GP Project

CSFC Department

Project ID: UPM-CSFC-F2022-01-Sem2

January 20, 2022

Machine Learning Framework for Intrusion Detection Systems

2022 – 2023

Dept. of Cyber Security and Forensic Computing

Faculty of Computer Science and Information Technology

University of Prince Mugrin, Madinah KSA

This research project is submitted to the department of cyber and computer science at university of prince Mugrin in order to fulfill the college graduation requirements.

Author(s):

- **Omar Belal**

E-mail: 3710137@upm.edu.sa

- **Rafif Altayar**

E-mail: 3910162@upm.edu.sa

- **Bayan Ghannam**

E-mail: 3810127@upm.edu.sa

- **Aghiad Massarani**

E-mail: 3810225@upm.edu.sa

- **Abdullah Basalama**

E-mail: 3810177@upm.edu.sa

Supervisor(s):

- **Mr. Mohamad Al Khatib**

E-mail: m.al-khatib@upm.edu.sa

- **Mr. Naveed Ahmad**

E-mail: n.ahmad@upm.edu.sa

Dept. of Cyber Security and Forensic Computing
Faculty of Computer Science and Information Technology
University of Prince Mugrin, Madinah KSA

Internet: http://upm.edu.sa
Phone: +966 014 - 831 8484

Intellectual Property Right Declaration

This is to declare that the work carried out under the supervision of Cyber Security and Forensic Computing with the title "Machine learning framework to detect network malicious traffic in cloud environments" in partial fulfillment of the requirements of the Bachelor of Science in Cyber Security and Forensic Computing is the sole property of the University of Prince Mugrin and the respective supervisor and is protected under intellectual property rights laws and convention. It can only be considered/used with the consent of the University and the corresponding supervisor for objectives such as extension for further enhancement, product development, adoption for commercial/organizational use, and so on.

This statement is applicable to all students and faculty members.

Date: 20/1/2022

Author(s):

Omar Belal Signature: OM

Rafif Altayar Signature: RT

Bayan Ghannam Signature: BG

Aghiad Massarani Signature: AM

Abdullah Basalama Signature: AB

Supervisor(s):

Mr. Mohamad Al Khatib Signature:

Mr. Naveed Ahmad Signature:

ABSTRACT

Machine learning algorithms are being consistently used to enhance intrusion detection systems (IDS) capabilities by applying several useful techniques that should be expected to allow high quality detection and classification methodologies for the corresponding network traffic in both host (HIDS) and network (NIDS) based systems. Yet researchers face different challenges while developing machine learning IDS systems as the surface of cyber network attacks is changing rapidly in today's age. This represents the challenge of finding a high-quality and updated analysis of newly created and publicly available malicious network datasets. In this study, a detailed analysis of three classification phases is performed on the CICIDS-2018 dataset (the dataset generated by Canadian institute researchers where the whole network structure is created on an AWS cloud service provider). Anomaly detection in one class classification phase provides comprehensive results using three different architectures of autoencoder (a type of anomaly detection method that utilizes the benefits of encoder and decoder functionalities). Denoising, Undercomplete, and Stacked All the autoencoder model's experiments ran with 50 epochs. In comparison with Autoencoder deep learning models, three classical machine learning algorithms have been created to ensure the usability of unsupervised learning and novelty detection in network traffic analysis using Isolation Forest, One Class SVM, and Local Outlier Factor (LOF). In terms of supervised learning, a comparison of five classical algorithms has been conducted to finalize the best and final estimator in the binary classification phase using algorithms represented as logistic regression, random forest, decision tree, and Catboost (a variation of the gradient boost algorithm). Final analysis was provided as a multiclass-classification phase that was performed only on the corresponding malicious attack labels with ignorance of benign network traffic. A total of five classical machine learning algorithms (XGboost, Random Forest, Decision Tree, SVM, and Naïve bayes) have been trained on the training malicious set only. All the experiments of binary, multiclass, and one-class algorithms were enhanced using two feature selection methods: filter correlation and selectKbest, in order to identify the final best predictors (features) from the whole dataset. Optimal model parameters were chosen for the Catboost estimator using Grid Search hyperparameter optimization. Through various experimental phases, it was confirmed that the Catboost algorithm performed well in comparison with all the classical and deep learning autoencoder models, with a recall and precision of 0.99 percent and an F1 score of 0.98. Finally, we have proposed a simple PoC script that performs a simple classification pipeline starting with the distinguishment of the network traffic from an input CSV file that will be generated from the CICFlowmeter tool (feature extraction tool). Our final binary estimator (Catboost) is going to classify the network packets (instances) as either malicious or benign. Then, if any malicious pattern is found in the CSV file, the pipeline will continue to the next multiclassification phase to reveal the attack type using the XGBoost algorithm.

Table of Contents

List of figures

List of Tables

CHAPTER 1

INTRODUCTION

1. CHAPTER 1: INTRODUCTION

In the age of technology and the explosion of information, it became common and important to use clouds to store data temporarily or permanently. Also, it became important to provide protection and continuous maintenance for these clouds and keep them secure against recent attacks that appear every day. One important part of the security process is to detect any malicious attempt to compromise and exploit any vulnerabilities that might be in the system. IDS devices have been brilliant solutions in this area, however attackers are evolving and using new techniques and inventing new zero-day attacks. Therefore, enhancing IDS capabilities is a must. This research is presenting a machine learning based detecting solution that will automate the network traffic detection process in different three phases of implementation.

1.1.PROBLEM DEFINITION

Throughout the history IDS systems have been used heavily by both companies and individuals to provide real-time monitoring and early detection of cyber threats to their networks; this process is applied by two types of detection mechanisms that could place in any IDS, one based on the attack signatures where the attack is matched based on a set of patterns identified in early stages and then alert will be generated to the user. The second type of detection is Anomaly detection-based methodology where the IDS will have a baseline that is set to identify any abnormal behaviors and distinguish between normal and malicious traffic such as Dos attacks.

An excellent solution centralizes on providing a new methodology for analyzing the patterns of the captured network traffic by utilizing machine learning classification algorithms that will classify network packets based on the net-flow most important features. Machine learning approaches are growing extremely fast in few years in most of the current domains that we see in our daily lives and will be very much affective in discovering new sets of attacks.

One main category of machine learning techniques is supervised learning, where the algorithm will learn from a past event that will be supplied as learning material to the

algorithms, this process could be so helpful in detecting new types of attacks that may go undetected in normal IDS.

Another type is unsupervised learning where the algorithm is going to learn from one set of behaviors and create a legit threshold for upcoming events to identify the anomalies, besides its capabilities to detect anomalies based on a set of criteria such as the behaviors and outliers in the data. Machine learning models could give us one-class, Binary, and multi classification based on the input labels that we give and the output that we desire starting as benign or malicious traffic, then classify the malicious traffic according to the attack type.

1.2.PROJECT DESCRIPTION

The following proposed system comprises two main phases.

1- In order to establish well-designed and functional machine learning algorithms that will detect intrusions, we start by collecting our data and finding the best-labeled dataset we could use in this research.

2- Second, we go through the collected data with different visualization methods that allow us to discover the data that we have and apply the necessary changes based on the requirements.

3- Third, we go through different machine learning algorithms and choose the best one based on the performance that we can take during the training and testing.

4- Second phase using machine learning will go through one of the most important security problems nowadays, which is network traffic fingerprinting where the network packet will be classified into many categories: going from the first one: BINARY classification problem where the network packet will be classified into two labels malicious and benign then convert to a full MULTI-CLASSIFICATION problem where several labels would be identified in advanced in the dataset, another stage that will allow the test of One class classification algorithms is creating one-class classifier from the benign traffic.

1.3.CHALLENGES

The most considered challenge that any machine learning project could phase is the dataset collection due to the lack of publicly accessible datasets or the quality of data that doesn't meet what we need. In order for this problem to be solved, we have chosen different machine learning IDS datasets that are mostly not so old and could be functional in this research: CICIDS-2018, CICIDS-2017, and UNW-2015.

This approach is pretty well and functional in our day-to-day work lives specifically for the well-known attack types, However, this solution could be vulnerable to new types of attacks that are generated and utilized constantly by threat actors such as Zero days attacks and APT groups.

1.4. CONTRIBUTION

This research provides a full detailed analysis of three classification phases of the malicious network traffic: one class, binary class, and multiclass classification, with a final comparison between all the used algorithms in this research through hypermeter optimization.

In addition, we go through multiple feature selection and scaling methods that will create a pretty well visualization of the most prominent features in the provided dataset. The following algorithms are applied to create the set of features: wrapper, filter, and K-best.

Finally, an implementation of a couple of deep learning models to create a final comparison between classical machine learning techniques and deep learning models. The following neural network structure will be mentioned: Autoencoder with its variation denoising, stacked, and undercomplete (simple); another model that proved its performance in intrusion detection [1] and research is Deep neural network DNN.

1.5. THESIS STRUCTURE

This research is constructed in five different Chapters:

- ***Chapter One: Introduction****:* that describes where the problem hits and what is the suggested solution, and what approach we are going to take in order to achieve the best possible solution.
- ***Chapter Two: Literature Review:*** this chapter presents a full overview of the background theory needed for this research, related existed knowledge about the problem, necessary tools, and methods that are going to be applied as a fulfillment of the research goal.
- ***Chapter Three: Research Methodology:*** the following chapter describes the design and processes of the implementation, the whole machine learning pipeline, and a quick look at the performance metrics.
- ***Chapter Four: Implementation:*** provides a detailed description of the implementation steps of this research from data preparation, pre-processing, visualization, and a final training and testing.
- ***Chapter Five: Result:*** full analysis of the datasets provided in previous chapters with a visualization of the final results of the machine learning models in different phases of implementation.
- ***Chapter Six: Conclusion:*** review the final research work with the final lessons learned and future development work.

CHAPTER 2

LITERATURE REVIEW

2. CHAPTER 2: LITERATURE REVIEW

2.1.IDS DATASET

Proposed tools and requirements are mentioned in [2] for creating a new dataset by generating encrypted network traffic in a real-world environment, where researchers presented a new dataset developed by generating encrypted network traffic in a real-world environment. The contributions of researchers are divided into two main parts. First, they have proposed several new crucial criteria for developing new datasets. Second, they built a new IDS dataset that includes encrypted traces of network traffic. Attacks such as brute force login and probing are labeled in the dataset. The background traffic (normal), ground-truth data, and packet traces with payloads are provided too. Using Zeek [3], an open-source network security monitoring tool, they retrieved and incorporated over 80 features from the CICIDS-2017 dataset for ground-truth, normal traffic, and malicious traffic.

In [4] they presented a phenomenon methodology for labeling datasets. If a supervised technique is selected, the lack of labels in data sets forces researchers to manually examine and attribute data according to the phenomena they are trying to model and identify network packets. Following [5] have proposed a detection system based on a collection of network traffic attributes to feed IDS that uses an ML model to detect malicious traffic. The ISOT-CID dataset was generated by Aldribi et al. [6]. Using Sklearn, Numpy, Matplotlib, and Pandas, the proposed model is prepared, created, fitted, and evaluated in Python. Their appealing model should be easy to build and fit in memory, so it can respond to network traffic features and forecast anomalies in real-time.

Datasets in this research context refer to the collection of huge network traffic pictured data that have been collected in the form of PCAP files, then were transferred to CSV files in the shape of rows and columns, where each column represent a specific feature or attribute. These collections of data are gathered and labeled in such a way that allows researchers to use them after cleaning them to train ML modules and to test their abilities to predict specific values that are of interest.

Datasets can be original (made by the researchers themselves) or taken from other people's work. However, it is really hard work to generate such a huge dataset that is big enough to train ML modules. This is the reason the researchers usually decide to find an online available dataset and use it in their research.

CICIDS2018

One of the most crucial factors of working with machine learning algorithms is the quality of the data that will be provided to the classifier, so it could be trained and give us the desired results we aim for to have a very well functional machine learning model. For the sake of this research, one of the recent, publicly accessible, and updated datasets was chosen as a source of data that will be input to the machine learning models. CICIDS2018 dataset is a representation of a cloud-based environment that was used to create and collect network traffic in distinct phases of attack scenarios, starting from network attacks, brute force attacks, and even web attacks.

CICIDS2018 is generated with the help of researchers in [7] where they have created a network subnet in AWS to launch cyber-attacks at different times and then collect the traffic using one of the most well-known network traffic collection tools: Wireshark.

The following categories were created to enhance machine learning research in the era of intrusion detection systems. Capturing data of these attacks was specified between 9:00 am on Monday third of July and 6:00 pm on Friday seventh of July where it ran for continues 5 days:

1- **Web attacks:** Researchers generated a handful sets of web attacks. The following attacks have been generated using one of the well-known vulnerable websites that are used for security testing Damn Vulnerable Web App (DVWA): SQL-Injection [7], Cross-Site Scripting (XSS), and Brute Force attacks on two main heavily used protocols FTP and SSH. All of these attacks have been automated using the selenium framework with the help of customized code that was implemented.

2- **Network attacks:** In terms of network attacks where most of the scenarios were generated, researchers have applied different variations of DoS attacks. Hulk,

GodldenEye, Slowloris, Slowhttptest, DDoS, PortScan, and Heartleech. All previous attacks took two days of the overall process targeting Ubuntu servers Apache and OpenSSL vulnerable versions.

3- **Infiltration attack:** Using a malicious payload integrated in Drobox software where it was installed in a windows vista machine; the attacker gained access and exploit the victim machine. Hence, the traffic collected at this point for a full two days.

4- **Botnet Attacks:** Specific malware called Zues was used to run in different versions of Microsoft windows. The malware was mostly launched by downloading facilities, with integrations of Ares botnet that will get back with some useful features and actions in the target machine, such as remote shell, screenshots, and persistence. Finally, the infected machine was asked for screenshots with the help of the infected zombies.

The full dataset was provided in the shape of CSV and PCAP files. Both PCAP and CSV 10 files were provided by the Canadian institute's official website. dataset features were extracted using a well-known feature extraction tool called CIC-FlowMeter. This tool will give a total of 85 statistical features extracted from PCAP network files. Some of the represented features are: flow duration, number of ACK flags, number of SYN flags, TTL of the network packet, and Windows length of the IP packet. The whole feature description is provided by the official website [8]

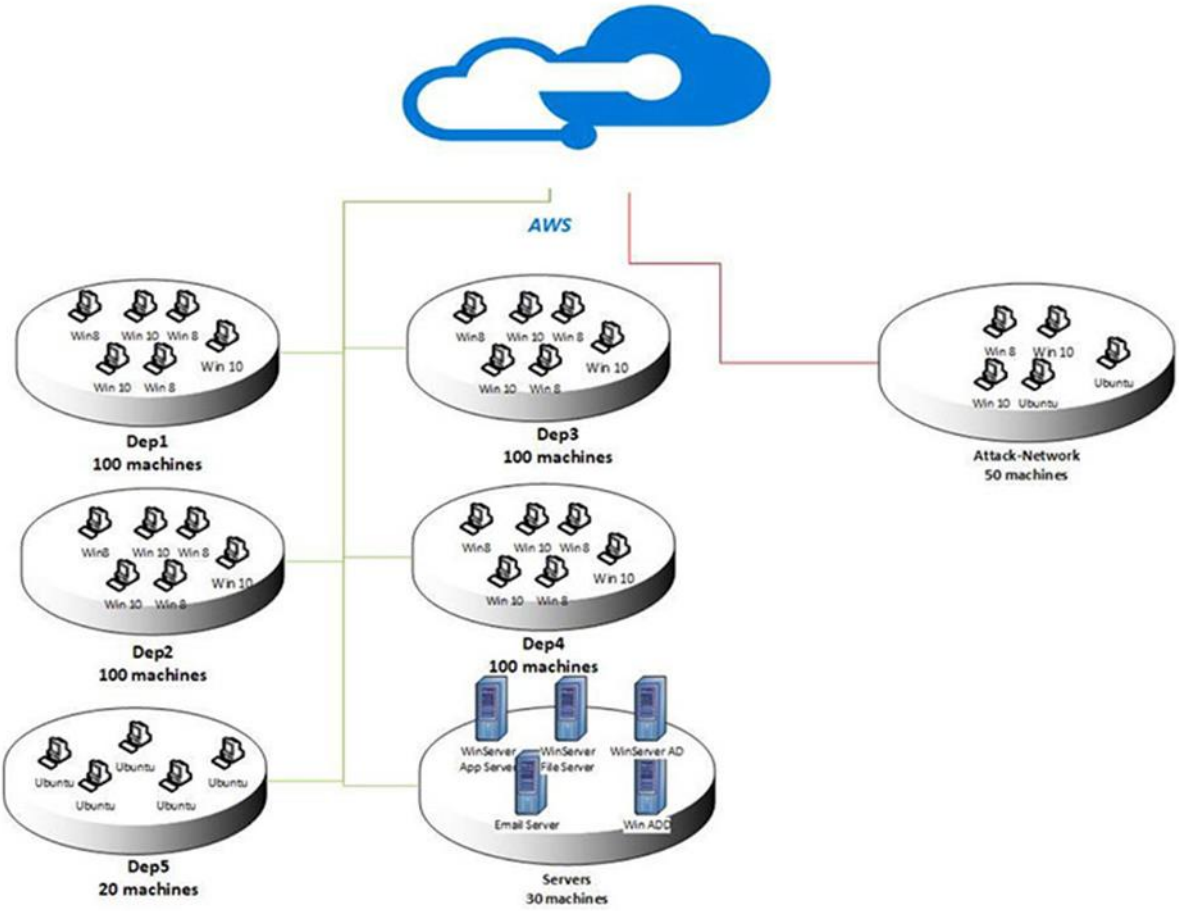

Figure 1 Network design of CICIDS-2018 on AWS

2.2.FEATURE SELECTION METHODS

Feature Selection is one of the core aspects of supervised learning that data scientists work hard to get right. As long as we have selected the best features that can go inside our model during the training process, we will not get many results. Feature selection ensures the quality and accuracy of our trained model. Multiple methods showed their effectiveness in this process such as SelectKbest, Filter, and Wrapper Methods. Each one has its own strength points that should be followed accordingly with respect to the needs.

SelectKbest:

SelectKbest is a class provided by scikit-learn that allows the extracting of the best features in the dataset with respect to value k. an important parameter is provided to specify if this problem is regression or classification. However, the default value is ‘f_classif which represents classification problems. Selecting the most important feature is an extremely critical process, especially when dealing with an exceptionally large dataset. This process limits the unused or less important inputs to the machine learning algorithms. Kbest is used for classification and regression problems.

Filter method:

An extremely useful and practical method is filtering. This method is mostly used in the pre-processing phase of the data. In this important method, the extraction of the most prominent features doesn’t rely on the algorithm itself, moreover on the correlation statics between the input features. This process is called Correlation Coefficients, and it has two main types: Continues and Categorical.

Wrapper method:

The Wrapper method works in a way where the method requires the model to be trained in different subsets of features and change from one to another and based on the previous subset performance we determine if the used features are good or not to go to the next iteration.

Some types of Wrapper Method:

Forward Selection

The model will go with multiple iterations. In the first iteration, it starts with no features, then with each iteration, a new feature will be added until it reaches a point where there is no improvement in the added features where it will stop.

Backward Elimination

As the name indicates we start the model from the end so the process will start with all the features and train the model, then with each iteration the less important features will be eliminated accordingly until it reaches a point where there is no improvement in the model's performance.

Recursive Feature Elimination

Using greedy optimization algorithms [9] this method aims to train the model with several iterations and multiple sets of features. However, the recursive method is going to identify the least performance features during the training and put them aside, then take the rest best features to the next iteration until all the features are finished. The method will specify the best features based on their order of elimination.

2.3.MACHINE LEARNING CLASSIFICATION ALGORITHMS

In creating the ML classification methodology, [10] proposed a full technical process ongoing through the traffic classification methodology by providing seven main steps to construct a network traffic classifier. The first step is to precisely state the classification objective. To achieve its goal, traffic classes can be divided into: Protocols, applications (e.g., Skype, WeChat, or Torrent), traffic types (e.g., browsing, downloading, or video chat), websites, user actions, operating systems, browsers, and so on. Hence, the goal is to categorize each flow with relevant traffic classes: The source IP, destination IP, source port, destination port, and protocol are commonly used to determine a flow, a huge and ground truth dataset, Data cleaning and preprocessing, feature categories, model selection, Training & Validation (tuning), and finally a periodic evaluation for the models.

This research objective centralized into working with different classification algorithms in three distinct phases: one class, binary class, and multiclass classification with each class have it is own feature and importance to get the most out of the data and the machine learning algorithms.

- **One Class**:

Several algorithms are utilized heavily on one class classification where the phase helps data scientists to work with one class of the given dataset with the elimination of other classes in the training phase then apply both classes by taking instances from both sides and feeding them to the classifier. This method helps researchers in different use cases, such as working with imbalanced datasets. We could train the classifier on the most available category in the distribution and then apply both unrepresented and overrepresented categories in the test phase. Another useful use case is discovering new classes that were not classified or identified before by looking for the abnormal behavior that the classifier has not trained on or by simply collaborating with outliers that are available in the dataset. The detection of such outliers could help in seeing the given data in new ways. This phase is mostly used in anomaly detection with several types of machine learning use cases, and it has had a fairly good result so far.

Some useful algorithms that are used in one class classification are OneClassSVM which is a sub model that is generated from the standard SVM classifier, local outlier factor, and Isolation Forest.

- **Binary Class:**

Instead of utilizing only the benign traffic as is the case in anomaly detection, we are going to use all available classes in the dataset and combine all similar into two types so all the attacks classes going to be as only (attack) despite the type of attack itself as it is not important to know the attack type in this specific phase. The only essential information that we need to detect/Classify is what type of traffic this traffic is, benign or attack? To do this

efficiently, several machine learning algorithms utilized for classification are used, such as Logistic regression and Decision tree.

- **Multi-Class:**

Working with multiclass is a bit of a nicer challenge, as we will be able to actually see which type of attack traffic is detected. However, our main target here is only looking for the attack traffic and then dissecting all the available types/classes, so there will be no benign or normal classes in this phase. Similarly, several classification algorithms could utilize as it is the case of binary classification, such as: Random Forest, XGboost, and Gradient Boost.

2.4.PERFORMANCE METRICS

After finalizing the training and validation processes of the machine learning models, several metrics are used to see how actually the model performed during the training and validation (where the model sees the new data that was never shown in the training process). let us see some of the practical metrics in this project.

Accuracy: The Most well-known metric that uses mostly in machine learning projects due to its ease of understanding and implementation, it is actually a particle metric used to identify the total number of the correct predictions divided by the overall number of predications that are done by the model.

Accuracy

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

However, accuracy could be the ultimate solution with respect to the severe imbalance of the datasets such as our project here.

Another useful metric for model evaluation and finding all the data points in the dataset is **Recall**, which is the ratio of the truly identified class divided by the sum of the correctly predicted and the wrongly predicted. To make it easier, it is the total of the truly predicted over the whole true occurrences of that class. High recall means low false negatives in the prediction process.

Recall

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

Precision is the ratio of the truly predicted divided by the sum of the truly predicted and the wrongly predicted as positive, where it is actually negative. High precision helps us determine that there are small numbers of false positives in the prediction process.

Precision

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

F1 score is the overall combination of the recall and precision results, which helps us define a balanced evaluation of the classifier as both the precision and recall give us important information.

F1 Score

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

Another important and extremely useful matrix that could be created after the model validation is the **confusion matrix**, which provides a clear visualization of the model performance by taking a closer look into the actual and the predicated classes in case it is a binary classification problem. All the four categories of FN, FP, TP, and TP will be clearly identified in two modes either without normalization where the actual numbers are shown in each cell in the table, or with normalization where a percentage is presented instead.

Confusion Matrix

True/Predict	Predicted as Attack	Predict Benign
Truly identified as attack	True Positive (TP)	False Negative (FN)
Truly identified as Benign	False Positive (FP)	True Negative (TN)

Table 1 Confusion Matrix of binary classification

2.5.SUMMARY

In this chapter, we have clearly identified our background knowledge that is required to implement and deploy this project, written plan of how to deal with the available data, and through different classification phases. Finally, we showed the evaluation metrics that are used to measure our model performance and usability.

CHAPTER 3

RESEARCH METHODOLOGY

3. CHAPTER 3: RESEARCH METHODOLOGY

This chapter is going to discuss the Main Tasks of the project requirements, tools of implementation, development environment, and all required utilities to make this project a success.

3.1.PROJECT REQUIREMENTS

The objective of this project was determined by the following characteristics to ensure useful participation in cyber security and data science communities:

- Building several machine learning classification algorithms in three different phases: one, binary, and multiclass. Then finalize the best performance model of each phase based on the precision, recall, and f1 scores.

- Enhance the implementation feasibility of using different deep learning models on provided network traffic dataset that is represented in tabular CSV format.

- Create a decent PoC for presentation and showcase purposes for future work and use cases.

Based on our above specific objectives, we construct a full requirement in table 2 that will specify and sort all project tasks based on their priority and difficulty in the different implementation phases. The difficulty measure was defined based on the current knowledge of the group by answering the question of ‘Do we know how to perform the following task?’ in terms of the importance of the assigned tasks we have measured the priority level based on the importance on the specific task and relevance to the overall project.

No.	Task	Priority	Difficulty
1	Search for high-quality updated IDS Dataset.	High	High
2	Perform Preprocessing Steps.	Medium	Low
3	Explore the Dataset using Exploratory Data Analysis (EDA).	Medium	Low
4	Perform Filter Method for Feature Selection.	Medium	Medium
5	Build Six Binary Classification algorithms.	High	High
6	Build five Multiclass Algorithms.	High	High
7	Create two phases of Anomaly detection using 3 ML algorithms and 3 variations of the Autoencoder Neural network.	High	High
8	Evaluate each model with its phase on tasks 5, 6, and 7 to get the best out of each phase.	Medium	Low
9	Perform the feature Importance method to choose the predictor's features.	Medium	Medium
10	Create Hyper meter optimization on the final estimator.	Medium	High
11	Create a PoC script that links both multiclass and Binary class best performing models then predict input network traffic according to the two, first if malicious or benign then if malicious predict which type of attack.	Medium	High

Table 2: Project Requirements List

3.2.IMPLEMENTATION TOOLS

Choosing the best tools for this project is an essential task to have the best output of this project to ensure the high efficiency and effectiveness of this project. The tools could be represented as libraries, development environments, and data used as the final dataset.

3.2.1. PROGRAMMING LANGUAGE

In order to build and maintain this research, we have used one of the best-considered programming languages for data science: Python. Python provides an easy, effective, and quick development approach with huge capabilities and support from the Tech and Machine Learning communities. However, it took some time to understand, build, and get familiar with the language, then things get easy.

3.2.2. DEVELOPMENT ENVIRONMENT

Jupyter notebook was the optimal solution for the development environment due to its easy configuration and use. Jupiter notebook provides a high-efficiency testing environment to run the code blocks individually in jupyter cells. This specific approach is superior to machine learning, so each task or face could be done individually without the need to run it again each time. In addition, it provides a high level of visualization that makes shaping and plotting graphs and tables much easier, especially during the Exploratory Data Analysis (EDA) Phase.

3.2.3. LIBRARIES

Several Data Science Libraries have been used in this project and all were utilized in python programming language. For the sake of data preprocessing and cleaning, Pandas was super useful in slicing, discovering, and editing the data frame stored in the jupyter notebook memory. Another helpful library in parallel to pandas is NumPy, which provides the data mostly in an array type format. One useful application of NumPy while normalizing the dataset into a specific range of values using one of the scaling methods. However, it was used much less than pandas due to the easy visualization of the panda’s data frame and the default tabular format of the data.

For Data visualization and providing high-quality figures and statistics of the dataset labels, features, and records, Seaborn and matplotlib were used to create a high-level

understanding of the data and the features used in the machine learning models training and testing.

The primary library used to build, train, and test the machine learning algorithms is Scikit learn. By providing several methods and functions Scikit learn helps in the automation process of several states of ready-use Machine learning algorithms that could do several tasks such as Regression by predicting numeric continuous values, Clustering and getting data in groups based on their similarities, and Dimensionality reeducation to reduce the number of given features in the dataset to the best minimum performed ones. In addition, it provides full preprocessing of the given data by performing normalization and feature extraction, and model selection by comparing, validating, and choosing the best parameters and models. Scikit learning capabilities are huge and super useful. Sklearn utilizes high computational power depends on the classifier that is being used in the model training process.

3.3.APPROACH DESIGN

The Project is designed in three main phases of development; the initial phase is a representation of a binary classification problem where the network traffic is given as an input in CSV format and the output is given by the binary classifier as malicious or benign.

The next face will utilize the multi-class classification that takes the malicious traffic determined by the binary classifier and classify it based on the Cyber-attack type.

The final Main Phase is an unsupervised learning approach that will take only one type of class and predict other classes based on the differences or the abnormal new situation of the given data. Specifically, benign network traffic will be the only input to the classification algorithms during the training phase, then both benign and malicious data going to be combined and fed to the classifier in the testing phase to predict the anomaly in that case. We are going to conduct two methods: one using Sklearn anomaly detection algorithms, and the second one using Autoencoder neural network architecture by using three structures of the Autoencoder. fig 1 presents the required phases.

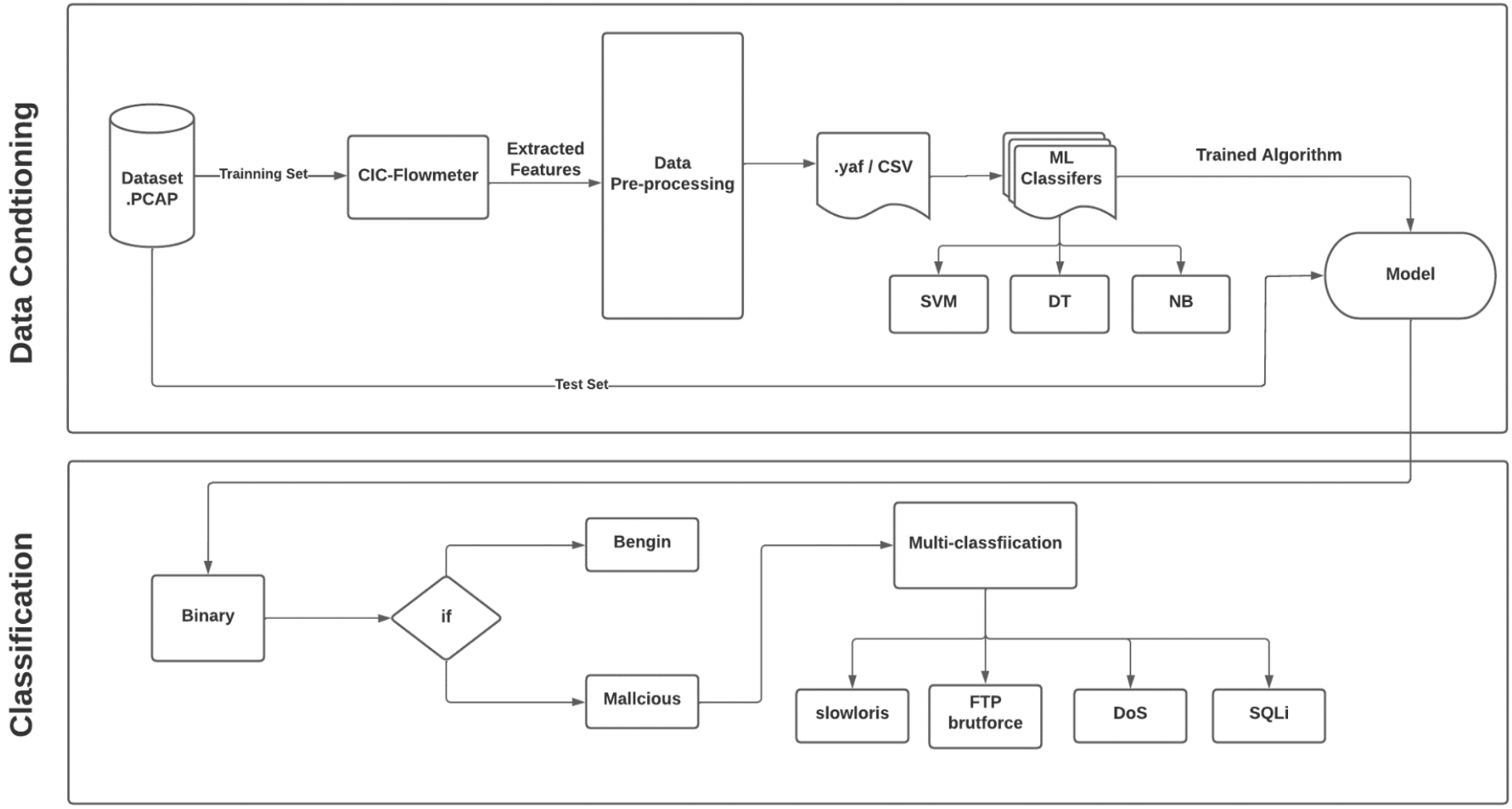

Figure 2 : Full Deployment Design

3.4.SUMMARY:

In this chapter, we have discussed all the important milestones of the project, and all required and essential tools in all development phases, whether it is in the preprocessing, data visualization, or model training. Finally, we proposed our implementation design and the followed approach that is going to be described in detail in the following chapters.

CHAPTER 4

IMPLEMENTATION

4. CHAPTER 4: IMPLEMENTATION

In order to get a high-level overview of the project implementation steps, the project will be divided into eight different milestones, starting from data collection, data preprocessing, EDA, feature selection methods, binary classification, multiclass classification, anomaly detection, and finally a model deployment for the sake of proof of concept (POC).

4.1.DATA COLLECTION

Datasets are always an ongoing challenge for data science and Subject Matter Experts (SME) in the research community, and that's due to several issues that are constructed with data. For example, the data must be publicly accessible to the research community and have no privacy-violating aspects. It must also have good quality, and it would be a perfect choice if it was collected in the same deployment environment.

To solve this problem, we have selected two good, recent, and high quality datasets provided by the Canadian Institute for Cyber Security [11]: CICIDS2017 and CICIDS2018. Both datasets represent the same high-level categories of network attack traffic, except the collection environment is different. The 2018 version was collected in various AWS subnets in various attack and normal behavior scenarios, whereas the 2017 version was collected in a regular LAN network with nearly identical labels. CICIDS-2018 Class Entries are shown in Table 3.

	label
Benign	13484708
DDOS attack-HOIC	686012
DDoS attacks-LOIC-HTTP	576191
DoS attacks-Hulk	461912
Bot	286191
FTP-BruteForce	193360
SSH-Bruteforce	187589
Infilteration	161934
DoS attacks-SlowHTTPTest	139890
DoS attacks-GoldenEye	41508
DoS attacks-Slowloris	10990
DDOS attack-LOIC-UDP	1730
Brute Force -Web	611
Brute Force -XSS	230
SQL Injection	87

Table 3 CICIDS 2018 Dataset Entries

The distribution of attack classes in both datasets is shown in the following table by the number of records in each label. It is super clear that the dataset is severally imbalanced and skewed in favor of the benign traffic with the highest record among all given labels. The percentage of the benign traffic in the dataset is almost 83% of the whole labeled combined. This information, determined by our dummy classifier that has the most frequent strategy assigned, gives us that the benign class is the most frequent class in the dataset.

	precision	recall	f1-score	support
0	0.83	1.00	0.91	1348471
1	0.00	0.00	0.00	274824
accuracy			0.83	1623295

Figure 3 Result of the dummy classifier

4.2.PREPROCESSING

Data cleaning, also known as data preprocessing, is typically the first step in any machine learning project, particularly when dealing with classical algorithms. This phase is divided into multiple tasks that we have followed in order to achieve a high-quality and clean dataset:

1- Missing data: usually after extracting the dataset from raw PCAP files to tabular format data (CSV), there will be some missing values present in the CSV files such as Null, Inf, and NaN values; to deal with this issue, we have decided to do two main operations dealing with it. The first one will be dropping the rows that contain missing data if the number of missing records isn't huge. The second option is replacing all missing values from the same types by the overall mean of the column. In most cases, it will not be a problem to delete the mentioned missing records if the dataset is huge enough, but the mean value will be a good choice when dealing with the features directly.

2- Feature Variances: Feature Variances: We have searched for the available features that have zero variance determined by a certain threshold defined earlier. By default, this approach will remove all features that have the same kind of values in all the given records.

3- Undesired Features: some of the given features will have no additional impact in the training process, such as the timestamp (date) of the instance and the destination port, where this specific feature has lots of string data contained and it will be of no use in the training process.

4- Negative Features: Negative values in the dataset could easily affect the training process. To deal with this problem, we will again replace all negative values with their mean by iterating over the whole dataset and detecting any negative record in the dataset.

5- Finally, before proceeding with training the machine learning classifiers, the dataset has to be splitted into X and Y, where X will contain all the training numeric features that will be used further to implement a specific feature selection method and identify all the necessary features only to pass them later to the training phase. Y mostly will only contain the dataset labels that will be parsed into numerical labels by label encoding or cat codes methods, both of which are going to be fine. The following is a small example of the dataset split into two data frames.

```
In [16]: # load the dataset and split it to X-training , Y-labels
         X = df.drop(columns=['label', 'binary_class'])
         y = df[['binary_class']]
```

Figure 4 Dataset Split into X and Y

4.3. EXPLORATORY DATA ANALYSIS

Data visualization is such an important task in machine learning and data science projects in general that it will allow the researchers to have a crystal-clear view of the dataset that they are dealing with. The following checklist has been followed in order to get the best visualization of the CICIDS2018 dataset:

1- Check feature datatypes: to see if the dataset contains any object or string features that could cause errors later in the training process.

2- Dataset Labels: Having a labeled dataset is a super useful advantage in any AI project that will make things much easier. The following fig 5 presents the distribution of each label in the CICIDS2018 dataset.

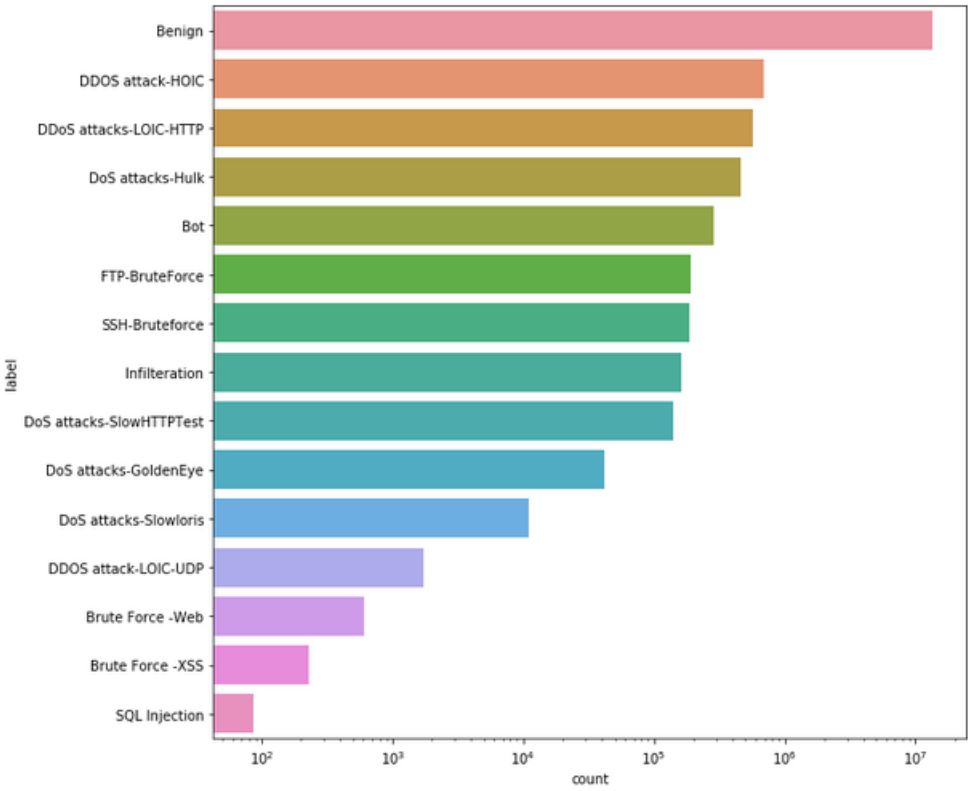

Figure 6 ration of All class labels CICIDS-2018

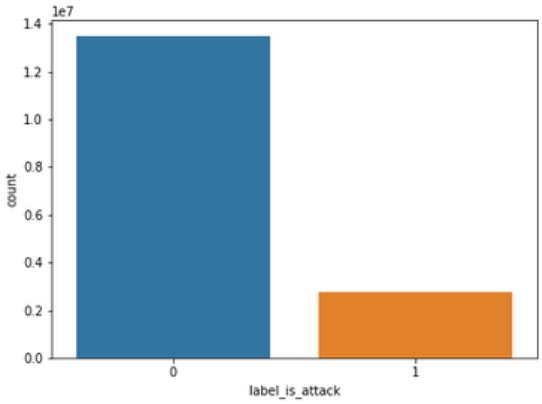

Figure 5 binary class distribution

3- Fig 6 presents the distribution between benign and attack traffic only in the CICIDS2018 dataset.

4- The distribution of all attack types only, which is a useful information that will help us during the multiclass classification as there will be no benign traffic.

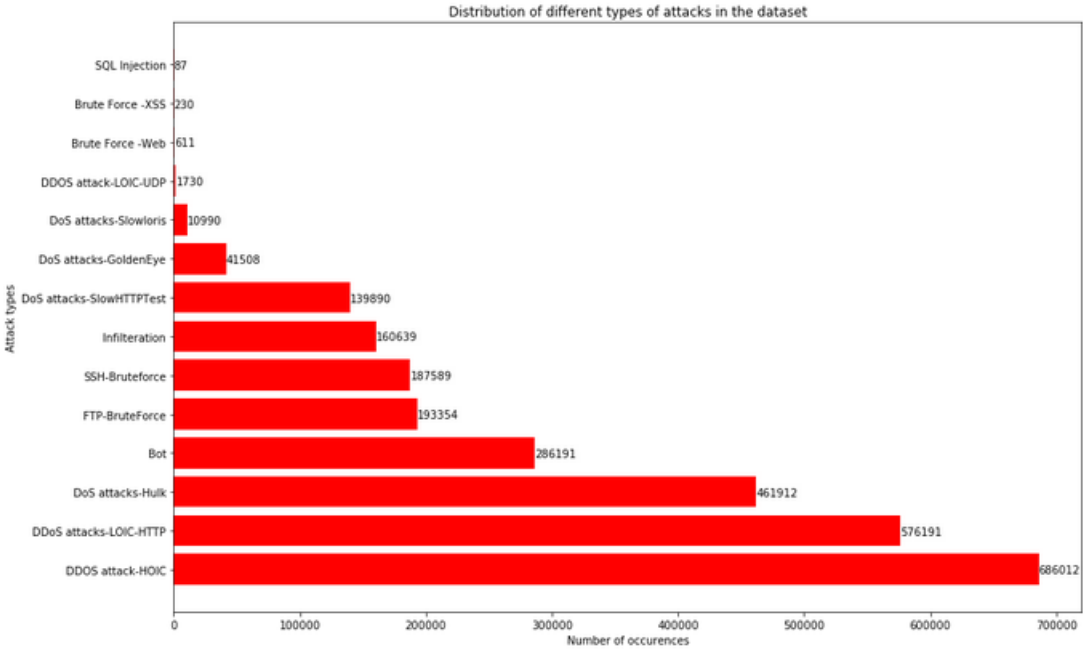

Figure 7 ratio of the Attack traffic types

5- Feature Correlation: A super useful way to create a clear visualization of the feature correlation is by using a correlation heatmap. The feature heatmap shown in fig 8 is conducted to capture feature correlation between each other and the attack labels in the dataset.

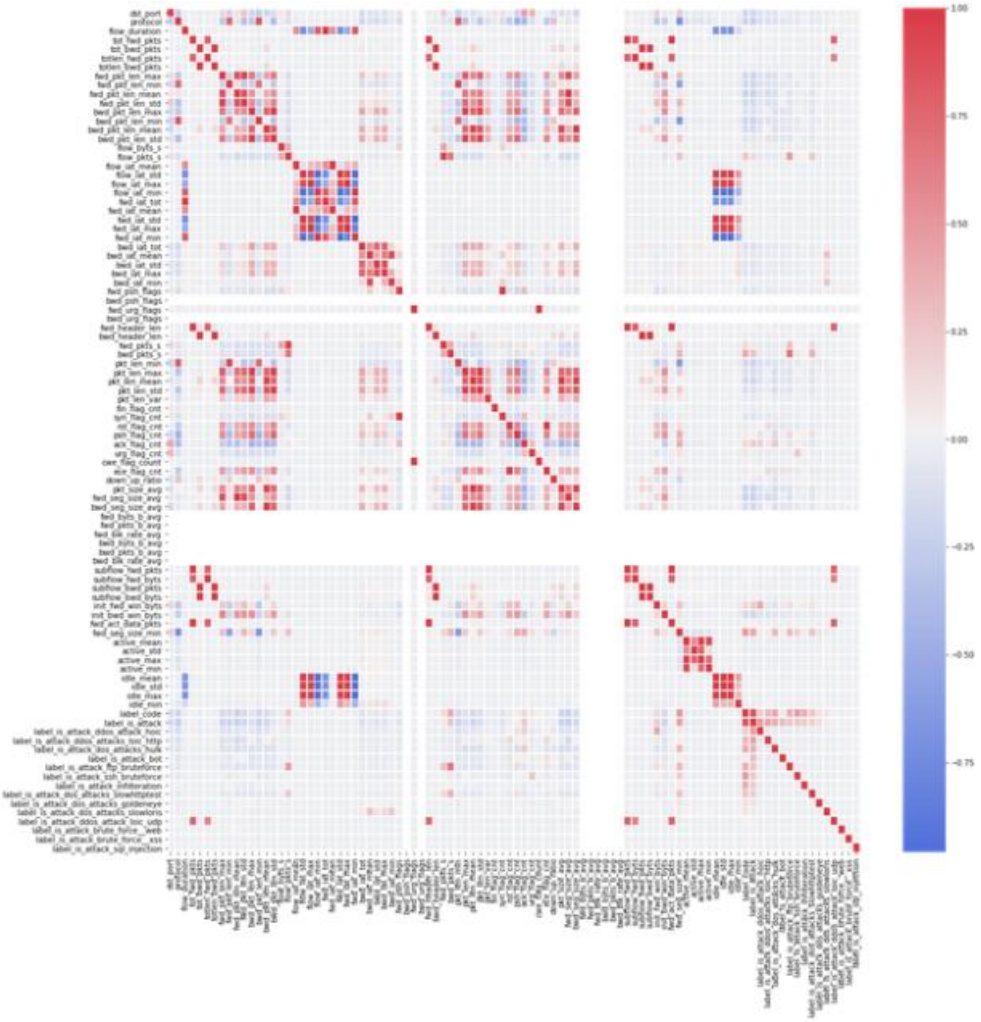

Figure 8 Feature Correlation Matrix with al labels

fwd_urg_flags	cwe_flag_count	1.000000
totlen_fwd_pkts	subflow_fwd_byts	1.000000
fwd_pkt_len_mean	fwd_seg_size_avg	1.000000
bwd_pkt_len_mean	bwd_seg_size_avg	1.000000
tot_bwd_pkts	subflow_bwd_pkts	1.000000
tot_fwd_pkts	subflow_fwd_pkts	1.000000
fwd_psh_flags	syn_flag_cnt	1.000000
totlen_bwd_pkts	subflow_bwd_byts	1.000000
flow_iat_min	fwd_iat_min	0.999996
flow_iat_max	fwd_iat_max	0.999994
rst_flag_cnt	ece_flag_cnt	0.999987
flow_duration	fwd_iat_tot	0.999986
flow_iat_std	fwd_iat_std	0.999981
flow_iat_mean	fwd_iat_mean	0.999963
subflow_fwd_pkts	fwd_act_data_pkts	0.999189
tot_fwd_pkts	fwd_act_data_pkts	0.999189
tot_bwd_pkts	bwd_header_len	0.997798
bwd_header_len	subflow_bwd_pkts	0.997798

5-1 high correlation: By taking a quick look into fig 9, we can determine that there is some high feature correlation between assigned features and by plotting the most correlated features by a 0.8 threshold, we can see all correlated features that exceed this baseline. These features are going to be possible to drop before the training process, which aims to ensure a better result and high quality.

Figure 9 Highly correlated features p > 0.8

5-2 features correlated with the binary classification labels, which could be a useful indicator of these features to have a good impact in the predication process:

Feature name	**Feature Description**	**Correlation with the binary labels**
fwd_seg_size_min	Minimum segment size observed in the forward direction	**0.429296**
bwd_pkts_s	Number of backward packets per second	**0.260203**
ack_flag_cnt	Number of packets with ACK flag	**0.220465**
bwd_pkt_len_min	Minimum size of packet in backward direction	**0.218390**
fwd_seg_size_min	Minimum segment size observed in the forward direction	**0.148600**

Figure 10 Features correlated with binary classification

4.4.FEATURE SELECTION

By using the CICFLOWMETER tool, the researchers extracted eighty-five features from the collected PCAP files so they could be utilized in the ML models. Table 5 gives a clear description of each feature extracted from the PCAP files to CSV useable files.

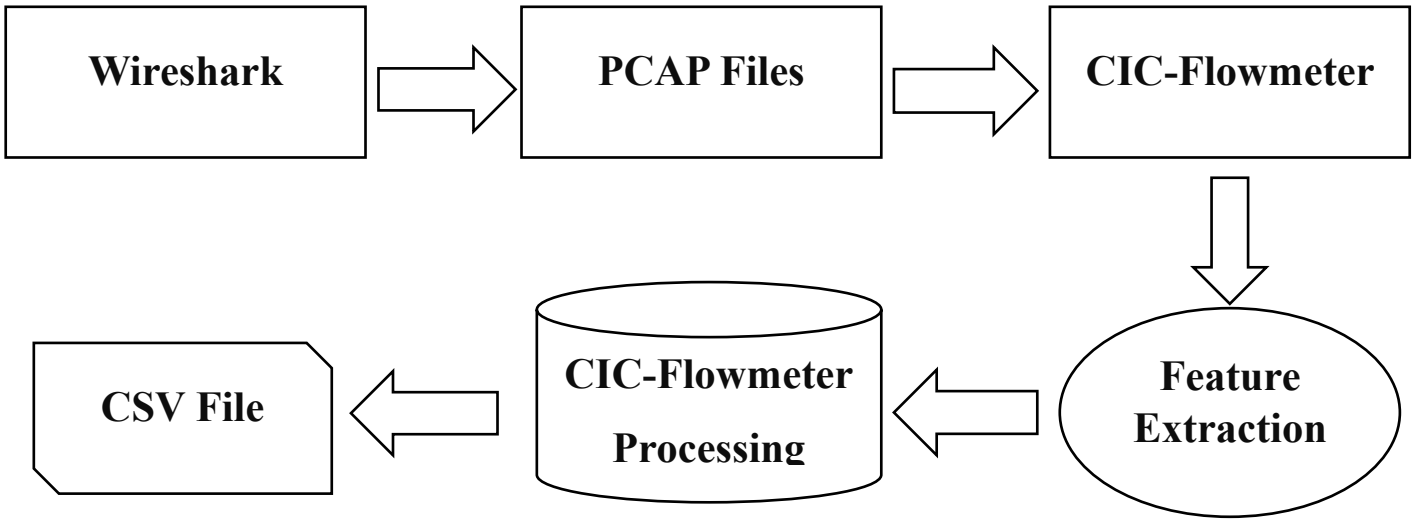

Figure 11Features extraction process

Feature	Feature Description
fl_dur	Flow duration
tot_fw_pk	Total packets in the forward direction
tot_bw_pk	Total packets in the backward direction
tot_l_fw_pkt	Total size of packet in forward direction
fw_pkt_l_max	Maximum size of packet in forward direction
fw_pkt_l_min	Minimum size of packet in forward direction
fw_pkt_l_avg	Average size of packet in forward direction
fw_pkt_l_std	Standard deviation size of packet in forward direction
Bw_pkt_l_max	Maximum size of packet in backward direction
Bw_pkt_l_min	Minimum size of packet in backward direction
Bw_pkt_l_avg	Mean size of packet in backward direction
Bw_pkt_l_std	Standard deviation size of packet in backward direction
fl_byt_s	flow byte rate that is number of packets transferred per second
fl_pkt_s	flow packets rate that is number of packets transferred per second
fl_iat_avg	Average time between two flows

Table 4 features description

As we have described in chapter 2, feature selection methods allow us to get the best number of features that will have the highest impact on the machine learning models. In later stages, we have selected three of the top methods that are used for dimensionality reduction and feature selection.

4.4.1. SERLECTKBEST

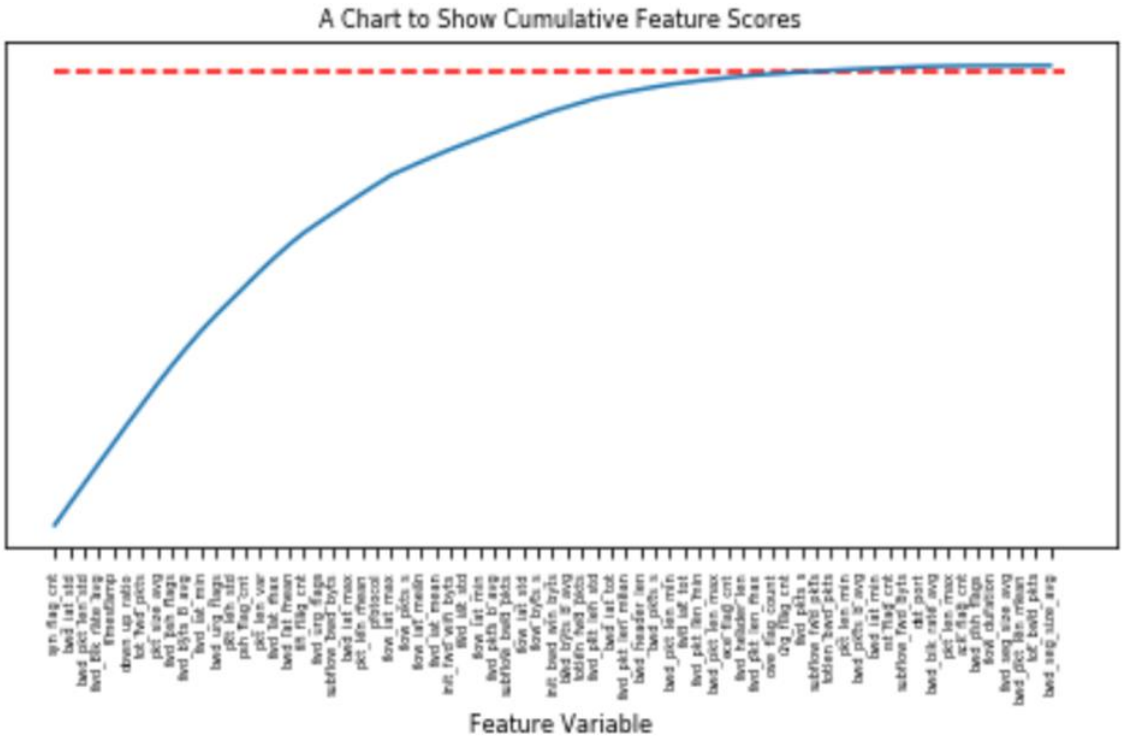

Figure 12 Best selected features using K = 40

Fig 10 shows a clear visualization of the best selected 40 features by the identifying k value with k = 40 and Score_fun = chi2, which is a unique parameter used for non-negative features in multiclass classification problems. The selected features have been selected according to the k score.

4.4.2. FILTER – CORRELATION

The filter method provides a unique implementation of the correlated features by plotting the spearman heatmap and all super red or super blue cells show high correlated that will be between -1 and 1.

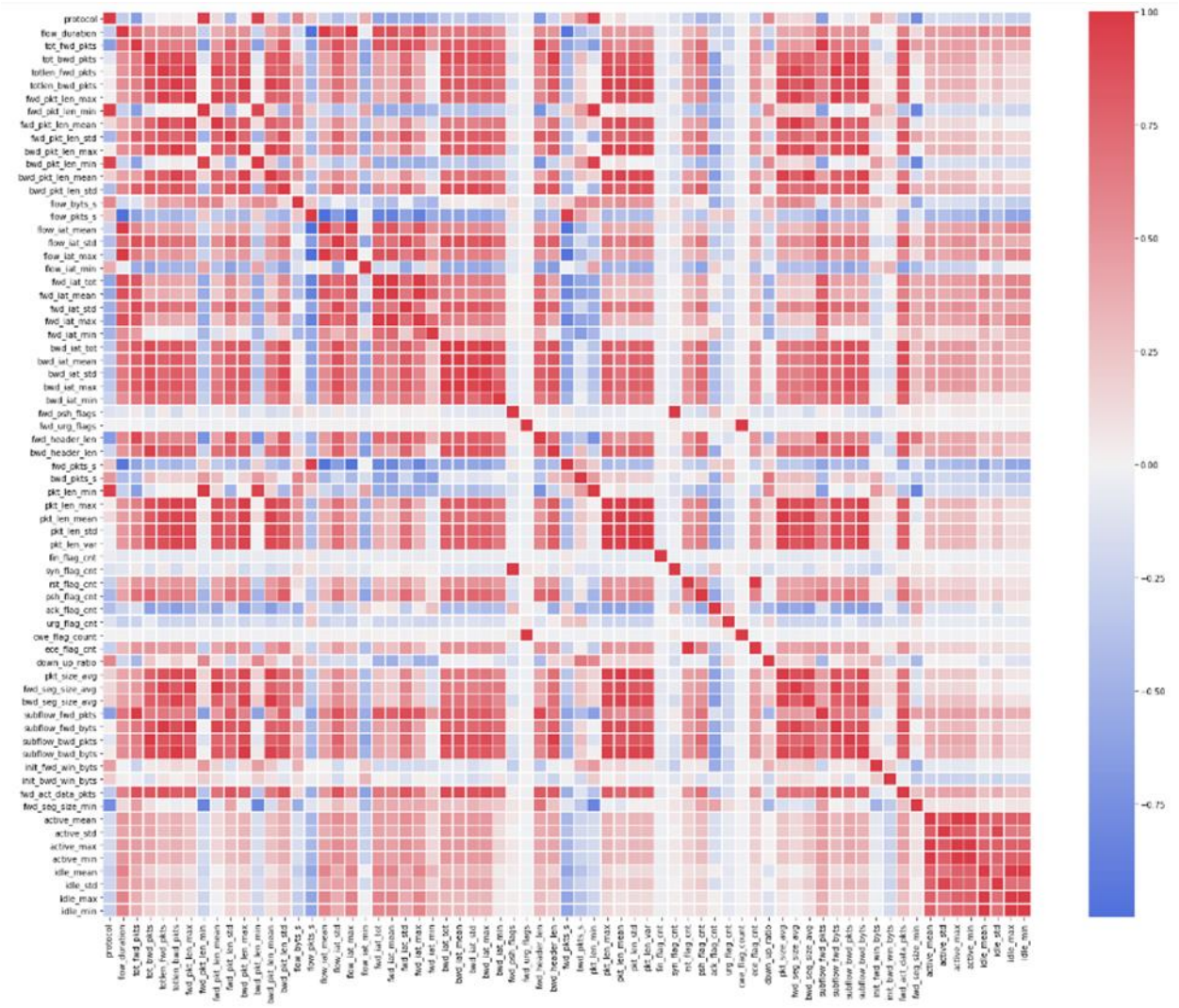

Figure 13 spearman Features correlation matrix - No labels

To ensure a high-quality decision is taken, we have iterated only on the selected features with low correlation and the rest are dropped, so the whole number of eighty features used in this dataset dropped to only thirty columns for further training and evaluation on

different classical algorithms. Fig. 13 shows an updated heatmap of the features after dropping the corelated ones.

The below heatmap represents the correlation after removing all the correlated features by iterating over the whole dataset columns and only selecting certain selected features to use in the training process.

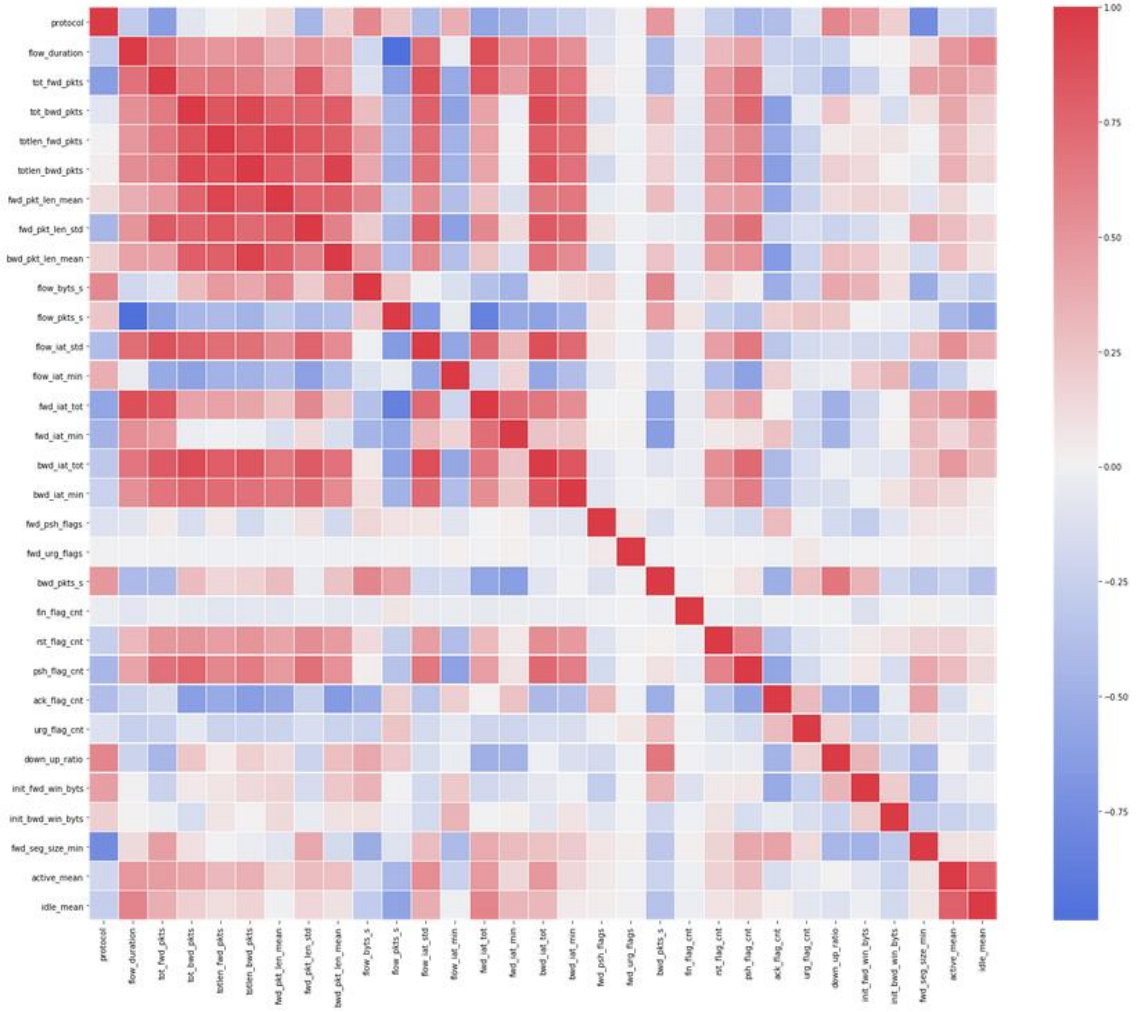

Figure 14 Updated Heatmap - after removing correlated features

4.5.DATASET SPLIT

For dataset splitting, the dataset has been split into most classification phases as 80/10/10, where 80 percent for the model training data is a set that is for the model training only by fitting the predefined parameters to the machine learning model, 10 percent for validation is a set of new data that the model has never seen before during the training and it will provide an unbiased assumption of the model performance, and the last 10 percent for final testing of the model. This set is also never seen by the model in the training phase, so it will be utilized for unbiased checking on the final fitted model after we finalized our parameter optimization of the evaluation sets.

```
In [28]: # Split X frame into training , testing , and evaluating sets
         # 0.8 training , 0.1 testing , 0.1 evaluation
         # split done with the feature "protocol" being one hot encoded in the training a

         X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                             test_size=0.2,
                                                             stratify=y ,
                                                             random_state = 101 )

         X_val, X_test, y_val, y_test = train_test_split(X_test, y_test,
                                                         test_size=0.5, stratify=y_test)

         # on hot encoding for the feature protocol cuz it is categorical
         X_trainh = pd.get_dummies(X_train, columns=['protocol'])
         X_valh = pd.get_dummies(X_val, columns=['protocol'])
         X_testh = pd.get_dummies(X_test, columns=['protocol'])
```

Figure 15 Dataset Split into train, validation, and test

It is also a good practice to split the data into 80 percent for the training and 20 percent for testing and evaluation. However, it always depends on the number of records and the types of data.

4.6. CLASSIFICATION PHASE

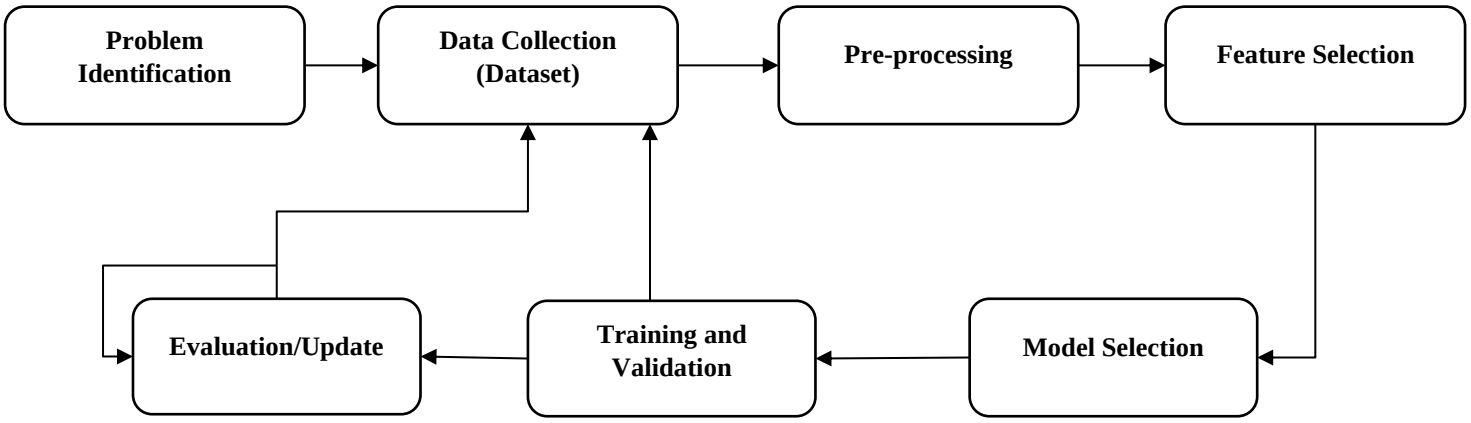

Figure 16 Building Classification Models Process

The following section describes all three phases of the machine learning algorithms. Following implementation will highlight the creation of the best performing algorithms in each phase, and a final estimator will be chosen from each phase for future deployment.

It is useful to note that our main target here is to perform multiple algorithms and follow a try-and-error approach to produce the best result possible from the running experiments.

Most of the classification algorithms are assigned to do only one type of classification unless some special parameter is passed to ensure the right type of classification. Another major component that must be clear to the machine learning classifier is the labels assigned in the training and testing, which must be either binary for a binary classifier or multiclass in terms of a multiclassification classifier. In a final stage, if the target experiment is anomaly detection, we will only apply one class of the labels during the training process where there is mostly benign traffic. Both malicious and benign traffic will be passed during the evaluation phase later on.

Figure 15 demonstrates the whole process of creating a highly effective and practical model that provides the best rate of accuracy as possible, starting with defining the overall problem; data collection and dataset traffic generation will be conducted according to that

specification; then preparing the dataset with different actions, starting from the preprocessing phase to the feature selection and extraction; and finally training and evaluating the model with periodic evaluation methodology to ensure effectiveness and the highest level of accuracy.

Below is a following list of some unique classifiers that we are going to use in different classification phases where they give the highest accuracy during both training, evaluation, and testing phases.

4.6.1. BINARY CLASS MODELS

In general, binary classification algorithms represent a quick and precise insight of the required precision. In this specific problem, we are targeting binary prediction models that will give us a precise result of the input traffic to know whether it is benign or suspicious traffic. Moreover, this is the first practical face of our project pipeline that will come up as the first output to the user, and it could be implemented in different machine learning algorithms to produce the best high-performing model.

4.6.1.1. Logistic Regression

The method of modeling the likelihood of a discrete result given an input variable is known is logistic regression. Most frequently, logistic regression models used for a binary classification problem, which might be true or false, yes or no, and so forth. Multinomial logistic regression can be used to model situations with more than two discrete outcomes. Logistic regression is a handy analytical tool for determining if a fresh sample fits best into a category in classification tasks. Because components of cyber security, such as threat detection, are classified issues, logistic regression is a valuable analytic tool.

4.6.1.2. Support Vector Machine (SVM)

Data points are represented as vertices in an n-dimensional space in the SVM, with "n" denoting the total number of extracted features. Each feature's magnitude is given by the position of a certain coordinate [13]. Estimating the ideal hyperplane, which effectively differentiates the two groups, is used to group features into various categories or classes. In a high-dimensional space, a hyperplane is a border that divides data points into their

corresponding or specialized classes. There are data points on both sides of the hyperplane that can be assigned to different classes.

4.6.1.3. Random Forest

Random forest (RF) is a learning technique based on ensemble trees. It consists of a set of decision trees derived from a randomly selected portion of the training dataset. It combines the votes from several decision trees to arrive at a consensus on the test data's ultimate classification. It creates decision trees for each random data sample and then uses them to make predictions. Following that, the best answer is determined by a vote. It combines a large number of DTs with a unique set of observations, and the splitting nodes are determined by a variety of factors. In addition, the RF's ultimate prediction is made by combining the projections of each individual tree.

4.6.1.4. Catboost for Gradient Boosting

Catboost is a highly efficient library that is a variant of gradient boost and decision tree. This amazing library can solve and be used for multiple types of machine learning problems, such as classification, regression, and even multiclass classification. Catboost is a superhot method that has exceeded some of its relevant algorithms, such as LightGBM, H2O, and even XGBoost. The comparison is on their official website [14]. Amazingly, Catboost could utilize the categorical features positively by using them in the training process and exceed the performance of the model. There will be no need to even do any sort of one-hot encoding. Plus, Catboost can handle overfitting problems perfectly, where small datasets can be easily overfitted in gradient boosting models. Catboost will enforce special parameters during the training, and the user will be thrilled with this spectacular algorithm.

The whole dataset was trained on the Catboost model with no one-hot encoding for the categorical features. In order to input the training, validation, and testing data, we created a train, vol, and test pools to use them as inputs to the model parameters while constricting the model before the fitting (training) phase. The model performed pretty well during the training and validation. However, during our implementation, this specific algorithm was the best performed among our three phases of implementation, which will be described in detail in the next chapter.

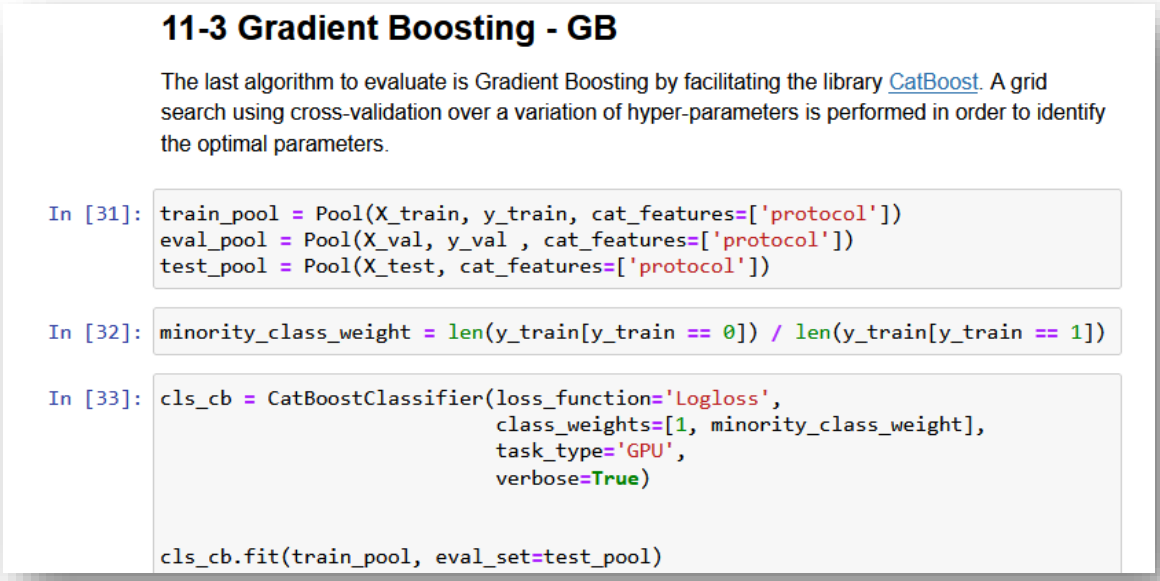

Figure 17 GB model Creation

To understand the model creation, we can go through some of the essential parameters that are identified above in fig 16.

Loss function: specify what machine learning problem to solve.

Class weight: the following parameter is used in both binary and multiclass problems to determine the multiplier values of the binary or multiclass labels.

Task type: to ensure the use of GPU

Verbose: having details output while the training process is running.

Cls_ch.fit: starting the training process by using fit transformer on the training dataset

Cat features: the categorical features that will be used in the training process, which may enhance the model's performance.

4.6.2. MULTICLASS MODELS

After the completion of the first phase: binary classification, in order to deal with our implemented dataset and ensure high model performance, we will transfer all the attack traffic to an external CSV file and eliminate any benign or normal traffic that could appear

in the multiclass phase. This section, however, goes deep into only the best performing models in our multiclass experiment.

Random Forest and XGBoost are two of the most efficient models in the multiclassification phase, and they are described in detail below, with XGBoost taking a more technical approach because it outperforms all other multiclass models in our multiclass prediction phase.

4.6.2.1. Random Forrest

As we discussed in the previous binary class section, the Random Forest algorithm could be useful even in multiclass classification by providing the correct type of data and the correct classes (aka labels) in the independent variables section. Thirteen attack classes, labeled from 1 to 13, by giving each class a specific number, have been given to this algorithm to evaluate its accuracy, recall, precision, and F1 score. The data was divided into testing and training groups with no validation set, and the features were extracted using the Select KBest feature selection method.

4.6.2.2. XGboost

In the same way, using data splitting and feature selection methodologies, XGBoost was given the right set of data to enhance the prediction and train scores. As we can see, the name indicates another boosting algorithm that is a variant of the god father of the boost's models the Gradient Boosting framework. XGBoost is a distributed gradient boosting toolkit that has been tuned for efficiency, flexibility, and portability. XGBoost is a parallel tree boosting (also known as GBDT or GBM) algorithm that solves a variety of data science issues quickly and accurately.

By using GBoost, we have applied the algorithm as it is on the training and testing data with all default parameters and no specific optimization or modification in the model construction phase.

4.6.3. ANOMALY DETECTION / ONE CLASS

One of the most resilient and practical methodologies in anomaly detection in general is one class classification. In the following section, we are only dealing with our benign traffic in the training phase. Basically, the algorithm will take the benign traffic as an input and perform the necessary training afterwards. We combine both training and testing to test the model if they can spot the outliers or abnormal behaviors in the testing or validation phase. Three of the best-performing models are described in the following subsections, which are divided into two main categories: one for one-class classification classical machine learning algorithms and the other for the use of autoencoder with its best-performing architecture denoising autoencoder.

4.6.3.1. Local Outlier Factor

In order to utilize the utilities of supervised learning, our local outlier factor algorithm (LOF) works in a special way where it computes the data points based on other neighbors in the dataset by using local density deviation. The density will be calculated for each data point, and the lower the density is with other neighbors, the data will be spotted as an outlier. So, one of the main uses of this algorithm is outlier detection in the dataset. Moreover, this model had another beautiful option, which is novelty detection, where we are only considering any new observations in the data or any abnormal behaviors that are not by any means related to the training data; this novel methodology is our concern in this spectacular research. The data has been trained only on the benign traffic as we mentioned in the main section, then both classes of traffic are combined in the testing phase to spot any novel observations.

4.6.3.2. One Class SVM

Both regression and classification problems could be targeted by the support vector machine algorithm. However, SVM is mostly used in classification challenges. The nice thing about this model is that it has so many variations that it could be used in one class, binary class, or multiclass problems with different architectures that have slightly different names. Normal SVM uses the functionalities of Hyperplane that will separate the two binary classes, one from the other, in order to perform the required classification.

Moreover, one class of SVM operates a little bit differently, with two main approaches: outlier and novelty detection. We are mostly concerned about novelty detection, so we are going to skip the outlier approach for the moment. Just like LOF but with a different flavor where it differentiates between the data and the feature space using the hyperplane. So, there will be two main regions: low space density and maximum space density. Each one will have a type of class to spot the anomalies in the lower one.

4.6.3.3. Autoencoder Denoising Architecture

An autoencoder is a type of neural network that learns a dense representation of the input data. To do this, an autoencoder is trained to reconstruct the supplied inputs by encoding the input characteristics as dense representations (also known as latent representations or coding) and then decoding the dense representations to reconstruct the original inputs. The model should learn the identity function of the input data using this method.

Autoencoders are similar to Principal Component Analysis (PCA), where it reduces the number of important features to the least important ones (dimensionality reduction). That way, they may be used for dimensionality reduction or feature extraction, but they can also be used for anomaly detection problems.

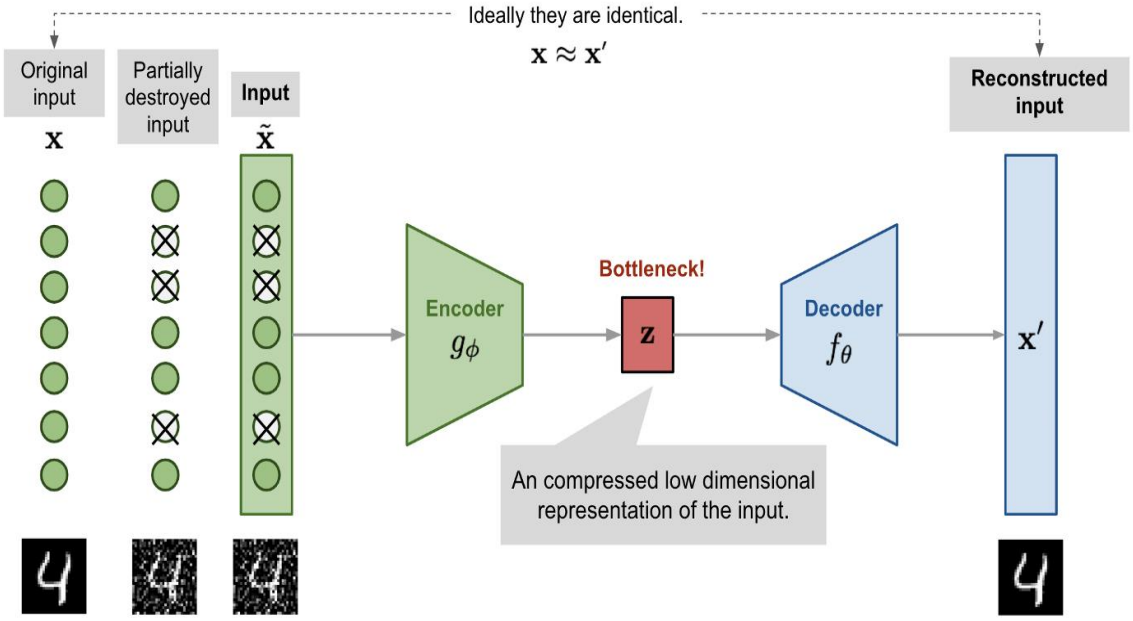

Figure 18 Denoising Autoencoder[12]

When there are more nodes in the hidden layer than inputs, the network runs the danger called the "Identity Function," also known as the "Null Function," which means that the output equals the input, and obviously that will make the autoencoder just useless and no output will be reconstructed in prober form.

Denoising Autoencoders tackle this problem by purposefully adding noise to the data by selecting some of the input data randomly and training them to zero values. That will be the first stage of the denoising autoencoder after the input of the data, just like it is mentioned in figure 18 [12].

So, the main advantage and goal of the denoising autoencoder is to re-build the nosing data after it passes the encoder and the bottleneck without any noise.

figure 19 below shows the architecture of our model. It is important to mention that the denoising autoencoder used in this project is overcomplete, so the hidden layers contain more units than the input layer. The model is made up of five levels, each with 512 units. The probability of introducing noise is set at 15%. The settings were found by trial and error.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               31232

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 512)               262656

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 512)               262656

 dropout_2 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 512)               262656

 dropout_3 (Dropout)         (None, 512)               0

 dense_4 (Dense)             (None, 512)               262656

 dropout_4 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 60)                30780

=================================================================
Total params: 1,112,636
Trainable params: 1,112,636
Non-trainable params: 0
```

Figure 19 Denoising Autoencoder NN construction

The learning curves suggest that the model does not overfit the training data. As we use Dropout layers, the validation loss is lower than the training loss. The training curves that will be presented in the results and evaluation chapter show us that there is no overfitting on the training data. That could be due to our usage of dropout layers. More details on the model performance and results are provided in chapter five.

4.7.SUMMARY

In this chapter we have carefully explained our implementation methodology for all the project phases, starting from data preprocessing by cleaning and eliminating all unnecessary feature records in the dataset; the use of some of the feature selection methods, filter correlation method and select KBest method, where each method returns the most useful features with respect to certain parameters; and dataset splitting into three sets: training, validation, and final testing. We have gone through all three phases of binary class, multiclass, and anomaly detection where we have used both the utilities of classical machine learning algorithms and the amazing efficient model architecture denoising autoencoder.

CHAPTER 5

RESULTS

5. CHAPTER 5: RESULTS

Using publicly available datasets, we were able to evaluate the performance of both classical machine learning models and the autoencoder on three different structures. This also allows us to find the optimal model that could generalize well and could be considered the baseline model for our research in terms of all phases: binary, multiclass, and one-class classifications. Both the CICIDS-2017 and main CICIDS-2018 datasets were combined to evaluate and optimize the final estimator. The datasets are split into train, test, and validate sets, and the final features are extracted using the filter correlation method. Train sets were used to train all the machine learning models for both benign and malicious traffic, except for the anomaly detection phase, which only had one type of class that was identified as the benign class following the novelty detection approach. The validated and test sets are used to evaluate the model performance in all phases by combining the numeric class labels and prediction data.

5.1. PERFORMANCE COMPARISON

The highest training and evaluating performance were spotted at the binary classification phase. Table 5 shows a good range between 98% to 99% weighted average recall and precision for Random Forest and our final estimator (CatBoost). In terms of multiclass algorithms, Random Forest and XGBoost in Table 6 showed quite good performance measures of between 96% and 97% for accuracy, recall, and precision. While in the case of the anomaly detection phase autoencoder in Table 8, which came with the highest performance of 88% weighted recall using the denoising autoencoder structure, the highest performing model out of all the one-class machine learning models in Table 9 was the one-class SVM and Local Outlier Factor, with accuracy, recall, and precision of between 70% to 79%.

Gradient Boost and RF performed the best of all the models. However, in terms of precision and recall, gradient boost-Catboost will be used as the final estimator.

5.1.1. BINARY CLASS

In this specific section, in Table 6 we have tested several machine learning models that will provide binary class behavior in the training process, for a total of five models: SVM, DT, RF, Catboost, and LR. They have been tested to come up with the best performing model as the final estimator for the experiment, where the thing was determined by the Catboost model with the highest 99% weighted recall and precision.

As it is mentioned below in the classification report, Figure 20, and the confusion matrix, Figure 21, the Catboost returns surprisingly promising results on both weighted scores of recall and precision.

```
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.99      1.00      0.99   1348471
           1       1.00      0.94      0.97    274824

    accuracy                           0.99   1623295
   macro avg       0.99      0.97      0.98   1623295
weighted avg       0.99      0.99      0.99   1623295

Avg Precision Score: 0.9484097143331572
```

Figure 20 Catboost classification report

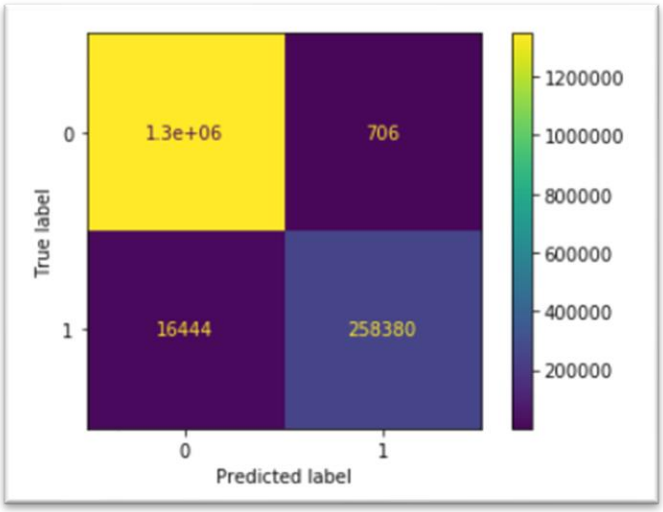

Figure 21 Catboost Confusion matrix

Model	Recall	Precision	F1	Recall Attack	Precision Attack
Baseline	0.83	0.69	0.75	0.00	0.00
Logistic Regression	0.92	0.92	0.92	0.70	0.83
Random Forest	**0.99**	0.96	0.99	**0.95**	**0.96**
Gradient Boost	**0.99**	0.99	0.99	**0.94**	**1.0**

Table 5 Binary Classification Performance Table

5.1.2. MULTICLASS

Multiclass classification has been tested with only a total of fourteen attack traffic classes by eliminating the benign traffic class in the first place. Using our classical ML approach, In table 6 all five multiclass algorithms (DT, RF, NB, SVM-RBF, XGBoost) were evaluated to finalize the best performed estimator in the multiclass phase, which was identified as XGBoost with the highest accuracy of 97% shown in figure 23.

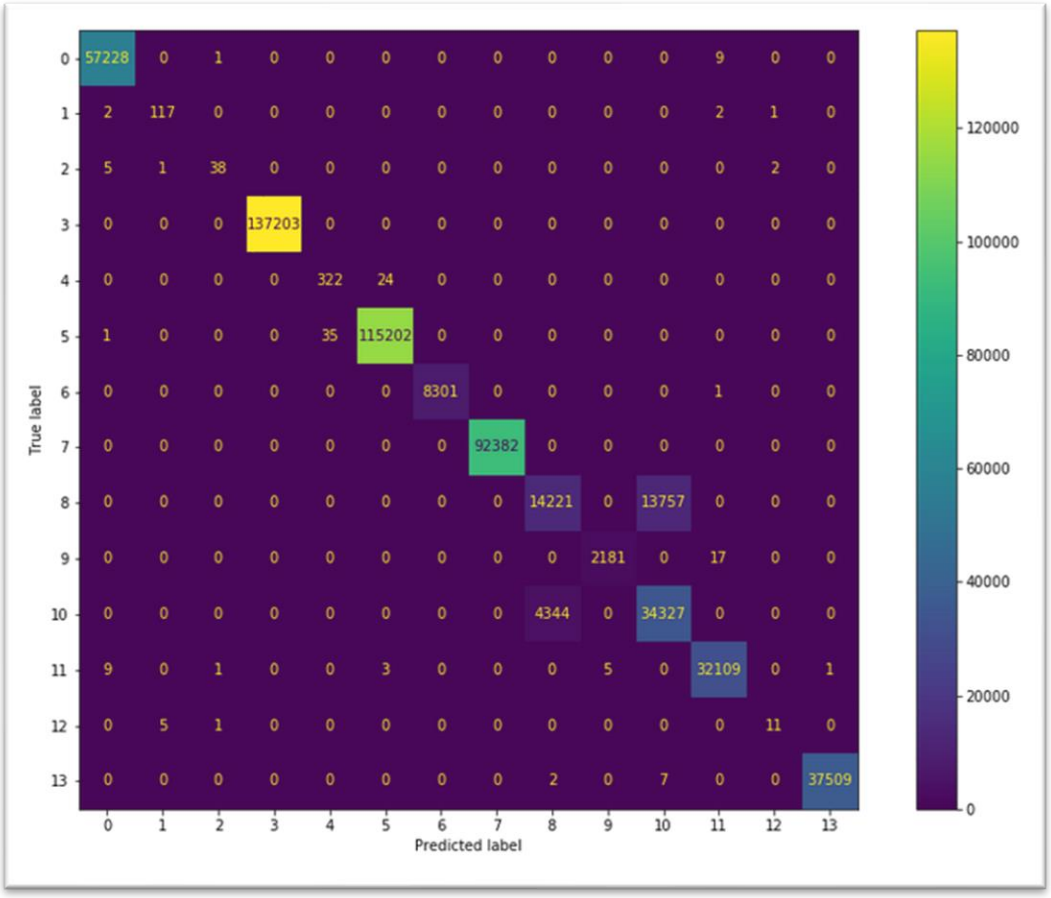

Figure 22 XGBoost Confusion Matrix

Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	57238
1	0.95	0.96	0.96	122
2	0.93	0.83	0.87	46
3	1.00	1.00	1.00	137203
4	0.90	0.93	0.92	346
5	1.00	1.00	1.00	115238
6	1.00	1.00	1.00	8302
7	1.00	1.00	1.00	92382
8	0.77	0.51	0.61	27978
9	1.00	0.99	0.99	2198
10	0.71	0.89	0.79	38671
11	1.00	1.00	1.00	32128
12	0.79	0.65	0.71	17
13	1.00	1.00	1.00	37518
accuracy			0.97	549387
macro avg	0.93	0.91	0.92	549387
weighted avg	0.97	0.97	0.97	549387

Figure 23 XGBoost Classification Report

Model	**Recall**	**Precision**	**F1**
Support Vector Machine	0.95	0.95	0.95
Naive Bayes	0.79	0.78	0.76
Random Forest	**0.97**	0.97	0.96
Gradient Boost	0.88	0.88	0.78
Decision Tree	**0.97**	0.97	0.97
XGboost	**0.97**	0.97	0.97

Table 6 Multiclass Model Performance

5.1.3. ONE CLASS

The provided results tell us that anomaly detection methods weren't the best option for real-life implementation. Surprisingly, the classical machine learning models table 7 in the previous experiments outperform the 3 structures of autoencoder neural networks table 8. However, the best advantage of this process is that anomaly detection can take only one class/label during the training, which is a very good option in the absence of malicious network traffic during the training phase.

Furthermore, the estimator's predictions are heavily dependent on the value of the decision boundary, which can only be estimated correctly given enough damaging data. The method's principal benefit is that only benign data is necessary to create the estimator, with the acquisition of harmful data being either not required or highly limited. This constraint restricts the applicability of this technique marginally.

However, the presented method may be helpful in situations where malicious training data is only available in small ratios or not at all. Setting an appropriate confidence interval based on the distribution of benign samples and adjusting the boundary as new data comes in could be used to determine a decision boundary if no malicious training data is available.

Model	Precision	Recall	F1
OneClassSVM	**0.82**	**0.71**	**0.70**
Isolation Forest	0.31	0.51	0.39
local outlier factor	0.71	0.71	0.70

Table 7 Classical One Class ML Model Performance

Model	Precision	Recall	F1
Simple Autoencoder	0.64	0.59	0.54
Denoising Autoencoder	**0.83**	**0.82**	0.82
Stacked Autoencoder	0.80	0.79	0.79

Table 8 Autoencoder Model Performance

```
Classification Report:
======================
              precision    recall  f1-score   support

           0       0.88      0.74      0.81   1348471
           1       0.78      0.90      0.84   1374117

    accuracy                           0.82   2722588
   macro avg       0.83      0.82      0.82   2722588
weighted avg       0.83      0.82      0.82   2722588
```

Figure 25 Denoising Autoencoder Classification report

```
In [82]: print("Accuracy :", metrics.accuracy_score(new_y_test , yhat))
         Accuracy : 0.7110855829982768

In [83]: print(classification_report(new_y_test , yhat))
                      precision    recall  f1-score   support

                  -1       0.60      1.00      0.75      3000
                   1       1.00      0.49      0.66      3964

            accuracy                           0.71      6964
           macro avg       0.80      0.75      0.70      6964
        weighted avg       0.82      0.71      0.70      6964
```

Figure 24 One Class SVM classification report

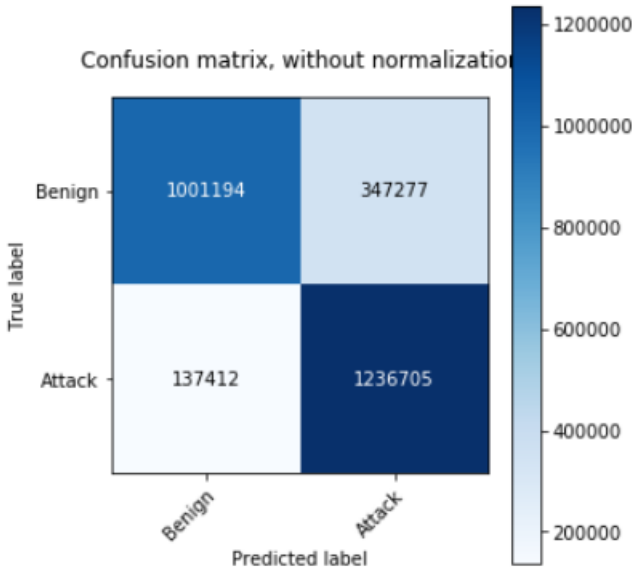

Figure 26 Denoising Autoencoder Confusion matrix

5.1.4. MISCLASSIFICATION

The misclassifications in the test dataset are listed in Table 9 & figure 27, demonstrating that attacks of type infiltration are frequently misclassified.

Misclassifications:	label	percentage
Infiltration	11937	0.737125
Brute Force -Web	32	0.524590
SQL Injection	3	0.375000
Brute Force -XSS	3	0.130435
DoS attacks-Slowloris	13	0.011829
Benign	8425	0.006248
Bot	35	0.001223
DDoS attacks-LOIC-HTTP	8	0.000139

Table 9 Misclassification report

Furthermore, minority attack types are frequently misclassified. To improve performance, synthetic minority oversampling can be applied to the train dataset for these classes.

```
print('Misclassifications:')
display(mis)
```

```
Misclassifications:
```

		0	percentage
binary_class	pred		
1	0	16249	1.0
0	1	682	1.0

Figure 27 Misclassification report of Both classes

5.1.1. SYNTHETIC MINORITY OVERSAMPLING (SMOTE)

Attack classes with a small number of occurrences appear in both databases present in figure 28. As it is shown in Figure 29, Synthetic Minority Oversampling is used to increase the occurrences of certain classes by 100,000 occurrences in order to improve the detection rate for these attacks.

```
In [54]: from collections import Counter
         print('Class occurrences:')
         Counter(y_train_s)

         Class occurrences:

Out[54]: Counter({0: 12606243,
                  7: 460953,
                  10: 116311,
                  1: 230526,
                  9: 554388,
                  5: 548809,
                  15: 127144,
                  4: 102422,
                  14: 129576,
                  12: 161038,
                  17: 154789,
                  11: 100000,
                  8: 100000,
                  2: 100000,
                  6: 100000,
                  3: 100000,
                  16: 100000,
                  13: 100000})

In [55]: y_train_s = (y_train_s != 0).astype('int')
         print('Binary label occurrences:')
         Counter(y_train_s)

         Binary label occurrences:

Out[55]: Counter({0: 12606243, 1: 3285956})
```

Figure 29 Class Labels after applying SMOTE

```
Benign                      12606243
DoS attacks-Hulk              554388
DDOS attack-HOIC              548809
DDoS attacks-LOIC-HTTP        460953
Bot                           230526
FTP-BruteForce                161038
SSH-Bruteforce                154789
Infilteration                 129576
PortScan                      127144
DoS attacks-SlowHTTPTest      116311
DDOS LOIT                     102422
DoS attacks-GoldenEye          41441
DoS attacks-Slowloris          13429
Brute Force -Web                1694
DDOS attack-LOIC-UDP            1384
Brute Force -XSS                 706
SQL Injection                     86
Heartbleed                         9
Name: label, dtype: int64
```

Figure 28 Original Class Labels

5.2.TEST NOVEL DATA

Despite the combined estimator's impressive performance, it's safe to presume that, due to observed disparities in distributions, it won't translate well to diverse network contexts. being the first dataset(cicids-2018) gathered in an AWS cloud environment, where is cicids2017 was collected in a traditional LAN setting.

In order to cope with this problem, the following suggestions are proposed:

- An estimator has to be trained with more varied attack data originating from different network environments.
- An estimator has to be trained with data originating from the network environment it will be deployed in. Such as cloud in the case of the cicids-2018 dataset

Because high-quality real-world network attack datasets are difficult to find, the second alternative solution appears to be more promising. The need for data collection in the target environment can be reduced by using only benign traffic and detecting network attacks using an anomaly detection method.

Classification Report (Novel):

	precision	recall	f1-score	support
0	0.85	0.97	0.91	2273097
1	0.71	0.30	0.42	557646
accuracy			0.84	2830743
macro avg	0.78	0.63	0.66	2830743
weighted avg	0.82	0.84	0.81	2830743

Avg Precision Score: 0.34918627997276036

Figure 30 Test on Novel CICIDS-2017 Data

5.3.FEATURE IMPORTANCE

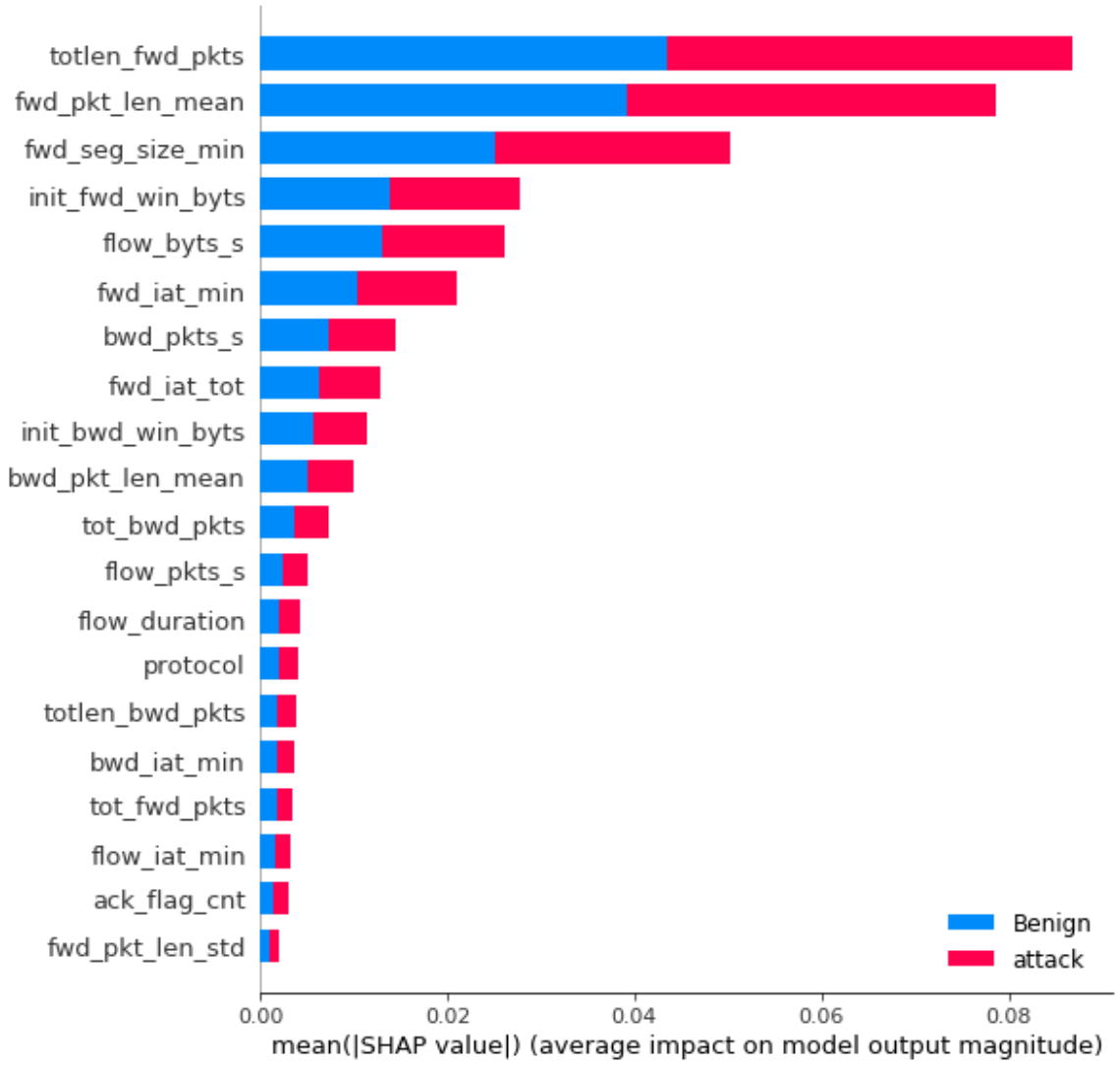

Figure 31 Shap Analysis : feature importance

SHAP analysis provides us with the features that have the highest influence on the predictions of the model:

1. `init_fwd_win_byts`: Number of bytes sent in initial window in the forward direction
2. `fwd_pkt_len_mean`: Mean size of packet in forward direction
3. `protocol`: Protocol
4. `init_bwd_win_byts`: Number of bytes sent in initial window in the backward direction
5. `fwd_seg_size_min`: Minimum segment size observed in the forward direction

5.4.FINAL COMBINED BINARY ESTIMOTR

OPTIMIZATION OF PARAMETERS

Given our training data, we use hyperparameter search to discover the best parameter configuration for the algorithm.

The search-spaces parameter is defined as follows:

- The greatest number of trees “nr_iterations”: uniform integer space between [100, 2000].
- Tree depth “depth”: uniform integer space in the range [4–10].
- L2 regularization coefficient “l2_leaf_reg”: uniform space in the [1, 10] interval.
- splits for numerical features “border_count”: a choice of 128 and 254.
- Randomness used for scoring splits “random_strength”: uniform integer space in the interval of [0, 5].

Conducting grid-search the following parameters were determined for an optimal estimator in Figure 32 and Table 10:

Parameter	Value
Depth	9
L2 Leaf Regularization	3
Iterations	1500
Learning Rate	0.3

Table 10 Optimal Estimator Parameters

```
Optimal Parameters:
{'depth': 9, 'l2_leaf_reg': 3, 'iterations': 1500, 'learning_rate': 0.3}
```

Figure 32 Optimal Parameters

The combined estimator shows promising performance on the test dataset with a high **recall of 0.99, precision of 0.99, and an attack detection rate (recall class 1) of 0.96.**

Classification Report (Test):

	precision	recall	f1-score	support
0	0.99	0.99	0.99	3151562
1	0.97	0.96	0.97	661176
accuracy			0.99	3812738
macro avg	0.98	0.98	0.98	3812738
weighted avg	0.99	0.99	0.99	3812738

Avg Precision Score: 0.9414048643545958

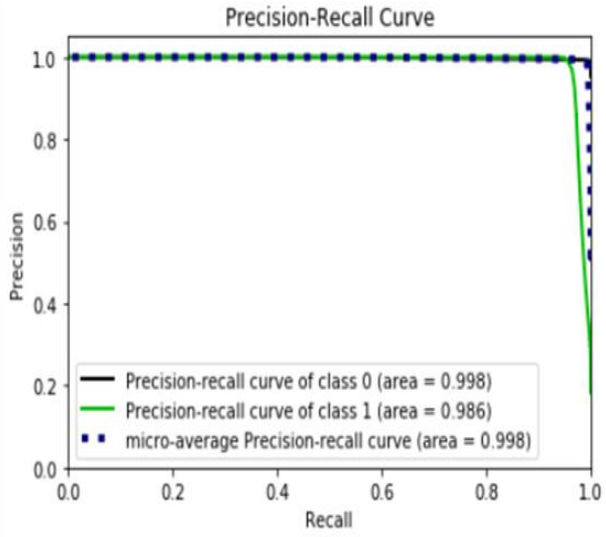

Figure 33 Final Binary Estimator Performance

5.5.MODEL DEPLOYMENT

For model deployment and getting the most promising models into work, we have extracted the chosen models as pkl files. Two final phases have been selected as shown in the previous pipeline in figure 2. In the first phase, the final binary estimator will be utilized to classify benign and attack traffic from the input CSV file. The attack traffic will then be transferred to an external dataframe that only stores the identified attack packets. Afterward, in the second phase, all the generated attack traffic is going to be classified into different attack traffic to know which type of attack has been captured.

Passive detector

In the detector.py script, the process will be mainly focused on CSV ready converted files that have the required features specified in advance. They will be preprocessed and taken as input to the pkl file and the prediction will start from the binary class process into the multiclass, focusing only on the detected packets that were identified as attack.

Real-Time Detector

The second script is a combination of two options that will be running in parallel. First of all, the CICFlowmeter will be started by specifying the network interface card of the system and the output csv file. Immediately, the tool will start capturing network packets and fill the CSV that was specified as the output in the command line. Afterward, the real time detector is going to run and wait for the incoming traffic to pkl the csv that was generated using the CICFlowmeter. Whenever any new packet comes, the real time detector will classify whether it is benign or malicious.

5.6.SUMMARY

In this chapter we have presented our final implementation results in detail for all the outperformed and promising models; the final two model deployment scripts; and a comprehensive comparison of each phase's performance.

CHAPTER 6

CONCLUSION

6. CHAPTER 6: CONCLUSION

Intrusion detection systems (IDS) are such a useful methodology to catch and analyze network attacks by performing network traffic monitoring. In this study, we took an extra step to leverage AI in IDS technologies by presenting a proof of concept of three different classification techniques (Novelty detection, binary class, and multiclass classifications) on network traffic at the NetFlow features level. Afterwards, we examined the performance of a total of 12 classical machine learning and deep learning models by splitting the dataset into training, validation, and testing. Finally, we have compared all machine learning models to the autoencoder different variations, where we observed a significant leverage of the machine learning algorithms on the deep learning autoencoder models. For future work, we would like to test different deep learning models against the classical algorithms, implement a wrapper feature selection method, and go for further real-live deployment of an AI-based IDS system.

7. REFERENCES

[1] K. Farhana, M. Rahman, and M. Tofael Ahmed, “An intrusion detection system for packet and flow based networks using deep neural network approach,” *Int. J. Electr. Comput. Eng.*, vol. 10, no. 5, pp. 5514–5525, Oct. 2020, doi: 10.11591/IJECE.V10I5.PP5514-5525.

[2] A. Ferriyan, A. H. Thamrin, K. Takeda, and J. Murai, “Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic,” *Appl. Sci.*, vol. 11, no. 17, Sep. 2021, doi: 10.3390/app11177868.

[3] Zeek, “ZEEK IDS.” .

[4] S. Abt and H. Baier, “Are We Missing Labels? A Study of the Availability of Ground-Truth in Network Security Research,” in *Proceedings - 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2014*, Apr. 2016, pp. 40–55, doi: 10.1109/BADGERS.2014.11.

[5] A. Alshammari and A. Aldribi, “Apply machine learning techniques to detect malicious network traffic in cloud computing,” *J. Big Data*, vol. 8, no. 1, Dec. 2021, doi: 10.1186/s40537-021-00475-1.

[6] A. Aldribi, I. Traoré, B. Moa, and O. Nwamuo, “Hypervisor-based cloud intrusion detection through online multivariate statistical change tracking,” *Comput. Secur.*, vol. 88, 2020, doi: 10.1016/j.cose.2019.101646.

[7] A. Alshammari and A. Aldribi, “Apply machine learning techniques to detect malicious network traffic in cloud computing,” *J. Big Data*, vol. 8, no. 1, Dec. 2021, doi: 10.1186/s40537-021-00475-1.

[8] “Canadian Institute for Cybersecurity.” https://www.unb.ca/cic/.

[9] C. Cerrone, R. Cerulli, and B. Golden, “Carousel greedy: A generalized greedy algorithm with applications in optimization,” *Comput. Oper. Res.*, vol. 85, no. April, pp. 97–112, 2017, doi: 10.1016/j.cor.2017.03.016.

[10] X. Fu, C. Zhang, J. Chen, L. Zhang, and L. Qiao, “Network traffic based virtual machine migration in cloud computing environment,” *Proc. 2019 IEEE 3rd Inf. Technol. Networking, Electron. Autom. Control Conf. ITNEC 2019*, no. Itnec, pp. 818–821, 2019, doi: 10.1109/ITNEC.2019.8729184.

[11] R. Panigrahi and S. Borah, “A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems,” *Int. J. Eng. Technol.*, vol. 7, no. 3.24 Special Issue 24, pp. 479–482, 2018.

[12] Lilian Weng, “denoising autoencoder.” https://lilianweng.github.io/posts/2018-08-12-vae/.