UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

MASTER DEGREE IN ICT FOR INTERNET AND MULTIMEDIA

NETWORK ANOMALY DETECTION USING MACHINE LEARNING

Supervisor: Prof. Nicola Laurenti

Co-supervisor: Prof. Stefano Tomasin

Internship supervisor: Antonios Atlasis

Candidate: Stefano Leggio

ACADEMIC YEAR: 2022-2023

Graduation date: 30/11/2023

Abstract

The constant increase of network attacks in the digital world creates a significant threat to system security and availability. Anomaly detection plays a crucial role in identifying previously unknown network attacks and potential malicious activities. This thesis focuses on leveraging machine learning techniques for effective network anomaly detection to enhance cybersecurity measures. The study explores various machine learning models to develop a robust and efficient anomaly detection system. At the end of the research, a novel framework based on *Autoencoders* (AE) is proposed to further enhance the detection capabilities.

Contents

List of Figures

List of Tables

List of abbreviations

AE *Autoencoder.*

AI *Artificial Intelligence.*

AUROC *Area Under the Receiver Operating Characteristics.*

AV *Antivirus.*

BiRNN *Bidirectional Recurrent Neural Network.*

CNN *Convolutional Neural Network.*

DAGMM *Deep Autoencoding Gaussian Mixture Model.*

DDOS *Distributed Denial Of Service.*

DL *Deep Learning.*

DOS *Denial Of Service.*

DT *Decision Tree.*

ESA *European Space Agency.*

FI *Feature Importance.*

FPR *False Positives Rate.*

GMM *Gaussian Mixture Model.*

HIDS *Host Intrusion Detection System.*

IDS *Intrusion Detection System.*

IoT *Internet of Things.*

IP *Internet Protocol.*

IPS *Intrusion Prevention Systems.*

LSTM *Long Short Time Memory.*

MAE *Mean Absolute Error.*

MCC *Matthew's Correlation Coefficient.*

ML *Machine Learning.*

MLP *Multi Layer Perceptron.*

MSE *Mean Squared Error.*

NIDS *Network Intrusion Detection System.*

NN *Neural Network*.

PCA *Principal Component Analysis*.

R2L *Remote to Local*.

ReLu *Rectified Linear Unit*.

RF *Random Forest*.

RFA *Recursive Feature Addition*.

RFE *Recursive Feature Elimination*.

RMSE *Root Mean Squared Error*.

RNN *Recurrent Neural Network*.

ROC *Receiver Operating Characteristic*.

SGD *Stochastic Gradient Descend*.

SVM *Support Vector Machine*.

t-SNE *t-distributed Stochastic Neighbor Embedding*.

TCP *Transmission Control Protocol*.

TOS *Type Of Service*.

TPR *True Positive Rate*.

U2R *User to Root*.

VAE *Variational Autoencoder*.

Chapter 1

Introduction

In this opening chapter, we set the stage for the study by providing preliminary information that will lay the foundation for the subsequent sections.

First, we immerse ourselves in the contemporary landscape of network anomaly detection, shedding light on its pivotal role in the realm of cybersecurity. Additionally, we offer insights into the background and context of this study, sharing that the author dedicated a semester to this endeavor.

Next, we outline the primary objectives of this work, which will guide our exploration in the following sections. Defining these objectives within the introduction is essential to give the reader a clear sense of the anticipated results.

Finally, we provide a comprehensive overview of the study's structure, offering a detailed breakdown of each chapter's content and its overarching purposes. This paves the way for a coherent and structured understanding of the study's progression.

1.1 Context

As technology continues to advance, an increasing number of devices become susceptible to cybersecurity threats. One of the primary defenses against these threats is the implementation of *Intrusion Detection System* (IDS), which are designed to intercept possible cyberattacks. In particular, IDS based on behavioral models, such as *Machine Learning* (ML), have the capability to identify unknown attacks, which constitutes the main challenge in this domain.

Cyberattacks are continuously advancing, becoming increasingly sophisticated and challenging to identify. The demand for research in the field of anomaly detection has become an imperative necessity. Moreover, with the growing computational capabilities of network devices like routers and firewalls, deploying real-time, complex machine learning models for network traffic analysis has become more accessible.

To contribute to this critical endeavor, the author of this thesis has dedicated a semester working with experts at the *European Space Agency* (ESA). While the topic may not be directly linked to the space sector, it is noteworthy that ESA prioritizes network security enhancements to safeguard its sensitive network infrastructure. Throughout this semester, I have had the privilege of collaborating with experts in the field and benefiting from the resources of a cybersecurity laboratory. ESA stands at the forefront of network security research, and I am deeply grateful of the opportunity to contribute to their ongoing research efforts.

1.2 Objectives

With this work, we aim to offer a comprehensive exploration of employing ML methodologies for network anomaly detection. This subject is continually evolving in response to scientific advancements within the field of *Artificial Intelligence* (AI). To achieve this goal, a thorough analysis of all components of a *Network Intrusion Detection System* (NIDS) was essential to comprehend both its strengths and weaknesses. The ultimate objective of the project is to devise a framework that outperforms the existing models in comparison, delivering superior results.

One of the primary objectives of this research is to identify suitable datasets for training a NIDS. The forthcoming methodologies section will delve into the chosen datasets and their distinctive characteristics.

Additionally, this study aims to explore the most appropriate ML models for addressing these security challenges. We seek to assess the significance of ML model selection and assess how the complexity can enhance their performance.

Furthermore, our research delves into understanding the predominant weaknesses in NIDS that leverage ML. This is prompted by the fact that such systems are not yet universally regarded as entirely reliable, despite years of investigation.

Our investigation also aims to differentiate between the performance and approach disparities of supervised and unsupervised ML models. The study encompasses the implementation of models belonging to both categories.

1.3 Structure

This work follows a rigorous structure, divided into four chapters and an introduction. The chapters are as follows: Introduction, Background Theory, Methodologies, Results, and Conclusions. Each chapter will feature a brief introduction and a summary, except for the Conclusions chapter, which will include only a concise introduction.

After this introductory chapter there will be the second chapter which is dedicated to the background theory. This chapter serves as a foundational resource for readers, equipping them with the necessary tools for comprehending the subsequent chapters. Within this chapter, we provide a concise history of IDS, enabling readers to position this study within the broader landscape. Additionally, we dedicate a section to ML, focusing on each model that will be subsequently implemented. Finally, this chapter concludes by delving into the theory related to the primary research topic: network anomaly detection using machine learning. Here, readers will also find a literature review on this subject.

The third chapter is dedicated to the methodologies, encompassing the techniques employed to yield the quantitative results elaborated upon in the subsequent chapter. This chapter begins with a comprehensive section devoted to the datasets used, providing detailed explanations of their strengths and weaknesses. Subsequently, we delve into the preprocessing stage and feature selection, elucidating the techniques applied. To conclude, we offer a brief overview of the models implemented and outline how we will evaluate them in the following chapter.

The fourth chapter, Results, serves as a repository for all the quantitative outcomes of the ML models we have implemented. It starts by presenting the results derived from the feature selection process, with particular emphasis on the selected features to be used in the subsequent models. For each implemented model, we report the results associated with each dataset under consideration. In the final section of this chapter, we expound upon the proposed framework, providing quantitative results from the benchmark conducted on one of the datasets considered. This section also offers insights into the composition of the framework and the achieved improvements.

In the concluding chapter, we will provide a comprehensive overview of the study's findings. This will include a specific focus on the key discoveries, with an emphasis on those achieved through the proposed framework. We will also address the primary limitations of the study, as these have significantly influenced the results obtained. Furthermore, we will outline avenues for future research, which are of paramount importance for those seeking to delve deeper into the subject matter. The chapter will culminate with a concluding remark from the author, offering personal insights into the topic and the results gathered through the study.

Chapter 2

Background theory

In this chapter, we will equip the reader with the necessary tools to comprehend the forthcoming research results. The inclusion of this chapter is imperative due to the extensive array of topics encompassed by the study, ranging from network security to ML.

The chapter will start with a comprehensive introduction to IDS, shedding light on their definition, functioning, and categorization. This will provide readers with a clear context within the cybersecurity landscape. Additionally, we will delve into various anomaly detection techniques, distinct from the ML method employed in this study, to offer a complete perspective.

The second section of this chapter will be devoted to ML. It will start with a comprehensive theoretical overview, elucidating the inner workings of ML models and exploring various learning paradigms. Following this, the discussion will zoom in on the class of problems predominantly addressed in this study, namely binary classification. Furthermore, a comprehensive inventory of metrics employed for assessing the effectiveness of ML models will be presented. In the last part of the chapter will be presented all the ML models that will be used in the following chapters.

The concluding section of this chapter will pivot to the core subject of our study: employing ML for network anomaly detection. Here, we will present an outline of the key components comprising the typical architecture used in such systems, with special emphasis on network flows — a topic that is often taken for granted. The final segment of this section will be dedicated to a survey of pertinent literature in this domain, offering readers an initial view into the latest advancements and state-of-the-art techniques.

2.1 Intusion detection systems

The cybersecurity landscape is in a state of constant evolution, marked by substantial threats to computer networks. An IDS serves as an application designed to identify and detect such attacks, thereby facilitating their mitigation before the attacker can successfully compromise the system. The ultimate goal of an IDS is to initiate an alert to the system or network administrator, enabling a prompt response to counteract the ongoing attack. In a broad and simplified perspective, we can define an IDS as a detection tool that examines system event logs. Based on a predefined configuration and reference database, the IDS generates alerts that signal the need for appropriate countermeasures to the attack. To evaluate the efficiency of the system, it is common to consider the accuracy, completeness, fault tolerance and timeliness [1].

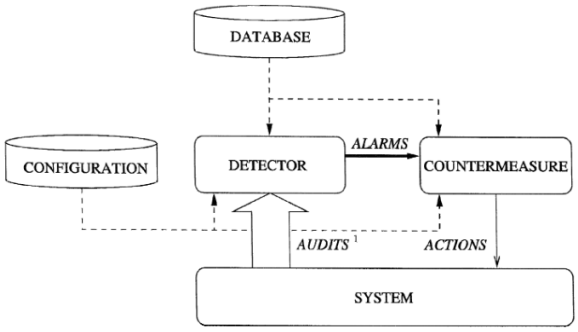


Figure 2.1: General *Intrusion Detection System* (IDS) scheme [1]

Advanced IDS can also autonomously mitigate these suspicious activities, so they are often referred to as *Intrusion Prevention Systems* (IPS). This may include blocking some incoming network traffic, modifying the firewall rules or implementing some measures at the operating system level.

2.1.1 Network and host based IDSs

A fundamental categorization of IDS can be made by distinguishing between network-based and host-based variants.

Network-based IDSs, commonly referred to as NIDSs, operate at the network level and are designed to identify suspicious network traffic patterns [1]. For example, they are particularly effective in detecting some typical network attacks such as *Denial Of Service* (DOS) attacks and *Distributed Denial Of Service* (DDOS) attacks orchestrated by botnets. Advanced NIDS possess the capability to perform deep packet inspection, allowing them to delve into the content of network packets to identify potential remote exploits.

The Host-based IDSs, commonly referred to as *Host Intrusion Detection System* (HIDS) are tools that aims to monitor individual systems operating directly on the host endpoint [1]. HIDS plays a critical role in ensuring the security and integrity of a host system by closely examining its operational state. It accomplishes this task by diligently inspecting various aspects of the host's activities, primarily relying on logs and processor-level instructions to detect any suspicious or potentially malicious behavior. HIDS may also employ low-level processor instructions to inspect memory access patterns, system calls, and other critical system-level operations. Some of the most common attacks that a HIDS can detect are: unauthorized access such as bruteforce attacks, malware or rootkit presence by detecting file changes

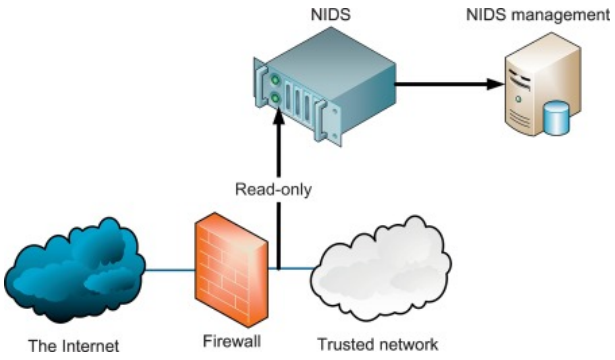


Figure 2.2: *Network Intrusion Detection System* (NIDS) scheme [2]

or register modification, file system changes and privilege escalation. The classical antivirus software commonly adopted at the endpoint of a system is a typical example of HIDS, most of them can also mitigate some attacks becoming an IPS host-based.

2.1.2 Signature based IDSs

Another important distinction that can be made among the IDS relates to its signature-based or anomaly-based nature. This distinction holds significance as it characterizes the methodology employed by the IDS in detecting attacks.

Signature-based IDS function by scrutinizing attack fingerprints stored in a database. When there is a match of an attack fingerprint in the database, the system identifies the attack. This approach is commonly employed in most *Antivirus* (AV) systems and, in a broader context, in many HIDS. However, a significant limitation of this method is its inability to detect unknown attacks or even variations of common attacks. On the other hand, this type of IDS offers several distinct advantages. Firstly, its implementation is remarkably straightforward, requiring nothing more than a comprehensive database of attack signatures and a relatively simple detection algorithm. Secondly, the *False Positives Rate* (FPR) is exceptionally low since a match occurs only when the attack is found in the database.

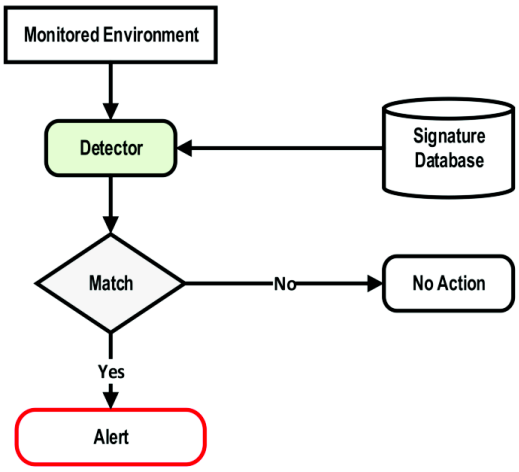


Figure 2.3: Signature based *Intrusion Detection System* (IDS) scheme [3]

2.1.3 Anomaly based IDSs

The other anomaly-based, also known as behavioral-based, are the one that this thesis will explore most. This type of IDSs have the capability to identify unknown attacks by analyzing the behavior of collected data [4]. Their implementation is not as straightforward and is considerably more complex compared to the signature-based approach. Additionally, this type of IDSs are susceptible to classifying all suspicious traffic as attacks, leading to an increase in the FPR.

Anomaly detection techniques can be classified into statistical, spectral, information theoretic, ML and streaming [5][4].In this study, our primary focus will be on ML techniques. Nonetheless, it is essential to comprehend the strengths and weaknesses of all these approaches.

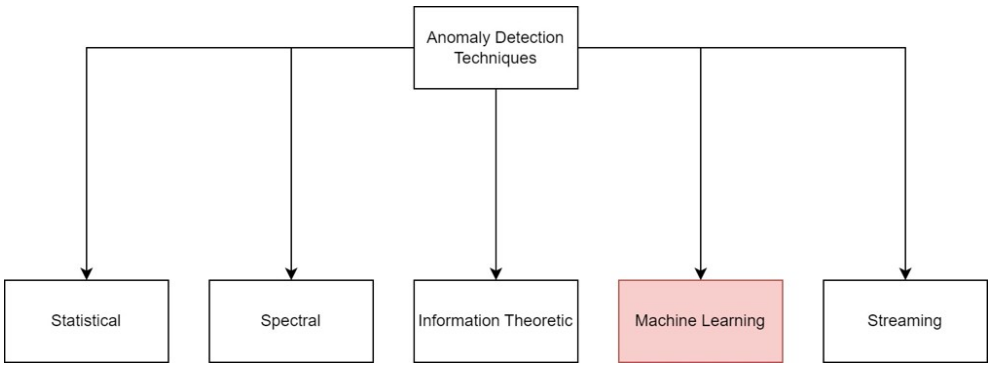


Figure 2.4: Anomaly detection techniques [5][4]

The statistical approach revolves around the fundamental idea of measuring an anomaly score, which indicates the extent to which a given data instance differs from the model [5]. Following this, the score is assessed against a pre-established threshold, serving as a test to determine whether the associated data instance is considered anomalous or not.

The spectral approach involves reducing the dimensionality of the original data instances to create a set of independent dimensions, a process formally known as dimensionality reduction [5]. One commonly employed dimensionality reduction technique is *Principal Component Analysis* (PCA). To determine whether an instance is an attack, this involves a dimensionality reduction step, followed by projection into the anomalous/normal subspace, which is separated by a specific threshold.

The information theoretic approach treats data instances as a collection of symbols generated independently with associated probabilities, as outlined in [5]. Subsequently, the entropy is computed within subsets of data instances and is utilized as an indicator of anomalous behavior. In other words, entropy serves as a key indicator of anomalous behavior. This is because higher entropy levels within a subset of data instances signify a greater degree of unpredictability or randomness, making it a useful metric for detecting anomalies in the dataset.

The ML approach involves the training of a ML algorithm for the classification of anomalous/normal instances. ML models are highly advantageous when the goal is to understand and identify anomalies in a particular behavior. This is a concise overview, with further elaboration to follow in subsequent chapters.

The last approach is the streaming one where the data instances are seen as a continuous flow instead of chunks [5]. In this approach there is a continuous processing of the data as it enters the system.

2.2 Machine learning

ML, a subset of AI, enables algorithms to learn and adapt their behavior through a training process [6]. The fundamental distinction between this approach and traditional coding is depicted in the image below.

In classical programming, an algorithm requires data and predefined rules as input, and it provides an output result based on those inputs. In contrast, a Machine Learning algorithm operates differently. It takes data and the desired outcomes as input, and in return, it generates a set of rules or a model that can be used to make predictions and decisions. This dynamic approach allows algorithms to generalize and handle a wide range of inputs, making it a powerful tool for tasks like pattern recognition, prediction, and classification.

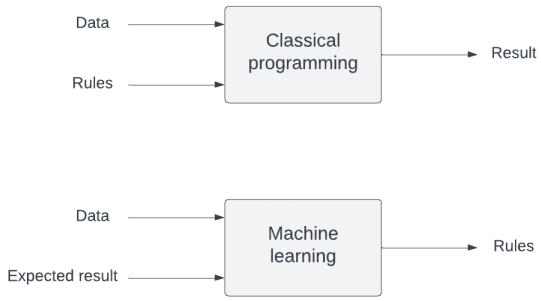


Figure 2.5: *Machine Learning* (ML) compared to classical programming

A ML model can be visualized as a matrix W of weights. The primary objective of the training process is to adjust these weights to enhance prediction accuracy. As illustrated in the following image, the model takes a sample denoted as x_i and associates it with a label represented by y_i. With this input and the weight values within the model, the algorithm generates a predicted label denoted as $\hat{y}_i$. The entire system can be conceptualized as a black-box function: $f(W, x_i)$, which produces a prediction for the label as its output.

The model's weights are acquired through the training process, which unfolds as an iterative procedure. In each iteration, the loss function, represented as $Loss(y_i, \hat{y}_i)$, is computed to quantify the disparity between the actual label and the predicted label [6]. Subsequently, the weights are fine-tuned by considering the gradient of the loss, using a technique for the propagation of the errors such as back-propagation which is commonly used in *Neural Network* (NN). This method enables the determination of weight values that minimize the loss function, thereby achieving the closest match between the labels.

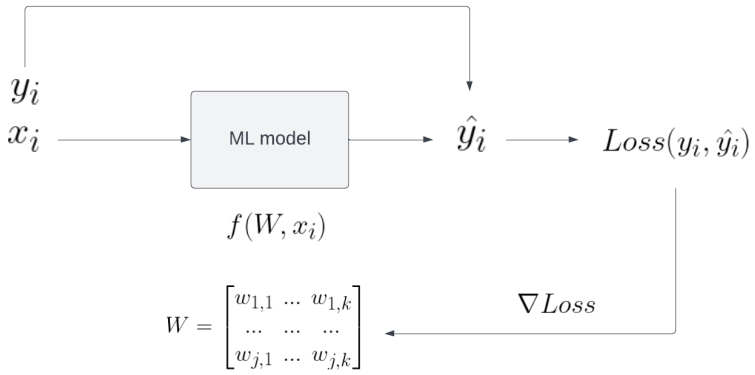


Figure 2.6: *Machine Learning* (ML) training scheme

Following the training phase, the model undergoes a testing phase, during which it is evaluated using data that it has not previously encountered. This testing phase holds significant importance as it provides the opportunity to assess the model's performance. Subsequently, as discussed in the upcoming section, it is essential for the test set to significantly differ from the training set. This divergence is crucial for obtaining reliable metrics for the algorithm.

In cases where the dataset is not pre-divided into subsets, a frequently adopted proportion of samples is a 70% allocation to the training set and a 30% allocation to the test set. Sometimes the test set is further splitted into validation and test set. The validation is used for assessing the performance and fine-tuning the parameters of a ML model during the training process.

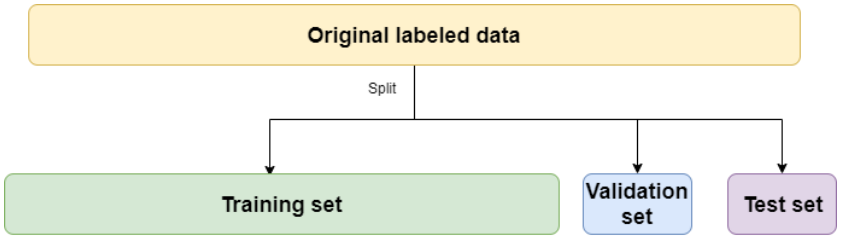


Figure 2.7: Common choice for dataset subsets [7]

2.2.1 Paradigms

In ML we can identify the following paradigms of learning [6]:

- Supervised: Labeled dataset.
- Unsupervised: Unlabeled dataset.
- Semi-Supervised: The labels are learned during the training.
- Reinforcement learning: Interactions with the environment.

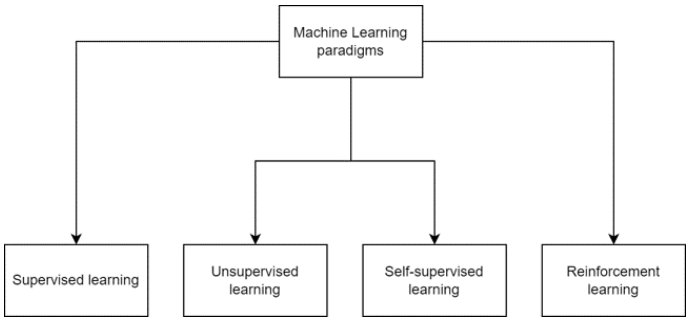


Figure 2.8: Learning paradigms

The distinction among these models lies in how the algorithm interacts with the data throughout the training process. Most of the models discussed in the following sections predominantly fall under either supervised or unsupervised learning categories.

2.2.2 Binary classification

ML can address a diverse range of problems in different domain like regression, clustering, dimensionality reduction and classification. In the case of classification, the objective is to allocate data points to predefined categories or classes. When there are precisely two classes involved, it is termed binary classification.

The image presented below illustrates a conventional ML model, specifically the *Support Vector Machine* (SVM). The SVM is designed to determine a hyperplane that effectively separates the two classes. While this model serves as an example for binary classification, it is not employed for anomaly detection due to the high non-linearity inherent in this particular problem.

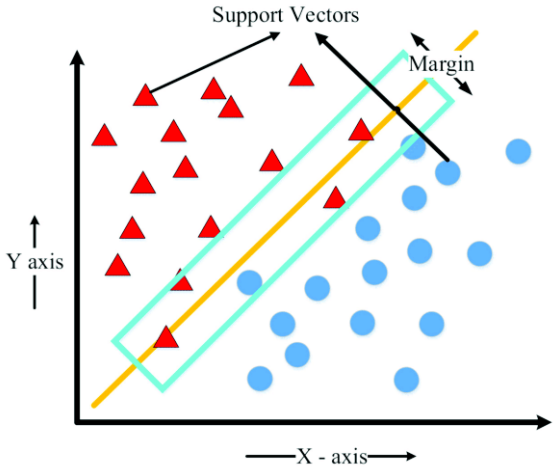


Figure 2.9: Binary classification [8]

The problem of network anomaly detection will be treated as a binary classification with two classes: attack or normal.

2.2.3 Metrics

To evaluate the performances of the models we use different metrics [6]: accuracy, FPR, precision, recall and the F1 score. In the context of binary classification, our model yields four potential prediction outcomes, which can be categorized as follows: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). The metrics that will be consider for the evaluation of the ML models are the following:

- Accuracy: The ratio of the correctly predicted instances to the total instances.

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{2.1}$$

- FPR: The proportion of negatives instances that are incorrectly classified as positive.

$$FPR = \frac{FP}{FP + TN} \tag{2.2}$$

- Precision: The ratio of true positives tot the total predicted positives.

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

- Recall: The ratio of true positives to the total actual positives.

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

- F1 score: The harmonic mean of the precision and recall.

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{2.5}$$

The supports indicates the number of samples considered for computing a specific metric. Another way to visualize the performances of a model is with the confusion matrix which contains the TP, TN, FP and FN. In this study it will be presented in the normalized form:

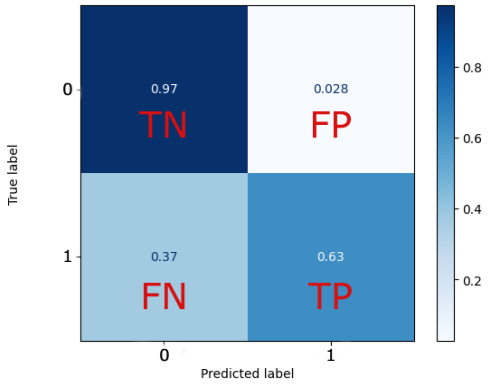


Figure 2.10: Example of confusion matrix

It is often valuable to assess the relationship between FPR and *True Positive Rate* (TPR). This evaluation is typically accomplished through the *Receiver Operating Characteristic* (ROC), plotting FPR against TPR at various thresholds. The *Area Under the Receiver Operating Characteristics* (AUROC) serves as a quantitative measure indicating the model's effectiveness in distinguishing between classes.

2.2.4 k-means clustering

k-means clustering, an unsupervised ML algorithm, seeks to segment a dataset into k clusters, with each data point finding its place in the cluster with the closest centroid [6]. In this context, a cluster is a collection of data points that share similarities, while dissimilar data points are expected to be allocated to separate clusters.

It's important to note that the determination of the number of clusters, denoted as k, is a critical parameter that must be decided to applying the algorithm.

Given μ_i the centroids of the clusters, k the number of the clusters, C_i the clusters and X the set of vectors that have to be clustered, the standard algorithm of k-means has the following steps:

1. Initialization: Select k random centroids.

2. Assignment: Associate each point to the closest centroid.

$$\forall i : C_i = \{x \in X : i = argmin_j ||x - \mu_j||^2\} \tag{2.6}$$

3. Update: compute the new centroids.

$$\forall i : \mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x \tag{2.7}$$

4. : Repeat step 2 and 3 until the algorithm converges.

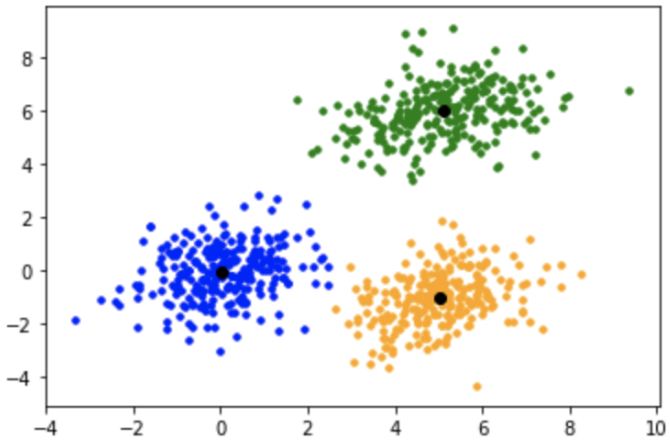


Figure 2.11: k-means clustering model [9]

2.2.5 Random forest

Before explaining the *Random Forest* (RF), it's important to discuss its component: the *Decision Tree* (DT). A DT is a tree-like structure in which each node represents a feature, the branch represents a decision rule, and each leaf node represents an outcome of a decision. Decision trees are widely used in ML for addressing classification and regression problems [6][10].

The Random Forest algorithm is a widely-used supervised ML technique. It operates by aggregating the outputs of multiple DTs to determine outcomes in classification tasks.

A collection of DTs are trained on different subsets of the data, and then the majority outcome is used to make the final prediction. This approach combines the predictive power of numerous trees to enhance the accuracy and robustness of classification results. RF is well-known for its ability to handle complex data without easily overfitting.

Furthermore, it provides valuable insights into feature importance, which will be discussed further in the next chapter in the section on feature selection.

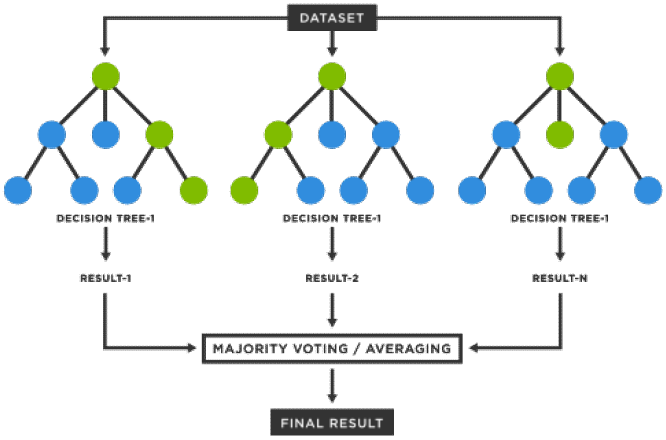


Figure 2.12: The *Random Forest* (RF) model [11]

2.2.6 Multi layer perceptron

The *Multi Layer Perceptron* (MLP) is a type of artificial NN widely used in ML. It can be considered as a basic building block of *Deep Learning* (DL) models. It is a feedforward network composed by neurons, which are the fundamental computational unit, organized in groups called layers [12].

The single neuron, which is a fundamental unit of a neural network, can be visualized as depicted below. It receives as input a vector $\mathbf{x} = [x_1, x_2, \ldots, x_m]$, which is then element-wise multiplied by a vector of weights $\mathbf{w} = [w_1, w_2, \ldots, w_m]$. To this product, the bias b, which is a specific parameter for the layer, is added. Finally, the entire result is passed through an activation function, denoted as $\varphi(z)$, to produce the final output. The whole system can be summarized with this function:

$$y = \varphi\left(b + \sum_{i=1}^{m}(x_i w_i)\right) \tag{2.8}$$

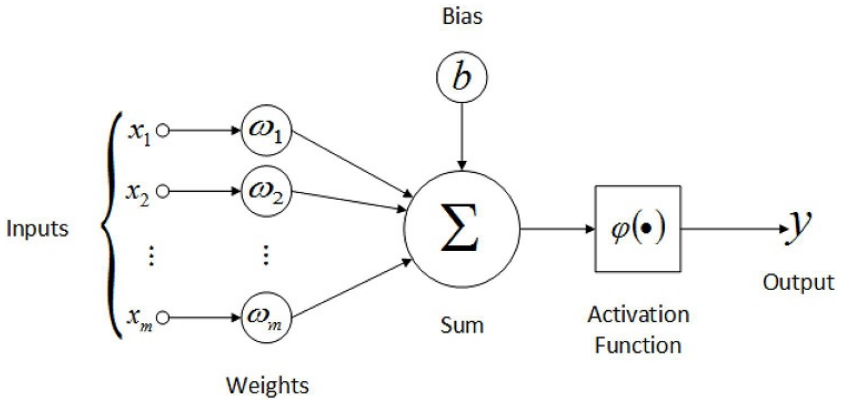


Figure 2.13: Neuron in a *Neural Network* (NN) scheme [13]

Each neuron has an activation function that gives non-linearity to the model. Common activation functions that will be used in the following chapters are the following:

- *Rectified Linear Unit* (ReLu):

$$\mathrm{ReLU}(z) = \max(0, z) \tag{2.9}$$

- Sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.10}$$

- Hyperbolic tangent:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.11}$$

The weights, which are contained in the connections between adjacent layers of neurons, constitute the parameters to be learned throughout the training process. These weights are updated using the fundamental backpropagation algorithm, facilitating the error propagation from the output layer to all neurons within the network.

The MLP stands out as a particular type of NN, where each neuron in a layer is connected to every neuron in the subsequent layers. It finds application in various tasks, such as classification, regression, pattern recognition, image and speech recognition, among others.

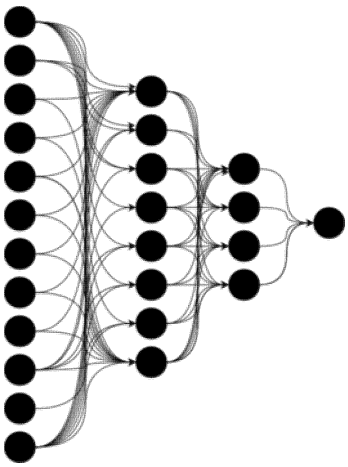


Figure 2.14: The *Multi Layer Perceptron* (MLP) model

The initial layer in an MLP is referred to as the input layer, while the final layer is designated as the output layer. All intermediary layers are known as hidden layers. When an MLP comprises multiple hidden layers (more than one), it is termed a deep neural network. It's important to mention that the more layers present, the greater the system's complexity, which can increase the risk of overfitting.

A significant challenge associated with MLP is their potential for computational intensity and susceptibility to overfitting during training. Often, advanced techniques for regularization and optimization are necessary to address this issue.

In summary, MLP models are among the most widely employed learning models in use today, capable of approximating complex and non-linear functions.

2.2.7 Convolutional neural networks

Convolutional Neural Network (CNN) represents a specialized class of NN primarily applied in image classification tasks due to their exceptional ability to recognize patterns in matrix-like structures, such as pixels in an image [12].

The essence of the CNN are the convolutional layers that make it able to learn automatically the features from the input data. The convolutional layer undertakes a dot product operation between two matrices: one with dimensions defined by the kernel containing learnable parameters, and the other extracted from a receptive field within the input. The convolutional filter g applied with the input signal f at position (x, y) can be expressed in the following equation:

$$\text{Convolution:} \quad (f * g)(x, y) = \sum_i \sum_j f(i, j) \cdot g(x - i, y - j) \tag{2.12}$$

It's crucial to highlight that a consistent set of weights is employed across various sections of the input. This practice effectively reduces the overall number of parameters, resulting in a more optimized model.

A commonly paired layer with CNNs is the Pooling layer, functioning as a filter to reduce the spatial dimensions of the input. For instance, in the case of max pooling, this layer selects the maximum element from a specific region within the feature map, dictated by the filter. The combination of this layer with the CNN make the model more robust to translation and minor deformations of the input data.

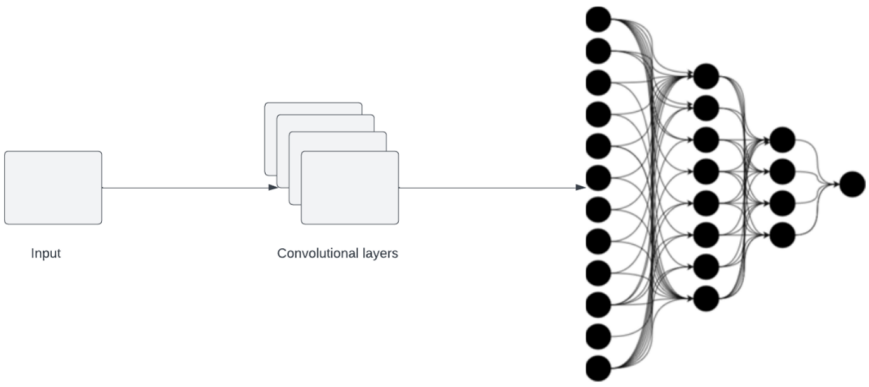


Figure 2.15: The *Convolutional Neural Network* (CNN) model

2.2.8 Autoencoder

The *Autoencoder* (AE), an unsupervised deep learning model, is a particular type of NN. The particularity of this model is that is able to transform the input into a lower dimension and then reconstruct it. In other words, the AE learns a meaningful representations of data by encoding and decoding the input [12].

It is composed by two fundamental components: the encoder, responsible for compressing the input into a lower-dimensional representation known as the latent space, and the decoder, tasked with reconstructing the input based on the information from this latent space.

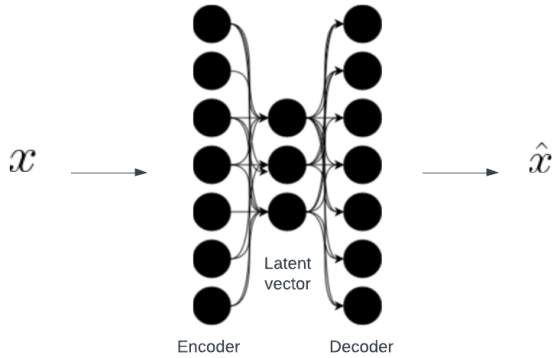


Figure 2.16: The *Autoencoder* (AE) model

The loss function of this model is the difference of the output in respect to the input which is called reconstruction error with x the input and $\hat{x}$ the reconstructed input. There are many losses that can be used, the most common are:

- *Mean Absolute Error* (MAE)

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i| \tag{2.13}$$

- *Mean Squared Error* (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \tag{2.14}$$

- *Root Mean Squared Error* (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2} \tag{2.15}$$

A specific type of AE is the *Variational Autoencoder* (VAE), which enhances its capabilities by incorporating probabilistic elements. VAEs have the unique ability to generate new data samples that resemble those encountered during training.

2.2.9 Long short time memory

Before delving into the explanation of the *Long Short Time Memory* (LSTM), it is essential to discuss the class of *Recurrent Neural Network* (RNN). A RNN is a type of NN specifically crafted to handle sequential data. It employs the hidden state as a form of memory, enabling it to capture information from prior time steps. This memory is leveraged by the network for making future predictions [14]. RNNs utilize a uniform set of weights and biases across all time steps, which contributes to computational efficiency.

The hidden state $h(t)$ at time step t can be expressed with the following formula where $x(t)$ represents the input at time step t, U is the matrix associated with the input, W is the weight matrix, $h(t-1)$ represents the hidden state of the previous time step and f is the activation function.

$$h(t) = f(U \cdot x(t) + W \cdot h(t-1)) \tag{2.16}$$

A particular kind of RNN is the *Bidirectional Recurrent Neural Network* (BiRNN) that consists of two opposite RNN running in opposite direction. Thanks to that the model can capture information from both future and past time steps.

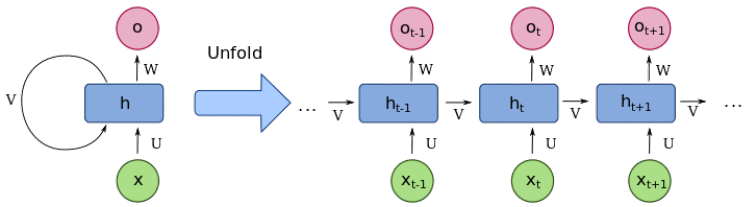


Figure 2.17: *Recurrent Neural Network* (RNN) scheme [15]

The LSTM is a type of RNN that effectively address a critical RNN issue: the vanishing gradient problem. It is composed by the following components:

- Cell state C_t
- Hidden state h_t
- Input gate i_t
- Forget gate f_t
- Output gate o_t

The input gate selects which information to retain from the previous cell state, the forget gate determines what to discard from the previous cell state, and the output gate controls what information is revealed as the output. Together, these gates efficiently manage information flow in sequential data processing.

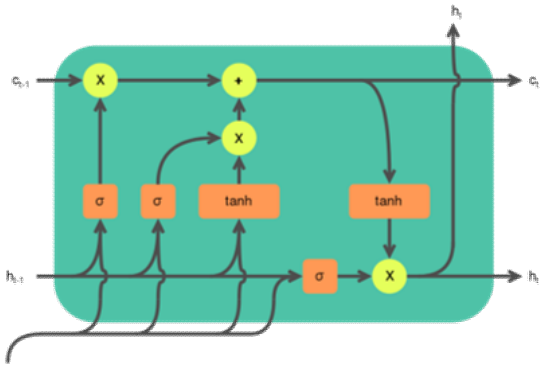


Figure 2.18: The *Long Short Time Memory* (LSTM) model [16]

2.3 Network anomaly detection using machine learning

In this section, we will explore the primary components of a NIDS that leverages ML. We'll also delve into the associated challenges and provide a literature review on this topic.

The initial stage of this system involves parsing network packets into flows. A more comprehensive explanation of this process and the concept of flows will be provided in the following section. Consequently, the total number of flows is split into two segments: one for training and the other for the test dataset. Both datasets require preprocessing, and in some cases, a feature selection process, which will be elaborated upon in the subsequent chapter.

Once the model is trained, it is subjected to testing using the test set. Many NIDS that employ ML are designed to address binary classification problems. As a result, the final output of the trained model, when provided with a flow, will be the classification decision between "normal" and "anomalous" traffic.

When the classification is "anomalous", it will initiate an alert to notify the administrator. The diagram depicted in the following image provides a visual representation of the description we've just outlined, representing the most generalized form of a NIDS employing ML.

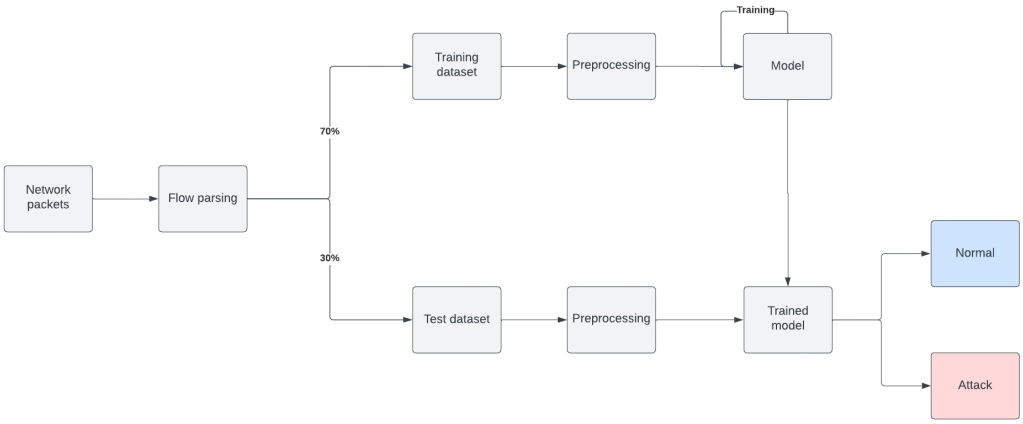


Figure 2.19: Network anomaly detection using *Machine Learning* (ML) scheme

2.3.1 Network flow

As demonstrated in the preceding section, the initial process following the capture of network packets involves parsing them into flows. A flow, in this context, serves as a statistical representation of a collection of packets within a defined time window, identified by a minimum of two key parameters: *Internet Protocol* (IP) addresses and ports. In some implementations, additional parameters such as the IP protocol number and *Type Of Service* (TOS) may also serve as identifiers. The general concept of a network flow is visually depicted in the image below [17][18].

Figure 2.20: Network flow scheme

It is possible to see the typical features that are collected in a network flow by taking the example of the NSL-KDD dataset [19] which is a typical dataset used for IDS that will be explained in the following chapter:

- Content: features related to the payload of the packets.
- Connection: features related to the network connection between two hosts.
- Traffic: Quantitative data about the amount of data exchanged during the flow. It provides an overview of network activity of the flow.
- Host: attributes related to the individual devices involved in the flow.

The absence of a standardized approach for generating network flows leads to significant variations in the features collected within datasets. This divergence can result in markedly different outcomes when assessing the same model's performance on identical network packets but with varying methods of flow parsing. However, it is a prevailing practice employed by the majority of NIDS due to its convenience as the simplest method for aggregating network packets.

If we consider real IDS implementations and not just benchmarking on training and testing dataset there is a common way to parsing the network packets into flows. The *NetFlow* protocol is a traffic monitoring technology developed by Cisco [18]. It works by collecting and exporting network traffic data from network devices to a central collector for the processing and analysis. The typical features extracted by *NetFlow* are:

- Source and destination IP addresses.
- Source and destination port numbers.
- Layer 3 protocol field.
- TOS.

The NetFlow architecture comprises three essential components: the exporter, responsible for generating flow records and transmitting them to a collector; the collector, which stores and aggregates incoming flows; and the console, the application used for displaying and analyzing flow data, as described in [18].

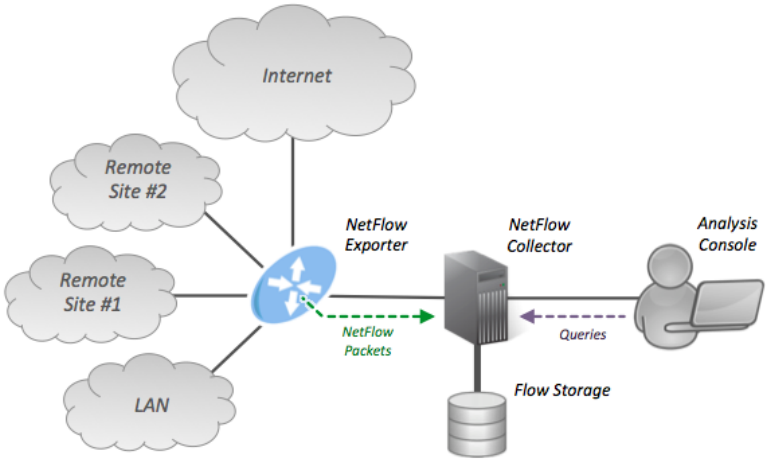


Figure 2.21: *NetFlow* architecture [20]

2.3.2 Challenges

ML-based IDS face a range of challenges. Firstly, the FPR increases in proportion to accuracy, which presents a significant challenge, potentially undermining the viability of practical implementations. Secondly, the adoption of state-of-the-art techniques often involves combining diverse ML models, leading to a substantial increase in computational costs. Moreover, acquiring a sizable, balanced, and up-to-date dataset is imperative for developing an effective IDS but can be challenging to fulfill. Finally, expressing a realistic lower bound for attack detection in an ML-based IDS is intricate, as this value heavily depends on the test dataset used for model evaluation.

2.3.3 Literature review

The final section of this chapter is devoted to a comprehensive literature review of developments in the field of network anomaly detection using ML. To gain insight into current trends, various surveys have been consulted as references, including [5], [21], and [22].

One challenge in some of the surveyed literature is the comparison of models tested on different datasets. This prompted the decision, as detailed in the following chapter, to implement fundamental models as reference benchmarks for a more consistent evaluation approach.

Broadly speaking, the prevailing trend in network anomaly detection, as documented in [21], is the adoption of DL models, which have now surpassed classical ML approaches. The models reviewed in this analysis primarily include CNN, LSTM, AEs, and hybrid models that combine elements from these. As computational power becomes increasingly accessible, the proposed IDS models are growing in complexity.

Several articles considered in this review utilize datasets that will be explained in the subsequent chapter of Methodology, specifically in the subsection dedicated to datasets. Readers seeking more detailed information about these datasets are encouraged to refer to section 3.2.

The work by [23] introduces a model featuring a sequence of CNNs and max pooling layers to extract optimal features. This is followed by two LSTM layers for classification. The results obtained on the NSL-KDD dataset are remarkable, showing an accuracy of 0.99 in the custom test set offered by the dataset: the NSL-KDDTest+. However, the analysis is incomplete as several other crucial metrics, including the FPR, are missing. Moreover, an adversarial ML analysis is notably absent, leaving room for further investigation and comprehensive evaluation.

In the research presented by [24], a comprehensive analysis delves into the performance of AEs for anomaly detection. This exploration specifically focuses on understanding the interplay between the model's performance and its latent size. The study encompasses multiple datasets, including the NSL-KDD dataset. The highest accuracy obtained through this study is an impressive 0.88, achieving a low FPR of 0.066. Also here, the test set used for the evaluation of the model is the original one: NSL-KDDTest+. The selection of the optimal latent size and network dimensions, informed by a comparison of the *Matthew's Correlation Coefficient* (MCC) and F1 score, shows the potential to enhance model performance through a thoughtful analysis of the latent space dimensions and network size.

The research presented in [25] introduces a model tailored for anomaly detection within *Internet of Things* (IoT) device networks using sparse AEs. IoT devices are an ideal use case for anomaly detection due to their predictable traffic patterns. The study involves custom feature extraction based on network flows. Multiple sparse AEs are trained to understand the legitimate communication profiles for each type of IoT device present in the network. During operation, the network traffic is fed to this set of sparse AEs, with each one providing a result based on its detection threshold. A communication event is classified as legitimate if at least one AE identifies it as such.

In the study outlined in [26], a novel model is proposed, comprising two AEs. One AE is trained using normal samples, while the other is trained using malicious samples. The reconstructed samples from both AEs traverse a series of dense layers, resulting in embedded representations. This process yields two embedded representations for the samples and an additional one referred to as the "anchor", representing the samples without the AEs. Subsequently, the distance between the anchor and the other two representations is computed. If the anchor is closer to the embedded representation of the normal reconstructed sample, the sample is classified as normal, and vice versa. This model represents a fusion of AEs and triplet networks, showing an innovative approach to anomaly detection.

The research presented in [27] introduces a model built upon the concept of *Deep Autoencoding Gaussian Mixture Model* (DAGMM). Initially, the input undergoes deep encoding to transform it into a latent space, followed by reconstruction using a decoder. This process results in a latent representation of the input. Subsequently, the latent representation is fed into a *Gaussian Mixture Model* (GMM). The GMM is a model designed to learn a combination of Gaussian distributions that effectively represent the data. Through this approach, the model considers the reconstruction error and employs outlier detection based on the Gaussian distribution, utilizing the Mahalanobis distance as a metric. This unique integration allows for robust anomaly detection within the dataset.

The research in [28] presents an enhanced version of DAGMM: SIGMOD [27] achieved by incorporating stacked sparse AEs and GMM with feature selection using Pearson correlation. The model's evaluation is conducted on the UNSW-NB15 dataset [29]. As elucidated in the paper, the model boasts several improvements: training exclusively on normal data samples, feature selection based on Pearson correlation, imposition of sparsity via stacked sparse AE, and refining the GMM model using Choelsky decomposition to compute sample energy. The model demonstrates superior performance compared to state-of-the-art NIDS. However, the paper does not delve into an analysis of the model's vulnerability to adversarial ML.

The work outlined in [30] introduces an innovative model composed of an ensemble of AEs. In a nutshell, the model generates various subsets from an input sample, each containing different features. Each of these subsets serves as input to an AE. The overall anomaly score is derived by considering the output from all the AEs. Kitsune transcends the standard anomaly detection model; It involves custom feature extraction directly from network packets. This distinctive feature makes it challenging to directly compare its performance with other models trained on the NSL-KDD dataset. The incorporation of custom feature extraction sets Kitsune apart, showing its uniqueness in the realm of anomaly detection.

The research presented in [31] introduces a model that employs *Recursive Feature Addition* (RFA) with SVMs for optimal feature selection within the NSL-KDD dataset. Following this feature selection process, detection is carried out using an AE trained on benign samples. However, the model lacks an in-depth analysis of FPR performances, which tend to be suboptimal in AEs. Furthermore, an adversarial ML analysis is notably absent from the study. Incorporating such analyses could provide a more comprehensive understanding of the model's performance and its robustness against adversarial attacks.

The table below displays selected metrics from models in the existing literature that have served as benchmarks for this study. It's worth noting that some of the studies referred to have been intentionally omitted from this list. This omission can be attributed to either their use of custom datasets or the unavailability of sufficient information regarding the metrics used. Nevertheless, these studies have played a significant role in shaping the research, as the underlying frameworks and methodologies have been valuable and insightful.

Model	**Dataset**	**Accuracy**	**FPR**	**F1**
AE [24]	NSL-KDD	0.88	0.06	0.89
CNN-LSTM [23]	NSL-KDD	0.99	/	/
AE-RFA [31]	NSL-KDD	0.91	/	/
DAGMM [27]	NSL-KDD	0.93	/	0.93
RENOIR [26]	CIC-IDS2017	0.98	/	0.95
SIGMOD [28]	UNSW-NB15	0.96	/	0.96

Table 2.1: Literature models comparison

2.4 Summary

By the chapter's conclusion, readers will have gained a comprehensive view of the fundamental concepts essential for comprehending the forthcoming research sections. The subsequent chapter will delve into the methodologies employed for conducting research in this domain. It will involve an examination of each component of the anomaly detector previously outlined. Additionally, the following section will witness the practical implementation of the various ML models introduced for empirical evaluation.

The literature review, a fundamental component of this study, will play a crucial role in making comparisons in the subsequent chapters. The models to be implemented in the results chapter, as well as the methodologies employed, are built upon the foundation laid by these essential studies and surveys, forming the cornerstone of this research.

Chapter 3

Methodology

In this chapter, we will explore the methodologies employed in this study to obtain the results presented in the subsequent chapter.

The chapter starts with an overview of the tools utilized in this project. This information is of greatest importance as it provides the reader with the means to reproduce the results that will be presented.

Following this, we provide a comprehensive explanation of the datasets considered: the CIC-IDS2017 and the NSL-KDD. For each of these datasets, we will show their composition, strengths, and weaknesses.

Next, we delve into the processing methodologies employed in the initial stage of anomaly detection, including data cleaning, normalization, and numericalization.

Subsequently, we elucidate the techniques used for feature selection, a critical step in the success of a ML model. To offer a comprehensive understanding, we will cover various types of feature selection methods: filter, wrapper, and embedded methods. Specific techniques employed will be explained in detail.

Finally, at the conclusion of this chapter, we will elaborate on the models chosen for implementation and how they were evaluated. This section is particularly crucial as it instructs the reader on how to interpret the results in the following chapter.

3.1 Tools

Throughout this study, a range of open-source tools have been harnessed for the development of ML models. These tools are accessible to the public, promoting transparency and collaboration.

Python [32] serves as the primary programming language for creating ML algorithms. It's widely adopted in this field due to its user-friendly nature and the extensive library ecosystem. Notable libraries employed include *NumPy* [33] and *Pandas* [34] for data analysis and management. For data visualization, we opted for graphic libraries such as *matplotlib* [35] and *Seaborn* [36]. Additionally, *scikit-learn* [37] proved invaluable for extracting metrics from model training and testing.

The ML framework of choice is *TensorFlow* [38], appreciated for its simplicity and high degree of customization. The availability of comprehensive documentation further enhances its appeal. Additionally, *TensorFlow* is precisely engineered for high performance and optimization, resulting in significant speed improvements during the training of all machine learning models. The inclusion of TensorBoard, a powerful monitoring tool provided by the framework, has played a pivotal role in fine-tuning model parameters, contributing to the overall efficiency of the training process.

To facilitate the execution of model training and testing, a cloud platform was indispensable for computational efficiency. Google Colab [39] emerged as a suitable choice due to its cost-effectiveness and customizable resource allocation options.

3.2 Dataset

The dataset is the central point for the success of a ML algorithm. We have conducted a deep research about which are the available dataset for training a ML model for network anomaly detection. As explained in the previous chapter, the dataset is composed by network flows which are a statistical representation of groups of network packets.

Before delving into the explanation of the datasets used in this study, it's crucial to outline the key criteria that distinguish a valuable dataset. This step is of significance as it will aid in comprehending the rationale behind the selection of the datasets employed.

Foremost, an ideal dataset should exhibit a substantial size and encompass a diverse range of data types. In the context of network anomaly detection, this criterion holds particular significance. It signifies exposure to a variety of attack types for effective learning and a more comprehensive understanding of normal network behavior. While this principle may seem straightforward, its fulfillment has proven to be a formidable challenge in the realm of network anomaly detection.

Furthermore, it is imperative that the dataset maintains a balance between the various attack types and normal packet categories. Such equilibrium is vital, as an imbalance in the number of samples across classes can introduce biases during the training phase. These biases, if left unaddressed, can significantly obstruct the model's performance when it undergoes testing and evaluation.

Equally crucial is the realism of the collected data. Many datasets for IDSs predominantly feature synthetic data. This propensity, at times, leads to the development of models that demonstrate strong performance within their test sets. However, upon deployment in real-world scenarios, the performance of such models may see a dramatic deterioration.

The final point, often overlooked in existing literature, underscores the significance of a challenging test set relative to the training data. It is crucial that the test set comprises entirely novel data that the trained model has never encountered before. This aspect holds paramount importance because without a rigorous challenge, benchmarking the model and even performing parameter tuning becomes a difficult task.

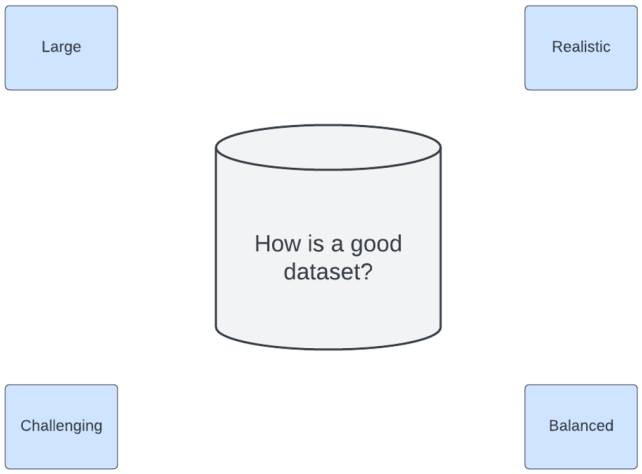


Figure 3.1: Requirements for a good dataset

3.2.1 CIC-IDS2017

The CIC-IDS2017 dataset [40] stands out as one of the most extensive datasets ever created for NIDS. It's noteworthy to mention that all the traffic within this dataset is simulated, yet it remains highly realistic. Additionally, there exists a larger variant of the same dataset known as CIC-IDS2018, featuring a diverse array of features. For computational efficiency, the decision was made to utilize the smaller version.

The dataset comprises multiple files, each representing distinct days of network activity. Each day exhibits a unique profile of network traffic, encompassing normal samples and specific types of attacks. This temporal diversity proves valuable for testing models as it allows for an exploration of traffic's evolution over time.

In the table below, the corresponding instances of malicious network activity for each day within the dataset are enumerated. This provides insight into the distribution of anomalous events alongside regular traffic patterns for each day.

Day	**Activity**
Monday	No attacks
Tuesday	Brute Force
Wednesday	DOS and Heartbleed
Thursday morning	Web attacks
Thursday afteroon	Infiltration
Friday morning	DDOS
Friday afternoon	Port scan and DDOS

Table 3.1: Days in the CIC-IDS2017

The dataset has already been parsed using the *CICFlowMeter* tool [41], specifically extracting flow features. Flows are identified based on source IP and port, destination IP and port, and the protocol, collectively forming the FlowID. A comprehensive analysis of *CICFlowMeter* can be found in [42]. The complete set of features extracted by *CICFlowMeter* is available for reference [43].

The dataset encompasses eight types of samples: benign (normal packets), DOS, DDOS, bot traffic, brute force, port scan, infiltration, and heartbleed.

It's evident that some of the attacks within the dataset are not adequately represented, potentially affecting the performance of attack classification. However, given that the research primarily focuses on anomaly detection rather than classification, specific data augmentation techniques to balance the dataset were deliberately omitted.

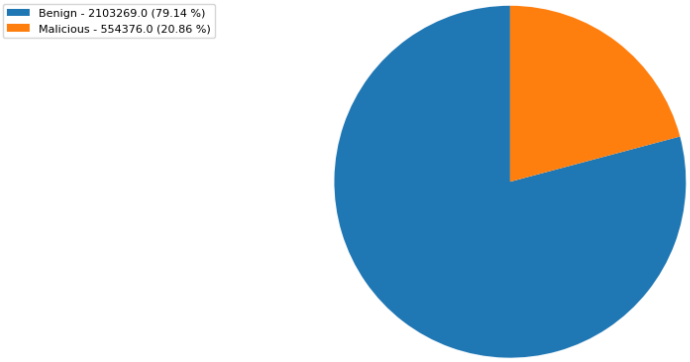


Figure 3.2: Benign and malicious samples in CIC-IDS2017

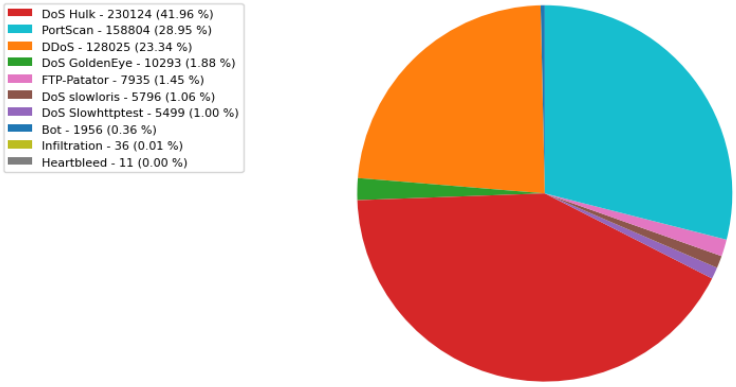


Figure 3.3: Attacks in CIC-IDS2017

In conclusion, this dataset offers numerous advantages, primarily notable for its extensive sample size and diverse range of attack types. However, a significant drawback is the absence of a dedicated test set, necessitating the utilization of a portion of the overall dataset for testing purposes. This poses a challenge due to the dataset containing numerous duplicate samples and attack flows with low variance. Consequently, the test set does not entirely consist of previously unseen data. It's important to consider this aspect when evaluating the results of various ML models using this dataset.

3.2.2 NSL-KDD

The NSL-KDD dataset [19] represents an enhancement of the KDD 99 dataset, addressing issues related to imbalances and redundancies as elucidated in the article [44]. It comprises 42 features categorized into four types: *Transmission Control Protocol* (TCP) connection attributes (B), content features (C), traffic features within a two-second time window (T), and host features for attacks lasting two seconds or longer (H) [45]. The following table shows all the features present in this dataset.

Sr. No	Label	Attribute Name	Sr. No	Label	Attribute Name	Sr. No	Label	Attribute Name	Sr. No	Label	Attribute Name
1	B	duration	10	C	hot	23	T	count	32	H	dst_host_count
2	B	protocol_type	11	C	num_failed_logins	24	T	serror_rate	33	H	dst_host_srv_count
3	B	service	12	C	logged_in	25	T	rerror_rate	34	H	dst_host_same_srv_rate
4	B	src_bytes	13	C	num_compromised	26	T	same_srv_rate	35	H	dst_host_diff_srv_rate
5	B	dst_bytes	14	C	root_shell	27	T	diff_srv_rate	36	H	dst_host_same_src_port_rate
6	B	flag	15	C	su_attempted	28	T	srv_count	37	H	dst_host_srv_diff_host_rate
7	B	land	16	C	num_root	29	T	srv_serror_rate	38	H	dst_host_serror_rate
8	B	wrong_fragment	17	C	num_file_creations	30	T	srv_rerror_rate	39	H	dst_host_srv_serror_rate
9	B	urgent	18	C	num_shells	31	T	srv_diff_host_rate	40	H	dst_host_rerror_rate
			19	C	num_access_files				41	H	dst_host_srv_rerror_rate
			20	C	num_outbound_cmds				42	-	class
			21	C	is_hot_login						
			22	C	is_guest_login						

Figure 3.4: NSL-KDD Features [19]

A notable feature of this dataset is its inherent division into distinct training and test sets, each residing in separate files. There are two test sets available: the NSL-KDDTest+ and the NSL-KDDTest-21. The latter is a subset of the first one, excluding the most challenging traffic records. As exemplified in the subsequent pie charts, the test set notably stands out due to its formidable challenge, stemming from the presence of a diverse variety of attacks that were not encountered during the training phase. This remarkable characteristic is a key strength of the dataset, although its relatively modest size represents a relevant weakness.

In the test set, there are 9,711 samples representing the “normal” class and 12,814 samples representing the “attack” class. Instead, in the training set, there are 67,325 samples for the “normal” class and 58,468 samples for the “attack” class.

The NSL-KDD dataset comprises a diverse set of network traffic data, encompassing both normal and attack patterns. Notably, the attacks within this dataset are categorized into four distinct classes: DOS, Probe, *User to Root* (U2R), and *Remote to Local* (R2L).

Attack	**Description**
DOS	Denial of service attacks
Probe	Attacks for vulnerability assessment
U2R	User to root attacks: priviledge escalation
R2L	Remote to local attacks: exploits

Table 3.2: Attack classes description in NSL-KDD

It is important to acknowledge that the attacks present in the NSL-KDD dataset remain synthetic in nature like the one in the CIC-IDS2017. This is due to the dataset's origin as a refined version of the KDD 99 dataset, which was entirely generated for research purposes. Due to the age of the KDD 99 dataset, it is crucial to approach its use with caution, recognizing that it may not accurately reflect contemporary real-world network intrusion scenarios.

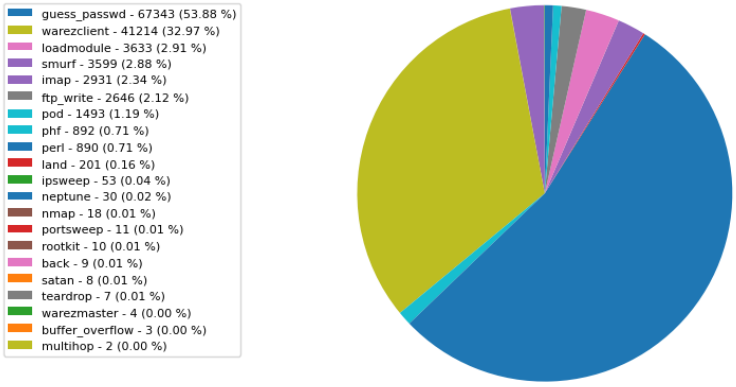


Figure 3.5: Training set of NSL-KDD

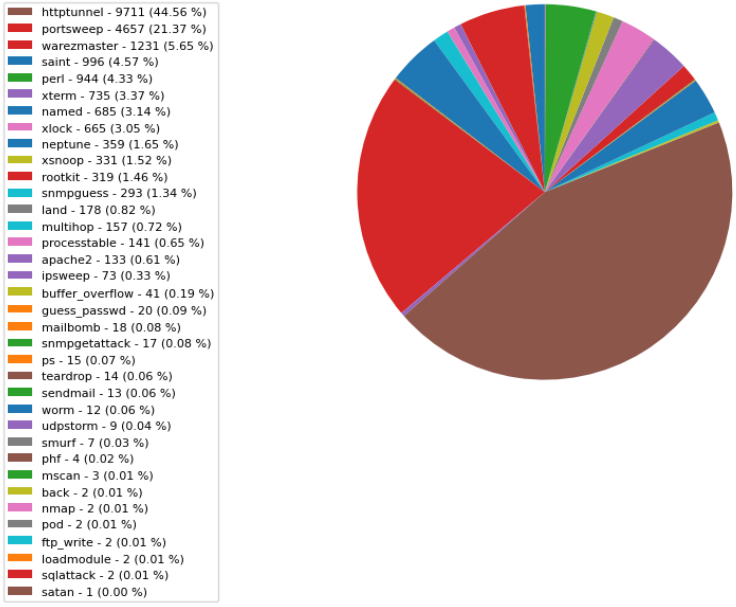


Figure 3.6: Test set of NSL-KDD

An alternative approach to show the complexity of this dataset is by employing *t-distributed Stochastic Neighbor Embedding* (t-SNE) [46] visualization, which effectively reveals the level of non-linearity within the data. This characteristic becomes notably apparent in the following image, where a pronounced overlap between the "normal" and "attack" classes is clearly evident.

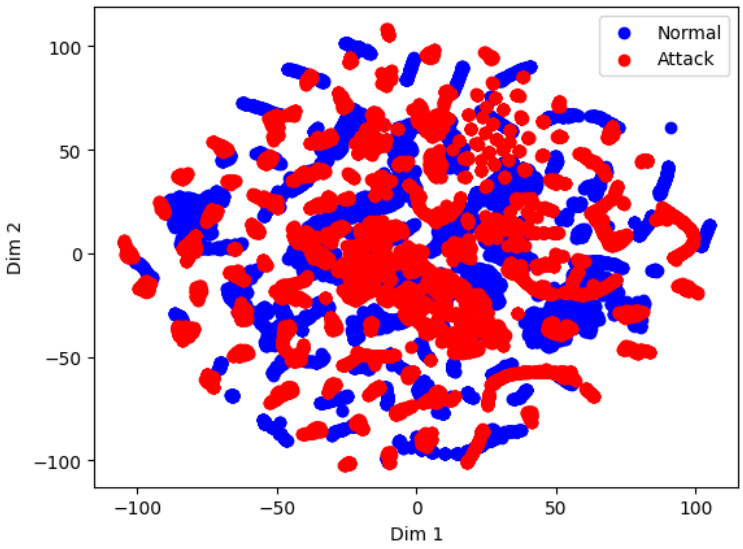


Figure 3.7: NSL-KDD *t-Distributed Stochastic Neighbor Embedding* (t-SNE) visualization

3.2.3 Comparison

Several notable distinctions exist between these two datasets. Firstly, the CIC-IDS2017 dataset impressively captures a deep understanding of diverse application models, network devices, and protocols, rendering it exceptionally realistic despite being entirely generated. Being one of the most recent datasets available, it holds significant relevance for the development of NIDS [47].

In contrast, the NSL-KDD dataset represents a network environment that may appear less reflective of today's real-world scenarios. Its predecessor, the KDD 99, was created back in 1999, rendering it somewhat outdated.

As mentioned earlier, the primary drawback of the CIC-IDS2017 dataset lies in its data redundancy, which simplifies the learning process for classifiers. Consequently, it may not serve as a robust benchmark in the field of intrusion detection. Conversely, the NSL-KDD dataset proves to be more challenging and, thus, a more reliable benchmark.

For this research, a deliberate choice was made to evaluate models on both these datasets, offering a comprehensive perspective on model performance across varying degrees of complexity and realism.

3.3 Preprocessing

Preprocessing stands as a fundamental cornerstone in the realm of ML applications [6]. It includes an array of techniques precisely designed to facilitate the learning process of the model. These methods are applied to the raw dataset, wielding a substantial influence over the model's overall performance. It involves techniques such as the cleaning, transformation, normalization and balancing of the dataset.

3.3.1 Cleaning

One of the initial and standard procedures when working with a ML dataset is data cleaning. This process involves the removal of null values and outliers, which can distort the data distribution due to potential errors. It is crucial to note that these datasets are synthetically generated, which makes encountering errors quite common.

Additionally, the network parser, such as *CICFlowMeter* [41], may occasionally introduce format errors in the CSV files, necessitating careful handling and rectification during the data cleaning process.

3.3.2 Numericalization

Several features within the dataset are non-numeric, and these need to be encoded into numerical values. Additionally, for enhanced performance when dealing with categorical data, employing the one-hot encoding technique is beneficial. This approach involves creating a new feature for each category, marking the feature corresponding to the specific category as one if present in a sample.

One particular type of numericalization is the one hot encoding technique [6]. This approach is able to encode a categorical variable with n possible values into a binary vector of length n where only one element is 1 and all the others are 0. In a mathematical way, any generic element of it x_i of the binary vector of the category $X = [x_0, x_1, ..., x_n]$ can be expressed as follow:

$$x_i = \begin{cases} 1 & \text{if } X \text{ encodes the category } i \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

3.3.3 Normalization

Normalization plays a pivotal role in dataset preprocessing [6]. When variables are measured on disparate scales, their contributions to model training are unequal. To address this, normalization is performed using the *MinMaxScaler* function from the *scikit-learn Python* library [37]. This function scales all feature variables to fall within the range of 0 and 1, ensuring a level playing field during model training. This can be expressed in the following equation where $\hat{x}_{i,f}$ is the value of the feature f of the sample i normalized and the min and max function are applied among all the values of the feature f.

$$\hat{x}_{i,f} = \frac{x_{i,f} - \min(X_f)}{\max(X_f) - \min(X_f)} \tag{3.2}$$

3.4 Feature selection

Feature selection plays a pivotal role in this study. It is a widely adopted technique within the realm of ML, often regarded as an integral part of the preprocessing phase. Its significance in our research is emphasized by dedicating a section to this critical process.

The primary objective of feature selection is to enhance model performance by carefully choosing a subset of the available features. Reducing the dimensionality of the data not only boosts model efficiency in terms of computation time but also results in improved predictive capabilities. This improvement derives from the identification and removal of features that may prove redundant or even detrimental to the decision-making process [48] [6].

Reducing the number of features serves as a valuable means to combat the overfitting challenge, a well-documented issue in high-dimensional data scenarios. In essence, this feature reduction scales down the complexity of the problem, ultimately promoting model generalization.

Furthermore, the act of simplifying the model through feature selection imparts clarity and comprehensibility, facilitating the interpretation of the intricate connections between the features and the predicted outcomes. Nevertheless, it's noteworthy that, especially with deep neural networks, an element of "black box" nature often endures, even when the feature set is reduced.

The subsequent subsection outlines the primary methodologies employed for feature selection in this study.

3.4.1 Filter methods

This feature selection approach involves evaluating each feature's individual characteristics to rank and identify their relative importance. Unlike other methods we'll discuss later, this approach does not assess the features in relation to any ML model to determine their significance [48].

Various approaches rely on statistical measures to assess the relationship between individual features and the target variable, often denoted as the "attack" or "normal" label. Consequently, conducting this type of analysis necessitates the availability of a labeled dataset.

Pearson correlation

The Pearson correlation algorithm is a classic feature selection method widely employed in the literature. Notably, a recent research paper applied this technique for feature classification in the context of IDS [49][28]. It is a statistical measure used to quantify the linear relationship between two variables. It is commonly used as a filter method for feature selection to understand how individual features relate to the target variable. The following equation defines the Pearson correlation coefficient between two variables X and Y where $\bar{X}$ and $\bar{Y}$ are their respective means.

$$r_{XY} = \frac{\sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n} (X_i - \bar{X})^2 \sum_{i=1}^{n} (Y_i - \bar{Y})^2}} \tag{3.3}$$

The Pearson correlation coefficient can assume values within the range of -1 (negative correlation) to 1 (positive correlation).

3.4.2 Wrapper methods

Wrapper methods are often regarded as computationally intensive approaches for feature selection [48]. This is due to their systematic search for the optimal subset of features that yields superior performance in a ML model.

These methods utilize a scoring mechanism to assess the ML algorithm's performance concerning the selected feature subset. They iteratively explore various feature combinations until they identify the best-performing one. However, it becomes evident that this approach is impractical when dealing with a large number of features.

Wrapper methods offer several advantages. They are known for their high accuracy and their ability to account for feature interactions. Moreover, the results are intricately tied to the model's performance, providing a strong guarantee of success. However, as previously mentioned, they can be computationally demanding and may potentially lead to overfitting of the model.

These methods are included in the discussion for the sake of comprehensiveness, but they have not been widely employed due to their high time complexity. Examples of such methods include sequential feature selection and genetic algorithms.

3.4.3 Embedded methods

Embedded methods are a class of feature selection techniques tailored to specific ML algorithms [48]. With these methods, feature selection becomes an integral part of the model training process.

One notable example of embedded methods is the *Recursive Feature Elimination* (RFE), which applies iteratively across various machine learning algorithms. RFE trains the model and progressively removes the least important features in each iteration until certain performance criteria are met.

Tree-based algorithms, such as RF and Gradient Boosting, are frequently employed for embedded feature selection. One notable algorithm in this category is *XGBoost*, which will be elaborated upon in the upcoming sections.

One of the key advantages of embedded methods is their computational efficiency since they seamlessly integrate feature selection into the training process. Furthermore, similar to wrapper methods, embedded methods are capable of capturing feature interactions.

However, they do have some limitations. They may not offer the same level of precision as the exhaustive search in wrapper methods. Additionally, embedded methods are inherently tied to the specific model being used, meaning that the selected features may be closely associated with that model. Consequently, when applying the chosen feature subset to different algorithms, the results can vary.

XGBoost

XGBoost [50] is a library that implements the gradient boosting decision tree algorithm for classification. The algorithm creates n decision trees, the *Feature Importance* (FI) is the counter of how many times a specific feature is used to split the tree during the training.

3.5 Model

We have opted to implement various ML models, both supervised and unsupervised, following a thorough review of the literature on the subject. Our model selection process centered on choosing the most widely employed approach for anomaly detection. Additionally, we considered the importance of incorporating diverse models into our study, encompassing traditional ML models like RF and k-means, as well as more intricate DL models such as MLP, CNN, and AE. Furthermore, we included a type of RNN, specifically LSTM.

This comprehensive approach allowed us to explore a broad spectrum of ML models, each distinct in terms of its architectural design. We believed that this strategy was best suited to cover the extensive landscape of ML.

In the following image, you can observe all the models that have been implemented, categorized into two groups: supervised and unsupervised. It's evident that the unsupervised models constitute the minority, which is not surprising, considering that training a model for classification with an unlabeled dataset is generally more intricate.

It's essential to note that the distinction between supervised and unsupervised models extends beyond dataset labeling. In the context of network anomaly detection, unsupervised models seek to discern anomalies by contrasting them with the normal behavior of the network, with the AE being a prime example of this approach.

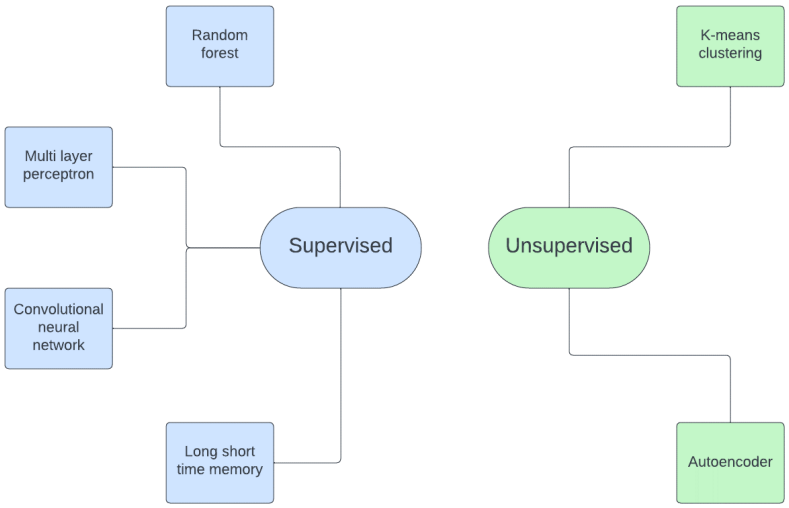


Figure 3.8: Models implemented scheme

3.5.1 Evaluation

For every model implemented, it has been decided to compute all the metrics explained in the background theory chapter in the section of ML, such as accuracy, FPR, and F1 score. Additionally, for each model, the confusion matrix will be computed and displayed in a normalized form.

Each model implemented will be tested on two datasets explained at the beginning of this chapter: the CIC-IDS2017 and the NSL-KDD. This choice provides the research with a more comprehensive view of the model's performance.

To present the performance data more clearly, the reader will find the two tables below for each model implemented. The first table displays the overall performance across the two datasets, followed by a second table that provides more specific performance metrics for each dataset.

Dataset	**Accuracy**	**FPR**
CIC-IDS2017		
NSL-KDD		

Table 3.3: Example of general metrics table

Class	**Precision**	**Recall**	**F1**	**Support**
Normal				
Attack				

Table 3.4: Example of metrics table

3.6 Summary

In this chapter, we have delved into the fundamental methodologies that will be applied in the subsequent chapter to yield the results of our study. This chapter plays a pivotal role in our research, as it rigorously outlines the path we have taken to reach these results. It serves as the bedrock of rigor and robustness in our investigation.

As we conclude this chapter, our readers are now well-prepared to explore the quantitative results of the various models in the next chapter. The comprehensive delineation of the two datasets offers the reader a thorough grasp of their composition, thereby providing a clearer perspective on the results to be presented in the subsequent chapter.

Chapter 4

Results

In this chapter, we will comprehensively analyze the results obtained in this study. The results will be presented using the methodologies described in the preceding chapter.

The chapter will commence with an examination of the results from the feature selection process, as detailed in the previous chapter. Specifically, we will show the outcomes of the Pearson correlation and *XGBoost*, a filter and an embedded method.

Following that, the chapter will progress to present the results achieved by each model implemented on both datasets: the CIC-IDS2017 and the NSL-KDD. For each dataset, a brief commentary on the attained metrics will be provided.

Subsequently, we will conduct a concise comparison of all the results obtained. This pivotal juncture is of significant importance as it elucidates the rationale behind the author's decision to implement the proposed framework, as outlined in the subsequent section.

The chapter culminates with a comprehensive explanation of the proposed framework, providing a detailed breakdown of its composition and strengths. The framework is then subjected to benchmarking on the NSL-KDD dataset, wherein not only quantitative results but also an in-depth exploration of all its constituent components are elaborated upon.

4.1 Feature selection

In this section we will show the results obtained in the feature selection part. We have decided to implement two algorithms for feature selection. The first is the Pearson correlation which is a filter method. The second is a more complex algorithm: *XGBoost* which is an embedded method. In the last subsection we will delve into some considerations about the results of the two methods implemented.

4.1.1 Pearson correlation

The image below illustrates the Pearson correlation coefficient between the label and each feature within the CIC-IDS2017 and NSL-KDD dataset. We have chosen to show only those features with an absolute correlation coefficient greater than 0.15. This selection is deliberate, as presenting all the features, in particular for the CIC-IDS2017 which are 88, could potentially overwhelm the reader.

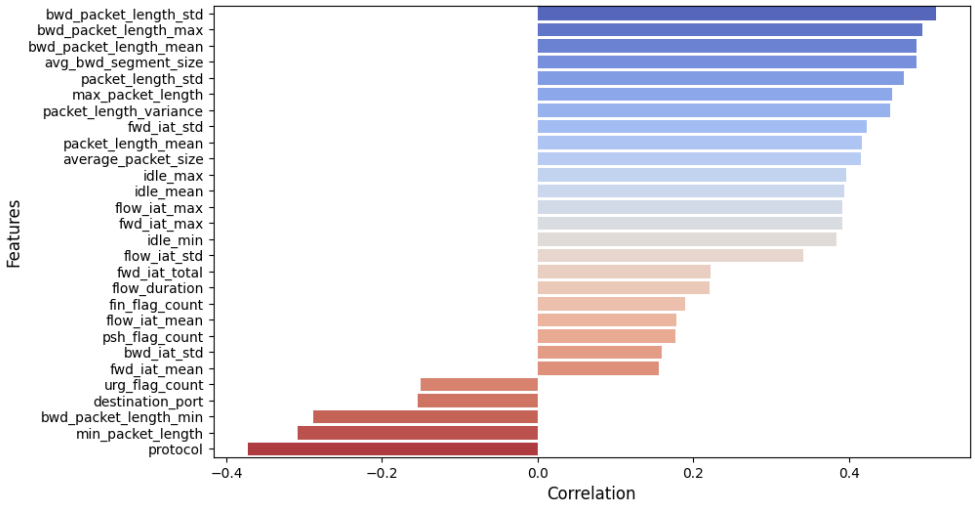


Figure 4.1: Pearson correlation in CIC-IDS2017

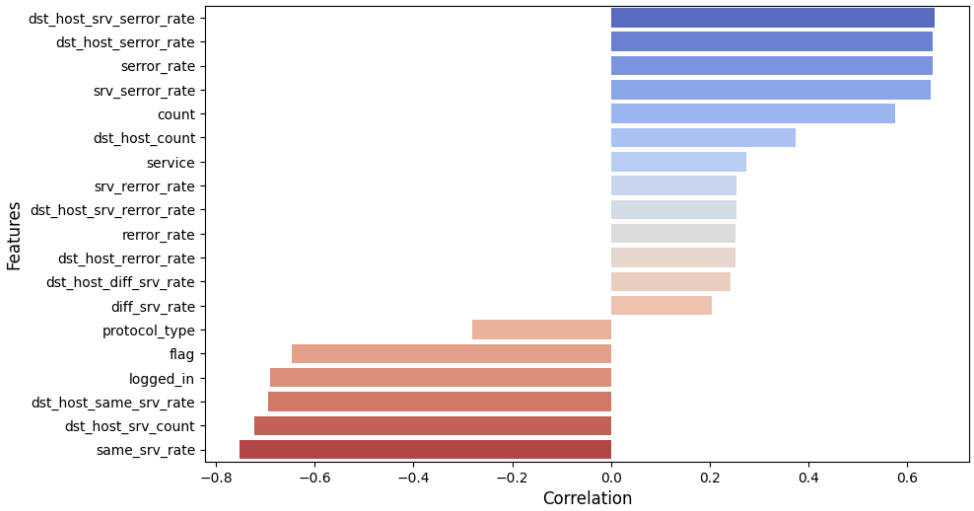


Figure 4.2: Pearson correlation in NSL-KDD

4.1.2 XGBoost

The following two images show the outcomes achieved by employing the *XGBoost* algorithm on two distinct datasets. Notably, we observe that certain features exhibit significantly greater importance compared to others. This distinction is notably more pronounced compared to the results obtained using the Pearson correlation.

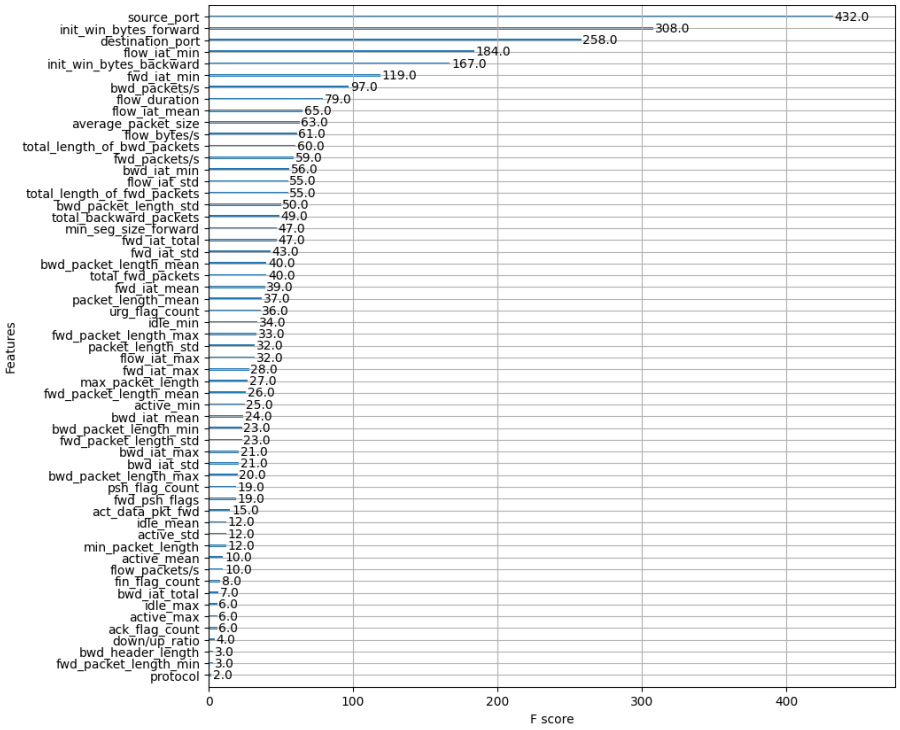


Figure 4.3: *XGBoost* feature ranking in CIC-IDS2017

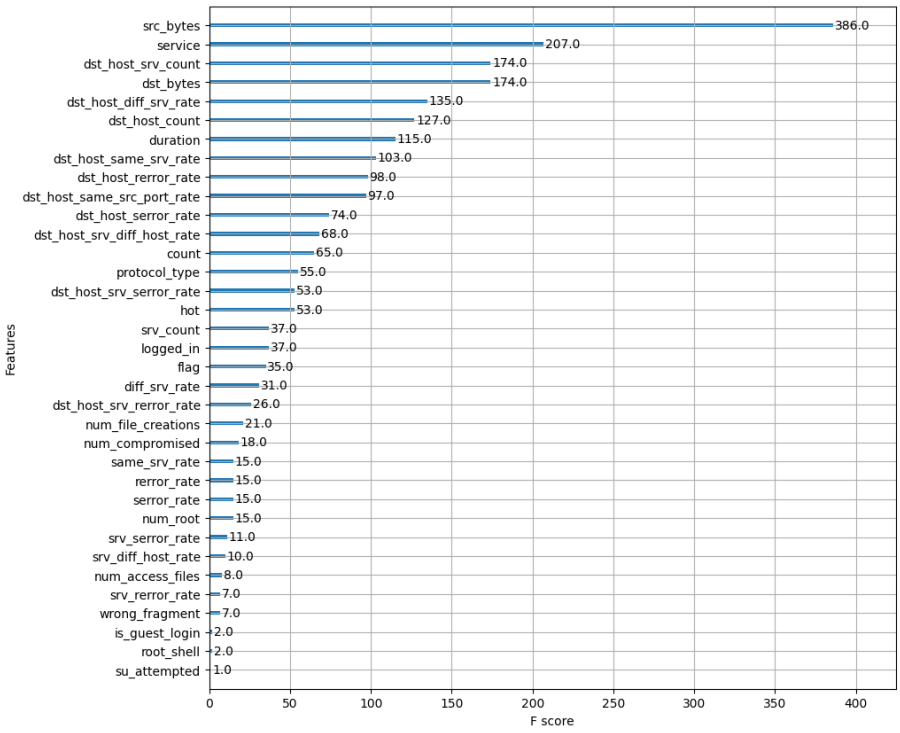


Figure 4.4: *XGBoost* feature ranking in NSL-KDD

4.1.3 Considerations

The two feature selection methods have yielded notably distinct results, which is not unexpected. It's plausible that certain features may have substantial importance when considered independently for prediction, yet when evaluated using Pearson correlation, they may receive lower rankings.

For example, *XGBoost* in the NSL-KDD indicates the src_bytes as the most relevant feature for the prediction of the target. At the same time, the Pearson correlation of src_bytes is not even listed in the plot because its value is below 0.15

By simulation with a MLP we have evaluated the top 10 features of both methods in order to choose which one is performing better. As expected, the *XGBoost* subset performed slightly better because some features are completely missed with the Pearson correlation.

The table below presents the smallest subset of features selected from the *XGBoost* ranking without compromising the model's performance. While *XGBoost* was used to select the features, there is one exception: the "flag" feature in the NSL-KDD dataset. This particular feature was assigned a lower ranking, potentially due to the fact that the algorithm was tested with the feature not encoded using the one-hot technique. Nevertheless, we deemed this feature to be of significant importance and retained it in the subset.

Index	Feature
f1	source_port
f2	init_win_bytes_forward
f3	destination_port
f4	flow_iat_min
f5	init_win_bytes_backward
f6	fwd_iat_min
f7	bwd_packets/s
f8	flow_duration
f9	flow_iat_mean
f10	average_packet_size
f11	flow_bytes/s
f12	total_length_of_bwd_packets
f13	fwd_packets/s
f14	bwd_iat_min
f15	flow_iat_std
f16	total_length_of_fwd_packets
f17	total_backward_packets

Table 4.1: Feature selected in CIC-IDS2017

Index	Feature
f1	src_bytes
f2	dst_bytes
f3	service
f4	flag
f5	duration
f6	count
f7	dst_host_srv_count
f8	dst_host_same_srv_rate
f9	dst_host_rerror_rate
f10	dst_host_same_src_port_rate
f11	dst_host_serror_rate
f12	dst_host_srv_diff_host_rate

Table 4.2: Feature selected in NSL-KDD

4.2 k-means clustering

In the process of implementing k-means clustering for anomaly detection, a crucial second step follows the creation of clusters. In this step, each cluster is assigned a label, designating it as either an "attack" or "normal" cluster. This labeling process relies on determining the majority class of samples within each cluster. Consequently, this procedure is not entirely unsupervised, as it employs training labels to categorize the clusters.

Once the clusters are properly created and identified, their performance is evaluated using a test set. Accuracy and FPR results are obtained by varying the number of clusters for both datasets, offering valuable insights into the effectiveness of the k-means clustering approach for anomaly detection. The number of clusters chosen is the one that achieved the best results.

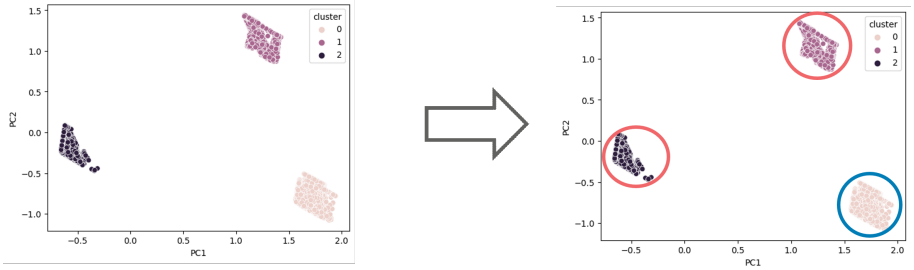


Figure 4.5: Anomaly detection with k-means scheme

Dataset	Accuracy	FPR
CIC-IDS2017	0.85	0.0067
NSL-KDD	0.78	0.02

Table 4.3: General metrics for k-means

4.2.1 CIC-IDS2017

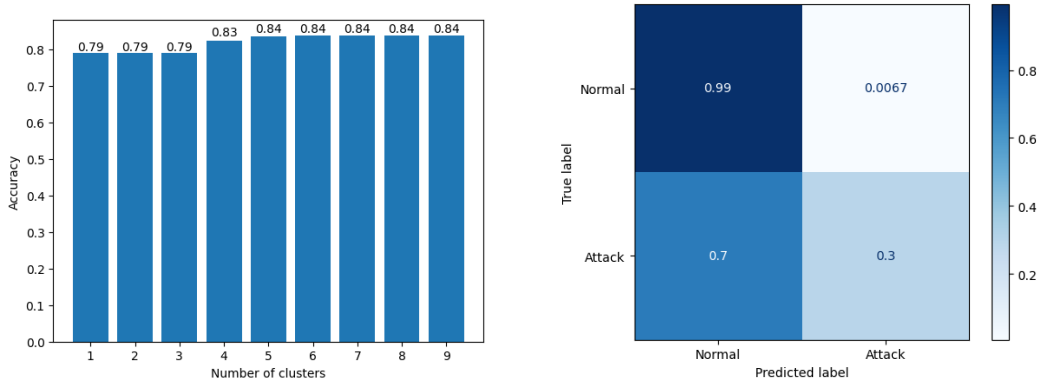
The optimal number of clusters for achieving the best results is determined to be 9. However, the obtained results fall short in comparison to other models tested using the same dataset. Specifically, the recall for the attack class is notably low in this scenario.

The low detection rate indicates that a significant portion of the attacks in this dataset is not effectively delineated within clusters; rather, they tend to be frequently mixed with normal-type clusters.

It's worth noting the stability of the cluster count concerning accuracy values. The accuracy achieved with 9 clusters closely resembles that with only one cluster. Improved results can be attained by addressing outliers in the dataset, allowing the algorithm to more effectively identify dataset clusters.

Class	Precision	Recall	F1	Support
Normal	0.84	0.99	0.91	630826
Attack	0.92	0.30	0.45	166468

Table 4.4: Metrics for k-means in CIC-IDS2017



(a) Cluster selection in CIC-IDS2017

(b) Confusion matrix for k-means in CIC-IDS2017

Figure 4.6: k-means clustering in CIC-IDS2017

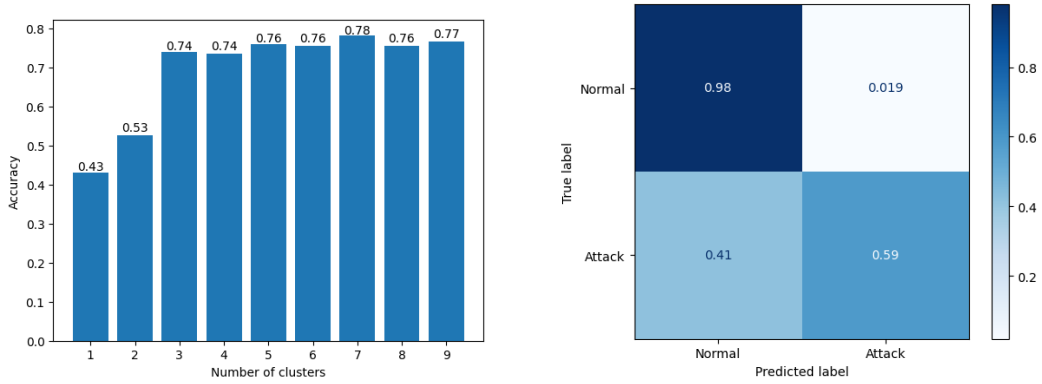
4.2.2 NSL-KDD

In the NSL-KDD dataset, the ideal cluster count chosen for optimal results is 7. Remarkably, the performance closely resembles that of other models tested on the NSL-KDD dataset. However, the detection rate is slightly lower compared to these other models. This is unsurprising, as solving the problem with an unsupervised approach is inherently more challenging.

In contrast, when compared to the CIC-IDS2017 dataset, the detection rate is notably improved, and accuracy exhibits greater variability relative to the clusters.

Class	Precision	Recall	F1	Support
Normal	0.64	0.98	0.78	9711
Attack	0.98	0.76	0.75	12814

Table 4.5: Metrics for k-means in NSL-KDD



(a) Cluster selection in NSL-KDD

(b) Confusion matrix for k-means in NSL-KDD

Figure 4.7: k-means clustering in NSL-KDD

4.3 Random forest

The RF, a classical ML algorithm frequently used in network anomaly detection, offers significant advantages. Due to its ensemble composition, it demonstrates robustness against overfitting and excels at managing imbalanced datasets. Moreover, it displays greater resilience to outliers and presents a comparatively straightforward implementation process in comparison to alternative models. Another interesting feature of this model is that it excels in data interpolation, which involves making predictions within the range of existing data points seen by the training.

A significant drawback of this model is its limited ability to extrapolate data, which translates to its challenges in identifying novel data that significantly deviates from the training dataset. This limitation is particularly critical, given that one of the primary objectives of an anomaly detector is to detect previously unknown or emerging attacks.

Dataset	**Accuracy**	**FPR**
CIC-IDS2017	1	0
NSL-KDD	0.78	0.02

Table 4.6: General metrics for *Random Forest* (RF)

4.3.1 CIC-IDS2017

The Random Forest algorithm produces the most favorable results in the context of the CIC-IDS2017 dataset. However, it becomes apparent that these outcomes are overly optimistic due to a test dataset that closely resembles the training set.

As mentioned in the preceding section, this model excels in interpolating data, effectively capturing patterns in data that closely resemble its training dataset. Consequently, when confronted with data that is not significantly dissimilar from the training set, it yields impressive results. This is notably evident in its strong performance on the CIC-IDS-2017 dataset, where the test set closely mirrors the training data.

Class	**Precision**	**Recall**	**F1**	**Support**
Normal	1	1	1	630826
Attack	1	1	1	166468

Table 4.7: Metrics for *Random Forest* (RF) in CIC-IDS2017

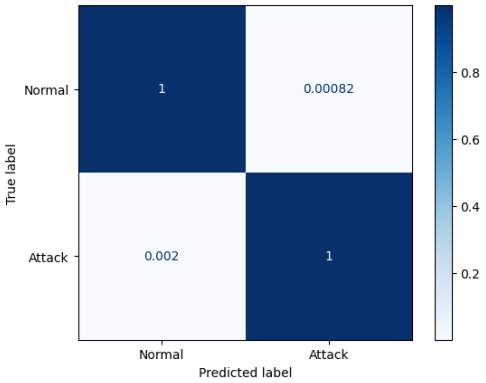


Figure 4.8: Confusion matrix for *Random Forest* (RF) in CIC-IDS2017

4.3.2 NSL-KDD

In the NSL-KDD dataset, we observe more realistic results when compared to those obtained from the CIC-IDS2017 dataset. It's worth noting that these results align with the average performance of other benchmarked models on this dataset. While the detection rate is slightly improved over k-means, it's important to highlight that approximately 37% of the attacks remain undetected.

Class	Precision	Recall	F1	Support
Normal	0.66	0.97	0.79	9711
Attack	0.97	0.63	0.76	12814

Table 4.8: Metrics for *Random Forest* (RF) in NSL-KDD

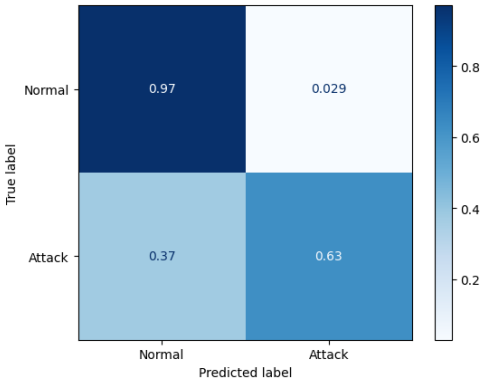


Figure 4.9: Confusion matrix for *Random Forest* (RF) in NSL-KDD

4.4 Multi layer perceptron

The MLP represents the first instance of using NN for network anomaly detection in our study. We conducted experiments on each of the two datasets, systematically varying the number of layers and neurons to identify the optimal combination.

A notable observation is that as we increase the model's complexity, its performance exhibits slight improvements. It is imperative to bear in mind that the number of parameters employed in this model significantly surpasses those in classical approaches such as the RF and k-means.

In both datasets, we employed the *Stochastic Gradient Descend* (SGD) or the adam optimizer along with binary cross-entropy loss. The optimal learning rate was determined through experimentation. We set the number of training epochs to 50 and implemented a *TensorFlow* callback that enables early stopping. This callback halts training if the loss stops to decrease over the last 5 epochs and restores the best weights. It's worth noting that this approach was consistently applied to all subsequent neural network-based models, as will be explained later.

Dataset	**Accuracy**	**FPR**
CIC-IDS2017	0.98	0.0068
NSL-KDD	0.82	0.03

Table 4.9: General metrics for *Multi Layer Perceptron* (MLP)

4.4.1 CIC-IDS2017

Similar to the other models, the outcomes obtained with the MLP on the CIC-IDS2017 dataset exhibit a level of over-optimism due to an excessively uncomplicated test set. The MLP architecture consists of three layers with 128, 64, and 32 neurons respectively, each utilizing the MLP activation function. The final layer comprises a single neuron with the sigmoid activation function. The optimizer that performed better, and so the one that has been chose is the adam.

Class	**Precision**	**Recall**	**F1**	**Support**
Normal	0.99	0.99	0.99	630826
Attack	0.97	0.98	0.98	166468

Table 4.10: Metrics for *Multi Layer Perceptron* (MLP) with CIC-IDS2017

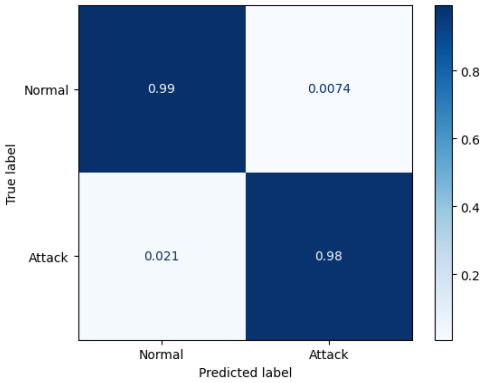


Figure 4.10: Confusion matrix for *Multi Layer Perceptron* (MLP) in CIC-IDS2017

4.4.2 NSL-KDD

The model evaluated on the NSL-KDD dataset demonstrates a more realistic and reliable result. This MLP architecture comprises two layers with 1024 and 32 neurons respectively. Following these layers is a dropout layer with a dropout rate of 0.01, strategically employed to mitigate overfitting. The final layer encompasses a single neuron utilizing the sigmoid activation function. Remarkably, this model shows exceptional performance, vaunting a recall rate of 0.7 and a mere 0.03 FPR.

Class	Precision	Recall	F1	Support
Normal	0.71	0.97	0.82	9711
Attack	0.97	0.70	0.81	12814

Table 4.11: Metrics for *Multi Layer Perceptron* (MLP) in NSL-KDD

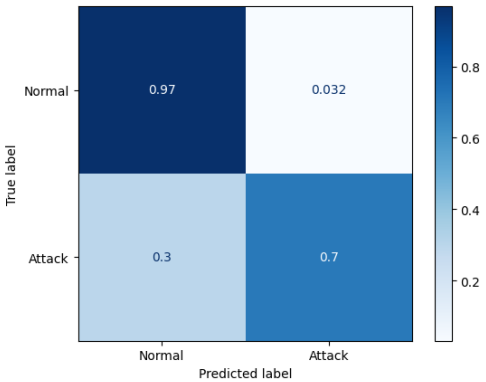


Figure 4.11: Confusion matrix for *Multi Layer Perceptron* (MLP) in NSL-KDD

4.5 Convolutional neural networks

The next model implemented is the CNN, which can be seen as a MLP with a convolutional layer before the dense one. It's crucial to emphasize that in this scenario, we're employing the one-dimensional version (1D) of convolutional filters and pooling layers. Given that our dataset comprises one-dimensional samples, the 1D counterparts are aptly utilized, following the same fundamental concept while considering a one-dimensional input.

In both datasets, the model architecture consists of a single CNN layer featuring 64 filters with a kernel size of 3x3 and employing the ReLu activation function. This is followed by a max pooling layer, a flatten layer, a dense layer housing 64 neurons, a dropout layer with a drop frequency of 0.5, and finally, the last layer with a single neuron utilizing the sigmoid activation function.

Dataset	Accuracy	FPR
CIC-IDS2017	1	0
NSL-KDD	0.8	0.02

Table 4.12: General metrics for *Convolutional Neural Network* (CNN)

4.5.1 CIC-IDS2017

In line with other models, the results obtained on the CIC-IDS2017 dataset seem overly optimistic due to a test set that lacks complexity. Furthermore, these outcomes closely align with the results produced by other models when applied to the same dataset.

Class	Precision	Recall	F1	Support
Normal	1	1	1	630826
Attack	0.99	0.99	0.99	166468

Table 4.13: Metrics for *Convolutional Neural Network* (CNN) in CIC-IDS2017

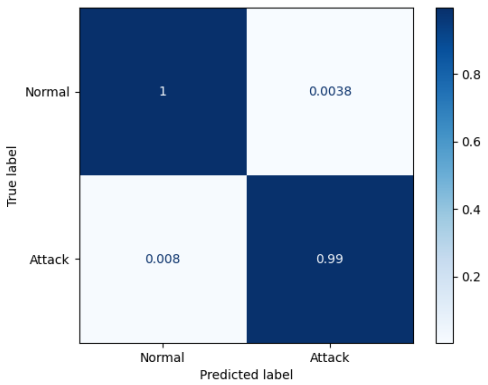


Figure 4.12: Confusion matrix for *Convolutional Neural Network* (CNN) in CIC-IDS2017

4.5.2 NSL-KDD

While the model displayed promising results, they fell short of the MLP performance. This disparity can be attributed to CNN susceptibility to overfitting during training. Determining the ideal number of filters and kernel size to achieve optimal results is a nontrivial task. However, with the current configuration, we managed to obtain the best achievable outcomes. Nonetheless, we believe there's room for enhancement by incorporating additional convolutional filters and pooling layers.

Class	Precision	Recall	F1	Support
Normal	0.69	0.97	0.81	9711
Attack	0.97	0.67	0.79	12814

Table 4.14: Metrics for *Convolutional Neural Network* (CNN) in NSL-KDD

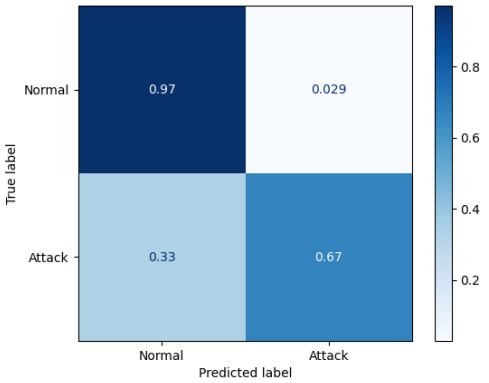


Figure 4.13: Confusion matrix for *Convolutional Neural Network* (CNN) in NSL-KDD

4.6 Autoencoder

The AE serves as a highly effective unsupervised tool for anomaly detection. However, comprehending its mechanism for detection isn't straightforward, necessitating a thorough explanation to grasp how it identifies attacks.

To begin, the AE undergoes training exclusively with normal traffic data from the training set. This allows it to acquire the ability of encoding these legitimate samples into a lower-dimensional representation and reconstructing them with minimal reconstruction error.

The loss function employed in this model is the MAE, as it yielded superior performance. We also conducted tests using the MSE and RMSE, but we achieved the best performances with the MAE.

The subsequent step involves calculating reconstruction errors for all training set samples and determining the optimal threshold value for distinguishing normal from attack instances. This process, known as threshold estimation, is a crucial aspect of the model. There are various methods to accomplish this task. One common approach involves averaging the reconstruction errors of the normal samples during training, or alternatively, using percentiles as a measure.

The image below clearly illustrates the composition of the anomaly detection process using AEs.

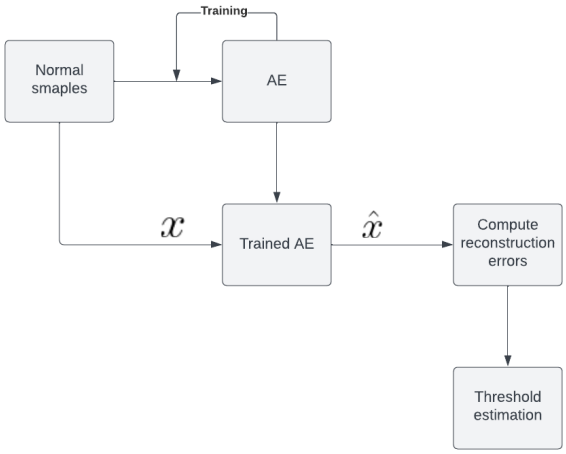


Figure 4.14: Anomaly detection with *Autoencoder* (AE) scheme

Dataset	Accuracy	FPR
CIC-IDS2017	0.84	0.01
NSL-KDD	0.77	0.02

Table 4.15: General metrics for *Autoencoder* (AE)

4.6.1 CIC-IDS2017

The performance of the AE model on the CIC-IDS2017 dataset was notably inferior compared to other models. Despite the dataset's pronounced redundancy, relying on a single AE to encapsulate normal behavior proved insufficient in effectively detecting a substantial portion of the attacks. The notable prevalence of false negatives indicates that the AE deciphers attacks with an unexpectedly low reconstruction error.

This implies that a majority of the attacks can be reconstructed effectively by an AE trained solely on normal samples. It's crucial to keep in mind that the optimal threshold was set based on the best results achieved.

Class	Precision	Recall	F1	Support
Normal	0.84	0.99	0.91	630826
Attack	0.88	0.28	0.43	166468

Table 4.16: Metrics for *Autoencoder* (AE) in CIC-IDS2017

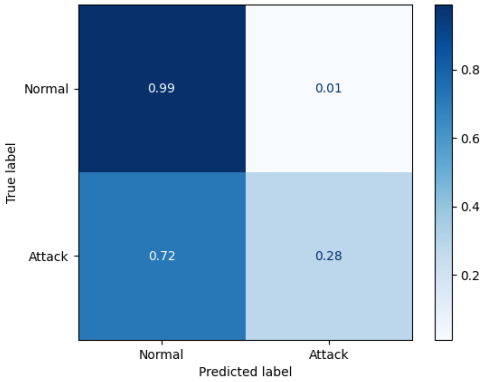


Figure 4.15: Confusion matrix for *Autoencoder* (AE) in CIC-IDS2017

4.6.2 NSL-KDD

The AE is composed by three layers, the input and the output which have the usual input space, and the latent space which has a smaller dimension and then another dense layer of the same dimension of the input. We have chosen the threshold that separate better the two classes among the training samples which is computed as the 0.97 percentile of the normal training samples reconstruction errors.

Class	Precision	Recall	F1	Support
Normal	0.66	0.97	0.79	9711
Attack	0.97	0.62	0.76	12814

Table 4.17: Metrics for *Autoencoder* (AE) in NSL-KDD

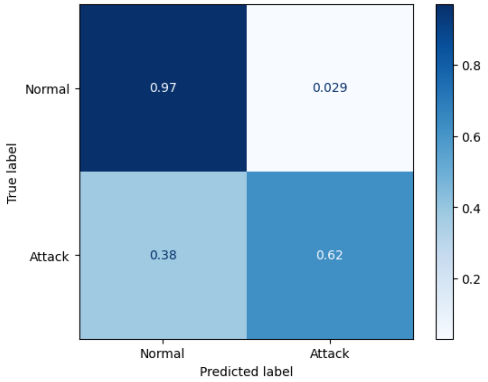


Figure 4.16: Confusion matrix for *Autoencoder* (AE) in NSL-KDD

The dimension of the latent space significantly impacts the model's performance, as it directly influences the threshold value. To determine the optimal latent space size, we conducted various simulations, comparing the resulting accuracies. Surprisingly, we observed that altering the latent space dimension doesn't lead to substantial changes in accuracy.

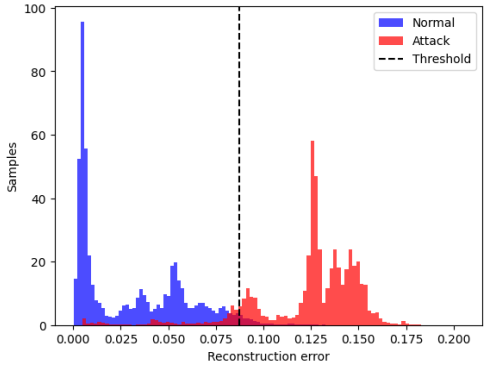


Figure 4.17: Threshold visualization for the *Autoencoder* (AE) in NSL-KDD

4.7 Long short time memory

The LSTM is the final model we will evaluate. As discussed in the background chapter, it belongs to the category of RNN, enabling it to capture temporal correlations within the data. In the context of network anomaly detection, this capability is crucial for capturing the temporal relationships among network flows.

From a logical perspective, leveraging this type of correlation can significantly enhance detection, as it allows for improved identification of many attacks by considering the time-based interactions of the flows.

Notably, this model stands out due to its exceptional performance. In both datasets, we utilized an LSTM with 32 units, consistently yielding optimal results.

Dataset	Accuracy	FPR
CIC-IDS2017	0.99	0
NSL-KDD	0.83	0.039

Table 4.18: General metrics for *Long Short Time Memory* (LSTM)

4.7.1 CIC-IDS2017

In line with other models, the outcomes on the CIC-IDS2017 dataset appear overly optimistic due to the simplified nature of the test set. These results closely resemble those achieved by other models applied to the same dataset.

Class	Precision	Recall	F1	Support
Normal	1	0.99	1	630826
Attack	0.97	0.99	0.98	166468

Table 4.19: Metrics for *Long Short Time Memory* (LSTM) in CIC-IDS2017

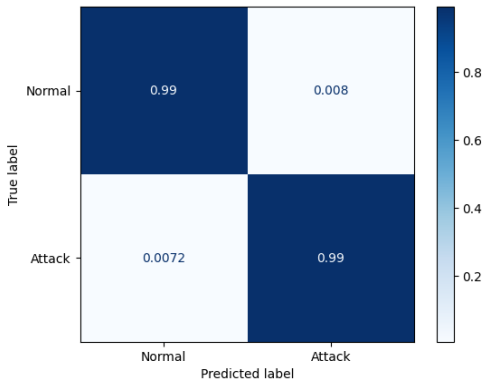


Figure 4.18: Confusion matrix for *Long Short Time Memory* (LSTM) in CIC-IDS2017

4.7.2 NSL-KDD

The LSTM model, when applied to the NSL-KDD dataset, exhibits remarkable performance, especially in achieving a recall rate of 0.74 for detecting attacks. This recall rate surpasses that of all the other models tested on this dataset, underscoring the significance of incorporating the temporal relationships among network flows to enhance attack detection.

Conversely, it's worth noting that the FPR is relatively high at 0.039, marking the highest value observed thus far.

Class	Precision	Recall	F1	Support
Normal	0.73	0.96	0.83	9711
Attack	0.96	0.74	0.83	12814

Table 4.20: Metrics for *Long Short Time Memory* (LSTM) in NSL-KDD

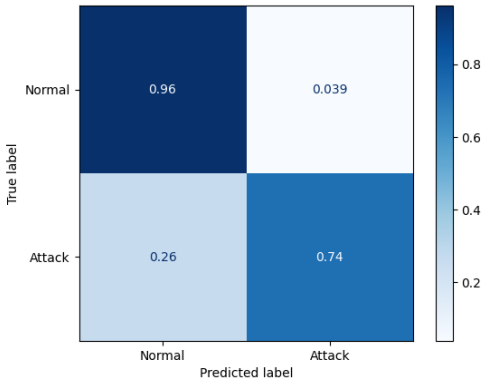


Figure 4.19: Confusion matrix for *Long Short Time Memory* (LSTM) in NSL-KDD

4.8 Comparison

Following a comprehensive assessment of all the models considered for both datasets, a number of noteworthy insights emerge. These observations hold particular significance as they underpin the author's subsequent research direction, culminating in the development of a framework to be expounded upon in the following section.

Firstly, the CIC-IDS2017 dataset proves to be an unreliable benchmark due to its inherent redundancy, rendering the test set substantially easier than the training set. Conversely, the NSL-KDD dataset, albeit smaller in size, presents a more challenging test set, making it better suited for evaluating models. Because of that, the research will continue without considering anymore this dataset as a benchmark. Indeed, in the next section the proposed framework will be tested only on the NSL-KDD dataset.

A noteworthy observation is the superior performance of deep learning models, specifically neural networks, in comparison to traditional ML models such as RF and k-means. The LSTM model exhibits the highest accuracy, reaching 0.83. However, it also demonstrates the highest FPR at 0.039. We can confirm that the trend of creating deep neural networks can effectively improve the results of the classification as recent literature surveys have anticipated [4].

Another important observation is that the unsupervised models have been the one that performed slightly worse. This is not surprisingly since they try to solve a more challenging problem. In particular, the AE model obtained the worse result in accuracy with 0.77.

In a broader context, the differences in performance among the models are relatively marginal. The LSTM model emerges as the most effective, while the AE model performs the least optimally. The discrepancy in accuracy between these models is merely 0.06. In the following table there is a list of all the model implemented with the correspondent accuracy and FPR.

Model	**Accuracy**	**FPR**
k-means	0.78	0.02
RF	0.78	0.02
MLP	0.82	0.03
CNN	0.8	0.02
AE	0.77	0.02
LSTM	0.83	0.039

Table 4.21: Models comparison in NSL-KDD

Considering these results, we have chosen to conduct a more in-depth investigation of the model that performed the least effectively: the AE. This decision stems from the potential for improvement in this model. Moreover, the unsupervised approach holds promise for efficient anomaly identification, as it focuses solely on normal behavior to identify attacks. Despite the challenges associated with this approach, we believe it offers a comprehensive solution for anomaly detection.

4.9 Proposed framework

In this section, we elucidate the architectural design of our framework, an amalgamation of techniques thoroughly seen previously. Our objective was to craft a model that optimally leverages the potential of ML within NIDS. Specifically, we aspired to create a model proficient in efficiently identifying previously unknown attacks and categorizing various permutations of learned attack patterns.

Many of the articles reviewed in the literature propose highly intricate models with a multitude of parameters, which are subsequently evaluated on widely-used datasets, such as the one examined in this research. We believed that, given the dataset's questionable reliability, constructing a deep neural network with numerous parameters might not ensure a genuine improvement, even if it exhibits strong performance in one of these benchmarks. As a result, we present a framework that serves as an explanatory guide to an architectural approach for effectively addressing the challenge of network anomaly detection using AEs.

In this section, we will enumerate all the enhancements made to the AE model for anomaly detection. In total, we have implemented three major improvements to the model, which will be detailed in the following subsections: outlier removal, AE specialization, and threshold estimation using a MLP. Ultimately, a benchmark on the NSL-KDD dataset will be conducted to assess performance quantitatively.

4.9.1 Outlier removing

As shown in this paper [51], the removal of outliers can enhance the performance of AEs. We employ a method based on the 0.95 percentile for each feature for the dataset. By removing data points beyond this threshold, we create a cleaner and more representative dataset. This process effectively reduces the impact of noisy or irrelevant data, enabling the AE to focus on learning and capturing meaningful patterns and features within the input data. Consequently, the AE's reconstruction accuracy and overall performance are expected to be significantly improved.

This marked the initial enhancement for the AEs, a model known for its susceptibility to outliers. We consider this step to be imperative.

4.9.2 Autoencoders specialization

A key constraint of the AE model lies in its mechanism of anomaly detection, predicated solely on scrutinizing the benign network behavior. However, normal behavior can vary significantly based on the specific network context. Consequently, the efficacy of this model is contingent upon how adeptly we tailor it to the contextual circumstances under test.

We have discerned two viable approaches for such tailoring: model retraining using contextual data and AE specialization. The former, model retraining, presents the most intuitive and commonly adopted option. Nonetheless, it harbors a significant drawback: susceptibility to data poisoning during training by a malevolent actor, posing a potential risk to the system's integrity.

The latter approach, AE specialization, involves the creation of distinct AEs, each tailored for a unique segment of the network. We have identified the following approaches:

- Subnet specialization.
- Device specialization.
- Protocol specialization.
- Service specialization.

We consider each of these approaches to be suitable for our framework, with the choice primarily contingent on the dataset's accessibility and structure used for the training set. In general, it is worth noting that device specialization has been widely adopted and has proven to be highly effective, as highlighted in prior work (e.g., [25]).

As illustrated in the preceding sections, the conventional approach to employing AEs for anomaly detection involves training them exclusively on normal samples. However, recent research [25] has suggested that incorporating attack samples in the training process can enhance the solution. Notably, prior studies have not ventured into the domain of AEs specialization specifically tailored for attack samples.

In the proposed framework, an AE specialization for attack samples is implemented, built on the assumption that samples from the same attack class exhibit shared similarities that can be efficiently encoded using an AE.

4.9.3 Threshold estimation with MLP

A crucial aspect of the anomaly detection process utilizing AEs is threshold estimation. Typically, this involves simulation on the training set to determine the optimal value. This step is non-trivial and often a primary contributor to lower performance.

In this framework, our approach involves estimating the threshold for each specialized AE using an MLP in a ML fashion. The MLP takes as input the reconstruction errors generated by all the AEs, and based on this information, classifies the network flow as normal or indicative of an attack.

The complete training of the model is conducted in two distinct steps: first, the training of each AE on its specific network segment, followed by the training of the MLP.

The MLP, trained with the reconstruction errors produced by the AEs, can provide more accurate predictions for the final classification. This is achieved by comparing the reconstruction errors from various AEs using the NN to determine the source of the network flow.

Moreover, the task of identifying the threshold for each AE in the network is a challenging process. However, this approach helps facilitate and improve the accuracy of this threshold determination.

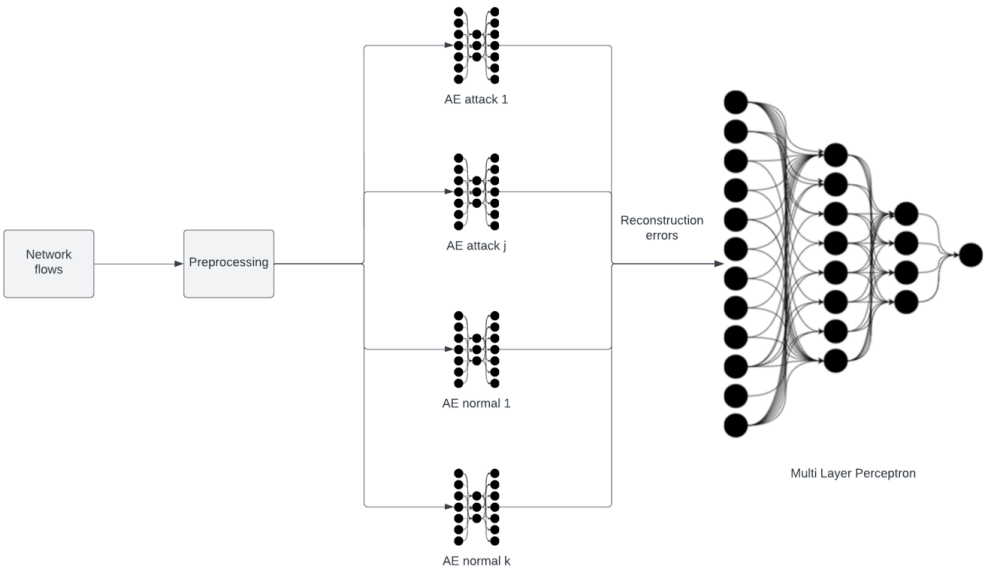


Figure 4.20: Proposed framework scheme

4.9.4 NSL-KDD

As delineated in the preceding section's summary, the selection of the NSL-KDD dataset as a benchmark dataset was motivated by the formidable challenge posed by its test set. To thoroughly evaluate the framework's performance, we deliberately chose this dataset. However, it is crucial to note that complete specialization of the AEs on this dataset was not feasible, primarily due to the insufficient normal samples available for segmentation based on device or protocol. Nevertheless, a targeted specialization of the AEs for various attack classes was successfully implemented.

Due to limited sample availability, we merged two classes: "U2R" and "R2L". Unfortunately, this amalgamation significantly impacted the performance of the AE due to the dissimilarity between the samples in these two classes. We made this choice out of necessity to test the model on this dataset. In a real-world scenario, it is advisable to train multiple AEs for specific attack classes. Additionally, as mentioned earlier, there should be the AE specialization also for the normal samples. In the following image there is the final representation of the dataset partitions used for train the model.

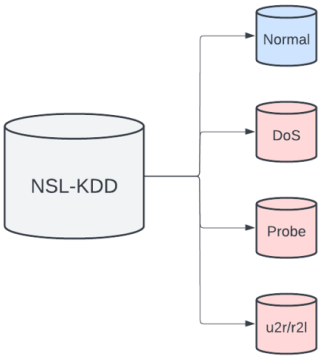


Figure 4.21: NSL-KDD partitions

Below are the results obtained from benchmarking the proposed framework on the NSL-KDD dataset. We accomplished an impressive overall accuracy of 0.91 with a relatively low AE of 0.13. Notably, this accuracy surpasses that of all other models previously implemented. It's relevant mentioning that the higher AE may be attributed to the absence of AE specialization for normal traffic.

Class	**Precision**	**Recall**	**F1**	**Support**
Normal	0.92	0.87	0.89	9711
Attack	0.91	0.94	0.92	12814

Table 4.22: Metrics for the proposed framework in NSL-KDD

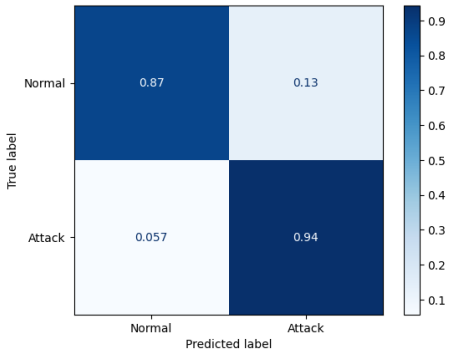


Figure 4.22: Confusion matrix for the proposed framework in NSL-KDD

In the following table there is a comparison of the proposed framework and the models previously implemented. A significant enhancement is observed compared to using a single AE for anomaly detection, which yielded an accuracy of 0.77. The proposed framework consistently demonstrated superior performance in comparison to all the other implemented models. As previously emphasized, the most significant drawback of this implementation remains the considerably higher FPR in comparison to the other models.

Model	**Accuracy**	**FPR**
k-means	0.78	0.02
RF	0.78	0.02
MLP	0.82	0.03
CNN	0.8	0.02
AE	0.77	0.02
LSTM	0.83	0.039
Proposed framwework	**0.91**	0.13

Table 4.23: Proposed framework and the implemented models comparison in NSL-KDD

4.10 Summary

This chapter has presented all the quantitative results derived from this study, with a brief commentary provided for each. Additionally, it includes a comprehensive comparison of all the models, including the proposed framework.

In the concluding section of this chapter, our approach extends beyond presenting solely quantitative results. We delve into an explanation of the design choices underpinning the proposed framework, as these decisions have played a pivotal role in driving the observed improvements.

At this stage, the reader is well-informed about the outcomes achieved in this study. In the subsequent chapter, titled "Conclusions", a detailed exploration of these achievements will be provided from the author's perspective.

Chapter 5

Conclusions

In this chapter, we will present the study's conclusions, summarizing all the results obtained through the methodologies described earlier.

We will outline the key findings from our research, encompassing the comparison of various models and the innovations introduced by our proposed framework. Notably, we will highlight the significant enhancements achieved through our framework.

Additionally, we will thoroughly examine the limitations that have had a profound impact on this project. This emphasis is critical for future researchers in this field, as they must take these limitations into account.

Then, we will provide a section on future work, where we will discuss unexplored ideas that should be considered for future research endeavors.

In closing this chapter, the author of this thesis will provide a concluding statement, summarizing the principal achievements, and offering personal insights and ideas concerning future research directions within this field.

5.1 Key findings

Our exploration of network anomaly detection through the lens of ML has provided us with a comprehensive understanding of this multifaceted problem. This research has yielded several pivotal discoveries, which we categorize into two distinct domains. The first block encompasses findings related to preprocessing and model implementation, while the second block delves into insights concerning the proposed framework.

The primary focus of the next section, which revolves around limitations, concerns the dataset. Throughout our research, we conducted an in-depth analysis of the available dataset to address the network anomaly detection problem. One notable discovery was that the widely used CIC-IDS2017 dataset, often considered a benchmark, proved to be an unreliable choice for this particular objective. On the other hand, the NSL-KDD dataset presented greater challenges, making it more suitable for training a ML model.

The preprocessing stage emerged as a crucial aspect of our work. The numericalization and normalization of the dataset were found to be pivotal. However, the feature selection component did not yield the anticipated improvements. In both datasets, it became evident that a significant portion of the features was unnecessary for detecting attacks, often resulting in a slowdown during training and testing. In our case, feature selection contributed mainly to speeding up the model without demonstrably enhancing its performance. Importantly, it's worth noting that the performance did not deteriorate as a result of these efforts.

Another intriguing observation was that the complexity of the ML models had a relatively minor impact on overall model performance. While we acknowledge that there is still room for improvement in this research domain, it's essential to note that the performance gains achieved by increasing the number of model parameters were not substantial for each model tested. Additionally, we found that the choice of different ML models had only a marginal impact on performance. The most significant performance enhancement was realized through the implementation of our proposed framework, where we not only tinkered with model parameters but also improved the system's architecture and conducted a comprehensive outlier cleaning of the dataset.

With the formulation of the proposed framework, we have demonstrated that an unsupervised approach like the AE can be significantly enhanced in various ways, leading to excellent quantitative results. Notably, we have shown that outlier cleaning for AE plays a pivotal role in ensuring improved learning and subsequently enhances attack detection. It's worth emphasizing that the majority of performance enhancements in our framework are attributed to this aspect.

Furthermore, we have discovered that specializing AEs is a valuable strategy for adapting the model to the specific context, which is crucial for unsupervised learning models. Our findings highlight that even attack samples can be leveraged by AEs to enhance network monitoring and improve attack detection.

Moreover, we have identified that the efficient estimation of thresholds for a group of AEs can be achieved using a MLP. This streamlined approach facilitates the process while ensuring the identification of optimal thresholds for the training set.

The proposed framework has yielded a remarkable improvement in performance on the NSL-KDD dataset, surpassing all previously implemented models. Furthermore, the results obtained are comparable with the models discussed in the literature review.

5.2 Limitations

One prominent limitation that significantly impacted this study revolved around the challenge of obtaining a reliable dataset for ML models. We encountered numerous obstacles in our quest to identify a dataset that simultaneously served as a dependable benchmark, contained a challenging test set, and included numerous up-to-date samples. A clear illustration of this limitation can be found in the CIC-IDS2017 dataset. Regrettably, the benchmarks conducted on models within this dataset proved to be entirely unrealistic, thereby offering a distorted perspective of the actual model performances.

Concurrently, even a dataset as valuable as NSL-KDD, which proved beneficial for benchmarking models in the initial phase of the study, ultimately revealed its unsuitability for the proposed framework. In this context, it was impossible to unlock the framework's full potential due to the lack of sample diversity, preventing the complete specialization of the AEs.

An additional noteworthy limitation that we encountered while working on this topic pertained to the lack of uniformity among the models benchmarked in the existing literature. The vast diversity in datasets used by various researchers posed a significant challenge when attempting to compare our results with theirs. It is evident that comparing identical models tested on different datasets is a futile exercise. Our case studies involving the CIC-IDS2017 and the NSL-KDD datasets serve as a prime example of the substantial variability in results. Furthermore, we observed that many studies in this field provided incomplete information regarding dataset performance, often omitting essential metrics such as FPR and exclusively reporting accuracy.

Additionally, several studies utilizing the NSL-KDD dataset failed to provide clear distinctions regarding whether the test set was integrated into the training set or if it was obtained from the dataset creators as a separate file. This particularity can significantly impact the results, as the test set furnished by the dataset creators poses a notably more challenging evaluation scenario compared to simply extracting a portion of the training set.

Finally, another limitation we encountered was the inability to fully reproduce the results presented in certain studies. Even when employing the exact parameters and architectures on the same dataset, we were unable to achieve identical outcomes. It would greatly facilitate the replication of results if these studies were accompanied by a code repository to ensure full replicability.

5.3 Future works

In this study, we had the opportunity to identify the primary areas upon which future researchers should concentrate their efforts. We propose that the following ideas should be promptly incorporated into research in this field.

The outcomes of the proposed framework on the NSL-KDD dataset display promise. However, it's important to note that while we have achieved notable performance with this dataset, it may not represent the optimal potential of our framework. We advocate for future research to prioritize the creation or discovery of a more suitable dataset for evaluation. Finding a dataset that allows for a complete specialization of AEs, especially with a challenging test set, poses a significant challenge. In forthcoming studies, efforts should be directed towards generating data and collecting a large amount of samples to facilitate AEs specialization for each device.

Another dataset-related challenge pertains to the development of a dependable dataset that accurately represents real attack classes, thereby enhancing the specialization of AEs for these. Our belief is that even within seemingly heterogeneous attack categories, there may exist underlying similarities that can be effectively identified and encoded using AEs.

We also recommend that future research endeavors incorporate techniques to safeguard models against adversarial ML attacks. This is an area that has only recently garnered attention in studies [52]. We firmly believe that all research in this domain should prioritize protection against such threats, especially given the increasing prevalence of deep and intricate models characterized by their black-box nature.

5.4 Final comment

In this study, we conducted a comprehensive analysis of network anomaly detection using ML. While we are satisfied with the results we've achieved, we acknowledge that there is room for improvement, particularly concerning the proposed framework.

We take satisfaction in having comprehensively examined the topic with a rigorous methodology, providing readers with the necessary background knowledge to fully grasp the research. Our hope is that this study proves beneficial to those delving into this subject, and we encourage future researchers to consider the ways for further exploration that we have outlined.

The convergence of human identity and digital twin is an increasingly relevant phenomenon. Therefore, research in areas like cyber attacks detection is of paramount importance, as it addresses the critical task of preventing cyber threats from succeeding. We emphasize the significance of continued studies in this domain, as they are essential for safeguarding our future.

Acknowledgments

Ho scelto di scrivere questa sezione in italiano con l'intenzione di esprimermi nel miglior modo possibile nei ringraziamenti alle persone che mi hanno accompagnato in questo lungo percorso. Nella mia tesi triennale, non ho incluso i ringraziamenti, forse perché ero scoraggiato dalla mancanza di ufficialità dovuta alla proclamazione online. Con questa parte del testo, desidero rimediare a questa assenza, poiché sono fermamente convinto che il mio successo è indissolubilmente legato alle persone che mi hanno circondato durante questo percorso.

Il primo ringraziamento va innanzitutto alla mia famiglia, che mi ha donato una serenità inestimabile lungo tutto il percorso, permettendomi di concentrarmi completamente sugli studi. Sono molto grato per il loro costante supporto e per avermi sempre saputo indirizzare anche nei momenti di più incertezza.

Un pensiero speciale è dedicato a mia sorella Chicchi, che, nonostante i numerosi episodi di schiavismo nei miei confronti, è stata una figura fondamentale nel mio percorso di crescita.

Grazie ad Angela, per avermi cresciuto ed essere stata presente come una madre fin dai primi ricordi della mia vita.

Ringrazio con affetto tutti i parenti e amici di famiglia in Sicilia. Nonostante la distanza fisica, ho sempre avvertito la vostra vicinanza nel profondo del cuore.

Desidero esprimere un ringraziamento speciale alle mie due nonne, che sono sempre state presenti e pronte a condividere la mia gioia per i successi ottenuti.

Vorrei dedicare un ringraziamento di cuore ai miei cari amici storici che ormai da diversi anni mi sopportano quotidianamente. Vorrei ringraziare Alberto, Carlo, Giacomo, Giulio, Marco e Sofia per essere sempre stati presenti in questi anni di cambiamenti. Un particolare pensiero va a un luogo magico che negli ultimi tempi ha contraddistinto dei momenti speciali: le "Chiuse di Voltabarozzo" e il "Telaio Bike café".

Nei primi tre anni di questo percorso, amaramente interrotti dalla pandemia, ho avuto l'opportunità di incontrare delle persone che si sono rivelate fondamentali. Grazie a loro ho mosso i miei primi passi dentro il mondo dell'università con meno paura. Ci tengo a ringraziare Alberto, Davide, Giacomo, Luca, Matteo, Paolo e Stefano per gli immensi aiuti e le risate condivise durante la triennale.

Nel primo semestre dell'ultimo anno trascorso a Barcellona, desidero ringraziare di cuore tutte le persone del gruppo "Barça". In particolare, vorrei menzionare Alessandro, Antonio, Clara, Diana, Generoso, Gabriel, Guillherme, Matteo, Peppe e Teresa, poiché hanno contribuito a rendere quel periodo uno dei più importanti della mia vita.

Vorrei conferire una menzione d'onore al ristorante "Garçon", di ispirazione Franco-Cinese, che ha rappresentato un eccezionale luogo di aggregazione e un'esperienza culinaria di alto livello.

Nel secondo semestre dell'ultimo anno trascorso in Olanda, desidero esprimere un sincero ringraziamento al mio supervisore, Antionios Atlasis, per avermi offerto questa straordinaria opportunità, e a tutto il team di TEC-ESS. Tra di loro, voglio riconoscere il contributo illuminante di Antonin, Gabriele, Loïc, Nicolas.

Vorrei esprimere la mia profonda gratitudine al meraviglioso gruppo di interns, dottorandi e dello "Shed" con cui ho condiviso l'intero semestre. Tra di loro, desidero ricordare tutti i membri del gruppo "A Lot of Italians" e "The Campus (Pep)peroni", con una menzione speciale per Alessandra, Andreas, Anna, Aurelio, Esther, Lucia, Michele, Nicolas, Samuele, Thomas e Viktor. Grazie a "Park de Put", sede di spensierati pomeriggi Olandesi.

Desidero concludere la sezione dei ringraziamenti con un pensiero speciale rivolto a tutti i professori che ho incontrato e che hanno contribuito in modo significativo alla mia formazione, sia a livello personale che professionale. In particolare, desidero esprimere la mia gratitudine al mio relatore, Nicola Laurenti, per la sua disponibilità e per essere stato un punto di riferimento fondamentale durante questi ultimi anni.

Bibliography

[1] Hervé Debar, Marc Dacier, and Andreas Wespi. "Towards a taxonomy of intrusion-detection systems". In: *Computer Networks* 31.8 (1999), pp. 805–822. ISSN: 1389-1286. DOI: https://doi.org/10.1016/S1389-1286(98)00017-6. URL: https://www.sciencedirect.com/science/article/pii/S1389128698000176.

[2] Eric Conrad, Seth Misenar, and Joshua Feldman. "Chapter 7 - Domain 7: Security operations". In: *Eleventh Hour CISSP® (Third Edition)*. Ed. by Eric Conrad, Seth Misenar, and Joshua Feldman. Third Edition. Syngress, 2017, pp. 145–183. ISBN: 978-0-12-811248-9. DOI: https://doi.org/10.1016/B978-0-12-811248-9.00007-3. URL: https://www.sciencedirect.com/science/article/pii/B9780128112489000073.

[3] Ammar Aldallal and Faisal Alisa. "Effective Intrusion Detection System to Secure Data in Cloud Using Machine Learning". In: *Symmetry* 13 (Dec. 2021), pp. 1–26. DOI: 10.3390/sym13122306.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41.3 (July 2009). ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: https://doi.org/10.1145/1541880.1541882.

[5] Sherenaz W Al-Haj Baddar, Alessio Merlo, and Mauro Migliardi. "Anomaly detection in computer networks: A state-of-the-art review." In: *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.* 5.4 (2014), pp. 29–64.

[6] Shai Shalev-Shwartz, Shai Ben-David, and Shai Shalev-Shwartz. *Understanding machine learning: from theory to algorithms / Shai Shalev-Shwartz, Shai Ben-David*. eng. Cambridge, 2014.

[7] *Test, training and validation sets*. URL: https://www.brainstobytes.com/test-training-and-validation-sets/.

[8] Raheel Muzzammel and Ali Raza. "A Support Vector Machine Learning-Based Protection Technique for MT-HVDC Systems". In: *Energies* 13 (Dec. 2020). DOI: 10.3390/en13246668.

[9] *Machine learning k-means algorithm*. URL: https://www.geeksforgeeks.org/ml-k-means-algorithm/.

[10] *Random forest*. URL: https://www.ibm.com/in-en/topics/random-forest#:~:text=Random%5C%20forest%5C%20is%5C%20a%5C%20commonly,both%5C%20classification%5C%20and%5C%20regression%5C%20problems..

[11] *What is a random forest*. URL: https://www.spotfire.com/glossary/what-is-a-random-forest.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. eng. Adaptive computation and machine learning. Cambridge, Massachusetts ; The MIT Press, 2016 - 2016. ISBN: 9780262337373.

[13] İrem Sahmutoglu, Talya Temizçeri, and Emine Bozkuş. "Evaluation of occupational accidents with artificial neural networks in occupational health and safety management systems". In: (Dec. 2021).

[14] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[15] *Recurrent neural network unfold*. URL: https://commons.wikimedia.org/wiki/User:Ixnay#/media/File:Recurrent_neural_network_unfold.svg.

[16] *LSTM cell illustration*. URL: https://en.wikipedia.org/wiki/File:LSTM_cell.svg.

[17] Bingdong Li et al. "A survey of network flow applications". In: *Journal of Network and Computer Applications* 36.2 (2013), pp. 567–581. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2012.12.020. URL: https://www.sciencedirect.com/science/article/pii/S1084804512002676.

[18] *Cisco NetFlow*. URL: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[19] *NSL-KDD Dataset*. URL: https://www.unb.ca/cic/datasets/nsl.html.

[20] *Netflow architecture illustration*. URL: https://en.wikipedia.org/wiki/NetFlow#/media/File:NetFlow_Architecture_2012.png.

[21] Ritesh K. Malaiya et al. "An Empirical Evaluation of Deep Learning for Network Anomaly Detection". In: (2018), pp. 893–898. DOI: 10.1109/ICCNC.2018.8390278.

[22] Hongyu Liu and Bo Lang. "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey". In: *Applied Sciences* 9.20 (2019). ISSN: 2076-3417. DOI: 10.3390/app9204396. URL: https://www.mdpi.com/2076-3417/9/20/4396.

[23] Chia-Ming Hsu et al. "Robust Network Intrusion Detection Scheme Using Long-Short Term Memory Based Convolutional Neural Networks". In: *Mobile Networks and Applications* 26 (June 2021). DOI: 10.1007/s11036-020-01623-2.

[24] Youngrok Song, Sangwon Hyun, and Yun-Gyung Cheong. "Analysis of Autoencoders for Network Intrusion Detection". In: *Sensors* 21.13 (2021). ISSN: 1424-8220. DOI: 10.3390/s21134294. URL: https://www.mdpi.com/1424-8220/21/13/4294.

[25] Mustafizur R. Shahid et al. "Anomalous Communications Detection in IoT Networks Using Sparse Autoencoders". In: (2019), pp. 1–5. DOI: 10.1109/NCA.2019.8935007.

[26] Giuseppina Andresini, Annalisa Appice, and Donato Malerba. "Autoencoder-based deep metric learning for network intrusion detection". In: *Information Sciences* 569 (2021), pp. 706–727. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2021.05.016. URL: https://www.sciencedirect.com/science/article/pii/S002002552100462X.

[27] Bo Zong et al. "Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection". In: *International Conference on Learning Representations*. 2018.

[28] Tianyue Zhang et al. “An intrusion detection method based on stacked sparse autoencoder and improved gaussian mixture model”. In: *Computers & Security* 128 (2023), p. 103144. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2023.103144. URL: https://www.sciencedirect.com/science/article/pii/S0167404823000548.

[29] *UNSW-NB15 dataset*. URL: https://research.unsw.edu.au/projects/unsw-nb15-dataset.

[30] Yisroel Mirsky et al. “Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection”. In: *ArXiv* abs/1802.09089 (2018).

[31] Chun Long et al. “Autoencoder ensembles for network intrusion detection”. In: (2022), pp. 323–333. DOI: 10.23919/ICACT53585.2022.9728934.

[32] *Python*. Python Software Foundation. 1991. URL: https://www.python.org/.

[33] Travis E. Oliphant. “NumPy: A fundamental package for scientific computing with Python”. In: *Nature* 585 (2020), pp. 357–362. URL: https://www.nature.com/articles/s41586-020-2649-2.

[34] *pandas: a foundational Python library for data manipulation and analysis*. URL: https://pandas.pydata.org/.

[35] *Matplotlib: A 2D Graphics Environment*. URL: https://matplotlib.org/.

[36] *Seaborn: A Python data visualization library*. URL: https://seaborn.pydata.org/.

[37] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf.

[38] Google Brain Team. *TensorFlow: A machine learning framework*. Google LLC. 2015. URL: https://www.tensorflow.org/.

[39] *Google colab*. URL: https://colab.research.google.com.

[40] *Intrusion Detection Evaluation Dataset (CIC-IDS2017)*. URL: https://www.unb.ca/cic/datasets/ids-2017.html.

[41] *CICFlowMeter tool*. URL: https://www.unb.ca/cic/research/applications.html#CICFlowMeter.

[42] Arnaud Rosay et al. “Network Intrusion Detection: A Comprehensive Analysis of CIC-IDS2017”. In: (Feb. 2022). DOI: 10.5220/0000157000003120.

[43] *CicFlowMeter features description*. URL: https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt.

[44] Mahbod Tavallaee et al. “A detailed analysis of the KDD CUP 99 data set”. In: (2009), pp. 1–6. DOI: 10.1109/CISDA.2009.5356528.

[45] Preeti Aggarwal and Sudhir Kumar Sharma. “Analysis of KDD Dataset Attributes - Class wise for Intrusion Detection”. In: *Procedia Computer Science* 57 (2015). 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015), pp. 842–851. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.07.490. URL: https://www.sciencedirect.com/science/article/pii/S1877050915020190.

[46] Laurens van der Maaten and Geoffrey Hinton. "t-Distributed Stochastic Neighbor Embedding". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

[47] Ankit Thakkar and Ritika Lohiya. "A Review of the Advancement in Intrusion Detection Datasets". In: *Procedia Computer Science* 167 (2020). International Conference on Computational Intelligence and Data Science, pp. 636–645. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.03.330. URL: https://www.sciencedirect.com/science/article/pii/S1877050920307961.

[48] Jiliang Tang, Salem Alelyani, and Huan Liu. "Feature selection for classification: A review". In: *Data classification: Algorithms and applications* (2014), p. 37.

[49] Pengtian Chen, Fei Li, and Chunwang Wu. "Research on Intrusion Detection Method Based on Pearson Correlation Coefficient Feature Selection Algorithm". In: *Journal of Physics: Conference Series* 1757.1 (Jan. 2021), p. 012054. DOI: 10.1088/1742-6596/1757/1/012054. URL: https://dx.doi.org/10.1088/1742-6596/1757/1/012054.

[50] *Matplotlib: A 2D Graphics Environment*. URL: https://xgboost.readthedocs.io/en/stable/.

[51] Wen Xu et al. "Improving Performance of Autoencoder-Based Network Anomaly Detection on NSL-KDD Dataset". In: *IEEE Access* 9 (2021), pp. 140136–140146. DOI: 10.1109/ACCESS.2021.3116612.

[52] Afnan Alotaibi and Murad A. Rassam. "Adversarial Machine Learning Attacks against Intrusion Detection Systems: A Survey on Strategies and Defense". In: *Future Internet* 15.2 (2023). ISSN: 1999-5903. DOI: 10.3390/fi15020062. URL: https://www.mdpi.com/1999-5903/15/2/62.