Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

AI-based algorithm for intrusion detection on a real Dataset

Estudante: Alejandro Esteban Martínez

Dirección: Bertha Guijarro Berdiñas

María Amparo Alonso Betanzos

Elena María Hernández Pereira

A Coruña, September of 2023.

To the ones that have limited knowledge, long live APCCEL

Acknowledgements

Thanks to all people that made this possible, starting from my uncle David that started my passion for science since I was little. Thank you to my great tutors Berta, Amparo and Elena that made this possible even with the complications I gave all the time. Thanks to all of my friends that supported me, even the ones that didn't support me that much and wanted me to go out drinking with them when I had to work on my Final Project. And special thanks to Fiordilatte for taking me every day of summer and letting me steal some wifi and a table to make this Project possible.

Abstract

In this Project, Novel Machine Learning proposals are given to produce a Network Intrusion Detection System (NIDS). For this, a state of the art Dataset for Cyclo Stationary NIDS has been used, together with a previously proposed standard methodology to compare the results of different models over the same Dataset. An extensive research has been done for this Project about the different Datasets available for NIDS, as has been done to expose the evolution and functioning of IDSs.

Finally, experiments have been made with Outlier Detectors, Ensemble Methods, Deep Learning and Conventional Classifiers to compare with previously published results over the same Dataset and with the same methodology. The findings reveal that the Ensemble Methods have been capable to improve the results from prior research being the best approach the Extreme Gradient Boosting method.

Resumo

En este Proyecto, se presentan novedosas propuestas de Aprendizaje Automático para producir un Sistema de Detección de Intrusos en Red (NIDS). Para ello, se ha utilizado un Dataset de última generación para NIDS Cicloestacionarios, junto con una metodología estándar previamente propuesta para comparar los resultados de diferentes modelos sobre el mismo Dataset. Para este Proyecto se ha realizado una extensa investigación sobre los diferentes conjuntos de datos disponibles para NIDS, así como se ha expuesto la evolución y funcionamiento de los IDSs.

Por último, se han realizado experimentos con Detectores de Anomalias, Métodos de Conjunto, Aprendizaje Profundo y Clasificadores Convencionales para comparar con resultados previamente publicados sobre el mismo Dataset y con la misma metodología. Los resultados revelan que los Métodos de Conjunto han sido capaces de mejorar los resultados de investigaciones previas siendo el mejor enfoque el método de Extreme Gradient Boosting.

Keywords:

- Machine Learning
- Intrusion Detection System
- Ensemble Methods
- Computer Network Attacks
- Deep Learning
- Outlier Detection

Palabras chave:

- Aprendizaje Automático
- Sistema de Detección de Intrusiones
- Métodos de Cojunto
- Ataques de Redes de Computadoras
- Aprendizaje Profundo
- Detección de Anomalías

Contents

1 Introduction **1**

1.1 Context and justification . . . 1
1.2 Materials used . . . 1
1.2.1 Dataset . . . 2
1.2.2 Programming Language . . . 2
1.2.3 Tools . . . 3
1.3 Development Methodology and Project costs . . . 4
1.3.1 Methodology and Phases . . . 4
1.3.2 Costs of the development . . . 6
1.4 Structure of the document . . . 7
1.5 Research objectives . . . 8

2 Theoretical foundations **9**

2.1 Intrusion detection systems . . . 9
2.1.1 Types of IDS based on deployment . . . 10
2.1.2 Types of IDS based on Detection Method . . . 11
2.1.3 Different Data Sources for Intrusion Detection . . . 12
2.1.4 Functioning of an IDS . . . 14
2.2 Computer Network Attacks . . . 14
2.2.1 Different Computer Network attacks . . . 15
2.3 Artificial Intelligence and Machine Learning . . . 17
2.3.1 Main Types of Problems in Machine Learning . . . 18
2.3.2 Classification Models . . . 19
2.3.3 Ensemble Methods . . . 27
2.3.4 Outlier Detection . . . 29

3 State of the Art in the Field **31**
3.1 Datasets for IDS . . . 31
3.1.1 DARPA datasets . . . 31
3.1.2 KDDCup'99 . . . 31
3.1.3 Improvements Proposed for KDDCup'99 . . . 32
3.1.4 More recent proposals . . . 33
3.2 The UGR'16 Dataset . . . 36
3.2.1 Calibration Set . . . 37
3.2.2 Test Set . . . 38
3.2.3 Format of the Dataset . . . 38
3.2.4 Composition of the Dataset . . . 39
3.3 Evolution of Machine Learning for IDS . . . 39
3.3.1 Early Attempts and Rule-Based Systems (late 1980s - early 1990s) . . . 39
3.3.2 Emergence of Anomaly Detection (1990s - early 2000s) . . . 39
3.3.3 Machine Learning Integration (2000s - early 2010s) . . . 39
3.3.4 Evolution of Deep Learning (2010s - Present) . . . 40
3.4 Standard Methodology for NIDS development . . . 40

4 Development of the IDS **41**
4.1 Free Framework for Machine Learning (FF4ML) . . . 41
4.1.1 Previous results from FF4ML methodology with UGR'16 . . . 42
4.2 Analysis of the UGR'16 Dataset . . . 43
4.2.1 Data Collection and Cleaning . . . 44
4.2.2 Data Transformation . . . 44
4.2.3 Evaluating the Hypothesis . . . 45
4.3 Replication of the original results . . . 47

5 Models Development and Results **51**
5.1 Metrics used . . . 51
5.2 Utilizing conventional classifiers . . . 52
5.2.1 Experimentation with Over and Under-Sampling . . . 55
5.3 Deep Learning models . . . 56
5.4 Combining Models: Ensemble Methods . . . 60
5.4.1 Voting Ensemble . . . 60
5.4.2 Stacking Ensemble . . . 63
5.5 Using models for Outlier Detection . . . 65
5.5.1 Experimentation with Outlier Detectors . . . 65
5.6 Discussion of Results . . . 67

6 Conclusions and Future Work **69**
6.1 Conclusions . . . 69
6.1.1 Researcher Conclusions . . . 69
6.1.2 Student Conclusions . . . 70
6.2 Future work . . . 71
6.2.1 Experiment with the raw UGR'16 Dataset . . . 71
6.2.2 Explainable and Green Artificial Intelligence . . . 71
6.2.3 Apply the methodology in other NIDS Datasets . . . 72
6.2.4 Develop a working NIDS . . . 72

A Additional Material **74**
A.1 Full Performance Metrics Tables . . . 74

List of Acronyms **77**

Bibliography **78**

List of Figures

1.1 Predicted Plan for the Project . . . 6
1.2 Final Plan compared with baseline . . . 6

2.1 Suport Vector Machine functioning from [1] . . . 21
2.2 Comparison of different SVM kernels from [2] . . . 21
2.3 Functioning of the Random Forest from [3] . . . 23
2.4 Example of Multilayer Perceptron from [4] . . . 26
2.5 kNN binary classification example from [5] . . . 27
2.6 Comparison of Bagging and Boosting from [6] . . . 28
2.7 Example of Isolation Forest functioning from [7] . . . 30

3.1 Diagram of the ISP network from [8] . . . 37

4.1 Diagram of the Methodology from [9] . . . 41
4.2 Confusion Matrix of the Random Forest . . . 50

5.1 Confusion Matrices for KNN and XGBOOST Models . . . 54
5.2 Comparison of performance of the proposed classifiers with the Random Forest 55
5.3 Confusion matrix for the 2 Layered MLP . . . 58
5.4 Comparison of performance for Random Forest and MLP . . . 59
5.5 Comparison of Elapsed Time for Random Forest and MLP . . . 59
5.6 Confusion Matrices for Voting with uniform and custom weights . . . 62
5.7 Comparison of performance for XGBoost and Voting Classifier with custom weights . . . 63
5.8 Confusion Matrices for Stacking with and without passthrough . . . 64
5.9 Comparison of best results over the considered groups . . . 68

List of Tables

1.1 Predicted cost . . . 7
1.2 Final cost . . . 7

2.1 Comparison of Network-based IDS (NIDS) and Host-based IDS (HIDS) . . . 11
2.2 Comparison of Signature-based Detection and Anomaly-based Detection . . . 12
2.3 Comparison of Gradient Boosting and Random Forest . . . 24

3.1 Comparison of Selected Datasets for IDS Evaluation from [8] . . . 35
3.2 Advantages and Disadvantages of Intrusion Detection Datasets . . . 36

4.1 Class Distribution from subsampled Test file . . . 46
4.2 Performance comparison for Random Forest trained with the Test set and tested with multiple sets . . . 47
4.3 Best hyper-parameter configurations for the replicated models . . . 48
4.4 Replicated performance for the original paper models following their methodology . . . 49
4.5 Elapsed time of training and testing for the replicated models . . . 50

5.1 Best configurations for the conventional classifiers . . . 53
5.2 Weighted Performance for proposed conventional models . . . 53
5.3 Elapsed time of training and testing for the conventional classifiers . . . 53
5.4 Class Distribution . . . 56
5.5 Best configurations for the MLPClassifier models . . . 57
5.6 Weighted Performance for the MLPClassifier models . . . 57
5.7 Elapsed time of training and testing for the MLP configurations . . . 57
5.8 Weighted Performance for Voting Models . . . 61
5.9 Weighted Performance for Stacking classifiers with and without passthrough . . . 64
5.10 Best configurations for the conventional classifiers . . . 66

5.11 Weighted Performance for Outlier Detection Models 66

A.1 Performance for proposed conventional models 74
A.2 Performance for the MLPClassifier models 75
A.3 Performance for Voting Models 76
A.4 Performance for Stacking classifiers with and without passthrough 76

Chapter 1
Introduction

1.1 Context and justification

With the start of the Information Era we find ourselves in a more connected world than ever before, it is possible to send information from one point to another separated by thousands of kilometers in mere seconds and all because of the Internet. The issue is that as we develop new technologies and make advancements, we also create new and more intricate problems. For example, the Internet, while not perfect, has been a significant technological advancement. However, it has also brought about a crucial problem since it can be accessed remotely, rendering traditional physical security methods ineffective against internet attacks that exploit vulnerabilities in our layered technology. This includes legacy components that are challenging to replace and susceptible to hacking.

One of the possible solutions is the implementation of an Intrusion Detection System (IDS), that monitors network traffic for suspicious activity and in case of a possible attack alerts the interested party, usually a system administrator.

In this work, we will explore different state-of-the-art technologies that are used or could be used to improve the IDS detection abilities, developing an Artificial Intelligence approach for our system trained with the UGR'16 dataset, one of the latest and more complete network flow datasets that are publicly available. To implement our AI system, we will follow a recently proposed standard methodology for the evaluation of Machine Learning models for Network Intrusion Detection Systems, and we will attempt to improve our results compared to the currently published for the UGR'16 dataset.

1.2 Materials used

In this section, we will introduce all the necessary materials and tools for the development of this Final Project. With this comprehensive overview, we will define the elements needed,

from datasets to software apps and programming languages.

1.2.1 Dataset

For this project, we decided to use the new **UGR'16 dataset**[8], since it's considered one of the newest and most reliable for this kind of problem. Some other datasets that had been used in the past do not contain this much data, it is mainly synthetic data or it is too old for it to be useful for a contemporary project, as some attacks are not included.

UGR'16 dataset is used for Intrusion Detection Systems that use cyclo stationarity-based techniques, which use the periodicity and correlation in the statistical features of data to distinguish between normal and anomalous behaviour and to classify the possible type of attack. It was created by a group of researchers in the Universidad de Granada in 2016, where it takes its name, and it is built with a mixture of real traffic and selected up-to-date synthetic attacks within a network of a Spanish Internet Service Provider collected into netflow packets. Divided into two different sets, calibration and test, the first one spans from March to June with only real traffic while the second one has introduced synthetic attacks apart from the background data from July to August. The dataset differentiates up to 7 different classes of data, from background traffic to Dos, DDos, port scanning, brute force, botnet and email spam campaigns. This makes it the state of the art in its field, including much more realistic data and diversity than its predecessors and capturing much longer time than the rest, making it possible to discern periodic patterns in the network activity.

This dataset is available to download in the Universidad de Granada's webpage[10] along with the 2017 paper exposing its results.

1.2.2 Programming Language

The selection of the language used and its libraries is fundamental for the development of our project. We decided to use **Python** [11] as it is the most used Programming Language in the field of Data Science and Artificial Intelligence.

Python is known for its simple and clean syntax and even though it is an Interpreted Programming Language which makes it quite slower than Compiled Languages such as C and C++, the community of developers is so rich that there is a huge number of open-source libraries available for any user that allow data manipulation, Artificial Intelligence and more in a much more straightforward and efficient way than other languages.

Specifically, we will use Python 3.9, and we also rely on the use of the following commonly known libraries:

- **scikit-learn (sklearn)** [12][13]: Probably the best known Machine Learning library. It offers a very big and powerful set of tools for classification, clustering, regression and

more such as tools to be able to explain the results of the model trained. It is easy to use even for beginner engineers so it has become standard in its field.

- **pandas**[14][15]: It is used for analysis and manipulation of data, simplifying the use of datasets in different formats and making possible and efficient the preprocessing of data and its transformation.

- **numpy**[16]: Important package for scientific computing, easing array operations and computations while making them efficient.

- **scikit-optimize (skopt)** [17]: Probably the less known library of the ones used, it is used for Bayesian Optimization techniques, mainly because of its BayesSearchCV implementation for hyper-parameter tuning. This has been tested as given almost optimal results in hyperparameter tuning in less time than the rest of tuners[18].

1.2.3 Tools

Without some tools, the development of this Project wouldn't have been possible, as these assist us by making our coding more efficient and comfortable as well as for allowing the transmission of information to the tutors for it to be reviewed and saved. Next, we will present the main tools used, how they work, and how we use them.

- **Pycharm Professional**[19]: This Integrated Development Environment (IDE) supports us with auto-completion of the functions and syntax correction tools. It also integrates functionalities like Github that will be presented later and enables debugging to make it possible to understand everything that happens within our program. Its license is given by the University of A Coruña (UDC)

- **CITIC Supercomputer**: A supercomputer SHH connection was lent by the CITIC[20] to make it possible to do computations of large amounts of data and memory-consuming tools that in our personal computers wouldn't have been possible. It also allowed us to compute very GPU and CPU intensive programs that couldn't have been used otherwise.

- **Personal Computer**: An ASUS laptop, property of the student, was used in the development of the project for writing the Final Report, having the project meetings and doing the proposed models computations, from training to evaluating. The hardware spec of the laptop are:

  - CPU: 11th Gen Intel Core i5 - 11400H (from 2.7GHz to 4.5GHz) with 6 cores
  - RAM: 16GB DDR4 SO-DIMM

  - GPU: NVIDIA GeForce RTX 3060 6GB GDDR6
  - Storage: 512 GB SSD M.2 NVMe PCIe 3.0

- **Github**[21]: It serves as a centralized hub for version control of our code. It made it possible for the review of each version of the code to return to older versions of it and to save newer ones for those to be accessed on any device that is available at the moment.
- **Microsoft Teams**[22]: This very important communication tool made it possible to have meetings with the tutors of the project and to send information such as important papers to take into account the results of multiple computations of models. The license was also given by the UDC.
- **Overleaf**[23]: Tool that enables the editing and writing of LaTex[24] documents, a markup language that allows us to write our document in plain text instead of having to do the format by ourselves like with Microsoft Word. Currently the standard in scientific and engineering publications.
- **arXiv**[25]: A platform that permits us to access, thanks to the University license, multiple scientific papers that turn out useful for the Project.
- **CherryTree**[26]: It is a hierarchical note-taking tool that allows us to organize the progress of every meeting and the work done outside of those. It allowed us to keep a comprehensive record of our short-term objectives.

1.3 Development Methodology and Project costs

In this section, we will introduce the proposed methodology to follow for the development of the Project and its costs.

1.3.1 Methodology and Phases

For the project a SCRUM Methodology was used from the beginning of the development, having progress checkups every two weeks and establishing new short-term objectives to fulfill the final ones. From the beginning of the project until half of June, that was the periodicity of meetings, but after that moment we decided to start weekly meetings.

Phases of the project

From the beginning of the project, we proposed a series of phases to be followed so as to be organized and ordered. These were:

1. **Study of Key IDS Development Techniques with Machine Learning**: In this step we immerse ourselves into a series of state-of-art techniques for IDS development, focusing mostly on Machine Learning-based technologies

2. **Dataset Analysis**: We fully analyze the contents of the UGR'16 dataset to be able to understand how it is formed, its structures, and how could we use it for the betterment of our IDS models.

3. **Acquaintance with Data Analysis Tools**: The dataset is a large one, with 4000 Million rows, each with 13 distinct variables and a size 1.8 TB. Thus, after having studied it and the different techniques for Machine Learning IDS development, we will also have to familiarize ourselves with the necessary tools to explore this data and transform it.

4. **Data Cleansing and Transformation**: Now that the dataset is known and also the useful tools for using it for our goals we can begin the process of modifying it for our use. As the Dataset is not in the necessary format for training Machine Learning models with values like the time of flow in normal string YYYY:MM:DD that have to be changed to Unix timestamps

5. **Replication of Previous Approaches**: It is important to replicate previous studies, with a double objective. On the one hand, checking the adequacy of the implementations carried out, and on the other hand, it is important to understand the nuances if we want to be able to improve the state of the art results.

6. **Development of Novel Proposals**: Once we have been able to replicate previous approaches, so as to compare the different results of the state-of-the-art models, we can begin with our proposals for an improvement in said results.

7. **Analysis and Comparison of Models**: After doing the necessary tests to be certain that the results we obtain are correct we can start analyzing them to understand their inner meaning and compare them with the previous results and also with our newer proposals.

8. **Finalizing and Reporting:** Final phase of the development of this project where the report is written with all the work done, needed information and the conclusions of the proposals including gained knowledge and resources used learned from the University Grade in Computer Science.

In figure 1.1 we do an initial planning of the project to be compared with the final development plan and analyze the deviations. Once established the baseline we can compare it with our final Plan in figure 1.2. As we can see the initial estimation of the duration of the project

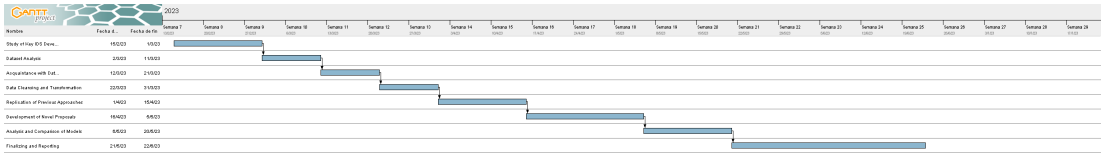


Figure 1.1: Predicted Plan for the Project

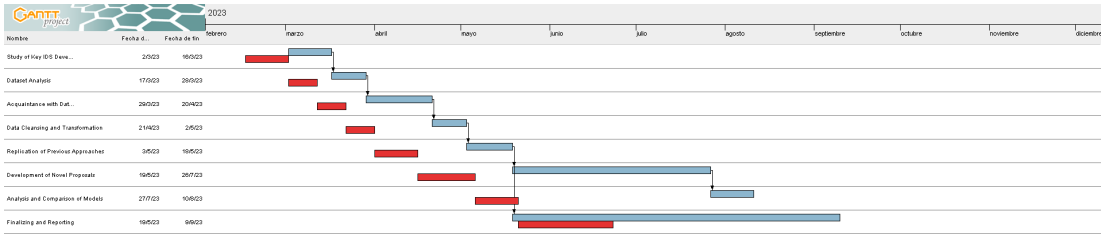


Figure 1.2: Final Plan compared with baseline

ended up being different from the final progress. The most notable change is the start date, as it started approximately 2 weeks after the predicted data. This can be explained by knowing that the student did an stay in a foreign country and had to still finish the examinations necessary in that country.

Also the sizes of the time span of the 'Development of Novel Proposals' phase and the 'Finalizing and Reporting' have increased due to the beginning of the professional career of the student. It was also decided, due to the time-consuming essence of training and testing Machine Learning models, to begin the writing process in parallel to the development phase. We had a deviation of 79 days from the original plan finish date.

1.3.2 Costs of the development

For the predicted costs originally we will use the European Credit Transfer System (ECTS) credits that are given to the subject 'Final Project. Mention in Computer Science' [27]. As the subject has 12 ECTS credits, the corresponding hours amount to a total of 300, of which 275 are to be done solely by the student as autonomous work and the remaining 25 should be worked with the student tutors. To calculate the costs we access the average salary of a Programmer and of a Project manager in Spain and multiply by their corresponding work hours. As for material costs, they will not be counted in the final prediction because the only cost there will be is the electricity costs and the amount to be insignificant compared to the personal costs.

We can observe the total predicted cost in the table 1.1. But as the initial prediction for the plan was wrong, as seen in figure 1.2, our cost calculations also changed. Both the Programmer and the Project Managers worked hours diverged from the prediction, with an approximate total of 400 hours spent working in the project for the Programmer and 20 for the

Project Managers, 12 total meetings and 8 hours each for analyzing the Project Report. the total calculated cost is given in table 1.2. As we can see the increase in cost is 29% from the original planning, and it makes sense since the project had a deviation of 79 days, 62% more than the predicted in figure 1.1

Resource	**Worked Hours**	**Hourly Rate**
Programmer	275	15 €/h
3 Project Managers	25	54 €/h
Total Cost		**Cost**
Programmer + Managers		5475€

Table 1.1: Predicted cost

Resource	**Worked Hours**	**Hourly Rate**
Programmer	400	15 €/h
3 Project Managers	20	54 €/h
Total Cost		**Cost**
Programmer + Managers		7080€ (+29%)

Table 1.2: Final cost

1.4 Structure of the document

In this section, we will introduce the chapters that this document contains:

- **Introduction**: It contains the project's motivation, presents the materials and methodology used and provides the structure of the memory
- **Theoretical Foundations**: We bring the most relevant theoretical context for the understanding of the project, delving into topics like Intrusion Detection Systems, Artificial Intelligence and Machine Learning.
- **State of the Art in the Project's field**: In this chapter we do a revision of the newest information available about our research, presenting the different databases that exist for IDS and how these are being used.
- **Development of the IDS:** In this chapter, the previous knowledge for the experiments to be done is presented, explaining the standard methodology that will be used, previous experiments with the raw Dataset and finally the replication of previous results to establish a baseline of comparison.

- **Implementations and Experiments**: We detail here the different proposals for improvement that were made, what kind of experiments were conducted, and which were the general results obtained. At the end, we will critically analyze the outcome of our experiments in detail, comparing them with the previous results from the latest research included in the second chapter.
- **Conclusions and Future Work**: Finally some conclusions of the project are provided with a summary of our discoveries and the contributions that were made through this project and what else could be explored in the future, with different applications and improvements to be made.

1.5 Research objectives

The main objective of this work is to develop an improved Intrusion Detection System (IDS) by utilizing the newly available datasets that consist of real and synthetic network traffic data. The work will focus on the following key points:

- Analysis of different techniques of Automatic Learning for IDS development.
- Utilization of Data Science techniques to handle the large size of the UGR'16 dataset.
- Proposal of an IDS based on artificial intelligence, trained and tested with the UGR'16 dataset, and comparison with previous AI models used for the same dataset.
- Analysis of the obtained results to evaluate the effectiveness of the proposed model and identify future research directions.

Chapter 2

Theoretical foundations

Intrusion detection is in the present days a fundamental component of computer security systems, safeguarding networks and devices against possible threats and unauthorized activities. For this project, it's necessary to fully understand the theoretical bases of these systems and the advantages brought by the use of artificial intelligence and Machine Learning techniques.

The following chapter will provide an overview of the field of intrusion detection. We will start with a presentation of a definition of it, the main objectives of this kind of system and the challenges that appear when trying to achieve an accurate and robust detection.

Next, we will expose its course of history. Since its conception in the 80s with rule-based systems, that use predefined rules to detect stated known attack patterns, to later systems that include anomaly detection methods that focus on finding deviations from "learned" normal behaviour. And finally the latest use of Machine Learning and artificial intelligence with the increase of large-scale available datasets of network traffic.

In summary, this chapter serves as an important introduction to all the knowledge needed for this Final Project Assignment, providing the background and understanding needed to comprehend the advancements and alternative methods for intrusion detection presented in our research.

2.1 Intrusion detection systems

Intrusion Detection Systems (IDS) act as the burglar alarm but in the cyber-security field. It defends the system by alerting when it could have been compromised by an outsider's attack or unauthorized access. It is different from the called Intrusion Prevention Systems (IPS) that identify potential threats like the IDS but take action to end the threat and solve its damage.

But a more proper definition could be a device or software that monitors network traffic and alerts when a possible malicious activity is detected. This warning is then often passed to

a Security Information and Event Management (SIEM)) or to an administrator that processes the information given and resolves whether the notification is a mere false alarm or if it represents a real danger.

For this, it uses multiple data sources like network packets, user activity or system logs that are afterwards analyzed, depending on the system, with many possible tools, like predefined rules, signatures or patterns of common attacks, statistical analysis, or, as is the main point in this project, with Artificial Intelligence (AI) or Machine Learning based algorithms.

IDSs can be categorized based on their deployment type and the detection methods they employ, as presented in the next sections of this document.

For research and writing of this next section [28] [29] and [30] have been used.

2.1.1 Types of IDS based on deployment

Intrusion Detection Systems can be differentiated by two main deployment types, Network based IDS (NIDS) and Host based IDS (HIDS). In this subsection, we will explore their differences.

Network Intrusion Detection System (NIDS)

These systems are deployed to at a planned point within a network to monitor all traffic from all the connected devices to the network. It verifies the content and the metadata of all packets and alerts the administrator of the network for any inconsistency.

It's normally located at the same location as the firewall of the network to provide a safeguard in case any attack or breaching of the firewall is tried.

Host Intrusion Detection System (HIDS)

These systems run each on its own independent device of the network. Instead of reading the whole flow of data of the network where it is deployed, it monitors only the entering and exiting packets from the host it is from. It uses snapshots of the state of the existing files to be able to check if after an alert any of them have been modified. In this case, the SIEM is notified.

These are most commonly used in mission-critical machines that don't normally change their configuration.

Hybrid Intrusion Detection System

A system of this kind is made by combining the two approaches explained before, using both the host with the system data and the network information to get a complete view of the network system.

The following table 2.1 shows the differences between the two main deployment types, Network and Host based IDSs, since the third type, Hybrid IDSs, are a combination of the previous two, therefore it is unnecessary to include it in the comparison.

Aspect	Network-based IDS (NIDS)	Host-based IDS (HIDS)
Focus	Monitors network traffic and events at network level	Monitors events on individual hosts or devices
Scope	Covers multiple devices and systems in the network	Limited to the host it is installed on
Data Source	Analyzes network flows, firewall logs, etc.	Analyzes system logs, file integrity, etc.
Detection	Detects network-level attacks, intrusions, and anomalies	Detects local host-level attacks and unusual activities
Visibility	Provides an overview of network-wide threats	Offers detailed insight into host-specific threats
Deployment	Positioned strategically within the network	Installed on individual hosts or devices
Performance	Can handle high-volume network traffic	Less affected by network traffic, but resource-intensive on hosts
Management	Centralized monitoring and alerting for the entire network	Distributed monitoring and alerting for individual hosts
Examples	Snort, Suricata, Bro/Zeek	OSSEC, Tripwire, AIDE

Table 2.1: Comparison of Network-based IDS (NIDS) and Host-based IDS (HIDS)

2.1.2 Types of IDS based on Detection Method

IDSs can use a variety of methods to analyze the given data and detect threats within its deployment field. These can be encased into three main categories that we will explore now: Signature-based detection, Anomaly-based detection and Hybrid Detection.

Each type has its own unique characteristics and their use will depend on the requirements of the environment where our IDS is deployed.

Signature-based Detection

Also known as Rule-based detection, the systems that employ this method of detection use a series of predefined rules or signatures for known attacks, also known as patterns, comparing the input data with these saved signatures and sending an alert if there is a match.

This method for detection is very effective towards already known attacks but can have problems with newer ones, as it requires a constant update of its signature database to avoid false negatives and won't be able to recognize infections that are too novel for it to know its patterns, such as zero-day attacks.

Anomaly-based Detection

Systems that use this method center their detection around developing what we can call a baseline of normal network behaviour and then comparing the input data with this baseline to identify a possible deviation, based on the principle that abnormal data will also hide abnormal patterns.

In contrast to the Signature-based detection systems, Anomaly-based detection systems can positively detect unseen attacks and zero-day malicious behaviour. But IDSs that employ anomaly detection can also be more prone to give false positives, identifying normal data as a possible attack, because of the intrinsic variations in the network traffic or the host system.

Hybrid Detection

Nowadays is more common to find IDSs that employ both methods for detection. This way these systems can sum the strengths of both while reducing their weaknesses. By combining both Signature-based and Anomaly-based detection methods we can have the ability to detect known and unknown attacks.

The following table 2.2 exposes in a simplified way the differences between the Signature and the Anomaly-based Detection Methods.

Method	**Signature-based Detection**	**Anomaly-based Detection**
Characteristics	Relies on predefined patterns or signatures of known attacks	Identifies deviations from normal behavior based on a baseline model
Advantages	• Highly effective at detecting known threats • Low false positive rate • Simple model creation	• Detects novel attacks • Adaptability • Covers a broader range of threats
Limitations	• Ineffective against new or zero-day attacks • Requires constant updates • Limited coverage	• Higher false positive rate • Complex model creation • Difficulty in setting thresholds

Table 2.2: Comparison of Signature-based Detection and Anomaly-based Detection

2.1.3 Different Data Sources for Intrusion Detection

We have now explained different types of IDSs, from the deployment to the method they use to classify the threats, but we still have to define what data they use to determine an attack.

In this subsection, we will explain what data is gathered by the network and host IDSs.

Network-deployment Data

The Network IDS obtains its data from the network traffic, getting an overview of the activities happening across the entire network where it is deployed. That data consists in:

- **Network Flow**: This flow is represented as a sequence of packets that detail the communication between different hosts, with the IPs, ports and protocols involved. This flow also contains more information about the transaction like its timestamp or the number of packets sent.
- **Network logs**: These logs are files produced by the different components of the network such as the router or a firewall are very valuable for the detection of abnormal traffic. They contain a record of the events that occurred in the network with their severity and information of the hosts involved.
- **Packet data**: The raw network packet is captured when it passes through the segment where our NIDS is located. With it, we are able to inspect all of the packet's information, including its headers, application data or commands.

Host-deployment Data

The Host IDS, instead of selecting the network traffic, is centered on gathering the events and activities that happen in a single host. The data that it will process is the following:

- **System logs**: Also called syslogs, these files record the events and activities of the device and provide essential information, including event types, severity, timestamps, event descriptions, and user involvement. With these logs, the HIDS can detect anomalies in the device record of events.
- **File Integrity Monitoring**: It consists of recording in a database the known normal status of the host files, from the file paths to their attributes. By having a baseline of the device's file normal status, the HIDS can detect modifications in size, permissions, etc. This way if any file has been tampered with, like if a worm has been hidden inside, there is a way to know it.
- **Application logs**: They are files with the record of the activity, errors and events produced by software applications. With information similar to the System logs but in a different scope, it is possible to detect if any unusual behaviour of unauthorized execution has happened.

- **Process Monitoring**: The HIDS can monitor the processes running in the machine to check for unauthorized executions and strange processes that consume too many resources, communicate with other hosts without permission or commit malicious activity.

2.1.4 Functioning of an IDS

In this section, we will explain how the IDS works.
As we stated before, the IDS monitors network traffic and notifies in case of a possible malicious intention. This process can be separated into multiple steps:

1. **Data Collection**: Data is gathered from multiple sources to be reviewed by the IDS.
2. **Preprocessing of Data**: Before all data can be analyzed, it passes this crucial step to extract all the valuable information and reject the useless or incomplete data.
3. **Use of Analysis Techniques**: After the data is correctly transformed, the next step is to discern if the incoming information is part of the normal network traffic and user data or if some part of the system might have been compromised. To make this happen, several methods are applied, as it was explained previously.
4. **Alert Generation**: If the possible malicious activity is identified by the analysis technique, an alert with the necessary information of the detected anomaly is generated and it passed to the SIEM.
5. **Response to the Alert**: In this step, the IDS is no longer involved directly as it is the SIEM or the system administrator the one that, given the proper information, assesses the importance of the alert and selects the proper actions to be performed.

2.2 Computer Network Attacks

While talking about the workings of an IDS we mentioned multiple times the words attack and malicious behaviour. In this section, we will introduce what this is and give some explained examples of these instances that the IDS has the objective to detect.

A Computer Network Attack is an action or activity that aims to compromise Confidentiality, Integrity or Availability. These 3 factors are commonly called the CIA Triad [31] as a model to guide information security policies in all organizations. Every Computer Network attack can be assigned to attempt to break one or a combination of these principles.

This attack can be the result of malicious intent or unintentional actions due to negligence or just a lack of knowledge that can result in a security breach.

As we will see next, these can range to a very big set of attacks from gaining access without proper authorization to making the functioning of the System complicated or impossible, overloading it.

2.2.1 Different Computer Network attacks

Now that we have explained what are the different kinds of attacks and what a Computer Network attack is we can explain the different kinds there are of this. But first, we will provide information about the previously presented term, the CIA Triad and its components: Confidentiality, Integrity and Availability.

The CIA Triad

The CIA Triad, as stated before, is used as a framework to take into account when designing and implementing security features against possible threats and risks.

We will now present the components:

- **Confidentiality**: This means that all data being used is only accessible by the authorized parties. By implementing it into our security measures we prevent unauthorized users from accessing, obtaining or viewing this data.

  It can be implemented in various ways like encryption of the information, authentication to interact with it, or access control.

- **Integrity**: It assures that all data remains as it is, accurate, complete and without alterations. When implemented we can prevent modifications that are not authorized, either by accident or by a user with bad intentions.

  For its implementation we can use from hashing to checksums or digital signing to be able to detect any tampering in our information or possible corruption of data and in some cases be able to revert it.

- **Availability**: All data is accessible by the authorized parties when needed, preventing service interruptions or disconnection of critical systems.

  It can be achieved by physical and digital means. A physical way to prevent an availability error can be to have multiple backups or redundancy in the system to be able to solve it as soon as possible or even make it unnoticeable by the interested user. By digital ways we mean implementing software tools that avoid possible attacks that try to rid us of our System response by overloading it like with Denial of Service (DOS) attacks.

Confidentiality Attacks

The most common attacks that attempt to surpass this principle are: Phishing attacks, Eavesdropping and Data breach.

- Phishing: This is a technique where attackers send fraudulent emails or messages that attempt to pass as legitimate sources, such as banks, companies, or government agencies to get the confidence of the recipient. Once the trust has been forged they trick the users into revealing their personal or financial information, like passwords or identification data that could be used for impersonation.
- Eavesdropping: This is based on the ability of the attacker to intercept the communication between the authorized parties without the proper permissions to attempt to capture said data for the possibility of extracting valuable information.
- Data breach: Common when there is a vulnerability in the System that allows unauthorized access to some or all of our data. This vulnerability can be in any part of the chain of the security process, either by faulty equipment, an error in software, or a personal mistake.

Integrity attacks

For attempting to tamper with data without permission there are three most common examples: Injections and Man-in-the-middle attacks.

- Injections: This refers to any attempt to trick the attacked parties into executing some unknown code to try to manipulate, access or damage their data. The most common example is the SQL injection. It consists of finding a vulnerability, normally in a search engine or a web application that interacts with a database, that allows the attacker to send in their command and expression that is read, instead of a search for the expression, as some other command to be run.
- Man-in-the-middle: With this technique, the attacker attempts to disguise themselves as a hidden intermediary between the connection of two or more parties, being able to analyze their communications like with a confidentiality attack and having the ability to alter or redirect the exchanged data.

Availability attacks

To disrupt the normal flow of data in a system or deny access to a network there are two main dangers: Denial-of-service and Ransomware.

- Denial-of-Service: most known for its acronym, DoS, is a type of attack that consists of flooding the system or network with more traffic or requests than it is able to receive in normal conditions. This causes slow functioning or even a complete stall. It can either be done by one or multiple attackers in a distributed way, called DDoS, Distributed Dos.
- Ransomware: is a kind of malware that encrypts the data of the system and demands, as the name says, a ransom to restore the victims' access to their data. It is probably the most renowned kind of attack because of the very big expansion in 2017 of the WannaCry ransomware that affected 200.000 devices worldwide, including hospitals and critical infrastructure of cities [32].

2.3 Artificial Intelligence and Machine Learning

Mostly known in popular culture as AI, it represents the intent of generating machines or software that is able to reproduce or perform tasks that would normally require human intelligence. This can be used in a very diverse number of domains from the most famous now like Natural Language Processing, due to the new Large Language Models like ChatGPT[33], to diverse fields as Healthcare of even Law.

In this section we will explore the concepts of Artificial Intelligence and Machine Learning that are instrumental in the development of this Final Project.

Machine Learning[34] is a branch of Artificial Intelligence that utilizes datasets to learn patterns from data and perform functions that would require human intelligence, with the objective of being able to solve new problems learning from the given dataset without having the process explicitly programmed. This capability allows the Machine Learning model to better adapt to future incoming information, to which it must apply the previously constructed model generated through prior training.

These Machine Learning models are made to solve different problems, that we will describe in a subsequent section, and can be categorized into three main groups: classification, regression, and clustering.

However, before delving into that, we must explain how the training of Machine Learning models works, which can be categorized into four distinct types.

Supervised Learning

In this type of learning, the dataset consists of tuples <attributes, value>. The attributes represent the information we wish to provide to the model during training, and the value or label represents the final outcome that a perfectly trained model should achieve after analyzing the given attributes.

In supervised learning, the typical training process of a Machine Learning model involves at least one training set and one testing set, which are necessarily distinct from each other. Generally, data should not be repeated between these two sets.

The first set, the training set, will be the larger one in terms of data quantity and will consist exclusively of the attribute portion of the <attribute-value> tuple. It is with this set that the model will attempt to learn hidden patterns within our data, enabling it to solve the given problem.

On the other hand, the testing set will only be used after the training is complete, allowing us to evaluate how our model performs on data it has never encountered before.

Unsupervised Learning

This type of learning differs from the previous one in that it lacks the value for each data point, having data consisting only of a set of unlabelled attributes. As a result, the goal is to identify hidden patterns in the data that allow grouping based on their attributes.

In this type of learning, there isn't a separation of the dataset into a training and a test set because we don't have any labeled data to compare with.

Semi-Supervised Learning

Sitting between supervised and unsupervised learning, this type of learning involves tuples of <attribute-value> pairs but only for a limited number of previously labeled data points. This technique is typically used when there is a scarcity of pre-labeled data but enough of it to provide support in our unsupervised training.

Reinforcement Learning

Unlike supervised learning, reinforcement learning doesn't initially provide data with a label for tuning the model. Instead, it employs a trial-and-error process where the model explores various possible actions with the aim of maximizing or minimizing a score depending on the problem at hand, in order to reach an optimal solution.

2.3.1 Main Types of Problems in Machine Learning

In this subsection, we will explore the three main types of problems that can be addressed using classical Machine Learning models: classification, regression, and clustering.

Classification

This type of problem, typically solved using supervised or semi-supervised learning, involves assigning a predefined class to each data point. Classification problems can be binary, where

the classification occurs between two exclusive classes, or multi-class, where there are three or more classes into which the data can be categorized. There is also a special type of classification known as multi-label classification, where classes are not completely exclusive, and each data point can belong to more than one class simultaneously.

An example of classification is predicting whether a given text is in English or Spanish or classifying an email as spam or non-spam based on its characteristics.

Regression

In regression problems, the goal is to predict continuous numeric values based on the provided attributes. The primary distinction between classification and regression problems lies in the type of output data. Classification involves discrete values, while regression deals with continuous values.

A typical example of a regression problem is predicting the value of a property based on its attributes such as location or size, or forecasting the temperature a day might reach by measuring its environmental factors.

Clustering

Clustering problems involve grouping similar data points together based on their intrinsic properties. Unlike classification, clustering is an unsupervised learning task, as it doesn't rely on predefined class labels. Clustering algorithms discover patterns and structures in the data by identifying data points that share common characteristics. Applications of clustering include customer segmentation for targeted marketing, identifying natural groupings in biological data, or image segmentation for object detection.

After presenting the main problems to solve with traditional Machine Learning techniques we will introduce the main algorithms used for solving these problems, centering the explanations in classification problems as the one to solve in our Final Project is a multi-class classification problem.

2.3.2 Classification Models

In this section we will present the algorithms for classification problems used in this project.

Support Vector Machines (SVM)

The Support Vector Machine (SVM), sometimes called Support Vector Classification (SVC) for Classification SVMs, originated in the 1960s and was further developed in the 1990s, finally

being presented in 1995 by Vladimir Vapnik and Alexey Chervonenkis [35]. Its primary objective is to find a hyperplane that optimally separates different data points of each class in high-dimensional spaces.

The goal of the SVM is to create a kind of separation between two classes by maximizing the margin's distance between each class so that if data points fall on different sides of the boundary, they can be classified. This algorithm is commonly used for binary classification problems and linearly separable cases, where the sought boundary is represented by the hyperplane function: $w^T x + b = 0$, where the weight vector w determines the orientation of the classification boundary by assigning significance to features in the input vector x, the bias term b permits the boundary to shift, accommodating data not centered at the origin, and the transpose w^T is employed to gauge the similarity between the weight vector and the feature vector x.

However, in reality, most problems are not linearly separable due to noise introduction or high complexity in data. To address such cases, two approaches have been devised: using a soft margin or employing the kernel trick.

With the solution of the soft margin, a parameter C is introduced, which represents the trade-off between allowing misclassification and minimizing the margin width. This allows some values to be classified within a margin they shouldn't be in, to avoid overfitting and excessively lengthy training.

The kernel trick is used for non-linear SVM problems. The kernel used by the model can be changed to measure the similarity between different data points and to find the best boundary with the maximum margin. The kernel trick involves using kernels that allow mapping our data into higher-dimensional spaces, where the problem might be linearly separable. Importantly, the kernel doesn't need to calculate the projection in this higher-dimensional space; it only needs to compute the dot product in the new space, which significantly reduces computational load.

The most commonly used kernels nowadays are:

- Linear Kernel: Represents the original feature space and performs linear classification. It calculates the dot product of the input data points.
- Polynomial Kernel: Maps the data into a higher-dimensional space using polynomial functions, enabling SVM to capture more complex relationships between data points.
- Radial Basis Function (RBF) Kernel: Measures similarity between data points based on a Gaussian distribution.

We can see in figure 2.1 a representation of a SVM solving a binary classification problem and in figure 2.2 an example of the different kernels and how it affects on the solution hyperplane is displayed

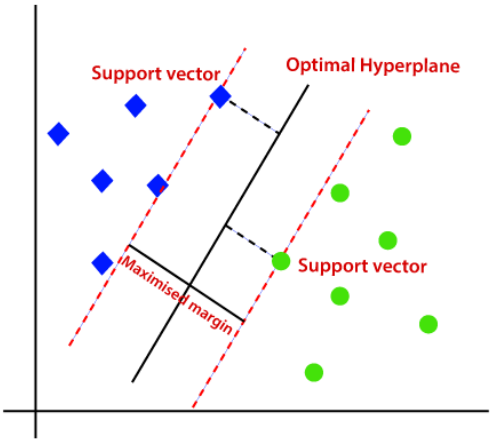


Figure 2.1: Suport Vector Machine functioning from [1]

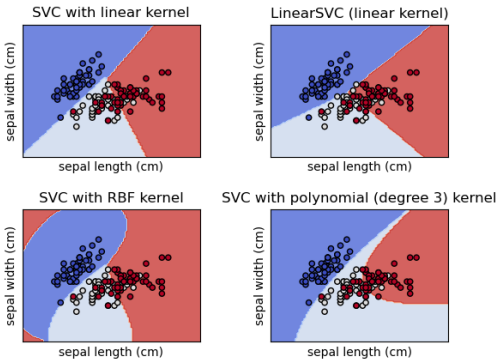


Figure 2.2: Comparison of different SVM kernels from [2]

Random Forest

The Random Forest (RF) algorithm, introduced by Leo Breiman and Adele Cutler in the year 2001 [36], is a versatile ensemble (2.3.3) learning method that combines multiple decision trees to achieve improved performance and robustness.

Random Forest's main objective is to address overfitting and enhance generalization by aggregating predictions from individual decision trees. It constructs an ensemble of decision trees during training, where each tree is trained on a random subset of the training data, introducing randomness into the process.

The algorithm makes predictions by averaging or voting on the predictions of individual trees within the forest. This aggregation reduces the impact of individual decision trees' errors, resulting in a more accurate and stable prediction.

Random Forest has several advantages, including handling non-linear relationships, feature importance estimation, and resistance to overfitting. It is suitable for both classification and regression tasks.

The key steps in Random Forest include:

1. Bootstrapping: Creating multiple subsets of the training data using a process called bootstrapping, allowing each decision tree to be trained on a different subset.

2. Random Feature Selection: Randomly selecting a subset of features at each node of a decision tree, reducing correlation among trees.

3. Growing Trees: Growing decision trees using the selected features and bootstrapped subsets, resulting in a diverse set of trees.

4. Aggregating Predictions: Combining the predictions of individual trees through averaging (regression) or voting (classification).

In figure 2.3 we can observe a simple representation of how the Random Forest model works

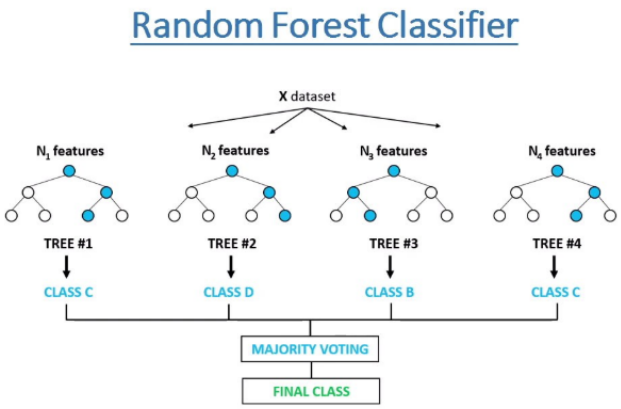


Figure 2.3: Functioning of the Random Forest from [3]

Gradient Boosting

Gradient Boosting, introduced by Jerome H. Friedman in 2001 [37], is an advanced ensemble (2.3.3) learning technique that aims to create a strong predictive model by combining the predictions of multiple weak models, typically decision trees.

The primary objective of Gradient Boosting is to improve the predictive accuracy of a model through an iterative process. It builds the model sequentially, where each new model corrects the errors made by the previous ones.

Here's how Gradient Boosting works:

1. Initialization: The first model is trained on the original data.
2. Residual Calculation: The difference between the actual target values and the predictions of the current model is calculated, creating a residual.
3. Training Weak Models: New weak models (typically decision trees) are trained to predict the calculated residuals.
4. Combining Predictions: The predictions of the new models are added to the previous model's predictions, updating the previous model.
5. Iteration: Steps 2 to 4 are repeated for a predefined number of iterations.

The process of Gradient Boosting is guided by optimization techniques like gradient descent, where the algorithm tries to minimize the loss function by adjusting the model's parameters.

There are several implementations of Gradient Boosting, with eXtreme Gradient Boosting (XGBoost) [38] being one of the most popular. XGBoost introduces enhancements to tradi-

tional Gradient Boosting by incorporating techniques such as regularization, parallel computing, and handling missing values. This leads to improved performance, efficiency, and generalization.

Since both Random Forest and Gradient Boosting are ensemble learners we expose their differences and similarities in the table 2.3:

Aspect	**Gradient Boosting**	**Random Forest**
Construction	Sequentially builds models by correcting errors of previous models	Constructs multiple decision trees independently in parallel
Weighting Errors	Assigns weights to data points based on errors made by previous models	Trains each tree on a random subset of data with replacement
Bias-Variance Tradeoff	Tends to have lower bias but more prone to overfitting	Typically has a slightly higher bias but less prone to overfitting
Interpretability	May be harder to interpret due to sequential construction	Easier to interpret as each tree is independent

Table 2.3: Comparison of Gradient Boosting and Random Forest

Logistic Regression

Logistic Regression (LR), introduced as a statistical model by Joseph Berkson in 1944[39], is a fundamental Machine Learning algorithm widely used for binary classification tasks. Unlike its name suggests, it's primarily used for classification rather than regression. It's a simple yet effective algorithm that can model the probability of a binary outcome based on input features.

The core idea behind Logistic Regression is to apply a logistic function (also known as the sigmoid function) to the linear combination of input features and corresponding weights. This function maps any input into a value between 0 and 1, which can be interpreted as the probability of the input belonging to the positive class.

The logistic function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where z is the linear combination of weights and input features.

In the training phase, the algorithm adjusts the weights to minimize the difference between predicted probabilities and the actual binary labels. This optimization process is typically done using techniques like gradient descent.

The key components of Logistic Regression are:

- **Linear Combination**: Logistic Regression calculates a linear combination of input

features and corresponding weights.

- **Sigmoid Function**: The logistic function maps the linear combination to a probability value between 0 and 1.
- **Thresholding**: A threshold value (usually 0.5) is applied to the predicted probabilities to classify instances into positive or negative classes.
- **Cost Function**: Logistic Regression uses the cross-entropy loss as the cost function to measure the difference between predicted and actual labels.

However, Logistic Regression has its limitations. It may struggle with handling highly imbalanced datasets or dealing with non-linear relationships. In such cases, more advanced algorithms like Support Vector Machines or ensemble methods like Random Forests and Gradient Boosting might be more suitable.

But it can be possible to use different training strategies to improve its ability of generalizing and making it able to work for multi-class problems, like One vs Rest or One vs One.

Multilayer Perceptrons

The Multilayer Perceptron (MLP) is a type of neural network that evolved from the original Perceptron, introduced in 1958 by Frank Rosenblatt [40]. It is a series of interconected nodes, also called neurons, that are organized into layers, including an input layer, one or more hidden layers and an output layer. These connections are weighted to determine their strength and are adjusted during training to minimize the difference between the predicted and real outputs.

The improvement of the MLP, from the original Perceptron is the inclusion of multiple hidden layers, that allows our model to pass through multiple steps of feature extraction of the input data and permits for solving non linear problems and capture more complex data patterns.

In the figure 2.4, an example of a simple MLP is shown.

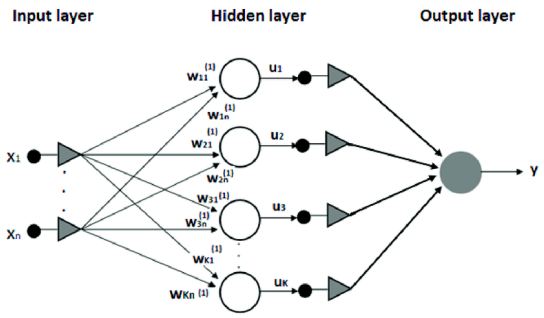


Figure 2.4: Example of Multilayer Perceptron from [4]

K-Nearest Neighbors (kNN) Algorithm

The K-Nearest Neighbors (kNN) algorithm is a versatile and intuitive classification technique that operates on the principle of similarity. Introduced in 1951 by Evelyn Fix and Joseph Hodges in 1951 [41] . It works by classifying a data point by examining its proximity to its neighboring data points. It assumes that data points with similar features tend to belong to the same class. In essence, kNN defines classes based on the majority class of its k nearest neighbors, where k is a user-defined parameter.

kNN is particularly useful when dealing with non-parametric and non-linear data distributions. Unlike some other algorithms, kNN doesn't make any underlying assumptions about the data's distribution or structure. This flexibility allows it to handle complex data scenarios, such as clusters, outliers, and noisy data.

The algorithm's steps are:

1. Calculate Distances: For a given data point, calculate the distance to all other data points in the dataset. The choice of distance metric, such as Euclidean or Manhattan distance, depends on the nature of the data.

2. Select Neighbors: Identify the k nearest data points based on their calculated distances. These data points form the "neighborhood" of the target data point.

3. Classify: Determine the class of the target data point by tallying the classes of its k neighbors. The class that appears most frequently among the neighbors is assigned to the target data point.

We can observe in figure 2.5 the functioning of the k Nearest Neighbours algorithm for a binary classification problem.

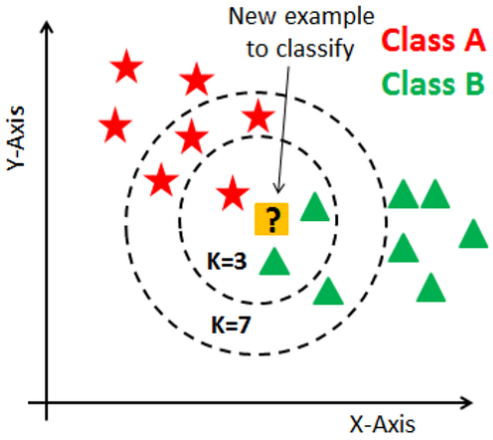


Figure 2.5: kNN binary classification example from [5]

An important consideration in kNN is the selection of the parameter k. A small k value can make the algorithm sensitive to noise and outliers, potentially leading to overfitting. On the other hand, a large k value can result in a loss of local information, making the algorithm less sensitive to the underlying patterns in the data.

While kNN is relatively simple, it may suffer from computational inefficiency when applied to large datasets. Calculating distances between data points can be time-consuming. But nowadays various techniques like KD-trees[42] can be employed to speed up the nearest neighbor search.

2.3.3 Ensemble Methods

Ensemble methods involve combining multiple individual models to create a stronger and more robust predictive model. These methods often outperform single models by leveraging the diversity and collective knowledge of the individual models.

There are multiple different methods for ensemble models but we will explain next the principal ones that are more used nowadays, apart from Random Forest and Gradient Boosting because we have already explained them in points 2.3.2 and 2.3.2.

Bagging (Bootstrap Aggregating)

Bagging, published originally in 1996 by Leo Breiman[43], is a fundamental ensemble technique that involves creating multiple subsets (bags) of the training dataset through random sampling with replacement.

Each subset is used to train a separate base model, typically a decision tree. During prediction, the outputs of the individual models are combined, often through majority voting

for classification tasks. Bagging helps reduce overfitting and variance by averaging out the individual model's errors.

Boosting

Boosting is another ensemble technique that focuses on sequentially improving the performance of weak base models by combining them. assigning higher weights to misclassified instances, making subsequent models focus on correcting those mistakes. The final prediction is a weighted combination of the predictions from all base models. One of the most known implementations of Boosting is AdaBoost (Adaptive Boosting) [44].

Boosting can lead to significant improvements in accuracy, especially when combined with simple models like decision stumps.

In the figure 2.6, a graphic representation of the difference of Boosting and Bagging Ensemble Methods is presented.

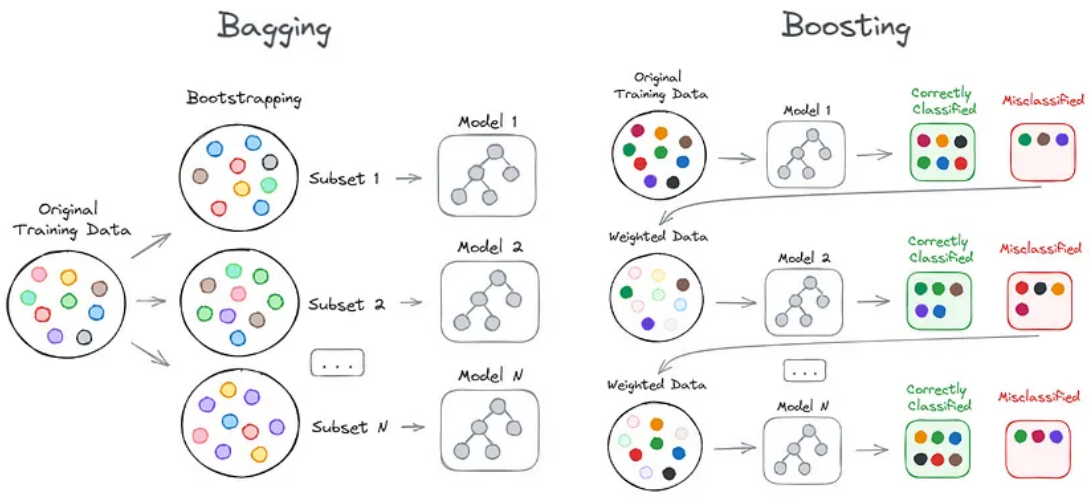


Figure 2.6: Comparison of Bagging and Boosting from [6]

Voting

Voting ensembles, introduced in 1992 by Robert M. French[45], involve combining the predictions of multiple individual models to make a final prediction, taking the opinion of each of the models to make the decision based on the collective insight. There are three main types of voting:

- **Hard Voting:** In hard voting, the final prediction is determined by majority rule. Each individual model's prediction contributes to the final decision, and the class with the most votes is selected.

It works better with multiple models that each differ in their area of expertise and have different errors, correcting misclassifications in the process.

- **Soft Voting:** In soft voting, the final prediction is based on the class probabilities predicted by the individual models. The class with the highest average probability across all models is chosen.
- **Weighted Voting:** Weighted voting assigns different weights to the predictions of individual models. This allows more influential models to have a larger say in the final decision.

  It introduces importance to the models, so for a model that has better known performance on the problem tried to solve, it can be given more importance to its prediction.

Stacking

Stacking, first proposed in 1992 by David H. Wolpert [46], is an ensemble method that entails training multiple base models and using their predictions as inputs to a higher-level model, often referred to as a meta-model. This meta-model learns how to combine the predictions from the base models for the final prediction. This ensemble method attempts to join the stenghts of the different base models to improve the final prediction.

2.3.4 Outlier Detection

As exposed previously, one of the main methods for IDSs to detect and classify their input data into possible threats is the Anomaly-based Detection (2.1.2), that attempts to establish what is considered common behaviour and detect the ones that deviate from that. And that is also the principle of this task to be solved with Machine Learning.

As there are multiple methods for Outlier Detection, we will center our explanation in three of them: Isolation Forest, Local Outlier Factor and One Class SVM.

Isolation Forest

It was introduced by Lie et al. in 2008 [47] and follows the principle that anomalies should normally be isolated data so with a set of binary trees that divide the data space we should require fewer partitioning steps to identify them than for normal data.

We randomly select a feature and divide the data space within the range of the feature value. This way we can measure the quantity of anomaly of the data by its difference of partitioning steps compared to the average of the inliers.

An example of functioning of the Isolation Forest can be seen in figure 2.7

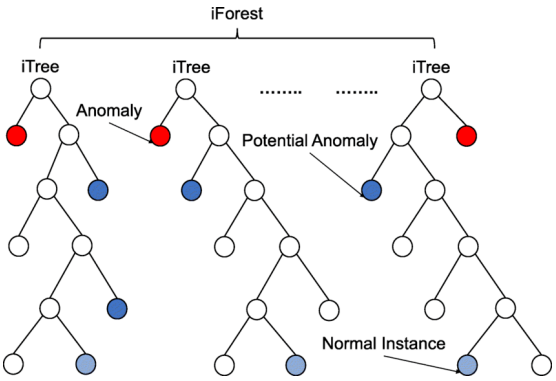


Figure 2.7: Example of Isolation Forest functioning from [7]

Local Outlier Factor

The Local Outlier Factor (LOF), introduced by Breunig et al. in 2000[48], presents a distinct perspective on anomaly detection that leverages the local density variations of data points.

LOF evaluates the deviation of the density of a data point from the densities of its neighbors, as outliers are often surrounded by regions of lower density compared to the broader data distribution. By comparing this local density with the densities of its neighbors, LOF quantifies the degree to which a point is an outlier. If the local density is notably lower compared to the neighbours', it is classified as anomaly.

One Class SVM

The One-Class Support Vector Machine (One-Class SVM), introduced by Schölkopf et al. in 2001 [49], is a variant of the traditional Support Vector Machine built for outlier and novelty detection.

In this method, the primary objective is to create a decision boundary that encapsulates the majority of the data points, defining a region that is considered "normal". Instances lying outside this boundary are classified as outliers. One-Class SVM operates with the notion that the majority of data points are expected to lie within the boundary, making the outliers' representation significantly smaller. To achieve this, the One-Class SVM seeks to find a hyperplane that best encapsulates the data, maximizing the margin while accounting for the prescribed tolerance of outliers. The resulting hyperplane and margin effectively establish the boundary within which most data points should reside.

Chapter 3

State of the Art in the Field

Machine learning for Network Intrusion Detection Systems is fundamental to be able to cope with the development of new attacks as well as to rid ourselves of unwanted exposure to bad-intention behaviour. But to train our Machine Learning models we first need to be able to establish a standard methodology to follow. Not only the methodology is relevant, but also the data used to train our system. Thus, in this chapter we will discuss the state of the art in both aspects, starting with the evolution of Datasets used for IDS Machine Learning models and their history.

3.1 Datasets for IDS

The evolution of Intrusion Detection System (IDS) datasets has been driven by the need to address various challenges posed by real-world network traffic and attacks. In the following, we will describe briefly the main datasets used in the field. Some of the information has been extracted from the dataset comparison present in [9] and [50].

3.1.1 DARPA datasets

The initial DARPA datasets from 1998 and 1999 [51] marked the inception of intrusion detection data, featuring synthetic network traffic collected over an extended period of several weeks. However, these datasets were criticized for having unrealistic traffic patterns, outdated attacks, and inaccurate labels.

3.1.2 KDDCup'99

After the DARPA'98 datasets, the KDDCup'99 dataset [52] was developed as a derivative, aiming to incorporate features extracted from TCP connections, traffic behaviors, and content attributes, deemed more useful to machine training. Despite becoming a widely utilized benchmark for Intrusion Detection System (IDS) evaluation, the KDDCup'99 dataset inherits

several limitations present in its predecessor, the DARPA'98 dataset, and even introduces new challenges [50]. The main problems this Dataset has are:

- **Subset of DARPA'98:** The KDDCup'99 dataset is a subset of the DARPA'98 dataset and therefore inherits its limitations in terms of traffic volume and diversity. The constrained scope of the dataset limits its representation of network traffic and attack scenarios, potentially affecting the generalizability of IDS models trained on it.
- **High Proportion of Attack Traffic:** With a skewed distribution, the KDDCup'99 dataset contains a significantly higher proportion of attack traffic (80.5%) in comparison to normal traffic (19.5%). This imbalance does not accurately reflect real-world traffic distributions, leading to biased training and evaluation results.
- **Imbalanced Attack Type Distribution:** The dataset also presents skewed distributions among attack types. Certain attacks, like Neptune, are overrepresented, while others, like U2R, are underrepresented. This imbalance can distort the effectiveness of IDS models in recognizing less frequent attack behaviors.
- **Unnecessary Features:** In the Dataset, there are included irrelevant, redundant and correlated features that bring difficulty in model interpretation and prediction, as they can potentially lead to increased complexity and reduced performance of Machine Learning algorithms.
- **Missing Values** Some features within the dataset contain missing values, introducing uncertainty and compromising the integrity of data analysis. Missing values can distort the training process and affect the reliability of IDS evaluations.

  As a result of these limitations, the KDDCup'99 dataset falls short in evaluating modern IDS methods, as it fails to accurately capture the intricacies and dynamics of contemporary network traffic and attacks. Consequently, there is an imperative for newer and more realistic datasets capable of serving as reliable benchmarks for the evaluation of IDS techniques. These datasets should reflect the evolving landscape of network threats and provide a foundation for training, testing, and refining IDS models under more accurate and challenging conditions.

3.1.3 Improvements Proposed for KDDCup'99

To address the limitations of previous datasets, several new datasets were developed, each aiming to enhance the representation of network traffic and attacks. These datasets, including gureKDDCup, ISCX 2012, and NSL-KDD, attempted to provide more realistic, diverse, and up-to-date network environments. They introduced new features, metadata, and attack details

to better capture the nuances of contemporary network threats. However, these datasets faced challenges such as incomplete network configurations, insufficient traffic diversity, and limited internal-external network interactions. Notable improvements and specific features of these datasets are outlined below.

gureKDDCup

The gureKDDCup dataset [53] aimed to overcome the limitations of KDDCup'99 by incorporating the missing connections from DARPA'98. This expansion resulted in a more comprehensive dataset with detailed attack descriptions and categories. While addressing some of KDDCup'99's issues, such as class imbalance and outdated attacks, gureKDDCup retained certain challenges like irrelevant features.

ISCX 2012

The ISCX 2012 dataset [54] simulated diverse network scenarios, including normal activities, brute force attacks, HTTP DoS attacks, and more, to create a realistic network environment. It offered raw network traffic in pcap format and extracted features with corresponding labels in xml format. Despite its merits, ISCX 2012 had limitations in terms of traffic volume, diversity, and label consistency.

NSL-KDD

The NSL-KDD dataset [55] utilized a selection method to rectify redundancy in the KDDCup'99 dataset and balance class distributions. It provided two distinct subsets: KDDTrain+ and KDDTest+, intended for training and testing respectively. Although addressing issues like redundant records, NSL-KDD shared some drawbacks with KDDCup'99, including irrelevant features and outdated attacks.

3.1.4 More recent proposals

Apart from the previous examples that were all based on legacy datasets, some newer ones have been developed in the past years to try to solve the problems that were criticized, such as the use of very old types of attacks and mostly or only synthetic data. Some main examples are:

- **MAWI Working Lab Dataset [56]:** This dataset contains data from 2001 to 2007, with only the last year being labeled. The tcpdump trace represents just 15 minutes per day. It provides real network traffic from a trans-Pacific backbone link, covering various types of applications and protocols. It also includes some attacks, such as DDoS,

port scanning, and worm propagation. However, it has some limitations, including low traffic volume, lack of diversity in attack types, and incomplete labels.

- **CTU-13 [57]:** This dataset features 13 different types of malware labeled in real network traffic from short periods between 2013 and 2016, primarily used for botnet behavior detection. It offers raw network traffic in pcap format, along with extracted features and labels in csv format. Detailed descriptions of malware samples and their characteristics are also provided. However, it has limitations such as low traffic volume, lack of diversity in normal traffic and attack types, and limited interaction between internal and external networks.

- **UNSW-NB15 [58]:** This dataset employs an attack generation tool called PerfectStorm to generate synthetic data spanning 31 hours. It provides network traffic with nine different attack categories, such as DoS, Fuzzers, Exploits, etc. Each network flow is accompanied by 49 features and labels. However, it has limitations including unrealistic traffic patterns, inaccurate labels, and irrelevant features.

- **UGR'16 [10] [8]:** This dataset comprises 4 months of anonymized network traffic flows from a Spanish ISP, divided into Calibration and Test parts. Test contains synthetically generated and real attacks, making it suitable for supervised classification using Machine Learning techniques. The dataset provides real network traffic with high volume and diversity, covering various types of applications and protocols. It also includes some attacks, such as DDoS, port scanning, brute force, etc. However, it has limitations like incomplete network configuration and lack of attack details and labels, as some attacks have been grouped into bigger categories and the Calibration files had to be passed through already existing models to classify the background data like One Class SVM and $MSNM_c$ (Multivariate Statistical Network Monitoring for anomaly detection).

These new datasets represent significant progress in addressing the limitations of previous datasets and provide more realistic and diverse environments for evaluating Intrusion Detection System (IDS) methods.

For this Final Project we will center our implementations around the use of the last dataset, UGR'16. We will use it to train our different models used in applying the necessary transformation to the netflow data.

The UGR'16 dataset offers a distinct advantage over its predecessors by being useful to the evaluation of Network Intrusion Detection Systems (NIDS) that emphasize long-term evolution and traffic periodicity. This dataset enables the training and evaluation of models that account for variations in daytime and nighttime, as well as differences between labor week-

days and weekends. Such considerations significantly enhance the dataset's utility in assessing NIDS performance across diverse temporal patterns.

To better reflect the properties of each dataset exposed we show in table 3.1 a reduced version of the table present in [8]. As in the original table, we use 'yes' to say that a requirement is met, '(yes)' if is met with some limitations and 'no' otherwise. We also expose in a summarized way the advantages and disadvantages over choosing each of the newest proposals exposed previously in the table 3.2.

As we end up choosing to use the UGR'16, in the next section we will present it in a more extended manner.

Dataset	Reference	Real back traffic	Updated attack traffic	Duration	Format
DARPA'98/'99	[51]	no	no	yes	pcap
KDDCup99	[52]	no	no	yes	csv
gureKDDCup'08	[53]	no	no	yes	csv
UNB ISCX'2012	[54]	no	yes	no	csv
NSL-KDD'09	[55]	no	no	yes	csv
MAWILab'2010	[56]	yes	yes	(yes)	pcap
CTU-13	[57]	(yes)	yes	(yes)	pcap & flow
UNSW-NB15	[58]	no	yes	no	csv
UGR'16	[10]	yes	yes	yes	flow

Table 3.1: Comparison of Selected Datasets for IDS Evaluation from [8]

Dataset	Advantages	Disadvantages
MAWI Working Lab	• Contains real network traffic data. • Covers various applications and protocols. • Includes some attacks.	• Low traffic volume. • Lack of diversity in attack types. • Incomplete labels.
CTU-13	• Features various types of labeled malware traffic. • Offers detailed descriptions of malware samples.	• Low traffic volume. • Lack of diversity in normal traffic and attack types. • Limited interaction between internal and external networks.
UNSW-NB15	• Provides data with different attack categories. • Includes features and labels.	• Unrealistic traffic patterns. • Inaccurate labels. • Irrelevant features.
UGR'16	• Contains real network traffic data from an ISP. • Covers various applications and protocols. • Includes some attacks.	• Lack of attack details and labels. • Limited interaction between internal and external networks.

Table 3.2: Advantages and Disadvantages of Intrusion Detection Datasets

3.2 The UGR'16 Dataset

As mentioned earlier in 1.2.1, the UGR-16 dataset, published in 2017 by the University of Granada, is used for Intrusion Detection Systems based on cyclostationary techniques. It comprises network traffic data from a Tier 3 ISP in Spain, with the aim of capturing heterogeneous traffic thanks to having multiple real services like email or WordPress hosted on the ISP network.

The network's architecture is sufficiently complex to mimic the operation of a typical network, featuring five attacking machines in their own subnet, five victim machines within the

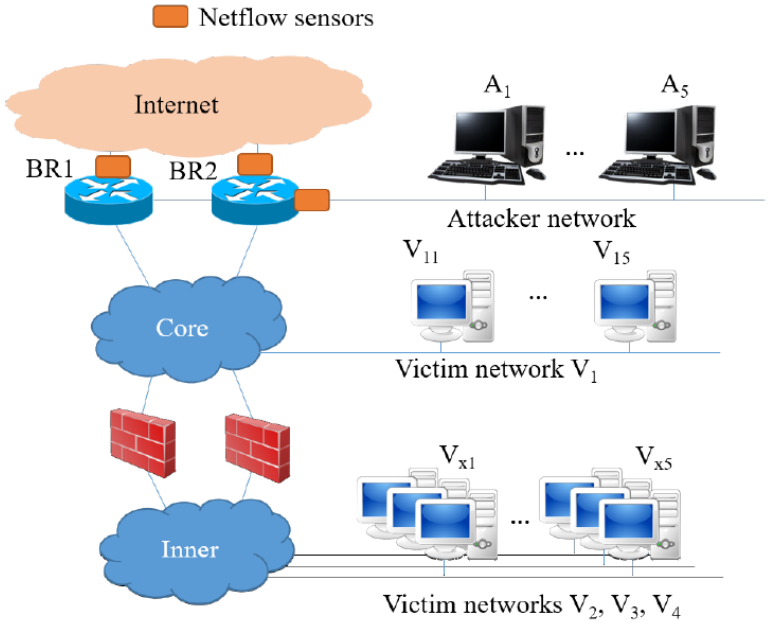


Figure 3.1: Diagram of the ISP network from [8]

network but outside the firewall, and an additional fifteen machines distributed across three subnets after the firewall. Both attackers and victims are located within the ISP's infrastructure to avoid potential detection and blocking of attacks by intermediate ISPs. However, the attackers' network is positioned at the border router to simulate attack traffic as if it were coming from the Internet. The explained structure can be better seen in figure 3.1.

To create the UGR'16 Dataset, it was divided into two distinct captures: a calibration capture spanning from 10:47 on March 18, 2016, to 18:27 on June 26, 2016, and a test capture from 13:38 on July 27, 2016, to 09:27 on August 29, 2016. The calibration capture exclusively contains real network traffic, while the test set introduces synthetically generated attacks in combination with real background traffic.

3.2.1 Calibration Set

The calibration set spans 100 days and Its primary purpose is to assist in building and calibrating normality models. It is important to note that while no attacks were synthetically generated during this period, it does not necessarily mean that no attacks are present, so, as the traffic does not include generated attacks, the authors needed to pass the obtained traffic through multiple filters to be able to establish with some certainty what is normal traffic and what could have be an outsider attack. This was done not only for the calibration set but also for the included background traffic in the test set.

To detect and classify the attacks they used multiple methods explained before in section 2.1.2, like signature-based, including the patterns of typical attacks and the known IPs from blacklists hpHosts, Malware DL, spamhaus and abuse.ch, and anomaly-based detection with models like MSNMC, MSNMS, and One Class SVM explained in 2.3.4.

With this methods three main attacks were identified: UDP scan, SSH scan and SPAM attacks.

3.2.2 Test Set

The test set last little more than a month and includes, as said previously, a traffic produced by combining the background traffic of the network with generated attacks by state-of-the-art hacking tools,generated in batches at different times and different days to attempt to be distributed over the set, and its porpouse is to be able to test IDS detection capabilities.

The generated attacks are grouped into three main categories: Low-rate Dos Attacks, Port Scanning and Botnet Traffic. For the Dos attacks the hping3 tool was used to send TCP SYN packets to port 80 of multiple victims and three different scenarios where genereted, one attacker to one victim, five attackers to three victims at the same time and the same but each with a difference of 30 seconds. For the Port Scanning they generated a SYN scanning with nmap with a one to one scan and a four to four. For the botnet the malware Neris, a type of computer worm that infects hosts and sends spam to spread and is also able to execute malicious files, traffic was inserted into the dataset converted from pcap files to nfcapd to be able to transform it to netflow v9 data that will be used for the UGR'16 dataset. They did not infect the computers of the nework as it is not a private network and the authors considered ethical complications as they cannot control its spread. For the generated traffic they use the available CTU-Malware-Capture-Botnet-42 [57]. As the dataset is proposed for use in a ciclo stationary IDS the generated traffic is inserted in a period of 12 consecutive days over all possible hours, with planned and random attacks.

3.2.3 Format of the Dataset

The format used for the capture of the flows of data is Netflow v9 from CISCO and it is stored in nfcapd binary files, that include all Netflow v9 details. It is later prepocessed with an anonymization process, where the IP addresses of the machines within the dataset undergo anonymization using the CryptoPan prefix-preserving anonymization with the tool nfanon, and after transformed into CSV format. This transformation attempts to simplify the data while preserving the most common features for Machine Learning, timestamps, flow duration, source and destination IP addresses, source and destination ports, protocol, flags, forwarding status, type of service, packets exchanged within the flow, and their corresponding byte counts.

3.2.4 Composition of the Dataset

As exposed by the authors, the total number of flows is more than 16900 Million, with a duration of 4 months and 600 Million different external IPs from 10 Million different networks.

As exposed in [8], the analysis of network traffic patterns in the dataset reveals several notable findings: the HTTP/S protocol dominates, accompanied by consistent DNS traffic, the authors express surprise as, despite security concerns, many companies still use Telnet for equipment management, and there are peaks in SMTP traffic occur sporadically, representing legitimate email campaigns and spam, as with SSH traffic that corresponds with SSH scan attacks.

3.3 Evolution of Machine Learning for IDS

The incorporation of Machine Learning algorithms in Intrusion Detection Systems (IDS) has marked a significant advancement in enhancing the security of computer networks. This section explores the historical evolution of Machine Learning's integration into IDS [30], tracing its progression from early attempts to the sophisticated models used today[59][60].

3.3.1 Early Attempts and Rule-Based Systems (late 1980s - early 1990s)

In the 1990s, the field of intrusion detection saw its early stages with the emergence of rule-based systems. These systems relied on predefined signatures and patterns to identify known attacks. While effective against specific threats, these rule-based approaches struggled to adapt to evolving attack techniques, as they necessitated manual updates for each new threat.

3.3.2 Emergence of Anomaly Detection (1990s - early 2000s)

As the nature of attacks evolved, so did the approach to intrusion detection. Anomaly-based detection methods gained prominence in the 1990s. These methods aimed to identify deviations from normal system behavior, making them more adaptable to new and previously unseen attacks. However, they often suffered from a high rate of false positives due to their reliance on complex statistical models that had difficulty capturing the intricacies of legitimate variations in network behavior.

3.3.3 Machine Learning Integration (2000s - early 2010s)

The 2000s witnessed a paradigm shift with the integration of Machine Learning techniques into IDS. This integration aimed to overcome the limitations of rule-based and anomaly-based systems. Supervised Machine Learning algorithms, such as Support Vector Machines (SVM) and Random Forests, were employed to classify network traffic as normal or malicious. These

models relied on labeled datasets for training and demonstrated improved accuracy and adaptability.

3.3.4 Evolution of Deep Learning (2010s - Present)

In recent years, the rise of deep learning has reshaped the landscape of IDS. Deep neural networks, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have shown remarkable capabilities in feature extraction and pattern recognition. Their hierarchical architecture enables them to capture complex relationships in network data, enhancing the accuracy of intrusion detection. Additionally, the advent of transfer learning has facilitated the use of pre-trained models, even with limited labeled data.

3.4 Standard Methodology for NIDS development

It is imperative to develop a standard methodology to test the different proposals of Machine Learning and Deep Learning models for Network Intrusion Detection Systems (NIDS). There have been various proposals like the one presented in [61] or [9].

Both methodologies share common ground in their use of Machine Learning techniques to develop and evaluate Machine Learning-based Network Intrusion Detection Systems (ML-NIDS). They both emphasize key aspects like data quality, feature engineering, model selection, parameter tuning, validation techniques, and performance metrics. Moreover, both methodologies strive to create structured approaches for systematically assessing and comparing ML-NIDS, employing established datasets for their experiments.

However, they differ in their primary goals and methods. The first methodology focuses on cross-evaluations, using multiple datasets to test the same ML-NIDS. This uncovers new qualities like detection capabilities and robustness. The second methodology is centered on single-dataset evaluations, providing guidelines for standardized assessments within one dataset. The former aims to discover hidden attributes, while the latter seeks to enhance reliability in comparisons and evaluations.

For the development of our Project we chose the second methodology, the one proposed in [9], as it is has been found more suitable for our project and it is the methodology used by previous works in the evaluation of the dataset UGR'16 [10].

In the next chapter, we will present the followed steps by the chosen methodology and how it has been used for our project.

Chapter 4

Development of the IDS

Artificial Intelligent systems and Machine Learning are nowadays one of the most used methods for detection in IDSs, but, as authors like [9] and [61] expose, there needs to be an standard methodology to be able to compare different Machine Learning approaches over a Dataset. For it, in this chapter we will introduce the Free Framework for Machine Learning Methodology presented in [9], we will attempt to solve a problem present in the methodology explained in their publication and finally we will replicate their results as a baseline for later attempts for improvement.

4.1 Free Framework for Machine Learning (FF4ML)

This methodology has been implemented in the open project Free Framework for Machine Learning (FF4ML) [62]. It follows a series of 4 differentiated steps to extract the important information from the UGR'16 dataset, transform it to be able to train consistently Machine Learning models, and obtain the final results with the proper format to be able to compare these models' performance with each other and previous state of the art models. The steps are shown in figure 4.1 and explained in continuation:

1. **Feature Engineering**

   As most of the current datasets for NIDS come in various formats, mostly containing

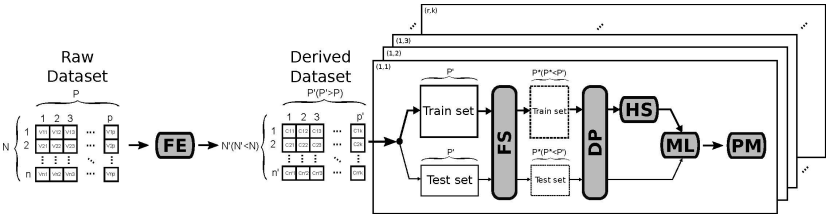


Figure 4.1: Diagram of the Methodology from [9]

very large files of binary data, we need to transform this into a suitable input for our models. For this, the authors in [9] propose the Feature as a Counter technique (FaaC). It transforms our huge number of rows of data into rows that have the information grouped within a given span of time.Then it adds each of the different features, as the name exposes, as a counter. It sums all of the occurrences of the predefined features in said span of time. This alone manages to reduce greatly the size of our data, going from approximately 4.000 million rows in the total UGR'16 dataset to 47000, but increasing the complexity of our problem from 11 features to a total of 132 in the reduced dataset.

To confront this increase of complexity in our dataset the authors propose the next step: Feature Selection.

2. **Feature Selection**

   In this step we aim to reduce the number of features that our dataset will contain, removing the least important ones that carry less information or those that are duplicated. The Least Absolute Shrinkage and Selection Operator (LASSO) model is employed, training 5 LASSO models, because they try to solve a 5-label classification problem. This models work in parallel with an added penalty, and only the variables that are proven necessary to predict all K labels remained in the dataset.

   After this step, the total number of features is reduced from 132 to 87.

3. **Data Preprocessing**

   Effective Machine Learning models necessitate normalized input features to account for varying variable distributions. A standard normalization process is applied to numeric input variables, ensuring consistent input for ML models.

4. **Hyper-parameters Selection**

   Any Machine Learning model that is used will not perform in an optimal way without a proper tuning of their inner parameters for the given problem. In this methodology it is proposed the use of Bayesian optimization[18] for this task, as it has been proven that it can get a pseudo-optimal solution for the hyper-parameter selection with less iteration than the given alternatives, Grid Search and Random Search.

4.1.1 Previous results from FF4ML methodology with UGR'16

After presenting the methodology proposed in the reference paper [9], we will briefly expose the models used, and a summary of the results obtained so as to be able to compare it later with our proposals.

To follow the necessary steps of their methodology they chose 3 distinct models, the Multinomial Logistic Regression model, the Support Vector Machine, choosing for the kernel both the Linear Function and the Radial Basis Function, and the Random Forest. The study [9] evaluates the performance of these Machine Learning (ML) models, trained and tested with the UGR'16 Dataset, in Intrusion Detection Systems (IDS).

Statistical tests, including pairwise Wilcoxon signed-rank comparisons and Holm post-hoc correction, were used to establish significance with a 95% confidence level.

The findings include:

- Significant differences in accuracy exist among the models for most cases, except in certain attack classes and accuracy metrics.
- Random Forest and Suport Vector Machine with Radial Basis Function kernel models exhibit strong performance, particularly for synthetically generated attacks (DoS, Botnet, Scan).
- Real attacks (Spam) and real network traffic (Background) prove more challenging to classify due to complex patterns.
- Random Forest demonstrates robustness for unbalanced classifications, outperforming other models in weighted averages of performance metrics.
- Botnet attacks, designed to hide network traffic, are effectively detected by the ML-based NIDS.
- Background class results are less accurate, possibly due to a reduction in observations.

4.2 Analysis of the UGR'16 Dataset

This exploratory phase is indispensable to get used to the data that is going to be used, its format and the information it includes. This is to be then able to properly propose improvements around the problem and avoid the design of unrealistic approaches that would be later seen impossible for the nature of our data.

As said in the previous paper 'authors envision that it (the general low accuracy in the class Background over the evaluated models) might be due to the reduction of the number of observations concerning the original ones as a consequence of the FaaC approach.' [9] For this reason, and the importance of working with the raw Dataset before using Feature Engineering, we decide to first test this hypothesis before the replication of the results of [9].

4.2.1 Data Collection and Cleaning

The UGR'16 dataset, as it turns out, surpasses initial expectations in terms of its size. Each week's dataset amounts to approximately 50 GB once uncompressed and with a total of 23 weeks in the whole Dataset the summed size amounts to approximately 1.8 TB.

This substantial size poses challenges for traditional analytical tools. For instance, software like Excel lacks the memory capacity to load such a voluminous dataset into a computer with only 16 GB of available RAM.

For this reason, it was decided to use CITIC's supercomputer, a high-performance machine offering sufficient storage space to hold the complete dataset, which allowed for the following transformation and analysis. Access was provided via SSH.

To facilitate initial analysis, a decision was made to download a single week's worth of test data (76GB uncompressed), corresponding to the first week of August, part of the UGR'16 Test set. This decision comes from the fact that the calibration subset does not feature synthetically generated attacks.

By conducting a random sampling of this week's data to retrieve only a 0.1% fraction of the original size of the file, we attempt to maintain a proportional representation of both attacks and background data to align with the original paper's [8] dataset composition, as analyzed in this paper [63], very imbalanced towards background data. It allowed us to employ the models used by the paper authors [9] for training, as with our processing capabilities we lack the necessary memory to train with very big sets of data. And with this previous steps we are able to determine if the Feature Engineering step truly makes our models have problems with the background class.

Subsequently, necessary modifications are applied to the CSV file of the selected week. This file lacks headers and some rows (described in subsection 3.2.3) have missing values, causing training errors in any model. Because of the missing values, we perform a filtering step to remove such rows and retain only those with complete data. It is done in bash code, removing the rows with a column number different from 13. Even though there are more sophisticated techniques to treat missing values, this approach is used due to the large size of our data that forces us on a simpler and faster approach.

After this preprocessing steps, the resulting dataset serves as a more manageable and representative training subset for our analyses and model evaluations.

4.2.2 Data Transformation

The transformation of our original netflow data is necessary due to its inherent diversity of data types, ranging from strings of dates and IP addresses to categorical variables. These diverse data types need a series of transformations to ensure the data's compatibility with

our Machine Learning models.To achieve this, we did the following transformations:

- **Conversion of Dates:** we convert strings of dates in format 'YYYY-mm-dd HH:MM:SS' into Unix timestamps. This conversion standardizes the time representation and enables consistent time-based analyses.
- **IP Address Transformation:** IP addresses are transformed from strings that represent them and include points into integer representations, from '125.149.122.195' to 125149122195. This transformation facilitates uniform numerical treatment, allowing these values to be efficiently processed by our models.
- **Binary Encoding of Protocol and Flags:** to accommodate Machine Learning algorithms that expect numerical inputs, we binary-encode protocol information and network packet flags. This ensures that these categorical variables are appropriately processed by our models.

4.2.3 Evaluating the Hypothesis

As said in [9], a possible problem in the implementation of the Methodology could have to be with the Feature Engineering part with FaaC, as it decreases the number of background samples and the authors say it could explain their lower results in the class that represents the normal traffic. For this, after doing the necessary transformation of the original dataset and extracting a section of it we attempt to reproduce their experiments but without the Feature Engineering.

For this, we chose to train our Machine Learning models with a subsample of the Test set of the UGR'16 Dataset [10], as it is the only set that contains generated attacks and as such it contains a class distribution better for evaluating the real performance of our model as seen in the table 4.1 in comparison with the Calibration set.

To test the Machine Learning models we have two possible ways, to test it also with the Test set or to test it with the Calibration set. By testing with the Test set we can analyze the capacity of our system to classify background data from the different generated attacks, but we fall into an unrealistic scenario, where it does not show the reality of our system with real traffic. And we can also test it with the Calibration test to find if our system can accurately differentiate between background data and real attacks. For the comparisons we will use the Random Forest (RF) Model, as it has the best results in [9].

Class	Count	Percentage from Total (%)
Background	811375	92.12%
Spam	27957	3.17%
Dos	4959	0.56%
Scan44	2417	0.27%
Blacklist	2374	0.27%
UDPScan	1005	0.11%
Botnet	961	0.11%
Scan11	566	0.06%

Table 4.1: Class Distribution from subsampled Test file

As we can see in table 4.2, Random Forest has very good results at deciding the class of each sample with the Test set, which includes the generated attacks, better than the ones shown in table 4.4 with the replicated results, but they can't be conclusive as it has been evaluated with only a very small part of the whole dataset because of the memory limitations that avoiding the Feature Engineering entails. But when we try to test the same model with the Calibration set, it proves that it does not have a perfect ability to differentiate background data from attacks in real traffic. For it to be evaluated with the necessary depth, it is necessary to train and test it with as much as possible of the UGR'16 Dataset, but it falls outside the scope of our Project, due to temporal restrictions.

Model	Class	P	R	F1	AUC
RF tested with Test set	Background	1.00	1.00	1.00	0.9752
	Dos	1.00	1.00	1.00	1.0000
	Scan44	0.99	0.98	0.99	0.9905
	Spam	1.00	1.00	1.00	0.9990
	Blacklist	0.90	0.22	0.36	0.6117
	Botnet	0.93	0.93	0.93	0.9635
	UDPscan	1.00	1.00	1.00	1.0000
	Scan11	0.97	0.98	0.98	0.9902
RF tested with Calibration set [1]	Background	1.00	0.92	0.96	0.4948
	Blacklist	0.00	0.00	0.00	0.5000

Table 4.2: Performance comparison for Random Forest trained with the Test set and tested with multiple sets

4.3 Replication of the original results

After finishing working with the raw dataset, it is the moment to attempt to replicate the results given in [9]. The aim is to ensure that we are not implementing their methodology badly, that the processing done in the dataset is correct and, therefore, that we will be able to make a fair comparison between the models developed later in this project (see chapter 5) and the results already published.

The first attempt to replicate the initial results was to analyze the original paper [9] to get the explanations of the used tools for each part of the process, starting with the *Feature Extraction* with the Feature as a Counter (FaaC)) method, implemented in the FCParser github repository [64]. After using the tool with the needed configuration, we got the expected results, from 11 initial features to 132 counters.

Afterwards, we commenced with the rest of the methodology steps. In order to achieve a deeper understanding of the tools used and as a more formative action for the student, our first attempt was implementing each of the steps by researching each tool that they used. However, soon we find some problems as some of the steps given in the original paper are not well explained to be able to exactly reproduce their experiments. We cannot say that our time was lost as with the research necessary to reproduce the methodology we improved our knowledge in all of the tools that are used in the original repository but, finally, we decided to employ the code uploaded by the authors of the article in the ff4ml repository [62]. Once

we were able to use the developed code for the methodology, we were able to retrain the models to reproduce the paper experiments. As part of the training process, we carried out the hyper-parameter selection and optimization step. Table 4.3 contains the set of optimal parameters obtained for each of the Machine Learning models considered in [9].

Model	Optimal Hyper-parameter values
LR	max_iter=10000, penalty='none'
RF	max_features=16, n_estimators=318
SVC-RBF	C=0.1804, gamma=0.1256
SVC-L	C=0.1008 , kernel='linear'

Table 4.3: Best hyper-parameter configurations for the replicated models
Note: LR - Logistic Regression, RF - Random Forest, SVC-RBF - Support Vector Classification with Radial Basis Function kernel, SVC-L - Support Vector Classification with Linear kernel.

Table 4.4 contains the results obtained by each model in terms of Precision (P), Recall (R), F1 Score (F1) and Area Under Curve, broken down for each class (Background, Dos, Botnet, Scan and Spam) except UDP and SSH Scan, as they have a very low number of samples and the models do not learn to classify them. Also, it is declared the percentage of difference in the weighted average (calculated by summing all classes' AUCs multiplied by the number of samples of each class in the test set and then divided by the total number of samples in the test set) compared to the original. The results are practically identical, with some minor changes due to the intrinsic randomness inside the methodology, mainly in the Hyper-parameter selection, as the Bayes Search does not try each time all the combinations so the sub-optimal hyper-parameters could change from different runs, and the training of the model. Therefore we can conclude this crucial step of the process and start implementing improvements to the proposed ones.

Model	Class	P	R	F1	AUC
LR	Background	0.8182	0.9251	0.8684	0.7750
	Dos	0.8111	0.8690	0.8391	0.9570
	Botnet	0.9533	0.8644	0.9067	0.9450
	Scan	0.7292	0.9722	0.8333	0.9570
	Spam	0.8016	0.6241	0.7018	0.7640
	Weighted avg.	0.8132 (+0.4%)	0.8216 (+1.2%)	0.8115 (+0.7%)	0.7830 (+0.9%)
RF	Background	0.9134	0.9553	0.9339	0.8710
	Dos	0.9487	0.8810	0.9136	0.9420
	Botnet	0.9732	0.9237	0.9478	0.9610
	Scan	1.0000	0.9722	0.9859	0.9620
	Spam	0.9155	0.8249	0.8678	0.8570
	Weighted avg.	**0.9147** (+2.0%)	**0.9095** (+2.5%)	**0.9110** (+2.6%)	**0.8977** (+3.4%)
SVC-RBF	Background	0.8190	0.9370	0.8860	0.8100
	Dos	0.9136	0.8810	0.8970	0.9150
	Botnet	0.9633	0.8898	0.9251	0.9430
	Scan	0.9429	0.9167	0.9296	0.7680
	Spam	0.8261	0.7393	0.7802	0.7825
	Weighted avg.	0.8261 (-1.3%)	0.8264 (-0.7%)	0.8183 (-1.0%)	0.7875 (-2.3%)
SVC-L	Background	0.8194	0.9364	0.8740	0.7876
	Dos	0.9136	0.8810	0.8970	0.9150
	Botnet	0.9633	0.8898	0.9251	0.9430
	Scan	0.9429	0.9167	0.9296	0.7680
	Spam	0.8321	0.6162	0.7081	0.7900
	Weighted avg.	0.8261 (-0.0%)	0.8264 (+0.7%)	0.8183 (+0.4%)	0.7875 (+0.3%)

Table 4.4: Replicated performance for the original paper models following their methodology

We find it also important for the future comparisons to show in table 4.5 the time each of the four Machine Learning models took to train and give their predictions, as some models were very costly to train and could be an important measure for the next results. As we can see, the Logistic Regression model is the fastest one to train, followed by the Random Forest that takes much more time but 10 times less than the SVC models.

Since Random Forest is the classifier with the best results we will review its confusion matrix in figure 4.2 to mark our baseline of improvement. We can observe that our Random Forest model has a clear bias towards the background and the spam classes, as those are the ones that have more samples, being the biggest problem the false mix up of instances from

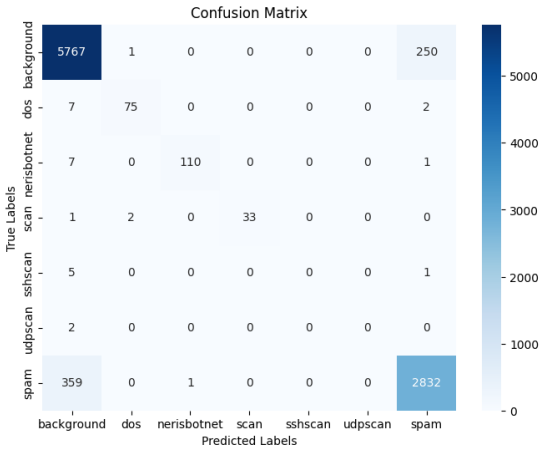


Figure 4.2: Confusion Matrix of the Random Forest

both classes. The bias towards the background class is bigger, as the smaller classes also tend to have some samples wrongly classified as background.

Model	**Elapsed Time (s)**
Logistic Regression	**17.24**
Random Forest	722.82
SVC-L	7205.18
SVC-RBF	10197.14

Table 4.5: Elapsed time of training and testing for the replicated models

Chapter 5

Models Development and Results

After the initial positive results reproducing the paper values for the given models, we will present each of the Novel proposal suggestions that have been evaluated during this Project's development, along with their outcomes, whether they are positive or not.

Despite instances where the ultimate result might not improve the previous from [9], this process of experimentation has provided us with an increased understanding on the field and will be helpful for analyzing future research lines.

To allow for a better understanding of the progress of the project, the next sections will attempt to follow a chronological order, but some of the experiments were carried in parallel, so selecting a closed time range for each section has been rendered unnecessary.

For this, the experiments consist of using Conventional Classifiers, Deep Learning models, Outlier Detectors, and Ensemble Models to combine the best resulting models. We will first begin with an explanation of the metrics used for our experiments, the same used in [9] to be able to compare results.

5.1 Metrics used

For measuring the quality of the models the authors of the paper that will be used as our state-of-the-art reference, propose the use of 5 different metrics: Recall, Precision, F1 Score, Area Under Curve and the Weighted Average of these metrics. [65]:

- **Recall**: Recall is the proportion of correctly predicted positive instances (true positives) out of all actual positive instances (true positives + false negatives). It measures how well the model captures the positive cases. For example, in a Network Intrusion Detection System (NIDS), recall would be the percentage of actual network attacks that are correctly identified by the model. Recall is calculated as:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

- **Precision**: Precision measures the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive (true positives + false positives). It measures how accurate the model is when it makes a positive prediction. For example, in a NIDS, precision would be the percentage of instances flagged as attacks by the model that are indeed actual attacks. Precision is calculated as:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

- **F1 Score**: F1 score is the harmonic mean of precision and recall. It is a single metric that balances both the concerns of precision and recall. It is useful when you want to compare two NIDS models that have different trade-offs between precision and recall. F1 score is calculated as:

$$\text{F1 score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision + Recall}}$$

- **Area Under Curve (AUC)**: The Area Under Curve (AUC) is the area under the Receiver Operating Characteristic (ROC) curve. The ROC curve plots the true positive rate against the false positive rate for different threshold values. AUC measures how well the model can distinguish between actual network attacks and non-attacks. The higher the AUC, the better the model is at predicting attacks as attacks and non-attacks as non-attacks. AUC ranges from 0 to 1, where 0.5 is a random classifier and 1 is a perfect classifier.

- **Weighted Average**: Weighted average is a way of combining the metrics for different classes in a multi-class classification problem. It assigns a weight to each class based on its frequency or importance, and then calculates the average of the metrics using those weights. For the followed methodology proposed in [9], the weighted average of each metric is calculated summing each class' metric value multiplied by the number of samples in that class and finally dividing the result by the total number of instances.

5.2 Utilizing conventional classifiers

For this attempt to improve the results in the problem we are trying to solve we decided to use different classifiers from the ones that were used for the original paper, as we found it possible that other classifiers could reach the problem from another view and could give some interesting results.

For this, we propose using K-Nearest Neighbours classifier and Extreme Gradient Boosting, even though the Extreme Gradient Boosting is an ensemble method, as explained in 2.3.2,

we differentiate this approach from the section 5.4 as the Ensemble Methods section is focused on methods that combine the predictions of multiple different models, and XGBoost does not fall in this category.

After computing the Bayes hyper-parameter optimization for both models the final parameters for the optimal evaluation are in the table 5.1:

Model	Configuration
k Nearest Neighbours	metric='euclidean', n_neighbors=3, weights='distance"
Extreme Gradient Boosting	max_depth=6, n_estimators=1000

Table 5.1: Best configurations for the conventional classifiers

We can observe the evaluation results of both of the models in table 5.2, and as we can see they both have very good results, with an interesting result of both models having perfect Precision and a very good performance in general on the Scan class that can be seen in annexed table A.1. However the Extreme Gradient Boosting has the best results of the two of them in all of the metrics.

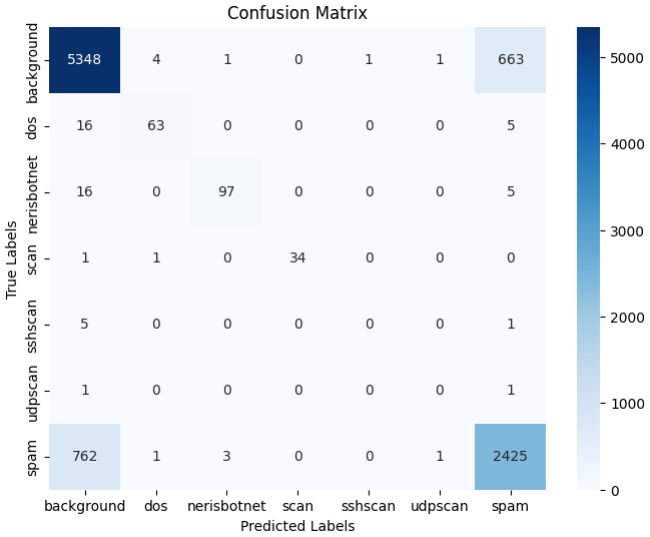
Also we can observe in the table 5.3 that both have very small training times and testing times, with the kNN significantly smaller than the XGBoost and for future improvements and to better know how each model treats each class we have the confusion matrix of both 5.1

Model	P	R	F1	AUC
k Nearest Neighbors r	0.8419	0.8417	0.8416	0.8260
Extreme Gradient Boosting	**0.9291**	**0.9290**	**0.9289**	**0.9219**

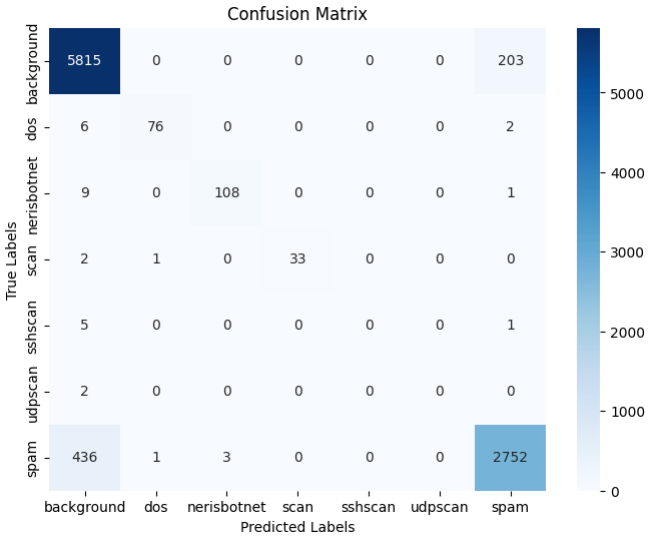
Table 5.2: Weighted Performance for proposed conventional models

Layer Configuration	Elapsed Time (s)
kNN	**72.00**
XGBOOST	173.18

Table 5.3: Elapsed time of training and testing for the conventional classifiers



(a) Confusion Matrix for KNN Model



(b) Confusion Matrix for XGBOOST Model

Figure 5.1: Confusion Matrices for KNN and XGBOOST Models

But if we want to know the level of our improvement we need to compare it in the figure 5.2 with the performance values of the replicated Random Forest that we selected as our baseline.

In the comparison we can observe clearly that the newly proposed Extreme Gradient Boosting turns out to be better in all metrics than the previous best, the Random Forest.

The K-nearest Neighbours is still worse than the Random Forest but still has good performance and very low training times that can be beneficial for larger quantities of data.

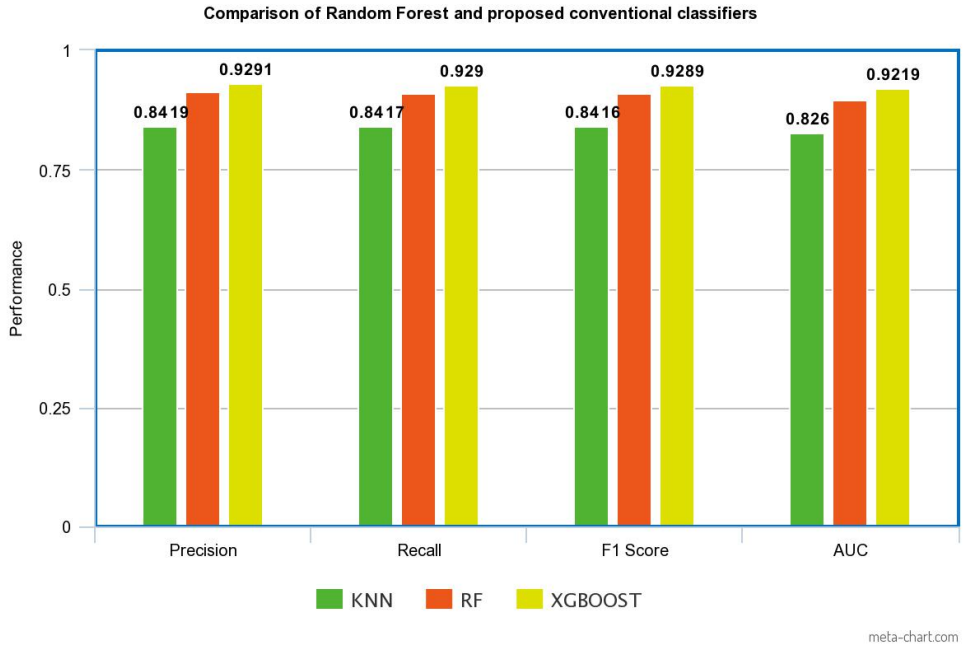


Figure 5.2: Comparison of performance of the proposed classifiers with the Random Forest

5.2.1 Experimentation with Over and Under-Sampling

After obtaining the positive results and also analyzing the structure of our dataset after the Feature Engineering, which can be seen in table 5.4, we added as a possible improvement to implement Oversampling, Undersampling or both with models like SMOTEEN[66] to level the distribution of our data over the different classes and possibly reduce the bias of our model.

But after trying multiple different techniques for each of the types of sampling we came to the conclusion that this did not only not help our models grasp the complexity of our classes with fewer instances, but degrade heavily the performance metrics in all classes.

Class	Count	Percentage from Total (%)
Background	30091	63.41%
Dos	417	0.88%
Botnet	594	1.25%
Scan	176	0.37%
Spam	15961	33.63%
SSHScan	27	0.06%
UDPScan	9	0.02%

Table 5.4: Class Distribution

5.3 Deep Learning models

After trying the conventional classifiers we want to compare them with another technique that is very used nowadays for wrongful behaviour detection in NIDS, Deep Learning[67] [68].

For this the proposed model to use is the sklearn Multilayer Perceptron[69].

Deep Learning method have been proven to be very good at predicting attacks in computer networks so they could be a good improvement over the reproduced results with the conventional classifiers.

In the case of this particular model, the initial crucial step entails hyper-parameter selection. However, this process might be deemed more intricate compared to that of conventional classifiers. This intricacy arises due to the need to explore diverse hidden layer architectures, alongside activation functions, solver choices, and several other pertinent attributes.

As, at the moment of development of this project, it is not possible to tune the MLP hyper-parameters with the hyper-parameter optimizer proposed in [9] directly, because it has still not been added the possibility of optimization with tuples, and we need them to be able to optimize our MLP with multiple hidden layers. For this we had to create a custom wrapper class that would let us find the best configuration.

After the hyper-parameter tuning, we had 4 different results, one for each configuration of layers.

As it is not possible to test with only one fit for the optimizer configurations with multiple number of hidden layers we need to run it 4 times, each selecting the layers we want to train

our tuner with.

The optimal results given by the Bayes Search are 5.5:

Configuration	Hidden Layer Sizes	Activation	Solver	Alpha	Max Iterations
1 Layer	45	relu	sgd	0.1	873
2 Layers	49, 59	relu	sgd	0.0001	229
3 Layers	34, 28, 30	relu	lbfgs	0.0001	678
4 Layers	39,49,48,32	tanh	lbfgs	0.002	748

Table 5.5: Best configurations for the MLPClassifier models

Model	P	R	F1	AUC
1 Layer	0.8812	0.8853	0.8832	0.8888
2 Layers	0.8893	**0.8933**	**0.8913**	0.8962
3 Layers	**0.8950**	0.8861	0.8902	**0.8980**
4 Layers	0.8662	0.8735	0.8698	0.8742

Table 5.6: Weighted Performance for the MLPClassifier models

Layer Configuration	Elapsed Time (s)
1 Layer	478.08
2 Layers	**108.87**
3 Layers	297.09
4 Layers	998.60

Table 5.7: Elapsed time of training and testing for the MLP configurations

To select what can be our best configuration we have two possible answers, 2 or 3 Layers. As for comparison we choose to use the weighted F1 Score and AUC.

If we take into account the scores they are leveled as with 3 Layers we obtain a better F1 Score but lower AUC, so to choose the best model we will apply the followed in Machine Learning principle of Occam's Razor, when faced with two approaches of similar performance, the simpler one is always chosen. So for the next results the 2 Layers model is chosen as the best one, as is less complex than the other best, the 3 layered one, with a lower number of hidden layers.

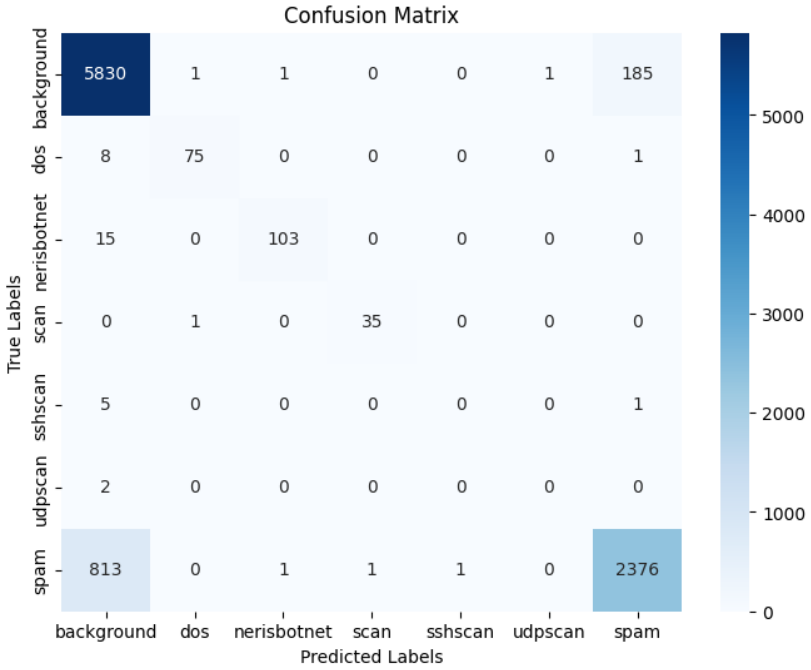


Figure 5.3: Confusion matrix for the 2 Layered MLP

To be able to analyze how our resulting model reacts to every class we can observe the figure 5.3, and to really understand these results we need to compare them with the previous ones obtained from the original paper.

For this comparison we will use also the results from the Random Forest Classifier that we replicated from the original paper, as it is determined in it that it has the best results in with all metrics in all different classes, and the 2 Layered MLP, as stated before.

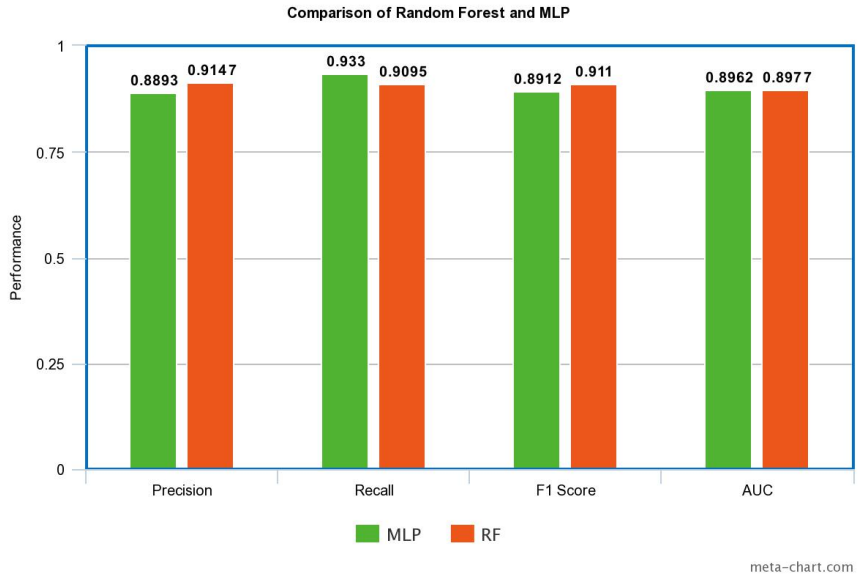


Figure 5.4: Comparison of performance for Random Forest and MLP

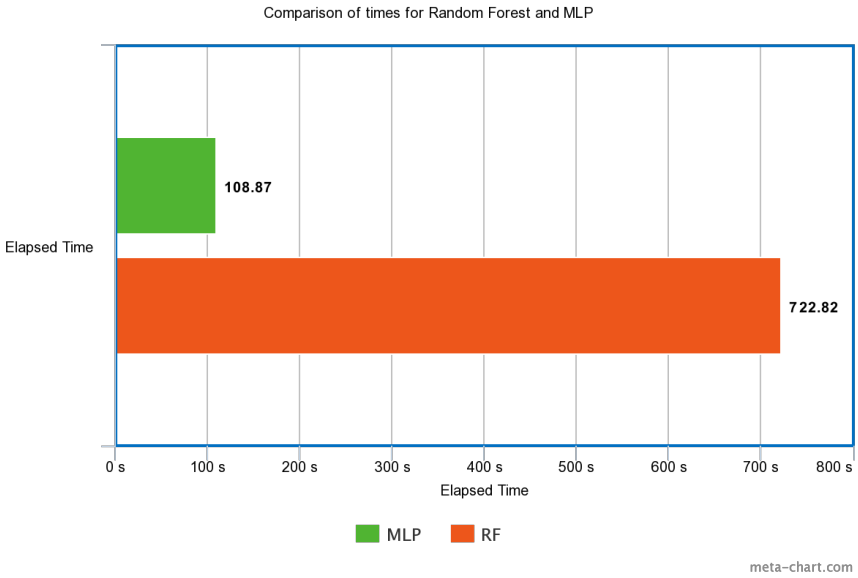


Figure 5.5: Comparison of Elapsed Time for Random Forest and MLP

As we can see in the graph 5.4 , both models have very similar performance, with a better Precision and F1 Score for the Random Forest, a better Recall for MLP and a similar AUC for both models.

The main difference can be seen in the graph 5.5, Random Forest took in total almost 7 times more time than the MLP with 2 Layers.

For this we can conclude that they are both interchangeable, but if used for a bigger set of data the best model would be the MLP for its lower training times.

5.4 Combining Models: Ensemble Methods

The last proposed approach consists in the use of Ensemble Models to try to mix the results of the classifiers that had the best outcomes and improve those results based on the principle that different classifiers can approach in multiple ways the same problem and be able to detect the class correctly of some data that others cannot.

There will be two tried different ensemble methods: Voting and Stacking.

5.4.1 Voting Ensemble

For the Voting Ensemble implementation we will use the 3 best models that we have gotten in the development of this project, XGBoost, MLP with 2 layers and the replicated Random Forest, combined with the sklearn Voting Classifier[70].

It is important use use an uneven number of estimators to avoid draws in the voting that would result in a possibly incorrect result of the voting process, as with draws in choosing the prediction by alphabetic order.

To do the experimentation we will use soft voting, using the prediction probabilities, as it is recommended for well calibrated estimators.

There are two possible solutions, to use uniform weights for the voting process or to find the best weights to combine the predictions of the estimators. Our hypothesis is that the second solution should work better as the XGBoost model should have more importance in the decision for its better overall performance.

As for the model without custom weights, it doesn't need a parameter optimization, as none can be changed. But for the second solution we can attempt to optimize with Bayes Search the weights that will give more importance or less to the model predictions. For this we need to implement a custom class for a Voting Wrapper, similar to the one used for the MLP layers optimization.

After this step, the optimal solution proposed by the Bayes optimizer is 6 for XGBoost and 7 for Random Forest and MLP. This seems non-intuitive as the model with the best performance is XGBoost, the one that the proposed optimal weights give less importance in the decision. But it could probably be because there are different predictions in the RF and MLP compared to the XGBoost and less decision power for the XGBoost could be beneficial in the final result.

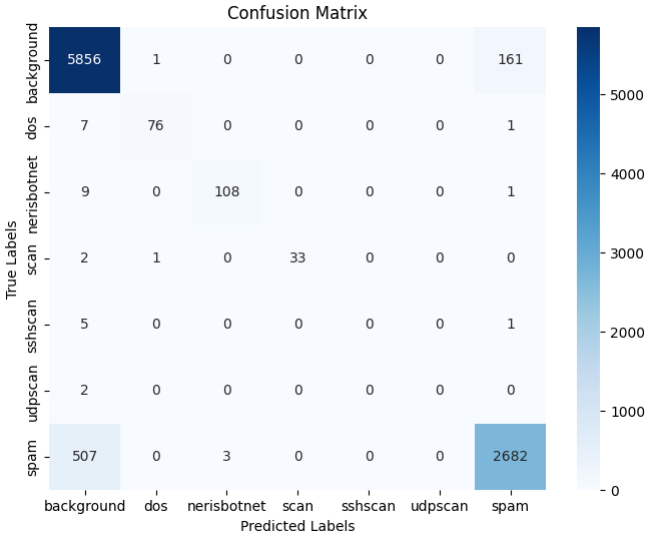
As we can see in the table 5.8, both models have a very similar performance, with a slight improvement with the custom weights, specially in the Recall and F1 Score metrics, proving

our initial hypothesis.

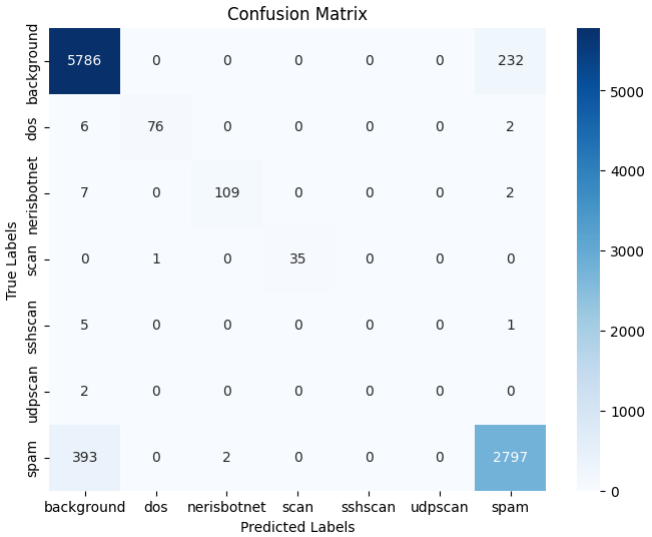
In the figures displayed in 5.6 we can appreciate the main difference between the two Voting models, the model with custom weights has more true positives for spam and less for background than with uniform weights.

Model	**P**	**R**	**F1**	**AUC**
Voting Classifier with uniform weights	0.9298	0.9264	0.9277	0.9188
Voting Classifier with custom weights	**0.9300**	**0.9309**	**0.9301**	**0.9209**

Table 5.8: Weighted Performance for Voting Models



(a) Confusion Matrix for Voting model with uniform weights



(b) Confusion Matrix for Voting Model with custom weights

Figure 5.6: Confusion Matrices for Voting with uniform and custom weights

However, to really analyze the possible improvement of our Voting Classifier we need to compare it with our best model for the moment, XGBoost. We propose the comparison with XGBoost and not with Random Forest because with Extreme Gradient Boosting we have been able to surpass the previous replicated results and, since we use it as an estimator for our Voting model, the comparison should be with it.

As we can see in the graph 5.7, there is a minimal improvement over the use of just the XGBoost model with the Voting Classifier, but it is so small it cannot be considered significant. And to follow the Occam's Razor principle, the XGBoost model is still considered best between

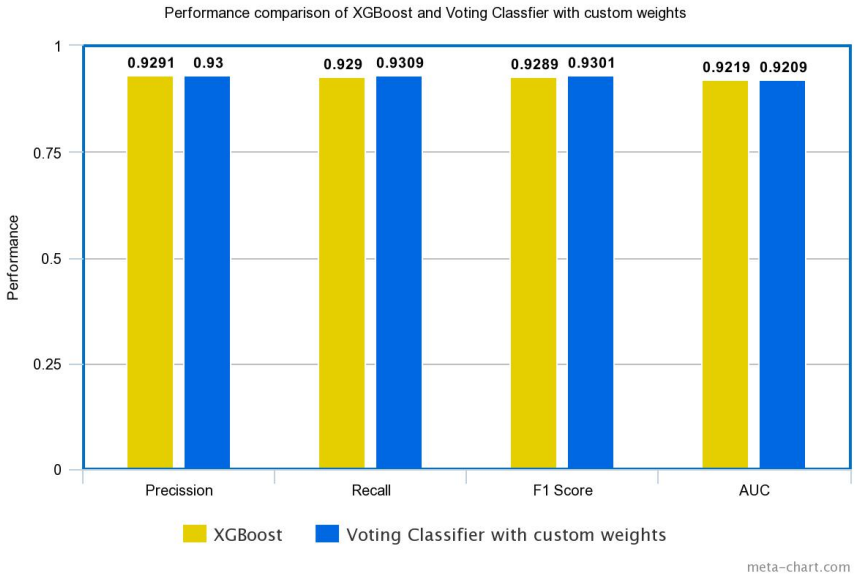


Figure 5.7: Comparison of performance for XGBoost and Voting Classifier with custom weights

the two, with lower complexity.

5.4.2 Stacking Ensemble

We will use the Stacking Ensemble given by the Sklearn library to attempt to improve the results of our three best models combined, the same used for the Voting Ensemble.

For this experimentation, hyper-parameter selection was dismissed, as the only hyper-parameter that makes sense to change is the passthrough that selects whether our final estimator will include for its training the original data or only the predictions of the list of estimators.

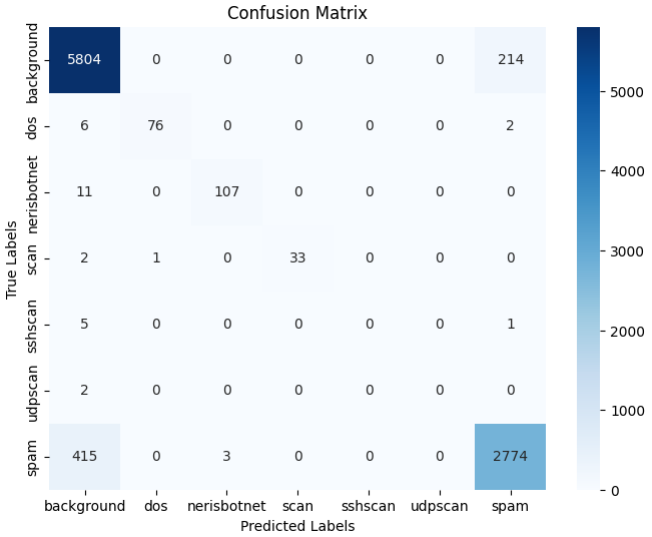
For our final estimator that will give the predictions we will use the default Logistic Regression model, as it is usually not recommended to change it.

As we can observe in the table 5.9, both with and without passthrough behave really well with our data, with slightly better results with the passthrough activated, as it can benefit apart from the stacking predictions from the original data to train the final estimator.

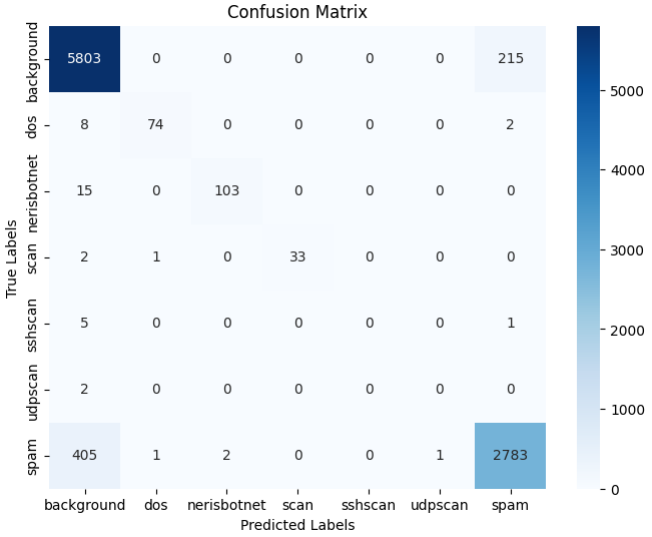
And in the figure 5.8 we can understand the difference between both approaches, with passthrough it is slightly better at differentiating the spam class from background data, with 9 more elements classified correctly as spam in the figure 5.8b than in 5.8a.

Model	P	R	F1	AUC
Stacking without passthrough	0.9307	0.9284	0.9294	0.9224
Stacking with passthrough	**0.9301**	**0.9308**	**0.9303**	**0.9237**

Table 5.9: Weighted Performance for Stacking classifiers with and without passthrough



(a) Confusion Matrix for Stacking model without passthrough



(b) Confusion Matrix for Stacking Model with passthrough

Figure 5.8: Confusion Matrices for Stacking with and without passthrough

The results are very similar to the XGBoost results in table 5.2, but with a slight improve-

ment that is not really significant, and following the Occam's Razor principle, the simplest and best model between the eXtreme Gradient Boosting and the Stacking Classifier is the XGBoost.

5.5 Using models for Outlier Detection

As we are using an imbalanced dataset like UGR'16 [63], even after the Feature Engineering proposed in [9], the problem could benefit from the use of Outlier Detectors, possibly improving the capacity to differentiate background from attack data and using the imbalances that bias towards background data in our favour. For an Outlier Detection problem, we will no longer be working with a multi-class classification problem, working only with inlier and outlier data, so multiple changes will have to be done to the Dataset to effectively train and test our Outlier Detection models.

To test the assumption that the Intrusion Detection problem can be solved under this perspective we used two different outlier detectors that work in completely different ways: Isolation Forest and One Class SVM.

Also, to work with the outlier detection methods with the UGR'16 dataset we have to make some modifications to the ff4ml methodology, as this is no longer a classification method and we cannot use the same structure of data or the same metrics to evaluate the trained models.

The first thing to solve is regarding the training data, as we have two proposed approaches: to train with the same training set as the previous experiments or to train the Outlier Detectors only with inlier data, the background class, to later be able to identify anomalies in the test set that differ from the normal baseline. So, for the second approach, we need to extract from the training data only the background cases to get the inlier data and we also need to transform our Dataset into a binary problem, i.e. *Background* and *Other*, as it is not possible to solve a multi-class problem with an Outlier Detector.

To obtain the best parameters for our models we also found a problem, as the prediction values of the outlier detector seem to not work with the hyper-parameter optimization scorer used in [9], in charge of computing the F1 Score over all classes without taking into account the number of samples of each class. It is important to use that score, as the rest of the models and the used in [9] use it, so, in order to maintain equal conditions, we needed to create our own scorer to evaluate the predictions of our Outlier Detectors

5.5.1 Experimentation with Outlier Detectors

For the hyper-parameter optimization step we had two possible approaches, to optimize our models to train with background data only or with a mix of background and attack data. So we decided to try both, as Outlier Detectors are implemented with the possibility of selecting

the level of contamination and it could work well depending on the parameters used.

The results in the table 5.10 are expected to take into account the data used for training. For the Isolation Forest, the contamination varies from near 0 for the background data to a third with attacks added into the training set. This makes sense compared with the distribution of data shown in table 5.4.

And for the One Class SVM, the *nu* parameter represents the percentage of samples that can be classified by an outlier by the decision boundary, and it matches, for the model trained with attacks, the distribution of our classes.

Model	Configuration
Isolation Forest without attacks	bootstrap=True, contamination=0.0289, max_features=30, max_samples=272, n_estimators=316
Isolation Forest with attacks	contamination=0.3210, max_features=36, max_samples=834, n_estimators=390
One Class SVM without attacks	gamma=0.0002, max_iter=12443, nu=0.7493, tol=3.8688e-05
One Class SVM with attacks	coef0=5.2869, degree=2, gamma=0.0023, kernel='poly', max_iter=24806, nu=0.3487, tol=6.5637e-05

Table 5.10: Best configurations for the conventional classifiers

As for this experiments we have multiple different sets of data that we can use, we have the opportunity to evaluate our optimized models each with and without attacks mixed inside our data. In the table 5.11 we can observe how each of the approaches performed, with a common bad result at being able to detect attacks in the test data except for the One Class SVM that had the hyper-parameter optimization done with the normal set of data that includes attacks, and trained only with the Background data, that has only samples labelled as normal traffic. As for times comparison, all of them had very small and similar training and testing times, around 10s, so it is not found relevant for the discussion.

Compared with the Random Forest model, the best outlier detector results are still much worse than the results of the RF over the Background class shown in table 4.4

Model	Trained With	Tested With	P	R	F1	AUC
Isolation Forest	N	N	0.5567	0.5609	0.5587	0.5208
Isolation Forest	B	B	0.6224	0.6449	0.5490	0.5226
Isolation Forest	B	N	0.6001	0.6165	0.5692	0.5731
Isolation Forest	N	B	0.5724	0.4923	0.5275	0.5055
One Class SVM	N	N	0.5611	0.5643	0.5626	0.5256
One Class SVM	B	B	0.5467	0.4407	0.4183	0.4957
One Class SVM	B	N	**0.7760**	**0.7323**	**0.7373**	**0.7556**
One Class SVM	N	B	0.5346	0.4325	0.4099	0.4981

Table 5.11: Weighted Performance for Outlier Detection Models

5.6 Discussion of Results

As seen in the previous sections, some models have improved the results given in [9], some have worsened them and others have very similar results. They have been compared separately but in this section we will compare all the experimentations done to select the best models with the best results.

For our comparison, we have to take into account Outlier detectors, Ensemble Methods to join multiple classifiers, conventional classifiers and Deep Learning Approaches. And we will compare them by their performance metrics and also their run times.

Within the 4 considered groups, the two that had the best results were the Ensemble Methods and the Conventional Classifiers, followed by the Deep Learning with the Multilayer Perceptron and finally the Outlier Detectors. As we can observe in figure 5.9, when we use for comparison the Weighted Area Over Curve of the evaluated models with the best results of each group, we can find the best results in the Extreme Gradient Boosting (XGBoost) and the Stacking Classifier with XGBoost, RF and MLP as estimators for the Logistic Regression final estimator. We can also observe that Anomaly Detection models don't work very well with the UGR'16 Dataset [8] and the Methodology ff4ml [9], possibly because of the reduction of background data produced in the Feature Engineering and the augmentation of the number features the process produces, from 11 to 132 and 87 after the Feature Selection.

Finally, we can conclude that the Ensemble Methods, like the XGBoost, the Stacking Classifier, and the previous better results of Random Forest presented in [9], are still the most capable of discerning the differences between the classes with very good results in all of them. The best of this Ensemble Models in terms of performance results is the Stacking Classifier with passthrough activated. It is capable of slightly improving the results of the XGBoost by combining its predictions with the Random Forest and the 2 Layers MLP. But because the improvement is not really considerable, and to follow the principle of the Occam's Razor, the model selected as the best in term of quality is the Extreme Gradient Boosting, having less complexity because we don't have to combine multiple different models and thus having much shorter training and testing times.

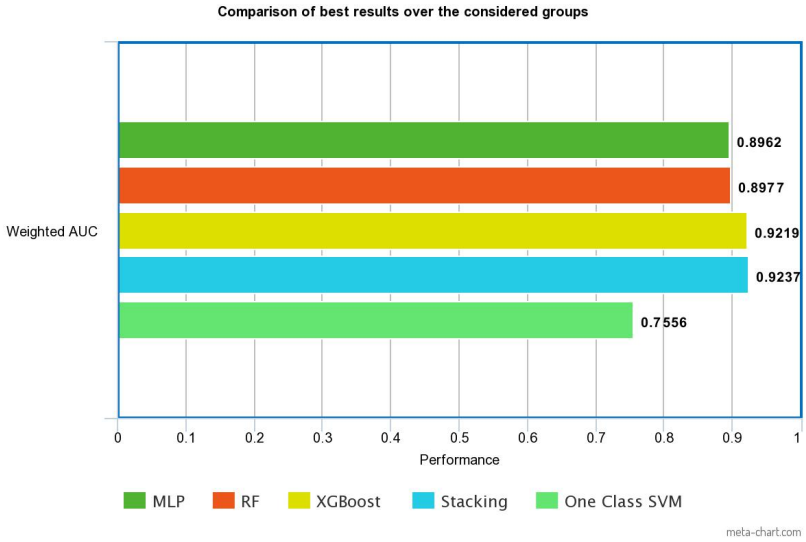


Figure 5.9: Comparison of best results over the considered groups

Chapter 6

Conclusions and Future Work

During the development of this Project, valuable knowledge has been gained about the field of Intrusion Detection Systems with Machine Learning techniques, and some important results have been extracted from it. In this next chapter the extracted conclusions, including the valuable lessons obtained from the process, will be exposed, and also the possible Future Approaches in this field that our work did not include and, thanks to the increased knowledge in the field, are selected as possible high interest research lines.

6.1 Conclusions

For this section we have decided to differentiate into two main point of views of the conclusions of the Project, the Researcher and the Student point of views, separating the main advances of the novel proposals for the development of a NIDS with the UGR'16 dataset from the gained and used knowledge over the course of this work, presenting the important set of information that was obtained during the Computer Engineering University Grade and was beneficial for the development of this work, and the gained knowledge with in the process of the Project.

6.1.1 Researcher Conclusions

In this Project we have developed new Machine Learning proposals of improvement for a Cyclo-Stationary Network Intrusion Detection System. For this we have used the UGR'16 Dataset [8] as a train and test set for our Machine Learning models and followed the methodology proposed in [9] to be able to compare our results and their previous results. The Dataset was composed of 16000 Million different flows in 4000 Million CSV rows and was latter reduced, following the methodology, to 47000 rows that compress all that information into more useful data that allowes our model to analyze the network traffic in bigger spans of time.

As it is custom, the previous results from [9] and their methodology were recreated to

establish a baseline of comparison for our improvement proposals and Random Forest was declared, as it was in the paper [9], the best model from all previously tested.

The approaches range from using Conventional Classifiers, like Extreme Gradient Boosting (XGBoost) and k - Nearest Neighbours, to Deep Learning with the Multilayer Perceptron, Outlier Detection methods likes Isolation Forest and One Class SVM and finally Ensemble Methods to combine the individual models that had the best results. The previous results were surpassed with the XGBoost and the Ensemble Methods, that used a combination of Random Forest, XGBoost and Multilayer Perceptron (as these methods had the best results) for Voting and Stacking Classifiers. The best results are given by the Stacking Classifier, but they are very close to the XGBoost alone, and considering the time of training the other models and following Occam's Razor principle, the XGBoost model is considered the best to classify network data and detect attacks as it has lower complexity than the Stacking Classifier.

6.1.2 Student Conclusions

For the development of this project, the student has had to put into practice a lot of the knowledge gained in the course of his studies, going into high detail of the functioning of the Intrusion Detection Systems and the different computer network attacks as seen in the subject of Networks (2º year), using methodologies like SCRUM that were taught in Software Process (2º Year) and using the planification techniques learned in Project Management (3º year). As the focus of the Project is in the development of Machine Learning approaches for the detection of attacks using the UGR'16 dataset[8], related knowledge was obtained in the subjects Intelligent Systems (2º year), where most of the used models were presented, and later improved with more depth into each of them in the subject Machine Learning (3º year), and relative to the Intrusion Detection Systems and the types of attacks, a lot of his knowledge comes from Computer Security and Legislation (3º year). And finally very important general knowledge was gained in the whole student carrier, with special mention to the Company Practices (4º year) that provided with knowledge on research with Machine Learning models and taught the student about the report process, together with the subject of Intelligent Systems Development (3º year), that focused more on the documentation for Artificial Intelligence models. The documentation of said subjects is available in [71]

Apart from the used resources obtained in the Degree, the student also had to do an intensive research on the different detection methods Intrusion Detection Systems employ and how the state-of-the-art Machine Learning methods are introduced into the field and what methodologies are used for comparing the performance of different models over the same Dataset. The student also had to research on the different Datasets available to train and test the Machine Learning models, outside of the scope of the University Degree. Outlier Detection methods also were not in the taught curriculum as well as newer models like Gradient

Boosting and there had to be autonomous work from the student to apply these models and the ones taught in the Degree, as most of them were only taught theoretically and never used in practice.

In conclusion, the development and report of this Project has been very beneficial for the student advancement and acquisition of the required competences for a student of Computer Engineering in the Computation itinerary, allowing the student to apply the learned knowledge over the course of his studies and contributing to his development as a Computer Scientist.

6.2 Future work

In this section we will explore four proposed future works that were not attempted in the scope of this project and have been found as a possible improvement over the final results and as future research approaches for the UGR'16 dataset [8] and the used methodology from [9].

6.2.1 Experiment with the raw UGR'16 Dataset

As explained in subsection 4.2.3, this approach fell outside the scope of our Project due to temporary restrictions. However it would be an important advance to train the models with the best results with the Dataset UGR'16 without Feature Engineering and compare it with the results given in this Project to evaluate posible improvements to the methodology proposed in [9] and followed for the development of this Project. We could extract conclusions on how the different steps proposed affect the final result for our detectors and compare the train and test times and energy consumption for our models.

6.2.2 Explainable and Green Artificial Intelligence

In consonance to the new directives about inclusive, sustainable and centered on the citizens AI from the European Union [72] and the Spanish Government [73], the use of Explicable and Green models are proposed as a future improvement.

Explainablility could help with transparency of how the system recognises threats, to open the possibilty to the general public to understand the process of decision behind detection.

Green models lie in the principle of the sustainable AI. With the start era of information, the size and complexity of Machine Learning models and the Dataset these use for training have increased exponentially. The EU and Spanish directives compel us to tend to use Machine Learning models with lower computational load and to improve our Datasets to reduce the unnecessary information to reduce the consumption of our models.

6.2.3 Apply the methodology in other NIDS Datasets

We propose this branch of experimentation for the methodology of [9] used in this Project to be able to empirically compare the performance of our models over mutiple Datasets for Cyclo Stationary NIDS. This would give important information to compare the state-of-the-art Datasets in the field with the used UGR'16 and extract conclusions over what features should a good NIDS Dataset contain.

6.2.4 Develop a working NIDS

As the Project found improvements over previous results in models for NIDS, the next step is to test this model in real conditions, building a working real time Network Intrusion Detection System to evaluate our model with real traffic and newer attacks.

Appendices

Appendix A

Additional Material

In this chapter we will include the necessary content that, due to the extent of it or because it is found to not add to the understanding of the project, has been excluded from the main corpus of this report.

A.1 Full Performance Metrics Tables

In this section, the full tables where the performance of the tested models over the different classes individually and collectively can be seen.

Model	Class	P	R	F1	AUC
KNeighborsClassifier	Background	0.8704	0.8873	0.8788	0.8280
	Dos	0.9130	0.7500	0.8235	0.8747
	Botnet	0.9604	0.8220	0.8858	0.9108
	Scan	1.0000	0.9444	0.9714	0.9722
	Spam	0.7823	0.7597	0.7708	0.8259
	Weighted avg.	0.8419	0.8417	0.8416	0.8260
XGBClassifier	Background	0.9377	0.9548	0.9462	0.9218
	Dos	0.9500	0.9048	0.9268	0.9522
	Botnet	0.9730	0.9153	0.9432	0.9575
	Scan	1.0000	0.9722	0.9859	0.9861
	Spam	0.9123	0.8835	0.8977	0.9200
	Weighted avg.	0.9291	0.9290	0.9289	0.9219

Table A.1: Performance for proposed conventional models

Model	Class	P	R	F1	AUC
1 Layer	Background	0.9026	0.9196	0.9110	0.8738
	Dos	0.9740	0.8929	0.9317	0.9463
	Botnet	0.9130	0.8898	0.9013	0.9444
	Scan	0.9722	0.9722	0.9722	0.9861
	Spam	0.8385	0.8214	0.8299	0.8704
	Weighted avg.	0.8812	0.8853	0.8832	0.8888
2 Layers	Background	0.9128	0.9184	0.9156	0.8845
	Dos	0.8864	0.9286	0.9070	0.9463
	Botnet	0.9145	0.9068	0.9106	0.9529
	Scan	0.9722	0.9722	0.9722	0.9861
	Spam	0.8453	0.8459	0.8456	0.8835
	Weighted avg.	0.8893	0.8933	0.8913	0.8962
3 Layers	Background	0.9110	0.9274	0.9191	0.8816
	Dos	0.8481	0.7976	0.8221	0.8982
	Botnet	0.9804	0.8475	0.9091	0.9236
	Scan	1.0000	0.9444	0.9714	0.9722
	Spam	0.8639	0.8136	0.8380	0.8742
	Weighted avg.	0.8950	0.8861	0.8902	0.8980
4 Layers	Background	0.8929	0.9018	0.8973	0.8603
	Dos	0.9277	0.9167	0.9222	0.9580
	Botnet	0.9310	0.9153	0.9231	0.9572
	Scan	0.9722	0.9722	0.9722	0.9861
	Spam	0.8128	0.8186	0.8157	0.8613
	Weighted avg.	0.8662	0.8735	0.8698	0.8742

Table A.2: Performance for the MLPClassifier models

Model	Class	P	R	F1	AUC
Voting Classifier with uniform weights	Background	0.9334	0.9598	0.9464	0.9199
	Dos	0.9620	0.9048	0.9325	0.9870
	Botnet	0.9730	0.9237	0.9432	0.9522
	Scan	1.0000	0.9722	0.9859	0.9575
	Spam	0.9219	0.8662	0.8932	0.9144
	Weighted avg.	0.9298	0.9264	0.9277	0.9188
Voting Classifier with custom weights	Background	0.9334	0.9614	0.9472	0.9207
	Dos	0.9870	0.9048	0.9441	0.9523
	Botnet	0.9819	0.9237	0.9519	0.9575
	Scan	1.0000	0.9722	0.9859	0.9861
	Spam	0.9219	0.8763	0.8985	0.9192
	Weighted avg.	0.9300	0.9309	0.9301	0.9209

Table A.3: Performance for Voting Models

Model	Class	P	R	F1	AUC
Stacking without passthrough	Background	0.9390	0.9538	0.9463	0.9225
	Dos	0.9870	0.9048	0.9441	0.9523
	Botnet	0.9727	0.9068	0.9386	0.9532
	Scan	1.0000	0.9444	0.9714	0.9722
	Spam	0.9136	0.8841	0.8986	0.9207
	Weighted avg.	0.9307	0.9284	0.9294	0.9224
Stacking with passthrough	Background	0.9392	0.9545	0.9468	0.9231
	Dos	0.9487	0.8810	0.9136	0.9403
	Botnet	0.9626	0.8729	0.9156	0.9362
	Scan	0.9189	0.9444	0.9315	0.9721
	Spam	0.9137	0.8919	0.9027	0.9245
	Weighted avg.	0.9301	0.9308	0.9303	0.9237

Table A.4: Performance for Stacking classifiers with and without passthrough

List of Acronyms

AI Artificial Intelligence. 10, 17

AUC Area Under Curve. 47–49, 52, 53, 57, 59, 61, 64, 66

FaaC Feature as a Counter. 42, 43, 45, 47

LR Logistic Regression. 24, 49

MLP Multilayer Perceptron. iv, v, 25, 56–60, 67

RF Random Forest. 22, 45, 47–49, 60, 66, 67

SIEM Security Information and Event Management. 10, 14

SVC Support Vector Classification. 19, 48, 49

SVM Support Vector Machine. 19, 65, 66

XGBoost eXtreme Gradient Boosting. iv, 23, 53, 60, 62–65, 67

Bibliography

[1] S. Sharma. (2021) Support vector machines(svm): A complete guide for beginners. Accessed on 2023-09-11. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

[2] scikit-learn developers. (2021) sklearn.svm.svc. Accessed on 2023-09-11. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[3] S. Saha. (2020) How to use the tree-based algorithm for machine learning. Accessed on 2023-09-11. [Online]. Available: https://www.freecodecamp.org/news/how-to-use-the-tree-based-algorithm-for-machine-learning/

[4] S. S. Mousavi, S. M. Hashemi, and M. A. Nekoui. (2019) Example of multilayer perceptron (mlp) network for regression analysis. Accessed on 2023-09-11. [Online]. Available: https://www.researchgate.net/figure/Example-of-multilayer-perceptron-MLP-network-for-regression-analysis-here-n-13-and_fig1_347244673

[5] S. Sharma. (2021) Introduction to k-nearest neighbors (knn) algorithm. Accessed on 2023-09-11. [Online]. Available: https://ai.plainenglish.io/introduction-to-k-nearest-neighbors-knn-algorithm-e8617a448fa8

[6] P. Kumar. (2021) Bagging vs boosting: The power of ensemble methods in machine learning. Accessed on 2023-09-11. [Online]. Available: https://pub.towardsai.net/bagging-vs-boosting-the-power-of-ensemble-methods-in-machine-learning-6404e33524e6

[7] Y. Regaya, F. Fadli, and A. Amira, "Point-denoise: Unsupervised outlier detection for 3d point clouds enhancement," *Multimedia Tools and Applications*, vol. 80, no. 20, p. 28161–28177, 2021. [Online]. Available: https://www.researchgate.net/figure/Isolation-Forest-learned-iForest-construction-for-toy-dataset_fig1_352017898

[8] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón, "Ugr'16: A new dataset for the evaluation of cyclostationarity-based network idss," *Computers & Security*, vol. 73, pp. 411–424, 2017.

[9] R. Magán-Carrión, D. Urda, I. Díaz-Cano, and B. Dorronsoro, "Towards a reliable comparison and evaluation of network intrusion detection systems based on machine learning approaches," *IEEE Access*, vol. 8, p. 112227–112247, 2020.

[10] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón, "Ugr'16 dataset - universidad de granada," https://nesg.ugr.es/nesg-ugr16/index.php, 2016.

[11] P. S. Foundation, "Python.org," https://www.python.org/, 2021, [Online; accessed 2021-08-10].

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "scikit-learn: machine learning in python — scikit-learn 1.3.0 documentation," https://scikit-learn.org/, 2021.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, p. 2825–2830, 2011.

[14] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 56 – 61.

[15] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://pandas.pydata.org/

[16] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[17] T. scikit-optimize contributors, "Scikit-optimize," https://scikit-optimize.github.io/stable/, 2023, accessed: 2023-08-09.

[18] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," 2012.

[19] JetBrains, "Pycharm professional," https://www.jetbrains.com/pycharm/, 2021.

[20] "Udc - citic – centro de investigación en tecnologías de la información y las comunicaciones," https://citic.udc.es/, accessed: 2023-08-09.

[21] "Github," https://github.com.

[22] "Microsoft teams," https://www.microsoft.com/en-us/microsoft-teams/group-chat-software.

[23] "Overleaf," https://www.overleaf.com.

[24] "Latex," https://www.latex-project.org.

[25] "arxiv," https://arxiv.org/.

[26] "Cherrytree," https://www.giuspen.net/cherrytree/.

[27] U. da Coruña, "Final project. mention in computer science," 2023, accessed: 2023-09-09. [Online]. Available: https://guiadocente.udc.es/guia_docent/index.php?centre=614&ensenyament=614G01&assignatura=614G01106&any_academic=2023_24&idioma_assig=&idioma=eng

[28] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 1–32, 2019. [Online]. Available: https://cybersecurity.springeropen.com/articles/10.1186/s42400-019-0038-7

[29] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," National Institute of Standards and Technology, NIST Special Publication 800-94, February 2007. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-94.pdf

[30] D. G. G'omez, *Sistemas de Detecci'on de Intrusiones*, July 2003, online; accessed 8-august-2023. [Online]. Available: https://dgonzalez.net/papers/ids/IDS_v1.0.pdf

[31] J. Fruhlinger, "The cia triad: Definition, components and examples," *CSO Online*, 2023. [Online]. Available: https://www.csoonline.com/article/568917/the-cia-triad-definition-components-and-examples.html

[32] Kaspersky, "Wannacry ransomware: What you need to know," 2021, online; accessed 1-September-2023. [Online]. Available: https://www.kaspersky.com/resource-center/threats/ransomware-wannacry

[33] "Openai chatgpt." [Online]. Available: https://chat.openai.com

[34] Coursera, "What is machine learning? definition, types, and examples," 2023, online; Accessed 02-09-2023. [Online]. Available: https://www.coursera.org/articles/what-is-machine-learning

[35] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, p. 273–297, 1995.

[36] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, p. 5–32, 2001.

[37] J. H. Friedman, "Greedy function approximation: A gradient boosting machine." *Annals of Statistics*, vol. 29, pp. 1189–1232, 2001. [Online]. Available: https://api.semanticscholar.org/CorpusID:39450643

[38] T. Chen and C. Guestrin, "XGBoost," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016. [Online]. Available: https://doi.org/10.1145%2F2939672.2939785

[39] J. Berkson, "Application of the logistic function to bio-assay," *Journal of the American Statistical Association*, vol. 39, no. 227, p. 357–365, 1944.

[40] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[41] E. Fix and J. Hodges, *Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties*. USAF School of Aviation Medicine, 1951. [Online]. Available: https://books.google.es/books?id=4XwytAEACAAJ

[42] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, p. 509–517, sep 1975. [Online]. Available: https://doi.org/10.1145/361002.361007

[43] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, p. 123–140, 1996.

[44] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, p. 119–139, 1997.

[45] R. M. French, *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*. Springer, 1992, p. 273–299.

[46] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608005800231

[47] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[48] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, p. 93–104, may 2000. [Online]. Available: https://doi.org/10.1145/335191.335388

[49] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, p. 1443–1471, 1999.

[50] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, "Kdd cup 99 data sets: A perspective on the role of data sets in network intrusion detection research," *Computer*, vol. 52, no. 2, pp. 41–51, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8672520

[51] M. Lincoln Laboratory, "1998 darpa intrusion detection evaluation dataset," 1998. [Online]. Available: https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset

[52] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "Kdd cup 1999 data," in *Proceedings of the Second International Workshop on Security and Artificial Intelligence*, 2009, p. 1–6. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[53] I. Perona, I. Gurrutxaga, O. Arbelaitz, J. I. Mart'ın, J. Muguerza, and J. M. P'erez, "gurekddcup data base," 2008. [Online]. Available: http://aldapa.eus/res/gureKddcup/

[54] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Intrusion detection evaluation dataset (iscxids2012)," in *Military Communications and Information Systems Conference (MilCIS), 2015*. IEEE, 2015, p. 1–6. [Online]. Available: https://www.unb.ca/cic/datasets/ids.html

[55] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications*, 2009. [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html

[56] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking,"

Proceedings of the ACM CoNEXT Conference (CoNEXT), 2010. [Online]. Available: http://www.fukuda-lab.org/mawilab/v1.1/2010.html

[57] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "The ctu-13 dataset. a labeled dataset with botnet, normal and background traffic," https://www.stratosphereips.org/datasets-ctu13/, 2014.

[58] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," *Information Security Journal: A Global Perspective*, vol. 25, no. 3, p. 125–126, 2016. [Online]. Available: https://research.unsw.edu.au/projects/unsw-nb15-dataset

[59] S. S. M. Aljawarneh, M. Aldwairi, and M. B. Yassein, "Machine learning in intrusion detection systems: A survey," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 1, p. 557–567, 2019.

[60] M. A. Ahmad, F. Alqahtani, A. A. Shah, F. A. Khan, M. U. R. Khan, J. Sher Khan, K.-A. Tait, and J. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *IEEE Access*, vol. 8, p. 132894–132917, 2020.

[61] G. Apruzzese, L. Pajola, and M. Conti, "The cross-evaluation of machine learning-based network intrusion detection systems," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 5152–5169, dec 2022. [Online]. Available: https://doi.org/10.1109%2Ftnsm.2022.3157344

[62] UCADatalab, "ff4ml: Free framework for machine learning," https://github.com/ucadatalab/ff4ml, 2022.

[63] I. Yilmaz and R. Masum, "Expansion of cyber attack data from unbalanced datasets using generative techniques," 2019.

[64] josecamachop, "Fcparser," https://github.com/josecamachop/FCParser, 2023.

[65] D. Powers, "Evaluation: From precision, recall and f-factor to roc, informedness, markedness correlation," *Mach. Learn. Technol.*, vol. 2, 01 2008.

[66] G. Lemaitre, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," 2016.

[67] S. Wankhede and D. Kshirsagar, "Dos attack detection using machine learning and neural network," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 2018, pp. 1–5.

[68] Y. N. Kunang, S. Nurmaini, D. Stiawan, and B. Y. Suprapto, "Attack classification of an intrusion detection system using deep learning and hyperparameter optimization," *Journal of Information Security and Applications*, vol. 58, p. 102804, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212621000430

[69] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "sklearn.neuralnetwork.mlpclassifier," https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html, 2011, accessed on 2023-08-31.

[70] ——, "Sklearn Voting Classifier," https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html, 2011, accessed on 2023-09-05.

[71] U. of A Coruña, "Teaching guide 2023/24 faculty of computer science," accessed on 2023-09-11. [Online]. Available: https://guiadocente.udc.es/guia_docent/index.php?centre=614&ensenyament=614G01&consulta=assignatures&any_academic=2023_24&idioma=eng

[72] E. Commission, "A european approach to artificial intelligence," accessed on 2023-09-11. [Online]. Available: https://digital-strategy.ec.europa.eu/en/policies/european-approach-artificial-intelligence

[73] M. de Ciencia e Innovación, "Inteligencia artificial," accessed on 2023-09-11. [Online]. Available: https://portal.mineco.gob.es/es-es/ministerio/areas-prioritarias/Paginas/inteligencia-artificial.aspx