University of Miskolc

Institute of Information Science

Department of Information Technology

Faculty of Mechanical Engineering and Informatics



**Analysis And Implementation Of ML And DL Techniques In The Detection Of DDoS Attacks**

**Master Thesis**

Author: Vinay Karthik Maradi Balachandra

Registration code: AZF2YA

Miskolc, Hungary

**2021**

|  |  |  |
| --- | --- | --- |
| ***University of Miskolc***  ***Faculty of Mechanical Engineering and Informatics*** |  | ***Department of***  ***Information Technology***  *Miskolc-Egyetemváros, 3515 Hungary* |
| *Major:* ***Computer Science and Engineering*** |  | *Thesis number:* ***GEIAL-535/Ma/MSc*** |
| *Level:**Master* |  | *National Uni. Id.: FI 87515* |

***THESIS SPECIFICATION***

### *Maradi Balachandra Vinay Karthik*

*Registration code: AZF2YA*

*Candidate for MSc degree in Computer Science and Engineering*

|  |  |
| --- | --- |
| *Subject of thesis:* | ***Artificial Neural Networks*** |
| *Title of thesis:* | ***Application of Artificial Neural Networks in the field of CyberSecurity*** |
|  |  |
| *Detailed specification:* ***Analysis of Viability of the use of Artificial Neural Networks in the Detection of DDoS Attacks*** | |

|  |  |
| --- | --- |
| *Supervisor:*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* | *Affiliation, position:*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
| *Consultant(s):*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* | *Affiliation/company, position:*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
| *Date of issue:* | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
| *Deadline for submission:* | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
| *Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* | **Dr. Kovács, László** *Head of Department* |

##### 1. *Place of practice field :* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2. Supervisor of practice field:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*3. Modifications to the thesis: required (should be attached separately)   
 not required (underline as appropriate)*

*Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_ Supervisor*

*4. Dates of consultation: (1)*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(2)*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(3)*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(4)*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Date Supervisor/Consultant*

*5. Thesis submission: accepted / not accepted (underline as appropriate)*

*Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Consultant/Supervisor*

*6. Thesis contains: ...… pages,*

*...… appendices,*

*...… CD attachment,*

*...… other attachment.*

*7. Thesis: approved / not approved (underline as appropriate)*

*Name and affiliation of Opponent:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Head of Department*

*8. Grades: Opponent:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Department:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Final Examination Board:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Chairman of FEB*

DECLARATION OF AUTHORSHIP

I,……………………………………………………………………………………, Neptun code: …………………………………………., MSc student of the Faculty of Mechanical Engineering and Informatics, University of Miskolc, being acutely aware of my legal liability, hereby confirm, declare and certify with my signature, that the assignment, entitled………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………… - except where indicated by referencing - , is my own work, is not copied from any other person’s work, and is not previously submitted for assessment at University of Miskolc or elsewhere, and all sources (both the electronic and printed literature, or any kind) referred to in it, have been used in accordance with the rules of copyright. I understand that a thesis work may be considered to be plagiarized if it consists of - Quoting word by word or referring to literature either with no quotation marks or no proper citation; - Referring to content without indicating the source of reference; - Representing previously published ideas as one’s own. I, hereby declare that I have been informed of the term of plagiarism, and I understand that in the case of plagiarism my thesis work is rejected.

Miskolc,

…………… (day) ................(month) ………………..(year)

------------------------------------------------

Student’s signature

**Table of Contents**

[Thesis Specification](#_TOC_250142) 2

[Declaration of Authorship](#_TOC_250142) 4

[Glossary](#_TOC_250142) 8

[Chapter](#_TOC_250141) 1

[Introduction 9](#_TOC_250140)

* Background 9
* Type of DDoS 12
* Motivation for DDoS 15
* Mitigation of DDoS 15

[Specification of the Problem 17](#_TOC_250135)

T[he Scope of the Thesis 18](#_TOC_250130)

Proposed Solution 19

Chapter 2

Literature Review [20](#_TOC_250107)

* Analysis of Present Available Solution 20
* Related Work 24

Chapter 3

[Definition of Technology 25](#_TOC_250127)

* Support Vector Machine 25
* K-Nearest Neighbors 27
* Artificial Neural Networks 30
* Random Forest Classifier 32
* Extra Trees Classifier 34

Chapter 4

Application of ML and DL techniques for Problem Statement 38

* KDD Cup ’99 DataSet 38
* Pre-Processing and Extensive Data Analysis 40
* Support Vector Machine 43
* K-Nearest Neighbors 46
* Artificial Neural Networks 51
* Extra Trees Classifier 53
* Random Forest Classifier 57

Chapter 5

Comparison of Performance of the ML and DL algorithms 60

* Evaluation Functions 60
* Performance of the different models 62

Chapter 6

Application and Performance Evaluation of the Proposed Solution 64

* Proposed Solution Algorithm 64
* Performance of the Proposed Solution 65

Chapter 7

Conclusion and Future Work 67

Appendix 68

* List of Figures 68
* List of Tables 69
* Program Code 71
* [References / Bibliography](#_TOC_250001) 99

|  |  |
| --- | --- |
| GLOSSARY | |
| D/DoS | Distributed/Denial-Of-Service |
| SVM | Support Vector Machine |
| ANN | Artificial Neural Networks. |
| ETC | Extra Trees Classifier |
| CNN | Convolutional Neural Network |
| KNN | K-Nearest Neighbors |
| RFC | Random Forest Classifier |
| DL | Deep Learning |
| ML | Machine Learning |
| IDS | Intrusion Detection System |

**Chapter 1**

**Introduction**

**Background**

Denial of service attack (DoS attack) is a type of cyberattack in which an Internet site is made unavailable, typically by using multiple computers to repeatedly make requests that tie up the site and prevent it from responding to requests from legitimate users.

The first documented DoS-style attack occurred during the week of February 7, 2000, when “mafiaboy,” a 15-year-old Canadian hacker, orchestrated a series of DoS attacks against several e-commerce sites, including Amazon and eBay. These attacks used computers at multiple locations to overwhelm the vendors’ computers and shut down their World Wide Web (WWW) sites to legitimate commercial traffic. The attacks crippled Internet commerce; the U.S. Federal Bureau of Investigation (FBI) estimated that the affected sites suffered $1.7 billion in damages. In its early years the Internet had played a role only in the lives of researchers and academics, but by 2000 it had become essential to the workings of many governments and economies. Cybercrime had moved from being an issue of individual wrongdoing to being a matter of national security.

Distributed DoS (DDoS) attacks are a special kind of hacking. A criminal salts an array of computers with computer programs that can be triggered by an external computer user. These programs are known as Trojan horses since they enter the unknowing users’ computers as something benign, such as a photo or document attached to an e-mail. At a predesignated time, this Trojan horse program begins to send messages to a predetermined site. If enough computers have been compromised, it is likely that the selected site can be tied up so effectively that little if any legitimate traffic can reach it. One important insight offered by these events has been that much software is insecure, making it easy for even an unskilled hacker to compromise a vast number of machines. Although software companies regularly offer patches to fix software vulnerabilities, not all users implement the updates, and their computers remain vulnerable to criminals wanting to launch DoS attacks.

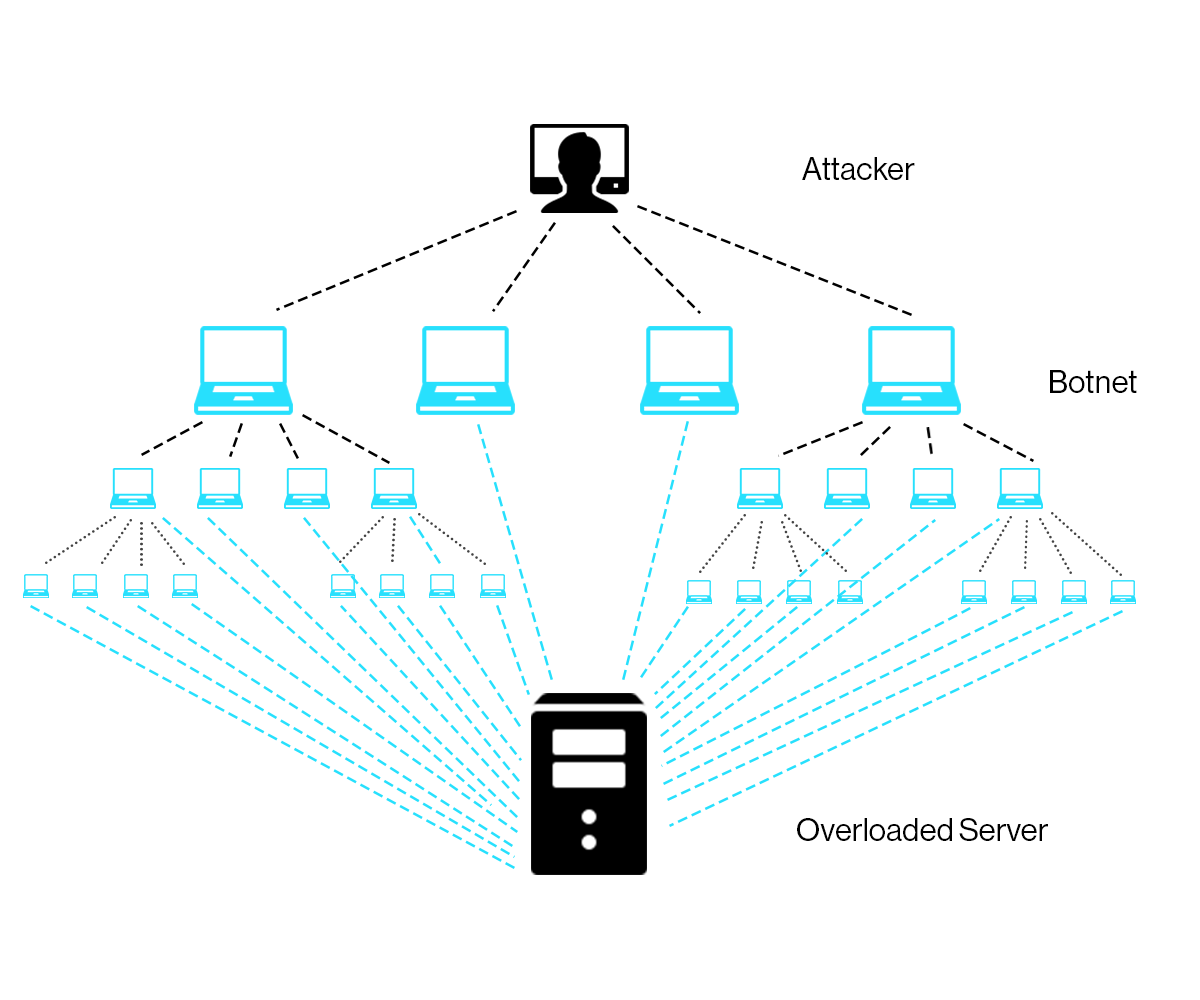


Figure 1: DDoS Attack

One of the worst DDoS attacks occurred in October 2016 when a botnet (a network of infected devices) called Mirai brought down the servers of Dyn, an American company that is in charge of much of the Internet’s domain name system (DNS). This attack interrupted much of North American Internet traffic. The Mirai botnet was not made up of infected computers but infected other devices, such as baby monitors, digital video recorders (DVRs), and digital cameras, that could connect to the Internet. Only vigorous security regimes can protect against such an environment.

DDoS and DoS attacks apparently have been used for political purposes, with neighbors of Russia (most notably Estonia in 2007, Georgia in 2008, and Ukraine in 2014 and 2015) having their Web sites targeted in times of conflict in the region. The Russian government has been suspected of being behind these attacks, but its involvement has not been definitively proven. The recent attacks on Russia in 2022 also is an example of DDoS

The key objective of a Distributed Denial of Service (DDoS) attack is to compile multiple systems across the Internet with infected zombies/agents and form botnets. Such zombies are designed to attack a particular target or network with different types of packets. The infected systems are remotely controlled either by an attacker or by self-installed Trojans that are programmed to launch packet floods.

It is possible to use VPSs to create a botnet for private use, and deploy virtual machines to attack a target. Tools like IP Stresser, Low Orbit Ion Cannon (LOIC), High Orbit Ion Cannon (HOIC), Slowloris, R.U.D.Y (R-U-Dead-Yet). Some of the methods have no permanent solution yet. A simple RUDY attack or Slowloris may cause millions in damage to the right service at the right time.

Cybersecurity experts have explained different DDoS architectural structures used by DDoS engineers to launch successful attacks. DDoS attacks are serious security issues that cost organizations and individuals a great deal of time, money and reputation, yet they do not usually result in the compromise of either credentials or data loss. They can damage one or a group of devices and their resources. A DDoS attack slows or halts communications between devices as well as the victim machine itself. It introduces loss of Internet services like email, online applications or program performance.

Protecting digital assets and intellectual property is becoming challenging for organizations. Recent studies describe external hacking as the primary cause of data loss in the corporate industry. Organizations are expected to take adequate measures to protect data from loss or leakage. Quite often, unchecked IT cyber security risk factors that remain unmitigated for too long-something that happens in almost all business are the cause for unexpected cyber attacks. Intrusion Detection Systems (IDS) are now mainly employed to secure company networks. An IDS is highly recommended to detect all attempted intrusions, and to protect the organization from the attack.

**Types of DDoS**

DoS and DDoS attacks can be divided into three types:

* Volume Based Attacks - Includes UDP floods, ICMP floods, and other spoofed-packet floods. The attack’s goal is to saturate the bandwidth of the attacked site, and magnitude is measured in bits per second (Bps).
* Protocol Attacks - Includes SYN floods, fragmented packet attacks, Ping of Death, Smurf DDoS and more. This type of attack consumes actual server resources, or those of intermediate communication equipment, such as firewalls and load balancers, and is measured in packets per second (PPS).
* Application Layer Attacks - Includes low-and-slow attacks, GET/POST floods, attacks that target Apache, Windows or OpenBSD vulnerabilities and more. Comprised of seemingly legitimate and innocent requests, the goal of these attacks is to crash the web server, and the magnitude is measured in Requests per second (RPS).

Some of the most commonly used DDoS attack types include:

1. UDP Flood

A UDP flood, by definition, is any DDoS attack that floods a target with User Datagram Protocol (UDP) packets. The goal of the attack is to flood random ports on a remote host. This causes the host to repeatedly check for the application listening at that port, and (when no application is found) reply with an ICMP ‘Destination Unreachable’ packet. This process saps host resources, which can ultimately lead to inaccessibility.

1. ICMP (Ping) Flood

Similar in principle to the UDP flood attack, an ICMP flood overwhelms the target resource with ICMP Echo Request (ping) packets, generally sending packets as fast as possible without waiting for replies. This type of attack can consume both outgoing and incoming bandwidth, since the victim’s servers will often attempt to respond with ICMP Echo Reply packets, resulting a significant overall system slowdown.

1. SYN Flood

A SYN flood DDoS attack exploits a known weakness in the TCP connection sequence (the “three-way handshake”), wherein a SYN request to initiate a TCP connection with a host must be answered by a SYN-ACK response from that host, and then confirmed by an ACK response from the requester. In a SYN flood scenario, the requester sends multiple SYN requests, but either does not respond to the host’s SYN-ACK response, or sends the SYN requests from a spoofed IP address. Either way, the host system continues to wait for acknowledgement for each of the requests, binding resources until no new connections can be made, and ultimately resulting in denial of service.

1. Ping of Death

A ping of death (“POD”) attack involves the attacker sending multiple malformed or malicious pings to a computer. The maximum packet length of an IP packet (including header) is 65,535 bytes. However, the Data Link Layer usually poses limits to the maximum frame size – for example 1500 bytes over an Ethernet network. In this case, a large IP packet is split across multiple IP packets (known as fragments), and the recipient host reassembles the IP fragments into the complete packet. In a Ping of Death scenario, following malicious manipulation of fragment content, the recipient ends up with an IP packet which is larger than 65,535 bytes when reassembled. This can overflow memory buffers allocated for the packet, causing denial of service for legitimate packets.

1. Slowloris

Slowloris is a highly-targeted attack, enabling one web server to take down another server, without affecting other services or ports on the target network. Slowloris does this by holding as many connections to the target web server open for as long as possible. It accomplishes this by creating connections to the target server, but sending only a partial request. Slowloris constantly sends more HTTP headers, but never completes a request. The targeted server keeps each of these false connections open. This eventually overflows the maximum concurrent connection pool, and leads to denial of additional connections from legitimate clients.

1. NTP Amplification

In NTP amplification attacks, the perpetrator exploits publically-accessible Network Time Protocol (NTP) servers to overwhelm a targeted server with UDP traffic. The attack is defined as an amplification assault because the query-to-response ratio in such scenarios is anywhere between 1:20 and 1:200 or more. This means that any attacker that obtains a list of open NTP servers (e.g., by a using tool like MetaSploit or data from the Open NTP Project) can easily generate a devastating high-bandwidth, high-volume DDoS attack.

1. HTTP Flood

In an HTTP flood DDoS attack, the attacker exploits seemingly-legitimate HTTP GET or POST requests to attack a web server or application. HTTP floods do not use malformed packets, spoofing or reflection techniques, and require less bandwidth than other attacks to bring down the targeted site or server. The attack is most effective when it forces the server or application to allocate the maximum resources possible in response to every single request.

1. Zero-day DDoS Attacks

The “Zero-day” definition encompasses all unknown or new attacks, exploiting vulnerabilities for which no patch has yet been released. The term is well-known amongst the members of the hacker community, where the practice of trading zero-day vulnerabilities has become a popular activity.

**Motivation for DDoS**

Attackers are primarily motivated by:

* Ideology – So called “hacktivists” use DDoS attacks as a means of targeting websites they disagree with ideologically.
* Business feuds – Businesses can use DDoS attacks to strategically take down competitor websites, e.g., to keep them from participating in a significant event, such as Cyber Monday.
* Boredom – Cyber vandals, a.k.a., “script-kiddies” use prewritten scripts to launch DDoS attacks. The perpetrators of these attacks are typically bored, would-be hackers looking for an adrenaline rush.
* Extortion – Perpetrators use DDoS attacks, or the threat of DDoS attacks as a means of extorting money from their targets.
* Cyber warfare – Government authorized DDoS attacks can be used to both cripple opposition websites and an enemy country’s infrastructure.
* Gaming- In gaming, there are two main concerns regarding DDoS.
  + First are attacks on individual gamers. Due to the competitive nature of online gaming, this tends to be the most common form of DDoS in gaming. Attackers find the IPs of other gamers and eat up their bandwidth by inundating their connections with traffic/requests. The victim of a DDoS attack experiences poor and unstable performance, giving the attacker a competitive gaming edge. These attacks can even force players to disconnect from lobbies and matches.
  + Outside of gamers attacking one another, the other type of DDoS attack in gaming directly targets servers of video game publishers or common platforms like PlayStation, Xbox, and Steam. As these attacks are intended to disrupt the entire system, they’re likely not motivated by competition. Like DDoS attacks on any other server, DDoS on game servers aims to overwhelm and bog down the servers with excessive amounts of traffic.

**Mitigation of DDoS Attacks**

DDoS mitigation is a set of network management techniques and/or tools for resisting or mitigating the impact of distributed denial-of-service (DDoS) attacks on networks attached to the Internet by protecting the target and relay networks. DDoS attacks are a constant threat to businesses and organizations by threatening service performance or to shut down a website entirely, even for a short time.

The foundation of DDoS mitigation certainly rests in building up robust infrastructure. Keeping resilience and redundancy top-of-mind through the following are all crucial first steps for DDoS mitigation:

* Strengthening bandwidth capabilities
* Securely segmenting networks and data centers
* Establishing mirroring and failover
* Configuring applications and protocols for resiliency
* Bolstering availability and performance through resources like content delivery networks (CDNs)

Security teams running DDoS mitigation programs usually seek out technology or services that help them automatically determine the difference between legitimate traffic spikes and actual DDoS Attacks.

* Traffic analysis

Most DDoS mitigation strategies lean on 24x7 traffic monitoring to keep an eye out for threats and spot the early signs of DDoS activity before it snowballs into unmanageable volumes or lingers on through low-and-slow DDoS techniques that may degrade performance without taking a system completely offline. Organizations that do not have the staff to provide around-the-cloud monitoring frequently turn to managed service providers to fill that role. Managed DDoS mitigation can make all the difference in minimizing the cost of downtime and productivity in the wake of an attack.

* Anomaly detection

Monitoring capabilities are typically backstopped by anomaly detection technology that's tuned to network baselines and polices, as well as to threat intelligence sources that track the latest indicators of compromise (IOCs) associated with the most recent DDoS attack tactics. These detections then trigger reactive responses from DDoS mitigation experts and/or automated technology.

* Rerouting and scrubbing

Many organizations utilize a combination of on-premises solutions such as DDoS mitigation appliances, firewalls, and unified threat management appliances to block DDoS activity as it is detected. However, this requires significant appliance tuning and the hardware limits how much traffic these devices can deflect or absorb.

Many organizations are turning to cloud-based DDoS mitigation solutions or managed security solution providers. When the monitoring and anomaly detection senses malicious traffic or activity, DDoS mitigation infrastructure will then ideally reroute that traffic through cloud-based filtering system before crossing the network edge, leaving legitimate traffic to continue unabated through existing systems as usual. The scrubbing done by that external resource helps organizations better block and absorb high-volume DDoS activity, maintaining uptime even in the face of targeting by massive botnets.

While much of the initial attack response is automated through technology, effective DDoS mitigation also requires a well-trained team to make changes on the fly when attack scenarios throw unusual volume, techniques, or extended attacks at the network. In addition to incident response capabilities, organizations may need to lean on security analysts to conduct post-mortem reviews that could help them adjust future DDoS mitigation planning or tuning of tools.

**Specification of the Problem**

DDoS (Distributed Denial of Service) attacks have become a pressing threat to the security and integrity of computer networks and information systems, which are indispensable infrastructures of modern times. The detection of DDoS attacks is a challenging issue before any mitigation measures can be taken. ML/DL (Machine Learning/Deep Learning) can be applied to the detection of DDoS attacks.

A new botnet called the Meris botnet emerged in mid-2021 and continued to bombard organizations around the world, launching some of the largest HTTP attacks on record — including a 17.2Mrps(Million requests per second) attack that Cloudflare automatically mitigated. The average losses from DDoS is anywhere from a few thousand Euroes to Millions depending on the target and the severity of the attack. Hence it is not viable to take DDoS attacks lightly and they need to be regarded in priority.

The Detection of the DDoS is of paramount importance as the detection is the root to the Solution in any Cyber Attack scenario. Hence there is a need for a real time system which monitors the network and analyzes the incoming data traffic packets to be classified as either a normal or an atttack data packet and to either close the connection or keep the connection alive.

To make the decision if the incoming packets are a threat or not, we can use the ML and DL algorithms and analyze their performance and implement the best performer in the proposed Intrusion Detection System

**Scope of the Thesis**

By flooding malicious traffic, DoS (Denial of Service) attacks deplete the network bandwidth and computing resources of a targeted system, preventing the target system from offering regular services to legitimate users. DDoS (Distributed Denial of Service) goes even further on a much larger scale. DDoS attacks take over the control of a large number of comprised systems, called a botnet, and launch coordinated attacks on the victim system. Along with the emergence and advancement of disruptive Internet technologies, DDoS attacks are evolving and proliferating in scale, frequency, and sophistication. Organizations face potential threats to their network environment that may cause severe impacts to their operations, such as business downtime, data breaches, or even ransom demands from hackers.

Upon the occurrence of DDoS attacks, actions for DDoS mitigation should be taken,

The detection of DDoS attacks is essential before any mitigation approaches can be taken. In the early era, the alarm of DDoS attacks was triggered by rules programmed by traffic engineers. This approach apparently failed to catch up with the dynamic and evolving natures of DDoS attacks.

As ML/DL unleashes their great potential in different fields, academics and industries are exploring the possibility of applying ML/DL to DDoS detection. Traditional manual methods suffer from low accuracy and long latency problems in risk identification. With ML approaches, such as Naive Bayesian, KNN, and Random Forest, threats can be captured more quickly and more accurately. In ML, features for classification must be selected by human experts or by certain feature selection schemes. On the other hand, feature selection is an integral part of DL. Deep Learning models such as CNN and RNN are built based on a series of nonlinear processing layers to learn many levels of data representation from a large volume of labeled samples. Therefore, DL can serve as a powerful tool for DDoS detection. BI-LSTM is capable of capturing essential characteristics of DDoS traffic, in particular, the time domain correlation.

ML and DL have proven themselves effective solutions to the detection of DDoS attacks. However, they are trained to recognize only instances drawn from the distribution models constructed from the training set. Thus, they could fail in cases they have never learned. To know what one doesn’t know is the problem called the Open Set Recognition problem. This problem has a severe impact on the detection of DDoS attacks since DDoS attacks’ technology keeps evolving and results in changing traffic characteristics.

A real time estimation of the number of zombies in DDoS attack scenario is helpful to suppress the effect of attack by choosing predicted number of most suspicious attack sources for either filtering or rate limiting. ANN can be employed to estimate number of zombies involved in a DDoS attack. The method does not depend on the frequency of attack and hence solves the problem of low detection precision and weak detection stability of ANN which occurs when used for low frequent attack estimation. The sample data used to train the feed forward neural networks is generated using NS-2 network simulator running on Linux platform. Various sizes of feed forward networks are compared for their estimation performance using MSE. The generalization capacity of the trained network is promising and the network is able to predict number of zombies involved in a DDoS attack with very less test error.

**Proposed Solution**

In the proposed solution we propose to use the KDD Cup ’99 Train and Test dataset to analyze the performance of various ML and DL techniques to implement the best performing algorithm into an Intrusion Detection System. The algorithms proposed to be used are Support Vector Machine, K-Nearest Neighbours, Multi-Level Perceptron, Extra Trees Classifier and Random Forest Classifier. The Datasets are preprocessed and the algorithms are used on the data sets with multiple variations to analyze the algorithms and based on the achieved accuracies to propose a model which can be implemented in the Intrusion Detection System.

**Chapter 2**

**Literature Review**

**Analysis of Present Available Solutions**

Various ML technologies have been employed, mainly as classifiers, in the detection of DDoS attacks. These include Support Vector Machines (SVM), k-Nearest Neighbors (KNN), the Naïve Bayes Classifier, Random Forest (RF), Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and Artificial Neural Networks (ANNs), to name a few. With SVM, based on labeled training data, a hyperplane is constructed in the transform domain to classify unseen data. Cheng and team has built an IAI (IP Address Interaction Feature) model that can effectively discriminate normal from abnormal flows in traffic streams and helps to identify attack flows quickly and precisely. In KNN,k nearest neighbors of incoming data are located. A majority of these k neighbors decide the classification of the incoming data.

In [1], the paper proposed Denial of service Attack Prediction using the Gradient Descent algorithm yields the result of accuracy 99.7% with a 3.3% error. To overcome the error rate use the Linear Regression algorithm in which the error can be reduced to 0.3%. In [2], discussed A Classification Framework to Detect DoS Attacks used NSL-KDD dataset. The performance is compared with the 10 commonly used classification technique Naive Bayes (NB), Support Vector Machine (SVM), Multilayer Perceptron (MLP), K-Nearest Neighbor (KNN), Decision Tree (DT), Radial Basis Function (RBF), One Rule (OneR), PART, Bayesian Network, and Random Tree.

In [3], Tahir Alyas discussed Detection and mitigation of DDoS attacks in cloud computing using a machine learning algorithm. The two most effective algorithms Naïve Bayes and Random forest, concluding with that Naïve Bayes produce more positives. In [4], focused on A study for the DDOS attack classification method. Here, three receptive classification algorithms can be compared to Naive Bayes, Decision tree, Artificial neural network from that Artificial neural network produces the best accuracy rate of 84.3% compared with the other two techniques. In [6], the paper proposed Research on multiple Machine learning for anomaly detection. In this approach NSL-KDD dataset is divided into two dataset its performance can be compared with the confusion matrix.

Fadil and team used the NB method to predict the existence of DDoS attacks based on the mean and standard deviation of network packets and achieved precise results. An RF is a collection of decision trees. The majority of the outcomes of individual decision trees determine the classification[5]. In [7], Wang and team proved that with well-computed key features in DDoS data, experimental results show an RF algorithm can attain valid classification performance and an optimal feature subset.

Li and Lu in [8] combined Long Short-Term Memory (LSTM) and Bayesian methods to detect DDoS attacks. LSTM is suitable for events with long intervals and delays in the time domain. In other words, LSTM can control the value of the indefinite length of time and decide whether the information should be retained or removed. The author used LSTM to identify the confidence index of DDoS attacks and further used the Bayesian method to make a second judgment to improve detection accuracy.

In [9], Yong and team applied machine learning models to detect webshell to build secure solutions for IoT networks. Ensemble methods, including random forest (RF), extremely randomized trees (ET), and Voting, are used to improve the performances of these machine learning models. Their findings show that RF and ET are suitable for lightweight IoT scenarios, and the Voting method is effective for heavyweight IoT scenarios.

Hemalatha and team in [10] propose an efficient malware detection system based on deep learning. The system uses a reweighted classbalanced loss function in the final classification layer of the DenseNet model to achieve significant performance improvements in classifying malware by handling imbalanced data issues. Comprehensive experiments performed on four benchmark malware datasets showed that the proposed scheme has superior performance with a higher detection rate and lower computational cost.

Idhammad et al. , presented a technique for detecting DDoS attacks using a feed-forward neural network in[11]. They gathered inbound traffic first, then used a correlation-based feature selection method to extract relevant information. The UNSW-NB15 and NSLKDD datasets were used to validate their method.

Bhuvanesvari, Amma et al in [12], realized skill, deep intellect. The authors extracted intelligence from radial basis functions composed of different levels of abstraction. Experiments were performed on the well-known Network Security Laboratory (NSL KDD) and UNSW-NB15 datasets. The authors claim that it is more accurate than other existing methods.

Intrusion detection using machine learning algorithms was proposed by Nuno Martins et al.in [13] . They used the KDD dataset. They used various supervised models to balance the unclassification method and improve its performance. In this paper, a comparative research was proposed using several classification methods, and the results were positive.

For evaluation, in [16] Larriva-Novo et al. presented two bench- mark datasets, particularly UGR16 and UNSW-NB15, as well as the most widely used dataset KDD99. The scalar and standardisation capabilities of the pre-processing technique are assessed. Various attribute arrangements are used to apply these pre-processing models. The purpose of this investigation is to assess this structure by utilising various information A comparison work for network traffic classification was proposed by Xianwei Gao et al. in [15]. For intrusion detection, they used classification algorithm. The datasets CI- CIDS and KDD were obtained from the UCI repository. In comparison to other methods, they discovered support vector machine SVM to be one of the better.

For detecting several sorts of abnormalities in the network, Xiao et al. in[14] devised the most widely used kNN technique. The kNN method was used to identify as many bots as possible in the network. When it came to detecting unknown attacks, the accuracy of any method was increased.

Using data mining techniques and machine learning algorithms, Esra SUT et al. created a DDoS detection system. On this system and KDDCUP99, they compared several machine learning algorithms in terms of speed and accuracy. They used empirical methods to determine the best value for several hyperparameters, such as 10 for Cross-Validation Ratio and 66% for Dataset for training model size. According to this study, the J48 algorithm has the highest success rate for correctly detecting DDoS attacks.

Based on machine learning approaches and traffic flow traces, Ajeetha G and Madhu Priya G created a DDoS detection method in [17]. They compared the accuracy of the Naive Bayes and Random Forest algorithms on datasets acquired from Sans and Isna and discovered that the Nave Bayes system, with an accuracy of 90.90 percent, outperforms the Random Forest algorithm, which has an accuracy of 78.18 percent.

Despite the success attained by the ML/DL-based schemes, a critical issue, the OSR problem, is overlooked. Bendale and Sabeel in [18] and [19] point out that, without proper measures, ML/DL could forcibly classify instances from new sample space into the wrong category. The correct action is to differentiate novel instances from training samples, which is the strategy adopted by. They examined the hyper distance from incoming data to known classes and marked it as a new instance if the distance exceeded a certain threshold.

Jie-Hao and Ming have used ANN to detect DDoS attacks where they compared the detection outcome with decision tree, ANN, entropy and Bayesian.The authors identified users' requests or demands to a specific resource and their communicative data.Then samples of such requests are sent to the detection systems to be judged for abnormalities.

Also, In [8] Liu and Gu have used LearningVectorQuantisation(LVQ) neural networks to detect attacks. This is a supervised version of quantisation,which can be used for pattern recognition,multi- class classification and data compression tasks.The data sets used in the experiments were converted into numerical form and given as inputs to the neural network.

In [7], the authors deployed their approach (Probabilistic Neural Network) over two periods (attack and normal traffic profiles) accuracy was measured at 92% (period one) and 97% (period two). In [8] the authors compared two different learning approaches, Back-Propagation (BP) and Learning Vector Quantization (LVQ). Our comparison is with BP algorithm since our approach is based on BP, which indicates good perfor-mance and accuracy. However, [22] used a statistical approach while [23] used Kernel Principle Component Analysis (KPCA) and Particle Swarm Optimization (PSO)-Support Vector Machine (SVM) to detect DDoS attacks. KPCA is used to remove redundant features and PSO is used to optimize SVM.

**Comparison of Related Work**

|  |  |  |  |
| --- | --- | --- | --- |
| **Authors** | **DataSet** | **Method** | **Efficiency** |
| Akilandeswari, V. Shalinie, S.M | NA | PNN | 92 97 |
| Li, J.; Liu, Y.; Gu, L | NA | BP | 90 |
| Leu F.; Pai C. | NA | Statistical Approach | 94 |
| Alan Saied, Richard Overill, Tomasz Radzik | NA | ANN | 98 |
| Bhuvanesvari, Amma | NSLKDD And Unsw-Nb15 | NA | NA |
| Gupta and Kulariya | NSLKDD And UNSW-NB15 | LR, NB, SVM, RF, DT | Highest Achieved:  90.0% |
| Isna And Team | NA | Naive Bayes Classifier and Random Forest Classifier | NBC: 90.90%  RFC:78.18% |

Table 1: Comparison of Related Work

**Chapter 3**

**Definitions of the Technology**

**Support Vector Machine**

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of support vector machines are: Effective in high dimensional spaces. Still effective in cases where number of dimensions is greater than the number of samples.

The original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. In 1992, Bernhard Boser, Isabelle Guyon and Vladimir Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes. The "soft margin" incarnation, as is commonly used software packages, was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995.Support Vector Machine are perhaps one of the most popular and talked about machine learning algorithms.They were extremely popular around the time they were developed in the 1990s and continue to be the go-to method for a high performing algorithm with little tuning.

Support vector machines so called as SVM is a supervised learning algorithm which can be used for classification and regression problems as support vector classification (SVC) and support vector regression (SVR). It is used for smaller dataset as it takes too long to process. In this set, we will be focusing on SVC.

SVM is based on the idea of finding a hyperplane that best separates the features into different domains.

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.The points closest to the hyperplane are called as the support vector points and the distance of the vectors from the hyperplane are called the margins.

The basic intuition to develop over here is that more the farther SV points, from the hyperplane, more is the probability of correctly classifying the points in their respective region or classes. SV points are very critical in determining the hyperplane because if the position of the vectors changes the hyperplane’s position is altered. Technically this hyperplane can also be called as margin maximizing hyperplane.

The kernel is a way of computing the dot product of two vectors x and y in some (very high dimensional) feature space, which is why kernel functions are sometimes called generalized dot product.

Applying kernel trick means just to the replace dot product of two vectors by the kernel function.

Types of kernels:

* Linear Kernel
* Polynomial Kernel
* Radial basis function kernel (RBF)/ Gaussian Kernel

Pros:

* It is really effective in the higher dimension.
* Effective when the number of features are more than training examples.
* Best algorithm when classes are separable
* The hyperplane is affected by only the support vectors thus outliers have less impact.
* SVM is suited for extreme case binary classification.
* Effective in high dimensional spaces.
* Still effective in cases where number of dimensions is greater than the number of samples.
* Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
* Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

Cons:

* For larger dataset, it requires a large amount of time to process.
* Does not perform well in case of overlapped classes.
* Selecting, appropriately hyperparameters of the SVM that will allow for sufficient generalization performance.
* Selecting the appropriate kernel function can be tricky.
* If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
* SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

Preparing data for SVM:

1. Numerical Conversion:

SVM assumes that you have inputs are numerical instead of categorical. So you can convert them using one of the most commonly used “one hot encoding , label-encoding etc”.

2. Binary Conversion:

Since SVM is able to classify only binary data so you would need to convert the multi-dimensional dataset into binary form using (one vs the rest method / one vs one method) conversion method.

Scikit-Learn SVM

The support vector machines in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered numpy.ndarray (dense) or scipy.sparse.csr\_matrix (sparse) with dtype=float64.

**K Nearest Neighbours**

K-nearest Neighbor (KNN) is a supervised classification algorithm that is based on predicting data by finding the similarities to the underlying data. KNN is most widely used for classification problems, but can also be used to solve regression problems. KNN stands for K-nearest neighbour, it’s one of the Supervised learning algorithm mostly used for classification of data on the basis how it’s neighbour are classified. KNN stores all available cases and classifies new cases based on a similarity measure. K in KNN is a parameter that refers to the number of the nearest neighbours to include in the majority voting process.

To choose K we use Sqrt(n), where n is a total number of data points(if in case n is even we have to make the value odd by adding 1 or subtracting 1 that helps in select better)

We can use KNN when Dataset is labelled and noise-free and it’s must be small because KNN is a “Lazy learner”.

KNN is one of many supervised learning algorithms used in data mining and machine learning, it’s a classifier algorithm where the learning is based “how similar” is a data from other .

The KNN’s steps are:

1. Receive an unclassified data
2. Measure the distance (Euclidian, Manhattan, Minkowski or Weighted) from the new data to all others data that is already classified
3. Gets the K(K is a parameter that you difine) smaller distances
4. Check the list of classes had the shortest distance and count the amount of each class that appears
5. Takes as correct class the class that appeared the most times
6. Classifies the new data with the class that you took in step 5

PROS

* K-NN is pretty intuitive and simple: K-NN algorithm is very simple to understand and equally easy to implement. To classify the new data point K-NN algorithm reads through whole dataset to find out K nearest neighbors.
* K-NN has no assumptions: K-NN is a non-parametric algorithm which means there are assumptions to be met to implement K-NN. Parametric models like linear regression has lots of assumptions to be met by data before it can be implemented which is not the case with K-NN.
* Very easy to implement for multi-class problem: Most of the classifier algorithms are easy to implement for binary problems and needs effort to implement for multi class whereas K-NN adjust to multi class without any extra efforts.
* Can be used both for Classification and Regression: One of the biggest advantages of K-NN is that K-NN can be used both for classification and regression problems.
* One Hyper Parameter: K-NN might take some time while selecting the first hyper parameter but after that rest of the parameters are aligned to it.

CONS

* K-NN slow algorithm: K-NN might be very easy to implement but as dataset grows efficiency or speed of algorithm declines very fast.
* Curse of Dimensionality: KNN works well with small number of input variables but as the numbers of variables grow K-NN algorithm struggles to predict the output of new data point.
* K-NN needs homogeneous features: If you decide to build k-NN using a common distance, like Euclidean or Manhattan distances, it is completely necessary that features have the same scale, since absolute differences in features weight are the same.
* Optimal number of neighbors: One of the biggest issues with K-NN is to choose the optimal number of neighbors to be consider while classifying the new data entry.
* Imbalanced data causes problems: k-NN doesn’t perform well on imbalanced data. If we consider two classes, A and B, and the majority of the training data is labeled as A, then the model will ultimately give a lot of preference to A. This might result in getting the less common class B wrongly classified.
* Outlier sensitivity: K-NN algorithm is very sensitive to outliers as it simply chose the neighbors based on distance criteria.
* Missing Value treatment: K-NN inherently has no capability of dealing with missing value problem.

Scikit-Learn KNN

Training and testing on the same data is not an optimal approach, so we do split the data into two pieces, training set and testing set. We use ‘train\_test\_split’ function to split the data. Optional parameter ‘test-size’ determines the split percentage. ‘random\_state’ parameter makes the data split the same way every time you run. Since we are training and testing on different sets of data, the resulting testing accuracy will be a better estimate of how well the model is likely to perform on unseen data.

Scikit-learn is carefully organized into modules, so that we can import the relevant classes easily. Import the class ‘KNeighborsClassifer’ from ‘neighbors’ module and Instantiate the estimator (‘estimator’ is scikit-learn’s term for a model). We are calling model as estimator because their primary role is to estimate unknown quantities.

‘fit’ method is used to train the model on training data (X\_train,y\_train) and ‘predict’ method to do the testing on testing data (X\_test).

**Artficial Neural Network**

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specfic problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true for ANNs as well. Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an expert" in the category of information it has been given to analyze. This expert can then be used to provide projections of new situations of interest and answer "what if" questions. Learning in Neural Networks Learning in the context of neural networks is the process of adjusting the connection weights and biases such that for a given input a desired output is achieved. There are two basic training modes.

1. Supervised learning - This is a learning paradigm where the neural network is given samples of the input and desired output and the error between the desired output and the actual output of the neural network is used to adjust the connection weights. A famous algorithm of supervised learning is back propagation.
2. Unsupervised learning - This does not need any feedback for adjustment of the weights.

PROS

* Neural networks are flexible and can be used for both regression and classification problems. Any data which can be made numeric can be used in the model, as neural network is a mathematical model with approximation functions.
* Neural networks are good to model with nonlinear data with large number of inputs; for example, images. It is reliable in an approach of tasks involving many features. It works by splitting the problem of classification into a layered network of simpler elements.
* Once trained, the predictions are pretty fast.
* Neural networks can be trained with any number of inputs and layers.
* Neural networks work best with more data points.

CONS

* Neural networks are black boxes, meaning we cannot know how much each independent variable is influencing the dependent variables.
* It is computationally very expensive and time consuming to train with traditional CPUs.
* Neural networks depend a lot on training data. This leads to the problem of over-fitting and generalization. The mode relies more on the training data and may be tuned to the data.

Multilayer Perceptron (MLP) in Scikit-learn

* There is no activation function in the output layer.
* For regression scenarios, the square error is the loss function, and cross-entropy is the loss function for the classification
* It can work with single as well as multiple target values regression.
* Unlike other popular packages, likes Keras the implementation of MLP in Scikit doesn’t support GPU.
* We cannot fine-tune the parameters like different activation functions, weight initializers etc. for each layer.
* “MLPClassifier” is available for Multilayer Perceptron (MLP) classification scenarios

**Random Forest Classifier**

Random forests are a supervised Machine learning algorithm that is widely used in regression and classification problems and produces, even without hyperparameter tuning a great result most of the time. It is perhaps the most used algorithm because of its simplicity. It builds a number of decision trees on different samples and then takes the majority vote if it’s a classification problem.

It uses ensemble learning, that combines many weak classifiers to provide solutions to complex problems.

As the name suggests random forest consists of many decision trees. Rather than depending on one tree it takes the prediction from each tree and based on the majority votes of predictions, predicts the final output.

Random forests (RF) construct many individual decision trees at training. Predictions from all trees are pooled to make the final prediction; the mode of the classes for classification or the mean prediction for regression. As they use a collection of results to make a final decision, they are referred to as Ensemble techniques.

Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value the more important the feature.

Scikit-learn Random Forest

For each decision tree, Scikit-learn calculates a nodes importance using Gini Importance. These can then be normalized to a value between 0 and 1 by dividing by the sum of all feature importance values.

Decision trees use a flowchart like a tree structure to show the predictions that result from a series of feature-based splits. It starts with a root node and ends with a decision made by leaves.It consists of 3 components which are the root node, decision node, leaf node. The node from where the population starts dividing is called a root node. The nodes we get after splitting a root node are called decision nodes and the node where further splitting is not possible is called a leaf node.

To select a feature to split further we need to know how impure or pure that split will be. A pure sub-split means that either you should be getting “yes” or “no”.When a feature is taken as root node, we get a pure split whereas when we take feature 2, the split is not pure.“Gini Index” specifies the impurity in the nodes.

### Steps involved in Random Forest Algorithm

**Step-1** – We first make subsets of our original data. We will do row sampling and feature sampling that means we’ll select rows and columns with replacement and create subsets of the training dataset

**Step- 2** – We create an individual decision tree for each subset we take

**Step-3** – Each decision tree will give an output

**Step 4** – Final output is considered based on Majority Voting if it’s a classification problem and average if it’s a regression problem.

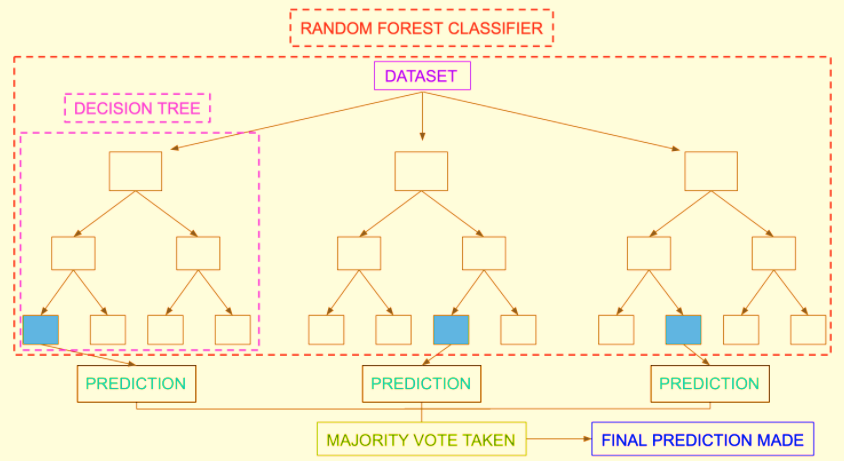


Figure 1: Random Forest Classifier

PROS

* One of the greatest benefits of a random forest algorithm is its flexibility. We can use this algorithm for regression as well as classification problems. It can be considered a handy algorithm because it produces better results even without hyperparameter tuning. Also, the parameters are pretty straightforward, they are easy to understand and there are also not that many of them.
* One of the biggest problems in machine learning is Overfitting. We need to make a generalized model which can get good results on the test data too. Random forest helps to overcome this situation by combining many Decision Trees which will eventually give us low bias and low variance.

CONS

* The main limitation of random forest is that due to a large number of trees the algorithm takes a long time to train which makes it slow and ineffective for real-time predictions. In general, these algorithms are fast to train but quite slow to create predictions once they are trained. In most real-world applications, the random forest algorithm is fast enough but there can certainly be situations where run-time performance is important and other approaches would be preferred.

**Extra Trees Classifier**

Extremely Randomized Trees, or Extra Trees for short, is an ensemble machine learning algorithm based on decision trees. The Extra Trees algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification. The predictions of the trees are aggregated to yield the final prediction, by majority vote in classification problems and arithmetic average in regression problems.

There are three main hyperparameters to tune in the algorithm; they are the number of decision trees in the ensemble, the number of input features to randomly select and consider for each split point, and the minimum number of samples required in a node to create a new split point.

The random selection of split points makes the decision trees in the ensemble less correlated, although this increases the variance of the algorithm. This increase in variance can be countered by increasing the number of trees used in the ensemble. The Extra Trees algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification.Extra Trees is an ensemble machine learning algorithm that combines the predictions from many decision trees.It is related to the widely used random forest algorithm. It can often achieve as-good or better performance than the random forest algorithm, although it uses a simpler algorithm to construct the decision trees used as members of the ensemble.

It is also easy to use given that it has few key hyperparameters and sensible heuristics for configuring these hyperparameters.Specifically, it is an ensemble of decision trees and is related to other ensembles of decision trees algorithms such as bootstrap aggregation (bagging) and random forest.

The Extra Trees algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification.

* Regression: Predictions made by averaging predictions from decision trees.
* Classification: Predictions made by majority voting from decision trees.

The Extra-Trees algorithm builds an ensemble of unpruned decision or regression trees according to the classical top-down procedure. Its two main differences with other tree-based ensemble methods are that it splits nodes by choosing cut-points fully at random and that it uses the whole learning sample (rather than a bootstrap replica) to grow the trees.

As such, there are three main hyperparameters to tune in the algorithm; they are the number of decision trees in the ensemble, the number of input features to randomly select and consider for each split point, and the minimum number of samples required in a node to create a new split point.

Each decision stump will be built with the following criteria:

1. All the data available in the training set is used to build each stump.
2. To form the root node or any node, the best split is determined by searching in a subset of randomly selected features of size sqrt (number of features). The split of each selected feature is chosen at random.
3. The maximum depth of the decision stump is one.

PRO

* It can be used for both classification and regression problems: Decision trees can be used to predict both continuous and discrete values
* It can capture nonlinear relationships: They can be used to classify non-linearly separable data.
* It does not require any transformation of the features if we are dealing with non-linear data because decision trees do not take multiple weighted combinations into account simultaneously.
* They are very fast and efficient compared to KNN and other classification algorithms.
* The data type of decision tree can handle any type of data whether it is numerical or categorical, or boolean.
* Normalization is not required in the Decision Tree.
* The decision tree is one of the machine learning algorithms where we don’t worry about its feature scaling.

CON

* The time complexity right for operating this operation is very huge keep on increasing as the number of records gets increased decision tree with to numerical variables takes a lot of time for training.
* Decision tree for many features take more time for training-time complexity to increase as the input increases.
* Method of overfitting: If we discuss overfitting, it is one of the most difficult methods for decision tree models. The overfitting problem can be solved by setting constraints on the parameters model and pruning method.
* A decision tree generally needs overfitting of data. In the overfitting problem, there is a very high variance in output which leads to many errors in the final estimation and can show highly inaccuracy in the output
* High variance which can be reduced by bagging.
* If the size of data is too big, then one single tree may grow a lot of nodes which might result in complexity and leads to overfitting.
* There is no guarantee to return the 100% efficient decision tree.

**Chapter 4**

**Application of ML and DL techniques for Problem Statement**

**Introduction**

Due to the huge flow of data and complications in mutable characteristics of the data, Distributed Denial ofServices attack are prevalent. Attacks over the networks have become an increasing menace in recent time, which tries to hack or illegally tamper the streaming data available overthe networks. On the other hand, there has been an increase in volume in research contributions to effectivelycounter these attacks and implement a strong defense mechanism.

There have been numerous algorithms and frameworks implemented in recent times that are intelligent and soft computing-based. These evolution-based algorithms play a vital role in self-adapting the system under attack towards increasing and new typesof attacks which are increasing day by day.

One such area of soft computing algorithms that needs to be investigated is the Artificial Neural Network or popularly known as ANNs. It works analogously to the biological neurons in the human body. It is systematically analyzed how the ANN-based network model counteract the DDoS attacks and the architecture, and implementation of ANNs, the experimental investigations and findings which help in drawing an inference of ANN-based defense models.

**Dataset Analysis and Preprocessing**

NSL-KDD is a data set suggested to solve some of the inherent problems of the KDD'99data set which are mentioned in . Although, this new version of the KDD dataset still suffers from some of the problems discussed by McHugh and may notbe a perfect representative of existing real networks, because of the lack of publicdata sets for network-based IDSs, we believe it still can be applied as an effectivebenchmark data set to help researchers compare different intrusion detection methods.Furthermore, the number of records in the NSL-KDD train and test sets are reasonable.This advantage makes it affordable to run the experiments on the complete set withoutthe need to randomly select a small portion. Consequently, evaluation results ofdifferent research work will be consistent and comparable.

The NSL-KDD data set has the following advantages over the original KDD data set: It does not include redundant records in the trainset, so the classifiers will not be biased towards more frequent records.

◾There is no duplicate records in the proposed testsets; therefore, the performance of the learners are not biased by the methods whichhave better detection rates on the frequent records.

◾The number of selected records from each difficultylevelgroup is inversely proportional to the percentage of records in the original KDDdata set. As a result, the classification rates of distinct machine learning methodsvary in a wider range, which makes it more efficient to have an accurate evaluationof different learning techniques.

◾The number of records in the train and test setsare reasonable, which makes it affordable to run the experiments on the completeset without the need to randomly select a small portion. Consequently, evaluationresults of different research works will be consistent and comparable.

One of the most important deficiencies in the KDD data set is the huge number ofredundant records, which causes the learning algorithms to be biased towards thefrequent records, and thus prevent them from learning unfrequent records which areusually more harmful to networks such as U2R and R2L attacks. In addition, the existenceof these repeated records in the test set will cause the evaluation results to bebiased by the methods which have better detection rates on the frequent records.

* Statistics of redundant records in the KDD train set:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Data | Train Data | Total |
| Attacks | 12833 | 58630 | 71463 |
| Normal | 9711 | 67343 | 77054 |
| Total | 22544 | 125973 | 148517 |

Table 2: Distribution of DATA in KDD train DataSet

**Preprocessing of Data and Extensive Data Analysis**

The dataset needs to be processed before being implemented on the algorithms, the raw data contains values that are non-integers. These headers need to be formatted to make the input data integer only.

The dataset originally contains 43 Headers:

"duration","protocol\_type","service","flag","src\_bytes","dst\_bytes","land","wrong\_fragment","urgent","hot", "num\_failed\_logins","logged\_in","num\_compromised","root\_shell","su\_attempted","num\_root","num\_file\_creations","num\_shells","num\_access\_files","num\_outbound\_cmds","is\_host\_login","is\_guest\_login","count","srv\_count","serror\_rate","srv\_serror\_rate","rerror\_rate","srv\_rerror\_rate","same\_srv\_rate","diff\_srv\_rate","srv\_diff\_host\_rate","dst\_host\_count","dst\_host\_srv\_count", "dst\_host\_same\_srv\_rate","dst\_host\_diff\_srv\_rate","dst\_host\_same\_src\_port\_rate","dst\_host\_srv\_diff\_host\_rate","dst\_host\_serror\_rate","dst\_host\_srv\_serror\_rate","dst\_host\_rerror\_rate","dst\_host\_srv\_rerror\_rate","label","difficulty"

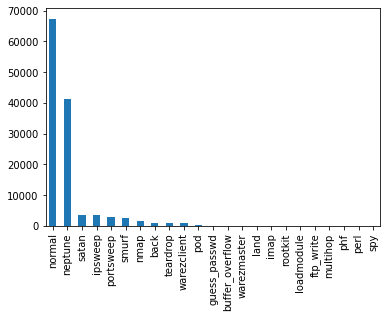


Figure 2: Distribution of different attack packets in the dataset

The test data has label values:

'neptune', 'normal', 'saint', 'mscan', 'guess\_passwd', 'smurf','apache2', 'satan', 'buffer\_overflow', 'back', 'warezmaster', 'snmpgetattack', 'processtable', 'pod', 'httptunnel', 'nmap', ps','snmpguess', 'ipsweep', 'mailbomb', 'portsweep', 'multihop', 'named', 'sendmail', 'loadmodule', 'xterm', 'worm', 'teardrop', 'rootkit', 'xlock', 'perl', 'land', 'xsnoop', 'sqlattack', 'ftp\_write', 'imap', 'udpstorm', 'phf’

The train data has label values:

'normal', 'neptune', 'warezclient', 'ipsweep', 'portsweep','teardrop', 'nmap', 'satan', 'smurf', 'pod', 'back','guess\_passwd', 'ftp\_write', 'multihop', 'rootkit','buffer\_overflow', 'imap', 'warezmaster', 'phf', 'land','loadmodule', 'spy', 'perl'

The label values are replaced with 0 if the packet is normal or with 1 if the packet is an intrusion attempt.

The protocol header has values:

'tcp', 'udp', 'icmp'

The service header has values:

'ftp\_data', 'other', 'private', 'http', 'remote\_job', 'name','netbios\_ns', 'eco\_i', 'mtp', 'telnet', 'finger', 'domain\_u','supdup', 'uucp\_path', 'Z39\_50', 'smtp', 'csnet\_ns', 'uucp','netbios\_dgm', 'urp\_i', 'auth', 'domain', 'ftp', 'bgp', 'ldap','ecr\_i', 'gopher', 'vmnet', 'systat', 'http\_443', 'efs', 'whois','imap4', 'iso\_tsap', 'echo', 'klogin', 'link', 'sunrpc', 'login','kshell', 'sql\_net', 'time', 'hostnames', 'exec', 'ntp\_u','discard', 'nntp', 'courier', 'ctf', 'ssh', 'daytime', 'shell','netstat', 'pop\_3', 'nnsp', 'IRC', 'pop\_2', 'printer', 'tim\_i', 'pm\_dump', 'red\_i', 'netbios\_ssn', 'rje', 'X11', 'urh\_i','http\_8001', 'aol', 'http\_2784', 'tftp\_u', 'harvest'

The flag header has values:

'SF', 'S0', 'REJ', 'RSTR', 'SH', 'RSTO', 'S1', 'RSTOS0', 'S3','S2', 'OTH'

The pandas function of get\_dummies() is used to replace the headers with dummy headers to replace the text values to get a binary matrix with columns denoting true or false. Figure 3: Visual Representation of Packet Label Distribution

This introduces new columns that modifies the shape of the data set.

Test data: (22544, 118)

Train data: (125973, 124)

These are of inconsistent shapes, the data needs to be of the same shape so we modify the data to hold all the common headers.

Test Data: (22544, 118)

Train Data: (125973, 118)

The data sets now are of equal shape, but they still hold the label header, which needs to be separated as the Y input. The column ’Label’ is stored into the Y variables and the dropped from the original data set.

Train Data has shape: (125973, 117)

Test Data has shape: (22544, 117)

Raw Data Shape

train has shape: (125973, 43)

test has shape: (22544, 43)

Data Shape after dummy row introduction

train has shape: (125973, 124)

test has shape: (22544, 118)

Data Shape after making shape uniform

train has shape: (125973, 118)

test has shape: (22544, 118)

Shape of Data after column label is separated

train has shape: (125973, 117)

test has shape: (22544, 117)

X\_train has shape: (125973, 117)

y\_train has shape: (125973,)

X\_test has shape: (22544, 117)

y\_test has shape: (22544,)

Figure 4: Output of Preprocessing.ipynb

**Support Vector Machine**

The preprocessing Notebook is run and the processed data is used to train the SVM and the Validation dataset is used to calculate the validation accuracies. The test data set is used to calculate the test accuracies and the evaluation function is called in both the cases. The results are recoded. The SVM is first configured with a Linear Kernel and then a polynomial kernel is used to compare the differneces and evaluate the best model.

SVM Parameters:

class sklearn.svm.SVC(\*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True,probability=False, tol=0.001, cache\_size=200, class\_weight=None, verbose=False,max\_iter=-1,decision\_function\_shape='ovr',break\_ties=False, random\_state=None)

* SVM Linear Kernel: parameter kernel = “linear”

╒═════════════════════════╤════════════╕

│ Mean Absolute Error │ 0.0100286 │

├─────────────────────────┼────────────┤

│ Mean Squared Error │ 0.0100286 │

├─────────────────────────┼────────────┤

│ Root Mean Squared Error │ 0.100143 │

├─────────────────────────┼────────────┤

│ R2 Score │ 95.9737 │

├─────────────────────────┼────────────┤

│ F1 Score │ 0.989254 │

├─────────────────────────┼────────────┤

│ Recall Score │ 0.986959 │

├─────────────────────────┼────────────┤

│ Accuracy │ 98.9971 │

╘═════════════════════════╧════════════╛

Table 3: SVM linear kernel validation performance

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.187944 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.187944 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.433525 │

├─────────────────────────┼───────────┤

│ R2 Score │ 29.6187 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.81611 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.921042 │

├─────────────────────────┼───────────┤

│ Accuracy │ 81.2056 │

╘═════════════════════════╧═══════════╛

Table 4: SVM linear kernel performance test performance

SVM Quadratic Kernel: parameter kernel =”poly”

╒═════════════════════════╤═════════════╕

│ Mean Absolute Error │ 0.00531859 │

├─────────────────────────┼─────────────┤

│ Mean Squared Error │ 0.00531859 │

├─────────────────────────┼─────────────┤

│ Root Mean Squared Error │ 0.0729286 │

├─────────────────────────┼─────────────┤

│ R2 Score │ 97.8633 │

├─────────────────────────┼─────────────┤

│ F1 Score │ 0.994293 │

├─────────────────────────┼─────────────┤

│ Recall Score │ 0.993306 │

├─────────────────────────┼─────────────┤

│ Accuracy │ 99.4681 │

╘═════════════════════════╧═════════════╛

Table 5: SVM Quadratic kernel validation preformance

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.189984 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.189984 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.435872 │

├─────────────────────────┼───────────┤

│ R2 Score │ 29.3076 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.812732 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.925882 │

├─────────────────────────┼───────────┤

│ Accuracy │ 81.0016 │

╘═════════════════════════╧═══════════╛

Table 6: SVM Quadratic kernel test performance

**K-Nearest Neighbors**

The K-Nearest Neighbors algorithm is used with different number of Neighbors to evaluate the optimal K value which was found to be in the proximity of 500.

The KNN algorithm is also validated on the Validation data Set and on the Test DataSet. KNN is configured with 20 Neighbours and the validation and the test runs are performed and the evaluation results are obtained. The classification report is also generated for comparision.

class sklearn.neighbors.KNeighborsClassifier(n\_neighbors=5, \*, weights='uniform', algorithm='auto', leaf\_size=30, p=2, metric='minkowski', metric\_params=None, n\_jobs=None)

KNN 20 Neighbours

Here we specify the Parameter n\_neighbors=20.

Classification Report

precision recall f1-score support

Normal 0.99 0.99 0.99 16774

Attack 0.99 0.99 0.99 14720

accuracy 0.99 31494

macro avg 0.99 0.99 0.99 31494

weighted avg 0.99 0.99 0.99 31494

Table 7: KNN 20 Neighbors Classification Report for validation

╒═════════════════════════╤═════════════╕

│ Mean Absolute Error │ 0.00666794 │

├─────────────────────────┼─────────────┤

│ Mean Squared Error │ 0.00666794 │

├─────────────────────────┼─────────────┤

│ Root Mean Squared Error │ 0.0816574 │

├─────────────────────────┼─────────────┤

│ R2 Score │ 97.3214 │

├─────────────────────────┼─────────────┤

│ F1 Score │ 0.992868 │

├─────────────────────────┼─────────────┤

│ Recall Score │ 0.993071 │

├─────────────────────────┼─────────────┤

│ Accuracy │ 99.3332 │

╘═════════════════════════╧═════════════╛

Table 8: KNN 20 Neighbors validation evaluation

Classification Report

precision recall f1-score support

Normal 0.66 0.97 0.79 9711

Attack 0.97 0.62 0.75 12833

accuracy 0.77 22544

macro avg 0.81 0.80 0.77 22544

weighted avg 0.84 0.77 0.77 22544

Table 9: KNN 20 Neighbors Classification Report for test

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.229019 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.229019 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.478559 │

├─────────────────────────┼───────────┤

│ R2 Score │ 23.9892 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.754272 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.617471 │

├─────────────────────────┼───────────┤

│ Accuracy │ 77.0981 │

╘═════════════════════════╧═══════════╛

Table 10: KNN 20 Neighbors test evaluation

KNN is then configured with 500 Neighbours and the validation and the test runs are performed and the evaluation results and classification report is generated and compared with the results of KNN with 20 neighbors. The results show an increase in the performance.

KNN 500 Neighbours

Here we specify the Parameter n\_neighbors=500.

Classification Report

precision recall f1-score support

Normal 0.97 0.96 0.96 16774

Attack 0.95 0.97 0.96 14720

accuracy 0.96 31494

macro avg 0.96 0.96 0.96 31494

weighted avg 0.96 0.96 0.96 31494

Table 11: KNN 500 Neighbors Classification Report for validation

╒═════════════════════════╤════════════╕

│ Mean Absolute Error │ 0.0374357 │

├─────────────────────────┼────────────┤

│ Mean Squared Error │ 0.0374357 │

├─────────────────────────┼────────────┤

│ Root Mean Squared Error │ 0.193483 │

├─────────────────────────┼────────────┤

│ R2 Score │ 84.9845 │

├─────────────────────────┼────────────┤

│ F1 Score │ 0.960272 │

├─────────────────────────┼────────────┤

│ Recall Score │ 0.968003 │

├─────────────────────────┼────────────┤

│ Accuracy │ 96.2564 │

╘═════════════════════════╧════════════╛

Table 12: KNN 500 Neighbors validation evaluation

Classification Report

precision recall f1-score support

Normal 0.68 0.96 0.80 9711

Attack 0.95 0.67 0.78 12833

accuracy 0.79 22544

macro avg 0.82 0.81 0.79 22544

weighted avg 0.84 0.79 0.79 22544

Table 13: KNN 500 Neighbors Classification Report

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.208614 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.208614 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.456743 │

├─────────────────────────┼───────────┤

│ R2 Score │ 26.9841 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.784157 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.665706 │

├─────────────────────────┼───────────┤

│ Accuracy │ 79.1386 │

╘═════════════════════════╧═══════════╛

Table 14: KNN 500 Neighbors test evaluation

As the KNN configured with 500 Neighbours showed better performance then the one with 20 Neighbors, we increase the number of neighbors to 2000 and the validation and the test runs are performed and the evaluation results are obtained. The classification report is also generated for comparision. But the evaluation shows that the efficiency reduces with increase in the number of neighbors. And the optimal number of neighbors are estimated to be around 500.

KNN 2000 neighbours

Here we specify the Parameter n\_neighbors=2000.

Classification Report

precision recall f1-score support

Normal 0.94 0.95 0.94 16774

Attack 0.94 0.93 0.93 14720

accuracy 0.94 31494

macro avg 0.94 0.94 0.94 31494

weighted avg 0.94 0.94 0.94 31494

Table 15: KNN 2000 Neighbors Classification Report

╒═════════════════════════╤════════════╕

│ Mean Absolute Error │ 0.0609957 │

├─────────────────────────┼────────────┤

│ Mean Squared Error │ 0.0609957 │

├─────────────────────────┼────────────┤

│ Root Mean Squared Error │ 0.246973 │

├─────────────────────────┼────────────┤

│ R2 Score │ 75.5162 │

├─────────────────────────┼────────────┤

│ F1 Score │ 0.934269 │

├─────────────────────────┼────────────┤

│ Recall Score │ 0.927446 │

├─────────────────────────┼────────────┤

│ Accuracy │ 93.9004 │

╘═════════════════════════╧════════════╛

Table 16: KNN 2000 Neighbors validation evaluation

Classification Report

precision recall f1-score support

Normal 0.65 0.96 0.77 9711

Attack 0.95 0.60 0.74 12833

accuracy 0.76 22544

macro avg 0.80 0.78 0.75 22544

weighted avg 0.82 0.76 0.75 22544

Table 17: KNN 2000 Neighbors Classification Report

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.244855 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.244855 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.494828 │

├─────────────────────────┼───────────┤

│ R2 Score │ 17.5684 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.737218 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.603366 │

├─────────────────────────┼───────────┤

│ Accuracy │ 75.5145 │

╘═════════════════════════╧═══════════╛

Table 18: KNN 2000 Neighbors test evaluation

**Neural Network**

NN is configured with 3 layers and the validation and the test runs are performed and the evaluation results are obtained. The evaluation is calculated on the validation and test data set and the ReLU activation efficiency is calculated with 10 Epochs.

class sklearn.neural\_network.MLPClassifier ( hidden\_layer\_sizes = (100,), activation = 'relu', \* , solver='adam' , alpha=0.0001, batch\_size='auto' , learning\_rate = 'constant' , learning\_rate\_init = 0.001, power\_t = 0.5, max\_iter = 200, shuffle = True, random\_state = None, tol = 0.0001, verbose = False, warm\_start = False, momentum=0.9, nesterovs\_momentum = True, early\_stopping = False, validation\_fraction = 0.1, beta\_1 = 0.9, beta\_2 = 0.999, epsilon = 1e-08, n\_iter\_no\_change = 10, max\_fun=15000)

╒═════════════════════════╤════════════╕

│ Mean Absolute Error │ 0.0150429 │

├─────────────────────────┼────────────┤

│ Mean Squared Error │ 0.0150429 │

├─────────────────────────┼────────────┤

│ Root Mean Squared Error │ 0.122649 │

├─────────────────────────┼────────────┤

│ R2 Score │ 93.9803 │

├─────────────────────────┼────────────┤

│ F1 Score │ 0.984015 │

├─────────────────────────┼────────────┤

│ Recall Score │ 0.992231 │

├─────────────────────────┼────────────┤

│ Accuracy │ 98.4957 │

╘═════════════════════════╧════════════╛

Table 19: NN Validation Set Evaluation Results

Here we specify the parameter activation is “relu”, the solver is “adam” and 3 layers specified and number of epochs = “10”

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.226535 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.226535 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.475957 │

├─────────────────────────┼───────────┤

│ R2 Score │ 18.9527 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.766878 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.654562 │

├─────────────────────────┼───────────┤

│ Accuracy │ 77.3465 │

╘═════════════════════════╧═══════════╛

Table 20: NN Test Evaluation Results

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.430758 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.430758 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.656321 │

├─────────────────────────┼───────────┤

│ R2 Score │ 0 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.7255 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.569242 │

├─────────────────────────┼───────────┤

│ Accuracy │ 56.9242 │

╘═════════════════════════╧═══════════╛

Table 21: NN evaluation with ReLU activation

The performance of the MLP Model is found to be inconsistent and the efficiency is less than the acceptable values and hence will be only used for comparison.

**Extra Trees Classifier**

The preprocessing notebook is run to prepare the data, the test and train datasets are used to validate and test the model and the evaluation of efficiency is noted.

class sklearn.ensemble.ExtraTreesClassifier(n\_estimators=100, \*, criterion='gini', max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features='auto', max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, bootstrap=False, oob\_score=False, n\_jobs=None, random\_state=None, verbose=0, warm\_start=False, class\_weight=None, ccp\_alpha=0.0, max\_samples=None)

In this model, we specify the parameter n\_estimators=100 and criterion as “gini”

╒═════════════════════════╤══════════════╕

│ Mean Absolute Error │ 0.000793802 │

├─────────────────────────┼──────────────┤

│ Mean Squared Error │ 0.000793802 │

├─────────────────────────┼──────────────┤

│ Root Mean Squared Error │ 0.0281745 │

├─────────────────────────┼──────────────┤

│ R2 Score │ 99.6812 │

├─────────────────────────┼──────────────┤

│ F1 Score │ 0.99915 │

├─────────────────────────┼──────────────┤

│ Recall Score │ 0.998777 │

├─────────────────────────┼──────────────┤

│ Accuracy │ 99.9206 │

╘═════════════════════════╧══════════════╛

Table 22: Extra Trees Classifier validation evaluation

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.204578 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.204578 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.452303 │

├─────────────────────────┼───────────┤

│ R2 Score │ 27.6351 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.786402 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.969289 │

├─────────────────────────┼───────────┤

│ Accuracy │ 79.5422 │

╘═════════════════════════╧═══════════╛

Table 23: Extra Trees Classifier test evaluation

The dataset is then dimensionally reduced based on the best features using Principal Component Analysis where the dimensions are reduced to 10 best features of the dataset. The resulting PCA datasets are used for further analysis.

Feature Selection is done to assess the impact of the features of the dataset.

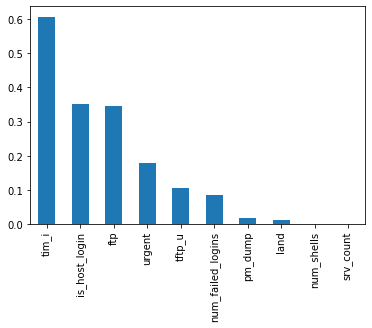


Figure 4: The 10 best features of the train dataset

The PCA output data sets are in the shape of:

test PCA has shape: (22544, 10)

train PCA has shape: (125973, 10)

The dimensionally reduced PCA is split to Validation Set as:

X\_train PCA has shape: (94479, 10)

y\_train PCA has shape: (94479,)

X\_val PCA has shape: (31494, 10)

y\_val PCA has shape: (31494,)

classsklearn.decomposition.PCA(n\_components=None, \*, copy=True, whiten=False, svd\_solver='auto', tol=0.0, iterated\_power='auto', random\_state=None)

Here for the Dimensionality Reduction we specify the parameter n\_components = 10.

╒═════════════════════════╤═════════════╕

│ Mean Absolute Error │ 0.00247666 │

├─────────────────────────┼─────────────┤

│ Mean Squared Error │ 0.00247666 │

├─────────────────────────┼─────────────┤

│ Root Mean Squared Error │ 0.0497661 │

├─────────────────────────┼─────────────┤

│ R2 Score │ 99.0053 │

├─────────────────────────┼─────────────┤

│ F1 Score │ 0.997352 │

├─────────────────────────┼─────────────┤

│ Recall Score │ 0.99803 │

├─────────────────────────┼─────────────┤

│ Accuracy │ 99.7523 │

╘═════════════════════════╧═════════════╛

Table 24: Extra Trees Classifier validation evaluation with PCA

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.170644 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.170644 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.413091 │

├─────────────────────────┼───────────┤

│ R2 Score │ 38.9723 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.828251 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.722824 │

├─────────────────────────┼───────────┤

│ Accuracy │ 82.9356 │

╘═════════════════════════╧═══════════╛

Table 25: Extra Trees Classifier test evaluation with PCA

**Random Forest Classifier**

The Preprocessing Notebook is called to prepare the datasets, and the validation and test datasets are used to evaluate the prediction accuracies. And the evaluation results are noted for comparision.

class sklearn.ensemble.RandomForestClassifier(n\_estimators=100, \*, criterion='gini', max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features='auto', max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, bootstrap=True, oob\_score=False, n\_jobs=None, random\_state=None, verbose=0, warm\_start=False, class\_weight=None, ccp\_alpha=0.0, max\_samples=None)

Here we specify the parameters as ccp\_alpha=0, criterion='entropy', max\_depth=10, max\_features='log2',max\_leaf\_nodes=100,max\_samples=10000,min\_impurity\_decrease=0,min\_samples\_leaf=10,min\_weight\_fraction\_leaf=0,n\_estimators=50, n\_jobs=5, oob\_score=True, random\_state=10, warm\_start=True

╒═════════════════════════╤════════════╕

│ Mean Absolute Error │ 0.0112402 │

├─────────────────────────┼────────────┤

│ Mean Squared Error │ 0.0112402 │

├─────────────────────────┼────────────┤

│ Root Mean Squared Error │ 0.10602 │

├─────────────────────────┼────────────┤

│ R2 Score │ 95.4952 │

├─────────────────────────┼────────────┤

│ F1 Score │ 0.987752 │

├─────────────────────────┼────────────┤

│ Recall Score │ 0.980761 │

├─────────────────────────┼────────────┤

│ Accuracy │ 98.876 │

╘═════════════════════════╧════════════╛

Table 26: Random Forest Classifier validation evaluation

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.217308 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.217308 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.466163 │

├─────────────────────────┼───────────┤

│ R2 Score │ 26.5541 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.770183 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.639679 │

├─────────────────────────┼───────────┤

│ Accuracy │ 78.2692 │

╘═════════════════════════╧═══════════╛

Table 27: Random Forest Classifier test evaluation

The PCA.ipynb file is run to apply Dimensionaltiy reduction on the data sets with n\_components set as 10. The PCA datasets areused to analyze the performance of Random Forest Classifier and the performance evaluation results are recorded.

The PCA output data sets are in the shape of:

test PCA has shape: (22544, 10)

train PCA has shape: (125973, 10)

The dimensionally reduced PCA is split to Validation Set as:

X\_train PCA has shape: (94479, 10)

y\_train PCA has shape: (94479,)

X\_val PCA has shape: (31494, 10)

y\_val PCA has shape: (31494,)

classsklearn.decomposition.PCA(n\_components=None, \*, copy=True, whiten=False, svd\_solver='auto', tol=0.0, iterated\_power='auto', random\_state=None)

Here for the Dimensionality Reduction we specify the parameter n\_components = 10.

╒═════════════════════════╤════════════╕

│ Mean Absolute Error │ 0.0136534 │

├─────────────────────────┼────────────┤

│ Mean Squared Error │ 0.0136534 │

├─────────────────────────┼────────────┤

│ Root Mean Squared Error │ 0.116848 │

├─────────────────────────┼────────────┤

│ R2 Score │ 94.5213 │

├─────────────────────────┼────────────┤

│ F1 Score │ 0.985454 │

├─────────────────────────┼────────────┤

│ Recall Score │ 0.989538 │

├─────────────────────────┼────────────┤

│ Accuracy │ 98.6347 │

╘═════════════════════════╧════════════╛

Table 28: Random Forest Classifier validation evaluation with PCA

╒═════════════════════════╤═══════════╕

│ Mean Absolute Error │ 0.124778 │

├─────────────────────────┼───────────┤

│ Mean Squared Error │ 0.124778 │

├─────────────────────────┼───────────┤

│ Root Mean Squared Error │ 0.35324 │

├─────────────────────────┼───────────┤

│ R2 Score │ 53.0213 │

├─────────────────────────┼───────────┤

│ F1 Score │ 0.880089 │

├─────────────────────────┼───────────┤

│ Recall Score │ 0.804411 │

├─────────────────────────┼───────────┤

│ Accuracy │ 87.5222 │

╘═════════════════════════╧═══════════╛

Table 29: Random Forest Classifier test evaluation with PCA

**Chapter 5**

**Comparison of Performance of the ML and DL algorithms**

**Performance Evaluation Functions**

To analyze the performance of the models we use sklearn.metrics library to call the functions to calculate Mean Absolute Error, Mean Squared Error, Mean Squared Root Error, R2 Score, F1 Score, Recall and Accuracy.

* Mean Absolute Error: Mean Absolute Error calculates the average difference between the calculated values and actual values. It is also known as scale-dependent accuracy as it calculates error in observations taken on the same scale. It is used as evaluation metrics for regression models in machine learning

sklearn.metrics.mean\_absolute\_error (y\_true, y\_pred)

* Mean Squared Error: The Mean Squared Error (MSE) is a measure of how close a fitted line is to data points. For every data point, you take the distance vertically from the point to the corresponding y value on the curve fit (the error), and square the value.

sklearn.metrics.mean\_squared\_error (y\_true, y\_pred)

* Root Mean Squared Error: The root mean squared error (RMSE) is used to measured the differences between values predicted by the model and observed values of the model. The root mean squared error (RMSE) is always non-negative, RMSE value near to 0 indicates a perfect fit to the data.

numpy.sqrt (sklearn.metrics.mean\_squared\_error (y\_true, y\_pred))

* R2 Score: R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model.

sklearn.metrics.explained\_variance\_score (y\_test, y\_pred)\*100

* F1 Score: F1 score of the positive class in binary classification or weighted average of the F1 scores of each class for the multiclass task. When true positive + false positive == 0 , precision is undefined.

When true positive + false negative == 0 , recall is undefined.

sklearn.metrics.f1\_score (y\_test, y\_pred)

* Recall: The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1 and the worst value is 0.

sklearn.metrics.recall\_score (y\_true, y\_pred)

* Accuracy: The accuracy\_score method is used to calculate the accuracy of either the faction or count of correct prediction in Python Scikit learn. Mathematically it represents the ratio of the sum of true positives and true negatives out of all the predictions

sklearn.metrics.accuracy\_score (y\_true, y\_pred)

def evaluation(y\_test, y\_pred):

    eMEA= metrics.mean\_absolute\_error(y\_test, y\_pred)

    MSE= metrics.mean\_squared\_error(y\_test, y\_pred)

    MSRE= np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

    R2= metrics.explained\_variance\_score(y\_test, y\_pred)\*100

    F1= metrics.f1\_score(y\_test,y\_pred)

    Recall= metrics.recall\_score(y\_test,y\_pred)

    accuracy = accuracy\_score(y\_test,y\_pred)\*100

    data = [["Mean Absolute Error",eMEA],["Mean Squared Error",MSE],["Root Mean Squared Error",MSRE],["R2 Score",R2],["F1 Score",F1],["Recall Score",Recall],["Accuracy",accuracy]]

    print (tabulate(data, tablefmt='fancy\_grid'))

    print('\n')

Figure 5: Evaluation Function

**Performance of the different models**

The results of the performance is consolidated by calling the different notebooks in analysis.ipynb and the accuracies are made note of and compared. We can see here that the performance of random forest is higher than the rest of the methods used. Here we consider the highest performance of the model on the test data set.

|  |  |
| --- | --- |
| Model | Accuracy |
| LSVM Validation | 98.9971 |
| LSVM Test | 81.2056 |
| QSVM Validation | 99.4681 |
| QSVM Test | 81.0016 |
| KNN 20 Validation | 99.3332 |
| KNN 20 Test | 77.0981 |
| KNN 500 Validation | 96.2564 |
| KNN 500 Test | 79.1386 |
| KNN 2000 Validation | 93.9004 |
| KNN 2000 Test | 75.5145 |
| NN Validation | 98.4957 |
| NN Test | 77.3465 |
| NN ReLU | 56.9242 |
| RF Validation | 98.876 |
| RF Test | 78.2692 |
| RF PCA Validation | 98.6347 |
| RF PCA Test | 87.5222 |
| ETC Validation | 99.9206 |
| ETC Test | 79.5422 |
| ETC PCA Validation | 99.7523 |
| ETC PCA Test | 82.9356 |

Table 30: All recorded Evaluation Results

The above table shows the performance of all the different models on the test dataset. We can see that the RandomForest Classifier working on the PCA dataset performs the best compared to the other methods.

Max Test Accuracy:

[87.5221788502484]

Figure 6: Max Test Accuracy Achieved

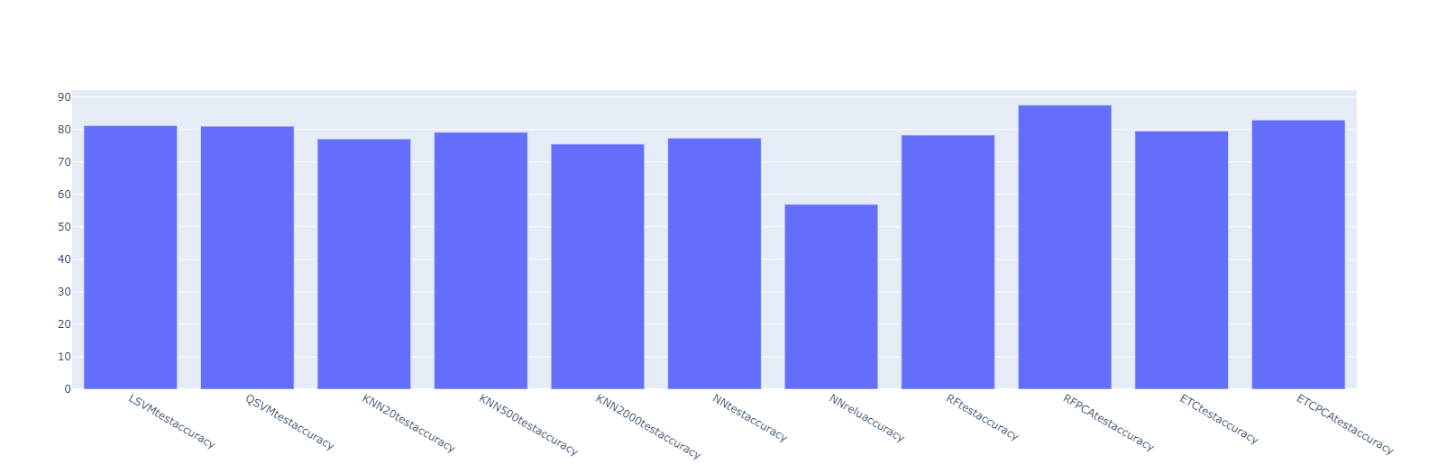
****

Figure 7: Bar Graph with Test Dataset Accuracies

**Chapter 6**

**Application and Performance Evaluation of the Proposed Solution**

**INTRODUCTION**

The evaluation section showed that the Random Forest Classifier showed the most promising solution that can be applied to the Intrusion Detection System, Once the model is trained on the Train Dataset, the trained model can be used to predict the incoming packet traffic and provide real time protection against cyber attack. The model can be further improved by training on much diverse datasets and improving the accuracy of the algorithm.

In this study, we have implemented 5 types of machine learning classifiers such as Random forest, KNN, SVM, MLP and ETC. The experiment was conducted on well known dataset NSL KDD dataset. The dataset contains normal and different kinds of attack types. Out of 5 classifications techniques, Random Forest performed well with an accuracy of 99.34% on train data and 86.17% on test data. Also, we have done feature preprocessing technique in which we have dropped some features which have low importance.

The Proposed Solution implements an Intrusion Detection System that has Random Forest Classifier as its core, the model is trained on an extensive and updated dataset. The trained model predicts the incoming packet as either attack or normal and provides real time protection of the network by allowing the normal packets to communicate and raise an alert when an intrusion is detected and withholding the connection with the sender. The model may also be configured to recursively train itself based on the predicted outcomes in a supervised manner.

The below flowchart depicts the proposed algorithm for the Intrusion Detection System

KDD Cup DataSet

Data PreProcessing

Random Forest Classifier

fit()

Trained Model

Prediction

model.predict()

Figure 8: Proposed Solution Algorithm

**Performance of the Proposed Solution**

The proposed solution provided the below results on the single packets that were input. The model provided similar results for both normal and attack packets.

╒═════════════════════════╤═════╕

│ Mean Absolute Error │ 0 │

├─────────────────────────┼─────┤

│ Mean Squared Error │ 0 │

├─────────────────────────┼─────┤

│ Root Mean Squared Error │ 0 │

├─────────────────────────┼─────┤

│ R2 Score │ 100 │

├─────────────────────────┼─────┤

│ F1 Score │ 0 │

├─────────────────────────┼─────┤

│ Recall Score │ 0 │

├─────────────────────────┼─────┤

│ Accuracy │ 100 │

╘═════════════════════════╧═════╛

Table 32: Proposed Solution Algorithm Performance

The perfect scores are due to the fact that the comparison takes place for one instance, hence the result is either correct or wrong. The solution was predicted accurately for most of the test cases run. The algorithm shows deviation from accuracy when the input data is not accurate, or if there are fields of packet information missing.

**Chapter 7**

**Conclusion**

**Conclusion and Future Work**

The work done here can be further improved by further experimentation by experimenting further with different DL/ML algorithms and processing the data and using different DataSets and further enhancing the model and achieving a higher accuracy. We can also work towards improving the prediction times to further suit the real-time necessities of the application in question. The proposed Intrusion System can be developed to implement protection against the real-time attacks.

Single Packet

Data Preprocessing

Trained Random Forest Classifier Predict ()

Allow Traffic

Block Traffic

Figure 9: Proposed Intrusion detection System Algorithm

**Appendix**

List of Figures

Figure 1- DDoS Structure 00

Figure 2- Random Forest Classifier 00

Figure 3- Distribution of different attack packets in the dataset 00

Figure 4- Visual Representation of Packet Label Distribution 00

Figure 5- Output of Preprocessing.ipynb 00

Figure 6- Evaluation Function 00

Figure 7- Max Test Accuracy Achieved 00

Figure 8- Bar Graph with Test Dataset Accuracies 00

Figure 9- Proposed Solution Algorithm 00

Figure 10- Proposed Intrusion detection System Algorithm 00

# 

# List of Tables

Table 1: Comparison of Related Work 24

Table 2: Distribution of DATA in KDD train DataSet 39

Table 3: SVM linear kernel validation performance 44

Table 4: SVM linear kernel performance test performance 44

Table 5: SVM Quadratic kernel validation preformance 45

Table 6: SVM Quadratic kernel test preformance 45

Table 7: KNN 20 Neighbors Classification Report for validation 46

Table 8: KNN 20 Neighbors validation evaluation 46

Table 9: KNN 20 Neighbors Classification Report for test 47

Table 10: KNN 20 Neighbors test evaluation 47

Table 11: KNN 500 Neighbors Classification Report for validation 48

Table 12: KNN 500 Neighbors validation evaluation 48

Table 13: KNN 500 Neighbors Classification Report 49

Table 14: KNN 500 Neighbors test evaluation 49

Table 15: KNN 2000 Neighbors Classification Report 50

Table 16: KNN 2000 Neighbors validation evaluation 50

Table 17: KNN 20 Neighbors Classification Report 50

Table 18: KNN 2000 Neighbors test evaluation 51

Table 19: NN Validation Set Evaluation Results 51

Table 20: NN Test Evaluation Results 52

Table 21: NN evaluation with ReLU activation 53

Table 22: Extra Trees Classifier validation evaluation 54

Table 23: Extra Trees Classifier test evaluation 54

Table 24: Extra Trees Classifier validation evaluation with PCA 56

Table 25: Extra Trees Classifier test evaluation with PCA 56

Table 26: Random Forest Classifier validation evaluation 57

Table 27: Random Forest Classifier test evaluation 58

Table 28: Random Forest Classifier validation evaluation with PCA 59

Table 29: Random Forest Classifier test evaluation with PCA 59

Table 30: All recorded Evaluation Results 62

Table 31: Proposed Solution Algorithm Performance 66

**Program Codes**

**The Program Codes can be found here**

[**https://github.com/vkvinay580/IDS-with-ML-DL**](https://github.com/vkvinay580/IDS-with-ML-DL)

Requirements.ipynb

import numpy as np

import pandas as pd

import pickle

from os import path

from sklearn import preprocessing

from sklearn.preprocessing import (StandardScaler, OrdinalEncoder,LabelEncoder, MinMaxScaler, OneHotEncoder)

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.preprocessing import Normalizer, MaxAbsScaler , RobustScaler, PowerTransformer

import matplotlib.pyplot as plt

import seaborn as sns

import joblib

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis

from sklearn import metrics

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import roc\_curve, auc

import tensorflow as tf

from keras.layers import Dense

from keras.models import Sequential

from keras.models import model\_from\_json

from keras.layers import LSTM

from keras.layers import Input

from keras.models import Model

from keras.utils.vis\_utils import plot\_model

from sklearn.ensemble import ExtraTreesClassifier

import matplotlib.pyplot as plt

from sklearn import metrics

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import f1\_score

from sklearn.metrics import recall\_score

from tabulate import tabulate

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import f1\_score

from sklearn.metrics import recall\_score

from tabulate import tabulate

import matplotlib.pyplot as plt

from sklearn import metrics

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import f1\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import confusion\_matrix

from sklearn.neural\_network import MLPClassifier

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Activation, Dropout

from sklearn.preprocessing import StandardScaler

from numpy.linalg import eig

from matplotlib import pyplot as plt

from sklearn.metrics import plot\_confusion\_matrix

from sklearn.decomposition import PCA

from tabulate import tabulate

import numpy as np

import pandas as pd

import pickle

from os import path

from sklearn import preprocessing

from sklearn.preprocessing import (StandardScaler, OrdinalEncoder,LabelEncoder, MinMaxScaler, OneHotEncoder)

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.preprocessing import Normalizer, MaxAbsScaler , RobustScaler, PowerTransformer

import matplotlib.pyplot as plt

import seaborn as sns

from tabulate import tabulate

import joblib

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis

from sklearn import metrics

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import roc\_curve, auc

from sklearn.metrics import confusion\_matrix

import tensorflow as tf

from keras.layers import Dense

from keras.models import Sequential

from keras.models import model\_from\_json

from keras.layers import LSTM

from keras.layers import Input

from keras.models import Model

from keras.utils.vis\_utils import plot\_model

from sklearn.feature\_selection import chi2

from sklearn.feature\_selection import SelectKBest

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

import matplotlib.pyplot as plt

from sklearn import \*

from sklearn import metrics

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import f1\_score

from sklearn.metrics import recall\_score

from tabulate import tabulate

from sklearn.model\_selection import train\_test\_split

from google.colab import data\_table

from vega\_datasets import data

PCA.ipynb

from sklearn.decomposition import PCA

cd /content/drive/MyDrive/Colab Notebooks

run Preprocessing.ipynb

pca = PCA(n\_components=10)

pca.fit(train\_data)

train\_pca = pca.transform(train\_data)

test\_pca = pca.transform(test\_data)

X\_train\_PCA, X\_val\_PCA, y\_train\_PCA, y\_val\_PCA = train\_test\_split(train\_pca,y\_train\_data, test\_size=0.25, random\_state=42)

print('X\_train PCA has shape:',X\_train\_PCA.shape,'\ny\_train PCA has shape:',y\_train\_PCA.shape)

print('X\_val PCA has shape:',X\_val\_PCA.shape,'\ny\_val PCA has shape:',y\_val\_PCA.shape)

print('test PCA has shape:',test\_pca.shape,'\ntrain PCA has shape:',train\_pca.shape)

Preprocessing.ipynb

cd /content/drive/MyDrive/Colab Notebooks

run Requirements.ipynb

feature=["duration","protocol\_type","service","flag","src\_bytes","dst\_bytes","land","wrong\_fragment","urgent","hot",

"num\_failed\_logins","logged\_in","num\_compromised","root\_shell","su\_attempted","num\_root","num\_file\_creations","num\_shells",

"num\_access\_files","num\_outbound\_cmds","is\_host\_login","is\_guest\_login","count","srv\_count","serror\_rate","srv\_serror\_rate",

"rerror\_rate","srv\_rerror\_rate","same\_srv\_rate","diff\_srv\_rate","srv\_diff\_host\_rate","dst\_host\_count","dst\_host\_srv\_count",

"dst\_host\_same\_srv\_rate","dst\_host\_diff\_srv\_rate","dst\_host\_same\_src\_port\_rate","dst\_host\_srv\_diff\_host\_rate","dst\_host\_serror\_rate",

"dst\_host\_srv\_serror\_rate","dst\_host\_rerror\_rate","dst\_host\_srv\_rerror\_rate","label","difficulty"]

flag=['OTH','RSTOS0','SF','SH','RSTO','S2','S1','REJ','S3','RSTR','S0']

protocol\_type=['tcp','udp','icmp']

service=['http','smtp','finger','domain\_u','auth','telnet','ftp','eco\_i','ntp\_u','ecr\_i','other','private','pop\_3','ftp\_data',

'rje','time','mtp','link','remote\_job','gopher','ssh','name','whois','domain','login','imap4','daytime','ctf','nntp',

'shell','IRC','nnsp','http\_443','exec','printer','efs','courier','uucp','klogin','kshell','echo','discard','systat',

'supdup','iso\_tsap','hostnames','csnet\_ns','pop\_2','sunrpc','uucp\_path','netbios\_ns','netbios\_ssn','netbios\_dgm',

'sql\_net','vmnet','bgp','Z39\_50','ldap','netstat','urh\_i','X11','urp\_i','pm\_dump','tftp\_u','tim\_i','red\_i','icmp',

'http\_2784','harvest','aol','http\_8001']

binary\_attack=['normal','ipsweep', 'nmap', 'portsweep','satan', 'saint', 'mscan','back', 'land', 'neptune', 'pod', 'smurf',

'teardrop', 'apache2', 'udpstorm', 'processtable','mailbomb','buffer\_overflow', 'loadmodule', 'perl', 'rootkit',

'xterm', 'ps', 'sqlattack','ftp\_write', 'guess\_passwd', 'imap', 'multihop','phf', 'spy', 'warezclient',

'warezmaster','snmpgetattack','named', 'xlock', 'xsnoop','sendmail', 'httptunnel', 'worm', 'snmpguess']

multiclass\_attack={ 'normal': 'normal',

'probe': ['ipsweep.', 'nmap.', 'portsweep.','satan.', 'saint.', 'mscan.'],

'dos': ['back.', 'land.', 'neptune.', 'pod.', 'smurf.','teardrop.', 'apache2.', 'udpstorm.', 'processtable.','mailbomb.'],

'u2r': ['buffer\_overflow.', 'loadmodule.', 'perl.', 'rootkit.','xterm.', 'ps.', 'sqlattack.'],

'r2l': ['ftp\_write.', 'guess\_passwd.', 'imap.', 'multihop.','phf.', 'spy.', 'warezclient.', 'warezmaster.','snmpgetattack.',

'named.', 'xlock.', 'xsnoop.','sendmail.', 'httptunnel.', 'worm.', 'snmpguess.']}

print(feature)

print(binary\_attack)

print(multiclass\_attack)

Import Data

---

from google.colab import drive

drive.mount('/content/drive')

test\_data=pd.read\_csv('/content/drive/MyDrive/KDDTest+.txt',names=feature)

train\_data=pd.read\_csv('/content/drive/MyDrive/KDDTrain+.txt',names=feature)

Data Shape

---

test\_data.shape

train\_data.shape

Change Labels

---

def change\_label(df):

df.label.replace(['apache2','back','land','neptune','mailbomb','pod','processtable','smurf','teardrop','udpstorm','worm'],'Dos',inplace=True)

df.label.replace(['ftp\_write','guess\_passwd','httptunnel','imap','multihop','named','phf','sendmail','snmpgetattack','snmpguess','spy','warezclient','warezmaster','xlock','xsnoop'],'R2L',inplace=True)

df.label.replace(['ipsweep','mscan','nmap','portsweep','saint','satan'],'Probe',inplace=True)

df.label.replace(['buffer\_overflow','loadmodule','perl','ps','rootkit','sqlattack','xterm'],'U2R',inplace=True)

change\_label(train\_data)

change\_label(test\_data)

traindata=train\_data.label.value\_counts()

traindata.plot.bar()

testdata=test\_data.label.value\_counts()

testdata.plot.bar()

train\_data.shape

test\_data.shape

print('Raw Data Shape')

print('train has shape:',train\_data.shape)

print('test has shape:',test\_data.shape)

print('\n')

Preprocessing Data

---

train\_data.label.unique()

train\_data.protocol\_type.unique()

train\_data.service.unique()

train\_data.flag.unique()

train\_data['label'] = pd.DataFrame(train\_data.label.map(lambda x:0 if x=='normal' else 1))

test\_data['label'] = pd.DataFrame(test\_data.label.map(lambda x:0 if x=='normal' else 1))

train\_data = pd.get\_dummies(train\_data,columns=['protocol\_type','service','flag'],prefix="",prefix\_sep="")

test\_data = pd.get\_dummies(test\_data,columns=['protocol\_type','service','flag'],prefix="",prefix\_sep="")

test\_data.shape

train\_data.shape

print('Data Shape after dummy row introduction')

print('train has shape:',train\_data.shape)

print('test has shape:',test\_data.shape)

print('\n')

Data shaping

trainColumns = train\_data.columns

testColumns = test\_data.columns

diff = []

for col in trainColumns:

if col not in testColumns:

diff.append(col)

diff

train\_data.drop(diff, axis=1, inplace=True)

test\_data.shape

train\_data.shape

print('Data Shape after making shape uniform')

print('train has shape:',train\_data.shape)

print('test has shape:',test\_data.shape)

print('\n')

Separate Label Column to Y

---

y\_train\_data = train\_data['label']

train\_data.drop(labels= [ 'label'], axis=1, inplace=True)

y\_test\_data = test\_data['label']

test\_data.drop(labels= [ 'label'], axis=1, inplace=True)

Data View

---

test\_data

test\_data.shape

train\_data

train\_data.shape

y\_test\_data

y\_train\_data

print('Shape of Data after column label is separated')

print('train has shape:',train\_data.shape)

print('test has shape:',test\_data.shape)

print('\n')

Feature Selection

#num\_dataset\_bin is just include numeric features with binary labels

X\_train\_data = train\_data.copy()

X\_test\_data = test\_data.copy()

# print('X\_train has shape:',X\_train\_data.shape,'\ny\_train has shape:',y\_train\_data.shape)

# print('X\_test has shape:',X\_test\_data.shape,'\ny\_test has shape:',y\_test\_data.shape)

chi\_scores = chi2(X\_train\_data,y\_train\_data)

chi\_scores

p\_values = pd.Series(chi\_scores[1],index = X\_train\_data.columns)

p\_values.sort\_values(ascending = False , inplace = True)

p\_values = p\_values[:10]

print('The best features of the dataset and their impact:')

p\_values.plot.bar()

print('\n')

bestfeatures = SelectKBest(score\_func=chi2, k=10)

fit = bestfeatures.fit(X\_train\_data,y\_train\_data)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(X\_train\_data.columns)

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

featureScores.columns = ['Specs','Score']

featureScores

print('X\_train has shape:',X\_train\_data.shape,'\ny\_train has shape:',y\_train\_data.shape)

print('X\_test has shape:',X\_test\_data.shape,'\ny\_test has shape:',y\_test\_data.shape)

print('\n')

SVM.ipynb

Requirements

cd /content/drive/MyDrive/Colab Notebooks

run Requirements.ipynb

Evaluation Function

def evaluation(y\_test, y\_pred):

eMEA= metrics.mean\_absolute\_error(y\_test, y\_pred)

MSE= metrics.mean\_squared\_error(y\_test, y\_pred)

MSRE= np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

R2= metrics.explained\_variance\_score(y\_test, y\_pred)\*100

F1= metrics.f1\_score(y\_test,y\_pred)

Recall= metrics.recall\_score(y\_test,y\_pred)

accuracy = accuracy\_score(y\_test,y\_pred)\*100

data = [["Mean Absolute Error",eMEA],["Mean Squared Error",MSE],["Root Mean Squared Error",MSRE],["R2 Score",R2],["F1 Score",F1],["Recall Score",Recall],["Accuracy",accuracy]]

print (tabulate(data, tablefmt='fancy\_grid'))

print('\n')

PreProcessing

cd /content/drive/MyDrive/Colab Notebooks

%%capture

run Preprocessing.ipynb

%%capture

run PCA.ipynb

scaler = StandardScaler()

scaler.fit(X\_train\_data)

xTrain = scaler.transform(X\_train\_data)

xTest = scaler.transform(X\_test\_data)

X\_train, X\_val , Y\_train, Y\_val = train\_test\_split(xTrain, y\_train\_data, test\_size=0.6)

LSVM

svm = SVC(kernel='linear',gamma='auto')

history = svm.fit(X\_train, Y\_train)

binary\_predictions = svm.predict(X\_val)

base\_rf\_score = accuracy\_score(binary\_predictions,Y\_val)

base\_rf\_score

print("SVM Linear validation evaluation \n")

LSVMvalaccuracy = accuracy\_score(binary\_predictions,Y\_val)\*100

evaluation(binary\_predictions,Y\_val)

svm = SVC(kernel='linear',gamma='auto')

history = svm.fit(X\_train, Y\_train)

binary\_predictions = svm.predict(xTest)

base\_rf\_score = accuracy\_score(binary\_predictions,y\_test\_data)

base\_rf\_score

print("SVM Linear test evaluation \n")

LSVMtestaccuracy = accuracy\_score(binary\_predictions,y\_test\_data)\*100

evaluation(binary\_predictions,y\_test\_data)

QSVM

svm = SVC(kernel='poly',gamma='auto')

history = svm.fit(X\_train, Y\_train)

binary\_predictions = svm.predict(X\_val)

base\_rf\_score = accuracy\_score(binary\_predictions,Y\_val)

base\_rf\_score

print("SVM Quadratic validation evaluation \n")

QSVMvalaccuracy = accuracy\_score(binary\_predictions,Y\_val)\*100

evaluation(binary\_predictions,Y\_val)

svm = SVC(kernel='poly',gamma='auto')

history = svm.fit(X\_train, Y\_train)

binary\_predictions = svm.predict(xTest)

base\_rf\_score = accuracy\_score(binary\_predictions,y\_test\_data)

base\_rf\_score

print("SVM Quadratic test evaluation \n")

QSVMtestaccuracy = accuracy\_score(binary\_predictions,y\_test\_data)\*100

evaluation(binary\_predictions,y\_test\_data)

KNN.ipynb

cd /content/drive/MyDrive/Colab Notebooks

run Requirements.ipynb

Preprocessing

cd /content/drive/MyDrive/Colab Notebooks

%%capture

run Preprocessing.ipynb

Evaluation Function

def evaluation(y\_test, y\_pred):

eMEA= metrics.mean\_absolute\_error(y\_test, y\_pred)

MSE= metrics.mean\_squared\_error(y\_test, y\_pred)

MSRE= np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

R2= metrics.explained\_variance\_score(y\_test, y\_pred)\*100

F1= metrics.f1\_score(y\_test,y\_pred)

Recall= metrics.recall\_score(y\_test,y\_pred)

accuracy = accuracy\_score(y\_test,y\_pred)\*100

data = [["Mean Absolute Error",eMEA],["Mean Squared Error",MSE],["Root Mean Squared Error",MSRE],["R2 Score",R2],["F1 Score",F1],["Recall Score",Recall],["Accuracy",accuracy]]

print (tabulate(data, tablefmt='fancy\_grid'))

print('\n')

KNN with 20 neighbours

knn=KNeighborsClassifier(n\_neighbors=20)

knn.fit(X\_train\_data,y\_train\_data)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(train\_data,y\_train\_data, test\_size=0.25, random\_state=42)

y\_pred=knn.predict(X\_val)

ac=accuracy\_score(y\_val, y\_pred)\*100

ac

ac

print("KNN 20 neighbours validation evaluation \n")

KNN20valaccuracy = accuracy\_score(y\_val, y\_pred)\*100

print('Classification Report \n')

print(classification\_report(y\_val, y\_pred,target\_names=['Normal', 'Attack']))

evaluation(y\_val, y\_pred)

y\_pred=knn.predict(test\_data)

ac=accuracy\_score(y\_test\_data, y\_pred)\*100

print("KNN 20 neighbours test evaluation \n")

KNN20testaccuracy = accuracy\_score(y\_test\_data, y\_pred)\*100

print('Classification Report \n')

print(classification\_report(y\_test\_data, y\_pred,target\_names=['Normal', 'Attack']))

evaluation(y\_test\_data, y\_pred)

KNN with \*\*500\*\* Neighbours

knn=KNeighborsClassifier(n\_neighbors=500)

knn.fit(X\_train\_data,y\_train\_data)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(train\_data,y\_train\_data, test\_size=0.25, random\_state=42)

y\_pred=knn.predict(X\_val)

ac=accuracy\_score(y\_val, y\_pred)\*100

print("KNN 500 neighbours validation evaluation \n")

KNN500valaccuracy = accuracy\_score(y\_val, y\_pred)\*100

print('Classification Report \n')

print(classification\_report(y\_val, y\_pred,target\_names=['Normal', 'Attack'] ))

evaluation(y\_val, y\_pred)

y\_pred=knn.predict(test\_data)

ac=accuracy\_score(y\_test\_data, y\_pred)\*100

print("KNN 500 neighbours test evaluation \n")

KNN500testaccuracy = accuracy\_score(y\_test\_data, y\_pred)\*100

print('Classification Report \n')

print(classification\_report(y\_test\_data, y\_pred,target\_names=['Normal', 'Attack']))

evaluation(y\_test\_data, y\_pred)

KNN with 2000 neighbours

knn=KNeighborsClassifier(n\_neighbors=2000)

knn.fit(X\_train\_data,y\_train\_data)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(train\_data,y\_train\_data, test\_size=0.25, random\_state=42)

y\_pred=knn.predict(X\_val)

ac=accuracy\_score(y\_val, y\_pred)\*100

print("KNN 2000 neighbours validation evaluation \n")

KNN2000valaccuracy = accuracy\_score(y\_val, y\_pred)\*100

print('Classification Report \n')

print(classification\_report(y\_val, y\_pred,target\_names=['Normal', 'Attack']))

evaluation(y\_val, y\_pred)

y\_pred=knn.predict(test\_data)

ac=accuracy\_score(y\_test\_data, y\_pred)\*100

print("KNN 2000 neighbours test evaluation \n")

KNN2000testaccuracy = accuracy\_score(y\_test\_data, y\_pred)\*100

print('Classification Report \n')

print(classification\_report(y\_test\_data, y\_pred,target\_names=['Normal', 'Attack']))

evaluation(y\_test\_data, y\_pred)

NN.ipynb

cd /content/drive/MyDrive/Colab Notebooks

run Requirements.ipynb

Preprocessing

cd /content/drive/MyDrive/Colab Notebooks

%%capture

run Preprocessing.ipynb

Evaluation Function

def evaluation(y\_test, y\_pred):

eMEA= metrics.mean\_absolute\_error(y\_test, y\_pred)

MSE= metrics.mean\_squared\_error(y\_test, y\_pred)

MSRE= np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

R2= metrics.explained\_variance\_score(y\_test, y\_pred)\*100

F1= metrics.f1\_score(y\_test,y\_pred)

Recall= metrics.recall\_score(y\_test,y\_pred)

accuracy = accuracy\_score(y\_test,y\_pred)\*100

data = [["Mean Absolute Error",eMEA],["Mean Squared Error",MSE],["Root Mean Squared Error",MSRE],["R2 Score",R2],["F1 Score",F1],["Recall Score",Recall],["Accuracy",accuracy]]

print (tabulate(data, tablefmt='fancy\_grid'))

print('\n')

PCA

X = X\_train\_data

X = StandardScaler().fit\_transform(X)

pca = PCA(n\_components=10)

newXTrain = pca.fit\_transform(X)

newXTrain.shape

X\_train, X\_val , Y\_train, Y\_val = train\_test\_split(X\_train\_data, y\_train\_data, test\_size=0.6)

X = X\_val

newXVal = pca.transform(X)

newXTest = pca.transform(X\_test\_data)

MLP

classifier = MLPClassifier(hidden\_layer\_sizes=(10,100,2), max\_iter=300,activation = 'relu',solver='adam',random\_state=1)

classifier.fit(X\_train, Y\_train)

print("MLP validation evaluation \n")

pred = classifier.predict(X\_val)

evaluation(Y\_val,pred)

NNvalaccuracy = accuracy\_score(Y\_val,pred)\*100

print('Confusion Matrix:')

cm = confusion\_matrix(pred, Y\_val)

print(cm)

print('\n')

print("MLP test evaluation \n")

pred = classifier.predict(test\_data)

evaluation(y\_test\_data,pred)

NNtestaccuracy = accuracy\_score(y\_test\_data,pred)\*100

print('Confusion Matrix:')

cm = confusion\_matrix(pred, y\_test\_data)

print(cm)

print('\n')

batch\_size = 64

hidden\_units = 256

dropout = 0.5

model = Sequential()

model.add(Dense(hidden\_units, input\_dim=10))

model.add(Activation('relu'))

model.add(Dropout(dropout))

model.add(Dense(hidden\_units))

model.add(Activation('relu'))

model.add(Dropout(dropout))

model.add(Dense(1))

model.add(Activation('softmax'))

model.summary()

X\_train.shape

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(newXTrain, y\_train\_data, epochs=10, batch\_size=batch\_size, validation\_data=(newXVal, Y\_val))

print('MLP relu test evaluation')

pred = model.predict(newXTest)

evaluation(pred, y\_test\_data)

NNreluaccuracy = accuracy\_score(y\_test\_data,pred)\*100

cm = confusion\_matrix(pred, y\_test\_data)

print('Confusion Matrix: \n')

print(cm)

ETC.ipynb

Requirements

cd /content/drive/MyDrive/Colab Notebooks

run Requirements.ipynb

Evaluation Function

def evaluation(y\_test, y\_pred):

eMEA= metrics.mean\_absolute\_error(y\_test, y\_pred)

MSE= metrics.mean\_squared\_error(y\_test, y\_pred)

MSRE= np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

R2= metrics.explained\_variance\_score(y\_test, y\_pred)\*100

F1= metrics.f1\_score(y\_test,y\_pred)

Recall= metrics.recall\_score(y\_test,y\_pred)

accuracy = accuracy\_score(y\_test,y\_pred)\*100

data = [["Mean Absolute Error",eMEA],["Mean Squared Error",MSE],["Root Mean Squared Error",MSRE],["R2 Score",R2],["F1 Score",F1],["Recall Score",Recall],["Accuracy",accuracy]]

print (tabulate(data, tablefmt='fancy\_grid'))

print('\n')

EDA and Preprocessing

cd /content/drive/MyDrive/Colab Notebooks

%%capture

run Preprocessing.ipynb

ExtraTreesClassifer

X\_train\_data, X\_val, y\_train\_data, y\_val = train\_test\_split(train\_data,y\_train\_data, test\_size=0.25, random\_state=42)

model = ExtraTreesClassifier()

model.fit(X\_train\_data,y\_train\_data)

Run inbuilt Importance Function and visualize

# print(model.feature\_importances\_)

feat\_importances = pd.Series(model.feature\_importances\_, index=X\_test\_data.columns)

feat\_importances.nlargest(10).plot(kind='barh')

plt.show()

Validation

pred = model.predict(X\_val)

print("Extra Trees Classifier validation evaluation \n")

ETCvalaccuracy = accuracy\_score(y\_val,pred)\*100

evaluation(y\_val,pred)

Prediction

pred = model.predict(X\_test\_data)

print("Extra Trees Classifier test evaluation \n")

ETCtestaccuracy = accuracy\_score(y\_test\_data,pred)\*100

evaluation(pred,y\_test\_data)

Extra Trees Classifier with PCA

cd /content/drive/MyDrive/Colab Notebooks

%%capture

run PCA.ipynb

X\_train\_PCA, X\_val\_PCA, y\_train\_PCA, y\_val\_PCA = train\_test\_split(train\_pca,y\_train\_data, test\_size=0.25, random\_state=42)

model = ExtraTreesClassifier()

model.fit(X\_train\_PCA,y\_train\_PCA)

Validation

print("Extra Trees Classifier validation evaluation with PCA \n")

pred = model.predict(X\_val\_PCA)

evaluation(y\_val\_PCA, pred)

ETCPCAvalaccuracy = accuracy\_score(y\_val\_PCA, pred)\*100

Prediction

print("Extra Trees Classifier test evaluation with PCA \n")

y\_pred=model.predict(test\_pca)

evaluation(y\_test\_data, y\_pred)

ETCPCAtestaccuracy = accuracy\_score(y\_test\_data, y\_pred)\*100

RF.ipynb

Random Forest

Requirements

cd /content/drive/MyDrive/Colab Notebooks

run Requirements.ipynb

Evaluation Function

def evaluation(y\_test, y\_pred):

eMEA= metrics.mean\_absolute\_error(y\_test, y\_pred)

MSE= metrics.mean\_squared\_error(y\_test, y\_pred)

MSRE= np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

R2= metrics.explained\_variance\_score(y\_test, y\_pred)\*100

F1= metrics.f1\_score(y\_test,y\_pred)

Recall= metrics.recall\_score(y\_test,y\_pred)

accuracy = accuracy\_score(y\_test,y\_pred)\*100

data = [["Mean Absolute Error",eMEA],["Mean Squared Error",MSE],["Root Mean Squared Error",MSRE],["R2 Score",R2],["F1 Score",F1],["Recall Score",Recall],["Accuracy",accuracy]]

print (tabulate(data, tablefmt='fancy\_grid'))

print('\n')

PreProcessing

cd /content/drive/MyDrive/Colab Notebooks

%%capture

run Preprocessing.ipynb

Random Forest Classifier

Random Forest

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train\_data,y\_train\_data, test\_size=0.25, random\_state=43)

model = RandomForestClassifier(n\_estimators=50, criterion="entropy", max\_depth=10, min\_samples\_split=2, min\_samples\_leaf=10, min\_weight\_fraction\_leaf=0, max\_features="log2", max\_leaf\_nodes=100, min\_impurity\_decrease=0, bootstrap=True, oob\_score=True, n\_jobs=5, random\_state=10, verbose=0, warm\_start=True, class\_weight=None, ccp\_alpha=0, max\_samples=10000)

validation

model.fit(X\_train,y\_train)

print("Random Forest Classifier validation evaluation \n")

y\_pred=model.predict(X\_val)

evaluation(y\_val, y\_pred)

RFvalaccuracy = accuracy\_score(y\_val,y\_pred)\*100

test

print("Random Forest Classifier test evaluation \n")

y\_pred=model.predict(X\_test\_data)

evaluation(y\_test\_data, y\_pred)

RFtestaccuracy = accuracy\_score(y\_test\_data,y\_pred)\*100

With PCA

print("Random Forest Classifier validation evaluation with PCA \n")

cd /content/drive/MyDrive/Colab Notebooks

%%capture

run PCA.ipynb

model = RandomForestClassifier(n\_estimators=45, criterion="entropy", max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0, max\_features="log2", max\_leaf\_nodes=10, min\_impurity\_decrease=0, bootstrap=True, oob\_score=True, n\_jobs=5, random\_state=10, verbose=0, warm\_start=True, class\_weight=None, ccp\_alpha=0, max\_samples=10000)

model.fit(X\_train\_PCA,y\_train\_PCA)

val

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_test\_data,y\_test\_data, test\_size=0.25, random\_state=43)

y\_pred=model.predict(X\_val\_PCA)

evaluation(y\_val\_PCA, y\_pred)

RFPCAvalaccuracy = accuracy\_score(y\_val\_PCA,y\_pred)\*100

test

print("Random Forest Classifier test evaluation with PCA\n")

y\_pred=model.predict(test\_pca)

evaluation(y\_test\_data, y\_pred)

RFPCAtestaccuracy = accuracy\_score(y\_test\_data,y\_pred)\*100

Analysis.ipynb

cd /content/drive/MyDrive/Colab Notebooks

run Preprocessing.ipynb

run SVM.ipynb

run KNN.ipynb

run NN.ipynb

run RandomForest.ipynb

run ExtraTreesClassifier.ipynb

acc=[LSVMtestaccuracy],[QSVMtestaccuracy],[KNN20testaccuracy],[KNN500testaccuracy],[KNN2000testaccuracy],[NNtestaccuracy],[NNreluaccuracy],[RFtestaccuracy],[RFPCAtestaccuracy],[ETCtestaccuracy],[ETCPCAtestaccuracy]

print('All Accuracies')

acctable=["LSVM Validation",LSVMvalaccuracy],["LSVM Test",LSVMtestaccuracy],["QSVM Validation",QSVMvalaccuracy],["QSVM Test",QSVMtestaccuracy],["KNN 20 Validation",KNN20valaccuracy],["KNN 20 Test",KNN20testaccuracy],["KNN 500 Validation",KNN500valaccuracy],["KNN 500 Test",KNN500testaccuracy],["KNN 2000 Validation",KNN2000valaccuracy],["KNN 2000 Test",KNN2000testaccuracy],["NN Validation",NNvalaccuracy],["NN Test",NNtestaccuracy],["NN ReLU",NNreluaccuracy],["RF Validation",RFvalaccuracy],["RF Test",RFtestaccuracy],["RF PCA Validation",RFPCAvalaccuracy],["RF PCA Test",RFPCAtestaccuracy],["ETC Validation",ETCvalaccuracy],["ETC Test",ETCtestaccuracy],["ETC PCA Validation",ETCPCAvalaccuracy],["ETC PCA Test",ETCPCAtestaccuracy]

print (tabulate(acctable,headers=["Model", "Accuracy"] , tablefmt='fancy\_grid'))

print('\nTest Accuracies')

TestAcc=["LSVM Test",LSVMtestaccuracy],["QSVM Test",QSVMtestaccuracy],["KNN 20 Test",KNN20testaccuracy],["KNN 500 Test",KNN500testaccuracy],["KNN 2000 Test",KNN2000testaccuracy],["NN Test",NNtestaccuracy],["NN ReLU",NNreluaccuracy],["RF Test",RFtestaccuracy],["RF PCA Test",RFPCAtestaccuracy],["ETC Test",ETCtestaccuracy],["ETC PCA Test",ETCPCAtestaccuracy]

print (tabulate(TestAcc,headers=["Model", "Accuracy"] , tablefmt='fancy\_grid'))

print('Max Test Accuracy:')

max(acc)

import plotly.graph\_objects as go

fig = go.Figure( go.Bar(x=["LSVMtestaccuracy","QSVMtestaccuracy","KNN20testaccuracy","KNN500testaccuracy","KNN2000testaccuracy","NNtestaccuracy","NNreluaccuracy","RFtestaccuracy","RFPCAtestaccuracy","ETCtestaccuracy","ETCPCAtestaccuracy"], y=[LSVMtestaccuracy,QSVMtestaccuracy,KNN20testaccuracy,KNN500testaccuracy,KNN2000testaccuracy,NNtestaccuracy,NNreluaccuracy,RFtestaccuracy,RFPCAtestaccuracy,ETCtestaccuracy,ETCPCAtestaccuracy] ))

fig.show()

Solution.ipynb

Requirements

cd /content/drive/MyDrive/Colab Notebooks

run Requirements.ipynb

feature=["duration","protocol\_type","service","flag","src\_bytes","dst\_bytes","land","wrong\_fragment","urgent","hot",

"num\_failed\_logins","logged\_in","num\_compromised","root\_shell","su\_attempted","num\_root","num\_file\_creations","num\_shells",

"num\_access\_files","num\_outbound\_cmds","is\_host\_login","is\_guest\_login","count","srv\_count","serror\_rate","srv\_serror\_rate",

"rerror\_rate","srv\_rerror\_rate","same\_srv\_rate","diff\_srv\_rate","srv\_diff\_host\_rate","dst\_host\_count","dst\_host\_srv\_count",

"dst\_host\_same\_srv\_rate","dst\_host\_diff\_srv\_rate","dst\_host\_same\_src\_port\_rate","dst\_host\_srv\_diff\_host\_rate","dst\_host\_serror\_rate",

"dst\_host\_srv\_serror\_rate","dst\_host\_rerror\_rate","dst\_host\_srv\_rerror\_rate","label","difficulty"]

Import Data

---

test\_data=pd.read\_csv('/content/drive/MyDrive/KDDTest+.txt',names=feature)

train\_data=pd.read\_csv('/content/drive/MyDrive/KDDTrain+.txt',names=feature)

testing=pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/test.txt',names=feature)

train\_data.label.unique()

Change Labels

---

def change\_label(df):

df.label.replace(['apache2','back','land','neptune','mailbomb','pod','processtable','smurf','teardrop','udpstorm','worm'],'Dos',inplace=True)

df.label.replace(['ftp\_write','guess\_passwd','httptunnel','imap','multihop','named','phf','sendmail','snmpgetattack','snmpguess','spy','warezclient','warezmaster','xlock','xsnoop'],'R2L',inplace=True)

df.label.replace(['ipsweep','mscan','nmap','portsweep','saint','satan'],'Probe',inplace=True)

df.label.replace(['buffer\_overflow','loadmodule','perl','ps','rootkit','sqlattack','xterm'],'U2R',inplace=True)

def evaluation(y\_test, y\_pred):

eMEA= metrics.mean\_absolute\_error(y\_test, y\_pred)

MSE= metrics.mean\_squared\_error(y\_test, y\_pred)

MSRE= np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

R2= metrics.explained\_variance\_score(y\_test, y\_pred)\*100

F1= metrics.f1\_score(y\_test,y\_pred)

Recall= metrics.recall\_score(y\_test,y\_pred)

accuracy = accuracy\_score(y\_test,y\_pred)\*100

data = [["Mean Absolute Error",eMEA],["Mean Squared Error",MSE],["Root Mean Squared Error",MSRE],["R2 Score",R2],["F1 Score",F1],["Recall Score",Recall],["Accuracy",accuracy]]

print (tabulate(data, tablefmt='fancy\_grid'))

print('\n')

# def preprocess(test\_data,train\_data,testing):

from numpy.ma.core import copy

change\_label(train\_data)

change\_label(test\_data)

change\_label(testing)

train\_data['label'] = pd.DataFrame(train\_data.label.map(lambda x:0 if x=='normal' else 1))

test\_data['label'] = pd.DataFrame(test\_data.label.map(lambda x:0 if x=='normal' else 1))

testing['label'] = pd.DataFrame(testing.label.map(lambda x:0 if x=='normal' else 1))

train\_data = pd.get\_dummies(train\_data,columns=['protocol\_type','service','flag'],prefix="",prefix\_sep="")

test\_data = pd.get\_dummies(test\_data,columns=['protocol\_type','service','flag'],prefix="",prefix\_sep="")

testing = pd.get\_dummies(testing,columns=['protocol\_type','service','flag'],prefix="",prefix\_sep="")

trainColumns = train\_data.columns

testColumns = test\_data.columns

testingColumns = testing.columns

diff = []

for col in testColumns:

if col not in testingColumns or col not in trainColumns:

diff.append(col)

test\_data.drop(diff, axis=1, inplace=True)

diff=[]

for col in trainColumns:

if col not in testingColumns or col not in testColumns:

diff.append(col)

train\_data.drop(diff, axis=1, inplace=True)

diff=[]

for col in testingColumns:

if col not in testColumns or col not in trainColumns:

diff.append(col)

testing.drop(diff, axis=1, inplace=True)

diff

print('Data Shape after making shape uniform')

print('train has shape:',train\_data.shape)

print('test has shape:',test\_data.shape)

print('testing has shape',testing.shape)

print('\n')

y\_train\_data = train\_data['label']

train\_data.drop(labels= [ 'label'], axis=1, inplace=True)

y\_train\_data

X\_train\_data = train\_data.copy()

X\_test\_data = test\_data.copy()

y\_test\_data = test\_data['label']

test\_data.drop(labels= [ 'label'], axis=1, inplace=True)

y\_test\_data

y\_testing = testing['label']

testing.drop(labels= [ 'label'], axis=1, inplace=True)

y\_testing

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train\_data,y\_train\_data, test\_size=0.25, random\_state=42)

pca = PCA(n\_components=5)

pca.fit(train\_data)

train\_pca = pca.transform(train\_data)

test\_pca = pca.transform(test\_data)

testing\_pca=pca.transform(testing)

X\_train\_PCA, X\_val\_PCA, y\_train\_PCA, y\_val\_PCA = train\_test\_split(train\_pca,y\_train\_data, test\_size=0.25, random\_state=42)

print('X\_train PCA has shape:',X\_train\_PCA.shape,'\ny\_train PCA has shape:',y\_train\_PCA.shape)

print('X\_val PCA has shape:',X\_val\_PCA.shape,'\ny\_val PCA has shape:',y\_val\_PCA.shape)

print('test PCA has shape:',test\_pca.shape,'\ntrain PCA has shape:',train\_pca.shape)

Single Packet Test

# preprocess(test\_data,train\_data,testing)

model = RandomForestClassifier(n\_estimators=45, criterion="entropy", max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0, max\_features="log2", max\_leaf\_nodes=10, min\_impurity\_decrease=0, bootstrap=True, oob\_score=True, n\_jobs=5, random\_state=10, verbose=0, warm\_start=True, class\_weight=None, ccp\_alpha=0, max\_samples=10000)

model.fit(X\_train\_PCA,y\_train\_PCA)

y\_pred=model.predict(testing\_pca)

y\_pred=model.predict(testing\_pca)

Check = accuracy\_score(y\_testing,y\_pred)\*100

print(Check)

if(Check):

print("Normal")

else:

print("Attack")

print(y\_pred)

print(y\_testing)

evaluation(y\_testing, y\_pred)

**Bibliography**

[1]Gayathri Rajakumaran1 · Neelanarayanan Venkataraman1.” Raghava Rao Mukkamala2. “Denial of Service Attack Prediction Using Gradient Descent Algorithm”. Received: 10 June 2019 / Accepted: 4 October 2019 © Springer Nature Singapore Pte Ltd 2019.

[2] C.Gong,K.Sarac,A more practical approach for single-packet I traceback using packet logging and marking, IEEETrans. Parallel Distrib Syst 19(10)(2008)

[3]Aroosh Amjad, Tahir Alyas\*, Umer Farooq, Muhammad Arslan Tariq. “Detection and mitigation of DDoS attack in cloud computing using machine learning algorithm”.

[4] Ahmad Sanmorino. “A study for DDOS attack classification method” 1st International Conference on Advance and Scientific Innovation (ICASI).

[5] Fadil, A.; Riadi, I.; Aji, S. Review of detection DDoS attack detection using Naïve Bayes classifier for network forensics. Bull. Electr. Eng. Inform. 2017

[6] Shravan K, Shridhar Allagi and Rashmi Rachh, An Approach for Securing Big Data Environment using Machine Learning for Dynamic Processing of Portable Executable in Network, International Journal of Advanced Research in Engineering and Technology, 12(1), 2021

[7] Yuanyuan Sun 1, 2, 3a, Yongming Wang 1, 2b, Lili Guo 3c, Zhongsong Ma3, Shan Jin3 and Huiping Wang. “Researching on Multiple Machine Learning for Anomaly Detection”.

[8] J.Li;Y.Liu;L.Gu, DDoS attack detection based on neural network,in: Proceedings of the 2nd International Symposium on Aware Computing (ISAC),Tainan,1–4 Nov.2010

[9] Yong, B.; Wei, W.; Li, K.C.; Shen, J.; Zhou, Q.; Wozniak, M.; Połap, D.; Damaševičius, R. Ensemble machine learning approaches for webshell detection in Internet of things environments. Trans. Emerg. Telecommun. Technol. 2020

[10] Hemalatha, J.; Roseline, S.A.; Geetha, S.; Kadry, S.; Damaševiˇcius, R. An efficient DenseNet-based deep learning model for malware detection. Entropy 2021

[11] Idhammad M, Afdel K, Belouch M (2017) DoS detection methodbased on artificial neural networks. Int J Adv Comput Sci Appl, 2017

[12]Bhuvaneswari Amma N G, Selvakumar S (2020) A statistical class center based triangle area vector method for detection of denial of service attacks. Cluster Computing

[13] N. Martins, J. M. Cruz, T. Cruz and P. H. Abreu, "Adversarial machine learning applied to intrusion and malware scenarios: A systematic review", IEEE Access, vol. 8, 2020

[14] Zhang, J.; Li, Y.; Xiao, W.; Zhang, Z. Non-iterative and Fast Deep Learning: Multilayer Extreme Learning Machines. J. Franklin Inst. 2020, 357, 8925–8955.

[15] X. Gao, C. Shan, C. Hu, Z. Niu, and Z. Liu, ‘‘An adaptive ensemble machine learning model for intrusion detection,’’ IEEE Access, vol. 7, 2019

[16] Larriva-Novo, X.A.; Vega-Barbas, M.; Villagra, V.A.; Sanz Rodrigo, M. Evaluation of Cybersecurity Data Set Characteristics for Their Applicability to Neural Networks Algorithms Detecting Cybersecurity Anomalies, 2020.

[17] G. Ajeetha and G. M. Priya, "Machine Learning Based DDoS Attack Detection," in 2019 Innovations in Power and Advanced Computing Technologies (i-PACT), 2019, vol. 1: IEEE

[18] Bendale, A.; Boult, T.E. Towards open set deep networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; IEEE: Piscataway Township, NJ, USA, 2016;.

[19] Sabeel, U.; Heydari, S.S.; Mohanka, H.; Bendhaou, Y.; Elgazzar, K.; El-Khatib, K. Evaluation of deep learning in detecting unknown network attacks. In Proceedings of the 2019 International Conference on Smart Applications, Communications and Networking, Sharm El Sheik, Egypt, 2019

[20] Jie-Hao, C.; Feng-Jiao, C.; Zhang. (2012) “DDoS defense system with test and neural network”. IEEE International Conference on Granular Computing (GrC), 11-13 Aug. 2012, Hangzhou, China,

[21] V.Akilandeswari; S.M.Shalinie, Probabilistic neural network based attack traffic classification, in:Proceedings of the Fourth International Conference on Advanced Computing(ICoAC),Chennai,13-15 Dec 2012.

[22]Leu F.; Pai C. (2010) “Detecting DoS and DDoS Attacks Using Chi -Square”, Fifth Inter-national Conference on Information Assurance and Security (IAS-09), 18-20 August 2009, Xian, pp.225-258 26.

[23]Xu, X. ;Wei, D. ; Zhang, Y. (2011) “Improved Detection Approach for Distributed Denial of Service Attack Based on SVM”. 2011 Third Pacific-Asia Conference on Circuits, Communications and Systems (PACCS) 17-18 July 2011, Wuhan.