University of Essex
Department of Mathematical Sciences

MA981: DISSERTATION

Predictive Modeling for Network Attack Detection Using Ensemble Methods

chandana somanahalli ashok
2310583

Supervisor:Wenxing Guo

September 18, 2024
Colchester

Contents

List of Figures

List of Tables

Abstract

According to the article of McAfee dec21,2020 In 2020, cyber-attacks caused damages amounting to over $1 trillion globally. Since the 2010s there has been rapid growth of Internet of Things (IoT) devices significantly changed and been adopted by various sectors like healthcare, manufacturing, and smart homes. Many more sectors have been positively influenced, and due to this rapid growth in different fields the challenge, and problems have been an increased risk of sophisticated cyber-attacks, this study focuses on the development and evaluation of predictive models for network attack detection using ensemble methods, here mainly targeting on IoT environments. By leveraging the RT-IoT2022 dataset, which encompasses a diverse range of IoT devices and network attack methodologies, this research aims to enhance the capabilities of Intrusion Detection Systems (IDS). The primary object of this study is to identify the most effective and best-performing methods for detecting network attacks and to understand the importance of the underlying feature contributing to detection accuracy. There are a variety of machine learning methods that can be used like Random Forests, Gradient Boosting Machines (GBM), XGBoost(An optimised version of gradient boosting), k-nearest Neighbors (k-NN), k mean etc. This study focuses on Random Forests, XGBoost, and k-nearest Neighbors (k-NN Additionally, the second objective is to determine which type of attack is most common within the dataset. using these advanced methods, we can identify which model provides the best predictive performance, while the k-means clustering method is employed to explore the data and reveal how it naturally clusters. This study begins with handling the data like handling the missing value, encode categorical features, and scaling the data. Principal Component Analysis (PCA) is applied to reduce dimensionality, ensuring computational efficiency while retaining critical information. XGBoost, k-NN, and Random Forest models are then trained and evaluated using key metrics such as accuracy, precision, recall, and F1-score to classify various types of network traffic, including cyber-attacks.

Keywords: IoT Security, Network Attack Detection, Random Forest, XGBoost, k-Nearest Neighbors (k-NN), k-Means Clustering, Principal Component Analysis (PCA)

CHAPTER 1

Introduction

IoT, “Internet of Things”. IoT is A network of physical things equipped with software, sensors, and various technologies to connect and exchange data with other devices and systems via the Internet. [1] IoT has played a significant role in the increase in the number of connected devices, transforming industries such as healthcare, manufacturing, and smart homes. However, this surge in connectivity has also introduced a vast array of security vulnerabilities, making IoT networks prime targets for cyber-attacks. The complexity, advancement in IoT technology and dynamic nature of network traffic, present unique challenges for traditional security measures. As a result, there is a growing need for advanced Intrusion Detection Systems (IDS) that effectively detect and mitigate these evolving threats.

1.1 APPLICATION OF IoT

One of the most significant traits of humans is our ability to collaborate and learn from one another. [2] Imagine if machines could do the same,interacting and sharing information and data. This would create a truly interconnected world, the essence of the Internet of Things (IoT). IoT represents a network of diverse devices that can collect and exchange data, deriving meaningful insights. The applications of IoT are vast and varied, extending into almost unimaginable areas. [2]

The top application of the IoT has made our lives more convenient through its various applications. Imagine a home where the lights turn on automatically, the TV

knows your favourite shows, and the fridge alerts you when you're running low on ice cream—how cool is that? In large factories, IoT enables machines to operate together seamlessly, like a team of robots. They can even perform self-maintenance when issues arise, showcasing their smart capabilities. [2] These impressive capabilities are made possible through the top applications of IoT across different industries. [2] like Smart Agriculture, Smart Vehicles, Smart Home, Smart Pollution Control, Smart Healthcare, Smart Cities, Smart Retail [2]

Figure

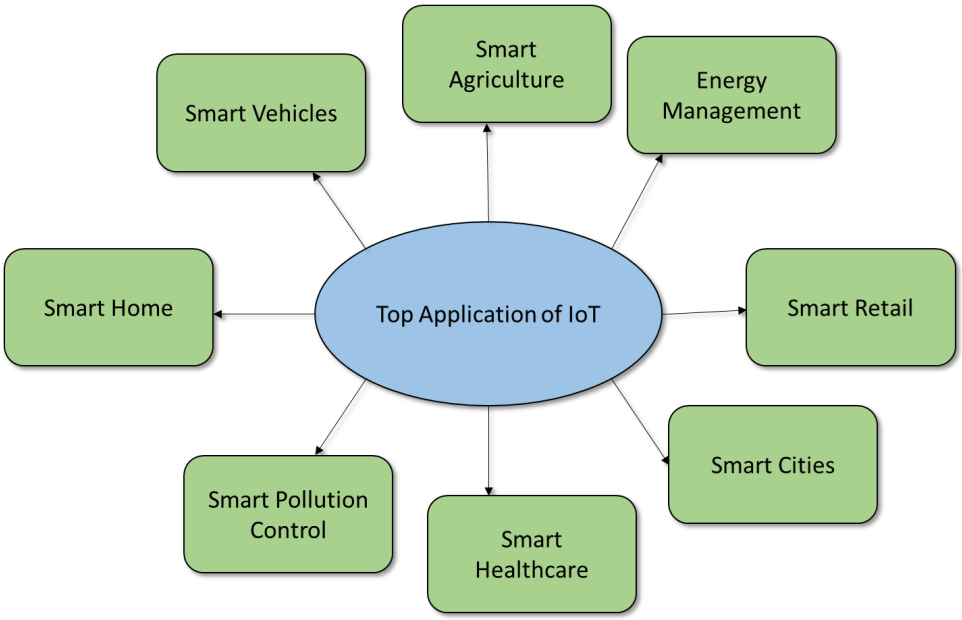

Figure 1.1: This is a sample figure.

1.2 IoT ARCHITECTURES

Multi-layered architectures have been proposed, such as three—layered, four—layered, and five—layered. Also, many recent proposed like Service Oriented Architecture (SOA), Work done comparing the Cloud Architecture, Fog Architecture, Edge Architecture and Mist Architecture specific to IoT. [3] If we consider the four—layered multi—layered architectures, the Perception layer, Network layer, Processing layer, and Application layer are the four layers. [1] a detailed explanation of each layer is given below,

1. Perception Layer: This layer includes sensors, actuators, RFIDs, QR codes, and

ZigBee technology to collect data on various parameters such as temperature, humidity, pH levels, motion, and air pressure. It gathers information from diverse sources. [1]

2. Network Layer: This layer encompasses physical components and software necessary for transferring data between different layers. [1]

3. Processing Layer: A crucial component of IoT, this layer is responsible for storing, analysing, and processing large volumes of data. It handles tasks such as data mining, data analytics, and generating recommendations. [1]

4. Application Layer: This layer interacts directly with the end user of the IoT device, providing services and functionalities based on the specific type of device and user needs. [1]

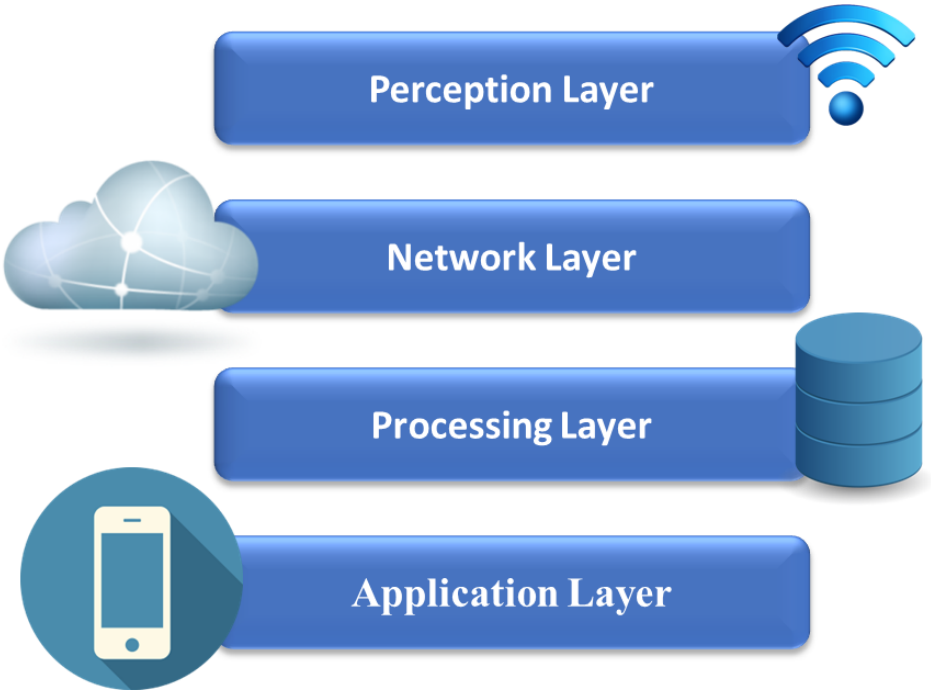

Figure 1.2: Four layered Architecture [1]

1.3 SECURITY CHALLENGES IN IoT

The security and privacy of IoT systems are critical challenges that must be resolved to ensure their continued growth and widespread use. IoT devices generate and store

vary large amount of sensitive personal and commercial data, making them susceptible to data breaches, hacking, and unauthorized access [5]. Among the most prevalent security threats to IoT systems are:

- Data Breaches: IoT devices frequently collect and store personal information, making them prime targets for cybercriminals seeking to steal or misuse this data [5].
- Hacking: IoT devices are particularly vulnerable to hacking, especially when inadequately secured or running outdated software. Compromised devices can be leveraged to attack other systems, steal sensitive information, or control connected devices. [5]
- Unauthorized Access: Weak passwords or system vulnerabilities can allow attackers to gain unauthorized access to IoT devices, exposing them to exploitation [5].
- Malware: IoT devices are also at risk of malware infections, which can spread across the network, causing extensive damage to the entire system [5].

An IoT network is composed of numerous devices connected to the internet, making them susceptible to security breaches. [4] These vulnerable devices can be exploited by cybercriminals to launch various types of attacks, such as Denial-of-Service (DoS), ARP Poisoning, NMAP Scans, Distributed Denial-of-Service (DDoS), and Metasploit Brute Force SSH (Secure Shell). Securing large-scale IoT systems presents significant challenges, as many devices possess limited processing power, memory, and storage, making it difficult to implement conventional security measures [4, 5]. Developing effective and efficient security strategies is essential to protect IoT systems and the sensitive data they store and process. To tackle these security and privacy concerns, organisations should adopt a multi-layered security approach that includes encryption, secure authentication, and regular software updates [5]. Furthermore, emerging technologies such as blockchain and machine learning hold great potential for improving the security and privacy of IoT systems. Here in this study the data set used

1.4 MACHINE LEARNING IN IOT

Machine learning models have become a cornerstone for securing IoT systems due to their ability to process large volumes of data, detect anomalies in real-time, and predict potential threats. By analyzing network traffic data, these models can identify suspicious behaviors and adapt to new attack patterns continuously evolving in IoT environments. One of the key models used in this study is k-means clustering, which is widely utilized for anomaly detection and exploratory data analysis. K-means allows us to segment the network traffic into different clusters, providing insights into the natural grouping of data and identifying any anomalous behaviour within those groups [15].

Another method used in this study is the k-nearest neighbors (k-NN) algorithm, which is a simple yet effective technique for classifying network traffic. The model classifies network activity based on its proximity to known data points and can be particularly useful in detecting similar attack patterns, given the vast amount of IoT traffic data available for comparison. By relying on the majority class of its nearest neighbors, k-NN has proven effective in identifying malicious activities in IoT networks [16].

XGBoost, a powerful gradient-boosting algorithm, was also employed in this study. XGBoost works by building trees sequentially, with each tree correcting the mistakes made by the previous one. It is known for its speed and efficiency, as well as its ability to handle high-dimensional data, making it an ideal choice for detecting complex attack patterns in IoT networks. XGBoost has been extensively used in cybersecurity applications for its superior performance in classification tasks and its ability to reduce bias by boosting weak learners [11].

Finally, this study employed Random Forest as part of the ensemble learning approach. This method builds multiple decision trees, each trained on different subsets of the data, and aggregates their outputs to provide a final prediction. Random Forest is especially effective in high-dimensional spaces like network traffic analysis and is robust to overfitting. It has shown great accuracy in identifying network attacks and distinguishing between different types of traffic, including normal behavior and various attack types [9].

These models provide a robust framework for detecting and classifying network

traffic in IoT systems. By integrating k-means clustering, k-NN, XGBoost, and Random Forest, this study aims to improve the accuracy and reliability of IoT security measures, ensuring that any potential threats are quickly identified and mitigated [1]. The use of ensemble methods in machine learning has proven to be particularly powerful, as it allows the system to leverage the strengths of multiple models, improving overall detection accuracy and resilience against various forms of cyberattacks [17].

1.5 Critical Issue and Real-time Example

the making IoT IoT networks prime targets for cyberattacks. The decentralized and heterogeneous nature of IoT networks, coupled with limited computational resources on many devices, makes them particularly vulnerable to a wide range of network attacks.when it comes to why critical issue.

- Inherent Vulnerabilities in IoT Devices: In many IoT devices have minimal built-in security due to their constrained resources, including limited processing power, memory, and storage.because of this the devices often lack proper encryption, authentication, and firmware updates, making them easy targets for attackers. [20]

  For example some IoT devices still use default or weak passwords, which can be exploited by attackers to gain unauthorized access and control. [21]

- Potential for Large-Scale Attacks:

  IoT devices can be used as entry points for larger-scale attacks.when it comes to example, Mirai botnet attack in (2016) where where thousands of poorly secured IoT devices were infected and used to launch a massive Distributed Denial-of-Service (DDoS) attack, disrupting major websites like Twitter, Netflix, and Reddit. This incident highlighted the potential for IoT devices to be weaponized, causing widespread internet outages and significant economic damage. [22]

- Impact on Critical Infrastructure:

  IoT devices has been frequently incorporated into critical infrastructure like smart grids, healthcare systems, and industrial control systems. A successful attack on these systems could lead to disastrous outcomes, such as power outages, compromised patient data, and disruptions in industrial processes. [22] when

it comes to real time example The Guardian or BBC covered the 2021 the water water treatment facility in Florida was targeted in a cyberattack where the attacker attempted to manipulate the chemical levels in the water supply. The attack was mitigated in time, but it demonstrated the potential risks associated with vulnerable IoT systems in critical infrastructure. [23]

- Data Privacy Concerns:

  where IoT devices collect vast amounts of sensitive data, including personal, medical, and financial information. where this data can lead to severe privacy violations, identity theft, and financial losses. the real time example is In 2017, a casino's network was compromised through an IoT-enabled thermometer in a fish tank. where the attackers used the thermometer to gain access to the casino's network and ex filtrate high-roller data, showing how even seemingly harmless IoT devices can be exploited to access sensitive information.

  the most impotent of this study is gives the vulnerabilities and the potential impact of IoT network attacks, developing effective detection mechanisms is crucial.the security measures such as firewalls and anti virus software are often insufficient in the dynamic and diverse landscape of IoT networks. so here is a pressing need for advanced Intrusion Detection Systems (IDS) that can monitor network traffic in real-time and accurately identify various types of cyber attacks.

In the next chapter 2.literature review, it provides an in-depth analysis of existing research related to network attack detection, machine learning, and IoT security. The literature review establishes the foundation for the proposed methodology and highlights the state-of-the-art methods used for network security and intrusion detection in IoT environments.

Chapter 3 gives the details of the methodologies employed to achieve this study's objectives. It describes the dataset used, including the preprocessing techniques applied to clean and prepare the data for analysis. The machine learning models used for predictive modeling, such as Random Forest, XGBoost, k-NN, and k-means clustering, are discussed. The chapter also explains the hyperparameter tuning and model evaluation techniques, such as cross-validation, and outlines the experimental setup for the study.

Chapter 4 'Results and Analysis' results of this study conducted on the dataset are

presented and analyzed. The performance of the machine learning models is evaluated using metrics such as accuracy, precision, recall, and F1-score. Furthermore, Visualizations, including confusion matrices and cluster visualizations, are also provided to support the analysis.

Chapter 5 The final chapter summarizes the key findings of the research and highlights the contributions made to the field of IoT security. It reflects on the effectiveness of the applied machine learning techniques and discusses potential improvements. The chapter also addresses the limitations of the study and proposes future research directions that could build on the findings of this study.

CHAPTER 2

Literature Review

The paper Systematic Literature Review of Machine Learning for IoT Security: provides an extensive review of machine learning (ML) techniques applied in the context of Internet of Things (IoT) security. It highlights the rapid growth of IoT devices and the increasing vulnerability these devices face due to their complex architecture and network exposure. The study emphasizes the need for robust security solutions and explores the role of machine learning in addressing various IoT security challenges, including the detection of cyberattacks, malware, and ransomware. [1]

Key contributions of the paper include a detailed classification of primary studies into five categories: attack detection, intrusion detection, DDoS attacks, malware, and ransomware detection. The authors systematically review ML techniques like Random Forest, Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), Artificial Neural Networks (ANN), and others, focusing on their application in IoT security. [1]

The paper concludes that while machine learning offers significant potential in enhancing IoT security, challenges remain in areas such as scalability, processing large datasets, and adapting to evolving threats. It also identifies the need for future research in combining multiple ML techniques (e.g., ensemble learning) and addressing computational limitations in IoT environments [1]

In this paper, Khan et al. propose a weighted ensemble model combining Random Forest (RF) and Deep Neural Networks (DNN) to detect phishing websites. he paper proposes a novel approach to detect phishing websites by utilizing a weighted ensemble

model that combines the Random Forest (RF) and Deep Neural Network (DNN) algorithms. Phishing websites, which are used to steal sensitive information by mimicking legitimate websites, present a major cybersecurity threat. The authors aim to improve the accuracy of phishing detection through the ensemble model, which leverages the strengths of both machine learning (RF) and deep learning (DNN) techniques. [9]

The study demonstrates that RF can efficiently handle high-dimensional data and is robust against noisy and missing data, while DNN excels in detecting complex patterns within the dataset. The ensemble model achieves a superior accuracy rate of 99%, surpassing the individual accuracies of RF (98%) and DNN (73%). The dataset used for the study contains 651,191 URLs, categorized into benign, defacement, phishing, and malware URLs. [9]

The paper highlights the importance of ensemble learning in phishing website detection, noting that combining multiple models leads to more robust predictions and better generalization performance. The study's results indicate that the ensemble model significantly improves phishing detection, making it a valuable tool in the fight against cybercrime. [9]

The paper titled "Deep Ensemble Learning With Pruning for DDoS Attack Detection in IoT Networks" introduces the DEEPShield system, a novel approach to detecting both high- and low-volume DDoS attacks in resource-constrained IoT environments. The study focuses on utilizing deep learning techniques, particularly a combination of Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. These models analyze network traffic, helping detect DDoS attacks with high accuracy. A key aspect of this work is the integration of unit pruning techniques, which optimize the ensemble model to improve efficiency and reduce computational resource demands, making it suitable for edge devices. [11]

The authors also developed a custom dataset, HL-IoT, specifically designed to evaluate DDoS attacks in IoT systems. The DEEPShield system achieved superior performance with over 90% accuracy across various datasets, including HL-IoT, ToN-IoT, and CICIDS-17. This approach allows for a balance between detection accuracy and computational efficiency, making it a promising solution for IoT network security. [11]

The paper highlights the growing complexity of IoT systems and the evolving threat of DDoS attacks, particularly those that are multi-vector in nature. By combining CNN

and LSTM models in an ensemble approach, the authors address both high-volume and low-volume attacks, offering a comprehensive solution that outperforms traditional machine learning models in terms of accuracy and efficiency. [11]

The paper "IoT Device Fingerprinting: Machine Learning based Encrypted Traffic Analysis" provides an insightful exploration of how machine learning techniques can be applied to identify IoT devices even when network traffic is encrypted. This research tackles the problem of IoT device fingerprinting, where attackers can passively observe encrypted traffic patterns to infer the type of IoT device transmitting the data. By employing a machine learning-based approach, the authors demonstrate that it is possible to achieve significant accuracy improvements (18.5% better than baseline) and efficiency in detecting device types using only the metadata of the traffic. [12]

The paper introduces an extended sliding window technique to improve traditional traffic segmentation and applies various classification algorithms such as Random Forest (RF), k-nearest neighbors (k-NN), and AdaBoost. The results show that, despite encryption, distinguishing between different IoT device types based on traffic features is possible and practical. This research underscores the potential security vulnerabilities of IoT devices and highlights the need for better device management and security protocols to prevent attackers from gaining sensitive information through traffic analysis. [12]

The paper titled "SDN-based Machine Learning Powered Alarm Manager for Mitigating the Traffic Spikes at the IoT Gateways" focuses on addressing the security challenges in IoT networks, particularly in mitigating traffic spikes that can overwhelm IoT gateways and application servers. By proposing a software-defined networking (SDN)-based alarm manager powered by machine learning, the study demonstrates how to detect and respond to abnormal traffic patterns, such as those caused by Distributed Denial of Service (DDoS) attacks. [4]

The alarm manager utilizes machine learning algorithms, specifically Random Forest and XGBoost, to classify traffic and distinguish between legitimate and malicious activities. The ensemble learning approach improves detection accuracy and reduces both bias and variance, leading to more effective real-time mitigation strategies. The SDN-based architecture enables centralized management, making it easier to block malicious flows directly at the gateway level, preventing attacks from propagating

through the network. [4]

The paper highlights the increasing adoption of IoT devices and the need for secure architectures to prevent cyberattacks. It also emphasizes the importance of machine learning in detecting and mitigating threats in real time, particularly in environments with high traffic volumes, such as IoT gateways. Through experimental results, the study demonstrates that the proposed solution achieves a precision rate of 98%, making it a promising approach for enhancing IoT security. [4]

Blockchain is presented as a decentralized and tamper-proof technology that can significantly enhance data integrity and security by eliminating central points of failure and providing transparent, immutable records. Additionally, machine learning algorithms can play a crucial role in threat detection by analyzing patterns in large datasets generated by IoT devices, enabling real-time detection and response to security threats. [5] The authors conducted a systematic literature review (SLR) that examined numerous peer-reviewed studies between 2018 and 2022, highlighting the convergence of blockchain, machine learning, and IoT technologies. The findings indicate that while blockchain can ensure data authenticity and prevent unauthorized tampering, machine learning can detect anomalies and predict threats in real-time, creating a layered security framework that is both proactive and robust. [5] This paper contributes to the literature by emphasizing the need for a multi-layered security approach in IoT, integrating both blockchain for decentralization and machine learning for real-time threat detection. It underscores that although these technologies offer great promise, challenges like computational costs and the need for large datasets remain key hurdles to widespread adoption. Future research should focus on optimizing these technologies for resource-constrained IoT environments to ensure scalability and efficiency. [5]

The paper titled "Security Frameworks for Internet of Things Systems - A Comprehensive Survey" provides a thorough review of the existing security frameworks developed to safeguard IoT systems. It highlights the critical role of security architectures in ensuring the robustness of IoT applications, given the growing IoT threat landscape. The paper discusses various IoT architectures, including three-layered, four-layered, and newer paradigms like fog and edge computing architectures, as well as blockchain-based security frameworks. [3]

The authors survey a range of security frameworks that address threats target-

ing IoT systems, which range from physical devices to protocols and middleware layers. They classify these frameworks into three categories: application-specific, functionality/protocol-specific, and generic frameworks. Each category is assessed based on its implementation, test-bed setup, and effectiveness in addressing IoT security challenges. The paper emphasizes the need for adaptable and application-agnostic security frameworks capable of handling threats at every layer of the IoT stack. It concludes by proposing enhancements, such as integrating machine learning and blockchain technologies, to improve the detection and mitigation of security threats in IoT environments. [3]

The paper "Phishing Website Detection using Machine Learning Techniques" (2023): addresses the detection of phishing websites using various machine learning techniques. Phishing attacks are a significant concern in cybersecurity, particularly in tricking users into divulging personal information by masquerading as legitimate websites. The authors explore different methods such as decision trees, random forests, and k-nearest neighbors (k-NN) for detecting phishing URLs. They highlight the advantages of machine learning over traditional blacklist methods, emphasizing that machine learning models can adapt to new phishing tactics more effectively. [10] The research demonstrates that models like Random Forest achieved high accuracy, around 97.14%, in detecting phishing websites, primarily due to their robustness and ability to reduce overfitting. The paper also explores the use of feature selection methods such as principal component analysis (PCA) to enhance model performance by reducing irrelevant data. [10]

The paper "IoT Network Traffic Classification Using Machine Learning Algorithms: An Experimental Analysis" focuses on the challenges of classifying IoT device traffic within smart environments using machine learning techniques. The authors highlight the increasing use of IoT devices across various sectors and the need for efficient methods to classify network traffic generated by these devices. The study investigates a range of machine learning models for IoT traffic classification, including Random Forest (RF), k-Nearest Neighbors (k-NN), and Support Vector Machine (SVM). The authors use a dataset containing 20 days of network traces from 20 different IoT devices. [15]

Key contributions of this paper include a structured approach to feature extraction from network traffic, the application of various machine learning models to classify IoT

traffic, and a comparative evaluation of these models based on performance metrics like accuracy, speed, and training time. Additionally, the paper provides insights into the limitations of existing classification techniques and offers suggestions for selecting appropriate machine learning models for different IoT traffic scenarios. [15] This research is relevant to your dissertation as it explores the use of machine learning techniques for IoT traffic classification, similar to your study on predictive modeling for network attack detection. It highlights the importance of feature selection, model tuning, and performance evaluation, all of which align with your work using models like Random Forest, XGBoost, and k-NN. This paper could provide a valuable reference for understanding the challenges and opportunities in IoT traffic classification. [15]

The paper "Research of Network Traffic Classification Based on Ensemble Learning" explores the application of ensemble learning methods for improving the accuracy and generalization of network traffic classification. The authors propose a hybrid model combining three distinct classifiers: Logistic Regression (LR), Support Vector Machine (SVM), and k-Nearest Neighbors (k-NN). Each classifier is applied individually to classify network traffic, and the final classification is determined using the majority voting ensemble method. [16]

Ensemble learning has been identified as a powerful tool for improving model performance, particularly in complex environments such as network traffic analysis. By combining multiple classifiers, ensemble learning aims to leverage the strengths of each model while compensating for their weaknesses. This paper emphasizes that ensemble methods are more robust and adaptable than single-classifier models, achieving higher accuracy and stronger generalization capabilities. [16]

The study highlights the challenges associated with modern network environments, where network attacks such as brute force hacking, malware, and distributed denial-of-service (DDoS) attacks are increasingly common. The goal of this research is to classify network traffic efficiently, a key step in identifying and mitigating these malicious activities. [16]

The paper discusses the ensemble method's improved accuracy over individual classifiers, showing a 15.64% and 1.23% improvement over Logistic Regression and k-NN, respectively, in terms of training accuracy. Additionally, the study demonstrates that ensemble models reduce the overfitting tendency seen in some single classifiers

like k-NN, as indicated by their more stable learning curves. The use of evaluation metrics such as ROC_ AUC (Receiver Operating Characteristic - Area Under the Curve) further validates the ensemble model's stronger generalization ability. [16]

The paper "Analysis of Machine Learning (ML) and Internet of Things (IoT) presents a comprehensive scientometric analysis of Machine Learning (ML) and Internet of Things (IoT) research, focusing on publication trends and the research landscape from 2015 to 2021. The authors used the Scopus database to identify and analyse research papers, performing bibliometric analysis to assess co-authorship, keyword occurrences, and citation trends using VOS viewer software. The results show a sharp increase in IoT-ML-related publications, with the number rising from just two papers in 2015 to 105 papers in 2021. The analysis reveals that India is the top country in terms of research output, with the Vellore Institute of Technology being the most prolific institution. [18]

The paper also identifies the key researchers and most impactful publications, with Mahdavinejad et al.'s 2018 survey on machine learning for IoT data analysis being the most cited work. The research landscape shows that IoT-ML research is being applied in various fields, including healthcare, computer security, and data analytics. The paper highlights the growing importance of IoT-ML in addressing security issues, particularly in the context of Big Data, and the potential for this research field to expand further in the coming years, especially in network security and big data analytics. [18]

This review of publication trends and thematic analysis contributes to the understanding of how ML and IoT have been evolving and their implications for future research, making it relevant to ongoing discussions in network security and predictive modelling. [18]

CHAPTER 3

Methodology

3.1 Introduction

This methodology will outline this study to evaluate the performance of ensemble machine learning methods for network attack detection in IoT environments. This study methodology involves data preprocessing, feature engineering, model selection, training, evaluation, and interpretation using an IoT dataset RT—IoT2022. The study employs a combination of unsupervised and supervised machine learning techniques. K—means clustering is used for exploratory data analysis, while Random Forest, XGBoost, and k—Nearest Neighbors (k—NN) are used for predictive modelling.

3.2 Data Description

The data set taken for this study is RT-IoT2022 dataset from Kaggle, where it consist of both normal and attack behaviours in IoT systems. It includes various IoT devices and attack types such as DDoS, ARP Poisoning, NMAP scans, and Metasploit Brute Force SSH attacks. In this dataset approximately consist of 120,000 records and 78 features, including network-related attributes like packet counts, flow rates, and flag statuses.

- Source: The dataset was collected using the Zeek network monitoring tool and Flow meter plug-in.

- Features: Network features such as proto, service, flow_ duration, and attack types (target variable: Attack_ type).
- Attacks: Includes a wide range of attacks, such as DOS_ SYN_ Hping, ARP_ poisoning, NMAP_ UDP_ SCAN, and DDOS_ Slowloris

This data has several types of network attacks as well as normal traffic patterns. attack types:

- Denial-of-Service (DoS) SYN Flooding using Hping
- ARP Poisoning.
- NMAP Scans
  - UDP Scan
  - XMAS Tree Scan
  - OS Detection
  - TCP Scan
  - FIN Scan
- Distributed Denial-of-Service (DDoS) Slowloris
- Metasploit Brute Force SSH
- Normal Traffic
  - MQTT
  - ThingSpeak
  - Wipro Bulb

The dataset has class imbalance with some attack types, like DoS SYN Flooding, having a very large number of samples compared to others, such as Metasploit Brute Force SSH. this imbalance is challenge for the machine learning models, which may become biased towards the majority classes and struggle to accurately detect under represented attacks.

when it comes to diversity of attack type this dataset has the wide range of of attacks,from port scans to brute force attempts. Each type of attack presents different characteristics, requiring models to differentiate between subtle variations in network traffic patterns.this density increases the complexity of the detection task,teh model has to learn increases the complexity of the detection task, because of the large number of samples for certain attacks like DoS SYN Flooding, there is a risk that models might overfit to these frequent patterns. This over fitting can lead to poor generalization and the inability to detect less common or novel attacks.

3.3 Data Preprocessing

This study the main step in preparing the dataset for machine learning models. Proper preprocessing ensures that the data is clean, consistent, and in a suitable format for analysis.

- Handling Missing Data

  In this study, handling missing data was a critical step, as missing data is a common challenge in large datasets and can severely affect the performance of machine learning models if not properly addressed. For the RT-IoT2022 dataset, the following approach was implemented to manage missing values effectively.

  1. Forward-Filling (ffill):

     The Python Data Analysis Library, commonly known as Pandas, is an open-source library that offers high-performance, user-friendly data structures and tools for data analysis. A key data structure in Pandas is the Series, a one-dimensional array that can hold various data types. Effectively managing missing data within a Series is crucial for data cleaning and preparation. A particularly useful method for addressing missing values is the ffill() method [7] The ffill() method fills NULL values using the value from the preceding row (or the preceding column if the axis parameter is set to 'columns'). [6] here the method is implemented The dataset was first scanned for missing values, and any detected were filled with the preceding non-missing value using the ffill method.

2. Imputation with Mean Value :

   After forward-filling, each numerical column was examined, and missing values were replaced with the column mean. This step was crucial to ensure that no missing data remained, allowing the machine learning algorithms to process the data without errors.

- Encoding Categorical Variables

  Machine learning models, require input features to be in numerical format. The RT-IoT2022 dataset contained categorical features that needed to be encoded:

  1. Label Encoding

     Categorical features such as proto (protocol type) and service (service type) represent non-numeric network-related attributes. These features were converted to numerical values using label encoding, which assigns an integer to each unique category. Here are Python' s 'panda' s and 'scikit-learn' libraries.is been used. Each unique category within the proto and service columns was assigned a unique integer value, transforming the categorical data into a numerical format compatible with machine learning algorithms. In case of the RT-IoT2022 dataset if 'proto' had values like TCP, UDP, and ICMP, these would be encoded as 0, 1, and 2, respectively.

- Feature Engineering:

  Feature engineering creates new features from existing data to enhance the model's ability to capture the underlying patterns in the data. For the RT-IoT2022 dataset, the following features were engineered:

  1. **pkt_ratio:** ('fwd_pkts_tot') the ratio of forward packets to backward packets('bwd_pkts_tot'), where This feature captures the directionality of network traffic. In network security, a significant imbalance between forward and backward packets might indicate anomalous behaviour, such as certain types of DDoS attacks where outgoing traffic heavily outweighs incoming traffic.
     $\text{pkt\_ratio} = \frac{\text{fwd\_pkts\_tot}}{\text{bwd\_pkts\_tot} + 1}$
  2. **data_pkt_ratio:** the ratio of forward data packets (fwd_data_pkts_tot) to total forward packets (fwd_pkts_tot). This feature helps differentiate between

packets containing data and those not. High ratios could indicate normal operation, while low ratios might suggest malicious activity, such as a flood of empty packets used in a DoS attack.

$$\text{data\_pkt\_ratio} = \frac{\text{fwd\_data\_pkts\_tot}}{\text{fwd\_pkts\_tott} + 1}$$

3. **pkt_per_sec_ratio:** The ratio of forward packets per second (fwd_pkts_per_sec) to backward packets per second (bwd_pkts_per_sec). This feature measures the rate of packet transmission in both directions. Anomalies in this ratio can help detect unusual traffic patterns, such as those seen in DDoS attacks where the rate of outgoing packets may spike.

$$\text{pkt\_per\_sec\_ratio} = \frac{\text{fwd\_pkts\_per\_sec}}{\text{bwd\_pkts\_per\_sec} + 1}$$

- Feature Scaling:

  Feature scaling is a part of data processing where it is the most essential in preparing the dataset for machine learning algorithms, especially those that rely on distance calculations, such as k-means clustering and k-Nearest Neighbors (k-NN).

  StandardScaler:

  Standardization was used to scale the numerical features so that they have a mean of 0 and a standard deviation of 1. StandardScaler ensures that all features contribute equally to the model and prevents any single feature from dominating due to its scale. The StandardScaler from scikit-learn was applied to the numerical columns in the dataset. Each feature was transformed by subtracting the mean and dividing by the standard deviation Scaling improved the performance of the machine learning models by ensuring that features with larger scales did not disproportionately influence the results.

$$X' = \frac{X - \mu}{\sigma}$$

  Where μ is the mean of the feature, and σ is the standard deviation.

- Outlier Detection and Removal

  Outlier detection and removal is a crucial step in data preprocessing, especially in network traffic and security datasets, where anomalous data points can either

represent true attacks or be the result of noise or measurement errors. The presence of outliers can significantly affect the performance of machine learning models, leading to biased results and decreased model accuracy.

3.4 Exploratory Data Analysis using k-means Clustering

Here the k-mean clustering is used to uncover natural groupings within the dataset. The main objective of this clustering is to observe the network traffic data naturally formed clusters that corresponded to attack and normal behaviours.

- Elbow Method for Determining k

  Where the elbow is the method is used here to determine the optimal number of clusters (k). Inertia (sum of squared distances to the nearest cluster centre) was calculated for different values of k, typically ranging from 1 to 10. The point where the decrease in inertia slows down significantly (forming an "elbow") was chosen as the optimal number of clusters. Here in this study after analysing the plot k is 4 or 5 because the elbow seems to occur around k=4 or k=5. This is where the line starts to bend and the reduction in inertia begins to taper off.

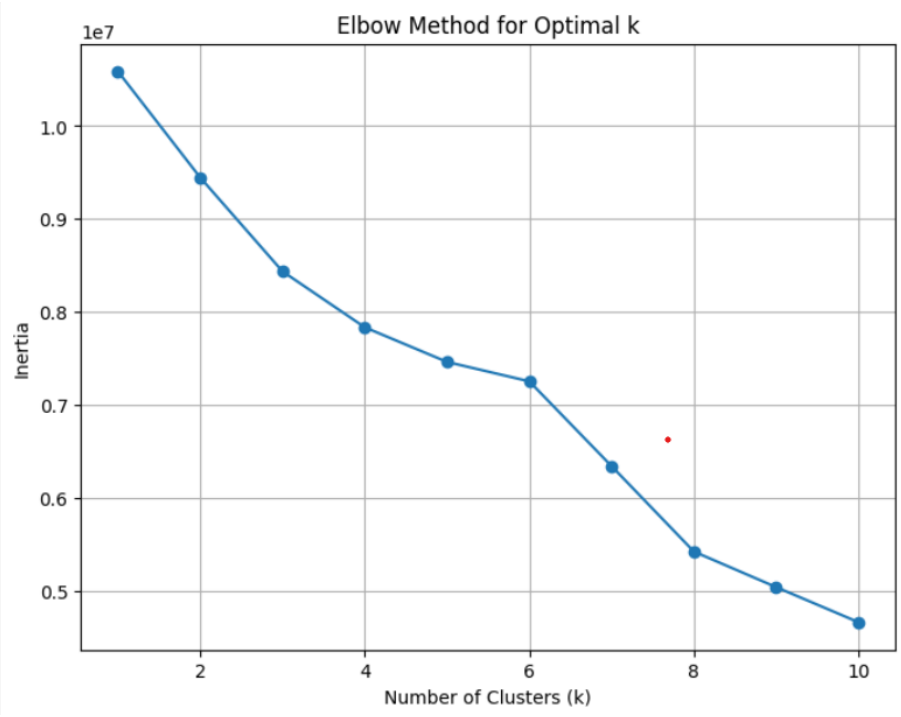

Figure 3.1: elbow

- Applying k-means Clustering

  After identification of the 'k,' the k-mean clustering is applied to the scaled dataset. Each data point was assigned to a cluster, and a new column was added to the dataset representing the cluster label. Here there are 5 cluster for k=5.

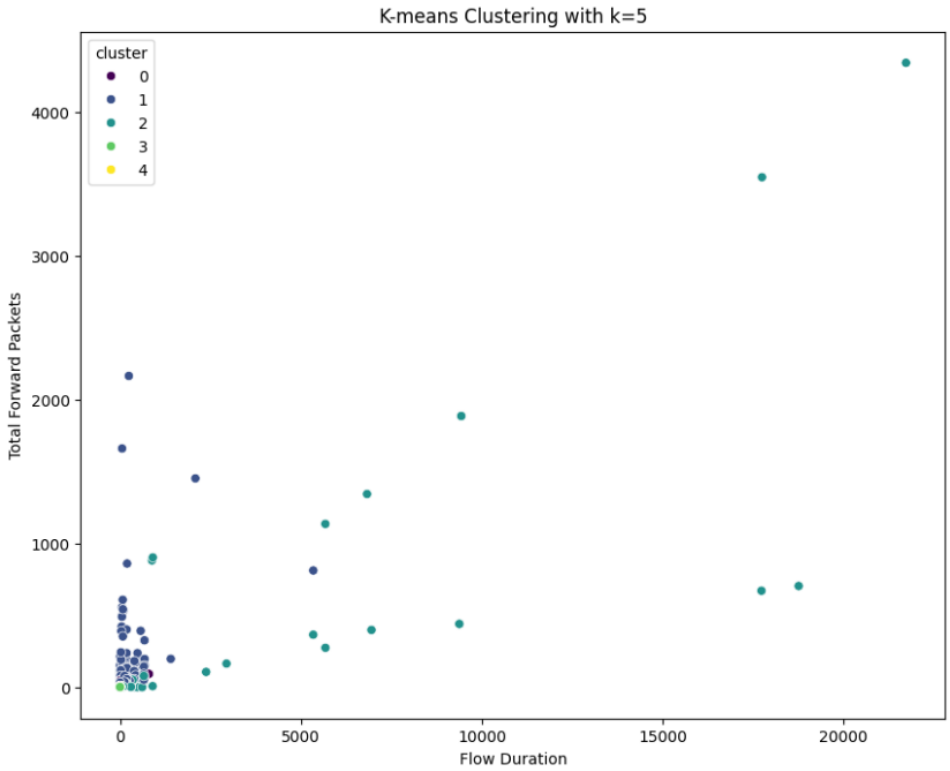

Figure 3.2: cluster

- Visualization and Analysis

  The most important part of clustering is analysing it and Visualization, The clusters were visualized using scatter plots of key features (e.g., flow_duration vs. fwd_pkts_tot).also the distribution of attack types across clusters was analyzed to understand how well the clustering corresponded to different attack behaviours. here all different attacks have been given with different colours to understand properly all attacts in 5 cluster

3.5 Dimensionality Reduction

Principal Component Analysis (PCA) reduces a large dataset with many variables per observation to a smaller set of summary indices. These indices preserve most of the

information from the original variables. Analysts refer to these new values as principal components. [8] Here in this study, it has been reducing the number of features while preserving 95% of the variance in the data. This step helped simplify the dataset, improve computational efficiency, and mitigate the risk of overfitting. Here are the Steps Involved:

- Application of PCA

  PCA was initially fitted on the processed dataset (data_processed) to compute the principal components and the variance explained by each component. The cumulative explained variance for the components was plotted to determine the number of components required to retain a desired level of variance This plot serves as a tool to visualize the trade-off between the number of components and the amount of variance they explain

- Selection of Components

  Here the variance threshold of 95% was set because the selected components should explain at least 95% of the variance in the data. This threshold balances between retaining significant information and reducing the dimensionality. The cumulative explained variance was calculated, and the minimum number of components needed to achieve the desired variance was identified. In this study, 28 components were required to explain 95% of the variance.

- Dimensionality Reduction

  PCA was applied with the selected 28 components to both the training and testing datasets. This transformation resulted in reduced datasets (X_train_reduced and X_test_reduced), which retain most of the information from the original data but with fewer features. By reducing the number of features from the original dataset to 28 principal components, the complexity of the models was reduced, improving computational efficiency and reducing the risk of overfitting

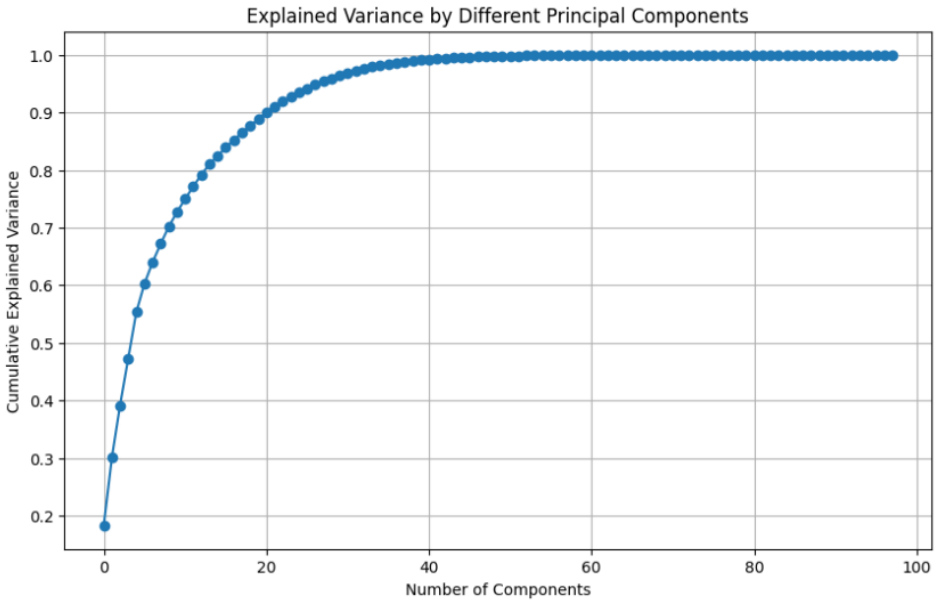

Figure 3.3: PCA

3.6 Data splitting

The main objective of data splitting is to create an independent test set that allows for a robust evaluation of the machine learning models. By training the models on one portion of the data and testing them on a separate, unseen portion, we can better understand how well the models generalize to new, unobserved data. This approach is fundamental in avoiding over fitting, where a model performs well on the training data but poorly on new data.

In this study, the dataset was split into a training set and a testing set, following an 8020 ratio. This means that 80% of the data was used to train the models, while the remaining 20% was set aside for testing. This ratio is a common practice in machine learning, providing a balance between having enough data to train the models effectively and retaining a sufficient portion for robust testing.

The train_test_split function from the scikit-learn library was used to perform the data splitting. This function randomly assigns data points to the training and testing sets while maintaining the specified ratio. A random_state parameter was set to ensure reproducibility of the results, meaning that the data split is consistent across different runs of the experiment.

3.7 Predictive Modelling using Ensemble Methods

The main objective of this study involves applying ensemble methods to classify network traffic as either normal or one of several attack types. The following ensemble methods were evaluated:

Random Forest:

Random Forest (RF) is an ensemble learning technique that constructs multiple decision trees during training, particularly suited for classification tasks. It operates by randomly selecting subsets of the training data (a process known as bootstrapping) and randomly choosing features at each node. The final prediction is made by aggregating the predictions from all the individual trees, typically using majority voting for classification. Random Forest is known for its robustness, ability to handle high-dimensional data, and resistance to overfitting. [9] This method is particularly robust against overfitting, which can be a significant concern with high-dimensional datasets like RT-IoT2022. Random Forest is also known for its ability to handle both numerical and categorical data, making it a versatile choice for this study.

- n_estimators=100: This parameter controls the number of decision trees in the forest. A larger number of trees generally improves performance but also increases computational cost.
- max_depth=10: The maximum depth of the trees was limited to prevent overfitting and to ensure that the model remains interpretable.
- random_state=42: A fixed random seed was used to ensure reproducibility of the results.

the Random Forest model demonstrated strong performance on the RT-IoT2022 dataset, achieving high accuracy, precision, recall, and F1 scores. The model's ability to generalize well to unseen data indicates its robustness and suitability for this classification task.

XGBoost: Extreme Gradient Boosting (XGBoost) is a decision tree-based ensemble method that utilizes the gradient descent algorithm to minimize errors in a sequential model. XGBoost reduces bias by boosting weak learners, where each learner is trained

to correct the errors made by its predecessor. This iterative process results in a highly accurate and efficient model, making XGBoost an effective tool for handling complex classification tasks in network attack detection. [4] XGBoost is well-suited for structured data and has been widely adopted in various machine-learning competitions due to its high efficiency and predictive power.

- n_ estimators=100: The model was set to construct 100 boosting rounds, balancing computational efficiency and model performance.
- max_ depth=3: The depth of each tree was limited to three levels to prevent overfitting, which can occur when the model becomes too complex.
- learning_ rate=0.1: This parameter controls the contribution of each tree to the final model. A lower learning rate makes the model more robust but requires more trees to achieve the same performance.
- random_ state=42: As with Random Forest, a fixed random seed was used to ensure consistent results.

  XGBoost showed excellent performance in classifying network traffic, with metrics similar to those of the Random Forest model. Its ability to handle complex interactions between features and its built-in regularization techniques make it a powerful tool for this kind of task. The model performed particularly well in correctly identifying the major attack types present in the dataset.

k-Nearest Neighbors (k-NN)

k-Nearest Neighbors (k-NN) is a simple, yet effective, non-parametric algorithm for classification and regression tasks. Unlike ensemble methods, k-NN does not require a training phase per se. Instead, it classifies a new data point based on the majority class of its k nearest neighbours in the feature space. This method is intuitive and can provide competitive performance, especially when the relationship between features and classes is relatively straightforward. [10]

- n_ neighbors=5: Based on empirical testing, the number of nearest neighbors was set to 5. This choice balances bias and variance, avoiding the extremes of underfitting or overfitting.

- Distance Metric: The Euclidean distance metric was used to measure the similarity between data points, which is standard for k-NN. The simplicity of this metric makes it a good fit for the normalized data used in this study.

  The k-NN algorithm performed surprisingly well on the RT-IoT2022 dataset. It achieved high accuracy, particularly in identifying the most common attack types. However, the model struggled with less frequent attack types, which is a known limitation of k-NN in imbalanced datasets. Nonetheless, k-NN serves as a useful baseline for comparison with more complex models like Random Forest and XGBoost.

CHAPTER 4

Results and Analysis

The results obtained from all the machine learning models implemented for the detection of network attacks in the RT-IoT2022 dataset. The models include Random Forest, Gradient Boosting, k-Nearest Neighbors (k-NN), and K-means clustering for exploratory analysis. Each model's performance is evaluated in terms of accuracy, precision, recall, and F1-score. Additional to that, the distribution of attack types across different clusters identified by the K-means algorithm.

- Random Forest Model

  The Random Forest model was trained and tested on the dataset after applying Principal Component Analysis (PCA) to reduce dimensionality. the Classification Report and model's performance metrics as follows: **performance metrics**

  - Accuracy: 0.9964
  - Precision: 0.9964
  - Recall: 0.9964
  - F1-Score: 0.9963

Classification Report

Class	Precision	Recall	F1-score	Support
ARP_poisoning	0.96	0.99	0.97	2306
DDOS_Slowloris	1.00	0.97	0.98	154
DOS_SYN_Hping	1.00	1.00	1.00	28409
MQTT_Publish	1.00	1.00	1.00	1273
Metasploit_Brute_Force_SSH	1.00	0.55	0.71	11
NMAP_FIN_SCAN	1.00	0.86	0.92	7
NMAP_OS_DETECTION	1.00	1.00	1.00	622
NMAP_TCP_scan	1.00	0.99	1.00	319
NMAP_UDP_SCAN	1.00	0.98	0.99	750
NMAP_XMAS_TREE_SCAN	1.00	0.99	1.00	582
Thing_Speak	0.99	0.98	0.98	2424
Wipro_bulb	0.95	0.72	0.82	79
Accuracy		0.9964		
Macro Avg	0.99	0.92	0.95	36936
Weighted Avg	1.00	1.00	1.00	36936

Table 4.1: Classification report of the Random Forest Model.

The confusion matrix for the Random Forest model shows that the model performed exceptionally well across most classes, with high precision and recall. However, there were minor misclassification's in categories such as Metasploit Brute Force SSH and Wipro bulb, where the model's precision and recall were slightly lower than for other classes.

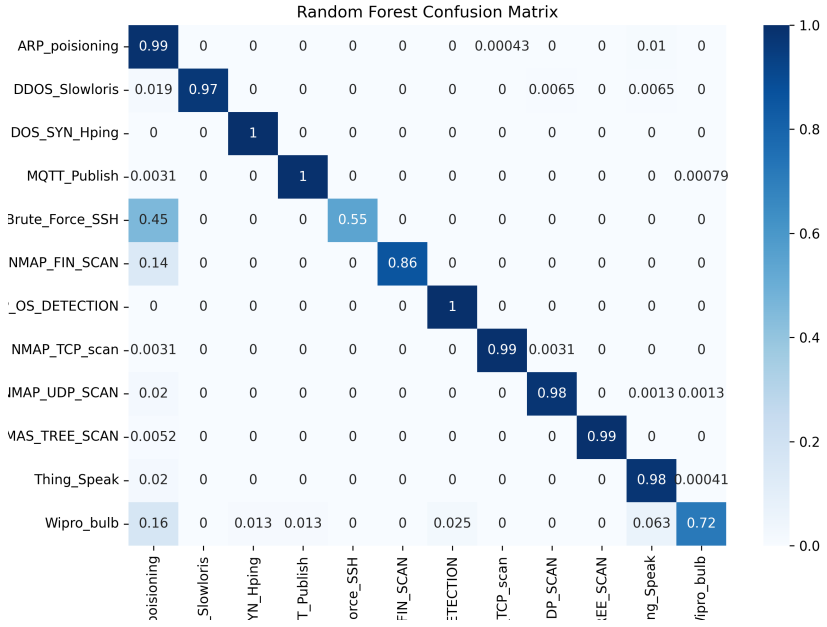

Figure 4.1: Random forest confusion matrix

- XGBoosting Model:

  The Gradient Boosting model was also trained on the reduced dataset. The performance of this model is slightly lower than the Random Forest, but still impressive:

  performance metrics

  - Accuracy: 0.9952
  - Precision: 0.9952
  - Recall: 0.9952
  - F1-Score: 0.9951

  Classification Report

Class	Precision	Recall	F1-score	Support
ARP_poisoning	0.96	0.97	0.97	2306
DDOS_Slowloris	1.00	0.97	0.98	154
DOS_SYN_Hping	1.00	1.00	1.00	28409
MQTT_Publish	1.00	1.00	1.00	1273
Metasploit_Brute_Force_SSH	1.00	0.64	0.78	11
NMAP_FIN_SCAN	1.00	0.86	0.92	7
NMAP_OS_DETECTION	1.00	1.00	1.00	622
NMAP_TCP_scan	1.00	1.00	1.00	319
NMAP_UDP_SCAN	0.99	0.97	0.98	750
NMAP_XMAS_TREE_SCAN	1.00	0.99	1.00	582
Thing_Speak	0.97	0.97	0.97	2424
Wipro_bulb	0.97	0.76	0.85	79
Accuracy		0.9952		
Macro Avg	0.99	0.93	0.95	36936
Weighted Avg	1.00	1.00	1.00	36936

Table 4.2: Detailed classification report for XGBoosting Model.

As with the Random Forest, the Gradient Boosting model achieved high accuracy, but there were more noticeable errors in predicting the Metasploit Brute Force SSH and Wipro bulb classes. These errors indicate that while Gradient Boosting is powerful, it may require further tuning or feature engineering to handle certain classes more effectively.

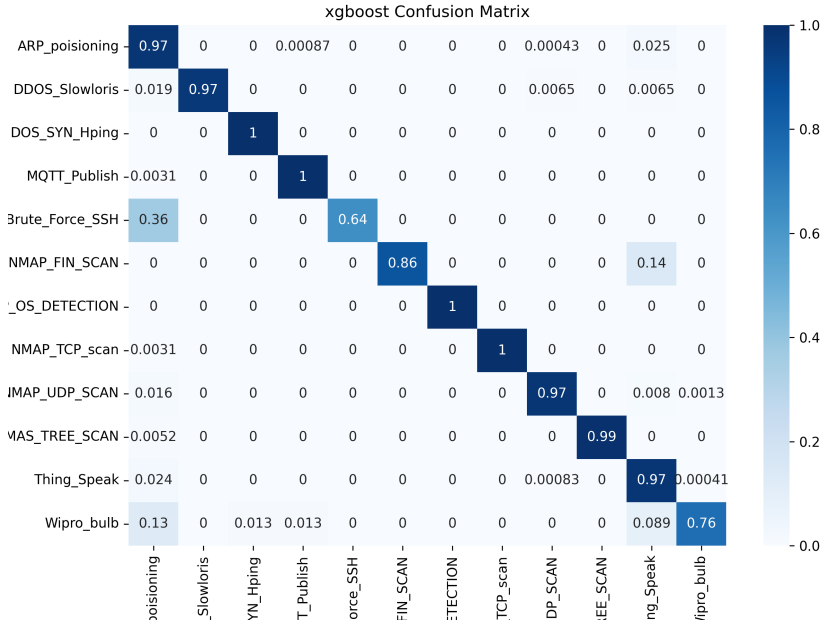

Figure 4.2: xgboost Confusion Matrix

- k-Nearest Neighbors (k-NN) Model:

  The k-NN model was also evaluated on the reduced dataset. The k-NN model performed similarly to the Gradient Boosting model, with the following classification report:

  - Accuracy: 0.9956
  - Precision: 0.9956
  - Recall: 0.9956
  - F1-Score: 0.9955

Class	Precision	Recall	F1-score	Support
ARP_poisoning	0.96	0.98	0.97	2306
DDOS_Slowloris	0.95	0.94	0.94	154
DOS_SYN_Hping	1.00	1.00	1.00	28409
MQTT_Publish	1.00	1.00	1.00	1273
Metasploit_Brute_Force_SSH	1.00	0.55	0.71	11
NMAP_FIN_SCAN	1.00	0.86	0.92	7
NMAP_OS_DETECTION	1.00	1.00	1.00	622
NMAP_TCP_scan	0.99	0.99	0.99	319
NMAP_UDP_SCAN	0.99	0.97	0.98	750
NMAP_XMAS_TREE_SCAN	1.00	0.99	1.00	582
Thing_Speak	0.98	0.98	0.98	2424
Wipro_bulb	0.98	0.73	0.84	79
Accuracy		0.9956		
Macro Avg	0.99	0.92	0.94	36936
Weighted Avg	1.00	1.00	1.00	36936

Table 4.3: Detailed Classification Report for the k-Nearest Neighbors

The confusion matrix for k-NN shows that, like the other models, it struggles with the Metasploit Brute Force SSH and Wipro bulb classes. However, it performs extremely well in predicting the major attack types such as DOS_ SYN_ Hping and MQTT_ Publish.

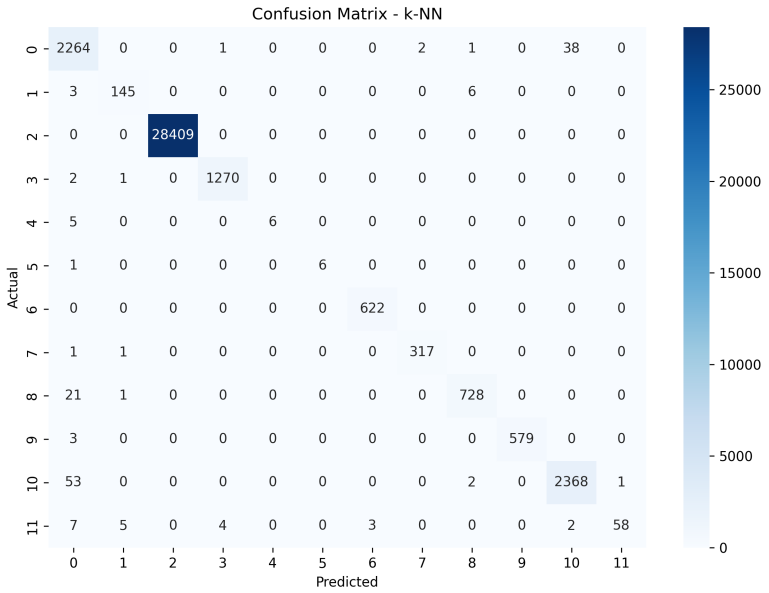

Figure 4.3: Confusion Matrix k-NN

- K-means Clustering:

  K-means clustering was used for exploratory data analysis to identify potential natural groupings within the dataset. The elbow method was used to determine the optimal number of clusters, which was found to be 5 based on the elbow plot. The clusters identified by the K-means algorithm showed the following distribution of attack types:

  - Cluster 0: Predominantly contains DOS_SYN_Hping attacks.
  - Cluster 1: A mix of several attacks, including ARP_poisioning and NMAP-related scans.
  - Cluster 2: Mostly contains MQTT_Publish and minor classes like Wipro_bulb.
  - Cluster 3: Contains a high number of NMAP scans and other less frequent attacks.
  - Cluster 4: Mainly consists of DOS_SYN_Hping attacks.

  The majority of the data points belong to Cluster 0, with a heavy concentration of DOS_SYN_Hping (green), which overwhelms the other attack types in this cluster. This suggests that DOS_SYN_Hping attacks dominate the network traffic within

the dataset, accounting for a significant portion of the dataset. Other attacks, such as ARP_poisioning, DDOS_Slowloris, and NMAP_TCP_scan, also exist in Cluster 0, but their representation is minimal compared to DOS_SYN_Hping.

The Cluster 4 also contains a large number of data points, dominated again by DOS_SYN_Hping (green), reinforcing the idea that this attack type is prevalent throughout the dataset. The near absence of other attack types in this cluster indicates that Cluster 4 likely represents a continuation or extension of the same type of DOS_SYN_Hping behavior, albeit with different characteristics that set it apart from Cluster 0.

Cluster 3 exhibits a diverse range of attack types, including ARP_poisioning, NMAP scans (various types), MQTT_Publish, and Thing_Speak. This cluster highlights multiple attack vectors, making it more heterogeneous than Clusters 0 and 4. The presence of multiple attack types suggests that this cluster captures more complex or mixed behaviors in network traffic, rather than being dominated by a single attack type.

Cluster 1 and Cluster 2 have relatively few data points. Cluster 1 shows a small concentration of ARP_poisioning, DDOS_Slowloris, and Thing_Speak attacks, while Cluster 2 has a modest number of MQTT_Publish and ARP_poisioning attacks. These clusters might represent rare or less intense attacks compared to the larger, more populated clusters (0, 3, 4).

Attack types like Metasploit_Brute_Force_SSH, NMAP_FIN_SCAN, and NMAP_UDP_SCAN are present in low quantities across several clusters, particularly in Clusters 0, 1, and 3. Their presence across different clusters indicates that these attacks are distributed across different network behaviors but do not form a dominant group in any specific cluster.

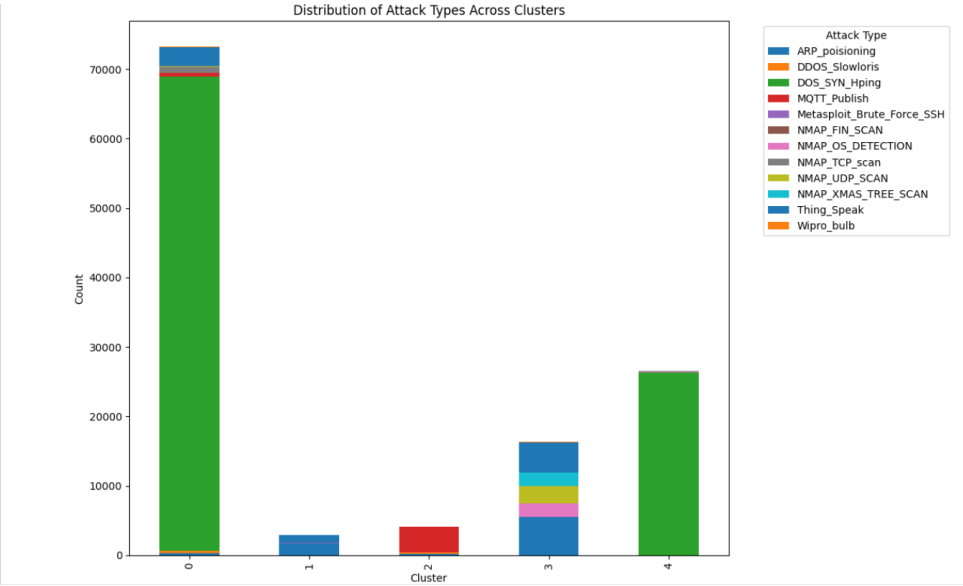

Figure 4.4: k-mean cluster

- Cross-Validation Results

  For this study, 5-fold cross-validation was applied to assess the consistency and performance of the machine learning models for network attack detection. This method splits the dataset into five equal parts, trains the model on four parts, and tests it on the remaining part. The process is repeated five times, each time with a different test fold, ensuring the robustness of the model's performance. Below are the detailed results for each model [14] [19]:

  – Random Forest Cross-Validation Accuracy: 0.9963 ± 0.0004

    The Random Forest model has exceptional performance with an accuracy close to 99.63%. This shows that the model can consistently classify network traffic with high precision. The very small standard deviation (± 0.0004) indicates that the model's predictions are stable across different data folds, meaning it generalizes well to unseen data without significant performance drops. Random Forest's robustness stems from its ensemble nature, where multiple decision trees contribute to the final prediction, minimizing the impact of individual misclassification's. This result suggests that Random Forest could be a preferred choice for real-time intrusion detection systems, as it offers a combination of high accuracy and reliability, ensuring minimal false positives and negatives.

- XGBoost Cross-Validation Accuracy: 0.9956 ± 0.0004

  XGBoost performers as well as Random Forest, achieving an accuracy of 99.56%. While slightly lower, this model showed remarkable consistency with a small margin of error (± 0.0004). XGBoost's ability to sequentially boost weak learners enables it to correct errors made by previous models, providing a high level of precision and adaptability in detecting complex patterns in network traffic. Given its excellent performance and scalability, XGBoost would be an effective model in environments with large volumes of network data that require fast, accurate intrusion detection.

- k-Nearest Neighbour (k-NN) Cross-Validation Accuracy: 0.9959 ± 0.0001

  Despite its simplicity, k-NN delivered impressive results with an accuracy of 99.59%, closely matching the performance of more sophisticated models like Random Forest and XGBoost. The extremely low standard deviation (± 0.0001) shows that k-NN's performance was incredibly consistent across all data folds. This result is noteworthy because k-NN is a non-parametric method that relies on the distance between data points to classify network traffic. k-NN's ability to maintain high accuracy, even as a simpler model, highlights its utility for network attack detection, particularly in environments where high interpret-ability and minimal computational resources are key factors.

Overall, of cross-validation results underscore the effectiveness of ensemble methods like Random Forest and XGBoost in detecting IoT-based network attacks, but also highlight the surprising strength of simpler models like k-NN in similar tasks. Each model has its advantages: Random Forest and XGBoost offer robust performance with high-dimensional data, while k-NN provides a lightweight and efficient alternative for quick classification tasks.

- Analysis of Results

  This study indicate that the three machine learning models used—Random Forest, Gradient Boosting, and k-Nearest Neighbors (k-NN)—achieved highly competitive results when applied to the RT-IoT2022 dataset. Each model performed with outstanding accuracy, precision, recall, and F1-scores, with values consistently

at or above 99%. Among the models, Random Forest slightly outperformed the others, showcasing its robustness and effectiveness in handling high-dimensional data and varied types of network traffic. This success can be attributed to its ensemble nature, which reduces over fitting and improves generalization.

All models exhibited some limitations, In classifying rare attack types such as Metasploit Brute Force SSH and Wipro bulb. These classes had fewer instances in the dataset, making it more difficult for the models to learn their patterns effectively. In machine learning, class imbalance can lead to skewed predictions, where models may perform exceptionally well on more prevalent classes while struggling with under-represented ones. Future work could address this issue by applying techniques like data augmentation, oversampling, or adjusting class weights to help the models better learn from under-represented classes.

In addition, the use of k-means clustering revealed valuable insights about the natural groupings of attack types within the network traffic data. The clustering analysis showed that certain attack types tend to cluster together, indicating similarities in their behaviour. For instance, attacks such as DOS_SYN_Hping and NMAP_UDP_SCAN may be more closely related in terms of packet-level behaviours or network characteristics. These findings provide a foundation for developing more nuanced detection models that not only classify attacks but also identify relationships between them, potentially enhancing detection accuracy.

An important future direction could be the integration of clustering with classification, where k-means clustering is used to pre-process the data by grouping similar attack types. Then, classification models like Random Forest or Gradient Boosting could be applied to refine the identification of specific attacks within those clusters. This hybrid approach could improve the detection of under-represented classes by leverage the inherent structure of the data, making the overall model more robust against class imbalance.

- Comparison of Results:

  Overall Performance: Random Forest slightly outperformed XGBoost in accuracy and F1-score, but both models delivered excellent results across the board. k-NN also delivered strong results but was slightly behind the ensemble methods due

to its simplicity and sensitivity to class distribution.

Handling of Rare Classes: Both Random Forest and XGBoost showed struggles with under-represented classes like Metasploit Brute Force SSH. However, ensemble methods generally handled these issues better than k-NN, which requires balanced and well-distributed data for optimal performance.

CHAPTER 5

Conclusion

The fast proliferation of Internet of Things (IoT) devices has introduced new opportunities, but also significant security challenges. [1] As IoT networks increase in complexity, they become more vulnerable to sophisticated cyder attacks. This study aimed to explore predictive modelling approaches using ensemble machine learning methods to enhance the detection and classification of network attacks in IoT systems. By utilizing the RT-IoT2022 dataset, which includes both normal and adversarial network behaviours from various IoT devices, we focused on improving the capabilities of Intrusion Detection Systems (IDS) for real-time threat detection. The primary objective was to identify the most effective machine learning models for detecting network attacks and to analyse the underlying features contributing to detection accuracy. Several models were applied, including Random Forest, XGBoost, k-nearest neighbour (k-NN), and k-means clustering, to classify network traffic and detect various attack types such as Denial-of-Service (DoS), Distributed Denial-of-Service (DDoS), ARP poisoning, and NMAP scans.

- Random Forest and XGBoost Performance: Both Random Forest and XGBoost demonstrated superior performance in terms of accuracy, precision, recall, and F1-score. Random Forest achieved an accuracy of 99.64%, closely followed by XGBoost with 99.51%. These models were particularly effective due to their ability to handle high-dimensional data and address the complexities of IoT network traffic. As demonstrated in prior research [16], ensemble methods like Random

Forest and XGBoost have been validated for their robustness and accuracy in network traffic classification.

- k-NN Effectiveness: The k-NN algorithm delivered strong results, achieving a cross-validation accuracy of 99.56%. Its ability to classify network activity based on proximity to known attack types made it useful for detecting repeated attack patterns, as seen in previous studies [15].

- K-Means Clustering: K-means clustering was employed to explore and identify natural groupings within network traffic. By analyzing clustered traffic patterns, insights into potential attack groupings were obtained, offering a deeper understanding of the segmentation of network behaviours. The optimal number of clusters (k=5) allowed meaningful distinctions between attack types, reflecting similar findings in IoT traffic clustering research [11].

- Feature Importance and Dimensionality Reduction: Feature engineering and Principal Component Analysis (PCA) were applied to reduce the dimensionality of the dataset while retaining 95% of the variance. This helped improve computational efficiency without sacrificing critical information. Similar to findings by [1], key features such as packet ratios and flow metrics contributed significantly to model performance, enhancing the detection of both normal and malicious network behaviours.

5.1 Contributions

This study contributes to the evolving field of IoT security by demonstrating the efficacy of ensemble machine-learning methods in detecting network attacks. By combining models like Random Forest, XGBoost, k-NN, and k-means clustering, we developed a comprehensive framework capable of accurately classifying network traffic. These findings align with current research, such as [12], which underscores the importance of using multiple machine learning models for better accuracy and resilience in IoT security.

5.2 Future Directions and Gaps

the machine learning-based models have shown great promise, challenges remain in terms of scalability, interpretability, and adaptability to emerging threats. Future research should focus on improving the real-time detection capabilities of these models, as well as addressing privacy concerns associated with IoT data [13]. Moreover, the development of lightweight models for resource-constrained IoT devices remains an open research problem, particularly in environments where computational resources are limited [14].

Bibliography

[1] Systematic Literature Review of Machine Learning for IoT Security, 2023.

[2] Top Applications of IoT in the World, GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/top-applications-of-iot-in-the-world/. [Accessed: 10-Sept-2024].

[3] Security Frameworks for Internet of Things Systems – A Comprehensive Survey, 2023.

[4] P. Thorat and N. Dubey, "SDN-based Machine Learning Powered Alarm Manager for Mitigating the Traffic Spikes at the IoT Gateways," IEEE International Conference on Advanced Networking and Telecommunications, 2020.

[5] Security and Privacy Concerns Associated with the Internet of Things, 2022.

[6] Pandas DataFrame ffill() Method, w3schools.com. [Online]. Available: https://www.w3schools.com/python/pandas/ref_dataframe_ffill.asp. [Accessed: 10-Sept-2024].

[7] Understanding pandas.Series.ffill() method (with examples), Sling Academy. [Online]. Available: https://www.slingacademy.com/understanding-pandas-series-ffill-method/. [Accessed: 10-Sept-2024].

[8] Principal Component Analysis Guide & Example, Statistics By Jim. [Online]. Available: https://www.statisticsbyjim.com/basics/principal-component-analysis/. [Accessed: 10-Sept-2024].

[9] R. Mahajan and I. Siddavatam, “A Weighted Ensemble Model for Phishing Website Detection using Random Forest and Deep Neural Network,” Proceedings of International Conference, 2023.

[10] S. Mathankar, S. R. Sharma, T. Wankhede, and M. Sahu, “Phishing Website Detection using Machine Learning Techniques,” 11th International Conference on Emerging Trends in Engineering & Technology - Signal and Information Processing (ICETET - SIP), 2023.

[11] P. Thorat and N. Dubey, “Deep Ensemble Learning With Pruning for DDoS Attack Detection in IoT Networks,” IEEE Access, 2020.

[12] IoT Device Fingerprinting Machine Learning Based Encrypted Traffic Analysis, 2021.

[13] F. Ghaleb, et al., “Cyber Threat Intelligence-Based Malicious URL Detection Model Using Ensemble Learning,” Sensors, vol. 22, no. 04, 2022.

[14] M. Frustaci, et al., “Evaluating Critical Security Issues of the IoT World: Present and Future Challenges,” IEEE Internet of Things Journal, vol. 5, no. 4, pp. 2483-2495, 2018.

[15] IoT Network Traffic Classification Using Machine Learning Algorithms: An Experimental Analysis, 2022.

[16] Research of Network Traffic Classification Based on Ensemble Learning, 2022.

[17] An Accuracy Network Anomaly Detection Method Based on an Ensemble Model, 2023.

[18] S. S. M. Ajibade, M. B. Jasser, A. R. Gimeno, A. O. Adediran, L. A. Ypanto, R. F. B. Legaspino, and J. C. C. Baird, "Analysis of Machine Learning (ML) and Internet of Things (IoT): A Scientometric Review," in *20th IEEE International Colloquium on Signal Processing & Its Applications (CSPA)*, 2024, DOI: 10.1109/CSPA60979.2024.10525638.

[19] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, vol. 2, no. 12, pp. 1137-1143, 1995.

[20] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Journal Name*, 2015.

[21] OWASP IoT Project, "Provides an overview of common security issues in IoT." [Online]. Available: https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project.

[22] A. Antonakakis et al., "Mirai and IoT Botnet: A Malware Attack and Analysis," 2017.

[23] "The Guardian/BBC coverage of the 2021 Florida water treatment facility cyberattack," The Guardian/BBC, February 2021. [Online]. Available: https://www.urltoarticle.com.

Appendix:

```
import pandas as pd

import numpy as np

import seaborn as sns

import shap

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix

import shap

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
AdaBoostClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

from sklearn.model_selection import cross_val_score, train_test_split

from sklearn.ensemble import GradientBoostingClassifier

import xgboost as xgb

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from sklearn.ensemble import RandomForestClassifier
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix

data = pd.read_csv('D:\\msc_data_science\\New folder\\archive (12)\\RT_IOT2022.csv')
print(data)

#plot for attack type distribution
import seaborn as sns
import matplotlib.pyplot as plt
# plot for attack type distribution
# Visualizations
sns.countplot(data['Attack_type'])
plt.title('Distribution of Attack Types')
plt.show()

# Forward fill using ffill() method
data = data.ffill()

# Fill remaining missing values in numeric columns with the mean of each column
```

```
numeric_cols = data.select_dtypes(include=['number']).columns
data[numeric_cols] = data[numeric_cols].fillna(data[numeric_cols].mean())

# Define a threshold for the proportion of zeros
zero_threshold = 0.5

# Calculate the proportion of zeros in each row
zero_proportion = (data == 0).sum(axis=1) / data.shape[1]

# Filter out rows where the proportion of zeros is greater than the threshold
data_cleaned = data[zero_proportion < zero_threshold]

# Reset the index if you want the cleaned DataFrame to have a continuous index
data_cleaned.reset_index(drop=True, inplace=True)

# Display the shape of the original and cleaned data to understand the impact
original_shape = data.shape
cleaned_shape = data_cleaned.shape

original_shape, cleaned_shape

def remove_outliers(df):
    # For each numerical column, remove outliers
    for col in df.select_dtypes(include=[np.number]).columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
```

```
    return df

# Apply the function to the dataset
data_no_outliers = remove_outliers(data_cleaned)

# Display the shape of the data before and after removing outliers
original_cleaned_shape = data_cleaned.shape
no_outliers_shape = data_no_outliers.shape

original_cleaned_shape, no_outliers_shape

# Calculate the percentage of missing values in each column
missing_percentage = data.isnull().mean()

# Identify columns with more than 50% missing values
columns_to_drop = missing_percentage[missing_percentage > 0.5].index

# Drop the identified columns
data_cleaned = data.drop(columns=columns_to_drop)
data.shape

# Ensure the target variable exists
target_variable = 'Attack_type'
if target_variable not in data.columns:
    raise ValueError(f"Target variable '{target_variable}' not found in data")

# Convert categorical columns to numerical using encoding
data_cleaned['proto'] = data_cleaned['proto'].astype('category').cat.codes
data_cleaned['service'] = data_cleaned['service'].astype('category').cat.codes
### Feature Engineering
# Create new features
```

```
data_cleaned['pkt_ratio'] = data_cleaned['fwd_pkts_tot'] / (data_cleaned['bwd_pkts_tot'] + 1)

data_cleaned['data_pkt_ratio'] = data_cleaned['fwd_data_pkts_tot'] / (data_cleaned['fwd_pkts_tot'] + 1)

data_cleaned['pkt_per_sec_ratio'] = data_cleaned['fwd_pkts_per_sec'] / (data_cleaned['bwd_pkts_per_sec'] + 1)


# Get the list of column names

column_names = data_cleaned.columns.tolist()

print("Column Names:")

print(column_names)


## StandardScaler


scaler = StandardScaler()

numerical_data = data_cleaned.select_dtypes(include=[np.number])

data_scaled = scaler.fit_transform(numerical_data)

data_scaled = pd.DataFrame(data_scaled, columns=numerical_data.columns)


categorical_data = data_cleaned.select_dtypes(include=[object])

data_encoded = pd.get_dummies(categorical_data, drop_first=True)


data_processed = pd.concat([data_scaled, data_encoded], axis=1)

print(data_cleaned.info())

print(data_cleaned.describe())

# Ensure the target variable exists

target_variable = 'Attack_type'

y = data_cleaned[target_variable]

data_processed = data_processed.drop(columns=[target_variable], errors='ignore')

# Get the list of column names

column_names = data_cleaned.columns.tolist()

print("Column Names:")

print(column_names)
```

k mean

```
# Applying k-means clustering

# Determine the optimal number of clusters using the elbow method
inertia = []  # Inertia is the sum of squared distances to the nearest cluster centre
K = range(1, 11)  # trying k = 1 to 10 clusters

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_scaled)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.figure(figsize=(8, 6))
plt.plot(K, inertia, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()
# Select optimal k based on the elbow plot (e.g., assume k=3 based on the elbow)
optimal_k = 5  # Change this based on the elbow plot

# Apply k-means with the chosen number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
data['cluster'] = kmeans.fit_predict(data_scaled)

# Visualize the clusters
plt.figure(figsize=(10, 8))
sns.scatterplot(data=data, x='flow_duration', y='fwd_pkts_tot', hue='cluster', palette='viridis')
```

```
plt.title(f'K-means Clustering with k={optimal_k}')
plt.xlabel('Flow Duration')
plt.ylabel('Total Forward Packets')
plt.show()
# Analyze the distribution of attack types across clusters
cluster_attack_distribution = pd.crosstab(data['cluster'], data['Attack_type'])
print(cluster_attack_distribution)

# Plot the distribution of attacks within each cluster
cluster_attack_distribution.plot(kind='bar', stacked=True, figsize=(12, 8))
plt.title('Distribution of Attack Types Across Clusters')
plt.xlabel('Cluster')
plt.ylabel('Count')
plt.legend(title='Attack Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

splitting the data

```
# Split Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data_processed, y, test_size=0.3, random_state=42)
# Instantiate the LabelEncoder
label_encoder = LabelEncoder()

# Encode the target variable
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

PCA

```
# Fit PCA
```

```
pca = PCA().fit(data_processed)

# Plot the cumulative explained variance
plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance by Different Principal Components')
plt.grid(True)
plt.show()

# Select the number of components to explain a desired amount of variance
desired_variance = 0.95
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
n_components = np.argmax(cumulative_variance >= desired_variance) + 1

print(f"Number of components to explain {desired_variance * 100}% variance: {n_components}")

# Apply PCA with the selected number of components to both training and test data
pca = PCA(n_components=n_components)
X_train_reduced = pca.fit_transform(X_train)
X_test_reduced = pca.transform(X_test)

# Train Random Forest with Hyperparameter Tuning
rf_model = RandomForestClassifier(n_estimators=100, max_depth=10, min_samples_split=10,
min_samples_leaf=5, random_state=42)
rf_model.fit(X_train_reduced, y_train_encoded)

# Get feature importances
feature_importances = rf_model.feature_importances_
```

```
# Create a DataFrame for visualization using PCA component names
importance_df = pd.DataFrame({
    'Feature': [f'PC{i+1}' for i in range(len(feature_importances))],  # PCA components
    'Importance': feature_importances
})

# Sort by importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot the feature importance
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=importance_df.head(20))  # Show top 20 features
plt.title('Top 20 Important Features in Random Forest')
plt.show()
```

randome forest

```
# Make predictions
rf_pred = rf_model.predict(X_test_reduced)
rf_pred_train = rf_model.predict(X_train_reduced)
def evaluate_model(y_test, y_pred, model_name):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"{model_name} Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print()
# Random Forest Evaluation
```

```
evaluate_model(y_test_encoded, rf_pred, "Random Forest")
evaluate_model(y_train_encoded, rf_pred_train, "Random Forest")

# Calculate accuracy
accuracy = accuracy_score(y_test_encoded, rf_pred)
print(f'Accuracy: {accuracy:.4f}')

# Print classification report
print('Classification Report:')
print(classification_report(y_test_encoded, rf_pred, target_names=label_encoder.classes_))

# Cross-Validation
def cross_validation(model, X, y, model_name):
    scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    print(f"{model_name} Cross-Validation Accuracy: {np.mean(scores):.4f} ± {np.std(scores):.4f}")

# Random Forest Cross-Validation
cross_validation(rf_model, X_train_reduced, y_train_encoded, "Random Forest")

# Confusion matrix for Random Forest
cm_rf = confusion_matrix(y_test_encoded, rf_pred)
cm_normalized_rf = cm_rf.astype('float') / cm_rf.sum(axis=1)[:, np.newaxis]

plt.figure(figsize=(10, 7))
sns.heatmap(cm_normalized_rf, annot=True, cmap='Blues', xticklabels=label_encoder.classes_,
yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Random Forest Confusion Matrix')
# Save the figure
plt.savefig('random_forest_confusion_matrix.png', dpi=300)
```

```
plt.show()

# Save the figure

#plt.savefig('random_forest_confusion_matrix.png', dpi=300)

#plt.close()
```

XGboosting

```
from sklearn.preprocessing import LabelEncoder

import xgboost as xgb



# Instantiate the model

xgb_model = xgb.XGBClassifier(n_estimators=100, max_depth=3, learning_rate=0.1,
random_state=42, n_jobs=-1)


# Train the model

xgb_model.fit(X_train_reduced, y_train_encoded)


# Make predictions

xgb_pred = xgb_model.predict(X_test_reduced)

xgb_pred_decoded = label_encoder.inverse_transform(xgb_pred)


# If you want to see the decoded predictions

print(xgb_pred_decoded)


# Evaluate the model

# Calculate accuracy

accuracy = accuracy_score(y_test_encoded, xgb_pred)

print(f'Accuracy: {accuracy}')


# Print classification report

print('Classification Report:')
```

```
print(classification_report(y_test_encoded, xgb_pred, target_names=label_encoder.classes_))

# Print confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test_encoded, xgb_pred))

def evaluate_model(y_test, y_pred, model_name):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"{model_name} Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print()
# xgboosting Evaluation
evaluate_model(y_test_encoded, xgb_pred, "xgb_model")
# Encode the target variable for cross-validation
y_encoded = label_encoder.fit_transform(y)

# Define the cross-validation function
def cross_validation(model, X, y, model_name):
    scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    print(f"{model_name} Cross-Validation Accuracy: {np.mean(scores):.4f} ± {np.std(scores):.4f}")

# Perform cross-validation
cross_validation(xgb_model, X_train_reduced, y_train_encoded, "XGBoost")
# Confusion matrix
cm = confusion_matrix(y_test_encoded, xgb_pred)
```

```
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


plt.figure(figsize=(10, 7))
sns.heatmap(cm_normalized, annot=True, cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('xgboost Confusion Matrix')
# Save the figure
plt.savefig('xgboost Confusion Matrix.png', dpi=300)
plt.show()
# Save the figure
#plt.savefig('xgboost Confusion Matrix.png', dpi=300)
#plt.close()
```

k-NN model

```
# Initialize the k-NN model
knn_model = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors

# Train the model
knn_model.fit(X_test_reduced, y_test_encoded)

# Make predictions
knn_pred = knn_model.predict(X_test_reduced)
# Evaluate the model
def evaluate_model(y_test, y_pred, model_name):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"{model_name} Performance:")
```

```
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print()
# k-NN Evaluation
evaluate_model(y_test_encoded, knn_pred, "k-NN")
# Cross-Validation for k-NN
cross_validation(knn_model, X_train_reduced, y_train_encoded, "k-NN")
# Evaluate the model
# Calculate accuracy
accuracy = accuracy_score(y_test_encoded, knn_pred)
print(f'Accuracy: {accuracy}')

# Print classification report
print('Classification Report:')
print(classification_report(y_test_encoded, knn_pred, target_names=label_encoder.classes_))

# Print confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test_encoded, knn_pred))

# Plot your confusion matrix (or any other graph)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - k-NN')
plt.xlabel('Predicted')
plt.ylabel('Actual')

# Save the figure first
plt.savefig('Confusion_Matrix_k-NN.png', dpi=300)
```

```
# Then show the plot
plt.show()
```