University of Mohamed Khider, Biskra
Faculty of Exact, Natural and Life Sciences
Computer Science Departement

Anomaly-Based Network Intrusion Detection System: A Machine Learning Approach

Ahmed Ramzi Bahlali

June 29, 2019

Abstract

At the present time, anomaly detection has attracted the attention of many researchers to overcome the weakness of signature-based IDSs in detecting novel attacks, and NSL-KDD benchmark data set is the most used in the literature which it was generated a decade ago, therefore, it does not reflect modern network traffic and low footprint attacks. A new data set has been developed named UNSW-NB15 network data set that came to solve the issues of NSL-KDD in which it has been used in this project to evaluate Anomaly-based NIDs by using different machine learning methods such as Logistic Regression, Decision Tree, Random Forest and particularly an Artificial Neural Network with both binary and multi-class classification that we have discussed their results in details and they have been compared with some previous related works.

keywords- *Network Securtiy; NIDS; UNSW-NB15 data set; Machine learning; Multi Layer Perceptron.*

Acknowledgements

I would like to express my special thanks of gratitude to my supervisor Professor Abdelmalik Bachir for his guidance and support during the project as well for giving me the golden opportunity to work on such a wonderful project, which also helped me doing a lot of research and learn about so many new things.
Secondly I would also like to thank my family members and friends who encouraged me a lot in finalizing this project within the limited time frame.

List of Abbreviations

NIDS	Network-based Intrusion Detection System
HIDS	Host-based Intrusion Detection System
ML	Machine Learning
DL	Deep Learning
ANN	Artificial Neural Network
GD	Gradient Descent
SGD	Stochastic Gradient Descent
MLP	Multi Layer Perceptron
LR	Logistic Regression
DT	Decision Tree
RF	Random forest
DMZ	DeMiliterized Zone
DPI	Deep Packet Inspection

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, due to the utilities and services that the Internet provides such as social media, E-learning, online purchasing, VoIP, cloud services and so on, and the affordability of electronic devices and Internet access, the number of connected devices has increased exponentially. However, it is a nightmare for organizations and corporations security managers to prevent their networks from being attacked and compromised and preserve their secrets and the sensitive information of their customers from leaking out. Hence, cybersecurity has become more challenging than ever.
The firewall with its variants has been shown that it could be easily bypassed by intruders, for instance, by using false source address. Also, it has failed to detect so many attacks such as DoS and DDoS [1]. A new security mechanism has come to existence called *Intrusion Detection System* to overcome the drawbacks of the conventional security schemes. IDS monitors the inbound and outbound traffic for the purpose of detecting malicious ones. IDS can be categorized regarding its placement or the technique it uses to detect abnormal activities [2]. With respects to its placement, IDS could be located at terminals to protect them from being attacked is called Host-based IDS (HIDS) or at network's entry to monitor incoming and outgoing packets to detect malicious ones in order to protect the whole network is called Network-based IDS (NIDS). IDS is classified into signature-based (misuse-based) and anomaly-based [3] depending on the detection technique used. Signature-based IDS uses a database of well-known attack patterns (signatures) and any incoming packet matches one of those patterns is considered as malicious. This class of IDS cannot detect new attacks and its database should be updated continuously [4]. Anomaly-based IDS creates a profile that represents normal behavior and any deviation from this profile is considered as attack. According to [2], anomalies could be categorized either as non-malicious (without any intention to cause harm just appeared because of software bugs or network load for instances) or as malicious (created with intention to harm and destroy network systems). The latter is the type of anomalies security researchers would deal with. The main advantage of anomaly-based IDS is its potential to detect previously unseen attacks [5]. Given the promising capabilities of anomaly-based IDS, it became a principal focus of research and the most investigated

topic among researchers in the literature [2] [5]. Hence, so many techniques have been used in order to build a profile or a model that can perform well in detecting anomalies such as evolutionary, information theory, statistical, and machine learning techniques. Machine learning approach has proved to do good job in problem solving from image classification [6] to machine translation [7]. Therefore, a lot of researchers have adopted this approach with the purpose to address the anomaly detection problem. Both classification and clustering techniques have been used, nevertheless, classification is the commonly used in anomaly detection approach [8]. Classifications techniques have performed well compared to the other existing techniques such as clustering [2]. Classification problem consists of training a model with labeled data and testing it with previously unseen data to check its performance.

Because machine learning is a data driven approach (i.e. it relies heavily on data to a build classifier that generalizes well), a comprehensive dataset that represents network traffic is needed. An older dataset called NSL-KDD has been developed and it was the most used as benchmark in the literature which it was generated a decade ago [9]. However, it has some drawbacks such as: it does not reflect modern network traffic and low footprint attacks[10]. Due to the mentioned reasons, Mustafa et al. [10] have provided an effort in creating a recent dataset named UNSW-NB15 that tackles the issues of NSL-KDD.

Even though is not the first work that tackles the problem of anomaly-detection using machine learning approach, it focuses to address the problem by using different machine learning methods on UNSW-NB15 dataset such as *Logisitic Regression*, *Decision Tree*, *Random Forest* and particularly, a deep learning approach called *Artificial Neural Network* in both cases binary and multi-classification in which their performance results have been discussed in details and they have been compared to previous related work such as the one that has been done by the author of UNSW-NB15 dataset [11].

Chapter 2

Intrusion Detection Systems: An Overview

Intrusion Detection Systems (IDS) are security tools that, like other measures such as antivirus software, firewalls and access control schemes, are intended to strengthen the security of information and communication systems. IDS have arisen due to the weakness of the conventional security mechanisms. But before to dive into the IDS details, let's take a look at network security concepts.

2.1 Network Security

According to [12], network security is any activity designed to protect the usability and integrity of your network and data. It includes both hardware and software technologies. Effective network security manages access to the network. It targets a variety of threats and stops them from entering or spreading on your network. Most security threats are intentionally caused by malicious people trying to gain some benefit, get attention, or harm someone.
Network security problems can be divided roughly into 5 closely intertwined areas such as [4]:

1. **Confidentiality:** only the sender and intended receiver should be able to understand the contents of the transmitted message. Because eavesdroppers may intercept the message. It is achieved using *encryption*.

2. **Message integrity:** ensures that the content of the sent message is not altered, either maliciously or by accident. This is achieved through *checksum* and *hash functions* techniques.

3. **Authentication:** Both the sender and receiver should offered a mechanism to be able to confirm the identity of the other party involved in the communication.

4. **Nonrepudiation:** deals with the issue that someone could deny sending a message or performing an activity. It is achieved through *digital signature*.

5. **Operational security:** Almost all organizations (companies, universities, and so on) today have networks that are attached to the public Internet. These networks therefore can potentially be compromised. Attackers can attempt to deposit worms into the hosts in the network, obtain corporate secrets, map the internal network configuration, and launch DoS attacks. Firewalls and IDSs are used to counter attacks against an organization's network.

Typically, organization networks are structured into two parts such as **internal network** and **DeMilitarized zone**(see Figure 2.1).

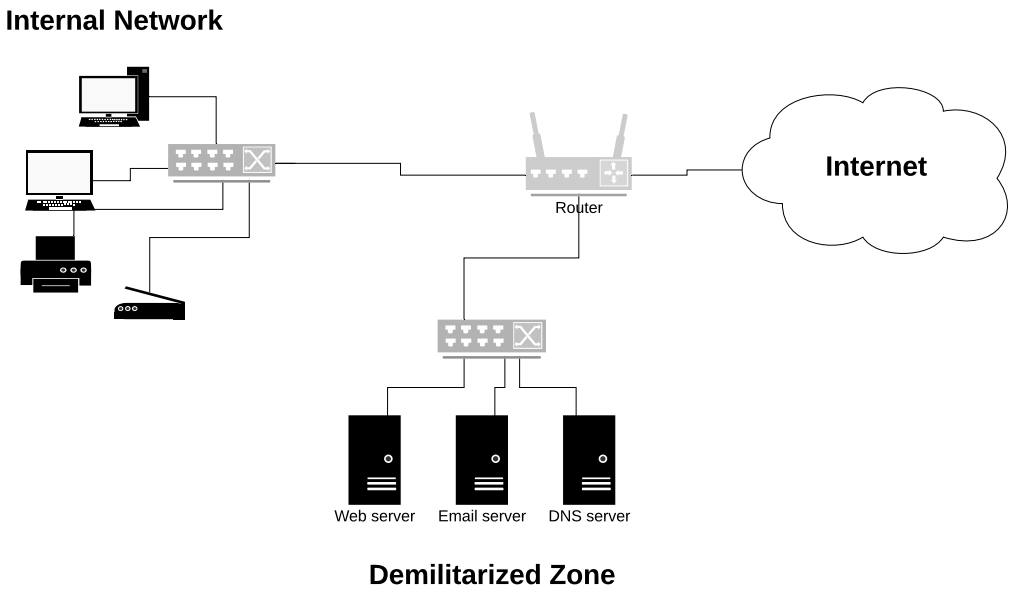


Figure 2.1: Organization network topology

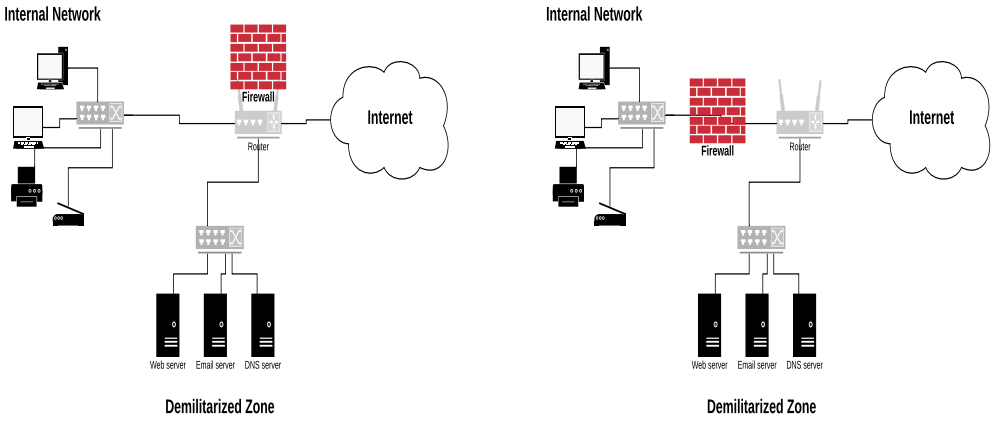
The internal network of the organization is usually accessed only by the network administrators or the employees whereas the demilitarized zone is the part of network's company that is accessed by anyone from the outside such as the Internet, it exposes the organization's services. The purpose of DMZ is to add an additional layer of security to an organization's internal network because the most vulnerable hosts to attack are those that provide services to users outside the internal network such as e-mail, web, and DNS servers. Because of the increased potential of these hosts suffering an attack, they are placed into this specific subnetwork in order to protect the rest of the network from being attacked or compromised. An external network host can only access what is exposed in the DMZ, while the rest of the organization's network is forbidden. However, separating the company's network without adding a mechanism for traffic control does not make a sense. Consequently a conventional mechanism security is added called **Firewall**.

2.2 Firewalls

A firewall is a combination of hardware and software that isolates an organization's network from the Internet, allowing some packets to pass and blocking others. It acts as a packet filter and inspects each and every incoming and outgoing packet. Packets meeting some criterion described in rules formulated by the network administrator are forwarded normally. Those that fail the test are dropped.

2.2.1 Firewall Placement

Within the organization's network, the firewall could take 2 common possible placement; at the router (gateway) that connects the organization's network with the public Internet, or between the router and the internal network (see Figure 2.2). Large organizations may use multiple levels of firewalls or distributed firewalls.



(a) (b)

Figure 2.2: Different Firewall Location

Placing the firewall within the company router could restrict access to authorized traffic (i.e. the traffic going to the DMZ) whereas placing it in between the gateway and the internal network make the DMZ without any defense measure. Each placement has its own drawbacks.

2.2.2 Firewall Categories

Firewalls can be classified into three categories [4]:

Traditional Packet Filters

In this category of firewalls, it examines each datagram in isolation, determining whether the datagram should be allowed to pass or should be dropped

based on a administrator-specific rules. Filtering decisions are typically based on :

- IP source or destination address.
- Protocol type in IP datagram field: TCP, UDP, ICMP, OSPF ... etc.
- TCP or UDP source and destination port.
- TCP flag bits: SYN, ACK, and so on
- ICMP message type
- Different rules for datagrams leaving and entering the network..

A network administrator configures the firewall based on the policy of the organization. For example, if the organization does not want any incoming TCP SYN segments except those for its public Web server. It can block all incoming TCP SYN segments except TCP SYN segments with destination port 80 and the destination IP address corresponding to the Web Server. If the organization doesnt want its internal network to be mapped (tracerouted) by an outsider, it can block all ICMP TTL expired messages leaving the organizations network. A filtering policy can be based on a combination of addresses and port numbers. The major drawback of this firewall category is: basing the policy on addresses provides no protection against datagrams that have their source addresses **spoofed**. Also it proved difficult to write rules that allowed useful functionality but blocked all unwanted traffic [1].

Stateful Packet Filters

Stateful firewalls map packets to connections and use TCP/IP header fields to keep track of connections by using a connection or session table. This allows for rules that, for example, allow an external Web server to send packets to an internal host, but only if the internal host first establishes a connection with the external Web server. Such a rule is not possible with stateless designs that must either pass or drop all packets from the external Web server. The major disadvantage is that sometimes can go into situations where it cannot forward traffic because it has exceeded the capacity of the firewall's connection table.

Application Gateway

This processing involves the firewall looking inside packets, beyond the TCP header, to see what the application is doing. With this capability, it is possible to distinguish HTTP traffic used for Web browsing from HTTP traffic used for peer-to-peer file sharing. Administrators can write rules to spare the company from peer-to-peer file sharing but allow Web browsing that is vital for business. For all of these methods, outgoing traffic can be inspected as well as incoming traffic, for example, to prevent sensitive documents from being emailed outside of the company. Application gateways do not come without

their disadvantages. First, a different application gateway is needed for each application. Second, there is a performance penalty to be paid, since all data will be relayed via the gateway. This becomes a concern particularly when multiple users or applications are using the same gateway machine.

2.2.3 Firewall Drawbacks

Even if the Firewall is perfectly configured, plenty of security problems still exist. For example, if a firewall is configured to allow in packets from only specific networks (e.g., the companys other plants), an intruder outside the firewall can put in false source addresses to bypass this check. If an insider wants to leave out secret documents, he can encrypt them or even photograph them and leave the photos as JPEG files, which bypasses any email filters. Three-quarters of all attacks come from outside the firewall, the attacks that come from inside the firewall, for example, from disgruntled employees, are typically the most damaging [1].

In addition, there is a whole other class of attacks that firewalls cannot deal with. The basic idea of a firewall is to prevent intruders from getting in and secret data from getting out. Unfortunately, there are people who have nothing better to do than try to bring certain sites down. They do this by sending legitimate packets at the target in great numbers until it collapses under the load. Such attack called **DoS** (**Denial of Service**). Usually, the sent packets have false source addresses so the intruder cannot be traced easily. An other variant of DoS attack in which the intruder compromises hundreds of computers elsewhere in the world, and then commands all of them to attack the same target at the same time. Such an attack is calle **DDoS (Distributed Denial of Service)** attack. This attack is difficult to defend against.

It still exist a lot of attacks that the firewall with its variant cannot cope with. Especially for companies and organizations that store sensitive data of their clients. Therefore, a new security mechanism is heavily needed to deal with the attacks that firewall cannot cope with.

2.3 Intrusion Detection Systems

In the previous section, we have seen that a packet filter (firewall) inspects IP, TCP, UDP and ICMP header fields when deciding which packets to let pass through the firewall. However, to detect many attack types especially those the packet filter cannot detect, we need to perfrom **Deep Packet Inspection**. Obviously, there is a place for another device that not only examines the headers of all packets passing through it (unlike a packet filter) but also performs a deep packet inspections. When such a device observes a suspicious packet, or a suspicious series of packets and prevent those packets from entering the organization's network by dropping them is called **Intrusion Prevention System**. Wheres, when the device could let the packets pass through it towards the organizational network, but send an alert to the network administrator or logs

the packet is called **Intrusion Detection System**. In this section we will study intrusion detection in details.

Intrusion Detection Systems (IDS) are automated defense and security systems for monitoring, detecting and analyzing malicious activities within a network or a host. The goal of IDS is to guarantee the security of a network or computer system with regard to *Confidentiality, Integrity* and *Availability*. A firewall is commonly the first defensive line in a network and an IDS is used when there is evidence of an intrusion/attack, which the firewall was unable to stop or mitigate [2]. IDSs could be categorized in many ways [13], whether depending on the monitored platform or regarding the technique they use to detect unusual activities (see Figure 2.3).

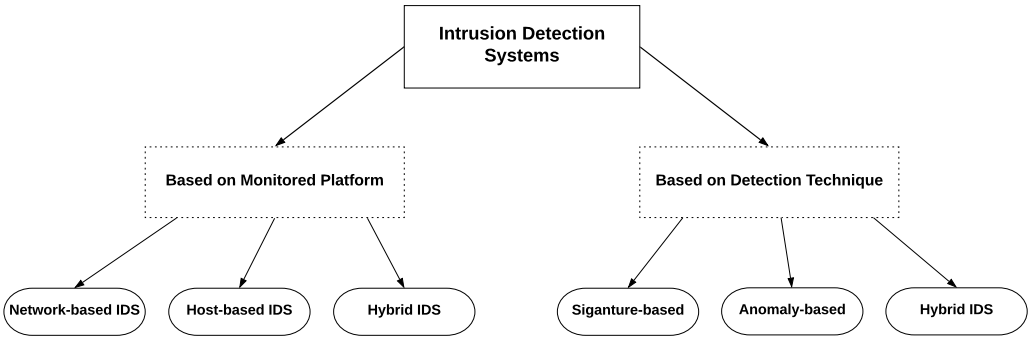


Figure 2.3: IDS Categorization

2.3.1 IDS types by Monitored Platform

Network-based IDS (NIDS)

This class of IDS is usually deployed at the entry of the network in order to protect all hosts such as organization's border router or at the entry of internal network or DMZ. It has different placement. Also an organization may deploy one or more IDS monitors in its organizational network with the purpose to balance the load, particularly it the organization receives gigabits/sec of traffic from the Internet. By placing the IDS monitors at different points within the network, each IDS monitor sees only a fraction of the organization's traffic. It is usually simple to add this type of IDS to a network and they are considered

well secured against attacks [2]. Nevertheless, they have some disadvantages such as the difficulty in analyzing all packets from a large and overloaded network especially when a few number of IDS is deployed. Moreover, they cannot inspect the payload of inbound and outbound packets because they are encrypted, since the decryption takes place on the terminals.

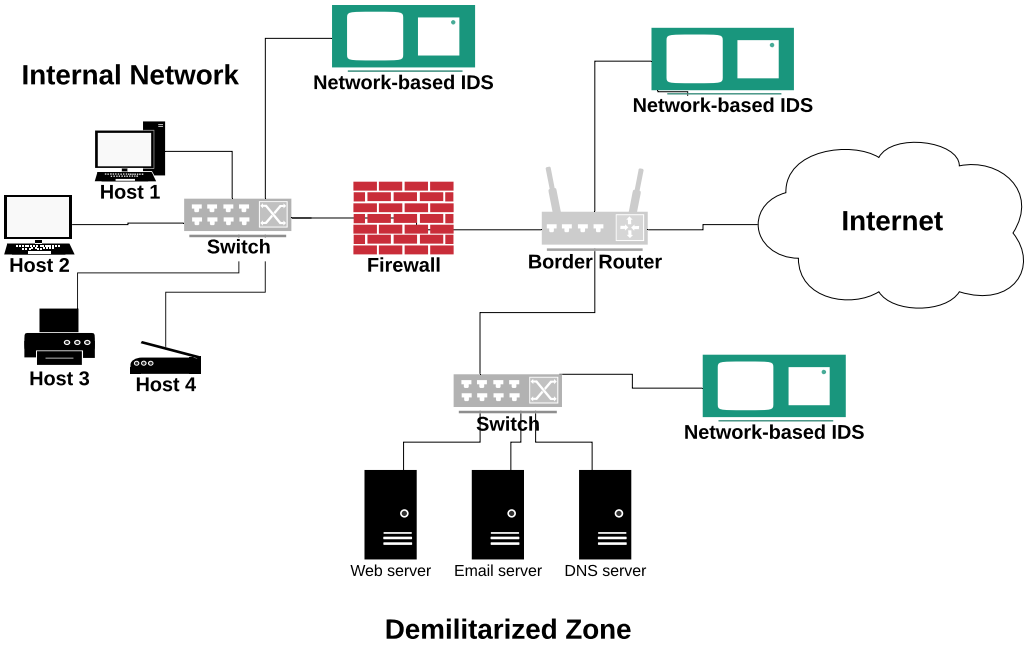


Figure 2.4: NIDS sensor placement

As shown in Figure 2.4, NIDS can be deployed at the 3 different places at the same time, or at one place at least.

Host-based IDS (HIDS)

HIDS is typically deployed at single hosts such as terminals. They are deployed as a software on the host. This class of IDS guarantee the safety of the host by monitoring the the inbound and outbound packets from the host only and will alert the user or administrator if abnormal activity is detected. It provides security against the types of attack that the firewall and NIDS do not detect, such as those based on encrypted protocols. Another benefit of HIDS over NIDS is that the success or failure of an attack can be immediately determined [14]. An illustration of HIDS placement in Figure 2.5

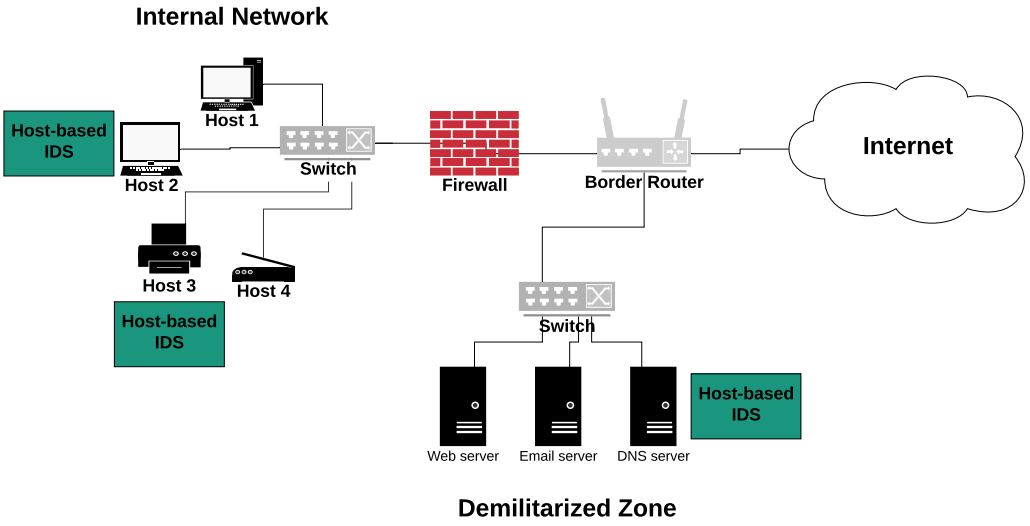


Figure 2.5: HIDS sensor placement

Hybrid IDS

It's a combination of both HIDS and NIDS. These class of systems aggregate the benefits of both approaches while overcoming many of the drawbacks.

2.3.2 IDS types by detection technique

Signature-based IDS

Also denoted as misuse-based IDS. A signature-based IDS maintains a huge and extensive database of attack signatures(rules). Fore this purpose, a signature database that defines the shape of known attack is specified a priori. A signature is simply a list of characteristics about a single packet(e.g., source and destination port numbers, protocol type...etc). Whenever an attempt matches a signature, the IDS triggers an alarm or logging the packet that has been involved for future inspection. This kind of IDS guarantees an efficient detection with low false alarms and good level of accuracy [14]. Despite its advantages, wide deployment and usage, it has a number of drawbacks. Most significantly, it requires previous knowledge of the attack to generate an accurate signature [4]. Moreover, any other packet pattern that does not match the IDS knowledge database is considered normal even it is a new attack or unknown anomalies ,or a little variations of known attacks(i.e. it cannot detect new attacks such as those are not defined in the IDS' dataset). Also, the IDS' dataset should be updated all times especially whenever a new attack pattern

is discovered.

Anomaly-based IDS

An anomaly-based IDS creates a traffic profile from observing usual or normal traffic that reflects normal usage, and any observed deviation of current traffic activity compared to the profile that reflects normal behavior is considered as an anomaly such as malicious. This profile is generated mostly through statistical and historical network traffic data. Although from being able to detect previously unseen attack in which is the main benefit and it does not rely on previous knowledge about existing attacks, the rate of false positives (or FP, events mistakenly classified as attacks) is anomaly-based IDS is usually higher than a signature based ones [5]. Hence, the challenge resides in building a traffic model that distinguishes between normal traffic and statistically unusual traffic [4] with low or maybe now false alarm rate . Anomaly detection technique are the most commonly used IDS detection type and is the most investigated topic in the literature among researchers [2].

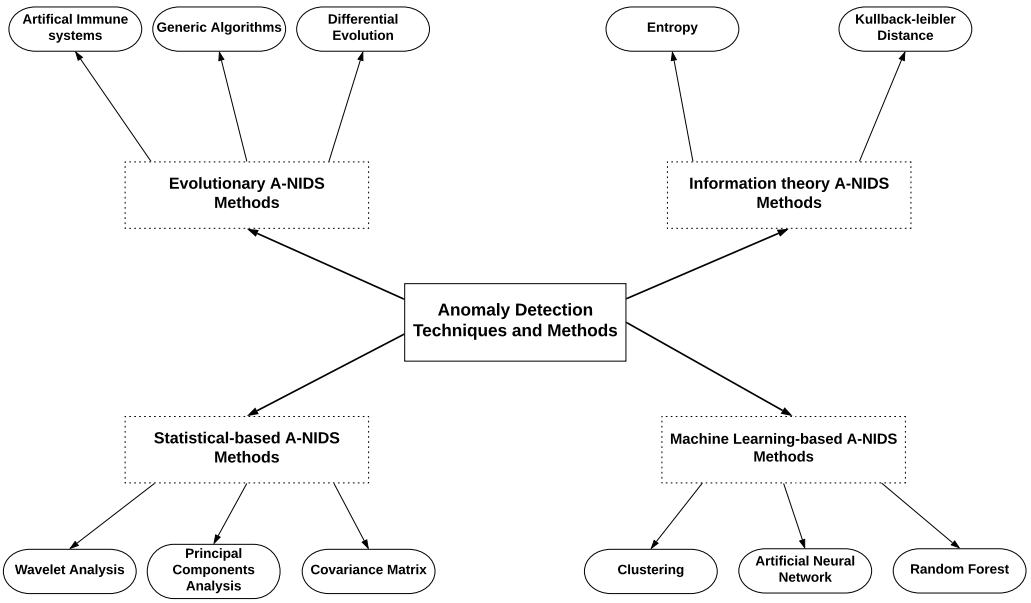


Figure 2.6: Anomaly detection techniques and methods

The subject is a bit far from mature and key issues remain to be solved before wide scale deployment of A-NIDS platforms can be practical [5]. There are many different techniques and algorithms in the literature used to build a network model that is capable to perform a good detection such as statistical procedures, machine learning based schemes, information theory procedures, and heuristics in which they are illustrated in the Figure 2.6.

Hybrid IDS

Hybrid IDSs are a combination of both signature and anomaly based IDS by deploying the functionalities of both.

2.3.3 Network traffic anomalies

There are several types of network traffic anomalies. to put simple, network anomalies can be categorized giving two relevant properties: according to their nature and according to their causal aspect. Figure 2.7 recapitulates the existed network traffic anomalies.

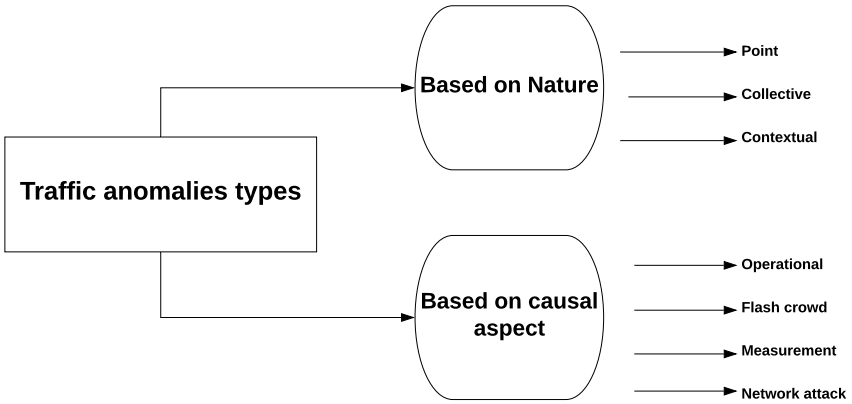


Figure 2.7: Network traffic anomalies categorization

Traffic anomaly based on its nature

- **Point anomaly:** is the alternation or deviation of an single (individual) data instance such as packet from the usual patter/ behavior. These anomalies are considered the simplest ones.
- **Collective anomaly:** happens when a collection of similar data instances(packet flow) behaves anomalously with respects to the whole set of instances. However, in this type of anomalies, individual packets them selfs are not anomalies in fact but the collective behavior is perceived as an anomaly.
- **Contextual anomaly:** are events considered as anomalous depending on the context in which they are found.

Traffic anomaly based on its causal aspect

Actually, network traffic anomalies are not always related to attacks with the intention to harm computer systems or steal information. According to Barford et al. [15] and Marnerides et al. [16], anomalies are grouped into four categories:

- **Misconfiguration events:** also known as Operational events or Failures, they are non-malicious anomalies in which can occur because of hardware failures, software bugs or human mistakes. Server crashes, power outages, misconfiguration, traffic congestion, non malicious large file transfers.
- **Flash crowds:** are large floods in the traffic, which occur when rapid growth of users attempts to access a specific network resource causing a dramatic surge in server load. This category of anomalies is considered as a legitimate (i.e. non-malicious). A cases where flash crowds could occur such as a contest result is published in a website (baccalaureate results), or when an e-commerce website announces a big sale.
- **Measurement:** are not network infrastructure problems, abnormal usage or malicious attacks. These anomalies are related to collection infrastructure problems and problems during data collections. Examples are the loss of flow data caused by router overload.
- **Network attack:** is the kind of anomalies that are created with the intention to disrupt, harm, degrade, deny or destroy information and services from computer network systems compromising their integrity, confidentiality or availability. This category is the most dangerous among the network traffic anomaly categories and to consider when designing anomaly detection systems.

2.4 Anomaly Detection using Machine Learning Approach

Machine learning has been proven theoretically and empirically to be an effective approach in solving so many problems such as image classification [6], speech recognition [17], natural language processing [18], object detection [19], and machine translation [7] ... etc. Applying machine learning to solve those problem has given good results in which we could see them in our daily life for example, Youtube videos recommendation, Google assistant, language translation, face recognition, and Facebook image tags ... etc. Hence, so many researchers have adopted this approach to address the anomaly detection problem. Both classification (supervised learning) and clustering(unsupervised) have been applied in the literature, some of them are shown in Figure 2.8.

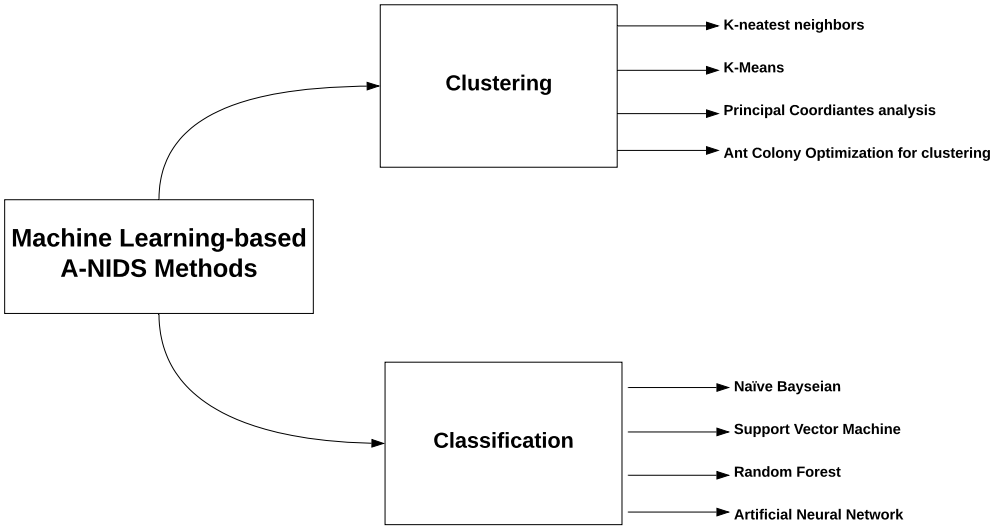


Figure 2.8: Clustering and classification techniques used in the literature

Clustering is the task of grouping a set of objects in such way that objects in the same group (called cluster) are more similar in some sense to each other than to those in other groups (clusters). The main advantage of clustering algorithms is that their implementation do not require a labeled data (i.e. the ground truth of the gathered data) and they provide a stable performance in terms of detection rate and complexity, however, they showed to be time-consuming and highly reliant on proximity measures [2]. Classification is widely used in the anomaly-detection field [8]. The main idea of a such techniques is training(learning) a classifier with labeled data and then testing the performance (i.e evaluating) of the trained classifier in previously unseen data called testing data. Classification based anomaly detection techniques can be either a binary classification (i.e. building a model to be able to distinguish between two classes) or a multi-class classification(i.e. building a classifier that is capable of predicting the class of a given instance among more than two classes). Classifications techniques have the highest detection accuracy amongt all used methods in the literature[2]. Nevertheless, they need labeled data to work and they are demanding in resource consumption.

Chapter 3

Statistical Modeling (Machine Learning and Deep Learning)

3.1 Machine Learning

Machine Learning is an application of Artificial Intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. It means with experience that we have to give the learning algorithm examples or instances and tell it to make inferences from that examples to design a hypothesis and uses it to do the required task. The task differ from application to another for instances Image Classification, House Price Prediction, Speech Recognition, Email Classification (Spam or not), Packet Classification (Normal or Malicious), ... etc.
When it comes to Email Classification For example, suppose that we would like to build an email filter that can distinguish spam (junk) email from non-spam. The machine-learning approach to this problem would be the following: Start by gathering as many examples as posible of both spam and non-spam emails. Next, feed these examples, together with labels indicating if they are spam or not, to your favorite machine-learning algorithm which will automatically produce a classification or prediction rule. Given a new, unlabeled email, such a rule attempts to predict if it is spam or not. The goal, of course, is to generate a rule that makes the most accurate predictions possible on new test examples [20].
In traditional programming, the programmer gives the inputs and tells to the algorithm what to do with these inputs (i.e. the algorithm is designed explicitly to solve a such problem), whereas with ML approach, the programmer must give the input and the output (results) and the ML algorithm will try to find a relationship between them (programe) as shown in the Figure 3.1

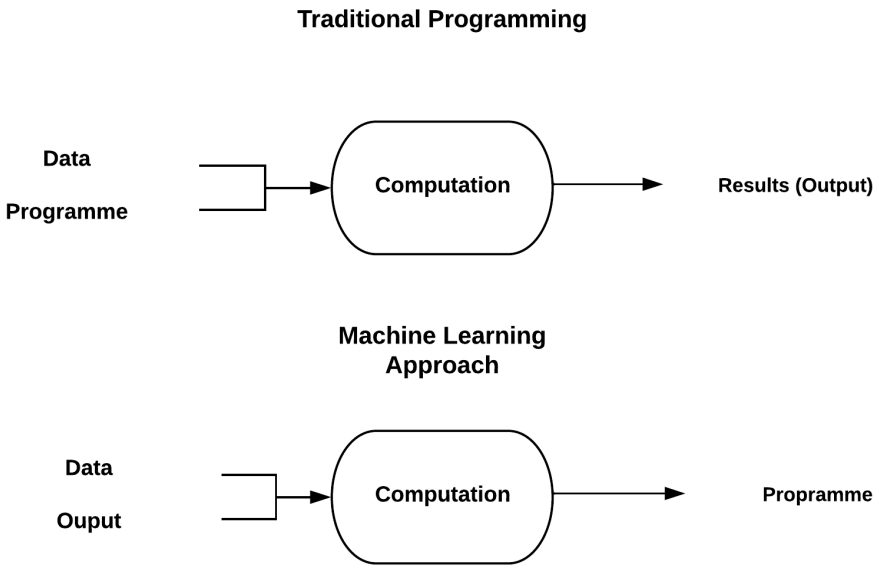


Figure 3.1: Traditional Programming vs Machine Learning Approach

3.1.1 Regression vs Classification

In ML Approach, we could classify the existing problems into two main categories as the following :

1. **Regression** in this category the model predict a continuous value giving an input variable. These are often quantities such as amounts and sizes. For example predicting house price giving its size and number of bedrooms.

2. **Classification** unlike regression, the model gives a discrete value (label or category) that determines in which class or category belong the input variables. For example giving a tumor size and the model try to classify if the tumor either is malignant or benign.

In this project, our aim is to solve a classification problem of network traffic if a packet is normal or malicious. Packet Classification is not a handy task at all, thus, we could not use the traditional approach to build a classifier manually that gives good results (e.g. accuracy or precision ... etc.) Instead of investing effort into writing that complex packet classification function $f(x)$. Its more common and practical to use a Machine Learning approach to tackle that kind of problems. Using labeled data to learn a classification function is called supervised learning.

3.1.2 Supervised Learning

Given a set of instances $\{x_1, x_2, x_3, ..., x_n\}$ in which each instances x_i may have one or more feature followed by the answers or labels of each instance $\{y_1, y_2, y_3, ..., y_n\}$, the learning algorithm try to find a relationship between the inputs(instances) and the outputs (labels) that is represented by a function $h(x) = y$ called the hypothesis and it will be used in predicting the output (label) of a given new instance $h(x') = y'$.
This is what we call Supervised Learning, We train a model with data (instances) given the ground truth (labels) and the model compute a function that map the input with the labels to use it in order to classify new instances. Some popular examples of supervised machine learning algorithms are:

1. **Decision Tree.**
2. **Random Forest.**
3. **Logistic Regression.**
4. **Artificial Neural Network . . . etc.**

In this master-thesis, the above mentioned algorithms have been used to evaluate the performance of anomaly-based IDS.

3.1.3 Features

Features are the set of values that usefully characterize the things we wish to classify. These values are usually aligned in a vector called the feature vector. Examples:

Things to classify	possible features
Packet	duration, ttl, service, protocol used, . . .
Tumor	tumor size, age, . . .
Email	source IP address, sent time, content title,. . .

There is several tool intended to feature extraction, however, the problem is not how to extract features but which features to use in order to construct the dataset to be used to train and evaluate the model. Features extraction usually requires domain knowledge.It is called Feature Engineering.

3.1.4 Machine Learning Model Evaluation

To evaluate the performance of a machine learning model on a given dataset we rely on some metrics such as accuracy, precision, recall, and others. Usually, the dataset is split into two subsets:

- the training set: used to train our model
- the test set: used to evaluate the performance of our model on examples that are not part of the training set

A validation set can be used to track the performance of the model during the learning process. Test and validation sets aren't meant to be as big as the training set.

However, there are two problems with splitting the dataset into two subsets:

- Sometimes we don't even have enough data, thus, selecting only a random subset of the available data may be counter-productive.
- What if the test data is biased? Therefore, the estimated performance can't be as reliable.

To deal with such kinds of problem, there is an evaluation strategy called **k-fold cross validation**. It works as follows:

- Pick an integer k, usually between 5 and 10.
- Split the dataset D into k equally-sized subsets. These subsets should be representative of the whole dataset, it means they contain samples from all classes.
- For each subset D_i, train the model (from scratch) on the other $k-1$ subsets, and evaluate it on D_i.

The average performance estimated over the k training is more reliable given we used all the data we have for both training and testing in the k setups.

3.1.5 Evaluation Metrics

Evaluation of a ML model is an essential part of any project. It is reached by using different evaluation metrics. In this section we will cover different types of evaluation metrics available. Before we dive into those metrics, it is necessary to understand some common terms such as :

- **True Positives (TP)** the number of positive instances classified as positive.
- **True Negatives (TN)** the number of negative instances classified as negative.
- **False Positives (FP)** the number of negative instances classified as positive.
- **False Negatives (FN)** the number of positive instances classified as negative.

Here are some of the most used evaluation metrics:

1. **Classification Accuracy:** the most commonly used metric to judge a model and is actually not a clear indicator of the performance. It is the ration of correctly classified samples to the total number of input samples. It is calculated by the following formula:

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

2. **Precision:** It is the number of correctly classified positive samples divided by the number of sample predicted as positive by the classifier (i.e. the proportion of positive samples correctly classified to the all predicted as positive). It's formula is:

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

   Also known as *positive predictive value (PPV).*

3. **Recall:** It is the number of correctly classified positive samples divided by the number of all passed positive samples .

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

   Also known as *sensitivity* or *true positive rate (TPR).*

4. **Specificity:** It is the number of correctly classified negative samples divided by the number of all passed negative samples .

$$Specificity = \frac{TN}{TN + FP}$$

   Also known as *Selectivity* or *True negative rate (TNR).*

5. **False Positive Rate (FPR):** It is the ratio of misclassified negative samples to all negative samples .

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

6. **False Negative Rate (FNR):** It is the ratio of misclassified postiive samples to all positive samples .

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

7. **F1-Score:** It is the harmonic mean of precision and recall. This takes the contribution of boch, so higher the F1 score, the better.

$$F1Score = \frac{2 * precision * recall}{precision + recall}$$

8. **Confusion Matrix:** It is a simple matrix that contains the four semi-metrics described above (TP, TN, FP and FN). It is structured as shown in Figure 3.2

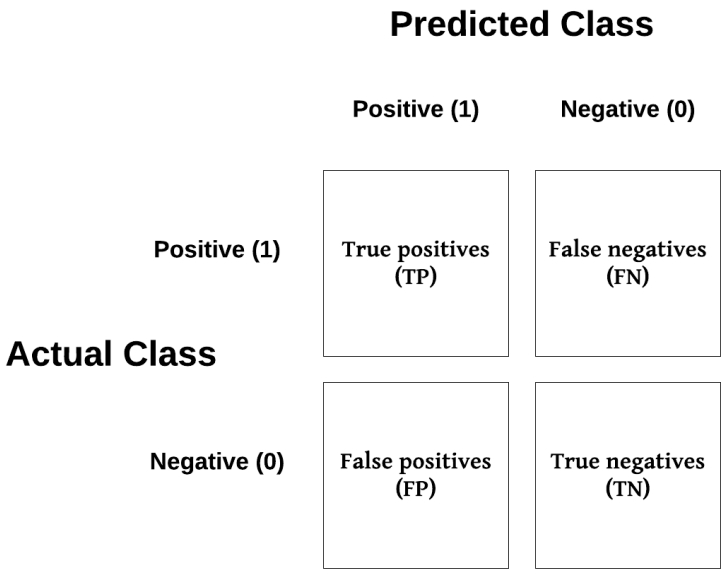


Figure 3.2: Confusion Matrix of a binary classification

3.2 Artificial Neural Networks

Artificial neural networks (ANNs, neural networks or neural nets) are learning models inspired by a simplification of neurons in a brain. Average human has about 100 billion neurons, each of these neurons is connected to a bunch of other neurons. A single neuron may receive an electrical signal from other neurons as an input, the strength of the signal coming from a particular neuron de-pends on how strong is the connection with that neuron, and depending on that signal, the neuron may fire or not. In this section we will explore the basics of artificial neural networks models and how they apply to machine learning.

3.2.1 Artificial neural network architecture

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another in which is a real number, and the output(the real

number) of each artificial neuron is computed by some non-linear function (we explained how it works in details later in this section). The connections between artificial neurons are called 'weights'. Artificial neurons are structured into layers conventionally Input layer, Hidden layer and ultimately the Output layer. The input layer consists of the features of the data set that you aim to train your neural net on where n features corresponds to n input.

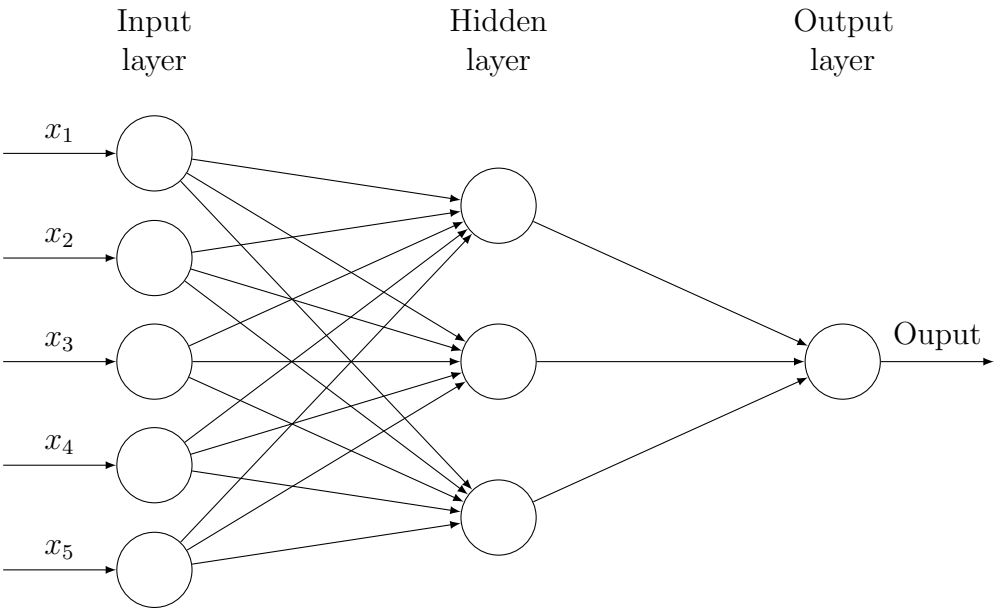


Figure 3.3: Example of a neural net architecture

3.2.2 Artificial Neuron

The artificial neuron also called perceptron is the elementary unit of ANNs exactly where the computation happens. We can perceive the artificial neuron as a function f that:

1. takes as inputs a set of values $x_0, x_1, \ldots x_n$,
2. each input value x_i is multiplied by a weight w_i that reflects how strong that connection is, giving us a weighted input $w_i \times x_i$
3. the weighted inputs now run through a sum

$$\sum_{i=0}^{n} w_i \times x_i$$

4. to follow the convention, they added a term to the weighted sum called the bias b. The bias is like the intercept added in linear equation. It is

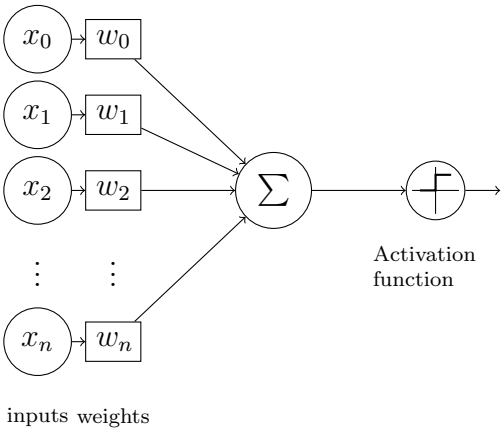


Figure 3.4: Computational model of a neuron

an additional parameter in the neural network which is used to adjust the output along with the weighted sum of the inputs to the neuron. Therefore is a constant which helps the model in a way that it can fit best for the given data.

$$\sum_{i=0}^{n} w_i \times x_i + b$$

The sum of the weighted inputs can be computed as a dot product of the vector x and W.

$$\sum_{i=0}^{n} w_i \times x_i + b = W \cdot x + b$$

5. then the sum is passed through a non-linear function known as an activation function noted $a(x)$.

In a nutshell, a neuron with an activation function $a(x)$ can be seen as a function f parameterized by a vector of weights $W = [w_1, w_2, \ldots, w_n] \in \mathbb{R}^n$ and a bias $b \in \mathbb{R}$ that takes as inputs a vector $x = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^n$:

$$f_{W,b}(x) = a(W \cdot x + b)$$

The output of the activation function used as an input in the next layer's neurons.

3.2.3 Activation functions

The artificial neuron calculates a weighted sum of its input and adds a bias.

$$h_{W,b}(x) = W \cdot x + b$$

The value of $h_{W,b}(x)$ can be anything ranging from $-\infty$ to $+\infty$, the neuron really doesn't know the bounds of the value. So how do we decide whether the neuron should fire or not ? Researchers decided to add a function called *Activation Function* $a(h_{W,b}(x))$ to check whether to fire it or not.
In Addition, the activation function introduce non-linear properties to the neural network because the weighted sum is a simple linear function and it is limited in their complexity and have less power to learn complex functional mapping from data. Hence the activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks. An important feature of the Activation Function is that is should be differentiable. Here are the mostly used activation function in the literature :

- **Sigmoid activiation function**
  It is an activation function of from (see Figure 3.1). Its range is between 0 and 1 and a S-Shaped curve (see Figure 3.5).

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.1}$$

  The Sigmoid function is typically used on the output layer especially

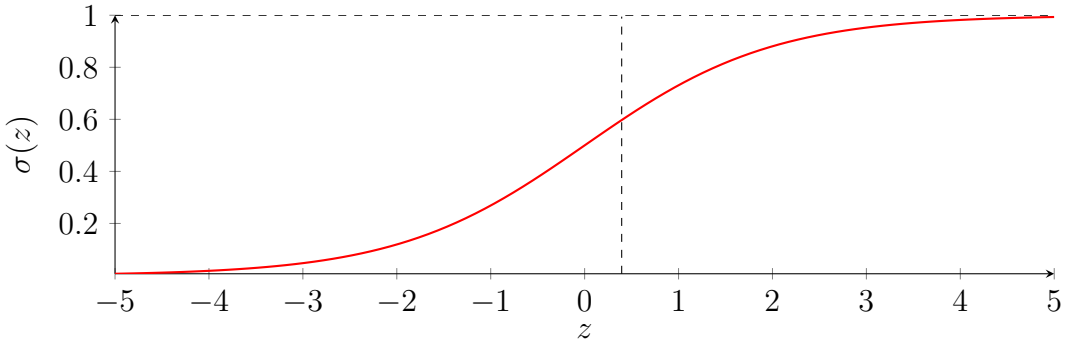


Figure 3.5: Sigmoid activation function

  when it comes to a binary classification problem.

- **Hyperbolic tangent activation function**
  Mathematically, it is a shifted version of the sigmoid function in which performs a squashing between -1 and 1,thus, the inactivity of a neuron is represented by a -1 rather than 0. It is abbreviated to *tanh* and it is used on both hidden and output layers.

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.2}$$

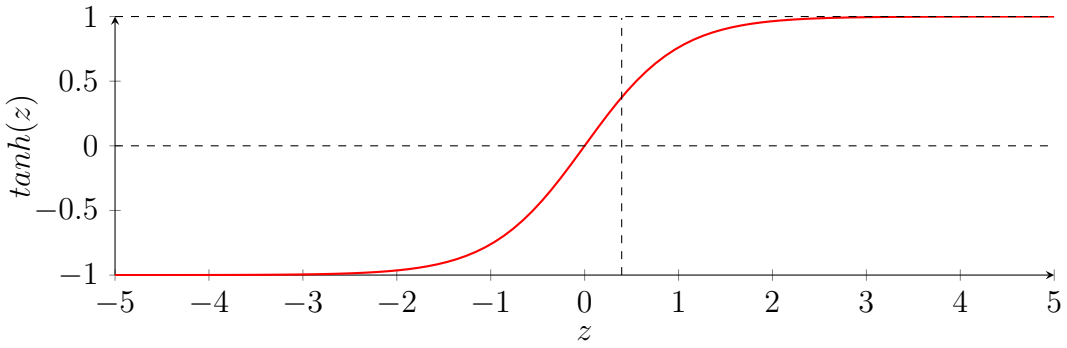


Figure 3.6: Hyperbolic tangent activation function

– **Rectified Linear activation function**
Also called ReLU (Rectified Linear Unit), the ReLU activation is considered by experts as the most popular activation function for deep learning [21]. It has strong biological motivations and mathematical justifications [22].

$$ReLU(z) = \max(z, 0) \tag{3.3}$$

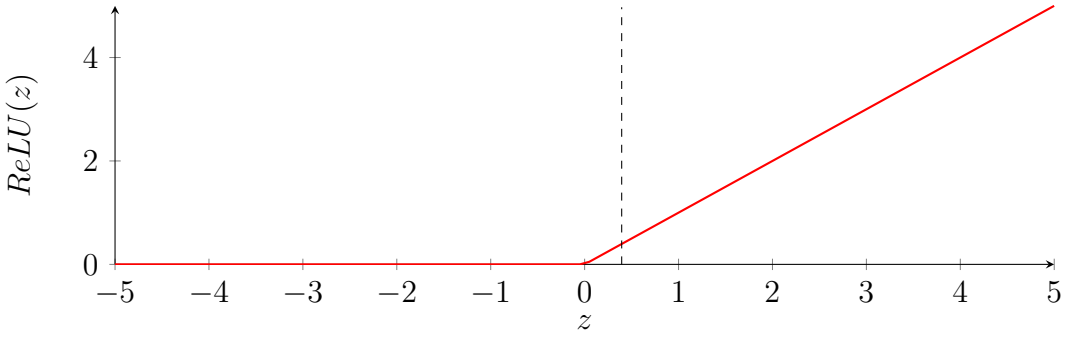


Figure 3.7: Rectified Linear activation function

– **Softmax activation function**
Also known as softargmax. It is a function that takes as input a vector of k real numbers and normalizes it into a probability distribution consisting of k probabilities. The softmax function of the i^{th} neuron is defined by the formula see Figure 3.4

$$\begin{aligned} Softmax(z)_i &= \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \\ \text{and} \quad z &= (z_1, \ldots, z_k) \in \mathbb{R}^k \end{aligned} \tag{3.4}$$

That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval $(0, 1)$, and the components will add up to 1, so that they can be interpreted as probabilities.

It is used mostly on the output layer when it comes to a multi-class classification to calculate the probability in which class a given input belong to.(the output that has the highest probability).

3.2.4 Feed-forward Neural Network

Also know as *multi-layer perceptrons(MLP)* or *Deep Feed-forward*. Multi-layer perceptrons are the foundation of most *deep learning* models. Networks like *Convolutional neural networks CNNs* and *Recurrent neural networks RNNs* are some special cases of feed-forward neural networks (FFNNs). FFNN was the first and simplest type of artificial neural network devised [23]. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden layer nodes and to the output nodes. There are no cycles of loops in the network. MLP consists of:

- first layer(input layer) consists of the input units, a vector $x = [x_1, \ldots, x_n]$. Also called features that we aim to train our model on.
- hidden layers (at least 1), usually used to perform representation transformation, it is exactly where the computation happens.
- a single output layer, that represents the output of the network, usually used to perform classification on the representation given by the last hidden layer.

A layer is a set of neurons that have the same activation function, the same number of inputs (thus, the same number of weights), but may have different values for their parameters (weights and bias)

- The first hidden layer, noted $h^{(1)}$, each neuron in this layer is noted $h_k^{(1)}$ where k represents the k^{th} neuron neuron in this layer. Each neuron $h_k^{(1)}$ takes as input a vector $x = [x_1, x_2, \ldots, x_n]$ that the represents the features of the data and a vector of weights associated to that neuron $W_k = [w_1, w_2, \ldots, w_n]$ and a bias b_k . We end up having the following formula:
$$h_k^{(1)}(x) = a(W_k \cdot x + b_k)$$
note: $a(.)$ is the activation functions.
The same to the other neurons at this layer.
- The second hidden layer noted $h^{(2)}$, each neuron $h_k^{(2)}$ takes as input a vector of weights W_k and a bias b_k but rather than taking a vector x(the input features), it takes a vector that represents the output h of each neuron of the previous layer noted $x = [h_1^{(1)}, h_2^{(1)}, \ldots, h_i^{(1)}]$. We end up having the following formula:

$$h_k^{(2)}(x) = a(W_k \cdot x + b_k)$$

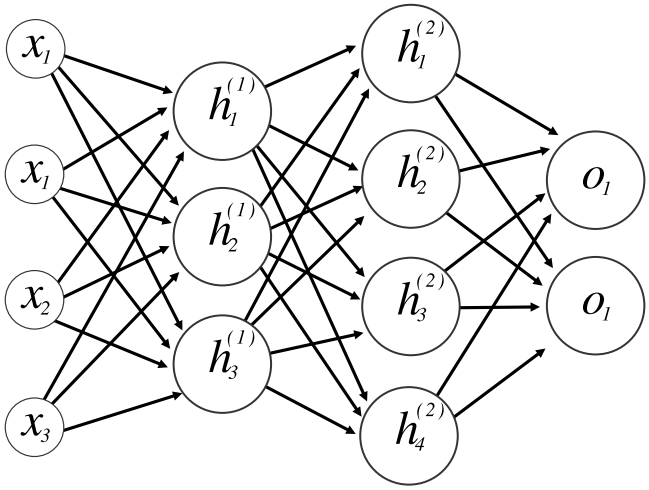


Figure 3.8: A simplified graphical visualization of the feed-forward neural network model. This FFNN takes 4 inputs, has 2 hidden layers $h^{(1)}$ and $h^{(2)}$ of sizes 3 and 4 respectively, and has an output layer of size 2. Biases were omitted only from this visualization. We can see how each two adjacent layers are fully connected to each other

The same with the other neurons at this layer, also the next layer until we reach the output layer. This is why it is called Feed-Forward (It keeps feeding the next layers with the result of the previous layers until it reaches the output layer).

In a nutshell, let's consider the following:

- $h_k^{(i)}$ is the k^{th} neuron of the i^{th} layer.
- $h^{(n)}$ is the neuron of the output layer.
- W^i is an $n \times m$ matrix ($W^k \in \mathbb{R}^{n \times m}$) that represents the weights of i^{th} layer where n is the number of neurons in this layer and m in the dimension of inputs (or the number of neurons at the previous layer).

$$W^i = \begin{bmatrix} w_{1,1} & w_{1,2}, w_{1,3}, \ldots, w_{1,m} \\ w_{2,1} & w_{2,2}, w_{2,3}, \ldots, w_{2,m} \\ \vdots & \\ w_{n,1} & w_{n,2}, w_{n,3}, \ldots, w_{n,m} \end{bmatrix} \tag{3.5}$$

  w_k^i row represents the set of weights of the k^{th} neuron and $w_{k,j}^i$ represents the j^{th} weight of k^{th} neuron at the i^{th} layer (see Figure 3.5).

- $b^i = [b_1, b_2, \ldots, b_n]$ is the vector of bias at the i^{th} layer where b_k is the k^{th} neuron's bias at this layer.
- $x^i = [x_1, x_2, \ldots, x_n]$ represents the input vector of the i^{th} layer.

We can write the hypothesis h of a given neuron(k^{th} neuron at the i^{th} layer) as the following:

$$\begin{aligned} h_k^{(i)}(x) &= a(W_k^i \cdot x^i + b_k^i) \\ h_k^{(i)}(x) &= a(W_{k,1}^i \cdot x_1^i + W_{k,2}^i \cdot x_2^i + \ldots + W_{k,m}^i \cdot x_n^i + b_k^i) \end{aligned} \tag{3.6}$$

Note: $x = h^{(i-1)} = [h_1^{i-1}, h_2^{i-1}, \ldots, h_n^{i-1}]$ is the output vector of the previous layer.

Each layer i output a vector of values noted $h^{(i)} = [h_1^i, h_2^i, \ldots, h_n^i]$ in which it will be used as input at the next layer, hence, we can say that a feed-forward neural network of L hidden layers is a composed function f that takes as input a vector x (features), and returns a vector of output y $\in \mathbb{R}^1$ (one dimension) or y $\in \mathbb{R}^n, n > 1$ in case of multi-class classification:

$$y = f_\theta(x) = a(h^{(L)}(h^{(L-1)}(\ldots(h^{(1)}(x))\ldots)))$$

where:
$\theta = \{W^1, W^2, \ldots, W^n\} \cup \{b^1, b^2, \ldots, b^n\}$ the set of network parameters.

3.2.5 Feed-froward neural network Training

In a classification problem, given a dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ where x_i is a set of features and y_i is the class or category where the given features belong to ($y \in [1, 0]$ 1 for malicious and 0 for normal packet when it comes to packet classification for example). We can see the dataset D as function g that maps some input x_i to an output y_i , $y = g(x)$ in which is not defined mathematically. As we have seen before, a feed-forward neural network is a function f parameterized by the θ (the set of all weights and biases). The goal of training a neural net is to find the right parameters θ for the function f_θ to approximate the function g. This is why FFNN is called *universal function approximator*.
The approximated function f can be interpreted as $f_\theta(x) = P(y = 1|x; \theta)$ the probability that $y = 1$ (by convention 1 represents the positive class) given the input vector of features x and paramatrized by θ. This is because the neurons of the output layer use function that outputs a probability distribution (a value between 0 and 1 as we have seen in the activation functions section).
Training a neural network it is all about choosing the set of model parameters θ so that $f_\theta(x)$ to be close to y (the actual class) for our training examples (x, y). In fact training a neural network mathematically is an *optimization problem* more precisely a *minimization problem* in which we need to choose the right parameters θ that minimize the difference between the predicted class and actual class $\mid f_\theta(x) - y \mid$ but it is not evident to choose θ with respect to one training example, hence, rather than using one example we will use the whole dataset by summing up over the dataset of size m. Also it is important to square the difference because mathematically it is more convenient. We end up having a function J called the *cost function* that is paramatrized by θ and

has the following formula :

$$\underset{\theta}{\operatorname{argmin}}\, J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(f_\theta(x^i) - y^i)^2$$

More precisely, the previous form of cost function called the *Mean Squared Error (MSE)* . It is the simplest and the most used among cost functions.
Briefly, training a neural network model is to find the set of parameters θ such that cost function $J(\theta)$ is minimum by changing the values of θ. Of course we can not keep changing them randomly because it takes a lot of time and often we can not fall at the optimum. Thus, It is required to use an algorithm to guide the training and guarantee to fall at the right parameters as fast as possible.

3.2.6 Optimization algorithms

Optimization algorithms helps us to minimize (or maximize) an objective function (cost function) $J(\theta)$ which is simply a mathematical function dependent of the Model's internal learnable parameters θ which are used in computing the predicted class $y^{'} = f_\theta(x)$. Optimization algorithms play an important role in model training. So, choosing the convenient one will make our model fit better to the data and consume much less time to converge (find the optimum parameters). It exists several optimization algorithms in the literature but in this section we will use most widely used and the foundation of how we train and optimize intelligent systems called **Gradient Descent**.
Gradient descent is an iterative optimization algorithm that finds a local minimum of a function.This algorithms uses the information that the gradient gives us about the function at a given input: the direction of the steepest increase, but since we are looking to decrease the functions output, we follow the negative gradient at each step. To update the parameters of the function $J(\theta)$ that we aim to minimize, we use the following formula:

$$\theta = \theta - \alpha \cdot \nabla J(\theta) \tag{3.7}$$

where:

- θ is the set of parameters of the network that they were initialized randomly.

$$\theta = \{W_1, W_2, \ldots, W_n\} \cup \{b_1, b_2, \ldots, b_n\} = \{\theta_1, \theta_2, \ldots, \theta_n\}$$

- α is the learning rate. It determines the size of the step towards the optimal parameters. It is required to set the learning rate to an appropriate value, which is neither too low nor too high this is because if the steps it takes are too big, it maybe misses the local minimum and if the steps are too small it will take a lot of time to converge (time consuming). See Figure 3.9.

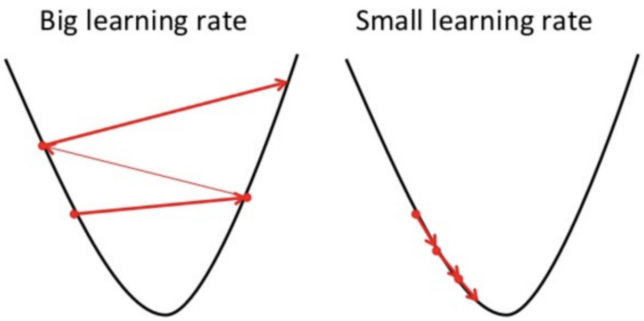


Figure 3.9: a simplified visualization of learning rate influence α in finding the local minimal of a the cost function

– $\nabla J(\theta)$ is the gradient(partial derivative) of the cost function with respect to its input because it is a multi-variable function that takes multiple variables as input(the set of parameters). For each $\theta_i \in \theta$:

$$\theta_i = \theta_i - \alpha \cdot \nabla J(\theta_i) \tag{3.8}$$

where :

$$\nabla J(\theta_i) = \frac{\partial J(\theta)}{\partial \theta_i}$$

An example (see Figure 3.10) where we need to find the local minimal of a function that has one parameter $J(\theta)$ (just to simplify). At the point A, the gradient is negative $\nabla J(\theta_i) < 0$, so the value of the next θ is going to increase in order to minimize a bit $J(\theta)$ accroding to the formula 3.10. in contrast to the point B where the gradient is positive $\nabla J(\theta_i) > 0$, hence, the next θ value according to the same formula is going to decrease to minimize $J(\theta)$.
It exists another sophisticated alternative to gradient descent such as:

– Stochastic gradient descent (SGD) in contrary, does this for each training example within the dataset. This means that it updates the parameters for each training example, one by one. It is called stochastic because the method uses randomly selected (or shuffled) samples to evaluate the gradients. This can make SGD faster than Gradient Descent, depending on the problem. One advantage is that the frequent updates allow us to have a pretty detailed rate of improvement.

$$\theta_i = \theta_i - \alpha \cdot \nabla J(\theta_i) \tag{3.9}$$

where:

$$J(\theta) = (f_\theta(x^i) - y^i)$$

– Mini-batch Gradient Descent is the go-to method since its a combination of the concepts of SGD and Gradient Descent. It simply splits the training dataset into small batches and performs an update for each of

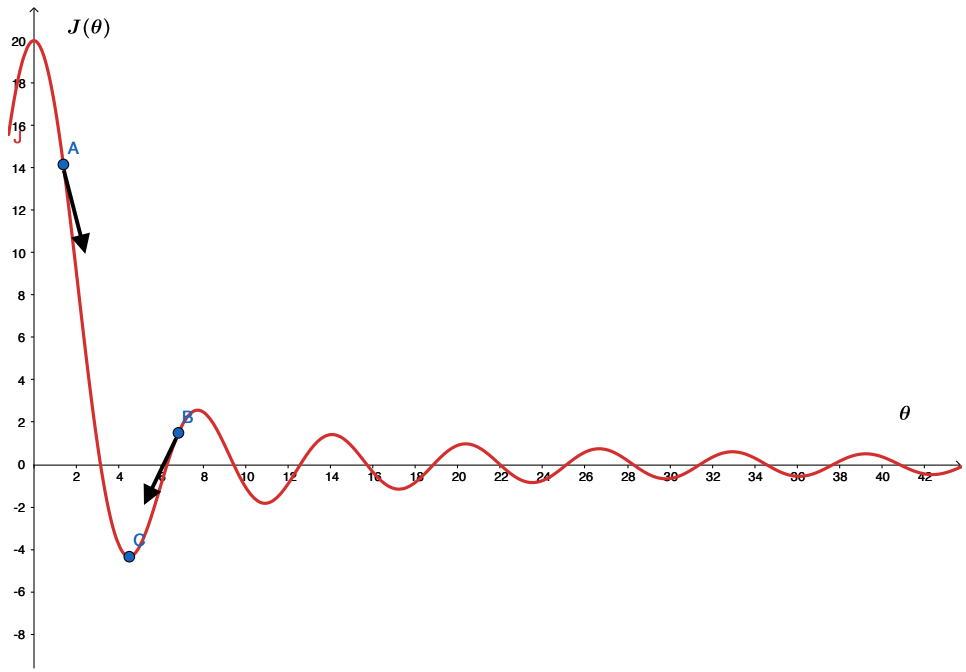


Figure 3.10: A simple example of a cost function
that has one parameter θ. From two point initialized randomly A and B, it visualizes which direction we should take to reach the local minimal of $J(\theta)$.

these batches. Therefore it creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

$$\theta_i = \theta_i - \alpha \cdot \nabla J(\theta_i) \tag{3.10}$$

where:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(f_\theta(x^i) - y^i)^2 \text{ and } 1 < m < \text{dataset size}$$

3.2.7 ANN Hyper-Parameters Tuning

Hyper-parameter is a configurable value which is set before the learning process begins. These hyper-parameter values dictate the behavior of the training algorithm and how it learns the set of parameters θ from the data. Choosing a set of optimal hyper-parameters for a learning algorithm requires expertise and extensive trial and error. However, it exist two common used methods called *Grid search* and *Random search* that can ease the process. Before to dive into those two method, we have to take a look to the ANNs' hyper-parameters that affect its training process and performance such as:

1. **Number of Layers:** It must be chosen wisely as a very high number may introduce problems like over-fitting and vanishing and exploding

gradient problems and a lower number may cause a model to have high bias and low potential model. Depends a lot on the size of data used for training.

2. **Number of hidden units per layer:** These too must be chosen reasonably to find a sweet spot between high bias and variance. Again depends on the data size used for training.

3. **Activation Function:** The popular choices in this are ReLU, Sigmoid and Tanh(only for shallow networks 1 or 2 hidden layers), and LeakyReLU. Generally choosing a ReLU/LeakyReLU do equally well. Sigmoid/Tanh may do well for shallow networks. Identity helps during regression problems.

4. **Optimizer:** It is the optimization algorithm used by the model to update weights of every layer after every iteration. Popular choices are SGD, RMSProp and Adam. SGD works well for shallow networks but cannot escape saddle points and local minima in such cases RMSProp could be a better choice, AdaDelta/AdaGrad for sparse data whereas Adam is a general favorite and could be used to achieve faster convergence.

5. **Learning Rate:**It is responsible for the core learning characteristic and must be chosen in such a way that it is not too high wherein we are unable to converge to minima and not too low such that we are unable to speed up the learning process. Recommended to try in powers of 10, specifically 0.001,0.01, 0.1,1. The value of the learning rate is dependent a lot on the optimizer used. For SGD - 0.1 generally works well whereas for Adam - 0.001/0.01 but it is recommended to always try all values from the range above. You can also use the decay parameter,to reduce your learning with number of iterations, to achieve convergence.Generally it is better to use adaptive learning rate algorithms like Adam than using a decaying learning rate.

6. **Initialization:** Doesn not play a very big role as defaults work well but still it is preferred to use He-normal/uniform initialization while using ReLUs and Glorot-normal/uniform (the default is Glorot-uniform) for Sigmoid for better results. One must avoid using zero or any constant value(same across all units) weight initialization.

7. **Batch Size:** It is indicative of number of patterns shown to the network before the weight matrix is updated. If batch size is less, patterns would be less repeating and hence the weights would be all over the place and convergence would become difficult. If batch size is high learning would become slow as only after many iterations will the batch size change. It is recommend to try out batch sizes in powers of 2 (for better memory optimization) based on the data-size.

8. **Number of Epochs:** The number of epochs is the number of times the entire training data is shown to the model. It plays an important role in how well does the model fit on the train data. High number of epochs may over-fit to the data and may have generalization problems on the test and validation set, also they could cause vanishing and exploding gradient problems. Lower number of epochs may limit the potential of the model. Try different values based on the time and computational resources you have.

There is not a deterministic rule or a pre-defined set of hyper-parameters that guarantee a good performance. For popular problems, like image classification, one can use hyper-parameters published in known paper and make small adjustments. However, for relatively under- developed problems like the one we are tackling for instance, one should experiment with different hyper-parameters until they get the best set, this might require time and/or computational resources. Evaluating the performance of a given set of hyper-parameters is done in the validation processes (by using a set of data that the model has never seen them before: validation set). As we mentioned before, there are two common approaches that ease the processes of choosing the optimal hyper-parameters:

- **Grid Search:**The traditional way of performing hyper-parameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyper-parameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set [24] or evaluation on a held-out validation set [25].

- **Random Search:** is a technique where random combinations of the hyper-parameters are used to find the best solution for the built model. It tries random combinations of a range of values. The chances of finding the optimal parameter are comparatively higher in random search. Researchers have shown empirically and theoretically that random search is more efficient for parameter optimization than grid search [26].

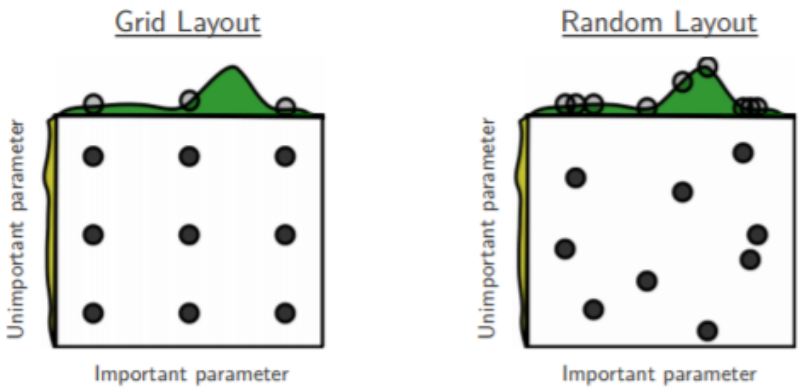


Figure 3.11: As shown in the figure, random search has found the optimal hyper-parameters less number of iterations. In grid search, however, the optimal hyper-parameter is not found since we do not have it in our grid.

Chapter 4

UNSW-NB15 Network Data Set

In the field of machine learning, the dataset used to train and test the model plays an important part in model's performance. ML depends heavily on data. Its the most crucial aspect that makes algorithm training possible and explains why machine learning became so popular in recent years. But regardless of your actual terabytes of information and data science expertise, if you cant make sense of data records, a machine will be nearly useless or perhaps even harmful. An expertise in the domain have to be part of the work team that they are constructing the dataset (a medicine in cancer detection for instance) to select the more convenient features (proprieties). This process called feature engineering, it means using domain knowledge of the data to create features that make machine learning algorithms work.
The effectiveness of a ML model is evaluated based on their performance on a given dataset in which must be reliable and reflects real data points.
In network security, evaluating the performance of NIDS has been done by using older benchmark dara sets are KDDCUP 99 and NSLKDD. It is percevied through several studies [27] [28] [29] [30]. Evaluating a NIDS using these data sets does not reflect realistic output performance due to several reasons [10]. First reason is the KDDCUP 99 data set contains a tremendous number of redundant records in the training set. The Redundant records affect the result of detection biases toward the frequent records [29]. Second, there are also multiple missing records that are a factor in changing the nature of the data [28]. Third, The NSLKDD data set is the improved version of the KDDCUP 99, it tackles the several issues such as data unbalancing among the normal/abnormal records and the missing values [9]. However, this data set is not a comprehensive representation of a modern low foot print attack environment [10]. the reasons mentioned above have pushed me to use UNSW-NB 15 dataset in this project.

4.1 Dataset description and details

4.1.1 Dataset generation

UNSW-NB15 data set is a hybrid of the real moder normal behaviors and the synthetical attack activities. Abnormal traffic has been generated using IXIA PerfectStorm traffic generator tool that simulates nine families of attacks from a CVE site (this site is a dictionary of publicly known information security vulnerabilities and exposures). Both normal and malicious traffic have captured using the TCPdump tool and saved in a form of pcap files as row packets. Extracting useful features from the raw packet has been done by using Argus and Bro-IDS. Additionally, twelve algorithms are developed using a C # language to analyze in depth the flows of the connection packets. The data set is labelled from a ground truth table that contains all simulated attack types. Ultimately, saving them into a CVP files. The Figure 4.1 recapitulate the process of UNSW-NB15 data set generation.

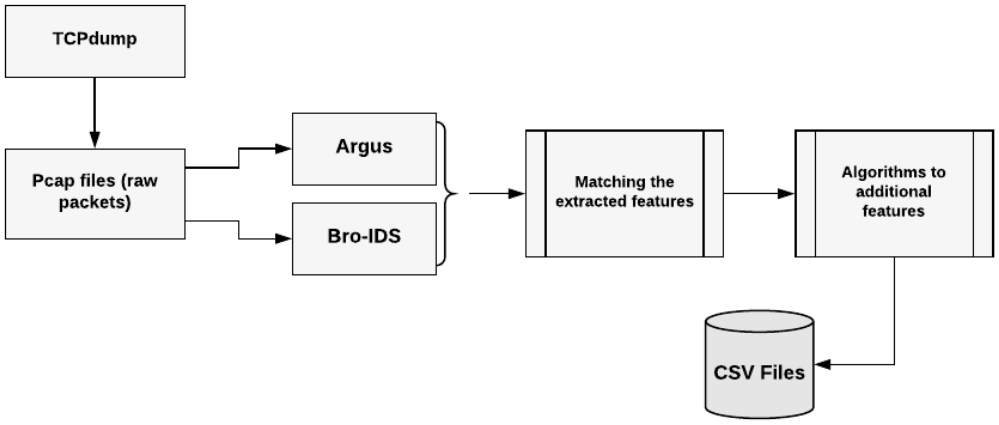


Figure 4.1: Framework Architecture for Generating UNSW-NB15 dataset

4.1.2 Dataset Features

the UNSW-NB15 dataset consists of 49 feature in each record in the form of 5 data types: nominal, integer, float, timestamp and binary [10]. The extracted features have been categorized into 6 groups: flow features, basic features, content features, time features, additional generated features and labelled features [10]. There are 4 features in the flow features (Source and destination IP, source and destination port number) that we can not use them in machine learning approach because they will cause the model to be biased towards those addresses, hence, it is more convenient to remove them from the feature set. Here are the 49 features and their description:

No	Name	Type	Description
1	*srcip*	Nominal	Source IP Address
2	*sport*	Integer	Source port number
3	*dstip*	Nominal	Destination IP address
4	*dsport*	Integer	Destination port number
5	*proto*	Nominal	Transaction protocol

Table 4.1: Flow features

No	Name	Type	Description
6	*state*	Nominal	The state and its dependent protocol, e.g. ACC, CLO, else(-)
7	*dur*	Float	Record total duration
8	*sbytes*	Integer	Source to destination bytes
9	*dbytes*	Integer	Destination to source bytes
10	*sttl*	Integer	Source to destination time to live
11	*dttl*	Integer	Destination to source time to live
12	*sloss*	Integer	Source packets retransmitted or dropped
13	*dloss*	Integer	Destination packets retransmitted or dropped
14	*service*	Nominal	http, ftp, ssh, dns,. . . , else (-)
15	*sload*	Float	Source bits per second
16	*dload*	Float	Destination bits per second
17	*spkts*	Integer	Source to destination packet count
18	*dpkts*	Integer	Destination to source packet count

Table 4.2: Basic Features

No	Name	Type	Description
19	*swin*	Integer	Source TCP window advertisement
20	*dwin*	Integer	Destination TCP window advertisement
21	*stcpb*	Integer	Source TCP sequence number
22	*dtcpb*	Integer	Destination TCP sequence number
23	*smeansz*	Integer	Mean of the flow packet size transmitted by the src
24	*trans-depth*	Integer	The depth into the connection of http request/response transaction
25	*res-bdy-len*	Integer	The content size of the data transferred from the servers http service.

Table 4.3: Content Features

No	Name	Type	Description
26	*sjit*	Float	Source jitter (msec)
27	*djit*	Float	Destination jitter (msec)
28	*stim*	Timestamp	Record start time
29	*ltime*	Timestamp	Record last time
30	*sintpkt*	Float	Source inter-packet arrival time (msec)
31	*dintpkt*	Float	Destination inter-packet arrival time (msec)
32	*tcprtt*	Float	The sum of synack and ackdat of the TCP.
33	*synack*	Float	The time between the SYN and the SYN-ACK packets of the TCP.
34	*ackdat*	Float	The time between the SYN-ACK and the ACK packets of the TCP.

Table 4.4: Time Features

Features are categorized into six groups:

1. **Flow features:** Includes the protocol used between hosts, as reflected in the table 4.1

2. **Basic features:** involves the attributes that represent protocols connections (i.e. session characteristics), as shown in the table 4.2

3. **Content features:** encapsulates the attributes of TCP protocol, also they contain some attributes of http services, as reflected in 4.3

4. **Time features:** contains the attributes time, for example, arrival time between packets(jitter), start/end packet time, and round trip time of TCP prtocol, as shown in the table 4.4

5. **Additional features:** in the table 4.5, this category can be further divided into two groups: general purpose features (i.e., 3640), whereby each feature has its own purpose, according to protect the service of protocols, and (2) connection features (i.e., 4147) that are built from the flow of 100 record connections based on the sequential order of the last time feature.

6. **Labelled features:** used to identify the class (i.e. label) the data set. Two attributes were provided: attack-category represents the nine categories of the attack and the normal, and label is 0 for normal and 1 otherwise, as shown in the table 4.6

No	Name	Type	Description
35	*is-sm-ips-ports*	Binary	If source (1) equals to destination (3)IP addresses and port numbers (2)(4) are equal, this variable takes value 1 else 0
36	*ct-state-ttl*	Integer	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).
37	*ct-flw-http-mthd*	Integer	No. of flows that has methods such as Get and Post in http service.
38	*is-ftp-login*	Binary	If the ftp session is accessed by user and password then 1 else 0
39	*ct-ftp-cmd*	Integer	No of flows that has a command in ftp session.
40	*ct-srv-src*	Integer	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).
41	*ct-srv-dst*	Integer	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
42	*ct-dst-ltm*	Integer	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
43	*ct-src-ltm*	Integer	No. of connections of the same source address (1) in 100 connections according to the last time (26).
44	*ct-src-dport-ltm*	Integer	No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
45	*ct-dst-sport-ltm*	Integer	No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).
46	*ct-dst-src-ltm*	Integer	No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).
47	*Label*	Binary	0 for normal and 1 for attack records.

Table 4.5: Additional Generated Features

No	Name	Type	Description
48	*attack-cat*	Nominal	The name of each attack category. In this data set, nine categories (e.g., Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms)
49	*Label*	Binary	0 for normal and 1 for attack records.

Table 4.6: Labelled Features

4.1.3 Dataset Attack Categories

Multi-class classification is the problem of classifying instances into one of three or more classes. For this purpose, the feature *Attack-category* has been introduced among UNSW-NB15 dataset features to categorize the records into specific classes are the following:

1. **Normal:** The class of the natural traffic that can not harm the target host or network (normal usage).

2. **fuzzers:** Ejecting invalid, unexpected or random data as an input to discover coding errors and security vulnerabilities in a software, operating systems or networks.

3. **Analysis:** It contains different attacks of port scan, spam and html files penetrations.

4. **Backdoors:** Allow attackers to establish a connection and gain command and control [C&C] with their target network while evading detection. In fact, research reveals that many of the backdoors used in targeted attacks have been especially designed with the ability to bypass and kind of intrusion detection system(IDS).

5. **DoS:** The denial of service attack is an attack meant to shut down a machine or network, making it inaccessible to its intended users. DoS attacks accomplish this by flooding the target with traffic, or sending it information that triggers a crash. In both instances, the DoS attack deprive legitimate users (i.e. employees, members, or account holders) of the service or resource they expected.

6. **Exploits:** Include any type of attacks that take advantages of the existing vulnerabilities at your operating system or your piece of software to cause harms such as unauthorized access, stealing information ... etc.

7. **Generic:** A technique works against all block- ciphers (with a given block and key size), without consideration about the structure of the block-cipher.

8. **Reconnaissance:** A type of attack in which an intruder engages with your system to gather information about vulnerabilities

9. **Shellcode:** Is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called *shellcode* because it typically starts a command shell from which the attacker can control the compromised machine.

10. **Worms:** A computer worm is a type of malware that spreads copies of itself from computer to computer. A worm can replicate itself without any human interaction, and it does not need to attach itself to a software program in order to cause damage.

4.1.4 Dataset Packets distribution

The total number of records(packets) is two million and 540,044. A partition from this dataset is configured as a training-set and testing-set to utilize them as a benchmark in anomaly-based NIDS evaluation. The number of records is 175,341 and 82,332 records in the train and test set respectively. The distribution of normal and malicious packets or of attack categories in both train and test set can be illustrated in the following:

- **Training Set:**

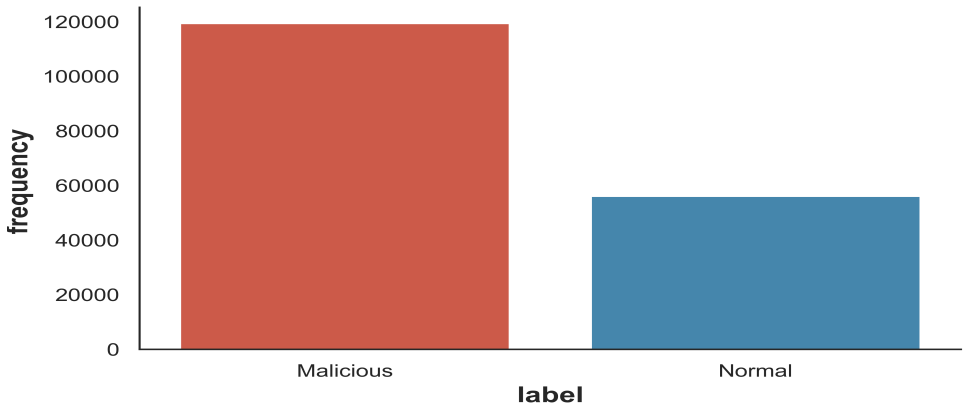


Figure 4.2: Normal and malicious packets distribution in training set

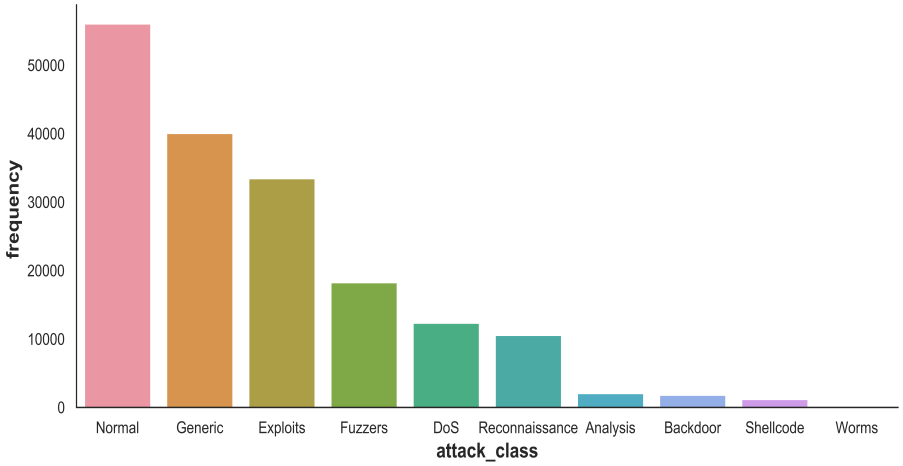


Figure 4.3: Attack categories distribution in training set

Category	**Nbr of records**	**Percentage**
binary classes distribution		
Normal	56000	31.93 %
Malicious	119341	68.07 %
multi classes distribution		
Normal	56000	31.93 %
Generic	40000	22.8 %
Exploits	33393	19 %
Fuzzers	18184	10.3 %
DoS	12264	7 %
Reconnaissance	10491	6 %
Analysis	2000	1.14 %
Backdoor	1746	1 %
Shellcode	1133	0.6 %
Worms	130	0.07 %

Table 4.7: Packet distribution in training set. The first two rows represent the number of instances in Normal and Malicious categories. The malicious category itself has 9 sub-categories that they are described in the remaining rows

– **Testing Set:**

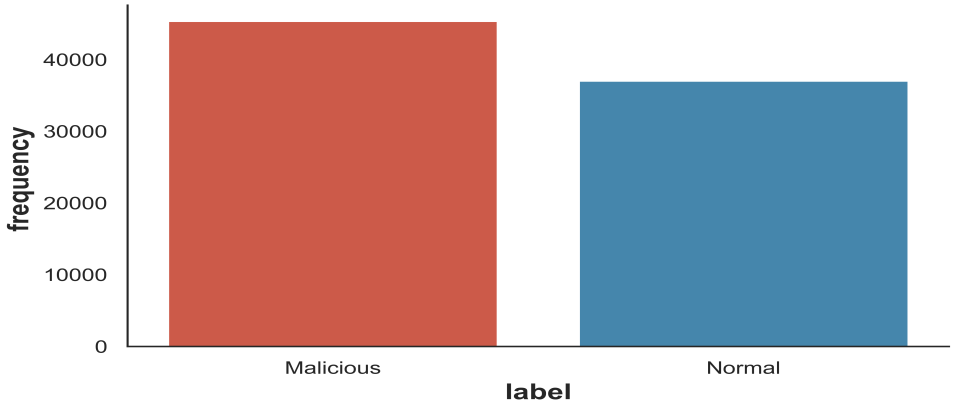


Figure 4.4: Normal and malicious packets distribution in testing set

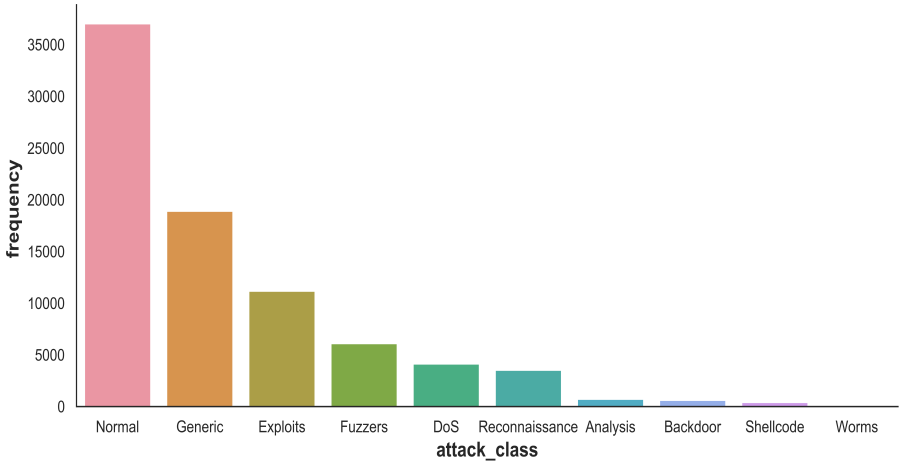


Figure 4.5: Attack categories distribution in testing set

Category	**Nbr of records**	**Percentage**
binary classes distribution		
Normal	37000	44.93%
Malicious	45332	55.07 %
multi classes distribution		
Normal	37000	44.93%
Generic	18871	22.92 %
Exploits	11132	13.52 %
Fuzzers	6062	7.36 %
DoS	4089	4.96 %
Reconnaissance	3496	4.26 %
Analysis	677	0.85 %
Backdoor	583	0.70 %
Shellcode	378	0.45 %
Worms	44	0.05 %

Table 4.8: Packet distribution in testing set. The first two rows represent the number of instances in Normal and Malicious categories. The malicious category itself has 9 sub-categories that they are described in the remaining rows

4.2 Dataset Pre-processing

Among the crucial steps in the data-driven approach(machine learning) is data pre-processing. Usually, the gathered data is not suitable or prepared to be used for a machine learning task . Hence, data pre-processing (also called data preparation) is preferable or sometimes is necessary to achieve better result. When the following proprieties are present in our data, we have to consider data pre-processing before doing a machine learning task such as:

- **Missing values:** occurs when no data value is stored for the feature in a record.
- **Redundant Records:** means when a given records are duplicated in the dataset.
- **Nominal features:** also know as categorical features. Typically, it means any feature which is categorical in nature represents discrete values which belong to a specific finite set of categories or classes(i.e. a no numerical data).
- **Non similar scale features:** which means that we have to make sure that all features take the same ranges of values. For instance the values' range of the feature x_1 is $[0 - 1000]$ and the feature x_2 takes on values between $[0 - 5]$. In this case we consider to normalize the data to make the features scale at the same range.

There are other proprieties to consider in data pre-processing phase. USNW-NB15 network dataset does not suffer from either missing values or redundant records. However, it has some nominal feature and furthermore its features are not at the same values range. To solve the past two problems we have used two techniques called *One-Hot Encoding* and *Feature scaling (also called data normalization)* respectively.

1. **One-Hot Encoding:** is a technique used to convert a categorical data into a numerical. The problem with categorical data is that some machine learning algorithms cannot work with them such as logistic regression, support vector machine and artificial neural network... etc. They only understand numerical or continuous data. One-hot encoding transforms each category into a feature where the all new created features can be perceived as a vector of binary values. The record that has a such category, a 1 value is placed into the feature that represents the such category and the rest are 0.
   Example: Let's assume a dataset that has a categorical feature *color*.

year	**color**	**nbr of seats**
2014	red	5
2000	green	5
2012	black	7

Table 4.9: Before Encoding

After applying one-hot encoding transformation:

year	nbr of seats	red	green	black
2014	5	1	0	0
2000	5	0	1	0
2012	7	0	0	1

Table 4.10: After Encoding

As mentioned above, USNW-NB15 has 4 categorical features (protocol, service, state and attack-category) that they have been encoded using One-Hot Encoding.

2. **Feature Scaling:** also knows as data normalization or Standardization. It helps to normalize the data features within a particular range often between 0 and 1 or -1 and 1. In gradient descent and its alternatives such as stochastic and mini-bath gradient descent, feature scaling improve the convergence speed of the algorithm [31]. For instance in support vector machine (SVM) algorithm, it can reduce the time to find support vectors [32]. Here is a brief overview of feature scaling techniques:
Note: let x_i and $\hat{x_i}$ denote the original and normalized feature value respectively.

   - **Standard scaler:** it assumes that your data is normally distributed within each feature, and it'll scale them as the following :

$$\hat{x_i} = \frac{x_i - mean(x_i)}{stdev(x_i)}$$

   - **Min-max scaler:** this technique performs a linear transformation on the original data. the normalized feature is given by:

$$\hat{x_i} = \frac{x_i - min(x_i)}{max(x_i) - min(x_i)}$$

     The advantage of *Min-Max* normalization is that it preserves the relationships among the original data values [33].

   - **Robust scaler:** it used a similar method to min-max scaler but it instead used the interquartile range, rather than the min-max, so that it is robust to outliers.

$$\hat{x_i} = \frac{x_i - Q_1(x_i)}{Q_3(x_i) - Q_1(x_i)}$$

   - **Z-score normalization:** a very common technique to normalize the features to zero mean and unit variance is Z- score normalization [34]. It is a linear technique in which, initially, mean $\bar{x}$ and

standard deviation σ of the specific feature values are computed using:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

The normalized feature is then given by:

$$\hat{x_i} = \frac{x_i - \bar{x}}{\sigma}$$

The issue is which one to choose to normalize our data. The best practice is to pick it empirically (i.e. the technique that gives better result it is the suitable to the nature of our data). Since that Min-Max technique preserves the relationships among the original data values [33], it has been used to normalize USNW-NB15 dataset features.

Chapter 5

Experiments and Results

5.1 Dataset used and pre-processing

USWN-NB15 is the dataset used to evaluate the performance of machine learning approaches in which it has been discussed in details in the previous chapter (chapter 4). At the pre-processing phase, the dataset features have been encoded using One-Hot-Encoding to transform categorical features into numerical. As well the dataset features have been scaled using Min-Max technique.

5.2 Experimental setup

5.2.1 Pre-processing

Data pre-processing has been done using the following libraries:

- **Pandas** [35] offers a better data manipulation and visualization as well used to perform some analysis.
- **NumPy** [36] (numerical python), offering a set of mathematical and numerical features , was used a long side pandas for data manipulation.
- **Scikit-learn** [37] also known as Sklearn. It provides the *sklearn.preprocessing* package that offers common utility functions, it has been used fo feature scaling and data encoding.

5.2.2 Machine learning algorithms implementation

The machine learning algorithms used to evaluate the performance of anomaly-based NIDS over the UNSW-NB15 dataset in this thesis such as *Decision Tree, Random Forest and Logisitic Regression* have been implemented using the framework **Scikit-learn 0.21.2** [37].

5.2.3 Neural network implementation

There is two ways to implement a neural network, either by programming it from scratch or using a pre-implemented libraries like *Keras* or *TensorFlow*. Writing a neural network from scratch is time-consuming and requires a lot of expertise in the domain. Hence, we chose to use the deep learning framework **Keras** [38]. Compared to TensorFlow, Keras is a high level API and it is more user-friendly. Keras is an open-source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive, Toolkit, Theano, or PlaiML.

5.2.4 Technical details

CPU	Intel Core i5 2.6GHz
Memory	8Gb 1600MHz DDR3
Operating system	Mac OS X, version = 10.14.4
Programming language	Python, version = 3.7.0
Libraries and frameworks	NumPy, version = 1.15.3 pandas, version = 0.23.4 sklearn, version = 0.20.3 Keras, version = 2.2.4 with TensorFlow 1.13.1

5.3 Model evaluation

The process of evaluating a ML model or classifier has been discussed previously in details. In this section, we are going to pass from the general meaning of the evaluation metrics into our specific case such as packet classification (either a normal or a malicious packet). Malicious packets are considered the positive class, in the other hand, negative class represents the normal traffic. Before to explain the evaluation metrics, it is necessary to explain the components of the confusion matrix.

5.3.1 Confusion matrix components

– **True Positives (TP)** the number of correctly classified malicious packets.

– **True Negatives (TN)** the number of correctly classified normal packets.

– **False Positives (FP)** the number of normal packets incorrectly classified as malicious.

– **False Negatives (FN)** the number of malicious packets incorrectly classified as normal.

5.3.2 Evaluation metrics

Here are the used evaluation metrics :

- **Accuracy:** is the percentage of correctly classified packet either is a malicious or a normal packet. In other words, it's the ration of the number of correctly classified packets to the total classified packets.

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

- **Precision:** defines how many malicious packet is correctly classified among the all classified as malicious. For instance, our model has classified 50 packets as malicious and 35 are actually malicious, so, the precision is 35 divided by 50 in which is 70%.

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

- **Recall:** means how many malicious packets have been classified correctly among the all malicious classified packets. For example, 60 malicious packets have been passed to the classifier, then it classified 40 packets as malicious. the recall is 40 divided by 60 in which is 66%.

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

- **False Positive Rate:** is the percentage of misclassified normal packets among all that they are actually normal packets.

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

- **False Negative Rate:** is the percentage of misclassified malicious packets among all that they are actually malicious packets.

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

The importance of evaluation metrics vary from problem to another. In critical systems like network intrusion detection, the effect or danger of misclassifying normal packets is not the same as misclassifying malicious packet (i.e. do not detecting a malicious packet is going to cause harm to the network's computers, in contrary, classifying as normal packet as malicious is going only to trigger a false alarm). Hence, the first metrics to look are **recall** and **precision**. Because of that the accuracy is commonly used in the classification problems, it is used as metric even it's not a good criterion of model's performance. Also, FPR and FNR have been used to determine the percentage of erroneously classified packets in that class.

5.4 Results and Discussions

This section shows the results obtained from training ML models on USNW-NB15 train set and evaluating them using USNW-NB15 test set in both cases binary and multi-class classification. The evaluation results are structured into two groups: machine learning based approach such as *Decision Tree, Random Forest and Logisitic Regression* and deep learning based approach *Artificial Neural Network*.

5.4.1 Machine learning based approach

Binary classification

metric / classifier	accuracy	precision	recall	FPR	FNR
LR	80.60 %	74.94%	97.20 %	39.80%	2.67 %
decision tree	86.20%	82.43%	95.23%	24.87%	4.76%
random forest	87.73%	84.09%	96.84%	22.71%	3.84%

Table 5.1: ML based approach results using binary classification

Multi-class classification

metric / classifier	accuracy
LR	69.15%
decision tree	73.30%
random forest	74.40%

Table 5.2: ML based approach results using multi-class classification

The three classifiers have been implemented with the default input hyper-parameters expect for random forest in which the hyper-parameter *n-estimators* is set to 4 and 12 in binary and multi-class classification respectively. The tables 5.1 and 5.2 illustrate the obtained results of the three used techniques by using different evaluation metrics in binary classification whereas using only the classification accuracy in multi-class because of that the other metrics such as precision, recall, FPR and FNR cannot be calculated when there is more than two classes. In other words, the concepts of TP,TN, FP and FN break down when there are multiple possibilities. The Random Forest classifier accomplish the highest score in both binary and multi class classification.

metric / classifier	accuracy	FAR	accuracy	FAR
	Moustafa et al. [11] result		our result	
LR	83.15%	18.48%	80.60%	21.23%
decision tree	85.56 %	15.78%	86.20%	14.81 %

Table 5.3: ML based algorithm results comparison

The author of the USNW-NB15 dataset contributed along with the dataset, with 2 ML based techniques among the techniques that we have used such as Decision Tree and Logisitic Regression with only binary classification. As shown in the table 5.3, Moustafa et al. [11] logisitic regression classifier has performed better whereas our decision tree classifier shows better result. Note: False Alarm Rate (FAR) = $\frac{FPR+FNR}{2}$

5.4.2 Deep learning based approach

After using grid search technique in order to find the best network's setting such as the hyper-parameters, we end up with hyper-parameter set as shown in the tables 5.4 and 5.6 for binary and multi-class classification respectively.

Binary classification

Hyper-parameters	**Setting**
Hidden layers	1
Neurons	64
Activation function in the hidden layer	ReLU
Activation function in the output layer	Sigmoid
Optimizer	RMSprop
Learning rate	0.001
Batch size	140
Epochs	100
Lost Function	binary-CrossEntropy
Regularizer	activity-regularizer l2=0.00001
kernel-initializer	Variance scaling

Table 5.4: Network configuration in binary classification

metric / classifier	accuracy	precision	recall	FPR	FNR
ANN	88.80 %	85%	96.75%	20.92%	3.24 %

Table 5.5: Binary classification DL based approach results

Multi-class classification

Hyper-parameters	Setting
Hidden layers	1
Neurons	64
Activation function in the hidden layer	ReLU
Activation function in the output layer	Softmax
Optimizer	RMSprop
Learning rate	0.001
Batch size	100
Epochs	90
Lost Function	sparse-categorical-crossentropy
Regularizer	None
kernel-initializer	Variance scaling

Table 5.6: Network configuration in multi-class classification

metric / classifier	accuracy
ANN	77.48%

Table 5.7: Multi-class classification DL based approach results

	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Normal	Reconnaissance	Shellcode	Worms
Analysis	2	0	9	658	4	0	4	0	0	0
Backdoor	2	5	12	536	9	1	3	14	1	0
DoS	3	32	148	3640	104	25	39	76	22	0
Exploits	20	48	251	10152	340	8	144	138	28	3
Fuzzers	4	3	18	1675	2616	0	1363	303	80	0
Generic	0	20	41	477	180	18107	11	27	8	0
Normal	219	0	15	807	5185	6	30096	629	42	1
Reconnaissance	0	2	9	803	40	5	67	2570	0	0
Shellcode	0	0	0	90	21	0	22	145	100	0
Worms	0	0	0	35	3	0	2	1	2	1

Table 5.8: Multi-class Confusion Matrix of ANN

The performance of the artificial neural network (i.e. multi layer perceptron) surpasses all ML based approaches with the selected hyper-parameters in both cases binary and multi-class classification as shown in the table 5.5 and 5.7 respectively. The performance results of ANN are provided with the confusion-matrix (see Figure 5.8) to the lack of other evaluation metrics in multi-class apart from classification accuracy.

metric / classifier	accuracy	FAR	accuracy	FAR
	Moustafa et al .[11] result		our result	
ANN	81.34%	21.13%	88.80%	12.08%

Table 5.9: DL based algorithm results comparison

Nour et al. [11] have also evaluated the anomaly-based NIDS' performance using ANN with 81.34% accuracy and 21.13% false alarm rate. As demonstrated in the table 5.9, our ANN model outperforms Nour et al. model with 88.80 % accuracy and 12.08% false alarm rate in binary classification.

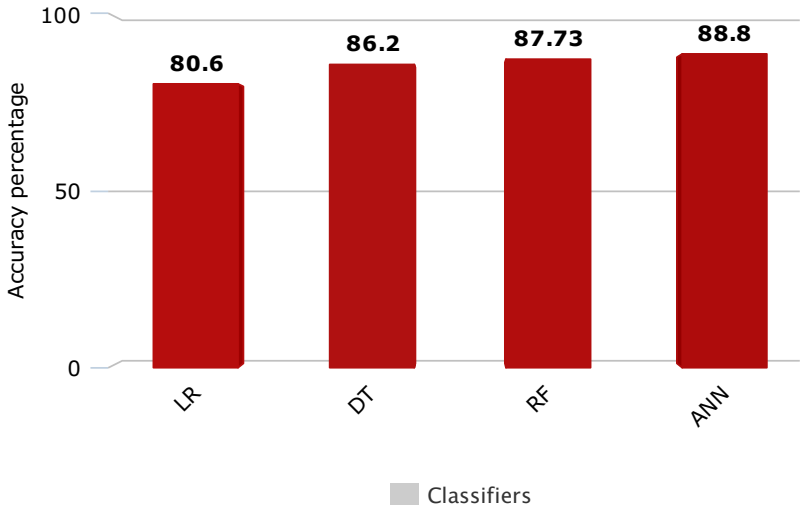


Figure 5.1: Results summary for binary classification

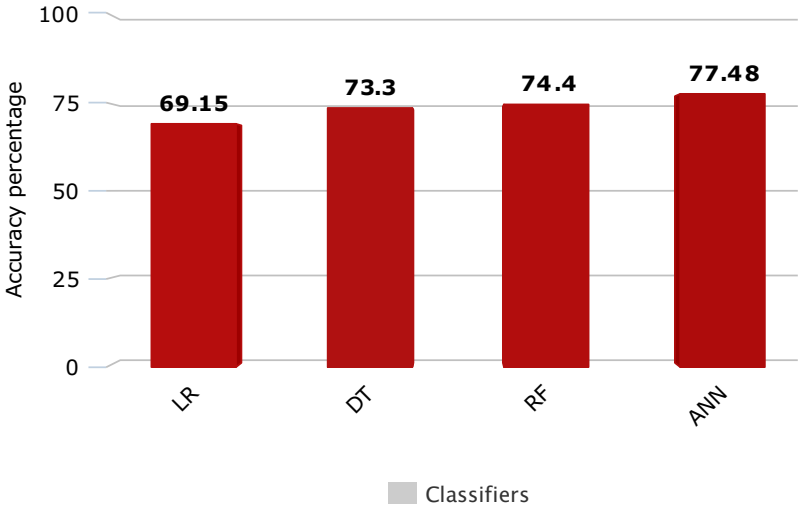


Figure 5.2: Results summary for multi-class classification

Chapter 6

Conclusion

In this thesis, we have evaluated the performance of anomaly-based NIDS by using multiple machine learning algorithms on the UNSW-NB15 dataset. The results shows that ANN performed well compared to the other methods in predicting the class of unseen packets especially in terms of accuracy, recall and false negative rate. Moreover, Our ANN classifier has shown better results than the one of Moustafa et al.[11]. Even if the USNW-NB15 dataset suffers from several issues such as imbalanced classes and the recorded malicious traffic are synthetic does not reflect real world attacks, the classifiers have given good results and they can be improved in future works.
In the literature review survey of G. Fernandes Jr. et al. [2] shows that the absence of reliable standard dataset is the most discussed issue among researchers in the literature. Hence, as long as we are concerned, building a reliable labeled dataset that does not suffer from the issues as the existing datasets is the first matter to look when addressing the anomlay-detection problem in future works.
Apart from the dataset issue, more sophisticated deep learning based methods such as Recurrent neural network and Convolutional neural network could be used to improve the performance of anomlay-based NIDS in detecting unseen abnormal behavior.

Bibliography

[1] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*. Upper Saddle River, NJ, USA: Prentice Hall Press, 5th ed., 2010.

[2] G. Fernandes, J. J. P. C. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proença, "A comprehensive survey on network anomaly detection," *Telecommunication Systems*, vol. 70, pp. 447–489, Mar 2019.

[3] C. Dartigue, H. I. Jang, and W. Zeng, "A new data-mining based approach for network intrusion detection," in *Proceedings of the 2009 Seventh Annual Communication Networks and Services Research Conference*, CNSR '09, (Washington, DC, USA), pp. 372–377, IEEE Computer Society, 2009.

[4] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th ed., 2012.

[5] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, pp. 18–28, Feb. 2009.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[7] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016.

[8] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2Nd Edition)*. New York, NY, USA: Wiley-Interscience, 2000.

[9] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, CISDA'09, (Piscataway, NJ, USA), pp. 53–58, IEEE Press, 2009.

[10] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6, Nov 2015.

[11] N. Moustafa and J. Slay, "The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set," pp. 1–14, 01 2016.

[12] Cisco, "What is network security?," 2019. Accessed: 2019-05-15.

[13] H. Bostani and M. Sheikhan, "Hybrid of anomaly-based and specification-based ids for internet of things using unsupervised opf based on mapreduce approach," *Computer Communications*, vol. 98, pp. 52–71, 12 2016.

[14] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools.," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[15] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*, IMW '02, (New York, NY, USA), pp. 71–82, ACM, 2002.

[16] A. Marnerides, A. Schaeffer-Filho, and A. Mauthe, "Traffic anomaly diagnosis in internet backbone networks," *Comput. Netw.*, vol. 73, pp. 224–243, Nov. 2014.

[17] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "Achieving human parity in conversational speech recognition," *CoRR*, vol. abs/1610.05256, 2016.

[18] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *CoRR*, vol. abs/1607.01759, 2016.

[19] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," *CoRR*, vol. abs/1605.06409, 2016.

[20] R. E. Schapire, *The Boosting Approach to Machine Learning: An Overview*, pp. 149–171. New York, NY: Springer New York, 2003.

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, p. 436, may 2015.

[22] R. H. R. Hahnloser, H. S. Seung, and J.-J. Slotine, "Permitted and Forbidden Sets in Symmetric Threshold-Linear Networks," *Neural Computation*, vol. 15, no. 3, pp. 621–638, 2003.

[23] J. Schmidhuber, "Deep learning in neural networks: An overview," *CoRR*, vol. abs/1404.7828, 2014.

[24] C. wei Hsu, C. chung Chang, and C. jen Lin, "A practical guide to support vector classification," 2010.

[25] D. Chicco, "Ten quick tips for machine learning in computational biology," *BioData Mining*, vol. 10, p. 35, Dec 2017.

[26] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[27] P. Gogoi, M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Packet and flow based network intrusion dataset," in *Contemporary Computing* (M. Parashar, D. Kaushik, O. F. Rana, R. Samtaney, Y. Yang, and A. Zomaya, eds.), (Berlin, Heidelberg), pp. 322–334, Springer Berlin Heidelberg, 2012.

[28] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, pp. 262–294, Nov. 2000.

[29] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection," in *Recent Advances in Intrusion Detection* (G. Vigna, C. Kruegel, and E. Jonsson, eds.), (Berlin, Heidelberg), pp. 220–237, Springer Berlin Heidelberg, 2003.

[30] A. R. Vasudevan, E. Harshini, and S. Selvakumar, "Ssenet-2011: A network intrusion detection system dataset and its comparison with kdd cup 99 dataset," in *2011 Second Asian Himalayas International Conference on Internet (AH-ICI)*, pp. 1–5, Nov 2011.

[31] S. Tsakalidis, V. Doumpiotis, and W. Byrne, "Discriminative linear transforms for feature normalization and speaker adaptation in hmm estimation," *IEEE Transactions on Speech and Audio Processing*, vol. 13, pp. 367–376, May 2005.

[32] L. Bo, L. Wang, and L. Jiao, "Feature scaling for kernel fisher discriminant analysis using leave-one-out cross validation," *Neural Computation*, vol. 18, pp. 961–978, April 2006.

[33] G. Manikandan, N. Sairam, S. Sharmili, and S. Venkatakrishnan, "Achieving privacy in data mining using normalization," vol. 6, pp. 4268–4272, 04 2013.

[34] T. Jayalakshmi and S. A, "Statistical normalization and back propagation for classification," *International Journal Computer Theory Engineering (IJCTE)*, vol. 3, pp. 89–93, 01 2011.

[35] W. McKinney, "pandas : powerful python data analysis toolkit," 2011.

[36] T. E. Oliphant, "Guide to NumPy," *Methods*, vol. 1, p. 378, 2010.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.

[38] F. Chollet *et al.*, "Keras," 2015.