

# **IoT Botnet Detection in Home IoT Environment using Machine Learning**



**FONG ZI KHANG**

**This report is submitted in partial fulfillment of the requirements for the  
Bachelor of Computer Science (Computer Security) with Honours.**


**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY  
UNIVERSITI TEKNIKAL MALAYSIA MELAKA**

## DECLARATION

I hereby declare that this project report entitled  
**IoT Botnet Detection in Home IoT Environment using Machine Learning**  
is written by me and is my own effort and that no part has been plagiarized without  
citations.

STUDENT :  Date : 26/08/2024  
(FONG ZI KHANG)

I hereby declare that I have read this project report and found this project report is  
sufficient in term of the scope and quality for the award of Bachelor of Computer  
Science (Computer Security) with Honours.

SUPERVISOR :  Date : 26/08/2024  
(ASSOC. PROF. DR. MOHD. FAIZAL ABDOLLAH)

## ACKNOWLEDGEMENTS

During the completion of this project, I received support and help from many people. Here, I would like to express my sincerest gratitude to all those who helped me.

First of all, I would like to thank my mentor, ASSOC. PROF. DR. MOHD. FAIZAL ABDOLLAH, for his valuable guidance and suggestions throughout the project, which were crucial to the smooth progress of the project. The mentor's deep insights into the research direction and strict requirements for details helped me to continuously improve and promoted the successful completion of the project.

Secondly, I would like to thank Kaggle.com for providing the N-BaIoT dataset, which provided a basis for the research and enabled the project to test and verify the project's model in real scenarios.

Finally, I would like to thank my family for their understanding and support, which allowed me to focus on my research work and gave me encouragement and help when I encountered difficulties.

Here, I would like to express my sincere gratitude to all those who supported and helped me. Without your help and support, this project would not have been completed so smoothly.

FONG ZI KHANG

## ABSTRACT

With the rapid development of Internet of Things (IoT) technology, more and more devices are connected to each other through the Internet, forming a large and vulnerable network system. The widespread application and connectivity of these devices enhance the risk of Botnet attacks. This paper aims to develop an intelligent detection system to detect Botnet activities in the Internet of Things through artificial intelligence algorithms to improve detection efficiency and accuracy. Through this detection system, potential network attacks can be discovered and blocked in a timely manner, effectively protecting the security of IoT devices and data. At the same time, this will also promote the development of Botnet detection technology and improve the security and stability of the network. The research also includes the analysis and comparison of different features to optimize the feature selection process, and by evaluating and comparing multiple machine learning classifiers, determine the most effective classifier in the home IoT environment. These contributions not only improve the overall performance of the Botnet detection system, but also provide valuable data and empirical basis for future research.

اونيورسيتي تيكنيكل مليسيا ملاك  
UNIVERSITI TEKNIKAL MALAYSIA MELAKA

## ABSTRAK

Dengan perkembangan pesat teknologi Internet of Things (IoT), semakin banyak peranti disambungkan antara satu sama lain melalui Internet, membentuk sistem rangkaian yang besar dan terdedah. Aplikasi dan ketersambungan meluas peranti ini meningkatkan risiko serangan Botnet. Kertas kerja ini bertujuan untuk membangunkan sistem pengesanan pintar untuk mengesan aktiviti Botnet dalam Internet Perkara melalui algoritma kecerdasan buatan untuk meningkatkan kecekapan dan ketepatan pengesanan. Melalui sistem pengesanan ini, kemungkinan serangan rangkaian boleh ditemui dan disekat tepat pada masanya, dengan berkesan melindungi keselamatan peranti dan data IoT. Pada masa yang sama, ini juga akan menggalakkan pembangunan teknologi pengesanan Botnet dan meningkatkan keselamatan dan kestabilan rangkaian. Penyelidikan ini juga termasuk analisis dan perbandingan ciri yang berbeza untuk mengoptimumkan proses pemilihan ciri, dan dengan menilai dan membandingkan berbilang pengelas pembelajaran mesin, tentukan pengelas yang paling berkesan dalam persekitaran IoT rumah. Sumbangan ini bukan sahaja meningkatkan prestasi keseluruhan sistem pengesanan Botnet, tetapi juga menyediakan data berharga dan asas empirikal untuk penyelidikan masa depan.

## Table of Contents

CHAPTER 01 : INTRODUCTION .....	1
1.1 Introduction .....	1
1.2 Problem Statement .....	2
1.3 Project Question .....	3
1.4 Project Objective .....	3
1.5 Project Scope .....	4
1.6 Project Contribution .....	5
1.7 Report organization .....	6
Chapter 1: Introduction .....	6
Chapter 2: Literature Review .....	6
Chapter 3: Methodology .....	6
Chapter 4: Design .....	6
Chapter 5: Implementation .....	6
Chapter 6: Testing .....	6
Chapter 7: Conclusion .....	6
1.8 Conclusion .....	7
CHAPTER 02 : LITERATURE REVIEW .....	8
2.1 Introduction .....	8
2.2 Related Work/Previous Work .....	8
2.2.1 Internet of Things(IoT) .....	8
2.2.1.1 What is Internet of Things(IoT) .....	8
2.2.1.2 IoT security and threats .....	13
2.2.2 Botnet .....	15
2.2.2.1 What is Botnet? .....	15
2.2.2.2 Botnet Behaviour in Internet of Things(IoT) .....	19
2.2.2.3 Mirai Botnet Behaviour in Internet of Things(IoT) .....	20
2.2.2.4 Bashlite Botnet Behaviour in Internet of Things(IoT) .....	22
2.2.3 Machine Learning Technique .....	24
2.2.4 Types of Dataset .....	25
2.2.5 Features Selection Methods .....	26
2.2.5.1 Lasso Feature Selection Method .....	28
2.2.5.2 Random Projection Feature Selection Method .....	28
2.2.5.3 Chi-Square Filter Feature Selection Method .....	28

2.2.5.4 Recursive Feature Elimination (RFE) Feature Selection Method	28
2.2.6 Performance Parameters .....	29
2.3 Related Paper .....	31
2.4 Critical Review .....	33
2.5 Proposed Solution .....	34
2.6 Conclusion .....	35
CHAPTER 03 : PROJECT METHODOLOGY .....	36
3.1 Introduction .....	36
3.2 Dataset Used .....	36
3.3 Methodology .....	38
3.4 Project Milestones .....	40
3.5 Gantt Chart .....	42
3.6 Conclusion .....	43
CHAPTER 04 : ANALYSIS AND DESIGN .....	44
4.1 Introduction .....	44
4.2 Data Structure .....	44
4.3 System Flowchart .....	46
4.3.1 Data Collection .....	47
4.3.2 Data Preprocessing .....	49
4.3.3 Feature Selection .....	50
4.3.4 Training and Testing Machine Learning Classifiers .....	53
4.3.5 Performance Evaluation .....	55
4.4 Conclusion .....	56
CHAPTER 05 : ANALYSIS AND DESIGN .....	57
5.1 Introduction .....	57
5.2 Botnet Detection System Configuration Management .....	57
5.2.1 Dataset Collection .....	57
5.2.2 Main Coding .....	61
5.2.3 Data Preprocessing .....	67
5.2.4 Feature Selection .....	69
5.2.4.1 Lasso .....	69
5.2.4.2 Random Projection .....	71
5.2.4.3 Chi Square .....	72
5.2.4.4 Recursive Feature Elimination (RFE) .....	74

5.2.4.5 Hybrid Feature Selection Method .....	76
5.2.4.6 Machine Learning Classifier and Performance Evaluation .....	79
5.3 Conclusion .....	82
CHAPTER 06 : TESTING .....	83
6.1 Introduction .....	83
6.2 TOP Feature Selection Method and Machine Learning Classifiers Model ....	83
6.3 TOP Feature .....	86
6.4 Comparison with the existing works .....	87
6.5 Conclusion .....	89
CHAPTER 07 : CONCLUSION .....	90
7.1 Introduction .....	90
7.2 Project Achievement .....	90
7.2.1 Best Feature Selection Method .....	91
7.2.2 Develop and Implement Machine Learning Classifiers .....	93
7.2.3 Best Machine Learning Classifiers .....	94
7.3 Future Work .....	95
7.4 Conclusion .....	95
Reference .....	97
Appendix(dataPrepare.py) .....	105
Appendix(main.py) .....	107
Appendix(Full Result) .....	116
Appendix(Feature Selected) .....	124



## CHAPTER 01 : INTRODUCTION

### 1.1 Introduction

With the rapid development and popularization of Internet of Things (IoT) technology, various devices in people's lives have begun to connect to each other through the Internet, forming a large and fragile network system. From smart phones to home appliances, to cameras, smart furniture, etc., the network connections of these devices have brought convenience to people's lives, but they have also brought new security risks(Kumar et al., 2019).

In this context, IoT devices face major security threats, one of which is Botnet intrusion. A Botnet is a network composed of a large number of infected devices that attackers can control to launch various network attacks, the most common of which is distributed denial of service (DDoS) attacks(Nicholson, 2022).

The popularity and increased connectivity of IoT devices have led to an increasing risk of Botnet attacks in the IoT. What is particularly noteworthy is that the covert nature of this attack is even more prominent in the IoT environment. The robot's disguised behaviour patterns and communication methods make it invisible in IoT devices, especially common household devices like cameras. People often install cameras for home security, and cameras connected to the Internet have become targets. Robots often communicate with their controllers through covert communication channels, which makes detecting robot activities more difficult because they do not cause obvious alarms or unusual behaviour(Cut Alna Fadhilla et al., 2023).

Therefore, timely detection and response to Botnet activity becomes critical. Network traffic monitoring has become an important means to solve this problem. By monitoring the communication traffic between devices, abnormal behaviours can be found and potential Botnet infections can be identified, such as abnormal DNS data, access to unfamiliar or blacklisted areas Names, frequent communication attempts, or large amounts of unauthorized data access can all be signs of Botnet activity(Stevanovic & Pedersen, 2014).

Therefore, ensuring the security of IoT devices has become an urgent task. The goal of this project is to develop an intelligent detection system to detect Botnet activities in the Internet of Things, and to improve detection efficiency and accuracy through artificial intelligence algorithms. By combining the intelligent detection systems into firewall, potential network attacks can be discovered and blocked in time, effectively protecting the security of IoT devices and data. At the same time, this will also promote the progress and development of Botnet detection technology and improve the security and stability of the network.

## 1.2 Problem Statement

Although the development of IoT technology has brought convenience to our lives, with the widespread application and enhanced connectivity of IoT devices, the risk of Botnet attacks is also increasing. In this context, how to effectively detect and respond to Botnet activities in the Internet of Things has become an urgent problem to be solved. As the number of IoT devices continues to increase, their security issues have become increasingly prominent (Dange & Chatterjee, 2020). Especially in the Internet of Things (IoT) environment, the concealment problem of Botnet attacks is even more serious. This concealment mainly comes from the camouflage of the behaviour patterns and communication methods of robots (bots). Among IoT devices, robots may be hidden among a large number of home devices, especially cameras that can be connected to a local area network. For home security reasons, many people choose to install cameras at home, and IP cameras that can be connected to mobile phones are the first choice. When the camera is able to connect to the network, it will become a target for Botnet attacks. Bots often communicate with their controllers through covert communication channels, such as leverage the normal communication traffic of IoT devices or mimicking legitimate traffic. This hidden communication pattern makes detecting bot activity more difficult because they do not cause obvious alarms or unusual behaviour. Therefore, for Botnet attacks in IoT environments, it becomes more challenging to detect and identify malicious behaviours in a timely manner (Rajesh Kumar Yadav & Karamveer Karamveer, 2022).

**Table 1.1 Summary of Problem Statement**

PS	Problem Statement
PS1	In the IoT environment, the covert nature of Botnet attacks makes it difficult to detect by using single feature.

### 1.3 Project Question

In the context of the rapid development of Internet of Things (IoT) technology, ensuring the security of IoT devices has become an important task. Among them, how to effectively detect Botnet activities in IoT using machine learning and what feature can help to effectively detect Botnet activities in IoT. By answering these questions, important references and solutions can be provided for building a safer and more reliable IoT environment.

**Table 1.2 Summary of Problem Question**

PS	PQ	Problem Question
PS1	PQ1	How to effectively detect Botnet activities in IoT using Machine Learning?
	PQ2	What feature can help to effectively detect Botnet activities in IoT?

### 1.4 Project Objective

As this project is developed, a number of objectives are being discussed

**Table 1.3 Summary of Problem Objective**

PS	PQ	PO	Problem Objective
PS1	PQ1	PO1	Identify the best feature selection for Botnet detection in IoT network
	PQ2	PO2	To develop and implement machine learning classifiers for detecting Botnet in IoT environment
		PO3	To select the best classifiers with higher accuracy in detecting Botnet in IoT environment

## 1.5 Project Scope

The project scope will be conducted based on the project objectives to ensure the correct and organized flow of the project. The scope of the project include:

I. Dataset collection - collecting the data from the Botnet behaviour in IoT environment.

II. Data preprocessing - pre-process the dataset before executing the algorithm for missing values, noisy data, and other inconsistencies.

III. Feature selection - only choose the important features to enhance the accuracy of the system.

IV. Training and testing machine learning classifiers - Use a dataset to train the machine learning classifiers and test it using unseen data before to evaluating its performance.

V. Performance evaluation - Validate the performance of machine learning classifiers based on accuracy, precision, recall, and f1 score to compare with existing literature.

VI. Home IoT environment - Analysis and dataset is based on Home IoT environment

## 1.6 Project Contribution

This project has made important contributions to botnet detection in home IoT environments in many ways. First, through in-depth research and experiments, the ability to detect botnets in home IoT environments has been improved. Second, by analyzing and comparing different features, this project identified the best features for detecting botnets in home IoT environments, thereby optimizing the feature selection process and improving the accuracy and efficiency of detection. Finally, by evaluating and comparing multiple machine learning classifiers, this project identified the most effective machine learning classifier in home IoT environments. These contributions not only improve the overall performance of botnet detection systems, but also provide valuable data and empirical foundations for future research.

**Table 1.4 Summary of Problem Contribution**

PS	PQ	PO	PC	Problem Contribution
PS1	PQ1	PO1	PC1	Improve Botnet detection in Home IoT environment
	PQ2	PO2	PC2	Identify best feature to detect Botnet in Home IoT environment
		PO3	PC3	Identify best Machine Learning classified in Home IoT environment

## **1.7 Report organization**

### **Chapter 1: Introduction**

This chapter introduces the initial part of the project, discussing the problem statement, questions, and objectives. Additionally, the scope and contribution of the project are covered. Briefly, this chapter will briefly discuss the background of the project.

### **Chapter 2: Literature Review**

This chapter describes the project in more detail, with supporting material drawn from past projects and research papers.

### **Chapter 3: Methodology**

This chapter describes the project methodology and how it was implemented and executed. It also describes how to prioritize projects through to completion.

### **Chapter 4: Design**

This chapter defines the user interface, requirements, and system design of the project.

### **Chapter 5: Implementation**

This chapter discusses the processes, activities, and desired outputs involved in software implementation.

### **Chapter 6: Testing**

This chapter describes the activities involved in the software testing phase, the project outcomes, and the performance of the Botnet detection system.

### **Chapter 7: Conclusion**

This chapter will summarize and summarize the entire project, along with the advantages and limitations involved. Future improvements and project contributions will also be outlined in this chapter.

## 1.8 Conclusion

With the continuous development and widespread application of Internet of Things technology, people have ushered in the convenience of life, but also face the challenge of security risks, especially the threat from Botnet attacks. In this context, this project aims to solve the problem of how to effectively detect and respond to Botnet activities in the Internet of Things, and proposes the goal of developing intelligent detect tools and new algorithms. By deploying detect tools systems and new algorithms to analyse and identify abnormal network traffic patterns, potential network attacks can be discovered and blocked in time, and the security of IoT devices can be protected. In addition, this project will promote the advancement of Botnet detection technology and improve network security and stability.

Setting the project scope helps ensure that the project process is correct and organized. Deploying an detect tools system in an IoT network can effectively monitor network traffic and detect abnormal activities, thereby improving the overall security of the IoT. At the same time, by developing new algorithms based on artificial intelligence, Botnet activities in the Internet of Things can be identified more accurately and efficiently, and threats can be detected and responded to in a timely manner. This will provide important references and solutions for building a safer and more reliable IoT environment and promote the healthy development of IoT technology.

## CHAPTER 02 : LITERATURE REVIEW

### 2.1 Introduction

The rapid development of Internet of Things (IoT) technologies has brought tremendous benefits and convenience to various fields such as healthcare, transportation, and smart homes. However, with the popularity of IoT devices, security vulnerabilities and threats have also increased accordingly. IoT devices often lack strong security measures and are a prime target for malicious actors to exploit these weaknesses. This chapter reviews the existing literature on intelligent detection of IoT malware, focusing on the nature of IoT devices, the machine learning techniques they use, feature selection techniques, and the datasets used. This review will cover the characteristics of IoT systems, common attack vectors, and the application of machine learning techniques in detecting IoT botnet activities.

### 2.2 Related Work/Previous Work

#### 2.2.1 Internet of Things(IoT)

##### 2.2.1.1 What is Internet of Things(IoT)

The Internet of Things (IoT) refers to a network of physical devices connected through the Internet. These devices can be anything from household appliances to industrial machinery to wearable. They are all capable of collecting and transmitting data and can be controlled via the Internet(IBM, 2024).

The concept of the Internet of Things can be traced back to the 1990s, when Kevin Ashton of MIT first proposed the concept of "Internet of Things". He believes that with the popularization of radio frequency identification (RFID) technology, more and more items will be connected to the Internet(Foote, 2022).

Bandyopadhyay & Sen(2011) pointed out that the Internet of Things (IoT) is a rapidly developing technology, and its definition and understanding are also constantly evolving. Nonetheless, IoT systems often have the following important characteristics:



- i. Interconnectivity
  - a) One of the core features of the Internet of Things is the interconnectedness of objects or devices. Whether they are sensors, home appliances, cars, or industrial machines, all devices are able to connect, communicate, and collaborate with each other through the Internet.
  
- ii. Data Collection and Transmission
  - a) IoT devices collect a large amount of environmental data or operational data through sensors and transmit it to the data processing center through the network. The collection and transmission of these data are the basis for the normal operation of the IoT system.
  
- iii. Data Processing and Analysis
  - a) The data collected needs to be processed and analyzed to extract valuable information. This process is usually performed on cloud or edge computing devices and may involve big data analysis, machine learning and other technologies.
  
- iv. Real-time Capability
  - a) IoT systems often require real-time response and processing capabilities to respond to environmental changes or emergencies in a timely manner. This feature is particularly important in applications such as intelligent transportation and telemedicine.
  
- v. Intelligence
  - a) Through data analysis and artificial intelligence technology, IoT systems can achieve intelligent decision-making and control. For example, smart home systems can automatically adjust the operating status of home appliances based on user behavior patterns.
  
- vi. Scalability
  - a) IoT systems need to be scalable to cope with the increasing number of devices and data volumes. This requires system architecture design to support large-scale device access and data processing.

vii. Security and Privacy

- a) Since IoT devices and systems handle large amounts of sensitive data, security and privacy protection are crucial features. Strong measures need to be taken to prevent data leakage, tampering and unauthorized access.

viii. Heterogeneity

- a) The Internet of Things encompasses various types of devices and communication protocols and requires the ability to handle interoperability issues between different devices and systems to achieve seamless connectivity and data sharing.

ix. Dynamic Nature

- a) The status and quantity of IoT devices and systems often change dynamically, and it is necessary to have the ability to adapt to these changes, including the addition and removal of devices, movement, status changes, etc.

x. Ecosystem Perspective

- a) The Internet of Things is not just a simple connection of devices, but a complex ecosystem including devices, networks, platforms, data and related stakeholders. Understanding and managing this ecosystem is critical to IoT success.

Since the 21st century, with the rapid development of Internet technology, the Internet of Things has also developed rapidly. In 2015, the International Telecommunications Union (ITU) released the "Global Standard for the Internet of Things", laying the foundation for the development of the Internet of Things (ITU, 2019). In 2015, the Chinese government released the "Internet +" Action Plan, which listed the Internet of Things as a key development area (China Government Network, n.d.).

Lionel Sujay Vailshery (2023) predicts that the number of global Internet of Things (IoT) devices will almost double by 2030, increasing from 15.1 billion units in 2020 to 29 billion units. IoT devices are used in all types of vertical industries and in the consumer market, which accounted for approximately 60% of all IoT connected

devices in 2020. This ratio is expected to remain at this level over the next decade. Key industry verticals with more than 100 million connected IoT devices currently include electricity, gas, steam and air conditioning, water supply and waste management, retail and wholesale, transportation and warehousing, and government. Overall, the number of IoT devices across all industry verticals is expected to grow to more than 8 billion by 2030. The most important use cases for IoT devices in the consumer sector are consumer internet and media devices such as smartphones, with the number of IoT devices expected to grow to more than 17 billion by 2030. Other use cases expected to exceed 1 billion IoT devices by 2030 include connected (self-driving) cars, IT infrastructure, asset tracking and monitoring, and smart grids(Vailshery, 2023).

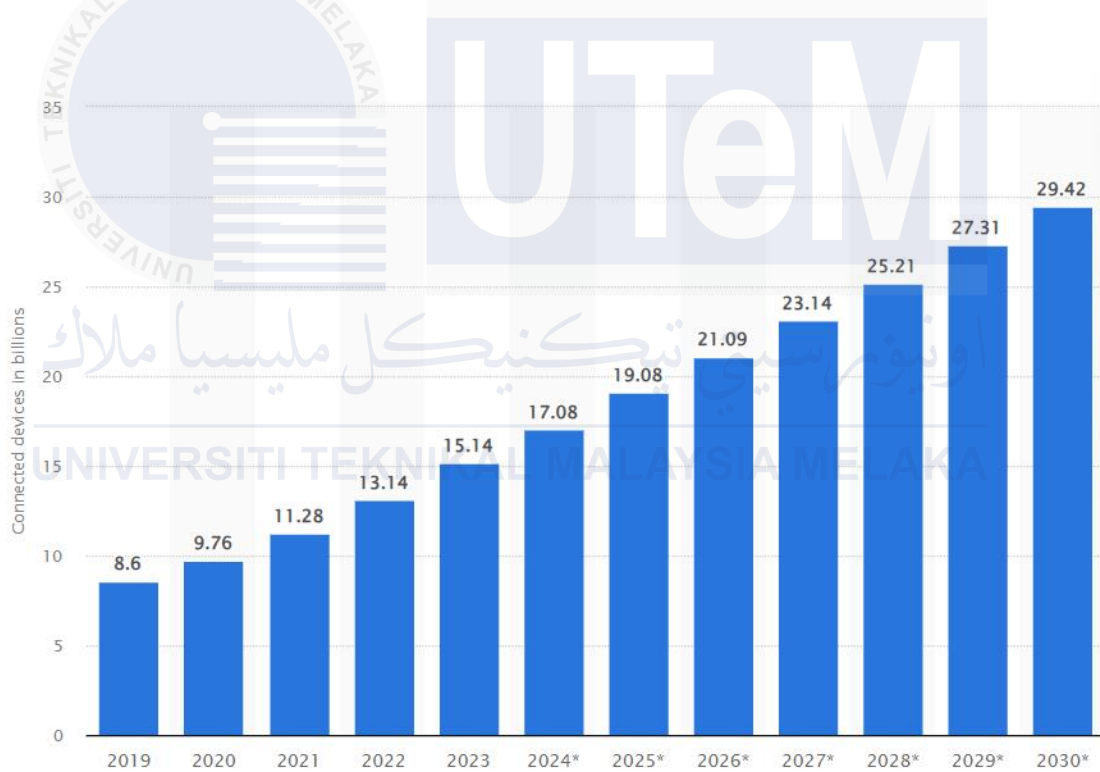


Figure 2.1: Statistic reports the number of IoT devices from 2019 to 2030 (Vailshery, 2023)

IoT has a wide range of applications, including smart homes, smart cities, industrial IoT, and healthcare. Smart home devices can automatically adjust temperature, lighting, and security systems; smart cities can use IoT sensors to monitor traffic, environment, and public safety; the industrial IoT can improve

production efficiency and reduce costs; and the IoT can be used to monitor patient health conditions and provide remote medicine services(Gillis, 2023).

The architecture of the Internet of Things is usually divided into four layers, namely the perception layer, network layer, data processing layer and application layer. This structure helps to understand and implement the various components of the IoT system and their functions. The following is a detailed explanation of this four-layer structure:

- i. Perception Layer(Gubbi et al., 2013)
  - a. The perception layer is the lowest layer of the IoT system and is responsible for the interface between the physical world and the digital world.
  - b. This layer mainly includes various sensors and actuators for data collection and equipment control.
  - c. Sensors collect environmental data (such as temperature, humidity, light intensity, pressure, etc.) and convert these data into digital signals for transmission to the upper layer.
  - d. For example: temperature sensor, humidity sensor, light sensor, RFID tag, barcode scanner, etc.
- ii. Network Layer(Xu et al., 2014)
  - a. The network layer is responsible for data transmission and is the bridge between the perception layer and the data processing layer.
  - b. It includes various communication technologies and protocols, such as Wi-Fi, Bluetooth, ZigBee, cellular networks (such as 4G/5G), etc., which are used to transmit data collected by the perception layer to the data processing layer.
  - c. This layer also includes gateway devices that play an important role in connecting local sensor networks to the larger Internet.
  - d. For example: Wi-Fi router, Bluetooth module, ZigBee device, cellular gateway, etc.
- iii. Data Processing Layer(Atzori et al., 2010)
  - a. The data processing layer is responsible for processing and storing data transmitted by the network layer.

- b. It usually includes cloud computing platforms and data centers for large-scale data storage, processing and analysis.
  - c. This layer may also use edge computing devices to process data close to the data source to reduce latency and increase processing efficiency.
  - d. For example: cloud servers, database systems, edge computing devices, data analysis platforms, etc.
- iv. Application Layer(Miorandi et al., 2012)
- a. The application layer is the top layer of the IoT system, directly facing users and applications.
  - b. This layer provides various applications and services to make decisions and control based on the data analyzed by the processing layer.
  - c. The application layer covers a wide range of IoT application fields, such as smart home, smart transportation, smart medical care, industrial automation, etc.
  - d. For example: smart home control system, telemedicine system, smart city management platform, Industrial Internet of Things (IIoT) applications, etc.

The key technologies of the Internet of Things include sensor technology, communication technology, data processing technology and security technology. Sensor technology is used to collect data, communication technology is used to transmit data from the perception layer to the application layer, data processing technology is used to process data and extract valuable information, and security technology is used to protect the security of IoT devices and networks(Kumar et al., 2019).

#### **2.2.1.2 IoT security and threats**

While the Internet of Things (IoT) brings great convenience and innovation, it also faces many security challenges and threats. The following is a detailed discussion of IoT security and threats, including attack sources and common attack types.

IoT devices usually collect and transmit large amounts of personal data and sensitive information, which, if accessed without authorization or leaked, will cause serious privacy and security issues. Khalid & Jhanjhi(2020) discussed four aspects of

IoT security: data integrity, data sharing, authentication and access control, and privacy protection.

- I. Data integrity: Ensure that the data generated by the IoT system is not modified without authorization and maintain the accuracy and reliability of the data.
- II. Data sharing: Securely exchange data between devices, ensuring the confidentiality and integrity of data during transmission.
- III. Authentication and access control: Ensure that only authorized users and devices can access system resources.
- IV. Privacy protection: Protect data from illegal collection and use, and prevent data leaks and privacy violations.

The sources of IoT attacks can be divided into devices, communication channels and application software. Ivan Lee(2024) pointed out the following sources of attacks:

- i. Device: including the device's memory, firmware, network interface and physical interface, etc. Attackers can exploit vulnerabilities in a device, such as outdated components or unpatched vulnerabilities.
- ii. Communication channels: Insecure communication channels can become the target of attacks, and attackers can attack by eavesdropping or tampering with communication data.
- iii. Application software: Application software connected to IoT devices also presents security risks. An attacker can gain access to a device or system by exploiting vulnerabilities in application software.

Common attack types on IoT devices include the following:

- I. Brute-force Password Attack: The attacker continuously tries various password combinations to gain access to the system.
- II. Distributed Denial of Service Attack (DDoS): Attackers use infected IoT devices to form a Botnet and initiate a large number of requests to the target system, causing the system to paralyze. Ivan Lee(2024) pointed out that this type of attack has become increasingly common and easy to carry out.
- III. Man-in-the-middle attack (MitM): An attacker steals or tamper with the data transmitted between the device and the server by hijacking the communication channel.

IV. Privilege Escalation Attack: An attacker exploits vulnerabilities in the system or device to elevate his or her privileges, thereby gaining complete control over the device or system.

Zscaler ThreatLabz 2023 Enterprise IoT and OT Threat Report show the number of malware attacks on IoT devices surged 400 percent compared to the 2022. IoT botnet activity, a growing concern in the IoT space, continued to dominate, with the Mirai and Gafgyt malware families accounting for 66% of attack payloads.

## **2.2.2 Botnet**

### **2.2.2.1 What is Botnet?**

Botnet is a network composed of a large number of computers controlled by hackers. These computers are called "bots" and are often exploited by hackers to perform various malicious activities (Kaspersky, 2019). These activities include sending spam, launching distributed denial-of-service (DDoS) attacks, stealing sensitive data and spreading malware.

Bot networks pose a serious threat to Internet security. They can be used to launch large-scale attacks, causing huge economic losses and social impact. For example, in 2016, the "Mirai" robot network launched an attack on the US Internet infrastructure, causing many websites and services to be paralysed (Cloudflare, 2023).

Typical Botnet creation and maintenance can be divided into four phases. First is the initial infection stage, where computers can be infected in a variety of ways, such as through actively exploited vulnerabilities, automated downloads of malware, or execution by opening email attachments. Next comes the secondary injection phase, where the infected host downloads and runs the bot code, becoming a true bot. This is followed by the malicious activity phase, where the robot communicates with the controller and receives commands to perform activities such as spam, DDoS and scanning. Command communication can use IRC, HTTP, DNS or P2P protocols. Finally, there is the maintenance and upgrade phase, where the robot continuously upgrades its binaries. Botnets are classified based on their command and control

architecture, for example, those that use the IRC protocol are called IRC-based Botnets(Zhu et al., 2008).

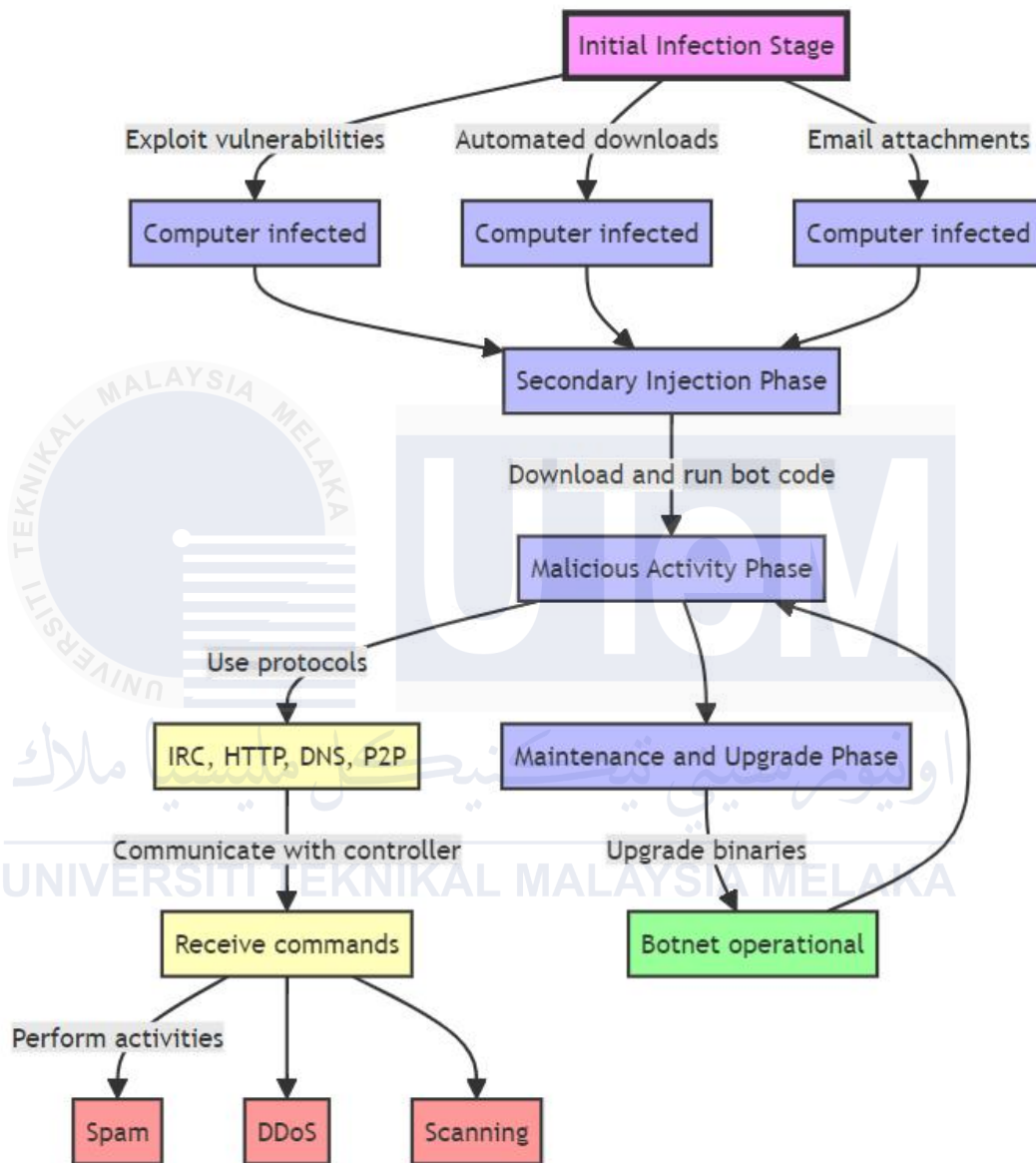


Figure 2.2: Diagram showing general Botnet behaviour

With the rapid growth in the number of IoT devices, robotic networks are paying more and more attention to IoT devices. IoT devices often lack security protections and can be easily hacked and incorporated into bot networks. Once IoT devices are infected, hackers can use them to perform a variety of malicious activities, such as launching DDoS attacks, stealing sensitive data, and damaging critical infrastructure(Tay, 2023).



IoT Botnet Distributed Denial of Service(DDoS) traffic grew fivefold between 2022 and 2023, driven by a rise in for-profit hacking groups run by cybercriminals, since Russia's invasion of Ukraine, Nokia's latest threat intelligence report found today. This traffic originates from large numbers of unsecured IoT devices with the goal of disrupting telecom network services for millions of users. This dramatic growth, coupled with increased consumer use of IoT devices around the world, first emerged at the start of the Russia-Ukraine conflict but has since spread to other parts of the world, with Botnet driven DDoS attacks being used to disrupt telecom networks and other critical infrastructure and services. The number of IoT devices (bots) involved in Botnet driven DDoS attacks has increased from around 200,000 in 2022 to around 1 million, generating more than 40% of all DDoS traffic today (Nokia Threat Intelligence Report Finds Malicious IoT Botnet Activity Has Sharply Increased | Nokia, 2023).

Using Slapper as an example to describe the process of P2P bot host infection. Slapper's propagation method is similar to worm, but its operating principle is Botnet. First, scans hosts on the network which using GET request packets to scan vulnerable hosts, hoping to obtain the fingerprint of the host (operating system version, web server version). Second, launch an attack on the detected vulnerable host, causing the SSL buffer area of the attacked host to overflow. The victim host leaks the heap pointer to the attacker. Third, launch a secondary attack on the vulnerable host. Interestingly, this bot uses two buffer overflows instead of one. First time, locate the location of the heap in the Apache process address space. The second time, inject its attack cache and shell code(Arce & Levy, 2003).

There are two good reasons for dividing the attack into two phases which is the attack cache must include the absolute address of the shell code, which is difficult to predict. Because when the shell code is placed in memory, it is dynamically allocated in the heap. In order to solve this problem, the bot first lets the server leak the address where the shell code is finally allocated. Then the attack cache is patched according to the address. Besides that, the attack will requires overwriting the password field behind the key\_arg[] cache in the password SSL\_SESSION data structure. The following is a more detailed introduction to these two stages. The first stage, when establishing an SSL v2 connection, an excessively large parameter is

deliberately placed, causing a buffer overflow. Open a connection to this port and initiate an SSLv2 handshake. For example, sends a "hello" message to the client, announces 8 different passwords (although, this bot only supports one of them) and receives the server-side authentication message. Then send master key and key parameters to the client, especially one key parameter that only allows a maximum of 8 bytes. When the packet data is parsed in the `get_client_master_key()` function in the `libssl` library, the code does no bounds checking on the parameters and copies the parameters to a fixed-length buffer in a heap-located `SSL_SESSION` data structure. In this way, bytes of any size can overwrite the information after `key_arg[]`, which can make the `SSL_SESSION` structure memory-blocking. Hand-crafted fields are the key to buffer overflows. Vulnerability explorers carefully cover these data fields without seriously affecting the SSL handshake. The second stage, secondary use of buffer overflow, executing shell code is divided into three steps. First, disrupt heap management data. Second, abuse the `free()` library call to add arbitrary code to the memory as an entry to `free()`'s own global offset table. Third, since `free()` is called a second time, this time the control is redirected to the location of the shell code. The attack cache used in the second overflow contains three parts which is the cache behind `key_arg[]` of the `SSL_SESSION` data structure, 24 bytes of carefully constructed data and 124 bytes shell code. When a buffer overflow occurs, all members of the `SSL_SESSION` data structure behind the `key_arg[]` cache are overwritten. Except for the password field, the numeric fields are filled with "A" bytes, and the pointer fields are made empty. This field is restored to the original leaked value.

When the shell code is executed, it first looks for the socket for the TCP connection to the attacking host. This process is performed by traversing all file descriptors and calling `getpeername()` on each one until the TCP port matching the shell code is successfully found. The shell code then copies the socket descriptor to standard input, output, and error outlets. Next it attempts to gain root privileges by calling `setresuid()` with a UID set to all zeros. Apache is usually started as root and then switched to the unprivileged "Apache" identity using the `setuid()` function. In this case, the `setresuid()` call will fail, because `setuid()`, contrary to `seteuid()`, is irreversible. The creators of the shell code are fully aware of this fact, and the worm does not require permissions to spread because it only needs to write to the `/tmp` folder. Finally,

a standard shell "/bin/sh" is executed with the `execve()` system call. The worm initiates a series of shell commands to upload itself to the server in UU-encoded form, and then decodes, compiles, and executes itself. This recompilation of source code on different platforms makes it more difficult to identify binary worm signatures. These operations are performed in the /tmp folder, and the worm files are named something like .uubugtraq, .bugtraq.c and .bugtraq. Note that the . before the file name is to hide the file from being discovered by the simple `ls` command(Citronnelle2, 2015).

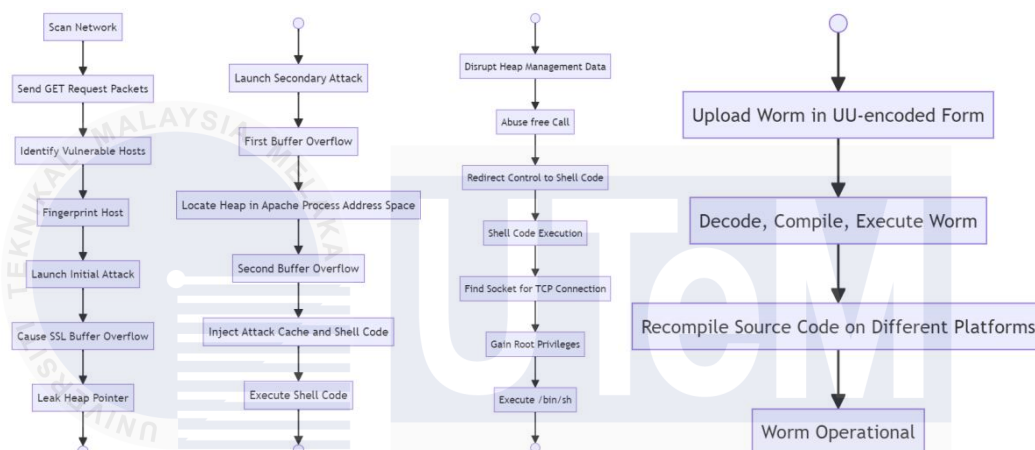


Figure 2.3 : Diagram for Slapper behaviour

### 2.2.2.2 Botnet Behaviour in Internet of Things(IoT)

An IoT Botnet consists of infected devices (called bots) that run malicious code under the command and control (C&C) of the Botnet controller (Bertino & Islam, 2017). According to Alzahrani et al.(2020), these bots can spread throughout a network by scanning the network range and exploiting known vulnerabilities or weak credentials of the devices. After invading an unprotected device, the bot will embed itself in the device and wait for instructions from the Botnet controller to perform various malicious activities, including but not limited to:

- i. DDoS attack: Sending a large number of illegal requests to the target device, causing the device to be unable to process legitimate requests, thereby causing a distributed denial of service (DDoS) attack.
- ii. Cryptocurrency mining: Using the computing power of the device to mine cryptocurrency.
- iii. Password cracking: Trying to crack the password of the target system.

- iv. Sending spam: Sending large amounts of spam through infected devices.
- v. Key logging: Recording the keystrokes of the device user to steal sensitive information.

The first IoT Botnet, Linux.Hydra, was discovered in 2008 (Kishore Angrishi, 2017). However, it was not until the emergence of the Mirai Botnet in 2016 that the security community truly realized the seriousness of IoT Botnets (Kolias et al., 2017). In September 2016, a Mirai attack targeting the Krebs on Security blog generated 620 Gbps of traffic. The release of Mirai's original source code led to the development of dozens of variants and inspired the creation of many other Botnets. For example, a Mirai variant compromised the service provider Dyn, resulting in one of the largest DDoS attacks in history.

IoT Botnets can attack in a variety of ways, and here are some common attack sources and attack types:

- I. Exploiting device vulnerabilities: Intrusion by scanning the network and exploiting known vulnerabilities in the device.
- II. Weak credential attacks: Gaining access to a device by guessing or cracking a weak password.
- III. Reflection attacks: Using IoT devices as reflectors for DDoS attacks, making the attack difficult to track (Zheng & Yang, 2019).

### **2.2.2.3 Mirai Botnet Behaviour in Internet of Things(IoT)**

Using a IoT Botnet which is Mirai as a example. Mirai targets Linux-based IoT devices and creates a network of robots controlled by a command and control (C&C) server. The attack is divided into two steps.

First, during discovery/infection phase, Mirai probes random blocks of IP addresses for possible telnet connections. Once a potential victim is identified, Mirai begins brute-force login attempts via a factory-set list of users and passwords. The device using default settings will be identified and its IP address, along with successful credentials, will be sent back to the C&C server(Tanaka & Yamaguchi, 2017).

Second, once a device is infected and executes a malware binary, it is added to the list of active bots to receive C&C commands and scanned to detect new victims. It is worth mentioning that the malware binary is removed from the system along with other competing processes, including other Mirai variants(Mirai Botnet Malware & Its Impact on the IoT, n.d.).

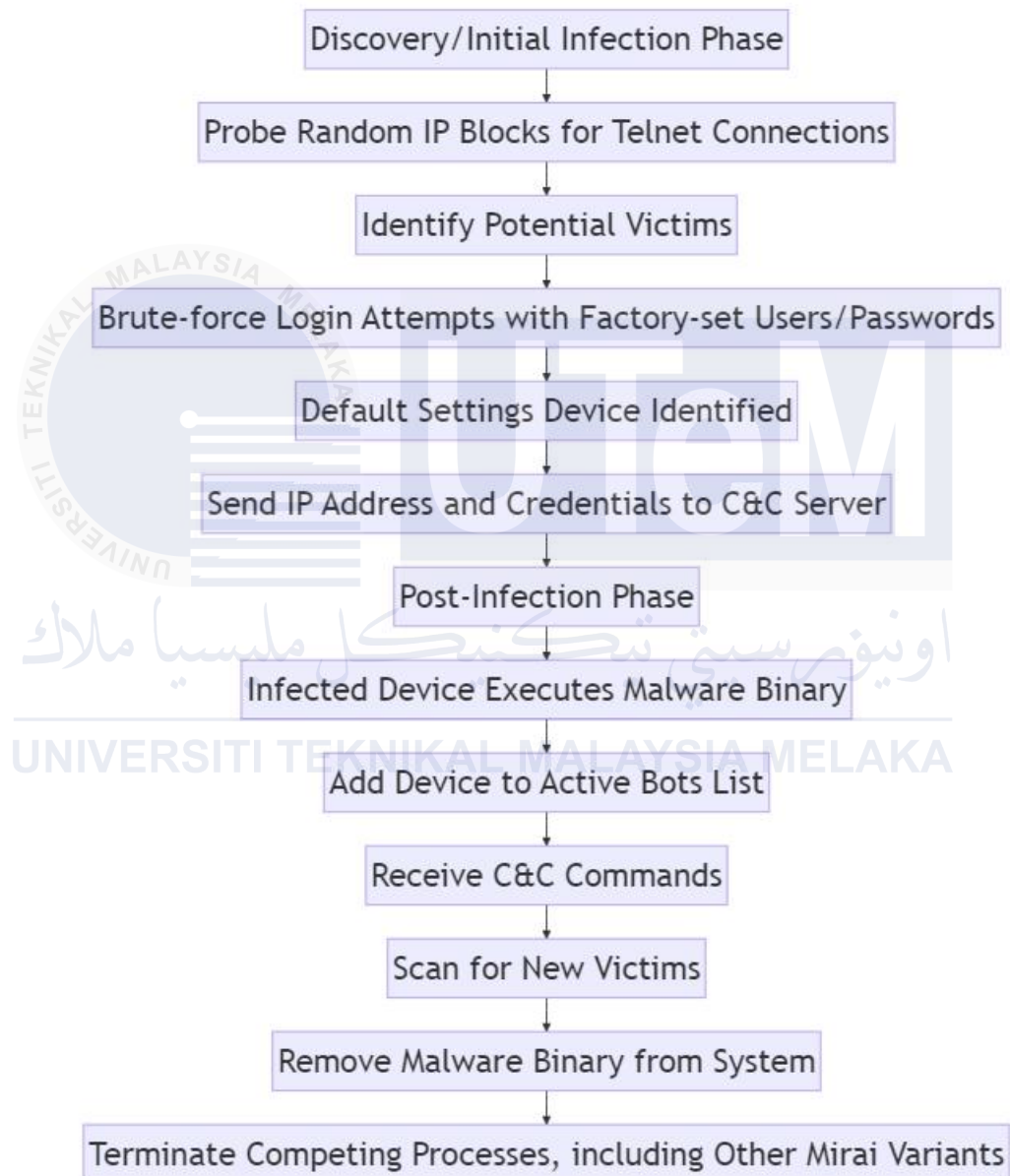


Figure 2.4 : Diagram for Mirai behaviour

#### 2.2.2.4 Bashlite Botnet Behaviour in Internet of Things(IoT)

Using Bashlite, also known as Gafgyt is a botnet affecting Internet of Things (IoT) devices and Linux-based systems. The malware aims to compromise and gain control of these devices, often by exploiting weak or default passwords, as well as known vulnerabilities. Gafgyt has been around since 2014 and has evolved into multiple variants, each with its own set of features and capabilities, including the ability to launch distributed denial of service (DDoS) attacks (Stanislav Gayvoronsky, 2024).

Bashlite is an Internet of Things (IoT) malware written in C that primarily infects devices through brute force and exploitation. Its behaviors include:

- i. Brute Force: Bashlite uses a built-in list of default Telnet and SSH credentials to attempt to brute force the device.
- ii. Exploit: Bashlite also exploits known vulnerabilities in the device firmware. For example:
  - a) CVE-2017-18368: This vulnerability exists in the remote system log forwarding function of Zyxel routers. Bashlite can exploit this vulnerability through the lack of proper input validation.
  - b) CVE-2023-1389: This vulnerability affects TP-Link Archer devices. Bashlite is able to execute unauthorized malicious commands in the national form of the web management interface.
- iii. Infected Devices: Once the device is infected, Bashlite downloads and executes a script from a preconfigured address to collect the device's IP address and system information.
- iv. Connecting to C2 Server: After infecting a device, Bashlite connects to its command and control server (C2) and receives instructions from the C2, such as launching a flood attack on a specified target.

Some versions of Bashlite have a persistence mechanism that allows them to continue running even if the device is restarted. Bashlite also has the ability to self-propagate, scanning the Internet for devices with open ports and trying to access them using default credentials (Marzano et al., 2018).

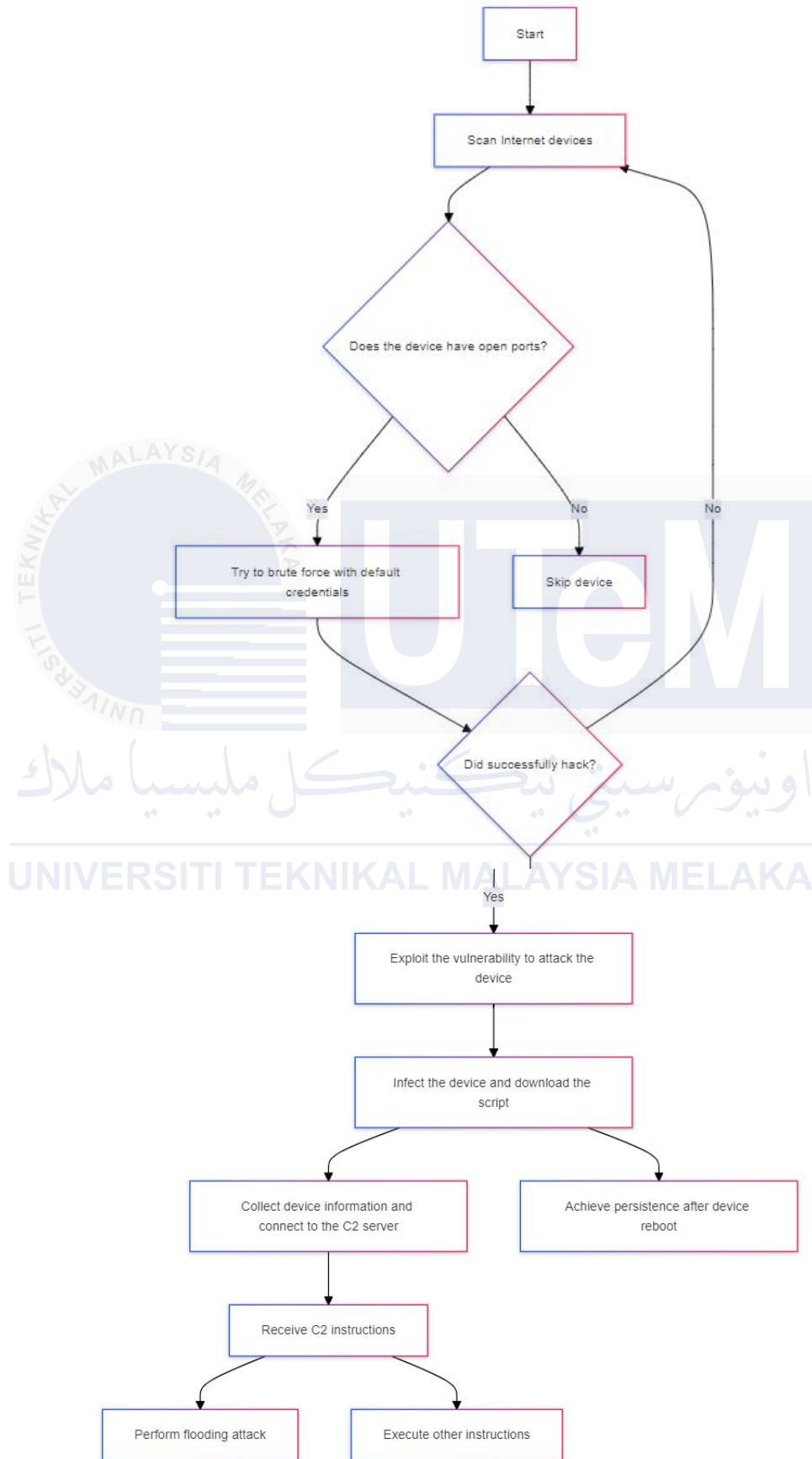


Figure 2.5: Diagram for Bashlite behaviour

### 2.2.3 Machine Learning Technique

In the context of detecting Botnets in IoT environments, machine learning has important application value and broad prospects. Xiao et al., 2018 discussed the importance of machine learning in IoT and introduced a variety of machine learning methods. The following is an introduction to and application of these techniques:

#### i. Supervised Learning

a. Supervised learning is a machine learning method that uses labeled data to train an algorithm. During the training process, the algorithm obtains input data and its associated correct output labels, with the goal of teaching the algorithm to accurately predict its labels when encountering new data. Common supervised learning algorithms include:

- i. Naive Bayes
- ii. Random Forest
- iii. Decision Tree
- iv. Support Vector Machine (SVM)

b. These techniques are widely used for time series prediction, regression, and classification tasks. Supervised learning has applications in many fields, such as generating predictions and extracting valuable information from data (Gupta et al., 2022).

#### ii. Unsupervised Learning

a. Unsupervised learning is a machine learning method that analyzes unlabeled data, in which the algorithm has no predefined output labels. Its goal is to discover patterns, relationships, or structures in the data. Unsupervised learning algorithms work independently to discover hidden insights and cluster similar data points together. Common unsupervised learning techniques include:

- i. Clustering algorithms (such as K-means, hierarchical clustering)
- ii. Dimensionality reduction methods (such as PCA and t-SNE)

b. These techniques are used in areas such as data mining, customer segmentation, and market analysis (Amruthnath & Gupta, 2018).



### iii. Semi-supervised Learning

a. Semi-supervised learning is a hybrid machine learning technique that combines labeled and unlabeled data for training. By using a large amount of unlabeled data and a small amount of labeled data, semi-supervised learning can improve the understanding and functionality of the model. The theoretical basis of this method is that unlabeled data provides context and additional information that helps enhance learning. Semi-supervised learning is very useful in practical applications, especially when labeled data is expensive or difficult to obtain (Yang et al., 2022).

### iv. Reinforcement Learning

a. Reinforcement learning is a machine learning algorithm that mimics the human learning process through trial and error. In this method, the agent interacts with the environment and learns by choosing behaviors that maximize the cumulative reward. Depending on the agent's behavior, it is rewarded or punished, providing feedback. Due to its dynamic learning approach, reinforcement learning is very effective in handling complex decision-making situations (Akanksha et al., 2021).

## 2.2.4 Types of Dataset

Datasets play a vital role in building botnet detection systems. These datasets provide rich training and testing data, allowing researchers to develop and evaluate the effects of various machine learning and deep learning algorithms (Importance of Datasets in Machine Learning and AI Research, n.d.).

According to Moustafa & Slay (2015), UNSW-NB15 dataset is a labeled dataset containing real network traffic, mainly used for network intrusion detection. The dataset contains 49 features and 2,540,444 records, including 175,341 records in the training set and 82,332 records in the test set. The data is divided into normal traffic and abnormal traffic (including nine different attack types), including backdoor, analysis, general, denial of service (DoS), vulnerability exploitation, reconnaissance, worm, fuzz testing, and ShellCode. This dataset is widely used to evaluate and verify the effectiveness of various machine learning methods in intrusion detection systems.

The N\_BaIoT dataset is captured from a testbed network and contains normal and attack traffic of nine IoT devices, including two doorbells, thermostats, baby monitors, four security cameras, and webcams. The dataset contains Mirai and Bashlite attack traffic, with a total of 115 features. The dataset is characterized by an unbalanced distribution of traffic and attack categories, and is often used to study malware classification and intrusion detection of IoT devices(Abbasi et al., 2021).

MohammadNoor Injadat et al.(2020) express Bot-IoT-2018 dataset is generated by designing a real IoT network environment, which contains three main components: network platform, simulated IoT service, and feature extraction platform. The entire dataset contains about 72 million records and 46 features. About 3.6 million records and 10 best features are usually used in research. The dataset is mainly used to detect botnet attacks on IoT devices and is often used to evaluate the effect of machine learning optimization frameworks.

The IoT-23 dataset focuses on detecting Mirai botnet attacks and contains traffic data from three IoT devices (smart LED lights, Amazon Echo smart assistants, and smart door locks). The dataset contains 19 features, and the data is divided into normal traffic and malicious traffic. This dataset is often used in conjunction with transfer learning methods to study the effectiveness of intrusion detection systems in IoT environments(Dutta et al., 2020).

### **2.2.5 Features Selection Methods**

Feature selection is primarily focused on removing non-informative or redundant predictors from the model. Some predictive modeling problems have a large number of variables that can slow the development and training of models and require a large amount of system memory. Additionally, the performance of some models can degrade when including input variables that are not relevant to the target variable(Kuhn & Johnson, 2018).

According to Benkessirat & Benblidia(2019), Feature selection is a key step in machine learning and data mining. It aims to extract the most representative and discriminative features from the raw data to improve the performance and training

efficiency of the model. Feature selection methods mainly include filtering, wrapping, embedding and combination methods.

RHNS Jayathissa & MWP Maduranga(2022), talk about filtering method selects by evaluating the statistical properties of each feature. The variance threshold method selects features with variance greater than a certain threshold and filters out features with too small variance because they have little impact on the classification results. The correlation coefficient rule calculates the correlation coefficient between each feature and the target variable and selects features with high correlation. Common methods include Pearson correlation coefficient and mutual information. The chi-square test is for classification problems. It uses the chi-square statistic to measure the independence between features and labels and selects features with higher chi-square values.

Yogesh Dhote et al.(2015) explain the wrapping method evaluates the effect of features by training models. Recursive feature elimination (RFE) uses a specific learning algorithm (such as linear regression, support vector machine) to train the model and recursively delete features according to their importance until a predetermined number is reached. Forward selection starts from an empty feature set and gradually adds features that improve the model performance the most. Backward elimination starts from the full feature set and gradually deletes features that have the least impact on model performance.

Embedding methods integrate the feature selection process directly into model training. Lasso regression achieves feature selection by introducing an L1 regularization term, which reduces the coefficients of some features to zero. Decision trees and random forests select features based on feature importance scores such as information gain and Gini coefficient(Yogesh Dhote et al., 2015).

MohammadNoor Injadat et al. (2020) explain Combination methods is combine multiple feature selection techniques to achieve better results. For example, use filtering methods to quickly filter out more important features, and then use wrapping methods to further refine the selection; or use filtering methods for preliminary screening, and then use embedding methods for final selection.

### **2.2.5.1 Lasso Feature Selection Method**

Lasso is a feature selection method based on linear regression. It achieves feature selection by adding L1 regularization (i.e., penalizing the absolute value of the coefficient) to the regression model, forcing some coefficients to approach zero. The goal of Lasso is to minimize the prediction error while minimizing the number of regression coefficients, so that the most important features can be automatically selected (Queen & Emrich, 2021).

### **2.2.5.2 Random Projection Feature Selection Method**

Random projection is a dimensionality reduction technique that achieves feature selection or dimensionality reduction by projecting high-dimensional data into a low-dimensional random subspace. This method uses a random matrix to linearly transform the original data so that the projected data still maintains the structural characteristics of the original data. Random projection follows the Johnson-Lindenstrauss Lemma, which ensures that the distances between data points are not distorted too much by random projection (Wang et al., 2019).

### **2.2.5.3 Chi-Square Filter Feature Selection Method**

Chi-square test is a method for evaluating the independence between features and target variables, especially for classification problems. It measures the degree of deviation of the distribution of features from the distribution of the target variable by calculating the chi-square statistic for each feature. The larger the chi-square value of a feature, the stronger the association between the feature and the target variable, and therefore the more valuable the feature is for classification (Dhalaria & Gandotra, 2020).

### **2.2.5.4 Recursive Feature Elimination (RFE) Feature Selection Method**

RFE is an iterative feature selection method that retains the most valuable features by continuously training the model, evaluating the importance of features, and recursively removing those that are not important. In each iteration, RFE trains the data using a specified learner (such as decision tree, linear regression, etc.) and eliminates the lowest-scoring features based on the feature importance score (Rukshani Puvanendran & Sharnidha Thangasundram, 2023).

### 2.2.6 Performance Parameters

According to Rashedun Nobil Chowdhury et al.(2020), in the detection system, performance parameters are used to evaluate the detection effect and classification performance of the model. The following are the definitions and explanations of commonly used performance parameters:

- I. True Positive (TP): The number of correctly detected positive samples, that is, the number of samples that are actually positive and correctly predicted as positive by the model.
- II. True Negative (TN): The number of correctly detected negative samples, that is, the number of samples that are actually negative and correctly predicted as negative by the model.
- III. False Positive (FP): The number of negative samples that are incorrectly detected as positive, that is, the number of samples that are actually negative but incorrectly predicted as positive by the model.
- IV. False Negative (FN): The number of positive samples that are incorrectly detected as negative, that is, the number of samples that are actually positive but incorrectly predicted as negative by the model.

Based on these basic parameters, the following key performance indicators can be calculated:

- i. Accuracy: Indicates the proportion of samples correctly predicted by the model to the total number of samples. The formula is:

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

Figure 2.6 : Accuracy Formula

- ii. Recall: Also known as sensitivity, it indicates the proportion of samples that are actually positive that are correctly predicted as positive. The formula is:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Figure 2.7 : Recall Formula

- iii. Precision: It indicates the proportion of samples that are actually positive that are predicted as positive. The formula is:

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

Figure 2.8 : Precision Formula

- iv. F-Measure: It is the harmonic mean of precision and recall, which is used to comprehensively evaluate the performance of the model. The common F-Measure is the F1 score. The formula is:

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Figure 2.9 : F1-Score Formula

These performance parameters can comprehensively evaluate the performance of the model in the detection system. In the research of IoT botnet detection system, these indicators are widely used to evaluate the effects of different machine learning algorithms. For example, Garg et al. (2021) compared the performance of different algorithms on the UNSW-NB15 dataset through these indicators, and Susanto et al. (2020) used these indicators to evaluate the performance of different classifiers in the N\_BaIoT dataset.

### 2.3 Related Paper

This section will provide an overview of the work related to Botnet detection in IoT environment using machine learning.

Garg et al. (2021) introduced the use of machine learning algorithms to detect intrusions and evaluate their accuracy in detecting malicious nodes. For this purpose, the UNSW-NB15 dataset was used to implement an intrusion detection system (IDS). The dataset contains labeled data of real network traffic, mainly divided into normal and abnormal (including nine different attack types). Six algorithms were used, including Decision Tree, Random Forest Gini, Support Vector Machine, Logistic Regression, RandomForest IG, and Gaussian Naive Bayes. The experimental results show that RandomForest IG has the highest accuracy of 92.63%.

Susanto et al. (2020) used the N\_BaIoT dataset, which has an unbalanced distribution of traffic and attack categories, containing normal traffic and attack traffic (such as Mirai and Bashlite attacks) of nine IoT devices, with a total of 115 features. Weka and Scikit-learn tools were used, and four classification methods were adopted: Random Forest, Decision Tree, Naive Bayes, and Adaboost. The results show that Random Forest has the highest classification accuracy of 99.99% in Scikit-learn and 100% in Weka .

Madhuri Gurunathrao Desai et al. (2021) used the N\_BaIoT public dataset, where each record contains 115 independent features. Through the feature selection process, the researchers selected 10 features with the highest accuracy for model building and evaluation. Unsupervised K-means clustering and supervised decision tree methods were used. The experimental results show that when the amount of normal data is larger than the amount of attack data, the hybrid method has higher accuracy which is 99.13% and fewer false positive values when  $k=2$ .

MohammadNoor Injadat et al. (2020) proposed a machine learning-based optimization framework that effectively detects botnet attacks on IoT devices by combining the Bayesian Optimized Gaussian Process (BO-GP) algorithm and the Decision Tree (DT) classification model. The Bot-IoT-2018 dataset used contains 72

million records and 46 features, and about 3.6 million records and 10 best features were used in the study. The experimental results show that the optimized DT model outperforms the default model and the support vector machine (SVM) model in various indicators which got 99.999% accuracy.

Rabhi et al. (2023) proposed a transfer learning-based intrusion detection system for detecting IoT botnets and compared deep learning and transfer learning models using artificial neural network (ANN) and long short-term memory network (LSTM) algorithms. Two datasets were used: N-BaIoT as the source domain and IoT-23 as the target domain. The results showed that the transfer learning model outperformed the deep learning model and got 98% accuracy under both algorithms.

Guerra-Manzanares et al. (2019) used the N-BaIoT dataset, which contains normal and malicious IoT traffic from 9 devices. 115 numerical features were defined for each data point. The k-nearest neighbor (k-NN) and random forest (RF) algorithms were used, and the results showed that the hybrid feature selection method significantly reduced the computational time while maintaining a high detection rate which 99.9% accuracy.

H Esha et al. (2023) used multiple datasets, including BoT-IoT, UNSW-NB15, and ToN\_IoT. The UNSW-NB15 dataset was finally selected and used. A variety of classification algorithms were used, including Linear SVC, Decision Tree, Random Forest, Logistic Regression, Gradient Boosting Decision Tree, and Ensemble Models. The results show that the Linear SVC has the best accuracy which is 98.865.

Bahsi et al. (2018) used the N-BaIoT dataset captured in a laboratory environment, simulating typical normal behaviors and attack cases. The dataset contains network traffic statistics of IoT devices of 9 different application categories. The results show that the decision tree classifier using three optimal features performs well in terms of detection accuracy and interpretability and got 98.97% with 10 feature set size.

Susanto et al. (2021) used the N-BaIoT dataset, which was extracted by statistical methods and contains 115 features and 89 files. These files include different



types of network traffic data such as normal traffic, Mirai (Ack, Scan, Syn, Udp, Udpplain) and Bashlite (Combo, Junk, Scan, TCP, UDP) attack traffic. The machine learning algorithms used include Random Forest (RF), Decision Tree (DT), Adaboost (AD), k-Nearest Neighbor (k-NN), and Gradient Boosting (GB). Experimental results show that the use of decision tree classifier combined with random projection method can reduce the number of features while still maintaining high accuracy and efficient detection performance which got 100% accuracy.

## 2.4 Critical Review

Table 2.1 shows some previous works that have been done by various people on IoT detection techniques using machine learning. The works have been categorized according to the detection techniques' names. These research papers have been reviewed to collect the studies about previous research and work implementing algorithms in IoT detection using machine learning. It can help to understand project scopes and system developments better.

Table 2.1 : Literature Review of Previous Works

Author	Dataset	Best Feature Selection	Best Algorithm	Accuracy
Garg et al., 2021	UNSW-NB15	None	Random Forest Gini	92.63%
Susanto et al., 2020	N_BaIoT	None	Random Forest	99.99% in Scikit-learn 100% in Weka
Madhuri Gurunathrao Desai et al., 2021	N_BaIoT	Recursive Feature Elimination(RFE)	K-means + Decision Tree	99.13%
Mohammad Noor Injadat et al., 2020	Bot-IoT-2018	Recursive Feature Elimination(RFE)	combination of BO-GP and DT classifier	99.99%

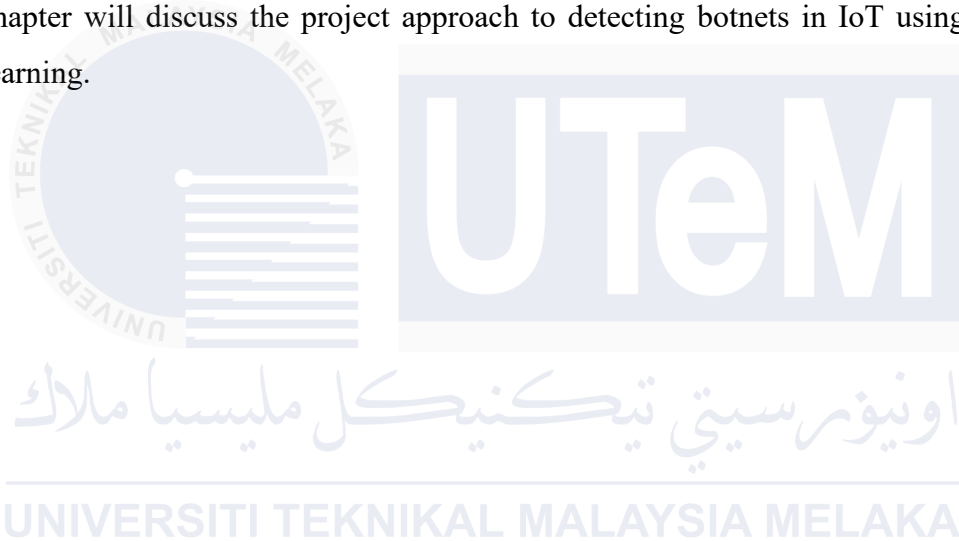
Rabhi et al., 2023	N-BaIoT as source domain, IoT-23 as target domain	None	Transfer Learning with LSTM	98%
Guerra-Manzanares et al., 2019	N_BaIoT	Filter Method + Wrapping Method + Embedding Method	k-Nearest Neighbors + Random Forest	99.9%
H Esha et al., 2023	UNSW-NB15	Lasso Method	Linear SVC	98.86%
Bahsi et al., 2018	N-BaIoT	Fisher's Score	Decision Tree	98.97% with 10 feature set size
Susanto et al., 2021	N-BaIoT	Random Projection	Decision Tree	100%

## 2.5 Proposed Solution

Table 2.1 summarizes the existing research and shows that N-BaIoT is adopted by a large number of studies, so this project will use N-BaIoT as a dataset. Recursive feature elimination (RFE), filtering method, and selection operator (Lasso) will be used as feature selection to select the most suitable features in the dataset. In addition, fire machine learning classifiers, namely linear SVC, random projection, random forest (RF), decision tree (DT), and k-nearest neighbor (KNN) will be implemented in the testing phase to determine the highest accuracy of this project. The performance evaluation for measuring the accuracy, precision, recall, and f1 score of the classifier will be used at the end of the botnet detection framework using machine learning methods.

## 2.6 Conclusion

In summary, this chapter introduced the details of IoT botnet detection based on machine learning, including the Internet of Things (IoT), botnets, botnets in IoT, datasets used by existing works, feature selection methods, and machine learning methods. In addition, related work shows how previous projects built the project, as well as the step-by-step process of system development. A critical review gives an overview of the existing work and will delve into the differences between the datasets used. Finally, the solution proposed in this project uses the same dataset used in most of the literature, and its performance will be evaluated in the testing chapter. The next chapter will discuss the project approach to detecting botnets in IoT using machine learning.



## CHAPTER 03 : PROJECT METHODOLOGY

### 3.1 Introduction

This chapter introduces the dataset used in this project, N\_BaIoT. It also discusses the project methodology that guided the project execution, namely the Agile Development Methodology, and briefly describes the process of this methodology. This methodology can be used as a roadmap to better understand what must be done and how it should be done.

### 3.2 Dataset Used

The dataset use in this project is N-BaIoT which consists of data samples with 115 features. The datasets were collected through the port mirroring of IoT devices with 9 device that authentically infected by Mirai and BASHLITE which are(Catillo et al., 2023) :

- i. Danmini - Doorbell
- ii. Ennio - Doorbell
- iii. Ecobee - Thermostat
- iv. Philips B120N/10 - Baby Monitor
- v. Provision PT-737E - Security Camera
- vi. Provision PT-838 - Security Camera
- vii. Simple Home XCS7-1002-WHT - Security Camera
- viii. Simple Home XCS7-1003-WHT - Security Camera
- ix. Samsung SNH 1011 N - Web cam

The benign data were captured immediately after setting the network to ensure that the data was benign. For two types of packet sizes (only outbound/both outbound and inbound), packet counts, and packet jitters, the times between packet arrival were extracted for each statistical value. A total of 23 features were extracted for each of the 5 time windows (100 ms, 500 ms, 1.5 s, 10 s, and 1 min), for a total of 115 features(Kim et al., 2020). Figure 3.1 shows the detailed features of this dataset.

Aggregated by	Value	Statistic	Total No. of Features
Source IP	Packet size (only outbound)	Mean, variance	3
	Packet count	Integer	
Source MAC-IP	Packet size (only outbound)	Mean, variance	3
	Packet count	Integer	
Channel	Packet size (only outbound)	Mean, variance	10
	Packet count	Integer	
	Amount of time between packet arrivals	Mean, variance, integer	
	Packet size (both inbound and outbound)	Magnitude, radius, covariance, correlation coefficient	
Socket	Packet size (only outbound)	Mean, variance	7
	Packet count	Integer	
	Packet size (both inbound and outbound)	Magnitude, radius, covariance, correlation coefficient	
<b>Total</b>			<b>23</b>

Figure 3.1 : Detailed features of the N-BaIoT dataset(Kim et al., 2020)

This dataset was chosen not only because it is the most commonly used dataset in relevant literature, but also because it has the following advantages(Catillo et al., 2023) (Abbasi et al., 2021):

- I. Multi-device coverage: The N\_BaIoT dataset covers 9 different IoT devices, including doorbells, thermostats, baby monitors, cameras, and webcams. These devices have different functions and network behaviors, which can fully reflect the traffic characteristics in the IoT environment.
- II. Multiple attack types: The dataset contains normal traffic and multiple attack traffic, especially Mirai and Bashlite attacks. These attacks are common malicious behaviors in the IoT environment and can be effectively used to study and evaluate different intrusion detection methods.
- III. Rich features: The N\_BaIoT dataset provides 115 features, which cover various statistical information and properties of the traffic, providing rich information for machine learning models and helping to improve the accuracy and robustness of detection.

- IV. Unbalanced data distribution: The traffic and attack categories of this dataset are unevenly distributed, which is highly similar to the real IoT environment. By training and testing on such a dataset, the applicability and performance of the model in real scenarios can be improved.
- V. Widely used and recognized: The N\_BaIoT dataset is widely used and recognized in many studies, with high credibility and practicality. Using such a recognized dataset helps to compare and verify with other research results and promote the progress of research.
- VI. Open access: The dataset is public and researchers can easily access and use it, which facilitates scientific research and development. At the same time, public datasets also contribute to the transparency and reproducibility of research.
- VII. High data quality: The N\_BaIoT dataset is captured in a laboratory environment, with high data quality and accuracy, and can provide reliable training and testing data for machine learning models.

### 3.3 Methodology

This project will use Agile Development Methodology to develop the system because of (Chaudhari & Joshi, 2021) (Kuhmann et al., 2021):

- i. Flexibility and adaptability: Agile methods allow researchers to flexibly respond to changing needs and environments during the development process. For academic projects, research directions and technical requirements may change with new discoveries or experimental results. Agile development can quickly adjust plans and strategies to maintain the flexibility and adaptability of the project.
- ii. Iterative development: Development is carried out through short cycles (iterations), and each iteration can produce a usable version. Iterative

development helps to gradually realize system functions, gradually verify the effectiveness of assumptions and algorithms, and reduce the risk of encountering major problems during final integration.

- iii. Continuous improvement: Agile development emphasizes continuous feedback and improvement. By communicating with supervisors regularly, researchers can reflect on and improve the development process, optimize system performance and code quality. This is particularly important for academic projects, because the detection effect and stability of the system can be gradually improved through continuous experiments and evaluations.
- iv. Results display and evaluation: After each iteration cycle, researchers can show their supervisors to evaluate the current research results and adjust the research direction and strategy if necessary to ensure the achievement of project goals.

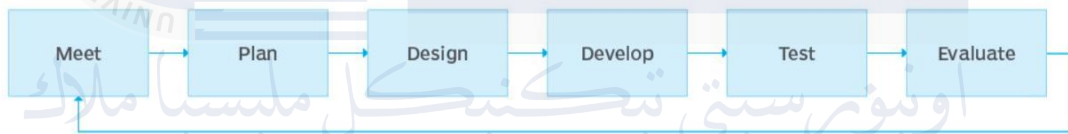


Figure 3.2 : Agile Development Methodology Cycle (Brush & Silverthorne, 2022)

- I. Meet: Meet and discuss with the supervisor regularly. The frequency of the meeting can be determined according to the progress of the project, with the purpose of discussing the progress of the project, sharing the challenges and problems encountered, and obtaining feedback and guidance from the supervisor.
- II. Plan: Make a plan for the time, including priority setting and scheduling. Planning in agile development is an ongoing process, and new plans are made at the beginning of each iteration.
- III. Design: Design the system and obtain a clear blueprint.
- IV. Develop: Develop code according to the design and gradually implement system functions.

V. Test: Test the developed functions to determine the current performance of the system.

VI. Evaluate: Evaluate the results of each iteration cycle, analyze the performance and effect of the system, and let the supervisor evaluate.

### 3.4 Project Milestones

Project milestones were specified actions or activities that needed to be completed within set time frames for the project to succeed. These milestones were essential for tracking progress and ensuring timely completion of tasks.

Table 3.1 : Project Milestone of FYP 1

WEEK	ACTIVITY
W1	Select an appropriate project topic and potential supervisor
	Meeting 1 for the proposal discussion with the supervisor for assessment, verification, correction, and improvement before the proposal approval and submission.
W2	Proposal approval and submission to the committee
	Official List of Supervisors
W3	Proposal Presentation
	Meeting 2 for the proposal presentation to the supervisor
W4	Chapter 1 Report Writing Progress 1 [PRJ-3]
W5	Chapter 2 Report Writing Progress 1 [PRJ-3]
	Meeting 3 for the discussion of chapter 1
W6	Chapter 2 Report Writing Progress 1 [PRJ-3]



	Project Progress 1 [PRJ-2]
W7	Chapter 3
W8	MID-SEMESTER BREAK
W9	Chapter 3 Report Writing Progress 1 [PRJ-3]
	Meeting 4 for the discussion of chapter 2
W10	Chapter 4
	Project Progress 2 [PRJ-5]
W11	Chapter 4
	PSM 1 Draft Report Preparation
W12	PSM 1 Draft Report Preparation
W13	PSM 1 Draft Report Preparation
	Meeting 5 for the discussion of chapter 3
W14	PSM 1 Draft Report Submission to Supervisor and Evaluator
	Report Evaluation [PRJ-9] [PRJ-10]
	Meeting 6 for the discussion of chapter 4
W15	Demonstration with Supervisor [PRJ-8]
	Demonstration with Evaluator [PRJ-7]
W16	Correction on the draft report based on the Supervisor and Evaluator's comments during the presentation •

Table 3.2 : Project Milestone of FYP 2

WEEK	ACTIVITY
W1	Discussion with supervisor for future planning

	Discussion with the supervisor for correction, and improvement from PSM1
W2	Chapter 5 Project Progress 1 [PRJ-1]
W3	Chapter 5 Project Progress 1 [PRJ-3]
W4	Chapter 6 Report Writing Progress 1 [PRJ-2]
W5	Chapter 6 Report Writing Progress 1 [PRJ-3]
	Meeting for the discussion of chapter 5
	Schedule for the presentation
W6	Chapter 7 Report Writing Progress 1 [PRJ-3]
	Determination of student status (Continue/Withdraw)
	PSM 2 Draft Report Preparation
W7	Demonstration with supervisor and evaluator
W8	Correction on the draft report based on the Supervisor and Evaluator's comments during the final presentation session.
	Submission of the final complete report, which is the updated & corrected PSM2 report
W9	Submission of the final complete report, which is the updated & corrected PSM2 report and Plagiarism Report etc. onto the OneDrive
	End of PSM 2

### 3.5 Gantt Chart

The table 3.3 and 3.4 is Gantt chart that outlines the time spent on each phase of the project, ensuring that all tasks were completed within the allocated period.

Progress \ Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
FYP Proposal	■	■												
Project Progress 1			■	■	■	■	■							
Report Writing			■	■	■	■	■	■						
Progress 1							■	■	■	■				
Project Progress 2									■	■	■	■	■	
Report Writing													■	■
Progress 2													■	■
Report Evaluation													■	■

Table 3.3 : Gantt chart Final Year Project 1

Progress \ Week	1	2	3	4	5	6	7	8
Chapter 5 - Project Progress	■	■						
Chapter 5 - Report Writing Progress		■	■					
Chapter 6 - Project Progress			■	■				
Chapter 6 - Report Writing Progress				■	■			
Chapter 7 - Project Progress					■	■		
Chapter 7 - Report Writing Progress						■	■	
Final Presentation								■

Table 3.4 : Gantt chart Final Year Project 2

### 3.6 Conclusion

In summary, this chapter outlined the project methodology for the system development of the N\_BaIoT dataset using an agile methodology. The six phases of the agile methodology are meet, plan, design, develop, test, evaluation. In addition, this chapter provides a visual representation of the project milestones and a Gantt chart to effectively monitor and manage the activities involved in developing a Botnet Detection in IoT using Machine Learning System. The next section describes the analysis and design of the project in more detail.

## CHAPTER 04 : ANALYSIS AND DESIGN

### 4.1 Introduction

This chapter will introduce the data structure and flowchart of the machine learning-based IoT botnet detection system.

### 4.2 Data Structure

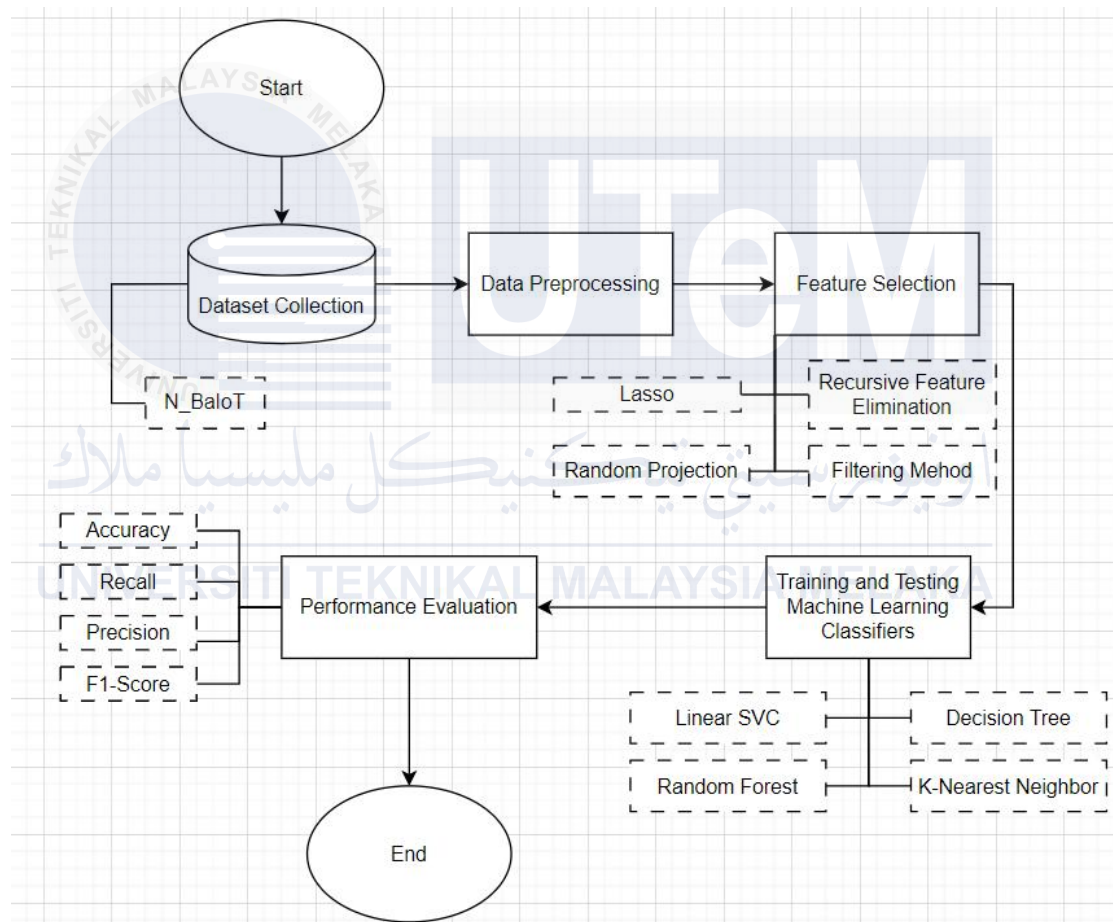


Figure 4.1 : Data Structure for Botnet Detection System

- I. Data Collection: This phase involves collecting data from various sources for subsequent processing and analysis.
  - a) N-BaIoT: This project uses the N-BaIoT dataset. This dataset contains normal and attack traffic from 9 types of IoT devices with a total of 115 features. These devices include doorbells, thermostats, baby monitors,

security cameras, and webcams. The collection of the dataset includes various statistical features of different time windows.

- II. Data Preprocessing: This phase involves cleaning and preparing the collected data to ensure its quality and consistency. Steps may include data cleaning, handling missing values, data standardization or normalization, etc.
- III. Feature Selection: In this step, the most representative and discriminative features are selected to improve the performance and training efficiency of the model.
  - a) Recursive Feature Elimination (RFE): The number of features is gradually reduced by recursively training the model and removing the least important features.
  - b) Filtering Method: Important features are selected by evaluating the statistical properties of each feature (such as variance, correlation coefficient).
  - c) Lasso (Least Absolute Shrinkage and Selection Operator): Features are selected using L1 regularization, which achieves feature selection by shrinking the coefficients of some features to zero.
- IV. Training and Testing ML Classifiers: This stage uses the selected features to train and test various machine learning models.
  - a) Linear SVC (Support Vector Classifier): Use linear support vector classifiers for training and testing, suitable for linearly separable data.
  - b) Random Projection: Use random projection technology to reduce data dimensions and perform classification.
  - c) Random Forest: Use the random forest algorithm to improve classification accuracy and generalization ability by integrating multiple decision trees.
  - d) Decision Tree: Use the decision tree algorithm to make classification decisions through a tree structure.
  - e) K-Nearest Neighbor (KNN): Use the K nearest neighbor algorithm to classify according to the categories of the nearest K data points.
- V. Performance Evaluation: This stage evaluates the performance of the model through a variety of indicators to ensure the accuracy and effectiveness of the model.

- a) Accuracy: Accuracy, which indicates the proportion of samples correctly predicted by the model.
- b) Recall: Recall rate, which indicates the proportion of actual positive class samples that are correctly predicted as positive.
- c) Precision: Precision rate, which indicates the proportion of samples predicted as positive that are actually positive.
- d) F1-Score: F1 score, the harmonic mean of precision and recall, used to comprehensively evaluate the performance of the model

### 4.3 System Flowchart

In the development of Botnet Detection in IoT using Machine Learning approach, this project proposed the framework as shown in Figure 4.2.

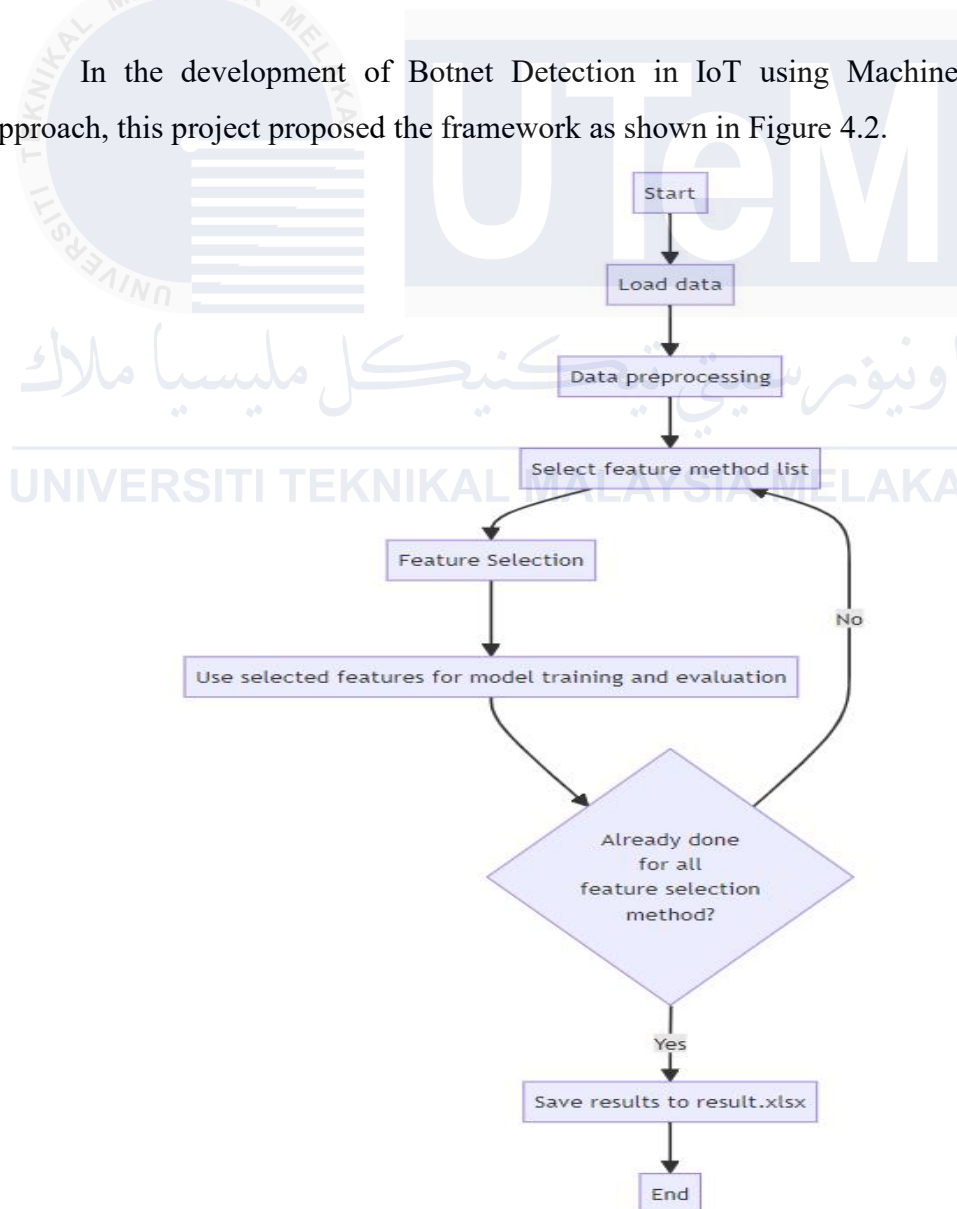


Figure 4.2 : Flowchart for Botnet Detection in IoT using Machine Learning

### 4.3.1 Data Collection

This project conducted the experiment using the N\_BaIoT dataset that suggests real traffic data, gathered from 9 commercial IoT devices authentically infected by Mirai and BASHLITE.

Total Size for this dataset is 8.14GB, get from [www.kaggle.com](https://www.kaggle.com/datasets/mkashifn/nbaiot-dataset) (https://www.kaggle.com/datasets/mkashifn/nbaiot-dataset). Figure 4.3 show a part of feature for this dataset.

△ Feature Name	△ Feature Descripti...
<b>115</b> unique values	<b>115</b> unique values
MI_dir_L5_weight	MI dir Lambda 5 weight
MI_dir_L5_mean	MI dir Lambda 5 mean
MI_dir_L5_variance	MI dir Lambda 5 variance
MI_dir_L3_weight	MI dir Lambda 3 weight
MI_dir_L3_mean	MI dir Lambda 3 mean
MI_dir_L3_variance	MI dir Lambda 3 variance
MI_dir_L1_weight	MI dir Lambda 1 weight
MI_dir_L1_mean	MI dir Lambda 1 mean
MI_dir_L1_variance	MI dir Lambda 1 variance
MI_dir_L0.1_weight	MI dir Lambda 0.1 weight

Figure 4.3 : Part of N\_BaIoT dataset feature

Feature information(Sundaram, 2023):

- i. Stream aggregation:
  - a) H: ("Source IP" in N-BaIoT paper) Stats summarizing the recent traffic from this packet's host (IP)

- b) MI: ("Source MAC-IP" in N-BaIoT paper) Stats summarizing the recent traffic from this packet's host (IP + MAC)
- c) HH: ("Channel" in N-BaIoT paper) Stats summarizing the recent traffic going from this packet's host (IP) to the packet's destination host.
- d) HH\_jit: ("Channel jitter" in N-BaIoT paper) Stats summarizing the jitter of the traffic going from this packet's host (IP) to the packet's destination host.
- e) HpHp: ("Socket" in N-BaIoT paper) Stats summarizing the recent traffic going from this packet's host+port (IP) to the packet's destination host+port.  
Example 192.168.4.2:1242 -> 192.168.4.12:80

- ii. Time-frame (The decay factor Lambda used in the damped window):
  - a) How much recent history of the stream is capture in these statistics
    - i. L5, L3, L1, L0.1 and L0.01
- iii. The statistics extracted from the packet stream:
  - a) weight: The weight of the stream (can be viewed as the number of items observed in recent history)
  - b) mean: ...
  - c) std: ...
  - d) radius: The root squared sum of the two streams' variances
  - e) magnitude: The root squared sum of the two streams' means
  - f) cov: An approximated covariance between two streams
  - g) pcc: An approximated correlation coefficient between two streams

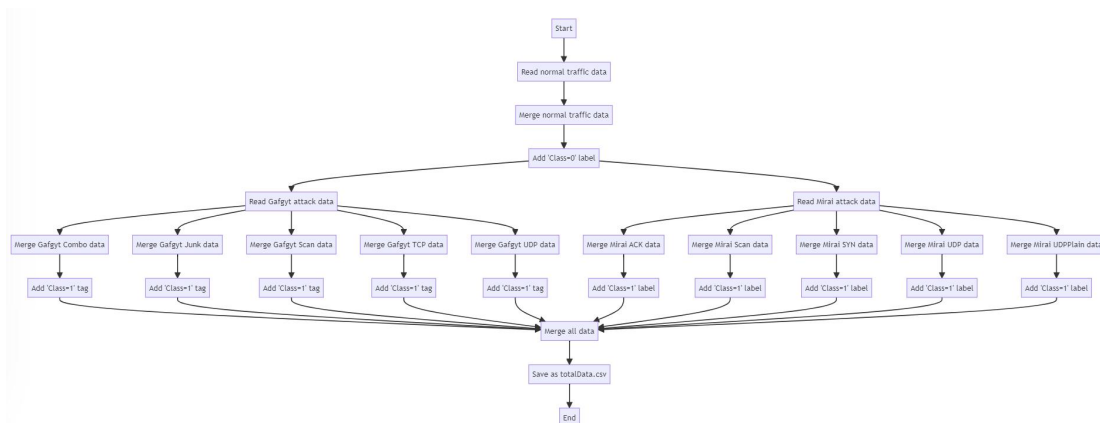


Figure 4.4 : Flowchart for Data Collection and Prepare



### 4.3.2 Data Preprocessing

#### I. Get the dataset

- a) First, download and get the N-BaIoT dataset from [www.Kaggle.com](http://www.Kaggle.com). This dataset contains normal traffic and attack traffic, covering 9 different IoT devices and two Botnet attacks. The data collection of these devices includes various statistical features in different time windows. Kaggle's data is saved in CSV file format and has been cleaned. CSV file is a widely used text file format that can be easily stored and transmitted(Tapsai, 2018).

#### II. Import necessary libraries

- a) In order to preprocess the data, need to import some Python related libraries. These libraries provide a wealth of functions and tools to help researchers perform data manipulation, analysis, and visualization.
- b) Required libraries:
  - i. Numpy: used to perform mathematical calculations and operate multidimensional arrays. Numpy provides efficient array manipulation methods and rich mathematical functions to facilitate the processing of numerical data(Python, 2020).
  - ii. Pandas: used for data manipulation and analysis. Pandas provides powerful data structures (such as DataFrame) that can easily clean, transform, and analyze data(Python, 2020).
  - iii. Matplotlib and Seaborn: used for data visualization. Matplotlib is a 2D drawing library. Seaborn is an advanced visualization library built on Matplotlib. It provides a simpler API and richer drawing functions, which helps to examine data distribution and characteristics(Python, 2020).

#### III. Loading the data set

- a) Use the Pandas library to read the CSV file and load the data into a data frame (DataFrame). DataFrame is a data structure in Pandas, which is similar to a spreadsheet or SQL table and can be easily used for data manipulation and analysis(Python, 2020).

#### IV. Data standardization or normalization

- a) In order to compare and calculate different features on the same scale, the data needs to be standardized or normalized. This helps to eliminate the dimensional differences between different features and improve the training effect and performance of the model.

#### V. Encoding categorical data

- a) If the data set contains categorical variables, these categorical variables need to be converted to numerical form. Machine learning algorithms usually can only process numerical data, so categorical variables need to be encoded.

#### VI. Data splitting

- a) Split the data set into a training set and a test set. This is to ensure the independence of model training and evaluation. Split it in a 70:30 ratio, that is, 70% of the data is used to train the model and 30% of the data is used to evaluate the model performance.

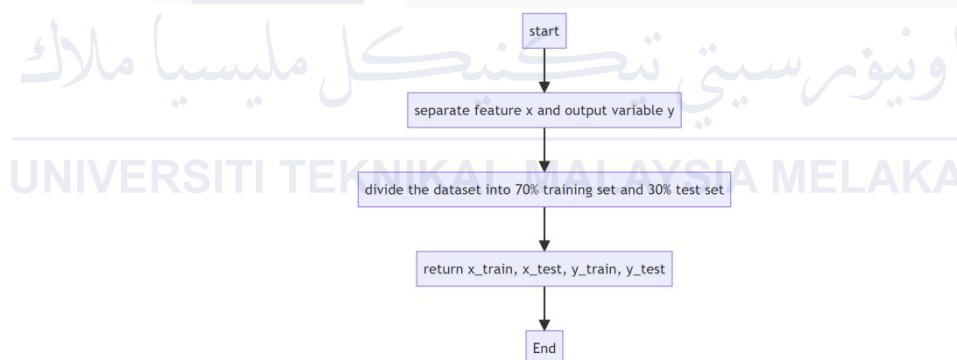


Figure 4.5 : Flowchart for Data Preprocessing

#### 4.3.3 Feature Selection

Feature selection is an important step in data preprocessing. Its purpose is to select the features that best represent and distinguish data categories, thereby improving the performance of the model and the training efficiency. Feature selection includes the following three methods in this project:

- i. Recursive Feature Elimination (RFE)

- a) According to Chen & Jeong(2007), recursive feature elimination is a method of recursively training a model and gradually reducing the number of features. The specific steps are as follows:
- i. Training model: First, train a base model (such as linear regression or support vector machine) using all features.
  - ii. Calculate feature importance: Calculate the importance of each feature based on the coefficients of the model or the feature importance score.
  - iii. Remove the least important features: Remove the features with the lowest importance score.
- b) Repeat the above steps: Repeat the above steps on the remaining feature set until the predetermined number of features is reached.
- c) This method can find the feature subset that contributes most to the model performance through repeated training and screening.
- ii. Chi-Square Filtering Method
- a) Filtering methods select important features based on the statistical properties of features and usually do not rely on any machine learning model. Common filtering methods include(Cherrington et al., 2019):
- i. Variance selection method: select features with variance greater than a certain threshold and ignore features with smaller variance because they have less impact on the target variable.
  - ii. Correlation coefficient method: calculate the correlation coefficient between the feature and the target variable and select features with high correlation.
  - iii. Chi-square test: for classification tasks, use the chi-square test to measure the independence between the feature and the target variable and select features with significant correlation.
- b) This method is simple and fast and suitable for preliminary feature screening.
- iii. Lasso (Least Absolute Shrinkage and Selection Operator)

- a) Lasso is a linear model based on L1 regularization. By adding regularization terms, the coefficients of some features are reduced to zero, thereby achieving feature selection (Muthukrishnan & Rohini, 2016). The specific steps are as follows:
- i. Training Lasso model: Add L1 regularization terms to the loss function of the model and train the model.
  - ii. Feature coefficient shrinkage: L1 regularization will cause some feature coefficients to become zero, thereby automatically selecting the features that contribute most to the model.
  - iii. Select non-zero coefficient features: Finally, select those features with non-zero coefficients as important features.
- b) Lasso not only can handle multicollinearity problems, but also achieve feature selection, which is very suitable for high-dimensional data sets.
- iv. Random Projection
- a. Random projection is a dimensionality reduction technique that randomly projects high-dimensional data into a low-dimensional space, maintaining the data structure while reducing computational complexity (Heidari et al., 2021). The steps are as follows:
    - i. Use random projection technology to reduce the dimension of the data.
    - ii. Train and test classification models (such as SVM, KNN, etc.) on the reduced-dimensional data.
- v. Hybrid Feature Selection Method
- a. Hybrid Feature Selection Method is a strategy that combines multiple feature selection techniques, aiming to comprehensively utilize the advantages of each method to improve the effectiveness of feature selection and the performance of the model. Through this hybrid strategy, it is possible to maximize the retention of important features related to the target variable while reducing the data dimension (Hsu et al., 2011). The steps are as follows:

- i. Use multiple feature selection methods (such as Lasso, Random Projection, Chi-Square, RFE, etc.) to select features separately.
- ii. Combine the selected feature sets and select the best feature combination through cross-validation or other evaluation indicators.
- iii. Train and test the classification model (such as SVM, Random Forest, KNN, etc.) on the selected best feature combination.

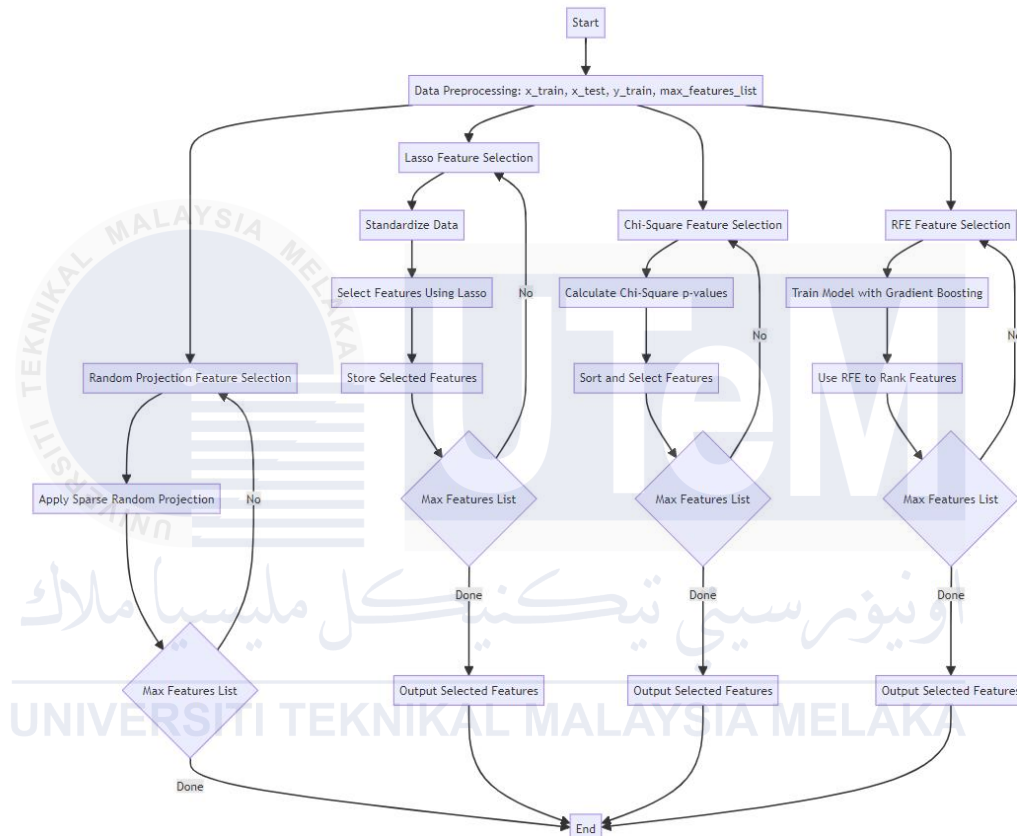


Figure 4.6 : Flowchart for Non-Hybrid Feature Selection

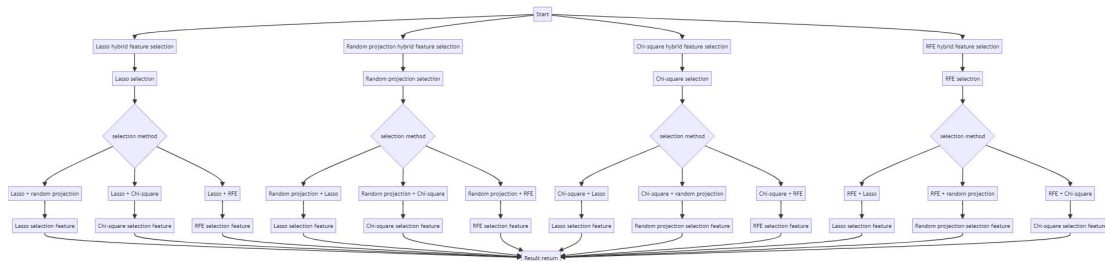


Figure 4.7 : Flowchart for Hybrid Feature Selection

### 4.3.4 Training and Testing Machine Learning Classifiers

After completing feature selection, use the selected features to train and test different machine learning models to find the best classifier. This project includes the following classifiers:

## I. Linear Support Vector Machine (Linear SVC)

b. Linear Support Vector Machine is a classification algorithm suitable for linearly separable data(Dixit et al., 2020). The steps are as follows:

- i. Train the linear support vector machine model using the training set.
- ii. Adjust hyperparameters (such as regularization parameters) to optimize model performance.

## II. Random Forest

a. Random Forest is an ensemble learning method that improves classification accuracy and generalization ability by integrating multiple decision trees(Ashari & Broschat, 2019). The steps are as follows:

- i. Use the training set to train multiple decision trees, and each tree is trained on a randomly selected feature subset.
- ii. Combining the prediction results of all decision trees, the majority voting method is used to determine the final classification result.

## III. Decision Tree

a. Decision tree is a classification algorithm based on tree structure. It achieves classification by dividing the decision boundary in the feature space(Ashari & Broschat, 2019). The steps are as follows:

- i. Use the training set to train the decision tree model and select the optimal features for data division.
- ii. Prune to avoid overfitting and improve the generalization ability of the model.

## IV. K-Nearest Neighbor (KNN)

a. K-Nearest Neighbor is an instance-based learning algorithm. By calculating the distance between the sample and each point in the training set, the K points closest to the sample are selected for classification(Nahin Ul Sadad et al., 2021). The steps are as follows:

- i. Select an appropriate K value (the number of nearest neighbors).
- ii. Calculate the distance between the sample to be classified and all samples in the training set.
- iii. Select the K samples closest to the sample and determine the classification result by majority voting.

#### 4.3.5 Performance Evaluation

1. Select evaluation data : Use the test data set from the previous data splitting stage to evaluate the model. Make sure that the test data is not used for model training to ensure the authenticity and generalization of the evaluation results.
2. Calculate evaluation indicators : Predict the test data and calculate various evaluation indicators. The number of various prediction results (true positive, false positive, false negative, true negative) can be easily obtained through the confusion matrix, so as to calculate the accuracy, recall, precision and F1-score.
3. Compare different models : Compare the evaluation results of each machine learning model and select the model with the best performance.
4. Model adjustment and optimization : According to the evaluation results, adjust the model's hyperparameters or perform further feature engineering to optimize the model performance. Keep iterating this process until a satisfactory model is obtained.

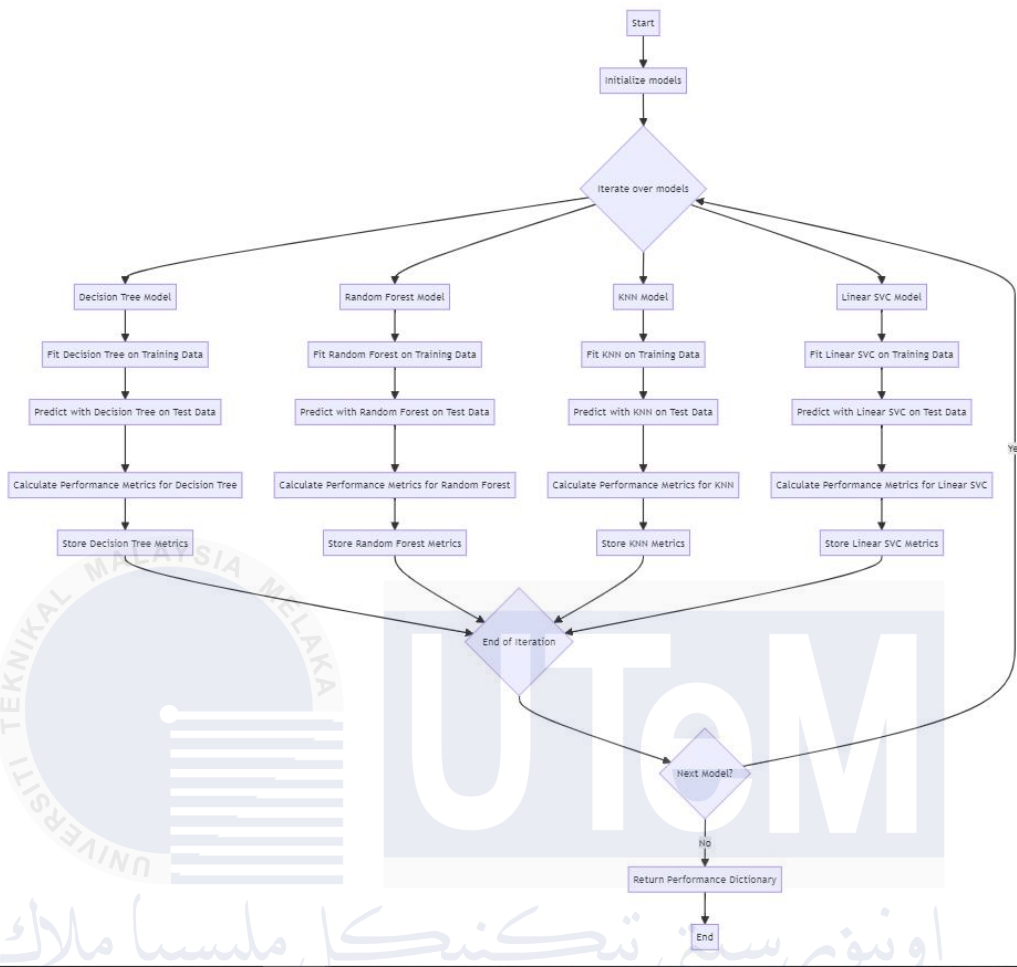


Figure 4.8 : Flowchart for Machine Learning Classifier and Performance Evaluation

#### 4.4 Conclusion

In summary, this chapter provides a comprehensive analysis and design of the machine learning-based IoT botnet detection system. First, this chapter explains the data structure of the system and details the collection, preprocessing, and feature selection process of the N-BaIoT dataset. By using different feature selection methods, such as recursive feature elimination (RFE), filtering methods, and Lasso, the researchers were able to select the most representative features. Subsequently, the researchers used these features to train and test a variety of machine learning classifiers, including linear support vector machines (Linear SVC), random projections, random forests, decision trees, and K-nearest neighbors (KNN). Finally, the models were evaluated using metrics such as accuracy, recall, precision, and F1 score. The next chapter will focus on the implementation phase of the Botnet detection system in home IoT devices.



## CHAPTER 05 : ANALYSIS AND DESIGN

### 5.1 Introduction

This chapter introduces the implementation of dataset description, data preprocessing, feature selection, machine learning implementation, and performance evaluation based on figure 4.1. This project will measure the model accuracy evaluation to obtain the expected output.

### 5.2 Botnet Detection System Configuration Management

#### 5.2.1 Dataset Collection

This project using N\_BaIoT dataset from Kaggle.com, total 89 files, the file with the name benign records normal traffic, file with mirai records Mirai attack traffic and file with gafgyt records Bashlite attack(Bashlite also knows as gafgyt).

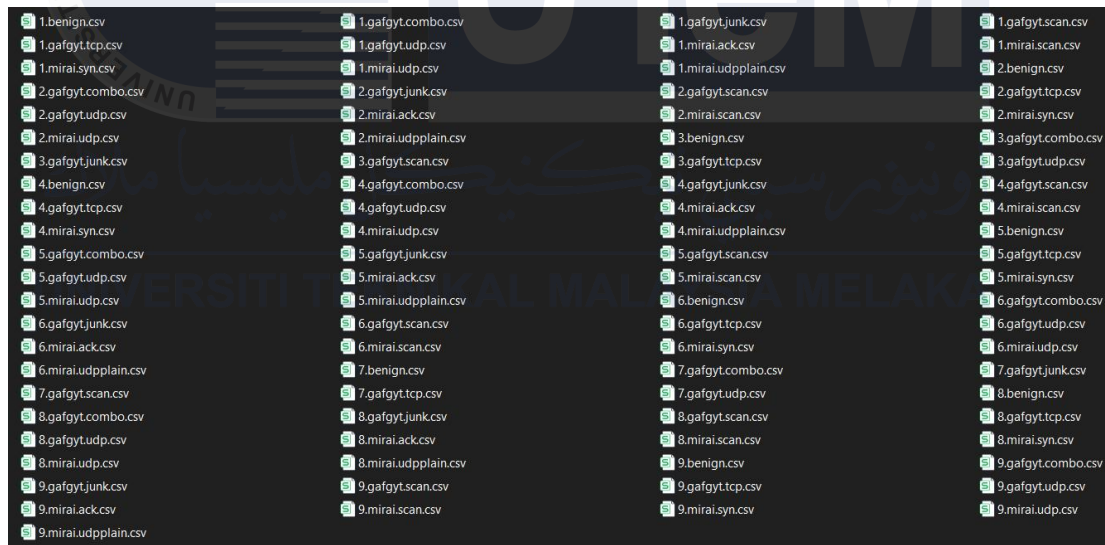


Figure 5.1 : Files of N\_BaIoT Dataset

Using dataPrepare.py to combine all the files of N\_BaIoT Dataset into totalData.csv. The code is mainly used to read different types of network traffic data (normal traffic and Mirai, Bashlite attack traffic), and then merge them into a large data set and save it to a CSV file. The final output file contains all the merged data and adds a Class label for each type of traffic to distinguish normal traffic from attack traffic.

#### i. Import necessary libraries

- a) pandas (pd): A powerful data processing and analysis library, especially suitable for processing structured data (such as CSV files).
- b) glob: A library for finding file paths that match a certain pattern. The library allows the use of wildcards to match file names.

```
import pandas as pd
import glob
```

Figure 5.2 : Coding for Import Necessary Libraries for Data Collection

#### ii. Define a data preparation function

- a) A function called dataPrepare is defined to prepare the dataset.

```
def dataPrepare():
```

Figure 5.3 : Coding for dataPrepare function define

#### iii. Read normal traffic data

- a) benign\_files: Use the glob library to find all files ending with benign.csv in the current directory. \*benign.csv is a wildcard pattern that matches all files whose names end with benign.csv.
- b) benign\_data: Use pd.read\_csv() to read these files one by one and merge them into a large DataFrame. pd.concat() is used to connect multiple DataFrames, and ignore\_index=True ensures that the merged DataFrame has a continuous row index.
- c) benign\_data['Class'] = 0: Add a new column Class for the normal traffic data and set its value to 0, indicating that these data are normal traffic.

```
benign_files = glob.glob("*benign.csv")
benign_data = pd.concat((pd.read_csv(file) for file in benign_files), ignore_index=True)
benign_data['Class'] = 0
```

Figure 5.4: Coding for Read normal traffic

#### iv. Reading Mirai attack data

- a) This block of code is similar to reading normal traffic data, but it targets five different types of Mirai attack data: ack, scan, syn, udp, and udpplain.
- b) The files for each attack type are read separately and merged, and then a new column Class is added to the data with a value of 1, indicating that this data is attack traffic.

```

mirai_ack_files = glob.glob("*mirai.ack.csv")
mirai_ack_data = pd.concat((pd.read_csv(file) for file in mirai_ack_files),
ignore_index=True)
mirai_ack_data['Class'] = 1
mirai_scan_files = glob.glob("*mirai.scan.csv")
mirai_scan_data = pd.concat((pd.read_csv(file) for file in mirai_scan_files),
ignore_index=True)
mirai_scan_data['Class'] = 1
mirai_syn_files = glob.glob("*mirai.syn.csv")
mirai_syn_data = pd.concat((pd.read_csv(file) for file in mirai_syn_files),
ignore_index=True)
mirai_syn_data['Class'] = 1
mirai_udp_files = glob.glob("*mirai.udp.csv")
mirai_udp_data = pd.concat((pd.read_csv(file) for file in mirai_udp_files),
ignore_index=True)
mirai_udp_data['Class'] = 1
mirai_udpplain_files = glob.glob("*mirai.udpplain.csv")
mirai_udpplain_data = pd.concat((pd.read_csv(file) for file in mirai_udpplain_files),
ignore_index=True)
mirai_udpplain_data['Class'] = 1

```

Figure 5.5 : Coding for Read Mirai Attack Traffic

#### v. Read Bashlite attack data

- a) This part of the code is similar to processing Mirai attack data, processing five different attack data of Bashlite (i.e. Gafgyt): combo, junk, scan, tcp and udp.
- b) The data of each attack type is also read and merged separately, and the Class column is added with a value of 1, indicating attack traffic.

```

gafgyt_combo_files = glob.glob("*gafgyt.combo.csv")
gafgyt_combo_data = pd.concat((pd.read_csv(file) for file in gafgyt_combo_files),
ignore_index=True)
gafgyt_combo_data['Class'] = 1
gafgyt_junk_files = glob.glob("*gafgyt.junk.csv")
gafgyt_junk_data = pd.concat((pd.read_csv(file) for file in gafgyt_junk_files),
ignore_index=True)
gafgyt_junk_data['Class'] = 1
gafgyt_scan_files = glob.glob("*gafgyt.scan.csv")
gafgyt_scan_data = pd.concat((pd.read_csv(file) for file in gafgyt_scan_files),
ignore_index=True)
gafgyt_scan_data['Class'] = 1
gafgyt_tcp_files = glob.glob("*gafgyt.tcp.csv")
gafgyt_tcp_data = pd.concat((pd.read_csv(file) for file in gafgyt_tcp_files),
ignore_index=True)
gafgyt_tcp_data['Class'] = 1
gafgyt_udp_files = glob.glob("*gafgyt.udp.csv")
gafgyt_udp_data = pd.concat((pd.read_csv(file) for file in gafgyt_udp_files),
ignore_index=True)
gafgyt_udp_data['Class'] = 1

```

Figure 5.6 : Coding for Read Bashlite Attack Data

#### vi. Merge all data and save

- a) frames: A list containing all data sets (normal data and various attack data).

- b) result: Use `pd.concat()` to merge all DataFrames in the list into one large DataFrame and reset the row index (`ignore_index=True`).
- c) `result.to_csv('totalData.csv')`: Save the merged data as `totalData.csv` file.
- d) `print("Starting to save data...")` and `print("Done for saving")`: Print messages to mark the start and end of the saving process.
- e) `print(result.shape)`: Print the shape (number of rows and columns) of the merged data.
- f) `print(result.head())`: Print the first five rows of the merged data for inspection.

```
frames = [benign_data, mirai_ack_data, mirai_scan_data, mirai_syn_data, mirai_udp_data,
mirai_udpplain_data, gafgyt_combo_data, gafgyt_junk_data, gafgyt_scan_data, gafgyt_tcp_data,
gafgyt_udp_data]
result = pd.concat(frames, ignore_index=True)
print("Starting to save data...")
result.to_csv('totalData.csv')
print("Done for saving")
print(result.shape)
print(result.head())
```

Figure 5.7 : Coding for Merge all data and save

#### vii. Main program part

- a) `if __name__ == '__main__':`: Make sure that the `dataPrepare` function is only executed when the script is run directly, and not when it is imported by other scripts.
- b) `dataPrepare()`: Call the `dataPrepare` function defined above to perform the data preparation process.
- c) The following code segment reads the `totalData.csv` file just saved through pandas, then removes the first column (redundant index column) and saves it back to the same file. Finally, read and print the file contents again to ensure that the data is saved correctly.

```
if __name__ == '__main__':
    dataPrepare()

    # using pandas to read csv file
    data = pd.read_csv('totalData.csv')
    data = data.iloc[:, 1:]
    data.to_csv('totalData.csv', index=False)
    data = pd.read_csv('totalData.csv')
    print(data)
```

Figure 5.8 : Coding for function call and display

Result of dataPrepare.py, total 7062606 rows and 116 columns of data save in totalData.csv.

```
[Running] set PYTHONIOENCODING=utf8 && python -u "c:\Coding\N_BaIoT\dataPrepare.py"
Starting to save data...
Done for saving
(7062606, 116)
  MI_dir_L5_weight  MI_dir_L5_mean  ...  HpHp_L0.01_pcc  Class
0      1.000000      60.000000  ...      0.0      0
1      1.000000     354.000000  ...      0.0      0
2      1.857879     360.458980  ...      0.0      0
3      1.000000     337.000000  ...      0.0      0
4      1.680223     172.140917  ...      0.0      0

[5 rows x 116 columns]
  MI_dir_L5_weight  MI_dir_L5_mean  ...  HpHp_L0.01_pcc  Class
0      1.000000      60.000000  ...      0.0      0
1      1.000000     354.000000  ...      0.0      0
2      1.857879     360.458980  ...      0.0      0
3      1.000000     337.000000  ...      0.0      0
4      1.680223     172.140917  ...      0.0      0
...
7062601      1.000000      60.000000  ...      0.0      1
7062602      1.000000      60.000000  ...      0.0      1
7062603      1.000000      60.000000  ...      0.0      1
7062604      1.000000      60.000000  ...      0.0      1
7062605      1.000000      60.000000  ...      0.0      1

[7062606 rows x 116 columns]
[Done] exited with code=0 in 1645.568 seconds
```

Figure 5.9: Output of Data Prepare

## 5.2.2 Main Coding

Main.py code implements the complete process of feature selection and model training, and saves the final results.

### I. Import required libraries

- from sklearnx import patch\_sklearn: This line of code imports the patch\_sklearn function in the sklearnx library to speed up the operation of scikit-learn.
- patch\_sklearn(): By calling the patch\_sklearn() function, the scikit-learn library is optimized to make some calculations faster, especially on Intel hardware.
- from scipy.stats import chi2\_contingency: Import the chi2\_contingency function in the scipy.stats module, which is used to calculate the chi-square test statistic for analyzing the correlation between two categorical variables.
- from sklearn.model\_selection import train\_test\_split: Import the train\_test\_split function to split the data set into training and test sets.

- e) `from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score`: Import four indicators for evaluating the performance of classification models, including accuracy (`accuracy_score`), recall rate (`recall_score`), precision (`precision_score`), F1 score (`f1_score`).
- f) `from sklearn.linear_model import LogisticRegression`: Import the logistic regression model (`LogisticRegression`), which is a commonly used classification model.
- g) `from sklearn.feature_selection import SelectFromModel`: Import the `SelectFromModel` class, which is used to select features based on an existing model.
- h) `from sklearn.preprocessing import StandardScaler`: Import the `StandardScaler` class, which is used to standardize features so that the mean of each feature is 0 and the standard deviation is 1.
- i) `from sklearn.tree import DecisionTreeClassifier`: Import the decision tree classifier (`DecisionTreeClassifier`), which is a tree model for classification tasks.
- j) `from sklearn.neighbors import KNeighborsClassifier`: Imports `KNeighborsClassifier`, a neighbor-based classification model.
- k) `from sklearn.feature_selection import RFE`: Imports the Recursive Feature Elimination (RFE) method for feature selection.
- l) `from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier`: Imports `RandomForestClassifier` and `GradientBoostingClassifier`, both of which are ensemble learning models for classification tasks.
- m) `from sklearn.svm import LinearSVC`: Imports the `LinearSVC` of the Support Vector Machine for linearly separable classification tasks.
- n) `from sklearn import random_projection`: Imports the `random_projection` module for random projection dimensionality reduction.
- o) `import pandas as pd`: Imports the pandas library for data processing, especially for processing DataFrames.
- p) `import numpy as np`: Import the numpy library, which is used for scientific computing, especially for processing arrays and matrices.

```
from sklearnx import patch_sklearn
patch_sklearn()
from scipy.stats import chi2_contingency
```

```

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import LinearSVC
from sklearn import random_projection

import pandas as pd
import numpy as np

```

Figure 5.10 : Coding for Import library for main.py

## II. Main program entry & data loading

- Detects whether the current script is running directly. If so, it executes the subsequent code.
- `print("Start load data")`: Prints a message indicating the start of data loading.
- `data = pd.read_csv('N_BaIoT/totalData.csv')`: Uses pandas's `read_csv` function to load the CSV file `totalData.csv` and stores it in the `data` variable.
- `print("Data loaded\n\nNow data preprocessing Start")`: Prints a message indicating that the data has been successfully loaded and data preprocessing is about to start.

```

if __name__ == '__main__':
    print("Start load data")
    data = pd.read_csv('N_BaIoT/totalData.csv')
    print("Data loaded\n\nNow data preprocessing Start")

```

Figure 5.11 : Coding for Main Program Entry and Data Loading

## III. Data Preprocessing

- `x_train, x_test, y_train, y_test = dataPreprocessing(data)`: Call the previously defined `dataPreprocessing` function to preprocess the data and divide it into training and test sets. `x_train` and `x_test` are the feature data of the training and test sets respectively, and `y_train` and `y_test` are the target variables of the training and test sets respectively.
- `print("Data Preprocessing End")`: Print a message indicating that data preprocessing is complete.

```

x_train, x_test, y_train, y_test = dataPreprocessing(data)
print("Data Preprocessing End")

```

Figure 5.12 : Coding for Data Preprocessing Function Call

#### IV. Define the number of features to be tested

- a) `max_features_list = [2, 3, 5, 10, 15]`: Define a list containing different numbers of features that will be used to test the performance of the model.
- b) `results = []`: Initialize an empty list to store the final test results.

```
max_features_list = [2, 3, 5, 10, 15]
results = []
```

Figure 5.13 : Coding for Define Number of Features List

#### V. Create a dictionary of feature selection methods

- a) `feature_selection_methods`: Create a dictionary with the key being the name of the feature selection method and the value being the corresponding feature selection function. These methods include Lasso, random projection, chi-square test, recursive feature elimination (RFE), and hybrids of these methods with other methods.

```
feature_selection_methods = {
    'Lasso': lasso_feature_selection,
    'Random Projection': random_projection_feature_selection,
    'Chi-Square': chi_square_feature_selection,
    'RFE': rfe_feature_selection,
    'Lasso + Hybrid': lasso_hybrid_feature_selection,
    'Random Projection + Hybrid': random_projection_hybrid_feature_selection,
    'Chi-Square + Hybrid': chi_square_hybrid_feature_selection,
    'RFE + Hybrid': rfe_hybrid_feature_selection
}
```

Figure 5.14 : Coding for Feature Selection Methods Dictionary

#### VI. Iterate over feature selection methods and store results

- a) `for method_name, feature_selection_function in feature_selection_methods.items():` Iterate over each feature selection method and its corresponding function in the dictionary.
- b) `print(f"\n{method_name} Feature Selection Start")`: Prints a message indicating the start of the current feature selection method.
- c) `selected_features_results_list = []`: Initializes an empty list to store the results of the current feature selection method.

```
for method_name, feature_selection_function in feature_selection_methods.items():
    print(f"\n{method_name} Feature Selection Start")
    selected_features_results_list = []
```

Figure 5.15 : Coding for Iterate over feature selection methods and store results



## VII. Handling hybrid feature selection methods

- a) if `method_name == 'Lasso + Hybrid'`: Check if the current method is a hybrid method of Lasso.
- b) `lasso_rp_results`, `lasso_chi_results`, `lasso_rfe_results` = `feature_selection_function(x_train, x_test, y_train, max_features_list)`: Call the feature selection function of the Lasso hybrid method to obtain the feature selection results of different combinations.
- c) `selected_features_results_list`: Store the above results in a list, where each element is a tuple containing the name of the submethod and its corresponding result.
- d) Other hybrid methods (random projection, chi-square test, RFE) are handled similarly.

```
if method_name == 'Lasso + Hybrid':
    lasso_rp_results, lasso_chi_results, lasso_rfe_results =
feature_selection_function(x_train, x_test, y_train, max_features_list)
    selected_features_results_list = [
        ('Random Projection', lasso_rp_results),
        ('Chi-Square', lasso_chi_results),
        ('RFE', lasso_rfe_results)
    ]
    elif method_name == 'Random Projection + Hybrid':
        rp_lasso_results, rp_chi_results, rp_rfe_results =
feature_selection_function(x_train, x_test, y_train, max_features_list)
        selected_features_results_list = [
            ('Lasso', rp_lasso_results),
            ('Chi-Square', rp_chi_results),
            ('RFE', rp_rfe_results)
        ]
    elif method_name == 'Chi-Square + Hybrid':
        chi_lasso_results, chi_rp_results, chi_rfe_results =
feature_selection_function(x_train, x_test, y_train, max_features_list)
        selected_features_results_list = [
            ('Lasso', chi_lasso_results),
            ('Random Projection', chi_rp_results),
            ('RFE', chi_rfe_results)
        ]
    elif method_name == 'RFE + Hybrid':
        rfe_lasso_results, rfe_rp_results, rfe_chi_results =
feature_selection_function(x_train, x_test, y_train, max_features_list)
        selected_features_results_list = [
            ('Lasso', rfe_lasso_results),
            ('Random Projection', rfe_rp_results),
            ('Chi-Square', rfe_chi_results)
        ]
]
```

Figure 5.16 : Coding for Handling hybrid feature selection method

## VIII. Handling non-hybrid feature selection methods

- a) For non-hybrid methods, the feature selection function is called directly and the results are added to `selected_features_results_list`.

```
else:
    selected_features_results = feature_selection_function(x_train, x_test, y_train,
max_features_list)
    selected_features_results_list.append((method_name, selected_features_results))
print(f"{method_name} Feature Selection End")
```

Figure 5.17 : Coding for handling non-hybrid feature selection method

## IX. Train and evaluate the model using selected features

- a) for `sub_method_name`, `selected_features_results` in `selected_features_results_list`: Iterate over all feature selection sub-methods and their corresponding results.
- b) for `num_features` in `max_features_list`: For each sub-method, iterate over all the number of features to be tested.
- c) `print(f"\nTrain and test model with {method_name} - {sub_method_name}-selected features Start for {num_features} features")`: Prints a message indicating that model training and testing has started with the current number of features.
- d) `performance = train_and_evaluate(selected_features_results[num_features][0], selected_features_results[num_features][1], y_train, y_test)`: Calls the `train_and_evaluate` function to train and evaluate the model using the selected number of features and returns the performance metrics of the model.

```
for sub_method_name, selected_features_results in selected_features_results_list:
    for num_features in max_features_list:
        print(f"\nTrain and test model with {method_name} - {sub_method_name}-selected
features Start for {num_features} features")
        performance = train_and_evaluate(selected_features_results[num_features][0],
selected_features_results[num_features][1], y_train, y_test)
        print(f"Train and test model with {method_name} - {sub_method_name}-selected
features End for {num_features} features\n")
```

Figure 5.18 : Coding for train and evaluate

## X. Display and store performance results

- a) `print(f"Performance with {method_name} + {sub_method_name} feature selection ({num_features} features):")`: Prints a message showing the performance results for the current number of features.
- b) for `model`, `metrics` in `performance.items()`: Iterates over the performance results and prints the performance metrics for each model separately.

- c) `results.append({...})`: Stores the performance metrics for each model in the results list for easy later saving.

```
print(f"Performance with {method_name} + {sub_method_name} feature selection
({num_features} features):")
for model, metrics in performance.items():
    print(f"{model}: {metrics}")
    results.append({
        "Number of Features": num_features,
        "Model": model,
        "Feature Selection Method": f'{method_name} - {sub_method_name}',
        "Accuracy(%)": metrics['Accuracy'] * 100,
        "Recall(%)": metrics['Recall'] * 100,
        "Precision(%)": metrics['Precision'] * 100,
        "F1-Score(%)": metrics['F1-Score'] * 100
    })
```

Figure 5.19 : Coding for display and store performance result

#### XI. Convert the results to a data frame and save it as an Excel file

- a) `results_df = pd.DataFrame(results)`: Convert the results list to a pandas data frame for easy saving.
- b) `results_df.to_excel('result.xlsx', index=False)`: Save the result data frame to an Excel file `result.xlsx` without saving the index column.
- c) `print("Results have been saved to result.xlsx")`: Print a message indicating that the results have been successfully saved to the Excel file.

```
results_df = pd.DataFrame(results)
results_df.to_excel('result.xlsx', index=False)
print("Results have been saved to result.xlsx")
```

Figure 5.20 : Coding for convert results into .xlsx file

### 5.2.3 Data Preprocessing

The main function of this Python code is to preprocess the data set, including the separation of data features and target variables, the division of data sets, etc.

#### I. Define data preprocessing function

- a) Define a function called `dataPreprocessing` to preprocess the input data set

```
def dataPreprocessing(data):
```

Figure 5.21 : `dataPreprocessing` function define

## II. Separation of features and target variables

- a) `x = data.iloc[:, data.columns != 'Class']`: Extract all features from the dataset (excluding the target variable Class column) and store them in the variable x.
  - i. `data.iloc[:, data.columns != 'Class']`: Use the `iloc` indexing method to select all data that is not equal to the Class column, which is the feature variable.
- b) `y = data.iloc[:, data.columns == 'Class'].values.ravel()`: Extract the data of the target variable (Class column) and convert it into a one-dimensional array and store it in the variable y.
  - i. `data.iloc[:, data.columns == 'Class']`: Select the data of the Class column.
  - ii. `.values.ravel()`: Convert the extracted Class column data from a two-dimensional form to a one-dimensional array form for subsequent processing.

```
x = data.iloc[:, data.columns != 'Class']  
y = data.iloc[:, data.columns == 'Class'].values.ravel()
```

Figure 5.22 : Coding for separation of feature and target variables

## III. Dataset division

- a) `train_test_split(x, y, test_size=0.3, random_state=0)`: Use the `train_test_split` function to split the dataset into training and test sets.
- b) `x`: Feature variable.
- c) `y`: Target variable.
- d) `test_size=0.3`: Specifies that the test set accounts for 30% of the entire dataset.
- e) `random_state=0`: Set the random seed to zero to ensure the repeatability of the results.
- f) `x_train, x_test, y_train, y_test`: Respectively represent the feature data of the training set, the feature data of the test set, the label data of the training set, and the label data of the test set.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

Figure 5.23 : Coding for dataset division

## IV. Return preprocessed data

- a) This line of code returns the training set and test set data to the caller, making it easier to use these data for model training and evaluation.

```
return x_train, x_test, y_train, y_test
```

Figure 5.24 : Coding for return preprocessed data

## 5.2.4 Feature Selection

This code defines 16 feature selection methods: Lasso, Random Projection, Chi-Square, Recursive Feature Elimination (RFE), Lasso Hybrid (3 in total), Random Projection Hybrid (3 in total), Chi-Square Hybrid (3 in total), RFE Hybrid (3 in total). Each function receives the feature data of the training set and the test set, and returns the selected feature data set based on the given list of the number of features.

### 5.2.4.1 Lasso

- i. `def lasso_feature_selection(x_train, x_test, y_train, max_features_list):` defines a function called `lasso_feature_selection` for feature selection using Lasso regression. The function takes four parameters: training set feature data `x_train`, test set feature data `x_test`, training set labels `y_train`, and a list `max_features_list` containing the number of features to be selected.
- ii. `scaler = StandardScaler():` creates a `StandardScaler` object that standardizes the data so that each feature has a mean of 0 and a variance of 1.
- iii. `x_train_scaled = scaler.fit_transform(x_train):` standardizes and transforms the training set feature data.
- iv. `x_test_scaled = scaler.transform(x_test):` transforms the test set feature data using the same standardizer (keeping the same standard).

```
def lasso_feature_selection(x_train, x_test, y_train, max_features_list):
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_train)
    x_test_scaled = scaler.transform(x_test)
```

Figure 5.25 : Coding for Lasso function define and data scale

- v. `x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_train.columns):` Convert the standardized training set data to `DataFrame` format and keep the original feature names.
- vi. `x_test_scaled = pd.DataFrame(x_test_scaled, columns=x_test.columns):` Convert the standardized test set data to `DataFrame` format and keep the original feature names.

```
x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_train.columns)
x_test_scaled = pd.DataFrame(x_test_scaled, columns=x_test.columns)
```

Figure 5.26 : Coding for convert standardized data into data frame

- vii. `results = {}:` Initialize an empty dictionary to store training set and test set data with different numbers of features.

```
results = {}
```

Figure 5.27 : Coding for results dictionary variable define

- viii. `sel_ = SelectFromModel(LogisticRegression(C=0.05, penalty='l1', solver='liblinear', random_state=10))`: Use a logistic regression model with L1 regularization for feature selection. L1 regularization imposes a penalty on the coefficients of some features, making the coefficients of some features become 0, thereby performing feature screening.
- ix. `sel_.fit(x_train_scaled, y_train)`: Train this model on the standardized training data.
- x. `all_selected_features = x_train_scaled.columns[(sel_.get_support())]`: Get the features selected by Lasso and store them as `all_selected_features`.

```
sel_ = SelectFromModel(LogisticRegression(C=0.05, penalty='l1', solver='liblinear',
random_state=10))
sel_.fit(x_train_scaled, y_train)
all_selected_features = x_train_scaled.columns[(sel_.get_support())]
```

Figure 5.28 : Coding for Lasso Model Setup and select feature

- xi. `for num_features in max_features_list`: Traverse the given feature list.
- xii. `selected_features = all_selected_features[:num_features]`: Select the first `num_features` features.
- xiii. `print(f'Lasso selected top {num_features} features: {selected_features.tolist()}')`:  
Print the first `num_features` features currently selected.
- xiv. `x_train_selected = x_train_scaled[selected_features]`: Extract the selected features in the training set.
- xv. `x_test_selected = x_test_scaled[selected_features]`: Extract the selected features in the test set.
- xvi. `results[num_features] = (x_train_selected, x_test_selected)`: Store the selected training and test set data in the results dictionary, with the key being the number of features and the value being the corresponding dataset.

```
for num_features in max_features_list:
    selected_features = all_selected_features[:num_features]
    print(f'Lasso selected top {num_features} features: {selected_features.tolist()}')
    x_train_selected = x_train_scaled[selected_features]
    x_test_selected = x_test_scaled[selected_features]
    results[num_features] = (x_train_selected, x_test_selected)
```

Figure 5.29 : Coding for select feature based on feature list(2,3,5,10,15)

- xvii. `return results`: Returns a dictionary containing training set and test set data with different numbers of features.

```
return results
```

Figure 5.30 : Coding for return results

#### Example Output :

```
Lasso Feature Selection Start
Lasso selected top 2 features: ['MI_dir_L0.1_weight', 'H_L0.1_weight']
Lasso selected top 3 features: ['MI_dir_L0.1_weight', 'H_L0.1_weight', 'HH_jit_L5_mean']
Lasso selected top 5 features: ['MI_dir_L0.1_weight', 'H_L0.1_weight', 'HH_L0.01_weight', 'HH_jit_L5_mean',
'HH_jit_L0.01_weight']
Lasso selected top 10 features: ['MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L0.1_weight', 'H_L0.01_weight',
'HH_L5_weight', 'HH_L0.01_weight', 'HH_L0.01_mean', 'HH_jit_L5_mean', 'HH_jit_L3_mean', 'HH_jit_L0.01_weight']
Lasso selected top 15 features: ['MI_dir_L5_weight', 'MI_dir_L1_weight', 'MI_dir_L0.1_weight', 'MI_dir_L0.01_weight',
'MI_dir_L0.01_mean', 'H_L0.1_weight', 'H_L0.01_weight', 'H_L0.01_mean', 'HH_L5_weight', 'HH_L0.01_weight',
'HH_L0.01_mean', 'HH_jit_L5_weight', 'HH_jit_L5_mean', 'HH_jit_L3_mean', 'HH_jit_L0.01_weight']
Lasso Feature Selection End
```

Figure 5.31 : Example Output for Lasso Feature Method

#### 5.2.4.2 Random Projection

- i. `def random_projection_feature_selection(x_train, x_test, y_train, max_features_list)`: defines a function called `random_projection_feature_selection`, which is used to select features by random projection. The parameters are the same as above.
- ii. `results = {}`: initialize an empty dictionary to store the results.
- iii. `for num_features in max_features_list`: traverse the given list of features.
- iv. `rp = random_projection.SparseRandomProjection(n_components=num_features)`: creates a `SparseRandomProjection` object, specifying the dimension after projection as `num_features`.
- v. `x_train_rp = rp.fit_transform(x_train)`: performs random projection on the training data and transforms it to low-dimensional data.
- vi. `x_test_rp = rp.transform(x_test)`: performs the same transformation on the test data.
- vii. `results[num_features] = (x_train_rp, x_test_rp)`: Store the converted training and test set data in the results dictionary.
- viii. `return results`: Returns the dictionary results containing the training and test set data with different numbers of features.

```

def random_projection_feature_selection(x_train, x_test, y_train, max_features_list):
    results = {}

    for num_features in max_features_list:
        rp = random_projection.SparseRandomProjection(n_components=num_features)
        x_train_rp = rp.fit_transform(x_train)
        x_test_rp = rp.transform(x_test)
        results[num_features] = (x_train_rp, x_test_rp)

    return results

```

Figure 5.32 : Coding for Random Projection Feature Selection Method

#### Example Output :

```

Random Projection Feature Selection Start
Random Projection Feature Selection End

```

Figure 5.33 : Example output for Random Projection Feature Selection Method

Random Projection can't display selected feature because it unlike feature selection methods such as recursive feature elimination (RFE) or Lasso regularization, Random Projection does not involve the process of selecting or screening features. Instead of selecting a part of the original feature set, it combines all the original features into new low-dimensional features in a random manner. The nature of this combination determines that it is not possible to directly identify which original features are retained or highlighted after projection(Johnson, 1984).

#### 5.2.4.3 Chi Square

- i. `def chi_square_feature_selection(x_train, x_test, y_train, max_features_list):` defines a function called `chi_square_feature_selection` for selecting features through chi-square test. The parameters are the same as above.
- ii. `results = {}`: initialize an empty dictionary to store the results.
- iii. `chi_ls = []`: initialize an empty list to store the p-value of each feature.
- iv. `for feature in x_train.columns:` loop through each feature in the training set.
- v. `c = pd.crosstab(y_train, x_train[feature])`: calculate the contingency table between the target variable `y_train` and the current feature `feature`.
- vi. `p_value = chi2_contingency(c)[1]`: perform a chi-square test, get the p-value, and store it in `p_value`.
- vii. `chi_ls.append(p_value)`: add the p-value of the current feature to the `chi_ls` list.



```

def chi_square_feature_selection(x_train, x_test, y_train, max_features_list):
    results = {}

    chi_ls = []
    for feature in x_train.columns:
        c = pd.crosstab(y_train, x_train[feature])
        p_value = chi2_contingency(c)[1]
        chi_ls.append(p_value)

```

Figure 5.34 : Coding for Chi-Square Filter function define and p\_value calculate

- viii. `all_selected_features = pd.Series(chi_ls, index=x_train.columns).sort_values(ascending=True).index`: Convert the p-value to a Series object with the index as the feature name. Sort by p-value in ascending order and get the feature name list `all_selected_features`.
- ix. `for num_features in max_features_list`: Traverse the given feature list.
- x. `selected_features = all_selected_features[:num_features]`: Select the first `num_features` features with the smallest p-value.
- xi. `print(f'Chi-Square selected top {num_features} features: {selected_features.tolist()}')`: Print the first `num_features` features currently selected.
- xii. `x_train_selected = x_train[selected_features]`: Extract the selected features in the training set.
- xiii. `x_test_selected = x_test[selected_features]`: Extract the selected features in the test set.
- xiv. `results[num_features] = (x_train_selected, x_test_selected)`: Store the selected training set and test set data in the results dictionary, with the key being the number of features and the value being the corresponding data set.
- xv. `return results`: Returns the dictionary results containing the training set and test set data with different numbers of features.

```

    all_selected_features = pd.Series(chi_ls,
index=x_train.columns).sort_values(ascending=True).index
    for num_features in max_features_list:
        selected_features = all_selected_features[:num_features]
        print(f"Chi-Square selected top {num_features} features: {selected_features.tolist()}")
        x_train_selected = x_train[selected_features]
        x_test_selected = x_test[selected_features]
        results[num_features] = (x_train_selected, x_test_selected)

return results

```

Figure 5.35 : Coding for select feature based on higher p\_value and feature list

## Example Output :

```
Chi-Square Feature Selection Start
Chi-Square selected top 2 features: ['MI_dir_L5_weight', 'MI_dir_L5_mean']
Chi-Square selected top 3 features: ['MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance']
Chi-Square selected top 5 features: ['MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance', 'MI_dir_L3_weight',
'MI_dir_L3_mean']
Chi-Square selected top 10 features: ['MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance', 'MI_dir_L3_weight',
'MI_dir_L3_mean', 'MI_dir_L3_variance', 'MI_dir_L1_weight', 'MI_dir_L1_mean', 'MI_dir_L1_variance', 'MI_dir_L0.1_weight']
Chi-Square selected top 15 features: ['MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance', 'MI_dir_L3_weight',
'MI_dir_L3_mean', 'MI_dir_L3_variance', 'MI_dir_L1_weight', 'MI_dir_L1_mean', 'MI_dir_L1_variance', 'MI_dir_L0.1_weight',
'MI_dir_L0.1_mean', 'MI_dir_L0.1_variance', 'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.01_variance']
Chi-Square Feature Selection End
```

Figure 5.36 : Example output for Chi-Square Feature Selection Method

### 5.2.4.4 Recursive Feature Elimination (RFE)

- i. `def rfe_feature_selection(x_train, x_test, y_train, max_features_list):` defines a function called `rfe_feature_selection` for selecting features through recursive feature elimination (RFE). The parameters are the same as above.
- ii. `results = {}:` initialize an empty dictionary to store the results.
- iii. `model = GradientBoostingClassifier(n_estimators=10, max_depth=4, random_state=10):` creates a gradient boosting classifier `GradientBoostingClassifier` for feature selection. The model has 10 trees (`n_estimators=10`) and the maximum depth of the tree is 4 (`max_depth=4`).

```
def rfe_feature_selection(x_train, x_test, y_train, max_features_list):
    results = {}
    model = GradientBoostingClassifier(n_estimators=10, max_depth=4, random_state=10)
```

Figure 5.37 : Coding for RFE function, results dictionary, model define

- iv. `max_features = max(max_features_list):` Get the maximum value in `max_features_list`, which is used to specify the maximum number of features to be selected during recursive feature elimination (RFE).
- v. `selector = RFE(model, n_features_to_select=max_features, step=10):` Create an RFE object. `model` is the base model used to evaluate feature importance, and `n_features_to_select=max_features` specifies the number of features that need to be selected in the end. `step=10` means that 10 features are eliminated (or the ranking of these features is reduced) in each recursion.
- vi. `selector.fit(x_train, y_train):` Train the RFE object using the training data `x_train` and the labels `y_train` to determine the importance and ranking of the features.

```
max_features = max(max_features_list)
```

```

selector = RFE(model, n_features_to_select=max_features, step=10)
selector.fit(x_train, y_train)

```

Figure 5.38 : Coding for select feature

- vii. `ranking = selector.ranking_`: Get the ranking of each feature. A feature with a value of 1 indicates that it is selected, and a feature with a larger value indicates that it is less important.
- viii. `selected_features_ordered = x_train.columns[np.argsort(ranking)]`: Sort the features according to their ranking order and return a list of sorted feature names. After sorting, the most important features are ranked first.

```

ranking = selector.ranking_
selected_features_ordered = x_train.columns[np.argsort(ranking)]

```

Figure 5.39 : Coding for ranking the feature selected

- ix. `for num_features in max_features_list`: traverse the given number of features in the list.
- x. `selected_features = selected_features_ordered[:num_features]`: select the top `num_features` features with the highest ranking.
- xi. `print(f'RFE selected top {num_features} features: {selected_features.tolist()}')`: print the names of the top `num_features` features currently selected.
- xii. `results[num_features] = (x_train[selected_features], x_test[selected_features])`: extract these selected features in the training set and test set, and store them as tuples in the results dictionary, with the key being the number of features and the value being the corresponding feature subset data.
- xiii. `return results`: returns a dictionary containing the training set and test set data under different numbers of features.

```

for num_features in max_features_list:
    selected_features = selected_features_ordered[:num_features]
    print(f'RFE selected top {num_features} features: {selected_features.tolist()}')
    results[num_features] = (x_train[selected_features], x_test[selected_features])

return results

```

Figure 5.40 : Coding for select top feature based on number feature list

## Example Output :

```
RFE Feature Selection Start
RFE selected top 2 features: ['MI_dir_L0.01_weight', 'MI_dir_L0.1_weight']
RFE selected top 3 features: ['MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight']
RFE selected top 5 features: ['MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L0.01_weight',
'HH_L0.01_magnitude']
RFE selected top 10 features: ['MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L0.01_weight',
'HH_L0.01_magnitude', 'HH_L0.01_radius', 'HH_L0.01_covariance', 'HH_L0.01_mean', 'HH_L0.1_mean',
'HH_jit_L0.01_variance']
RFE selected top 15 features: ['MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L0.01_weight',
'HH_L0.01_magnitude', 'HH_L0.01_radius', 'HH_L0.01_covariance', 'HH_L0.01_mean', 'HH_L0.1_mean',
'HH_jit_L0.01_variance', 'HpHp_L1_magnitude', 'HpHp_L0.01_radius', 'HH_jit_L0.01_mean', 'HH_L0.01_pcc',
'HH_jit_L3_variance']
RFE Feature Selection End
```

Figure 5.41 : Example output for RFE feature selection method

### 5.2.4.5 Hybrid Feature Selection Method

In lasso hybrid feature selection function, First, use StandardScaler to standardize the training and test data. The purpose of standardization is to scale the feature data to a range of mean 0 and standard deviation 1 to eliminate the impact of different dimensions between features.

```
def lasso_hybrid_feature_selection(x_train, x_test, y_train, max_features_list):
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_train)
    x_test_scaled = scaler.transform(x_test)
```

Figure 5.42 : Coding for Lasso Hybrid function define and data scale

After standardization, convert the standardized data back to DataFrame format and retain the original feature names. This will facilitate subsequent feature selection and operations.

```
x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_train.columns)
x_test_scaled = pd.DataFrame(x_test_scaled, columns=x_test.columns)
```

Figure 5.43 : Coding for convert data standardized into data frame

Initialize three dictionaries lasso\_rp\_results, lasso\_chi\_results, and lasso\_rfe\_results to save the results of different feature selection methods.

```
lasso_rp_results = {}
lasso_chi_results = {}
lasso_rfe_results = {}
```

Figure 5.44 : Coding for Results dictionaries define

Lasso regression (logistic regression with L1 regularization) is used for feature selection. SelectFromModel selects the most important features based on the model

weights (L1 regularization). The final selected features are stored in `all_selected_features_lasso` and used to filter the training and test data.

```

sel_ = SelectFromModel(LogisticRegression(C=0.05, penalty='l1', solver='liblinear',
random_state=10))
sel_.fit(x_train_scaled, y_train)
all_selected_features_lasso = x_train_scaled.columns[(sel_.get_support())]
x_train_selected_lasso = x_train_scaled[all_selected_features_lasso]
x_test_selected_lasso = x_test_scaled[all_selected_features_lasso]

```

Figure 5.45 : Coding for select feature by Lasso

The features selected by Lasso are further screened by the chi-square test. The chi-square test selects the most significant features by calculating the correlation between different categories and feature values. Finally, the features are sorted by p-value and the feature list `all_selected_features_chi` is obtained.

```

chi_ls = []
for feature in x_train_selected_lasso.columns:
    c = pd.crosstab(y_train, x_train_selected_lasso[feature])
    p_value = chi2_contingency(c)[1]
    chi_ls.append(p_value)
all_selected_features_chi = pd.Series(chi_ls,
index=x_train_selected_lasso.columns).sort_values(ascending=True).index

```

Figure 5.46 : Coding for select feature by Chi-Square

Use the recursive feature elimination (RFE) method for feature selection. First, build a base model (here is `GradientBoostingClassifier`), and then recursively remove the least important features until the required number of features `max_features` is reached. The ranking result of each feature is obtained through the `ranking_` attribute, and `selected_features_ordered` arranges the features in the order of selection.

```

model = GradientBoostingClassifier(n_estimators=10, max_depth=4, random_state=10)
max_features = max(max_features_list)
selector = RFE(model, n_features_to_select=max_features, step=10)
selector.fit(x_train_selected_lasso, y_train)
ranking = selector.ranking_
selected_features_ordered = x_train_selected_lasso.columns[np.argsort(ranking)]

```

Figure 5.47 : Coding for select feature by RFE

For each number of features `num_features` in the given `max_features_list`, do the following in order:

- I. Select features using Random Projection, select `num_features` features from the features filtered by Lasso, and save the results to `lasso_rp_results`.
- II. Select features using Chi-Square, select `num_features` features from the features filtered by Lasso, and save the results to `lasso_chi_results`.

III. Select features using Recursive Feature Elimination (RFE), select `num_features` features from the features filtered by Lasso, and save the results to `lasso_rfe_results`.

```

for num_features in max_features_list:
    # let random projection select feature from lasso
    rp = random_projection.SparseRandomProjection(n_components=num_features)
    x_train_rp = rp.fit_transform(x_train_selected_lasso)
    x_test_rp = rp.transform(x_test_selected_lasso)
    lasso_rp_results[num_features] = (x_train_rp, x_test_rp)
    # let chi square filter select top feature from itself
    selected_features_chi = all_selected_features_chi[:num_features]
    print(f"Lasso + Chi-Square selected top {num_features} features:
{selected_features_chi.tolist()}")
    x_train_selected_chi = x_train_selected_lasso[selected_features_chi]
    x_test_selected_chi = x_test_selected_lasso[selected_features_chi]
    lasso_chi_results[num_features] = (x_train_selected_chi, x_test_selected_chi)
    # let rfe select top feature from itself
    selected_features_rfe = selected_features_ordered[:num_features]
    print(f'Lasso + RFE selected top {num_features} features:
{selected_features_rfe.tolist()}')
    lasso_rfe_results[num_features] = (x_train_selected_lasso[selected_features_rfe],
x_test_selected_lasso[selected_features_rfe])

```

Figure 5.48 : Coding for Random Projection select feature and all feaute selection method list out feature selected based on number feature list

Finally, three dictionaries, `lasso_rp_results`, `lasso_chi_results` and `lasso_rfe_results`, are returned, which contain the feature selection results of different combination methods respectively.

```

return lasso_rp_results, lasso_chi_results, lasso_rfe_results

```

Figure 5.49 : Coding for return result

The remaining three functions, `random_projection_hybrid_feature_selection`, `chi_square_hybrid_feature_selection`, and `rfe_hybrid_feature_selection`, have similar structures, but the order of feature selection is different.

- `random_projection_hybrid_feature_selection`: first perform random projection, then Lasso, chi-square test, and RFE.
- `chi_square_hybrid_feature_selection`: first perform chi-square test, then Lasso, random projection, and RFE.
- `rfe_hybrid_feature_selection`: first perform RFE, then Lasso, random projection, and chi-square test.

## Example Output :

```
Lasso + Hybrid Feature Selection Start
Lasso + Chi-Square selected top 2 features: ['MI_dir_L5_weight', 'MI_dir_L5_mean']
Lasso + RFE selected top 2 features: ['MI_dir_L0.1_weight', 'MI_dir_L0.01_weight']
Lasso + Chi-Square selected top 3 features: ['MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance']
Lasso + RFE selected top 3 features: ['MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L5_mean']
Lasso + Chi-Square selected top 5 features: ['MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance',
'MI_dir_L3_variance', 'MI_dir_L1_weight']
Lasso + RFE selected top 5 features: ['MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L5_mean', 'H_L0.1_weight',
'H_L0.01_weight']
Lasso + Chi-Square selected top 10 features: ['MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance',
'MI_dir_L3_variance', 'MI_dir_L1_weight', 'MI_dir_L1_variance', 'MI_dir_L0.1_weight', 'MI_dir_L0.1_variance',
'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean']
Lasso + RFE selected top 10 features: ['MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L5_mean', 'H_L0.1_weight',
'H_L0.01_weight', 'HH_L0.01_std', 'HpHp_L0.1_std', 'HpHp_L0.1_weight', 'HH_jit_L0.1_variance', 'HpHp_L5_magnitude']
Lasso + Chi-Square selected top 15 features: ['MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance',
'MI_dir_L3_variance', 'MI_dir_L1_weight', 'MI_dir_L1_variance', 'MI_dir_L0.1_weight', 'MI_dir_L0.1_variance',
'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.01_variance', 'H_L5_weight', 'H_L5_mean', 'H_L5_variance',
'H_L3_variance']
Lasso + RFE selected top 15 features: ['MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L5_mean', 'H_L0.1_weight',
'H_L0.01_weight', 'HH_L0.01_std', 'HpHp_L0.1_std', 'HpHp_L0.1_weight', 'HH_jit_L0.1_variance', 'HpHp_L5_magnitude',
'HH_L0.01_radius', 'HH_L0.01_pcc', 'HH_L0.01_mean', 'HH_L0.1_std', 'HH_L0.1_pcc']
Lasso + Hybrid Feature Selection End
```

Figure 5.50 : Example output for Lasso Hybrid Feature Selection

### 5.2.4.6 Machine Learning Classifier and Performance Evaluation

The main function of this code is to train and evaluate different machine learning models and return the performance indicators of each model.

i. A function called `train_and_evaluate` is defined to train and evaluate the model. It receives four parameters:

- `x_train`: the feature data of the training set.
- `x_test`: the feature data of the test set.
- `y_train`: the label data (target variable) of the training set.
- `y_test`: the label data (target variable) of the test set.

ii. A dictionary called `models` is defined, which contains four different machine learning models. Each model is associated with a string name:

- Decision Tree: decision tree model, using the `DecisionTreeClassifier` class.
- Random Forest: random forest model, using the `RandomForestClassifier` class.
- KNN: K nearest neighbor model, using the `KNeighborsClassifier` class.

- Linear SVC: linear support vector machine model, using the LinearSVC class.
- iii. The `random_state=0` parameter is used to ensure that the results of the model are the same each time it is run, that is, to set the random number seed for easy reproduction of the results.

```
def train_and_evaluate(x_train, x_test, y_train, y_test):
    models = {
        'Decision Tree': DecisionTreeClassifier(random_state=0),
        'Random Forest': RandomForestClassifier(random_state=0),
        'KNN': KNeighborsClassifier(),
        'Linear SVC': LinearSVC(random_state=0),
    }
```

Figure 5.51 : Coding for train and evaluate function define and models define

- iv. An empty dictionary performance is created to store the performance evaluation results of each model.
- v. Use a for loop to iterate over each model in the models dictionary. `model_name` is the name of the model, and `model` is the corresponding model object.
- vi. Output the name of the model currently being trained and evaluated in the console so that the user knows the progress of the program.
- vii. Call the fit method to fit the model to the training data, that is, train the model to learn the relationship between the features and labels in the training data.
- viii. Use the predict method to use the trained model to predict the data of the test set and get the predicted label `y_pred`.

```
performance = {}

for model_name, model in models.items():
    print(model_name + " Start")
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
```

Figure 5.52 : Coding for performance dictionary define and looping for machine learning algorithm execute

- ix. Use different evaluation metrics to measure the performance of the model:
- x. accuracy: Accuracy, which indicates the proportion of correctly classified by the model, calculated using the `accuracy_score` function.
- xi. recall: Recall, which indicates the proportion of all positive samples that the model correctly identified, calculated using the `recall_score` function.



xii. precision: Precision, which indicates the proportion of samples predicted by the model as positive that are actually positive, calculated using the `precision_score` function.

xiii. f1: F1 score, which is the harmonic mean of precision and recall, calculated using the `f1_score` function.

```
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

Figure 5.53 : Coding for calculate and performance results

xiv. Store the evaluation results of the current model in the performance dictionary. The key of the dictionary is the model name, and the value is a sub-dictionary containing various evaluation metrics.

xv. Output The training and evaluation process of the current model has been completed.

xvi. The function returns the performance dictionary, which contains the performance evaluation results of all models.

```
performance[model_name] = {
    'Accuracy': accuracy,
    'Recall': recall,
    'Precision': precision,
    'F1-Score': f1
}
print(model_name + " Done")

return performance
```

Figure 5.54 : Coding for store performance results and return

## Example Output :

```
Train and test model with Random Projection-selected features Start for 2 features
Decision Tree Start
Decision Tree Done
Random Forest Start
Random Forest Done
KNN Start
KNN Done
Linear SVC Start
Linear SVC Done
Train and test model with Random Projection-selected features End for 2 features

Performance with Random Projection feature selection (2 features):
Decision Tree: {'Accuracy': 0.995746612912513, 'Recall': np.float64(0.9986523912369563), 'Precision':
np.float64(0.9967361458802664), 'F1-Score': np.float64(0.9976933484379935)}
Random Forest: {'Accuracy': 0.9913771213838894, 'Recall': np.float64(0.9959141322142544), 'Precision':
np.float64(0.9947306117372859), 'F1-Score': np.float64(0.9953220201498797)}
KNN: {'Accuracy': 0.9935613951789283, 'Recall': np.float64(0.9976685856000573), 'Precision':
np.float64(0.9953520875476127), 'F1-Score': np.float64(0.9965089903351083)}
Linear SVC: {'Accuracy': 0.9217715649840332, 'Recall': np.float64(0.999943636135386), 'Precision':
np.float64(0.9217626281567439), 'F1-Score': np.float64(0.9592628084524468)}
```

Figure 5.55 : Example Output for performance of Random Project Feature Selection in all machine learning method

## 5.3 Conclusion

In summary, this chapter detailed the step-by-step process of developing an IoT Botnet Detection in Home IoT Environment using Machine Learning. The entire process began with Kaggle data collection, followed by comprehensive data preprocessing, feature selection, training and testing of machine learning classifiers, and finally a comprehensive performance evaluation. Throughout the implementation, the accuracy of sixteen classifiers was measured, which will serve as a valuable basis for comparison with existing literature in the next chapter. This chapter also showed how the system development of this project was successfully completed, providing a framework for further research. Detail coding and output already put in appendix.

## CHAPTER 06 : TESTING

### 6.1 Introduction

This section will explain the results of project tests the system's ability to accurately detect Botnet from Home IoT Environment. This section also determines whether the top feature selection method and top machine learning classifiers Model.

### 6.2 TOP Feature Selection Method and Machine Learning Classifiers Model

Table 6.1: Top 3 Results with highest accuracy in each feature selected situation

Number of Features Selected	Model	Feature Selection Method	Accuracy (%)	Recall (%)	Precision (%)	F1-Score (%)
2	Decision Tree	Chi-Square + RFE	99.9934	99.9986	99.9942	99.9964
2	Decision Tree	RFE + Chi-Square	99.9883	99.9975	99.9899	99.9937
2	Decision Tree	RFE	99.9883	99.9975	99.9899	99.9937
3	Decision Tree	Chi-Square + RFE	99.9973	99.9994	99.9976	99.9985
3	Decision Tree	Lasso + RFE	99.9947	99.9988	99.9954	99.9971
3	Decision Tree	Chi-Square + Random Projection	99.9933	99.9988	99.9939	99.9964
5	Decision Tree	Chi-Square + RFE	99.9973	99.9994	99.9976	99.9985
5	Decision Tree	Random Projection	99.9972	99.9995	99.9975	99.9985
5	Decision Tree	RFE	99.9969	99.9991	99.9975	99.9983
10	Decision Tree	Lasso	99.9994	99.9998	99.9995	99.9997
10	Decision Tree	Random Projection	99.9991	99.9999	99.9991	99.9995
10	Decision Tree	Random Projection + RFE	99.9990	99.9997	99.9992	99.9995
15	Decision Tree	RFE	99.9999	99.9999	99.9999	99.9999
15	Decision Tree	Lasso	99.9994	99.9999	99.9995	99.9997
15	Decision Tree	Random Projection + RFE	99.9991	100	99.9990	99.9995

#### I. 2-feature selection case

- a) In the case of 2-feature selection, Chi-Square + Recursive Feature Elimination(RFE) method performs best, achieving 99.9934% accuracy and 99.9986% recall. This shows that this method is very effective in selecting the two most discriminative features from the original data, ensuring efficient classification performance.

- b) Recursive Feature Elimination(RFE) + Chi-Square method performs closely behind, with an accuracy of 99.9883%. Although slightly lower than Chi-Square + Recursive Feature Elimination(RFE), its recall is still close to perfect, reaching 99.9975%. This shows that this method can also effectively select key features and has a slightly different processing order, but still maintains excellent classification results.
- c) Recursive Feature Elimination(RFE) method alone also achieved a high accuracy of 99.9883% and a recall of 99.9975%. This shows that RFE, as a feature selection technique, can well identify effective feature combinations with a small number of features and ensure the excellent performance of the model.

## II. 3 feature selection cases

- a) When the number of features increases to 3, the Chi-Square + Recursive Feature Elimination(RFE) method continues to maintain its superior performance with an accuracy of 99.9973%. This result shows that with one more feature, the method can still robustly select the most meaningful features to ensure the efficiency of the model.
- b) The Lasso + Recursive Feature Elimination(RFE) method also performs very well with an accuracy of 99.9947%. By combining Lasso regularization with recursive feature elimination, this method has strong robustness in feature selection, especially when dealing with fewer features, it can still maintain a high level of classification accuracy.
- c) The Chi-Square + Random Projection method performs slightly worse than the previous two when selecting 3 features, but still has a high accuracy of 99.9933%. This shows that by combining statistical screening and dimensionality reduction techniques, this method can find important features and effectively compress the feature space, thereby maintaining high performance.

## III. 5 feature selection case

- a) When selecting 5 features, the Chi-Square + Recursive Feature Elimination(RFE) method continues to show its stability and excellent performance, maintaining a high accuracy of 99.9973%. This shows that this

method can still ensure the efficiency and accuracy of the model in the case of more features.

- b) The performance of the Random Projection method alone is also very close, with an accuracy of 99.9972%. This shows that this dimensionality reduction method can effectively reduce dimensions in the case of multiple features while maintaining the integrity of information, thereby maintaining a high level of classification performance.
- c) Recursive Feature Elimination(RFE) still performs well with 5 feature selections, with an accuracy of 99.9969%. Although slightly lower than the first two, its recall and F1-Score are still very high, indicating that this method still has good selection ability when dealing with complex features.

#### IV. 10 feature selections

- a) When the number of features increases to 10, the Lasso method performs well with an accuracy of 99.9994%. This performance shows that Lasso can effectively prevent overfitting when dealing with high-dimensional data while maintaining high accuracy and robustness of the model.
- b) Random Projection still performs well with 10 feature selections, with an accuracy of 99.9991%. This shows that when dealing with more features, this method can effectively compress high-dimensional space to low-dimensional space while retaining key information to ensure classification performance.
- c) Combining Random Projection with Recursive Feature Elimination(RFE) also achieves a very high accuracy (99.9990%). This shows that combining dimensionality reduction technology with feature elimination technology can further optimize the effect of feature selection, thereby improving model performance.

#### V. 15 feature selection cases

- a) When the number of features increases to 15, the Recursive Feature Elimination(RFE) method shows a near-perfect classification ability with an accuracy of 99.9999%. This shows that RFE can still select the most valuable features when dealing with a large number of features, ensuring the top performance of the classifier.

- b) The Lasso method continues to perform well with an accuracy of 99.9994% when 15 features are selected. This further proves the advantage of Lasso in high-dimensional feature selection, which can effectively improve the robustness of the model and prevent overfitting.
- c) The method combining Random Projection and Recursive Feature Elimination(RFE) also performed well when selecting 15 features, with an accuracy of 99.9991%. This shows that the combined method can effectively extract key features when dealing with large-scale feature selection tasks while ensuring the efficiency and accuracy of the model.

Based on table 6.1, researcher ensure that Decision Tree is the best machine learning classifiers model compare with Linear SVC, k-Nearest Neighbour(KNN), and Random Forest.

### 6.3 TOP Feature

Table 6.2: Feature Selected by each Top Feature Selection Method

Number of feature selected	Feature Selection Method with highest accuracy	Feature Selected
2	Chi-Square + RFE	'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean'
3	Chi-Square + RFE	'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.1_weight'
5	Chi-Square + RFE	'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.1_weight', 'H_L0.01_weight', 'H_L0.01_mean'
10	Lasso	'MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L5_weight', 'HH_L0.01_weight', 'HH_L0.01_mean', 'HH_jit_L5_mean', 'HH_jit_L3_mean', 'HH_jit_L0.01_weight'
15	RFE	'MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L0.01_magnitude', 'HH_L0.01_radius', 'HH_L0.01_covariance', 'HH_L0.01_mean', 'HH_L0.1_mean', 'HH_jit_L0.01_variance', 'HpHp_L1_magnitude', 'HpHp_L0.01_radius', 'HH_jit_L0.01_mean', 'HH_L0.01_pcc', 'HH_jit_L3_variance'

Based on table 6.2, the top 5 feature in N\_BaIoT Dataset is 'MI\_dir\_L0.01\_weight', 'MI\_dir\_L0.1\_weight', 'H\_L0.01\_weight', 'HH\_L0.01\_mean', 'H\_L0.1\_weight'.

- i. MI\_dir\_L0.01\_weight, This feature represents the weight of the packet flow from a specified host (based on IP and MAC addresses) in a very short time (100ms). The weight of the flow can be regarded as the number of packets observed in this time window, so this feature captures the intensity of network activity of the source host in a very short time.
- ii. MI\_dir\_L0.1\_weight, This feature represents the weight of the packet flow from a specified host (based on IP and MAC addresses) in a slightly longer short time (500ms). Compared with L0.01, this feature captures the intensity of network activity of the source host in a slightly longer window.
- iii. H\_L0.01\_weight, This feature reflects the weight of network traffic from a specific IP address in a very short time (100ms). This feature captures the traffic activity of a specified host (based on IP address) in a very short time range, representing the network load during this period.
- iv. HH\_L0.01\_mean, Explanation: This feature represents the mean value of the traffic from a specific host (source IP) to the destination host in a very short time (100ms). This feature measures the average level of network communication strength from the source host to the destination host in a short time window.
- v. H\_L0.1\_weight, This feature represents the weight of network traffic from a specific IP address in a shorter time (500ms). This reflects the strength and frequency of network traffic generated by a specified host (based on IP address) in this shorter time range.

#### 6.4 Comparison with the existing works

Table 6.3: Comparison with the existing works

References	Feature Selection Method	Number of feature selected	Best Model	Accuracy
Bahsi et al., 2018	Fisher's Score	10	Decision Tree	98.97%
Proposed	Chi-Square + RFE	2	Decision Tree	99.9934%

Proposed	Chi-Square + RFE	3	Decision Tree	99.9973%
Proposed	Chi-Square + RFE	5	Decision Tree	99.9973%
Proposed	Lasso	10	Decision Tree	99.9994%
Proposed	RFE	15	Decision Tree	99.9999%

Table 6.3 shows the comparison between the proposed project and the existing work. According to this table, the proposed project achieved higher accuracy which is minimum 99.99% in all number of feature selected using same machine learning classifiers model.

From Table 6.3, also know about when 15 features are selected, the performance is better than when 10, 5, 3, or 2 features are selected. There are several main reasons:

#### I. Feature diversity and increased information

- a) As the number of features increases, the model can access more information.

The goal of feature selection is to remove redundant or irrelevant features while retaining features that have predictive power for the target variable.

When the number of features selected is 15, it may cover a wider range of patterns and correlations in the data, allowing the model to better understand the complex relationships in the data. Therefore, compared to selecting fewer features (such as 2 or 3 features), using 15 features can provide more information to help the model make more accurate predictions.

#### II. Avoid underfitting

- a) Too few features may cause the model to fail to capture key patterns in the data, resulting in insufficient performance of the model. This situation is often referred to as "underfitting". When there are only 2 or 3 features, the model may not be able to fully describe the complex relationships in the data, resulting in degraded performance. 15 features can provide enough information to reduce the risk of underfitting, thereby making the model perform better.

#### III. Capture complex feature interactions

- a) Some complex classification problems, especially tasks like Botnet detection, may involve complex interactions between multiple features. A small



number of features may not be able to fully capture these interaction effects. Selecting 15 features allows the model to better capture these interactions, thereby improving the model's classification accuracy and overall performance.

#### IV. Balance of feature selection

- a) During the feature selection process, it is necessary to find a balance between retaining enough information and avoiding noise features. According to the results of this project, 15 features seem to be an ideal balance point, which avoids insufficient information due to too few features and too much noise due to too many features. This balance enables the model to achieve the best classification performance.

### 6.5 Conclusion

In the analysis of feature selection and classification performance, different feature selection methods were used and tested for different numbers of features (2, 3, 5, 10, and 15). In the case of 2 feature selection, the Chi-Square + RFE method performed best, achieving an accuracy of 99.9934%, while the RFE + Chi-Square and RFE methods followed closely with an accuracy of 99.9883%. When the number of features increased to 3, Chi-Square + RFE continued to perform well, achieving an accuracy of 99.9973%, while Lasso + RFE and Chi-Square + Random Projection methods achieved high accuracies of 99.9947% and 99.9933%, respectively. As the number of features increased to 5, Chi-Square + RFE and Random Projection methods achieved high accuracies of 99.9973% and 99.9972%, respectively, while the RFE method also performed well with an accuracy of 99.9969%. When the number of features increased to 10, the Lasso method performed excellently, with an accuracy of 99.9994%, while the Random Projection and Random Projection + RFE methods achieved accuracies of 99.9991% and 99.9990%, respectively. Finally, with 15 features selected, the RFE method performed best with a near-perfect accuracy of 99.9999%, while the Lasso and Random Projection + RFE methods achieved high accuracies of 99.9994% and 99.9991%, respectively. These results show that under different numbers of features, various methods have demonstrated excellent feature selection capabilities and classification performance, especially RFE, Chi-Square + RFE, and Lasso methods, which are stable under various numbers of features,

ensuring the efficiency and accuracy of the model. The best classifier is undoubtedly the Decision Tree. Full results already put in appendix.

## CHAPTER 07 : CONCLUSION

### 7.1 Introduction

This chapter provides a overview of the project, compare the achievement with the objective and potential future works.

### 7.2 Project Achievement

Project objective is identify the best feature selection method, develop and implement machine learning classifiers, and select the best classifiers with higher accuracy in detecting Botnet in home IoT environment. Figure 7.1 list out the top 10 result get from this project.

Table 7.1: TOP 10 Result List

Number of Features Selected	Model	Feature Selection Method	Accuracy (%)	Recall (%)	Precision (%)	F1-Score (%)
15	Decision Tree	RFE	99.9999	99.9999	99.9999	99.9999
10	Decision Tree	Lasso	99.9994	99.9998	99.9995	99.9997
15	Decision Tree	Lasso	99.9994	99.9999	99.9995	99.9997
10	Decision Tree	Random Projection	99.9991	99.9999	99.9991	99.9995
15	Decision Tree	Random Projection + RFE	99.9991	100	99.9990	99.9995
15	Decision Tree	RFE + Random Projection	99.9991	99.9999	99.9990	99.9995
10	Decision Tree	Random Projection + RFE	99.9990	99.9997	99.9992	99.9995
15	Decision Tree	Random Projection	99.9989	99.9997	99.9991	99.9994
10	Decision Tree	Random Projection + Lasso	99.9982	99.9997	99.9983	99.9990
15	Decision Tree	Lasso + Chi-Square	99.9981	99.9997	99.9982	99.9989

## 7.2.1 Best Feature Selection Method

Based on table 7.1, Recursive Feature Elimination (RFE) has proven to be the best feature selection method with its almost perfect accuracy and F1 score.

```
def rfe_feature_selection(x_train, x_test, y_train, max_features_list):
    results = {}

    # Build initial model
    model = GradientBoostingClassifier(n_estimators=10, max_depth=4, random_state=10)

    # Use RFE select feature
    max_features = max(max_features_list)
    selector = RFE(model, n_features_to_select=max_features, step=10)
    selector.fit(x_train, y_train)

    # Get the ranking of features
    ranking = selector.ranking_
    selected_features_ordered = x_train.columns[np.argsort(ranking)]

    # Get and show selected features for each required number of features
    for num_features in max_features_list:
        selected_features = selected_features_ordered[:num_features]
        print(f'RFE selected top {num_features} features: {selected_features.tolist()}')
        results[num_features] = (x_train[selected_features], x_test[selected_features])

    return results
```

Figure 7.1 : Coding for RFE feature selection

In the above code, Recursive Feature Elimination (RFE) is used as a feature selection method to iteratively train the model and gradually eliminate unimportant features to finally select the most valuable features. In the case of selecting 15 features, RFE method proves to be the best feature selection method for the following reasons:

- i. Gradient Boosting Classifier model is gradually eliminated by RFE, eliminating the least important features in each iteration. This process continues until the required number of features are left. In this process, RFE can accurately identify which features are critical to the model's predictive ability.
- ii. Combined with Gradient Boosting Model, RFE is combined with Gradient Boosting Classifier as the base model. This combination can fully utilize the advantages of the gradient boosting model, allowing RFE to effectively evaluate the importance of features at each step. The gradient boosting model can provide

a more accurate feature importance assessment when dealing with high-dimensional data due to its powerful learning ability.

- iii. Excellent performance of selecting 15 features, when RFE finally selects 15 features, these features are considered to be the most helpful for the classification task. By sorting the features using `np.argsort(ranking)` and selecting the top 15 features, the RFE method ensures that the combination of these features can provide the best classification performance for the model. This is why in the evaluation results, the model performance reached a near-perfect level when the 15 features selected by RFE were used.
- iv. Balancing performance and complexity when the number of features reaches 15, RFE can not only ensure the high performance of the model, but also effectively control the complexity of the model. Compared with using more or fewer features, this combination of 15 features shows the best balance between performance and complexity, and is therefore considered the best choice.

In summary, RFE combines the gradient boosting model to gradually select the most important features in multiple rounds of iterations. The 15 features finally selected perform well in the classification task, making RFE the best feature selection method in this case.

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

## 7.2.2 Develop and Implement Machine Learning Classifiers

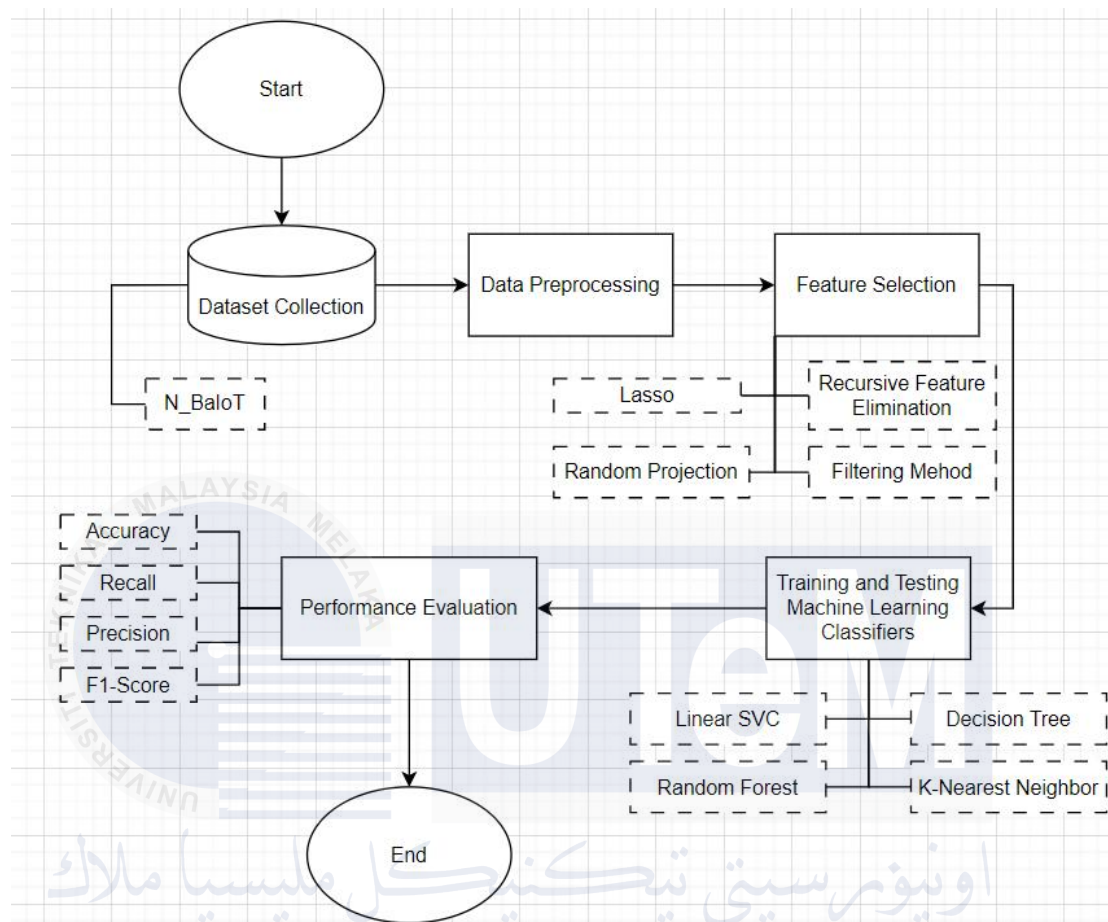


Figure 7.2 : Data Structure for Botnet Detection System

This project completed the development and implementation of machine learning classifiers through the following steps:

### I. Dataset Collection:

- First, the N-BaIoT dataset was selected as the basic data for analysis. This step is the starting point of the project to ensure that the project has enough data to train and test the model.

### II. Data Preprocessing:

- After the dataset is collected, the data is preprocessed to ensure that the data is suitable for training machine learning models.

### III. Feature Selection:

- In the feature selection process, a variety of methods are used, including Lasso, Recursive Feature Elimination (RFE) and dimensionality reduction

methods (such as random projection) and their mixed use, a total of 16 feature selection methods. These methods help the project extract the most representative features from the data, thereby improving the training efficiency and prediction accuracy of the model.

#### IV. Training and Testing Machine Learning Classifiers:

- After selecting features, the project used a variety of machine learning models to train and test the data, including Linear SVC, Decision Tree, Random Forest, and K-Nearest Neighbor. These models represent different algorithmic ideas and can process and analyze data from different angles.

#### V. Performance Evaluation:

- Finally, the performance of each model was evaluated, using indicators such as Accuracy, Recall, Precision, and F1-Score to quantify the classification effect of the model. These evaluation indicators help people understand the performance of the model in practical applications.

Through these steps, the project successfully developed and implemented multiple machine learning classifiers.

### 7.2.3 Best Machine Learning Classifiers

Based on table 7.1, the top 10 result all using Decision Tree Classifiers Model and this situation has proven Decision Tree is the best Machine Learning Classifiers Model. The reasons why Decision Tree is considered the best classifier can be summarized as follows:

#### i. Handling complex feature interactions:

- Decision trees can effectively handle complex interactions between features. In the N\_BaIoT dataset, there are nonlinear relationships between features, and decision trees can capture these complex interactions by recursively splitting data, thereby improving classification accuracy.

#### ii. Efficient handling of class imbalance:

- In the N\_BaIoT dataset, there is a class imbalance problem between normal traffic and Botnet traffic. Decision trees can adapt to this imbalance by

adjusting the node splitting criteria, thereby maintaining high efficiency in identifying the minority class (normal traffic).

iii. Less parameter tuning requirements:

- Compared to other complex machine learning models, such as random forests or support vector machines, decision trees are less sensitive to hyperparameters and do not require a lot of parameter tuning. This makes the training and application of decision tree models on datasets simpler and more straightforward.

In summary, in Botnet Detection System project, the decision tree model has become the best performing classifier through its ability to handle complex feature interactions, adaptability to class imbalance, and less parameter tuning requirements.

### **7.3 Future Work**

As for the direction of future work, can consider developing a real-time detection system, deploying the model in a real environment, and building a system that can analyze network traffic and detect Botnet activities in real time. This can achieve immediate threat response and protect network security. The system can also be embedded in a firewall or IDS for use in conjunction with it, and a user-friendly interface can be developed to build an easy-to-use user interface to facilitate security personnel to monitor and analyze Botnet activities. The interface can include real-time traffic monitoring, detection reports, and visualization tools.

### **7.4 Conclusion**

Through the implementation of this project, multiple machine learning classifiers were successfully developed and implemented, and the best classifier for detecting Botnet traffic in a home IoT environment was determined through a comprehensive performance evaluation. In terms of feature selection, the combination of the RFE method and the gradient boosting model demonstrated its strong ability in high-dimensional data processing, especially when selecting 15 key features, which can significantly improve the accuracy and prediction performance of the model. In the selection of classifiers, the decision tree model has become the best choice for

detecting Botnet traffic due to its excellent performance, less parameter tuning requirements, and effective processing of complex feature interactions.

Future work will focus on developing a real-time detection system to deploy the model in a real environment to achieve real-time monitoring of network traffic and rapid detection of Botnet activities. At the same time, by building a user-friendly interface, security personnel can monitor and analyze network security threats more efficiently. These further research and development work will help improve the practicality and scalability of the system, so that it can maintain efficient threat detection capabilities in a diverse network environment.





## Reference

- Abbasi, F., Naderan, M., & Alavi, S. E. (2021, May 1). *Anomaly detection in Internet of Things using feature selection and classification based on Logistic Regression and Artificial Neural Network on N-BaloT dataset*. IEEE Xplore.  
<https://doi.org/10.1109/loT52625.2021.9469605>
- Abbasi, F., Naderan, M., & Alavi, S. E. (2021, May 1). *Anomaly detection in Internet of Things using feature selection and classification based on Logistic Regression and Artificial Neural Network on N-BaloT dataset*. IEEE Xplore.  
<https://doi.org/10.1109/loT52625.2021.9469605>
- Akanksha, E., Jyoti, Sharma, N., & Gulati, K. (2021, April 1). *Review on Reinforcement Learning, Research Evolution and Scope of Application*. IEEE Xplore.  
<https://doi.org/10.1109/ICCMC51019.2021.9418283>
- Alzahrani, H., Abulkhair, M., & Alkayal, E. (2020). A Multi-Class Neural Network Model for Rapid Detection of IoT Botnet Attacks. *International Journal of Advanced Computer Science and Applications*, 11(7). <https://doi.org/10.14569/ijacsa.2020.0110783>
- Amruthnath, N., & Gupta, T. (2018). A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance. *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*.  
<https://doi.org/10.1109/iea.2018.8387124>
- Arce, I., & Levy, E. (2003). An analysis of the Slapper worm. *IEEE Security & Privacy*, 1(1), 82–87.  
<https://doi.org/10.1109/msecp.2003.1177002>
- Ashari, Z. E., & Broschat, S. L. (2019, November 1). *t-Tree and t-Forest: Decision Tree and Random Forest Algorithms Including the Relevance Factor with Applications in Bioinformatics*. IEEE Xplore. <https://doi.org/10.1109/BIBM47256.2019.8983065>
- Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, 54(15), 2787–2805. <https://doi.org/10.1016/j.comnet.2010.05.010>
- Bahsi, H., Nomm, S., & La Torre, F. B. (2018). Dimensionality Reduction for Machine Learning Based IoT Botnet Detection. *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. <https://doi.org/10.1109/icarcv.2018.8581205>
- Bandyopadhyay, D., & Sen, J. (2011). Internet of Things: Applications and Challenges in Technology and Standardization. *Wireless Personal Communications*, 58(1), 49–69.  
<https://doi.org/10.1007/s11277-011-0288-5>

- Benkessirat, A., & Benblidia, N. (2019). Fundamentals of Feature Selection: An Overview and Comparison. *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*. <https://doi.org/10.1109/aiccsa47632.2019.9035281>
- Bertino, E., & Islam, N. (2017). Botnets and Internet of Things Security. *Computer*, *50*(2), 76–79. <https://doi.org/10.1109/mc.2017.62>
- Brush, K., & Silverthorne, V. (2022, November). *What is Agile Software Development (Agile Methodologies)?* TechTarget. <https://www.techtarget.com/searchsoftwarequality/definition/agile-software-development>
- Catillo, M., Pecchia, A., & Villano, U. (2023). A Deep Learning Method for Lightweight and Cross-Device IoT Botnet Detection. *Applied Sciences*, *13*(2), 837. <https://doi.org/10.3390/app13020837>
- Catillo, M., Pecchia, A., & Villano, U. (2023). A Deep Learning Method for Lightweight and Cross-Device IoT Botnet Detection. *Applied Sciences*, *13*(2), 837. <https://doi.org/10.3390/app13020837>
- Chaudhari, A. R., & Joshi, S. D. (2021). Study of effect of Agile software development Methodology on Software Development Process. *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*. <https://doi.org/10.1109/icesc51422.2021.9532842>
- Chen, X., & Jeong, J. C. (2007). Enhanced recursive feature elimination. *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*. <https://doi.org/10.1109/icmla.2007.35>
- Cherrington, M., Thabtah, F., Lu, J., & Xu, Q. (2019, April 1). *Feature Selection: Filter Methods Performance Challenges*. IEEE Xplore. <https://doi.org/10.1109/ICCISci.2019.8716478>
- Citronnelle2. (2015, September 5). *The working principle and defense of botnets* [Review of *The working principle and defense of botnets*]. Blog.csdn.net. <https://blog.csdn.net/zhouwei1221q/article/details/48222917>
- Cloudflare. (2023). What is the Mirai Botnet? | Cloudflare. *Cloudflare*. <https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/>
- Cut Alna Fadhillah, Muhammad Dany Alfikri, & Kaliski, R. (2023). Lightweight Meta-Learning BotNet Attack Detection. *IEEE Internet of Things Journal*, *10*(10), 8455–8466. <https://doi.org/10.1109/jiot.2022.3229463>

- Dange, S., & Chatterjee, M. (2020). IoT Botnet: The Largest Threat to the IoT Network [Review of IoT Botnet: The Largest Threat to the IoT Network]. *Data Communication and Networks*, pp.137-157. [https://doi.org/10.1007/978-981-15-0132-6\\_10](https://doi.org/10.1007/978-981-15-0132-6_10)
- Dhalaria, M., & Gandotra, E. (2020, November 1). *Android Malware Detection using Chi-Square Feature Selection and Ensemble Learning Method*. IEEE Xplore. <https://doi.org/10.1109/PDGC50313.2020.9315818>
- Dixit, R., Kushwah, R., & Pashine, S. (2020). Handwritten Digit Recognition using Machine and Deep Learning Algorithms. *International Journal of Computer Applications*, 176(42), 27–33. <https://doi.org/10.5120/ijca2020920550>
- Dutta, V., Choraś, M., Pawlicki, M., & Kozik, R. (2020). A Deep Learning Ensemble for Network Anomaly and Cyber-Attack Detection. *Sensors*, 20(16), 4583. <https://doi.org/10.3390/s20164583>
- Foote, K. (2022, January 14). *A Brief History of the Internet of Things - DATAVERSITY*. DATAVERSITY. <https://www.dataversity.net/brief-history-internet-things/>
- Garg, U., Kaushik, V., Panwar, A., & Gupta, N. (2021, May 1). *Analysis of Machine Learning Algorithms for IoT Botnet*. IEEE Xplore. <https://doi.org/10.1109/INCET51464.2021.9456246>
- Gillis, A. (2023, August 1). *What Is IoT (Internet of Things) and How Does It Work?* TechTarget. <https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT>
- Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), 1645–1660. <https://doi.org/10.1016/j.future.2013.01.010>
- Guerra-Manzanares, A., Hayretin Bahsi, & Sven Nömm. (2019). Hybrid Feature Selection Models for Machine Learning Based Botnet Detection in IoT Networks. *Cyberworlds*. <https://doi.org/10.1109/cw.2019.00059>
- Guiding Opinions of the State Council on Actively Promoting the “Internet +” Action\_Information Industry (Including Telecommunications)\_China Government Network. (n.d.). [https://www.gov.cn/zhengce/content/2015-07/04/content\\_10002.htm](https://www.gov.cn/zhengce/content/2015-07/04/content_10002.htm)
- Gupta, V., Mishra, V. K., Singhal, P., & Kumar, A. (2022, December 1). *An Overview of Supervised Machine Learning Algorithm*. IEEE Xplore. <https://doi.org/10.1109/SMART55829.2022.10047618>

- H Esha, Hadimani, B. S., Devika, S. P., Shanthala, P. T., & R Bhavana. (2023). *IoT Botnet Creation and Detection using Machine Learning*.  
<https://doi.org/10.1109/incacct57535.2023.10141717>
- Heidari, M., Lakshmivarahan, S., Mirniaharikandehei, S., Danala, G., Maryada, S. K. R., Liu, H., & Zheng, B. (2021). Applying a Random Projection Algorithm to Optimize Machine Learning Model for Breast Lesion Classification. *IEEE Transactions on Biomedical Engineering*, 68(9), 2764–2775. <https://doi.org/10.1109/tbme.2021.3054248>
- Hsu, H.-H., Hsieh, C.-W., & Lu, M.-D. (2011). Hybrid feature selection by combining filters and wrappers. *Expert Systems with Applications*, 38(7), 8144–8150.  
<https://doi.org/10.1016/j.eswa.2010.12.156>
- IBM. (2024). *What is the internet of things?* Wwww.ibm.com.  
<https://www.ibm.com/topics/internet-of-things>
- Importance of Datasets in Machine Learning and AI Research*. (n.d.). Wwww.linkedin.com.  
 Retrieved June 4, 2024, from <https://www.linkedin.com/pulse/importance-datasets-machine-learning-ai-research-datatobiz>
- ITU. (2019). *Internet of Things Global Standards Initiative*. Itu.int. <https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx>
- Ivan Lee. (2024, May 13). *What are IoT Attack?* [Review of *What are IoT Attack?*]. Ivan, L. (n.d.).  
 Lee, I. (n.d.). *What are IoT Attacks? Vectors Examples and Prevention*. (n.d.).  
 Wwww.wallarm.com. Retrieved May 26, 2024, from <https://www.wallarm.com/what/iot-attack#:~:text=To%20prevent%20IoT%20attacks%2C%20it>
- Johnson, W. B. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Conference on Modern Analysis and Probability*. <https://doi.org/10.1090/conm/026/737400>
- Kaspersky. (2019). *What is a Botnet?* Kaspersky.com. <https://usa.kaspersky.com/resource-center/threats/botnet-attacks>
- Khalid, D., & Jhanjhi, N. (2020). A Review on Security and Privacy Issues and Challenges in Internet of Things. *IJCSNS International Journal of Computer Science and Network Security*, 20(4), 263.  
[https://expert.taylors.edu.my/file/rems/publication/109566\\_7213\\_1.pdf](https://expert.taylors.edu.my/file/rems/publication/109566_7213_1.pdf)
- Kim, J., Shim, M., Hong, S., Shin, Y., & Choi, E. (2020). Intelligent Detection of IoT Botnets Using Machine Learning and Deep Learning. *Applied Sciences*, 10(19), 7009.  
<https://doi.org/10.3390/app10197009>

- Kishore Angrishi. (2017). Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets. *ArXiv (Cornell University)*. <https://doi.org/10.48550/arxiv.1702.03681>
- Kolias, C., Kambourakis, G., Stavrou, A., & Voas, J. (2017). DDoS in the IoT: Mirai and Other Botnets. *Computer*, 50(7), 80–84. <https://doi.org/10.1109/mc.2017.201>
- Kuhn, M., & Johnson, K. (2018). *Applied Predictive Modeling* (2013th ed.) [Review of *Applied Predictive Modeling*]. Springer.
- Kuhrmann, M., Tell, P., Hebig, R., Klunder, J. A.-C., Munch, J., Linssen, O., Pfahl, D., Felderer, M., Prause, C., Macdonell, S., Nakatumba-Nabende, J., Raffo, D., Beecham, S., Tuzun, E., Lopez, G., Paez, N., Fontdevila, D., Licorish, S., Kupper, S., & Ruhe, G. (2021). What Makes Agile Software Development Agile. *IEEE Transactions on Software Engineering*, 1–1. <https://doi.org/10.1109/tse.2021.3099532>
- Kumar, S., Tiwari, P., & Zymbler, M. (2019). Internet of Things is a revolutionary approach for future technology enhancement: a review. *Journal of Big Data*, 6(1). springeropen. <https://doi.org/10.1186/s40537-019-0268-2>
- Madhuri Gurunathrao Desai, Shi, Y., & Suo, K. (2021). A Hybrid Approach for IoT Botnet Attack Detection. *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. <https://doi.org/10.1109/iemcon53756.2021.9623102>
- Marzano, A., Alexander, D., Fonseca, O., Fazzion, E., Hoepers, C., Steding-Jessen, K., Chaves, M. H. P. C., Cunha, Í., Guedes, D., & Meira, W. (2018, June 1). *The Evolution of Bashlite and Mirai IoT Botnets*. IEEE Xplore. <https://doi.org/10.1109/ISCC.2018.8538636>
- Miorandi, D., Sicari, S., De Pellegrini, F., & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7), 1497–1516. <https://doi.org/10.1016/j.adhoc.2012.02.016>
- Mirai Botnet Malware & Its Impact On The IoT*. (n.d.). [www.linkedin.com](http://www.linkedin.com). <https://www.linkedin.com/pulse/mirai-botnet-malware-its-impact-iot-skillfield/>
- MohammadNoor Injadat, Abdallah Moubayed, & Shami, A. (2020). Detecting Botnet Attacks in IoT Environments: An Optimized Machine Learning Approach. *ArXiv (Cornell University)*. <https://doi.org/10.1109/icm50269.2020.9331794>
- Moustafa, N., & Slay, J. (2015, November 1). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). IEEE Xplore. <https://doi.org/10.1109/MilCIS.2015.7348942>

- Muthukrishnan, R., & Rohini, R. (2016, October 1). *LASSO: A feature selection technique in predictive modeling for machine learning*. IEEE Xplore.  
<https://doi.org/10.1109/ICACA.2016.7887916>
- N\_BaloT Dataset to Detect IoT Botnet Attacks. (n.d.). Www.kaggle.com.  
<https://www.kaggle.com/datasets/mkashifn/nbaiot-dataset>
- Nahin Ul Sadad, Afsana Afrin, & Islam, N. (2021). *Binary Classification using K-Nearest Neighbor Algorithm on FPGA*. <https://doi.org/10.1109/ic4me253898.2021.9768439>
- Nicholson, P. (2022, January 21). *5 Most Famous DDoS Attacks | A10 Networks*. A10 Networks.  
<https://www.a10networks.com/blog/5-most-famous-ddos-attacks/>
- Nokia Threat Intelligence Report finds malicious IoT botnet activity has sharply increased | Nokia*. (2023). Nokia; Nokia. <https://www.nokia.com/about-us/news/releases/2023/06/07/nokia-threat-intelligence-report-finds-malicious-iot-botnet-activity-has-sharply-increased/#:~:text=The%20number%20of%20IoT%20devices>
- Python. (2020). *The Python Standard Library — Python 3.8.1 documentation*. Python.org.  
<https://docs.python.org/3/library/index.html>
- Queen, O., & Emrich, S. J. (2021). LASSO-based feature selection for improved microbial and microbiome classification. *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. <https://doi.org/10.1109/bibm52615.2021.9669485>
- Rabhi, S., Tarek Abbes, & Faouzi Zarai. (2023). *Transfer learning-based Mirai botnet detection in IoT networks*. <https://doi.org/10.1109/inista59065.2023.10310379>
- Rajesh Kumar Yadav, & Karamveer Karamveer. (2022). *A Survey on IOT Botnets and their Detection Approaches*. <https://doi.org/10.1109/icac3n56670.2022.10074482>
- Rashedun Nobi Chowdhury, Chowdhury, M. M., Chowdhury, S., Mohammed Rashedul Islam, Md. Ahsan Ayub, Chowdhury, A., & Kalpoma, K. A. (2020). *Parameter Optimization and Performance Analysis of State-of-the-Art Machine Learning Techniques for Intrusion Detection System (IDS)*. <https://doi.org/10.1109/iccit51783.2020.9392683>
- RHNS Jayathissa, & MWP Maduranga. (2022). *Study on Using Signal Filtering Techniques for Machine Learning-based Indoor Positioning Systems (IPS)*. <https://doi.org/10.1109/slaai-icai56923.2022.10002655>
- Rukshani Puvanendran, & Sharnidha Thangasundram. (2023). *Performance Analysis for Exercise Pose Prediction Using RFE Based Feature Selection*.  
<https://doi.org/10.1109/iciis58898.2023.10253489>

- Stanislav Gayvoronsky. (2024, May 30). *Gafgyt – Malware Trends Tracker by ANY.RUN*. Gafgyt | Malware Trends Tracker; ANY.RUN. <https://any.run/malware-trends/gafgyt#:~:text=Gafgyt%2C%20also%20known%20as%20BASHLITE>
- Stevanovic, M., & Pedersen, J. M. (2014). An efficient flow-based botnet detection using supervised machine learning. *2014 International Conference on Computing, Networking and Communications (ICNC)*. <https://doi.org/10.1109/icnc.2014.6785439>
- Sundaram, D. (2023, October 12). *Botnet dection on IoT Devices*. GitHub. <https://github.com/dineshh912/IoT-botnet-attack-detection>
- Susanto, Deris Stiawan, Agus, M., Mohd. Yazid Idris, & Rahmat Budiarto. (2020). *IoT Botnet Malware Classification Using Weka Tool and Scikit-learn Machine Learning*. <https://doi.org/10.23919/eecsi50503.2020.9251304>
- Susanto, N., Deris Stiawan, Agus, M., Rejito, J., Mohd. Yazid Idris, & Rahmat Budiarto. (2021). *A Dimensionality Reduction Approach for Machine Learning Based IoT Botnet Detection*. <https://doi.org/10.23919/eecsi53397.2021.9624299>
- Tanaka, H., & Yamaguchi, S. (2017, November 1). *On modeling and simulation of the behavior of IoT malwares Mirai and Hajime*. IEEE Xplore. <https://doi.org/10.1109/ISCE.2017.8355547>
- Tapsai, C. (2018, December 1). *Information Processing and Retrieval from CSV File by Natural Language*. IEEE Xplore. <https://doi.org/10.1109/ICOMIS.2018.8644947>
- Tay, E. A., Joie Salvio and Roy. (2023, January 13). *2022 IoT Threat Review | FortiGuard Labs*. Fortinet Blog. <https://www.fortinet.com/blog/threat-research/2022-iot-threat-review>
- Vailshery, L. (2023). *IoT Connected Devices Worldwide 2019-2030*. Statista. <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>
- Wang, Q., Wan, J., Nie, F., Liu, B., Yan, C., & Li, X. (2019). Hierarchical Feature Selection for Random Projection. *IEEE Transactions on Neural Networks and Learning Systems*, 30(5), 1581–1586. <https://doi.org/10.1109/tnnls.2018.2868836>
- What is Reinforcement Learning? - Reinforcement Learning Explained - AWS*. (n.d.). Amazon Web Services, Inc. [https://aws.amazon.com/what-is/reinforcement-learning/#:~:text=Reinforcement%20learning%20\(RL\)%20is%20a](https://aws.amazon.com/what-is/reinforcement-learning/#:~:text=Reinforcement%20learning%20(RL)%20is%20a)
- What is semi-supervised learning? | IBM*. (n.d.). Wwww.ibm.com. <https://www.ibm.com/topics/semi-supervised-learning>
- What is Supervised Learning?* (n.d.). Google Cloud. <https://cloud.google.com/discover/what-is-supervised-learning#:~:text=Supervised%20learning%20is%20a%20category>

- What is unsupervised learning?* (n.d.). Google Cloud. <https://cloud.google.com/discover/what-is-unsupervised-learning>
- Xiao, L., Wan, X., Lu, X., Zhang, Y., & Wu, D. (2018). IoT Security Techniques Based on Machine Learning: How Do IoT Devices Use AI to Enhance Security? *IEEE Signal Processing Magazine*, 35(5), 41–49. <https://doi.org/10.1109/msp.2018.2825478>
- Xu, L. D., He, W., & Li, S. (2014). Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, 10(4), 2233–2243. <https://doi.org/10.1109/tii.2014.2300753>
- Yang, X., Song, Z., King, I., & Xu, Z. (2022). A Survey on Deep Semi-Supervised Learning. *IEEE Transactions on Knowledge and Data Engineering*, 1–20. <https://doi.org/10.1109/tkde.2022.3220219>
- Yogesh Dhote, Agrawal, S., & Anjana Jayant Deen. (2015). *A Survey on Feature Selection Techniques for Internet Traffic Classification*. <https://doi.org/10.1109/cicn.2015.267>
- Zheng, S., & Yang, X. (2019). {DynaShield}: Reducing the Cost of {DDoS} Defense using Cloud Services. <https://www.usenix.org/conference/hotcloud19/presentation/zheng>
- Zhu, Z., Lu, G., Chen, Y., Fu, Z. J., Roberts, P., & Han, K. (2008). Botnet Research Survey. *2008 32nd Annual IEEE International Computer Software and Applications Conference*. <https://doi.org/10.1109/compsac.2008.205>
- Zscaler ThreatLabz 2023 Enterprise IoT and OT Threat Report | Zscaler. (n.d.). Info.zscaler.com. Retrieved April 30, 2024, from <https://info.zscaler.com/resources-industry-reports-threatlabz-2023-enterprise-iot-ot-threat-report>



## Appendix(dataPrepare.py)

```
import pandas as pd
import glob

def dataPrepare():
    # read normal traffic data.
    benign_files = glob.glob("*benign.csv")
    benign_data = pd.concat((pd.read_csv(file) for file in benign_files), ignore_index=True)
    benign_data['Class'] = 0

    # read mirai attack data
    mirai_ack_files = glob.glob("*mirai.ack.csv")
    mirai_ack_data = pd.concat((pd.read_csv(file) for file in mirai_ack_files),
    ignore_index=True)
    mirai_ack_data['Class'] = 1
    mirai_scan_files = glob.glob("*mirai.scan.csv")
    mirai_scan_data = pd.concat((pd.read_csv(file) for file in mirai_scan_files),
    ignore_index=True)
    mirai_scan_data['Class'] = 1
    mirai_syn_files = glob.glob("*mirai.syn.csv")
    mirai_syn_data = pd.concat((pd.read_csv(file) for file in mirai_syn_files),
    ignore_index=True)
    mirai_syn_data['Class'] = 1
    mirai_udp_files = glob.glob("*mirai.udp.csv")
    mirai_udp_data = pd.concat((pd.read_csv(file) for file in mirai_udp_files),
    ignore_index=True)
    mirai_udp_data['Class'] = 1
    mirai_udpplain_files = glob.glob("*mirai.udpplain.csv")
    mirai_udpplain_data = pd.concat((pd.read_csv(file) for file in mirai_udpplain_files),
    ignore_index=True)
    mirai_udpplain_data['Class'] = 1

    #read gafgyt attack data
    gafgyt_combo_files = glob.glob("*gafgyt.combo.csv")
    gafgyt_combo_data = pd.concat((pd.read_csv(file) for file in gafgyt_combo_files),
    ignore_index=True)
    gafgyt_combo_data['Class'] = 1
    gafgyt_junk_files = glob.glob("*gafgyt.junk.csv")
    gafgyt_junk_data = pd.concat((pd.read_csv(file) for file in gafgyt_junk_files),
    ignore_index=True)
    gafgyt_junk_data['Class'] = 1
    gafgyt_scan_files = glob.glob("*gafgyt.scan.csv")
    gafgyt_scan_data = pd.concat((pd.read_csv(file) for file in gafgyt_scan_files),
    ignore_index=True)
    gafgyt_scan_data['Class'] = 1
    gafgyt_tcp_files = glob.glob("*gafgyt.tcp.csv")
    gafgyt_tcp_data = pd.concat((pd.read_csv(file) for file in gafgyt_tcp_files),
    ignore_index=True)
    gafgyt_tcp_data['Class'] = 1
    gafgyt_udp_files = glob.glob("*gafgyt.udp.csv")
    gafgyt_udp_data = pd.concat((pd.read_csv(file) for file in gafgyt_udp_files),
    ignore_index=True)
    gafgyt_udp_data['Class'] = 1

    #combine normal data and attack data
    frames = [benign_data, mirai_ack_data, mirai_scan_data, mirai_syn_data, mirai_udp_data,
    mirai_udpplain_data, gafgyt_combo_data, gafgyt_junk_data, gafgyt_scan_data, gafgyt_tcp_data,
    gafgyt_udp_data]
```

```
result = pd.concat(frames, ignore_index=True)
print("Starting to save data...")
result.to_csv('totalData.csv')
print("Done for saving")
print(result.shape)
print(result.head())

if __name__ == '__main__':
    dataPrepare()

# using pandas to read csv file
data = pd.read_csv('totalData.csv')
data = data.iloc[:, 1:]
data.to_csv('totalData.csv', index=False)
data = pd.read_csv('totalData.csv')
print(data)
```



اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

## Appendix(main.py)

```
from sklearnex import patch_sklearn
patch_sklearn()
from scipy.stats import chi2_contingency
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import LinearSVC
from sklearn import random_projection

import pandas as pd
import numpy as np

def dataPreprocessing(data):
    # Separate out the features x and output variable y
    x = data.iloc[:, data.columns != 'Class']
    y = data.iloc[:, data.columns == 'Class'].values.ravel()

    # Dataset division: Divide the dataset into 70% for training and 30% for testing
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)

    return x_train, x_test, y_train, y_test

# feature selection function
def lasso_hybrid_feature_selection(x_train, x_test, y_train, max_features_list):
    # standard the feature
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_train)
    x_test_scaled = scaler.transform(x_test)

    # convert the standardized data into Data Frame and keep the feature name
    x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_train.columns)
    x_test_scaled = pd.DataFrame(x_test_scaled, columns=x_test.columns)

    lasso_rp_results = {}
    lasso_chi_results = {}
    lasso_rfe_results = {}

    # let lasso to select feature
    sel_ = SelectFromModel(LogisticRegression(C=0.05, penalty='l1', solver='liblinear',
random_state=10))
    sel_.fit(x_train_scaled, y_train)
    all_selected_features_lasso = x_train_scaled.columns[(sel_.get_support())]
    x_train_selected_lasso = x_train_scaled[all_selected_features_lasso]
    x_test_selected_lasso = x_test_scaled[all_selected_features_lasso]

    # let chi square feature filter select feature from lasso
    chi_ls = []
    for feature in x_train_selected_lasso.columns:
        c = pd.crosstab(y_train, x_train_selected_lasso[feature])
        p_value = chi2_contingency(c)[1]
        chi_ls.append(p_value)
```

```

    all_selected_features_chi = pd.Series(chi_ls,
index=x_train_selected_lasso.columns).sort_values(ascending=True).index

```

```

# let rfe select feature from lasso
# Build initial model
model = GradientBoostingClassifier(n_estimators=10, max_depth=4, random_state=10)
# Use RFE select feature
max_features = max(max_features_list)
selector = RFE(model, n_features_to_select=max_features, step=10)
selector.fit(x_train_selected_lasso, y_train)
# Get the ranking of features
ranking = selector.ranking_
selected_features_ordered = x_train_selected_lasso.columns[np.argsort(ranking)]

for num_features in max_features_list:
    # let random projection select feature from lasso
    rp = random_projection.SparseRandomProjection(n_components=num_features)
    x_train_rp = rp.fit_transform(x_train_selected_lasso)
    x_test_rp = rp.transform(x_test_selected_lasso)
    lasso_rp_results[num_features] = (x_train_rp, x_test_rp)

    # let chi square filter select top feature from itself
    selected_features_chi = all_selected_features_chi[:num_features]
    print(f"Lasso + Chi-Square selected top {num_features} features:
[selected_features_chi.tolist()]")
    x_train_selected_chi = x_train_selected_lasso[selected_features_chi]
    x_test_selected_chi = x_test_selected_lasso[selected_features_chi]
    lasso_chi_results[num_features] = (x_train_selected_chi, x_test_selected_chi)

    # let rfe select top feature from itself
    selected_features_rfe = selected_features_ordered[:num_features]
    print(f'Lasso + RFE selected top {num_features} features:
[selected_features_rfe.tolist()]')
    lasso_rfe_results[num_features] = (x_train_selected_lasso[selected_features_rfe],
x_test_selected_lasso[selected_features_rfe])

return lasso_rp_results, lasso_chi_results, lasso_rfe_results

def random_projection_hybrid_feature_selection(x_train, x_test, y_train, max_features_list):
    rp_lasso_results = {}
    rp_chi_results = {}
    rp_rfe_results = {}

    # let random projection to select feature first
    rp = random_projection.SparseRandomProjection(n_components=57)
    x_train_selected_rp = rp.fit_transform(x_train)
    x_test_selected_rp = rp.transform(x_test)

    # Convert numpy arrays to DataFrame
    x_train_selected_rp = pd.DataFrame(x_train_selected_rp, columns=[f"RP_Feature_{i}" for i
in range(x_train_selected_rp.shape[1])])
    x_test_selected_rp = pd.DataFrame(x_test_selected_rp, columns=[f"RP_Feature_{i}" for i in
range(x_test_selected_rp.shape[1])])

    # let lasso selection feature from random projection
    print("Lasso Selection Start")
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_train_selected_rp)
    x_test_scaled = scaler.transform(x_test_selected_rp)

```

```

    x_train_scaled = pd.DataFrame(x_train_scaled, columns=[f"RP_Feature_{i}" for i in
range(x_train_scaled.shape[1])])
    x_test_scaled = pd.DataFrame(x_test_scaled, columns=[f"RP_Feature_{i}" for i in
range(x_test_scaled.shape[1])])

    sel_ = SelectFromModel(LogisticRegression(C=0.05, penalty='l1', solver='liblinear',
random_state=10))
    sel_.fit(x_train_scaled, y_train)
    all_selected_features_lasso = x_train_scaled.columns[(sel_.get_support())]

    # let chi square select feature from random projection
    print("Chi Square Selection Start")
    chi_ls = []
    for feature in x_train_selected_rp.columns:
        c = pd.crosstab(y_train, x_train_selected_rp[feature])
        p_value = chi2_contingency(c)[1]
        chi_ls.append(p_value)
    all_selected_features_chi = pd.Series(chi_ls,
index=x_train_selected_rp.columns).sort_values(ascending=True).index

    # let rfe select feature from random projection
    print("RFE Selection Start")
    # Build initial model
    model = GradientBoostingClassifier(n_estimators=10, max_depth=4, random_state=10)
    # Use RFE select feature
    max_features = max(max_features_list)
    selector = RFE(model, n_features_to_select=max_features, step=10)
    selector.fit(x_train_selected_rp, y_train)
    # Get the ranking of features
    ranking = selector.ranking_
    selected_features_ordered = x_train_selected_rp.columns[np.argsort(ranking)]

    for num_features in max_features_list:
        # let lasso select the top feature
        lasso_selected_features = all_selected_features_lasso[:num_features]
        print(f'Random Projection + Lasso selected top {num_features} features:
{lasso_selected_features.tolist()}')
        x_train_selected_lasso = x_train_scaled[lasso_selected_features]
        x_test_selected_lasso = x_test_scaled[lasso_selected_features]
        rp_lasso_results[num_features] = (x_train_selected_lasso, x_test_selected_lasso)

        # let chi square select the top feature
        chi_selected_features = all_selected_features_chi[:num_features]
        print(f"Random Projection + Chi-Square selected top {num_features} features:
{chi_selected_features.tolist()}")
        x_train_selected_chi = x_train_selected_rp[chi_selected_features]
        x_test_selected_chi = x_test_selected_rp[chi_selected_features]
        rp_chi_results[num_features] = (x_train_selected_chi, x_test_selected_chi)

        # let rfe select top feature from itself
        selected_features_rfe = selected_features_ordered[:num_features]
        print(f'Random Projection + RFE selected top {num_features} features:
{selected_features_rfe.tolist()}')
        rp_rfe_results[num_features] = (x_train_selected_rp[selected_features_rfe],
x_test_selected_rp[selected_features_rfe])

    return rp_lasso_results, rp_chi_results, rp_rfe_results

def chi_square_hybrid_feature_selection(x_train, x_test, y_train, max_features_list):
    chi_lasso_results = {}
    chi_rp_results = {}

```

```

chi_rfe_results = {}

# chi square filter feature selection
print("Chi Square Filter Feature Selection Start")
chi_ls = []
for feature in x_train.columns:
    c = pd.crosstab(y_train, x_train[feature])
    p_value = chi2_contingency(c)[1]
    chi_ls.append(p_value)

# Select top 57 features based on p-value
all_selected_features_chi = pd.Series(chi_ls,
index=x_train.columns).sort_values(ascending=True).index
chi_selected_features = all_selected_features_chi[:57]
x_train_selected_chi = x_train[chi_selected_features]
x_test_selected_chi = x_test[chi_selected_features]

# let lasso selection feature from chi square filter
print("Lasso Selection Start")
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train_selected_chi)
x_test_scaled = scaler.transform(x_test_selected_chi)

x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_train_selected_chi.columns)
x_test_scaled = pd.DataFrame(x_test_scaled, columns=x_test_selected_chi.columns)

sel_ = SelectFromModel(LogisticRegression(C=0.05, penalty='l1', solver='liblinear',
random_state=10))
sel_.fit(x_train_scaled, y_train)
all_selected_features_lasso = x_train_scaled.columns[(sel_.get_support())]

# let rfe select feature from chi square filter
print("RFE Selection Start")
# Build initial model
model = GradientBoostingClassifier(n_estimators=10, max_depth=4, random_state=10)
# Use RFE select feature
max_features = max(max_features_list)
selector = RFE(model, n_features_to_select=max_features, step=10)
selector.fit(x_train_selected_chi, y_train)
# Get the ranking of features
ranking = selector.ranking_
selected_features_ordered = x_train_selected_chi.columns[np.argsort(ranking)]

for num_features in max_features_list:
    # let lasso select the top feature from itself
    lasso_selected_features = all_selected_features_lasso[:num_features]
    print(f'Chi Square Filter + Lasso selected top {num_features} features:
{lasso_selected_features.tolist()}')
    x_train_selected_lasso = x_train_scaled[lasso_selected_features]
    x_test_selected_lasso = x_test_scaled[lasso_selected_features]
    chi_lasso_results[num_features] = (x_train_selected_lasso, x_test_selected_lasso)

# let random projection select feature from chi square
rp = random_projection.SparseRandomProjection(n_components=num_features)
x_train_rp = rp.fit_transform(x_train_selected_chi)
x_test_rp = rp.transform(x_test_selected_chi)
chi_rp_results[num_features] = (x_train_rp, x_test_rp)

# let rfe select top feature from itself
selected_features_rfe = selected_features_ordered[:num_features]

```

```

        print(f'Chi Square Filter + RFE selected top {num_features} features:
{selected_features_rfe.tolist()}')
        chi_rfe_results[num_features] = (x_train_selected_chi[selected_features_rfe],
x_test_selected_chi[selected_features_rfe])

    return chi_lasso_results, chi_rp_results, chi_rfe_results

def rfe_hybrid_feature_selection(x_train, x_test, y_train, max_features_list):
    rfe_lasso_results = {}
    rfe_rp_results = {}
    rfe_chi_results = {}

    # let rfe to select feature first
    print("RFE Feature Selection Start")
    # Build initial model
    model = GradientBoostingClassifier(n_estimators=10, max_depth=4, random_state=10)
    # Use RFE select feature
    selector = RFE(model, n_features_to_select=57, step=10)
    selector.fit(x_train, y_train)
    # Get the ranking of features
    ranking = selector.ranking_
    selected_features = x_train.columns[np.argsort(ranking)]
    x_train_selected_rfe = x_train[selected_features]
    x_test_selected_rfe = x_test[selected_features]

    # let lasso selection feature from RFE
    print("Lasso Feature Selection Start")
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_train_selected_rfe)
    x_test_scaled = scaler.transform(x_test_selected_rfe)

    x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_train_selected_rfe.columns)
    x_test_scaled = pd.DataFrame(x_test_scaled, columns=x_test_selected_rfe.columns)

    sel_ = SelectFromModel(LogisticRegression(C=0.05, penalty='l1', solver='liblinear',
random_state=10))
    sel_.fit(x_train_scaled, y_train)
    all_selected_features_lasso = x_train_scaled.columns[(sel_.get_support())]

    # let chi square feature filter select feature from RFE
    print("Chi-Square Filter Feature Selection Start")
    chi_ls = []
    for feature in x_train_selected_rfe.columns:
        c = pd.crosstab(y_train, x_train_selected_rfe[feature])
        p_value = chi2_contingency(c)[1]
        chi_ls.append(p_value)
    all_selected_features_chi = pd.Series(chi_ls,
index=x_train_selected_rfe.columns).sort_values(ascending=True).index

    for num_features in max_features_list:
        # let random projection select feature from rfe
        rp = random_projection.SparseRandomProjection(n_components=num_features)
        x_train_rp = rp.fit_transform(x_train_selected_rfe)
        x_test_rp = rp.transform(x_test_selected_rfe)
        rfe_rp_results[num_features] = (x_train_rp, x_test_rp)

        # let lasso select the top feature
        lasso_selected_features = all_selected_features_lasso[:num_features]
        print(f'RFE + Lasso selected top {num_features} features:
{lasso_selected_features.tolist()}')
        x_train_selected_lasso = x_train_scaled[lasso_selected_features]

```

```

x_test_selected_lasso = x_test_scaled[lasso_selected_features]
rfe_lasso_results[num_features] = (x_train_selected_lasso, x_test_selected_lasso)

# let chi square select the top feature
chi_selected_features = all_selected_features_chi[:num_features]
print(f"RFE + Chi-Square selected top {num_features} features:
{chi_selected_features.tolist()}")
x_train_selected_chi = x_train_selected_rfe[chi_selected_features]
x_test_selected_chi = x_test_selected_rfe[chi_selected_features]
rfe_chi_results[num_features] = (x_train_selected_chi, x_test_selected_chi)

return rfe_lasso_results, rfe_rp_results, rfe_chi_results

def lasso_feature_selection(x_train, x_test, y_train, max_features_list):
    # standard the feature
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_train)
    x_test_scaled = scaler.transform(x_test)

    # convert the standardized data into Data Frame and keep the feature name
    x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_train.columns)
    x_test_scaled = pd.DataFrame(x_test_scaled, columns=x_test.columns)

    results = {}

    # let lasso to select feature
    sel_ = SelectFromModel(LogisticRegression(C=0.05, penalty='l1', solver='liblinear',
random_state=10))
    sel_.fit(x_train_scaled, y_train)
    all_selected_features = x_train_scaled.columns[(sel_.get_support())]

    # show selected feature
    for num_features in max_features_list:
        selected_features = all_selected_features[:num_features]
        print(f'Lasso selected top {num_features} features: {selected_features.tolist()}')
        x_train_selected = x_train_scaled[selected_features]
        x_test_selected = x_test_scaled[selected_features]
        results[num_features] = (x_train_selected, x_test_selected)

    return results

def random_projection_feature_selection(x_train, x_test, y_train, max_features_list):
    results = {}

    for num_features in max_features_list:
        rp = random_projection.SparseRandomProjection(n_components=num_features)
        x_train_rp = rp.fit_transform(x_train)
        x_test_rp = rp.transform(x_test)
        results[num_features] = (x_train_rp, x_test_rp)

    return results

def chi_square_feature_selection(x_train, x_test, y_train, max_features_list):
    results = {}

    chi_ls = []
    for feature in x_train.columns:
        c = pd.crosstab(y_train, x_train[feature])
        p_value = chi2_contingency(c)[1]
        chi_ls.append(p_value)

```



```

# Select top features based on p-value
all_selected_features = pd.Series(chi_ls,
index=x_train.columns).sort_values(ascending=True).index
for num_features in max_features_list:
    selected_features = all_selected_features[:num_features]
    print(f"Chi-Square selected top {num_features} features: {selected_features.tolist()}")
    x_train_selected = x_train[selected_features]
    x_test_selected = x_test[selected_features]
    results[num_features] = (x_train_selected, x_test_selected)

return results

def rfe_feature_selection(x_train, x_test, y_train, max_features_list):
    results = {}

    # Build initial model
    model = GradientBoostingClassifier(n_estimators=10, max_depth=4, random_state=10)

    # Use RFE select feature
    max_features = max(max_features_list)
    selector = RFE(model, n_features_to_select=max_features, step=10)
    selector.fit(x_train, y_train)

    # Get the ranking of features
    ranking = selector.ranking_
    selected_features_ordered = x_train.columns[np.argsort(ranking)]

    # Get and show selected features for each required number of features
    for num_features in max_features_list:
        selected_features = selected_features_ordered[:num_features]
        print(f'RFE selected top {num_features} features: {selected_features.tolist()}')
        results[num_features] = (x_train[selected_features], x_test[selected_features])

    return results

# model train and test
def train_and_evaluate(x_train, x_test, y_train, y_test):
    models = {
        'Decision Tree': DecisionTreeClassifier(random_state=0),
        'Random Forest': RandomForestClassifier(random_state=0),
        'KNN': KNeighborsClassifier(),
        'Linear SVC': LinearSVC(random_state=0),
    }

    performance = {}

    for model_name, model in models.items():
        print(model_name + " Start")
        model.fit(x_train, y_train)
        y_pred = model.predict(x_test)

        accuracy = accuracy_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        performance[model_name] = {
            'Accuracy': accuracy,
            'Recall': recall,
            'Precision': precision,
            'F1-Score': f1

```

```

    }
    print(model_name + " Done")

    return performance

if __name__ == '__main__':
    # load data
    print("Start load data")
    data = pd.read_csv('N_BaIoT/totalData.csv')
    print("Data loaded\n\nNow data preprocessing Start")
    x_train, x_test, y_train, y_test = dataPreprocessing(data)
    print("Data Preprocessing End")

    # number of feature want to test
    max_features_list = [2, 3, 5, 10, 15]
    results = []

    # create dictionary for feature selection function
    feature_selection_methods = {
        'Lasso': lasso_feature_selection,
        'Random Projection': random_projection_feature_selection,
        'Chi-Square': chi_square_feature_selection,
        'RFE': rfe_feature_selection,
        'Lasso + Hybrid': lasso_hybrid_feature_selection,
        'Random Projection + Hybrid': random_projection_hybrid_feature_selection,
        'Chi-Square + Hybrid': chi_square_hybrid_feature_selection,
        'RFE + Hybrid': rfe_hybrid_feature_selection
    }

    # function call for feature selection function, and store result into
    selected_features_results_list
    for method_name, feature_selection_function in feature_selection_methods.items():
        print(f"\n{method_name} Feature Selection Start")
        selected_features_results_list = []
        if method_name == 'Lasso + Hybrid':
            lasso_rp_results, lasso_chi_results, lasso_rfe_results =
feature_selection_function(x_train, x_test, y_train, max_features_list)
            selected_features_results_list = [
                ('Random Projection', lasso_rp_results),
                ('Chi-Square', lasso_chi_results),
                ('RFE', lasso_rfe_results)
            ]
        elif method_name == 'Random Projection + Hybrid':
            rp_lasso_results, rp_chi_results, rp_rfe_results =
feature_selection_function(x_train, x_test, y_train, max_features_list)
            selected_features_results_list = [
                ('Lasso', rp_lasso_results),
                ('Chi-Square', rp_chi_results),
                ('RFE', rp_rfe_results)
            ]
        elif method_name == 'Chi-Square + Hybrid':
            chi_lasso_results, chi_rp_results, chi_rfe_results =
feature_selection_function(x_train, x_test, y_train, max_features_list)
            selected_features_results_list = [
                ('Lasso', chi_lasso_results),
                ('Random Projection', chi_rp_results),
                ('RFE', chi_rfe_results)
            ]
        elif method_name == 'RFE + Hybrid':
            rfe_lasso_results, rfe_rp_results, rfe_chi_results =
feature_selection_function(x_train, x_test, y_train, max_features_list)

```

```

        selected_features_results_list = [
            ('Lasso', rfe_lasso_results),
            ('Random Projection', rfe_rp_results),
            ('Chi-Square', rfe_chi_results)
        ]
    else:
        selected_features_results = feature_selection_function(x_train, x_test, y_train,
max_features_list)
        selected_features_results_list.append((method_name, selected_features_results))
        print(f"{method_name} Feature Selection End")

    # train model using feature selected
    for sub_method_name, selected_features_results in selected_features_results_list:
        for num_features in max_features_list:
            print(f"\nTrain and test model with {method_name} - {sub_method_name}-selected
features Start for {num_features} features")
            performance = train_and_evaluate(selected_features_results[num_features][0],
selected_features_results[num_features][1], y_train, y_test)
            print(f"Train and test model with {method_name} - {sub_method_name}-selected
features End for {num_features} features\n")

            # Show results for current feature count
            print(f"Performance with {method_name} + {sub_method_name} feature selection
({num_features} features):")
            for model, metrics in performance.items():
                print(f"{model}: {metrics}")
                results.append({
                    "Number of Features": num_features,
                    "Model": model,
                    "Feature Selection Method": f'{method_name} - {sub_method_name}',
                    "Accuracy(%)": metrics['Accuracy'] * 100,
                    "Recall(%)": metrics['Recall'] * 100,
                    "Precision(%)": metrics['Precision'] * 100,
                    "F1-Score(%)": metrics['F1-Score'] * 100
                })

    # convert result into Data Frame
    results_df = pd.DataFrame(results)

    # save result into csv file
    results_df.to_excel('result.xlsx', index=False)

    print("Results have been saved to result.xlsx")

```

## Appendix(Full Result)

Number of Features Selected	Model	Feature Selection Method	Accuracy (%)	Recall (%)	Precision (%)	F1-Score (%)
2	Decision Tree	Chi-Square	99.5412	99.9744	99.5295	99.7515
2	Decision Tree	Chi-Square + Lasso	96.9377	99.9180	96.8567	98.3636
2	Decision Tree	Chi-Square + Random Projection	99.6278	99.9631	99.6340	99.7983
2	Decision Tree	Chi-Square + RFE	99.9934	99.9986	99.9942	99.9964
2	Decision Tree	Lasso	99.8663	99.9674	99.8875	99.9274
2	Decision Tree	Lasso + Chi-Square	99.0803	99.9753	99.0353	99.5031
2	Decision Tree	Lasso + Random Projection	99.5747	99.8652	99.6736	99.7693
2	Decision Tree	Lasso + RFE	99.9851	99.9970	99.9869	99.9919
2	Decision Tree	Random Projection	99.9881	99.9975	99.9894	99.9935
2	Decision Tree	Random Projection + Chi-Square	99.8048	99.9331	99.8551	99.8941
2	Decision Tree	Random Projection + Lasso	92.8935	99.9664	92.8641	96.2845
2	Decision Tree	Random Projection + RFE	99.5016	99.9017	99.5587	99.7299
2	Decision Tree	RFE	99.9883	99.9975	99.9899	99.9937
2	Decision Tree	RFE + Chi-Square	99.9883	99.9975	99.9899	99.9937
2	Decision Tree	RFE + Lasso	99.9851	99.9970	99.9868	99.9919
2	Decision Tree	RFE + Random Projection	99.9358	99.9821	99.9482	99.9652
2	KNN	Chi-Square	99.7380	99.9269	99.7889	99.8579
2	KNN	Chi-Square + Lasso	98.5423	99.7299	98.7011	99.2128
2	KNN	Chi-Square + Random Projection	99.2890	99.9461	99.2867	99.6153
2	KNN	Chi-Square + RFE	99.9732	99.9825	99.9884	99.9855
2	KNN	Lasso	99.9015	99.9446	99.9485	99.9466
2	KNN	Lasso + Chi-Square	99.6645	99.8637	99.7723	99.8180
2	KNN	Lasso + Random Projection	99.3561	99.7669	99.5352	99.6509
2	KNN	Lasso + RFE	99.9470	99.9720	99.9704	99.9712
2	KNN	Random Projection	99.4999	99.8573	99.6008	99.7289
2	KNN	Random Projection + Chi-Square	99.4999	99.8572	99.6008	99.7289
2	KNN	Random Projection + Lasso	99.6754	99.8904	99.7575	99.8239
2	KNN	Random Projection + RFE	99.0104	99.8010	99.1304	99.4646
2	KNN	RFE	99.9575	99.9795	99.9744	99.9777
2	KNN	RFE + Chi-Square	99.9575	99.9795	99.9744	99.9769
2	KNN	RFE + Lasso	99.9470	99.9720	99.9704	99.9712
2	KNN	RFE + Random Projection	99.9081	99.9913	99.9090	99.9501
2	Linear SVC	Chi-Square	92.8824	99.8535	92.9438	96.2748
2	Linear SVC	Chi-Square + Lasso	92.1305	99.9776	92.1462	95.9023
2	Linear SVC	Chi-Square + Random Projection	93.0592	99.9935	92.9978	96.3689
2	Linear SVC	Chi-Square + RFE	92.1098	100	92.1098	95.8928
2	Linear SVC	Lasso	92.1098	100	92.1098	95.8928
2	Linear SVC	Lasso + Chi-Square	92.8822	99.8533	92.9438	96.2747
2	Linear SVC	Lasso + Random Projection	92.1772	99.9944	92.1763	95.9263
2	Linear SVC	Lasso + RFE	92.1098	100	92.1098	95.8928

2	Linear SVC	Random Projection	92.8246	100	92.7730	96.2510
2	Linear SVC	Random Projection + Chi-Square	46.7684	46.8690	90.9557	61.8612
2	Linear SVC	Random Projection + Lasso	92.1098	100	92.1098	95.8928
2	Linear SVC	Random Projection + RFE	92.2453	99.9984	92.2360	95.9605
2	Linear SVC	RFE	92.1098	100	92.1098	95.8928
2	Linear SVC	RFE + Chi-Square	92.1098	100	92.1098	95.8928
2	Linear SVC	RFE + Lasso	92.1098	100	92.1098	95.8928
2	Linear SVC	RFE + Random Projection	92.2800	100	92.2668	95.9779
2	Random Forest	Chi-Square	99.5877	99.7929	99.7595	99.7762
2	Random Forest	Chi-Square + Lasso	98.5083	99.6735	98.7194	99.1942
2	Random Forest	Chi-Square + Random Projection	98.6507	99.5452	98.9954	99.2696
2	Random Forest	Chi-Square + RFE	99.9531	99.9576	99.9915	99.9745
2	Random Forest	Lasso	99.8106	99.9299	99.8646	99.8972
2	Random Forest	Lasso + Chi-Square	99.5841	99.7981	99.7504	99.7743
2	Random Forest	Lasso + Random Projection	99.1377	99.5914	99.4731	99.5322
2	Random Forest	Lasso + RFE	99.9098	99.9409	99.9612	99.9510
2	Random Forest	Random Projection	99.7724	99.9444	99.8087	99.8765
2	Random Forest	Random Projection + Chi-Square	99.3072	99.8258	99.4243	99.6247
2	Random Forest	Random Projection + Lasso	99.5595	99.8155	99.7066	99.7610
2	Random Forest	Random Projection + RFE	97.8678	98.8517	98.8337	98.8427
2	Random Forest	RFE	99.8249	99.9439	99.8661	99.9050
2	Random Forest	RFE + Chi-Square	99.8249	99.9439	99.8661	99.9050
2	Random Forest	RFE + Lasso	99.8249	99.9439	99.8661	99.9050
2	Random Forest	RFE + Random Projection	99.8050	99.9353	99.8531	99.8942
3	Decision Tree	Chi-Square	99.7734	99.9932	99.7613	99.8771
3	Decision Tree	Chi-Square + Lasso	98.0257	99.9961	97.9052	98.9396
3	Decision Tree	Chi-Square + Random Projection	99.9933	99.9988	99.9939	99.9964
3	Decision Tree	Chi-Square + RFE	99.9973	99.9994	99.9976	99.9985
3	Decision Tree	Lasso	99.9858	99.9913	99.9933	99.9923
3	Decision Tree	Lasso + Chi-Square	99.1447	99.9939	99.0859	99.5378
3	Decision Tree	Lasso + Random Projection	99.8631	99.9548	99.8966	99.9257
3	Decision Tree	Lasso + RFE	99.9947	99.9988	99.9954	99.9971
3	Decision Tree	Random Projection	99.9789	99.9930	99.9841	99.9885
3	Decision Tree	Random Projection + Chi-Square	99.9578	99.9821	99.9721	99.9771
3	Decision Tree	Random Projection + Lasso	99.9909	99.9981	99.9921	99.9951
3	Decision Tree	Random Projection + RFE	99.9341	99.9860	99.9425	99.9642
3	Decision Tree	RFE	99.9883	99.9973	99.9900	99.9937
3	Decision Tree	RFE + Chi-Square	99.9883	99.9973	99.9900	99.9937
3	Decision Tree	RFE + Lasso	99.9864	99.9968	99.9884	99.9926
3	Decision Tree	RFE + Random Projection	99.9916	99.9977	99.9931	99.9954
3	KNN	Chi-Square	99.8268	99.9653	99.8469	99.9061
3	KNN	Chi-Square + Lasso	99.6885	99.9601	99.7025	99.8311
3	KNN	Chi-Square + Random Projection	99.9879	99.9977	99.9891	99.9934
3	KNN	Chi-Square + RFE	99.9909	99.9953	99.9948	99.9951

3	KNN	Lasso	99.9602	99.9822	99.9746	99.9784
3	KNN	Lasso + Chi-Square	99.8024	99.9412	99.8444	99.8928
3	KNN	Lasso + Random Projection	99.8263	99.9641	99.8474	99.9057
3	KNN	Lasso + RFE	99.9784	99.9876	99.9889	99.9883
3	KNN	Random Projection	99.6032	99.9241	99.6462	99.7849
3	KNN	Random Projection + Chi-Square	99.4999	99.8572	99.6008	99.7289
3	KNN	Random Projection + Lasso	99.7665	99.9833	99.7637	99.8734
3	KNN	Random Projection + RFE	99.6335	99.8919	99.7107	99.8012
3	KNN	RFE	99.9577	99.9795	99.9746	99.9770
3	KNN	RFE + Chi-Square	99.9577	99.9795	99.9746	99.9770
3	KNN	RFE + Lasso	99.9444	99.9712	99.9684	99.9698
3	KNN	RFE + Random Projection	99.4999	99.8573	99.6008	99.7289
3	Linear SVC	Chi-Square	92.8135	99.7373	92.9719	96.2359
3	Linear SVC	Chi-Square + Lasso	92.6447	99.9510	92.6438	96.1588
3	Linear SVC	Chi-Square + Random Projection	73.6829	71.6635	99.6732	83.3789
3	Linear SVC	Chi-Square + RFE	95.0363	99.9498	94.9295	97.3750
3	Linear SVC	Lasso	99.9276	99.9597	99.9618	99.9607
3	Linear SVC	Lasso + Chi-Square	92.8662	99.7984	92.9726	96.2647
3	Linear SVC	Lasso + Random Projection	92.2316	99.9982	92.2235	95.9536
3	Linear SVC	Lasso + RFE	94.9297	99.9449	94.8294	97.3200
3	Linear SVC	Random Projection	78.6828	83.5401	92.5925	87.8337
3	Linear SVC	Random Projection + Chi-Square	46.7684	46.8690	90.9557	61.8612
3	Linear SVC	Random Projection + Lasso	92.2017	100	92.1946	95.9388
3	Linear SVC	Random Projection + RFE	53.2234	57.7489	87.1270	69.4593
3	Linear SVC	RFE	92.1098	100	92.1098	95.8928
3	Linear SVC	RFE + Chi-Square	92.1098	100	92.1098	95.8928
3	Linear SVC	RFE + Lasso	92.1098	100	92.1098	95.8928
3	Linear SVC	RFE + Random Projection	92.1614	100	92.1573	95.9186
3	Random Forest	Chi-Square	99.8181	99.9343	99.8683	99.9013
3	Random Forest	Chi-Square + Lasso	99.5552	99.9356	99.5829	99.7590
3	Random Forest	Chi-Square + Random Projection	99.9045	99.9777	99.9187	99.9482
3	Random Forest	Chi-Square + RFE	99.9788	99.9854	99.9916	99.9885
3	Random Forest	Lasso	99.9396	99.9586	99.9758	99.9672
3	Random Forest	Lasso + Chi-Square	99.8150	99.9340	99.8652	99.8996
3	Random Forest	Lasso + Random Projection	99.8097	99.9231	99.8703	99.8967
3	Random Forest	Lasso + RFE	99.9646	99.9672	99.9944	99.9808
3	Random Forest	Random Projection	99.9068	99.9682	99.9306	99.9494
3	Random Forest	Random Projection + Chi-Square	99.7378	99.9093	99.8063	99.8578
3	Random Forest	Random Projection + Lasso	99.9213	99.9543	99.9602	99.9573
3	Random Forest	Random Projection + RFE	99.8129	99.8934	99.9035	99.8984
3	Random Forest	RFE	99.9102	99.9365	99.9660	99.9512
3	Random Forest	RFE + Chi-Square	99.9102	99.9365	99.9660	99.9512
3	Random Forest	RFE + Lasso	99.9102	99.9365	99.9660	99.9512
3	Random Forest	RFE + Random Projection	99.8439	99.9671	99.8636	99.9153

5	Decision Tree	Chi-Square	99.8907	99.9974	99.8841	99.9407
5	Decision Tree	Chi-Square + Lasso	99.9276	99.9986	99.9229	99.9607
5	Decision Tree	Chi-Square + Random Projection	99.9788	99.9977	99.9793	99.9885
5	Decision Tree	Chi-Square + RFE	99.9973	99.9994	99.9976	99.9985
5	Decision Tree	Lasso	99.9963	99.9993	99.9967	99.9980
5	Decision Tree	Lasso + Chi-Square	99.9061	99.9983	99.8999	99.9491
5	Decision Tree	Lasso + Random Projection	99.9834	99.9958	99.9862	99.9910
5	Decision Tree	Lasso + RFE	99.9947	99.9989	99.9954	99.9971
5	Decision Tree	Random Projection	99.9972	99.9995	99.9975	99.9985
5	Decision Tree	Random Projection + Chi-Square	99.9892	99.9955	99.9928	99.9942
5	Decision Tree	Random Projection + Lasso	99.9956	99.9993	99.9960	99.9976
5	Decision Tree	Random Projection + RFE	99.9967	99.9993	99.9971	99.9982
5	Decision Tree	RFE	99.9969	99.9991	99.9975	99.9983
5	Decision Tree	RFE + Chi-Square	99.9935	99.9987	99.9942	99.9965
5	Decision Tree	RFE + Lasso	99.9945	99.9992	99.9949	99.9970
5	Decision Tree	RFE + Random Projection	99.9952	99.9993	99.9955	99.9974
5	KNN	Chi-Square	99.8698	99.9811	99.8777	99.9294
5	KNN	Chi-Square + Lasso	99.9689	99.9801	99.9861	99.9831
5	KNN	Chi-Square + Random Projection	99.9247	99.9949	99.9233	99.9591
5	KNN	Chi-Square + RFE	99.9908	99.9949	99.9952	99.9950
5	KNN	Lasso	99.9834	99.9947	99.9873	99.9909
5	KNN	Lasso + Chi-Square	99.9556	99.9802	99.9716	99.9759
5	KNN	Lasso + Random Projection	99.9714	99.9946	99.9744	99.9845
5	KNN	Lasso + RFE	99.9773	99.9858	99.9895	99.9877
5	KNN	Random Projection	99.9146	99.9724	99.9349	99.9536
5	KNN	Random Projection + Chi-Square	99.4999	99.8572	99.6008	99.7289
5	KNN	Random Projection + Lasso	99.8759	99.9902	99.8752	99.9327
5	KNN	Random Projection + RFE	99.9200	99.9738	99.9394	99.9566
5	KNN	RFE	99.9892	99.9949	99.9933	99.9941
5	KNN	RFE + Chi-Square	99.9788	99.9948	99.9822	99.9885
5	KNN	RFE + Lasso	99.9869	99.9937	99.9921	99.9929
5	KNN	RFE + Random Projection	99.9145	99.9719	99.9353	99.9536
5	Linear SVC	Chi-Square	93.4050	99.7613	93.5123	96.5358
5	Linear SVC	Chi-Square + Lasso	95.4458	99.8549	95.4142	97.5841
5	Linear SVC	Chi-Square + Random Projection	91.4415	98.9575	92.3054	95.5158
5	Linear SVC	Chi-Square + RFE	92.1098	100	92.1098	95.8928
5	Linear SVC	Lasso	99.9289	99.9588	99.9640	99.9614
5	Linear SVC	Lasso + Chi-Square	93.9863	99.8706	93.9782	96.8348
5	Linear SVC	Lasso + Random Projection	94.0539	99.8737	94.0405	96.8694
5	Linear SVC	Lasso + RFE	94.9371	99.9448	94.8368	97.3238
5	Linear SVC	Random Projection	94.0204	99.9699	93.9287	96.8552
5	Linear SVC	Random Projection + Chi-Square	46.7684	46.8690	90.9557	61.8612
5	Linear SVC	Random Projection + Lasso	92.5477	99.9811	92.5298	96.1112
5	Linear SVC	Random Projection + RFE	92.1992	99.9994	92.1927	95.9375

5	Linear SVC	RFE	94.3264	99.9496	94.2398	97.0108
5	Linear SVC	RFE + Chi-Square	92.0811	99.9458	92.1254	95.8764
5	Linear SVC	RFE + Lasso	93.1481	99.9795	93.0927	96.4133
5	Linear SVC	RFE + Random Projection	23.7532	18.4106	93.9342	30.7872
5	Random Forest	Chi-Square	99.8858	99.9733	99.9028	99.9380
5	Random Forest	Chi-Square + Lasso	99.9855	99.9901	99.9942	99.9921
5	Random Forest	Chi-Square + Random Projection	99.9720	99.9922	99.9774	99.9848
5	Random Forest	Chi-Square + RFE	99.9724	99.9868	99.9832	99.9850
5	Random Forest	Lasso	99.9487	99.9497	99.9946	99.9721
5	Random Forest	Lasso + Chi-Square	99.9189	99.9832	99.9288	99.9560
5	Random Forest	Lasso + Random Projection	99.9857	99.9890	99.9955	99.9923
5	Random Forest	Lasso + RFE	99.9634	99.9691	99.9911	99.9801
5	Random Forest	Random Projection	99.9903	99.9950	99.9945	99.9947
5	Random Forest	Random Projection + Chi-Square	99.9123	99.9452	99.9596	99.9524
5	Random Forest	Random Projection + Lasso	99.9745	99.9894	99.9829	99.9862
5	Random Forest	Random Projection + RFE	99.9910	99.9964	99.9939	99.9951
5	Random Forest	RFE	99.9618	99.9674	99.9911	99.9792
5	Random Forest	RFE + Chi-Square	99.8572	99.9601	99.8850	99.9225
5	Random Forest	RFE + Lasso	99.9453	99.9745	99.9662	99.9703
5	Random Forest	RFE + Random Projection	99.9946	99.9983	99.9958	99.9971
10	Decision Tree	Chi-Square	99.9958	99.9994	99.9961	99.9977
10	Decision Tree	Chi-Square + Lasso	99.9980	99.9996	99.9982	99.9989
10	Decision Tree	Chi-Square + Random Projection	99.9964	99.9991	99.9970	99.9980
10	Decision Tree	Chi-Square + RFE	99.9975	99.9996	99.9977	99.9987
10	Decision Tree	Lasso	99.9994	99.9998	99.9995	99.9997
10	Decision Tree	Lasso + Chi-Square	99.9977	99.9995	99.9981	99.9988
10	Decision Tree	Lasso + Random Projection	99.9965	99.9994	99.9968	99.9981
10	Decision Tree	Lasso + RFE	99.9978	99.9996	99.9980	99.9988
10	Decision Tree	Random Projection	99.9991	99.9999	99.9991	99.9995
10	Decision Tree	Random Projection + Chi-Square	99.9974	99.9991	99.9980	99.9986
10	Decision Tree	Random Projection + Lasso	99.9982	99.9997	99.9983	99.9990
10	Decision Tree	Random Projection + RFE	99.9990	99.9997	99.9992	99.9995
10	Decision Tree	RFE	99.9980	99.9996	99.9983	99.9989
10	Decision Tree	RFE + Chi-Square	99.9978	99.9995	99.9981	99.9988
10	Decision Tree	RFE + Lasso	99.9978	99.9996	99.9981	99.9988
10	Decision Tree	RFE + Random Projection	99.9969	99.9993	99.9973	99.9983
10	KNN	Chi-Square	99.9833	99.9962	99.9857	99.9909
10	KNN	Chi-Square + Lasso	99.9947	99.9978	99.9964	99.9971
10	KNN	Chi-Square + Random Projection	99.9914	99.9974	99.9932	99.9953
10	KNN	Chi-Square + RFE	99.9881	99.9933	99.9938	99.9935
10	KNN	Lasso	99.9956	99.9988	99.9964	99.9976
10	KNN	Lasso + Chi-Square	99.9915	99.9967	99.9941	99.9954
10	KNN	Lasso + Random Projection	99.9926	99.9988	99.9932	99.9960
10	KNN	Lasso + RFE	99.9911	99.9954	99.9949	99.9952



10	KNN	Random Projection	99.9164	99.9694	99.9398	99.9546
10	KNN	Random Projection + Chi-Square	99.4998	99.8572	99.6007	99.7288
10	KNN	Random Projection + Lasso	99.9909	99.9984	99.9918	99.9951
10	KNN	Random Projection + RFE	99.9546	99.9885	99.9622	99.9754
10	KNN	RFE	99.4961	99.8561	99.5979	99.7268
10	KNN	RFE + Chi-Square	99.9907	99.9948	99.9951	99.9949
10	KNN	RFE + Lasso	99.9924	99.9973	99.9944	99.9959
10	KNN	RFE + Random Projection	99.6967	99.8999	99.7712	99.8355
10	Linear SVC	Chi-Square	93.1760	99.9591	93.1353	96.4266
10	Linear SVC	Chi-Square + Lasso	97.3752	99.9286	97.2950	98.5942
10	Linear SVC	Chi-Square + Random Projection	95.0234	99.8844	94.9728	97.3666
10	Linear SVC	Chi-Square + RFE	92.9764	99.9946	92.9193	96.3272
10	Linear SVC	Lasso	99.9326	99.9526	99.9743	99.9634
10	Linear SVC	Lasso + Chi-Square	97.3067	99.9290	97.2242	98.5581
10	Linear SVC	Lasso + Random Projection	97.9373	99.8827	97.9196	98.8914
10	Linear SVC	Lasso + RFE	98.1992	99.9644	98.1160	99.0316
10	Linear SVC	Random Projection	68.2754	65.9757	99.3706	79.3010
10	Linear SVC	Random Projection + Chi-Square	46.7684	46.8690	90.9557	61.8612
10	Linear SVC	Random Projection + Lasso	99.4204	99.6468	99.7238	99.6853
10	Linear SVC	Random Projection + RFE	96.3325	99.8658	96.2903	98.0455
10	Linear SVC	RFE	92.1103	100	92.1102	95.8931
10	Linear SVC	RFE + Chi-Square	97.9686	99.9633	97.8766	98.9089
10	Linear SVC	RFE + Lasso	98.8555	99.9446	98.8262	99.3822
10	Linear SVC	RFE + Random Projection	47.8448	46.8690	93.0663	62.3420
10	Random Forest	Chi-Square	99.9948	99.9969	99.9974	99.9972
10	Random Forest	Chi-Square + Lasso	99.9968	99.9982	99.9984	99.9983
10	Random Forest	Chi-Square + Random Projection	99.9969	99.9986	99.9980	99.9983
10	Random Forest	Chi-Square + RFE	99.9908	99.9934	99.9966	99.9950
10	Random Forest	Lasso	99.9920	99.9944	99.9970	99.9957
10	Random Forest	Lasso + Chi-Square	99.9973	99.9986	99.9985	99.9985
10	Random Forest	Lasso + Random Projection	99.9974	99.9977	99.9994	99.9986
10	Random Forest	Lasso + RFE	99.9910	99.9921	99.9982	99.9951
10	Random Forest	Random Projection	99.9962	100	99.9958	99.9979
10	Random Forest	Random Projection + Chi-Square	99.9927	99.9974	99.9947	99.9961
10	Random Forest	Random Projection + Lasso	99.9955	99.9998	99.9952	99.9975
10	Random Forest	Random Projection + RFE	99.9974	99.9996	99.9976	99.9986
10	Random Forest	RFE	99.9859	99.9879	99.9968	99.9923
10	Random Forest	RFE + Chi-Square	99.9912	99.9923	99.9982	99.9952
10	Random Forest	RFE + Lasso	99.9823	99.9842	99.9966	99.9904
10	Random Forest	RFE + Random Projection	99.9932	99.9958	99.9967	99.9963
15	Decision Tree	Chi-Square	99.9980	99.9998	99.9981	99.9989
15	Decision Tree	Chi-Square + Lasso	99.9980	99.9998	99.9981	99.9989
15	Decision Tree	Chi-Square + Random Projection	99.9954	99.9995	99.9954	99.9975
15	Decision Tree	Chi-Square + RFE	99.9977	99.9995	99.9980	99.9987
15	Decision Tree	Lasso	99.9994	99.9999	99.9995	99.9997

15	Decision Tree	Lasso + Chi-Square	99.9981	99.9997	99.9982	99.9989
15	Decision Tree	Lasso + Random Projection	99.9965	99.9989	99.9973	99.9981
15	Decision Tree	Lasso + RFE	99.9979	99.9996	99.9981	99.9988
15	Decision Tree	Random Projection	99.9989	99.9997	99.9991	99.9994
15	Decision Tree	Random Projection + Chi-Square	99.9971	99.9989	99.9980	99.9984
15	Decision Tree	Random Projection + Lasso	99.9977	99.9994	99.9981	99.9988
15	Decision Tree	Random Projection + RFE	99.9991	100	99.9990	99.9995
15	Decision Tree	RFE	99.9999	99.9999	99.9999	99.9999
15	Decision Tree	RFE + Chi-Square	99.9979	99.9995	99.9982	99.9989
15	Decision Tree	RFE + Lasso	99.9976	99.9994	99.9981	99.9987
15	Decision Tree	RFE + Random Projection	99.9991	99.9999	99.9990	99.9995
15	KNN	Chi-Square	99.9932	99.9995	99.9931	99.9963
15	KNN	Chi-Square + Lasso	99.9946	99.9977	99.9964	99.9971
15	KNN	Chi-Square + Random Projection	99.9930	99.9989	99.9935	99.9962
15	KNN	Chi-Square + RFE	99.9916	99.9949	99.9960	99.9954
15	KNN	Lasso	99.9962	99.9992	99.9967	99.9979
15	KNN	Lasso + Chi-Square	99.9921	99.9964	99.9950	99.9957
15	KNN	Lasso + Random Projection	99.9948	99.9987	99.9956	99.9972
15	KNN	Lasso + RFE	99.9913	99.9960	99.9946	99.9953
15	KNN	Random Projection	99.7096	99.9123	99.7728	99.8425
15	KNN	Random Projection + Chi-Square	99.7099	99.9109	99.7745	99.8426
15	KNN	Random Projection + Lasso	99.9909	99.9984	99.9918	99.9951
15	KNN	Random Projection + RFE	99.9116	99.9653	99.9388	99.9520
15	KNN	RFE	99.5217	99.8624	99.6192	99.7407
15	KNN	RFE + Chi-Square	99.9919	99.9959	99.9953	99.9956
15	KNN	RFE + Lasso	99.9940	99.9982	99.9953	99.9967
15	KNN	RFE + Random Projection	99.7146	99.9264	99.7641	99.8452
15	Linear SVC	Chi-Square	95.5785	99.9599	95.4544	97.6552
15	Linear SVC	Chi-Square + Lasso	97.4311	99.9284	97.3527	98.6237
15	Linear SVC	Chi-Square + Random Projection	95.4976	99.9287	95.4015	97.6126
15	Linear SVC	Chi-Square + RFE	97.8582	99.9760	97.7500	98.8505
15	Linear SVC	Lasso	99.9379	99.9561	99.9765	99.9663
15	Linear SVC	Lasso + Chi-Square	97.4135	99.9284	97.3346	98.6144
15	Linear SVC	Lasso + Random Projection	99.8628	99.9307	99.9203	99.9255
15	Linear SVC	Lasso + RFE	98.5067	99.9648	98.4382	99.1956
15	Linear SVC	Random Projection	97.6914	99.9738	97.5792	98.7620
15	Linear SVC	Random Projection + Chi-Square	46.5258	46.8504	90.5223	61.7445
15	Linear SVC	Random Projection + Lasso	99.6919	99.7136	99.9519	99.8326
15	Linear SVC	Random Projection + RFE	96.3685	99.9817	96.2233	98.0665
15	Linear SVC	RFE	92.9467	99.9872	92.8974	96.3120
15	Linear SVC	RFE + Chi-Square	98.2969	99.9677	98.2152	99.0837
15	Linear SVC	RFE + Lasso	98.8764	99.9519	98.8412	99.3935
15	Linear SVC	RFE + Random Projection	96.5414	99.9746	96.4037	98.1567
15	Random Forest	Chi-Square	99.9976	99.9988	99.9987	99.9987
15	Random Forest	Chi-Square + Lasso	99.9978	99.9993	99.9984	99.9988
15	Random Forest	Chi-Square + Random Projection	99.9974	99.9994	99.9977	99.9986

15	Random Forest	Chi-Square + RFE	99.9976	99.9991	99.9983	99.9987
15	Random Forest	Lasso	99.9967	99.9997	99.9967	99.9982
15	Random Forest	Lasso + Chi-Square	99.9980	99.9996	99.9982	99.9989
15	Random Forest	Lasso + Random Projection	99.9975	99.9985	99.9988	99.9986
15	Random Forest	Lasso + RFE	99.9924	99.9933	99.9984	99.9958
15	Random Forest	Random Projection	99.9969	99.9993	99.9973	99.9983
15	Random Forest	Random Projection + Chi-Square	99.9936	99.9981	99.9950	99.9965
15	Random Forest	Random Projection + Lasso	99.9971	99.9997	99.9971	99.9984
15	Random Forest	Random Projection + RFE	99.9980	99.9996	99.9982	99.9989
15	Random Forest	RFE	99.9947	99.9961	99.9982	99.9971
15	Random Forest	RFE + Chi-Square	99.9933	99.9944	99.9984	99.9964
15	Random Forest	RFE + Lasso	99.9828	99.9841	99.9972	99.9907
15	Random Forest	RFE + Random Projection	99.9974	99.9984	99.9988	99.9986



اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

### Appendix(Feature Selected)

Number of Features Selected	Feature Selection Method	Feature Selected
2	Chi-Square	MI_dir_L5_weight', 'MI_dir_L5_mean'
2	Chi-Square + Lasso	MI_dir_L5_weight', 'MI_dir_L5_variance'
2	Chi-Square + Random Projection	N/A
2	Chi-Square + RFE	MI_dir_L0.01_weight', 'MI_dir_L0.01_mean'
2	Lasso	MI_dir_L0.1_weight', 'H_L0.1_weight'
2	Lasso + Chi-Square	MI_dir_L5_weight', 'MI_dir_L5_mean'
2	Lasso + Random Projection	N/A
2	Lasso + RFE	MI_dir_L0.1_weight', 'MI_dir_L0.01_weight'
2	Random Projection	N/A
2	Random Projection + Chi-Square	N/A
2	Random Projection + Lasso	N/A
2	Random Projection + RFE	N/A
2	RFE	MI_dir_L0.01_weight', 'MI_dir_L0.1_weight'
2	RFE + Chi-Square	MI_dir_L0.01_weight', 'MI_dir_L0.1_weight'
2	RFE + Lasso	MI_dir_L0.01_weight', 'MI_dir_L0.1_weight'
2	RFE + Random Projection	N/A
3	Chi-Square	MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance'
3	Chi-Square + Lasso	MI_dir_L5_weight', 'MI_dir_L5_variance', 'MI_dir_L1_weight'
3	Chi-Square + Random Projection	N/A

3	Chi-Square + RFE	MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.1_weight'
3	Lasso	MI_dir_L0.1_weight', 'H_L0.1_weight', 'HH_jit_L5_mean'
3	Lasso + Chi-Square	MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance'
3	Lasso + Random Projection	N/A
3	Lasso + RFE	MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L5_mean'
3	Random Projection	N/A
3	Random Projection + Chi-Square	N/A
3	Random Projection + Lasso	N/A
3	Random Projection + RFE	N/A
3	RFE	MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight'
3	RFE + Chi-Square	MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight'
3	RFE + Lasso	MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight'
3	RFE + Random Projection	N/A
5	Chi-Square	MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance', 'MI_dir_L3_weight', 'MI_dir_L3_mean'
5	Chi-Square + Lasso	MI_dir_L5_weight', 'MI_dir_L5_variance', 'MI_dir_L1_weight', 'MI_dir_L1_mean', 'MI_dir_L1_variance'
5	Chi-Square + Random Projection	N/A
5	Chi-Square + RFE	MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.1_weight', 'H_L0.01_weight', 'H_L0.01_mean'
5	Lasso	MI_dir_L0.1_weight', 'H_L0.1_weight', 'HH_L0.01_weight', 'HH_jit_L5_mean', 'HH_jit_L0.01_weight'
5	Lasso + Chi-Square	MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance', 'MI_dir_L3_variance', 'MI_dir_L1_weight'
5	Lasso + Random Projection	N/A
5	Lasso + RFE	MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L5_mean', 'H_L0.1_weight', 'H_L0.01_weight'
5	Random Projection	N/A

5	Random Projection + Chi-Square	N/A
5	Random Projection + Lasso	N/A
5	Random Projection + RFE	N/A
5	RFE	MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L0.01_magnitude'
5	RFE + Chi-Square	MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L3_weight', 'H_L0.01_weight'
5	RFE + Lasso	MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L0.01_radius'
5	RFE + Random Projection	N/A
10	Chi-Square	MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance', 'MI_dir_L3_weight', 'MI_dir_L3_mean', 'MI_dir_L3_variance', 'MI_dir_L1_weight', 'MI_dir_L1_mean', 'MI_dir_L1_variance', 'MI_dir_L0.1_weight'
10	Chi-Square + Lasso	MI_dir_L5_weight', 'MI_dir_L5_variance', 'MI_dir_L1_weight', 'MI_dir_L1_mean', 'MI_dir_L1_variance', 'MI_dir_L0.1_weight', 'MI_dir_L0.1_mean', 'MI_dir_L0.1_variance', 'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean'
10	Chi-Square + Random Projection	N/A
10	Chi-Square + RFE	MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.1_weight', 'H_L0.01_weight', 'H_L0.01_mean', 'H_L0.1_weight', 'HH_L5_radius', 'HH_L0.1_covariance', 'HH_L1_radius', 'HH_L0.1_weight'
10	Lasso	MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L5_weight', 'HH_L0.01_weight', 'HH_L0.01_mean', 'HH_jit_L5_mean', 'HH_jit_L3_mean', 'HH_jit_L0.01_weight'
10	Lasso + Chi-Square	MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance', 'MI_dir_L3_variance', 'MI_dir_L1_weight', 'MI_dir_L1_variance', 'MI_dir_L0.1_weight', 'MI_dir_L0.1_variance', 'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean'
10	Lasso + Random Projection	N/A
10	Lasso + RFE	MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L5_mean', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L0.01_std', 'HpHp_L0.1_std', 'HpHp_L0.1_weight', 'HH_jit_L0.1_variance', 'HpHp_L5_magnitude'
10	Random Projection	N/A
10	Random Projection + Chi-Square	N/A

10	Random Projection + Lasso	N/A
10	Random Projection + RFE	N/A
10	RFE	'MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L0.01_magnitude', 'HH_L0.01_radius', 'HH_L0.01_covariance', 'HH_L0.01_mean', 'HH_L0.1_mean', 'HH_jit_L0.01_variance'
10	RFE + Chi-Square	'MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L3_weight', 'H_L0.01_weight', 'HH_L0.01_radius', 'HH_L0.01_covariance', 'HH_L1_magnitude', 'HH_L0.01_magnitude', 'HH_L0.1_mean'
10	RFE + Lasso	'MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L0.01_radius', 'HH_L0.01_covariance', 'HH_L0.01_magnitude', 'HH_L1_radius', 'HH_L0.01_mean', 'HpHp_L3_std'
10	RFE + Random Projection	N/A
15	Chi-Square	'MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance', 'MI_dir_L3_weight', 'MI_dir_L3_mean', 'MI_dir_L3_variance', 'MI_dir_L1_weight', 'MI_dir_L1_mean', 'MI_dir_L1_variance', 'MI_dir_L0.1_weight', 'MI_dir_L0.1_mean', 'MI_dir_L0.1_variance', 'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.01_variance'
15	Chi-Square + Lasso	'MI_dir_L5_weight', 'MI_dir_L5_variance', 'MI_dir_L1_weight', 'MI_dir_L1_mean', 'MI_dir_L1_variance', 'MI_dir_L0.1_weight', 'MI_dir_L0.1_mean', 'MI_dir_L0.1_variance', 'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.01_variance', 'H_L5_weight', 'H_L5_variance', 'H_L1_weight', 'H_L1_mean'
15	Chi-Square + Random Projection	N/A
15	Chi-Square + RFE	'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.1_weight', 'H_L0.01_weight', 'H_L0.01_mean', 'H_L0.1_weight', 'HH_L5_radius', 'HH_L0.1_covariance', 'HH_L1_radius', 'HH_L0.1_weight', 'HH_L0.1_magnitude', 'HH_L0.1_mean', 'HH_L0.1_radius', 'HH_L1_weight', 'HH_L5_magnitude'
15	Lasso	'MI_dir_L5_weight', 'MI_dir_L1_weight', 'MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'H_L0.1_weight', 'H_L0.01_weight', 'H_L0.01_mean', 'HH_L5_weight', 'HH_L0.01_weight', 'HH_L0.01_mean', 'HH_jit_L5_weight', 'HH_jit_L5_mean', 'HH_jit_L3_mean', 'HH_jit_L0.01_weight'
15	Lasso + Chi-Square	'MI_dir_L5_weight', 'MI_dir_L5_mean', 'MI_dir_L5_variance', 'MI_dir_L3_variance', 'MI_dir_L1_weight', 'MI_dir_L1_variance', 'MI_dir_L0.1_weight', 'MI_dir_L0.1_variance', 'MI_dir_L0.01_weight', 'MI_dir_L0.01_mean', 'MI_dir_L0.01_variance', 'H_L5_weight', 'H_L5_mean', 'H_L5_variance', 'H_L3_variance'

15	Lasso + Random Projection	N/A
15	Lasso + RFE	'MI_dir_L0.1_weight', 'MI_dir_L0.01_weight', 'H_L5_mean', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L0.01_std', 'HpHp_L0.1_std', 'HpHp_L0.1_weight', 'HH_jit_L0.1_variance', 'HpHp_L5_magnitude', 'HH_L0.01_radius', 'HH_L0.01_pcc', 'HH_L0.01_mean', 'HH_L0.1_std', 'HH_L0.1_pcc'
15	Random Projection	N/A
15	Random Projection + Chi-Square	N/A
15	Random Projection + Lasso	N/A
15	Random Projection + RFE	N/A
15	RFE	'MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L0.01_magnitude', 'HH_L0.01_radius', 'HH_L0.01_covariance', 'HH_L0.01_mean', 'HH_L0.1_mean', 'HH_jit_L0.01_variance', 'HpHp_L1_magnitude', 'HpHp_L0.01_radius', 'HH_jit_L0.01_mean', 'HH_L0.01_pcc', 'HH_jit_L3_variance'
15	RFE + Chi-Square	'MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L3_weight', 'H_L0.01_weight', 'HH_L0.01_radius', 'HH_L0.01_covariance', 'HH_L1_magnitude', 'HH_L0.01_magnitude', 'HH_L0.1_mean', 'HH_L1_radius', 'HH_L0.01_mean', 'HpHp_L3_radius', 'HpHp_L3_magnitude', 'HpHp_L3_std'
15	RFE + Lasso	'MI_dir_L0.01_weight', 'MI_dir_L0.1_weight', 'H_L0.1_weight', 'H_L0.01_weight', 'HH_L0.01_radius', 'HH_L0.01_covariance', 'HH_L0.01_magnitude', 'HH_L1_radius', 'HH_L0.01_mean', 'HpHp_L3_std', 'HpHp_L5_weight', 'HpHp_L0.1_radius', 'HpHp_L0.1_std', 'HpHp_L0.1_pcc', 'HpHp_L0.01_pcc'
15	RFE + Random Projection	N/A