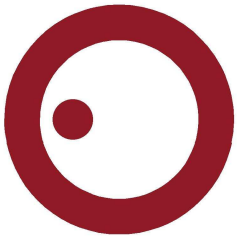


IPL

escola superior de tecnologia e gestão
instituto politécnico de leiria

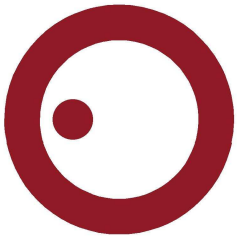
Polytechnic Institute of Leiria
School of Technology and Management
Department of Computer Engineering
Master in Cybersecurity and Forensic computing

DATA SCIENCE IN CYBERSECURITY

EVALUATING THE USE OF MACHINE LEARNING IN AN IOT-IDS

STUDENT NUNO ALEXANDRE GONÇALVES DOS PRAZERES

Leiria, November 2021



escola superior de tecnologia e gestão
instituto politécnico de leiria

Polytechnic Institute of Leiria
School of Technology and Management
Department of Computer Engineering
Master in Cybersecurity and Forensic computing

DATA SCIENCE IN CYBERSECURITY

EVALUATING THE USE OF MACHINE LEARNING IN AN IOT-IDS

STUDENT NUNO ALEXANDRE GONÇALVES DOS PRAZERES
Number: 2192642

Dissertation conducted under the guidance of the Professor Doctor Carlos Manuel da Silva Rabadão (carlos.rabadao@ipleiria.pt), Professor Doctor Leonel Filipe Simões Santos (leonel.santos@ipleiria.pt) and Professor Doctor Rogério Luís de Carvalho Costa (rogerio.l.costa@ipleiria.pt).

Leiria, November 2021

ACKNOWLEDGMENTS

Acknowledgment to my Parents and Micaela for the support, love, and patience.

Thankful also to my Advisors that caught a guy from telecommunications that only knew how to talk about fiber optics and to whom they taught the language of cybersecurity, machine learning, and datasets.

ABSTRACT

The new generation of communication networks has brought with them the digitalization of companies and services that have changed not only the way we communicate with each other but also the way we exchange personal and confidential data between people and entities. The IoT is one of the technological paradigms that benefits the most from these new forms of connectivity. The IoT allows us to be always connected to people, companies, our homes, our cities, our intelligent equipment and allows us to automate tasks or control situations remotely that would not be possible without this type of equipment and technology. But with the globalization of networks and services, the need to protect our data and our privacy is something to be concerned about. Although there are already several security options, both in companies and in our service providers, the amount of data that is currently generated far exceeds the capacity, of humans and systems, to analyze what is happening on our networks.

In this context, the dissertation presented here will make use of data science and implement machine learning techniques to deal with the volume of data generated by an IoT network. As a scenario, the network of a smart city was chosen, where an intrusion detection system will be placed, supported by a machine learning model so that it is possible to detect any type of activity that is not recognized as being its normal production behavior. The anomaly detection methodology was implemented through machine learning algorithms that enabled the classification of network flows as benign or malicious. By comparing supervised and unsupervised classification algorithms, we found that with a dataset from an IoT network and with flows previously categorized as normal traffic and malicious traffic, supervised classifiers manage to obtain the best results, although they are limited if there is one attack that has not been considered in the given dataset.

By combining, in this dissertation, an intrusion detection system with data science and specifically with machine learning models, it was demonstrated that this is a valid cybersecurity solution and that it constitutes an additional layer in terms of ensuring the security of our networks, services, and data.

INDEX

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AD	Anomaly Detection.
AI	Artificial Intelligence.
AMQP	Advanced Message Queuing Protocol.
AP	Access Point.
CoAP	Constrained Application Protocol.
CPU	Central Processing Unit.
DDoS	Distributed Denial of Service.
DL	Deep Learning.
DoS	Denial of Service.
DT	Decision Trees.
FC	Fog Computing.
FN	Fog Node.
HTTP	Hypertext Transfer Protocol.
IE	Information Element.
IETF	Internet Engineering Task Force.
IoE	Internet of Everything.
IoT	Internet of Things.
IP	Internet Protocol.
IPFIX	IP Flow Information Export.
LR	Logistic Regression.

ML	Machine Learning.
MLP	Multi-Layer Perceptron.
MQTT	Message Queuing Telemetry Transport.
NB	Naïve Bayes.
PCAP	Packet Capture.
QoS	Quality of Service.
RDP	Remote Desktop Protocol.
RF	Random Forest.
ROC	Receiver Operation Characteristic curve.
TCP	Transmission Control Protocol.
UDP	User Datagram Protocol.
VLAN	Virtual Local Area Network.
Wi-Fi	Wireless Fidelty.

REPORT

1

INTRODUCTION

Nowadays, data is present everywhere in many shapes, sizes, structured and unstructured, and is increasingly becoming a key asset for people and organizations. Driven by new communication networks, people are getting used to data-based infrastructure, and this is driving research more towards machine learning-based applications along with IoT [16]. The Internet of Things (IoT) can be viewed as a global network that provides the communication between human-to-human, human-to-things, and things-to-things by making a unique identity for each object [1]. Being present in various environments, such as our homes, cities, organizations, or even in people, it has become one of the great data generators in the world today. The IoT objects many times perform crucial tasks, interacting automatically with their surroundings and the Internet automatically and independently [1], as a result, attack and anomaly detection in IoT infrastructure is a rising concern in the domain of IoT [16]. For example, the existence of a Denial of Service (DoS) attack will disrupt client systems compromising the service availability[22].

With this type of scenario emerges the need to have a robust cybersecurity solution that guarantees the protection of the IoT devices and of the networks where they are in. Only then we can give the end-user the trust he needs in the use of IoT and obtain, like in any other network, the confidentiality, integrity, and availability of the data. Encryption is an approach that could help us to preserve our networks. Encrypting our IoT traffic and guaranteeing a secure communication channel is one of the mandatory features that should exist in any network. But unfortunately, objects in IoT networks are seriously resource-constrained, being difficult to implement heavy operations on them which are required by ciphering algorithms [1]. On the other hand, even if we were able to use encryption, we will not have the possibility to see if the network is under attack or if a device is compromised and generating suspicious traffic. Providing a network with attack or anomaly detection capabilities is one of the approaches to be considered.

There are devices like Intrusion Detection Systems (IDS) that scan the network looking for signature attacks or anomalies, alerting the network administrator in case of registering something potentially dangerous for the network. An IDS is

important in the cybersecurity field for achieving a solid line of protection against cyber adversaries [22]. But the traditional knowledge-based IDS must now give way to intelligent, data-driven systems [30].

Machine Learning (ML) is a branch of data science and is a subset of artificial intelligence techniques, that applies algorithms to extract patterns by using mathematics, statistics, optimization, and knowledge discovery methods [29]. ML frameworks can perform complex tasks by learning from information rather than following pre-programmed rules [20], exactly what we need for the upgrade of intelligence and adaptability based on data for an IDS.

Given the above, this dissertation aims to present a cybersecurity solution based on an IDS, associated with a Machine Learning (ML) module, which will help to improve its anomaly detection skills in an IoT network flow. The network flow will be represented through the IP Flow Information Export (IPFIX) [9] protocol. As a use case, it will be considered the IoT environment of the smart city where is proposed an IDS architecture regarding the technologies used and the placement of the IDS. In sum, the objectives of this dissertation are the following:

- Define the architecture in an IoT environment of an machine learning-based IDS.
- Test various ML analysis models and techniques, choosing the most suitable set to be implemented.
- Anomaly detection based on data flows described by the IPFIX protocol.

Regarding existing research, most of the verified works are based on only one type of ML, or only on one type of classification using ML. Many focus on the accuracy of the ML models, but depending on the topic at hand, this metric is not always the one that defines the model's performance as being the best. The use of datasets unrelated to the environment where the IDS is located is also one of the situations verified. There are also works where an IDS supported by ML models is mentioned, but without any indication of the IDS architecture, how it will function, its location on the network, or how the data to be analyzed will be collected and in what format. Based on the described, in this dissertation, our contributions are:

- Framing an machine learning-based IDS for IoT, in a real applicability scenario with a detailed description of how it works, from traffic capture to anomaly detection.

- The comparison of several ML models using supervised and unsupervised models, including the use of deep learning (DL) models versus the traditional ML models.
- The use of more than one data classification technique (binary and multi-class).
- Evaluation of a real world dataset from an IoT environment, using network flows to determine anomalies.

To accomplish our intents we have structured this dissertation into five chapters that will present the state of the art of our research, the environment chosen for the use case, the experimental results, and consequent conclusions. Chapter 2 presents the state of the art, through the theoretical concepts of the themes that make up our dissertation, as well as the summary of the research carried out to achieve our proposal. In chapter 3, the network of a smart city is described and is where the architecture of our security solution is presented. In chapter 4, we have our experimental results, where the results obtained are compared and discussed. And finally, in Chapter 5, we have identified the points for improvement and suggestions for future work, ending with the conclusion of the work developed.

2

STATE OF THE ART

There are massive quantities of data from applications, servers, smart devices, and other cyber-enabled resources generated by machine-to-machine and human-to-machine interactions [5]. In this context, data processing becomes complicated for people, computers and systems, who will look for tools and analysis techniques to help in this type of task, likewise, ensuring data and network protection is a challenge to be addressed. The communications data flows will be presented in the most varied sizes and formats, for which a solution with high computational resources will be needed, so that they can be processed and analyzed accordingly. The usage of data science can help in the study of large data volume and diversity environments, by correlating events, identifying patterns, and by detecting anomalous behavior [5] that, otherwise, would remain hidden. Cybersecurity systems can also take advantage of the characteristics of data science, to develop increasingly robust attack detection methods and solutions.

2.1 INTERNET OF THINGS

The IoT paradigm is one of the drivers for the new generation of communication networks, combining a wide variety of hardware and software that provide customers easy-to-use experience and low-cost solutions. In this kind of environment, there is a large data volume and diversity, mainly controlled and generated by IoT devices. These devices are present in our cities, homes, industry, healthcare facilities, vehicles, personal gadgets and could perform critical tasks in those areas. At the same time, they deal with all sorts of data in which we can find confidential and private information regarding their users.

2.1.1 *Smart Cities*

The smart city concept has become a reality with the help of the IoT, as a consequence, the smart city became one of the major drivers for the IoT applications [27].

Cities started to provide information to their citizens like car parking availability, transportation routes or schedules, traffic congestion, but also critical services like environmental disasters detection, energy or water supply management, street surveillance, among others. All to provide a better life quality, in a controlled and secure environment and to prevent the waste of resources. Due to this context, the security researchers look for the smart city environment with interest, challenging themselves to ensure the availability of the services and, at the same time, to ensure the privacy and integrity of data in this kind of network.

Figure 1: IoT Smart City

Being directly related to the IoT, the smart city also shares its network architecture. There are several proposals for its representation, but from our point of view, the one that best exposes the network architecture of a smart city is the approach represented by four layers [12], in which we have the perception layer, the network layer, the support layer and the application layer (Figure 2).

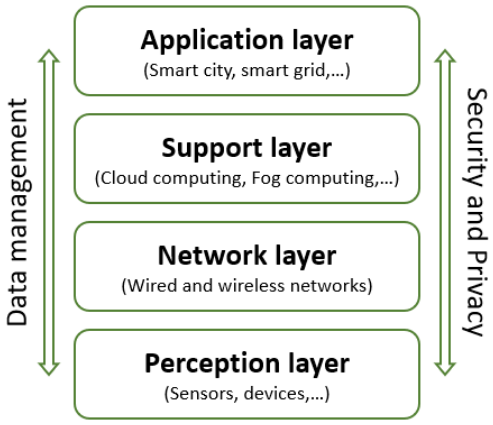


Figure 2: IoT based architecture for a smart city [12]

Perception layer

The perception layer is the source of the data. It includes the sensors, the meters, and all the IoT heterogeneous hardware that is deployed across the city, to achieve the intended services.

Network layer

In the network layer, we found all the technologies that will enable the communications between devices and the remaining layers. Wi-Fi, 5G, Bluetooth, RFID, Ethernet will provide connectivity to all the devices and allow the availability of the services.

Support layer

The support layer will provide services to the IoT smart city context and their applications. Here we can deploy the command center of the network, with the aid of computer resources and techniques, this is one of the most important layers and where the network monitoring and issues detection can be done.

Application layer

The application layer is the top layer and will provide to the end-user all the information regarding the data collected in the perception layer of the smart city.

2.1.2 *IoT Challenges*

Despite handling and generating a large volume of data, generally, IoT devices are cheap and, due to this fact, they have low CPU capacity, low storage, and low memory resources. Similarly, the software implemented in the IoT devices is based on open-source solutions or in obsolete software and sometimes even with faulty software. In a worst-case scenario, we would have one IoT device plugged in a network with software with no security patches or updates, without knowing which ports are open and visible from the internet, and with no access

control or credentials. Being connected to public networks, the IoT devices can be vulnerable to attacks if there are no security measures in place, producing unexpected behaviors in the private networks, compromising services availability, data confidentiality, and the user's privacy. Some of the most frequent attacks these days are the following:

Denial of Service

A Denial of Service (DoS) is an attempt by an attacker to prevent legitimate access to websites by overwhelming the amount of available bandwidth or resources of the computer system [22]. This type of attack typically evolves to a Distributed Denial of Service (DDoS) when is registered several types of equipment in different locations or computers systems are producing this type of attack against a specific target.

Botnets

A botnet denotes the number of hijacked computer systems remotely operated by one or many malicious actors which coordinate their activities by Command and Control (C&C) [22]. These botnets are many times responsible for DoS or DDoS attacks, where the owner of a hijacked device isn't aware of being part of an attack on a computer system, denoting only some slow processing of its device.

Brute force

Brute force endeavors to illegally obtain pairs of user names and passwords by trying all predefined pairs to gain access to network services, with automated applications often used to guess password combinations [22]. When this type of attack succeeds the device is under control of a malicious actor that now can access all the data that is processed by the device, gaining visibility over the victim network. This device can be transformed into a bot or perform DoS attacks on other devices or computer systems.

Ransomware

Ransomware is malware that harms computer and network systems by encrypting computer resources and blocking access till a ransom is paid [22]. This type of

attack could be the result of a brute force attack, compromising service availability and data access.

Taking into account the challenges and constraints presented, it will be necessary to adopt a security solution that will help protect these IoT environments.

2.2 INTRUSION DETECTION SYSTEMS

One solution can be an intrusion detection system or IDS, which is a tool or mechanism used to detect non-authorized accesses and attacks against systems, analyzing the communications, internal activities, and other events, making this type of security solution very important for any type of networks or systems [25]. Considering that the IDS has the information source based on network information and that it will detect any anomaly or attack in real-time, these systems must be strategically placed in several locations of the network, working with different methods of detection.

2.2.1 *IDS deployment strategy*

Aiming to have a centralized architecture, the IDS will rely on multiple sensors that will collect information about the devices installed inside the network and oversee the communications made between them and the rest of the network. The IDS sensors can be placed as host-based, network-based, or in a hybrid approach.

Host-based

The host-based approach is centered on a device, having a real-time perception of what is happening on that network node and with a view of what flows and communications the device participates in on the network. This approach depends highly on the device's capabilities of processing and exporting the data flows.

Network-based

The network-based approach has a broader view of the network, converging multiple hosts and dealing with a greater amount of data, being regularly deployed in the network gateways.

Hybrid

The hybrid approach tries to take advantage of the best of the two worlds of the host-based and network-based approaches. To do this solution it must be created clusters of hosts or VLANs aggregate hosts with similar traffic properties then, when setting up an IDS in the gateway of the network and the subnetworks gateways, we can achieve the broader view of the network-based approach. Additionally, with the host-based sensors is placed on the lower levels of the network, the traffic will be inspected by the gateways providing a better precision of analysis. At the same time, when deploying both host and network sensors we can assure some redundancy to the IDS in case of a gateway or a device going down.

2.2.2 *Detection techniques*

In addition to the IDS deployment location, it is necessary to adjust the type of detection that will be performed. There are several detection techniques related to IDSs, such as signature-based, anomaly-based, specification-based, and hybrid.

Signature-based

The signature-based or the specification-based IDSs have on the system preconceived data to look for on the network that when matched raised an alert. This produces a low number of alerts but on the other hand, if exists a new threat or attack is not listed on the IDS database as a specification or signature the IDS will not detect this new behavior inside the network.

Anomaly-based

An anomaly-based IDS can detect abnormal behaviors or anomalies when comparing network traffic with expected communications or with expected normal behavior that was previously registered inside the network. In this case, will see a raised number of alerts but compared with the signature and specification-based approaches new threats or attacks could be detected on time.

Hybrid

The hybrid approach will combine more than one technique, trying to avoid that new attacks or threats going undetected and to reduce the amount alerts that will make a network administrator lose the focus of the important ones.

But, to adapt an IDS, regarding his implementation and how we will proceed with the detection of intrusions, we need to have a notion not only of the network where we are inserted but also what type of data we will analyze.

2.3 IPFIX

As mentioned above, there are several flows within the communication networks and due to their large volume, there was a need to standardize them. IPFIX [9] [10] [11] stands for IP Flow Information Export and is an IETF protocol that was born due to the need of having a common universal standard for exporting IP flow information from network devices and probes that facilitates services such measurement, accounting, and billing. This protocol is used to transport information elements (IE) that allow network administrators to have a broader view of the traffic that flows in and out of the network. IPFIX can be deployed across network elements like routers where it performs passive flow measurements [33]. The IEs are grouped into 12 groups according to their semantics and their applicability [10] [11]:

1. Identifiers
2. Metering and Exporting Process Configuration
3. Metering and Exporting Process Statistics
4. IP Header Fields
5. Transport Header Fields
6. Sub-IP Header Fields
7. Derived Packet Properties
8. Min/Max Flow Properties
9. Flow Timestamps
10. Per-Flow Counters

11. Miscellaneous Flow Properties
12. Padding

With IPFIX IEs a network admin will be able to answer the following questions:

Who?

Who is originating a network flow, or which devices are intervening in this communication? This is an easy question to answer when looking at fields such as the source and destination IP addresses of packets flowing through the network. In the IPFIX flows we have access to this information through their IEs sourceIPv4Address (8), sourceIPv6Address (27), destinationIPv4Address (12), and destinationIPv6Address (28).

What?

What application is generating this flow? Through its IEs, IPFIX intends to describe a given application, depending on the transport ports used and the protocol or IP version used. When describing the application's transport protocol, some IEs allow us to check the flags, type, codes, and characteristics of the protocol used. Some of the IEs that can be used to retrieve this information are as follows:

- destinationTransportPort (11)
- applicationName (96)
- sourceTransportPort (7)
- ipVersion (60)
- flowEndReason (136)
- protocolIdentifier (4)
- udpSourcePort (180)
- udpDestinationPort (181)
- tcpSourcePort (182)
- tcpDestinationPort (183)
- icmpTypeIPv4 (176)

- icmpCodeIPv4 (177)
- icmpTypeIPv6 (178)
- icmpCodeIPv6 (179)
- tcpSequenceNumber (184)
- tcpAcknowledgementNumber (185)
- tcpWindowSize (186)
- tcpUrgentPointer (187)
- tcpHeaderLength (188)
- totalLengthIPv4 (190)
- ipPayloadLength (204)
- ipTotalLength (224)
- ipTTL (192)
- udpMessageLength (205)
- ipv4IHL (207)
- ipv4Options (208)
- tcpOptions (209)
- paddingOctets (210)
- isMulticast (206)
- ipDiffServCodePoint (195)
- ipPrecedence (196)
- fragmentFlags (197)

Where?

Where did this flow take place? Determining the input and output ports of the flow is information that can be useful for a network administrator to be able to understand where a given flow is occurring. Although we have previously managed to answer which machine is at the origin of the flow, it is necessary to determine within the equipment which ports are involved in its routing, its direction, or the network segment where the flow is inserted. Some of the IPFIX IEs that can return this information are the ingressInterface (10), the egressInterface

(14), the sourceMacAddress (56), the destinationMacAddress (80), the vlanID (58), the flowDirection (61), the observationPointId (138) and the flowId (148).

How?

How do a flow is working? What are its characteristics? IPFIX manages to describe the functioning of a flow-through quantity and statistical indicators that allow determining how the application behaves or how the flow in the network is affected. The following IEs record quantities and can give us visibility into anomalies that may be occurring in the network. Situations such as dropped packets or their significant increase within the network can be indicators of ongoing computer attacks.

- octetTotalCount (85)
- packetTotalCount (86)
- droppedOctetTotalCount (134)
- droppedPacketTotalCount (135)
- ignoredOctetTotalCount (165)
- ignoredPacketTotalCount (164)
- tcpSynTotalCount (218)
- tcpFinTotalCount (219)
- tcpRstTotalCount (220)
- tcpPshTotalCount (221)
- tcpAckTotalCount (222)
- tcpUrgTotalCount (223)
- observedFlowTotalCount (163)
- ingressUnicastPacketTotalCount (354)
- ingressMulticastPacketTotalCount (355)
- ingressBroadcastPacketTotalCount (356)
- egressUnicastPacketTotalCount (357)
- egressBroadcastPacketTotalCount (358)
- layer2FrameTotalCount (431)

When?

When did this happen? By determining when a given flow has occurred on the network, it allows the network administrator or a forensic audit to temporarily mark a given event. The IEs that will indicate the beginning and end of a given flow, and that is fundamental to understand when a given anomaly has occurred in the network, are the flowStartMilliseconds (152), the flowEndMilliseconds (153), the flowActiveTimeout (36), the flowIdleTimeout (37), the flowDurationMilliseconds (161) and the observationTimeMilliseconds (323).

Currently, IPFIX flows are described by 444 fields [10] (shown on table 17 of Appendix A). This protocol allows its configuration and customization, exporting only the IEs to perform the intended task. IPFIX IEs allow us to implement simple algorithms and impose thresholds that could define a network anomaly. As it is a light and flexible protocol, IPFIX can be deployed in IoT networks, contributing to its security. Successful detection of brute force attacks can be done on the data available in the IPFIX [33].

2.4 MACHINE LEARNING

Regarding the amount of data generated by several sources like computer systems and network devices, files like computer logs became overwhelming to human analysis and it's here where ML became essential to help digest all this information by spotting patterns, behaviors, and anomalies inside the datasets. The ML uses mathematical techniques [17], to build their models having two types of learning associated, supervised and unsupervised learning. The main objective of machine learning is to allow a system to learn from the past or present and use the knowledge to make predictions or decisions regarding unknown future events [2].

Supervised Learning

Supervised learning is based on learning from labeled data, which means that training data includes both the input and the desired results [6]. It tries to identify automatically rules from the available datasets and define various classes, and finally predicts the belonging of elements (objects, individuals, and criteria) to

a given class [17]. One of the challenges of this type of learning is to get proper labeled datasets, due to the specificity of the networks is recommended to train the models based on the scope of the project. For instance, if the ML model is to be deployed in a IoT network, the training of the models should be done with a IoT labeled dataset. Algorithms like Logistic Regression (LR) and Naïve Bayes (NB), belong to the family of probabilistic classifiers that are used to predict a target variable, being two of the simplest ML algorithm that we can use for classification problems. Another type of simple ML algorithms are the Decision Trees (DT), as figure 3 shows, based on the input data, the tree has several node leaf or decision nodes that classify the data, in a binary form, until the last node return the outcome.

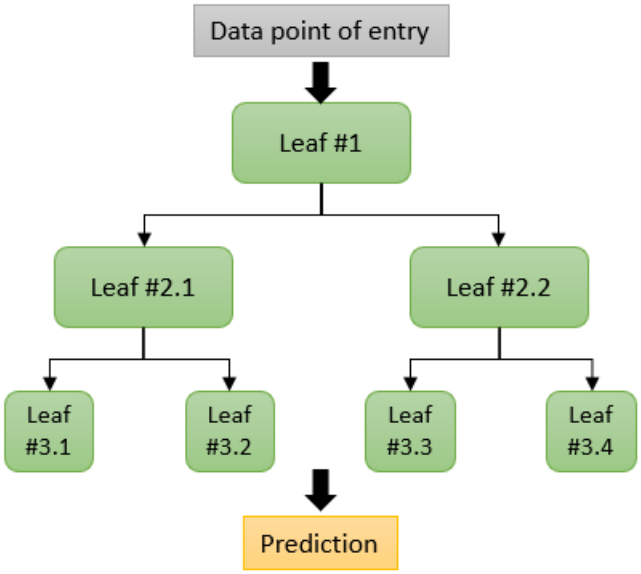


Figure 3: Decision Tree example

By taking advantage of the Decision Tree, the Random Forest (RF) algorithm aggregates a number of trees, that work independently with the same data entry, assembling a committee of trees (Figure 4). This correlation of classifiers will make a more resilient model, protecting the trees of individual errors.

Unsupervised Learning

Unsupervised learning is based on clustering the input data in classes on the basis of their statistical properties only [6] (Figure 5). This means that no labeled dataset is need, it will be the machine alone that will determine the class of the network traffic that is present to it by choosing a cluster to put it. The number of clusters

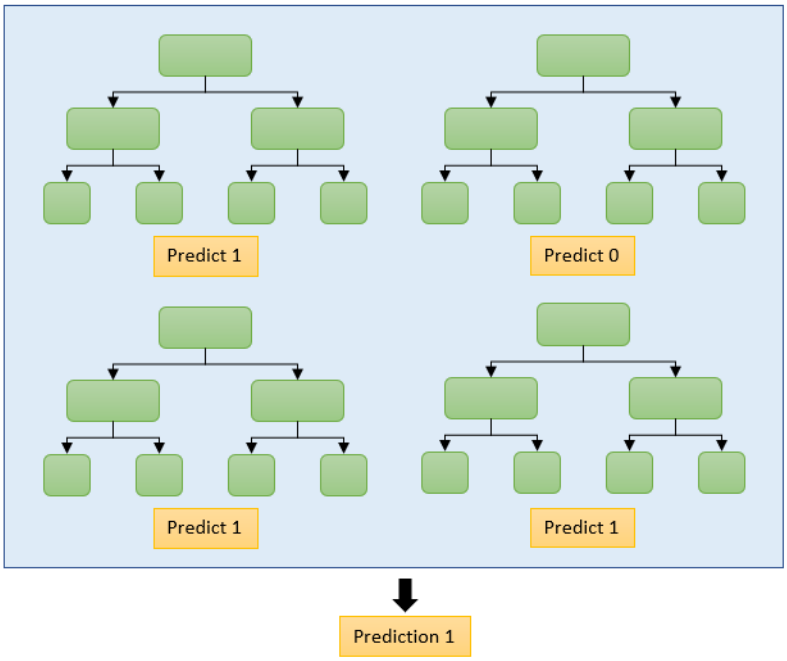


Figure 4: Random Forest example

can be defined by the ML model programmer. As an example, K-Means is an algorithm used for unsupervised learning to classify unlabeled data into several clusters based on similarities.

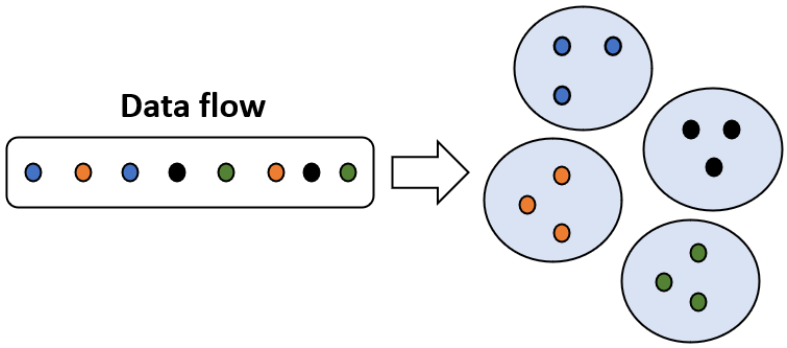


Figure 5: Unsupervised learning clusters

Deep Learning

Deep Learning (DL), is part of a broader family of ML methods based on learning high-level abstract representations [6]. DL groups generic algorithms mimicking the biological functioning of a brain without being intended for a specific task [6]. Technically, DL is the application of artificial neural networks (ANNs) which contain multiple hidden layers [6] (Figure 6), it achieves good performance and

flexibility by learning to represent the data as a nested hierarchy of concepts within layers of the neural network [7]. An advantage of DL over traditional ML is its superior performance in large datasets [15]. A Multilayer Perceptron (MLP) is one of the simpler DL algorithms. The MLP has an input layer to receive data, an output layer that makes a decision or prediction about the input, and in between those two, an arbitrary number of perceptrons or hidden layers that are the true computational engine of the MLP. By definition, a perceptron is a device capable of computing all predicates which are linear in some given set ϕ of partial predicates [21], in other words, is a simple algorithm intended to perform binary classification.

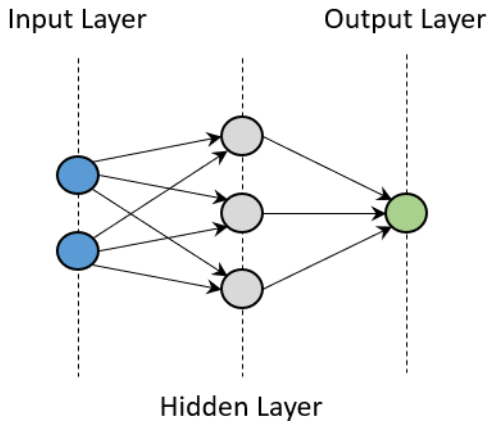


Figure 6: Simple neural network

Performance metrics

To verify the ML model feasibility performance metrics are put in place to help understand if the model has been trained correctly and if it is suited for the task at hands. The most suitable indicator depends on the problem of interest, in our case we face a classification problem, classifying the network flow as benign or malicious. If we look at it as a binary classification problem, we will have 4 outcomes:

- True negatives: correctly predicted negatives (zeros)
- True positives: correctly predicted positives (ones)
- False negatives: incorrectly predicted negatives (zeros)
- False positives: incorrectly predicted positives (ones)

The ROC (Receiver Operation Characteristic curve) verifies if we are towards a random classifier or not. The ROC will show the performance of the classification model at all classification thresholds, and it will have 2 parameters (True positive rate and False positive rate). The true positive rate is a synonym for recall, and is defined in the following Equation:

$$Recall = TPR = \frac{TP}{TP + FN} \tag{1}$$

Where TP is the number of true positives and FN the number of false negatives.

Being the False Positive Rate defined by:

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

Where the FP is the number of false positives and TN the number of true negatives.

Having a classification problem in hands, for our network is more harmful if we misclassify a flow as benign when in fact the flow is malicious the metric that must be prioritized is the recall metric. However, most academic papers pay attention to the precision or accuracy metrics. The precision metric shows us how accurate the model classifies a flow:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

When combining these two metrics, if we have a high precision plus a high recall, we can state that the prediction model is highly dependable, which means that the model doesn't misclassify benign flows and doesn't wrongly leave out the malicious flows. In another hand low recall with a high precision rate doesn't generate lots of false positives but will miss out on lots of malicious flows, as consequence this kind of model cannot be used to perform critical tasks like the one, we have because it could create catastrophic repercussions to our network. Lastly, when having a high recall with a low precision although the model will be able to detect most of the malicious flows, we will also face lots of false alarms which can create entropy in our security system.

To reach the full potential of the IDS and the analysis power of the ML models, it is also necessary to look for a computational solution that will guarantee us the necessary resources for data processing.

2.5 FOG COMPUTING & FOG NODE

Fog computing (FC) is the extension of cloud computing towards the network edge to enable cloud-things service continuum [13] (Figure 7). This paradigm is used to reduce the energy consumption and latency, for the heterogeneous communication approaches, in the smart cities applications [27]. Due to its edge location, it can provide network context information, such as local network condition, traffic statistics and client status information, which can be used by fog applications [28].

By having location awareness, the FC allows the distribution and installation of network nodes, with all the characteristics associated with FC, to offer local network services and resources if need. These network nodes are called Fog Nodes and allow for closer contact with the data source. For instance, in fog computing, an IDS can be deployed on the fog node system side, to detect intrusive behavior by monitoring and analyzing log files [28].

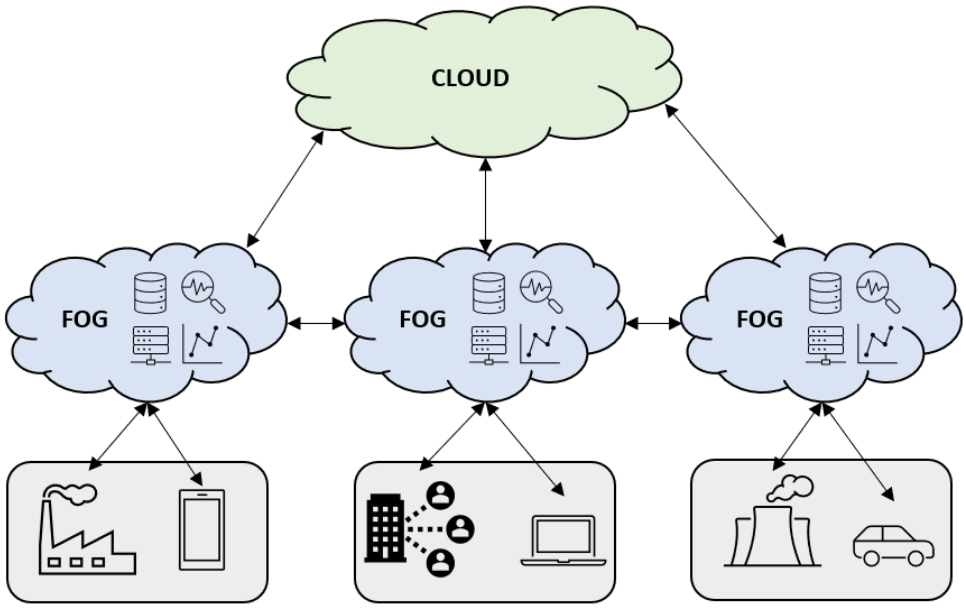


Figure 7: Fog computing example

2.6 RELATED WORK

Taking into account the state of the art and the various topics covered, a lot of work has already been developed to find solutions to protect IoT environments, using an intrusion detection system and data analysis using machine learning.

Ijaz et. al [18] give three information security perspectives to help to identify the correct and feasible solutions in a smart city context. As the figure 8 shows, we will have governance, technological and social/economics factors that will influence and identify the information security issues of a smart city. From a governance point of view, it must exist a way of mange and maintaining the whole infrastructure where is essential a proper implementation of what will be the core of the smart city functioning, also in governance critical infrastructure must be identified to ensure critical services availability and protection. Social/Economic factors will determine the number of services offered and how can be the people, commerce, and enterprises of the city be protected and ensure that their privacy remains intact. The technological solution will also raise questions from a security perspective, the choice of hardware and software and communication type between machines, users, and maintenance will be crucial to adopt security measures. To all this work together, is mentioned that the bridge between governance, social/economics, and technology will be the telecommunications sector that will provide the needed connectivity and support. Being the door to the world of this type of solution, the telecom service providers must also have security services and warn their customers of any security incident that could impact the smart city's normal functionality. Despite providing a comprehensive overview of threats, vulnerabilities, and some solutions when suggesting some encryption protocols and the deployment of firewalls or antivirus, Ijax et. al [18] present no practical implementation of a security solution.

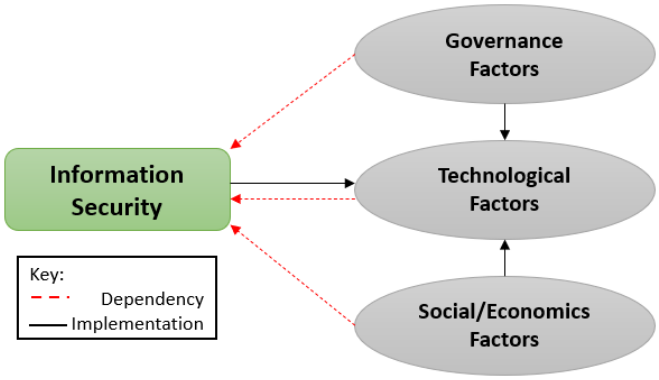


Figure 8: Information security factors[18]

In Cui et al. [12], the authors describe an IoT-based architecture for a smart city (Figure 2) which is crucial to understand all the network layers involved in this context. Concerns regarding privacy and availability are also mentioned, where is referred that the smart metering infrastructure can monitor the private lives of residents or how botnet activities can take the IoT devices as hostages and then perform DDoS attacks which will impact the ecosystem and the availability of the services. Various security and privacy protection mechanisms are presented and as the previous authors, cryptography is present as a solution to guarantee confidentiality but due to the limited CPU resources of IoT devices and the lack of lightweight solutions, the cryptography itself will not be sufficient. In this survey, the blockchain technique is also aborded once the decentralized feature of blockchain enables applications to operate in a distributed manner, which is the main reason behind the popularity of many blockchain-based IoT applications. Besides blockchain, ML is viewed as a technique that will improve traditional intrusion detection systems in the protection of the network. Nevertheless, the authors do not present any practical implementation nor how could be implemented a security solution.

With the IoT-based architecture for smart cities in mind (Figure 2) we can verify the existence of the support layer that will work very closely with the application layer and will provide support for the requirements of diversified applications via intelligent computing techniques [12]. One of these intelligent computing techniques is named fog computing in which Singh et al. [27] present a blockchain and fog-based architecture for the Internet of Everything in smart cities. Fog computing is like cloud computing but is an extension of the cloud to the edge of the network from which it receives the data reducing the communication latency due to the data source proximity but keeping all the remaining attributes providing storage, computational power, and applicational services to the end-users. The suggested architecture is based on a multi-layered solution where data sources are connected to fog nodes that handle the IoE devices traffic and then pass it through to the application layer. The intercommunication takes place between connected devices and security is provided with Blockchain technology. This solution focuses on the scalability, energy consumption, and reduced latency provided to the network.

Elrawy et al. [14] states that many IoT applications can run in real-time which means that network delay and latency will affect their performance. The authors also refer to the need for powerful security measures in the IoT network due to life-threatening attacks that can occur in smart environments like e-health

systems. The security solution must at the same time protect the IoT network and its resources without impacting the system performance or user privacy. The study relies upon an Intrusion Detection System where interoperability and standardization issues must be considered in terms of design. The operation of an IDS is described as a 3-stage process that includes a monitoring phase that relies on sensors that could be network or host-based, the analysis phase where features extraction methods or pattern identification methods are performed, and finally the detection stage where anomaly or misuse are detected. Additionally, is stated the importance of the placement of an IDS in an IoT network cause this matter will affect the overall efficiency of the IDS, while a centralized strategy offers centralized management a distributed approach has the advantage of reducing the amount of traffic monitored and increasing processing capacity. Despite several recommendations like the proposal of an IDS based on a hybrid intrusion detection technique to detect different types of attacks from different computational environments no practical implementation is made. Like Elrawy et al. [14], Zeadally & Tsikerdekis [32] discuss the use of traditional network monitoring, like Intrusion Detection Systems with the help of machine learning algorithms to give a viable alternative to existing IoT security solutions. They summarize the needs of host-based and network-based approaches to perform traffic capture of the network using machine learning to process the data. In this case, no solution is given as optimal, highlighting only the strength and limitations of the machine learning algorithms regarding the IoT devices characteristics.

Chaabouni et al. [6] present a comprehensive survey where it pointed out the design challenges of IoT security and classification of IoT threats, giving to the readers several options about how to build a Network Intrusion Detection System for IoT based on ML techniques. As future research directions are stated the advantages of exploring the edge and fog computing paradigms that will give the ability to push the intelligence and processing logic employment down near to data sources. To train and deploy an IoT NIDS based on ML a real-world IoT-dedicated dataset is needed. Diro & Chilamkurti [13] present a distributed attack detection scheme using a deep learning approach for IoT and place their IDS based on deep learning in the fog network. The fog nodes are responsible for training models and hosting attack detection systems at the edge of the distributed fog network since they are closer to the IoT data layer, here, it is also used a central node that updates the parameters of each cooperative node and propagates the resulting update back to the worker nodes. Regarding the model training, no IoT-based dataset was used.

In Habeeb et al. [2] its highlighted that adoption of real-time architecture for the smart city will assure effective and seamless communication among sensing devices within the smart city infrastructure. It also includes the quality of services support in the network, which is extremely crucial for the real-time application for smart cities. Due to big data production in several environments, Habeeb et al. investigated real-time big data processing and ML with the possibility of anomalous detection where the most used anomaly detection techniques are described. Chalapathy & Chawla [7] add that anomaly detection as better performance with the use of deep learning or deep neural networks being suitable for IoT big data anomaly detection and intrusion detection systems but this type of solution can add computation complexity depending on the deep learning models that can be implemented.

To complement, Moustafa et al. [22] made a holistic review about network anomaly detection systems where ML and deep learning techniques are presented and where all the components of a NADS are described. The authors suggest that combination-based approaches get better and effective NADS but again from a real-time point of view this could be a problem due to the increase of processing time.

Austin [3] used the IoT23 dataset in his trials to answer which ML model that performs the best in terms of classification accuracy, recall, precision, F1 score and to discover which features have the best predictive power in the dataset. Additionally, he [3] has extensible described the methods of how the dataset was collected, all the types of attacks used in the network, and how the ML approach helps to classify traffic in the dataset. The analysis made focuses only on the binary classification problem, which means the traffic in the trials was simply classified as benign or was malicious.

After considering the works carried out and the state of the art previously presented, in this dissertation the goal is to present a realistic scenario combining an IoT environment and a cybersecurity solution to address part of the IoT security challenges. As a scenario, it was chosen a smart city environment where is proposed an integrated security architecture with an IDS in place inside the smart city network that will be supported by fog nodes. To do this and regarding the amount of data produced by the IoT devices, is intended to deploy an IDS gifted with an ML model that will help the system to spot anomalies and to be a more robust solution than a traditional IDS. Some ML models will be compared

among them, and more than one classification technique will be used to have more detail about what is occurring inside the network.

3

STUDY SCENARIO

To present a cybersecurity solution for an IoT environment like a smart city we have to analyze the four layers that represent the architecture of the smart city (Figure 2) [12]. Starting in the perception layer, where we have the sensors, we assume that it will be possible to segment the IoT network traffic flow based on the application messaging protocols, like MQTT or CoAP, or simply by service. This type of segmentation or VLANs will allow us to create observation points (OP) of each service, is simpler to perform network analysis, and understand network behaviors. The network segmentation is explained by the different needs of each service, for instance, if we need a reliable connection or a service based on QoS the MQTT protocol can be used, since it relies on TCP as transport protocol and its lightweight features make it one of the most used protocols in IoT environments, however, the MQTT protocol have a low interoperability capability.

In another hand, TCP-based messaging protocol HTTP guarantees high interoperability but since it was built for web applications is utilization cannot be feasible in some IoT solutions due to high resources consumption. From a business point of view, the AMQP is preferred because of its wide range of services related to messaging and lightweight features.Another key feature in IoT applications can be latency and for this case, the use of the messaging protocol CoAP, which is based on the transport protocol UDP, can be preferred being also a lightweight protocol with small messaging overhead ideal to the IoT context [23].

The network layer has the necessary technology to transport data to the support layer. Whether through Wi-Fi APs, Ethernet, or 4G/5G radio links, the gateway of these devices will be set to the support layer node that will provide its services before the data reaches the application layer.

In the support layer, we assume that we will have an infrastructure based on fog computing, capable of providing all the necessary processing and storage capacity resources, being the place through which all the traffic generated not only by the perception layer will pass, but also the traffic originating from the application layer.

3.1 PROPOSED SOLUTION

After this analysis and since the support layer is the nerve center of the network and without any kind of constraints regarding processing power, we believe that this will be the ideal place to place our detection system. Being based on fog computing, it will provide the scalability, and flexibility to implement the fog nodes that will be the gateways for the IoT devices. At the same time, the fog nodes will act as network observation points (OP), by passively capturing the traffic, from which the IDS with the help of the ML module will perform the detection of potential attacks. The proposed architecture is shown in figure 9.

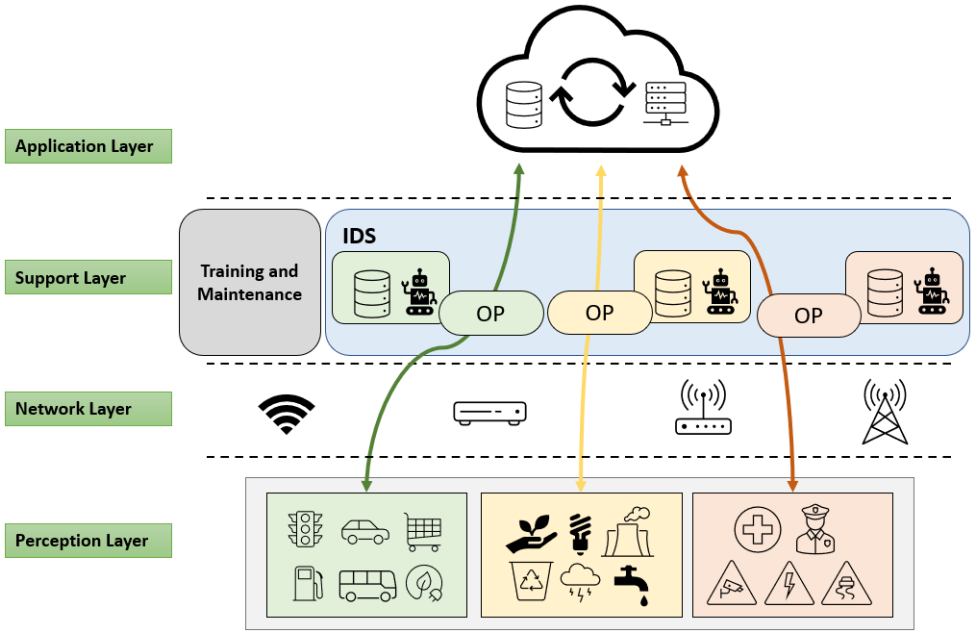


Figure 9: IDS architecture proposal

The fog node will be the network element responsible for observing, storing, transforming, evaluating and forwarding the data generated by the IoT network. When obtaining the network traffic, it will be transformed into IPFIX flows and will pass to a data processing module before being delivered to the ML model that will make the flow classification. The ML model will analyze it and, depending on its verdict, alert the network administrator to the fact that there is a network anomaly. Regardless of the classification result, it will be saved together with the flows and packets that originated it. Likewise, the flow forwarding to or from the application layer is guaranteed since it is just a detection system. The detail of the functioning of the fog node can be seen in the figure 10.

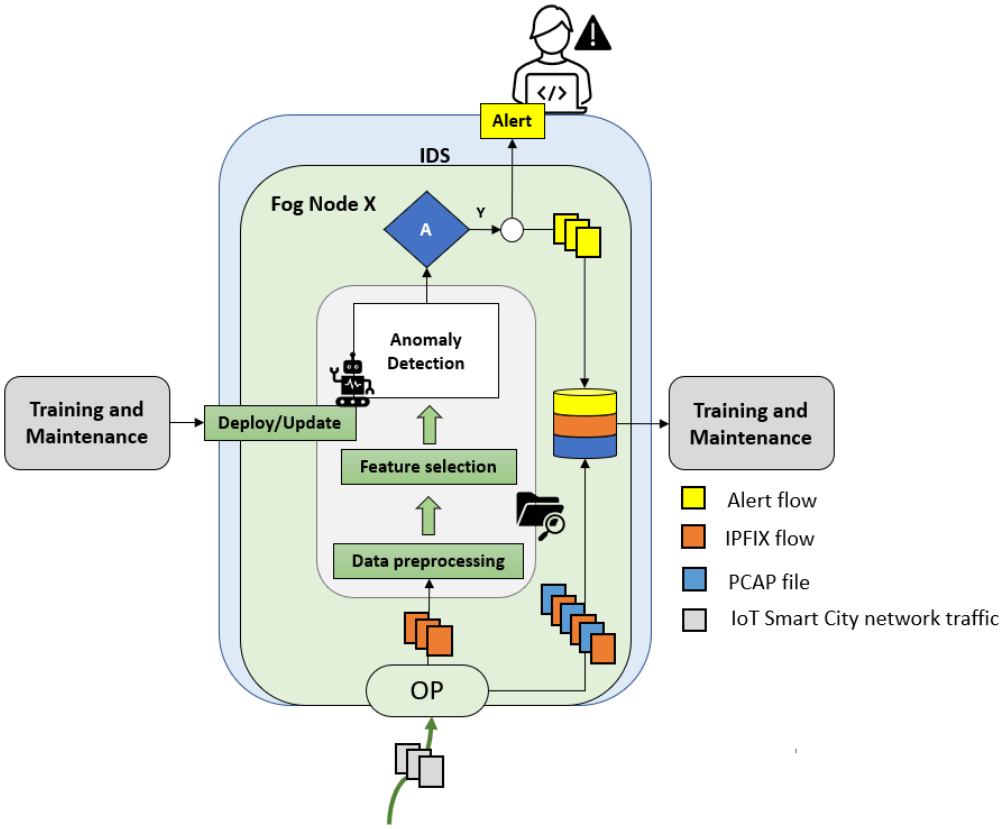


Figure 10: Support layer FN

But before we get into the anomaly detection process, there is some pre-work to be done. When considering the form of supervised learning, the ML module has to be previously trained to be able to distinguish what will be the normal behavior of the network and what will be an anomaly of the same. The training can be carried out through datasets built and labeled in controlled laboratory environments, or through pre-production networks that use protocols or equipment similar to those that will be installed in the smart city. That is why, in our architecture, we considered the Training and Maintenance module described in figure 11, which has a close working process with the IDS fog node.

In this module (Figure 11), the ML models that best detect malicious flows will be chosen, by comparing performance metrics, and later implemented in the production ML module of our IDS. Additionally, it is intended that continuous improvement work is carried out, by testing new ML models, by verifying and using the traffic that is generated in the production network, and by carrying out network audits and forensic analyses.

The idea is that by working together not only does the training module feeds the production IDS but also that the production IDS fog node feedback the training

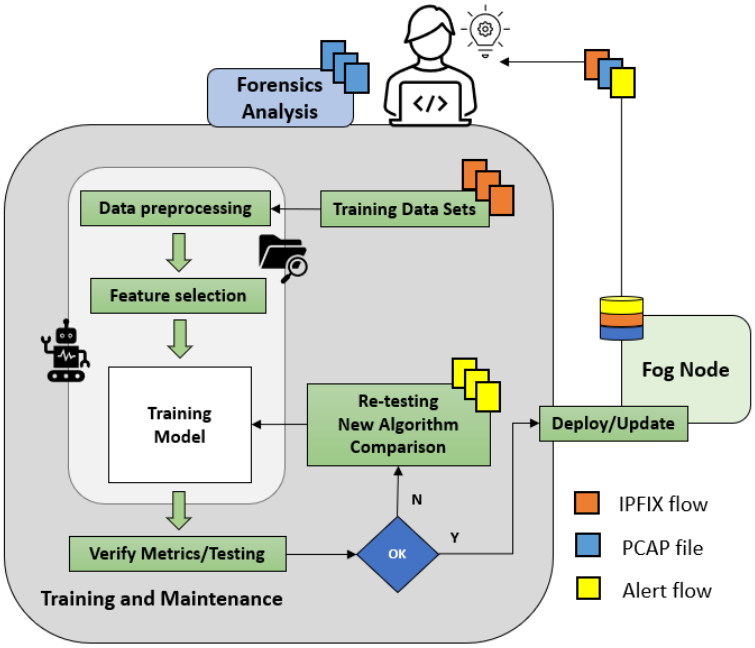


Figure 11: Support layer Training/Maintenence Module

module. When capturing the network traffic the OP will store the PCAP files on a database, alongside the respective IPFIX flows. Also, the features that raise alerts will be stored in the same database (Figure 10). The database will then be replicated to the training and maintenance module, which will be responsible for enhancing the ML models, by feeding the training mechanism with the datasets collected directly from the IoT devices and updating the production ML module if needed. In order to simulate a training process of ML algorithms, we will use a dataset made in the laboratory and which will serve as the basis for our IDS.

3.2 TRAINING DATASET

To evaluate the use of ML in anomaly detection, we looked for a dataset built with real-world traffic. The used dataset was made in the Malware Capture Facility Project of Prague and was retrieved from https://stratosphereips.org [26]. It's a large network traffic dataset from IoT devices, labeled as benign or malicious, to help researchers to develop machine learning algorithms. The IoT23 dataset introduced the columns "label" and "detailed-label" to describe if a flow is part of a network attack or if it is a normal network flow. The malware that produced the network attacks is described in table 1.

Table 1: Dataset Malware description

Malware	**Description**
Mirai	Piece of software that is used to create a botnet.When infected the devices (bots) can be remotely controlled to attack other devices over the internet.These attacks take the form of a DDoS (Distributed Denial of Service) where the bots send traffic to a server, consuming the target resources and forcing the server to stop responding.
Torii	Piece of software that is used to create a botnet. More sophisticated and stealthier than Mirai.Generally targeting IoT devices, with the capability of stealing information from a wide range of devices.
Trojan	Malicious code that looks legit but that can take control of a device being designed to damage, disrupt steal or in general inflict some other harmful action on data or network.
Gagfyt	Does not directly damage the infected device but is used to conduct attacks agaist bigger targets, in another words, this kind of software is used to perform DDoS attacks.
Kenjiro	Hakai variant.The bonet targets Telnet looking for devices with the port open and simple or default passwords.
Okiru	Mirai variant designed to hijack insecure devices that run ARC embedded processors.
Hakai	Based on Mirai and Gagfyt malware, Hakai is used to perform HTTP, UDP, TCP and STD flood DDoS attacks.
IRCBot	Backdoor trojan that allows unauthorized access and control of an affected computer by a remote attacker via IRC.
Linux, Mirai	Trojan that is specifically developed to target Linux-based devices and conduct DDoS attacks.
Linux, Hajime	Appears to be similar to the Wifatch malware that in appears to attempt to secure devices.Also is more advanced than Mirai.
Muhstik	Uses popular web app exploits to compromise IoT devices and mine cryptocurrency.Is an infamous cryptocurrency mining botnet.
Hide and seek	Has quietly been creating a robust botnet of IoT devices using advanced communications methods.This strain of malware cannot be removed by simply restoring the infected device to its original factory settings.

The flows were captured through a passive open-source network traffic analyzer called Zeek, formerly known as Bro. Zeek has an extensive set of logs to describe network activity and in this case, are provided the log files named “conn.log”

or in other words the connection log. Zeek's main data structure is a connection that follows typical flow identification mechanisms, such as 5-tuple approaches, this structure consists of the source IP address/port number, destination IP address/port number, and the protocol in use. The provided dataset is made by 23 columns to represent the communications made inside the network that are reflected in table 2.

Table 2: Zeek fields description

Zeek filed	**Field Description**
ts	time stamp of the captured flow
uid	Unique ID of Connection
id.orig_h	source IP address
id.orig_p	source communication port
id.resp_h	destination IP address
id.resp_p	destination communication port
proto	protocol type
service	service type
duration	flow interval
orig_bytes	count of source generated bytes
resp_bytes	count of destination generated bytes
conn_state	string containing the connection state
local_orig	Boolean flag that indicates if the flow was generated in the source
local_resp	Boolean flag that indicates if the flow was generated in the destination
missed_bytes	count of missed bytes
history	connection state history
orig_pkts	count of packets from the source
orig_ip_bytes	count of IP bytes from the source
resp_pkts	count of packets from the destination
resp_ip_bytes	count of IP bytes from the destination
tunnel_parents	If tunneled, connection UID of encapsulating parent(s)
label	labels the flow as Benign or Malicious
detailed-label	describes the attack type

Zeek flow vs IPFIX

Although Zeek will be used in our experiment, from a theoretical point of view, we can correlate it to the main contender in network flow traffic analysis, which is IPFIX. Being an IETF protocol the IPFIX is widely used, in this way, to give a more general view of the network flow, a comparison was made between the Zeek fields and the corresponding IPFIX elements in table 3.

Table 3: Zeek flow vs IPFIX

Zeek	**IPFIX**	
conn.log	**ElementID**	**Name**
ts	22	flowStartSysUpTime
uid	148	flowId
id.orig_h	8	sourceIPv4Address
id.orig_p	7	sourceTransportPort
id.resp_h	12	destinationIPv4Address
id.resp_p	11	destinationTransportPort
proto	4	protocolIdentifier
service	5	ipClassOfService
duration	161	flowDurationMilliseconds
orig_bytes	231	initiatorOctets
resp_bytes	232	responderOctets
conn_state	136, 218, 219, 220, 221, 222, 223	flowEndReason, tcpSynTotalCount, tcpFinTotalCount, tcpRstTotalCount, tcpPshTotalCount, tcpAckTotalCount, tcpUrgTotalCount
local_orig	149	observationDomainId
local_resp	149	observationDomainId
missed_bytes	165	ignoredOctetTotalCount
history	6	tcpControlBits
orig_pkts	298	initiatorPackets
orig_ip_bytes	1	octetDeltaCount
resp_pkts	299	responderPackets
resp_ip_bytes	1	octetDeltaCount
tunnel_parents	148	flowId
label	n.a.	n.a.
detailed-label	n.a.	n.a.

Table 4: conn.log: conn_state

conn.log: conn_state	
State	**Meaning**
S0	Connection attempt seen, no reply.
S1	Connection established, not terminated (0 byte counts).
SF	Normal establish & termination (>0 byte counts).
REJ	Connection attempt rejected.
S2	Established, ORIG attempts close, no reply from RESP.
S3	Established, RESP attempts close, no reply from ORIG.
RSTO	Established, ORIG aborted (RST).
RSTR	Established, RESP aborted (RST).
RSTOS0	ORIG sent SYN then RST; no RESP SYN-ACK.
RSTRH	RESP sent SYN-ACK then RST; no ORIG SYN.
SH	ORIG sent SYN then FIN; no RESP SYN-ACK (“half-open”).
SHR	RESP sent SYN-ACK then FIN; no ORIG SYN.
OTH	No SYN, not closed. Midstream traffic. Partial connection.

The correlation between the Zeek and IPFIX flow is almost straight forward and for each case, we have the IPFIX feature and correspondent description. We have the source and destination information like the host IP or the host ports, the timestamp, the unique label that identifies the flow, the counters of packets and bytes, local designations of the flow, and the type of service.

Maybe the main difference resides in the conn_state and history features since we are looking for a TCP connection. The conn_state in Zeek is a string that represents the TCP connection state (Table 4). From our point of view, to have the same information as an IPFIX flow, we must correlate several counters to verify the amount of SYN, FIN, RST, PSH, ACK, and URG flags to verify if a given connection is correctly established.

The history feature is the history of the connection state and as we can see in the table 5 it looks for the bits of the flags that we mentioned earlier, so to get that information we can use IPFIX tcpControlBits.

3.3 DATA PREPROCESSING AND FEATURE SELECTION

The log files that represent the network flows were divided by the used malware, being our objective to have a sample of the dataset that represents all the types of

Table 5: conn.log: history

conn.log: history	
Letter	**Meaning**
S	A SYN without the ACK bit set.
H	A SYN-ACK(“handshake”).
A	A pure ACK.
D	Packet with payload(“data”).
F	Packet with FIN bit set.
R	Packet with RST bit set.
C	Packet with a bad checksum.
I	Inconsistent packet(Both SYN & RST).

network attacks that were used in the lab. To do so, we took several log files to have a complete insight into what has happened in this IoT network. The table 6 gives us the distribution of the number of flows that we used to construct our sample dataset, and which log files were used to extract the flows.

Our sample will have a total of 1,244,220 flows, where was ensured that the number of malicious flows would be equal to the normal flows so that our dataset was balanced. The number of flows removed from each log file is equal to the sum of the number of benign and malicious flows considering the smallest value of the two. For example, considering the first line of table 6, the number of flows in our sample Dataset is equal to 3.665 benign flows added to 3.665 malicious flows. Now that we have our sample, and to simply describe it, the dashboard in figure 12 was made to help us better understand which are the big numbers, which protocols and communication ports are used, and also have an overview of the detailed-label introduced by the experiment, to see the type of attacks that occurred in the network, originated by the malware.

As we can see in figure 12, the "Type of flow" tab inform us that we have a balanced sample regarding the number of benign and malicious flows. At the same figure, in the "Protocols by type of flow" tab we can verify that the malicious traffic is transported mainly by the TCP protocol, furthermore, in the "IP Source and Destination ports flow" tab we can see the occurrence of malicious flows that targets specific destination ports (id.resp_p) and that the biggest amount of malicious flow is related to port scan actions in the "Types of network attacks in the flow" tab.

Table 6: Dataset sample

Log file	Attack	#Zeek	#Bening	#Malicious	#Dataset
CTU-IoT-Malware-Capture-49-1	Mirai	5.410.562	3.665	5.406.897	7.330
CTU-IoT-Malware-Capture-20-1	Torii	3.209	3.193	16	32
CTU-IoT-Malware-Capture-21-1	Torii	3.286	3.272	14	28
CTU-IoT-Malware-Capture-42-1	Trojan	4.427	4.420	6	12
CTU-IoT-Malware-Capture-60-1	Gagfyt	3.581.029	2.476	3.578.553	4.952
CTU-IoT-Malware-Capture-17-1	Kenjiro	54.659.864	31.438	54.628.426	62.876
CTU-IoT-Malware-Capture-36-1	Okiru	13.645.107	2.663	13.642.444	5.326
CTU-IoT-Malware-Capture-8-1	Hakai	10.404	2.181	8.223	4.362
CTU-IoT-Malware-Capture-39-1	IRCBot	73.568.982	7.337	73.561.645	14.674
CTU-IoT-Malware-Capture-7-1	Linux, Mirai	11.454.723	75.955	11.378.768	151.910
CTU-IoT-Malware-Capture-9-1	Linux, Hajime	6.378.294	22.548	6.355.746	45.096
CTU-IoT-Malware-Capture-3-1	Muhstik	156.104	4.536	151.568	9.072
CTU-IoT-Malware-Capture-1-1	Hide and seek	1.008.749	469.275	539.474	938.550
Total	-	169.884.740	632.959	169.251.780	1.244.220

To investigate further the relationships inside this sample, we need to implement mechanisms to help us with this task. Feature selection is one of the core concepts in machine learning which hugely impacts the performance of the ML models. This is a process where automatically or manually features are selected based on their contribution to predicting a variable or output in which we are interested. Having irrelevant features in the data can decrease the ML models accuracy and make them learn based on irrelevant features. Additionally, by reducing the number of data points the algorithm complexity is reduced, resulting in faster training phases of the ML models.

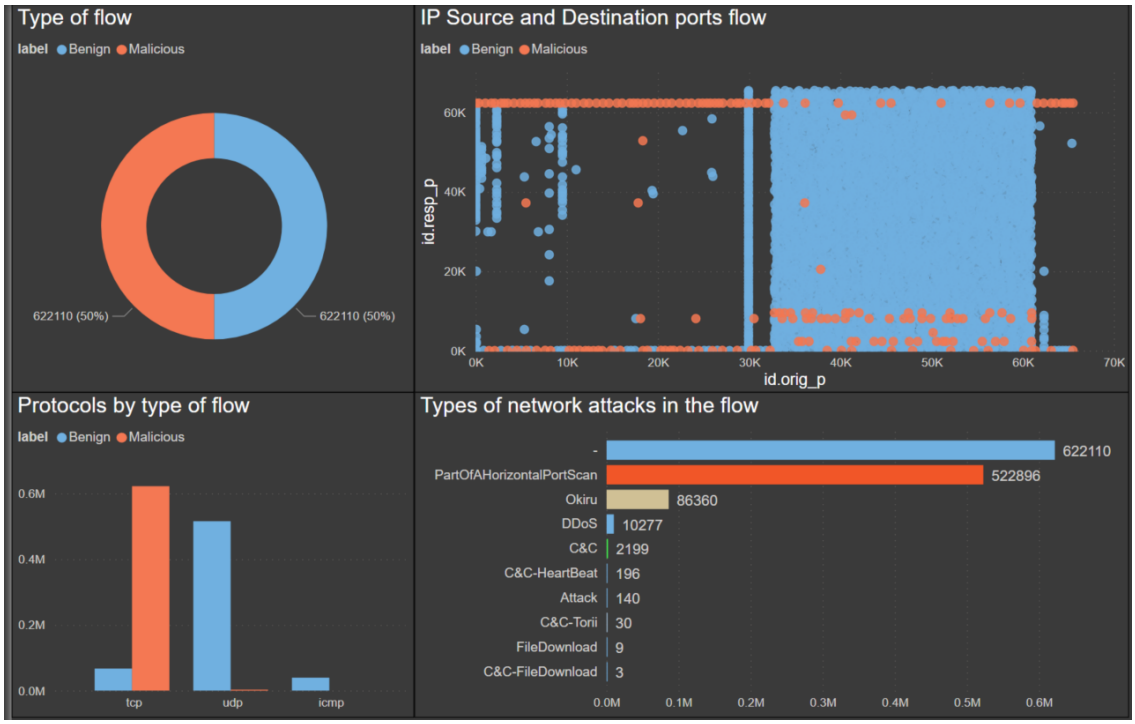


Figure 12: Dataset dashboard

With feature selection in mind, and by looking briefly at the dataset, the columns local_orig and local_resp have always the same value, on the other hand, the columns ts and uid have always different values. In both cases, we assume that these columns have no significance in determining if a flow is malicious or not, therefore the columns will be manually removed. Due to the fact of dismissing the UID column, we will also dismiss the tunnel-parents column, since both are related to one another.

From this point on, we will need automatic mechanisms to ensure a correct feature selection for training our ML models. For this purpose, the analysis was divided into two distinct sections, we will have the feature selection for binary classification of the data, that is, the models, in this case, will only reference whether a given flow is benign or malicious, or we will do the feature selection to a classification by type of attack or by class, also named as multi-class classification.

Binary classification

Considering the binary classification approach, we will drop the column detailed-labeled, since we are only interested in knowing if the flow is Benign or Malicious. At this stage, we have reduced the number of columns to 17 and from this, the

first 16 will be the features that determine if the flow (label column) is benign or malicious (Figure 13).

```
Data columns (total 17 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   id.orig_h      1542506 non-null  object
 1   id.orig_p      1542506 non-null  object
 2   id.resp_h      1542506 non-null  object
 3   id.resp_p      1542506 non-null  float64
 4   proto          1542506 non-null  object
 5   service        1542506 non-null  object
 6   duration       1542506 non-null  object
 7   orig_bytes     1542506 non-null  object
 8   resp_bytes     1542506 non-null  object
 9   conn_state     1542506 non-null  object
 10  missed_bytes   1542506 non-null  float64
 11  history        1542506 non-null  object
 12  orig_pkts      1542506 non-null  float64
 13  orig_ip_bytes  1542506 non-null  float64
 14  resp_pkts      1542506 non-null  float64
 15  resp_ip_bytes  1542506 non-null  float64
 16  label          1542506 non-null  object
dtypes: float64(6), object(11)
memory usage: 211.8+ MB
```

Figure 13: All dataset features

The decrease of the number of columns is a vital task for our ML models, among others benefits lower training time and less noise in our dataset can simply be achieved. Considering the obtained 16 features, we probably have columns that are expendable and that are hard to spot without the help of data science mechanisms and algorithms. So, to obtain the features that are more relevant for our goal, we need some computational aid to perform this task. First a transformation of the features from categorical (strings) to numerical values and its normalization is needed. Using algorithms like the LabelEncoder [24] and the SimpleImputer [24], to transform strings into values, and the StandardScaler [24] algorithm for normalization, we can finally verify which features have bigger contribution for classify the flow inside our sample by using the SelectKBest [24], ExtraTreeClassifier [24] (Figure 14) or even Heatmap [24] algorithms (Figure 29 at appendix B). In our case, the ExtraTreeClassifier algorithm was used due to the clarity of the obtained results [24], but we can see the implemented python code in figure 30 and 31 in appendix B.

The figure 14 was obtained by running the ExtraTreeClassifier[24] algorithm and, as we can see, the 4 most relevant features are the proto (Protocol), history, Id.resp_p (Destination port), and the Id.orig_p (Source port).

As seen above in our dashboard in figure 12 the protocol (proto) and IP source/destination ports (id.orig_p /id.resp_p) used in the communication are one of the most preponderant features that will help us to classify the flows. On

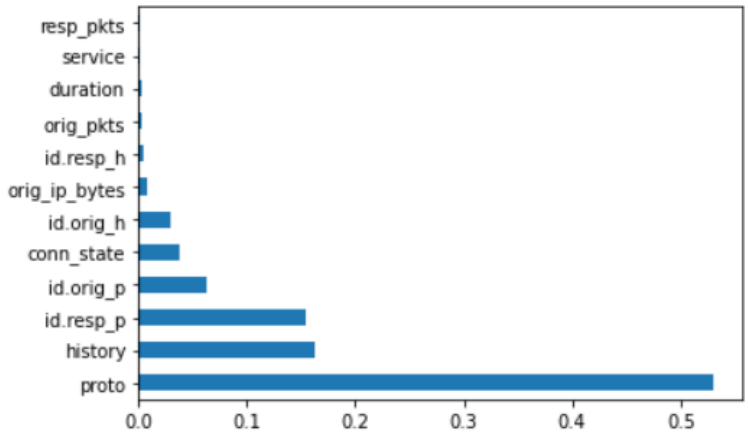


Figure 14: ExtraTree Classifier [24] feature selection

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1244220 entries, 0 to 1244219
Data columns (total 4 columns):
#   Column     Non-Null Count    Dtype
---  ------     --------------    -----
0   proto      1244220 non-null  int32
1   history    1244220 non-null  int32
2   id.resp_p  1244220 non-null  float64
3   id.orig_p  1244220 non-null  float64
dtypes: float64(2), int32(2)
memory usage: 28.5 MB
```

Figure 15: Features after selection

the other hand, also the history feature appears, as a feature that can determine if a flow is malicious or not. This situation is not surprising in the present context since, as we have seen before, the history feature is related to the TCP control bits, and since TCP is one of the drivers for malicious flows in this network, is normal that also history enters here as an important feature. Now that we made all the data preprocessing and feature selection we can move forward and apply our ML models, regarding binary classification, which is demonstrated in section 4.1.

Multiclass classification

Here we go back to our original dataset already with ts, uid, local_orig, local_resp, and tunnel-parents removed. The difference to binary classification will be, instead of removing the detailed-label, we will remove the label feature. Verifying the values count of the detailed-label column, in the "Types of network attacks in the flow" tab (Figure 12), in our python code, we can see that we have 9 detailed attacks and that the string “-” represent the benign flows. If we sum the attacks numbers, we can see that we still have a balanced dataset between malicious and

benign flows, being in line with the dashboard of the figure 12. For simplicity all attacks that begin with C&C were grouped, reducing the number of labels to seven and producing the following distribution shown in figure 16.

```
-                            622110
PartOfAHorizontalPortScan    522896
Okiru                         86360
DDoS                          10277
C&C                            2428
Attack                          140
FileDownload                      9
Name: detailed-label, dtype: int64
```

Figure 16: Detailed-label classes

Like in binary classification we now must change from categorical to numeric features and by using the LabelEnconder algorithm, we will now have our detailed-label distributed as represented in table 7. By analyzing the table, our sample has a reduced number of "File Download" attacks, "Attack" attacks, "C&C" attacks, "DDoS" attacks, or even "Okiru" attacks, compared to the attack "PartOfHorizontalPortScan". This situation can be explained by the fact that most of the time, intrusion attempts on computer systems are initiated precisely by port scans to identify ports of entry into the systems, ports to open Telnet, Remote Desktop Protocol (RDP), or even for SSH applications, that may later be targeted by brute force attacks.

Table 7: Multicalss label encoder

Label code	**Detailed-label**	**#Flows**
0	-	622.110
6	PartOfAHorizontalPortScan	522.896
5	Okiru	86.360
3	DDoS	10.277
2	C&C	2.428
1	Attack	140
4	FileDownload	9

Considering the same feature selection, that was made in the binary classification, we can now proceed with our ML models training, which will be demonstrated in section 4.2.

With the pre-processing performed, both for the binary classification exercise and for the multiclass classification, we can proceed with the evaluation of the ML models for each case.

4

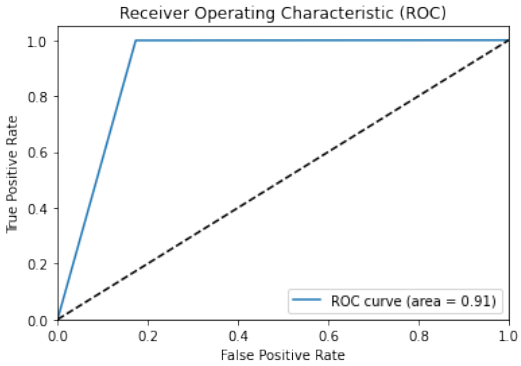
EXPERIMENTAL RESULTS

It is time to apply the ML models, to discover the existing relationships inside the data, and to verify if our feature selection returns to us satisfactory results in terms of identifying malicious flows, in our sample. In this section is made a proof of concept by training and evaluating some ML models, that could be deployed in our smart city architecture. To do the experimental evaluation we have used the programming language Python [31] in a jupyter notebook [19] to handle the dataset.

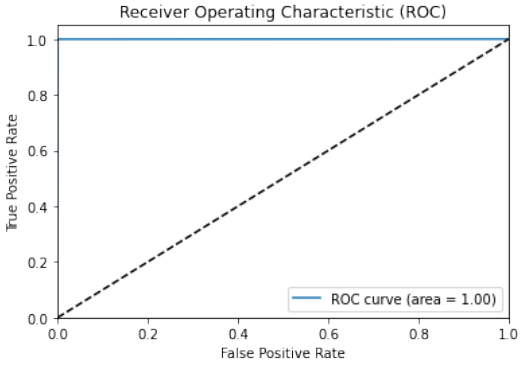
4.1 BINARY CLASSIFICATION RESULTS

Using python library scikit-learn [24] we will proceed with the implementation of the ML algorithms Logistic Regression, Random Forest, and Naïve Bayes. The dataset was loaded into our python environment, where we have done all the data preprocessing and applied all the necessary algorithms. After obtaining the selected features, the dataset was split in a test sample and into a training sample, and as the name suggests, the models are trained with the training sample and then, to verify the success of the training, the test sample is used to generate a prediction made by ML model. Based on the expected outcome, and comparing it with the predictions made, we will take a look at some performance metrics to validate if we have a random classifier or not.

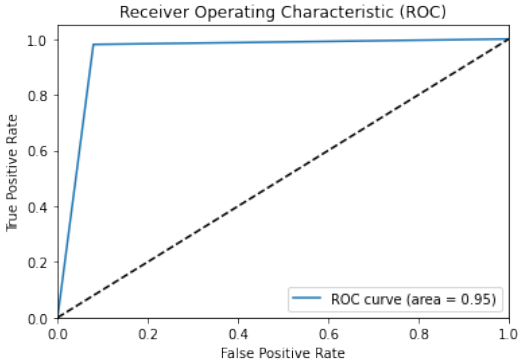
In figure 17 we can see the ROC curve results for each of the ML models used. Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). Here we can state that all three models are not random, and all of them have high percentage of true positives. To see exactly the number of hits and misses, we have the figure 18, from which we can confirm the high percentage of true positives.



(a) ROC chart Logistic Regression



(b) ROC chart Random Forest

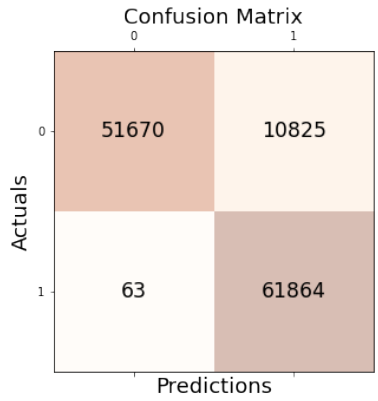


(c) ROC chart Naïve Bayes

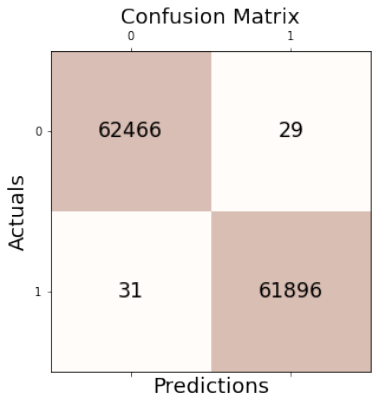
Figure 17: ROC charts ML models

Having the numbers of true positives, false positives, and false negatives to each case, we can simply compare the ML models through Recall and Precision metrics.

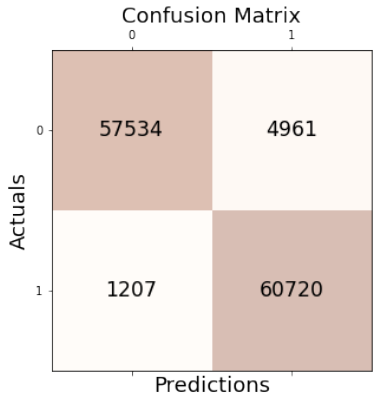
Based on the figures 17 and 18, and in the table 8, Random Forest obtains the higher results in terms of Recall and Precision, which makes him the model with the better performance when compared with the other two.



(a) Confusion Matrix Logistic Regression



(b) Confusion Matrix Random Forest



(c) Confusion Matrix Naïve Bayes

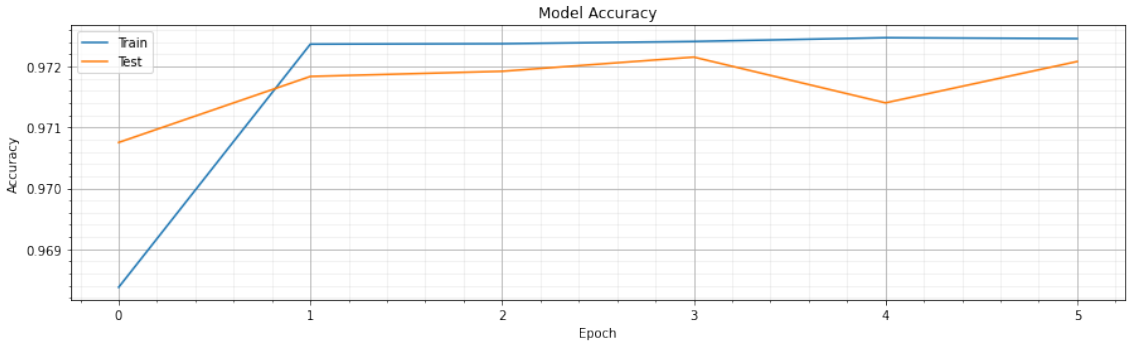
Figure 18: Confusion Matrix ML models

Table 8: Recall and Precision metrics of the ML models

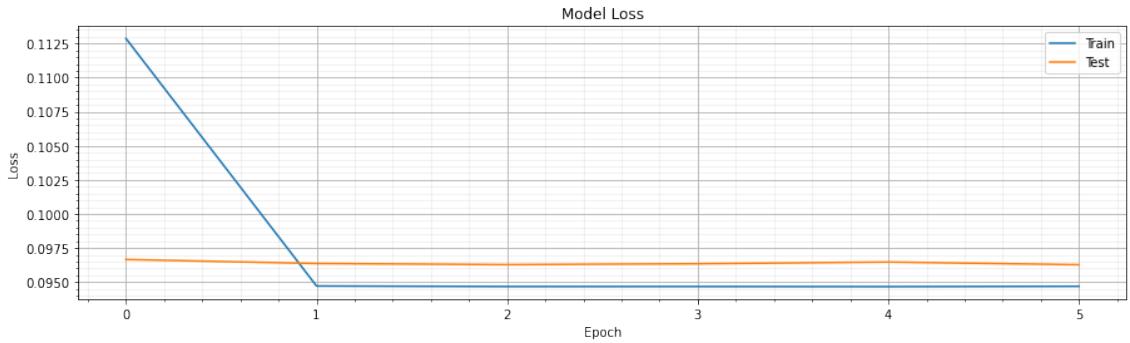
ML Models	**Recall**	**Precision**	**TP**	**FP**	**FN**
Logistic Regression	99,90%	85,11%	61.864	10.825	63
Random Forest	99,95%	99,95%	61.896	29	31
Naïve Bayes	98,05%	92,45%	60.720	4.961	1.207

Nevertheless, in our binary classification exercise, we want to assess if deep learning models have some advantage over the traditional ML models already tested. To start the comparison was implemented a baseline model that will help us to better understand the differences between a simple artificial neural network (ANN), and a more complex one with several layers or so-called neurons. To show the work done by the training phase, of this kind of ML model, the figure 19 shows us the several interactions made by the model algorithm to discover the point (EPOCH) where he cannot enhance further his prediction capability. This is achieved by using the EarlyStopping class of Keras API reference [8], which

stops the training when monitored metrics have stopped improving. In this case, the monitored metric was the loss function, which means the error between the output of our algorithm and a given target value (closer to zero the better).



(a) ANN Baseline Model Accuracy Training



(b) ANN Baseline Model Loss Training

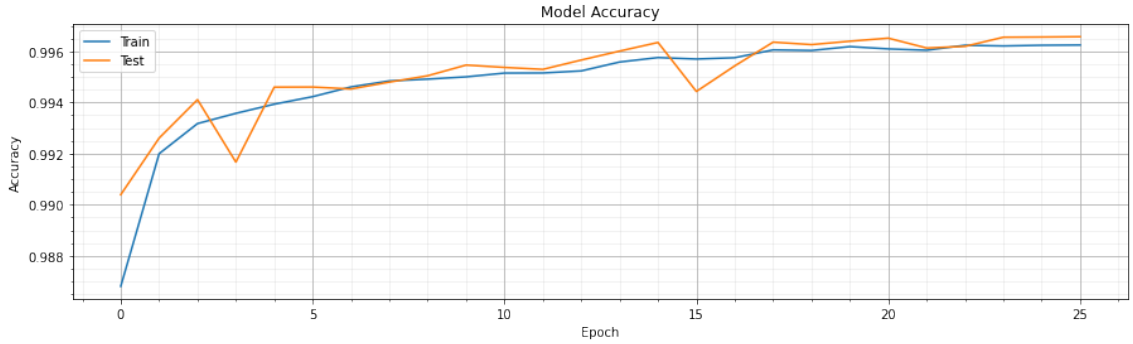
Figure 19: ANN Baseline Model Training

Likewise, a more complex ANN model was trained. With an MLP model, we are waiting for an improvement, of the results, since we are placing more computational power by adding layers of neurons to the model. That improvement can be seen in the accuracy curve and loss curve, of the model, during the training phase and reflected in figure 20.

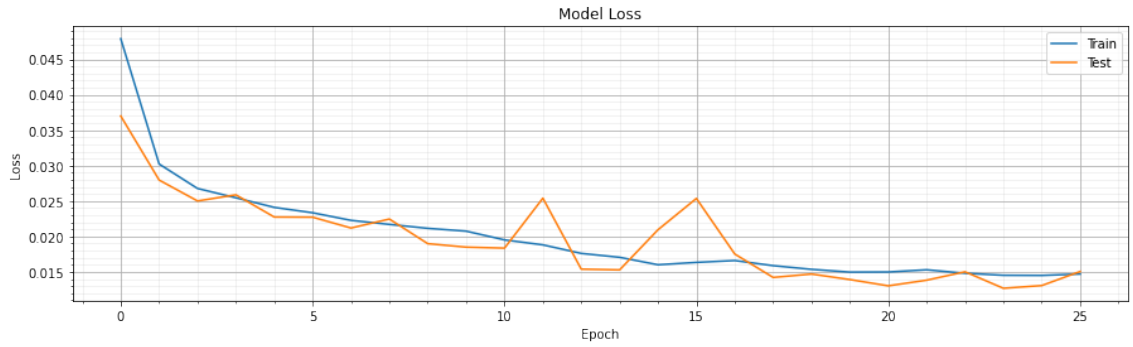
After the training phase, we can compare the ANN models between them. Based on the figures 21 and 22 we see that through deep learning we will achieve valid classifiers. And when calculating the performance metrics (Table 9), the results show us the benefit of having some extra neurons, especially regarding the Precision metric.

Table 9: Recall and Precision metrics of the ANN models

ANN Learning Models	**Recall**	**Precision**	**TP**	**FP**	**FN**
Baseline Model	99,02%	95,29%	61.322	3.029	605
MLP Model	99,70%	99,57%	61.744	265	183

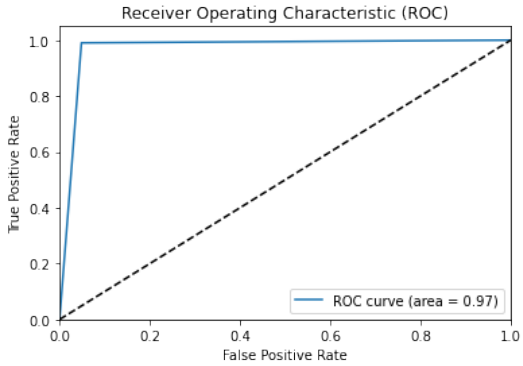


(a) ANN Baseline Model Accuracy Training

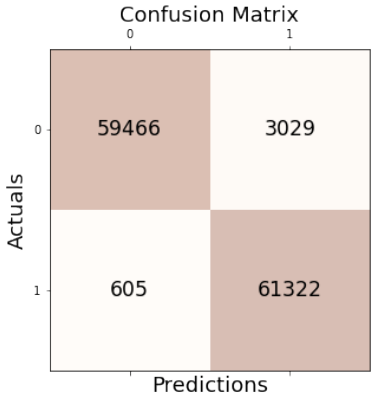


(b) ANN Baseline Model Loss Training

Figure 20: ANN MLP Model Training



(a) ANN Baseline ROC curve

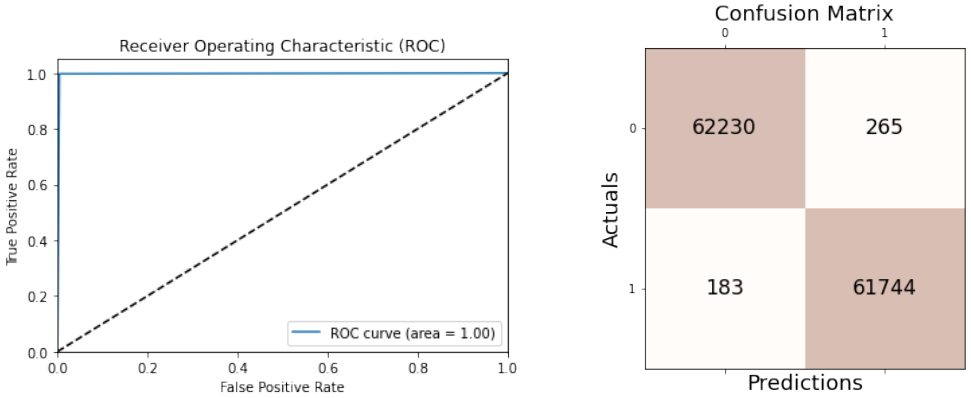


(b) ANN Baseline Confusion Matrix

Figure 21: ANN Baseline Model Training

4.2 MULTICLASS CLASSIFICATION RESULTS

Following the same procedures of the binary classification approach, it was tried to classify this dataset based on the 7 classes of attack (table 7) that were labeled in the IoT23 dataset. The testing with the traditional ML models, as shown in the



(a) ANN MLP ROC curve

(b) ANN MLP Confusion Matrix

Figure 22: ANN MLP Model Training

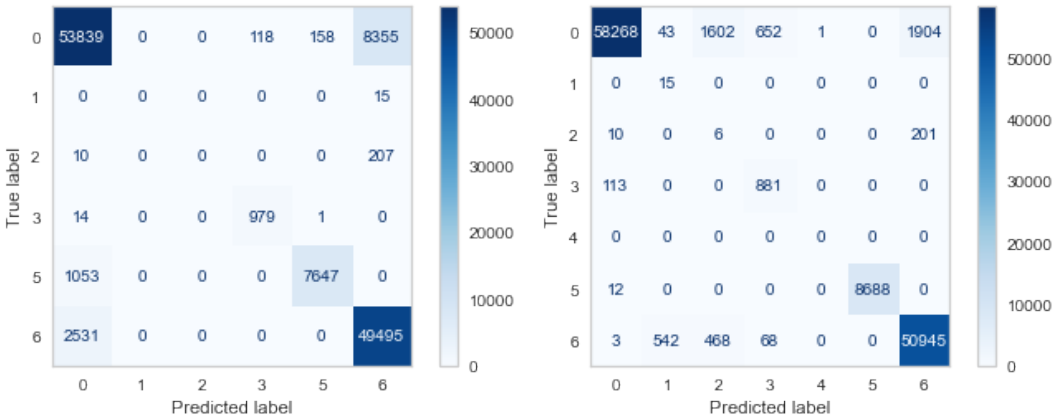
figure 23, reveals the struggle of accurately classifying all the flows of the test sample.

Tables 10 and 11 present the recall and precision metrics of multiclass when using Logistic Regression and Naïve Bayes respectively, while figure 23 presents the confusion matrixes. We can state that the Naïve Bayes classifier is the only model that acknowledges the existence of the attack 4, but even then the classifier returns it as a false positive. Even against the attack types 1, 2, and 3, the precision and recall decrease significantly compared with the attack types that are the majority of the flows represented.

Table 10: Multiclass Recall and Precision metrics of Logistic Regression

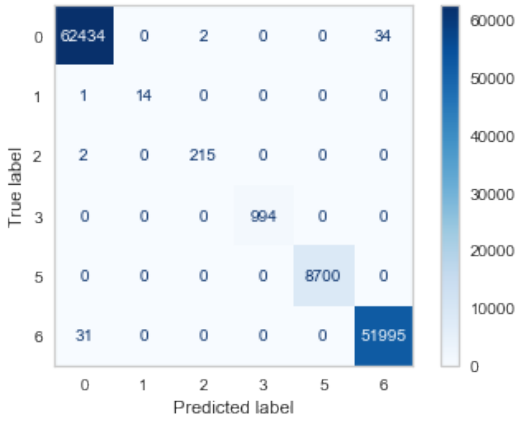
Class	Recall	Precision	TP	FP	FN
0	86,18%	93,72%	53.839	3.608	8.631
1	0,00%	-	0	0	15
2	0,00%	-	0	0	217
3	98,49%	89,24%	979	118	15
5	87,90%	97,96%	7.647	159	1.053
6	95,14%	85,23%	49.495	8.577	2.531

The ANN models will not be an exception when facing the multiclass classification problem (Figure 24 and tables 13 and 14), both baseline and the MLP models, are unable to identify the attack type 4. Nevertheless, we can see the advantage of the processing power extra of MLP, by obtaining better values of Recall and Precision, comparing with the baseline model, for example in the attack type 1.



(a) Multiclass LR Confusion Matrix

(b) Multiclass RF Confusion Matrix



(c) Multiclass NB Confusion Matrix

Figure 23: Multiclass Confusion Matrix ML models

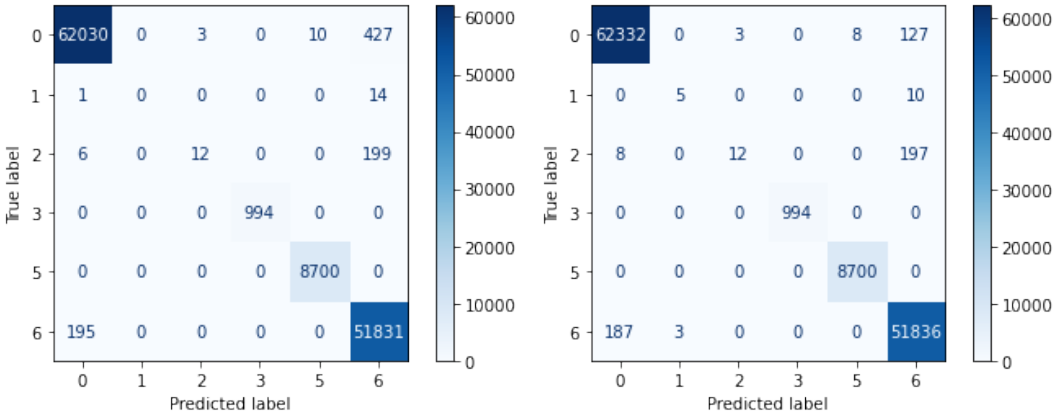
Table 11: Multiclass Recall and Precision metrics of Naïve Bayes

Class	Recall	Precision	TP	FP	FN
0	93,27%	99,76%	58.268	138	4.202
1	100,00%	2,50%	15	585	0
2	2,76%	0,29%	6	2.070	211
3	88,63%	55,03%	881	720	113
4	-	0,00%	0	1	0
5	99,86%	100,00%	8.688	0	12
6	97,92%	96,03%	50.945	2.105	1.081

As a conclusion of the multiclass exercise, it demonstrated that to training properly the ML and ANN models, is needed a significant amount of data from each type of attack, otherwise, the ML models will not learn to identify a given flow of that type of attack.

Table 12: Multiclass Recall and Precision metrics of Random Forest

Class	Recall	Precision	TP	FP	FN
0	99,94%	99,95%	62.434	34	36
1	93,33%	100,00%	14	0	1
2	99,08%	99,08%	215	2	2
3	100,00%	100,00%	994	0	0
5	100,00%	100,00%	8.700	0	0
6	99,94%	99,93%	51.995	34	31



(a) Multiclass ANN baseline model Confusion Ma-(b) Multiclass ANN MLP model Confusion Matrix trix

Figure 24: Multiclass Confusion Matrix ANN models

Table 13: Multiclass Recall and Precision metrics of ANN Baseline model

Class	Recall	Precision	TP	FP	FN
0	99,30%	99,68%	62.030	202	440
1	0,00%	-	0	0	15
2	5,53%	80,00%	12	3	205
3	100,00%	100,00%	994	0	0
5	100,00%	99,89%	8.700	10	0
6	99,63%	98,78%	51.831	640	195

4.3 UNSUPERVISED CLASSIFICATION RESULTS

Also, a simple unsupervised model based on clustering was tested, where to do the binary classification problem was considered 2 clusters. Without any clue, the K-Means algorithm will try to find on its own the relations between the data and group each data point (also called as coordinate) to a cluster. With the help

Table 14: Multiclass Recall and Precision metrics of ANN MLP model

Class	Recall	Precision	TP	FP	FN
0	99,78%	99,69%	62.332	195	138
1	33,33%	62,50%	5	3	10
2	5,53%	80,00%	12	3	205
3	100,00%	100,00%	994	0	0
5	100,00%	99,91%	8.700	8	0
6	99,63%	99,36%	51.836	334	190

of the yellowbrick library [4] , and to verify the formed clusters, and to see if it was a good split, the clustering algorithm has the silhouette plot that is ranked between -1 and +1, it is closer to +1 is because we have a good separation of the data between the cluster, otherwise, the algorithm is struggling to choose which is the best cluster. Since we have forced the number of clusters to 2, the separation in figure 25 is not the best, but, if we consider the "Benign flow" equals to 0 and the "Malicious flow" equal to 1, considering these values as the prediction of the model, and compare it with the label of the test dataset we will have the confusion matrix in the figure 26 and the correspondent performance metrics of the table 15.

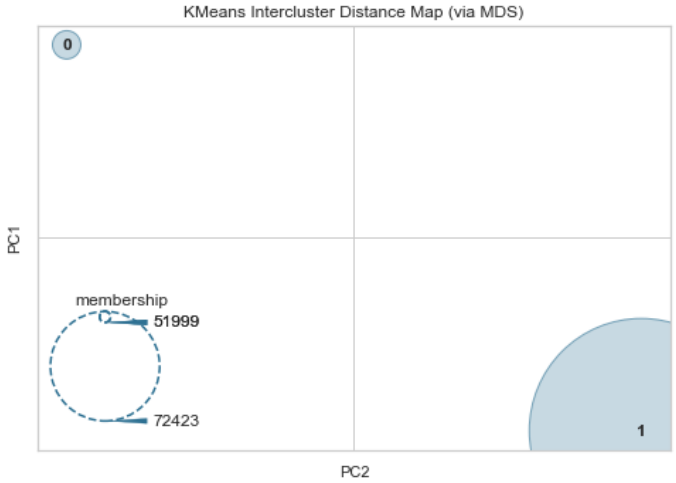
By making the assessment of the results obtained in the table 15, this unsupervised algorithm gets very good results on our binary classification exercise.

Table 15: Binary Classification K-Means Performance Metrics

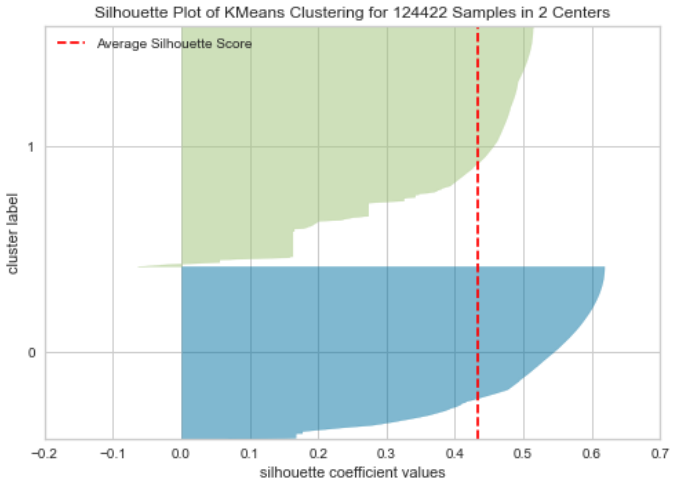
Unsupervised Model	**Recall**	**Precision**	**TP**	**FP**	**FN**
K-Means	99,60%	85,17%	61.681	10.742	246

In unsupervised clustering models the elbow technique is used to find the optimal number of clusters in which the data will be divided. Regarding our test dataset, the obtained k value was 4 (Figure 27).

When verifying the silhouette score, we can see the improvement the score has by approaching the +1 value (Figure 28). In this case, regarding the number of clusters, it's more difficult to establish a direct connection between the clusters labels and the attacks types of the multiclass classification exercise, being necessary additional analysis to determine which cluster belongs to a given label, and in this dissertation, no further investigation was made at this point.



(a) Binary Classification K-Means Clusters



(b) Binary Classification K-Means Silhouette

Figure 25: Binary Classification K-Means (k=2)

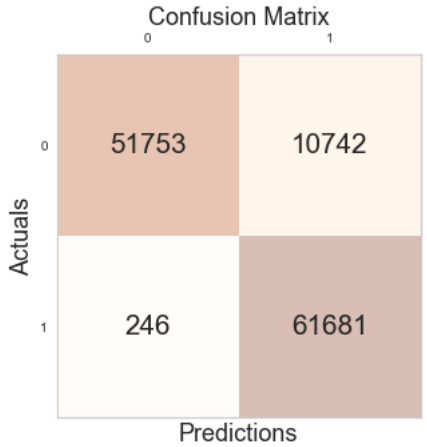


Figure 26: Binary Classification K-Means Confusion Matrix

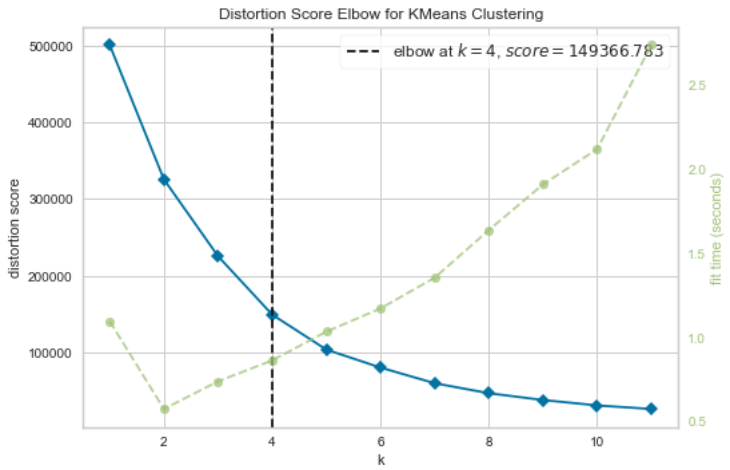
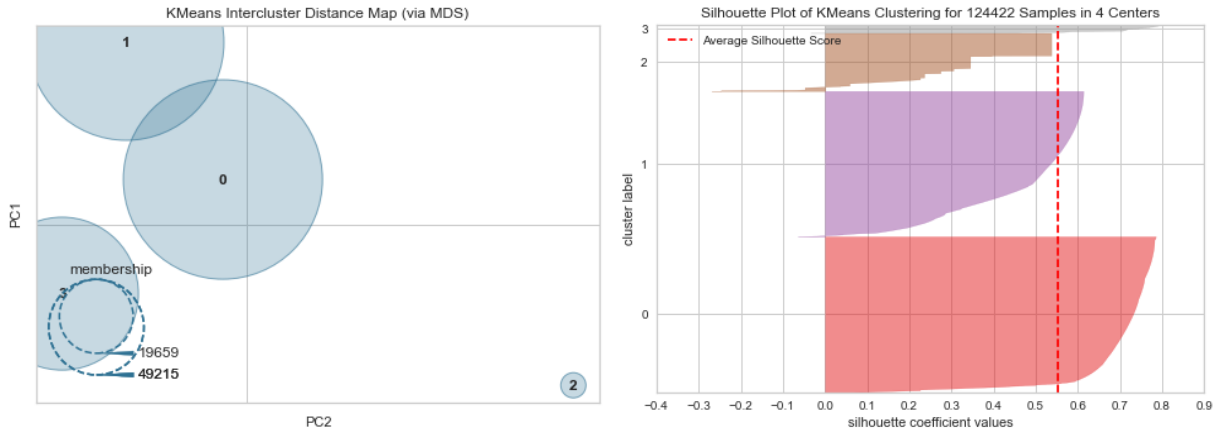


Figure 27: K-Means elbow curve



(a) Multiclass Classification K-Means Clusters

(b) Multiclass Classification K-Means Silhouette

Figure 28: Multiclass Classification K-Means (k=4)

4.4 DISCUSSION

To compare our results with a similar academic paper we have looked to the master thesis called IoT Malicious Traffic Classification Using Machine Learning [3]. In Austin's [3] dissertation it was used the same dataset and also ML models like the Random Forest or the Naïve Bayes. Nevertheless, some aspects of the data preprocessing are different, for instance, our dissertation discards the timestamp and in Austin's work [3] is the top feature. Here we consider that the time associated with flows should be removed since we always have different values throughout the analyzed dataset.

This premise allowed us to look only and only at the characteristics of the flows from a more technical point of view when analyzing the ports and protocols used.

Anyway, Austin's approach is not wrong, far from it, because many of the attacks that occur in communication networks are repetitive and if it is possible to foresee that a given attack will occur in a given time window, it is an asset.

Regarding the models and analysis carried out, our dissertation also presents deep learning solutions and unsupervised models that allowed us to have a more comprehensive view of the machine learning universe. Additionally, taking advantage of the data set already tagged, the analysis was carried out from the point of view of the multiclass classification to observe the behavior of the models in this type of scenario as well.

The comparison regarding the binary classification of the dataset between Austin's [3] and our results are shown in table 16. In both cases, we can verify that the Random Forest model has the best performance, in what we consider to be the most relevant metrics for this type of exercise. However, when comparing with the MLP deep learning model, which obtained results very similar to those of Random Forest, we consider that this can be a more scalable and flexible solution if there is any change during the network operation.

Table 16: Binary classification comparison

	Our results		**Austin [3] results**	
Models	**Recall**	**Precision**	**Recall**	**Precision**
Logistic Regression	99.90%	85.11%	-	-
Random Forest	99.95%	99.95%	94.98%	99.92%
Naïve Bayes	98.05%	92.45%	99.75%	89.76%
Linear SVM	-	-	99.90%	85.85%
ANN - Single layer	99.02%	95.29%	-	-
ANN - Multilayer Perception	99.70%	99.57%	-	-
K-Means	99,60%	85,17%	-	-

At the multiclass experiment we can see that in some cases, due to the low number of samples of a given attack type, the models cannot always detect accurately that kind of flow. Probably with more samples of each type of attack, the model could adapt the classification to better values but what we want to expose is that in this kind of supervised learning, if the ML model is not trained properly and with sufficient samples it can also overlook attacks that may exist in the flow. Also, as referred before in the unsupervised exercise, additional analysis must be made to make "reference data" between the detail labels of the dataset and the numbers of clusters obtained.

Facing these results, the best course of action should be to continue adopting the supervised binary classification approach to be implemented in our scenario.

5

CONCLUSION AND NEXT STEPS

The positioning of the machine learning-based IDS for IoT in a practical case provided a more comprehensive view of the needs, not only in terms of detection but also in relation to the network in which it was inserted. The ML models should be trained and tested inside the network context to realize what the normal behaviors are, and what can be an anomaly produced by an attack. One of the biggest initial constraints was not having a dataset referring to a properly categorized IoT environment to train our supervised ML models. Although there are some alternatives to the IoT23 dataset, none had the necessary size or exemption in the data to have a free biasing workout.

Based on Recall and Precision performance metrics, we were able to determine among all the models tested that the Random Forest algorithm has the best pair of metrics, being chosen the most reliable model to apply in our scenario. However, supervised DL models like MLP manage to have superior adaptability and scalability. Here we could have taken an extra step in the investigation by trying to determine which number of neurons is most suitable for this solution, in order to try to achieve more interesting results similar to those obtained by Random Forest. The reason that this did not happen was for the simple fact that, with our implementation, we have obtained very interesting results, in the binary classification, without having required great effort. This is the reason that leads us to believe that DL can in fact be very useful in more complex environments or contexts even if it is necessary to apply more processing power on the machines where the algorithm is running. When looking at the support layer of the IoT enviroment, using fog computing, the need for processing, which often haunts ML solutions associated with IoT, will fade away as this type of technology is increasingly disseminated. At this point, we can state that the use of data science can be a valid and powerful layer in a computer security system.

Additionally, by representing the data flow through the IPFIX protocol, we were able to normalize the structure of the data received and implement ways to extract information about the traffic that circulates in the network. As an open-source

protocol, it has great flexibility and scope, which allows its use in a simple way, not only in an IoT environment but also in "traditional" networks.

Based on our work, and as a continuous improvement perspective, the next two sections propose paths that can be taken into consideration as guiding lines of future work.

5.1 DEEPER LEARNING AND UNSUPERVISED MODELS

Deep Learning uses more advanced techniques and has more flexibility when implemented than “traditional” ML models. Being based on neural networks, deep learning has the ability of learning based on experience, without human intervention, and since computer processing is increasingly a common resource it has all the conditions to prosper. Image and language recognition, language translation, timed based events are some of the features that can be achieved with this kind of learning. As the next step of this dissertation, we believe that the use and improvement of this kind of learning could help us to go deeper and find more robust anomaly detection systems.

5.2 MODELS ATTACK RESILIENCE

Another path that can be taken, is to test the ML models’ resilience when they are under attack. The models can suffer from evasion, poisoning, inference, trojaning, and backdooring and all these attacks will try to interfere with the ML model behavior either in training or production phases, which poses a real threat in applications related to cybersecurity.

BIBLIOGRAPHY

[1] Sanaz Amnalou and Khairul Azmi Abu Bakar. "Lightweight security mechanism over MQTT protocol for IoT devices". In: *International Journal of Advanced Computer Science and Applications* 11.7 (2020), pp. 202–207. ISSN: 21565570. DOI: 10.14569/IJACSA.2020.0110726.

[2] Riyaz Ahamed Ariyaluran Habeeb et al. "Real-time big data processing for anomaly detection: A Survey". In: *International Journal of Information Management* 45.August 2018 (2019), pp. 289–307. ISSN: 02684012. DOI: 10.1016/j.ijinfomgt.2018.08.006.

[3] Michael Austin. "IoT Malicious Traffic Classification Using Machine Learning IoT Malicious Traffic Classification Using Machine Learning". In: (2021).

[4] Benjamin Bengfort and Rebecca Bilbro. "Yellowbrick: Visualizing the Scikit-Learn Model Selection Process". In: *The Journal of Open Source Software*. 1075th ser. 4.35 (Mar. 24, 2019). DOI: 10.21105/joss.01075. URL: http://joss.theoj.org/papers/10.21105/joss.01075.

[5] Daniel S. Berman et al. "A survey of deep learning methods for cyber security". In: *Information (Switzerland)* 10.4 (2019). ISSN: 20782489. DOI: 10.3390/info10040122.

[6] Nadia Chaabouni et al. "Network Intrusion Detection for IoT Security Based on Learning Techniques". In: *IEEE Communications Surveys and Tutorials* 21.3 (2019), pp. 2671–2701. ISSN: 1553877X. DOI: 10.1109/COMST.2019.2896380.

[7] Raghavendra Chalapathy and Sanjay Chawla. "Deep Learning for Anomaly Detection: A Survey". In: (2019), pp. 1–50. arXiv: 1901.03407. URL: http://arxiv.org/abs/1901.03407.

[8] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[9] Benoît Claise. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information*. RFC 5101. Jan. 2008. DOI: 10.17487/RFC5101. URL: https://rfc-editor.org/rfc/rfc5101.txt.

[10] Benoît Claise and Brian Trammell. *Information Model for IP Flow Information Export (IPFIX)*. RFC 7012. Sept. 2013. DOI: 10.17487/RFC7012. URL: https://rfc-editor.org/rfc/rfc7012.txt.

[11] Benoît Claise et al. *Information Model for IP Flow Information Export*. RFC 5102. Jan. 2008. DOI: 10.17487/RFC5102. URL: https://rfc-editor.org/rfc/rfc5102.txt.

[12] Lei Cui et al. "Security and privacy in smart cities: Challenges and opportunities". In: *IEEE Access* 6 (2018), pp. 46134–46145. ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2853985.

[13] Abebe Abeshu Diro and Naveen Chilamkurti. "Distributed attack detection scheme using deep learning approach for Internet of Things". In: *Future Generation Computer Systems* 82 (2018), pp. 761–768. ISSN: 0167739X. DOI: 10.1016/j.future.2017.08.043.

[14] Mohamed Faisal Elrawy, Ali Ismail Awad, and Hesham F.A. Hamed. "Intrusion detection systems for IoT-based smart environments: a survey". In: *Journal of Cloud Computing* 7.1 (2018), pp. 1–20. ISSN: 2192113X. DOI: 10.1186/s13677-018-0123-6.

[15] Mohammed Ali Al-Garadi et al. "A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security". In: *IEEE Communications Surveys and Tutorials* 22.3 (2020), pp. 1646–1685. ISSN: 1553877X. DOI: 10.1109/COMST.2020.2988293. arXiv: 1807.11023.

[16] Mahmudul Hasan et al. "Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches". In: *Internet of Things (Netherlands)* 7 (2019), p. 100059. ISSN: 25426605. DOI: 10.1016/j.iot.2019.100059. URL: https://doi.org/10.1016/j.iot.2019.100059.

[17] Fatima Hussain et al. "Machine Learning in IoT Security: Current Solutions and Future Challenges". In: *IEEE Communications Surveys and Tutorials* 22.3 (2020), pp. 1686–1721. ISSN: 1553877X. DOI: 10.1109/COMST.2020.2986444. arXiv: 1904.05735.

[18] Sidra Ijaz et al. "Smart Cities: A Survey on Security Concerns". In: *International Journal of Advanced Computer Science and Applications* 7.2 (2016). DOI: 10.14569/ijacsa.2016.070277.

[19] Thomas Kluyver et al. "Jupyter Notebooks – a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87–90.

[20] Arun Rana Kumar et al. "A Survey of Machine Learning Methods for IoT and their Future Applications". In: *Amity Journal of Computational Sciences* 2.2 (2018), pp. 1–5. URL: www.amity.edu.in/ajcs.

[21] Marvin L. Minsky and Seymour Papert. *Perceptrons, Expanded Edition An Introduction to Computational Geometry*. 1969, p. 292. ISBN: 9780262534772.

[22] Nour Moustafa, Jiankun Hu, and Jill Slay. "A holistic review of Network Anomaly Detection Systems: A comprehensive survey". In: *Journal of Network and Computer Applications* 128.December 2018 (2019), pp. 33–55. ISSN: 10958592. DOI: 10.1016/j.jnca.2018.12.006.

[23] Nitin Naik. "Choice of effective messaging protocols for IoT systems MQTT, CoAP, AMQP and HTTP". In: *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings* (2017). DOI: 10.1109/SysEng.2017.8088251.

[24] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[25] FERNANDO A. SANTOS. "UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO DEPARTAMENTO DE AGRONOMIA A Internet das Coisas aplicada à agricultura". In: (2020).

[26] & Maria Jose Erquiaga Sebastian Garcia Agustin Parmisano. "IoT-23: A labeled dataset with malicious and benign IoT network traffic". In: (2020). URL: http://doi.org/10.5281/zenodo.4743746.

[27] Parminder Singh et al. "Blockchain and fog based architecture for internet of everything in smart cities". In: *Future Internet* 12.4 (2020), pp. 1–12. ISSN: 19995903. DOI: 10.3390/FI12040061.

[28] V. Sucharitha, P. Prakash, and Ganesh Neelakanta Iyer. "Agrifog-a fog computing based IoT for smart agriculture". In: *International Journal of Recent Technology and Engineering* 7.6 (2019), pp. 210–217. ISSN: 22773878.

[29] Akbar Telikani et al. "Evolutionary Machine Learning: A Survey". In: *ACM Computing Surveys* 54.8 (2022). ISSN: 15577341. DOI: 10.1145/3467477.

[30] Stefanos Tsimenidis, Thomas Lagkas, and Konstantinos Rantos. *Deep Learning in IoT Intrusion Detection*. Vol. 30. 1. Springer US, 2022. ISBN: 0123456789. DOI: 10.1007/s10922-021-09621-9. URL: https://doi.org/10.1007/s10922-021-09621-9.

[31] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[32] Sherali Zeadally and Michail Tsikerdekis. “Securing Internet of Things (IoT) with machine learning”. In: *International Journal of Communication Systems* 33.1 (Jan. 2020), e4169. ISSN: 10745351. DOI: 10.1002/dac.4169. URL: https://doi.org/10.1002/dac.4169%20http://doi.wiley.com/10.1002/dac.4169.

[33] Tanja Zseby et al. “IPFIX / PSAMP : What Future Standards can Offer to Network Security”. In: ().

APPENDICES

APPENDIX A

The appendix A present all the current IEs of the IPFIX protocol.

Table 17: IPFIX current IEs

ElementID	Name	Abstract Data Type
1	octetDeltaCount	unsigned64
2	packetDeltaCount	unsigned64
3	deltaFlowCount	unsigned64
4	protocolIdentifier	unsigned8
5	ipClassOfService	unsigned8
6	tcpControlBits	unsigned16
7	sourceTransportPort	unsigned16
8	sourceIPv4Address	ipv4Address
9	sourceIPv4PrefixLength	unsigned8
10	ingressInterface	unsigned32
11	destinationTransportPort	unsigned16
12	destinationIPv4Address	ipv4Address
13	destinationIPv4PrefixLength	unsigned8
14	egressInterface	unsigned32
15	ipNextHopIPv4Address	ipv4Address
16	bgpSourceAsNumber	unsigned32
17	bgpDestinationAsNumber	unsigned32
18	bgpNextHopIPv4Address	ipv4Address
19	postMCastPacketDeltaCount	unsigned64
20	postMCastOctetDeltaCount	unsigned64
21	flowEndSysUpTime	unsigned32
22	flowStartSysUpTime	unsigned32

23	postOctetDeltaCount	unsigned64
24	postPacketDeltaCount	unsigned64
25	minimumIpTotalLength	unsigned64
26	maximumIpTotalLength	unsigned64
27	sourceIPv6Address	ipv6Address
28	destinationIPv6Address	ipv6Address
29	sourceIPv6PrefixLength	unsigned8
30	destinationIPv6PrefixLength	unsigned8
31	flowLabelIPv6	unsigned32
32	icmpTypeCodeIPv4	unsigned16
33	igmpType	unsigned8
37	flowIdleTimeout	unsigned16
40	exportedOctetTotalCount	unsigned64
41	exportedMessageTotalCount	unsigned64
42	exportedFlowRecordTotalCount	unsigned64
44	sourceIPv4Prefix	ipv4Address
45	destinationIPv4Prefix	ipv4Address
46	mplsTopLabelType	unsigned8
47	mplsTopLabelIPv4Address	ipv4Address
52	minimumTTL	unsigned8
53	maximumTTL	unsigned8
54	fragmentIdentification	unsigned32
55	postIpClassOfService	unsigned8
56	sourceMacAddress	macAddress
57	postDestinationMacAddress	macAddress
58	vlanId	unsigned16
59	postVlanId	unsigned16
60	ipVersion	unsigned8
61	flowDirection	unsigned8
62	ipNextHopIPv6Address	ipv6Address
63	bgpNextHopIPv6Address	ipv6Address

64	ipv6ExtensionHeaders	unsigned32
70	mplsTopLabelStackSection	octetArray
71	mplsLabelStackSection2	octetArray
72	mplsLabelStackSection3	octetArray
73	mplsLabelStackSection4	octetArray
74	mplsLabelStackSection5	octetArray
75	mplsLabelStackSection6	octetArray
76	mplsLabelStackSection7	octetArray
77	mplsLabelStackSection8	octetArray
78	mplsLabelStackSection9	octetArray
79	mplsLabelStackSection10	octetArray
80	destinationMacAddress	macAddress
81	postSourceMacAddress	macAddress
82	interfaceName	string
83	interfaceDescription	string
85	octetTotalCount	unsigned64
86	packetTotalCount	unsigned64
88	fragmentOffset	unsigned16
89	forwardingStatus	unsigned8
90	mplsVpnRouteDistinguisher	octetArray
91	mplsTopLabelPrefixLength	unsigned8
92	srcTrafficIndex	unsigned32
93	dstTrafficIndex	unsigned32
94	applicationDescription	string
95	applicationId	octetArray
96	applicationName	string
98	postIpDiffServCodePoint	unsigned8
99	multicastReplicationFactor	unsigned32
101	classificationEngineId	unsigned8
128	bgpNextAdjacentAsNumber	unsigned32
129	bgpPrevAdjacentAsNumber	unsigned32

130	exporterIPv4Address	ipv4Address
131	exporterIPv6Address	ipv6Address
132	droppedOctetDeltaCount	unsigned64
133	droppedPacketDeltaCount	unsigned64
134	droppedOctetTotalCount	unsigned64
135	droppedPacketTotalCount	unsigned64
136	flowEndReason	unsigned8
137	commonPropertiesId	unsigned64
138	observationPointId	unsigned64
139	icmpTypeCodeIPv6	unsigned16
140	mplsTopLabelIPv6Address	ipv6Address
141	lineCardId	unsigned32
142	portId	unsigned32
143	meteringProcessId	unsigned32
144	exportingProcessId	unsigned32
145	templateId	unsigned16
146	wlanChannelId	unsigned8
147	wlanSSID	string
148	flowId	unsigned64
149	observationDomainId	unsigned32
150	flowStartSeconds	dateTimeSeconds
151	flowEndSeconds	dateTimeSeconds
152	flowStartMilliseconds	dateTimeMilliseconds
153	flowEndMilliseconds	dateTimeMilliseconds
154	flowStartMicroseconds	dateTimeMicroseconds
155	flowEndMicroseconds	dateTimeMicroseconds
156	flowStartNanoseconds	dateTimeNanoseconds
157	flowEndNanoseconds	dateTimeNanoseconds
158	flowStartDeltaMicroseconds	unsigned32
159	flowEndDeltaMicroseconds	unsigned32
160	systemInitTimeMilliseconds	dateTimeMilliseconds

161	flowDurationMilliseconds	unsigned32
162	flowDurationMicroseconds	unsigned32
163	observedFlowTotalCount	unsigned64
164	ignoredPacketTotalCount	unsigned64
165	ignoredOctetTotalCount	unsigned64
166	notSentFlowTotalCount	unsigned64
167	notSentPacketTotalCount	unsigned64
168	notSentOctetTotalCount	unsigned64
169	destinationIPv6Prefix	ipv6Address
170	sourceIPv6Prefix	ipv6Address
171	postOctetTotalCount	unsigned64
172	postPacketTotalCount	unsigned64
173	flowKeyIndicator	unsigned64
174	postMCastPacketTotalCount	unsigned64
175	postMCastOctetTotalCount	unsigned64
176	icmpTypeIPv4	unsigned8
177	icmpCodeIPv4	unsigned8
178	icmpTypeIPv6	unsigned8
179	icmpCodeIPv6	unsigned8
180	udpSourcePort	unsigned16
181	udpDestinationPort	unsigned16
182	tcpSourcePort	unsigned16
183	tcpDestinationPort	unsigned16
184	tcpSequenceNumber	unsigned32
185	tcpAcknowledgementNumber	unsigned32
186	tcpWindowSize	unsigned16
187	tcpUrgentPointer	unsigned16
188	tcpHeaderLength	unsigned8
189	ipHeaderLength	unsigned8
190	totalLengthIPv4	unsigned16
191	payloadLengthIPv6	unsigned16

192	ipTTL	unsigned8
193	nextHeaderIPv6	unsigned8
194	mplsPayloadLength	unsigned32
195	ipDiffServCodePoint	unsigned8
196	ipPrecedence	unsigned8
197	fragmentFlags	unsigned8
198	octetDeltaSumOfSquares	unsigned64
199	octetTotalSumOfSquares	unsigned64
200	mplsTopLabelTTL	unsigned8
201	mplsLabelStackLength	unsigned32
202	mplsLabelStackDepth	unsigned32
203	mplsTopLabelExp	unsigned8
204	ipPayloadLength	unsigned32
205	udpMessageLength	unsigned16
206	isMulticast	unsigned8
207	ipv4IHL	unsigned8
208	ipv4Options	unsigned32
209	tcpOptions	unsigned64
210	paddingOctets	octetArray
211	collectorIPv4Address	ipv4Address
212	collectorIPv6Address	ipv6Address
213	exportInterface	unsigned32
214	exportProtocolVersion	unsigned8
215	exportTransportProtocol	unsigned8
216	collectorTransportPort	unsigned16
217	exporterTransportPort	unsigned16
218	tcpSynTotalCount	unsigned64
219	tcpFinTotalCount	unsigned64
220	tcpRstTotalCount	unsigned64
221	tcpPshTotalCount	unsigned64
222	tcpAckTotalCount	unsigned64

223	tcpUrgTotalCount	unsigned64
224	ipTotalLength	unsigned64
225	postNATSourceIPv4Address	ipv4Address
226	postNATDestinationIPv4Address	ipv4Address
227	postNAPTSourceTransportPort	unsigned16
228	postNAPTDestinationTransportPort	unsigned16
229	natOriginatingAddressRealm	unsigned8
230	natEvent	unsigned8
231	initiatorOctets	unsigned64
232	responderOctets	unsigned64
233	firewallEvent	unsigned8
234	ingressVRFID	unsigned32
235	egressVRFID	unsigned32
236	VRFname	string
237	postMplsTopLabelExp	unsigned8
238	tcpWindowScale	unsigned16
239	biflowDirection	unsigned8
240	ethernetHeaderLength	unsigned8
241	ethernetPayloadLength	unsigned16
242	ethernetTotalLength	unsigned16
243	dot1qVlanId	unsigned16
244	dot1qPriority	unsigned8
245	dot1qCustomerVlanId	unsigned16
246	dot1qCustomerPriority	unsigned8
247	metroEvcId	string
248	metroEvcType	unsigned8
249	pseudoWireId	unsigned32
250	pseudoWireType	unsigned16
251	pseudoWireControlWord	unsigned32
252	ingressPhysicalInterface	unsigned32
253	egressPhysicalInterface	unsigned32

254	postDot1qVlanId	unsigned16
255	postDot1qCustomerVlanId	unsigned16
256	ethernetType	unsigned16
257	postIpPrecedence	unsigned8
258	collectionTimeMilliseconds	dateTimeMilliseconds
259	exportSctpStreamId	unsigned16
260	maxExportSeconds	dateTimeSeconds
261	maxFlowEndSeconds	dateTimeSeconds
262	messageMD5Checksum	octetArray
263	messageScope	unsigned8
264	minExportSeconds	dateTimeSeconds
265	minFlowStartSeconds	dateTimeSeconds
266	opaqueOctets	octetArray
267	sessionScope	unsigned8
268	maxFlowEndMicroseconds	dateTimeMicroseconds
269	maxFlowEndMilliseconds	dateTimeMilliseconds
270	maxFlowEndNanoseconds	dateTimeNanoseconds
271	minFlowStartMicroseconds	dateTimeMicroseconds
272	minFlowStartMilliseconds	dateTimeMilliseconds
273	minFlowStartNanoseconds	dateTimeNanoseconds
274	collectorCertificate	octetArray
275	exporterCertificate	octetArray
276	dataRecordsReliability	boolean
277	observationPointType	unsigned8
278	newConnectionDeltaCount	unsigned32
279	connectionSumDurationSeconds	unsigned64
280	connectionTransactionId	unsigned64
281	postNATSourceIPv6Address	ipv6Address
282	postNATDestinationIPv6Address	ipv6Address
283	natPoolId	unsigned32
284	natPoolName	string

285	anonymizationFlags	unsigned16
286	anonymizationTechnique	unsigned16
287	informationElementIndex	unsigned16
288	p2pTechnology	string
289	tunnelTechnology	string
290	encryptedTechnology	string
291	basicList	basicList
292	subTemplateList	subTemplateList
293	subTemplateMultiList	subTemplateMultiList
294	bgpValidityState	unsigned8
295	IPSecSPI	unsigned32
296	greKey	unsigned32
297	natType	unsigned8
298	initiatorPackets	unsigned64
299	responderPackets	unsigned64
300	observationDomainName	string
301	selectionSequenceId	unsigned64
302	selectorId	unsigned64
303	informationElementId	unsigned16
304	selectorAlgorithm	unsigned16
305	samplingPacketInterval	unsigned32
306	samplingPacketSpace	unsigned32
307	samplingTimeInterval	unsigned32
308	samplingTimeSpace	unsigned32
309	samplingSize	unsigned32
310	samplingPopulation	unsigned32
311	samplingProbability	float64
312	dataLinkFrameSize	unsigned16
313	ipHeaderPacketSection	octetArray
314	ipPayloadPacketSection	octetArray
315	dataLinkFrameSection	octetArray

316	mplsLabelStackSection	octetArray
317	mplsPayloadPacketSection	octetArray
318	selectorIdTotalPktsObserved	unsigned64
319	selectorIdTotalPktsSelected	unsigned64
320	absoluteError	float64
321	relativeError	float64
322	observationTimeSeconds	dateTimeSeconds
323	observationTimeMilliseconds	dateTimeMilliseconds
324	observationTimeMicroseconds	dateTimeMicroseconds
325	observationTimeNanoseconds	dateTimeNanoseconds
326	digestHashValue	unsigned64
327	hashIPPayloadOffset	unsigned64
328	hashIPPayloadSize	unsigned64
329	hashOutputRangeMin	unsigned64
330	hashOutputRangeMax	unsigned64
331	hashSelectedRangeMin	unsigned64
332	hashSelectedRangeMax	unsigned64
333	hashDigestOutput	boolean
334	hashInitialiserValue	unsigned64
335	selectorName	string
336	upperCILimit	float64
337	lowerCILimit	float64
338	confidenceLevel	float64
339	informationElementDataType	unsigned8
340	informationElementDescription	string
341	informationElementName	string
342	informationElementRangeBegin	unsigned64
343	informationElementRangeEnd	unsigned64
344	informationElementSemantics	unsigned8
345	informationElementUnits	unsigned16
346	privateEnterpriseNumber	unsigned32

347	virtualStationInterfaceId	octetArray
348	virtualStationInterfaceName	string
349	virtualStationUUID	octetArray
350	virtualStationName	string
351	layer2SegmentId	unsigned64
352	layer2OctetDeltaCount	unsigned64
353	layer2OctetTotalCount	unsigned64
354	ingressUnicastPacketTotalCount	unsigned64
355	ingressMulticastPacketTotalCount	unsigned64
356	ingressBroadcastPacketTotalCount	unsigned64
357	egressUnicastPacketTotalCount	unsigned64
358	egressBroadcastPacketTotalCount	unsigned64
359	monitoringIntervalStartMilliSeconds	dateTimeMilliseconds
360	monitoringIntervalEndMilliSeconds	dateTimeMilliseconds
361	portRangeStart	unsigned16
362	portRangeEnd	unsigned16
363	portRangeStepSize	unsigned16
364	portRangeNumPorts	unsigned16
365	staMacAddress	macAddress
366	staIPv4Address	ipv4Address
367	wtpMacAddress	macAddress
368	ingressInterfaceType	unsigned32
369	egressInterfaceType	unsigned32
370	rtpSequenceNumber	unsigned16
371	userName	string
372	applicationCategoryName	string
373	applicationSubCategoryName	string
374	applicationGroupName	string
375	originalFlowsPresent	unsigned64
376	originalFlowsInitiated	unsigned64
377	originalFlowsCompleted	unsigned64

378	distinctCountOfSourceIPAddress	unsigned64
379	distinctCountOfDestinationIPAddress	unsigned64
380	distinctCountOfSourceIPv4Address	unsigned32
381	distinctCountOfDestinationIPv4Address	unsigned32
382	distinctCountOfSourceIPv6Address	unsigned64
383	distinctCountOfDestinationIPv6Address	unsigned64
384	valueDistributionMethod	unsigned8
385	rfc3550JitterMilliseconds	unsigned32
386	rfc3550JitterMicroseconds	unsigned32
387	rfc3550JitterNanoseconds	unsigned32
388	dot1qDEI	boolean
389	dot1qCustomerDEI	boolean
390	flowSelectorAlgorithm	unsigned16
391	flowSelectedOctetDeltaCount	unsigned64
392	flowSelectedPacketDeltaCount	unsigned64
393	flowSelectedFlowDeltaCount	unsigned64
394	selectorIDTotalFlowsObserved	unsigned64
395	selectorIDTotalFlowsSelected	unsigned64
396	samplingFlowInterval	unsigned64
397	samplingFlowSpacing	unsigned64
398	flowSamplingTimeInterval	unsigned64
399	flowSamplingTimeSpacing	unsigned64
400	hashFlowDomain	unsigned16
401	transportOctetDeltaCount	unsigned64
402	transportPacketDeltaCount	unsigned64
403	originalExporterIPv4Address	ipv4Address
404	originalExporterIPv6Address	ipv6Address
405	originalObservationDomainId	unsigned32
406	intermediateProcessId	unsigned32
407	ignoredDataRecordTotalCount	unsigned64
408	dataLinkFrameType	unsigned16

409	sectionOffset	unsigned16
410	sectionExportedOctets	unsigned16
411	dot1qServiceInstanceTag	octetArray
412	dot1qServiceInstanceId	unsigned32
413	dot1qServiceInstancePriority	unsigned8
414	dot1qCustomerSourceMacAddress	macAddress
415	dot1qCustomerDestinationMacAddress	macAddress
417	postLayer2OctetDeltaCount	unsigned64
418	postMCastLayer2OctetDeltaCount	unsigned64
420	postLayer2OctetTotalCount	unsigned64
421	postMCastLayer2OctetTotalCount	unsigned64
422	minimumLayer2TotalLength	unsigned64
423	maximumLayer2TotalLength	unsigned64
424	droppedLayer2OctetDeltaCount	unsigned64
425	droppedLayer2OctetTotalCount	unsigned64
426	ignoredLayer2OctetTotalCount	unsigned64
427	notSentLayer2OctetTotalCount	unsigned64
428	layer2OctetDeltaSumOfSquares	unsigned64
429	layer2OctetTotalSumOfSquares	unsigned64
430	layer2FrameDeltaCount	unsigned64
431	layer2FrameTotalCount	unsigned64
432	pseudoWireDestinationIPv4Address	ipv4Address
433	ignoredLayer2FrameTotalCount	unsigned64
434	mibObjectValueInteger	signed32
435	mibObjectValueOctetString	octetArray
436	mibObjectValueOID	octetArray
437	mibObjectValueBits	octetArray
438	mibObjectValueIPAddress	ipv4Address
439	mibObjectValueCounter	unsigned64
440	mibObjectValueGauge	unsigned32
441	mibObjectValueTimeTicks	unsigned32

442	mibObjectValueUnsigned	unsigned32
443	mibObjectValueTable	subTemplateList
444	mibObjectValueRow	subTemplateList
445	mibObjectIdentifier	octetArray
446	mibSubIdentifier	unsigned32
447	mibIndexIndicator	unsigned64
448	mibCaptureTimeSemantics	unsigned8
449	mibContextEngineID	octetArray
450	mibContextName	string
451	mibObjectName	string
452	mibObjectDescription	string
453	mibObjectSyntax	string
454	mibModuleName	string
455	mobileIMSI	string
456	mobileMSISDN	string
457	httpStatusCode	unsigned16
458	sourceTransportPortsLimit	unsigned16
459	httpRequestMethod	string
460	httpRequestHost	string
461	httpRequestTarget	string
462	httpMessageVersion	string
463	natInstanceID	unsigned32
464	internalAddressRealm	octetArray
465	externalAddressRealm	octetArray
466	natQuotaExceededEvent	unsigned32
467	natThresholdEvent	unsigned32
468	httpUserAgent	string
469	httpContentType	string
470	httpReasonPhrase	string
471	maxSessionEntries	unsigned32
472	maxBIBEntries	unsigned32

473	maxEntriesPerUser	unsigned32
474	maxSubscribers	unsigned32
475	maxFragmentsPendingReassembly	unsigned32
476	addressPoolHighThreshold	unsigned32
477	addressPoolLowThreshold	unsigned32
478	addressPortMappingHighThreshold	unsigned32
479	addressPortMappingLowThreshold	unsigned32
480	addressPortMappingPerUserHighThreshold	unsigned32
481	globalAddressMappingHighThreshold	unsigned32
482	vpnIdentifier	octetArray
483	bgpCommunity	unsigned32
484	bgpSourceCommunityList	basicList
485	bgpDestinationCommunityList	basicList
486	bgpExtendedCommunity	octetArray
487	bgpSourceExtendedCommunityList	basicList
488	bgpDestinationExtendedCommunityList	basicList
489	bgpLargeCommunity	octetArray
490	bgpSourceLargeCommunityList	basicList
491	bgpDestinationLargeCommunityList	basicList

B

APPENDIX B

The appendix B present additional figures, mainly referring to code examples in python.

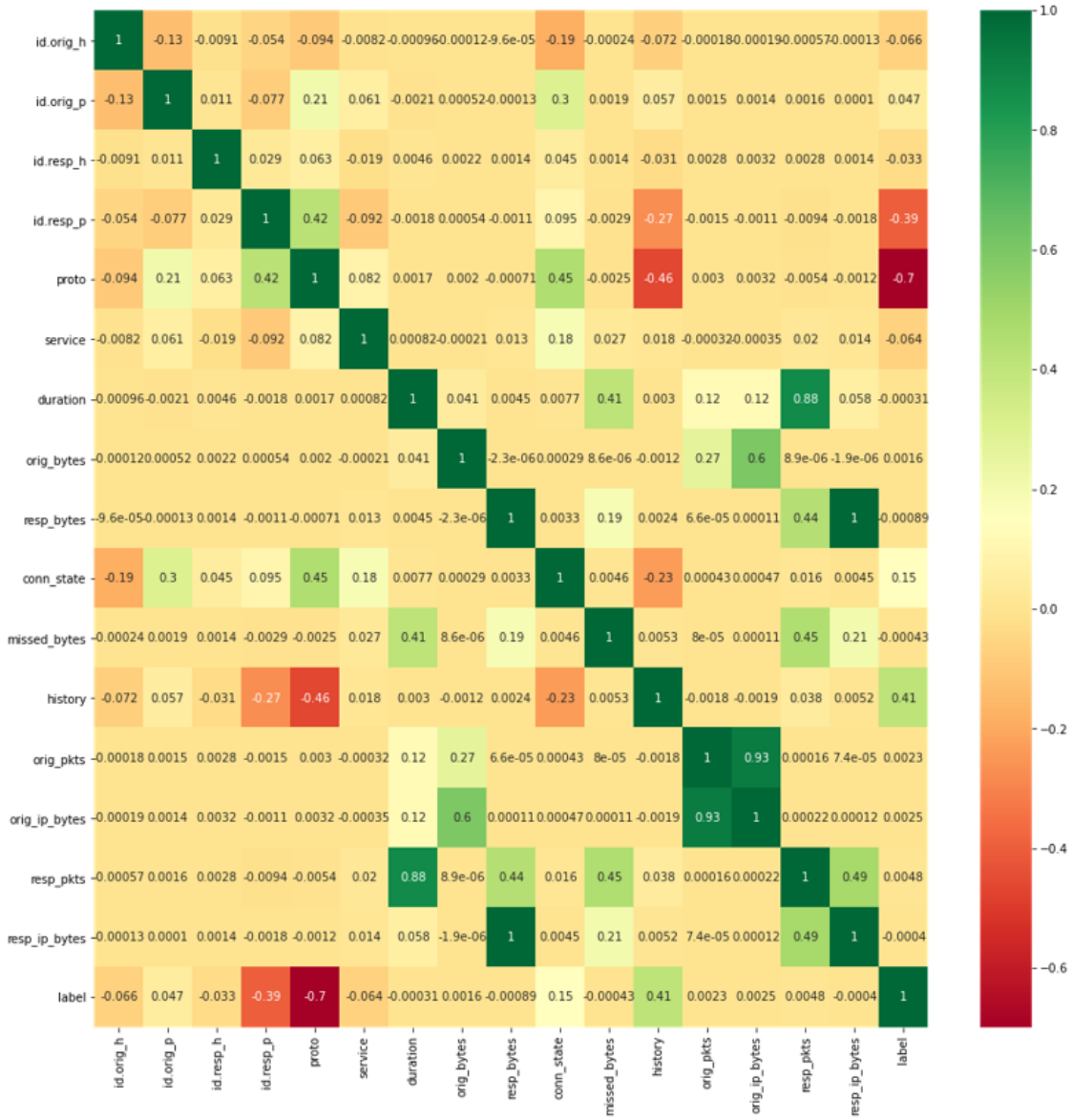


Figure 29: Heatmap features

```
    if x==1:
        print('--------ExtraTreesClassifier-----------')
        model = ExtraTreesClassifier()
        model.fit(df_X,df_Y.values.ravel())
        print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
        #plot graph of feature importances for better visualization
        feat_importances = pd.Series(model.feature_importances_, index=df_X.columns)
        feat_importances.nlargest(12).plot(kind='barh')
        plt.show()
        ft_cols = feat_importances.nlargest(4)
        del model
        del feat_importances

    if x==2:
        print('--------Heatmap-----------')
        aux_df = df_X
        aux_df ['label']= df_Y
        corrmat = aux_df.corr()
        top_corr_features = corrmat.index
        plt.figure(figsize=(16,16))
        #plot heat map
        sns.heatmap(aux_df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
        del aux_df
        del corrmat
        del top_corr_features
return ft_cols
```

Figure 30: Feature selection methods in python

```
#droping columns for binary classification only
df=df.drop(columns=['tunnel_parents','detailed-label'])
df.info()

#Fitting imputer object to the independent variables X.
from sklearn.impute import SimpleImputer
intFill=float(0)
X = df.iloc[:, 0:df.shape[1]-1].values
imputer= SimpleImputer(missing_values = '-', strategy='constant', fill_value=intFill)
imputerimputer= imputer.fit(X[:, 6:9])
#Replacing missing data with the calculated mean value
X[:, 6:9]= imputer.transform(X[:, 6:9])
X_cols = df.iloc[:, 0:df.shape[1]-1]
df_X = pd.DataFrame(X,columns=X_cols.columns)
df_X = df_X.astype({'duration':np.float,'orig_bytes':np.float,'resp_bytes':np.float,#})#,
                    'id.orig_p':np.float,'id.resp_p':np.float,'orig_pkts':np.float,'orig_ip_bytes':np.float,
                    'resp_pkts':np.float,'resp_ip_bytes':np.float,'missed_bytes':np.float})

Y = df.iloc[:, -1].values

#encode string features
cols=df_X.select_dtypes(include="object").columns
from sklearn.preprocessing import LabelEncoder
df_X[cols]=df_X[cols].apply(LabelEncoder().fit_transform)
labelencoder_Y = LabelEncoder()
Y = labelencoder_Y.fit_transform(Y)
#print(Y)
df_Y = pd.DataFrame(Y,columns=['label'])

#print(df_X.info())
#print(df_Y.info())

input_features = df_X.shape[1]
ft_cols = feature_selection_sub(df_X,df_Y,input_features)
```

Figure 31: Binary Feature selection in python

```
if x==0:
    print('---Logistic Regression_Classifier---')
    #Logistic Regression
    linear_classifier = LogisticRegression()
    classify_sub(linear_classifier,MLorDP,X_train, X_valid, y_train, y_valid, X_test, y_test,verbose)
    del linear_classifier
if x==1:
    print('---Random Forest_Classifier---')
    random_forest_classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
    classify_sub(random_forest_classifier,MLorDP,X_train, X_valid, y_train, y_valid, X_test, y_test,verbose)
    del random_forest_classifier
if x==2:
    print('---Naive Bayes_Classifier---')
    naive_classifier = GaussianNB()
    classify_sub(naive_classifier,MLorDP,X_train, X_valid, y_train, y_valid, X_test, y_test,verbose)
    del naive_classifier
```

Figure 32: Traditional ML models in python

```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

# baseline model - model will have a single fully connected hidden layer with the same number of ne
# as input variables.
def dl_model_baseline(input_variables):
    # create model
    model = Sequential()
    model.add(Dense(input_variables, input_dim=input_variables, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model - logarithmic loss function (binary_crossentropy)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    print(model.summary())
    return model

def dl_model_mlp(input_variables):
    # create model
    model = Sequential()
    model.add(Dense(32, input_dim=input_variables, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(4, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model - logarithmic loss function (binary_crossentropy)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    print(model.summary())
    return model
```

Figure 33: DL models in python

```
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
from yellowbrick.cluster import InterclusterDistance

model = KMeans(n_clusters=2, random_state=42)
visualizer = SilhouetteVisualizer(model,colors='yellowbrick')
visualizer.fit(X_test)
visualizer.poof()

model = KMeans(n_clusters=2, random_state=42)
visualizer = InterclusterDistance(model)
visualizer.fit(X_test)
visualizer.poof()

model.fit(X_test)
centroids = model.cluster_centers_
labels = model.labels_
```

Figure 34: K-Means model in python

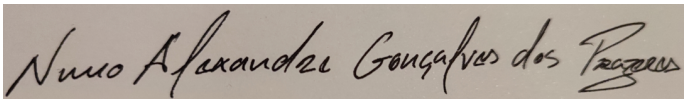
```
#ML models training
def classify_sub(classifier, MLorDP, X_train, X_valid, y_train, y_valid, X_test, y_test,verbose = True):

    from tensorflow.keras.callbacks import EarlyStopping
    from sklearn.metrics import classification_report
    from sklearn.model_selection import train_test_split
    from sklearn import metrics
    import numpy as np

    if MLorDP == 0:
        classifier.fit(X_train, y_train)
        #visual_ml(classifier,x_train, y_train)
    if MLorDP == 1:
        monitor = EarlyStopping(monitor="val_loss",min_delta=1e-3, patience=5,verbose=1,mode='auto',
                                restore_best_weights=True)

        classifier.fit(X_train, y_train,validation_data=(X_valid,y_valid),callbacks=[monitor],
                       verbose=2,epochs=1000)

        plot_validate(classifier, 'acc')
        plot_validate(classifier, 'loss')

    pred = np.around(classifier.predict(X_test))
    print(classification_report(y_test, pred))
    plot_roc(pred,y_test)
    plot_cm(y_test,pred)

    del classifier
    del pred
```

Figure 35: ML models training in python

DECLARATION

I declare, under commitment of honor, that the work presented in this dissertation, with the title *"Data Science in Cybersecurity*
Evaluating the use of Machine Learning in an IoT-IDS", is original and was performed by Student Nuno Alexandre Gonçalves dos Prazeres (2192642) under the guidance of Professor Doctor Carlos Manuel da Silva Rabadão (carlos.rabadao@ipleiria.pt), Professor Doctor Leonel Filipe Simões Santos (leonel.santos@ipleiria.pt) and Professor Doctor Rogério Luís de Carvalho Costa (rogerio.l.costa@ipleiria.pt).

Leiria, November 2021



Student Nuno Alexandre Gonçalves dos Prazeres