Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Intrusion Detection in IoT networks using Machine Learning

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Francisco Camilo Mejías Espinosa

ADVISOR: Olga León Abarca

DATE: October 18th, 2023

Title: Intrusion Detection in IoT networks using Machine Learning

Author: Francisco Camilo Mejías Espinosa

Advisor: Olga León Abarca

Date: October 18th, 2023

Abstract

The exponential growth of Internet of Things (IoT) infrastructure has introduced significant security challenges due to the large-scale deployment of interconnected devices. IoT devices are present in every aspect of our modern life; they are essential components of Industry 4.0, smart cities, and critical infrastructures. Therefore, the detection of attacks on this platform becomes necessary through an Intrusion Detection Systems (IDS). These tools are dedicated hardware devices or software that monitors a network to detect and automatically alert the presence of malicious activity.

This study aimed to assess the viability of Machine Learning Models for IDS within IoT infrastructures. Five classifiers, encompassing a spectrum from linear models like Logistic Regression, Decision Trees from Trees Algorithms, Gaussian Naïve Bayes from Probabilistic models, Random Forest from ensemble family and Multi-Layer Perceptron from Artificial Neural Networks, were analysed. These models were trained using supervised methods on a public IoT attacks dataset, with three tasks ranging from binary classification (determining if a sample was part of an attack) to multiclassification of 8 groups of attack categories and the multiclassification of 33 individual attacks. Various metrics were considered, from performance to execution times and all models were trained and tuned using cross-validation of 10 k-folds.

On the three classification tasks, Random Forest was found to be the model with best performance, at expenses of time consumption. Gaussian Naïve Bayes was the fastest algorithm in all classification's tasks, but with a lower performance detecting attacks. Whereas Decision Trees shows a good balance between performance and processing speed.

Classifying among 8 attack categories, most models showed vulnerabilities to specific attack types, especially those in minority classes due to dataset imbalances. In more granular 33 attack type classifications, all models generally faced challenges, but Random Forest remained the most reliable, despite vulnerabilities.

In conclusion, Machine Learning algorithms proves to be effective for IDS in IoT infrastructure, with Random Forest model being the most robust, but with Decision Trees offering a good balance between speed and performance.

I would like to thank to my Advisor Olga León Abarca,
for her guidance, support and patience during the execution of this project,
to my sister Amalia and niece Emily, for their hospitality towards me and my family, which made my stay in Barcelona possible.
And last but not least, to my wife Olga and my daughter Eva for their unwavering support and encouragement in the execution and completion of this Master. To both of them, I dedicate this work.

Francisco Camilo.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

The Internet of Things (IoT) [1] refers to the network of physical objects, or "things", implanted with sensors, software, and other technologies with the goal of connecting and exchanging data with other devices and systems via the internet. These objects range from common household items to sophisticated industrial equipment. The current growth of the Internet of Things (IoT) is experiencing an explosive surge, as these networks become more complex and widely available, they bring with them a host of benefits and advances. However, the expansion of IoT systems also poses considerable cybersecurity threats, amplifying the difficulty of safeguarding against malicious intrusions. Given the diverse nature and widespread deployment of IoT devices, they are frequently exposed to a variety of cyber threats, emphasising the significance of their security.

Traditional security measures, which are based on rule-based and signature detection techniques, often struggle to keep up with the dynamic and evolving nature of cyber threats. In consideration of the resource constraints exhibited by many IoT devices, including low energy storage, limited memory and low CPU, these devices are susceptible to cyber-attacks. A notable factor contributing to such vulnerability is their inability to support existing general-purpose security software. By this reason, a Network Intrusion Detection Systems (NIDS) or Intrusion Detection Systems (IDS), functions as a vigilant monitor for internet traffic in an IoT network, acting as a protective barrier against intruders and potential threats. Its primary role is to identify known and unknown malicious attacks by scrutinising network actions, user behaviour, and device activities. IDS not only detects unauthorised intrusions, but it also encourages context-awareness among devices in the network, thereby facilitating defence mechanisms such as firewall rules. The IDS system detects both internal attacks, emanating from compromised IoT devices, and external attacks initiated by third parties. Its primary components include Observation, which tracks network patterns, and Analysis and Detection, which form the core of the system and identify intrusions by means of algorithms. The IDS also has an alert system that indicates any detected threat [2].

Machine learning (ML), with its capacity to analyse vast amounts of data and identify patterns, presents a promising approach for enhancing IDS in IoT environments and offers the promise of adaptive, data-driven solutions capable of identifying anomalous patterns and behaviours in real-time. Given the vast amounts of data generated by IoT devices, ML algorithms are particularly well-suited to sifting through large datasets to identify potential threats swiftly and accurately. However, the effectiveness of these algorithms is dependent on the quality and importance of the datasets they have been trained on.

This master thesis project explores the evaluation of five different supervised machine learning classifier models (Logistic Regression, Decision Trees, Naïve Bayes, Random Forests and Multi-Layer Perceptron) in a chosen IoT attacks dataset, analysing their performance metrics, execution times and their adaptability in discerning legitimate from malicious activities within IoT networks.

The dataset for the study is labelled with a total of 34 classes: 33 distinct classes corresponding to specific IoT attack types and an additional class dedicated to benign traffic. These classes are further categorised into eight groups, with seven of them representing different attack categories and one signifying benign traffic. Moreover, at a broader level, the data can be bifurcated into two primary classifications: 'attack' and 'benign', facilitating binary classification tasks.

The results obtained from this study, show the effectiveness of employing Machine Learning algorithms for the Intrusion Detection System in IoT infrastructure. On the three classification tasks, Random Forest was found to be the model with best performance, at expenses of time consumption. Gaussian Naïve Bayes was the fastest algorithm in all classification's tasks, but with a lower performance detecting attacks. Whereas Decision Trees shows a good balance between performance and processing speed.

The rest of the document follows this structure. Chapter number 2 describes the State of the Art about the IoT infrastructure, Intrusion Detection Systems in IoT, Machine learning techniques in the detection of IoT attacks. Section one explains concepts of the IoT infrastructure and common types of cyberattacks. Section two describes the functions and classification of an IoT Intrusion Detection System. Section three explains the machine learning techniques used on this work, like models and datasets and current IoT attack datasets and related work.

Chapter number 3 explains the methodology of this project. Section one describes the dataset used for the study. Section two covers the dimensionality reduction of the dataset's features. In Section three, the selection of the models used in the work is made. Section four concerns about the metrics employed on this study.

Chapter number 4 focuses on the obtained results and discussion of this study. Section One deals with the tuning of hyperparameters for every model for each of the three proposed classification tasks. In section Two, each model received final evaluation on the test set for the three supervised classification tasks: binary, one for group categories of attacks, and another for a finer classification of individual attacks. In the following sections results and final metrics are analysed and compared across different models. Section Three presents and discusses the results of the binary classification models. Section Four presents the outcomes of the group attack classification models, which are analysed objectively. Following this, Section Five explores the findings and interpretations of the individual attack classification models.

CHAPTER 2. STATE OF THE ART

This chapter is divided in three sections. Section one explains concepts of the IoT infrastructure and common types of cyberattacks. Section two describes the functions and classification of an IoT Intrusion Detection System. Section three explains the machine learning techniques used on this work, like models and datasets and current IoT attack datasets and related work.

2.1. Internet of Things

Internet of Things (IoT) devices have become an integral part of our daily lives, playing vital roles in various sectors such as healthcare, transportation, smart homes, Industry 4.0 and critical infrastructures. Figure 2.1 illustrates the estimation of the last Cisco Annual Internet Report global device and connection growth, where Machine to Machine (M2M) connections, also referred as IoT, will be half of the global connected devices and connections by this year of 2023, having 14.7 billion connections by 2023, with a grown tendency of the 50 percent in 2023 [3].

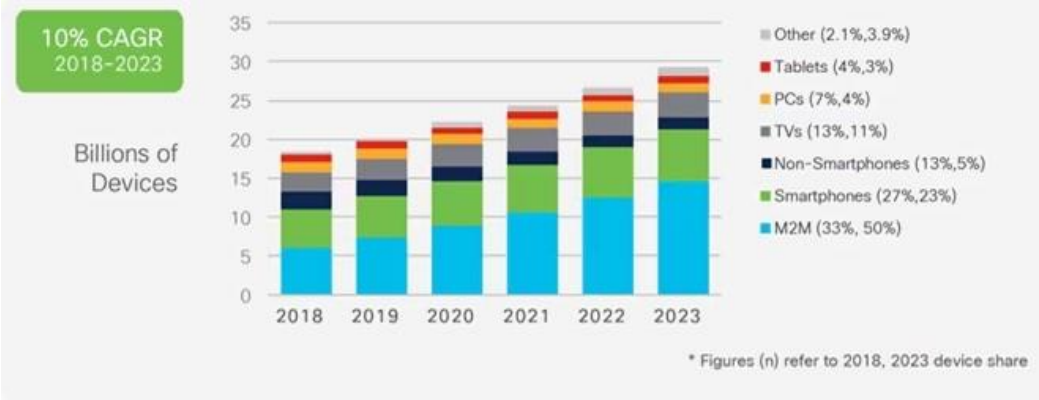


Fig. 2.1 Global devices and connection growth [3]

Figure 2.2 shows that, connected home applications, including home automation, home security, video surveillance, connected white goods, and tracking applications, are expected to comprise 48% or almost half of all M2M connections by 2023. This demonstrates the widespread integration of M2M in our daily life [3].

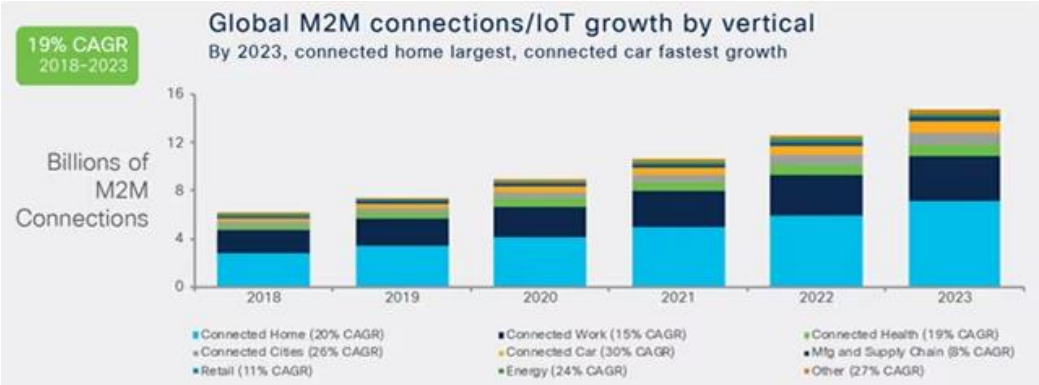


Fig. 2.2 Global M2M connection growth by industries [3]

IoT communication systems enable everyday devices to send and receive data and are crucial for the functionality of IoT devices, including smart home appliances and industrial sensors. To meet various requirements, such as range, power consumption, and data transmission rates, a wide number of communication protocols have been developed at different layers of the IoT protocol stack such as: [2][4].

- MQTT (Message Queuing Telemetry Transport): lightweight and ideal for use in low-bandwidth, high-latency, or unreliable networks. It is commonly used for remote monitoring, particularly in scenarios where bandwidth is limited.
- CoAP (Constrained Application Protocol): This web transfer protocol is specifically designed for use with constrained nodes and networks, such as those found in IoT environments. However, CoAP is more suitable for devices with limited processing capabilities.
- Zigbee: is a wireless protocol explicitly designed for short-range, low-power communications. it has become particularly popular among home automation systems.
- Z-Wave: Another protocol for home automation, with similar functionalities to Zigbee but different operating frequencies and specifications.
- LoRa (Long Range): As the name implies, it's specifically designed for long-range communication. This makes it an ideal choice for agricultural and other large-scale applications due to its long reach, even in challenging environments.
- Bluetooth and BLE (Bluetooth Low Energy): Although Bluetooth is commonly used for short-range communication, BLE offers a comparable range with much lower power consumption, thus being ideal for IoT devices that are battery-operated.
- NB-IoT (NarrowBand IoT): This cellular technology allows IoT devices to utilize the reliable communication infrastructure of existing mobile networks. It is engineered for usage in applications that do not need high bandwidth but necessitate power efficiency and extended range.
- Wi-Fi: Suitable for devices that require high data rates and are within range of a Wi-Fi network, this protocol is commonly used in smart home devices.

2.1.1. IoT Infrastructure

The IoT infrastructure [2] is complex, multi-layered, and specifically designed to facilitate data exchange, processing, and analytics across a vast network of interlinked devices and is dependent upon three significant elements: the cloud, fog, and edge layers, each serving a unique purpose, as shown in Figure 2.3.

1. The cloud layer is the central repository where vast amounts of data are stored, processed, and analysed on a grand scale. The cloud offers unparalleled storage and computational capabilities, facilitating intricate

analytics and insights. The IoT infrastructure is designed to be scalable, adjusting its capacity based on the influx of data from the ecosystem.

2. The fog layer, widely known as fog computing, functions as a central processing intermediary between edge devices and the central cloud. Situated nearer to the data source than centralised data centres but more widely dispersed than the edge, fog nodes can locally process data, thereby making real-time decisions and reducing the need for all data to travel to the cloud. This leads to a reduction in latency and bandwidth usage.

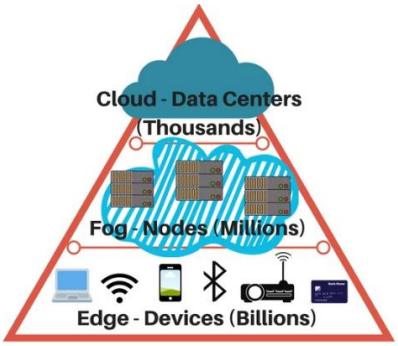


Fig. 2.3 Cloud, Fog and Edge Computing on IoT [5]

3. The edge layer sits adjacently to the fog layer and operates even closer to IoT devices. Edge computing allows devices to process data on-site even before it reaches the fog layer. By handling immediate data processing needs, it further streamlines the data that must be sent to the central system, conserving bandwidth, and ensuring timely actions.

Another essential component of the Internet of Things (IoT) infrastructure are the Gateways. They serve as connectors between IoT devices and communication networks, allowing the transmission of data between these devices and cloud-based management or storage platforms. Besides their data-relaying function, gateways have the capability to perform local data processing and analysis, thus enabling real-time decision-making without the need for cloud-based resources. This proves particularly beneficial in applications where latency is critical or when the volume of data generated by devices is too vast to be continuously sent to a centralised server [5].

Beyond their transmitting and processing capacities, gateways offer an added layer of security for IoT systems. By acting as an intermediary, they can implement IDS, authenticating devices, and filtering out unwanted traffic, which shields vulnerable devices and guarantees that only valid and secure data is transmitted to the cloud or other devices. Figure 2.4 shows an example of IoT gateways connected to IoT networks.

2.1.2. Attacks on IoT ecosystem

While IoT devices provide tremendous benefits in terms of automation, efficiency, and convenience, they also present significant security challenges. IoT devices are often designed with functionality in mind, but not always with sufficient attention to security considerations, making them a prime target for cyber-attacks.

There are numerous types of attacks targeting Internet of Things (IoT) devices and platforms. Below, there is a list of some of the most common ones and description of their main characteristics [2][7][8]:

- Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks shown in Figure 2.4, are aimed at rendering a device, network or service unusable by inundating it with a rush of internet traffic. The traffic in a DDoS attack comes from numerous sources, most commonly from botnets consisting of breached machines, making it more difficult to stop. IoT devices are frequently the subject of DDoS attacks and can also play a role in the botnets responsible for carrying out the attacks.

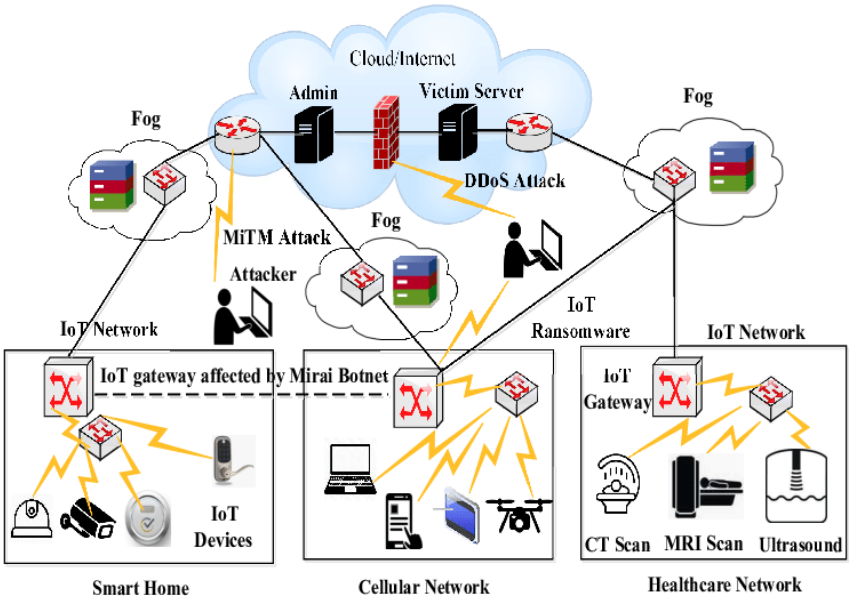


Fig. 2.4 Security attacks on IoT devices [5]

- Eavesdropping/Interception/Sniffing Attacks: Eavesdropping constitutes a passive attack wherein an attacker monitors network traffic. Through this, they could potentially obtain access to sensitive data, such as passwords, credit card numbers, or other personal information. Additionally, the attacker might be capable of identifying communication patterns between devices or systems, which they could then exploit in other manners.
- Man-in-the-Middle (MitM) Attacks: where an attacker secretly interferes with and potentially modifies the communication between two parties who assume that they are contacting directly. In the IoT framework, MitM attack can include an attacker intercepting communications between a user and a smart home device, allowing unauthorized access to sensitive data or control of the device.
- Spoofing Attacks: an aggressor impersonates another user or device on a network, with the intention of launching assaults against network hosts, stealing data, proliferating malware, or circumventing access controls. In an IP address spoofing attack, for example, an attacker may transmit packets that appear to originate from a fake source IP address, thus concealing their true identity.

- Replay Attacks: involves an assailant recording a data stream (such as a user entering a password) and subsequently repeating it to imitate legitimate user actions. This can enable the aggressor to acquire system access or cause system malfunctions.
- Jamming: The vast majority of IoT devices use wireless networks to communicate with one another. In this type of attack, the perpetrators target a specific IoT system and send a falsified signal to disrupt the radio transmission, leading to a reduction in available bandwidth, processing power, and memory.
- Malware Attacks: a type of malevolent software utilised or programmed by attackers to sabotage device operation, obtain confidential information, or acquire entry to private systems, poses a dangerous threat to individuals and corporations alike. In the realm of IoT devices, malware can be deployed to commandeer a device or incorporate the same into a botnet specifically for DDoS attacks.
- Mirai Botnet Attack: this is a type of malicious software that converts networked devices into remotely controlled bots that cyber attackers use as part of a botnet to conduct large-scale attacks. It focuses primarily on online consumer devices like IP cameras and home routers. Mirai is also commonly used as an initiator of DoS/DDoS attacks, an example is shown in Figure 2.4.
- Sybil Attack: Sybil attacks are discovered in peer-to-peer networks. A Sybil attack undermines the identity of an IoT device to generate numerous anonymous identities and consume a disproportionate amount of power. In networks where an IoT device functions with multiple identities, it frequently undermines authorized network access in reputation systems.
- Side-channel attacks target systems, like cryptographic ones, by studying their physical characteristics during operation. By observing elements such as power consumption, timing, and electromagnetic leaks, attackers can gather extra information. This acquired data can subsequently be exploited to breach cryptographic systems and access sensitive information.
- Cryptanalysis Attacks: These attacks aim to overcome the cryptographic protections implemented on data. They may include approaches such as ciphertext-only attacks (where the attacker only possesses encrypted data), known-plaintext attacks (where the attacker has access to both encrypted and unencrypted data), or chosen-plaintext attacks (where the attacker can encrypt any data and observe the outcome).
- Tampering: Modifies the hardware or software of a device to change its operation. Tampering may involve altering the device's programming to disrupt its functionality, gain access to sensitive data, or enable an attacker to control the device. Given that IoT devices are frequently deployed in unsecured public or remote locations and are often small and constrained, they are more vulnerable to physical attacks.
- Password Cracking: Password cracking refers to the process of guessing passwords to gain unauthorized access to a system. A range of techniques can be employed by attackers, including educated guessing, dictionary

attacks (where all words in a dictionary are tried), and brute force attacks (where all possible combinations are attempted).

- Advanced Persistent Threats (APT): refers to prolonged and targeted attacks by unauthorized individuals who attain access to a network and evade detection for a considerable duration. The primary objective is often data theft rather than network damage or disruption. Large organizations are typically the targets of APTs, which require an intricate level of stealth and sophistication.
- AI-Based Attacks: Modern hackers utilise AI-driven tools that are faster, scalable, and more efficient compared to manual interventions. This represents a growing threat to the ever-expanding IoT landscape. While traditional IoT threats may appear similar, the intensity, automation, and specificity of AI-powered attacks will make their mitigation increasingly challenging.

Examples of real-life attacks [7]:

- In 2016, Mirai malware attacked IoT devices, including cameras and routers, by using default credentials. Due to this malicious software, a botnet was formed using these compromised devices to launch distributed denial-of-service (DDoS) attacks. The DNS provider Dyn was notably affected, causing temporary disruptions to services such as Netflix, Twitter, and The New York Times.
- In 2015, researchers showcased remote manipulation of a Jeep Cherokee's telematics system, gaining control over its engine, brakes, and other key functions. Given the potential life-threatening consequences, such as hackers tampering with the brake system, Fiat Chrysler allocated 1.4 million USD to address the system's vulnerabilities.

2.2. Intrusion Detection Systems (IDS)

In the context of IoT infrastructure, an Intrusion Detection System (IDS) is a necessary element to identify and neutralise potential threats and attacks objectively. IoT devices are vulnerable due to their inherent constraints and vast numbers, making them inviting targets for attackers. State of the art network intrusion detection systems (NIDS) used in IoT systems can be categorised into Signature, Anomaly, Specification, and Hybrid types based on their framework, implementation, and operation [2]:

1. Signature-based NIDS are challenging to implement in IoT systems due to the resource limitations of IoT devices, such as memory, processing capability, and energy constraints. Traditional signature-based systems necessitate extensive datasets for robust detection. Therefore, there's a need to restructure traditional signature-based NIDS to be compatible with the resource constraints of IoT devices.

2. Anomaly-based NIDS for IoT systems compare current activities against a standard behavioural profile and generate alerts when deviations exceed a predetermined threshold. This approach efficiently detects emerging attacks,

particularly those that exploit IoT device resources. Unlike signature-based NIDS that depend on recognised attack patterns, anomaly-based systems rely on either typical or anomalous data. Given their potential for being lightweight, many NIDS for IoT security make use of anomaly-based approaches.

3. Specification-based Network Intrusion Detection Systems (NIDS) utilise a set of rules, manually updated by the IoT system administrator, to detect network intrusions. High-level rules are established based on the IoT network's surroundings and performance to ensure comprehensibility. Resource constraints in IoT systems make this technique preferable. However, this approach can only locate intrusions that match its predetermined rules, limiting its efficacy. The primary differentiation between specification-based and anomaly-based NIDS resides in the individualised manual rule-setting for each attack.

4. Hybrid NIDS for IoT systems combine several detection strategies, including signature-based, anomaly-based, and specification-based methods, to enhance strengths and minimize weaknesses. Two primary classifications for hybrid NIDS exist [2]:
   a) Sequence-based, which applies either anomaly or misuse detection first, followed by a distinct technique.
   b) Parallel-based. Multiple detectors operate simultaneously, and decisions are reached by considering outputs from multiple sources.

Typically, hybrid NIDS incorporate both signature-based and anomaly detection. Signature-based detection identifies known attacks, while anomaly detection identifies new or unidentified attacks. Because IoT systems have limited resources, implementing a hybrid-based NIDS directly in these systems often proves impractical.

IoT NIDS designs can also be categorized based on their operational mode into two main types: Host-based and Network-based. Table 2.1 illustrates the main differences between them.

Table 2.1. Differences between Host-Based and Network-Based IDS [2]

	Host-Based NIDS	Network-Based IDS
Data Source	System call logs	Captured network traffic
Placement Strategy	Locally on the hosted device or machine	Specific IoT devices on the same subnet
Detection Rate	Low, difficult to detect new attacks	High, can detect new attacks in real time
Threats Trace-ability	Based on system calls	Trace intrusion based on network addresses and timestamps
Limitations	Cannot analyse network attacks, rules created can be obsolete, depends on the operating system (OS)	Monitor only network traffic within a specific subnet

The placement of an IDS in an IoT infrastructure is contingent on the specific use case, the characteristics of the devices involved, and the network architecture. Here are several potential locations for an IDS in an IoT infrastructure [2]:

1. Device Level (Edge IDS): Directly on IoT devices, suitable for those with ample computational power, enabling real-time detection.
2. Gateway Level: At IoT gateways that collate data from multiple devices, ideal when individual devices can't host their own IDS.
3. Network Level (Network-based IDS or NIDS): Monitors entire IoT network traffic and is strategically positioned to oversee traffic flow.
4. Cloud level: Essential due to reliance on cloud platforms, it checks the authenticity of data being transmitted.
5. Fog level: this placement helps decrease threat detection latency, which is crucial for timely responses.
6. Host-based IDS or HIDS focuses on specific applications' activities, ensuring they process IoT data correctly and without malicious interference.

The most effective IoT security strategies often employ multiple IDS placements across the infrastructure, providing layered security. This multi-tiered approach ensures that if one IDS misses an intrusion, another might catch it. However, it's crucial to balance security needs with the performance overhead introduced by these systems, especially in resource constrained IoT environments.

2.3. Machine Learning techniques

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that enables computers to learn without explicit programming. It involves developing a predictive algorithm for each problem to be solved. These algorithms learn from data in order to identify patterns and trends, creating a model for predicting or classifying elements [10]. Deep Learning is a subset of Machine Learning that uses multiple layers to progressively extract higher-level features from the raw input. The term "deep" in "deep learning" indicates the number of layers utilized to transform the data. Most of Deep Learning algorithms are based on Artificial Neuron Networks (ANN) [11].

Machine learning systems can be categorised based on the level and type of supervision during training. Three main categories exist [12]:

1. Supervised learning: where the training dataset fed into the algorithm consists of target outcomes, referred to as labels. Supervised learning can be subcategorised according to their tasks into two types:
   a) Classification: where the target label is categorical and can be binary or multiclass, resulting in models known as classifiers.
   b) Regression: in this approach, the algorithm attempts to forecast a numerical value. The models used for this purpose are referred to as regressors.
2. Unsupervised learning: where training takes place on a dataset without previous labelling or defined classes. In advance, there is no known objective or class value, whether categorical or numerical. Typically, they are dedicated to clustering or segmentation tasks.

3. Reinforcement learning: the learning system, called an agent within this context, can observe the environment, choose and execute actions, and receive rewards in return (or penalties in the guise of negative rewards). It must subsequently learn on its own what is the optimal strategy, known as a policy, to accumulate the maximum reward over time.

The objective of this Master Thesis is to compare different machine learning algorithms to be used in Intrusion Detection Systems on IoT infrastructure. Specifically, employing supervised classification models to detect various types of attack. A typical configuration of an IDS using ML is shown in Figure 2.5.

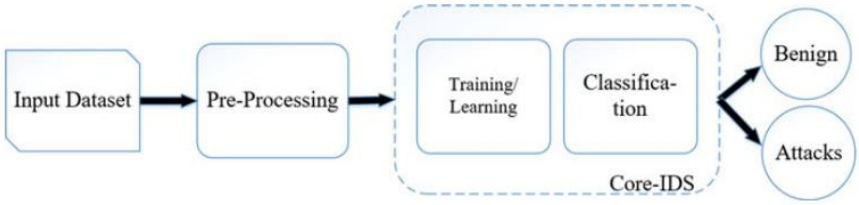


Fig. 2.5 Typical IDS scheme using ML [13]

Therefore, an Artificial intelligence system using machine learning for detecting attacks on the Internet of Things (IoT), must undertake the following steps:

1. Data Collection: begins by acquiring relevant data from IoT devices, can be during normal conditions and simulated or real cyberattacks.
2. Data Cleansing: cleanse the data by omitting redundancies, rectifying missing values, and encoding or normalising data types.
3. Feature Extraction: identify key data attributes that can enhance the learning process.
4. Model selection and training: choose and train appropriate models for the data.
5. Evaluation of the Model: post-training, validate the model's accuracy using metrics.
6. Deployment and Monitoring: deploy the effective model to monitoring the network, making adjustments as required.
7. Batch Learning: For continuously growing data, adopt a batched or incremental learning approach to keep the model updated.

Detecting an attack is primary a binary classification task, with only one main objective, detect or classify the traffic sample as been part of an attack or not. But nowadays, when the world tends to specialisation, more and more data to be analysed, more complex devices in the infrastructures, a more detailed classification of an attack is required in order to take the appropriate measures and possible future fixes and workarounds. Just binary classifying is not enough to proper handle the detected threat, a more granular classification is needed. Therefore, the classification of groups of attacks or specific attacks is the task of multiclassification problems. This study analysed two additional multiclassification tasks: one to detect/classify the group of the attack and another to detect/classify the individual attack.

2.3.1. Datasets

IoT attack datasets are essential tools for comprehending cyber threats and crafting effective protective strategies. These datasets, either drawn from real-world scenarios or artificially constructed, detail typical and anomalous behaviours within IoT settings. They contain several data entries, showcasing elements like packet dimensions, time markers, IP details, and more. These valuable resources aid researchers and cybersecurity experts in several ways: from moulding machine learning frameworks for intrusion detection to gauging the efficacy of security algorithms.

As IoT device adoption surges, so will the relevance of these datasets. A diverse array of IoT datasets exists within the cybersecurity realm, but the most of them typically exhibit certain standard attributes to ensure their utility, such as [14][15]:

1. Multidimensionality: Commonly known as features, entries often detail several attributes, spanning from IP addresses to protocol types across various IoT protocol stack layers.
2. Attack Diversity: An effective dataset encompasses various attack modalities, from DDoS to malware threats like botnets. This variety aids in building robust security models.
3. Device Diversity: Given IoT's vast device landscape, datasets should encapsulate varying devices, makers, and OS types.
4. Temporal Characteristics: The time-sensitive essence of many IoT transactions and threats necessitates datasets to capture time-related features, such as inter-packet intervals.
5. Scalability: Given the large-scale nature of many IoT deployments, datasets should be sufficiently large and scalable.
6. Labelling: For supervised learning approaches, the objective of this project, the data points should be labelled, identifying whether the data point represents normal behaviour or a specific type of attack.
7. Real-world Data: While synthetic data can be useful, real-world data provides the opportunity to capture and understand the complexity of IoT environments and attacks more accurately.

Creating datasets for Internet of Things (IoT) attacks is a complex process that involves simulating or capturing real-world IoT environments and their associated vulnerabilities as shown in Figure 2.6 [16] [17]:

1. IoT Environment Setup: An environment replicating authentic IoT scenarios is established, incorporating diverse IoT devices, communication protocols, and applications.
2. Attack Simulation: After environment setup, different types of attacks are simulated. This can include DDoS, MitM, malware, sniffing/eavesdropping, spoofing, etc. Attacks are often performed using penetration testing tools and automated scripts, for example, several setups with Kali Linux executing its known vulnerability tools.
3. Data Capture: All traffic in the IoT environment, including both normal and malicious traffic, is captured using network sniffing tools like the well-known

Wireshark, Tcpdump, Argus, etc. The data captured includes packet-level data like source and destination IP addresses, TCP/UDP ports, payload size, packet timestamps, and protocol type.

4. Pre-processing and Feature Extraction: The raw captured data often needs to be processed before it is useful. Pre-processing can include steps like noise removal, dealing with missing data, and normalisation. Feature extraction is performed to derive meaningful features from the raw data that can be used for modelling and analysis. This could include statistical features, time-based features, content-based features, etc. This step is crucial for the database.
5. Labelling: For supervised learning tasks, the dataset needs to be labelled. This involves identifying and marking the data points associated with different types of attacks. Labelling can be a challenging and time-consuming process, often requiring domain expertise.
6. Data Validation: Finally, the dataset is validated to ensure its accuracy and effectiveness. This involves using the dataset in various machine learning models to test if it can accurately detect and classify IoT attacks. The dataset might need to be refined and re-validated multiple times before it is considered ready for use.

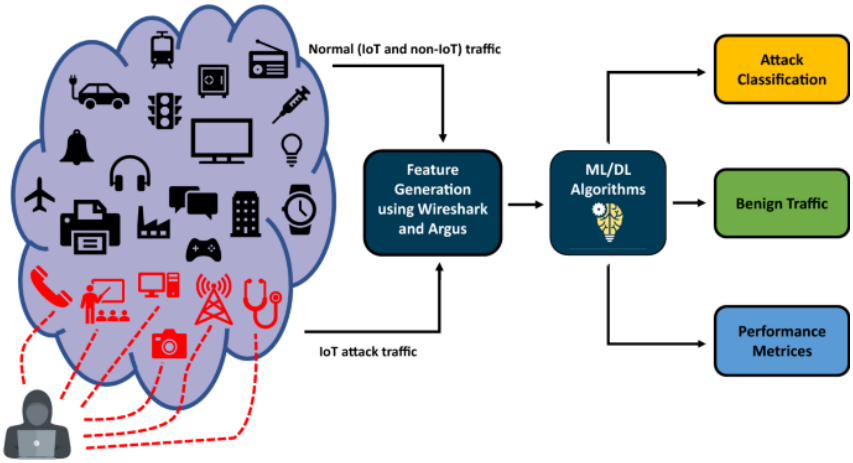


Fig. 2.6 Example of Creation of an IoT attacks dataset [18]

There are no standard procedures for creating an IoT attack dataset, it depends on the IoT context and the types of attacks. While some datasets are generated from controlled simulations to prevent actual damage, others are based on genuine IoT traffic, prioritising privacy and legality.

2.3.2. Public IoT Attack Datasets

Realistic IoT attack datasets are used by researchers to design and test new security mechanisms. These datasets are fundamental for investigating the numerous cyber threats that target IoT systems. It is crucial to keep the datasets up-to-date and appropriate as the nature of IoT threats changes constantly. This facilitates the creation of robust security measures and better understanding of emerging threats in the IoT sector [2]. For general IDSs, there are well known and standards datasets like KDD99, NSL-KDD, UNSW-NB 15, a review of them

are explained in [19] [20]. But more specific IoT attack datasets and related work are listed below:

- TON-IoT: is crafted for AI-based IDS within the Industry 4.0 and Industrial Internet of Things (IIoT) setting. It aids in evaluating AI models' intrusion detection performance using a realistic large-scale network with multiple virtual machines and sensors. The dataset creators ran different models from Naïve Bayes to Neural Networks, obtaining metrics from 90% to 98% [21].

- Bot-IoT: combines genuine IoT traffic with a variety of attack traffics, especially Botnet, filling the gap in the comprehensive collection of network traffic and attack diversity in existing datasets [22]. The dataset, available in both PCAP and CSV formats, organises files by attack types for easier labelling, authors in [23] obtained a 99.99% of accuracy employing Decision Trees models.

- MQTTset: targets intrusions against the MQTT protocol, a common IoT application protocol. The dataset, derived from real-life sensors, encompasses multiple attack types, including an innovative DoS attack named SlowTe. With 33 features, it highlights various TCP and MQTT protocol attributes. Researchers obtained accuracy between 64% and 91% running several models, from Naïve Bayes to Neural networks [24].

- N-BaIoT: use nine IoT devices. This dataset encompasses four kinds of attacks: reconnaissance, man-in-the-middle, denial-of-service, and botnet malware. From the PCAP files, 115 features are derived, capturing details from packets, steam, and time frames. The authors claimed a 100% of True Positive Rate (TPR) using deep autoencoders with neural networks [25].

2.3.3. Machine Learning Classifier Models

The supervised classifiers that were trained and assessed fell into five distinct categories: Logistic Regression from linear models, Naïve Bayes from probabilistic models, Decision Trees from tree models, Random Forests from ensemble models, and Multilayer Perceptron (MLP) from the Artificial Neural Network category.

1. **Logistic Regression (LR)**: Logistic Regression, also known as Logit Regression, is a widely used method for estimating the probability of an instance belonging to a particular class. If the estimated probability is above 50%, the model predicts that the instance belongs to that class (known as the positive class, labelled "1"). If the estimated probability is below 50%, the model predicts that the instance does not belong to that class (i.e., it is labelled as "0" being part of the negative class) [12][26][27].
2. **Gaussian Naive Bayes (NB)**: refer to a set of supervised learning algorithms that apply Bayes' probabilistic theorem. These methods make the "naive" assumption of conditional independence between every pair of features, given the value of the class variable. The likelihood of the features is assumed to be Gaussian [26][27].

3. **Decision Trees (DT)**: is a supervised learning method used for both classification and regression tasks. It graphically models decisions and their possible outcomes, including associated probabilities and costs. Starting with a root node containing the entire dataset, it splits data based on a feature that maximises separation. The choice of splitting is determined by criteria such as information gain or the Gini index. Internal nodes denote decisions, while leaf nodes represent final outcomes. They are easily interpretable and versatile, handling both categorical and numerical data [27][28][29].
4. **Random Forest (RF)**: Ensemble learning is the process of combining classifiers to increase the predictive performance of using singular models, this is the case of Random Forest where multiple decision trees are combined into one classifier, increasing accuracy by reducing the effects a single decision tree has on the overall outcome. This means if a decision tree has low accuracy, it will not have a significant impact on the overall accuracy of the random forest. Random forest utilises bagging (short for *bootstrap aggregating*) where sampling is performed with replacement which is what allows this method to achieve high classification accuracy [12][26][27][28].
5. **Multilayer Perceptron (MLP)**: is a form of feedforward Artificial Neural Network (FFNN) that falls into the category of supervised learning algorithms. FFNN it's a type of artificial neural network where connections between the nodes (also known as neurons) do not form a cycle and the data moves in only one direction, forward. Perceptrons are the foundational units of a neural network, behaving as single artificial neurons and functioning as single-layer networks. In contrast, MLPs have multiple layers. The neural network can comprise one or many hidden layers, and each layer can consist of one or several neurons. Neurons generally employ activation functions such as ReLU (rectified linear unit), sigmoid, or hyperbolic tangent. The output layer is used for classification tasks and can feature a SoftMax function for multi-class issues or a logistic (sigmoid) activation for binary classification problems. The MLP adjusts the neurons' weights according to the output error, which is propagated from the output layer to update the weights throughout the network, this algorithm is called backpropagation [12] [26][27][28].

One of the common problems associated when applying ML models is the overfitting. Overfitting happens when a model can predict very well on the training dataset but very badly on untrained/test data. The opposite behaviour is the underfitting, that happens when the algorithm performs very bad on the training and test set. [12] [26].

2.3.4. Cutting-edge ML techniques

There are several cutting-edge techniques were emerging in the field of IoT attack detection using machine learning, such as:

- Deep Learning Models: neural networks with many layers are capable of learning complex patterns. Types of deep learning models are Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM). CNNs process grid-like data, like images, and have potential for processing IoT network traffic. RNNs are

useful for analysing time-series IoT data. Long Short-Term Memory is a type of recurrent neural network (RNN) designed to learn and remember long-term dependencies in data sequences. [30][31].

- Federated Learning: this approach entails using ML to train a model across multiple devices or servers containing local data samples, without sharing them. Its effectiveness has been acknowledged in IoT networks, where individual devices can learn from their data and share model updates with a central server, thereby enhancing privacy [32].
- Transfer Learning: on the other hand, involves applying knowledge gained from solving one problem to a different, but related, problem. For example, a ML model trained on one IoT network may be adapted for use on a different network. This can save considerable computational resources and time as opposed to training the model from scratch [33].
- Automated Machine Learning (AutoML): encompasses a range of techniques and procedures to automate the complete machine learning process when dealing with real-world issues. Its implementation would be particularly advantageous in an IoT environment where models may need to be deployed across a large number of devices. Automated algorithm and hyperparameter selection can reduce reliance on expert knowledge [34].
- Reinforcement Learning: involves ML where an agent learns to make decisions by taking actions in an environment to maximise a reward. On the Internet of Things context, a model may learn to adjust network parameters to maximise security and respond to new threats in real-time [35].
- Explainable AI: refers to a suite of tools and frameworks that aid humans in comprehending and interpreting ML model predictions. Concerning IoT, it would enable a network operator to discern why the model identified specific network traffic as possibly harmful, thereby providing an enhanced understanding of both the model and the prospective threat [36].

CHAPTER 3. METHODOLOGY

This chapter explain the methodology used in this work and is organised as follow. Section one describes the dataset used for the study. Section two covers the dimensionality reduction of the dataset's features. In Section three, the selection of the models used in the work is made. Section four concerns about the metrics employed on this study.

The working environment utilised in the study had the following characteristics: Operating system with Windows 11 Pro 22H2 with a 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz Processor, 16GB of RAM and 512GB SSD, a NVIDIA GeForce RTX 3050 Laptop GPU graphic card with 4GB GDDR6. The main software used for the development was the Visual Studio Code 1.80.0 with Jupyter Notebook support, the language used was Python 3.11.2, for manipulating datasets in form of dataframes was used the library pandas 2.0.3, and finally, all modelling were done using the scikit-learn library 1.3.0. Scikit-learn is an open-source machine learning library that facilitates both supervised and unsupervised learning. It offers a range of facilities for model training, model selection, model evaluation, pre-processing of data, etc.

Evaluating the performance of ML models to be employed in IDS to properly identifying and classifying attacks in IoT, is accomplished through the following steps:

- Dataset selection.
- Feature selection and reduction.
- Model selection.
- Model Hyperparameters tuning.
- Model final evaluation.

3.1. IoT Dataset selection

Researchers from the University of New Brunswick of Canada (UnB), with the support of the Canadian Institute for Cybersecurity (CIC), proposed an extensive multiclass labelled dataset of IoT cyber-attacks, named by its authors, CICIoT2023 [37]. This dataset includes 33 different types of attacks (categorised into seven groups) performed within a topology of up to 105 IoT devices from various types and manufacturers as shown on Figure 3.1. The diversity of this dataset was one of the main reasons for its selection for the study. Additionally, it is one of the most recent IoT attack datasets available within the scientific community.

The entire dataset was released in the second quarter of 2023 and is publicly available in [38], with 46686579 samples, exceeds 13GB and is distributed across 169 .csv files. The original traffic captures in .pcap files are also provided by the authors, enabling researchers to develop their own feature extraction and labelling processes.

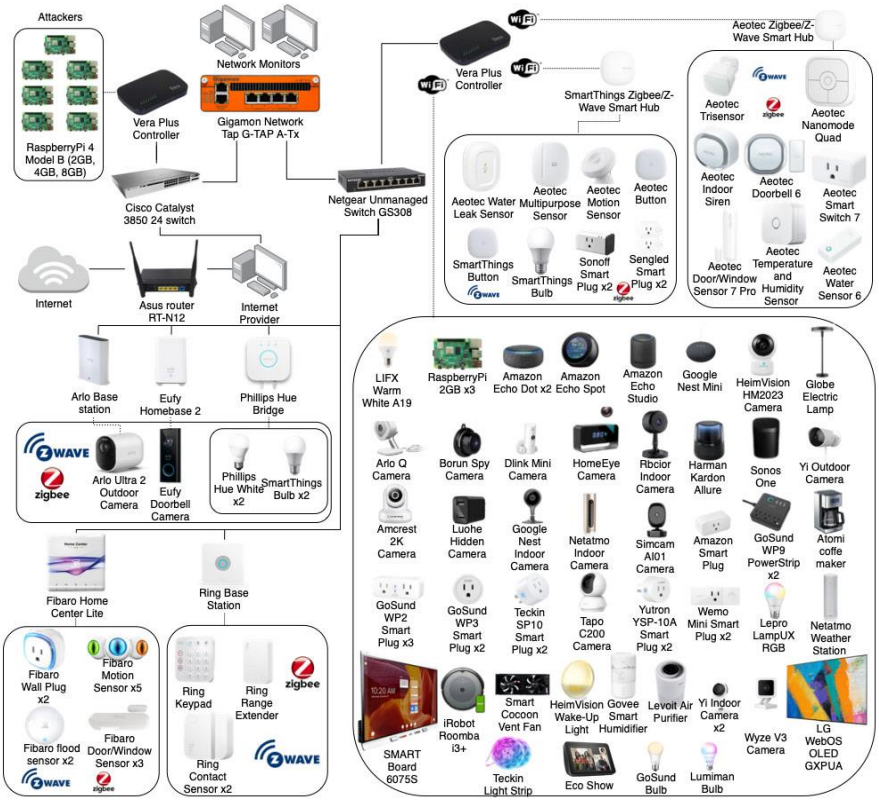


Fig. 3.1 Topology diagram of the scenario of Dataset. [37]

Due to technical constraints and limited resources, the study analysed only the 6 percent of the total dataset, while ensuring it maintained the same characteristics and class proportions as the original. The time required for implementing algorithms and machine learning models was meticulously recorded for performance analysis. The new Dataset for the analysis (a subset of the Full Dataset) has been obtained using the function **train_test_split** from the **scikit-learn** library, having the parameters: **test_size** equals to the new size requested in percentage and **stratify**=df['label'] pointing to the target variable, making a small new dataset of 2917910 samples but maintaining the same proportion of the classes of the target variable. The resulting dataset has been saved to an intermediate .csv file for further processing.

The obtained dataset was split into training and test sets, with an 80:20 proportion, respectively. This proportion is a common practice in machine learning and statistics, where is desirable to have a good balance of enough data for training and to validate and test the model. Some authors explain the relationship with the 80-20 Pareto rule, (where 80% of the effect are due to 20% of the cause, and vice versa) [39], and others had mathematically demonstrated this ratio in [40], but at the end this proportion has been proved to produce the best balanced results through many years of experimentation. The training dataset is used to train and adjust the model whereas the test dataset is utilised to finally evaluate the model's performance, ensuring that it can make accurate predictions on unseen data. All subsequent operations of feature extraction, training and work related are done in the training dataset, until the last step of evaluation of every model where the test dataset is used.

3.1.1 Exploratory dataset analysis

All features in the entire dataset are numeric, except for the 'label' variable, which is the target. It contains the outcome of the labelling process. Table 3.1 describes the number of observations or instances of each packet analysed. whereas Table 3.2 shown the names of the features and its description, these features are the result of the process of construction of the Dataset done by the authors.

Table 3.1. Dataset description

Samples	**Variables**	**Label Classes**
2917910	47 (including 46 features and one label)	34
Training Dataset Samples	**Test Dataset Samples**	
2334328	583582	

Table A.1 from the Annex A, describes the numerical properties (mean, standard deviation, min, max and percentiles) of each feature. It shows that every feature has no empty records, all features show the same count of instances.

The histograms displayed in Figure A.1 and the numerical descriptions presented in Table A.1 of Annex A reveal that the 'Telnet' feature lacks relevant points, as it shows all zeroes. This is not a surprise, because Secure Shell Protocol (SSH) obsoletes the old Telnet to login into remote systems. Therefore, “Telnet” Features should be eliminated from further analysis, reducing the number of features to 45. From a detailed analysis of the figures and tables previously mentioned, it can be inferred that several of the features shown are binary. This observation leads us to treat these variables as such during the processing and normalisation. Binary features don’t require normalisation they are in range [0,1]. Some of the models analysed such as LR, NB, MLP, require variable scaling or normalisation, because they rely on distances or gradients. Without normalisation, features with larger scales could disproportionately influence the model, potentially leading to misleading results or longer training times. By scaling or normalising, we ensure that each feature contributes proportionately to the final decision [27].

The target variable, named ‘label’ is a categorical one, describing 33 types of attacks and one class for benign traffic, and are also classified in 7 attack groups and one for benign traffic as shown in Table 3.3. For binary classification (last column) it has the ‘Attack’ and ‘Benign’ labels.

Table 3.2. Feature names and descriptions

Feature name	Description
flow_duration	Duration of the packet's flow
Header_Length	Header Length
Protocol Type	Protocol Type: IP, UDP, TCP, IGMP, ICMP, Unknown (Integers)
Duration	Time-to-Live (ttl)
Rate	Rate of packet transmission in a flow
Srate	Rate of outbound packets transmission in a flow
Drate	Rate of inbound packets transmission in a flow
fin_flag_number	Fin flag value
syn_flag_number	Syn flag value
rst_flag_number	Rst flag value
psh_flag_number	Psh flag value
ack_flag_number	Ack flag value
ece_flag_number	Ece flag value
cwr_flag_number	Cwr flag value
ack_count	Number of packets with ack flag set in the same flow
syn_count	Number of packets with syn flag set in the same flow
fin_count	Number of packets with fin flag set in the same flow
urg_count	Number of packets with urg flag set in the same flow
rst_count	Number of packets with rst flag set in the same flow
HTTP	Indicates if the application layer protocol is HTTP
HTTPS	Indicates if the application layer protocol is HTTPS
DNS	Indicates if the application layer protocol is DNS
Telnet	Indicates if the application layer protocol is Telnet
SMTP	Indicates if the application layer protocol is SMTP
SSH	Indicates if the application layer protocol is SSH
IRC	Indicates if the application layer protocol is IRC
TCP	Indicates if the application layer protocol is TCP
UDP	Indicates if the application layer protocol is UDP
DHCP	Indicates if the application layer protocol is DHCP
ARP	Indicates if the application layer protocol is ARP
ICMP	Indicates if the application layer protocol is ICMP
IPv	Indicates if the application layer protocol is IP
LLC	Indicates if the application layer protocol is LLC
Tot sum	Summation of packets lengths in flow
Min	Minimum packet length in the flow
Max	Maximum packet length in the flow
AVG	Average packet length in the flow
Std	Standard deviation of packet length in the flow
Tot size	Packet's length
IAT	The time difference with the previous packet
Number	The number of packets in the flow
Magnitue	(Average of the lengths of incoming packets in the flow + Average of the lengths of outgoing packets in the flow) ** 0.5
Radius	(Variance of the lengths of incoming packets in the flow + Variance of the lengths of outgoing packets in the flow) ** 0.5
Covariance	Covariance of the lengths of incoming and outgoing packets
Variance	Variance of the lengths of incoming packets in the flow / The variance of the lengths of outgoing packets in the flow
Weight	Number of incoming packets * Number of outgoing packets

Table 3.3. Label Classes attacks description, groups and binary classification.

Attack number	Multiclassification Class Name	Group Classification	Binary Classification
1	DDoS-RSTFINFlood	DDoS	Attack
2	DDoS-PSHACK_Flood	DDoS	Attack
3	DDoS-SYN_Flood	DDoS	Attack
4	DDoS-UDP_Flood	DDoS	Attack
5	DDoS-TCP_Flood	DDoS	Attack
6	DDoS-ICMP_Flood	DDoS	Attack
7	DDoS-SynonymousIP_Flood	DDoS	Attack
8	DDoS-ACK_Fragmentation	DDoS	Attack
9	DDoS-UDP_Fragmentation	DDoS	Attack
10	DDoS-ICMP_Fragmentation	DDoS	Attack
11	DDoS-SlowLoris	DDoS	Attack
12	DDoS-HTTP_Flood	DDoS	Attack
13	DoS-UDP_Flood	DoS	Attack
14	DoS-SYN_Flood	DoS	Attack
15	DoS-TCP_Flood	DoS	Attack
16	DoS-HTTP_Flood	DoS	Attack
17	Mirai-greeth_flood	Mirai	Attack
18	Mirai-greip_flood	Mirai	Attack
19	Mirai-udpplain	Mirai	Attack
20	Recon-PingSweep	Recon	Attack
21	Recon-OSScan	Recon	Attack
22	Recon-PortScan	Recon	Attack
23	VulnerabilityScan	Recon	Attack
24	Recon-HostDiscovery	Recon	Attack
25	DNS_Spoofing	Spoofing	Attack
26	MITM-ArpSpoofing	Spoofing	Attack
27	BrowserHijacking	Web	Attack
28	Backdoor_Malware	Web	Attack
29	XSS	Web	Attack
30	Uploading_Attack	Web	Attack
31	SqlInjection	Web	Attack
32	CommandInjection	Web	Attack
33	DictionaryBruteForce	BruteForce	Attack
34	BenignTraffic	Benign	Benign

It should be noted that throughout the development of the study, reference is made to the term 'group classification'. This is essentially a more refined multi-classification task related to 8 categories of attack groups. Whereas multiclassification belongs to the individual identification of 34 types of attacks, binary classification concerns determining whether a sample is part of an attack or not. A more detailed description of the group attack categories and the number of individual attacks is shown in Table 3.4.

Table 3.4. Group attack description.

Attack groups	Description name	Short name	Number of individual attacks
1	Distributed Denial of Service (DDoS)	DDoS	12
2	Denial of Service (DoS)	DoS	4
3	Mirai	Mirai	3
4	Reconnaissance attack	Recon	5
5	Spoofing	Spoofing	2
6	Web-Based vulnerabilities	Web	6
7	Brute force attacks	BruteForce	1
8	Benign	Benign	1 (benign)
		Total	34

A brief description of each attack group and their individual attacks can be found below:

1. **Distributed Denial of Service (DDoS) attacks:** described in section 2.1.2., these attacks are executed from several infected devices [41].
   a) **DDoS-RSTFINFlood**: The DDoS-RST/FIN Flood attack can disrupt servers by overwhelming them with traffic. This type of attack exploits the RST and FIN flags in TCP, which are responsible for controlling data transmission. The RST flag resets connections legitimately, while the FIN flag closes them after transmission. In the DDoS variant of this attack, an attacker sends a multitude of TCP packets with set RST or FIN flags, confusing the targeted server.
   b) **DDoS-PSHACK_Flood**: an attacker overwhelms a target using TCP packets with both the PSH and ACK flags set. PSH, representing the "push function" in TCP, bypasses standard buffering to send data immediately. When combined with the ACK flag in this attack, a flood of "urgent" packets strains the target's resources, causing service disruptions by exploiting the TCP push function.
   c) **DDoS-SYN_Flood**: targeting the initial handshake of a TCP/IP connection. In this attack, the attacker sends numerous SYN requests to a server from a fake IP address. The server responds with SYN-ACK packets, expecting ACK responses to establish a connection. Due to the spoofed IPs, these responses never come, leading to many half-open connections. This consumes server resources, potentially slowing it down or causing it to crash, preventing service for genuine users.
   d) **DDoS-UDP_Flood**: In this attack, the attacker bombards a host with numerous UDP packets targeting random ports. The host, failing to find an application for each port, responds with an ICMP 'Destination Unreachable' packet. This activity uses up the host's resources, potentially making it inaccessible.
   e) **DDoS-TCP_Flood**: this attack disrupts servers by inundating them with numerous TCP connections. The attacker forms an excessive number of

connections to the target server, saturating its capacity for concurrent connections. Consequently, the server fails to process genuine requests due to the surfeit of connections.

f) **DDoS-ICMP_Flood**: this attack overwhelms the target by sending numerous Internet Control Message Protocol (ICMP) echo-request packets or "pings". The objective is to saturate the target's network. In this assault, attackers flood the target with a deluge of "pings". Since each ping demands a response from the target, a constant stream of requests can quickly consume its bandwidth and processing capacity, making it unable to cater to genuine requests.

g) **DDoS-SynonymousIP_Flood**: a substantial quantity of manipulated TCP-SYN packets is sent, their source and destination addresses being the targeted address. This causes the server to utilise its resources in order to process the incoming traffic.

h) **DDoS-ACK_Fragmentation**: manipulates the packet fragmentation process in TCP/IP to overload a target. Here, the attacker dispatches a barrage of fragmented ACK (Acknowledgement) packets to the victim. As the target expends resources attempting to piece these packets back together, it can become sluggish or entirely non-functional due to the depletion of its resources.

i) **DDoS-UDP_Fragmentation**: bombards the target with numerous fragmented UDP packets. As the victim system tries to reconstruct these fragments, it expends considerable resources, which can cause system slowdowns or render it non-operational.

j) **DDoS-ICMP_Fragmentation**: floods the target with numerous fragmented ICMP packets. When the victim system attempts to piece these fragments together, it uses up a lot of its resources, potentially leading to reduced performance or system unavailability.

k) **DDoS-SlowLoris**: aimed to incapacitate specific web servers by consuming all available connections. It does so by sending deliberately prolonged partial HTTP requests, ensuring connections stay open. This eventually maxes out the server's connection limit, preventing legitimate users from connecting.

2. **Denial of Service (DoS) attacks**: very similar with DDoS and described in section 2.1.2, a DoS attack typically comes from a single device and Internet connection [42] [43].

a) **DoS-UDP_Flood**: same as the **DDoS-UDP_Flood**, being originated from one single spoofed IP.

b) **DoS-SYN_Flood**: similar as **DDoS-SYN_Flood,** originating from one single spoofed IP.

c) **DoS-TCP_Flood**: same effect as the **DoS-TCP_Flood** but conducted from a single spoofed host.

d) **DoS-HTTP_Flood**: overwhelms a server with HTTP requests, aiming to deplete its resources. These can be either GET (retrieving information) or POST (accepting data) requests. As the server processes these, it may

neglect legitimate user requests. Notably, this attack requires minimal resources or bandwidth from the attacker.

3. **Mirai attacks** are variants of the Mirai Bot, explained in section 2.1.2 [44].
   a) **Mirai-greeth_flood**: overwhelms a system with Generic Routing Encapsulation (GRE) packets, where the inner data consists of random IPs and ports, and the outer layer has genuine IPs.
   b) **Mirai-greip_flood**: like **Mirai-greeth_flood**, but the target is the packet encapsulation on the ethernet header.
   c) **Mirai-udpplain**: utilises a fixed string payload in the UDP packet known to the Mirai malware. Its constant size and content allow rapid packet generation to flood the target. As UDP doesn't necessitate a receiver's response, the attacker can intensify the attack by continuously dispatching packets.

4. **Reconnaissance attacks**: also known as information gathering attacks, are initial steps by attackers to gather detailed information about their targets, aiming to understand system vulnerabilities and plan subsequent attacks. Being passive in nature, these attacks are challenging to detect [45] [46].
   a) **Recon-PingSweep**: involves sending ICMP Echo Request packets across an IP range to identify active devices.
   b) Recon-OSScan: uses network probes to deduce the operating system of target devices.
   c) **Recon-PortScan**: also known as port scanning. This attack technique involves systematically scanning a target network or system to identify open ports and services available on the target devices to find potential vulnerabilities.
   d) **VulnerabilityScan**: specialized scanning tools are used to identify common security flaws, such as outdated software versions, misconfigurations, weak passwords, or missing security patches.
   e) **Recon-HostDiscovery**: employs techniques like ping sweeps and ARP scans to identify live hosts in a network.

5. **Spoofing attacks**: described in section 2.1.2, the main objective is to masquerades as another device or user on a network to launch attacks [47] [48].
   a) **DNS_Spoofing**: also known as DNS cache poisoning or DNS hijacking, is a malicious method where the DNS resolution process is altered to direct users to harmful websites by modifying DNS records.
   b) **MITM-ArpSpoofing**: is a tactic where an attacker disrupts network communication between two parties by impersonating a device's MAC address, enabling them to intercept and alter communications.

6. **Brute force attacks**: involve attackers using trial and error to access a system, systematically guessing passwords until the correct one is found.

These attacks leverage computational power to check all potential combinations [49].

a) **DictionaryBruteForce**: The goal of the attack is to find the correct password by trying all the words in the dictionary.

7. **Exploiting Web-Based vulnerabilities:** involves attackers taking advantage of flaws in web applications due to reasons like poor configuration, insecure coding, or using outdated software. When exploited, these vulnerabilities can give unauthorized access, retrieve data, or even control the system [50].
   a) **BrowserHijacking**: Unauthorized changes to browser settings by malicious code to redirect users to harmful websites.
   b) **Backdoor_Malware**: Malicious software that creates a hidden entry, or "backdoor", allowing attackers persistent unauthorized access to a system.
   c) **XSS**: Attackers inject malicious scripts into trusted websites, which unsuspecting users execute, potentially allowing data theft or unauthorized actions.
   d) **Uploading_Attack**: Attackers exploit file upload vulnerabilities to upload and execute harmful files on a target, compromising it.
   e) **SqlInjection**: Attackers manipulate database queries by injecting malicious SQL statements, enabling unauthorized access, data retrieval, or modifications.
   f) **CommandInjection**: Attackers execute arbitrary system commands due to the application's failure to sanitise user input.

8. **Benign traffic**: as its name indicates, these are packets labelled for normal benign traffic, considered not part of an attack.

In Figure 3.2 is shown the distribution of each class of attack of the training set, for group classification the distribution of the seven attacks and the benign traffic is shown in Figure 3.3, and finally for the binary classification, be part of an attack or a normal/benign packet, Figure 3.4 shown the distribution. From the analysis of three classes distributions, it can be observed a class imbalance in the dataset for every classification task, and this situation will be taken into consideration in subsequent tasks.

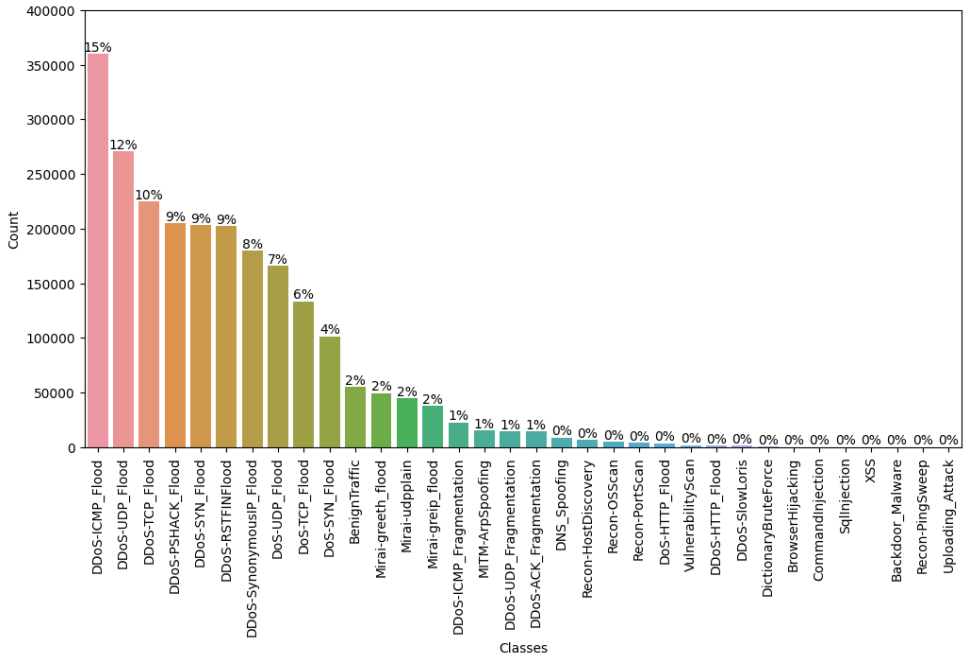


Fig. 3.2 Multiclass distribution of the target variable ‘label’

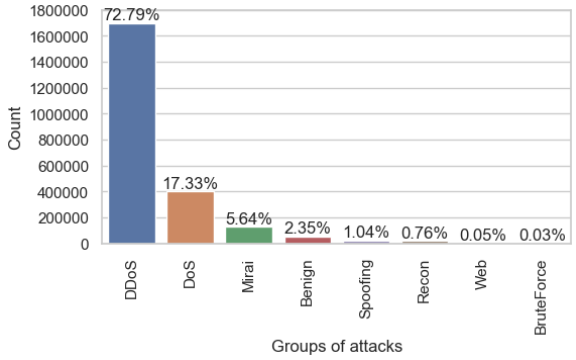


Fig. 3.3 Attack groups categories distribution

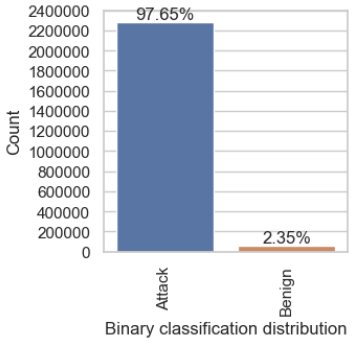


Fig. 3.4 Binary classes distribution

3.2. Feature extraction and reduction

Feature reduction is the process of selecting a group of most relevant features from all the available features of a dataset. The objective of feature reduction is to reduce the dimensionality of the dataset, preserving the most important

information. This can help improve the efficiency, speed and effectiveness of machine learning models, and also reduce the risk of overfitting. Some of these techniques are listed below [51] [52] [53]:

1. Principal Component Analysis (PCA): is a dimensionality reduction technique that reduces the feature space by retaining the maximum possible information. It creates new features from linear combinations of the old ones and selects those capturing the most variance in the data.

2. Feature Selection using Correlation Metrics: Feature selection via correlation metrics identifies highly correlated features using correlation matrices. However, a strong correlation does not necessarily indicate the significance of a feature, as it may fail to identify non-linear relationships.

3. Feature Importance from Decision Trees: Feature importance in decision trees is determined using models like Random Forest, Gradient Boosting, and Extra Trees. The significance of a feature is measured by the overall reduction of the criterion attributed to that feature.

4. Permutation Importance: Permutation feature importance measures the significance of features by shuffling their values and evaluating the decrease in model accuracy. Features that greatly influence the model's performance upon permutation are considered essential.

5. Mutual Information or Information Gain: The mutual information between two variables determines whether they are dependent on each other. If two random variables are independent, the mutual information will be zero, but a higher value indicates a stronger dependence.

In this work it has been adopted a hybrid approach by combining Pearson's coefficient [54][55], which is a standard correlation coefficient, with Mutual Information (MI) [56][57]. The Pearson's coefficient indicates the linear relationship between two variables. The range of the coefficient goes from -1 to 1. A score of 1 shows a perfect positive correlation and the score of -1 shows a perfect negative correlation. A score of zero, shows no correlation between the variables. Scores in the modular range of 0.7 to 0.99 considered the variables to be strong correlated linearly [55].

Mutual Information (MI) measures the amount of information gained about one variable as a result of observing another variable, as quantified by information theory. Mutual Information is closely related to entropy, which measures the amount of uncertainty or randomness of a variable. If two variables are independent, their mutual information is zero; they have no information about each other. A higher mutual information indicates a stronger association between the two variables [12] [56].

This election was based on removing the variables/features that are linear correlated first, based on the Mutual Information that its results are not dependent of any model evaluation, concerning models, is neutral, recalling that in this study five models from different types are analysed, not only models related to tress

like in Feature Importance from Decision Trees. Concerning PCA, a wide technique used, assumes linearity of the variables and a normal distribution of them, but none of the features follow this pattern, many of them are binary as shown in the Feature distribution of Figure A.1 of Annex A.

First, the mutual information is obtained of every feature against the target variable by the function **mutual_info_classif** from the Scikit-learn library [57]. Figure 3.5 shown the normalised mutual information importance relative to the target.

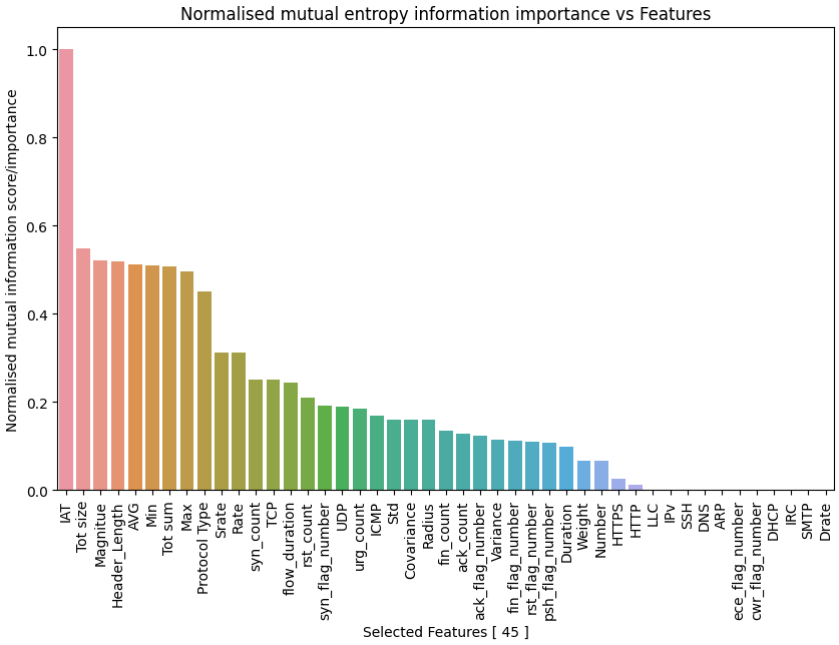


Fig. 3.5 Normalised mutual information importance of every feature

In order to reduce linear correlated variables, the Pearson coefficients are obtained in a correlation matrix of dimension [m x m], where m is the number of features, Figure 3.6 shows the correlation matrix as a heatmap, stronger colours denote high correlation between features. A threshold of 0.7 was set to find sets of strong correlated features. The objective is to select the most important feature of every set found based of the mutual importance of the feature. For example, if a set of three variables are found to be heavy correlated, having its correlation coefficient between each other above the threshold, the best feature to be selected is the one with better mutual information importance. After this procedure, the number of features is reduced from 45 to 27 features. The value 0.7 is a conservative number between 0.5 and 0.99 of strong correlation [55].

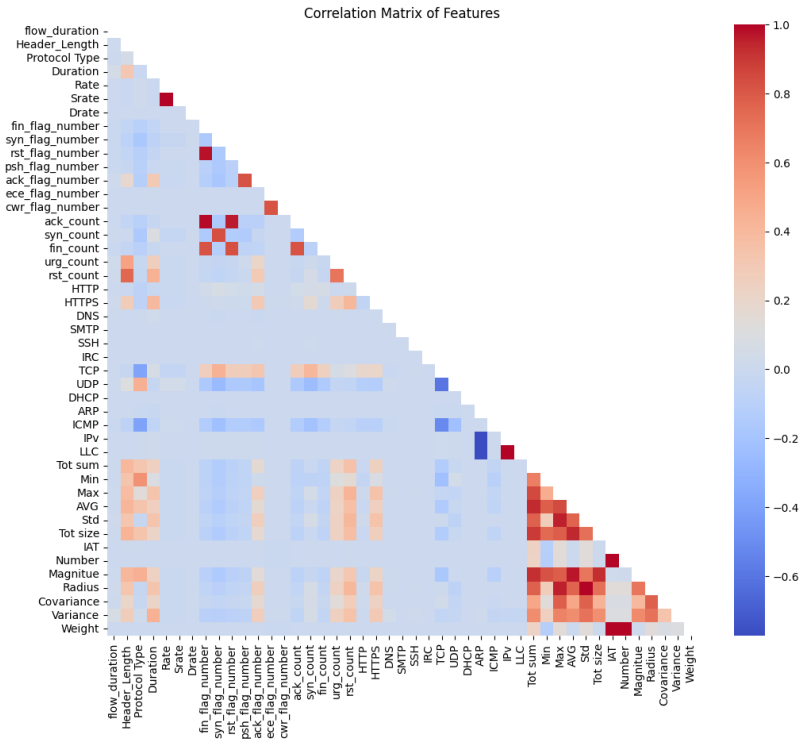


Fig. 3.6 Correlation matrix of features vs features

With the result of 27 features, another threshold of 0.1 is set to eliminate the less important features based on the mutual information importance, features that have an entropy coefficient with respect the target variable below the 10% are eliminated. After this procedure, is obtained the final reduced number of features of **16** as shown in Figure 3.7. Notice than the same results would be obtained if a threshold of 0.05, selecting the most important feature with more that 5% of importance of Information Gain.

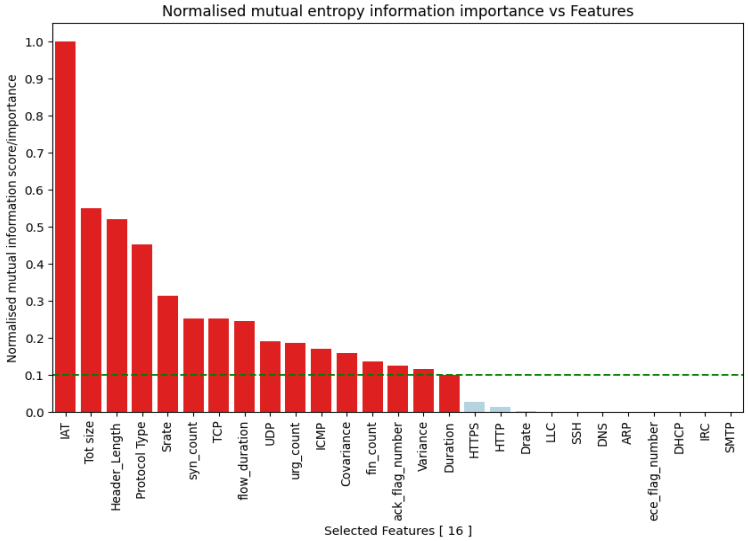


Fig. 3.7 Selected features

3.3. Model selection

As mentioned in Section 2.3.3, a selection of classification models from different model families was made:

- From linear models: Logistic Regression (LR).
- From probabilistic models: Gaussian Naïve Bayes (NB).
- From trees models: Decision Trees (DT).
- From ensemble models: Random Forests (RF).
- From the Artificial Neural Network category: Multilayer Perceptron (MLP).

3.4. Performance evaluation and metrics

For evaluating a classifier, of the most important measure is the Confusion Matrix (CM) [58] [59]. Although is primary oriented for binary classifying task, can be extrapolated to multiclassification problems. Figure 3.8 shows a typical binary confusion matrix. The purpose of the Confusion Matrix is to demonstrate how often instances of a class are classified as another class. Actual or real instances are represented by each row and predicted instances by each column. The diagonal of the matrix displays values that have been predicted accurately.

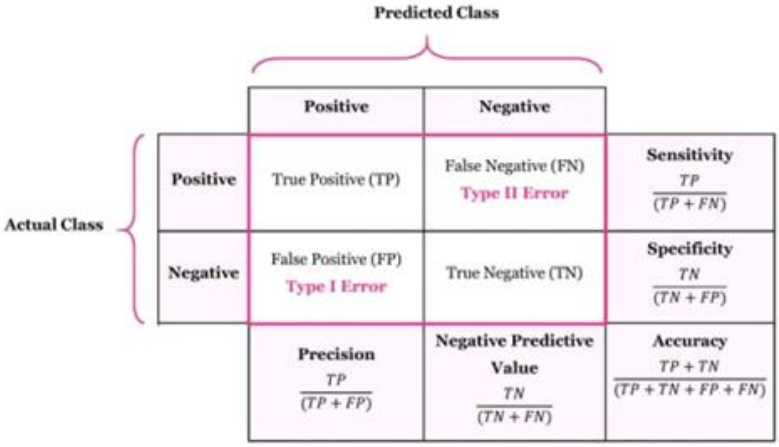


Fig. 3.8 Confusion Matrix [52]

Considering as positives, the samples belonging to a real attack, and as negatives, the samples for benign traffic, it can be found that: True positives (TP) are accurate predictions of true events like detecting a real attack sample. False positives (FP) are incorrect predictions of true events, for example detecting a benign/normal traffic as an attack “false alarm”. True negatives (TN) accurately predict false events, in this case, detecting regular traffic as such. False negatives (FN) are inaccurate predictions of false events. It is worth noting that, in this case, the model (IDS) misidentifies actual attacks as legitimate traffic, which enables access to the IoT infrastructure [58] [59].

Accuracy is the proportion of correctly predicted instances (TN, TP) out of the total number of instances evaluated and is defined in Equation (3.1) [59]. In imbalanced datasets, where one class is far more prevalent than the other, a

model that always predicts the dominant class can achieve high accuracy, yet it's essentially a poor classifier. For these reasons, in contexts with imbalanced data, it's recommended to use other metrics like recall, precision and F1-score [12].

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.1}$$

Precision/Recall Trade-off: in short terms, the precision is the accuracy of the positive predictions and is defined in Equation (3.2) [59]. Whereas recall also called sensitivity is the ratio of positive instances that are correctly detected and is defined by the Equation (3.3) [59]. It's desirable for a classifier to have both parameters high; however, this is not always achievable, unfortunately, an increase in precision often results in a decrease in recall, and vice versa [12].

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.3}$$

By this reason, it can be advantageous to merge precision and recall into an individual metric known as the F1 score (F1), particularly when seeking a straightforward method of comparing two classifiers. The F1 score defined in Equation (3.4) [59] represents the harmonic mean of precision and recall. Unlike the regular mean, which treats all values equally, the harmonic mean places greater emphasis on lower values. Consequently, the classifier will achieve a high F1 score only when both recall, and precision are high [12].

$$\text{F1} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 * \text{TP}}{2 * \text{TP} + \text{FP} + \text{FN}} \tag{3.4}$$

When dealing with imbalanced datasets and multi-class classification problems, macro metrics offers several advantages. Macro metrics give equal weight to all classes, computing the metric independently for each class and then take the average. They provide a holistic view of performance across all classes, without letting the potentially skewed distribution of one class dominate the overall evaluation metric. This helps to differentiate between good models (those that perform well across all classes) and mediocre ones (those that only perform well on the dominant classes). The Macro metrics is computed as the average of the metric of every class. For instance, in a binary classification the Precision Macro is the average of the sum of the precision of the positive class and the negative class, on the other hand, in multiclassification tasks the precision macro is obtained by the sum of individual precisions of each class divided by the number of classes.

Therefore, the target metric of the tuning of hyper-parameters of every model on the three classifications will be the F1 macro. An exhaustive search (brute force) of every combination of the most important hyper parameters of every model were done using for-loops, even though the scikit-learn library includes several functions like **GridsearchCV** [27] to find the best combination of

hyperparameters. But due to the long execution time of each optimisation, and some interruptions, causing irretrievable data loss, it has been decided to use to use for-loops and store in a table all the metrics involved with all the combinations of hyperparameters for each run.

CHAPTER 4. RESULTS AND DISCUSSION

Results and discussion are presented in five sections, the first one corresponding to the tuning of hyperparameters of every model (Logistic Regression, Decision Trees, Gaussian Naïve Bayes, Random Forest and Multilayer Perceptron) the second for the model evaluations and the rest of the sections corresponding to each classification task: binary, group, and multi classification on every model.

4.1. Hyper-parameters tuning

In machine learning, hyper-parameter tuning involves selecting optimal values for controlling the learning algorithm [60][61]. A learning algorithm estimates model parameters from a dataset and updates them throughout the learning process. Once complete, these parameters become integral to the model. On the other hand, hyperparameters are inherent to the algorithm and can't be derived from data. They influence the computation of model parameters, with different hyperparameter values yielding different model parameters for a dataset [60].

Cross-validation [61] [62] is a common technique used to evaluate a model's performance by splitting the data into multiple subsets and testing the model on each subset, helping in hyperparameter tuning and mitigating the risks of overfitting.

In the Cross-validation, the model is trained using the data from k -1 of the folds, and the remaining fold is used as the test set to evaluate performance. The final target metric is averaged from the k evaluations as shown in Figure 4.1 [61]. In this study is used a cross-validation of 10 **stratified** k folds for the tuning of hyper-parameters [63]. The term stratified as mentioned in Section 3.1, ensures that each fold maintains the same proportion of classes as the original imbalanced dataset.

Fig. 4.1 Cross-validation with 10 folds [64]

4.1.1 Logistic Regression Hyper-parameter tuning

The most important hyper-parameters in the Logistic Regression classifier are [12] [26] [27] [65] [66]:

- parameter 'C': inversely controls the strength of regularisation, with smaller values specifying stronger regularisation. Regularisation is a technique to prevent overfitting adding a penalty to the different parameters of the model to reduce their magnitude.
- The 'solver': specifies the algorithm used for optimization, such as 'lbfgs' or 'sag'. Solvers are algorithms used to optimize the cost function and find the best-fitting parameters for the model. They determine how the logistic regression model learns and adjusts itself to minimize the prediction errors.
- The 'max_iter': sets the maximum number of iterations for the solver to converge.
- The 'penalty': determines the type of regularisation applied, typically either 'l1' or 'l2', corresponding to Lasso and Ridge regularisation respectively. Lasso regularisation adds "absolute value of magnitude" of coefficient as penalty term to the loss function, whereas Ridge regularisation adds "squared magnitude" of coefficient as penalty term to the loss function.

The model of LR of Scikit-learn library comes with default hyper-parameters of Solver = 'lbfgs', Penalty = 'l2', Regularisation C=1 and max_iter = 100 [67].

Table 4.1 shows the hyper-parameters results for the Logistic Regression classifier. The first rows show each hyperparameter searched, the first column the name of the hyperparameter, the “Values” column corresponds to the range of search, the values inside this range are determined by the type of hyperparameter, and their recommended values in [65] [66], for numerical values is frequent to search below and above the default value of the model [67]. The last columns represent the value selected for that hyperparameter that achieved the best score of metric F1 macro on each of the three models (Binary, Group, Multi). The rows that follow, shows a metric comparison between the F1 macro with default parameters and the F1 score obtained with the selected hyperparameter. The last rows represent the average time of the training for a single run inside the cross-validation process with the selected hyperparameters, and finally the total execution time of the whole hypertuning process (including the training, predict and scoring time). The total number of runs of cross-validation for every classification task was 756 given by the combination of ranges of all hyper-parameters searched.

Table 4.1. Hyper-parameters search and result for LR model.

Hyperparameter	Values	Selected Values per model		
		Binary	**Group**	**Multi**
Solver	['lbfgs','sag', 'saga']	'lbfgs'	'saga'	'lbfgs'
Penalty	['l1', 'l2', 'elasticnet']	'l2'	'l1'	'l2'
Regularisation C	[0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100]	100	0.1	5
Max_iter	[5, 10, 25, 50, 100, 150, 200]	50	50	200
F1 macro with default hyper-parameters		0.780842	0.436387	0.442185
F1 macro achieved		**0.782046**	**0.438288**	**0.453426**
F1 macro increase (%)		0.15	0.43	2.54
Average training time for one run of ten, inside the cross validation (s)		11.02	189.82	1269.49
Total execution time of hypertuning for 756 cross-validations (h)		48.09	100.65	123.76

4.1.2 Decision Trees Hypertuning

For the Decision Trees classifier, the most important hyper-parameters are the following [27][28][29][68][69][70][71]:

- 'max_feature': limits the number of features to consider when searching for the best split, which can help in reducing variance and speeding up the training process.
- 'splitter': defines the strategy employed to choose the split at each node, either 'best' to select the best split or 'random' for a random split.
- 'max depth': limits the maximum depth of the tree, preventing the tree from growing too deep and potentially overfitting the data.
- 'min samples split': specifies the smallest number of samples required to make a node split, ensuring that minor data fluctuations don't create unnecessary branches.
- 'min samples leaf': sets the minimum number of samples a leaf node must have, preventing the creation of leaves with very few samples which can lead to overfitting.

The model of DT of Scikit-learn library comes with default hyper-parameters of max_feature = None, equivalent to the maximum number of features of 16, splitter = 'best', max_depth = None, that results on an unlimited depth, min samples split = 2, and min samples leaf = 1 [72].

Table 4.2 shows the same descriptions and results as Table 4.1 but for the Decision Trees classifier. The "Values" column corresponds to the range of search, the values inside this range are determined by the type of hyperparameter, and their recommended values in [69][70][71], for numerical

values is frequent to search below and above the default value of the model [72]. The total number of runs of cross-validation for every classification task was 960 given by the combination of ranges of all hyper-parameters searched.

Table 4.2. Hyper-parameters search and results for DT model.

Hyperparameter	Values	Selected Values per model		
		Binary	**Group**	**Multi**
max_feature	[4, 16]	16	16	16
splitter	[‘best’, ‘random’]	‘best’	‘best’	‘best’
max_depth	[3, 5, 8, 10, 20, 30, 40, None]	30	40	40
min_samples_split	[2, 5, 10, 20, 30]	2	2	2
min samples leaf	[1, 2, 5, 10, 20, 30, 40]	5	1	1
F1 macro with default hyper-parameters		0.958546	0.839600	0.839454
F1 macro increase (%)		**0.963659**	**0.841027**	**0.841635**
Performance increase (%)		0.53	0.17	0.26
Average training time for one run of ten, inside the cross validation (s)		16.12	20.38	36.45
Total execution time of hypertuning for 960 cross-validations (h)		14.52	19.85	19.89

4.1.3 Gaussian Naive Bayes Hypertuning

The Gaussian Naive Bayes classifier, has only one hyper-parameter: var_smoothing which adds a portion of the largest variance of all features to the variances for calculation stability, aiding in preventing zero probabilities in the model [26][27][73]. The model of GaussianNB of Scikit-learn library comes with default of var_smoothing = 1e-9 [74].

Table 4.3 shows the hyper-parameters searched for the Gaussian Naive Bayes classifier, the description of the values and columns are the same of Table 4.1. The “Values” column corresponds to the range of search, the values inside this range are determined by the type of hyperparameter, and their recommended values in [73], for this numerical value the search was done below and above the default value of the model [74].

Table 4.3. Hyper-parameters search and results for NB model

Hyperparameter	Values	Selected Values per model		
		Binary	Group	Multi
var_smoothing	[1e-12, 1e-11, 1e-10, 1e-09, 1e-08, 1e-07, 1e-06, 1e-05, 1e-04, 1e-03, 1e-02, 1e-01, 1e+00, 1e+01, 1e+02]	1	0.1	0.1
F1 macro with default hyper-parameters		0.755147	0.425963	0.398259
F1 macro achieved		**0.844691**	**0.464120**	**0.414518**
F1 macro increase (%)		11.86	8.96	4.08
Average training time for one run of ten, inside the cross validation (s)		2.02	2.17	2.70
Total execution time of hypertuning for 15 cross-validations (h)		0.15	0.17	0.41

4.1.4 Random Forest Hypertuning

For the Random Forest classifier, the most important hyper-parameters analysed were the following [12][26][27][28][65][75][76][77]:

- ‘n estimators': denotes the number of trees in the forest, with a larger number typically resulting in a more robust model at the cost of computational complexity, in the hypertuning was set to a maximum of 300.
- 'max depth': specifies the maximum depth of each tree, limiting the number of splits and thereby preventing the model from becoming too complex and potentially overfitting the data.
- 'min samples split’: sets the minimum number of samples required to make a node split, ensuring that trees don't branch out on small fluctuations or anomalies in the data.
- ‘min samples leaf’: defines the minimum number of samples a leaf node must contain, preventing the creation of leaves with very few samples which can lead to overfitting.

The model of RF of Scikit-learn library comes with default hyper-parameters of n_estimators = 100, max_depth = None, that results on an unlimited depth, min samples split = 2, and min samples leaf = 1 [78].

For the Random Forest model Table 4.4 shows the same descriptions and results as Table 4.1. The “Values” column corresponds to the range of search, the values inside this range are determined by the type of hyperparameter, and their recommended values in [65][75][76][77], for numerical values is frequent to search below and above the default value of the model [78]. The total number of runs of cross-validation for every classification task was 720 given by the combination of ranges of all hyper-parameters searched.

Table 4.4. Hyper-parameters search and results for RF model.

Hyperparameter	Values	Selected Values per model		
		Binary	**Group**	**Multi**
n_estimators	[25, 50, 100, 150, 300]	300	300	300
max_depth	[1, 2, 5, 10, 20, 30,40, None]	40	40	None/ unlimited
min_samples_split	[2, 5, 10, 20, 30, 40]	5	2	10
min_samples_leaf	[1, 2, 4]	1	2	1
F1 macro with default hyper-parameters		0.971426	0.871982	0.853270
F1 macro achieved		**0.972413**	**0.876360**	**0.855529**
F1 macro increase (%)		0.10	0.50	0.26
Average training time for one run of ten, inside the cross validation (s)		608.19	1741.48	1730.22
Total execution time of hypertuning for 720 cross-validations (h)		155.46	213.57	226.99

4.1.5 Multilayer Perceptron Hypertuning

Lastly, the hypertuning of the Multilayer Perceptron, which belongs to the Artificial Neural Networks family, involves several hyperparameters to consider. The study utilised just a single hidden layer approach, including more hidden layers falls into the category of deep learning [11] [12]. There is no consensus or recipe to find the best number of neurons for each layer, some authors suggest a minimum and maximum number according to the number of input features and the number of outputs, but in general, only through experimentation and evaluation can one determine the optimal number of neurons in an MLP. Other parameters searched are [26] [27] [28] [79] [80]:

- The 'solver' parameter determines the algorithm used for weight optimization.
- The 'activation' parameter sets the activation function for the neurons of the hidden layer.
- The 'alpha' parameter represents the L2 penalty (regularisation term) which combats overfitting by constraining the magnitude of the weights in the model.
- The 'learning_rate_init' parameter defines the initial learning rate for weight updates, controlling the step size during optimization.

The model of MLP of Scikit-learn library comes with default hyper-parameters of hidden_layer_sizes = (100,) defining 100 neurons in this layer, activation = ‘relu’, solver = ‘adam’, alpha = 0.0001, learning_rate_init = 0.001 [81].

The descriptions and values in Table 4.5 share the same descriptions as the previous tables, except for the first two rows. The first row specifies the range of the number of neurons in the hidden_layer to be searched by each model, the second row specify the value found. For the binary classification is set a range

from 2 neurons to 16 neurons [(2,)…(16,)], for group classification a range of [(8,)…(16,)] is used, and for the multiclassification a range from [(16,)...(34,)] neurons, all with step size = 1. Thus, for the binary task the number of runs was of 810 cross-validation, in group classification the combination was of 486 and for multiclassification a number of 1026 cross-validation were done.

Table 4.5. Hyper-parameters search and results for MLP model.

Hyperparameter	Values	Selected Values per model		
		Binary	**Group**	**Multi**
hidden_layer_size searched		[(2,).(16,)]	[(8,).(16,)]	[(16,).(34,)]
hidden_layer_size selected		13	16	33
solver	[‘adam’, ‘sgd’]	‘adam’	‘adam’	‘adam’
activation	[‘logistic’, ‘relu’, ‘tanh’]	‘tanh’	‘logistic’	‘tanh’
alpha	[0.0001, 0.001, 0.01]	0.0001	0.0001	0.0001
learning_rate_init	[0.0001, 0.001, 0.01]	0.001	0.001	0.001
F1 macro with default hyper-parameters		0.930927	0.665187	0.658210
F1 macro achieved		**0.928152**	**0.644880**	**0.652424**
F1 macro increase (%)		-0.29	-3.05	-0.87
Average training time for one run of ten, inside the cross validation (s)		120.37	155.85	189.31
Total execution time of hypertuning (h)		96.26	117,20	293,42

It can be observed that the F1 macro resulting of the best combination of the hyperparameter found of every model, is slightly below the default parameters of the MLPClassifier, therefore the hypertuning procedure couldn’t find best combination of hyper-parameters other that the default parameters. Thus, the best hyperparameters to be used on the final evaluation of the MLPClassifier were the default hyperparameters.

4.2. Model evaluation

After obtaining the best hyper parameters for each model for the three classification tasks, every model is finally trained on the whole training set and evaluated on the test set of unseen data created for this purpose.

Table 4.6 shows the F1 macro metric comparison after the hypertuning and the final F1 score obtained in the prediction with the test set, on most of the cases, there is an increment of the performance on the unseen data. Negatives values of the differences are in acceptable ranges, because is expected that on unseen data, the performance of the models will be slightly lower than the achieved with the trained cross validation hypertuning procedure.

Table 4.6. Comparison for the evaluation test of the F1 versus the achieved in the hypertuning.

Macro metric	Logistic regression	Decision Trees	Gaussian Naïve Bayes	Random Forest	Multilayer Perceptron
Binary Classifiers					
F1 tuning	0.782046	0.963659	0.844691	0.972413	0.930927
F1 test	**0.782126**	**0.964580**	**0.845388**	**0.973025**	**0.933232**
Difference (%)	0.01	0.10	0.08	0.06	0.25
Group Classifiers					
F1 tuning	0.438288	0.841027	0.464120	0.876360	0.665187
F1 test	**0.433896**	**0.847993**	**0.465141**	**0.888024**	**0.665175**
Difference (%)	-1.00	0.83	0.22	1.33	0.00
Multi Classifiers					
F1 tuning	0.453426	0.841635	0.414518	0.855529	0.658210
F1 test	**0.449604**	**0.846256**	**0.414240**	**0.865478**	**0.662976**
Difference (%)	-0.84	0.55	-0.07	1.16	0.72

4.3. Binary classification

The final evaluation/validation on the test set for every classifier yields the following macro metrics and accuracy and is shown in Table 4.7. It also included the times involved in the whole training and prediction using the training and test set respectively. The last parameter of “prediction time per sample”, is calculated by the ratio of the prediction time and the number of samples of the test set (583582); and gives an approximate idea of the theoretical delay of the prediction of a single packet, after all, classifier models should be part of the core of an IDS.

Table 4.7. Metrics and Results from the binary classification by model

Macro metric	Logistic regression	Decision Trees	Gaussian Naïve Bayes	Random Forest	Multilayer Perceptron
F1 score	0.782126	0.964580	0.845388	0.973025	0.933232
Precision	0.705891	0.946219	0.772325	0.963447	0.922242
Recall	0.977292	0.984613	0.980954	0.983029	0.944844
Accuracy	0.966551	0.996605	0.980298	0.997469	0.993701
training time (s)	8.36	14.82	2.95	447.68	266.86
prediction time (s)	1.26	0.04	0.56	5.92	2.58
prediction time per sample (µs)	2.16	0.06	0.97	10.14	4.43

The comparison between the models is shown graphically on Figure 4.2, it should be noted that models of Decision Trees and Random Forest outperforms the

other classifiers in Precision, followed by the MLPClassifier, being Logistic Regression the worst with a value below 80%. Gaussian Naïve Bayes shows a discrete performance above 80%. This leads that Decision Trees and Random Forest can identify most of legitimate traffic without mistakenly flagging it as malicious.

Respecting the recall or sensitivity, the MLPClassifier shows a minor performance below the other classifiers, being Decision Trees, Gaussian Naïve Bayes and Random Forest, the best algorithms indicating that the IDS should miss fewer real attacks. The Logistic Regression performance is slightly lower than the others, but in general all models are around the same region of values.

Concerning the training and prediction times, there are huge differences among the classifiers, being the faster training algorithm the Gaussian Naïve Bayes, and the worsts the Random Forest, followed by the MLPClassifier. Random Forest speed is affected by the number of estimators that increase the learning time, and the MLPclassifier by the number of neurons among other parameters.

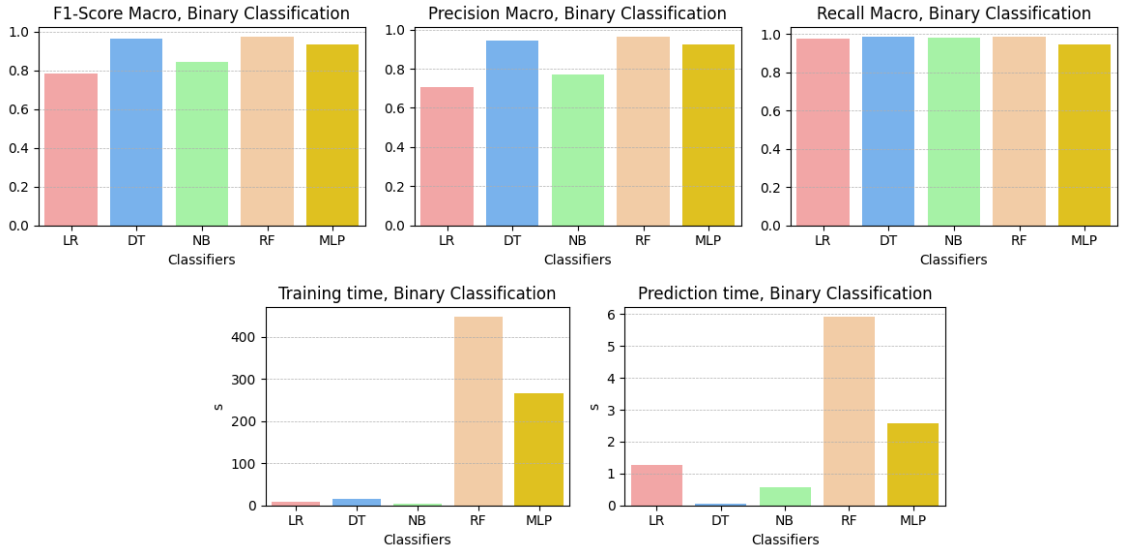


Fig. 4.2 Comparison of metrics and results of binary classifiers

A detailed analysis for each classifier is done through the confusion matrix and the classification report. Figure 4.3 and Table 4.8 shows the normalised confusion matrix and classification report of the Binary Logistic Regression model. The normalised confusion matrix shows information on the accuracy of a classification model's predictions in relative terms, instead of absolute values. When the confusion matrix is normalised, each value in the matrix is divided by the sum of its corresponding row. The classification report shows the Precision and Recall of each class individually, where the average macro is the average of every metric of each class, the support column shows the count of every class, being the last value the total count of samples of the test set.

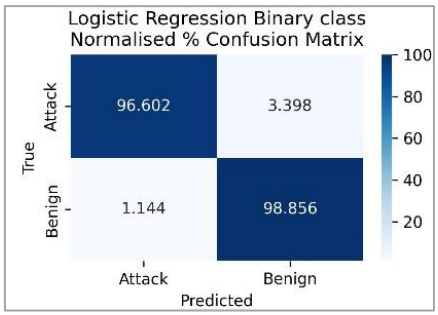


Fig. 4.3 Confusion matrix of binary Logistic Regression model

From the confusion matrix is shown that LR can classify precisely with more than 96% both Attack and Normal samples, having a miss rate of 3.39% of True attacks been undetected, and 1.14% of false alarms. But from the individual report, the precision of the “benign” traffic is below the 50% degrading the overall precision of the model.

Table 4.8. Classification report by classes of binary Logistic Regression model

	precision	**recall**	**F1 score**	**support**
Attack	0.9997	0.9660	0.9826	569854
Benign	0.4121	0.9886	0.5817	13728
macro average	0.7059	0.9773	0.7821	583582

In Figure 4.4 and Table 4.9 are shown respectively, the Normalised Confusion Matrix and classification report for binary Decision Tree algorithm.

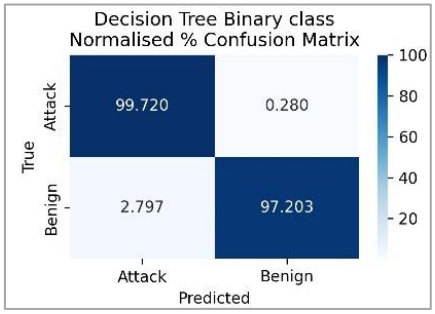


Fig. 4.4 Confusion matrix of binary Decision Tree model

Table 4.9. Classification report by classes of binary Decision Tree model

	precision	**recall**	**F1 score**	**support**
Attack	0.9993	0.9972	0.9983	569854
Benign	0.8931	0.9720	0.9309	13728
macro average	0.9462	0.9846	0.9646	583582

The Decision Tree model shows better performances than the Logistic regression, except in the case of false positive being twice of the LR model. Again, the precision of the “Benign” class reduces the overall metric consequence of being the minority class. a good 0.28% of attacks overrun the classifier and got undetected.

For the Bayes Naïve model, Figure 4.5 and Table 4.10 shows respectively, the Normalised Confusion Matrix and Classification report.

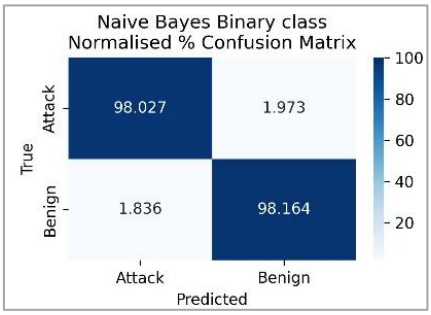


Fig. 4.5 Confusion matrix of binary Naive Bayes model

The Binary Naive Model present results very similar to the Logistic Regression being better in the precision of the Benign traffic. But affecting the overall metric.

Table 4.10. Classification report by classes of binary Naive Bayes model

	precision	**recall**	**F1 score**	**support**
Attack	0.9995	0.9803	0.9898	569854
Benign	0.5451	0.9816	0.7010	13728
macro average	0.7723	0.9810	0.8454	583582

Figure 4.6 shows the Normalised Confusion Matrix, and Table 4.11 shows the classification report for binary Random Forest algorithm.

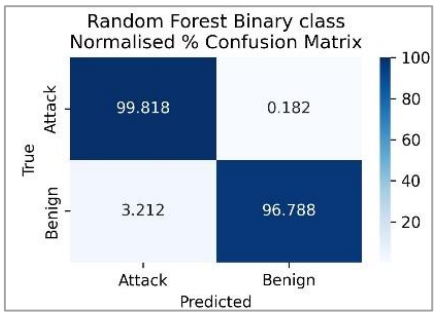


Fig. 4.6 Confusion matrix of binary Random Forest model

The Random Forest shows excellent detection of attacks 99.82% and a very good number of False negatives of 0.182, at expense to have a bit higher false positive

score of 3.21%. It also presents a good precision even in the minority class of the “Benign” class.

Table 4.11. Classification report by classes of binary Random Forest model

	precision	**recall**	**F1 score**	**support**
Attack	0.9992	0.9982	0.9987	569854
Benign	0.9277	0.9679	0.9473	13728
macro average	0.9634	0.9830	0.9730	583582

Finally, the binary MLPClassifier results are shown in Figure 4.7 for the Confusion Matrix and Table 4.12 for Classification report by classes.

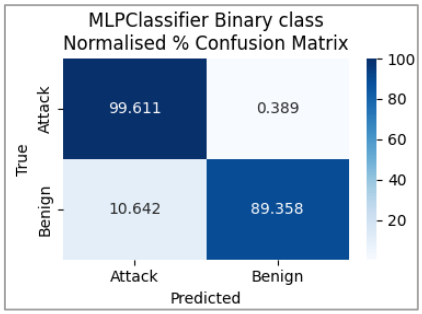


Fig. 4.7 Confusion matrix of binary MLPClassifier model

The MLPClassifier has an average precision and recall for the “Benign” class that reduces the average of both metrics. Note to mention that had the worst False Positive with 10.64%.

Table 4.12. Classification report by classes of binary MLPClassifier model

	precision	**recall**	**F1 score**	**support**
Attack	0.9974	0.9961	0.9968	569854
Benign	0.8471	0.8936	0.8697	13728
macro average	0.9222	0.9448	0.9332	583582

A summary for the binary classification is as follows: the best Model is the Random Forest at expense of higher computational cost, followed by the Decision Trees Algorithm which present a good time of training and prediction. MLPClassifier had worst False Positive with 10.64%. Another important metric from the confusion matrix and crucial for an IDS is the miss rate, where attack traffic is detected as “normal”, compromising the security of the system, letting pass to the infrastructure. In this case the Logistic Regression had the worst value

of 3.40%, then 3.40% of the anomalous traffic evade this model. The Gaussian Naïve Bayes has this parameter near 2%.

4.4. Group classification

As with the binary classification results, Table 4.13 shows the relevant metrics and results for every model analysed. Here, all metrics begins to suffer in performance in comparison with the binary classification task, due to the more complex classification involving the identification of 8 classes of attack categories.

Table 4.13. Metrics and Results from the group classification by model

Macro metric	Logistic regression	Decision Trees	Gaussian Naïve Bayes	Random Forest	Multilayer Perceptron
F1 score	0.433896	0.847993	0.465141	0.888024	0.665175
Precision	0.460550	0.846068	0.521091	0.952584	0.704064
Recall	0.581369	0.850187	0.501922	0.850432	0.652549
Accuracy	0.579727	0.994791	0.717877	0.996045	0.990118
training time (s)	186.92	18.89	5.57	820.30	218.03
prediction time (s)	1.76	0.06	2.48	9.80	1.28
prediction time per sample (µs)	3.02	0.10	4.26	16.79	2.19

The comparison between the models is shown graphically on Figure 4.8, where Random Forest outperforms the other classifiers in the Precision with a score of 95%. The Decision Trees model follows, lagging by 10%. The MLPClassifier has a discrete performance of 70%, being again Logistic Regression the worst model with a value below 50%. Gaussian Naïve Bayes is merely above 50%. This leads that Random Forest can identify most of legitimate traffic, around 95% without mistakenly flagging it as malicious.

Regarding recall or sensitivity, both the Random Forest and Decision Trees achieve near identical values of around 85%, with the Random Forest performing marginally better. The MLPClassifier's performance is reduced at 65%, whilst Gaussian Naïve Bayes and Logistic Regression fare the worst, recording 50% and 58% respectively. Thus, the Decision Trees and Random Forest models emerge as the superior algorithms, suggesting that an IDS would likely miss fewer genuine attacks when using them.

In terms of timing, for both metrics, the Random Forest once again proves to be the least efficient model by a significant margin. The faster model in the training process is the Gaussian Naïve Bayes with around 5 seconds with the whole training set of 2334328 samples. And the best model predicting the test set is again the Decision Trees algorithm with only 0.06 seconds.

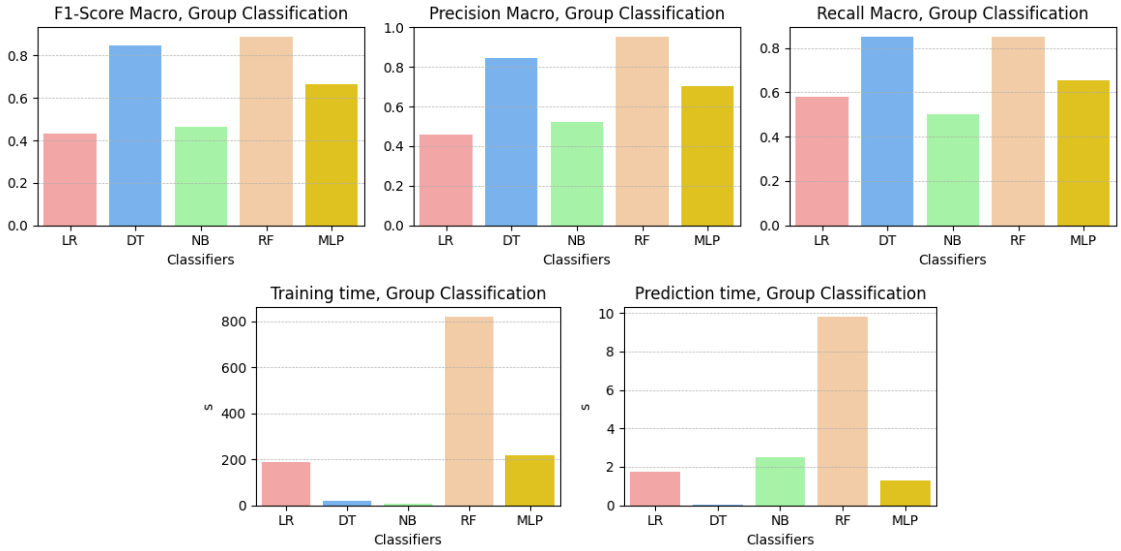


Fig. 4.8 Comparison of metrics and results of Group classifiers

A detailed analysis for each classifier is done through the confusion matrix and the classification report. Figure 4.9 and Table 4.14 shows the normalised confusion matrix and classification report of the Group Logistic Regression model.

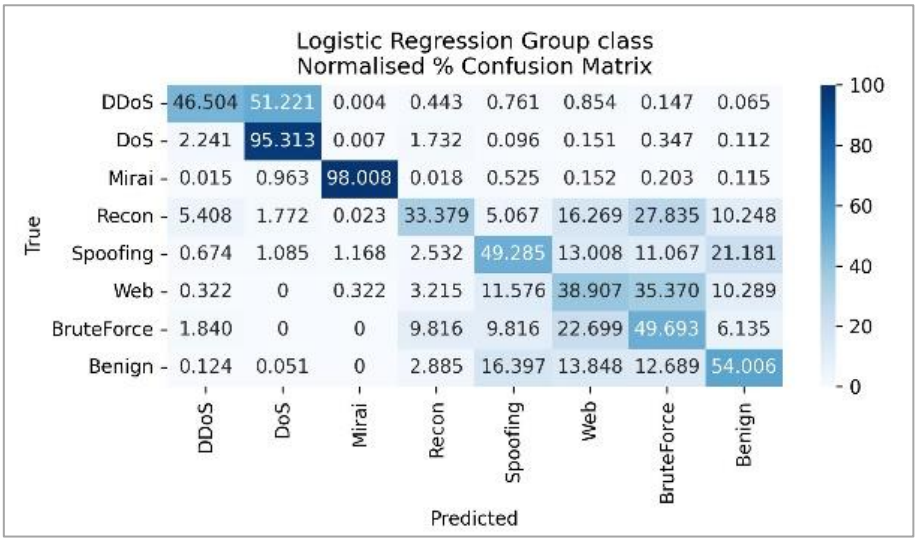


Fig. 4.9 Confusion matrix of group Logistic Regression model

In the Logistic Regression case, the model performs well above 95% detecting “DoS” and “Mirai” attacks, while the other groups of categories suffer. For example, the 51% of “DDoS” traffic are detected as “DoS”, a value even higher that its own score, this is why, all “DDoS” and “DoS” attacks share many attributes, and the model is unable to distinguish between them. “Recon” attacks are recognised as five distinct categories with more than a 5%, with a special note of a 10% of attacks detected as “benign” traffic. “Spoofing” attacks on the other hand, is detected with a score of more than 11% as “Web”, “Bruteforce” and again as “benign traffic” with a dramatic score of 21%. “Web” attacks detections

are distributed on “Spoofing” with 12%, “Bruteforce” with 35% and “Benign” traffic with a 10%. “Bruteforce” attacks are detected with almost 50%, while is misclassified with more than 6% as “Recon”, “Spoofing”, “Web”, and “Benign” category. All traffic that consists of true attacks but is incorrectly predicted as "Benign" eludes the model and IDS, entering or exiting the IoT infrastructure. This is why a special attention should be made to the last column containing the label “benign”. Is worth to mention, the fact that “benign” traffic is only recognised at the 54% and rising false alarms of other categories of attacks. If one were to comment on this model, it could be said that it is vulnerable to “Recon”, “Spoofing”, “Web”, and “Bruteforce” attacks, while it performs well in detecting “DoS” and “Mirai botnet” attacks and have a false alarm ratio of around 46%.

The numbers in the classification report contrasts with the above, note the very low values of Precision for the affected attack categories and mediocre values for the recall. This model is unable to perform well on the minority classes, although in the “DoS” category also suffer.

Table 4.14. Classification report by classes of group Logistic Regression model

	precision	**recall**	F**1 score**	**support**
DDoS	0.9871	0.4650	0.6322	424829
DoS	0.3065	0.9531	0.4639	101140
Mirai	0.9969	0.9801	0.9884	32929
Recon	0.2583	0.3338	0.2912	4401
Spoofing	0.3321	0.4928	0.3968	6081
Web	0.0164	0.3891	0.0314	311
BruteForce	0.0166	0.4969	0.0322	163
Benign	0.7704	0.5401	0.6350	13728
macro average	0.4606	0.5814	0.4339	583582

Decision Trees results are shown in Figure 4.10 and Table 4.15 for the confusion matrix and the classification report respectively. This Model has excellent detection rate of “DDoS”, “DoS” and “Mirai” attacks with more than 99.95%, with a decent score in the “Recon”, “Spoofing” and “Benign” categories. Decision Trees is vulnerable to category of “Recon”, “Spoofing”, “Web” and “Bruteforce”, being the metrics on these categories affecting the overall metric score. Nevertheless, is far better than Logistic Regression Model even in the minority classes.

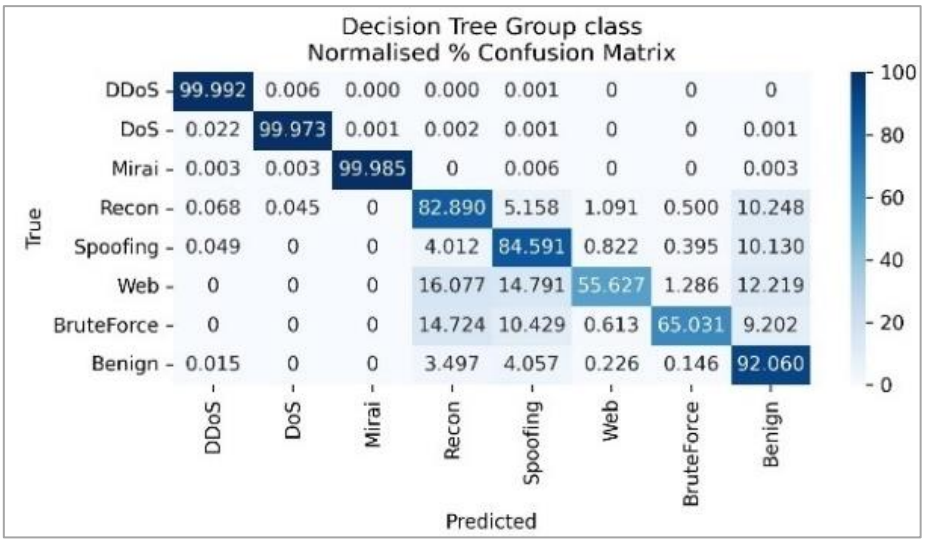


Fig. 4.10 Confusion matrix of group Decision Trees model

Table 4.15. Classification report by classes of group Decision Trees model

	precision	**recall**	F**1 score**	**support**
DDoS	0.9999	0.9999	0.9999	424829
DoS	0.9997	0.9997	0.9997	101140
Mirai	0.9999	0.9998	0.9999	32929
Recon	0.8200	0.8289	0.8244	4401
Spoofing	0.8573	0.8459	0.8516	6081
Web	0.5710	0.5563	0.5635	311
BruteForce	0.6023	0.6503	0.6254	163
Benign	0.9185	0.9206	0.9195	13728
macro average	0.8461	0.8502	0.8480	583582

For Gaussian Naive Bayes, Figure 4.11 and Table 4.16 shows the normalised confusion matrix and classification report respectively. The Gaussian Naive Bayes only perform very well on “Mirai botnet” attack with a 98%, on “Spoofing” and “Bruteforce” attacks the model performs very badly, and the worst behaviour of the model, is having a lot of missed rates identified as “benign traffic” with more than 40% on “Recon”, “Spoofing”, “Web” and “Bruteforce” categories. This model is very vulnerable to these categories of attacks.

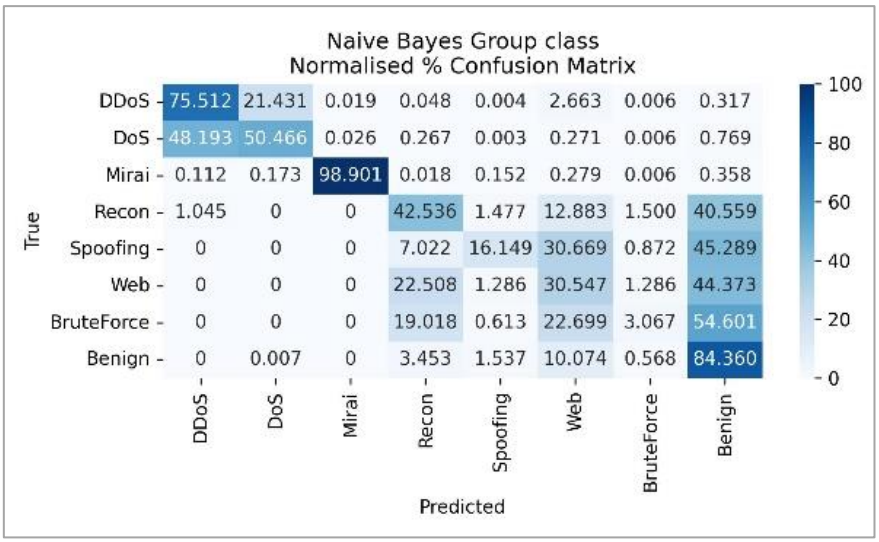


Fig. 4.11 Confusion matrix of group Naive Bayes model

Table 4.16. Classification report by classes of group Naive Bayes model

	precision	**recall**	**F1 score**	**support**
DDoS	0.8679	0.7551	0.8076	424829
DoS	0.3591	0.5047	0.4196	101140
Mirai	0.9968	0.9890	0.9929	32929
Recon	0.5578	0.4254	0.4827	4401
Spoofing	0.7372	0.1615	0.2649	6081
Web	0.0061	0.3055	0.0119	311
BruteForce	0.0208	0.0307	0.0248	163
Benign	0.6230	0.8436	0.7167	13728
macro average	0.5211	0.5019	0.4651	583582

For the model Random Forest Classifier, the Confusion matrix is shown on Figure 4.12 and the Classification report of classes is shown in Table 4.17.

The Random Forest model has excellent detection rate of “DDoS”, “DoS” and “Mirai” attacks with more than 99.95%, and a very good score of 97% detecting the normal traffic as such, leading to a low false alarm rate of 3%. On the categories of “Recon” and “Spoofing” has a decent performance with more than 85%, whereas it suffers in the “Web” and “Bruteforce” categories, with a score of 49% and 64% respectively. Please take note of the values that are zero outside the diagonal, indicating that the model is performing better because it not detecting other true categories as this one, an ideal classifier would have all values outside the diagonal in zero. Despite the good metrics in several categories, it has some Miss rate of classes belonging to “Recon”, “Spoofing”, “Web” and “Bruteforce” attacks detected as “benign traffic”.

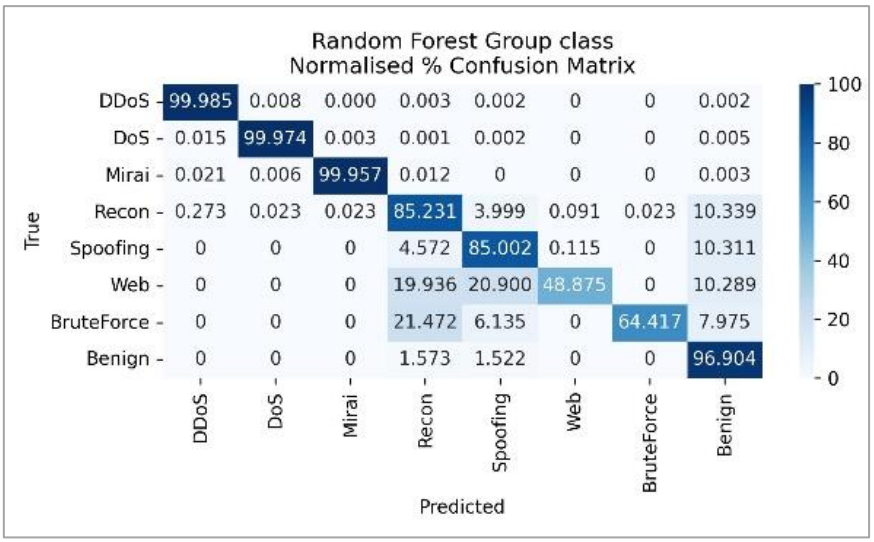


Fig. 4.12 Confusion matrix of group Random Forest model

Table 4.17. Classification report by classes of group Random Forest model

	precision	**recall**	F**1 score**	**support**
DDoS	0.9999	0.9998	0.9999	424829
DoS	0.9996	0.9997	0.9997	101140
Mirai	0.9998	0.9996	0.9997	32929
Recon	0.8603	0.8523	0.8563	4401
Spoofing	0.9168	0.8500	0.8822	6081
Web	0.9325	0.4887	0.6414	311
BruteForce	0.9906	0.6442	0.7807	163
Benign	0.9211	0.9690	0.9444	13728
macro average	0.9526	0.8504	0.8880	583582

And finally, for the model MLPClassifier Figure 4.13 and Table 4.18 shows the Confusion matrix and the Classification report of classes.

The MLPClassifier as the Decision Trees and Random Forest models perform very well on the “DDoS”, “DoS” and “Mirai” categories with more than 99.80% of score. Has a decent 93.74% detecting normal traffic as such, but surprisingly it cannot detect the “Bruteforce” category, only a spurious misclassification from the “Spoofing” category. Note the zero in both precision and recall of the report, the minority class. The “Web” category also suffers with only a 5.8% score of detection. Besides that, the models detect negatively attack traffic from the categories of “Recon”, “Spoofing”, “Web” and “Bruteforce” as “benign”.

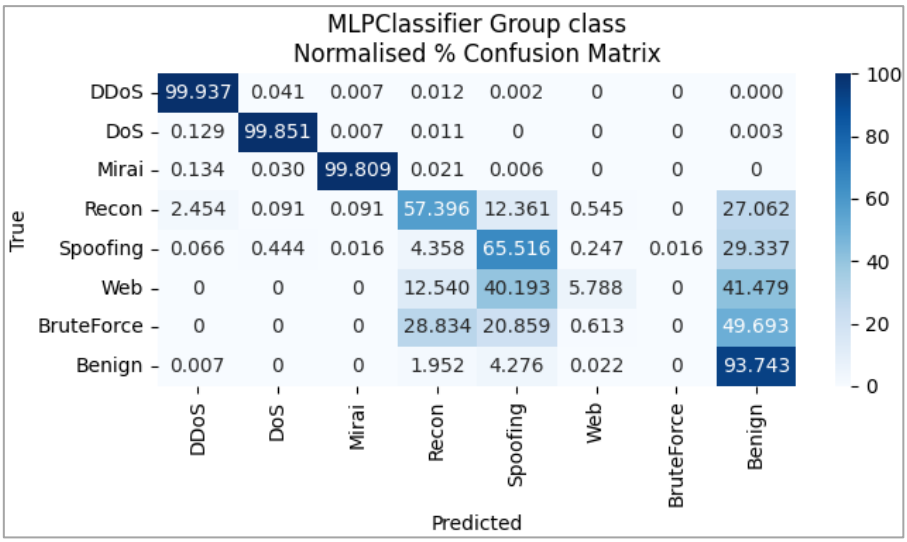


Fig. 4.13 Confusion matrix of group MLPClassifier model

Table 4.18. Classification report by classes of group MLPClassifier model

	precision	**recall**	**F1 score**	**support**
DDoS	0.9993	0.9994	0.9993	424829
DoS	0.9979	0.9985	0.9982	101140
Mirai	0.9987	0.9981	0.9984	32929
Recon	0.7862	0.5740	0.6635	4401
Spoofing	0.7540	0.6552	0.7011	6081
Web	0.2951	0.0579	0.0968	311
BruteForce	0.0000	0.0000	0.0000	163
Benign	0.8014	0.9374	0.8641	13728
macro average	0.7041	0.6525	0.6652	583582

As summary, it can be said that every model suffers classifying the minority classes, due to the imbalance of the dataset. However, models such as Decision Trees and Random Forest have proven to be quite robust, achieving nearly acceptable scores. Random Forest model is the overall winner for the group classification task, but again at a higher computational cost, Decision Trees performs slightly below but has a good training time and the best predict time. The worst model classifying categories of attack is the Logistic Regression.

4.5. Multi classification

The multiclassification is the most demanding task from the model, it must classify 34 individual classes of attacks. Therefore, all metrics begins to suffer in performance in comparison with the binary classification task. Values of these metrics are shown in Table 4.19.

Table 4.19. Metrics and Results from the multi classification by model

Macro metric	Logistic regression	Decision Trees	Gaussian Naïve Bayes	Random Forest	Multilayer Perceptron
F1 score	0.449604	0.846256	0.414240	0.865478	0.662976
Precision	0.464502	0.844037	0.481019	0.916407	0.725428
Recall	0.520232	0.850472	0.482601	0.839369	0.657903
Accuracy	0.777056	0.993178	0.764043	0.994484	0.985382
training time (s)	452.31	32.24	6.31	1297.71	1148.80
prediction time (s)	2.29	0.10	4.15	31.76	4.09
prediction time per sample (µs)	3.92	0.18	7.11	54.42	7.01

The comparison between the models is shown graphically on Figure 4.14, where Random Forest outperforms the other classifiers in the Precision with a score of 91%. The Decision Trees model follows, lagging by 7%. The MLPClassifier shows a discrete performance of 72%, being one more time the Logistic Regression the worst model with a value below 50% followed by the Gaussian Naïve Bayes that slightly better in a 2%. This leads that Random Forest can identify most of legitimate traffic, around 91% without mistakenly flagging it as malicious.

Regarding recall or sensitivity, Decision Trees achieved the best value of 85%, while is seconded by the Random Forest with a near value around 84%. The MLPClassifier's performance is reduced at 65%, while Gaussian Naïve Bayes and Logistic Regression achieves worst, recording 50% and 58% respectively.

In terms of timing, for both metrics, the Random Forest once again proves to be the least efficient model by a significant margin, but now has MLPClassifier as a closer competitor regarding the slowest models. The faster model in the training process is the Gaussian Naïve Bayes with around 6 seconds with the whole training set of 2334328 samples. And finally, Decision Trees algorithm emerges again as the fastest model predicting the test set the with only 0.10 seconds.

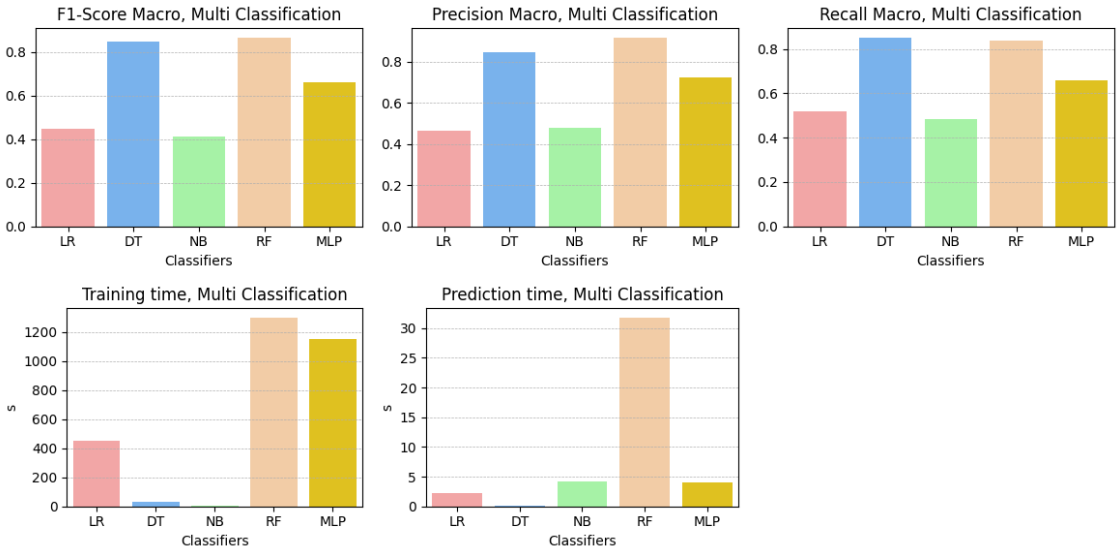


Fig. 4.14 Comparison of metrics and results of Multi classifiers

As in the binary and group classification models, a detailed analysis must be conducted using the confusion matrix of dimension [34x34], that are shown from Figure B.1 to Figure B.5 of the Annex B with the corresponding classification report for multiclass evaluation from Table B.1 to Table B.5 of the same Annex.

Beginning the analysis of the confusion matrix shown in Figure B.1 and classification report on Table B.1 of the Annex of the worst model, the Logistic Regression. This algorithm is able to detect many DDoS attacks, from “DDoS-RSTFINFlood” to “DDoS-ICMPFragmentation”, achieving detection scores from 80 to 99,7%, it can detect with a score of 99% attacks of “Mirai-udpplain”, but in general the model suffers on detecting other individual attacks being the “Recon-OSScan” and “Recon-PortScan” the worst attack detected below 1%. The algorithm detecting “normal” traffic has a poor metric of 39% provoking a rise of the false alarms. Some attacks such as “Recon-PingSweep”, “Recon-OSScan”, “DNS_Spoofing” and “MITM-ArpSpoofing” are detected with more than 10% as benign traffic, leading to a security breach, as this abnormal traffic passed undetected. The model misclassified several individual attacks as other types of attacks, from the point of view of security, a positive attack detection is made, but loose the granularity of been able to detect and correctly classified an attack. Resuming, Logistic Regression can only detect 9 types of attack of 34 including normal traffic.

Analysing the confusion matrix of the Random Forest algorithm in Figure B.4 and Classification Report in Table B.4 of the Annex C, the model is able to detect 19 types of attack of 34 classes with a precision of more than 99%, being the majority of the “DDoS”, “DoS” and “Mirai” categories. It can also detect 3 types of attack with more than 90% rate, including the “normal traffic” with 96%. The worst detection goes for the “Recon-PingSweep” with only 29% of score with a 46% to be detected as the “Recon-OSScan” attack. Concerning the missing rates, the model is vulnerable to the “Recon-OSScan” with a 21% to be detected as “normal” traffic. Another attack with a miss rate of 14% is the “Recon-PingSweep” and “Recon-PortScan” with 9%. It’s also vulnerable with a score of 10% to

“DNS_Spoofing” and “SqlInjection” with 11%. It worth to mention that, despite this model of Random Forest is very robust against several types of attack, is vulnerable to the family of Reconnaissance Attacks. Considering false alarms rate the model achieves a reasonable 4.5% score. Just to mention, that values of 1.0000 in the Classification report are shown by the rounded decimal ciphers, but they are all below 1.0000.

Summarising the other three models as: Decision Trees shown in Figure B.2 and Table B.2 of the Annex, performs very similar to the Random Forest achieving high scores of 99% in 20 types of attacks. The normal attack is detected with a 92% score, lower than the RF model, but more important are the vulnerabilities that cannot handle very well, it has above 10% of miss rate in several types of attacks, including the ones of the RF model of the Reconnaissance category, from the Spoofing family and the Web category. The False alarms rate falls around 8%.

In Figure B.3 and Table B.3 of the Annex is shown the confusion matrix of the Gaussian Naïve Bayes, this algorithm performs well with more than 90% in 11 types of attacks, has several attacks detected below 1%, and is totally unable to detect “BrowserHijacking” with a zero score. With a poor 65% the normal traffic is detected, leading to a high false alarm rate of 35%. But all worries rely on the vast number of attacks that go unnoticed from, with a ranging score of 10% up to 32% falling into the “Reconnaissance”, “Spoofing” and Web category.

Figure B.5 and Table B.5 of the Annex, shows the confusion matrix of the MLPClassifier, this model performs well detecting several types of attack of the majority classes, the “DDoS”, “DoS” and “Mirai” categories, achieving scores above 98%, has a good score of 95% detecting the “normal traffic” resulting in a low 5% of false alarms. The model is totally blind to 6 types of attacks with zero score, these attacks are “Recon-PingSweep”, and from most of the “Web” category, with this behaviour the algorithm adds those samples not detected to the “normal traffic” category, increasing the miss rate, enhancing security breach. Noticed that these attacks range from 13% up to 63%, an inacceptable situation. Thus, MLPClassifier is vulnerable to the “Reconnaissance”, “Spoofing”, “Web” and “BruteForce” categories.

CHAPTER 5. CONCLUSIONS

The main objective of this thesis was the evaluation of Machine Learning Models to be used as Intrusion Detection Systems (IDS) in the IoT infrastructure. A selection of five classifier from different families of models such as Logistic Regression from the Linear models, Decision Trees from the Trees models, Naïve Bayes from the probabilistic models, Random Forests from the ensemble models, and Multi-Layer Perceptron from the Artificial Neural Networks, for the study.

Using supervised techniques, all models were trained on a chosen public IoT attacks dataset. Moreover, this dataset allowed the implementation of three types of classification tasks. Binary classification, the simple one, to detect and identified if a sample of traffic is being part of an attack or not. Group classification, it's nothing more than a multiclassification task to proper classified samples into eight categories of attacks. And a multiclassification of individual 33 types of attacks. Typical tasks of exploration and features selection and reduction were done in the selected dataset. All models were tuned with cross-validation of 10 k-folds to find the best hyperparameter using the F1 macro as the scoring metric. Finally, the resulting model was evaluated in the test set to validate the model on unseen data. All results were analysed regarding performance metrics, execution times and detection capacity of every algorithm through the confusion matrix.

As expected, all models on the binary classification performs relatively well to classifying attacks, the worst model in this category was the Logistic regression from the linear models and the best, was the Random Forest from the ensemble family. The Random Forest achieved the highest score but at expense of computational power and time, including the time of tuning of hyperparameters, the training and prediction time. On the other hand, the fastest model was the Gaussian Naïve Bayes, requiring only one hyperparameter to be tuned.

If an IDS had already the trained model running, there is no need to hyper tuning or training in the same device, unless an incremental learning approach is done on the fly. Then, an IDS should look for the fastest prediction model, having the Decision Trees algorithm the fastest prediction time. With the environment used in this project, the Decision Trees reach a theoretical prediction time of 0.06 µs per sample for the binary task, being an important parameter to be taken into consideration to use a model inside the core on an IDS. Perhaps in a "slow" IoT environment, this requirement is not necessary, and is always depending on the application.

In the group classification the objective was classify a sample to be part of 8 categories of attacks, considering the normal traffic as the last category. Here the classification was more challenging, observing that some models had misclassified anomaly attacks as normal traffic, phenomena identified as miss rate, and in an IDS this metric is important, trying to lower this value increase other metric that is known as the False alarms or False Positive rate, that can cause what is known as "alarm-fatigue". All models are found to be somehow

vulnerable to some types of “Recon”, “Spoofing”, “Web” and “Bruteforce” attack categories, these attacks belong to the minority classes of the imbalanced dataset. The most vulnerable model to this attack category is the Gaussian Naïve Bayes, whereas Random Forest is less affected by the same categories, it has the best average precision and recall metrics for every class. The MLPClassifier in the only model to completely fail to detect the Bruteforce category of attacks.

The time involved in the group classification increased with respect the binary classification, and results in the same models to be the fastest tuned and trained with Gaussian Naïve Bayes as the first on the list, again Random Forest is the slowest model for hypertuning, training and prediction times. Decision Trees offers the trade-off of good detection metrics with acceptable timing.

Identifying each individual attack between 33 types of attacks concern to the multiclassification task. All models suffer a degradation of their metrics, but Random Forest and Decision Trees maintain acceptable values being the Random Forest the best classifier. Despite Random Forest is the best model, is not perfect and is vulnerable to the family of Reconnaissance Attacks. The MLPClassifier model is found to be totally blind to 6 types of attacks with zero detection score, these attacks are “Recon-PingSweep”, and from most of the “Web” category. Decision Trees maintain high detection rates of many types of attacks, whereas Gaussian Naïve Bayes failed in the detection of “BrowserHijacking” attack and has many miss rates.

Timing in multi-classification tasks was consistent with previous classifications, with the Gaussian Naïve Bayes being the most efficient model and the Random Forest the slowest. Notice that Decision Trees has the best time in the prediction process.

With regards to the group categories and individual classification of attacks the results are not so good as with the binary classification, but as mentioned in section 2.3 and recalling Figure 2.5, the primary objective of an IDS is to binary detect, attack or not attack, then and depending on the application, resources assigned, a classification can be done in order to track and record the types of attack being held to the infrastructure. IoT gateways with more computationally resources are main target to install an IDS or an IDS being part of dedicated hardware running inside the network. The speed of processing is also a concern, when dealing with detection algorithms of Machine Learning.

Based on the presented findings, it can be inferred that the application of machine learning algorithms is effective to be used as Intrusion detection system (IDS) on the IoT infrastructure. These results, recommends the Random Forest to be the most robust model when the speed is not an issue, whereas the application of Decision Trees algorithms is intended for faster requirements and offering good performance.

5.1. Future lines of development and research

The field of application of artificial intelligence and machine learning in the intrusion detection is in constant evolution. In the specific field of IoT infrastructure

application of IDS is critical to preserve the integrity and continuity of the network and IoT devices. Some future lines of research are listed below:

- Based on the obtained results, further improvement is required to strength the models, for example: the MLPClassifier.
- Analysis of other models on the same dataset including the deep learning models is strongly advised. Other techniques like Federation learning and Transfer Learning can be explored.
- Apply the same methodology on other public IoT datasets.
- Combining multiple models in an ensemble approach, such as stacking or boosting, might provide more robust and accurate results compared to using individual models.
- Another line of research can be the construction of an IoT dataset by the UPC or by collaborative efforts between academic institutions, industries, and cybersecurity communities.
- Construction and deployment of the model in a laboratory environment to test with real or synthetic data.
- Apply adversarial and poisoning attacks to the models to evaluate the robustness and resilience of the algorithms.
- Use new and efficient GPU library optimised to accelerate, training, hyper-parameters tuning, etc.

5.2. Sustainability considerations

A ML IDS can rapidly detect and prevent threats in real-time. As a result, it has the potential to decrease the time and resources necessary to manage and mitigate security incidents. Consequently, energy consumption can be reduced, as unnecessary or duplicate operations are avoided. Rather than relying on hardware-centric solutions that consistently draw power, an ML IDS can optimise the use of existing hardware, allowing for more efficient energy use. Machine learning systems can adapt and learn from new threats without the constant need for hardware upgrades. This can reduce the manufacturing and disposal of electronic devices, both processes with environmental impacts. By proactively detecting threats, a ML IDS can minimise downtime, which in turn reduces energy and resource consumption associated with system recovery and restarts and infections. Worth to mention, this work consumed a lot of time of heavy computational resources, most of the time, Machine Learning is not very eco-friendly in that aspect, trials and error, re-runs of algorithms, etc, but the objective is justified and many resources and even lives can be saved applying Machine Learning Techniques to Intrusion Detection.

5.3. Ethical considerations

The implementation of ML-IDS in cybersecurity provides advanced protection against threats and poses notable ethical implications, securing Homes up to critical infrastructures. The adoption of ML-IDS can be viewed as a proactive measure to protect sensitive data and maintain system integrity, which is

consistent with ethical principles of user safeguarding and data confidentiality. Therefore, the deployment of machine learning intrusion detection systems (ML-IDS) can greatly enhance system security. However, it is crucial to approach the use of ML-IDS with an ethical perspective by ensuring transparency, fairness, and responsible handling of data at every stage of implementation. For instance, in dataset construction, the acquisition, storage, and sharing of such data must be founded on ethical principles. It is vital to guarantee that the gathering of data does not violate privacy rights, that it has been obtained with consent where applicable, and that it does not unintentionally reveal vulnerabilities that malevolent actors might exploit.

ACRONYMS

AI	Artificial Intelligence
ANN	Artificial Neural Network
ARP	Address Resolution Protocol
APT	Advanced Persistent Threats
AutoML	Automated Machine Learning
BLE	Bluetooth Low Energy
C2	Command and Control
CAGR	Compound Annual Growth Rate
CIC	Canadian Institute for Cybersecurity
CM	Confusion Matrix
CoAP	Constrained Application Protocol
CPU	Central Processor Unit
DDoS	Distributed Denial of Service
DL	Deep Learning
DNS	Domain Name System
DoS	Denial of Service
DT	Decision Trees
F1	Score F1
FFNN	Feedforward Neural Network
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
HIDS	Host-based Intrusion Detection Systems
IDS	Intrusion Detection Systems
IIoT	Industrial Internet of Things
IoT	Internet of Things
IP	Internet Protocol
LR	Logistic Regression
M2M	Machine to Machine
MAC	Media Access Control
MI	Mutual Information
ML	Machine Learning
MLP	Multilayer Perceptron
MQTT	Message Queuing Telemetry Transport
NB	Naïve Bayes
NB-IoT	Narrow Band IoT
NIDS	Network Intrusion Detection Systems
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RF	Random Forest
SSH	Secure Shell Protocol
TCP	Transfer Control Protocol
TN	True Negative
TNR	True Negative Rate
TP	True Positive

TPR	True Positive Rate
UDP	User Datagram Protocol
UnB	University of New Brunswick of Canada
USD	United States of America Dollars

REFERENCES

[1] Oracle Corporation, (n.d.), “What is IoT?”, last accessed on 10 October 2023, https://www.oracle.com/internet-of-things/what-is-iot/

[2] Gyamfi, E.; Jurcut, A., “Intrusion Detection in Internet of Things Systems: A Review on Design Approaches Leveraging Multi-Access Edge Computing, Machine Learning, and Datasets”, *Sensors 2022*, 22, 3744. https://doi.org/10.3390/s22103744

[3] Cisco Systems, Inc., March 10, 2020, “Cisco Annual Internet Report (2018–2023) White Paper”, last accessed on 5 May 2023, https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[4] Joyanes Aguilar, L., “Hiperconectividad: Infraestructuras de Comunicaciones”, “Infraestructuras de la Internet de las Cosas: Cloud Computing, Edge, y Fog computing”, “Seguridad y ciberseguridad en Internet de las Cosas”, *Internet de las cosas, Un futuro hiperconectado: 5G, Inteligencia Artificial, Big Data, Cloud, Blockchain, Ciberseguridad*, pp. 55-90, 141-178, 281-309, Marcombo, S.L., 2021

[5] Power Solution, (n.d.), “Fog Computing and Edge Computing: What You Need to Know”, last accessed on 28 June 2023, https://www.power-solutions.com/industry-trends/fog-computing-and-edge-computing-what-you-need-to-know/

[6] Chaudhary, R., Aujla, G., Kumar, N., Zeadally, S., “Lattice-Based Public Key Cryptosystem for Internet of Things Environment: Challenges and Solutions”, *IEEE Internet of Things Journal*. pp. 1-1. 10.1109/JIOT.2018.2878707, 2018

[7] NordVPN, (n.d.), “What are IoT attacks?”, last accessed on 28 June 2023, https://nordvpn.com/es/blog/iot-attacks/

[8] MicroAI, (n.d.), “10 Types of Cyber Security Attacks in IoT”, last accessed on 28 June 2023, https://micro.ai/blog/10-types-of-cyber-security-attacks-in-the-iot

[9] Jullian, O., Otero, B., Rodriguez, E. et al. Deep-Learning Based Detection for Cyber-Attacks in IoT Networks: A Distributed Attack Detection Framework. J Netw Syst Manage 31, 33 (2023). https://doi.org/10.1007/s10922-023-09722-7

[10] Torres i Viñals, J., “Que es el deep learning”, *Python Deep Learning, Introducción práctica con Keras y TensorFlow 2*, pp. 25-44, Marcombo, S.L.,2020

[11] Girones Roig, J.,Casas Roma J., Minguillon Alfonso, J., Caihuelas Quiles, R., “Introducción a la Minería de Datos”, Minería de Datos. Modelos y algoritmos, pp. 23-33, Editorial UOC, 2017

[12] Géron A., “The Machine Learning Landscape”, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, pp. 25-66, O’Reilly Media, Inc., 2019

[13] Kim K., Aminanto M. E., Tanuwidjaja H. C., “Network Intrusion Detection using Deep Learning, A Feature Learning Approach. SpringerBriefs on Cyber Security Systems and Networks., Springer Singapore, 2018. https://doi.org/10.1007/978-981-13-1444-5

[14] Taşkın D., “IoT Learning Algorithms and Predictive Maintenance — Part III: Few-shot Learning”, last accessed on 5 July 2023, https://medium.com/iot-and-cloud/iot-learning-algorithms-and-predictive-maintenance-3-few-shot-learning-95154b606197

[15] Stratosphere. (2015). Stratosphere Laboratory Datasets. Retrieved March 13, 2020, from https://www.stratosphereips.org/datasets-overview

[16] HaddadPajouh H., Dehghantanha A., Khayami R., Raymond Choo K., “A deep Recurrent Neural Network based approach for Internet of Things malware threat hunting”, *Future Generation Computer Systems*, Volume 85, 2018, pp. 88-96, ISSN 0167-739X, https://doi.org/10.1016/j.future.2018.03.007

[17] Alex C., Creado G., Almobaideen W., Abu Alghanam O., Saadeh M., “A Comprehensive Survey for IoT Security Datasets Taxonomy, Classification and Machine Learning Mechanisms”, *Computers & Security*, Volume 132, 2023,103283, ISSN 0167-4048, https://doi.org/10.1016/j.cose.2023.103283

[18] Kumar P., Bagga H., Netam B.S., “SAD-IoT: Security Analysis of DDoS Attacks in IoT Networks”, *Wireless Pers Commun* 122, pp. 87–108 (2022). https://doi.org/10.1007/s11277-021-08890-6

[19] Alshamy, R., Ghurab, M. “A Review of Big Data in Network Intrusion Detection System: Challenges, Approaches, Datasets, and Tools”, 2020

[20] Divekar, A., Parekh, M., Savla, V., Mishra, R., & Shirole, M. (2018). “Benchmarking datasets for Anomaly-based Network Intrusion Detection: KDD CUP 99 alternatives”, *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*. doi:10.1109/cccs.2018.8586840

[21] Moustafa N., “A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets”, *Sustainable Cities and Society*, Volume 72, 2021, 102994, ISSN 2210-6707, https://doi.org/10.1016/j.scs.2021.102994.

[22] Koroniotis N., Moustafa N., Sitnikova E., Turnbull B., “Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset”, 2018. https://www.researchgate.net/publication/328736466_Towards_the_Development_of_Realistic_Botnet_Dataset_in_the_Internet_of_Things_for_Network_Forensic_Analytics_Bot-IoT_Dataset

[23] Muhammad Shafiq, Zhihong Tian, Yanbin Sun, Xiaojiang Du, and Mohsen Guizani. 2020. Selection of effective machine learning algorithm and Bot-IoT attacks traffic identification for internet of things in smart city. Future Generation Computer Systems 107 (June 2020), pp. 433–442. https://doi.org/10.1016/j.future.2020.02.017

[24] Vaccari, I.; Chiola, G.; Aiello, M.; Mongelli, M.; Cambiaso, E. “MQTTset, a New Dataset for Machine Learning Techniques on MQTT”, *Sensors* 2020-nov 18 vol. 20 iss. 22.

[25] Meidan, Y.; Bohadana, M.; Mathov, Y.; Mirsky, Y.; Shabtai, A.; Breitenbacher, D.; Elovici, Y., “N-BaIoT - Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders”, *IEEE Pervasive Computing* 2018-jul vol. 17 iss. 3.

[26] Albon C., “Machine Learning with Python Cookbook”, O'Reilly Media, Inc. ISBN: 9781491989388, 2018.

[27] Scikit-learn 1.3.1 User Guide, last accessed on 1 Sep 2023, https://scikit-learn.org/stable/user_guide.html

[28] Fernández A., García S., Galar M., Prati R. C., Krawczyk B., Herrera F., “Learning from Imbalanced Data Sets”, Springer Cham, 2018, https://doi.org/10.1007/978-3-319-98074-4

[29] IBM (n. d.) “What is a Decision Tree?”, last accessed on 7 Jul 2023, https://www.ibm.com/topics/decision-trees

[30] Banaamah, A.M.; Ahmad, I. “Intrusion Detection in IoT Using Deep Learning”, *Sensors 2022*, 22, 8417. https://doi.org/10.3390/s22218417

[31] Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). “A Deep Learning Approach to Network Intrusion Detection”, *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1), pp. 41–50. https://doi.org/10.1109/tetci.2017.2772792

[32] Dinh C. N.,Ming D., Pubudu N. P., Aruna S., Li J., Poor H. V., “Federated Learning for Internet of Things: A Comprehensive Survey”, *IEEE Communications Surveys Tutorials*, 23 (3), pp. 1622-1658, 2021, https://doi.org/10.1109%2Fcomst.2021.3075439

[33] Prendki, J., July 8, 2018, “Transfer learning for the IoT”, last accessed on 29 July 2023, https://www.embedded.com/transfer-learning-for-the-iot/

[34] Li Y., Abdallah S., “IoT data analytics in dynamic environments: From an automated machine learning perspective”, *Engineering Applications of Artificial Intelligence*, Volume 116, 2022, 105366, ISSN 0952-1976, https://doi.org/10.1016/j.engappai.2022.105366

[35] Uprety A., Rawat D. B., "Reinforcement Learning for IoT Security: A Comprehensive Survey," in IEEE Internet of Things Journal, vol. 8, no. 11, pp. 8693-8706, June 1, 2021, https://doi.org/10.1109/JIOT.2020.3040957

[36] Kalutharage, C.S.; Liu X.; Chrysoulas C.; Pitropakis N.; Papadopoulos P., “Explainable AI-Based DDOS Attack Identification Method for IoT Networks.” *Computers*. *2023*; 12(2):32. https://doi.org/10.3390/computers12020032

[37] Neto, E.C.P.; Dadkhah, S.; Ferreira, R.; Zohourian, A.; Lu, R.; Ghorbani, A.A. "CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment," *Sensors 2023,* 23(13), 5941*;* https://doi.org/10.3390/s23135941

[38] University of New Brunswick (UnB), (n.d.), “CIC_IOT_Dataset2023”, last accessed on 25 June 2023, https://www.unb.ca/cic/datasets/iotdataset-2023.html

[39] Ahmed, October 18, 2022, “The Motivation for Train-Test Split”, last accessed on 20 October 2023, https://medium.com/@nahmed3536/the-motivation-for-train-test-split-2b1837f596c3

[40] Gholamy, A.; Kreinovich, V.; Kosheleva, O., "Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation" (2018). Departmental Technical Reports (CS). 1209, last accessed on 20 October 2023, https://scholarworks.utep.edu/cs_techrep/1209

[41] Allot, See, Control, Secure, (n.d.), "Glossary of Common DDoS Attacks", last accessed on 1 July 2023, https://www.allot.com/ddos-attack-glossary/

[42] Wikipedia, (n.d.), "Denial-of-service attack", last accessed on 1 July 2023, https://en.wikipedia.org/wiki/Denial-of-service_attack

[43] Cloudfare, (n.d.), "What is a denial-of-service (DoS) attack?", last accessed on 1 July 2023, https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/

[44] Cloudfare, (n.d.), "What is the Mirai Botnet?", last accessed on 1 July 2023, https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/

[45] Computer Networking Notes, (n.d.), "Reconnaissance attacks, Tools, Types, and Prevention", last accessed on 1 July 2023, https://www.computernetworkingnotes.com/ccna-study-guide/reconnaissance-attacks-tools-types-and-prevention.html#:~:text=A%20reconnaissance%20attack%20is%20a,tool%20for%20an%20actual%20attack

[46] Odogwu C., March 22, 2023, "What Are Reconnaissance Attacks and How Do They Work?", last accessed on 1 July 2023, https://www.makeuseof.com/what-are-reconnaissance-attacks-and-how-do-they-work/

[47] Rapid7, (n.d.), "Spoofing Attacks", last accessed on 1 July 2023, https://www.rapid7.com/fundamentals/spoofing-attacks/#:~:text=Spoofing%20is%20the%20act%20of,often%20used%20to%20commit%20fraud

[48] Malwarebytes, (n.d.), "What is a spoofing attack?", last accessed on 1 July 2023, https://www.malwarebytes.com/spoofing

[49] Kaspersky, (n.d.), "Brute Force Attack: Definition and Examples", last accessed on 1 July 2023, https://www.kaspersky.com/resource-center/definitions/brute-force-attack

[50] OWASP Foundation, (n.d.), "Vulnerabilities, What is a vulnerability?", last accessed on 1 July 2023, https://owasp.org/www-community/vulnerabilities/

[51] Javatpoint, (n.d.), “Introduction to Dimensionality Reduction Technique”, last accessed on 8 July 2023, https://www.javatpoint.com/dimensionality-reduction-technique

[52] Kavika R., September 22, 2022, “Dimensionality Reduction Techniques in Data Science”, last accessed on 8 July 2023, https://www.kdnuggets.com/2022/09/dimensionality-reduction-techniques-data-science.html

[53] Vadapalli P., August 7, 2020, “Top 10 Dimensionality Reduction Techniques For Machine Learning”, last accessed on 8 July 2023, https://www.upgrad.com/blog/top-dimensionality-reduction-techniques-for-machine-learning/

[54] Statistical tools for high-throughput data analysis, “Correlation matrix : A quick start guide to analyze, format and visualize a correlation matrix using R software”, last accessed on 8 July 2023, http://www.sthda.com/english/wiki/correlation-matrix-a-quick-start-guide-to-analyze-format-and-visualize-a-correlation-matrix-using-r-software

[55] Turney S., June 22, 2023, “Pearson Correlation Coefficient (r) | Guide & Examples”, last accessed on 8 July 2023, https://www.scribbr.com/statistics/pearson-correlation-coefficient/

[56] Guhanesvar, June 26, 2021, “Feature Selection Based on Mutual Information Gain for Classification and Regression”, last accessed on 8 July 2023, https://guhanesvar.medium.com/feature-selection-based-on-mutual-information-gain-for-classification-and-regression-d0f86ea5262a

[57] Scikit-learn 1.3.1 User Guide, (n.d.) “Mutual information (MI)”, last accessed on 12 July 2023, https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html

[58] Wikipedia, (n.d.), Confusion matrix, (n.d.), last accessed on 19 August 2023, https://en.wikipedia.org/wiki/Confusion_matrix

[59] Saurabh D., Confusion Matrix Explained: Calculating Accuracy, TPR, FPR, TNR, Precision, and Prevalence. last accessed on 19 August 2023, https://medium.com/@saurabhdhandeblog/confusion-matrix-explained-calculating-accuracy-tpr-fpr-tnr-precision-and-prevalence-87557fe8714d

[60] Navas J. February 8, 2022, “What is hyperparameter tuning?”, last accessed on 23 July 2023, https://www.anyscale.com/blog/what-is-hyperparameter-tuning

[61] Alhamid M., December 24, 2020, “What is Cross-Validation?”, last accessed on 23 July 2023, https://towardsdatascience.com/what-is-cross-validation-60c01f9d9e75

[62] Refaeilzadeh, P., Tang, L., Liu, H. (2009). Cross-Validation. In: LIU, L., ÖZSU, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_565

[63] Scikit-learn 1.3.1 User Guide, (n.d.) “Cross-validation: evaluating estimator performance”, last accessed on 18 August 2023, https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators

[64] Rosaen K., June 06, 2016, last accessed on 18 August 2023 http://karlrosaen.com/ml/learning-log/2016-06-20/

[65] Brownlee J., August 28, 2020, “Tune Hyperparameters for Classification Machine Learning Algorithms”, last accessed on 24 July 2023, https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/

[66] Melanee Group, May 2021, “A Comprehensive Analysis of Hyperparameter Optimization in Logistic Regression Models”, last accessed on 24 July 2023, https://levelup.gitconnected.com/a-

comprehensive-analysis-of-hyperparameter-optimization-in-logistic-regression-models-521564c1bfc0

[67] Scikit-learn 1.3.1 User Guide, (n.d.), "sklearn.linear_model.LogisticRegression", last accessed on 29 July 2023, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

[68] Singh Chauhan N., February 9, 2022, "Decision Tree Algorithm, Explained", last accessed on 29 July 2023, https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

[69] Mithrakumar M., November 11, 2019,"How to tune a Decision Tree?", last accessed on 29 July 2023, https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680

[70] Ben Fraj, M., December 20, 2017 ,"InDepth: Parameter tuning for Decision Tree", last accessed on 29 July 2023, https://medium.com/@mohtedibf/indepth-parameter-tuning-for-decision-tree-6753118a03c3

[71] Keldenich T., September 16, 2022, "Decision Tree How to Use It and Its Hyperparameters", last accessed on 29 July 2023, https://inside-machinelearning.com/en/decision-tree-and-hyperparameters/

[72] Scikit-learn 1.3.1 User Guide, (n.d.), "sklearn.tree.DecisionTreeClassifier", last accessed on 29 July 2023, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

[73] Jain K., Apr 2, 2021, "The Naive Bayes Guide, How to Improve Naive Bayes?", last accessed on 30 July 2023, https://medium.com/analytics-vidhya/how-to-improve-naive-bayes-9fa698e14cba

[74] Scikit-learn 1.3.1 User Guide, (n.d.), "sklearn.naive_bayes.GaussianNB", last accessed on 30 July 2023, https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

[75] Koehrsen W., "Hyperparameter Tuning the Random Forest in Python", last accessed on 30 July 2023, https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

[76] Arya N., August 22, 2022, "Tuning Random Forest Hyperparameters", last accessed on 30 July 2023,

https://www.kdnuggets.com/2022/08/tuning-random-forest-hyperparameters.html

[77] “Hyperparameter Tuning Using Grid Search and Random Search in Python”, last accessed on 30 July 2023, https://www.kdnuggets.com/2022/10/hyperparameter-tuning-grid-search-random-search-python.html

[78] Scikit-learn 1.3.1 User Guide, (n.d.), “sklearn.ensemble.RandomForestClassifier”, last accessed on 30 July 2023, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier

[79] Fuchs M., February 3, 2021, “NN - Multi-layer Perceptron Classifier (MLPClassifier)”, last accessed on 31 July, https://michael-fuchs-python.netlify.app/2021/02/03/nn-multi-layer-perceptron-classifier-mlpclassifier/

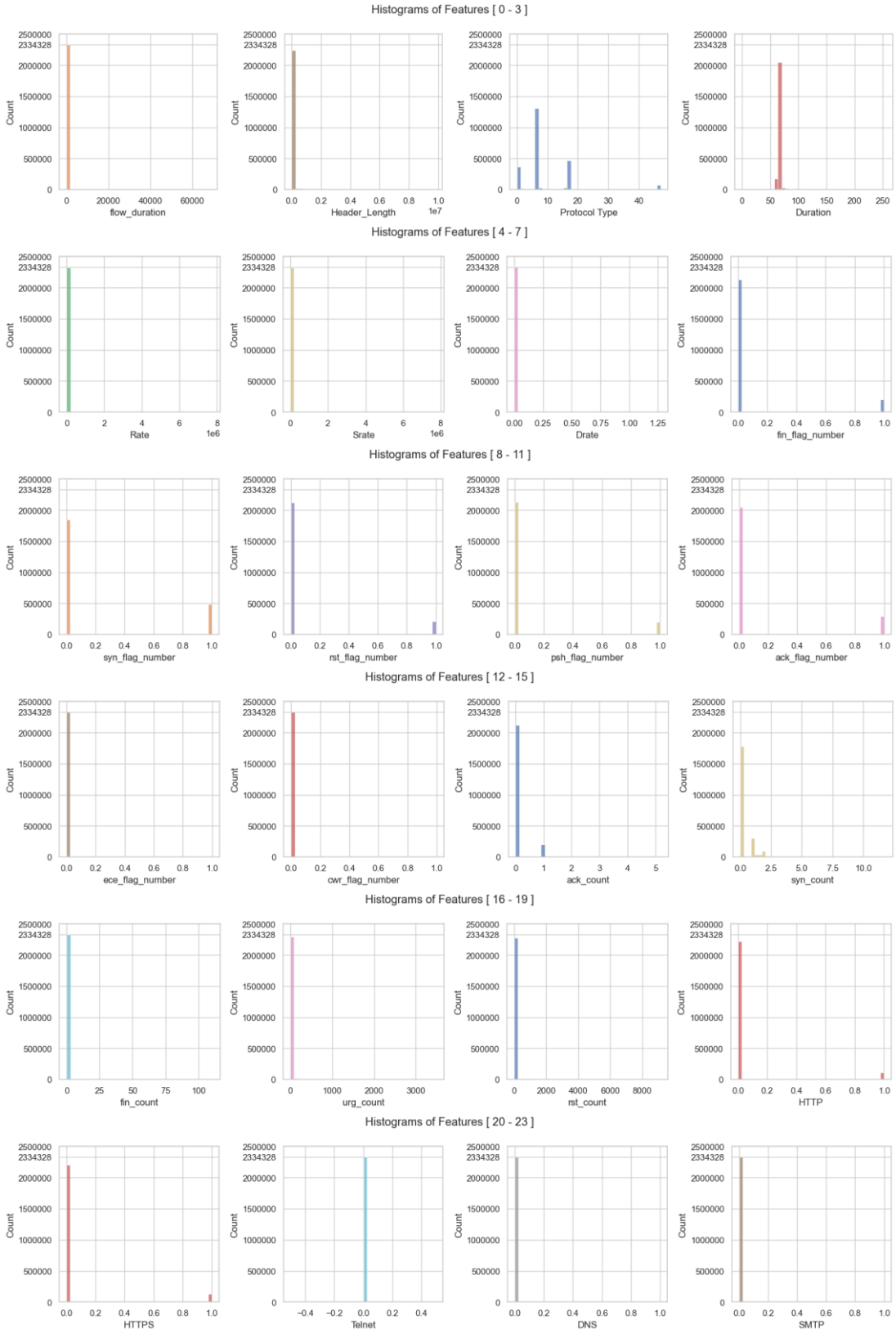
[80] Databall, (n.d.), “Parameter Tuning”, last accessed on 31 July, https://klane.github.io/databall/model/parameters/

[81] Scikit-learn 1.3.1 User Guide, (n.d.), “sklearn.neural_network.MLPClassifier”, last accessed on 31 July 2023, https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

ANNEXES

Annex A: Dataset exploratory analysis

Fig. A.1 Dataset features distribution



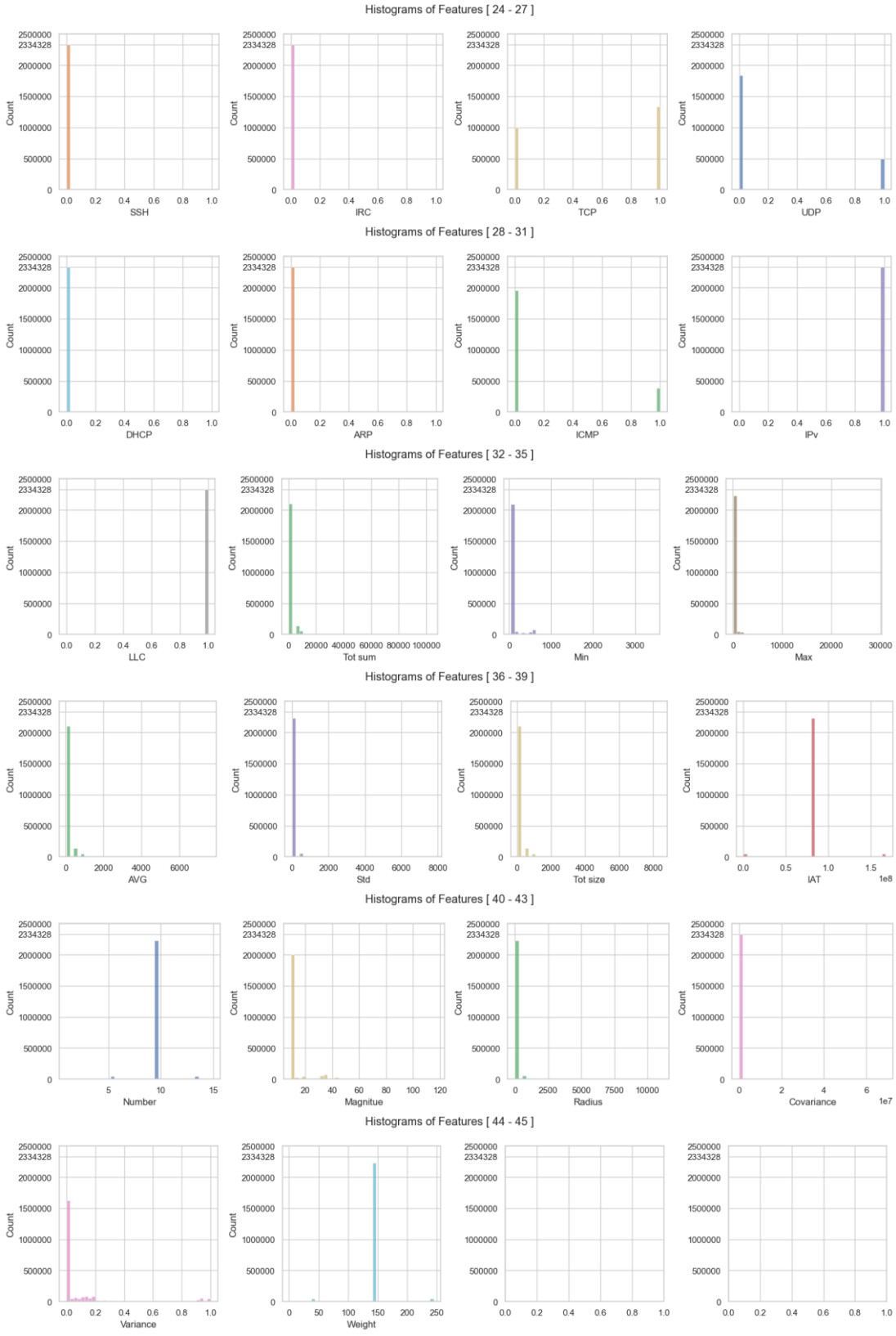

Histograms of Features [24 - 27]
SSH
IRC
TCP
UDP
Histograms of Features [28 - 31]
DHCP
ARP
ICMP
IPv
Histograms of Features [32 - 35]
LLC
Tot sum
Min
Max
Histograms of Features [36 - 39]
AVG
Std
Tot size
IAT
1e8
Histograms of Features [40 - 43]
Number
Magnitue
Radius
Covariance
1e7
Histograms of Features [44 - 45]
Variance
Weight
Count

Table A.1. Dataset Features numerical description.

Feature	count	mean	std	min	25%	50%	75%	max
flow_duration	2334328	5.63530458	262.091923	0	0	0	0.10491436	68430.7122
Header_Length	2334328	76778.2503	462321.857	0	54	54	276.5325	9806497.8
Protocol Type	2334328	9.06568768	8.94656026	0	6	6	14.31	47
Duration	2334328	66.3356124	13.9416923	0	64	64	64	255
Rate	2334328	9063.94016	99428.7596	0	2.0875004	15.7039793	117.386375	7780435.6
Srate	2334328	9063.94016	99428.7596	0	2.0875004	15.7039793	117.386375	7780435.6
Drate	2334328	3.41E-06	0.001371	0	0	0	0	1.27071333
fin_flag_number	2334328	0.08656281	0.2811934	0	0	0	0	1
syn_flag_number	2334328	0.2073736	0.40542553	0	0	0	0	1
rst_flag_number	2334328	0.09047872	0.28686644	0	0	0	0	1
psh_flag_number	2334328	0.08771732	0.28288342	0	0	0	0	1
ack_flag_number	2334328	0.1234698	0.32897576	0	0	0	0	1
ece_flag_number	2334328	1.29E-06	0.00113365	0	0	0	0	1
cwr_flag_number	2334328	8.57E-07	0.00092562	0	0	0	0	1
ack_count	2334328	0.09049278	0.28619119	0	0	0	0	5.2
syn_count	2334328	0.33046888	0.66327069	0	0	0	0.06	11.8
fin_count	2334328	0.09916589	0.34230291	0	0	0	0	110.86
urg_count	2334328	6.23206058	71.7457522	0	0	0	0	3494.3
rst_count	2334328	38.4082763	324.708597	0	0	0	0.01	9126.5
HTTP	2334328	0.04829013	0.21437867	0	0	0	0	1
HTTPS	2334328	0.05497299	0.22792759	0	0	0	0	1
DNS	2334328	0.00012038	0.010971	0	0	0	0	1
Telnet	2334328	0	0	0	0	0	0	0
SMTP	2334328	4.28E-07	0.00065451	0	0	0	0	1
SSH	2334328	3.77E-05	0.00613977	0	0	0	0	1
IRC	2334328	4.28E-07	0.00065451	0	0	0	0	1
TCP	2334328	0.57391763	0.49450611	0	0	1	1	1
UDP	2334328	0.21188753	0.40864566	0	0	0	0	1
DHCP	2334328	1.29E-06	0.00113365	0	0	0	0	1
ARP	2334328	6.47E-05	0.00804255	0	0	0	0	1
ICMP	2334328	0.16372935	0.37002987	0	0	0	0	1
IPv	2334328	0.99989162	0.01041012	0	1	1	1	1
LLC	2334328	0.99989162	0.01041012	0	1	1	1	1
Tot sum	2334328	1310.78792	2626.21915	42	525	567	567.54	103112.2
Min	2334328	91.6382647	139.621352	42	50	54	54	3416.4
Max	2334328	182.399327	526.80039	42	50	54	55.26	28854
AVG	2334328	124.841536	241.491161	42	50	54	54.0501127	7549.36127
Std	2334328	33.4557917	161.109364	0	0	0	0.37190955	7814.29924
Tot size	2334328	124.950134	242.662375	42	50	54	54.06	8409.2
IAT	2334328	83186605	17048120.7	0	83071564.1	83124522.1	83343908.2	167639430
Number	2334328	9.49871808	0.81896897	1	9.5	9.5	9.5	15
Magnitue	2334328	13.1266727	8.64063988	9.16515139	10	10.3923048	10.3967252	117.978075
Radius	2334328	47.2799398	227.862704	0	0	0	0.50592128	11051.088
Covariance	2334328	31017.2362	322733.368	0	0	0	1.35404398	69004153
Variance	2334328	0.09643977	0.23293241	0	0	0	0.08	1
Weight	2334328	141.517847	21.0664315	1	141.55	141.55	141.55	244.6

Annex B: Confusion Matrices and Reports of Multiclassification.

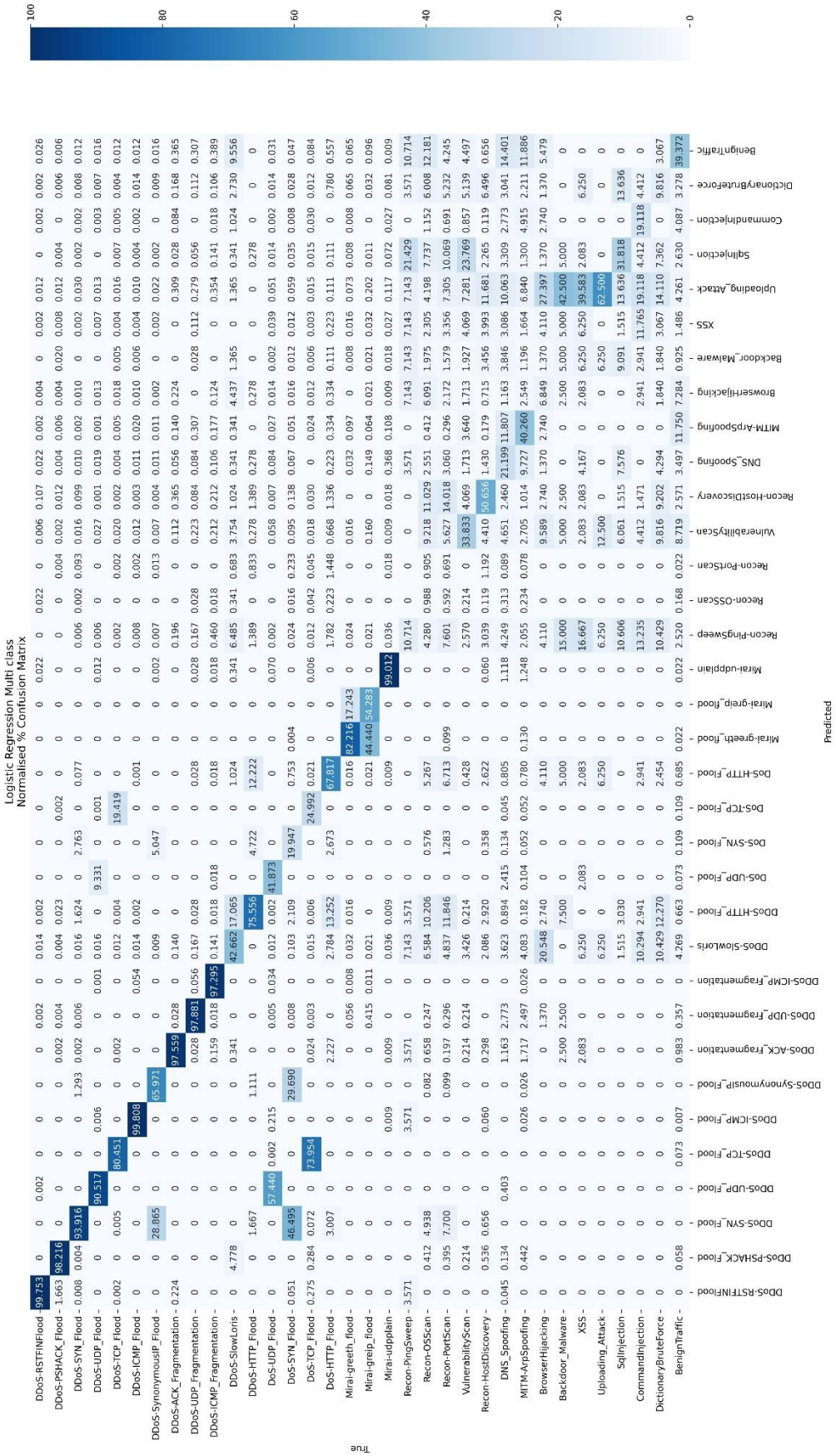


Fig. B.1 Confusion matrix for Logistic Regression multiclassification

Table B.1. Classification report by classes of multiclass Logistic Regression model

	precision	recall	F1 score	support
DDoS-RSTFINFlood	0.9811	0.9975	0.9893	50569
DDoS-PSHACK_Flood	0.9969	0.9822	0.9895	51187
DDoS-SYN_Flood	0.6561	0.9392	0.7725	50742
DDoS-UDP_Flood	0.7198	0.9052	0.8019	67658
DDoS-TCP_Flood	0.6467	0.8045	0.7170	56223
DDoS-ICMP_Flood	0.9989	0.9981	0.9985	90011
DDoS-SynonymousIP_Flood	0.7836	0.6597	0.7163	44979
DDoS-ACK_Fragmentation	0.9233	0.9756	0.9487	3564
DDoS-UDP_Fragmentation	0.9274	0.9788	0.9524	3587
DDoS-ICMP_Fragmentation	0.9876	0.9729	0.9802	5656
DDoS-SlowLoris	0.0949	0.4266	0.1553	293
DDoS-HTTP_Flood	0.1200	0.7556	0.2071	360
DoS-UDP_Flood	0.7313	0.4187	0.5325	41485
DoS-SYN_Flood	0.5737	0.1995	0.2960	25362
DoS-TCP_Flood	0.4328	0.2499	0.3169	33395
DoS-HTTP_Flood	0.4935	0.6782	0.5713	898
Mirai-greeth_flood	0.7089	0.8222	0.7613	12399
Mirai-greip_flood	0.7047	0.5428	0.6132	9397
Mirai-udpplain	0.9883	0.9901	0.9892	11133
Recon-PingSweep	0.0034	0.1071	0.0066	28
Recon-OSScan	0.1263	0.0099	0.0183	1215
Recon-PortScan	0.0354	0.0069	0.0116	1013
VulnerabilityScan	0.0786	0.3383	0.1276	467
Recon-HostDiscovery	0.4650	0.5066	0.4849	1678
DNS_Spoofing	0.2940	0.2120	0.2464	2236
MITM-ArpSpoofing	0.4325	0.4026	0.4170	3845
BrowserHijacking	0.0037	0.0685	0.0071	73
Backdoor_Malware	0.0048	0.0500	0.0087	40
XSS	0.0053	0.0625	0.0099	48
Uploading_Attack	0.0058	0.6250	0.0114	16
SqlInjection	0.0225	0.3182	0.0421	66
CommandInjection	0.0147	0.1912	0.0272	68
DictionaryBruteForce	0.0161	0.0982	0.0276	163
BenignTraffic	0.8156	0.3937	0.5311	13728
macro average	0.4645	0.5202	0.4496	583582

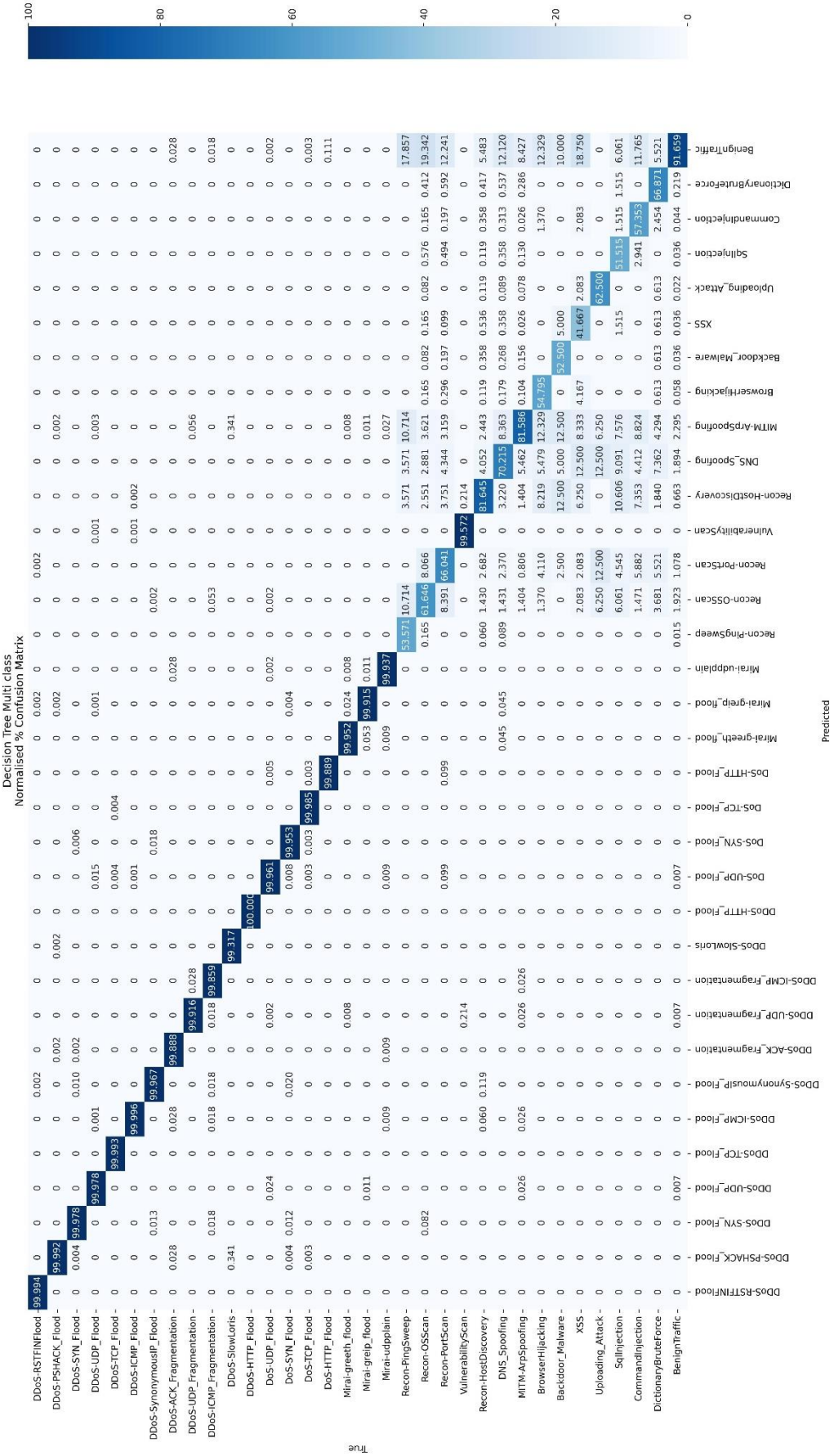


Fig. B.2 Confusion matrix for Decision Trees multiclassification

Table B.2. Classification report by classes of multiclassification Decision Trees model

	precision	recall	F1 score	support
DDoS-RSTFINFlood	1.0000	0.9999	1.0000	50569
DDoS-PSHACK_Flood	0.9999	0.9999	0.9999	51187
DDoS-SYN_Flood	0.9998	0.9998	0.9998	50742
DDoS-UDP_Flood	0.9998	0.9998	0.9998	67658
DDoS-TCP_Flood	1.0000	0.9999	1.0000	56223
DDoS-ICMP_Flood	0.9999	1.0000	0.9999	90011
DDoS-SynonymousIP_Flood	0.9997	0.9997	0.9997	44979
DDoS-ACK_Fragmentation	0.9992	0.9989	0.9990	3564
DDoS-UDP_Fragmentation	0.9983	0.9992	0.9987	3587
DDoS-ICMP_Fragmentation	0.9996	0.9986	0.9991	5656
DDoS-SlowLoris	0.9966	0.9932	0.9949	293
DDoS-HTTP_Flood	1.0000	1.0000	1.0000	360
DoS-UDP_Flood	0.9995	0.9996	0.9996	41485
DoS-SYN_Flood	0.9995	0.9995	0.9995	25362
DoS-TCP_Flood	0.9999	0.9999	0.9999	33395
DoS-HTTP_Flood	0.9956	0.9989	0.9972	898
Mirai-greeth_flood	0.9994	0.9995	0.9995	12399
Mirai-greip_flood	0.9991	0.9991	0.9991	9397
Mirai-udpplain	0.9996	0.9994	0.9995	11133
Recon-PingSweep	0.6818	0.5357	0.6000	28
Recon-OSScan	0.6089	0.6165	0.6127	1215
Recon-PortScan	0.6264	0.6604	0.6430	1013
VulnerabilityScan	0.9957	0.9957	0.9957	467
Recon-HostDiscovery	0.8111	0.8164	0.8138	1678
DNS_Spoofing	0.7063	0.7021	0.7042	2236
MITM-ArpSpoofing	0.8240	0.8159	0.8199	3845
BrowserHijacking	0.6061	0.5479	0.5755	73
Backdoor_Malware	0.4375	0.5250	0.4773	40
XSS	0.4000	0.4167	0.4082	48
Uploading_Attack	0.4348	0.6250	0.5128	16
SqlInjection	0.5000	0.5152	0.5075	66
CommandInjection	0.5571	0.5735	0.5652	68
DictionaryBruteForce	0.6022	0.6687	0.6337	163
BenignTraffic	0.9197	0.9166	0.9181	13728
macro average	0.8440	0.8505	0.8463	583582

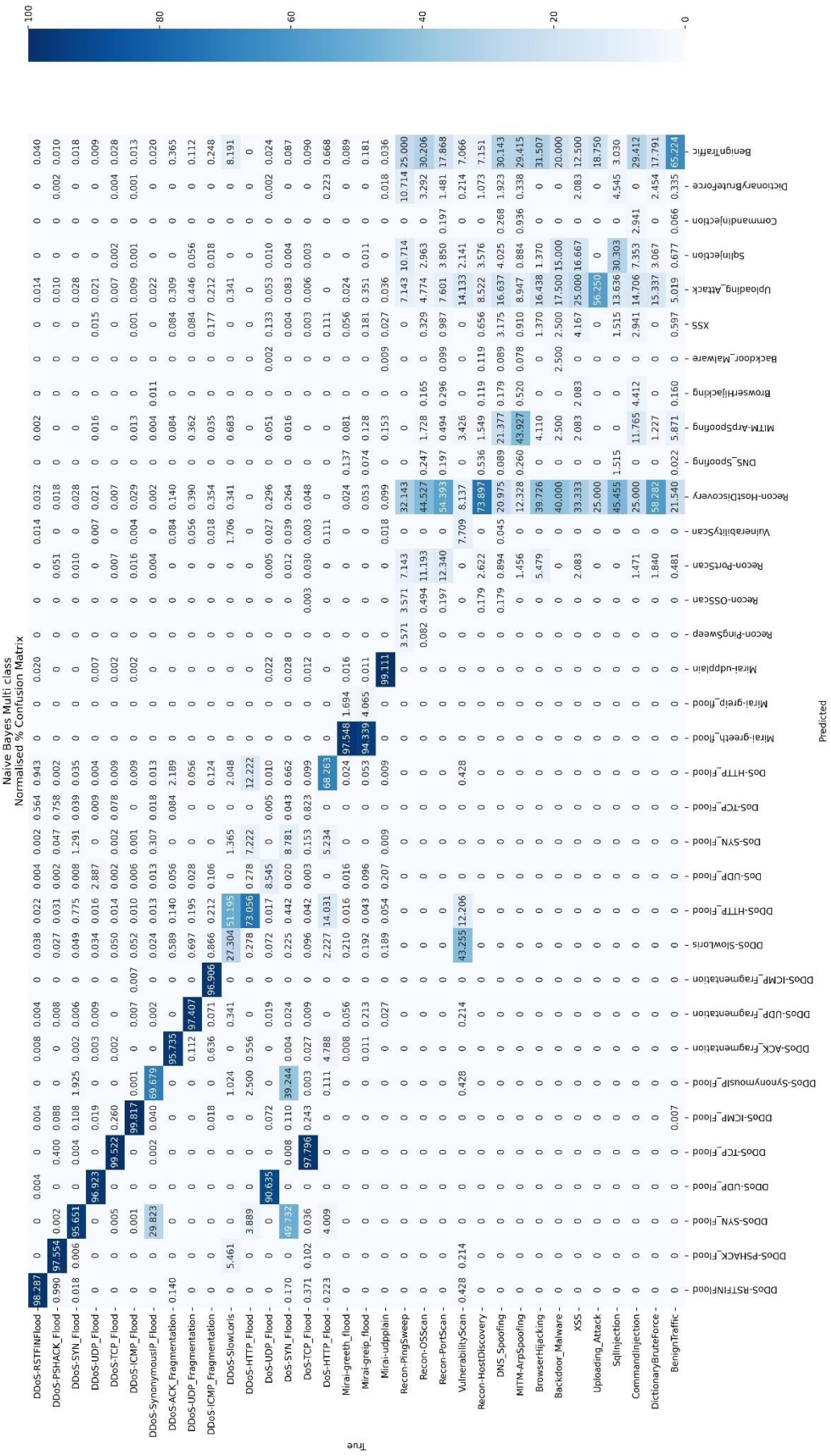


Fig. B.3 Confusion matrix for Naive Bayes multiclassification

Table B.3. Classification report by classes of multiclassification Naive Bayes model

	precision	recall	F1 score	support
DDoS-RSTFINFlood	0.9863	0.9829	0.9846	50569
DDoS-PSHACK_Flood	0.9989	0.9755	0.9871	51187
DDoS-SYN_Flood	0.6504	0.9565	0.7743	50742
DDoS-UDP_Flood	0.6356	0.9692	0.7677	67658
DDoS-TCP_Flood	0.6299	0.9952	0.7715	56223
DDoS-ICMP_Flood	0.9953	0.9982	0.9968	90011
DDoS-SynonymousIP_Flood	0.7411	0.6968	0.7183	44979
DDoS-ACK_Fragmentation	0.9701	0.9574	0.9637	3564
DDoS-UDP_Fragmentation	0.9790	0.9741	0.9765	3587
DDoS-ICMP_Fragmentation	0.9989	0.9691	0.9838	5656
DDoS-SlowLoris	0.1068	0.2730	0.1536	293
DDoS-HTTP_Flood	0.2158	0.7306	0.3331	360
DoS-UDP_Flood	0.6368	0.0855	0.1507	41485
DoS-SYN_Flood	0.7012	0.0878	0.1561	25362
DoS-TCP_Flood	0.2639	0.0082	0.0160	33395
DoS-HTTP_Flood	0.4131	0.6826	0.5147	898
Mirai-greeth_flood	0.5771	0.9755	0.7251	12399
Mirai-greip_flood	0.6453	0.0407	0.0765	9397
Mirai-udpplain	0.9963	0.9911	0.9937	11133
Recon-PingSweep	0.5000	0.0357	0.0667	28
Recon-OSScan	0.3529	0.0049	0.0097	1215
Recon-PortScan	0.2385	0.1234	0.1627	1013
VulnerabilityScan	0.4045	0.0771	0.1295	467
Recon-HostDiscovery	0.1814	0.7390	0.2913	1678
DNS_Spoofing	0.0370	0.0009	0.0017	2236
MITM-ArpSpoofing	0.5335	0.4393	0.4818	3845
BrowserHijacking	0.0000	0.0000	0.0000	73
Backdoor_Malware	0.0909	0.0250	0.0392	40
XSS	0.0060	0.0417	0.0105	48
Uploading_Attack	0.0045	0.5625	0.0088	16
SqlInjection	0.0474	0.3030	0.0820	66
CommandInjection	0.0364	0.0294	0.0325	68
DictionaryBruteForce	0.0204	0.0245	0.0223	163
BenignTraffic	0.7595	0.6522	0.7018	13728
macro average	0.4810	0.4826	0.4142	583582

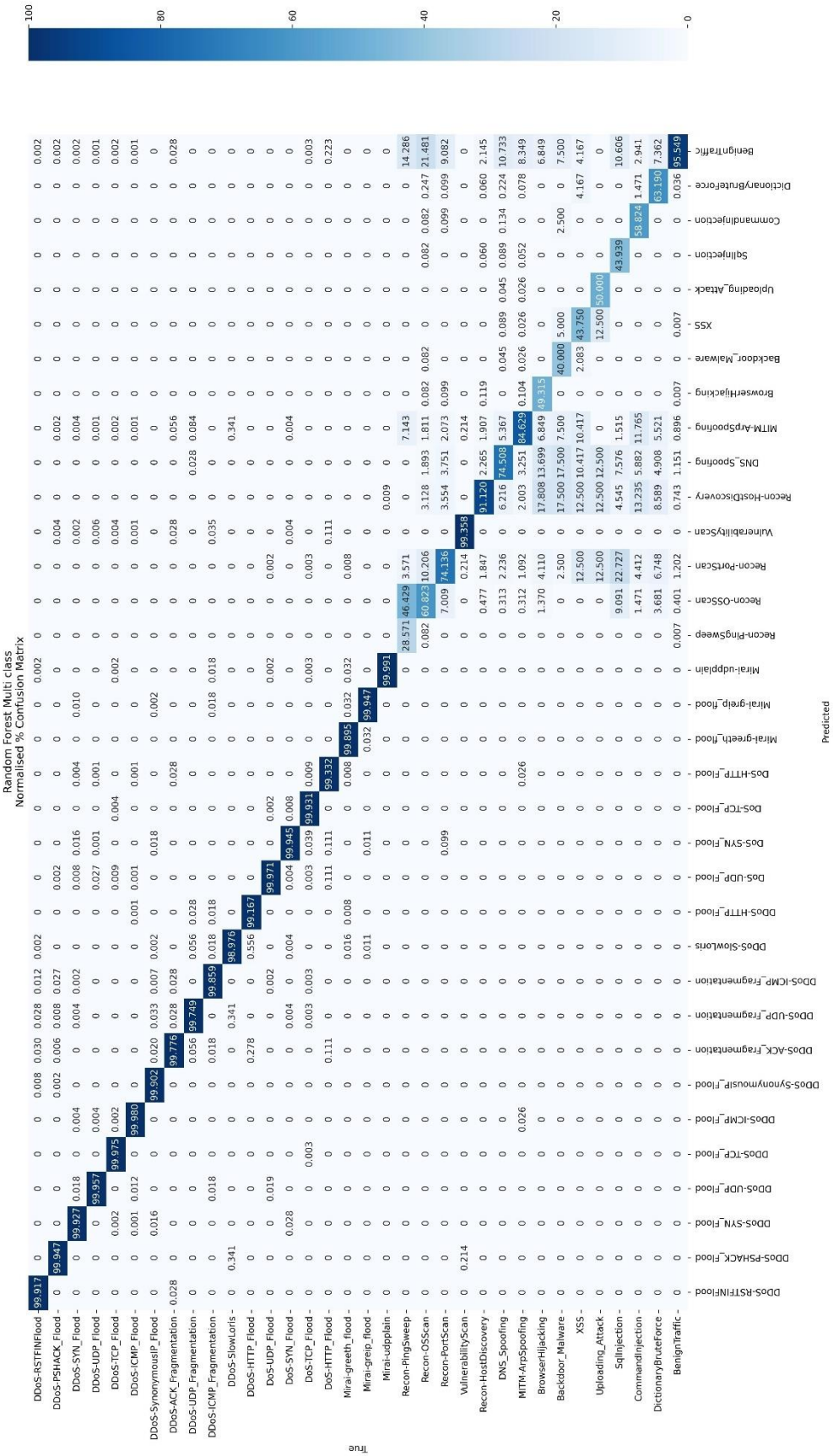


Fig. B.4 Confusion matrix for Random Forest multiclassification

Table B.4. Classification report by classes of multiclassification Random Forest model

	precision	recall	F1 score	support
DDoS-RSTFINFlood	1.0000	0.9992	0.9996	50569
DDoS-PSHACK_Flood	1.0000	0.9995	0.9997	51187
DDoS-SYN_Flood	0.9997	0.9993	0.9995	50742
DDoS-UDP_Flood	0.9996	0.9996	0.9996	67658
DDoS-TCP_Flood	1.0000	0.9998	0.9999	56223
DDoS-ICMP_Flood	0.9999	0.9998	0.9999	90011
DDoS-SynonymousIP_Flood	0.9999	0.9990	0.9995	44979
DDoS-ACK_Fragmentation	0.9911	0.9978	0.9944	3564
DDoS-UDP_Fragmentation	0.9892	0.9975	0.9933	3587
DDoS-ICMP_Fragmentation	0.9952	0.9986	0.9969	5656
DDoS-SlowLoris	0.9635	0.9898	0.9764	293
DDoS-HTTP_Flood	0.9889	0.9917	0.9903	360
DoS-UDP_Flood	0.9992	0.9997	0.9995	41485
DoS-SYN_Flood	0.9987	0.9994	0.9991	25362
DoS-TCP_Flood	0.9999	0.9993	0.9996	33395
DoS-HTTP_Flood	0.9889	0.9933	0.9911	898
Mirai-greeth_flood	0.9998	0.9990	0.9994	12399
Mirai-greip_flood	0.9988	0.9995	0.9991	9397
Mirai-udpplain	0.9992	0.9999	0.9996	11133
Recon-PingSweep	0.8000	0.2857	0.4211	28
Recon-OSScan	0.8041	0.6082	0.6926	1215
Recon-PortScan	0.6212	0.7414	0.6760	1013
VulnerabilityScan	0.9687	0.9936	0.9810	467
Recon-HostDiscovery	0.7738	0.9112	0.8369	1678
DNS_Spoofing	0.7971	0.7451	0.7702	2236
MITM-ArpSpoofing	0.8991	0.8463	0.8719	3845
BrowserHijacking	0.8000	0.4932	0.6102	73
Backdoor_Malware	0.8000	0.4000	0.5333	40
XSS	0.7241	0.4375	0.5455	48
Uploading_Attack	0.8000	0.5000	0.6154	16
SqlInjection	0.8286	0.4394	0.5743	66
CommandInjection	0.8696	0.5882	0.7018	68
DictionaryBruteForce	0.8306	0.6319	0.7178	163
BenignTraffic	0.9295	0.9555	0.9423	13728
macro average	0.9164	0.8394	0.8655	583582

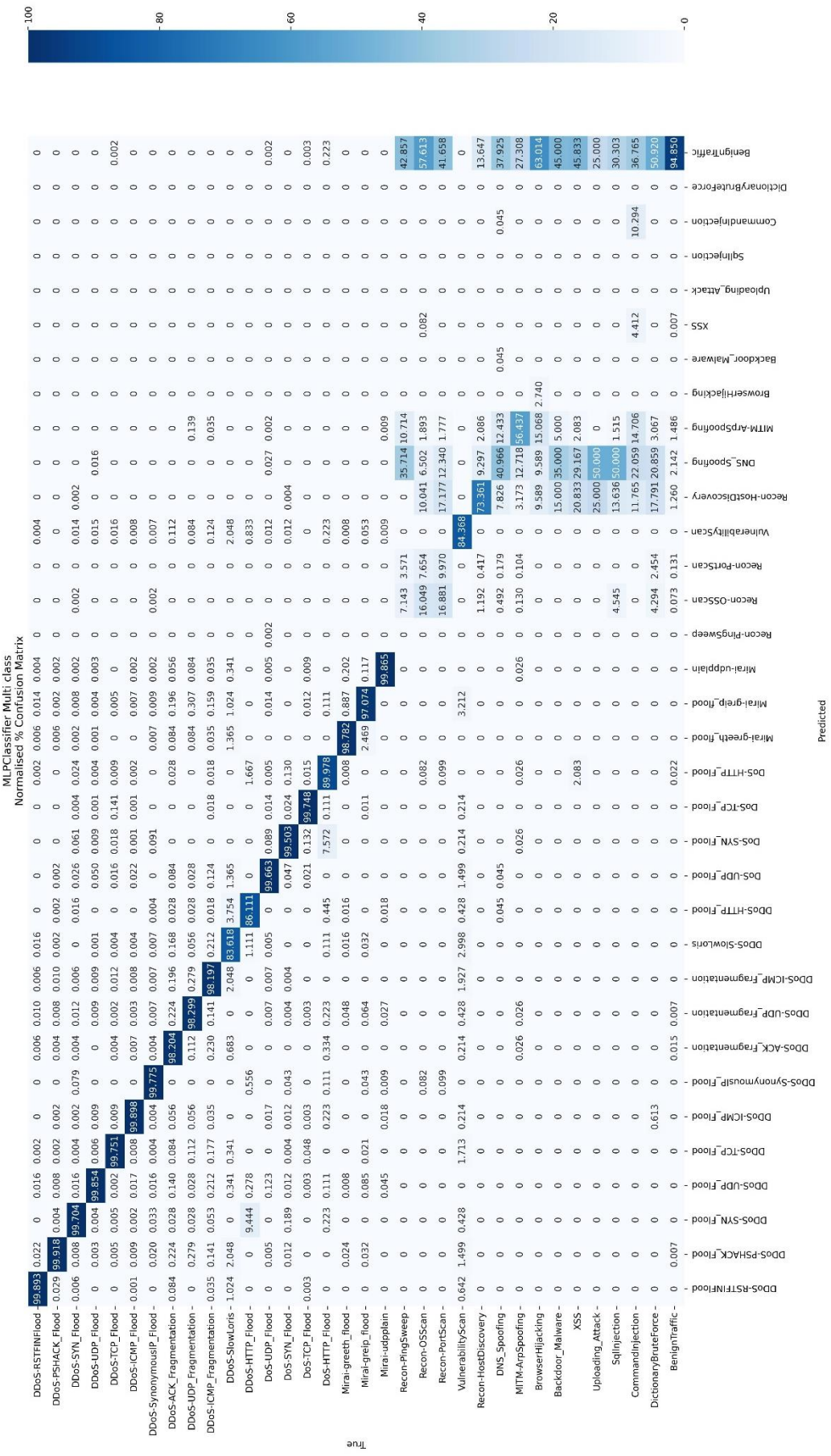


Fig. B.5 Confusion matrix for MLPClassifier multiclassification

Table B.5. Classification report by classes of multiclassification MLPClassifier model

	precision	recall	F1 score	support
DDoS-RSTFINFlood	0.9994	0.9989	0.9992	50569
DDoS-PSHACK_Flood	0.9983	0.9992	0.9987	51187
DDoS-SYN_Flood	0.9977	0.9970	0.9974	50742
DDoS-UDP_Flood	0.9980	0.9985	0.9983	67658
DDoS-TCP_Flood	0.9989	0.9975	0.9982	56223
DDoS-ICMP_Flood	0.9996	0.9990	0.9993	90011
DDoS-SynonymousIP_Flood	0.9986	0.9978	0.9982	44979
DDoS-ACK_Fragmentation	0.9879	0.9820	0.9849	3564
DDoS-UDP_Fragmentation	0.9805	0.9830	0.9818	3587
DDoS-ICMP_Fragmentation	0.9876	0.9820	0.9848	5656
DDoS-SlowLoris	0.7903	0.8362	0.8126	293
DDoS-HTTP_Flood	0.8960	0.8611	0.8782	360
DoS-UDP_Flood	0.9971	0.9966	0.9969	41485
DoS-SYN_Flood	0.9906	0.9950	0.9928	25362
DoS-TCP_Flood	0.9970	0.9975	0.9973	33395
DoS-HTTP_Flood	0.9109	0.8998	0.9053	898
Mirai-greeth_flood	0.9796	0.9878	0.9837	12399
Mirai-greip_flood	0.9792	0.9707	0.9749	9397
Mirai-udpplain	0.9947	0.9987	0.9967	11133
Recon-PingSweep	0.0000	0.0000	0.0000	28
Recon-OSScan	0.4577	0.1605	0.2377	1215
Recon-PortScan	0.4353	0.0997	0.1622	1013
VulnerabilityScan	0.8347	0.8437	0.8392	467
Recon-HostDiscovery	0.5941	0.7336	0.6565	1678
DNS_Spoofing	0.4134	0.4097	0.4115	2236
MITM-ArpSpoofing	0.7834	0.5644	0.6561	3845
BrowserHijacking	1.0000	0.0274	0.0533	73
Backdoor_Malware	0.0000	0.0000	0.0000	40
XSS	0.0000	0.0000	0.0000	48
Uploading_Attack	0.0000	0.0000	0.0000	16
SqlInjection	0.0000	0.0000	0.0000	66
CommandInjection	0.8750	0.1029	0.1842	68
DictionaryBruteForce	0.0000	0.0000	0.0000	163
BenignTraffic	0.7889	0.9485	0.8614	13728
macro average	0.7254	0.6579	0.6630	583582