MACHINE LEARNING ALGORITHMS FOR CYBER ATTACK DETECTION AND CLASSIFICATION

A THESIS SUBMITTED TO
THE FACULTY OF ARCHITECTURE AND ENGINEERING
OF
EPOKA UNIVERSITY

BY

ERINDI MULLALLI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

MARCH, 2024

MACHINE LEARNING ALGORITHMS FOR CYBER ATTACK DETECTION AND CLASSIFICATION

A THESIS SUBMITTED TO
THE FACULTY OF ARCHITECTURE AND ENGINEERING
OF
EPOKA UNIVERSITY

BY

ERINDI MULLALLI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

MARCH, 2024

Approval sheet of the Thesis

This is to certify that we have read this thesis entitled **"Machine Learning Algorithms for Cyber Attack Detection and Classification"** and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Assoc. Prof. Dr. Arban Uka
Head of Department
Date: March, 01, 2024

Examining Committee Members:

Assoc.Prof.Dr. Dimitrios Karras (Computer Engineering) _____

Prof.Dr. Gëzim Karapici (Computer Engineering) _____

Prof.Dr. Betim Çiço (Computer Engineering) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Surname: Erindi Mullalli

Signature: _____

ABSTRACT

MACHINE LEARNING ALGORITHMS FOR CYBER ATTACK DETECTION AND CLASSIFICATION

Mullalli, Erindi
Master of Science, Department of Computer Engineering
Supervisor: Prof.Dr. Betim Çiço

As a result of the accelerated development and expansion of technology in the present day, a new concern has emerged: cyberattacks. This has generated significant concern across various domains globally, leading to considerable disruption in networks and presenting PC users with a multitude of challenges. Presently, a multitude of organisations are striving to combat these types of cyber-attacks through the implementation of novel detection and subsequent destruction methods. The domain of machine learning enables computers to acquire knowledge and skills without requiring explicit programming. There are an abundance of implementation strategies for this technology. This study aims to demonstrate a diverse array of algorithms utilised in the defence against various cyber-attacks. This paper will examine various classification algorithms utilised to defend against diverse cyber-attacks, as well as the methods of defence against these attacks. The implementation, accuracy, and testing time of these algorithms will vary depending on the classification of the attack. This thesis will discuss various varieties of these algorithms.

Keywords: *Performance, cyber-attack, cyber-defense, machine learning, and deep learning*

ABSTRAKT

Algoritmet e mësimit të makinerisë për zbulimin dhe klasifikimin e sulmeve kibernetike

Mullalli, Erindi

Master of Science, Department of Computer Engineering

Udhëheqësi: Prof.Dr. Betim Çiço

Si rezultat i zhvillimit të përshpejtuar dhe zgjerimit të teknologjisë në ditët e sotme, është shfaqur një shqetësim i ri: sulmet kibernetike. Kjo ka krijuar shqetësim të madh në fusha të ndryshme globalisht, duke çuar në ndërprerje të konsiderueshme në rrjete dhe duke i paraqitur përdoruesit e PC me një mori sfidash. Aktualisht, një mori organizatash po përpiqen të luftojnë këto lloje të sulmeve kibernetike përmes zbatimit të metodave të reja të zbulimit dhe shkatërrimit të mëvonshëm. Fusha e mësimit të makinerive u mundëson kompjuterëve të fitojnë njohuri dhe aftësi pa kërkuar programim të qartë. Ka një bollëk strategjish zbatimi për këtë teknologji. Ky studim synon të demonstrojë një grup të larmishëm algoritmesh të përdorura në mbrojtjen kundër sulmeve të ndryshme kibernetike. Ky punim do të shqyrtojë algoritme të ndryshme klasifikimi të përdorura për t'u mbrojtur kundër sulmeve të ndryshme kibernetike, si dhe metodat e mbrojtjes kundër këtyre sulmeve. Zbatimi, saktësia dhe koha e testimit të këtyre algoritmeve do të ndryshojnë në varësi të klasifikimit të sulmit. Kjo tezë do të diskutojë varietete të ndryshme të këtyre algoritmeve.

***Fjalë kyce**: Performanca, sulmi kibernetik, mbrojtja kibernetike, mësimi i makinerive dhe mësimi i thellë*

Dedication

I dedicate this work to my family who supported me, to my friends who encouraged me for this work and supervisor who has always been ready to help me.

TABLE OF CONTENTS

ABSTRACT iii
ABSTRAKT iv
LIST OF TABLES ix
LIST OF FIGURES x
CHAPTER 1 1
INTRODUCTION 1
I.1 Purpose of this thesis 1
I.2 The thesis synopsis 1
CHAPTER II 3
REVIEW OF THE LITERATURE 3
II.1 Assaults on the system 3
II.1.1 Abuse of resources as a target of attack 3
II.1.2 User-access compromise 4
II.1.3 Root access compromise 5
II.1.4 Web access compromise 5
II.1.5 Malware attack 5
II.1.6 Denial of Service 6
II.2 Security measures to ward against cyberattacks 7
II.2.1 Intrusion Detection Systems 7
II.2.2 Protection against an assault on resource misuse 8
II.2.3 Protective measures against attacks that compromise both root and user access 9
II.2.4 Provides protection against attacks that compromise the web 9
II.2.5 Protection against malicious software 9
II.2.6 Defense against Denial of Service attacks 11
II.3 Artificial Intelligence (Learning Machine) 12
II.3.1 Decision Tree algorithm 13
II.3.2 Random Forest algorithm 14
II.3.4 Supporting Vector Machine algorithm 16
II.3.5 Gaussian Naïve-Bayes algorithm 17

II.3.6 K-means clustering algorithm 18
II.3.7 K-Nearest Neighbor (KNN) algorithm 19
II.3.8 Network of Artificial Neural Circuits 20
II.3.9 Convolutional Neural Network 21
II.3.10 Neuronal networks that are recurrent 23
CHAPTER III 25
THE METHODS AND THE MATERIALS 25
III.1 Datasets 25
III.1.1 Dataset KDD 25
III.1.2 The dataset, NSL-KDD 26
III.1.3 Kyoto dataset 28
III.1.4 UNSW-NB15 dataset 28
III.2 Machine Learning Algorithms 28
III.2.1 Gaussian Naïve Bayes Algorithm 29
III.2.2 Logistic Regression 30
III.2.3 Decision Tree Algorithm 32
III.2.4 Random Forest Algorithm 33
III.2.5 Supporting Vector Machine algorithm 34
III.2.6 Gradient Boosting algorithm 36
III.2.7 K-Nearest Neighbor Classifier 37
III.2.8 Network of Artificial Neural Circuits 38
III.2.9 Convolutional Neural Network 40
III.2.10 Recurrent Neural Network 42
CHAPTER IV 44
DISCUSSIONS BASED ON THE RESULTS 44
IV.1 KDD dataset results 44
IV.2 NSL-KDD dataset results 46
IV.3 Kyoto dataset results 49
IV.4 UNSW-NB15 52
CHAPTER V 58
SUMMARY AND SUGGESTIONS FOR FUTURE WORK 58
V.1 Summary 58
V.2 Future Work 59

REFERENCES 60
APPENDIX..... 62

LIST OF TABLES

Table 1 : THE OUTCOMES OF THE TRAINING AND TESTING METHODS FOR THE KDDCUP DATASET ……..……

Error! Bookmark not defined.

Table 2 : THE OUTCOMES OF THE TRAINING AND TESTING METHODS FOR THE NSL-KDD DATASET ……..……..50

Table 3: THE OUTCOMES OF THE TRAINING AND TESTING METHODS FOR THE KYOTO DATASET …….……..53

Table 4 : RESULT FOR TRAINING AND TESTING ALGORITHM FOR UNSW-NB15 DATASET ……..…..53

Table 5 : RESULTS FOR TRAINING TIME, TESTING TIME, TRAINING ACCURACY AND TESTING ACCURACY FOR ALL 4 DATASETS TESTED ……..…..53

Table 6 : THE RESULT FOR SENSITIVITY AND SPECIFICITY FOR EACH ALGORITHM IN THE 4 DATASETS TESTED …….…..53

LIST OF FIGURES

Figure 1. The decision Tree algorithm……... 13

Figure 2. Random Forest algorithm……. 14

Figure 3. Logistic Regression Algorithm…… 15

Figure 4. Supporting Vector Machine algorithm……. 16

Figure 5. Gaussian Naïve-Bayer algorithm……. 17

Figure 6. K-means clustering algorithm…….. 18

Figure 7. K Nearest Neighbor……..20

Figure 8. Artificial Neural Networks……... 21

Figure 9. Convolutional Neural Networks……. 21

Figure 10. Recurrent Neural Network…… 23

Figure 11. Implementation of Naïve Bayes Algorithm in Python……. 30

Figure 12. Logistic Regression Implementation in python…… 31

Figure 13. Decision Tree Classifier Implementation in Python…….. 32

Figure 14. Random Forest Classifier implementation in python…… 33

Figure 15. Linear SVM classifier implemented in python……. 34

Figure 16. Non-Linear SVM Classifier implementation in python……. 35

Figure 17. Gradient Boosting Classifier algorithm in python…… 36

Figure 18. Implementation of K-Nearest Neighbor classifier in python……. 37

Figure 19. Artificial Neural Network Classifier in Python……. 38

Figure 20. Multilevel Perception Classifier implemented in python…….. 41

Figure 21 : Convolutional Neural Network classifier using Python……... 42

Figure 22: Recurrent Neural Network implementation in python…….. 44

Figure 23: Block Scheme……..…….. 45

Figure 24: The outcomes of the training and testing methods for the KDDCUP dataset…… 48

Figure 25: The outcomes of the training and testing methods for the nsl-kdd dataset ……51

Figure 26: The outcomes of the training and testing methods for the Kyoto dataset ……54

Figure 27: The outcomes of the training and testing methods for the UNSW-NB15 dataset 56

Figure 28: Training accuracy values for all algorithms 57

Figure 29: Testing accuracy values for all algorithms 57

Figure 30: Sensitivity and Specificity of the 12 algorithms for all tested datasets ...60

CHAPTER 1

INTRODUCTION

Presently, scientists are devoting considerable effort to the development of an intelligent system that can detect various types of intrusions.

I.1 Purpose of this thesis

This is to achieve theThe objective of this endeavour is to develop an intelligent system capable of identifying various types of network anomalies and to evaluate the efficacy of each approach we implement. Machine learning techniques were selected due to their widespread recognition and acceptance as a preferred approach. They are capable of acquiring knowledge independently, without requiring explicit programming. Obtaining a more in-depth comprehension of deep learning and machine learning classification methods is an additional goal that this endeavour aims to accomplish. With the use of four different kinds of datasets, the purpose of this investigation is to determine how accurate the learning algorithms are.

I.2 The thesis synopsis

Including an introduction, a literature review, materials and methods, results and discussion, conclusions, and a section on future work are the six chapters that make up this project. The introduction section comprises a succinct delineation of the project, its intended objective, and an outline of the thesis. The literature review is structured into three sections, each of which pertains to cyber-attacks and identifies which varieties cause the most damage. The subsequent segment delineates the various forms of safeguards against cyber-criminals and cyber-attacks. The collection of machine learning algorithms that are utilised in the area of cyber security is presented in the third part of this introduction. The next section provides an analysis of the achievements of this endeavour in light of the sources that were mentioned. In the

section under "Materials and Methods," a comprehensive overview of the datasets that were utilised for the research is included, along with an explanation of the reasons behind their selection. The structure is divided into two distinct portions. In the first part of the presentation, an overview of the datasets that were used is presented. Detailed information on the results of the training time, testing time, training accuracy, and testing accuracy of the algorithms that were applied for machine learning and deep learning may be found in the section under "Result and Discussion." Even things like the Discussion and Results are broken up into four distinct parts for each dataset that is being offered. In the section, "Conclusions," you will find a summary of the results and consequences of the study, as well as some recommendations for further research that may be conducted in the future. Finally, in the section under "References," we provide hyperlinks to the many sources that we have relied on during our research.

CHAPTER II

REVIEW OF THE LITERATURE

II.1 Assaults on the system

A method that is used to compromise the data that is kept on a computer that is the subject of an assault is referred to as a cyber-attack. This may be accomplished by unauthorised access, theft, acquisition, alteration, or disabling. There is another way to describe a cyber-attack, which is an incursion that targets the system of a computer with the purpose of compromising the integrity, availability, and confidentiality of the data. In addition to that, it is also known as the CIA trinity. The protection of data from being accessed by those who are not permitted to do so is an example of secrecy. When a person who is not permitted to do so acquires your credit card information or password, this is a breach of confidence. Integrity refers to the guarantee that data cannot be altered by anyone who are not permitted to do so. The provision of permitted users with access to the data whenever it is necessary is what we mean by "availability." The deletion of each and every file on your computer is an example of a circumstance in which the availability is lost. It is possible to launch a broad range of cyberattacks.

II.1.1 Abuse of resources as a target of attack

Employees of an organisation may, on occasion, inadvertently allow access to information belonging to the institution to persons who are not permitted to have access to it. A Man-in-the-Middle attack, often known as a MitM assault, is an example of an attack that takes advantage of resource mismanagement. An instance of this kind of intrusion takes place when a criminal places oneself in the middle of a client and a server's connection that requires reliability. In this kind of attack, the cybercriminal makes changes to the communication that takes place between the server and the corresponding client. Both the server and the client are unaware of the fact that a third party is acting as an intermediary between them and is aware of all of the communications that are being sent back and forth between them. When it comes to

ModM attacks, session hijacking is one kind. During this kind of attack, a cybercriminal inserts themselves into the middle of a session that is taking place between the server and the client (the client in this case being the victim). After this, the cybercriminal will change the Internet Protocol (IP) address of the client to a different one of his choice. As a consequence of this, he resumes the connection with the server, which, ignorant to the circumstances, identifies the cybercriminal's IP address as that of a trusted client. After this, the device used by the cybercriminal creates a link with the computer belonging to the client, who is the victim, and then forges the sequence number and internal data of the client's machine.

II.1.2 User-access compromise

Presently, a prevalent form of attack involves the compromise of personally identifiable information (Passwords, Credit Card Numbers, and so forth). Personal information can be compromised through a variety of means, including social engineering, surveillance, brute force, dictionary, phishing, spearfishing, and so forth. Sniffing is a technique by which a cybercriminal can intercept data during its wireless transmission from one personal computer to another. Numerous gratis software programmes, including Wireshark, aid cybercriminals in conducting surveillance. Typically, this type of assault is employed in public areas with wireless access, such as cafeterias. This type of attack enables the cybercriminal to observe the content that is submitted on the website as well as the requests that are returned. Social engineering is a form of attack in which a cybercriminal manipulates a victim psychologically into divulging sensitive information. There exist numerous methods by which this may be accomplished. In class, for instance, when the instructor inquired as to how each student had entered the Facebook password, they began to demonstrate it individually. A brute force attack occurs when a cybercriminal attempts to deduce the password of an account by trying each possible combination of characters until they succeed. The dictionary assault is a more sophisticated form of intrusion than the brute force attack. Similar to the brute force attack, the cybercriminal endeavours to discover the passwords by trying every possible combination of characters. However, in this instance, he possesses a list consisting of the most frequently used passwords, including '1234' and others. Phishing is a form of cyberattack in which a target is duped into divulging sensitive information. Phishing, also referred to as the theft of sensitive

personal information like credit card numbers and passwords, is executed by a cybercriminal.

II.1.3 Root access compromise

This form of attack bears significant resemblance to user access compromise; nevertheless, in contrast to the latter, the cybercriminal gains access to the administrator account rather than the specific host. The administrator account possesses distinct privileges that distinguish it from the majority of other accounts within the network system.

II.1.4 Web access compromise

This form of intrusion is carried out through the exploitation of vulnerabilities present on various websites. Web compromise assaults are frequently executed through the utilisation of SQL (structured query language) injection and XSS (cross-site scripting).SQL injection is a form of injection that enables cybercriminals to compromise data, disrupt operations, or cause damage to information by impersonating their identities or rendering it ineffective, among other things.

II.1.5 Malware attack

Malware (short for malicious software) refers to a type of software that is capable of causing damage to a computer system. Malware has been utilised by hackers for decades to achieve a variety of goals, including disabling or destroying cyber-systems, compromising systems or networks, stealing massive amounts of data, injecting malicious programmes, and so forth. On the basis of their propagation frequency and intended use, malware can be categorised into various categories. Among this category, the most prevalent are ransomware, spyware, viruses, Trojans, and worms.

Virus assault: Comparable to how a biological virus replicates within the human body, a computer virus can also duplicate itself. It can infect other files on your computer after cloning itself and may be attached to a software application. Numerous varieties of viruses exist, including the Elk Cloner virus and the Melissa virus.

Worms: Unlike viruses, worms can replicate without the assistance of software. Self-cloning is possible via propagation across the network. The solitary worm is capable of cloning itself via email attachments. Worms are incapable of infecting computer files. Worms are capable of executing Denial-of-Service (DoS) attacks by replicating themselves across all contacts in the victim's email and by utilising all available network resources.Trojans are fundamentally dissimilar to viruses and worms in essence. Trojan-launching cybercriminals typically employ social engineering techniques to convince their targets to install the Trojan on their own systems. A Trojan does not possess the capability to replicate or infect the files present on a computer. Its sole purpose is to provide cybercriminals with a gateway through which they can execute malware whenever they deem it necessary.

Spyware is a form of malicious software designed to monitor the activities of targets instead of initiating an actual attack. Without the cognizance of the victim, this type of malware steals sensitive information from them, including passwords, credit card numbers, and logon credentials.

Ransomware is a form of malicious software that obstructs a collection of applications on a system with the intention of extracting a ransom, which is typically monetary. Typically, such assaults are executed with the assistance of a Trojan. Ransomware is illustrated by the name Wannacry.

II.1.6 Denial of Service

The primary objective of this category of cyber-attack is to disrupt the typical functioning state of a system or network. Distributed Denial of Service attacks, network-based attacks, and host-based attacks are the three primary classifications of denial of service attacks.

Host-based assault: This category of attacks involves the installation of viruses and malware within computer systems with the purpose of carrying out their payload or operation, which is to inundate the entire network system with an infinite number of host requests.

Network-based attack: In contrast to the aforementioned form of attack, which requires a specific computer system as its target, cybercriminals infiltrate the entire

network with the intention of executing their payload and subsequently disrupting the network's regular operations.

A Distributed Denial of Service (DDoS) attack is typically executed by a computer system or network with the intention of entirely deactivating the victim's network.

II.2 Security measures to ward against cyberattacks

As a result of the existence of several defensive mechanisms, the system is protected, either totally or partly, from the aforementioned types of attack. As an alternative, these measures are often referred to as Intrusion Detection Systems, or IDS for short.

II.2.1 Intrusion Detection Systems

An intrusion prevention mechanism and an intrusion detection mechanism are both components of the intrusion detection system. In order to identify and regulate network activities that are taking place inside the network, an intrusion detection system, also known as an IDS, is developed by using a mix of hardware and software components. There are two separate categories that make up the Intrusion Detection System. These categories are determined by the detection technique and the aim. detection-based and data source-based classification techniques are the two types of classification methods that are employed by an intrusion detection system. A misuse-based detection and an anomaly-based detection are the two unique subgroups that fall under the category of detection-based methods. The source-based techniques and the network-based methods are the two subcategories that fall under the umbrella of the data source approach.

Technique that is based on detection: Other names for signature-based detection are misuse detection and signature-based detection. The preservation of

recognisable attack behaviours, such as database signatures, is the overriding objective of this strategy. In addition to being extraordinarily rapid, the abuse approach generates an extremely low amount of false alerts. This technique, on the other hand, has a significant false alarm rate when it comes to situations in which there are no attacks or when there are attacks that have not been discovered.

Method that is depending on the source of the data: Without much difficulty, the host-based technique is able to identify intrusions that originate from a particular machine. In addition to being able to accurately identify the behaviour of network objects like programmes, ports, and files, this approach also has the power to do so. On the other hand, the host-based strategy is dependent on host resources, which in this case are computers, and as a result, it is unable to recognise instances of network abnormalities or assaults. When compared to host-based techniques, network-based methods are able to function independently of the resources provided by the host, such as computers. By and large, routers and switches are the devices that are used to implement network-based approaches. This system is not reliant on any particular operating system and is able to differentiate between many types of network protocols. One of the limitations is that its use is limited to monitoring the flow of data inside a particular network and not beyond it.

II.2.2 Protection against an assault on resource misuse

A network intrusion detection system based on anomalies is required to prevent attacks involving the exploitation of resources. A system that falls into this category is able to monitor network flows and will raise the alarm in the event that an attempt is made to take control of a network session. This specific kind of system will produce a high number of false alerts in the event that zero-day threats are present, despite the fact that it has shown remarkable performance against known network assaults. It is advised that businesses use preventative measures, such as a Virtual Private Network (VPN), while accessing resources inside their network in order to maintain network security against zero-day attacks. This is done in order to protect the network from being compromised.

II.2.3 Protective measures against attacks that compromise both root and user access

The term "phishing" refers to a situation in which both root and user access have been effectively compromised. Furthermore, this indicates that the prevention and protection against spoofing attacks may be used to ensure that both root and user access to a particular system are protected. Utilising a strategy that is based on email as a defence mechanism against phishing attempts is one option that can be used to ensure its security.

II.2.4 Provides protection against attacks that compromise the web

SQL injection and cross-site scripting attacks are two methods that cybercriminals use to launch a web access compromise attack against a particular website address. These attacks are carried out when this vulnerability is discovered. Anomaly detection and signature-based detection are the two separate detection approaches that are used in order to protect the system from assaults that are aimed at compromising the web. In addition to this, it is necessary to establish a secure coding practice, keep an up-to-date knowledge of vulnerabilities inside the database, and apply updates for programmes that are capable of preventing vulnerabilities of this kind.

II.2.5 Protection against malicious software

The current worldwide epidemic that has affected the whole digital environment is malware, which is a kind of malicious software. Cybercriminals use this strategy to their advantage in order to secure computer systems and get access to sensitive information. Malware detection methods are an essential part of the defensive system since they serve as the first line of defence against assaults that are carried out by malicious software. There are three distinct categories that make up the detection mechanism. These categories are based on the method in which the malware detection process will be carried out.

Signature-based: This technique is used rather often in the detection of malicious software. Companies that specialise in anti-malware techniques do malware analysis and then proceed to produce signatures, which are made up of a string of bytes. A pattern-matching algorithm is implemented, and signatures are used to assure

the safety of their customers. This is accomplished via the use of signatures. The most significant disadvantage of this technique, on the other hand, is that it is possible for malicious actors to modify a section of code and the programme that came before it in order to avoid detection by signature-based systems. It is also not possible to use this strategy to defend against zero-day attacks.

The notion of behavior-based malware detection is strongly tied with signature-based detection; however, it utilises a different process for extracting characteristics. Signature-based detection is closely related to behavior-based malware detection. By using a detection method that examines the behaviours of the virus rather than its vocal communication, this approach is able to identify malicious software. Identification of malware that is capable of obfuscation and aberrant malware may be accomplished via the use of behavior-based malware detection, which is an appropriate approach. Malware that displays characteristics that are similar to one another is categorised under a single signature rather than establishing separate signatures for each individual byte code sequence. A considerable reduction in the number of false alarms generated by behavior-based approaches is achieved as a result of this. The behavior-based detection approach is distinguished by the presence of three separate components. The first component, which is identified as the data collector, is used for the purpose of collecting data that is associated with executable element information. In order to transform the data that has been acquired, the alternative component acts as a medium that acts as an intermediate. The last step in the process of generating the output involves comparing the representations to the database that contains the behaviour signature.

Although the behavior-based detection strategy is far more effective than the signature-based approach, cybercriminals are nevertheless able to defeat this method by using tough countermeasures. This is the case despite the fact that the signature-based approach is significantly more effective. Researchers in the modern day use a heuristic strategy, which is a combination of machine learning and data mining methods, in order to solve this difficulty.

II.2.6 Defense against Denial of Service attacks

Defending a system from DoS (Denial of Service) attacks constitutes an extensive area of study. Defending against a DoS attack requires the implementation of two distinct strategies: attack prevention and detection.

Preventing attacks: This technique is predominantly implemented in networking routers to identify malicious traffic based on signatures. An assault prevention method is also the initial line of defence in the event of a DoS attack. The following are some techniques utilised to filter packets:

Ingress and egress filtering (a) The filtration packet grants access to internal network traffic contingent upon the equivalence of ingress (traffic entering the local network) and egress (traffic leaving the local network) traffic with the expected traffic from the originating IP.

b. Router-based packet filtering: This form of packet filtering operates based on routing information pertaining to the source and destination IP addresses of the incoming packets.

c. Packet-filtering by Hop Count: The hop count refers to the discrepancy between the initial value and the Time to Live (TTL) value of a packet in network traffic. Through this process, a network router generates a database table that contains the hop count of each user in relation to a particular destination. As a result, should the router detect an irregularity in the expected step counts, it will discard the packet and generate an alert to safeguard the network from potential threats or attacks.

This category of detection mechanisms employed to thwart DoS attacks is comprised of the following two groups:

a. Detection based on signatures: Malicious traffic is identified using this method by analysing the signatures of attack traffic data.

b. DoS detection based on anomalies: This method is widely employed today for DoS detection due to the fact that attack patterns are significantly more complex than they were previously. In general, machine learning techniques are applied to this method.

This approach consists of two fundamental components. Initially, network characteristics such as Time to Live (TTL), IP packet length, and so forth, are extracted from network traffic data using Data Mining (DM) techniques. Subsequently, a detection model is built upon this feature representation. Second, incoming traffic is evaluated by this model to determine whether or not it is malevolent, based on the value of a predetermined threshold.

II.3 Artificial Intelligence (Learning Machine)

Machine learning is an umbrella term that is used to represent computational techniques that try to duplicate the learning processes of people via the use of computers in order to gain information automatically. These approaches are used in order to acquire knowledge. It includes a wide range of fields, some of which are computer science, statistics, psychology, and neurology, amongst others. It is a very large area of research. Significant progress has been made in the way that learning algorithms are implemented in the current day. This is the result of recent developments in the performance of processors and the storage of expansive amounts of data. Supervised learning, unsupervised learning, and reinforcement learning are the three separate categories that machine learning algorithms fall into. These categories are dependent on the learning strategies that they apply. During the training process of supervised learning algorithms, models are trained to the degree that they are mapped to the actual output labels. This allows the models to understand the connection that exists between the labels and the feature value that corresponds to them. Supporting Vector Machines, Logistic Regression, Random Forest, Decision Tree, and K-Nearest Neighbours are all examples of supervised learning algorithms. Neural networks, such as Convolutional Neural Networks, Recurrent Neural Networks, and Artificial Neural Networks (which also include Multilevel Perception), are also included in this category of neural networks. Unsupervised learning algorithms, on the other hand, are able to gain information from the complete training dataset without being aware of the outcome for individual inputs. Data that does not include any labels is used to train algorithms that are used for unsupervised learning.

The K-means clustering technique is only one of the many examples of an unsupervised learning algorithm that is now available. The purpose of a reinforcement learning (RL) algorithm is to learn from the environment in which it deploys an agent. This is the aim of the algorithm. The agent is able to gain knowledge by the activities that it does within the environment, and it uses this information to decide whether it will make a mistake or achieve success. A combination of supervised and unsupervised learning strategies is what makes up the algorithms that make up reinforcement learning.

II.3.1 Decision Tree algorithm

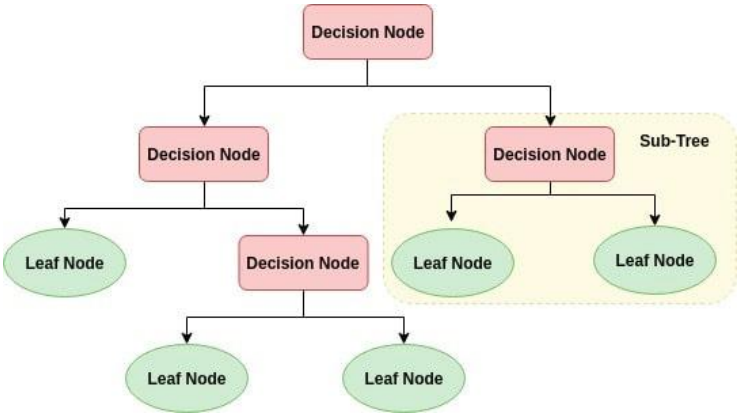


Fig 1. The decision Tree algorithm

The Decision Tree algorithm is a classification model based on rules. It is represented by a tree structure in which each feature is represented by a vertex, and the feature value is determined by each branch. The vertex positioned at the apex of the tree is referred to as the root. The component in question retains the largest proportion of the information gain (entropy differences) among all the features and is utilised to divide the training data appropriately. The term "leaves" refers to the vertices located at the bottom of the algorithm. The class is represented by a leaf. Throughout the classification process, the decision tree transitions to a top-down methodology to satisfy the classification instance. The following equation describes the information gain utilised in a decision tree to precisely divide samples in a tree-structured method:The Decision Tree algorithm is a classification model based on rules. It is represented by a tree structure in which each feature is represented by a vertex, and the feature value is determined by each branch. The

vertex positioned at the apex of the tree is referred to as the root. The component in question retains the largest proportion of the information gain (entropy differences) among all the features and is utilised to divide the training data appropriately. The term "leaves" refers to the vertices located at the bottom of the algorithm. The class is represented by a leaf. Throughout the classification process, the decision tree transitions to a top-down methodology to satisfy the classification instance. The following equation describes the information gain utilised in a decision tree to precisely divide samples in a tree-structured method:

$$Gain(P, Q) = Entropy(P) - \sum_{v \in D_Q} \frac{P_v}{P} Entropy(P_v) \qquad Eq\ 1$$

When applied to this scenario, Gain(P,Q) represents the entropy reduction that was applied in order to sort P according to feature Q. In a strategy that works from the top down, nodes are defined by characteristics that have an information gain value that is always growing. The uncomplicated implementation of the decision tree method and the high classification accuracy it offers are the two key advantages of using this technique. The complexity of the decision tree classifier's computations is, on the other hand, the most significant disadvantage related to it.

II.3.2 Random Forest algorithm

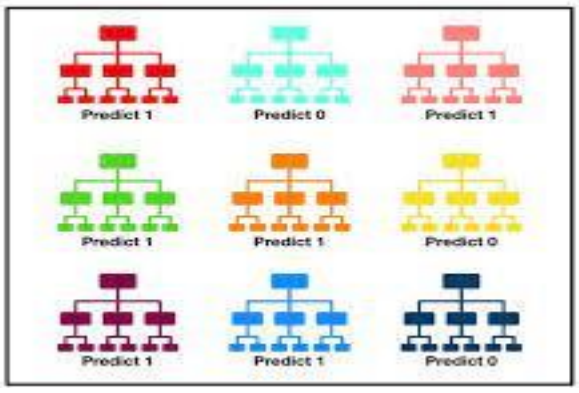


Fig 2. Random Forest algorithm

The algorithm known as random forest is made up of a collection of many different decision trees coming together. Every single tree that is still a part of the Random Forest has the ability to produce a prediction class. In reality, decision trees

are able to select the prediction of the model based on the class that obtains the most votes. It is possible to use an algorithm of this kind for both classification and regression purposes. Through the use of this strategy, an extra element of unpredictability is added throughout the process of tree development. As opposed to searching for the best attribute at the moment of splitting a node, it seeks for the best attribute from a random selection of attributes. Because of this, the random forest creates a substantially wider variety of trees, which allows for a higher-level bias to be accommodated in exchange for a lower-level variance. As a consequence, the random forest often results in a significantly better model.

II.3.3 The Algorithm behind Logistic Regression

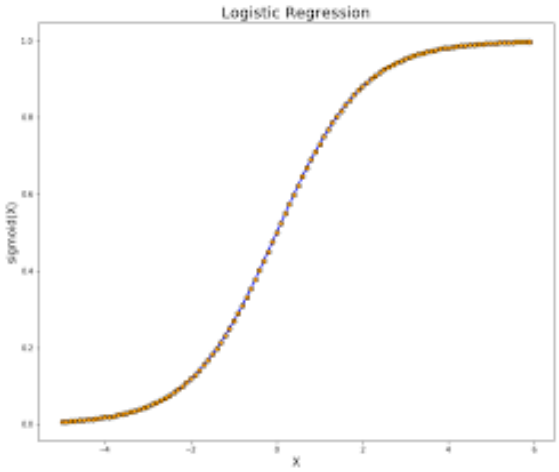


Fig 3. The Algorithm behind Logistic Regression

It is standard practice to use the Logistic Regression technique in order to ascertain the probability that a certain sample belongs to a particular category (for instance, to ascertain whether or not this file contains a pathogen). If the estimated probability is more than fifty percent, then our training model will make the prediction that the provided sample is a member of the class that has been defined (the sample that belongs to the positive class will be labelled with the value 1). On the other hand, if the estimated probability is less than fifty percent, the training model will make a prediction that the sample that was supplied does not belong to the class that was specified (which is referred to as the "negative class") and will assign the value zero

to the sample. As a result of these calculations, Logistic Regression may be used as a technique for binary classification. The output of a logistic regression method is not the actual result itself, despite the fact that it is feasible for the algorithm to calculate a weighted sum of a set of input characteristics (along with a bias). In this particular scenario, the output is the logistic of the result. With a range of values ranging from 0 to 1, the Logistic function, which is represented by the symbol σ(.), is a sigmoid function.One way to characterise the Logistic function is as follows:

$$\sigma(t) = \frac{1}{1+e^{-t}} \quad or \quad Y = \frac{1}{1+e^{-x}} \qquad Eq\ 2$$

Upon receiving an input of x (or t), the function proceeds to return an output of Y (or σ(t)).

II.3.4 Supporting Vector Machine algorithm

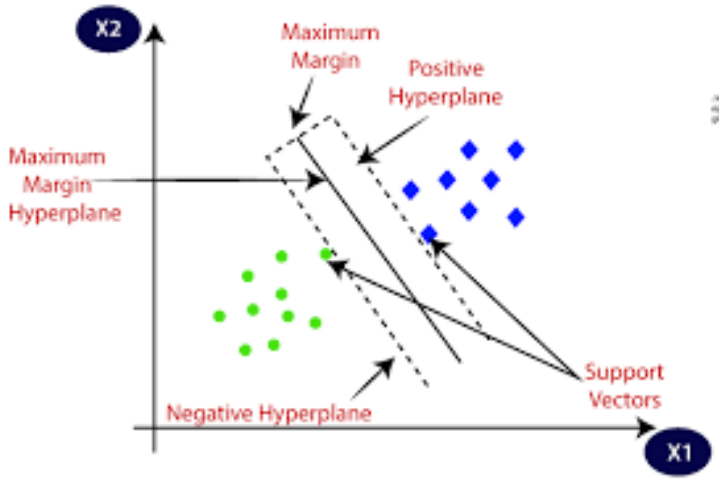


Fig 4. Supporting Vector Machine algorithm

Among the support vector machine algorithms that are used in the field of cyber security, Supporting Vector Machine is among the favourite methods. In order to divide the different classes, this technique makes use of a hyperplane, which is one of the significant characteristics of this approach. The technique makes use of a hyperplane, the degree of which is maximised in order to maximise the gap that exists between it and the data point that is closest to it. The method may be used in both two-dimensional and three-dimensional planes to get the desired results. The objective of the Supporting Vector Machine is to accurately classify the data. A few of the benefits

of the Supporting Vector Machine include its simplicity of implementation, its demonstrated exceptionally high accuracy rate, and its capability to generate hyperplanes with time complexity. One drawback associated with this approach is the challenge in determining the most effective kernel size. This algorithm finds applicability in a wide range of domains, including medicine, security applications, pattern recognition, and more.

II.3.5 Gaussian Naïve-Bayes algorithm

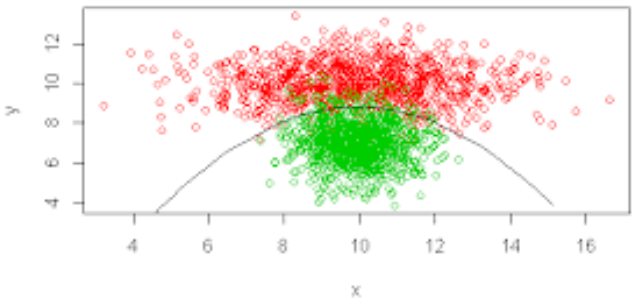


Fig 5. Gaussian Naïve-Bayer algorithm

The Naïve-Bayes classifier is an additional probabilistic type of supervised algorithm. The algorithm calculates the probability of a given class given inputs for all attributes. The model for this algorithm is constructed using the Bayes rule. The alternative name for the Naïve-Bayes algorithm is the generative model. The Naïve-Bayes classifier calculates the conditional probability of each attribute given in a class p (a/b) using the initial probability of all classes, p (b), in order to determine the probability of a class p (b/a). Generalisation of the Naive-Bayes algorithm formula:

$$p\left(\frac{b}{a}\right) = \frac{p(a,b)}{p(b)} = \frac{p(a/b)p(a)}{p(b)} \qquad Eq\ 3$$

"b" represents the class vector, whereas "a" represents the input vector. The primary benefit of the Naive-Bayes classifier is its robustness when confronted with chaotic training data. Due to its reliance on probabilistic values for all attributes, the performance of a Naive-Bayes classifier remains unaffected by low-level training

samples. The principal drawback of the Naive-Bayes classifier is that it treats all attributes as independent, despite the fact that this is rarely the case in practice.

II.3.6 K-means clustering algorithm

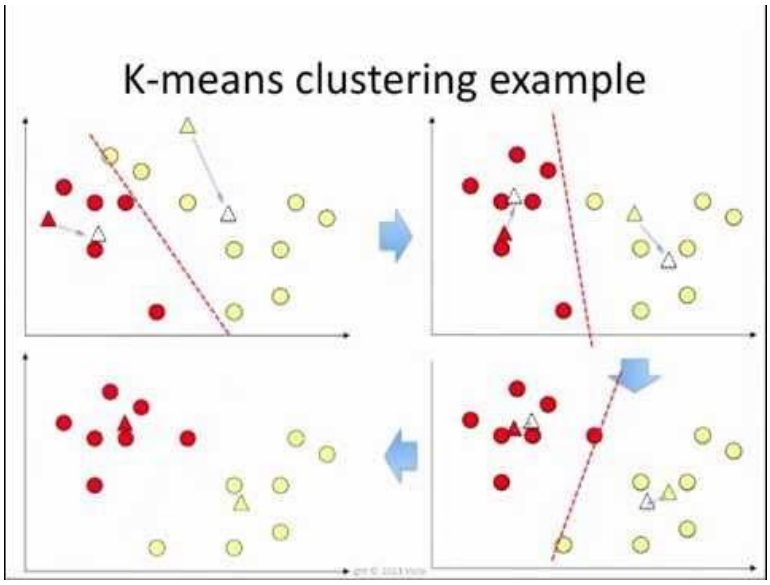


Figure 6. K-means clustering algorithm

The objective of the well-known unsupervised machine learning algorithm K-means clustering is to identify predefined clusters within a given dataset; the value of each cluster group is denoted by k. Clusters are generated on the basis of shared characteristics among all the data points within the given set. Consider an illustration of k-means clustering. Assigned to its nearest centroids are a number of m data points in accordance with Euclidean distance measures. The Euclidian distance equation is:

$$distance = \sum_{i=1}^{m} d(x_i, centroids(x_i)) \qquad Eq\ 4$$

where centoids (x_i) represents the centroid to which the data point x_i is assigned. Subsequently, the centroids are recalculated using the average distance between each data point that was assigned to the centroids. These steps are iterated throughout the algorithm until no data point can modify the cluster centroids. By performing these operations, the distance between each centroid and the corresponding data points within a cluster is diminished. These algorithms are utilised to identify data patterns and data clusters within the context of big data, where data labelling becomes a laborious task. One drawback of k-means clustering is that the k value must be

specified in advance. For the computation of attribute similarity, k-means clustering is utilised in security applications.

II.3.7 K-Nearest Neighbor (KNN) algorithm

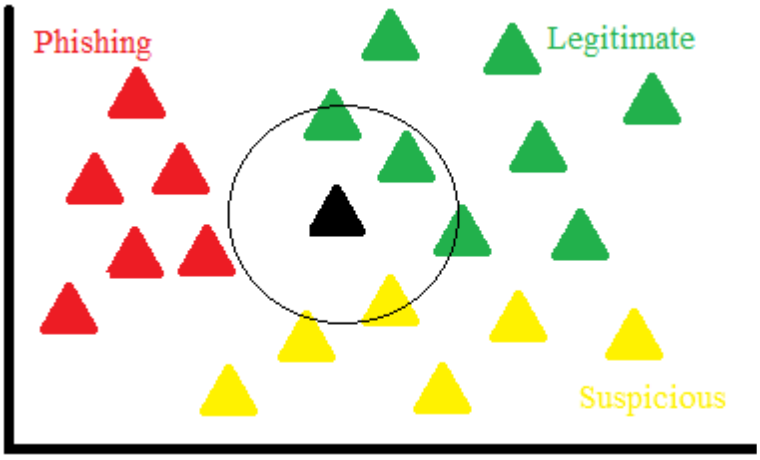


Fig 7. K Nearest Neighbor

The K-Nearest Neighbour algorithm, often known as KNN and shortened as KNN, is a well-known algorithm that is recognised for its simplicity throughout the execution of a programme. The method displays a great amount of usefulness whether it is used to either classification or regression problems. The supervised algorithms that are the most well-known to the general public. This technology is now being employed in a wide variety of technical fields. The scale of the dataset as well as the classification or regression issue that is being addressed both have a role in determining the k-value that is incorporated inside the K-Nearest Neighbour algorithm. In situations when continuous variables are being considered, the Equation of Euclidean Distance is often used. When calculating the distance between the data to be tested (x) and the data to be trained (k), the Euclidean distance is utilised. This allows for the determination of the elements that are located in the closest vicinity to one another. The following equation represents the Euclidean distance in a dimensional space with k dimensions, and it is based on the two characteristics x=[x_1,x_2,—…,x_k] and y=[y_1,y_2,… y_k].

$$D(x,y) = \sqrt{\sum_{i=1}^{k}(y_i - x_i)^2} \qquad \text{Eq 5}$$

Once the complete accumulation of KNN data is complete, the KNN majority will be utilised as a class for the data that will be evaluated.

II.3.8 Network of Artificial Neural Circuits

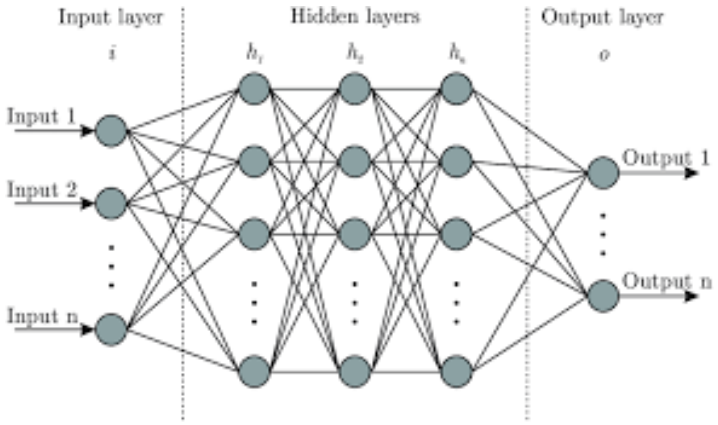


Fig 8. Network of Artificial Neural Circuits

Nodes are the building blocks of Artificial Neural Networks (ANN), which are susceptible to the impact of neurons found in the natural brain. It is necessary for an Artificial Neural Network (ANN) to have a minimum of three layers, which are the input, the hidden, and the output layers. The architecture of its network makes it possible to determine whether or not it has more than one disguised layer on the inside. It is the buried layer that receives the output of the input layer, which is then sent to the output of the layer that comes after it, and so on and so forth. Throughout the learning process within artificial neural networks, inputs (x_1, x_2, x_3,..., x_(n-2)), x_(n-1), x_n) are provided with an output label denoted by the value y. The label assigns a weight to the information assimilated by the input, which is represented by a weight vector (w_1, w_2,..., w_(n-2), w_(n-1), w_n). During the entirety of the learning process, the weights underwent modifications that effectively mitigated the learning error. The formula for calculating error is as follows:

$$E = \sum_{i=1}^{n} |d_i - y_i| \qquad Eq\ 6$$

The variable "d_i" represents the desired output, "y_i" denotes the current output, and "E" signifies the discrepancy between the two; this variance represents the error. The modification is achieved through the utilisation of back-propagation, a gradient algorithm wherein the learning process iteratively practices backwards and forwards until the model achieves an error value below a predetermined threshold. The weighted vector is adjusted in accordance with the following equation:

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j} \qquad Eq\ 7$$

where "i" represents the input node and "j" represents the concealed node.

II.3.9 Convolutional Neural Network

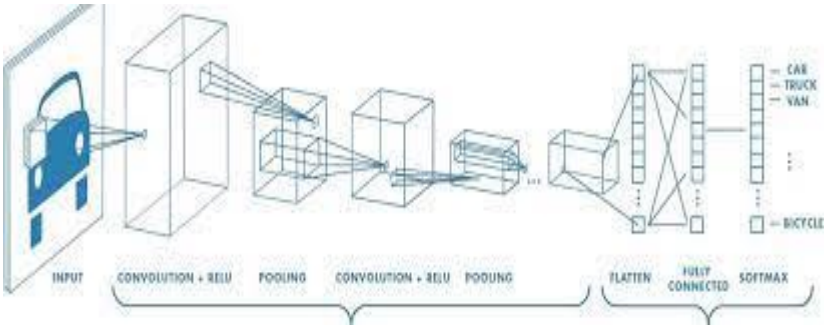


Fig 9. Convolutional Neural Networks

An additional category of algorithms utilised in deep learning is the Convolutional Neural Network (CNN). Generally, it is utilised to manage enormous training datasets through the abstraction and representation of attributes in a hierarchical fashion. The performance of conventional machine learning algorithms is negatively impacted when confronted with a very large dataset or when the data is dimensional. To tackle this challenge, the implementation of Deep Learning is supported by graphic processing units (GPUs) for the purpose of processing extensive datasets. Among all deep learning algorithms, convolutional neural networks are utilised in cyber security applications to a significant degree. There are two main layers that make up the convolutional neural network. These levels are the convolutional layer and the pooling layer. The convolutional layer is responsible for making use of many kernels of the same size in order to execute the convolution of the input data. In the event that the desired attribute is available in the input data, the convolution

operation will retrieve it by assigning a high value to a random place. On the other hand, if the desired attribute is not there, the operation will return it. This is the equation that is used to determine the value that is anticipated:

$$h = \sum_{k=1}^{m} \sum_{l=1}^{m} w_{k,l} x_{i+k-1,j+l-1} \qquad Eq\ 8$$

The input is denoted by x, the convolution kernel is denoted by w, and the output of the convolution is determined by h. The succeeding layer, which is referred to as the pooling layer, is used to lower the size of features by using two separate pooling strategies: maximum pooling and average pooling. All of these techniques are distinct from one another. In contrast to the method of average pooling, which determines the average value of the characteristics, the method of max-pooling chooses the value that is the highest. Additionally, convolutional neural networks make use of an activation layer that is known as a rectified linear unit. This activation layer combines perceptions by using a recognised activation function:

$$f(x) = \max(0, a) \qquad Eq\ 9$$

The expense is one disadvantage of the convolutional neural network. The time required to implement this algorithm is an additional drawback, which is attributable to the number of layers.

II.3.10 Neuronal networks that are recurrent

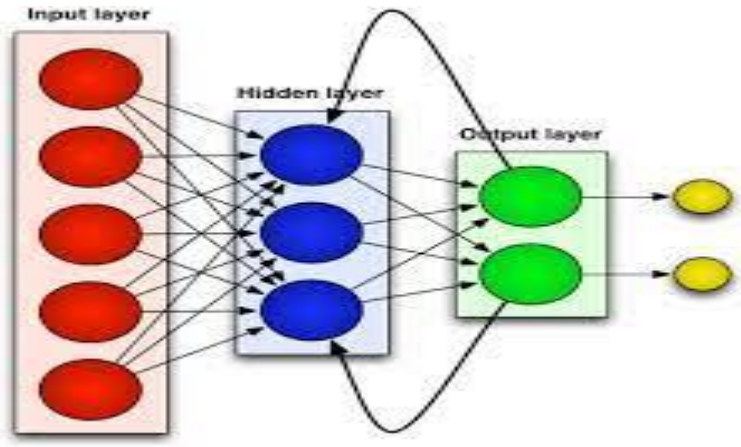


Fig 10. Neuronal networks that are recurrent

The performance of typical machine learning algorithms is unacceptable in a wide variety of applications, particularly those in which the output of the present state is reliant on the output of the past states of a variety of states. This problem occurs as a result of the lack of dependency that exists between the input and output in algorithms of this kind. In comparison to all other algorithms, the Recurrent Neural Network algorithm, which is an extra algorithm for Deep Learning, accomplishes the management of various sequential data kinds with an outstanding level of effectiveness. There are at least three levels that make up the Recurrent Neural Network. These layers include the hidden layer, the output layer, and the input layer. Recurrent neural networks are only capable of transferring data in a single way, from the input layer to the buried layer inside the network. This one-way data flow is mixed with data from a layer that came before it that was successively disguised, and then it is added to the layers that are now hidden. Each and every piece of information is stored inside the hidden layers of the recurrent neural network architecture. The vector sequence of the hidden layer, denoted as h=(h_1,h_2,h_3…h_(N-1),h_N), is computed by the Recurrent Neural Network to determine the vector of the output layer, y=(y_1,y_2,y_3…y_(T-1),y_T) [13]. from t=1 to t=T iteration of the given equations

$$h_t = H(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \qquad Eq\ 10$$

$$y_t = W_{hy}h_t + b_y \quad Eq\ 11$$

The symbols W (weight matrix), b (bias vector), and H (hidden layer) denote individual components of the model.

CHAPTER III

THE METHODS AND THE MATERIALS

III.1 Datasets

The kdd cup dataset, the nsl-kdd dataset, the Kyoto dataset, and the UNSW-NB15 dataset were all employed in this experiment.

III.1.1 Dataset KDD

There are roughly 4.9 million single-connection vectors that are included in the KDD Cup dataset. Each vector is comprised of 41 properties that may be categorised as either normal or harmful. There are four unique types of assaults that are included in the KDD Cup dataset. These are as follows: Cancellation of service (revocation)As a result of this kind of attack, the memory of the device becomes overloaded and occupied, which prevents it from responding to the request when it is received. It is the most effective protection against this kind of assault to turn off the device while it is being attacked. Attack from the User to the Root()In this kind of attack, a cybercriminal who has privileged access to a device makes an effort to get access to the router by taking advantage of weaknesses in the system. There are various approaches to accomplish this, including phishing attacks, sniffing (also referred to as packet controlling), social engineering, remote to local attacks (in which a cybercriminal without access to the device delivers packets from a computer device to a network system and exploits system vulnerabilities to gain access to the device), and probe attacks (in which a cybercriminal without access to the device delivers packets from the device to the network system and exploits the system vulnerabilities to gain access to the computer device). Fundamental attributes, traffic attributes, and content attributes are the three categories that are used to classify the characteristics that are included in the kdd collection of information. The category of essential attributes covers all features that are capable of being deleted from an IP/TCP connection. All of these characteristics contribute to a complete delay in detection, and a large part of

them do so. The category of traffic characteristics is comprised of attributes that are calculated with regard to a window interval that has been set. Additionally, it is subdivided into two subcategories, which are referred to as "same host attributes" and "same service attributes." Connections that have had the same destination host as the current connection for the previous two seconds are examined using the same host characteristics as the current connection they are examining. The capacity to compute statistics that are related with the behaviour of the protocol is another feature that this characteristic has. Exactly the same service qualities are being used by both the connection that is now being investigated and the connection that was examined just two seconds ago. Other names for these two subcategories of traffic characteristics are time-based and time-based characteristics. On the other hand, there are a vast number of slow-moving probing attacks that are able to scan ports (or hosts) by making use of time intervals that are longer than two seconds; for instance, one of these assaults could manifest itself once per minute. As a consequence of this, the intrusion patterns that such attacks attempt to establish within a time span of two seconds are not successful. As a result, in order to resolve this problem, the characteristics of the same host and service are recalculated, but this time they are based on the connection window of one hundred connections. Different names for this are connection-based traffic characteristics and connection-based traffic attributes. There are many other types of intrusions, such as User to Root Attacks and Remote to Local Attacks; yet, none of these types of intrusions display regular sequential patterns. The data component of the transmission is often where these specific types of attacks are inserted, and they only have a single link between them. The possession of a number of characteristics that are able to examine the data section for suspicious activity is very necessary in order to identify these types of assaults. There is another term for this, and that is content attributes.

III.1.2 The dataset, NSL-KDD

Both the KDD dataset and the NSL-KDD dataset are comparable to one another. Each of the four separate subcategories that make up the dataset are as follows: KDDTest+, KDDTest-21, KDDTrain+, and KDDTest+. These include

KDDTrain+ and KDDTest+, both of which are comprehensive datasets that include all of the components. The KDDTest-21 and KDDTtrain-21 datasets, on the other hand, individually account for just twenty percent of the whole dataset. All of the NSL-KDD datasets have a total of 43 characteristics, which are included in each dataset. Each dataset has a total of 41 characteristics, the first 41 of which are referred to as the traffic input. The remaining two attributes are referred to as Label, which indicates whether the traffic input is an attack or not, and Score, which indicates the severity of the attack traffic input. As was previously described, this dataset includes four categories that are identical to those found in the KDD dataset. These categories are as follows: denial of service attack, remote to local assault, user to root attack, and probe attack. A total of four categories have been established for the qualities that are included in the traffic problem data record. These categories are as follows: intrinsic, content, host-based, and time-based. These groups are employed by the intrusion detection system in order to successfully face the traffic that is coming in. For the purpose of transporting the core information that is available, the inherent characteristics of a packet are exploited. There are characteristics in the dataset that fall into the intrinsic category, and they range from 11 to 9. There is information that is stored in content attributes that is related to the initial segments. Rather of being communicated in a single chunk, these characteristics are actually conveyed in numerous pieces. The infrastructure of the network is able to acquire access to the cargo by making use of these data via their utilisation. There are twenty-two different qualities that fall under the content category that is included inside the document. The study of traffic input takes place within a two-second period, and time-based characteristics include this analysis. These characteristics include information such as the number of connections that the programme tried to make with the same host. There are many other properties. Quantities and rates make up the vast bulk of the many qualities that fall under this category. In the same way as time-based characteristics do not evaluate the traffic that occurred inside the preceding two seconds, host-based attributes do not do so either. The traffic, on the other hand, is examined over a series of links that have been formed. In order to take into consideration accessing attacks that last for more than two seconds, host-based characteristics have been developed expressly for this purpose. Within the dataset, the host-based category includes characteristics that range from 32 to 41 in various numbers.

III.1.3 Kyoto dataset

The Kyoto dataset is a collection of data that was generated at Kyoto University. It was constructed utilising actual traffic data. Fourteen attributes were extracted from the twenty-four statistical attributes comprising this type of dataset. A further ten attributes are affixed to these fourteen. The dataset exhibits a remarkable degree of precision. Through the utilisation of honeypots, web crawlers, darknet sensors, and email servers, this category of data was made accessible.

III.1.4 UNSW-NB15 dataset

In this programme, we will also utilise the UNSW-NB15 dataset. The dataset comprises a total of 42 attributes. Three of the characteristics listed contain categorical (i.e., non-numerical) values, whereas the remaining 39 attributes exclusively contain numeric values. This particular dataset was selected due to its suitability for implementation in intrusion detection systems..

III.2 Machine Learning Algorithms

During the course of this experiment, the algorithms that were used include the linear and nonlinear Supporting Vector Machine (SVM) method, the Gaussian Naïve-Bayes method, and the Logistic Regression Algorithm. Among the several neural network algorithms, the stochastic gradient descent algorithm, the random forest algorithm, the gradient boosting algorithm, the K-nearest neighbour algorithm, the convolutional neural network algorithm, and the recurrent neural network algorithm are all examples. The Multilevel Perception Algorithm, which is a subset of the Artificial Neural Network, is also implemented inside the Artificial Neural Network.

III.2.1 Gaussian Naïve Bayes Algorithm

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

clfg = GaussianNB()
start_time = time.time()
clfg.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Gaussian Naive-Bayes Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfg.predict(X_train)
end_time = time.time()
print("Gaussian Naive-Bayes Testing time: ", end_time-start_time)

print("Gaussian Naive-Bayes Train score is:", clfg.score(X_train, y_train))
print("Gaussian Naive-Bayes Test score is:", clfg.score(X_test, y_test))
```

Fig 11. Implementation of Naïve Bayes Algorithm in Python

In accordance with what was said before, the Gaussian Naïve-Bayes algorithm is built on the Bayes theorem. The operation of this algorithm is carried out in a succession of stages. Performing the calculation of the earlier probability for each of the class designations that have been supplied is an essential step before moving on. After that, we need to build a table by making use of the preceding data that we have available to us and determining the frequency of recurrence for each phenomena. Following this, the technique that follows involves determining the likelihood probability that is associated with each characteristic across all of the classes combined. Once each of them has been finished, it is important to establish each of the values that have been determined inside the Bayes Formula in order to establish the posterior probability. This is necessary in order to calculate the posterior probability. Following the completion of each of these steps, the class that has the greatest probability is determined to be the sample output. In spite of the fact that the implementation of this method is very short, it has a somewhat lower accuracy rate in comparison to the other algorithms.

III.2.2 Logistic Regression

```
from sklearn.linear_model import LogisticRegression

clfl = LogisticRegression(max_iter = 1200000)
start_time = time.time()
clfl.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Logistic Regression Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfl.predict(X_train)
end_time = time.time()
print("Logistic Regression Testing time: ", end_time-start_time)

print("Logistic Regression Train score is:", clfl.score(X_train, y_train))
print("Logistic Regression Test score is:", clfl.score(X_test, y_test))
```

Fig 12. Logistic Regression Implementation in python

Logistic regression is an extra classification procedure that is often employed in this process. This type of Python experiment employs Multinomial Logistic Regression due to the fact that there are more than two expected outcomes. The algorithm in question is a derivative form of Binary Logistic Regression. Comparable to the Binary Logistic Regression, the Multinomial Logistic Regression estimates the probability of a categorical shift using the maximal likelihood evaluation. In the context of Multinomial Logistic Regression, a meticulous examination of model sizes and consideration of enigmatic events are not required. It finds application in numerous disciplines, including medicine, statistics, and more. We are now going to examine the training process of this algorithm. The objective of the training process is to position the vector represented as θ; consequently, the model assigns a high probability to positive samples and a low probability to negative samples. The cost function of the trained sample x is:

$$c(\theta) = \begin{cases} -\log(p) & if\ y = 1 \\ -\log(1-p) & if\ y = 0 \end{cases} \qquad Eq\ 12$$

P represents the calculated probability, while y denotes the result. This type of cost function is logical due to the fact that -log∗(x) significantly increases as the input value x approaches zero. Consequently, if the model computes a probability near zero

for a positive sample, the cost will be extremely high. However, if the model computes a probability near one for a negative sample, the cost will also be extremely high. However, since -log;(x) approaches 0 when x approaches 1, the cost will be approximately zero if the calculated probability for a negative sample is close to zero and for a positive sample is close to one; this is precisely what we desire. The following describes the cost function of logistic regression:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(p^{(i)}\right) + \left(1 - y^{(i)}\right)\left(\log\left(1 - p^{(i)}\right)\right)\right] \qquad Eq\ 13$$

Where m represents the total number of samples and $\sum_{i=1}^{m}\left[y^{(i)}\log\left(p^{(i)}\right) + (1 - y^{(i)})(\log\left(1 - p^{(i)}\right))\right]$ is the total cost of the training sample. Regrettably, a closed-form equation of the kind required to compute the θ value, which is utilised to minimise the cost function, is not yet available. Nevertheless, this specific cost function has the advantage of being convex; hence, any optimisation technique should be able to ensure that it will locate the global minimum. It is possible to describe the partial derivatives of the cost function using the equation that follows:

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\sigma(\theta^T * x^{(i)}) - y^i)x_j^{(i)} \qquad Eq\ 14$$

As shown in the equation above, the algorithm computes the prediction error for each sample, multiplies it by the value of the j attribute, and then computes the mean of all training samples. It exhibits exceptional precision on both the training and testing datasets, and its implementation is remarkably rapid. I made the decision to utilise this algorithm due to its exceptional precision.

III.2.3 Decision Tree Algorithm

```
from sklearn.tree import DecisionTreeClassifier

clfd = DecisionTreeClassifier(criterion ="entropy", max_depth = 4)
start_time = time.time()
clfd.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Decision Tree Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfd.predict(X_train)
end_time = time.time()
print("Decision Tree Testing time: ", end_time-start_time)

print("Decision Tree Train score is:", clfd.score(X_train, y_train))
print("Decision Tree Test score is:", clfd.score(X_test, y_test))
```

Fig 13. Decision Tree Classifier Implementation in Python

From the perspective of cyber security, illicit activities may be traced back to a certain extent if network records are scanned. To effectively segregate legitimate activities from illicit ones, it is imperative to implement a classification-based strategy within a networking system. The Decision Tree algorithm is also among the most straightforward algorithms. It exhibits an exceptionally high degree of precision in comparison to alternative Gaussian algorithms. The fundamental principle of the Decision Tree is to partition the dataset according to the data gain of the attributes. The Python library Sickit-Learn, abbreviated sklearn, utilises the "Classification and Regression Tree" algorithm to train its decision trees. These trees are also referred to as "growing trees." The underlying principle of this algorithm is straightforward: the training algorithm initially partitions the set to be trained into two subsets using a distinct attribute k and a threshold t_k corresponding to this attribute. However, this prompts an inquiry into the methodology employed to determine the attribute k parameters and the threshold t_k. The algorithm seeks the set (k, t_k) that produces the subsets with the highest degree of clarity. The following is the equation representing the algorithm that endeavours to decrease the cost function:

$$J(k,\ t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} \qquad Eq\ 15$$

The impurity estimates for the left-wing and right-wing subsets are derived from m_left and m_right, respectively, while m denotes the total number of samples and G_left and G_right represent the sample numbers for the left-wing and right-wing subsets, respectively. As soon as this algorithm divides the set to be trained into two distinct subsets, it proceeds to divide the remaining subsets using the same logic. Subsequently, it divides the remaining subsets in the same fashion, and so forth, in a recursive fashion. The algorithm terminates the recursion when it reaches the maximum depth or when it is unable to identify a division capable of reducing impurity.

III.2.4 Random Forest Algorithm

```
from sklearn.ensemble import RandomForestClassifier

clfr = RandomForestClassifier(n_estimators = 30)
start_time = time.time()
clfr.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Random Forest Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfr.predict(X_train)
end_time = time.time()
print("Random Forest Testing time: ", end_time-start_time)

print("Random Forest Train score is:", clfr.score(X_train, y_train))
print("Random Forest Test score is:", clfr.score(X_test, y_test))
```

Fig 14. Random Forest Classifier implementation in python

The collective term for a collection of decision trees is "random forest classifier." Furthermore, its implementation typically requires only a few seconds and possesses an exceptionally high degree of precision. In this instance, thirty trees are utilised as inputs. It performs exceptionally well within the algorithms.

III.2.5 Supporting Vector Machine algorithm

```
from sklearn.svm import LinearSVC
clfls = LinearSVC(max_iter = 1000)
start_time = time.time()
clfls.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Linear SVM Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfls.predict(X_train)
end_time = time.time()
print("Linear SVM Testing time: ", end_time-start_time)
print("Linear SVM Train score is:", clfls.score(X_train, y_train))
print("Linear SVM Test score is:", clfls.score(X_test, y_test))
```

Fig 15. Linear SVM classifier implemented in python

As stated in the preceding chapter, the Supporting Vector Machine utilises hyperplanes to partition the various component classes. Linear SVM classification algorithms are distinguished from non-linear SVM classification algorithms to classify SVM algorithms. In the linear support vector machine (SVM) classification algorithm, classes are divided along a straight line. Typically, this classification algorithm is implemented on two-dimensional planes. In Linear Support Vector Machine (SVM) classification, the straight line serves the dual purpose of separating distinct classes and maintaining a safe distance from the nearest training samples, if possible. Evidently, the SVM classification algorithm suits the widest possible range, which is contained within the various classes that it has partitioned. This is also referred to as classification with a wide margin. The decision boundary remains unaffected by the inclusion of additional training samples. This boundary is determined by the samples situated along the way's perimeter. The samples situated at the boundary are alternatively referred to as the supporting vectors, which provide the nomenclature for this algorithm. Classification by hard margin entails compelling every sample to be positioned at an extreme right angle. The implementation of rigid margin classification raises two fundamental concerns. To begin with, its functionality is contingent upon the data being separable linearly, and it exhibits a high degree of sensitivity to outliers. To circumvent these concerns, it is preferable to utilise a more adaptable model. The objective of this model is to identify a delicate equilibrium between preserving the path to the greatest extent possible and limiting margin violations. The term for this is "soft margin classification." The stability of the Supporting Vector Machine Classes

of Sickit-Learn is effectively managed through the utilisation of the hypermeter C. The benefit of this approach is that Linear SVM in Sickit-Learn achieves an impressive accuracy ranging from 95% to 99%; nevertheless, its implementation requires a significant amount of time. Despite exhibiting a high efficiency rate and performing exceptionally well in many scenarios, a significant proportion of datasets do not approach linear separability. One approach to managing these types of datasets is to incorporate a substantial number of additional features, such as polynomial features.

```
from sklearn.svm import SVC

clfs = SVC(gamma = 'scale')
start_time = time.time()
clfs.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Non-Linear SVM Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfs.predict(X_train)
end_time = time.time()
print("Non-Linear SVM Testing time: ", end_time-start_time)

print("Non-Linear SVM Train score is:", clfs.score(X_train, y_train))
print("Non-Linear SVM Test score is:", clfs.score(X_test, y_test))
```

Fig 16. Non-Linear SVM Classifier implementation in python

In 3-D models, the non-linear SVM classifier is frequently implemented. In this classifier, the hyperplane is a two-dimensional plane, in contrast to the linear plane. The utilisation of SVM algorithms incorporates an exceptional mathematical technique referred to as the kernel trick. This approach enables one to achieve the same outcome as if a large number of polynomial features had been added, even if the polynomial in question has a very high degree, without the need to add each of these polynomials.An additional model that can be implemented in linear SVM is known as the stochastic gradient. This estimator is utilised to implement linear models (Linear Supporting Vector Machines in this instance) with SGD learning. SGD learning entails the computation of the loss gradient following each sample within a specified time interval, and the model is subsequently modified in accordance with a strength reduction schedule. It is an extremely potent classifier, and the tolerance level of this algorithm can be determined using it.

III.2.6 Gradient Boosting algorithm

```
from sklearn.ensemble import GradientBoostingClassifier

clfg = GradientBoostingClassifier(random_state = 0)
start_time = time.time()
clfg.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Gradient Boosting Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfg.predict(X_train)
end_time = time.time()
print("Gradient Boosting Testing time: ", end_time-start_time)

print("Gradient Boosting Train score is:", clfg.score(X_train, y_train))
print("Gradient Boosting Test score is:", clfg.score(X_test, y_test))
```

Fig 17. Gradient Boosting Classifier algorithm in python

In the realm of boosting algorithms, the Gradient Boosting algorithm is one of the more influential ones. In order to perform its duties, this category of algorithms incorporates predictors into an ensemble in a sequential manner. The aim of each subsequent predictor that is added to the ensemble is to correct the mistake that was produced by the predictor that came before it in the sequence. This technique, on the other hand, makes use of the residual errors that were produced by the previous predictor in order to modify the current predictor. Rather than altering the sample weights at each iteration, this method uses the residual errors. Due to the fact that it is successful when applied to complex datasets, this type of algorithm is acquiring more and more prominence. Not only can the method gradient boosting be used to solve classification difficulties, but it can also solve regression problems.

III.2.7 K-Nearest Neighbor Classifier

```
from sklearn.neighbors import KNeighborsClassifier
clfk= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
start_time = time.time()
clfk.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("KNN Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfk.predict(X_train)
end_time = time.time()
print("KNN Testing time: ", end_time-start_time)

print("KNN Train score is:", clfk.score(X_train,y_train))
print("KNN Test score is:", clfk.score(X_test,y_test))
```

Fig 18. Implementation of K-Nearest Neighbor classifier in python

For supervised learning, this is an algorithm that may be used. An example of how the system imports the object from the Sklearn algorithm is shown in the preceding image. When implemented as an algorithm, it takes a significant amount of time to complete. The number of neighbours is represented by the variable n_neighbors, which is included in the code. For the purpose of making predictions, this variable is included into the model. The code in question has five neighbours; due to the fact that this is such a small number, it displays an exceptionally high level of accuracy. My determination of this categorization was accomplished by the use of the Minkowski distance. In order to compute this particular kind of distance, distances must be represented as vectors inside a certain space, and each vector must have a length that is different from the others. It is recommended that the power parameter, which is represented by the tiny p, be set to 2 in this particular scenario when the Minkowski distance is employed. The capability of the K-Nearest Neighbour classifier to display a high degree of accuracy is one of the most noticeable advantages of this classification method. On the other hand, the construction of this kind of classifier is time-consuming, and it performs badly when it is presented with excessively large datasets.

III.2.8 Network of Artificial Neural Circuits

```
def ANNClassifier():
    annclassifier = Sequential()
    annclassifier.add(Dense(19, activation = 'relu', kernel_initializer = "random_uniform"))
    annclassifier.add(Dense(1, activation = 'sigmoid', kernel_initializer = "random_uniform"))
    annclassifier.add(Dense(2, activation = 'softmax'))
    annclassifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
                          metrics = ['accuracy'])
    return annclassifier

annclassifier = KerasClassifier(build_fn = ANNClassifier, epochs=100,batch_size=1000)

start_time = time.time()
annclassifier.fit(X_train, y_train)
end_time = time.time()
print("Artificial Neural Network Training time ", end_time - start_time)


start_time = time.time()
y_test_pred = annclassifier.predict(X_test)
end_time = time.time()
print("Artificial Neural Network Testing time ", end_time - start_time)

start_time = time.time()
y_train_pred = annclassifier.predict(X_train)
end_time = time.time()
print("Artificial Neural Network Training accuracy", accuracy_score(y_train, y_train_pred))
print("Artificial Neural Network Testing accuracy", accuracy_score(y_test, y_test_pred))
```

Fig 19. Network of Artificial Neural Circuits Classifier in Python

It was said in the chapter that came before this one that an artificial neural network (ANN) classifier is made up of nodes, which are also known as perceptions, and they take their inspiration from the neuronal architecture of the brain. Access to the Python train of this artificial neural network may be gained via the use of the keras package. The training of neural network algorithms is the specific aim for which this library is being used. An artificial neural network, often known as an ANN, is the algorithm that is considered to be the most fundamental of them. During the course of our experiment, I developed an artificial neural network (ANN) programme that has three layers: input, hidden, and output. Both the activation function of the layer and the number of dimensions that are included inside the layer are contained within each layer. In this particular case, the activation function of the input layer is represented by the symbol relu, which is an abbreviation for Rectified Linear Unit. In the experiment that we conducted, the activation function of the buried layer was a sigmoid function, and the activation function of the output layer was a Softmax function. A particular activation function that is referred to as the Rectified Linear Unit activation function is responsible for converting all negative values to zero while maintaining positive values. The term "Sigmoid" refers to an activation function that has values that are totally contained within the range of 0 to 1. Because of this, the activation function in question offers a great deal of benefit when it comes to probability prediction. The sigmoid function's significance in the formula is as follows:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \qquad Eq\ 16$$

The output, denoted by σ(z), is the representation of the input, which is z. σ(z) reaches its highest possible value when the value of z that is being input is comparable to infinity. Using the Softmax activation function, an input value that is included inside a values vector is normalised to a probability distribution. Based on this distribution, the total probability is predicted to be larger than 1. The values of the output continue to fall within the range of 0 to 1, notwithstanding this phenomenon. One is able to calculate the predicted losses that take place throughout the training process of a given dataset by making use of this particular sort of activation function. Equation that represents the activation function of the Softmax algorithm:

$$P\left(y = j \middle| \theta^{(i)}\right) = \frac{e^{\theta^{(i)}}}{\sum_{j=0}^{k} e^{\theta_k^{(i)}}} \qquad Eq\ 17$$

From which:

$$\theta = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_{k-1} x_{k-1} + w_k x_k \quad Eq\ 18.1$$

$$\sum_{i=0}^{k} w_i x_i = w^T x \qquad Eq\ 18.2$$

```
from sklearn.neural_network import MLPClassifier


mlp_clf = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=42)
start_time = time.time()
mlp_clf.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Multilevel Perception Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = mlp_clf.predict(X_train)
end_time = time.time()
print("Multilevel Perception Testing time: ", end_time-start_time)

print("Multilevel Perception Train score is: ", mlp_clf.score(X_train, y_train))
print("Multilevel PerceptionTest score is: ", mlp_clf.score(X_test, y_test))
```

Fig 20. Multilevel Perception Classifier implemented in python

MLP, which stands for Multilayer Perception, is another subclass of ANN. The MLP possesses the identical architecture as the Artificial Neural Network. With the exception of the output layer, every layer includes a bias neuron that is completely connected to the layer below it. Despite being a subclass of neural networks, this function is invoked via the Sklearn library as opposed to Keras. The MLP classifier in our algorithms consists of two hidden layers, whereas the initial classifier has five hidden layers and the second classifier has only two. Adam is the optimisation parameter, which is also referred to as the solver, in the programme. This optimisation parameter combines the most advantageous features of the RMSprop algorithm and the ADAgrad algorithm to produce an optimisation algorithm capable of regulating gradients that are sparse in noise-related problems. The activation function in this algorithm is also a Rectified Linear Unit, abbreviated relu. The rationale behind my selection of these algorithms is their exceptional performance.

III.2.9 Convolutional Neural Network

```
def CNNClassifier():
    cnnclassifier = Sequential()#add input layer and first hidden layer
    cnnclassifier.add(Conv2D(32, (3, 3), activation='relu', input_shape=(19, 1, 1),
                             padding='same'))
    cnnclassifier.add(LeakyReLU(alpha = 0.1))
    cnnclassifier.add(MaxPooling2D(pool_size=(4, 4), padding = 'same'))
    cnnclassifier.add(Conv2D(64, (3, 3), activation = 'relu', padding = 'same'))
    cnnclassifier.add(LeakyReLU(alpha = 0.1))
    cnnclassifier.add(MaxPooling2D(pool_size = (4,4), padding = 'same'))
    cnnclassifier.add(Flatten())
    cnnclassifier.add(LeakyReLU(alpha = 0.1))
    cnnclassifier.add(Dense(2, activation = 'softmax'))
    cnnclassifier.compile(optimizer = 'adam', loss='categorical_crossentropy',
                          metrics=['accuracy'])
    return cnnclassifier

cnnclassifier = KerasClassifier(build_fn = CNNClassifier, epochs=10, batch_size=1000)

start_time = time.time()
cnnclassifier.fit(X_train, y_train.values.ravel())
end_time = time.time()

print("Convolutional Neural Network Training time:", end_time-start_time)

start_time = time.time()
y_test_pred = cnnclassifier.predict(X_test)
end_time = time.time()
print("Convolutional Neural Network Testing time ", end_time - start_time)

start_time = time.time()
y_train_pred = cnnclassifier.predict(X_train)
end_time = time.time()

print("Convolutional Neural Network Training accuracy", accuracy_score(y_train, y_train_pred))
print("Convolutional Neural Network Testing accuracy", accuracy_score(y_test, y_test_pred))
```

Fig 21: Convolutional Neural Network classifier using Python

Convolutional Neural Network (CNN) algorithms are an additional type of deep learning technology. This algorithm is effective at classifying images. The CNN classifier in my programme consists of nine layers. Prior to commencing the CNN classification process, the dataset is resized from its original two dimensions to four. The activation function for the input layer, which is a two-dimensional convolutional layer, is identical to that of the preceding Rectified Linear Unit algorithms. In our equation, values contained within three by three dimensions are convolved. The input geometry is identical to the three reshaped dimensions that the dataset's reshaping has enabled. Padding is another term used within a convolutional network. Padding refers to the quantity of pixels appended within an image during kernel processing. It may contain two values, both of which must be legitimate. In this instance, our function remains unchanged. Consequently, this convolutional layer has the capability to utilise zero buffering if required. LeakyReLU, or Leaky Rectified Linear Unit, constitutes the second stratum. The distinction between Rectified Linear Unit and Leaky Rectified Linear Unit is that in Leaky ReLU, negative values are assigned a minor slope less than 1, as opposed to being set to zero. The experiment yielded a slope value of 0.1, despite the fact that negative inputs in this function produced minor negative values instead of zero. The following represents the Leaky ReLU function:

$$Y = 1(x < 0) * (ax) + 1(x \geq 0) * (x) \qquad Eq\ 19$$

In the context that has been presented, the output is denoted by Y, and the input is denoted by x. In the event that x is negative, the equation for Y is axe; otherwise, it is Y=x; the letter a denotes a minor constant that is always less than 0. The purpose of this Leaky ReLU is to address the problem of ReLUs that are about to expire. Subsequent to this, the Maxpooling two-dimensional layer is shown. A method known as maxpooling is an algorithm that can locate the highest value that is included inside a collection of numbers. As part of our inquiry, each of the groups has dimensions that are four by four. Even during maxpooling, the value of padding does not change, which enables the Maxpooling Layer to make use of zero padding if it is required to do so. The replication of these three layers is carried out in the same way as it was done before. The seventh tier of this method is the Flatten layer, which is the seventh

position. Its name gives the impression that its purpose is to arrange the inputs in a vertical pattern, or conversely, it may flatten the inputs according to the user's preferences. Following this layer is an extra Leaky ReLU layer that uses the same 0.1 slope as the previous layer. A softmax activation function and a numeric value that represents the input categories are the last components of the output layer. In situations when the datasets are extremely huge, CNN algorithms are able to recognise a broad range of Distributed Denial of Service assaults as well as malware attacks. My decision to choose this particular algorithm was based on this particular logic. On the other hand, the implementation of this method calls for a period of time that is quite long.

III.2.10 Recurrent Neural Network

```
def RNNClassifier():
    rnnclassifier = Sequential()
    rnnclassifier.add(LSTM(19))
    rnnclassifier.add(Dense(1, activation = 'sigmoid'))
    rnnclassifier.add(Dense(2, activation = 'softmax'))
    rnnclassifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
                          metrics = ['accuracy'])
    return rnnclassifier

rnnclassifier = KerasClassifier(build_fn = RNNClassifier, epochs=100 ,batch_size=1000)
start_time = time.time()
annclassifier.fit(X_train, y_train)
end_time = time.time()
print("Recurrent Neural Network Training time ", end_time - start_time)


start_time = time.time()
y_test_pred = annclassifier.predict(X_test)
end_time = time.time()
print("Recurrent Neural Network Testing time ", end_time - start_time)

start_time = time.time()
y_train_pred = annclassifier.predict(X_train)
end_time = time.time()

print("Recurrent Neural Network Training accuracy", accuracy_score(y_train, y_train_pred))
print("Recurrent Neural Network Testing accuracy", accuracy_score(y_test, y_test_pred))
```

Fig 22: Recurrent Neural Network implementation in python

The Recurrent Neural Network (RNN) algorithm is a kind of neural network algorithm that, in contrast to other RNN algorithms, adds feedback connections in addition to feedforward connections. This particular algorithm is used for the purpose of identifying irregularities. In the experimental configuration of our Recurrent Neural Network method, there are three separate layers as follows: an input layer for Long-Short Term Memory, a hidden layer that makes use of the sigmoid activation function, and an output layer that makes use of the softmax activation function. When it comes to optimising speed, Long Short-Term Memory (LSTM), which is capable of handling two state vectors, keeps both state vectors disconnected by default. This is done in

order to maximise efficiency. The Long Short-Term Memory (LSTM) is broken down into four parts. Analysis of the current inputs x_t and the prior (short-term) state h_(t-1) is the major purpose of the first unit, which is also the first unit. This is the only objective of the first unit. In certain contexts, the remaining three components are also referred to as gate controllers. Utilising sigmoid activation functions causes the outputs of these units to oscillate between 0 and 1, which is a consequence of the aforementioned factor. As a result, the gate will be closed if the outputs of the units are determined to be zero. On the other hand, the gate will be permitted to open when the units provide outputs of value 1, which is the expected value. The following are the components that make up the gate controllers: the forget gate is responsible for determining which long-term sections need to be removed, the input gate is in charge of determining which long-term portions need to be entered, and the output gate is in charge of determining which long-term portions need to be read and producing the output y_t in this particular time step. This particular method is not only incredibly useful but also quite easy to put into practice. The Keras library provides access to the LSTM layer for users to manipulate.

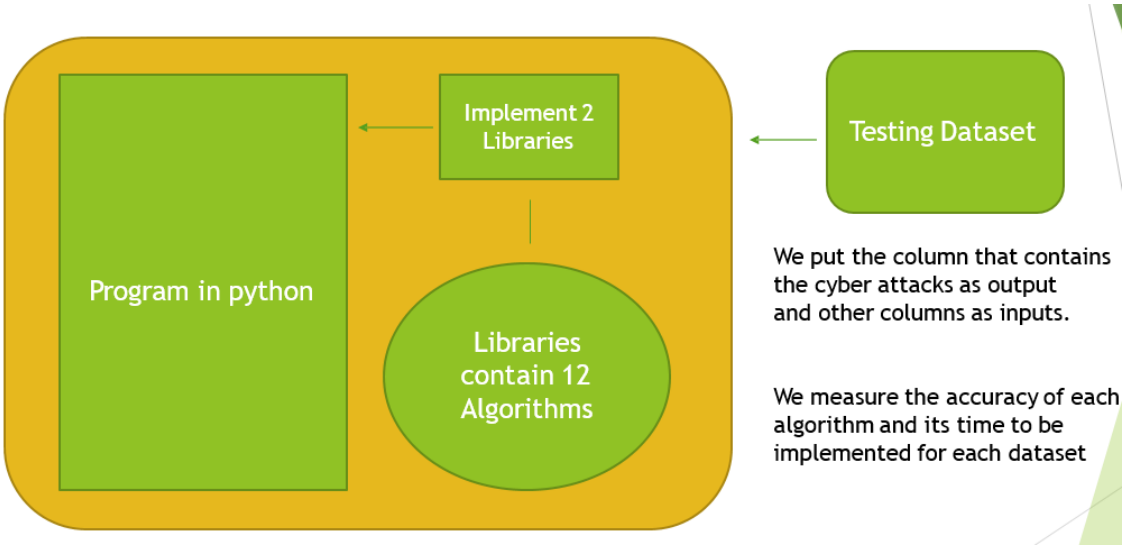


Fig 23: Block Scheme

CHAPTER IV

DISCUSSIONS BASED ON THE RESULTS

IV.1 KDD dataset results

The following are the outcomes extracted from the kddcup dataset: The training duration of Gaussian Naive Bayes is approximately 2.25 seconds. The estimated amount of time required for the Gaussian Naive-Bayes test is 3.42 seconds. The approximate training accuracy of Gaussian Naïve Bayes is 88.21%. A Gaussian Naive-Bayes test has an approximate 88.21% accuracy rate. Training a decision tree takes approximately 4.92 seconds. Testing a decision tree takes approximately 0.16 seconds. Accuracy in decision tree training is approximately 99.1%. Tree-based decision assessment The accuracy is nearly 99 percent. Training with a stochastic gradient takes approximately 15.5 seconds. The approximate testing time for stochastic gradients is 0.33 seconds. Accuracy in stochastic gradient training is approximately 99.21%. The accuracy of stochastic gradient testing is near 99.21%. The approximate training time for Random Forest is 44.64 seconds. The approximate Random Forest trial duration is 3:55 seconds. The approximate training accuracy of Random Forest is 99.99%. The accuracy of Random Forest testing is approximately 99.97%. The training duration of a Non-Linear Supporting Vector Machine classifier is approximately 2173.8 seconds. The trial duration for a Non-Linear Supporting Vector Machine classifier is approximately 226.42 seconds. The training accuracy of a Non-Linear Supporting Vector Machine classifier is approximately 99.89%. The accuracy of evaluating a non-linear support vector machine classifier is approximately 99.88%. The training duration of a Linear Supporting Vector Machine classifier is approximately 48.26 seconds. The trial duration for a Linear Supporting Vector Machine classifier is approximately 0.23 seconds. The training accuracy of a Linear Supporting Vector Machine classifier is approximately 99.71%. The accuracy of evaluating a Linear Supporting Vector Machine classifier is approximately 99.69%. Training for Logistic Regression takes approximately 40.87 seconds. Testing for Logistic Regression takes approximately 0.23 seconds. Accuracy in logistic regression

training is approximately 99.35%. The accuracy of logistic regression testing is approximately 99.31%. Training for Multilevel Perception takes approximately 498.51 seconds. Testing for multilevel perception takes approximately 0.63 seconds. The accuracy of Multilevel Perception training is approximately 99.42%. The accuracy of Multilevel Perception testing is approximately 99.38%. The training duration for a gradient boosting classifier is approximately 1433.56 seconds. The approximate trial duration for a Gradient Boosting Classifier is 7.99 seconds. Training Classifiers with Gradient Boosting Accuracy is approximately 99.96%. The testing accuracy of gradient boosting classifiers is approximately 99.93%. The training period of an artificial neural network is approximately 371.18 seconds. The assessment duration for artificial neural networks is approximately 1.02 seconds. The training accuracy of the artificial neural network is approximately 98.47%. The assessment accuracy of artificial neural networks is approximately 98.47%. The training duration of a recurrent neural network is approximately 448.51 seconds. The trial duration for recurrent neural networks is approximately 1.1 seconds. The accuracy of Recurrent Neural Network training is approximately 98.48%. The assessment accuracy of recurrent neural networks is approximately 98.48%. The training period of a convolutional neural network is approximately 870.8 seconds. The assessment duration for convolutional neural networks is approximately 21.46 seconds. The training accuracy of the convolutional neural network is approximately 99.87%. The accuracy of the Convolutional Neural Network testing is approximately 99.87%.

The outcomes of the training and testing methods for the kddcup dataset are shown in Table 1.

The Algorithms Used in Machine Learning	Time of Training	Time of Test	Training Performance	Testing Performance
Gaussian Naïve-Bayes	2.25 seconds	3.42 seconds	88.21%	88.21%
Decision Tree	4.92 seconds	0.16 seconds	99.1%	99.1%
Stochastic Gradient	15.5 seconds	0.33 seconds	99.21%	99.21%
Random Forest	44.64 seconds	3.55 seconds	99.99%	99.97%

Non-Linear SVM	2173.8 seconds	226.42 seconds	99.89%	99.89%
Linear SVM	48.26 seconds	0.23 seconds	99.71%	99.69%
Logistic Regression	40.87 seconds	0.23 seconds	99.35%	99.31%
Multilevel Perception	498.51 seconds	0.63 seconds	99.42%	99.38%
Gradient Boosting	1433.56 seconds	7.99 seconds	99.96%	99.93%
K-Nearest Neig	312.26 seconds	432.7 seconds	98.1%	94.2%
Artificial Neural Network	371.18 seconds	1.02 seconds	98.47%	98.47%
Recurrent Neural Network	448.51 seconds	1.1 seconds	98.46%	98.48%
Convolutional Neural Network	870.8 seconds	21.46 seconds	99.87%	99.87%

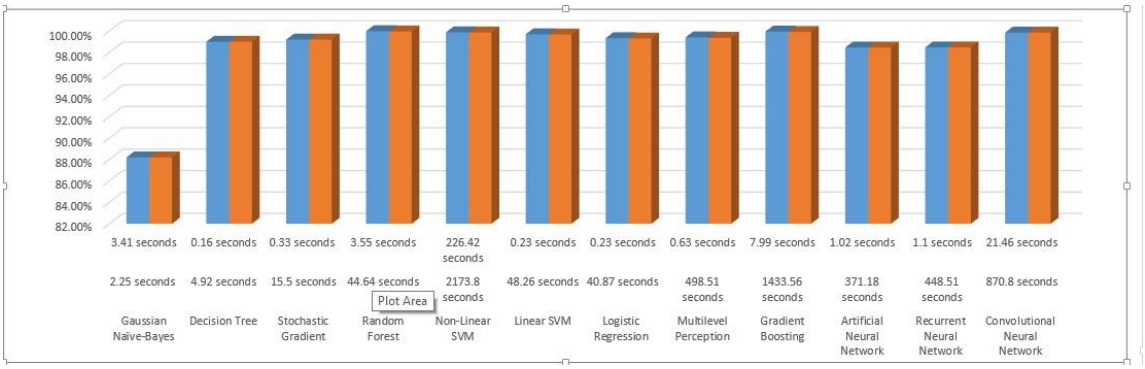


Fig 24: The outcomes of the training and testing methods for the kddcup dataset

IV.2 NSL-KDD dataset results

The approximate Gaussian Naïve-Bayes training time for the nsl kdd dataset is 0.12 seconds. The approximate Gaussian Naive-Bayes testing time is 0.12 seconds. The training accuracy of Gaussian Naive Bayes is approximately 51.28%. The approximate Gaussian Naïve-Bayes assessment accuracy is 51.56%. Training a decision tree takes approximately 0.2 seconds. Testing a decision tree takes approximately 0.016 seconds. Accuracy in decision tree training is approximately 95.8%. Tree-based decision assessment The approximate accuracy rate is 95.85%.

Training with a stochastic gradient takes approximately 0.62 seconds. The approximate trial time for stochastic gradients is 0.016 seconds. The approximate training accuracy of stochastic gradients is 97.21%. The accuracy of stochastic gradient testing is approximately 97.13%. Training with Random Forest takes approximately 1.18 seconds. The approximate Random Forest testing time is 0.154 seconds. The accuracy of Random Forest training is close to one hundred percent. The accuracy of Random Forest testing is approximately 99.64%. The training duration of a Non-Linear Supporting Vector Machine classifier is approximately 3.011 seconds. The trial duration for a Non-Linear Supporting Vector Machine classifier is approximately 2.04 seconds. The training accuracy of a Non-Linear Supporting Vector Machine classifier is approximately 99.26%. The testing accuracy of non-linear supporting vector machine classifiers is approximately 99.05%. The training duration of a Linear Supporting Vector Machine classifier is approximately 1.4 seconds. The trial duration for a Linear Supporting Vector Machine classifier is approximately 0.016 seconds. The training accuracy of a Linear Supporting Vector Machine classifier is approximately 96.97%. The testing accuracy of linear support vector machine classifiers is approximately 97.13%. Training for Logistic Regression takes approximately 1.62 seconds. Testing for Logistic Regression takes approximately 0.015 seconds. Accuracy in logistic regression training is approximately 96.74%. The accuracy of logistic regression testing is approximately 96.68%. Training for Multilevel Perception takes approximately 33.53 seconds. Testing for multilevel perception takes approximately 0.03 seconds. The accuracy of Multilevel Perception instruction is approximately 97.47%. The accuracy of multilevel perception testing is approximately 97.5%. The approximate training duration for a gradient boosting classifier is 60.55 seconds. The approximate trial duration for a Gradient Boosting Classifier is 0.4 seconds. The training accuracy of a gradient boosting classifier is approximately 99.95%. The testing accuracy of gradient boosting classifiers is approximately 99.6%. Training the K-Nearest Neighbour classifier takes approximately 3.89 seconds. The approximate K-Nearest Neighbour measurement time is 21.32 seconds. The accuracy of K-Nearest Neighbour classifier training is approximately 99.48%. Approximate K-Nearest Neighbour testing precision is 99.32%. The training period of an artificial neural network is approximately 192.79 seconds. The assessment duration for artificial neural networks is approximately 0.74

seconds. The training accuracy of the artificial neural network is approximately 98.68%. The testing accuracy of artificial neural networks is approximately 98.49%. The training duration of a recurrent neural network is approximately 234.92 seconds. The assessment duration for recurrent neural networks is approximately 0.83 seconds. The accuracy of Recurrent Neural Network training is approximately 97.82%. The assessment accuracy of recurrent neural networks is approximately 97.61%. The training period of a convolutional neural network is approximately 571.16 seconds. The assessment duration for convolutional neural networks is approximately 2.35 seconds. The training accuracy of the convolutional neural network is approximately 99.16%. The accuracy of the Convolutional Neural Network testing is approximately 98.94%.

The outcomes of the training and testing methods for the nsl-kdd dataset are shown in Table 2.

The Algorithms Used in Machine Learning	Time of Training	Time of Test	Training Performance	Testing Performance
Gaussian N-B	0.12 seconds	0.12 seconds	51.28%	51.56%
Dec Tree	0.2 seconds	0.016 seconds	95.89%	95.85%
Stoch Gradient	0.62 seconds	0.016 seconds	97.21%	97.13%
Ran Forest	1.18 seconds	0.154 seconds	100%	99.64%
Non-Linear SVM	3.011 seconds	2.04 seconds	99.26%	99.05%
Linear SVM	1.4 seconds	0.016 seconds	96.97%	97.12%
Logistic Reg	1.62 seconds	0.015 seconds	96.74%	96.68%
M-level Perception	33.53 seconds	0.03 seconds	97.47%	97.5%
Gradient Boosting	60.55 seconds	0.4 seconds	99.95%	99.6%
K-Nearest Neig	3.89 seconds	21.32 seconds	99.48%	99.32%

Artificial N.N.	192.79 seconds	0.74 seconds	98.68%	98.49%
Recurrent N.N	234.92 seconds	0.83 seconds	97.82%	97.61%
Convolutional N.N	571.16 seconds	2.35 seconds	99.16%	98.94%

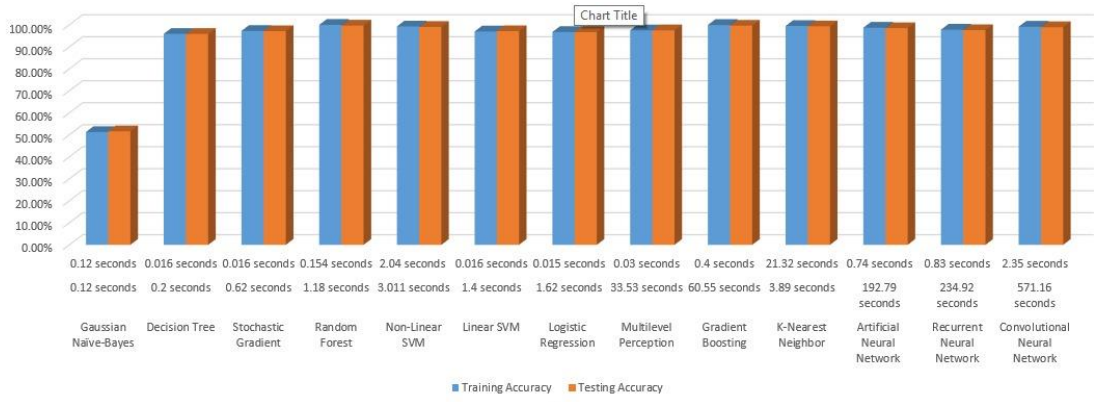


Fig 25: The outcomes of the training and testing methods for the nsl-kdd dataset

IV.3 Kyoto dataset results

The training duration of Gaussian Naive Bayes is approximately 1.4 seconds. The approximate Gaussian Naive-Bayes testing time is 0.61 seconds. The training accuracy of Gaussian Naive Bayes is approximately 71.15%. The approximate Gaussian Naïve-Bayes assessment accuracy is 71.11%. Training a decision tree takes approximately 3.14 seconds. Testing a decision tree takes approximately 0.078 seconds. Accuracy in decision tree training is approximately 98.21%. The assessment accuracy of decision trees is regarding 98.2%. Training with a stochastic gradient takes approximately 1.96 seconds. Testing with stochastic gradients takes approximately 0.2 seconds. The approximate training accuracy of stochastic gradients is 94.97%. The typical accuracy of stochastic gradient testing is 94.84%. Training with Random Forest takes approximately 38.47 seconds. Testing with Random Forest takes approximately 2.37 seconds. Random Forest has an approximate 99.95% training accuracy. The accuracy of Random Forest testing is approximately 99.75%. The training duration of a Non-Linear Supporting Vector Machine classifier is approximately 1958.71 seconds. The trial duration for a Non-Linear Supporting Vector Machine classifier is approximately 491.59 seconds. The training accuracy of a Non-Linear Supporting Vector Machine classifier is approximately 96.06%. The accuracy of evaluating a non-

linear support vector machine classifier is approximately 95.89%. The training duration of a Linear Supporting Vector Machine classifier is approximately 39.5 seconds. The trial duration for a Linear Supporting Vector Machine classifier is approximately 0.054 seconds. The training accuracy of a Linear Supporting Vector Machine classifier is approximately 94.91%. The testing accuracy of linear support vector machine classifiers is approximately 94.81%. Training for Logistic Regression takes approximately 10.89 seconds. Testing for Logistic Regression takes approximately 0.046 seconds. Accuracy in logistic regression training is approximately 94.86%. The accuracy of logistic regression testing is approximately 94.79%. Training in Multilevel Perception takes approximately 54 seconds. Testing for multilevel perception takes approximately 0.2 seconds. The accuracy of Multilevel Perception training is approximately 92.55%. The accuracy of multilevel perception testing is approximately 92.55%. The training duration for a gradient boosting classifier is approximately 173.34 seconds. The approximate trial duration for a Gradient Boosting Classifier is 1.06 seconds. The training accuracy of a gradient boosting classifier is approximately 99.42%. The testing accuracy of gradient boosting classifiers is approximately 99.36%. Training the K-Nearest Neighbour classifier takes approximately 259.34 seconds. The approximate K-Nearest Neighbour assessment time is 540.64 seconds. The accuracy of K-Nearest Neighbour classifier training is approximately 98.72%. Approximate K-Nearest Neighbour testing precision is 98.2 percent. The training duration of an artificial neural network is approximately 348.93 seconds. The assessment duration for artificial neural networks is approximately 1.69 seconds. The training accuracy of the artificial neural network is approximately 98.46%. The assessment accuracy of artificial neural networks is approximately 98.36%. The training duration of a recurrent neural network is approximately 294.88 seconds. The trial duration for recurrent neural networks is approximately 1.55 seconds. The training accuracy of the Recurrent Neural Network is approximately 98.77%. The assessment accuracy of recurrent neural networks is approximately 98.73%. The training period of a convolutional neural network is approximately 703.48 seconds. The assessment duration for convolutional neural networks is approximately 19.54 seconds. The training accuracy of the convolutional neural network is approximately 96.02%. The assessment accuracy of the convolutional neural network is approximately 96.07%.

The outcomes of the training and testing methods for the Kyoto dataset are shown in Table 3.

The Algorithms Used in Machine Learning	Time of Training	Time of Test	Training Performance	Testing Performance
Gaussian N-B	1.4 seconds	0.61 seconds	71.15%	71.11%
Dec Tree	3.14 seconds	0.078 seconds	98.21%	98.2%
Stoch Gradient	1.96 seconds	0.2 seconds	94.97%	94.84%
Ran Forest	38.47 seconds	2.37 seconds	99.95%	99.75%
Non-Linear SVM	1958.71 seconds	491.59 seconds	96.06%	95.89%
Linear SVM	39.5 seconds	0.054 seconds	94.91%	94.81%
Logistic Reg	10.89 seconds	0.046 seconds	94.86%	94.79%
M-level Perception	54 seconds	0.2 seconds	92.55%	92.55%
Gradient Boosting	173.34 seconds	1.06 seconds	99.42%	99.36%
K-Nearest Neig	259.34 seconds	540.64 seconds	98.72%	98.2 %
Artificial N.N.	348.93 seconds	1.69 seconds	98.46%	98.36%
Recurrent N.N	294.88 seconds	1.55 seconds	98.77%	98.73%
Convolutional N.N	703.48 seconds	19.54 seconds	96.02%	96.07%

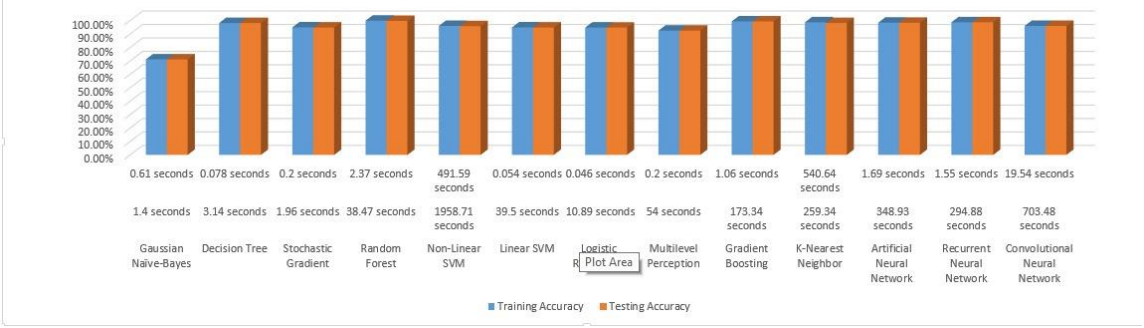


Fig 26: The outcomes of the training and testing methods for the Kyoto dataset

IV.4 UNSW-NB15

The training duration of Gaussian Naive Bayes is approximately 1.24 seconds. The approximate Gaussian Naive-Bayes testing time is 0.995 seconds. The Gaussian Naive Bayes algorithm has an accuracy of roughly 87.95% when it comes to training. Gaussian Naive-Bayes testing has an approximate accuracy of 86.05%. Training a decision tree takes approximately 2.94 seconds. Testing a decision tree takes approximately 0.1 seconds. The accuracy of decision tree training is at least 96.76%. The accuracy of decision tree testing is approximately 96.82%. Training with a stochastic gradient takes approximately 1.49 seconds. Testing with stochastic gradients takes approximately 0.14 seconds. The approximate training accuracy of stochastic gradients is 95.09%. The accuracy of stochastic gradient testing is approximately 95.13%. Training in Random Forest takes approximately 24.6 seconds. Testing with Random Forest takes approximately 1.6 seconds. Random Forest has an approximate 99.98% training accuracy. The accuracy of Random Forest testing is approximately 98.71%. The training duration for a Non-Linear Supporting Vector Machine classifier is approximately 471.77 seconds. The trial duration for a Non-Linear Supporting Vector Machine classifier is approximately 308.58 seconds. The training accuracy of a Non-Linear Supporting Vector Machine classifier is approximately 95.46%. The accuracy of evaluating a non-linear support vector machine classifier is approximately 94.24%. The training duration of a Linear Supporting Vector Machine classifier is approximately 9.47 seconds. The trial duration for a Linear Supporting Vector Machine classifier is approximately 0.116 seconds. The training accuracy of a Linear Supporting Vector Machine classifier is approximately 95.1%. The accuracy of evaluating a Linear Supporting Vector Machine classifier is approximately 93.89%. Training for Logistic Regression takes approximately 6.79 seconds. Testing for Logistic Regression takes approximately 0.072 seconds. Accuracy in logistic regression training is approximately 95.08%. The accuracy of logistic regression testing is approximately 94.23%. Training for Multilevel Perception takes approximately 656.31 seconds. Testing for multilevel perception takes approximately 0.178 seconds. The accuracy of Multilevel Perception instruction is approximately 97.37%. The accuracy of Multilevel Perception testing is approximately 95.25%. The training duration for a gradient boosting classifier is approximately 186.66 seconds. The approximate trial duration for a Gradient Boosting

Classifier is 1.06 seconds. The training accuracy of a gradient boosting classifier is approximately 98.03%. The testing accuracy of gradient boosting classifiers is approximately 98.03%. The training duration of a K-Nearest Neighbour classifier is approximately 162.62 seconds. The approximate K-Nearest Neighbour assessment time is 540.64 seconds. The accuracy of K-Nearest Neighbour classifier training is approximately 97.8%. Approximate K-Nearest Neighbour testing precision is 96.6%. The training period of an artificial neural network is approximately 206.17 seconds. The assessment period for artificial neural networks is approximately 0.51 seconds. The training accuracy of the artificial neural network is approximately 96.54%. The assessment accuracy of artificial neural networks is approximately 95.77%. The training duration of a recurrent neural network is approximately 294.88 seconds. The trial duration for recurrent neural networks is approximately 1.55 seconds. The accuracy of Recurrent Neural Network training is approximately 96.73%. The assessment accuracy of recurrent neural networks is approximately 96.78%. The training period of a convolutional neural network is approximately 703.48 seconds. The assessment duration for convolutional neural networks is approximately 19.54 seconds. The training accuracy of the convolutional neural network is approximately 95.5%. The assessment accuracy of the convolutional neural network is approximately 95.58%.

Table 4: Result for training and testing algorithm for UNSW-NB15 dataset

Machine Learning Algorithms	Training Time Seconds (s)	Testing Time Seconds (s)	Training Accuracy	Testing Accuracy
Gaussian N-B	1.24 seconds	0.995 seconds	87.95%	86.05%
Dec Tree	2.94 seconds	0.1 seconds	96.76%	96.82%
Stoch Gradient	1.49 seconds	0.14 seconds	95.09%	95.13%
Ran Forest	24.6 seconds	1.6 seconds	99.98%	98.71%
Non-Linear SVM	471.77 seconds	308.58 seconds	95.46%	94.24%
Linear SVM	9.47 seconds	0.116 seconds	95.1%	93.89%
Logistic Reg	6.79 seconds	0.072 seconds	95.08%	94.23%
M-level Perception	656.31 seconds	0.178 seconds	97.38%	95.25%

Gradient Boosting	186.66 seconds	1.06 seconds	98.03%	98.03%
K-Nearest Neig	162.62 seconds	540.64 seconds	97.8%	96.6 %
Artificial N.N.	206.17 seconds	0.51 seconds	96.54%	95.77%
Recurrent N.N	294.88 seconds	1.55 seconds	96.73%	96.78%
Convolutional N.N	703.48 seconds	19.54 seconds	95.5%	95.58%

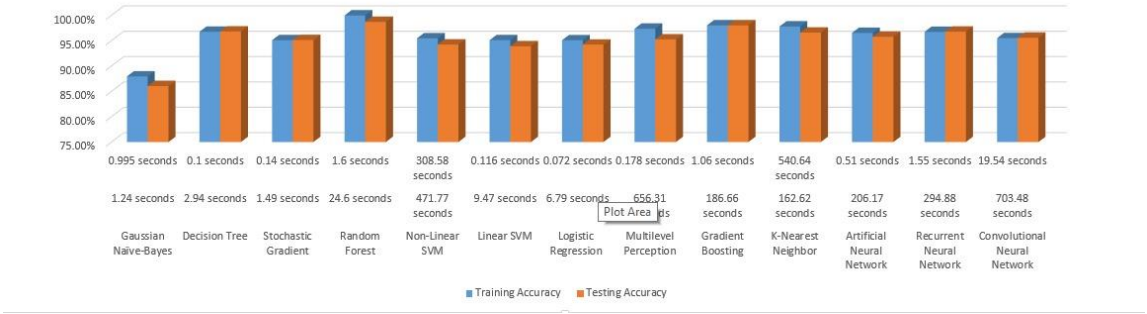


Fig 27: The outcomes of the training and testing methods for the UNSW-NB15 dataset

In the graphs below, you can see the comparison of the values in % of the training accuracy and testing accuracy of the 13 algorithms for the 4 tested datasets.

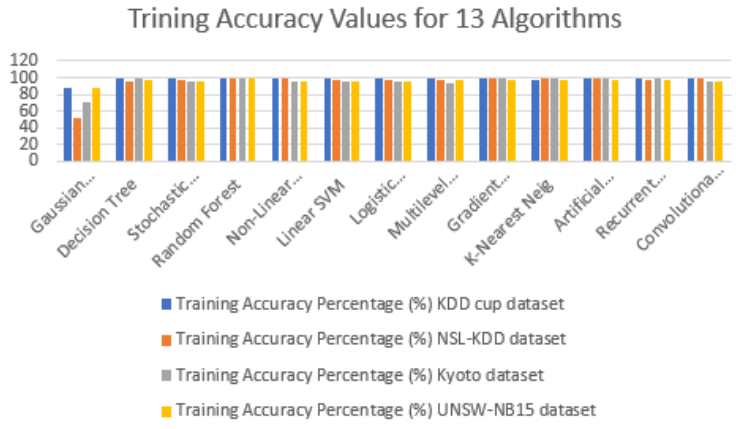


Fig 28: Training accuracy values for all algorithms

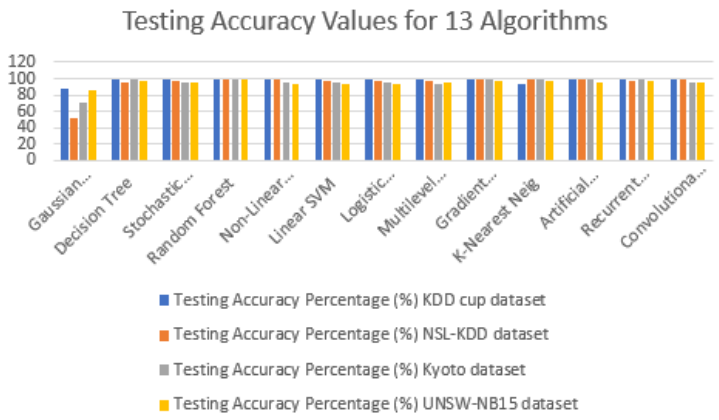


Fig 29: Testing accuracy values for all algorithms

Table 5 : RESULTS FOR TRAINING TIME, TESTING TIME, TRAINING ACCURACY AND TESTING ACCURACY FOR ALL 4 DATASETS TESTED

Machine Learning Algorithms	Training Time Seconds (s)				Testing Time Seconds (s)				Training Accuracy Percentage (%)				Testing Accuracy Percentage (%)			
	D 1	**D 2**	**D 3**	**D 4**	**D 1**	**D 2**	**D 3**	**D 4**	**D 1**	**D 2**	**D 3**	**D 4**	**D 1**	**D 2**	**D 3**	**D 4**
Gaussian Naïve-Bayes	2.25	0.12	1.4	1.24	3.42	0.12	0.61	0.995	88.21	51.28	71.15	87.95	88.21	51.56	71.11	86.05
Decision Tree	4.92	0.2	3.14	2.94	0.16	0.016	0.078	0.1	99.1	95.89	98.21	96.76	99.1	95.85	98.2	96.82
Stochastic Gradient	15.5	0.62	1.96	1.49	0.33	0.016	0.2	0.14	99.21	97.21	94.97	95.09	99.21	97.13	94.84	95.13
Random Forest	44.64	1.18	38.47	24.6	3.55	0.154	2.37	1.6	99.99	100	99.95	99.98	99.97	99.64	99.75	98.71
Non-Linear SVM	2173.8	3.011	1958.71	471.77	226.42	2.04	491.59	308.58	99.89	99.26	96.06	95.46	99.89	99.05	95.89	94.24
Linear SVM	48.26	1.4	39.5	9.47	0.23	0.016	0.054	0.116	99.71	96.97	94.91	95.1	99.69	97.12	94.81	93.89
Logistic Regression	40.87	1.62	10.89	6.79	0.23	0.015	0.046	0.072	99.33	96.74	94.86	95.08	99.31	96.68	94.79	94.23
Multilevel Perception	498.51	33.53	54	656.31	0.63	0.03	0.2	0.178	99.42	97.47	92.55	97.38	99.38	97.5	92.55	95.25
Gradient Boosting	1433.56	60.55	173.34	186.66	7.99	0.4	1.06	1.06	99.96	99.95	99.42	98.03	99.93	99.6	99.36	98.03
K-Nearest Neig	312.26	3.89	259.34	162.62	432.7	21.32	540.64	540.64	98.1	99.48	98.72	97.8	94.2	99.32	98.2	96.6
Artificial Neural Network	371.18	192.79	348.93	206.17	1.02	0.74	1.69	0.51	98.47	98.68	98.46	96.54	98.47	98.49	98.36	95.77
Recurrent Neural Network	448.51	234.92	294.88	294.88	1.1	0.83	1.55	1.55	98.46	97.82	98.77	96.73	98.48	97.61	98.73	96.78
Convolutional Neural Network	870.8	571.16	703.48	703.48	21.46	2.35	19.54	19.54	99.87	99.16	96.02	95.5	99.87	98.94	96.07	95.58

D 1 – KDD cup dataset

D 2 – NSL-KDD dataset

D 3 – Kyoto dataset

D 4 – UNSW-NB15 dataset

Gaussian Naïve-Bayes, Decision Tree and Stochastic Gradient have a short time of training and testing because of the credentials they have make them able to be implemented quickly. Thay have different accuracies because Gaussian Naïve-Bayes is based on probability, Decision Tree is an hierarchical algorithm and Stochastic Gradient is an optimization algorithm.

Random Forest, Linear SVM and Logistic Regression take a little more time to be implemented than algorithms mentioned above because there structure is a little bit more complex. Random forest has a great accuracy since it is a set of hierarchical structures (Decision Trees), Linear SVM which divides different types of cyber attacks using hyper planes and Logistic Regression which is a classification algorithm, in this case an multinomial statistical algorithm.

The other algorithms have a much more complex structure especially ANNs algorithms in different from machine learning algorithms is composed from more than one layer. Since they have a complex structure there accuracy is great.

Preprocessing of datasets: First we take the datasets and delete the unnecessary columns from the dataset. We take the cyber attacks column as output and other columns as input. If the columns are numbers we leave them as they are, if they are a combination of letters or a group of number and letters we Map them as integers.

By implementing supervised ML algorithms, we have been able to detect what types of output are a cyber attacks and what types are not.

Table 6 : THE RESULT FOR SENSITIVITY AND SPECIFICITY FOR EACH ALGORITHM IN THE 4 DATASETS TESTED

Machine Learning Algorithms	Sensitivity (%)				Specificity (%)			
	D 1	**D 2**	**D 3**	**D 4**	**D 1**	**D 2**	**D 3**	**D 4**
Gaussian Naïve-Bayes	88.21	51.56	71.11	86.05	87.23	52.44	69.2	85.2
Decision Tree	99.1	95.85	98.2	96.82	98.9	95.65	97.9	95.95
Stochastic Gradient	99.21	97.13	94.84	95.13	98.51	96.3	92.99	93.2

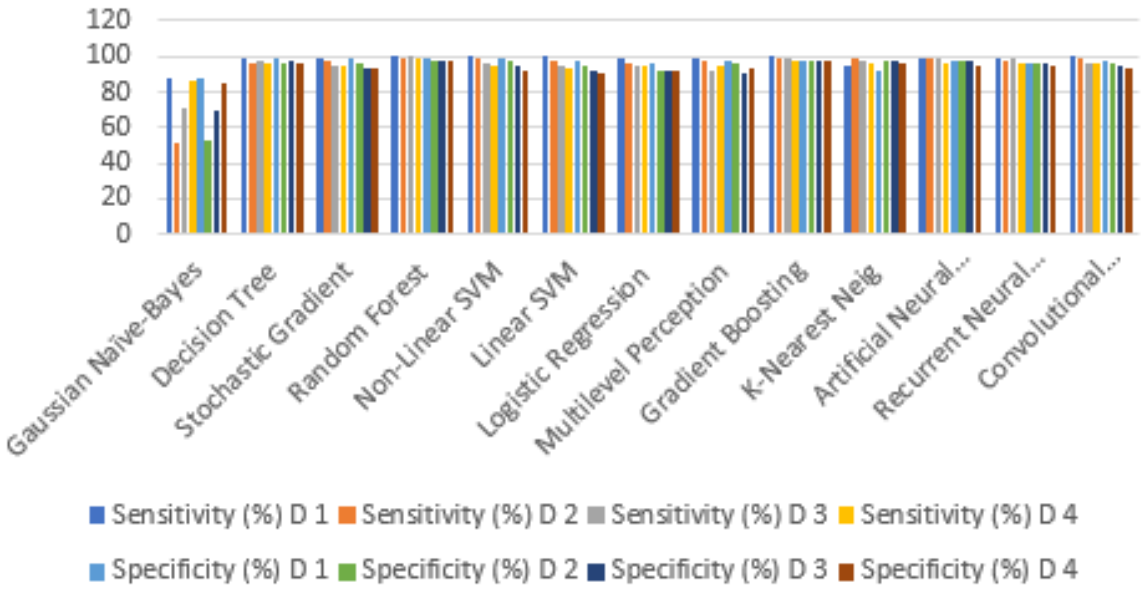
Random Forest	99.97	99.64	99.75	98.71	98.99	97.88	98.2	97.1
Non-Linear SVM	99.89	99.05	95.89	94.24	98.71	97.06	94.83	92.32
Linear SVM	99.69	97.12	94.81	93.89	97.86	95.32	92.11	90.63
Logistic Regression	99.31	96.68	94.79	94.23	96.81	92.54	92.23	92.33
Multilevel Perception	99.38	97.5	92.55	95.25	98.02	96.06	90.66	93.96
Gradient Boosting	99.93	99.6	99.36	98.03	98.02	98.04	97.85	96.96
K-Nearest Neig	94.2	99.32	98.2	96.6	92.35	97.2	96.92	95.66
Artificial Neural Network	98.47	98.49	98.36	95.77	97.36	97.36	97.05	94.27
Recurrent Neural Network	98.48	97.61	98.73	96.78	96.26	96.53	96.37	94.69
Convolutional Neural Network	99.87	98.94	96.07	95.58	97.78	96.52	94.56	93.06

D 1 – KDD cup dataset

D 2 – NSL-KDD dataset

D 3 – Kyoto dataset

D 4 – UNSW-NB15 dataset



D 1 – KDD cup dataset

D 2 – NSL-KDD dataset

D 3 – Kyoto dataset

D 4 – UNSW-NB15 dataset

Fig 30: Sensitivity and Specificity of the 12 algorithms for all tested datasets

CHAPTER V

SUMMARY AND SUGGESTIONS FOR FUTURE WORK

V.1 Summary

The purpose of these studies was to identify separate categories of malware that were present within the datasets. To do this, a variety of machine learning and deep learning methods were applied. As the algorithm for deep learning is taught more, it will generate findings that are more accurate. The great majority of the algorithms that were used in the detection of malware had an accuracy that was somewhat comparable to acceptable. On the other hand, one must exert a considerable amount of work in order to enhance the efficiency of these algorithms. The technical frontier of the future is represented by the incorporation of machine learning algorithms into a wide variety of fields, including cyber security. Within the scope of this thesis, we identified malware by using a number of different classification techniques across four different datasets. We have noticed that different machine learning algorithms have varying degrees of accuracy and different amounts of time required to apply them. According to the findings presented in this research, the amount of time required to develop algorithms like the Gaussian Naive-Bayes classifier, Logistic Regression, and Decision Tree classifier is very short. With the exception of the Gaussian Naïve-Bayes classifier algorithm, which achieves an accuracy ranging from 51% to 88%, all other algorithms surpass 90% in terms of accuracy.The implementation process for alternative classification algorithms, including Multilevel Perception, non-linear SVM, and Gradient Boosting, is considerably protracted. The algorithm that has achieved the highest level of accuracy among all the algorithms considered is the Random Forest Classification algorithm. However, each of them operates flawlessly. We have constructed a sophisticated system with an exceptional level of performance. This intelligent system is extremely potent and useful for identifying a great number of cyber threats.

V.2 Future Work

A prospective work proposal calls for the application of natural language processing and reinforcement learning algorithms to malware detection. They are exceptionally effective approaches.

REFERENCES

[1] Y. Qin and T. Xia, “Sensi tivity analysis of ring oscillator based hardware Trojan detection,” *Int. Conf. Commun. Technol. Proceedings, ICCT*, vol. 2017-Octob, pp. 1979–1983, 2018, doi: 10.1109ICCT.2017.8359975.

[2] S. Najari, “Malware Detection Using Data Mining Techniques,” *Int. J. Intell. Inf. Syst.*, vol. 3, no. 6, p. 33, 2014, doi: 10.11648/j.ijiis.s.2014030601.16.

[3] D. Jacobson and J. Idziorek, *Computer security literacy: staying safe in a digital world*, vol. 50, no. 10. 2013.

[4] D. Dasgupta, Z. Akhtar, and S. Sen, “Machine learning in cybersecurity: a comprehensive survey,” *J. Def. Model. Simul.*, 2020, doi: 10.1177/1548512920951275.

[5] R. A. Katole, “Parameter Values of SQL Query,” *2018 2nd Int. Conf. Inven. Syst. Control*, no. Icisc, pp. 736–741, 2018.

[6] H. M. Farooq and N. M. Otaibi, “Optimal machine learning algorithms for cyber threat detection,” *Proc. - 2018 UKSim-AMSS 20th Int. Conf. Model. Simulation, UKSim 2018*, pp. 32–37, 2018, doi: 10.1109/UKSim.2018.00018.

[7] V. Bhatia, S. Choudhary, and K. R. Ramkumar, “A Comparative Study on Various Intrusion Detection Techniques Using Machine Learning and Neural Network,” *ICRITO 2020 - IEEE 8th Int. Conf. Reliab. Infocom Technol. Optim. (Trends Futur. Dir.*, pp. 232–236, 2020, doi: 10.1109/ICRITO48877.2020.9198008.

[8] W. A. Ali, K. N. Manasa, M. Bendechache, M. F. Aljunaid, and P. Sandhya, “A review of current machine learning approaches for anomaly detection in network traffic,” *J. Telecommun. Digit. Econ.*, vol. 8, no. 4, pp. 64–95, 2020, doi: 10.18080/JTDE.V8N4.307.

[9] V. Mahesh and K. A. Sumithra Devi, “Prevention from Security Risks of Spyware by the use of Ai,” *1st Int. Conf. Adv. Technol. Intell. Control. Environ. Comput. Commun. Eng. ICATIECE 2019*, pp. 131–135, 2019, doi: 10.1109/ICATIECE45860.2019.9063838.

[10] I. Sumantra and S. Indira Gandhi, “DDoS attack Detection and Mitigation in

Software Defined Networks," *2020 Int. Conf. Syst. Comput. Autom. Networking, ICSCAN 2020*, 2020, doi: 10.1109/ICSCAN49426.2020.9262408.

APPENDIX

Kyoto dataset

```
# -*- coding: utf-8 -*-"""
Created on Sun Jun 20 16:34:53 2021

@author: User"""

import pandas as pd

import time
df = pd.read_csv("kyoto.csv")

df.drop('Source_IP_Address', axis = 1, inplace = True)
df.drop('Destination_IP_address', axis = 1, inplace = True )
df.drop('Start_time', axis = 1, inplace = True)
df.drop('IDS_detection', axis = 1, inplace = True)

fmap = {'OTH':0, 'REJ':1, 'RSTO':2, 'RSTOS0':3, 'RSTR':4, 'RSTRH':5, 'S0':6, 'S1':7, 'SF':8,
      'SHR':9}
df['Flag'] = df['Flag'].map(fmap)



pmap = {'icmp':0, 'tcp':1, 'udp':2}
df['Protocol'] = df['Protocol'].map(pmap)



smap = {'other':0, 'dns':1, 'smtp':2, 'snmp':3, 'ssh':4,'rdp':5, 'sip':6, 'http':7 ,'dhcp':8, 'ssl' : 9}
df['Service'] = df['Service'].map(smap)

lmap = {1: 'normal', -1: 'attack', -2:'unknown'}
df['Label'] = df['Label'].map(lmap)
y = df['Label']
```

```
X = df.drop(['Label'], axis = 1)

from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
X = sc.fit_transform(X)


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33)
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)


from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

clfg = GaussianNB()
start_time = time.time()
clfg.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Gaussian Naive-Bayes Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfg.predict(X_train)
end_time = time.time()
print("Gaussian Naive-Bayes Testing time: ", end_time-start_time)

print("Gaussian Naive-Bayes Train score is:", clfg.score(X_train, y_train))
print("Gaussian Naive-Bayes Test score is:", clfg.score(X_test, y_test))

from sklearn.tree import DecisionTreeClassifier

clfd = DecisionTreeClassifier(criterion ="entropy", max_depth = 4)
start_time = time.time()
```

```
clfd.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Decision Tree Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfd.predict(X_train)
end_time = time.time()
print("Decision Tree Testing time: ", end_time-start_time)

print("Decision Tree Train score is:", clfd.score(X_train, y_train))
print("Decision Tree Test score is:", clfd.score(X_test, y_test))

from sklearn.linear_model import SGDClassifier
clfsg = SGDClassifier(loss = 'hinge', max_iter = 10, tol = 0.0000000001)
start_time = time.time()
clfsg.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Stochastic Gradient Training time: ", end_time-start_time)
start_time = time.time()
y_test_pred = clfsg.predict(X_train)
end_time = time.time()
print("Stochastic Gradient Testing time: ", end_time-start_time)
print("Stochastic Gradient Train score is:", clfsg.score(X_train, y_train))
print("Stochastic Gradient Test score is:", clfsg.score(X_test, y_test))

from sklearn.ensemble import RandomForestClassifier

clfr = RandomForestClassifier(n_estimators = 30)
start_time = time.time()
clfr.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Random Forest Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfr.predict(X_train)
```

```
end_time = time.time()
print("Random Forest Testing time: ", end_time-start_time)

print("Random Forest Train score is:", clfr.score(X_train, y_train))
print("Random Forest Test score is:", clfr.score(X_test, y_test))

from sklearn.svm import SVC

clfs = SVC(gamma = 'scale')
start_time = time.time()
clfs.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Non-Linear SVM Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfs.predict(X_train)
end_time = time.time()
print("Non-Linear SVM Testing time: ", end_time-start_time)

print("Non-Linear SVM Train score is:", clfs.score(X_train, y_train))
print("Non-Linear SVM Test score is:", clfs.score(X_test, y_test))

from sklearn.svm import LinearSVC
clfls = LinearSVC(max_iter = 1000)
start_time = time.time()
clfls.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Linear SVM Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfls.predict(X_train)
end_time = time.time()
print("Linear SVM Testing time: ", end_time-start_time)
print("Linear SVM Train score is:", clfls.score(X_train, y_train))
print("Linear SVM Test score is:", clfls.score(X_test, y_test))
```

```
from sklearn.linear_model import LogisticRegression

clfl = LogisticRegression(max_iter = 1200000)
start_time = time.time()
clfl.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Logistic Regression Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfl.predict(X_train)
end_time = time.time()
print("Logistic Regression Testing time: ", end_time-start_time)

print("Logistic Regression Train score is:", clfl.score(X_train, y_train))
print("Logistic Regression Test score is:", clfl.score(X_test, y_test))

from sklearn.neural_network import MLPClassifier


mlp_clf = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(5, 2),
random_state=42)
start_time = time.time()
mlp_clf.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Multilevel Perception Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = mlp_clf.predict(X_train)
end_time = time.time()
print("Multilevel Perception Testing time: ", end_time-start_time)

print("Multilevel Perception Train score is: ", mlp_clf.score(X_train, y_train))
print("Multilevel PerceptionTest score is: ", mlp_clf.score(X_test, y_test))
```

```
from sklearn.ensemble import GradientBoostingClassifier

clfg = GradientBoostingClassifier(random_state = 0)
start_time = time.time()
clfg.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Gradient Boosting Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfg.predict(X_train)
end_time = time.time()
print("Gradient Boosting Testing time: ", end_time-start_time)

print("Gradient Boosting Train score is:", clfg.score(X_train, y_train))
print("Gradient Boosting Test score is:", clfg.score(X_test, y_test))



from sklearn.neighbors import KNeighborsClassifier
clfk= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
start_time = time.time()
clfk.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("KNN Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfk.predict(X_train)
end_time = time.time()
print("KNN Testing time: ", end_time-start_time)

print("KNN Train score is:", clfk.score(X_train,y_train))
print("KNN Test score is:", clfk.score(X_test,y_test))

from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
```

```
def ANNClassifier():
    annclassifier = Sequential()
    annclassifier.add(Dense(19, activation = 'relu', kernel_initializer = "random_uniform"))
    annclassifier.add(Dense(1, activation = 'sigmoid', kernel_initializer = "random_uniform"))
    annclassifier.add(Dense(2, activation = 'softmax'))
    annclassifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
    return annclassifier


annclassifier = KerasClassifier(build_fn = ANNClassifier, epochs=100,batch_size=1000)


start_time = time.time()
annclassifier.fit(X_train, y_train)
end_time = time.time()
print("Artificial Neural Network Training time ", end_time - start_time)



start_time = time.time()
y_test_pred = annclassifier.predict(X_test)
end_time = time.time()
print("Artificial Neural Network Testing time ", end_time - start_time)


start_time = time.time()
y_train_pred = annclassifier.predict(X_train)
end_time = time.time()
print("Artificial Neural Network Training accuracy", accuracy_score(y_train, y_train_pred))
print("Artificial Neural Network Testing accuracy", accuracy_score(y_test, y_test_pred))
from keras.layers import LSTM
```

```
def RNNClassifier():
    rnnclassifier = Sequential()
    rnnclassifier.add(LSTM(19))
    rnnclassifier.add(Dense(1, activation = 'sigmoid'))
    rnnclassifier.add(Dense(2, activation = 'softmax'))
    rnnclassifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
    return rnnclassifier

rnnclassifier = KerasClassifier(build_fn = RNNClassifier, epochs=100 ,batch_size=1000)
start_time = time.time()
annclassifier.fit(X_train, y_train)
end_time = time.time()
print("Recurrent Neural Network Training time ", end_time - start_time)


start_time = time.time()
y_test_pred = annclassifier.predict(X_test)
end_time = time.time()
print("Recurrent Neural Network Testing time ", end_time - start_time)

start_time = time.time()
y_train_pred = annclassifier.predict(X_train)
end_time = time.time()

print("Recurrent Neural Network Training accuracy", accuracy_score(y_train, y_train_pred))
print("Recurrent Neural Network Testing accuracy", accuracy_score(y_test, y_test_pred))

from keras.layers import Conv2D, MaxPooling2D
from keras.layers.advanced_activations import LeakyReLU
from keras.layers import Flatten
```

```
X_train = X_train.reshape(X_train.shape[0],19,1,1)
X_test = X_test.reshape(X_test.shape[0],19,1,1)



def CNNClassifier():
    cnnclassifier = Sequential()#add input layer and first hidden layer
    cnnclassifier.add(Conv2D(32, (3, 3), activation='relu', input_shape=(19, 1, 1), padding='same'))
    cnnclassifier.add(LeakyReLU(alpha = 0.1))
    cnnclassifier.add(MaxPooling2D(pool_size=(4, 4), padding = 'same'))
    cnnclassifier.add(Conv2D(64, (3, 3), activation = 'relu', padding = 'same'))
    cnnclassifier.add(LeakyReLU(alpha = 0.1))
    cnnclassifier.add(MaxPooling2D(pool_size = (4,4), padding = 'same'))
    cnnclassifier.add(Flatten())
    cnnclassifier.add(LeakyReLU(alpha = 0.1))
    cnnclassifier.add(Dense(2, activation = 'softmax'))
    cnnclassifier.compile(optimizer = 'adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return cnnclassifier


cnnclassifier = KerasClassifier(build_fn = CNNClassifier, epochs=10, batch_size=1000)


start_time = time.time()
cnnclassifier.fit(X_train, y_train.values.ravel())
end_time = time.time()


print("Convolutional Neural Network Training time:", end_time-start_time)


start_time = time.time()
y_test_pred = cnnclassifier.predict(X_test)
end_time = time.time()
print("Convolutional Neural Network Testing time ", end_time - start_time)


start_time = time.time()
y_train_pred = cnnclassifier.predict(X_train)
```

```
end_time = time.time()


print("Convolutional Neural Network Training accuracy", accuracy_score(y_train, y_train_pred))
print("Convolutional Neural Network Testing accuracy", accuracy_score(y_test, y_test_pred))
```