BOTNET ATTACK DETECTION IN IoT USING MACHINE LEARNING

A major project report submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering / Information Technology

Submitted by

Ananya Agarwal (201181)

Aditi Saxena (201433)

Under the guidance & supervision of

Dr. Ruchi Verma

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

CERTIFICATE

This is to certify that the work which is being presented in the project report titled 'Botnet Detection in IoT using Machine Learning' in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering / Information Technology submitted in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by Ananya Agarwal(201181) and Aditi Saxena(201433) during the period from August 2023 to May 2024 under the supervision of Dr. Ruchi Verma, Assistant Professor(SG), Department of Computer Science and Engineering Jaypee University of Information Technology, Waknaghat.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Ananya Agarwal

201181

Aditi Saxena

201433

This is to certify that the above statement made by the candidate is true to the best of our knowledge.

Dr. Ruchi Verma

Associate Professor (SG)

Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat

Candidate's Declaration

We hereby declare that the work presented in this report entitled **'Botnet Attack Detection in IoT using Machine Learning'** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering / Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of our work carried out over a period from August 2023 to May 2024 under the supervision of **Dr. Ruchi Verma** (Assistant Professor (SG), Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Ananya Agarwal
201181

Aditi Saxena
201433

This is to certify that the above statement made by the candidate is true to the best of our knowledge.

Dr. Ruchi Verma
Assistant Professor (SG)
Department of Computer Science & Engineering
Dated:

ACKNOWLEDGEMENT

First of all, we would like to sincerely thank Almighty God for His divine favour, which has made it possible for the project work to be successfully completed.

We express our gratitude and deep debt to our supervisor, Dr. Ruchi Verma, Senior Grade Assistant Professor, CSE Department, Jaypee University of Information Technology, Waknaghat. To complete this project, our supervisor has deep expertise in the field of "machine learning" and a strong interest in it. This project could not have been finished without her unending patience, academic advice, persistent encouragement, active supervision, constructive criticism, insightful counsel, and reading numerous subpar drafts and editing them at every turn.

Additionally, we would like to thank Mr. Mohan Sharma and Mr. Ravi Raina, our respective lab coordinators, for giving us the resources we needed to finish our project.

In addition, we would like to extend a warm welcome to everyone who has assisted us, directly or indirectly, in bringing success to this endeavour. Given the particular circumstances, we wish to express our gratitude to the numerous staff members—both instructional and non-instructional—who have provided helpful assistance and enabled our project.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

Ananya Agarwal 201181

Aditi Saxena 201433

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1.	**ML**:	Machine Learning
2.	**DL:**	Deep Learning
3.	**DDoS**:	distributed denial-of-service
4.	**IoT**:	Internet of Things
5.	**C&C**:	Command and Control
6.	**P2P** :	Peer-to-peer
7.	**LR**:	Logistic Regression
8.	**RF**:	Random Forest
9.	**DT**:	Decision Tree
10.	**NB**:	Naive Bayesian
11.	**TP**:	true positives
12.	**Fp**:	false positives
13.	**Tn**:	true negatives
14.	**Sf**:	specificity
15.	**FPR:**	false positive rate
16.	**Fn**:	false negatives
17.	**TCP:**	Transmission Control Protocol
18.	**IPS**:	Intrusion Prevention Systems
19.	**MFA**:	Multi-Factor Authentication

ABSTRACT

The traditional way of living has been altered by the Internet of Things (IoT) to a lifestyle where technology plays a significant role in our daily lives—sometimes even more so than people. There is technology involved in everything we see, touch, and experience. The Internet of Things has brought about life-changing innovations such as smartphones, smart homes, smart cities, and smart energy-saving systems.

Even though the technologies are getting better by the second, the Internet of Things has not yet reached its full potential. But as technology advances, so do the risks associated with it. The issue of hacked networks and broken systems is getting worse very quickly. As technologies advance, so do these threats—which are becoming more sophisticated and changing quickly. In computer language, these dangers are known as bots.

The significance of identifying and stopping these bots is increasing every day. Among the tools and techniques created for this purpose are antivirus software, network sniffers, secure passwords, and regular system checks. Anti-botnet software, honeypots and honeynets, signature-based and anomaly-based detection methods, and more are available for detection. Though there are tried-and-true methods for dealing with botnet threats, innovation is never out of the question.

That's the reason we developed this project, in which hot-pot technology is employed. For this reason, we developed this project in which we detect botnet attacks on other network using machine learning.

CHAPTER 1: INTRODUCTION

1.1 Introduction

In 1999, Kevin Ashton came up with the term "Internet of Things" to emphasize the seemingly endless possibilities of using sensor technologies to gather data. The use of IoT in the fields of machinery, law enforcement, healthcare, physical security, and transportation may decrease, according to Gartner data.

IoT, or "Internet of things," is a "network of objects," or "matters," that use a variety of software, sensors, and technological advancements to connect multiple devices over the internet. These gadgets include everything from simple household objects to complex commercial machnies. This concept is still in the infancy and does not have thorough security strategies, which puts important statistics at risk. On the IoT network, modern security features must be implemented in order to protect IoT entities, agencies, and individuals. The most serious security risk associated with the this is DDoS attacks, where intruders harms system with scripts.

"Robot networks," or "botnets," are groups of infected computers under the direction of attacker or "bot herdsman.". ‘bot’ is a device that is controlled by herdsman. The botnet's biases can cooperate to carry out illegal operations under a single truth. It is acceptable for adversaries to conduct extensive malware operations using botnets, some of which may have hundreds of thousands of bots. However, biased content can be updated and modified externally because botnets are managed by foreign adversaries. Consequently, bot herders can choose to profit financially by continuously leasing access to botnet components on the black market. The bushwhacker, also known as the Bot master, viruses, malware, or both, including the range of devices on the internet.

Even though dispatch is thought of as an antiquated attack channel, spam botnets are among the most significant real-world botnet attacks. Examples of such attacks include spam and junk mail dispatch. Bots are typically employed to send out hundreds of thousands of spam emails, most of which include malware.

1.1.1 How a Botnet works

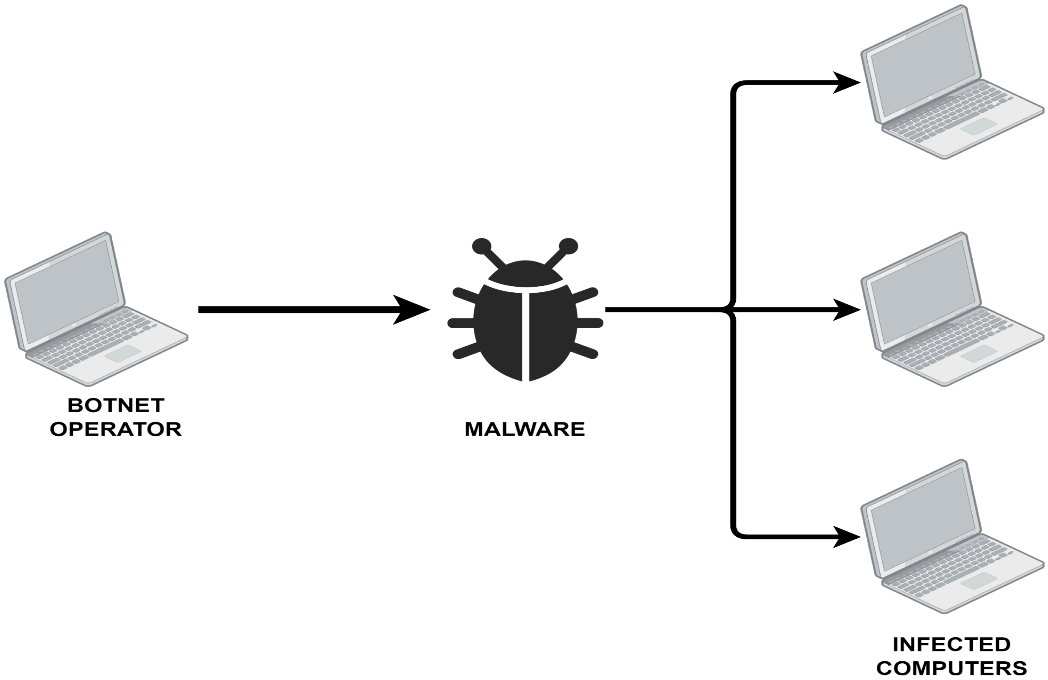


Fig 1.1 Botnet architecture

DDoS assaults take advantage of the large-scale botnet to flood a target network or garçon with requests, making it inaccessible to the intended drug dealers. DDoS attacks target organizations in order to achieve specific or political goals, or to compel oligarchies to halt the attack. Botnets designed expressly to steal money from companies and credit card details are examples of financial attacks. Financial botnets have held responsible for attacks that have swiftly taken thousands bones from many agencies, much like the Zeus botnet.Botnets can be distinguished from other forms of malware by their unique armature.

Botnets are similar to worms in that they can spread across tens of thousands of devices. Moreover, promoting awareness about cybersecurity risks among IoT stakeholders and fostering a culture of proactive risk management are crucial steps towards safeguarding IoT ecosystems and mitigating the impact of botnet threats on businesses and individuals alike. It is not possible to distribute botnets as distinct malware types due to their dispersed armature. The phylogeny of botnets is attempted to be represented in many courses. The propagation medium, exploitation strategy, bushwhacker-accessible set of operations, and C&C structure topology are critical areas for botnet classification.

1.1.2 COMMAND AND CONTROL TOPOLOGY

"Command and Control" servers, these are the centralised platforms with the ability to issue commands and reveal information from within a botnet. In the event that an interloper wishes to launch a DDoS attack, the packet sniffers interacted with the server communicated the server could additionally launch a collaborated assault.

Alternatively, they could send some instructions to the command and manage servers to instruct them to launch an attack against a designated target. One of four architectures is commonly used to prepare botnet C&C servers; these are superstar, a handful of servers, hierarchical, and arbitrary, and each has advantages and disadvantages.

In essence, C&C servers are the linchpin of botnet operations, providing attackers with the means to orchestrate large-scale cyberattacks, manipulate compromised devices, and evade detection by law enforcement or security measures. Effectively targeting and disturbing C&C infrastructure is essential in reducing the impact of bot network driven cyber threats and safeguarding digital ecosystems from malicious activities.

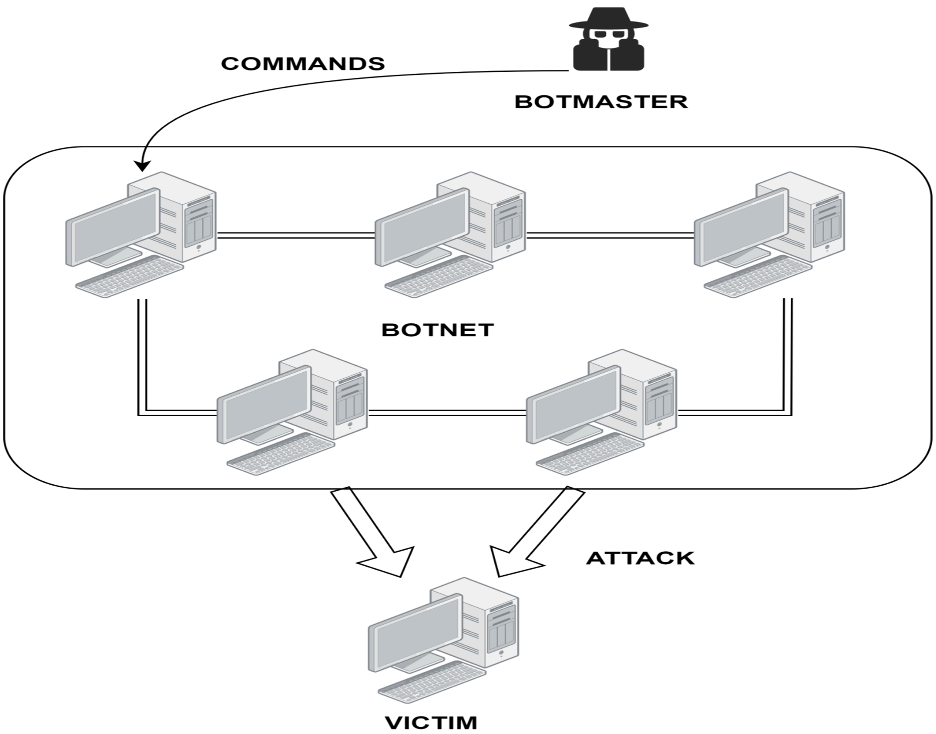


Fig 1.2 Architecture of Command and Control

1.1.3 P2P TOPOLOGY

It give resilience precedence over centralized command and control botnets by creating a peer to peer network. Peer-to-peer botnets are the same as centralised botnets in many other respects. Devices connected to peer to peer networks instantly share resources without going through the authority that manages centralized resources.

P2P botnets enable the sharing of commands, updates, and other data amongst bots via a range of communication protocols, including HTTP, UDP, and TCP/IP. The botnet can operate without a single point of failure because the bots can dynamically switch between functioning as servers and clients. Because of this, it is not easy to stop the bot network by bringing down its main C and C server.

Botmasters benefit from P2P topology in a number of ways. In the beginning, this gives the botnet greater adaptability to takedown attempts because there isn't a single point of failure that can be interfered with. In fact, the botnet can still function even if some of the bots are eliminated because they can still communicate and receive commands from one another.

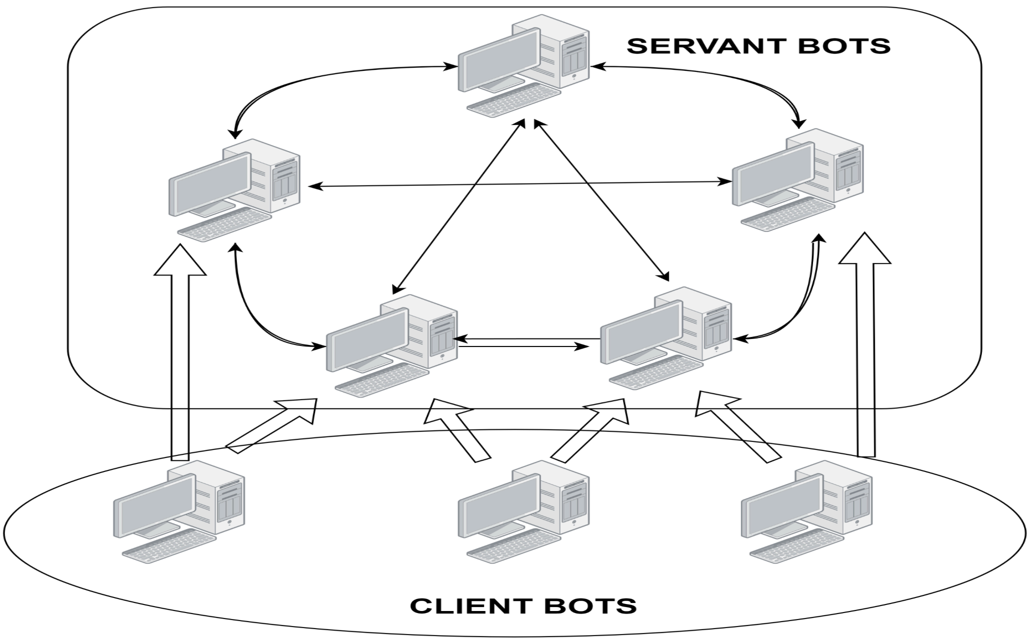


Fig 1.3 Architecture of P2P

1.1.4 Why can botnet attacks affect IoT devices?

The botnet attacks in IoT are common due to the following reasons:

1. Limited Computing Resources: A lot of Internet of Things devices have low amounts of memory, processing power, and storage. This restriction makes it difficult to put strong security measures on these devices, which increases their susceptibility to botnet exploitation.

2. Inadequate Security Measures:
    a. Default Credentials: Manufacturers often ship IoT devices with default usernames and passwords, and users frequently neglect to change them. This common practice makes it easier for botnets to gain unauthorized access through known credentials.
    b. Lack of Updates: Some IoT devices lack mechanisms for regular security updates. Without firmware updates to patch vulnerabilities, devices remain exposed to evolving threats.

3. Interconnected Ecosystems: The interconnected nature of IoT ecosystems means that a compromise in one device can potentially affect others within the network. Botnets leverage this interdependency to rapidly propagate and amplify their impact.

4. Inherent Design Flaws: In a few instances, safety isn't always given importance all through the layout and improvement phase of IoT gadgets. Manufacturers may awareness greater on capability and value, leaving vulnerabilities unaddressed until they may be exploited.

5. Lack of Standardization: The loss of standardized in security protocols across all IoT gadgets generally outcomes in a heterogeneous landscape. This range makes it difficult to apply uniform safety practices, leaving gaps in system that botnets can exploit.

6. Absence of Uniform Security Standards: IoT devices use a whole lot of communique protocols, and the absence of a customary general makes it difficult to implement regular protection practices. This diversity affords attackers with more than one avenues to compromise devices.

7. Weak Authentication Mechanisms: Some IoT gadgets might also moreover hire weak or insecure authentication mechanisms. Inadequate authentication makes it less complicated for malicious actors to advantage unauthorized get entry to and manipulate over the device.

8. Lack of Encryption: In scenarios where IoT devices communicate over networks without encryption, the transmitted data is vulnerable to interception. This lack of encryption can expose sensitive information and facilitate unauthorised access.

9. Remote Locations: Many IoT devices are deployed in remote or inaccessible locations, making it challenging to apply security patches promptly. This delay provides a window of opportunity for botnets to exploit unpatched vulnerabilities.

Multiple Botnet prevention techniques exist. Mitigating and preventing botnet attacks involves a multi-faceted approach that addresses various aspects of cybersecurity.

1. Strong Authentication:

- Use Complex Passwords: Practice using strong, specific passwords for all gadgets and structures, to keep devices safe
- Implement Multi-Factor Authentication (MFA): Require an additional layer of authentication to decorate protection.

2. Regular Software Updates:

- Firmware and Software Patches: Make sure the vulnerabilities are addressed by maintaining each tool have normal update.
- Automate Updates: adoption automated replace mechanisms within the system for quicker and much less tough patching technique.

3. Network Segmentation: Network segmentation to restriction the effect of a credibility attack particular segments now not permitting facet to facet movement.

4. Firewall Configuration:

- Strict Firewall Rules: Set up firewalls to permit important web site traffic and close greater chatting.

- Intrusion Detection and Prevention Systems (IDPS): Use intrusion prevention machine (IDPS) to detect and block sport primarily based attacks.

5. Security Awareness Training: Train customers to apprehend phishing attempts, keep away from clicking on suspicious links, and take a look at consistent practices.

6. Device Monitoring:

- Traffic Analysis: Ensure that there are ordinary tests on uncommon traffics that could enter into the groups and additionally unauthorised sports activities.
- Behavioral Analysis: Conductivity of device may be checked the use of analyzing gear.

7. Access Control:

- Least Privilege Principle: Limit person and tool permissions to the minimum essential for his or her functions.
- Regular Access Reviews: Periodically review and update get entry to permissions..

8. Network Monitoring Tools: Know community sports using a device that gives visibility, see any unusual pattern or anomaly.

9. Encrypted Communication:Ensure that information transmitted among gadgets is encrypted the use of stable protocols.

10. Intrusion Prevention Systems (IPS): Real-Time Threat Prevention: Deploy IPS to detect and prevent known and unknown threats in real time.

Detecting botnets poses a significant challenge due to their minimal use of computational resources, making them elusive and hard to identify. In recent years, network security experts have extensively explored the identification and tracking of botnets.

1.1.5 Anomaly-based Detection Technique

Several hours are expended on studies of new algorithms for the identification of botnets, taking into account indicators of Internet traffic. Botnet detection based upon anomalous network behavior considers unusual latency delays, NetFlow on atypical and unused ports, heavy traffic load to a semi-network, or irregular structure behaviors that might indicate rogue elements in the network.It especially enables the identification of zero-day botnet attacks that have no signatures or patterns. It is also capable of identifying botnets with advanced evasion tactics, like cryptography, camouflage, and dynamism, by interpreting the changing behaviors as abnormalities.

Nevertheless, anomaly-based detection is not without its challenges. "False positives" can also arise from the way the system responds to an alert or from typical variations in network traffic. An increase in network traffic, for instance, brought on by regular occurrences like software updates and high user activity, may also seem unusual. Setting appropriate thresholds and improving anomaly detection algorithms are both necessary to reduce false positives. To do this, it uses historical data, including packet size, average traffic volume, and protocol distribution, to create baseline data.

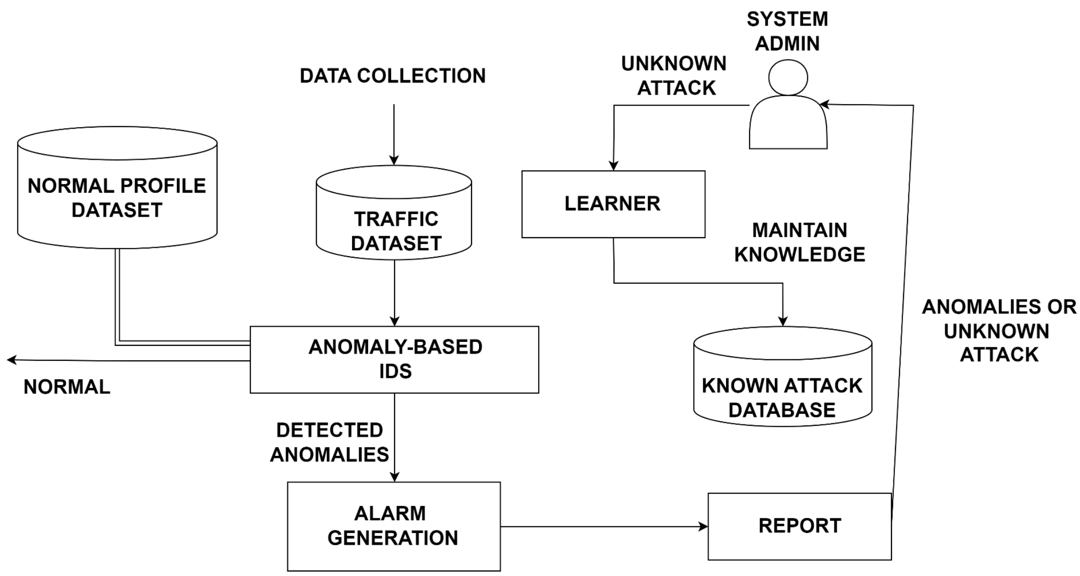


Fig 1.4 Anomaly-Based Detection System

1.2 Problem Statement

This project aims at developing an optimal approach for botnet attack detection within IoT systems via machine learning and deep learning algorithms. This would entail developing a means of analyzing and managing huge amounts of data produced in the form of network activity, log files, and even device conduct, detecting signs of botnets by recognizing such trends, behavior, and abnormalities.

The remedy should be employing ML model for instance supervised or unsupervised learning to detect botnets in real time but limiting the occurrence of faulty positive and faulty negative. The associated problems that come along with this problem statement include having to handle heterogeneity of data formats, limited computing resources, variable connectivity issues.

Another issue is that of dynamic botnet attack, botnet constantly changes its structure to avoid detection. Upon attainment of this undertaking, there will be an innovative approach to Botnet attack detection and suppression by using Machine learning algorithms within IoT.

In addition, such outcomes may become the groundwork for subsequent machine learning studies concerning the IoT botnet detection.

1.3 Objective

The objectives of the project are:

- This project seeks to determine whether a device is being attacked by a botnet.
- Developing methods for identifying botnet-induced denial-of-service attacks is the main goal of this work in an Internet of Things context.
- Developing an approach that will include designing and implementing an IoT data collection and preprocessing system.
- To build the best classification model using machine and deep learning algorithms to determine the optimal accuracy in predicting whether a system is under botnet detection or not.
- Optimising the models for real-time detection

1.4 Significance and Motivation of the Project Work

In researching cybersecurity, this is always a daunting assignment. Therefore, this has forced cybercriminals to remain on guard to look for ways of identifying vulnerabilities with which they can carry out unlawful activities. New and ingenious means for malware propagation are becoming popular. After that, malware is used to carry out secondary attacks like denial-of-service attacks and data exfiltration that target or use compromised systems.

1.5 Organization

The project overview is covered in this chapter, along with information on what a botnet attack is, how and why it occurs, modern methods for prevention and detection, the necessity of stopping or at least halting them, and our strategy for doing so through the use of machine learning technologies. The following is the report's structure.

In **Chapter 2 - Literature Survey,** To ascertain what makes our approach special and better than others, a brief review of other research papers on the topic will be studied.

In **Chapter 3 - System Development**, We are going to examine the conception, development, and application of our model before conducting an analysis.

In **Chapter 4 - Testing,** We are going to study the different evaluation metrics that we have used for evaluating our model accuracy.

In **Chapter 5 - Performance Analysis,** We are going to review our implemented model's performance statistics, outcomes, and output at different phases. We will also make a comparison between these findings and the previous models and hypotheses evolved.

In **Chapter 6 - Conclusion**, The project will be summarized, along with restrictions or extra work that may come up before, during, or after our model is put to use.

Chapter 2: Literature Survey

The proliferation of insecure IoT devices has inevitably brought about serious bot assault in the IoT networks. Cyber-attack detection mechanisms are deployed which help to continue the intended operation of connected devices and their data. Through the use of machine and deep learning algorithms, which are capable to recognize patterns and variances from a given data set, they are also capable of detecting such bot attacks in IoT data.

Like that of other sectors, the recent research in this field concentrates on utilizing machine learning algorithms to reveal botnet assaults performed by the Internet of Things. The research course must include different methods and techniques that are focused on the detection and prevention of a botnets', which threatens various IoT devices.

Botnet detection in IoT has also made use of deep learning algorithms, including recurrent neural networks (RNNs) and convolutional neural networks (CNNs). According to Tan et al. (2020), a CNN-based method for identifying botnet attacks in the Internet of Things by examining network traffic patters. The authors used the CNN model's learned features to detect botnet attacks with high accuracy. By examining the behavior of IoT devices, Singh et al. (2021) proposed an RNN-based method for botnet detection in the Internet of Things. The writers utilizing sequential data from the activities of IoT devices, RNNs were able to detect botnet attacks with high accuracy.

In order to simplify and boost the effectiveness of machine learning algorithms, feature selection and dimensionality reduction techniques are frequently employed in botnet detection in the Internet of Things.Yang et al. (2018) proposed a feature selection approach using information gain and recursive feature elimination to select the most relevant features for botnet detection in IoT.

The literature survey highlights that machine learning algorithms have been widely used for the detection of botnet attacks in IoT. Various techniques, including decision trees, support vector machines, neural networks, and ensemble learning methods, have been employed to detect botnet attacks in IoT data.

2.1 Overview of Relevant Literature

Title	**[1]**Ensemble Machine Learning Techniques for Accurate and Efficient Detection of Botnet Attacks in Connected Computers
Authors	Stephen Afrifa, Vijayakumar Varadarajan, Peter Appiahene, Tao Zhang and Emmanuel Adjei Domfeh
Year of Publication	2023
Summary	Communication is crucial, and the surge in IoT devices results from high internet message volumes. This paper employs quantitative, and qualitative approaches, and machine learning algorithms like random forest and a stacked prediction ensemble model for Internet of Things botnet detection. It suggests preventive actions, emphasizing importance of combating these attacks.

Title	**[2]**Botnet Attack Detection in IoT Using Machine Learning
Authors	Khalid Alissa, Tahir Alyas, Kashif Zafar, Qaiser Abbas, Nadia Tabassum and Shadman Sakib
Year of Publication	2022
Summary	Addressing cyber intrusions like botnets and the growing IoT devices, this study employs machine learning in one-class classification with balanced UNSW-NB15 datasets oversampled using SMOTE-Oversampling. The machine learning pipeline, comprising six steps, includes data analysis and preprocessing. Three models—XGBoost, logistic regression, and decision tree—are trained and evaluated for accuracy, F1 score, recall, and precision. Experiments show the decision tree's superior performance with a 94% test accuracy rate.

Title	**[3]**BOTNET Attacks Detection in Internet of Things Using Machine Learning.
Authors	Nookala Venu, Sanyasi Rao Allanki, Aarun Kumar
Year of Publication	2022
Summary	Network becomes vulnerable with growing dependence on rising IoT devices. Such a growth surge also puts the IoT networks at risk, rendering the devices even more vulnerable to attacks. The need for quick detection of such attacks grows rapidly. Attackers exploit vulnerabilities in devices so they can carry out BotNet attacks and ultimately develop more sophisticated DDoS attacks. Dataset challenges facing existing botnet detection. The work herein outlines common features in ML model with better performance than existing solutions and detection of botnets among various data sets.
Title	**[4]**Detection of Botnet Attacks against Industrial IoT Systems by Multilayer Deep Learning Approaches
Authors	Devrim Unal, Mohammad Hammoudeh, Mohammed Mudassir, and Farag Azzedin
Year of Publication	2022
Summary	The high rise in reliance on surgung IoT devices makes the networks vulnerable. With the larger exposure of the attack surface, IoT networks are prone to quick detection coupled with efficient response management. Attackers utilize botnet's ability to exploit device vulnerabilities in order to perform sophisticated DDOS incidences. Botnet detection currently has problems concerning datasets. The presented paper describes a unified feature set for the ML models that exceeds existing approaches of detecting botnet attacks using different datasets

Title	**[5]**IoT Security: Botnet detection in IoT using Machine learning
Authors	Satish Pokhrel, Robert Abbas, Bhulok Aryal
Year of Publication	2021
Summary	As devices used when accessing IoT applications increase, hackers become interested in conducting cyber-attack, which makes security the most significant concern. The proposed model makes use of KNN, NB and MLP ANNs with feature engineering and smart oversampling technique (SMOTE) for performance evaluation of both imbalanced and balanced classes datasets.

Title	**[6]**A Two-Fold Machine Learning Approach to Prevent and Detect IoT Botnet Attacks
Authors	Faisal Hussain; Syed Ghazanfar Abbas; Ivan Miguel Pires; Sabeeha Tanveer; Ubaid U. Fayyaz
Year of Publication	2022
Summary	Increased dependency on the internet has accelerated growth and exposure of IoT devices creating network vulnerability. The increased attack surface generated by this surge leaves IoT networks and devices increasingly vulnerable. This implies that it calls for the speedy detection and management of effective attacks. Attackers take advantage of flaws and vulnerabilities in IoT devices which form Botnet attacks resulting into advanced DDoS attacks. The challenges facing the existing botnet detection approaches are mainly attributed to data set problems.

Title	**[7]** Botnet Attack Detection by Using CNN-LSTM Model for Internet of Things Applications
Authors	Hasan Alkahtani and Theyazn H. H. Aldhyani
Year of Publication	2022
Summary	The internet of things is expanding at a quick pace causing security concerns making it necessary to come up with an IoT algorithm that involves a cnn lstm algorithm. The algorithm detects various BASHLITE and Mirai botnets targeted at commercial IoT devices with superior accuracies as high as 90.88% for doorbells and 89.64% for security cameras. EmpIRical experimentation uses the RN-BaIoT dataset, whereby the algorithm proves useful.

Title	**[8]** Machine Learning-Based IoT-Botnet Attack Detection with Sequential Architecture
Authors	Yan Naung Soe, Yaokai Feng, Paulus Insap Santosa, Rudy Hartanto and Kouichi Sakurai
Year of Publication	2020
Summary	With the spreading of IoT devices, there are various types of bots like botnets which contribute largely to cyber attacks.This article proposes a sequence architecture ML-based botnet detection system. The results achieved almost 99% accuracy with J48 decision tree, ANN, and Naïve Bayes ML algorithms, thus confirming the applicability of the suggested structure for detecting novel attack types.

Title	**[9]**Performance evaluation of Botnet DDoS attack detection using machine learning
Authors	Tong Anh Tuan, Hoang Viet Long, Le Hoang Son, Raghvendra Kumar, Ishaani Priyadarshini & Nguyen Thi Kim Son
Year of Publication	2020
Summary	Botnets; major vulnerabilities in internet attacks and use of ANN, SVM, DT, NB, and USML for the combating of distributed denial of service. The study evaluates in terms of the KDD99 and UNBS-NB 15 dataset. Remarks about KDD99 indicate better results which are indicative of computer security. The system provides a benchmark of botnet DDoS attacks based on ML.
Title	**[10]**Botnet Attack Detection Using Machine Learning
Authors	Mustafa Alshamkhany, Wisam Alshamkhany, Mohamed Mansour, Salam Dhou
Year of Publication	2020
Summary	This paper uses the Bot-IoT and UNSW datasets to detect botnets and evolving security threats using machine learning. Four classifiers are used: NB, KNN, SVM, Decision Trees. Using 82 thousand records from the UNSW-NB15 dataset, Decision Tree model was used to detect botnet attacks. It achieved an unprecedented accuracy of 99.89% and precision of 100%.

2.2 Key Gaps in the Literature

[1] The paper lacks discussion on the practical challenges and limitations of implementing the suggested preventive measures in diverse IoT environments. This limitation restricts a comprehensive understanding of the proposed solutions' real-world applicability and effectiveness, highlighting the need for further research and practical validation. Considering the variability and complexity of IoT ecosystems, addressing practical implementation challenges is crucial for ensuring the feasibility and success of proposed cybersecurity measures. In-depth analysis and testing in diverse IoT scenarios are necessary to validate the efficacy of preventive measures and their ability to mitigate evolving cyber threats effectively.

[2] While useful for botnet detection, this paper's focus on a single model and configuration may overlook nuances in attack types across diverse Internet-of-Things environments. This limitation calls for broader model exploration and validation to ensure comprehensive threat coverage. Exploring various models, configurations, and datasets can enhance the research's robustness and applicability to real-world IoT security challenges. It's crucial to consider the variability and complexity of IoT ecosystems to develop effective and adaptive botnet detection systems that can address evolving cyber threats effectively.

[3] The potential limitations of this study stem from its exclusive focus on the UNSW-NB15 dataset, which restricts its generalizability to other datasets and real-world scenarios. Additionally, while the study acknowledges dataset imbalance, it lacks specific recommendations to address this issue. There is a clear need for more research to enhance robustness against adversarial attacks. However, practical deployment challenges and scalability considerations are not thoroughly explored, and the absence of open-source implementations hinders reproducibility and wider adoption of proposed solutions. Addressing these gaps would strengthen the study's impact and practical relevance in the field of cybersecurity for IoT environments.

[4] Using a publicly available dataset limits the real-life applicability of the research findings. The paper lacks explicit discussions on dataset properties and biases, hindering generalization of proposed criteria for diverse IoT environments. Understanding dataset characteristics is crucial for assessing research validity and reliability. Acknowledging biases is essential for ensuring proposed criteria effectiveness across IoT scenarios. Thorough discussions on dataset properties and biases would enhance research transparency and credibility.

[5] The feature engineering process plays a crucial role in enhancing botnet detection capabilities, but it can be complex, particularly when applied to IoT devices. This approach requires a delicate balance between comprehensive evaluation of features and practical implementation feasibility to ensure adaptability and effectiveness in real-world scenarios. The challenge lies in identifying relevant features that capture meaningful patterns indicative of botnet activity while considering the computational constraints and resource limitations inherent in IoT environments. Striking this balance is essential to develop robust and scalable botnet detection solutions that can effectively safeguard IoT ecosystems against evolving cyber threats.

[6] The improvements achieved in our models on the BoT-IoT dataset may not seamlessly transfer to diverse IoT scenarios, leading to limitations in the generalizability of our findings. This necessitates the development of customized methods tailored to specific contexts within the IoT ecosystem. Each IoT environment may present unique challenges and characteristics that demand specialized approaches for effective threat detection and mitigation. Therefore, while our advancements on the BoT-IoT dataset are valuable, they serve as a starting point for further research and adaptation to varied IoT deployment scenarios.

[7] The suggested CNN LSTM model shows promising accuracy in identifying attacks on IoT devices, particularly those targeting doorbell devices. However, it faces limitations in detecting certain types of attacks, such as Scan and TCP flooding attacks. This underscores the challenges in effectively addressing cybersecurity threats, especially in the constantly evolving landscape of cyberattacks. The model's success in specific scenarios highlights the potential of advanced machine learning techniques in enhancing security measures but also emphasizes the ongoing need for innovation and adaptation to combat a wide range of cyber threats effectively.

[8]The educational algorithms within the machine mastering model are not entirely tailored to the proposed framework, potentially causing them to struggle in adapting to evolving attack techniques. The system lacks explicit handling of learning processes or adaptive changes required by this model, which could limit its capacity to respond effectively to emerging threats. As a result, there may be gaps in the model's capability to mitigate and detect new and sophisticated attacks, highlighting the importance of ongoing refinement and adaptation of machine learning algorithms within cybersecurity frameworks.

[9]The filter-based feature selection mechanism may introduce bias, affecting overall performance. Relying on statistical measures like Information Gain or Chi-square could overlook relevant features crucial for accurate DDoS attack detection. This bias could lead to false positives or negatives, compromising the detecting system's effectiveness. Therefore, careful consideration and validation of feature selection methods are essential to ensure robust and reliable DDoS detection capabilities.

[10]Certain classifiers, especially ones like Support Vector Machines (SVMs) that demand significant computational resources, may face challenges in real-time applications or large-scale deployment. This is due to their computational complexity and resource-intensive nature, leading to scalability issues, longer processing times, and higher resource utilization. These challenges are particularly pronounced in environments with constrained computational resources or demanding throughput requirements, highlighting the need for efficient algorithmic optimizations and hardware support in such contexts.

Chapter 3: System Development

3.1 Requirements and Analysis

Let us look at the libraries and platforms that we employed in the development of our project.

3.1.1 Python

Python is a popular, object-oriented, highly motivated, and highly interactive programming language that supports HLL. It is a garbage-gathering programming language that has dynamic typing. Around 1985–1990, Guido van Rossum designed it. It is a powerful and versatile programming language that is easy to learn, making it a fascinating choice for developing applications.

It's syntax, dynamic typing, and interpreted nature make it the ideal language for scripting and rapid software development. It supports a wide variety of programming patterns, including imperative, practical, and item-oriented programming patterns.

3.1.2 Numpy

The Python module NumPy is used to handle arrays. Matrix multiplication exercises, the Fourier transform, and matrices are also included. Travis Oliphant founded NumPy in 2005. We could use Numpy because it is an open-supply tool. The acronym for Numerical Python is NumPy.

NumPy adds more computational power to Python by integrating FORTRAN and C. The NumPy module in Python allows us to paint with multidimensional arrays and matrices. It is useful for mathematical or medical operations due to its speed and performance. NumPy also has linear algebra and sign processing features. NumPy provides tools for generating random numbers and sampling from various probability distributions, making it useful for simulations and statistical analysis. NumPy allows us to read and write data from/to files, including CSV files, binary files, and more, facilitating data handling and interoperability with other formats.

3.1.3 Pandas

An information evaluation program with a Python core is called Pandas. A strong and flexible tool for mathematical modeling was needed, so Wes McKinney founded Pandas in 2008. One of the most used Python programs right now is called Pandas. Pandas serves as a foundation of Key Python libraries. We can create a chart with little code because the Plot() function integrates multiple Matplotlib exercises into a single method. This integration allows users to create various types of charts and plots with minimal code, streamlining the process of visualizing data and gaining insights.

Pandas provides robust support for time series data analysis, including date/time indexing, resampling, and time zone handling. This makes it well-suited for analyzing temporal data trends and patterns.

3.1.4 Sklearn

The most reliable and practical Python machine learning library is called Scikit-learn, or Sklearn. Through a Python consistency interface, it offers a range of effective tools for statistical modeling and machine learning, such as regression, clustering, classification, and dimensionality reduction.

It supports all of the device study techniques, including random forests, ok-way clustering, logistic regression, linear regression, and selection timber.

3.1.5 Matplotlib

A well-liked Python graph charting tool for data science and device learning applications is called Matplotlib. Matplotlib is the primary plotting library used by Seaborn, but it also includes a few extra functions to enhance the visual appeal and usability of the graphs.

Matplotlib seamlessly integrates with Pandas, enabling easy creation of plots directly from DataFrame and Series objects, streamlining the data visualization process.Matplotlib is scalable and can handle large datasets, making it suitable for both small-scale and enterprise-level projects with diverse data visualization needs.

3.1.6 Logistic Regression

For binary classification tasks, such as estimating the likelihood that an instance will belong to one of two classes, statistical methods such as logistic regression are employed.

The relationship between the independent variables (features) and the likelihood of an outcome occurring is modeled by logistic regression. It accomplishes this by fitting the observed data to a logistic curve. A function is used in Logistic Regression referred to as sigmoid function used to convert predicted values into probabilities.
This function converts any real value lying between 0 and 1 to another value.
This function has precisely one inflection point and a non-negative derivative at each point.

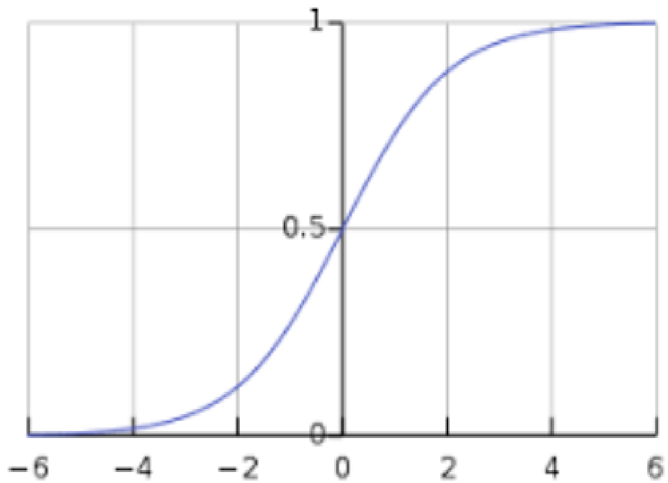


Fig 3.1.1 Sigmoid curve

3.1.7 Decision Tree

A decision tree is a well-known machine learning algorithm that is used for both classification and regression tasks. In this structure, which resembles a flowchart, each internal node stands for a "decision" made in response to a feature, each branch for the decision's result, and each leaf node for the ultimate choice or forecast.

It moves through the tree from the root node to a leaf node in order to generate a prediction for new instance. Based on the value of a feature, each internal node makes a decision before moving on to the next node by following the corresponding branch.

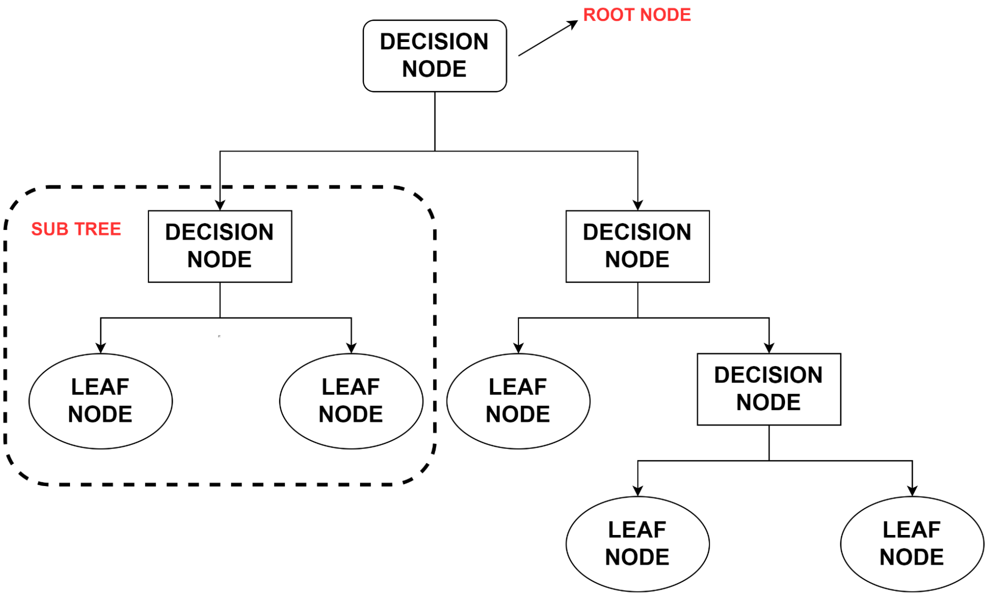


Fig 3.1.2 Decision tree Classifier

3.1.8 Random Forest

Based on decision trees, Random Forest is an effective ensemble learning technique. It is extensively utilized in machine learning for tasks involving both regression and classification. Random Forest bootstraps the training dataset to create multiple decision trees. Bootstrapping is the process of generating multiple new datasets of the same size as the original by randomly sampling the training data with replacement. Predictions are created by combining the predictions of each decision tree after they have all been constructed. The class that receives the most votes (the mode) among the trees is the one that is ultimately predicted for classification tasks.

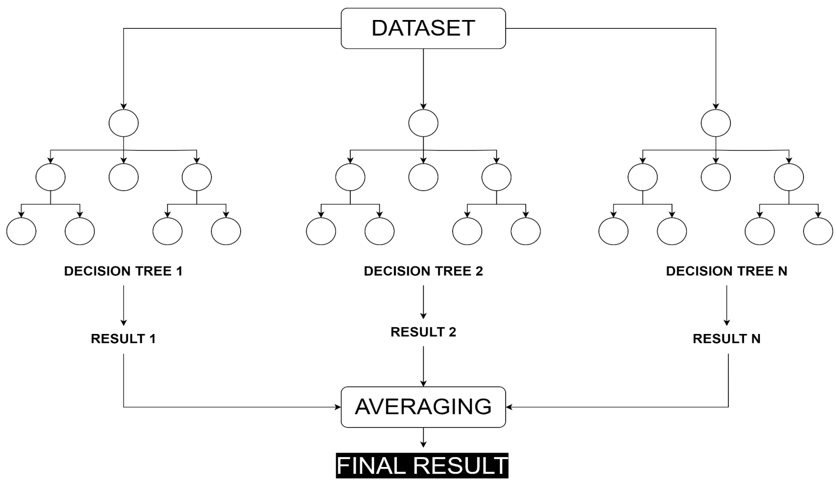


Fig 3.1.3 Random Forest

3.1.9 XgBoost

The powerful ensemble learning method known as gradient boosting machines is implemented in an efficient and scalable way by XGBoost, or eXtreme Gradient Boosting. It has gained popularity for its performance and adaptability in numerous machine learning competitions and real-world applications. It is widely used for both classification and regression tasks.

The gradient boosting framework, on which XGBoost is based, aims to iteratively add new models (usually decision trees) to an ensemble, each of which corrects the mistakes made by the earlier models. XGBoost is widely adopted in various domains such as finance, healthcare, marketing, and computer vision, showcasing its versatility and effectiveness in solving complex predictive modeling tasks.

XGBoost is an open-source library available in multiple programming languages, including Python, R, Java, and Scala, making it accessible to a broad community of developers and data scientists.XGBoost uses tree pruning to control model complexity and reduce computation time, resulting in faster training and prediction speeds.

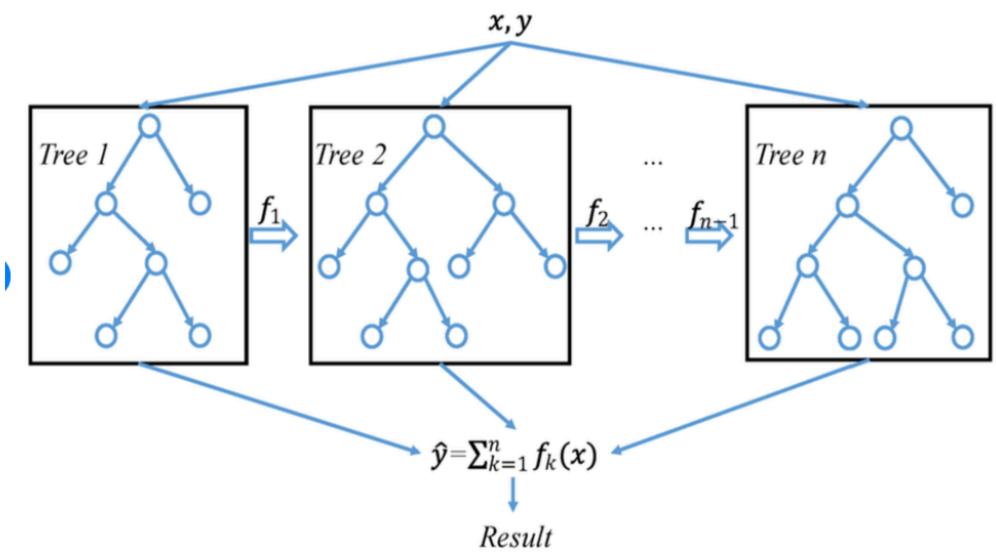


Fig 3.1.4 XgBoost architecture

3.1.10 Ensembling Method in Machine Learning

Ensemble learning is a machine learning technique that combines predictions from several models to improve forecasting accuracy and resilience. It uses the collective intelligence of the ensemble to reduce errors or biases that can be present in individual models. Ensemble learning has shown to be a strong technique in a variety of disciplines, providing more robust and trustworthy forecasts by efficiently combining predictions from numerous models.

The simple ensembling techniques are:

1. Max Voting
2. Averaging
3. Weighted Averaging

Max Voting:

Max Voting ensembling technique is generally used for classification tasks. The multiple models are used to make predictions on each data point and the predictions that we get from the majority of the models are considered as final prediction.

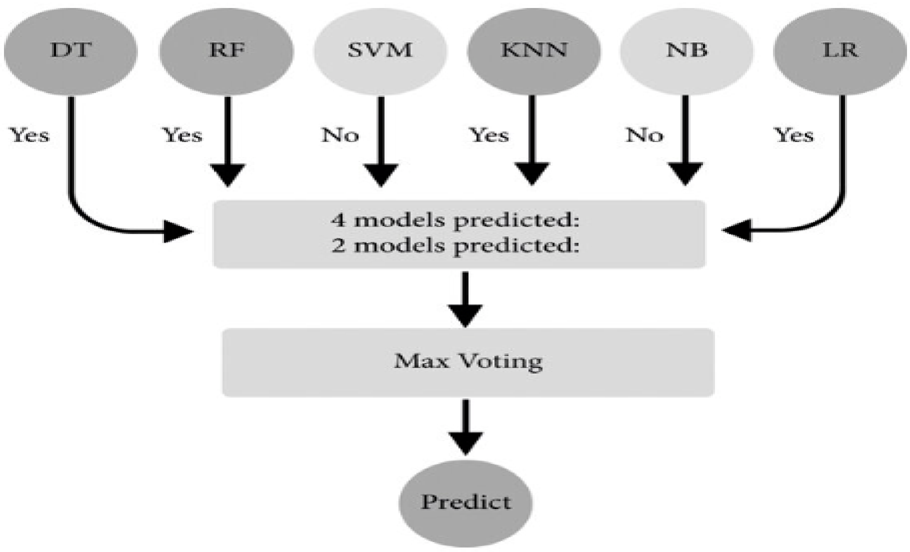


Fig 3.1.5 Max voting

Averaging:

In averaging, multiple predictions are made from each data point and we take average of predictions from all the models and use it as the final prediction.

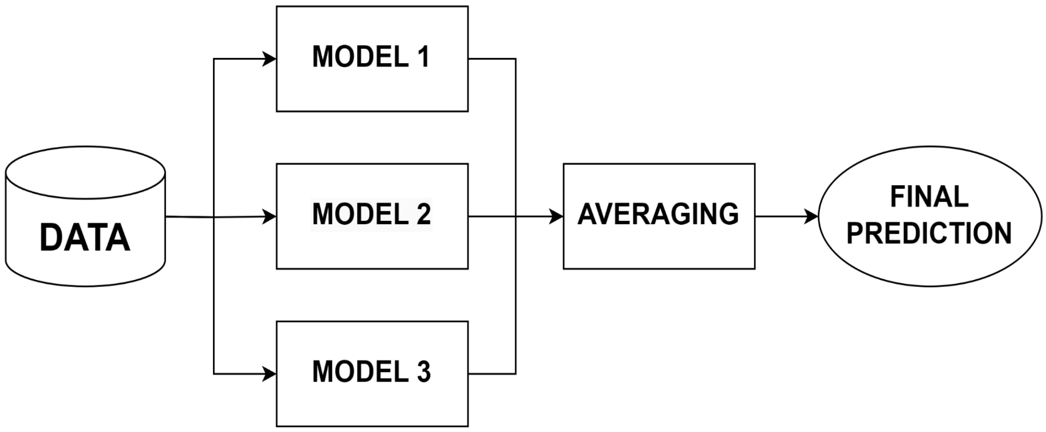


Fig. 3.1.6 Averaging Method

Weighted Averaging:

In this all the models are assigned different weights according to the importance of each model for prediction. We multiply each models prediction by the weight and then sum up the weighted predictions to obtain the final prediction.

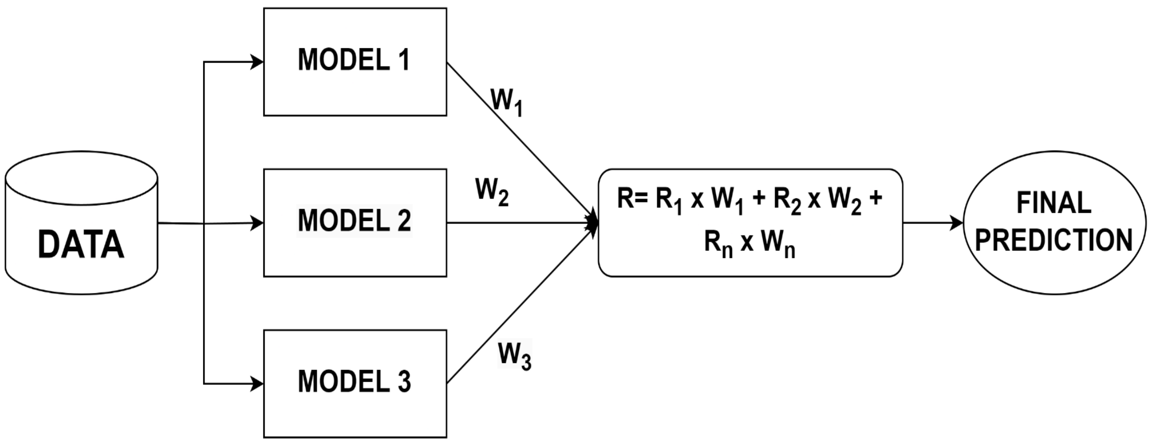


Fig. 3.1.7 Weighted averaging

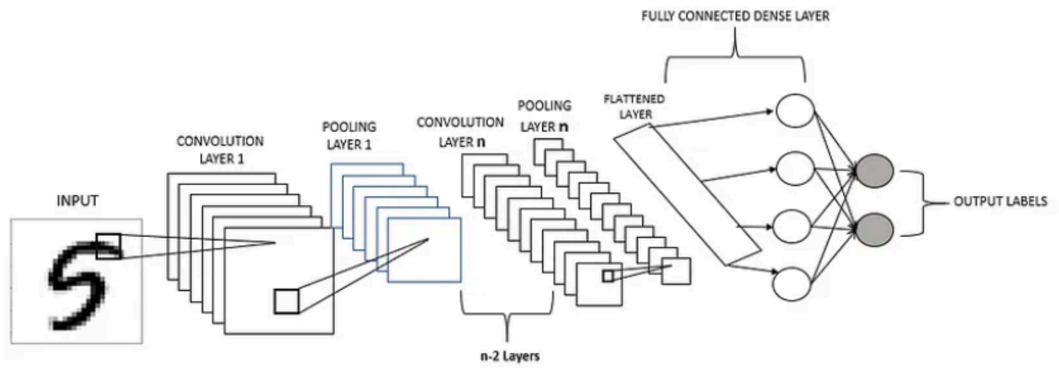
3.1.11 CNN

CNN stands for Convolutional Neural Network. It's a kind of deep neural network that's mostly employed for image analysis. For tasks like object detection, image classification, and image recognition, CNNs are particularly well-suited.

CNNs are made up of several layers, the main components of which are convolutional layers. In these layers, the input image is subjected to a set of learnable filters, also known as kernels, by the network.

Every filter applies convolution operations to the input image in order to extract various features, including textures, edges, and more intricate patterns. To add non-linearity to the network and help it understand intricate relationships in the data, non-linear activation functions such as Rectified Linear Unit, or ReLU, are applied following each convolutional and pooling layer. CNNs often leverage data augmentation techniques (e.g., rotation, flipping, scaling) to increase the diversity of training data and improve model generalization.

A softmax layer is added at the end of the network to transform the class probabilities from the raw scores generated by the preceding layers in classification tasks. The probability distribution across all classes is represented by the softmax function's output, and each node in the softmax layer corresponds to a class.



3.1.8 Convolutional Neural Network

3.1.12 RNN

Recurrent Neural Network is referred to as RNN. This kind of artificial neural network is intended to handle time-series or sequential data, where the elements' sequence holds significant information. Time-series prediction, speech recognition, natural language processing (NLP), and other sequential data-related tasks are among the many applications for RNNs.

Recurrent connections enable RNNs to remember information about prior inputs, in contrast to feedforward neural networks, which process input data in a single pass.The network's output at one time step becomes a portion of the network's input at the subsequent time step due to this recurrent connection, creating a loop.

There are different types of RNN architectures, including vanilla RNNs, Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRUs), each with variations in handling memory and learning long-term dependencies. RNNs can be implemented using deep learning frameworks like TensorFlow, PyTorch, Keras, and MXNet, providing tools and libraries for building, training, and deploying RNN models.

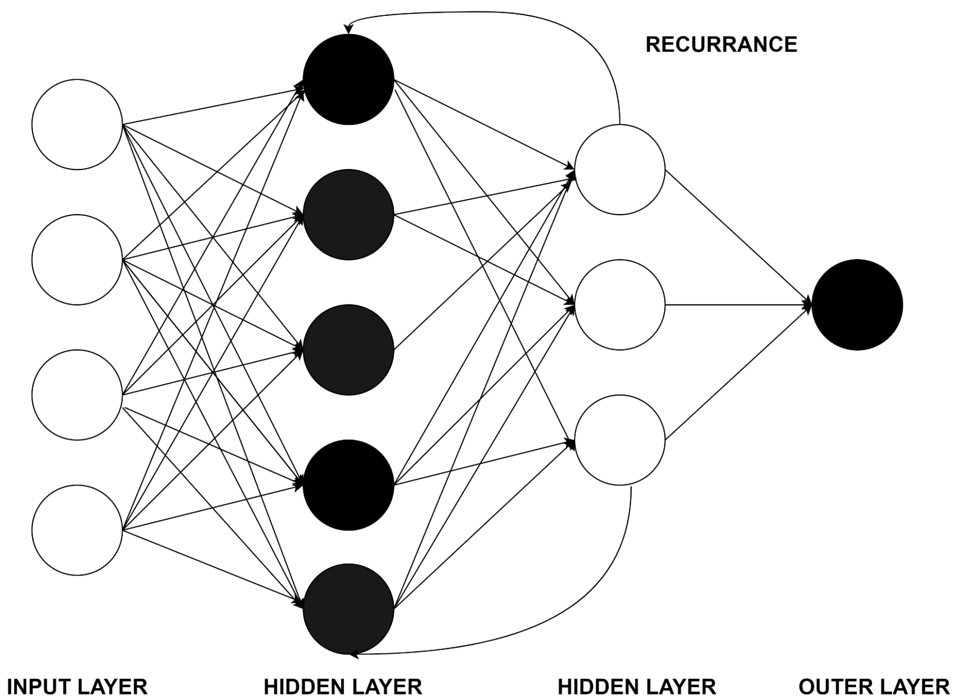


Fig 3.1.8 Recurrent Neural Network

3.2 Project Design and Architecture

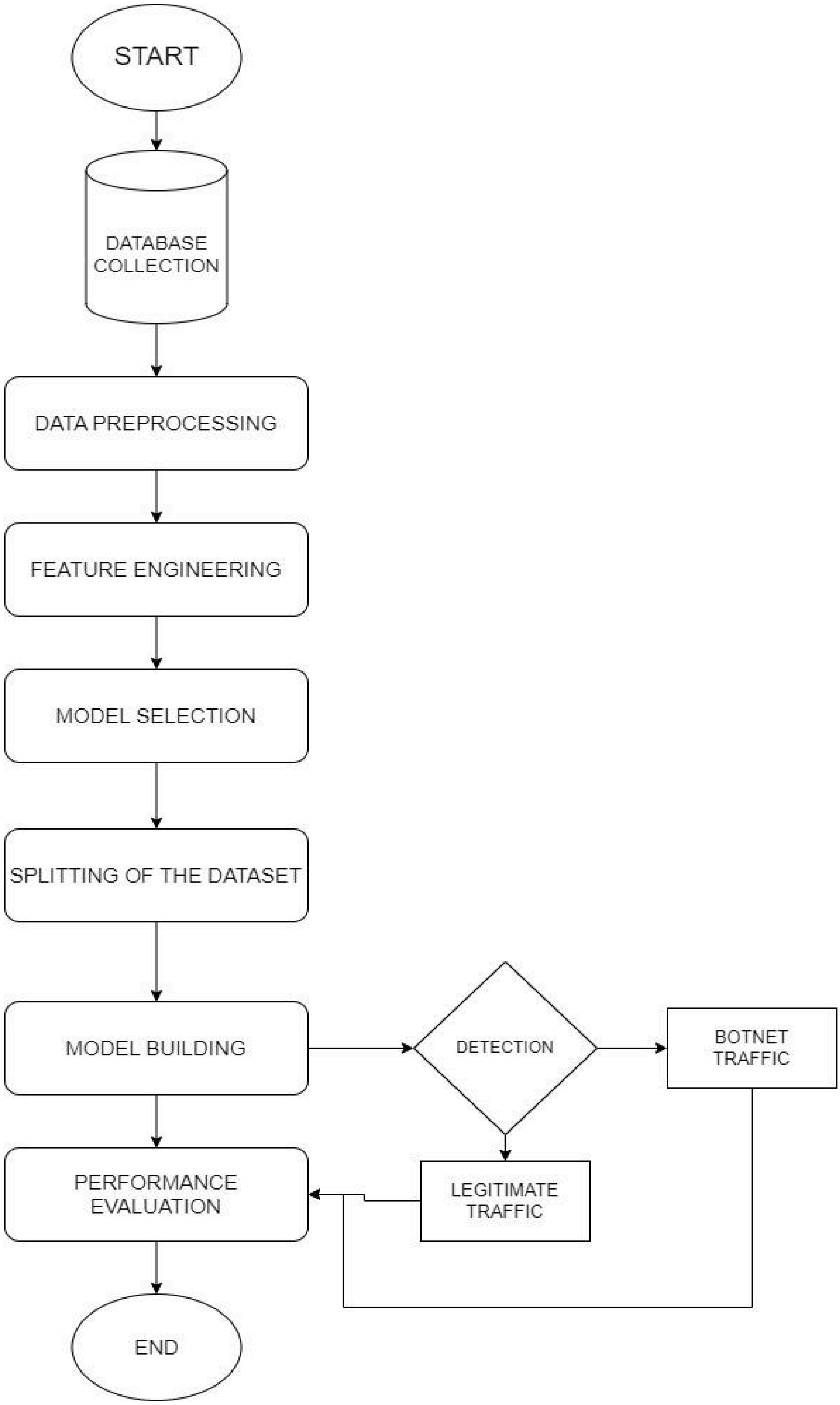


Fig 3.2.1 Flowchart

3.3 Data Preparation

Data Collection:

- Collected the dataset from Kaggle repository, The UNSW-NB15 dataset has been used in our project.
- The dataset consists of 44 features and and 2.5 million records in which we have used 2,57,673 records which are labeled as either attack traffic or normal traffic and further expanded to the category of attack and the subcategory.
- This dataset includes a wide range of network attributes, including protocol types, source and destination IP addresses, service-related information, and timestamps.
- Numerical data was coded using categorical information such as "proto," "service type," "state," "sptks," "sload," and "attack cat."

Data Labeling:

- The dataset has only two classes that is 0 and 1 that indicates that this dataset is for binary class classification.
- The dataset is imbalanced as we have more number of 0s in our dataset than 1.

Data Splitting:

- Divided the dataset into two training and testing subsets. 70% of the data was used for training and the rest 30% of the data was used for testing.

id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	dload	sloss
1	0.000011	udp	-	INT	2	0	496	0	90909.09	254	0	1.8E+08	0	0
2	0.000008	udp	-	INT	2	0	1762	0	125000	254	0	8.81E+08	0	0
3	0.000005	udp	-	INT	2	0	1068	0	200000	254	0	8.54E+08	0	0
4	0.000006	udp	-	INT	2	0	900	0	166666.7	254	0	6E+08	0	0
5	0.00001	udp	-	INT	2	0	2126	0	100000	254	0	8.5E+08	0	0

dloss	sinpkt	dinpkt	sjit	djit	swin	stcpb	dtcpb	dwin	tcprtt	synack	ackdat	smean	dmean	trans_dep
0	0.011	0	0	0	0	0	0	0	0	0	0	248	0	0
0	0.008	0	0	0	0	0	0	0	0	0	0	881	0	0
0	0.005	0	0	0	0	0	0	0	0	0	0	534	0	0
0	0.006	0	0	0	0	0	0	0	0	0	0	450	0	0
0	0.01	0	0	0	0	0	0	0	0	0	0	1063	0	0

response_	ct_srv_src	ct_state_t	ct_dst_ltm	ct_src_dpc	ct_dst_spc	ct_dst_src	is_ftp_logi	ct_ftp_cm	ct_flw_htt	ct_src_ltm	ct_srv_dst	is_sm_ips_	attack_cat	label
0	2	2	1	1	1	2	0	0	0	1	2	0	Normal	0
0	2	2	1	1	1	2	0	0	0	1	2	0	Normal	0
0	3	2	1	1	1	3	0	0	0	1	3	0	Normal	0
0	3	2	2	2	1	3	0	0	0	2	3	0	Normal	0
0	3	2	2	2	1	3	0	0	0	2	3	0	Normal	0

Fig 3.3.1 Features of Dataset

3.4 Implementation

➢ IMPORTING LIBRARIES:

The model was implemented and the dataset was trained using the following libraries:

```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb

from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import  classification_report
from sklearn.ensemble import RandomForestClassifier
```

Fig 3.4.1 Importing Libraries

A file in Python is considered as a module. It needs to be implemented using the import keyword before it is used. We need not to import all the functions only the necessary functions can be implemented using the “from” keyword.Using import and module name which is the filename or library to be used all module can be imported.

➢ IMPORTING DATASET:

```
#uploading file

from google.colab import files
uploaded = files.upload()
```

```
Choose Files UNSW_NB...ining-set.csv
• UNSW_NB15_training-set.csv(text/csv) - 15380800 bytes, last modified: 3/16/2024 - 100% done
Saving UNSW_NB15_training-set.csv to UNSW_NB15_training-set.csv
```

```
#uploading file
from google.colab import files
uploaded = files.upload()
```

```
Choose Files UNSW_NB...sting-set.csv
• UNSW_NB15_testing-set.csv(text/csv) - 32293018 bytes, last modified: 3/16/2024 - 100% done
Saving UNSW_NB15_testing-set.csv to UNSW_NB15_testing-set.csv
```

Fig 3.4.2 Importing Dataset

Importing the UNSW_NB15 training and testing datasets from the Kaggle repository and merging them. There are 83223 rows in the training dataset and 175341 rows in the testing dataset.

➢ PREPROCESSING OF THE DATASET:

```
#Performing label encoding to convert categorical data to numerical data
for col in ['proto', 'service', 'state']:
    df[col] = df[col].astype('category').cat.codes

df['attack_cat'] = df['attack_cat'].astype('category')
```

```
# Counting and visualizing the distribution of attack categories for labeled (label=1) data using a pie chart
validAttacks = df[df['label']==1]['attack_cat'].value_counts()
print(validAttacks)

plt.figure(figsize = (15,8))
#plt.pie(validAttacks,labels = validAttacks.index, autopct = '%1.1f%%',explode = [0,0,0,0,0,0.2,0.2,0.2,0.2,1.2])
plt.show()
```

Fig 3.4.3 Preprocessing of the Dataset

Preprocessing is carried out on the dataset. Some features in the dataset consists of the categorical data which were converted to numerical data. Next, the distribution of attack categories is counted and visualized. Important features are updated and the covariance matrix is computed.

➢SPLITTING THE DATASET INTO TRAINING AND TESTING:

```
#Splitting the data into training and testing and then dispalying their dimesnions
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=11)

feature_names = list(X.columns)

print("X_train shape: ", X_train.shape)
print("y_train shape: ", y_train.shape)
print("X_test shape: ", X_test.shape)
print("y_test shape: ", y_test.shape)
```

Fig 3.4.4 Splitting dataset into Training & Testing

As the next step, we need to divide our dataset into training and testing. The ratio we had taken is 70:30 which means that the model is trained in the 70% of the dataset and the rest is used for testing. Then the machine learning algorithm are used to generate the predictions based on the data that was not used in the training, their performance is evaluated

➢ IMPLEMENTATION OF LOGISTIC REGRESSION

▪ IMPORTING LOGISTIC REGRESSION

```
# Logistic Regression
# fitting linear model with coefficients to minimize residual sum of squares between the observed targets
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
```

Fig - 3.4.5 Implementing Logistic Regression

This code creates a logistic regression model, then it trains it on the training data (X_train for features, y_train for labels). After training, the model is ready to make predictions.

▪ ACCURACY, MATRICES, CONFUSION MATRIX OF LOGISTIC REGRESSION

```
y_pred_logreg = logreg.predict(X_test)


# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred_logreg)
print(f"Accuracy: {accuracy:.2f}")
print('\n')
# Print classification report (precision, recall, f1-score, etc.)
print("Classification Report:")
print(classification_report(y_test, y_pred_logreg))
print('\n')
cm = confusion_matrix(y_test, y_pred_logreg)
plt.figure(figsize=(3, 2))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Fig 3.4.6 Implementing Confusion Matrix, Accuracy and Classification Report for LR

In this code snippet, the logistic regression model (logreg) is trained to predict labels for the test dataset. We have calculated the accuracy of the model, and evaluated the classification report which includes metrics like precision, recall and F1-score. We then generated a confusion matrix to visualize the result.

➢ IMPLEMENTATION OF DECISION TREE

▪ IMPORTING DECISION TREE

```
# Decision Tree
dectree = DecisionTreeClassifier()
dectree.fit(X_train,y_train)
```

Fig 3.4.7 Implementation of Decision Tree

This code creates a decision tree classifier, then it trains it on the training data (X_train for features, y_train for labels). After training, the model is ready to make predictions.

▪ ACCURACY, MATRICES, CONFUSION MATRIX OF DECISION TREE

```
y_pred_dectree = dectree.predict(X_test)


# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred_dectree)
print(f"Accuracy: {accuracy:.2f}")
print('\n')

print("Classification Report:")
print(classification_report(y_test, y_pred_dectree))
print('\n')
cm = confusion_matrix(y_test, y_pred_dectree)
plt.figure(figsize=(3, 2))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Fig 3.4.8 Implementing Confusion Matrix, Accuracy and Classification Report for DT

In this code snippet, the decision tree model (dectree) is trained to predict labels for the test dataset. We have calculated the accuracy of the model, and evaluated the classification report which includes metrics like precision, recall and F1-score. We then generated a confusion matrix to visualize the result.

➢ IMPLEMENTATION OF RANDOM FOREST

▪ IMPORTING RANDOM FOREST

```
#Applying random forest

# Initialize Random Forest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
clf.fit(X_train, y_train)
```

Fig 3.4.9 Implementing Random Forest

This code creates a random forest classifier, then it trains it on the training data (X_train for features, y_train for labels). After training, the model is ready to make predictions.

▪ ACCURACY, MATRICES, CONFUSION MATRIX OF RANDOM FOREST

```
# predictions on the test set
y_pred_random = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred_random)
print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")
print(classification_report(y_test, y_pred_random))
cm = confusion_matrix(y_test,y_pred_random )
plt.figure(figsize=(3, 2))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Fig 3.4.10 Implementing Confusion Matrix, Accuracy and Classification Report for RF

In this code snippet, the random forest model is trained to predict labels for the test dataset. We have calculated the accuracy of the model, and evaluated the classification report which includes metrics like precision, recall and F1-score. We then generated a confusion matrix to visualize the result. Sns.heatmap creates a heatmap of the confusion matrix using seaborn library.

➢ IMPLEMENTATION OF ENSEMBLE METHOD MAX VOTING USING LOGISTIC REGRESSION AND DECISION TREE

▪ IMPORTING MODELS

```
decision_tree_classifier = DecisionTreeClassifier(random_state=42)
logistic_classifier = LogisticRegression(max_iter=1000, random_state=42)

# Train the classifiers
decision_tree_classifier.fit(X_train, y_train)
logistic_classifier.fit(X_train, y_train)

# Make predictions
decision_tree_predictions = decision_tree_classifier.predict(X_test)
logistic_predictions = logistic_classifier.predict(X_test)
```

Fig 3.4.11 Importing models of LR and DT

This code trains two classifiers that are logistic regression and decision tree, then it trains it on the training data (X_train for features, y_train for labels). After training, the model is ready to make predictions.

▪ IMPLEMENTING MAX VOTING

```
# Implement Max Voting
def max_voting(predictions):
    return np.apply_along_axis(lambda x: np.argmax(np.bincount(x)), axis=0, arr=predictions)

# Aggregate predictions using Max Voting
ensemble_predictions = max_voting([decision_tree_predictions, logistic_predictions])
```

Fig 3.4.12 Implementing Max Voting

▪ ACCURACY, MATRICES, CONFUSION MATRIX OF MAX VOTING

```
# Evaluate ensemble model
ensemble_accuracy = accuracy_score(y_test, ensemble_predictions)
print("Ensemble Accuracy:", ensemble_accuracy)

binary_predictions = [1 if pred >= 0.5 else 0 for pred in ensemble_predictions]

# Calculate classification report
print("Classification Report:")
print(classification_report(y_test, binary_predictions))

# Calculate confusion matrix
print("Confusion Matrix:")
```

Fig 3.4.13 Implementing Confusion Matrix, Accuracy and Report for Max Voting

➢IMPLEMENTATION OF ENSEMBLE METHOD AVERAGING USING LOGISTIC REGRESSION AND DECISION TREE AND KNN

▪ IMPORTING MODELS

```
# Initialize classifiers
logistic_classifier = LogisticRegression(max_iter=1000, random_state=42)
knn_classifier = KNeighborsClassifier()
decision_tree_classifier = DecisionTreeClassifier(random_state=42)

# Train the classifiers
logistic_classifier.fit(X_train, y_train)
knn_classifier.fit(X_train, y_train)
decision_tree_classifier.fit(X_train, y_train)

logistic_predictions = logistic_classifier.predict(X_test)
knn_predictions = knn_classifier.predict(X_test)
decision_tree_predictions = decision_tree_classifier.predict(X_test)
```

Fig 3.4.14 Importing Models LR, KNN and DT

This code trains three classifiers that are logistic regression, decision tree and KNN, then it trains it on the training data (X_train for features, y_train for labels). After training, the model is ready to make predictions.

▪ IMPLEMENTING AVERAGING

```
# Implement Averaging
def averaging(predictions):
    return sum(predictions) / len(predictions)

# Aggregate predictions using Averaging
ensemble_predictions = averaging([logistic_predictions, knn_predictions, decision_tree_predictions])

ensemble_predictions_rounded = [round(pred) for pred in ensemble_predictions]
```

Fig 3.4.15 Implementing Averaging on Models LR, KNN and DT

In the above code, a function called averaging is used, which takes a list of predictions from multiple models as input and returns the average prediction. Each prediction in the list is assumed to be a numeric value representing the model's confidence or probability for a certain class.

After defining the averaging function, the code aggregates predictions from three different classifiers (Logistic Regression, K-Nearest Neighbors, and Decision Tree) using this averaging technique. The predictions from these classifiers are stored in the variables logistic_predictions, knn_predictions, and decision_tree_predictions

- ACCURACY, MATRICES, CONFUSION MATRIX OF AVERAGING

```
# Evaluate ensemble model
ensemble_accuracy = accuracy_score(y_test, ensemble_predictions)
print("Ensemble Accuracy:", ensemble_accuracy)

binary_predictions = [1 if pred >= 0.5 else 0 for pred in ensemble_predictions]

# Calculate classification report
print("Classification Report:")
print(classification_report(y_test, binary_predictions))

# Calculate confusion matrix
print("Confusion Matrix:")
cm = confusion_matrix(y_test,binary_predictions )
```

Fig 3.4.16 Implementing Confusion Matrix, Accuracy and Report for Averaging

In the above code, we print the accuracy of the ensemble method.We convert the ensemble predictions to binary format based on a threshold of 0.5. If the prediction is greater than or equal to 0.5, it's classified as 1 (positive), else, it is classified as 0 (negative). Then we calculate and print the classification report which is a summary of various classification metrics such as precision, recall and f1-socre. Then the model visualise the confusion matrix displaying the number of true positives, true negatives, false positives, and false negatives.

➢ IMPLEMENTATION OF XgBOOST

▪ IMPORTING XgBOOST

```
# Initialize XGBoost Classifier
clf = xgb.XGBClassifier(objective='multi:softmax', num_class=3, random_state=42)

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)
```

Fig 3.4.17 Implementing XgBOOST model

This code creates a XgBoost model, then it trains it on the training data (X_train for features, y_train for labels). After training, the model is ready to make predictions.

▪ ACCURACY, MATRICES, CONFUSION MATRIX OF XgBOOST

```
# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report (precision, recall, f1-score, etc.)
print("Classification Report:")
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test,y_pred_naive )
plt.figure(figsize=(4, 3))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Fig 3.4.18 Implementing Confusion Matrix, Accuracy and Classification Report for XgBOOST

In this code snippet, the XgBoost model is trained to predict labels for the test dataset. We have calculated the accuracy of the model using the accuracy_socre, and evaluated the classification report which includes metrics like precision, recall and F1-score. We then generated a confusion matrix to visualize the result. We have used the sns.heatmap to create the heatmap of the confusion matrix .

➢ IMPLEMENTATION OF CNN

▪ IMPORTING CNN

```
# Define the CNN model
model = models.Sequential([
    layers.Reshape((X_train.shape[1], 1), input_shape=(X_train.shape[1],)),
    layers.Conv1D(32, 3, activation='relu'),
    layers.MaxPooling1D(2),
    layers.Conv1D(64, 3, activation='relu'),
    layers.MaxPooling1D(2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
```

Fig 3.4.19 Implementing CNN model

This code defines a Convolutional Neural Network (CNN) model using the Keras API with TensorFlow backend.

▪ ACCURACY, MATRICES, CONFUSION MATRIX OF CNN

```
# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
y_pred_prob = model.predict(X_test)

# Convert predicted probabilities to class labels based on a threshold (e.g., 0.5)
y_pred_labels = (y_pred_prob >= 0.5).astype(int)

# Generate a classification report
print(classification_report(y_test, y_pred_labels))
cm = confusion_matrix(y_test, y_pred_labels)

# Plot the confusion matrix
plt.figure(figsize=(3, 2))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

Fig 3.4.20 Implementing Confusion Matrix, Accuracy and Classification Report for CNN

This code trains a Convolutional Neural Network (CNN) model, evaluates its performance on a test dataset, and visualizes the results using a confusion matrix. The model accuracy is evaluated and then the Classification Report is printed . Then we have displayed the confusion matrix to visualize the model's performance.

➢ IMPLEMENTATION OF RNN

▪ IMPORTING RNN

```
X_train_rnn = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_val_rnn = np.reshape(X_val, (X_val.shape[0], X_val.shape[1], 1))
X_test_rnn = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Define the RNN model
model = models.Sequential([
    layers.SimpleRNN(32, input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2]), return_sequences=True),
    layers.Dropout(0.2),
    layers.SimpleRNN(32),
    layers.Dropout(0.2),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Fig 3.4.21 Implementing RNN model

This code defines a Recurrent Neural Network (RNN) model using the Keras API with TensorFlow backend.

▪ ACCURACY, MATRICES, CONFUSION MATRIX OF RNN

```
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
y_pred_prob = model.predict(X_test)

# Convert predicted probabilities to class labels based on a threshold (e.g., 0.5)
y_pred_labels = (y_pred_prob >= 0.5).astype(int)

# Generate a classification report
print(classification_report(y_test, y_pred_labels))
cm = confusion_matrix(y_test, y_pred_labels)

# Plot the confusion matrix
plt.figure(figsize=(3, 2))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

Fig 3.4.22 Implementing Confusion Matrix, Accuracy and Classification Report for RNN

This code trains a Recurrent Neural Network (RNN) model, evaluates its performance on a test dataset, and visualizes the results using a confusion matrix. The model accuracy is evaluated and then the Classification Report is printed . Then we have displayed the confusion matrix to visualize the model's performance.

3.5 Key Challenges

1. Dataset Quality:

Managing the various features of Internet of Things devices, including their diverse data, constrained mathematical capabilities, and dynamic internet, is project's main challenges. As we have imbalance data which is a challenge in our project as imbalance data can lead to biased result that favor the majority class .

2. Nature of Botnet Attacks:

The dynamic nature of botnet attacks presents another difficulty because the networks constantly change and adapt in order to avoid detection. IoT devices may contain sensitive data, so it's also critical to protect the IoT data during the observation process.

3. Feature Engineering:

As our dataset has high number of features so we need to use dimension reduction technique to reduce the number of features. Extracting and recognising the important features in our project pose as a challenge in our project.

4. Handling Categorical Data:

The presence of categorical variables in the dataset posed a significant challenge because they needed to be transformed into a numerical format for analysis and modeling. Since categorical variables cannot be numerically valued by nature, they cannot be used with some machine learning algorithms that require numerical inputs.

5. Identifying optimal Machine Learning models:

The project presented a significant challenge in determining the best machine learning algorithm. Discovering which algorithm best fits the problem statement required testing a number of them due to the dataset's complexity and variety of features. Comprehending the features of the dataset, the subtleties of various algorithms, and their suitability for the given task posed a difficulty.

Chapter 4: Testing

4.1 Testing Strategy

In the testing phase, we have applied various evaluation techniques to calculate the performance of our model.

Evaluation metrics are quantitative measures that are used to assess the effectiveness and performance of a statistical or machine learning model. These metrics provide useful information about the model's performance when comparing different models or algorithms.

The predictive power, generalizability, and general quality of a machine learning model should all be considered when assessing it. Objective standards for measuring these elements are provided by evaluation metrics. The evaluation metrics selected will vary depending on the particular problem domain, data type, and intended result.

Confusion Matrix: A confusion matrix is a N*N matrix , in which N is the number of predicted classes. It is used to visualise the performance of a classification model. It has four terms that are true positive, false negative, true negative and false positive. Since for our problem, the N=2, so we obtain a 2*2 matrix.

The terms that are used in the confusion matrix are:

- **True Positive:** True Positive, which shows how many positive examples are correctly classified
- **True Negative:** A True Negative indicates how many negative examples were correctly classified.
- **False Positive:** If a model predicts a positive outcome for a negative instance then it is false positive
- **False Negative:** If a model predicts a negative outcome for a positive instance then it is false negative.

Accuracy: Accuracy is an evaluation metric which is used to find the overall performance of a classification model in Machine Learning. A high accuracy value indicates that the model is making correct predictions across all classes in the dataset.
The formula for calculating Accuracy is:

Number of correct Predictions/Total number of Predictions

Precision: Precision measures the proportion of true positive predictions among all positive predictions made by the model. A high precision value indicates that the model makes fewer false positive predictions, leading to more reliable positive predictions. The formula for calculating Precision is:

True Positive/(True Positives+False Positives)

Recall: Recall is an evaluation metrics and it the proportion of all the true positive instances among all the positive instances in the dataset.A high recall value implies that the model is successful in obtaining the positive instances, even if they are rare.The formula for calculating Recall is:

True Positive/(True Positive+False Negatives)

F1-Score: A common evaluation metric for classification tasks that combines recall and precision into a single number is the F1-score. When working with unbalanced datasets that have an uneven class distribution, it is extremely helpful. It is used in binary classification tasks, where there are two classes. It provides a single metric to assess the overall performance of a classifier in terms of both false positives and false negatives.The formula for calculating F1-score is:

2*(Precision*Recall)/(Precision+Recall)

4.2 Test Cases and Outcomes

4.2.1 Results of Logistic Regression

```
Accuracy: 0.90


Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.78      0.85     27814
           1       0.89      0.97      0.93     49488

    accuracy                           0.90     77302
   macro avg       0.91      0.87      0.89     77302
weighted avg       0.90      0.90      0.90     77302
```

Fig 4.2.1.1 Accuracy and Classification Report of LR Model

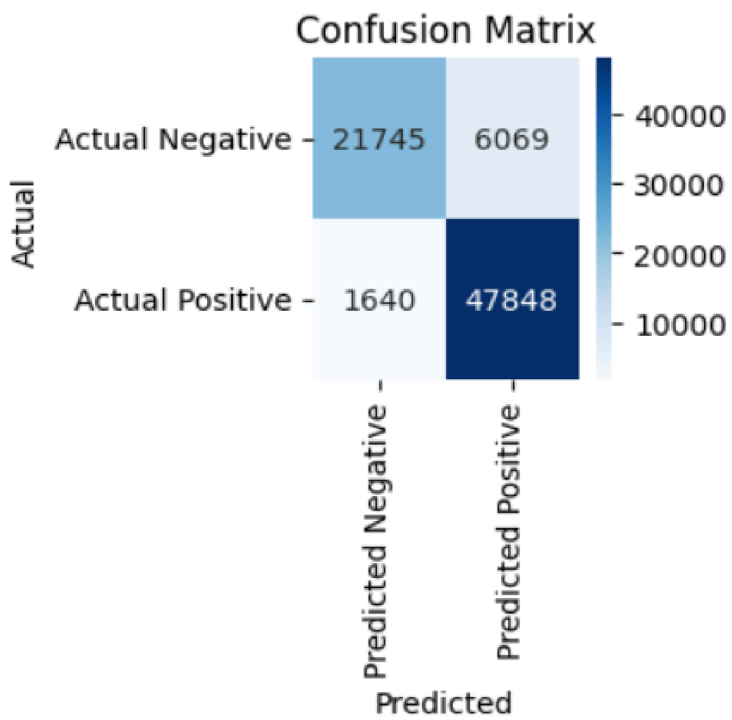


Fig 4.2.1.2 Confusion Matrix of LR Model

The accuracy achieved by the Logistic Regression is 90%

4.2.2 Results of Decision Tree

```
Accuracy: 0.94


Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91     27814
           1       0.95      0.95      0.95     49488

    accuracy                           0.94     77302
   macro avg       0.93      0.93      0.93     77302
weighted avg       0.94      0.94      0.94     77302
```

Fig 4.2.2.1 Accuracy and Classification Report of DT Model

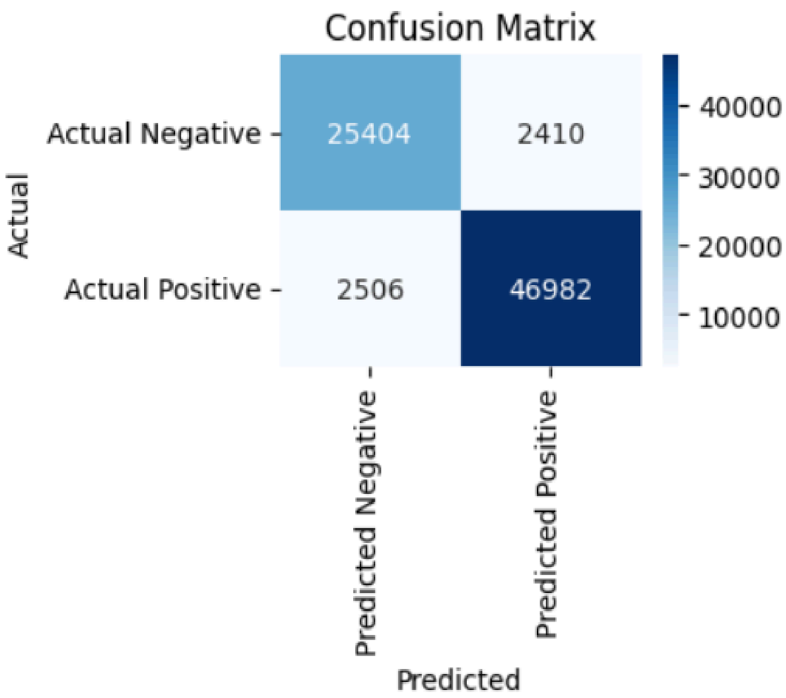


Fig 4.2.2.2 Confusion Matrix of DT Model

The accuracy achieved by the Decision Tree is 94%

4.2.3 Results of Random Forest

```
Accuracy: 0.95
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93     27814
           1       0.96      0.96      0.96     49488

    accuracy                           0.95     77302
   macro avg       0.95      0.95      0.95     77302
weighted avg       0.95      0.95      0.95     77302
```

Fig 4.2.3.1 Accuracy and Classification Report of RF Model

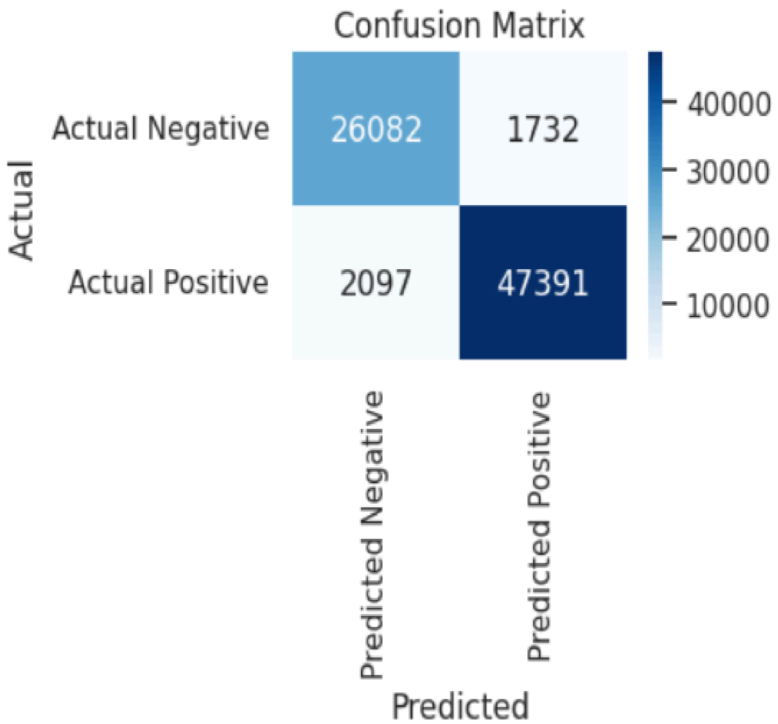


Fig 4.2.2 Confusion Matrix of RF Model

The accuracy achieved by the Random Forest is 95%

4.2.4 Results of Max Voting using LR and DT

```
Ensemble Accuracy: 0.9279449432097487
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.93      0.90     27814
           1       0.96      0.93      0.94     49488

    accuracy                           0.93     77302
   macro avg       0.92      0.93      0.92     77302
weighted avg       0.93      0.93      0.93     77302
```

Fig 4.2.4.1 Accuracy and Classification Report of Max Voting

```
Confusion Matrix:
```

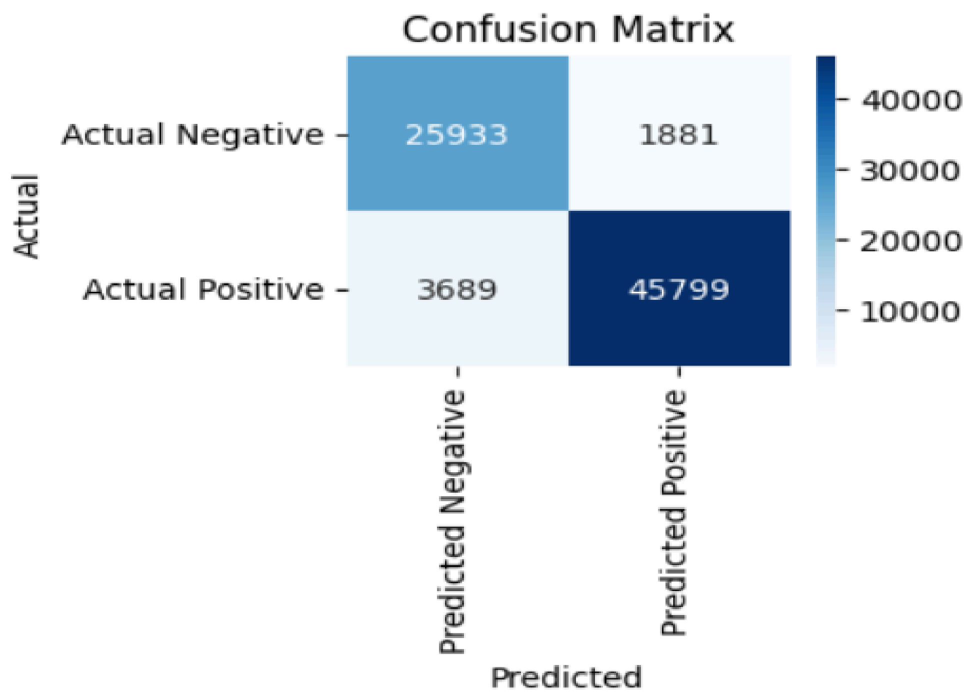


Fig 4.2.4.2 Confusion Matrix of Max Voting

The accuracy achieved by the Ensemble Method Max Voting is 92%

4.2.5 Results of Averaging using LR, DT and KNN

```
Ensemble Accuracy: 0.9307650513570154
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.87      0.90     27814
           1       0.93      0.96      0.95     49488

    accuracy                           0.93     77302
   macro avg       0.93      0.92      0.92     77302
weighted avg       0.93      0.93      0.93     77302
```

Fig 4.2.5.1 Accuracy and Classification Report of Averaging

```
Confusion Matrix:
```

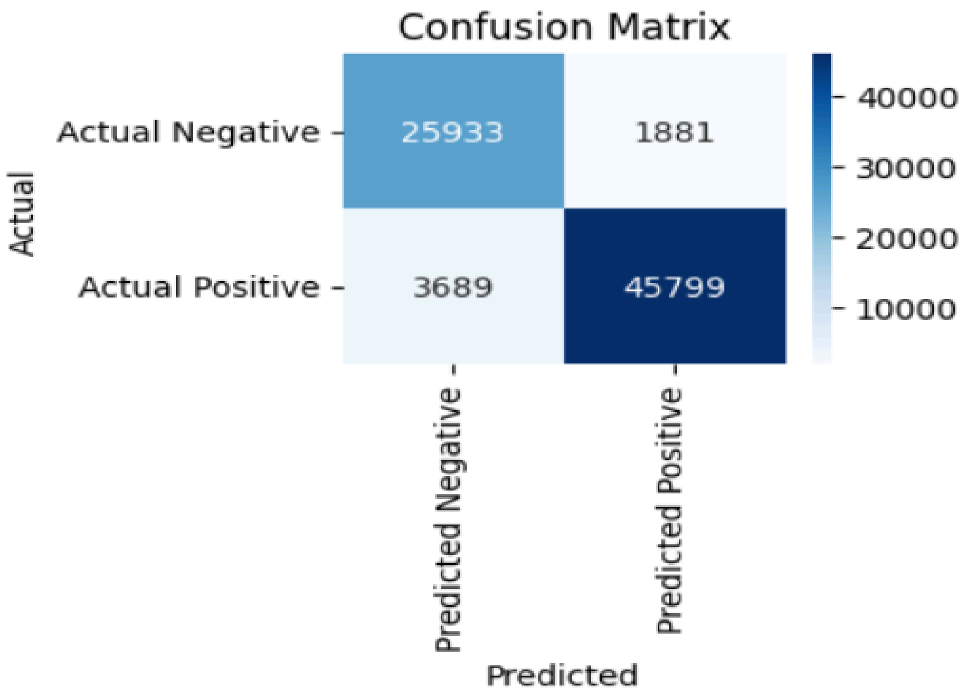


Fig 4.2.5.2 Confusion Matrix of Averaging

The accuracy achieved by the Ensemble Method Averaging is 93%

4.2.6 Results of XgBoost

```
Accuracy: 0.95
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.94      0.93     27814
           1       0.97      0.95      0.96     49488

    accuracy                           0.95     77302
   macro avg       0.94      0.95      0.94     77302
weighted avg       0.95      0.95      0.95     77302
```

Fig 4.2.6.1 Accuracy and Classification Report of XgBoost Model

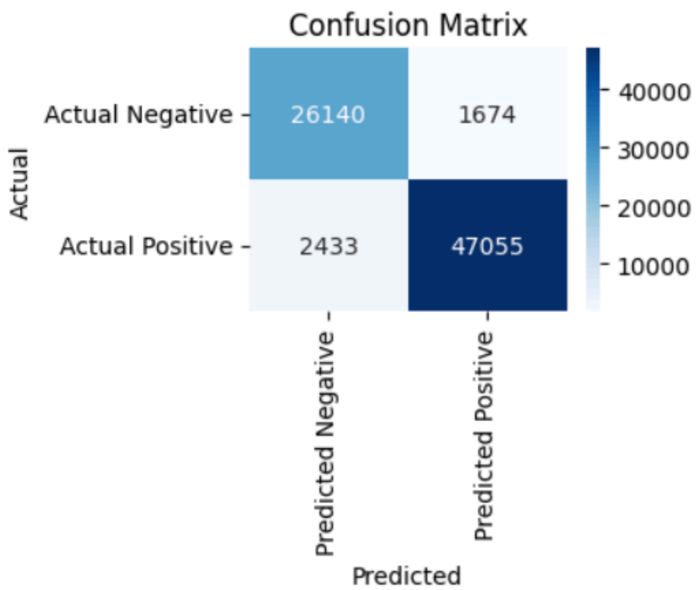


Fig 4.2.6.2 Confusion Matrix of XgBoost Model

The accuracy achieved by the XgBoost is 95%

4.2.7 Results of CNN

```
Epoch 1/10
2577/2577 [==============================] - 32s 12ms/step - loss: 0.1827 - accuracy: 0.9127 - val_loss: 0.1491 - val_accuracy: 0.9276
Epoch 2/10
2577/2577 [==============================] - 30s 12ms/step - loss: 0.1510 - accuracy: 0.9261 - val_loss: 0.1426 - val_accuracy: 0.9284
Epoch 3/10
2577/2577 [==============================] - 29s 11ms/step - loss: 0.1465 - accuracy: 0.9283 - val_loss: 0.1410 - val_accuracy: 0.9292
Epoch 4/10
2577/2577 [==============================] - 33s 13ms/step - loss: 0.1431 - accuracy: 0.9297 - val_loss: 0.1396 - val_accuracy: 0.9299
Epoch 5/10
2577/2577 [==============================] - 29s 11ms/step - loss: 0.1408 - accuracy: 0.9313 - val_loss: 0.1358 - val_accuracy: 0.9327
Epoch 6/10
2577/2577 [==============================] - 29s 11ms/step - loss: 0.1382 - accuracy: 0.9322 - val_loss: 0.1353 - val_accuracy: 0.9325
Epoch 7/10
2577/2577 [==============================] - 29s 11ms/step - loss: 0.1370 - accuracy: 0.9326 - val_loss: 0.1365 - val_accuracy: 0.9322
Epoch 8/10
2577/2577 [==============================] - 28s 11ms/step - loss: 0.1353 - accuracy: 0.9339 - val_loss: 0.1349 - val_accuracy: 0.9326
Epoch 9/10
2577/2577 [==============================] - 28s 11ms/step - loss: 0.1343 - accuracy: 0.9342 - val_loss: 0.1352 - val_accuracy: 0.9334
Epoch 10/10
2577/2577 [==============================] - 28s 11ms/step - loss: 0.1338 - accuracy: 0.9344 - val_loss: 0.1329 - val_accuracy: 0.9345
1611/1611 [==============================] - 5s 3ms/step - loss: 0.1304 - accuracy: 0.9359
```

Fig 4.2.7.1 Epochs values of CNN Model

```
Test Loss: 0.12908633053302765
Test Accuracy: 0.9356553554534912
1611/1611 [==============================] - 4s 3ms/step
              precision    recall  f1-score   support

           0       0.89      0.93      0.91     18675
           1       0.96      0.94      0.95     32860

    accuracy                           0.94     51535
   macro avg       0.93      0.94      0.93     51535
weighted avg       0.94      0.94      0.94     51535
```

Fig 4.2.7.2 Accuracy and Classification Report of CNN Model

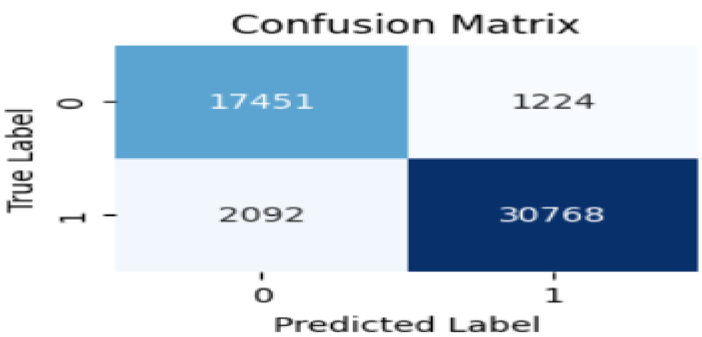


Fig 4.2.7.3 Confusion Matrix of CNN Model

The accuracy achieved by the CNN is 93%

4.2.8 Results of RNN

```
Epoch 1/10
2577/2577 [==============================] - 65s 24ms/step - loss: 0.2186 - accuracy: 0.8971 - val_loss: 0.1739 - val_accuracy: 0.9175
Epoch 2/10
2577/2577 [==============================] - 62s 24ms/step - loss: 0.1767 - accuracy: 0.9138 - val_loss: 0.1638 - val_accuracy: 0.9205
Epoch 3/10
2577/2577 [==============================] - 83s 32ms/step - loss: 0.1689 - accuracy: 0.9168 - val_loss: 0.1592 - val_accuracy: 0.9206
Epoch 4/10
2577/2577 [==============================] - 66s 25ms/step - loss: 0.1648 - accuracy: 0.9184 - val_loss: 0.1582 - val_accuracy: 0.9203
Epoch 5/10
2577/2577 [==============================] - 62s 24ms/step - loss: 0.1607 - accuracy: 0.9204 - val_loss: 0.1522 - val_accuracy: 0.9267
Epoch 6/10
2577/2577 [==============================] - 62s 24ms/step - loss: 0.1590 - accuracy: 0.9216 - val_loss: 0.1501 - val_accuracy: 0.9275
Epoch 7/10
2577/2577 [==============================] - 64s 25ms/step - loss: 0.1568 - accuracy: 0.9221 - val_loss: 0.1442 - val_accuracy: 0.9290
Epoch 8/10
2577/2577 [==============================] - 64s 25ms/step - loss: 0.1547 - accuracy: 0.9233 - val_loss: 0.1438 - val_accuracy: 0.9303
Epoch 9/10
2577/2577 [==============================] - 65s 25ms/step - loss: 0.1539 - accuracy: 0.9244 - val_loss: 0.1453 - val_accuracy: 0.9307
Epoch 10/10
2577/2577 [==============================] - 64s 25ms/step - loss: 0.1526 - accuracy: 0.9255 - val_loss: 0.1444 - val_accuracy: 0.9294
1611/1611 [==============================] - 12s 7ms/step - loss: 0.1422 - accuracy: 0.9310
```

Fig 4.2.8.1 Epoch values of RNN Model

```
Test Loss: 0.14381015300750732
Test Accuracy: 0.9276220202445984
1611/1611 [==============================] - 10s 6ms/step
              precision    recall  f1-score   support

           0       0.86      0.95      0.90     18675
           1       0.97      0.92      0.94     32860

    accuracy                           0.93     51535
   macro avg       0.92      0.93      0.92     51535
weighted avg       0.93      0.93      0.93     51535
```

Fig 4.2.8.2 Accuracy and Classification Report of CNN Model

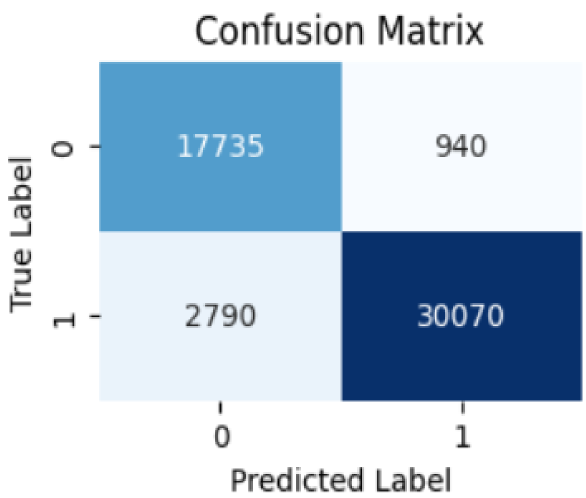


Fig 4.2.8.3 Confusion Matrix of RNN Model

The accuracy achieved by the RNN is 92%

Chapter 5: Results and Evaluation

Evaluations done during testing are informed by a numbers of statistical tools. False positives (Fp) are machines that are mistakenly identified as positives, whereas true positives (Tp) are machines that are correctly identified as being under a botnet attack. False negatives are known as false negatives (Fn), and true negatives (Tn) are ground truth negatives that have been recognized as negatives.

ML/DL MODELS	Different Performance Measure (in %)			
	Precision	Recall	F1-score	Accuracy
Logistics Regression	89	97	93	90
Decision tree	95	95	95	94
Random Forest	96	96	96	95
Xg Boost	97	95	96	95
Max Voting	96	93	94	92
Averaging	93	96	95	93
CNN	96	94	95	93
RNN	97	92	94	92

Table. 5.1 Performance Analysis Table

In the above Table 5.1 , the valued that we have obtained in the multiple machine learning and deep learning models that we have applied on our project.

The results here of Machine Learning and Deep Learning algorithms are shown in percentage. We have achieved the highest accuracy in the Random Forest model.

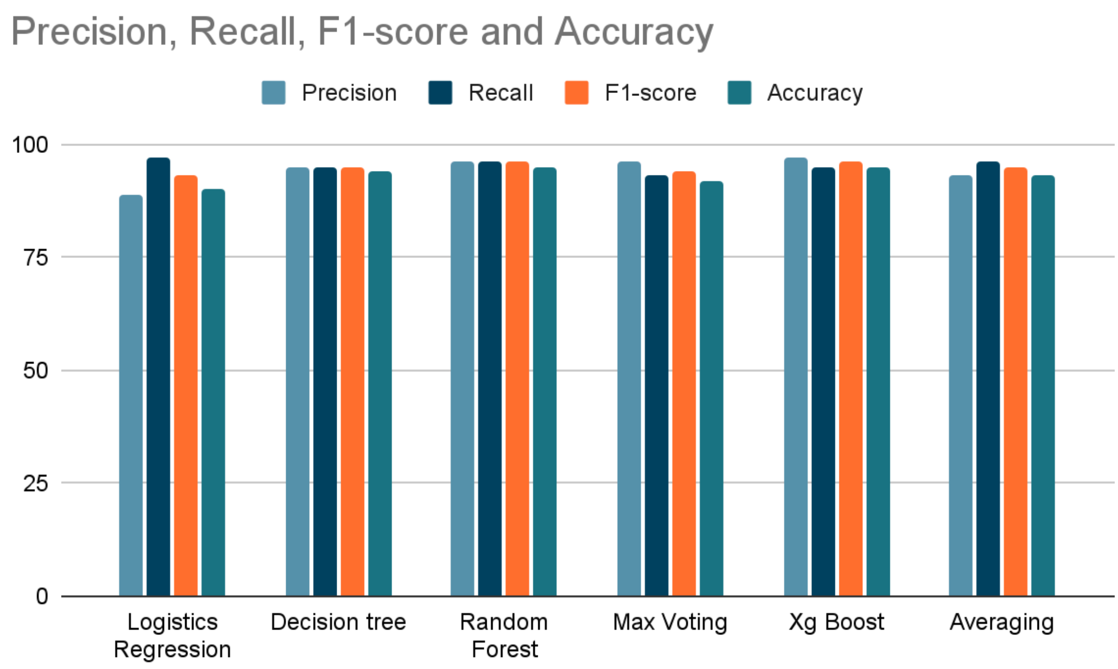


Fig 5.1.1 ML Model Comparisons

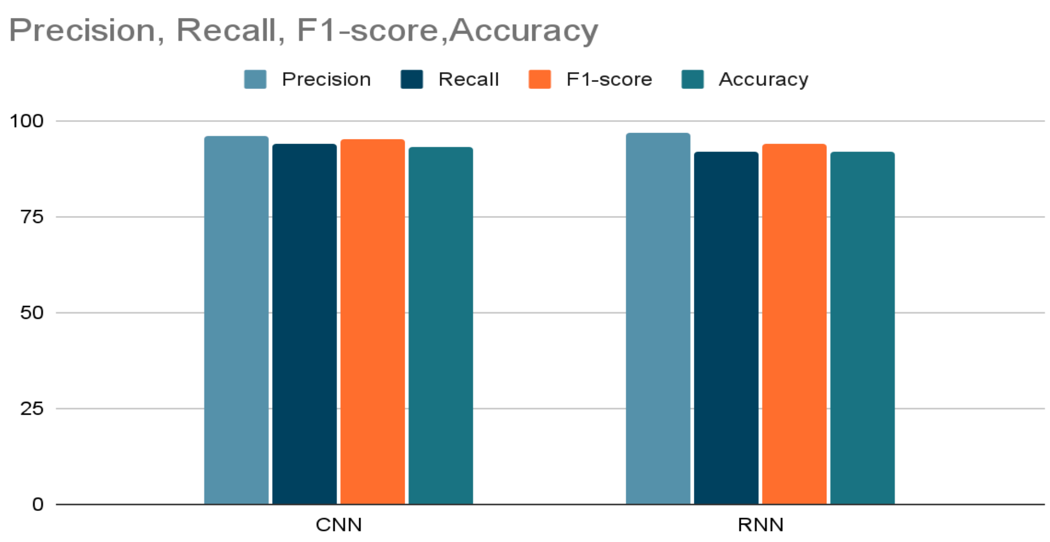


Fig 5.1.2 DL Model Comparisons

In the above Figure 5.1.1 and 5.1.2 , the results that we have obtained in our project have been displayed visually in a bar plot. The bar plot shows the comparison between Performance parameters accuracy, precision, recall, f1-score achieved by different Machine and Deep learning algorithms.

Chapter 6: Conclusions and Future Scope

6.1 Conclusion

Explored real-world scenarios of Botnet Attacks and their global impact. Developed a machine learning and deep learning based models using classifiers for botnet detection in IoT devices

As we have applied six machine learning algorithms that is Logistic Regression, Decision Tree, XgBoost,, Random Forest,Ensembling Methods like Max Voting and Averaging, CNN and RNN, we found that Random Forest has shown the best performance with the accuracy of 95%. It can be due to, Random Forest might be better at capturing the complex relationships and patterns present in your IoT data.

Botnet detection in IoT environments can involve intricate interactions and behaviors, and Random Forest's ability to handle such complexity could give it an advantage over simpler models like logistic regression. Random Forest has shown the best accuracy it can also be due to Random Forest is an ensemble learning method that combines multiple decision trees. This ensemble approach helps reduce overfitting and improves generalization compared to individual decision trees or simpler models like logistic regression.

Our future plan is to investigate additional classifiers and algorithms to enhance model performance. Aim to continuously update and expand dataset for improved applicability and efficacy. Intend to validate model performance on larger datasets like UNSW_NB15 and compare results with universal datasets.

6.2 Future Scope

We see several opportunities to further develop and expand our model as we move forward with our work. Improving our model's performance on the recently created dataset is our primary priority right now. In order to develop a new algorithm that might possibly produce better accuracy in botnet detection, we intend to investigate the use of various classifiers and combine them.

We intend to constantly add new data as it becomes available to our dataset in order to maintain it more robust and up to date. In order to give a more thorough assessment of our model's efficacy, we also hope to validate its performance on the whole UNSW_NB15 dataset. Furthermore, we might think about adding more universal datasets to our dataset in order to boost its variability and improve our model's ability to adapt to real-world situations.

We plan to explore the application of additional classifiers, such as SVM, as well as additional supervised, unsupervised, ensemble machine learning techniques and deep learning algorithms in addition to the current classifiers, which include Decision Tree, Logistics Regression, and Random Forest. We can identify which classifiers perform best by comparing their results, and we may even be able to create a new algorithm that combines the benefits of multiple classifiers to produce even better results.

In order to assess our machine learning model's accuracy outside of carefully controlled laboratory experiments, we also want to test it in real-time settings. This will enable us to better understand our model's performance in real-world scenarios and its ability to handle various threats, both known and unknown.

In conclusion, our next research endeavours will centre around refining our model's performance on the dataset, verifying its efficacy on more extensive datasets, investigating supplementary classifiers, and assessing its performance in real-time scenarios. Through continued improvement and development, these initiatives will help our botnet detection model become more accurate and useful.

REFERENCES

[1] S. Afrifa, V. Varadarajan, P. Appiahene, T. Zhang, and E. A. Domfeh, "Ensemble Machine Learning Techniques for Accurate and Efficient Detection of Botnet Attacks in Connected Computers," Eng, vol. 4, no. 1, pp. 650-664, Feb. 16, 2023, doi: 10.3390/eng4010039. https://www.mdpi.com/2673-4117/4/1/39

[2]K. Alissa, T. Alyas, K. Zafar, Q. Abbas, N. Tabassum, and S. Sakib, "Botnet Attack Detection in IoT Using Machine Learning," Open Access, vol. 2022, Article ID 4515642, Oct.4, 2022, doi:10.1155/2022/4515642. https://www.hindawi.com/journals/cin/2022/4515642/

[3]N. Venu, A. Kumar, and A. Rao, "BOTNET Attacks Detection in the Internet of Things Using Machine Learning," NeuroQuantology, vol. 20, no. 4, pp. 743-754, April 2022, doi: 10.14704/NQ.2022.20.4.NQ22298. [Online]. https://www.researchgate.net/publication/363846022

[4] M. Mudassir et al., "Detection of Botnet Attacks in Industrial IoT Systems with Multilayer Deep Learning," Open Access, vol. 2022, Article ID 2845446, 2022, doi:10.1155/2022/2845446. https://www.hindawi.com/journals/wcmc/2022/2845446/

[5]S. Pokhrel, R. Abbas, and B. Aryal, "IoT Security: Botnet Detection in IoT using Machine Learning," arXiv:2104.02231 [cs.LG], Apr. 6, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2104.02231.

[6] F. Hussain et al., "A Two-Fold Machine Learning Approach for IoT Botnet Attack Prevention and Detection, " IEEE Access, vol. 9, pp. 163412-163430, Nov. 25, 2021. [Online]. doi:10.1109/ACCESS.2021.3131014. Available: https:// ieeexplore.ieee.org /document/9627657.

[7]H. Alkahtani and T. H. H. Aldhyani, "Botnet Attack Detection by Using CNN-LSTM Model for Internet of Things Applications," Open Access, vol. 2021, Article ID 3806459, Sept. 10, 2021, doi: 10.1155/2021/3806459. https://www.hindawi.com/journals/scn/2021/3806459/

[8] Y. N. Soe et al., "A Sequential Scheme for Detecting Cyber Attacks in IoT Environment," in Proceedings of the 4th IEEE Cyber Science and Technology Congress,Japan, 5–8 Aug.2019. https://www.mdpi.com/1424-8220/20/16/4372

[9] T. A. Tuan, H. V. Long, L. H. Son, R. Kumar, I. Priyadarshini, and N. T. K. Son, "Performance evaluation of Botnet DDoS attack detection using machine learning," Evolutionary Intelligence, vol. 13, pp. 283–294, Nov. 20, 2019. https://link.springer.com/article/10.1007/s12065-019-00310-w#Abs1

[10]M. Alshamkhany, W. Alshamkhany, and M. Mansour, "Botnet Attack Detection Using Machine Learning," November 2020. doi: 10.1109/IIT50501.2020.9299061. https://www.researchgate.net/publication/347445002

[11] Dr.Nookala Venu, Dr.A.ArunKumar, Karthik Kumar Vaigandla, "Investigation on Internet of Things(IoT): Technologies, Challenges and Applications in Healthcare," International Journal of Research, Volume XI, Issue II, February/2022, pp.143-153

[12] Y. Meidan, M. Bohadana, A. Shabtai et al., "Detection of Unauthorized IoT Devices Using Machine Learning Techniques," 2017. [Online]. Available: https://arxiv.org/abs/1709.04647.

[13] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer Internet of things devices," in Proceedings of the 2018 IEEE Security and Privacy Workshops, 2018. View at: DOI: 10.1109/SPW.2018.00013

[14] E. Hodo, X. Bellekens, A. Hamilton, et al., "Threat analysis of IoT networks using artificial neural network intrusion detection system," in Proceedings of the 2016 International Symposium on Networks, 2016. DOI: 10.1109/ISNCC.2016.7746067

[15] W. Jo, S. Kim, C. Lee, and T. Shon, "Packet Preprocessing in CNN-based network intrusion detection system," Electronics, vol. 9, no. 7, p. 1151, 2020. View at: https://doi.org/10.3390/electronics9071151

[16] R. Yao, N. Wang, Z. Liu, P. Chen, and X. Sheng, "Intrusion detection system in the advanced Metering infrastructure: a cross-layer feature-Fusion CNN-LSTM-Based approach," Sensors, vol. 21, no. 2, 2021. https://doi.org/10.3390/s21020626

[17] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto, and K. Sakurai, "Machine learning-based IoT-botnet attack detection with Sequential architecture," Sensors, vol. 20, p. 4372, 2020. https://doi.org/10.3390/s20164372

[18] B. Selvakumar and K. Muneeswaran, "Firefly algorithm based feature selection for network intrusion detection," Computers & Security, vol. 81, 2019. https://doi.org/10.1016/j.cose.2018.11.005

[19] C.Kolias, G. Kambourakis, A.Stavrou, and J. Voas, " DDoSinthe IoT: Mirai and Other Botnets, " Computer, vol.50, 2017. DOI:10.1109/MC.2017.201

[20] Joshi, C., Bharti, V., & Ranjan, R. K. (2021). Botnet Detection Using Machine Learning Algorithms. In International Conference on Paradigms of Computing, Communication and Data Sciences (PCCDS-2020) (pp. 1–2). DOI:10.1007/978-981-15-7533-4_56.

[21] Haider, W.; Creech, G.; Xie, Y.; Hu, J. Windows based data sets for evaluation of robustness of Host based Intrusion Detection Systems (IDS) to zero-day and stealth attacks. Futur. Internet 2016, 8, 29. https://doi.org/10.3390/fi8030029

[22] Singh, K.J.; Thongam, K.; De, T. Entropy-based application layer DDoS attack detection using artificial neural networks. Entropy 2016, 18, 350. https://doi.org/10.3390/e18100350

[23] Shafiq, M.; Tian, Z.; Sun, Y.; Du, X.; Guizani, M. Selection of effective machine learning algorithm and Bot-IoT attacks traffic identification for Internet of things in the smart city. Futur. Gener. Comput. Syst. 2020 https://doi.org/10.1016/j.future.2020.02.017

[24]V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal and B. Sikdar, "A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures," in *IEEE Access*, vol. 7, pp. 82721-82743, 2019, doi: 10.1109/ACCESS.2019.2924045

[25] Mobile network intrusion detection for IoT system based on transfer learning algorithm. Cluster Comput 22 (Suppl 4), 9889–9904 (2019). https://doi.org/10.1007/s10586-018-1847-2

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: ____________________________ **Department:** ________________ **Enrolment No** _________

Contact No. ____________________________ **E-mail.** ____________________________________

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _______________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at.................... (%). Therefore, we
are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index (%)**	**Generated Plagiarism Report Details** (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com