

DSpace Institution

DSpace Repository

<http://dspace.org>

Computer Science

thesis

2024-07

IMPROVING INTRUSION DETECTION SYSTEM USING HYBRID FEATURE SELECTION APPROACH

Mebrahtu, Gebremedhin Gebreyohannes

<http://ir.bdu.edu.et/handle/123456789/16276>

Downloaded from DSpace Repository, DSpace Institution's institutional repository



BAHIR DAR UNIVERSITY

BAHIR DAR INSTITUTE OF TECHNOLOGY

SCHOOL OF RESEARCH AND POSTGRADUATE STUDIES

FACULTY OF COMPUTING

**IMPROVING INTRUSION DETECTION SYSTEM USING HYBRID
FEATURE SELECTION APPROACH**

By

Mebrahtu Gebremedhin Gebreyohannes

Bahir Dar, Ethiopia

July, 2024

IMPROVING INTRUSION DETECTION SYSTEM USING HYBRID FEATURE
SELECTION APPROACH

Mebrahtu Gebremedhin Gebreyohannes

A thesis submitted to the school of Graduate Bahirdar Institute of Technology, BDU in
partial fulfillment of requirements for the degree of Master of Science in Computer
Science Faculty of Computing.

Advisor: **Mekuanint Agegnehu (PhD)**

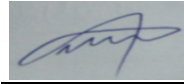
BahirDar, Ethiopia

July, 2024

DECLARATION

This is to certify that the thesis entitled “IMPROVING INTRUSION DETECTION SYSTEM USING HYBRID FEATURE SELECTION APPROACH”, submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science under Computing Faculty, Bahir Dar Institute of Technology, is a record of original work carried out by me and has never been submitted to this or any other organization to get any other degree or certificates. The guidance and support I received through this process have been properly recognized.

Mebrahtu Gebremedhin Gebreyohannes



July 30, 2024

Name of student


Signature

Date

**BAHIR DAR UNIVERSITY
BAHIR DAR INSTITUTE OF TECHNOLOGY
SCHOOL OF GRADUATE STUDIES
FACULTY OF COMPUTING**

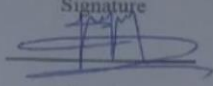
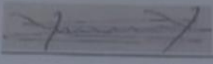
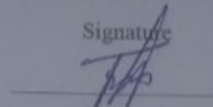
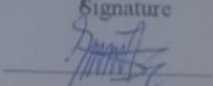
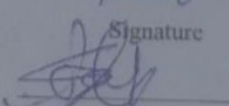
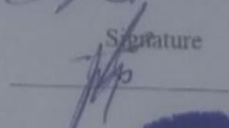
Approval of thesis for defense result

I hereby confirm that the changes required by the examiners have been carried out and incorporated in the final thesis.

Name of Student Mebrahtu Gebremedhin Signature  Date July 30, 2024

As members of the board of examiners, we examined this thesis entitled "IMPROVING INTRUSTION DETECTION SYSTEM USING HYBRID FEATURE SELECTION APPROACH" by Mebrahtu Gebremedhin. We hereby certify that the thesis is accepted for fulfilling the requirements for the award of the degree of Masters of Science in "Computer Science".

Board of Examiners

Name of Advisor	Signature	Date
<u>Mekuanint Agegnehu (PhD)</u>		_____
Name of External examiner	Signature	Date
<u>Yirga Yayehu (PhD)</u>		<u>July 26, 2024</u>
Name of Internal Examiner	Signature	Date
<u>Tesfa Tegegne (PhD)</u>		_____
Name of Chairperson	Signature	Date
<u>Mr. Seffi Gebevehu</u>		<u>Aug 15/2024</u>
Name of Chair Holder	Signature	Date
<u>Ms. Kidist Meshesha</u>		<u>Aug 15/2024</u>
Name of Faculty Dean	Signature	Date
<u>Tesfa Tegegne (PhD)</u>		_____

Faculty Stamp



ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Mekuanint Agegnehu for their unwavering support, guidance, and encouragement throughout this research work. Their insightful feedback and invaluable expertise were crucial to the completion of this thesis.

Finally, I must express my profound gratitude to my family for their unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This completion would not have been possible without their support.

ABSTRACT

With the rapid increase in intrusion attempts exhibiting nonlinear behavior, network traffic behaves unpredictably, and there is a massive feature in the problem domain, intrusion detection systems pose a complex challenge. Dealing with high-dimensional and imbalanced datasets becomes an obstacle in real-world applications like intrusion detection systems (IDS). To overcome this problem we adopted feature selection considering the classification performance and computational efficiency. In this research work, we propose MI-RFE, a hybrid feature selection method tasked with a binary class intrusion detection system that exploits the qualities of both a filter method chosen because of its speed and a wrapper method because of its relevance in search. In the first phase of our approach, we utilize Mutual Information (MI) for its computational efficiency and ability to handle nonlinear datasets to rank the features based on their importance. In the second phase, we employ recursive feature elimination (RFE) which is a machine learning-based wrapper method to further reduce the feature dimensions. Additionally, we apply the Synthetic Minority Over-sampling Technique (SMOTE) to address class imbalances in the dataset. The optimal features obtained from the proposed method were evaluated using a Decision Tree (DT), K-nearest neighbors (KNN), Random Forest (RF), and XGBOOST and these algorithms were then combined using Stacking (DT + KNN + RF + XGBoost) techniques to improve their performance. Our experimental results obtained based on the CICIDS 2017 dataset confirmed that the proposed method improves the performance and computational time. The results show that the feature is reduced from 78 to 25 while the accuracy of DT is improved from 93.94% to 99.34% and we achieve 99.92% by Stacking (DT + KNN + RF + XGBoost) using the reduced features.

Keywords: MI, RFE, Stacking, SMOTE

TABLE OF CONTENTS

DECLARATION	ii
ACKNOWLEDGEMENTS	iv
ABSTRACT	v
LIST OF ABBREVIATIONS	ix
LIST OF FIGURES	xi
LIST OF TABLES	xii
CHAPTER ONE	1
INTRODUCTION	1
1.1. Background	1
1.2. Motivation and Statement of Problem	3
1.3. Objective	4
1.3.1. General Objective	4
1.3.2. Specific Objectives	4
1.4. Scope and Limitation	4
1.5. Significance of the Study	5
1.6. Organization of the research	6
CHAPTER TWO	7
LITERATURE REVIEW	7
2.1 Cyber Security	7
2.2 Intrusion detection system	8
2.2.1 IDS Classification by Detection Methods	9
2.2.1.1 Signature-based intrusion detection systems	9
2.2.1.2 Anomaly-based intrusion detection system (AIDS)	9
2.3 Machine learning	10
2.3.1 Supervised Learning	11
2.3.2 Unsupervised Learning	11

2.3.3	Semi-Supervised Learning	11
2.3.4	Machine learning Algorithms	12
2.3.4.1	Decision Tree	12
2.3.4.2	Random Forest	13
2.3.4.3	K-Nearest Neighbors	14
2.3.4.4	XGBoost	15
2.3.4.5	Stacking.....	16
2.4	Feature selection methods	16
2.4.1	Filter feature selection method	17
2.4.1.1	Mutual information	17
2.4.2	Wrapper feature selection Method	18
2.4.3	Hybrid feature selection method.....	18
2.5	Related work	19
2.5.1	Gap Analysis.....	24
CHAPTER THREE		25
METHODOLOGY		25
3.1	Dataset collection and preparation	25
3.1.1	Dataset description	26
3.2	Data preprocessing	32
3.3	Feature Selection	34
3.3.1	Mutual Information Feature Selection.....	35
3.3.2	Recursive Feature Elimination Feature Selection	37
3.4	Synthetic Minority Oversampling Technique (SMOTE).....	39
3.5	Tools used for Experiment	40
3.5.1	The Hardware Tools Used.....	40
3.5.2	The Software Tools Used	40
3.6	System Architecture of Proposed Method	42
CHAPTER FOUR.....		44
RESULTS AND DISCUSSION		44
4.1	Dataset used for Experiments.....	44
4.2	Handling Class Imbalance.....	44
4.3	Performance Evaluation Metrics	47

4.4	Hyper-parameters	48
4.5	Experiments and Results	50
4.5.1	Experimental Results of Feature Selection Using Mutual Information.....	51
4.5.2	Experimental Results of Feature Selection Using RFE	52
4.5.3	Confusion Matrix Experiment Result	54
4.5.4	Experimental results without feature selection with full features.....	59
4.5.5	Experimental results of features selected by MI.....	62
4.5.6	Experimental results of hybrid feature selection (MI + RFE)	64
4.5.7	Sample Validation results	69
4.6	Answer to the research questions	71
CHAPTER FIVE		74
CONCLUSIONS AND RECOMMENDATIONS		74
5.1	Conclusions	74
5.2	Recommendations	75
REFERENCES		76

LIST OF ABBREVIATIONS

AIDS	Anomaly Intrusion Detection System
ANN	Artificial Neural Network
CIC	Canadian Institute for Cyber security
CVS	Comma Separated Values
DDOS	Distributed Denial of Service
DE	Differential Evolution
DoS	Denial-of-service
DT	Decision Tree
ELM	Extreme Learning Machine
FS	Feature selection
HIDS	Host Intrusion Detection Systems
IDS	Intrusion detection system
IoT	Internet of Things
IPS	Intrusion Prevention System
KNN	K-Nearest Neighbors
LDA	Linear Discriminant Analysis
LR	Logistic Regression
MI	Mutual Information
ML	Machine Learning

NIDS	Network Intrusion Detection System
RF	Random Forest
RFE	Recursive Feature Elimination
SIDS	Signature Intrusion Detection System
SMOTE	Synthetic Minority Over-sampling Technique
SVM	Support Vector Machine

LIST OF FIGURES

Figure 2. 1 Structure of DT algorithm. (Hafeez et al., 2021)	12
Figure 2. 2 How RF is working.....	14
Figure 2. 3 Classification of Feature selection approach	19
Figure 3. 1 Class distribution of the total CICIDC 2017 dataset	33
Figure 3. 2 Selecting 20% of CICIDS 2017 dataset	33
Figure 3. 3 Data cleaning output	34
Figure 3. 4 How SMOTE works	39
Figure 3. 5 System architecture for the proposed model	42
Figure 4. 1 Experiment result of SMOTE in Training dataset.....	45
Figure 4. 2 Mutual Information (MI) Score of all the features	51
Figure 4. 3 Selected features by MI.....	52
Figure 4. 4 Feature importance score of the selected features.....	52
Figure 4. 5 Selected features using Recursive feature elimination (RFE)	53
Figure 4. 6 Confusion matrix for DT	54
Figure 4. 7 Confusion matrix for KNN	55
Figure 4. 8 Confusion matrix for RF.....	56
Figure 4. 9 Confusion matrix for XGBoost	57
Figure 4. 10 Confusion matrix for Stacking (DT + KNN + RF + XGBoost)	58
Figure 4. 11 Accuracy Comparison of Full Features.....	61
Figure 4. 12 Accuracy Comparison of Reduced Features.....	66
Figure 4. 13 Accuracy Comparison of the 25 and 52 Features.....	67
Figure 4. 14 Accuracy Comparison of the 25, 52 and full (78) Features	68
Figure 4. 15 Cross-validation for DT	69
Figure 4. 16 Cross-validation for RF	69
Figure 4. 17 Cross-validation for stacking (DT+ KNN + RF + XGBoost).....	70

LIST OF TABLES

Table 2. 1 Comparison of IDS based on detection methods.....	10
Table 2. 2 Summary of the related works.....	21
Table 3. 1 Victim and Attacker Network Information in the CICIDS 2017 Dataset	26
Table 3. 2 Description of files containing the CICIDS2017 dataset with attack found	27
Table 3. 3 The 78 features and the label class of the CICIDS 2017 dataset record ..	28
Table 4. 1 Distribution of label class in training and testing before applying SMOTE	45
Table 4. 2 Distribution of label class in training and testing after applying SMOTE	46
Table 4. 3 Confusion matrix for two class classification	47
Table 4. 4 Hyper-parameters for the classifier algorithm	49
Table 4. 5 Experiment results using full features	59
Table 4. 6 Experiment result of 52 feature selected by MI	62
Table 4. 7 Experiment results using reduced features	64
Table 4. 8 Comparison of training and testing time	70

CHAPTER ONE

INTRODUCTION

1.1. Background

In our modern world, the Internet has become an integral part of everyone's daily life, enabling individuals to carry out essential activities seamlessly. With the growing reliance on digitalization and the Internet of Things (IoT), there has been a significant rise in security incidents including unauthorized access, malware attacks, zero-day exploits, data breaches, denial-of-service (DoS) attacks, and social engineering or phishing attempts, among others, expanding at an exponential pace in recent times(Sarker et al., 2020).

Cyber security is the proactive defense of systems, networks, and software against digital attacks. These cyber-attacks typically target sensitive information, aim to extort money or disrupt business operations. The evolving landscape, with more devices than people and increasingly sophisticated attackers, makes implementing effective cyber security measures a significant challenge. It entails safeguarding vital data and devices and ensuring confidentiality, integrity, and availability. While preventing breaches entirely may seem unattainable, security mechanisms aim to deter unauthorized access. Intrusion Detection, a field of research, focuses on identifying and responding to intrusion attempts, and mitigating potential damage(B. B. Gupta et al., 2020).

Intrusion detection involves systematically observing the activities within a computer system or network, and examining them for indications of potential incidents. These incidents may include breaches or impending threats to computer security policies, acceptable use guidelines, or established security protocols(Thakkar & Lohiya, 2022). According to (Thakkar & Lohiya, 2022) intrusion detection systems (IDSs) are typically categorized into two primary types: anomaly detection systems and misuse detection systems, also known as signature-based systems. Anomaly detection systems operate by establishing rules that define normal behavior within a system. Any activity that

significantly deviates from these established norms is flagged as potentially suspicious or indicative of an attack. In contrast, misuse detection systems maintain a database of known attack signatures or patterns. When monitoring network traffic or system activity, the system compares this data against the stored signatures to identify known threats. However, this approach may be limited in detecting novel attacks that do not match existing signatures.

According to (Thakkar & Lohiya, 2022) intrusion detection systems (IDSs) can also be categorized based on their installation method into Network Intrusion Detection Systems (NIDS) and Host Intrusion Detection Systems (HIDS). NIDS is strategically deployed at specific points within the network to monitor inbound and outbound traffic across all network devices. Conversely, HIDS operates directly on individual computers or devices within the network, providing monitoring capabilities for those systems with both internet and internal network access.

IDS can't be perfect, primarily because network traffic is so complicated(Thakkar & Lohiya, 2022). According to those authors errors in IDS can be classified into two categories: false positives and false negatives. A false positive occurs when the IDS mistakenly identifies legitimate activity as malicious, generating a false alarm. On the other hand, a false negative represents a more severe scenario where the IDS fails to detect an actual attack, classifying malicious activity as acceptable. This failure to recognize an attack poses the most serious risk as security professionals remain unaware of the breach. While false positives are inconvenient and can lead to complications, false negatives present a significant security concern.

Feature Selection (FS) is the act of identifying relevant features or a potential subset of features from a larger set of available features(Kumar, 2014). The utilization of evaluation criteria is essential in acquiring an optimal feature subset, a critical aspect for enhancing the efficiency of Intrusion Detection Systems (IDS). Given the extensive network connections and massive data flow on the Internet, IDS encounters challenges in accurately detecting attacks. Furthermore, the presence of irrelevant and redundant features significantly affects the quality of IDS, impacting both detection accuracy and processing costs. Consequently, Feature Selection (FS) emerges as a pivotal technique aimed at bolstering detection performance among these complexities(Sofiane Maza &

Mohamed Touahria, 2018). In this research work to reduce the irrelevant and redundant features, we use the FS approach. In general, two different approaches for FS can be distinguished(Jabali, 2017): filter and wrapper approaches. Using a filter approach, the selection of appropriate features is based on distance and information measures in the feature space and is carried out completely independently from the classifier deployed. In contrast, with a wrapper approach, the selection of features is based on the classifier's accuracy using a machine-learning classification or learning algorithm. Although a filter approach might be faster, we can apply a wrapper approach to achieve better classification results. To take advantage of both the filter and wrapper approach in this study we applied the Hybrid FS approach.

1.2. Motivation and Statement of Problem

In our modern world, the internet is a vital tool for work and everyday life, providing countless benefits. However, it also brings challenges, especially when it comes to security. With the internet expanding rapidly, we are seeing more and more cyber-attacks and intrusions. These attacks can cause serious harm to our computer systems and result in significant data loss. That's why it's crucial to develop effective ways to detect and prevent these harmful activities, ensuring the safety and reliability of our systems(B. B. Gupta et al., 2020).

Because intrusion attempts exhibit nonlinear behavior, network traffic behaves unpredictably, and there is a massive feature in the problem domain, intrusion detection systems pose a complex challenge(Aghdam & Kabiri, 2016). Given the size and nature of intrusion detection datasets, intrusion detection systems (IDS) also often require high computational complexity to analyze data features and identify intrusive patterns (Umar et al., n.d.). Redundant and irrelevant features in IDS data have been a persistent issue in network traffic classification.

Selecting effective and essential features is crucial in network IDS to enhance the detection rate of IDS models. While most IDSs employ a single-feature selection approach to categorize network traffic data as either normal behavior or anomalous, these single-feature selection approaches often fall short of achieving the optimal attack detection rate while maintaining a low false alarm rate(Panda et al., 2012). As other researchers start finding out different ways to increase accuracy and improve the model

of the intrusion detection system, there is still enough scope to increase the accuracy and efficiency of the model. In this area, the authors in(Khan et al., 2020) recommend future works to build a hybrid model to improve the accuracy of the model and to speed up the computation time of the model. In this study, we proposed a hybrid FS approach and stacking classification method.

In the end, this study answers the following research questions:

1. What are the features selected through the Hybrid Feature Selection method?
2. Does implementing a Hybrid Feature Selection method improve the performance of the model?
3. Does a combining classifier algorithm enhance the performance of the proposed model?

1.3. Objective

The objective of the study is described as general and specific objectives.

1.3.1. General Objective

The general objective of this thesis is to build a machine-learning intrusion detection model using a Hybrid feature selection approach.

1.3.2. Specific Objectives

- To propose an effective and efficient classifier model for the IDS using the Hybrid FS approach.
- To explore features best suited for the IDS implementation.
- To combine classifier algorithms to enhance the performance of the proposed model.
- To compare the effectiveness of the proposed method with single Methods.

1.4. Scope and Limitation

This research paper focuses on building an intrusion detection model using a hybrid feature selection approach. The approach combines the strengths of both filter and wrapper methods to handle nonlinear datasets and reduce feature dimensions. Mutual Information (MI) from the filter method and Recursive Feature Elimination (RFE) from the wrapper method are selected for feature selection. Decision Tree (DT), K-nearest

neighbors (KNN), Random Forest (RF), and XGBoost are selected for classification and stacking techniques is used to combine those classifiers.

Due to the challenge of obtaining a local dataset, the data used for this thesis work was obtained from publicly available data sources from the Canadian Institute of Cyber Security which is CICIDS 2017, and due to the lack of a high-performance computer, we used 20% of the data for our experiments.

1.5. Significance of the Study

This study addresses pressing issues in the field of cyber security, particularly in the realm of intrusion detection systems (IDS). By proposing a hybrid feature selection method (MI-RFE) and incorporating advanced machine learning techniques, the research offers substantial improvements in detecting cyber-attacks, which is of paramount importance in today's digital age where security threats are increasingly sophisticated and prevalent. The combination of MI from filter and RFE from wrapper methods for feature selection with ensemble learning techniques improves the performance, efficiency, and robustness of intrusion detection models, leading to better overall results. This approach leverages the strengths of each method, resulting in a highly effective IDS framework that can be adapted for various cyber security applications and improving cyber security measures across various sectors, by safeguarding critical digital infrastructure and sensitive information from malicious attacks.

The following are the main contributions of this research work;

- We build a hybrid feature selection method named MI-RFE for intrusion detection systems.
- Our proposed model successfully reduces feature dimensionality from 78 to 25 with accuracy of 99.34% for Decision Trees (DT), compared to 93.94% accuracy before feature selection.
- By employing a stacking approach that combines Decision Trees (DT), K-Nearest Neighbors (KNN), Random Forest (RF), and XGBoost classifiers, we get an accuracy of 99.92% with reduced features and 99.75% with the full feature set.

- Our proposed feature selection method not only enhances model performance but also reduces computational overhead by decreasing the training and testing time of the classifier.

1.6. Organization of the research

The research is organized into five different chapters.

Chapter One: This chapter presents an introduction to the study; including background, problem statement, objective, and significance of the study this provides sufficient information about the study to assist the reader to know the purpose of the study.

Chapter Two: This chapter presents a literature review that explains cyber security, IDS, feature selection approaches, some machine learning algorithms, and related works that are discussed.

Chapter Three: In this chapter, we discussed methods and methodology conducted for the model building including the dataset used, the preprocessing of the dataset, steps in feature selection, and the block diagram of the proposed model.

Chapter Four: In this chapter, we discussed on results and discussion of the proposed model using the selected dataset. We conduct experiments using the Anaconda Navigator tool which has a Jupyter Notebook interface.

Chapter Five: In this chapter, we discussed the conclusions and recommendations.

CHAPTER TWO

LITERATURE REVIEW

2.1 Cyber Security

The widespread adoption and usage of the Internet and mobile applications have expanded the realm of cyberspace. Consequently, cyberspace has become increasingly susceptible to automated and prolonged cyber-attacks. Cyber security techniques play a crucial role in bolstering security measures to detect and respond effectively to such cyber threats(Shaukat et al., 2020).

According to (Humayun et al., 2020) cyber security encompasses a comprehensive collection of elements including strategies, policies, standards, procedures, guidelines, risk mitigation strategies, training, techniques, technologies, tools, and processes. These components work in concert to safeguard the confidentiality, integrity, and availability of computing resources, networks, software programs, and assets from potential attacks. Cyberspace represents a worldwide field within the realm of information, distinguished by its utilization of the electronic and electromagnetic spectrum. Its defining feature lies in the creation, updating, storage, sharing, and exploitation of information through interconnected and interdependent networks, facilitated by cutting-edge information and communication technologies(Humayun et al., 2020). Currently, cyber security stands as a pressing concern in the realm of cyberspace, demonstrating a marked escalation across diverse application domains, including finance, industry, healthcare, and other vital sectors(Alom & Taha, 2017). Cyber security defense mechanisms can be implemented at various levels, including application, network, host, and data. Examples of these defense mechanisms include access control, authentication methods, encryption techniques, firewalls, antivirus software, intrusion detection systems (IDSs), and intrusion prevention systems (IPSs). These measures work to both prevent attacks and detect security breaches(Berman et al., 2019). According to (Sarker et al., 2020) the primary objectives

of cyber security revolve around ensuring the confidentiality, availability, and integrity of information.

- **Confidentiality** is a technique used to prevent the access and disclosure of information to unauthorized individuals, entities, or systems.
- **Integrity** is a technique used to prevent any modification or destruction of information in an unauthorized manner.
- **Availability** is a technique used to ensure timely and reliable access to information assets and systems to an authorized entity.

2.2 Intrusion detection system

The rapid growth of technology has certainly made life easier, yet it has also brought forth an excess of security concerns. With the evolution of the internet, the frequency of cyber-attacks has surged over the years. Intrusion Detection Systems (IDS) play a vital role in maintaining a secure environment for businesses by safeguarding against suspicious network activities.

According to (Khraisat et al., 2019) intrusion is characterized as any unauthorized activity that results in harm to an information system. This encompasses any attack that could potentially endanger the confidentiality, integrity, or availability of information, thus qualifying as an intrusion.

An Intrusion Detection System (IDS) is a software or hardware mechanism designed to detect and identify malicious or unauthorized activities on computer systems. Its primary function is to uphold system security by promptly identifying and responding to potential threats (Khraisat et al., 2019). The internet's growing impact on modern life underscores the significance of cyber security as a crucial area of research. Cyber security protection strategies primarily encompass anti-virus software, firewalls, access control, endpoint security, intrusion prevention systems, and intrusion detection systems (IDSs). These techniques are instrumental in safeguarding networks from both internal and external threats. Specifically, an IDS serves as a pivotal detection system, monitoring the operational states of software and hardware within a network to bolster cyber security (Liu & Lang, 2019a).

An IDS is a computer security application designed to identify various security breaches, spanning from attempted external break-ins to system intrusions and misuse by internal

users (Liu & Lang, 2019a). The primary functions of IDSs include monitoring hosts and networks, analyzing the behaviors of computer systems, generating alerts, and responding to suspicious activities. Due to their role in monitoring connected hosts and networks, IDSs are commonly deployed in proximity to protected network nodes, such as switches within major network segments.

Based on (Liu & Lang, 2019a) IDSs are classified into two main categories: detection-based methods and data source-based methods. Detection methods further categorize IDSs into misuse detection and anomaly detection. Data source methods classify IDSs into host-based and network-based methods.

2.2.1 IDS Classification by Detection Methods

2.2.1.1 Signature-based intrusion detection systems

Signature Intrusion Detection Systems (SIDS) rely on pattern-matching techniques to identify known attacks. These systems are also referred to as Knowledge-based Detection or Misuse Detection (Khraisat et al., 2019). In SIDS, matching methods are employed to identify a past intrusion. Simply, when an intrusion signature aligns with the signature of a previous intrusion stored in the signature database, an alarm signal is activated.

One of the primary advantages of SIDS is its ease of development and comprehension, particularly when there is a clear understanding of network traffic behavior and system activity (Jose et al., 2018). SIDS also excels in combating attacks with fixed behavioral patterns.

The primary disadvantages of SIDS include the constant need for updating and maintaining the collection of signatures, which can fail to detect unique attacks. Additionally, they struggle to effectively handle self-modifying behavioral characteristics (Jose et al., 2018).

2.2.1.2 Anomaly-based intrusion detection system (AIDS)

AIDS establishes a baseline or learned pattern of typical system activity to discern active intrusion attempts. When deviations from this baseline or pattern occur, an alarm is triggered. In an anomaly detection engine, events are generated by any behaviors that deviate from the predefined or accepted model of behavior (Jose et al., 2018).

In AIDS, a computer system's normal behavior model is established using machine learning, statistical-based, or knowledge-based techniques. Any substantial deviation

between observed behavior and the model is classified as an anomaly, potentially indicating an intrusion. The development of AIDS typically involves two phases: the training phase and the testing phase. During the training phase, the normal traffic profile is utilized to construct a model of typical behavior. Following this, during the testing phase, a new dataset is utilized to evaluate the system's capability to generalize to intrusions that it has not encountered before (Khraisat et al., 2019).

The primary advantage of AIDS lies in its capability to detect zero-day attacks, as it doesn't depend on a signature database for recognizing abnormal user activity (Khraisat et al., 2019). The primary drawback of an anomaly detection system is the challenge of defining rules. It requires precise definition, implementation, and difficult testing of all protocols being analyzed to ensure accuracy (Jose et al., 2018).

Table 2. 1 Comparison of IDS based on detection methods.

Anomaly-based	Misuse or Signature-based
Detects unknown attacks and vulnerabilities along with known attacks.	Effective in identifying known attacks by performing contextual analysis
It is less dependent on the operating system and rather examines the network patterns for identifying attacks.	It depends on its software and operating system to identify both attacks and vulnerabilities
It builds profiles of the observed network communication for identifying the attack patterns.	The attack signature database should be updated regularly.
The anomaly-based IDS conducts protocol analysis to examine the specifics of packets.	The signature-based IDS have a minimum knowledge of protocols.
Low missed alarm rate; High false alarm rate	Low false alarm rate; High missed alarm rate

2.3 Machine learning

Machine learning (ML) is defined as the scientific exploration of algorithms and statistical models that empower computer systems to accomplish specific tasks without the need for explicit programming, as stated (Mahesh, 2018). Machine learning (ML) is employed to instruct machines on handling data more efficiently. In the context of

Intrusion Detection Systems (IDS), ML algorithms enable more accurate detection of attacks within large volumes of data in shorter timeframes(Saranya et al., 2020). ML algorithms can be classified into three categories(Saranya et al., 2020).

- Supervised
- Unsupervised
- Semi-supervised

2.3.1 Supervised Learning

Supervised learning is a machine learning task focused on learning a function that links an input to an output, guided by example input-output pairs. It deduces a function from labeled training data, which comprises a collection of training examples(Mahesh, 2018). Supervised learning pertains to fully class-labeled data, establishing the relationship between data and its respective class. This can be achieved through either classification or regression. Classification involves two key steps: training and testing. During training, the model learns from the labeled data with the assistance of the response variable(Saranya et al., 2020).

2.3.2 Unsupervised Learning

Unsupervised learning extracts valuable feature information from unlabeled data, simplifying the process of acquiring training data. However, the detection performance of unsupervised learning methods typically falls short of that achieved by supervised learning methods(Liu & Lang, 2019b). In intrusion detection, unsupervised learning algorithms aim to uncover hidden structures within unlabeled data. Unsupervised learning, unlike supervised learning, operates independently of any training data. This can be accomplished through methods such as clustering, association analysis, or dimensionality reduction(Saranya et al., 2020). In unsupervised learning, we lack knowledge of the correct output for a given input. Clustering stands out as one of the most widely used unsupervised approaches for detecting intrusion(A. R. bhai Gupta & Agrawal, 2020).

2.3.3 Semi-Supervised Learning

The semi-supervised machine learning (ML) algorithm resides between unsupervised learning and supervised learning. These techniques leverage both unlabeled data for training and a small portion of labeled data, especially in scenarios involving a large set of unlabeled data(Saranya et al., 2020). These algorithms are employed when we

possess a portion of data where the output is known for the corresponding input, while the output remains unknown for the rest of the data (A. R. bhai Gupta & Agrawal, 2020).

2.3.4 Machine learning Algorithms

2.3.4.1 Decision Tree

Decision Tree (DT) serves as a supervised ML technique utilized for classification and regression tasks. The resultant model from the decision-making process resembles a tree structure, making DT relatively straightforward for users to interpret. Additionally, many ML tools offer the capability to visualize the resulting trees, further aiding in comprehension(Kasongo & Sun, 2020a).

A decision tree is a graphical representation of choices and their outcomes in the form of a tree structure. Nodes in the graph depict events or decisions, while edges represent decision rules or conditions. Each tree comprises nodes and branches, with each node representing attributes grouped for classification and each branch indicating a possible value the node can assume(Mahesh, 2018).

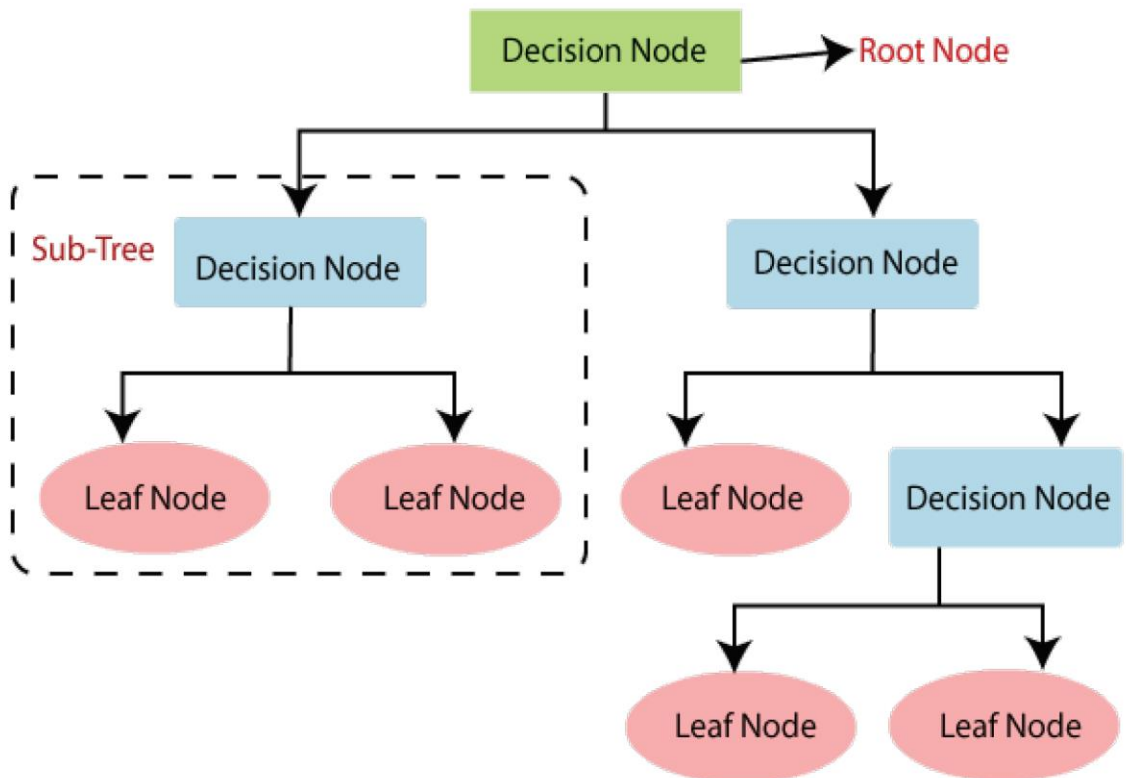


Figure 2. 1 Structure of DT algorithm. (Hafeez et al., 2021)

Advantages:

The interpretation of a Decision Tree (DT) classifier is straightforward and can be easily grasped with a concise explanation. Inferences can be drawn from various probability estimations and costs, enabling the generation of precise outputs. Furthermore, it seamlessly integrates with other classification models, enhancing accuracy. DT classifiers excel in scenarios where the model is familiar with existing intrusion methods and scenarios, showcasing superior performance(Thakkar & Lohiya, 2021).

Disadvantages:

The DT classifier lacks adaptability to minor data alterations, often leading to unstable decision tree structures even with slight changes. When handling similar datasets, its accuracy tends to be relatively lower. Complexity arises when deriving nodes, especially with interconnected or uncertain data. Consequently, it's ill-suited for problems with limited information availability(Thakkar & Lohiya, 2021).

2.3.4.2 Random Forest

Random Forest is a prominent machine learning algorithm that employs a bagging approach to generate multiple decision trees, each trained on a random subset of the data. Through repeated training on random samples of the dataset, the Random Forest algorithm aims to achieve robust prediction performance. In this ensemble learning technique, the output of all decision trees within the Random Forest is aggregated to form the final prediction. This can be accomplished either by polling the results of each decision tree or by selecting the prediction that occurs most frequently across the decision trees. RF constructs many decision trees based on the instances of class and the root of a random forest tree is selected by suitable voting from each class of the tree constructed(Thakkar & Lohiya, 2021). RF works by creating a forest of decision trees and merges them to form a more accurate model that can be used for both classification and regression(Jhaveri et al., 2019).

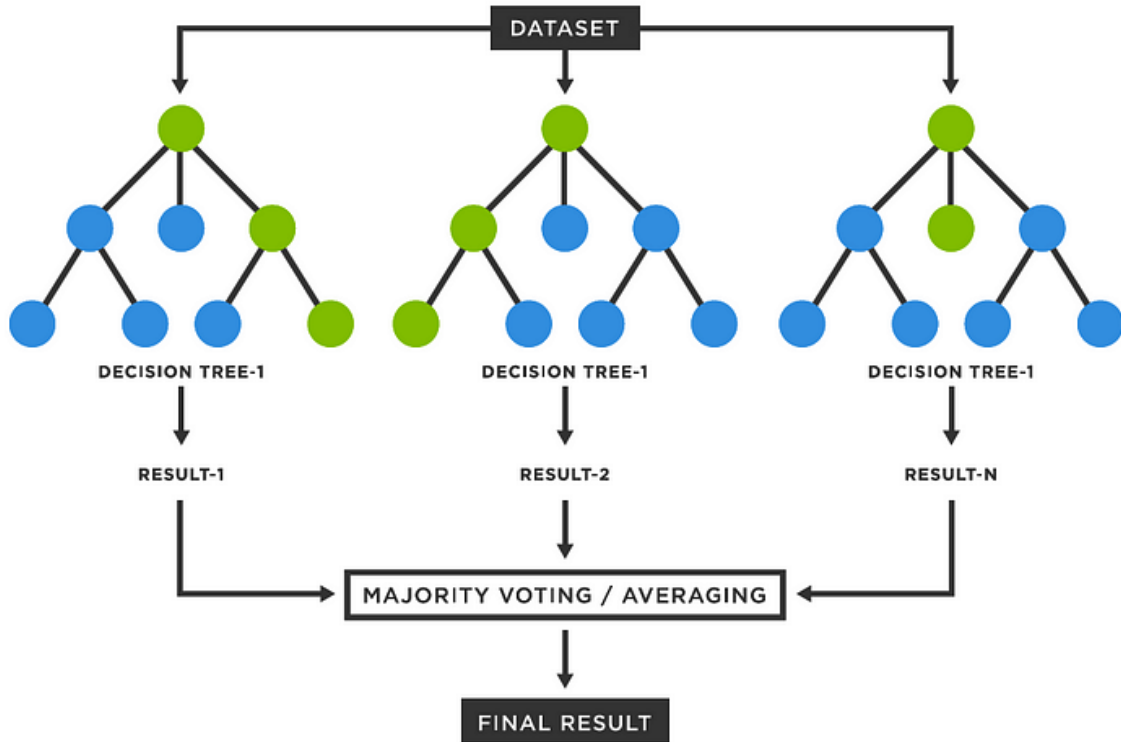


Figure 2. 2 How RF is working

Advantages:

Random Forest (RF) applies even to large datasets with numerous features. It evaluates the significance of each feature individually, ensuring balanced model performance. Notably, RF mitigates over-fitting issues and adeptly manages unbalanced datasets(Thakkar & Lohiya, 2021).

Disadvantages:

RF model isn't straightforward due to the numerous trees it generates. These complexities can difficult real-time classification, predictions, especially with a large number of trees. Consequently, the model's speed decreases as the number of trees increases(Thakkar & Lohiya, 2021).

2.3.4.3 K-Nearest Neighbors

K-nearest neighbor (KNN) is a supervised classification technique that assigns a class to data based on the class of its nearest neighbor. The algorithm determines the classes by considering the value of k, which represents the number of nearest neighbors to consider. It predicts the class of a data sample by assessing consistency and distance

with its closest neighbor. Distance metrics such as Euclidean and Manhattan distances are commonly used to measure the distances between data points and their nearest neighbors. Since all data points are stored in memory, KNN is often referred to as a memory-based technique. To enhance algorithm performance, weights can be assigned to training points based on their distances from the data points. However, managing computational complexity and memory requirements are significant concerns for this technique. These challenges can be addressed by reducing the dataset size or eliminating data points that do not contribute to recurring patterns(Thakkar & Lohiya, 2021).

Advantages:

It is a cost-effective algorithm as no computational time is required to learn the data. Even with large datasets, it performs effectively by employing simple methods to assume local approximation.

Disadvantages:

The interpretation of the model is very complex due to the absence of any description of the training data. The learning process can be costly, given the laborious task of identifying the k-nearest neighbors, particularly when dealing with extensive datasets stored in memory. It might require a very large dataset. The performance of the algorithm is completely dependent upon the attributes selected for computation and hence, it results in a curse of dimensionality for the dataset considered(Thakkar & Lohiya, 2021).

2.3.4.4 XGBoost

XGBoost stands for Extreme Gradient Boosting, which implements a gradient boosting technique using decision trees. It constructs small, simple decision trees iteratively, with each tree acting as a weak learner due to its high bias. XGBoost starts by building an initial basic tree that performs poorly. The next tree is trained to predict the errors of the first tree, improving upon what the initial weak learner couldn't capture. This process continues, with each new tree correcting the mistakes of the previous ones until a stopping condition is met, such as the predetermined number of trees (estimators) to be created. XGBoost also offers significant advantages: it trains quickly and can be parallelized or distributed across clusters for enhanced performance(Arif Ali et al., 2023). XGBoost combines weak classifiers to create a strong model. It incorporates feedback from previously constructed decision trees. Each iteration of gradient boosting optimizes

the loss function, aiming to minimize the residuals from the previous step. These residuals represent the difference between the predicted values and the true values. By iteratively refining these predictions, XGBoost enhances the overall model accuracy(Faysal et al., 2022).

2.3.4.5 Stacking

Stacking is an ensemble machine learning method that permits the mixture of two or more classifier algorithms and prepares a final model with accurate prediction. The best thing about the stacking ensemble is that it often used the benefits of the various algorithms at the identical time and therefore the capabilities of well-performing models to solve classification and regression problems.

Stacking is an effective strategy as it serves as a useful framework that integrates various ensemble methods. It operates on two levels of learning: base learning and meta-learning. During base learning, initial (base) learners are trained using the training dataset. Subsequently, these base learners generate a new dataset for the meta-learner. The meta-learner is then trained using this new dataset. Once trained, the meta-learner is utilized to classify the test set. A critical aspect of stacking lies in selecting the best base learner. Instead of relying on a single base learner, employing multiple base learners enhances performance on the training dataset(Rajadurai & Gandhi, 2022).

2.4 Feature selection methods

Feature selection involves the elimination of irrelevant and redundant features from a dataset, aiming to enhance the performance of machine learning algorithms in terms of accuracy and the time required to build the model (Asir et al., 2016). Feature selection methods are crucial in enhancing the performance and accuracy of machine learning algorithms. Feature selection (FS) serves as an essential preprocessing technique, enabling the selection of an optimal subset of relevant features from the original feature set for constructing machine learning models(Umar et al., 2021). This is accomplished by reducing the number of features through the removal of irrelevant, redundant, or noisy data, which directly impacts the performance of the subsequent model built.

According to (Umar et al., 2021), The typical feature selection procedures encompass the following steps.

1. **Subset generation:** produces candidate feature subsets for evaluation based on search starting point, direction, and strategy.
2. **Subset evaluation:** evaluates and compares candidate subset with the preceding best subset based on certain evaluation criteria. If the new subset happens to outperform the previous best subset, it will be replaced. The evaluation criteria can be independent (used in filter methods) or dependent (used in wrapper methods).
3. **Stopping criterion:** controls the stoppage of the feature selection process. The iteration of subset generation and evaluation continues until a predetermined stopping criterion is met.
4. **Result validation:** The selected best subset is validated through prior knowledge or various tests using synthetic and/or real-world datasets.

According to (Asir et al., 2016) feature selection approaches are categorized into three main categories: wrapper, filter, and hybrid methods.

2.4.1 Filter feature selection method

Filters aim to select an optimal feature subset based on the inherent characteristics of the data rather than a specific learning algorithm. Generally, filters compute the score of a feature (subset) using certain evaluation criteria and then select features with the highest scores (Hancer et al., 2020). Within the filter method, the model initially incorporates all features and subsequently identifies the optimal feature subset using statistical measures like Pearson's correlation, Linear Discriminant Analysis (LDA), Chi-square, and Mutual Information (MI). These statistical techniques rely on both the response and feature variables within the dataset(Hancer et al., 2020).

2.4.1.1 Mutual information

MI feature selection is a commonly used filter method for improving the performance of intrusion detection systems (IDS) (Alalhareth & Hong, 2023). This method evaluates the relationship between each feature and the class label or the target variable, choosing features with the highest mutual information scores. One of the primary benefits of using mutual information is its ability to consider nonlinear relationships between features and class labels. This characteristic renders mutual information making it suitable for handling complex and nonlinear data patterns in IDS. By selecting the most relevant and informative features, mutual information feature

selection can effectively reduce the dimensionality of the data and improve the degree of mutual dependence between two variables (x, y). MI assesses the amount of information acquired about one arbitrary variable through the other random variable (Venkatesh & Anuradha, 2019).

Mutual Information (MI) serves as a statistical method employed in feature selection. It quantifies the two discrete variables, the mutual information MI criterion is the amount of information these variables share about each other (Al-rimy et al., 2021). Based on those authors the criterion is calculated according to (1) as follows.

$$MI(X;Y)=\sum_{y\in Y}\sum_{x\in X}P(x,y)\log\frac{p(x,y)}{p(x)p(y)} \quad 1$$

$p(x)$ and $p(y)$ represent the marginal distributions of x and y , while $p(x, y)$ denotes their joint distribution.

2.4.2 Wrapper feature selection Method

Wrapper-based feature selection methods rely on the predictive power of the learning algorithm used to assess the qualitative attributes of selected features. In these methods, feature selection occurs in two primary steps for a specific learning model: i) identifying an optimal feature subset from the dataset, and ii) evaluating the chosen features. This iterative process continues until certain termination criteria are satisfied. The feature set search component generates subsets of features, and subsequently, the machine learning technique is employed to gauge the effectiveness of the selected feature set based on performance metrics (Thakkar & Lohiya, 2022).

Wrappers necessitate a learner to assess the quality of potential feature subsets. Consequently, wrappers can achieve superior feature subsets to improve the performance of the predefined learning algorithm. However, they often entail higher computational intensity compared to filters. Firstly, wrappers obtain a feature subset utilizing search strategies. Subsequently, the selected feature subset's quality is evaluated through a learning algorithm. This iterative process continues until the stopping criterion is satisfied (Hancer et al., 2020).

2.4.3 Hybrid feature selection method

Hybrid approaches strive to leverage the benefits of both wrappers and filters. Two common hybridization methods are typically employed to combine wrappers and

filters. One approach involves a two-stage process wherein a filter method is first applied to reduce the feature set, followed by a wrapper method on the reduced set to obtain the final subset. Alternatively, a filter (wrapper) method can serve as a local search mechanism within a wrapper (filter) method. The latter method is anticipated to yield superior performance in terms of both learning outcomes and feature subset size (Hancer et al., 2020). In this study, we apply the hybrid method to obtain the advantages of both the wrapper and filter methods.

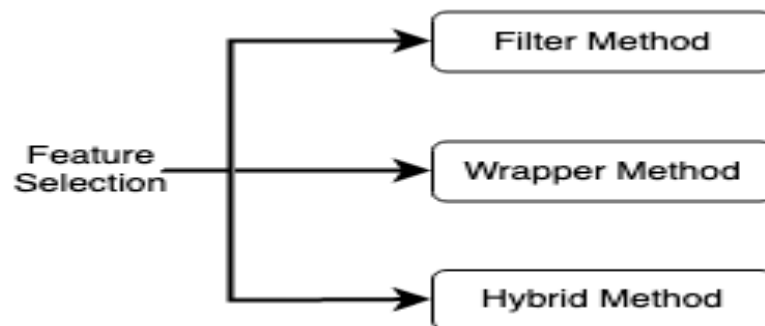


Figure 2. 3 Classification of Feature selection approach

2.5 Related work

The authors (Kasongo & Sun, 2020b) proposed a filter-based feature reduction technique using the XGBoost algorithm to score the feature importance of each feature of the dataset. They evaluated several ML algorithms including SVM, KNN, LR, ANN, and DT classifiers using the UNSW-NB15 dataset. Finally, their experiment achieved an accuracy of 90.85% by DT. The authors trained their proposed method on highly imbalanced datasets without employing any techniques to address this imbalance, which can lead to a high false positive rate and low detection accuracy.

The author (Almasoudy et al., 2020) proposed wrapper feature selection for an Intrusion Detection system using Differential Evolution (DE) for feature selection and they applied an Extreme Learning Machine (ELM) algorithm for classification, their experimental showed that this technique achieved an accuracy of 80.15 % using NSL_KDD datasets. The author uses only the Differential Evolution (DE) which is the wrapper feature selection method without reducing the features which leads to high computational time.

The author (Khan et al., 2020) introduced a wrapper-based feature selection approach using the Feature Importance model with an RF classifier for intrusion detection systems. They used KNN, DT, RF, Bagging Meta Estimator, and XGBoost ML algorithms for classification. Their experimental results have shown that RF achieved the highest accuracy of 74.87% using UNSW-NB 15 datasets. The authors in this paper recommend building a hybrid architecture of the method to further improve the accuracy of the model as well as speed up the computation time of the model. The author uses ensemble feature selection methods which require higher computational costs particularly challenging when handling large-scale datasets.

In this (Nazir & Khan, 2021) paper the authors introduce a wrapper-based feature selection technique Tabu Search - Random Forest (TS-RF). Tabu search serves as the search method, while RF acts as the learning algorithm for NIDS. They achieved an accuracy of 83.12% using UNSW-NB 15 datasets. The authors use only a Tabu search wrapper feature selection approach without reducing the feature which leads to high computational time and they have not used any technique to solve the imbalance problem in the dataset because this has an impact on the classifier accuracy and also increases misclassification rate and false positives.

The authors (Mebawondu et al., 2020) proposed IDS using a filter-based information gain to rank the features as feature selection and ANN as classification, and finally their experiment result achieved 76.96% accuracy using UNSW-NB 15 datasets. The author uses only information gain for feature selection without evaluated using machine learning based techniques whether those features are relevant or not. Using hybrid feature selection algorithms, the system's performance can be further enhanced.

The author (Taher et al., 2019) proposes an intrusion detection system to detect malicious attacks using ANN and SVM algorithms for classification, the applied Chi-Square filter method, and the Correlation-based wrapper feature selection techniques. Their Experimental results showed that achieved the highest accuracy of 94.02% using the NSL-KDD dataset by ANN.

The authors (Jing & Chen, 2019) proposed network intrusion detection without a feature selection approach using an SVM classifier, and finally, their experiment achieved an accuracy of 85.99% using UNSW-NB 15. The proposed method was evaluated using only a single classifier and SVM unsuitable for large datasets, not considering computational efficiency.

The author (Ahmim et al., 2018) proposes an intrusion detection system (IDS) that combines different classifier approaches that are based on decision tree and rules-based concepts, namely, REP Tree, JRip algorithm, and Forest PA without applying any feature selection method using CICIDS 2017 datasets and he achieved an accuracy of 96.6% by Forest PA. The author does not apply any feature selection method which may lead to overfitting, decreased model performance, longer training and prediction times, and he does not use any technique for handling the imbalance classes in the datasets.

The authors (Ambikavathi & Srivatsa, 2020) propose an intrusion detection system (IDS) using the Random forest’s variable importance function VarImp() to obtain the optimal features, RF is also used for classification and they achieved an accuracy of 97.34% utilized CICIDS 2017 datasets. The authors focus on the Random Forest algorithm for both feature selection and classification this single-algorithm focus might restrict the ability to handle various types of data patterns and attack behaviors optimally. Incorporating and comparing multiple algorithms could provide a more comprehensive evaluation and possibly better performance for intrusion detection systems. They do not apply any techniques for solving imbalance classes in the dataset.

Table 2. 2 Summary of the related works

Proposed by	Feature Selection Technique	Classifier Algorithm	Dataset used	Limitation
(Kasongo & Sun, 2020b)	XGBoost	SVM, KNN, LR, ANN, and DT	UNSW-NB15	The authors trained their proposed method on highly imbalanced datasets without employing any

				techniques to address this imbalance class, which can lead to a high false positive rate and low detection accuracy.
(Almasoudy et al., 2020)	Differential Evolution (DE)	Extreme Learning Machine (ELM)	NSL_KDD	The author uses only the Differential Evolution (DE) which is the wrapper feature selection method without reducing the features which leads to high computational time.
(Khan et al., 2020)	Feature Importance(RF)	KNN, DT, RF, Bagging Meta Estimator, and XGBoost	UNSW-NB 15	The author uses ensemble feature selection methods which require higher computational costs particularly challenging when handling large-scale datasets.
(Nazir & Khan, 2021)	Tabu Search	RF	UNSW-NB 15	The authors use only a Tabu search wrapper feature selection approach without reducing the feature which leads to high computational time and they have not used any technique to solve the imbalance problem in the dataset because this has an impact on the classifier accuracy and also

				increases misclassification rate and false positives
(Mebawondu et al., 2020)	Information Gain	ANN	UNSW-NB 15	The author uses information gain for feature selection without using advanced feature selection techniques to evaluate whether those features are relevant or not
(Jing & Chen, 2019)	Not applied	SVM	KDDCUP99	The proposed method was evaluated using only a single classifier and SVM unsuitable for large datasets, not considering computational efficiency.
(Ahmim et al., 2018)	Not applied	REP Tree, JRip, and Forest PA	CICIDS 2017	The author does not apply any feature selection method which may lead to overfitting, decreased model performance, longer training and prediction times, and he does not use any technique for handling the imbalance classes in the datasets.
(Ambikavathi & Srivatsa, 2020)	Random forest's variable importance function VarImp()	RF	CICIDS 2017	The authors focus on the Random Forest algorithm for both feature selection and classification this single-algorithm focus might restrict the ability to

				handle various types of data patterns and attack behaviors optimally
--	--	--	--	--

2.5.1 Gap Analysis

The previous existing works have several limitations in feature selection, classification, class imbalance handling, and computational time. Those using the filter approach often fail to identify the most relevant features accurately, as this method assesses features statically without incorporating a machine learning algorithm. Conversely, studies employing the wrapper method face high computational costs due to the iterative nature of using a machine-learning algorithm for feature selection. A combined approach, where the filter method initially reduces the feature set and the wrapper method subsequently evaluates the reduced set, can mitigate these issues. Additionally, some studies neglect feature selection entirely, resulting in over fitting, reduced model performance, and longer training and prediction times. Furthermore, many of these works do not employ resampling techniques to address class imbalance in the datasets. To address the limitations identified in the literature we reviewed, we propose a Hybrid Feature Selection approach and combine classifiers using Stacking (DT + KNN + RF + XGBoost). Additionally, we apply resampling techniques to address the class imbalance in the dataset.

CHAPTER THREE

METHODOLOGY

In this section, we outline the dataset used, the experimental setup, the feature selection methodology employed, the classification algorithms utilized, the tools employed for experimentation, and the block diagram of the proposed model.

3.1 Dataset collection and preparation

Due to the lack of an adequate dataset, anomaly-based approaches in intrusion detection systems are suffering from accurate deployment, analysis, and evaluation(Sharafaldin et al., 2018). Privacy concerns make it difficult to obtain private local datasets for Intrusion Detection Systems (IDS). As a result, many researchers rely on public datasets to evaluate their machine-learning models. Datasets like DARPA98, KDD99, ISC2012, and ADFA13 are commonly used benchmarks in IDS research for assessing the performance of proposed intrusion detection and prevention approaches(Sharafaldin et al., 2018).

Several of the above-mentioned datasets are outdated and deemed unreliable due to various shortcomings. Some lack traffic diversity and volume, while others fail to encompass a wide range of attacks. Additionally, certain datasets contain payloads that do not reflect current trends, or they lack comprehensive feature sets and metadata(Sharafaldin et al., 2018). Assessing datasets is crucial in validating IDS approaches, as it enables us to evaluate the effectiveness of proposed methods in detecting intrusive behavior (Khraisat et al., 2019).

(Gharib et al., 2016) proposed evaluation framework for Intrusion Detection datasets outlines 11 criteria for evaluation. The criteria include Complete Network Configuration, Complete Traffic Representation, Labeled Dataset, Comprehensive Interaction Coverage, Complete Capture, Availability of Protocols, Attack Diversity, Anonymity, Heterogeneity, Feature Set, and Metadata.

According to (Sharafaldin et al., 2018) the CICIDS 2017 dataset meets all 11 criteria outlined in the intrusion detection dataset evaluation framework. Hence, for this study, we chose the CIC-IDS 2017 dataset, prepared by the University of New Brunswick Institute Of Cyber Security (Canada). This dataset comprises both benign traffic and the latest common attacks, closely resembling real-world data. Additionally, it provides results of network traffic analysis using the CIC Flow Meter, with labeled flows categorized by timestamp, source and destination IPs, source and destination ports, protocols, and attacks in Comma Separated Values (CSV) files.

3.1.1 Dataset description

The CICIDS2017 dataset consists of both benign traffic and the latest common attacks, mirroring real-world data (PCAPs). Additionally, it offers results from network traffic analysis conducted using CICFlowMeter, with labeled flows including timestamps, source and destination IPs, source and destination ports, protocols, and attacks (stored in CSV files). Table 4 provides details on the victim and attacker network information, including their IP addresses.

Table 3. 1 Victim and Attacker Network Information in the CICIDS 2017 Dataset

Network device	IP Address
Firewall	205.174.165.80, 172.16.0.1
DNS+DC Server	192.168.10.3
Outsiders (Attackers network)	
Kali	205.174.165.73
Win	205.174.165.69, 70, 71
Insiders (Victim network)	
Web server 16 Public	192.168.10.50, 205.174.165.68
Ubuntu Server 12 Public	192.168.10.51, 205.174.165.66
Ubuntu 14.4, 32B	192.168.10.19
Ubuntu 14.4, 64B	192.168.10.17
Ubuntu 16.4, 32B	192.168.10.16
Ubuntu 16.4, 64B	192.168.10.12
Win 7 Pro, 64B	192.168.10.9

Win 8.1, 64B	192.168.10.5
Win Vista, 64B	192.168.10.8
Win 10, pro 32B	192.168.10.14
Win 10, 64B	192.168.10.15
MAC	192.168.10.25

The data capture period spanned from 9 a.m. on Monday, July 3, 2017, to 5 p.m. on Friday, July 7, 2017, totaling 5 days. Monday's data comprises normal benign traffic only. The attacks including Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS, happened both in the morning and afternoon on Tuesday, Wednesday, Thursday, and Friday. Table 4 details the types of attacks observed on each of the five days.

Table 3. 2 Description of files containing the CICIDS2017 dataset with attack found

No	Name of the file	Day Activity	Attacks Found	Total attack
1.	Monday-WorkingHours.pcap_ISCX.csv	Monday	Benign (Normal human activities)	0
2.	Tuesday-WorkingHours.pcap_ISCX.csv	Tuesday	Benign, FTP-Patator, SSH-Patator	2
3.	Wednesday-workingHours.pcap_ISCX.csv	Wednesday	Benign, DoS GoldenEye, DoSHulk, DoSSlowhttpstest, DoS slow loris, Heartbleed	5
4.	Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv	Thursday	Benign, Web Attack – Brute Force, Web Attack – SQL Injection, Web Attack – XSS	3
5.	Thursday-WorkingHours-Afternoon-Infiltration.pcap_ISCX.csv	Thursday	Benign, Infiltration	1
6.	Friday-WorkingHours-Morning.pcap_ISCX.csv	Friday	Benign, Bot	1

7.	Friday-WorkingHours- Afternoon- PortScan.pcap_ISCX.csv	Friday	Benign, PortScan	1
8.	Friday-WorkingHours- Afternoon- DDos.pcap_ISCX.csv	Friday	Benign, DDoS	1
Total	8 fillies	5 days	15 (1 normal + 14 Attack)	14

The CICIDS2017 dataset closely mimics real-world network data (PCAPs) and employs CICFlowmeter-V3.0 to extract 78 features and 1 label. It encompasses the behavioral profiles of 25 users across HTTP, HTTPS, FTP, SSH, and email protocols(Maseer et al., 2021). Table 6 lists the 78 features and 1 label class with their descriptions.

Table 3. 3 The 78 features and the label class of the CICIDS 2017 dataset record

Future no.	Feature Name	Description
1.	Destination Port	The port number on the destination host
2.	Flow Duration	Duration of the flow in Microseconds
3.	Total Fwd Packets	Total packets in the forward direction
4.	Total Backward Packets	Total packets in the backward direction
5.	Total Length of Fwd Packets	The total size of the packet in a forward direction
6.	Total Length of Bwd Packets	The total size of the packet in a backward direction
7.	Fwd Packet Length Max	Maximum size of the packet in a forward direction
8.	Fwd Packet Length Min	Minimum size of the packet in a forward direction
9.	Fwd Packet Length Mean	The mean size of the packet in a forward direction
10.	Fwd Packet Length Std	Standard deviation size of the packet in a forward direction
11.	Bwd Packet Length Max	Maximum size of the packet in a backward direction

12.	Bwd Packet Length Min	The minimum size of the packet in a backward direction
13.	Bwd Packet Length Mean	The mean size of the packet in a backward direction
14.	Bwd Packet Length Std	Standard deviation size of the packet in a backward direction
15.	Flow Bytes/s	Number of flow bytes per second
16.	Flow Packets/s	Number of flow packets per second
17.	Flow IAT Mean	Mean time between two packets sent in the flow
18.	Flow IAT Std	Standard deviation time between two packets sent in the flow
19.	Flow IAT Max,	Maximum time between two packets sent in the flow
20.	Flow IAT Min	Minimum time between two packets sent in the flow
21.	Fwd IAT Total	Total time between two packets sent in the forward direction
22.	Fwd IAT Mean	Mean time between two packets sent in the forward direction
23.	Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
24.	Fwd IAT Max	Maximum time between two packets sent in the forward direction
25.	Fwd IAT Min	Minimum time between two packets sent in the forward direction
26.	Bwd IAT Total	Total time between two packets sent in the backward direction
27.	Bwd IAT Mean	Mean time between two packets sent in the backward direction
28.	Bwd IAT Std	Standard deviation time between two packets sent

		in the backward direction
29.	Bwd IAT Max	Maximum time between two packets sent in the backward direction
30.	Bwd IAT Min	Minimum time between two packets sent in the backward direction
31.	Fwd PSH Flags	Number of times the PSH flag was set in packets traveling in the forward direction (0 for UDP)
32.	Bwd PSH Flags	Number of times the PSH flag was set in packets traveling in the backward direction (0 for UDP)
33.	Fwd URG Flags	Number of times the URG flag was set in packets traveling in the forward direction (0 for UDP)
34.	Bwd URG Flags	Number of times the URG flag was set in packets traveling in the backward direction (0 for UDP)
35.	Fwd Header Length	Total bytes used for headers in the forward direction
36.	Bwd Header Length	Total bytes used for headers in the backward direction
37.	Fwd Packets/s	Number of forwarding packets per second
38.	Bwd Packets/s	Number of backward packets per second
39.	Min Packet Length	Minimum length of a packet
40.	Max Packet Length	Maximum length of a packet
41.	Packet Length Mean	The mean length of a packet
42.	Packet Length Std	Standard deviation length of a packet
43.	Packet Length Variance	Variance length of a packet
44.	FIN Flag Count	Number of packets with FIN
45.	SYN Flag Count	Number of packets with SYN
46.	RST Flag Count	Number of packets with RST
47.	PSH Flag Count	Number of packets with PUSH
48.	ACK Flag Count	Number of packets with ACK
49.	URG Flag Count	Number of packets with URG

50.	CWE Flag Count	Number of packets with CWE
51.	ECE Flag Count	Number of packets with ECE
52.	Down/Up Ratio	Download and upload ratio
53.	Average Packet Size	The average size of a packet
54.	Avg Fwd Segment Size	Average size observed in the forward direction
55.	Avg Bwd Segment Size	Average number of bytes bulk rate in the backward direction
56.	Fwd Header Length	Total bytes used for headers in the forward direction
57.	Fwd Avg Bytes/Bulk	Average number of bytes bulk rate in the forward direction
58.	Fwd Avg Packets/Bulk	Average number of packets bulk rate in the forward direction
59.	Fwd Avg Bulk Rate	Average number of bulk rates in the forward direction
60.	Bwd Avg Bytes/Bulk	Average number of bytes bulk rate in the backward direction
61.	BwdAvg Packets/Bulk	Average number of packets bulk rate in the backward direction
62.	Bwd Avg Bulk Rate	Average number of bulk rates in the backward direction
63.	Subflow Fwd Packets	The average number of packets in a sub-flow in the forward direction
64.	Subflow Fwd Bytes	The average number of bytes in a sub-flow in the forward direction
65.	Subflow Bwd Packets	The average number of packets in a sub-flow in the backward direction
66.	Subflow Bwd Bytes	The average number of bytes in a sub-flow in the backward direction
67.	Init_Win_bytes_forward	The total number of bytes sent in an initial window

		in the forward direction
68.	Init_Win_bytes_backward	The total number of bytes sent in an initial window in the backward direction
69.	act_data_pkt_fwd	Count of packets with at least 1 byte of TCP data payload in the forward direction
70.	min_seg_size_forward	Minimum segment size observed in the forward direction
71.	Active Mean	Meantime a flow was active before becoming idle
72.	Active Std	Standard deviation time a flow was active before becoming idle
73.	Active Max	The maximum time a flow was active before becoming idle
74.	Active Min	The minimum time a flow was active before becoming idle
75.	Idle Mean	Meantime a flow was idle before becoming active
76.	Idle Std	Standard deviation time a flow was idle before becoming active
77.	Idle Max	The maximum time a flow was idle before becoming active
78.	Idle Min	The minimum time a flow was idle before becoming active
79.	Label	Target class

3.2 Data preprocessing

The primary objective of preprocessing in machine learning is to optimize the training and testing process by effectively transforming and scaling the dataset. This critical step in the ML workflow involves preparing the data before applying it to an ML algorithm. Preprocessing scales features to a consistent range, enhancing accuracy, reducing the time and resources needed for model training, preventing over fitting, and improving the model's interpretability(Al Lail et al., 2023).

In our research, we used the MachineLearning.CSV data subset from the CICIDS-2017 dataset. This file comprises eight (8) traffic monitoring sessions, each presented in a

comma-separated value (CSV) format(Maseer et al., 2021). This file encompasses both normal traffic labeled as BENIGN and anomaly traffic labeled as ATTACKS. The specific attack types are further detailed in the fourth column of Table 4. Apart from benign traffic, this dataset includes 14 types of attacks. As outlined in Table 4, the label class indicates whether the traffic is normal or an attack, where normal corresponds to Benign and the listed attacks.

The dataset used have a total of 2,830,743 records out of the total 2,273,097 records are benign and 557,646 are attacks.

```
In [20]: df12['Class'].value_counts()
Out[20]: 0      2273097
         1      557646
         Name: Class, dtype: int64
```

Figure 3. 1 Class distribution of the total CICIDC 2017 dataset

In the preprocessing phase, we first concatenate the eight files into one containing all files. Then we applied random sampling to select 20% of the dataset for our preprocessing because of a shortage of resources out of 2,830,743 records, we selected 566,149 records and 79 columns.

```
In [22]: #Sampling
         #generating one row
         df1 = df12.sample(frac =0.20)
         # checking if sample is 0.20 times data or not
         if (0.20*(len(df12))== len(df1)):
             print( "Cool")
         print(len(df12), len(df1))
2830743 566149
```

Figure 3. 2 Selecting 20% of CICIDS 2017 dataset

A crucial step in preprocessing is cleaning the dataset(Al Lail et al., 2023). The cleaning of the data includes finding incomplete, improper, inaccurate, or unnecessary data, by replacing, modifying, or deleting these data from the dataset. In our preprocessing rows with null values, duplicates, and empty cells, such as infinity (Inf) and not a number (NaN) are dropped. To improve quality the dataset we have remove total of 1,172 infinity (Inf) values and 38,255 duplicate rows using python code.

```
Before Data Clean:
Total -inf values: 0
Total inf values: 893
Total null values: 279
Total special values (-inf, inf, null): 1172
Total duplicate rows: 38255

After Data Clean:
Total -inf values: 0
Total inf values: 0
Total null values: 0
Total special values (-inf, inf, null): 0
Total duplicate rows: 0
```

Figure 3. 3 Data cleaning output

The initial CICIDS-2017 dataset also contains categorical features (e.g., labels) that need to be converted into numerical values to prepare them for the ML algorithms. We have converted the categorical values in the label class to numeric values by designating the normal class as “0” and the attack class as “1”.

We employed the MinMaxScaler to normalize the data, ensuring algorithms sensitive to feature magnitudes are not biased towards features with larger values. Without scaling, such algorithms may prioritize these larger numerical values, skewing the results. By applying MinMaxScaler, we transform all feature values in the cleaned data to a uniform range between zero and one, enabling the models to learn more effectively and improving their performance.

Finally, the cleaned 20% of the dataset is split into two sub-datasets: training (80%), and testing (20%). The split is performed randomly meaning that the class distribution percentage of the 20% of the dataset is retained in the training and testing sets. The split is created in this way to ensure that there are sufficient samples to train the models, while also ensuring sufficient samples for the testing of the results. The training sub-datasets are used for model training while the testing dataset is used for the final evaluation of the models.

3.3 Feature Selection

In Machine Learning (ML), Feature Selection (FS) holds significant importance by diminishing data dimensionality and amplifying the efficiency of proposed frameworks. Nonetheless, in practical scenarios, FS endeavors encounter obstacles such

as daunting dimensionality, computational and storage intricacies, noisy or ambiguous attributes, and the demand for high performance. The domain of FS presents a vast and formidable landscape, fraught with challenges(Dhal & Azad, 2022). In this study, we proposed a hybrid feature selection method that combines Mutual Information (MI) from the filter feature selection approach with Recursive Feature Elimination using a Random Forest Classifier from the wrapper feature selection approach.

3.3.1 Mutual Information Feature Selection

Mutual information feature selection stands as a widely used filter method to enhance the efficiency of intrusion detection systems (IDS). It measures the dependence between individual features and the target variables and selects the features with the highest mutual information scores. Mutual information yields a non-negative value, where a value of zero indicates that the two observed variables are statistically independent(Ambusaidi et al., 2016).

Based on (Dhal & Azad, 2022),(Valenzuela et al., n.d.),(Kou et al., 2020),(Priscilla & Prabha, 2021) the following are Advantages of MI feature selection.

Non-linearity data handling: MI, being a non-parametric measure can capture complex non-linear dependencies between features and the target variable. It assesses the amount of information shared between variables, making it suitable for identifying non-linear associations.

Efficiency: The MI feature selection method was computationally efficient, especially when dealing with high-dimensional datasets, by focusing on the information content of features rather than exhaustive search.

Model Agnostic: MI feature selection can be used with any machine learning algorithm without requiring any changes because it is model agnostic.

Feature Independence: MI accounts for the independence between features but also takes into account the mutual dependence between features and the target variable. It finds features that have distinct and important information for target prediction, resulting in a more effective and comprehensible feature set.

No Assumptions about Data Distribution: MI doesn't require data transformation or preprocessing and can handle discrete and continuous data, making it appropriate for a variety of datasets.

Feature Ranking: Features can be prioritized or their relative importance in the context of the problem domain can be understood by using the feature importance ranking that MI provides, which is based on how relevant each feature is to the target variable.

Given that the dataset used was large and showed nonlinear relationships between the features and the target variable, a straight-line model would not suffice to describe the input-output relationships. This means that changes in the output are not directly proportional to changes in the input features. The choice of Mutual Information (MI) feature selection was made because of its computational efficiency for high-dimensional data and its ability to handle nonlinear datasets. Furthermore, MI feature selection is adaptable and does not require modification for a wide range of machine learning algorithms. It considers the mutual dependence between features and the target variable and can handle both continuous and discrete data types. Furthermore, MI assigns a ranking to features, which is helpful in order to prioritize features or determine their relative importance. These factors led to the selection of the MI feature selection method over the filter feature selection techniques.

The following feature selection steps are followed by the Mutual Information algorithm:

Compute MI Scores: First, the mutual information scores between every feature and the target variable are computed by the algorithm.

Rank Features: The algorithm ranks the features according to their scores after calculating the mutual information scores for each feature. Higher mutual information scores indicate that a feature is more relevant or informative about the target variable.

Choose the Best Features: The top 'k' features with the highest mutual information scores are chosen by the algorithm. One can predetermine the value of 'k' or use optimization criteria to determine it.

Feature Subset Selection: The selected feature subset consists of the identified top features. While features that are superfluous or less useful are eliminated.

In our proposed model, we first compute mutual information scores for each feature and rank the features according to their scores to ascertain the relationship between individual features and the class label. Next, we have selected features that have Mutual Information

(MI) values of at least 0.10. Out of the 78 features in total, 52 features were chosen based on this criterion.

3.3.2 Recursive Feature Elimination Feature Selection

Recursive Feature Elimination (RFE) constitutes the second phase of our feature selection approach. Operating as a wrapper method, RFE iteratively assesses feature importance by eliminating them based on machine learning performance. It progressively discards the least significant features until optimal performance is achieved or a predetermined number of features are attained(Thakkar & Lohiya, 2021). RFE is a wrapper feature selection approach that fits a learning model and eliminates the less important features. Based on the scores obtained by the learning model, features are ranked and recursively eliminated through iterations. RFE removes the dependency and collinearity among features(Priscilla & Prabha, 2021).

Advantages of RFE (Priscilla & Prabha, 2021), (Assistant Professor, Department of Information Technology, Bishop Heber College, Affiliated to Bharathidasan University, Tiruchirappalli, 620 024, Tamil Nadu, India, et al., 2023), (Habeeb & Babu, 2024), (Yin et al., 2023).

Model Agnosticism: RFE is model agnostic because it works with a variety of machine learning techniques. Without any changes, it can be used with clustering, regression, or classification techniques.

Scalability: RFE's iterative operation on subsets of features makes it scalable to huge datasets. Because of its ability to manage datasets with tens of thousands or even millions of characteristics, it is appropriate for big data applications where effective modeling requires dimensionality reduction.

Diminishes Dimensionality: By removing unimportant characteristics, RFE successfully lowers the dimensionality of the feature space.

Feature Ranking: RFE offers a feature priority ranking according to how much each feature improves model performance. This score can direct feature engineering efforts or assist in prioritizing features for more study.

As previously indicated, there were nonlinear relationships between each feature and the target variable in our large dataset. Because it is capable of handling datasets with thousands or even millions of features, Recursive Feature Elimination (RFE) was selected. It decreases dimensionality, ranks feature to prioritize them for additional examination, and works with a variety of machine-learning algorithms. RFE was chosen from the wrapper feature selection techniques for the aforementioned reasons.

The RFE algorithm works through the following steps in feature selection:

- 1. Initialization:** Begin by deciding on a machine learning algorithm that can be used for importance or feature ranking.
- 2. Feature Ranking:** Utilizing the full dataset, train the selected model to determine the features' relative importance or coefficients. In this step, each feature is given a weight that represents its contribution to the predictive performance of the model.
- 3. Feature Elimination:** Extract the dataset's least significant feature or features.
- 4. Model Training:** Utilizing the smaller feature set that was acquired in step 3, train the model. A subset of features that are judged most relevant based on the previously acquired feature ranking is used to train this model.
- 5. Performance Evaluation:** Employing a suitable assessment metric, assess the model's performance.
- 6. Stopping Criterion:** Determine if the specified stopping criterion has been satisfied.
- 7. Iteration:** Up until the stopping requirement is met, recursively repeat steps 2 through 5. The model is retrained on the smaller feature set after the least significant features are eliminated in each iteration.
- 8. Final Model Selection:** Using performance metrics gathered from the iterations, select the final model.

The input dataset for our RFE FS algorithm only includes the 52 features that have been reduced using the mutual information algorithm from the first stage. The significance of the features is then graphically displayed by the RFE algorithm, and we have chosen the top 25 features from a total of 52 features for the assessment of the suggested model.

3.4 Synthetic Minority Oversampling Technique (SMOTE)

One of the most significant challenges in classification problems is the prevalence of majority classes compared to minority ones. When minority and majority classes coexist in the same dataset, they create an imbalanced class distribution. In binary (two-class) classification problems, it is common to designate class 0 as the majority class and class 1 as the minority class. To address this imbalance, an oversampling technique was adopted to balance the data before training the prediction models. The Synthetic Minority Over-sampling Technique (SMOTE) was employed to oversample the minority class, effectively mitigating the over fitting problem associated with the majority class, SMOTE generates new samples, reducing the likelihood of over fitting compared to simply duplicating minority class samples, avoids potential loss of valuable majority class information by not discarding any samples, and effective for High-Dimensional Data(Ahsan et al., 2022),(Fan et al., 2024).

SMOTE works first by randomly selecting a minority class instance, denoted as a . The algorithm then searches for the k nearest neighbors of a , which are also minority class instances. One of these neighbors, b is randomly chosen. A synthetic instance is created by forming a line segment between a and b in the feature space. This new synthetic instance is a convex combination of the two chosen instances, a and b (Fan et al., 2024).

Synthetic Minority Oversampling Technique



Figure 3. 4 How SMOTE works

3.5 Tools used for Experiment

3.5.1 The Hardware Tools Used

The proposed model implemented on the hardware with the specification of model: Toshiba satellite L855, operating system: Microsoft Windows 10 Enterprise, system type: 64-bit operating system, x64-based processor core i7 CPU @ 2.4GHz, installed memory (RAM): 8.00 GB, hard disk: 500 GB.

3.5.2 The Software Tools Used

Python, a high-level programming language, stands out as a powerful, object-oriented, and general-purpose tool that has seen widespread adoption in recent years. Its design focuses on readability, and its syntax enables programmers to convey concepts with fewer lines of code compared to languages like C(Python_for_Data_Analysis.Pdf, n.d.).

Based on the above-mentioned reasons and other advantages as a high-level programming language, Python is rapidly growing, platform-independent, powerful, easy to understand, and open-source with an object-oriented approach we choose Python for implementing the proposed methodology.

The following are some Python libraries:

- **Math:** The math library offers mathematical functions and constants essential for numerical computations. It encompasses operations like basic arithmetic, trigonometry, exponentiation, logarithms, and more.
- **Random:** The random library serves for generating random numbers and conducting random sampling. It facilitates the creation of random integers, floating-point numbers, and the selection of random elements from sequences.
- **NumPy (np):** NumPy is a robust library for numerical computing in Python. It enables support for large, multi-dimensional arrays and matrices, alongside a suite of mathematical functions tailored for efficient array operations.
- **matplotlib.pyplot (plt):** Matplotlib is a widely-used plotting library in Python, instrumental in crafting static, interactive, and animated visualizations. The pyplot module furnishes a MATLAB-like interface, simplifying the creation of various plots and visualizations.

- **Pandas (Pd):** Pandas is a potent library for data manipulation and analysis in Python. It furnishes data structures and functions for the seamless handling of structured data, such as tabular data and time series data.
- **Seaborn (sb):** Seaborn stands as a statistical data visualization library, built upon Matplotlib. It delivers a high-level interface for crafting informative and visually appealing statistical graphics.

3.6 System Architecture of Proposed Method

The following diagram shows our proposed Hybrid feature selection intrusion detection system to classify as normal or attack based on the dataset.

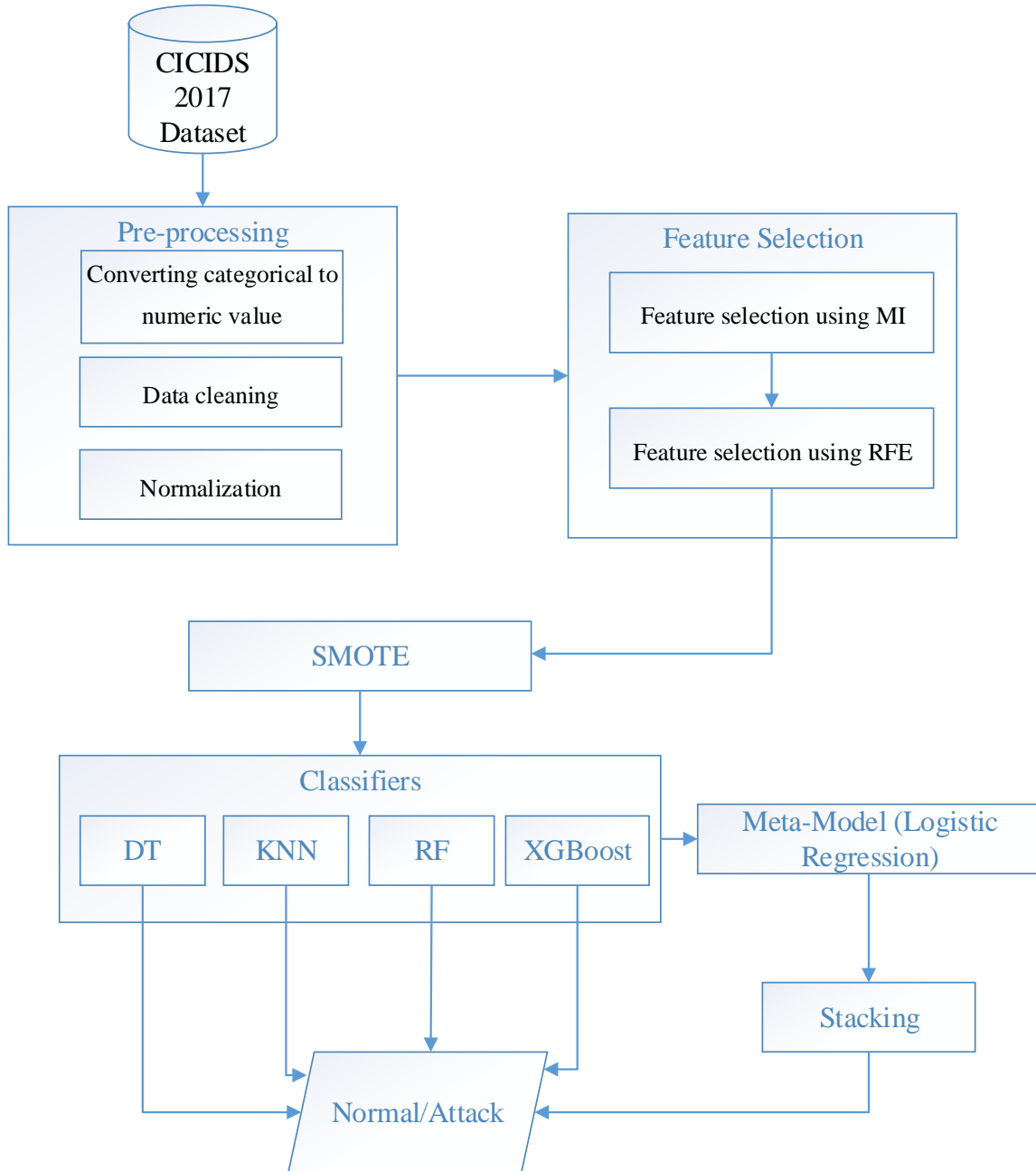


Figure 3. 5 System architecture for the proposed model

As shown in Figure 3.5 our proposed model begins by acquiring the Intrusion Detection System (IDS) dataset namely CICIDS 2017 and proceeds with preprocessing techniques.

We remove the duplicates, infinity values, and NaN values from the dataset to ensure data integrity.

We have converted the categorical value in the label class to a numeric value by assigning the normal class as 0 and the attack class as 1.

Normalization is then performed using MinMaxScaler, which rescales the features to a fixed range, typically within [0, 1]. This transformation standardizes the feature values, aiding in uniformity across the dataset.

Following preprocessing, feature selection is conducted. Initially, Mutual Information (MI) is employed as a filter feature selection approach to identify relevant features. Subsequently, Recursive Feature Elimination (RFE) from the wrapper feature selection approach is utilized to further refine the feature set.

After selecting the top-importance features through RFE, the model is evaluated using various machine learning algorithms including DT, KNN, RF, and XGBoost, and these algorithms are then combined using stacking a technique that merges the predictions of multiple models to enhance overall performance and robustness.

CHAPTER FOUR

RESULTS AND DISCUSSION

This chapter focuses on the experimentation of our proposed method using the CICIDS 2017 dataset, employing Python programming language for implementation. We detail the simulation and analysis process, including the steps involved in simulating and analyzing the dataset. Following the analysis conducted using Python tools; we thoroughly discuss and interpret the findings and results.

The experiment was conducted using the Anaconda Navigator tool which has a Jupyter Notebook interface. The algorithms chosen for our analysis are, DT, KNN, RF, and XGBoost, and we combined those using stacking concepts.

4.1 Dataset used for Experiments

As described in Chapter Three, our experiment utilized the CICIDS 2017 dataset. This dataset comprises a total of 2,830,743 records. Due to resource constraints, specifically a lack of high-performance computing resources, we used 20% (566,149) records of the CICIDS 2017 dataset for our experiment.

Following preprocessing of the 20% we divided the dataset into training and testing sets. Specifically, 80% (422,086) records were allocated for training purposes, while the remaining 20% (105,522) records were reserved for testing.

4.2 Handling Class Imbalance

As mentioned in chapter three to balance the distribution of class in the dataset we applied SMOTE and we trained the model using balanced class.

Table 4.1 and Table 4.2 shows the distribution of the class before and after applying SMOTE.

Table 4. 1 Distribution of label class in training and testing before applying SMOTE

class	Training	Testing	Total
0	346,464	86,603	433,067
1	75,622	18,919	94,541
Total	422,086	105,522	527,608

The above table 4.1 shows the distribution of label classes in the training and testing datasets before applying SMOTE (Synthetic Minority Over-sampling Technique). In the training dataset, there are a total of 422,086 instances. Out of these, 346,464 instances belong to class 0 (normal), and 75,622 instances belong to class 1 (attack). In the testing dataset, there are a total of 105,522 instances. Out of these, 86,603 instances belong to class 0 (normal), and 18,919 instances belong to class 1 (attack). The total number of instances across both the training and testing datasets is 527,608, with 433,067 instances belonging to class 0 (normal) and 94,541 instances belonging to class 1 (attack). The data suggests an imbalance in the class distribution, with a significantly higher number of instances in class 0 (normal) compared to class 1(attack).

```
Class distribution before SMOTE: Counter({0: 346464, 1: 75622})  
Class distribution after SMOTE: Counter({1: 346464, 0: 346464})
```

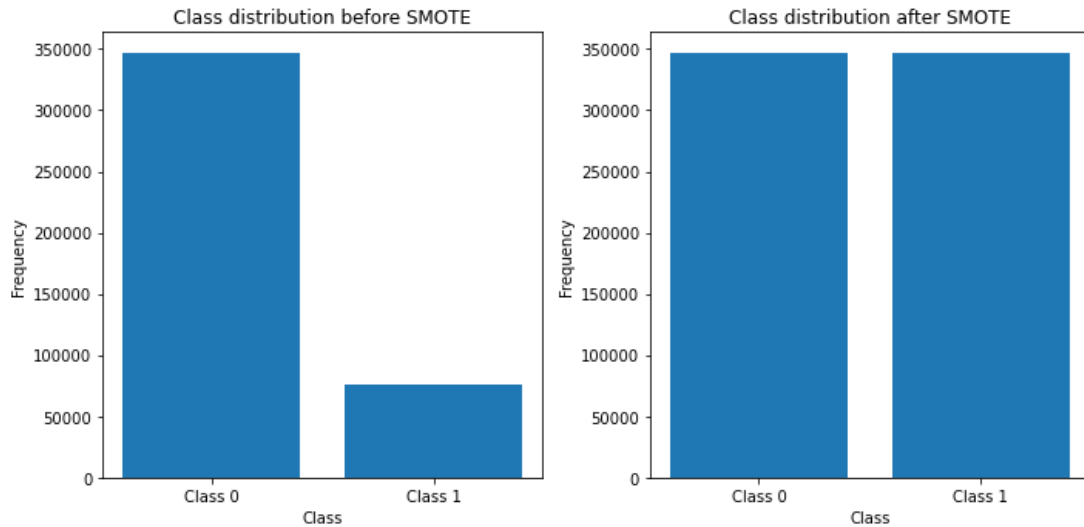


Figure 4. 1 Experiment result of SMOTE in Training dataset

The above Figure 4.1 illustrates the class distribution in the dataset before and after the applying of SMOTE (Synthetic Minority Over-sampling Technique). SMOTE is

employed to address class imbalance, a common issue where one class is significantly underrepresented compared to another. Before applying SMOTE, our dataset shows a considerable imbalance the training set contains 346,464 instances of Class 0 (majority class) and only 75,622 instances of Class 1 (minority class).

After applying SMOTE, the training set is balanced, with both Class 0 and Class 1 having 346,464 instances each. This adjustment ensures that the proposed model receives an equal representation of both classes during the training process, which can significantly enhance its ability to correctly predict the minority class.

Table 4. 2 Distribution of label class in training and testing after applying SMOTE

class	Training	Testing	Total
0	346,464	86,603	433,067
1	346,464	18,919	365,383
Total	692,928	105,522	798,450

The above table 4.2 shows the distribution of label classes in the training and testing datasets after applying the Synthetic Minority Over-sampling Technique (SMOTE). In the training dataset, there are a total of 692,928 instances. Out of these, 346,464 instances belong to class 0 (normal), and the same number, 346,464, belong to class 1 (attack). In the testing dataset, there are a total of 105,522 instances. Out of these, 86,603 instances belong to class 0 (normal), and 18,919 instances belong to class 1 (attack). The total number of instances across both the training and testing datasets is 798,450, with 433,067 instances belonging to class 0 (normal) and 365,383 instances belonging to class 1 (attack).

After applying SMOTE, the class distribution in the training dataset has been balanced, with an equal number of instances in both class 0 (normal) and class 1 (attack). This is a common technique used to address the problem of class imbalance, where one class significantly outnumbers the other. By oversampling the minority class (class 1 in this case), the model can learn more effectively from both classes, potentially improving its overall performance. The testing set remains unchanged, retaining its original class

distribution. This decision allows for an unbiased evaluation of the model's performance on the natural, imbalanced data.

4.3 Performance Evaluation Metrics

In our proposed model we have evaluated the model using the confusion matrix, accuracy, precision, recall, f1-score, and computational times.

Confusion matrix:

A confusion matrix is a square matrix where the rows represent the actual classes of instances, and the columns represent the predicted classes. The confusion matrix is a 2 X 2 matrix when dealing with a binary classification task.

The confusion matrix for binary class classification has 2 outputs, the inputs for this classification will fall in either of the 2 outputs or classes. The confusion matrix for two-class and is shown in Tables 4.3. The attack class is taken as a positive and the normal class as a negative.

Table 4. 3 Confusion matrix for two class classification

	Predicted Class		
		Normal	Attack
Actual class	Normal	TN	FP
	Attack	FN	TP

Where:

True Positive (TP): classified as an attack by the model and that is an attack.

True Negative (TN): classified a normal by the model and that is normal.

False Positive (FP): the model classified as an attack but that is normal.

False Negative (FN): the model classified as normal but that is an attack.

Accuracy, Precision, Recall, and F1-Score are then defined as follows(Krstinić et al., 2020).

Accuracy: is one of the evaluation metrics that calculate how many correct predictions your classification model made for the whole test dataset, and it is a good basic metric to measure the performance of the model. It is calculated as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad 2$$

Precision: is a form of performance evaluation that determines what number of the correctly predicted cases turned out to be positive. This can determine whether our model is reliable or not and it is a valuable metric in cases where a false positive is a higher concern than a false negative. It is calculated as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad 3$$

Recall: Recall is a type of performance evaluation metric that measures how many actual positive cases our model correctly predicted. It is particularly important in situations where false negatives are more critical than false positives. It is calculated as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad 4$$

F1-Score: is a type of performance evaluation that has a harmonic mean of precision and recall and so it gives a combined idea about these two metrics and it will be maximum when the precision is equal to recall. It is calculated as follows:

$$\text{F1-Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad 5$$

Computational time: The computation time represents the duration required for an algorithm to finish its operations. In our study, we evaluate each classifier by training and prediction time in seconds.

Cross validation:

To check whether the proposed model is efficient enough to predict the outcome of an unseen data point and performance we used k-fold cross-validation with k=3.

4.4 Hyper-parameters

The distributions of all the optimal hyper-parameters that were used in this experiment are described in Table 4.4.

We determined the optimal hyper-parameter for each classifier through a manual trial-and-error process using different hyper-parameters. We experiment with different hyper-parameter combinations, training the model, and evaluating its performance until to identify the best results.

Table 4. 4 Hyper-parameters for the classifier algorithm

Model	Optimal Hyper-parameters	Description
DT	max_depth=10, random_state=42	max_depth: Hyper-parameter that limits the maximum depth of the tree. random_state: Hyper-parameter that ensures reproducibility.
KNN	n_neighbors=5	n_neighbors: Hyper-parameter that specifies the number of neighbors to use
RF	n_estimators=100 max_depth=10 random_state=42	n_estimators: Hyper-parameter that specifies the number of trees in the forest. max_depth: Hyper-parameter that limits the maximum depth of the trees. random_state: Hyper-parameter that ensures reproducibility.
XGBoost	params = { 'objective': 'binary:logistic', 'eval_metric': ['error', 'logloss'], 'max_depth': 12, 'eta': 0.3, 'subsample': 0.7, 'colsample_bytree': 0.7, 'seed': 42	Objective: Hyper-parameter that defines the learning task and the corresponding learning objective. eval_metric: Hyper-parameter that specifies evaluation metrics for validation data. max_depth: Hyper-parameter that limits the maximum depth of a tree.

	<pre> } bst = xgb.XGBClassifier(**params) </pre>	<p>eta: Hyper-parameter for step size shrinkage to prevent over fitting.</p> <p>subsample: Hyper-parameter that specifies the fraction of samples to be used for each tree.</p> <p>colsample_bytree: Hyper-parameter that specifies the fraction of features to be used for each tree.</p> <p>seed: Hyper-parameter that ensures reproducibility.</p>
Stacking (DT+ KNN+RF +XGBoost)	<pre> estimators = [('dt', clf_dt), ('knn', clf_knn), ('rf', clf_rf), ('xgb', bst)] stacking_clf = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression()) </pre>	<p>Estimators: Hyper parameter that lists the base classifiers used in the stacking ensemble.</p> <p>final_estimator: Hyper-parameter that specifies the classifier used to aggregate the predictions of the base classifiers.</p>

4.5 Experiments and Results

Our experimental design was categorized into three main processes by considering DT, KNN, RF, XGBOOST, and Stacking (DT + KNN + RF + XGBoost) ML classification techniques. In the first step of our experiments, we employed the full features (78 features) of the CICIDS 2017 dataset for classification. In the second of our experiment we evaluated using 52 features which selected by MI. In the third phase, we evaluated those classifiers using the 25 features selected using the RFE. The results of our experimental processes are listed in **Tables 4.5, 4.6, and 4.7** whereby **Table 4.5** provides the results obtained by the ML methods for the classification technique using the full features of the CICIDS 2017 dataset. **Table 4.6** provides the result using 52 features, and

Table 4.7 lists the results obtain by the ML algorithms for the classification scheme using the reduced 25 features.

4.5.1 Experimental Results of Feature Selection Using Mutual Information

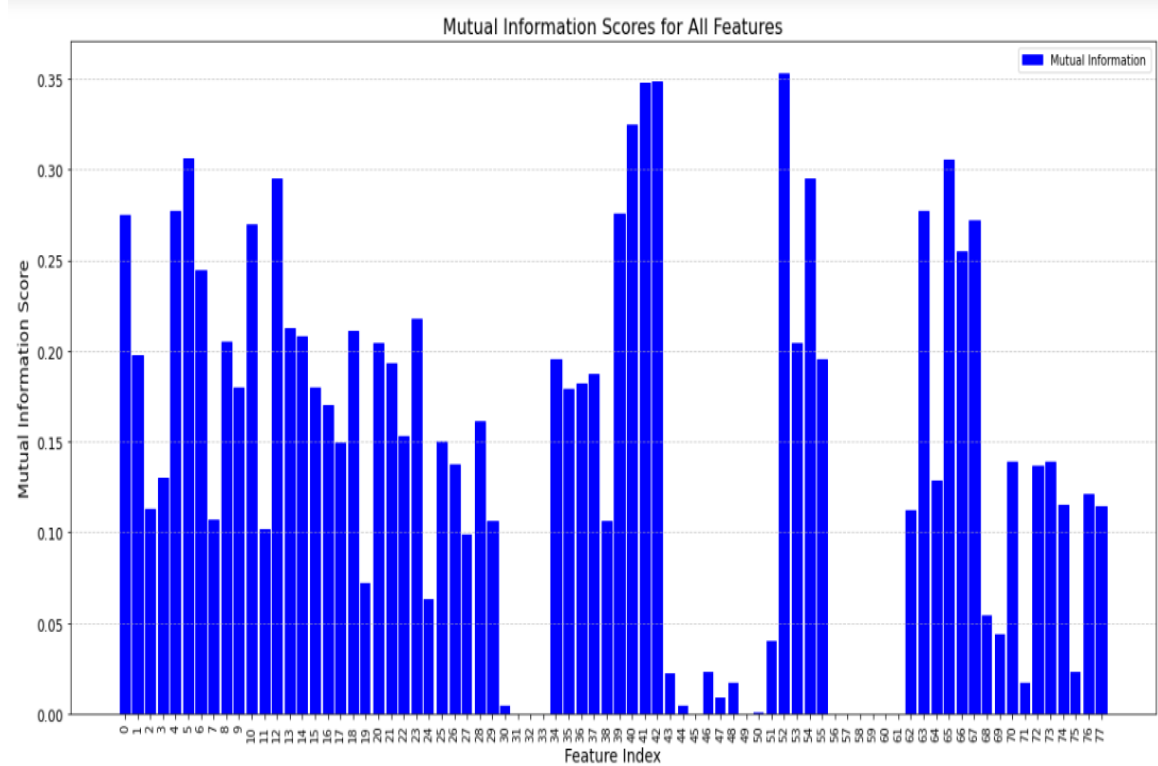


Figure 4. 2 Mutual Information (MI) Score of all the features

Figure 4.2 displays the mutual information scores for features indexed from 0 to 77, with the scores ranging from 0.00 to 0.36 on the y-axis and the feature indices on the x-axis. The highest mutual information score observed is approximately 0.36, indicating significant variability across different features. Notably, features indexed at 5, 40, 41, 42, 52, and 65 show the highest mutual information scores, exceeding 0.30, suggesting their strong relevance to the target variable.

Many features show moderate mutual information scores ranging between 0.10 and 0.30, including indices such as 0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 25, 26, 28, 29, 34, 35, 36, 37, 38, 39, 53, 54, 55, 62, 63, 64, 66, 67, 70, 72, 73, 74, 76, and 77. In contrast, features indexed at 19, 24, 27, 30, 31, 32, 33, 43, 44, 45, 46, 47, 48, 49, 50, 51, 56, 57, 58, 59, 60, 61, 68, 69, 71, and 75 have low mutual information

scores, often below 0.10, indicating minimal influence on the target variable and potential candidates for exclusion during feature selection.

Based on these observations, it is advisable to focus on features with higher mutual information scores for the second feature selection process. Evaluating the impact of removing low-scoring features could simplify models and improve computational efficiency. Out of a total of 78 features we selected only the top 52 features that are MI scores greater or equal to 0.10 for the second stage of feature selection.

Figure 4. 3 Selected features by MI

Total selected features by MI: 52
 Selected feature indices by MI: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 25, 26, 28, 29, 34, 35, 36, 37, 38, 39, 40, 41, 42, 52, 53, 54, 55, 62, 63, 64, 65, 66, 67, 70, 72, 73, 74, 76, 77]

4.5.2 Experimental Results of Feature Selection Using RFE

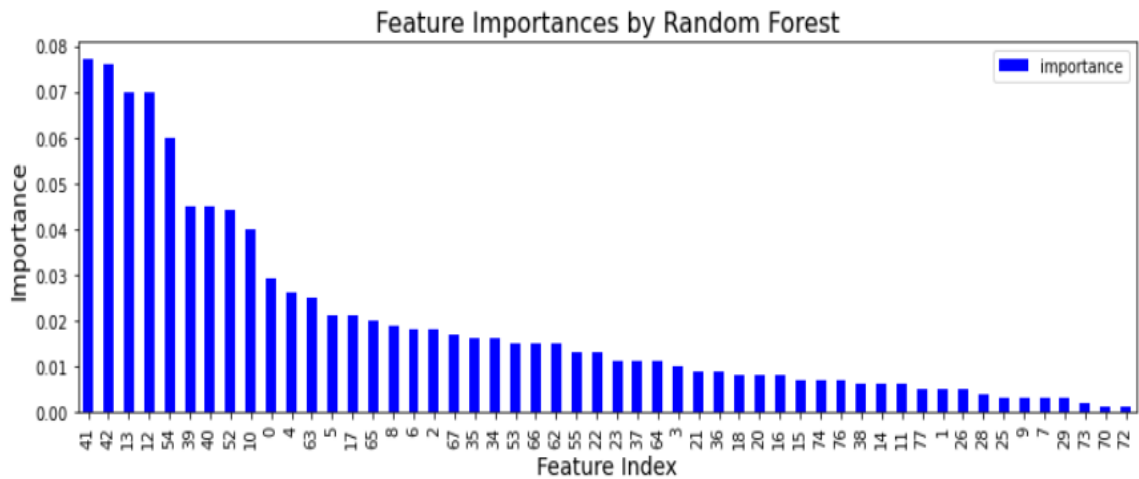


Figure 4. 4 Feature importance score of the selected features

Figure 4.4 illustrates the importance scores for 52 features, with the scores ranging from 0.00 to 0.08 using RFE with RF classifier. The highest importance scores are observed for features indexed at 41 and 42, each scoring around 0.07 to 0.08, indicating their significant relevance. Other features with high importance scores include indices 13, 12, 54, 39, 40, 52 and 10, with scores ranging from 0.04 to 0.06. Several features show moderate importance scores between 0.02 and 0.04, such as those indexed at 0, 4, 63, 5,

and 17. While these features are not as critical as the top-scoring ones, they still contribute significantly to the model.

In contrast, features indexed at 65, 8, 6, 2, 67, 35, 34, 53, 66, 62, 55, 22, 23, 37, 64, 3, 21, 36, 18, 20, 16, 15, 74, 76, 38, 14, and 11 have lower importance scores, generally below 0.02. The lowest importance scores are observed for features indexed at 17, 1, 26, 28, 25, 9, 7, 29, 73, 70, and 72, all scoring near 0.00. The figure shows a steep decline in importance scores from the highest-ranked feature to those ranked lower, with a noticeable drop after the top ten features.

Based on the figure features with high importance scores, such as indexed 41, 42, 13, and 12, should be prioritized in model evaluation. Moderate-importance features should be evaluated for their potential contributions and considered for inclusion based on their impact on the model's performance. Low-importance features might be candidates for exclusion to simplify the model and reduce complexity. Overall, emphasizing high-importance features could enhance model accuracy and efficiency. Out of the 52 features based on their importance we have selected 25 features to evaluate our proposed model.

Selected features after RFE: [0, 2, 4, 5, 6, 8, 12, 13, 18, 23, 34, 35, 37, 39, 40, 41, 42, 52, 54, 55, 62, 63, 65, 66, 67]
Final selected features (MI -> RFE) by original index:
[0, 2, 4, 5, 6, 8, 12, 13, 18, 23, 34, 35, 37, 39, 40, 41, 42, 52, 54, 55, 62, 63, 65, 66, 67]

Figure 4. 5 Selected features using Recursive feature elimination (RFE)

The above figure 4.5 displays a list of feature indices, which represent the most important features selected by the RFE algorithm. The feature indices are [0, 2, 4, 5, 6, 8, 12, 13, 18, 23, 34, 35, 37, 39, 40, 41, 42, 52, 54, 55, 62, 63, 65, 66, and 67]. The selected features shown in this figure are likely the top-ranked features based on the RFE algorithm's evaluation. These features are considered the most important as they provide the most relevant information for the model to make accurate predictions.

4.5.3 Confusion Matrix Experiment Result

The list below figures 4.6 to 4.10 shows the confusion matrices for the DT, KNN, RF, XGBoost, and Stacking (DT + KNN + RF + XGBoost) classifiers using the reduced set 25 features.

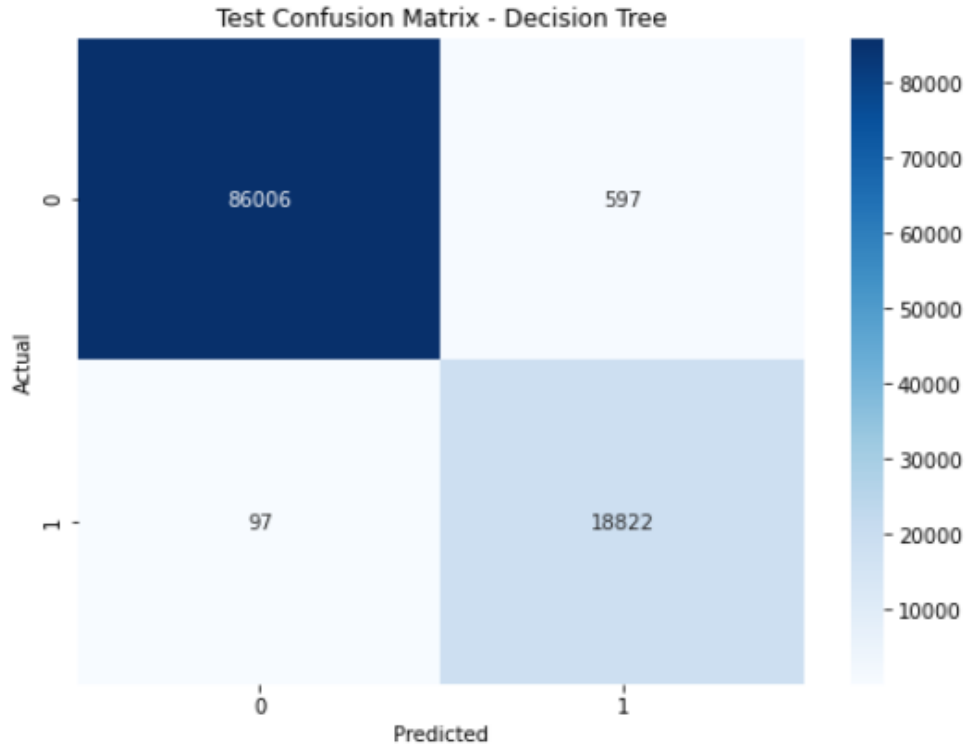


Figure 4. 6 Confusion matrix for DT

Figure 4.6 shows the confusion matrix for the DT classifier and reveals the distribution of actual versus predicted classifications. In the matrix, 86,006 instances were correctly predicted as class 0, while 18,822 instances were correctly predicted as class 1. However, there were also 597 instances where the model incorrectly predicted class 1 but the actual class was 0, and 97 instances where the model incorrectly predicted class 0 but the actual class was 1.

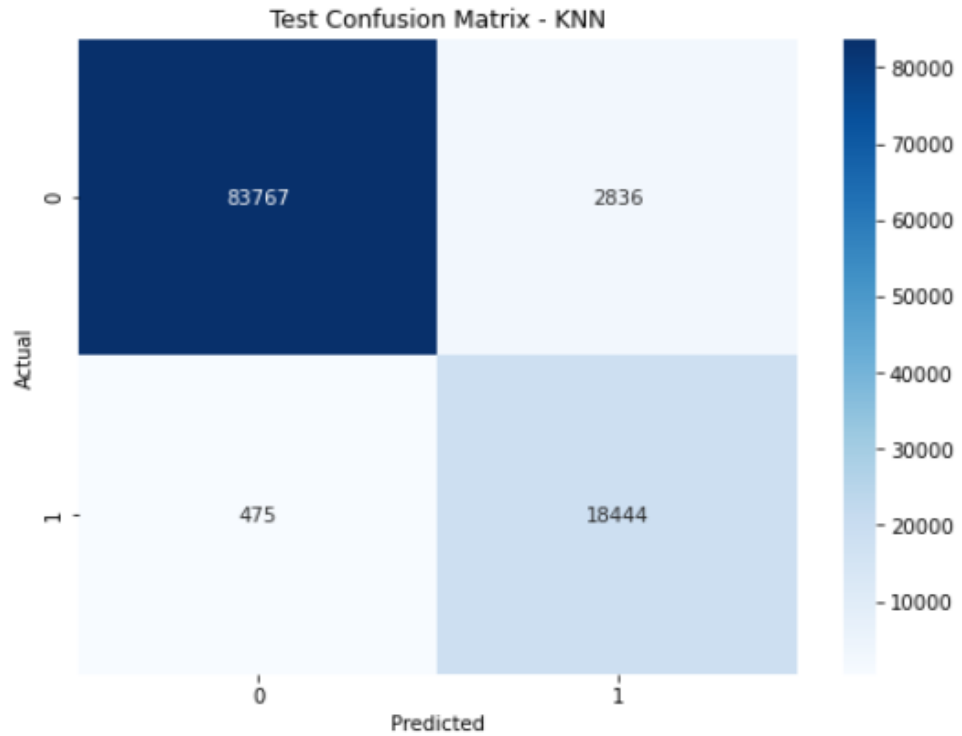


Figure 4. 7 Confusion matrix for KNN

Figure 4.7 shows the confusion matrix for the KNN classifier and demonstrates the model's performance in predicting class labels. It shows that 83,767 instances were correctly identified as class 0, while 18,444 instances were correctly identified as class 1. However, the model also made some errors, with 2,836 instances incorrectly predicted as class 1 when they were actual class 0, and 475 instances incorrectly predicted as class 0 when they were actual class 1.

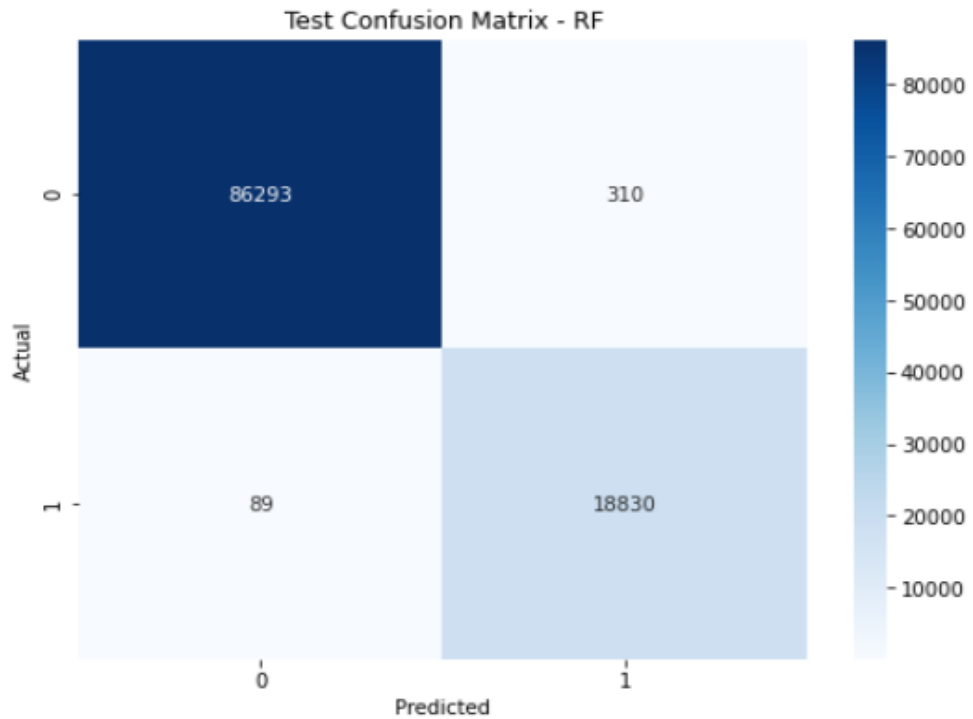


Figure 4. 8 Confusion matrix for RF

Figure 4.8 shows the confusion matrix for the RF classifier and reveals the model's performance in classifying instances. In this matrix, 86,293 instances were correctly predicted as class 0, and 18,830 instances were correctly predicted as class 1. However, the model also made some errors, with 310 instances incorrectly predicted as class 1 when they were actual class 0, and 89 instances incorrectly predicted as class 0 when they were actual class 1.

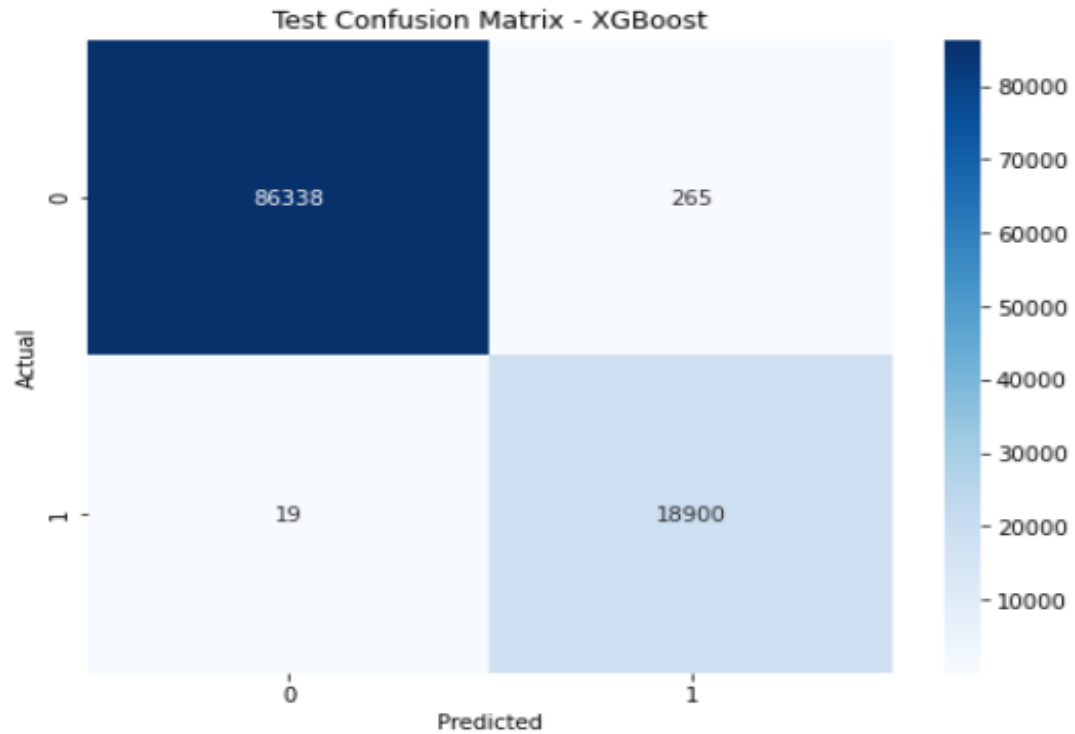


Figure 4. 9 Confusion matrix for XGBoost

Figure 4.9 shows the confusion matrix for the XGBoost classifier and reveals the model's performance in classifying instances. In this matrix, 86,338 instances were correctly predicted as class 0, and 18,900 instances were correctly predicted as class 1. However, the model also made some errors, with 265 instances incorrectly predicted as class 1 when they were actual class 0, and 19 instances incorrectly predicted as class 0 when they were actual class 1.

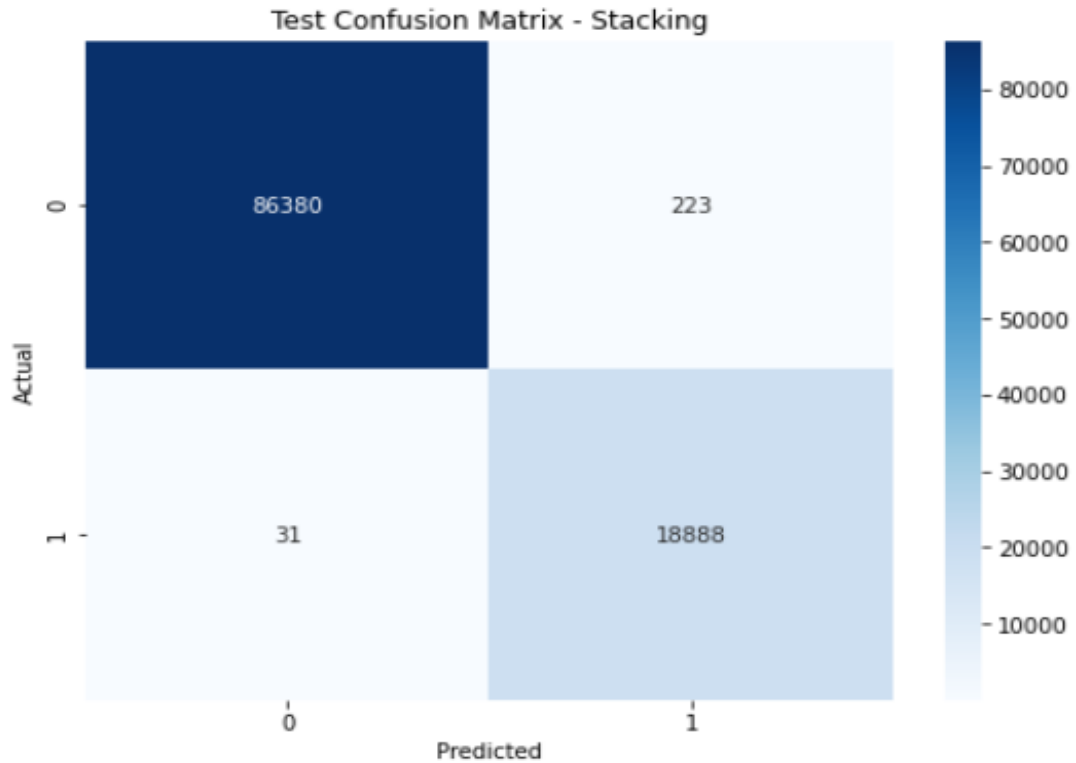


Figure 4. 10 Confusion matrix for Stacking (DT + KNN + RF + XGBoost)

Figure 4.10 shows the confusion matrix for the Stacking (DT + KNN + RF + XGBoost) classifier and reveals the model's performance in classifying instances. In this matrix, 86,380 instances were correctly predicted as class 0, and 18,888 instances were correctly predicted as class 1. However, the model also made some errors, with 223 instances incorrectly predicted as class 1 when they were actual class 0, and 31 instances incorrectly predicted as class 0 when they were actual class 1.

4.5.4 Experimental results without feature selection with full features

Table 4. 5 Experiment results using full features

Algorithms	Classes	Precision	Recall	F1-score	Accuracy	Training Time (s)	Testing Time(s)
DT	0	0.9885	0.9371	0.9621	0.9394	12.5006	0.0312
	1	0.7676	0.9503	0.8492			
KNN	0	0.9979	0.9890	0.9934	0.9772	131.8696	40.0321
	1	0.9518	0.9905	0.9708			
RF	0	0.9980	0.9980	0.980	0.9941	488.2859	1.3570
	1	0.9907	0.9910	0.9908			
XGBoost	0	0.9997	0.9991	0.9994	0.9954	22.2843	0.1269
	1	0.9957	0.9988	0.9973			
Staking (DT+KNN+ RF+ XGBoost)	0	0.9997	0.9991	0.9994	0.9975	595.9680	302.612 3
	1	0.9958	0.9987	0.9973			

The above table 4.5 shows the results of the performance of Decision Tree (DT), K-Nearest Neighbors (KNN), Random Forest (RF), XGBoost, and Stacking (DT + KNN + RF + XGBoost) algorithms without applying the feature selection method with full features. We evaluate those algorithms using precision, recall, F1-score, accuracy, training time, and testing time evaluation metrics for each algorithm for both classes (0 and 1).

In DT we conducted experiments using different models based on the tree's maximum depth using `maximum_depth_values = {3, 5, 10, and 11}` and we got better results with `maximum_depth` values 10. Its performance for class 0, with a precision of 98.85% for class 0 and 76.76% for class 1, recall of 93.71% for class 0 and 95.03% for class 1, F1-score of 96.21% for class 0 and 84.92% for class 1.. The overall accuracy is 93.94%. The key advantage of DT is its fast training of 12.5006 seconds and testing 0.0312 seconds

times, making it suitable for real-time applications or scenarios with limited computational resources.

In KNN we trained the models with multiple number_of_neighbours = {3, 5, 7, and 10}, and the results show that a KNN classifier with 5 neighbors achieved better results. KNN performs with precision of 99.79% for class 0 and 95.18% for class 1, recall 98.90% for class 0 and 99.05% for class 1, and F1-scores 99.34% for class 0 and 97.08% for class 1. This leads to an overall accuracy of 97.72%. However, KNN requires significantly more time for training 131.8696 seconds and testing 40.0321 seconds compared to DT, which may limit its applicability in time-sensitive or resource-constrained environments.

RF performed precision 99.80% for class 0 and 99.07% for class 1, recall 99.80% for class 0 and 99.10% for class 1, and F1-scores 99.80% for class 0 and 99.08% for class 1. It performs an overall accuracy of 99.41%. The high precision, recall, and F1 scores indicate that RF is highly effective at correctly classifying instances. However, its long training time of 488.2859 seconds is a drawback, making this RF less suitable for applications requiring rapid model updates. The testing time of 1.3570 seconds is relatively short making RF a good choice for scenarios where model inference needs to be fast, but training time is less of a concern.

XGBoost performed precision of 99.97% for class 0 and 99.57% for class 1, recall 99.91% for class 0 and 99.88% for class 1, and F1-scores 99.94% for class 0 and 99.73% for class 1, its overall accuracy of 99.54%. It also offers a good balance between training at 22.2843 seconds and testing at 0.1269 seconds times. The relatively short training time compared to RF makes XGBoost suitable for scenarios requiring frequent model retraining, while its quick testing time supports fast inference. XGBoost is a strong candidate for a wide range of applications where both performance and computational efficiency are important.

Stacking (DT + KNN + RF + XGBoost) performs a precision of 99.97% for class 0 and 99.58% for class 1, recall of 99.91% for class 0 and 99.87% for class 1, and F1-scores 99.94% for class 0 and 99.73% for class 1, resulting in an overall accuracy of 99.75%. Despite its superior performance, the computational cost is substantial, with the longest training of 595.9680 seconds, and testing 302.6123 seconds times among all algorithms

evaluated. Stacking (DT + KNN + RF + XGBoost) is best suited for applications where the highest possible accuracy is critical and computational resources are not a limiting factor. Its extensive training and testing times make it less practical for real-time applications or environments with limited computational capacity.

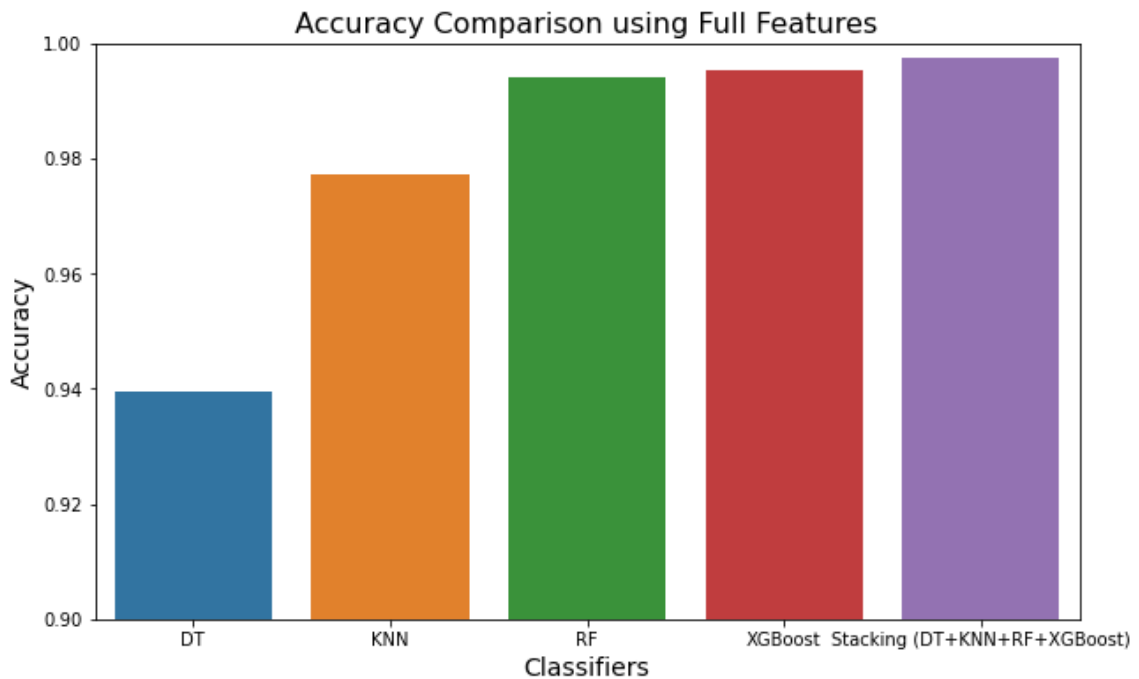


Figure 4. 11 Accuracy Comparison of Full Features

Figure 4.11 histogram charts compare the accuracy of DT, KNN, RF, XGBoost, and Stacking (DT + KNN + RF + XGBoost) using the full features of the dataset. As shown in the figure when we combined the algorithm using stacking the model improved its accuracy than the individual classifier's accuracy.

4.5.5 Experimental results of features selected by MI

Table 4. 6 Experiment result of 52 feature selected by MI

Algorithm	Classes	Precision	Recall	F1-score	Accuracy	Training Time (s)	Testing Time(s)
DT	0	0.9887	0.9373	0.9623	0.9396	11.9254	0.0284
	1	0.7666	0.9507	0.8488			
KNN	0	0.9978	0.9881	0.9929	0.9887	120.0365	33.0215
	1	0.9473	0.9899	0.9681			
RF	0	0.9976	0.9981	0.9979	0.9950	327.6088	1.3535
	1	0.9913	0.9892	0.9902			
XGBoost	0	0.9997	0.9987	0.9992	0.9961	14.1059	0.1086
	1	0.9940	0.9988	0.9964			
Stacking (DT+KNN+ RF +XGBoost)	0	0.9997	0.9990	0.9993	0.9980	540.5462	250.8457
	1	0.9953	0.9985	0.9969			

Table 4.6 shows the performance metrics of the Decision Tree (DT), K-Nearest Neighbors (KNN), Random Forest (RF), XGBoost, and Stacking (DT +KNN +RF + XGBoost) using the 52 features for both 0 and 1 classes. The evaluation metrics include precision, recall, F1-score, accuracy, training time, and testing time for each class (0 and 1).

The DT model shows moderate performance with class 0 achieving a precision of 98.87 %, recall of 93.73 %, and F1-score of 96.23%. For class 1, the precision is 76.66%, with a recall of 95.07% and F1-score of 84.88%. The overall accuracy of the DT model is 93.96%. In terms of computational efficiency, it has a training time of 11.9254 seconds and a very quick testing time of 0.0284 seconds, making it suitable for real-time.

The KNN model shows class 0 with a precision of 99.78%, recall of 98.81%, and F1-score of 99.29%. For class 1, it achieves a precision of 94.73%, recall of 98.99%, and F1-score of 96.81%. The overall accuracy stands at 98.87%. However, the model is computationally intensive, with a training time of 120.0365 seconds and a testing time of 33.0215 seconds. Despite its high accuracy, the substantial computational cost may limit its usability in time-sensitive applications.

The RF model performs for class 0 with a precision of 99.76%, recall of 99.81%, and F1-score of 99.79%. For class 1, a precision of 99.13%, recall of 98.92%, and F1-score of 99.02%. The overall accuracy of the RF model is 99.50%. In terms of computational resources, it has a training time of 327.6088 seconds and a testing time of 1.3535 seconds, making it a robust choice with a good balance of accuracy and computational efficiency.

The XGBoost model shows performance for class 0 a precision of 99.97%, recall of 99.87%, and F1-score of 99.95%. For class 1, the precision is 99.40%, recall is 99.88%, and F1-score is 99.64%. The overall accuracy of the XGBoost model is 99.61%. It is relatively efficient, with a training time of 14.1059 seconds and a testing time of 0.1086 seconds, offering a strong balance between high performance and computational efficiency.

The Stacking (DT + KNN + RF + XGBoost) model, which combines DT, KNN, RF, and XGBoost, achieves the highest performance metrics across all categories. For class 0, it attains a precision of 99.97%, recall of 99.90%, and F1-score of 99.93%. For class 1, the precision is 99.53%, recall is 99.85%, and F1-score is 99.69%. The overall accuracy of the Stacking model is 99.80%, the highest among all models tested. However, it has the longest training time at 540.5462 seconds and a significant testing time of 250.8457 seconds, indicating a high computational cost which may limit its practical application despite its superior accuracy.

4.5.6 Experimental results of hybrid feature selection (MI + RFE)

Table 4.7 Experiment results using reduced features

Algorithm	Classes	Precision	Recall	F1-score	Accuracy	Training Time (s)	Testing Time(s)
DT	0	0.9989	0.9931	0.9969	0.9934	11.8113	0.0200
	1	0.9693	0.9949	0.9819			
KNN	0	0.9944	0.9673	0.9806	0.9945	115.2157	29.8378
	1	0.8667	0.9749	0.9176			
RF	0	0.9990	0.9964	0.9977	0.9962	309.3665	1.3832
	1	0.9838	0.9953	0.9895			
XGBoost	0	0.9998	0.9969	0.9984	0.9973	12.7620	0.1042
	1	0.9862	0.9990	0.9925			
Stacking (DT+KNN + RF +XGBoost)	0	0.9996	0.9974	0.9985	0.9992	515.8013	188.5048
	1	0.9956	0.9982	0.9969			

Table 4.7 shows the performance metrics of the Decision Tree (DT), K-Nearest Neighbors (KNN), Random Forest (RF), XGBoost, and Stacking (DT +KNN +RF + XGBoost) using the reduced features for both 0 and 1 classes. The evaluation metrics

include precision, recall, F1-score, accuracy, training time, and testing time for each class (0 and 1).

DT performs an overall accuracy of 99.34%, with a precision of 99.89% for class 0 and 96.93% for class 1, and a recall of 99.31% for class 0 and 99.49% for class 1. The F1 scores are also 99.69% for class 0 and 98.19% for class 1, indicating a balanced performance. The model benefits from very short training and testing times, making it computationally efficient. This makes DT a feasible option when interpretability and speed are prioritized.

KNN achieves an accuracy of 96.86%. The precision 99.44% for class 0 and 86.67% for class 1, recall 96.73% for class 0 and 97.47% for class 1, and F1-scores 98.06% for class 0 and 91.76% for class 1. The training and testing times are significantly higher, suggesting that KNN may not be suitable for large datasets or real-time applications due to its computational inefficiency.

RF performs an accuracy of 99.62% and balanced high precision 99.90% for class 0 and 98.38% for class 1, recall 99.64% for class 0 and 99.53% for class 1, and F1-scores 99.77% for class 0 and 98.95% for class 1. However, the training time is significantly longer than DT and XGBoost. RF's robustness and high accuracy make it suitable for applications where performance is critical, but the computational cost must be considered.

XGBoost performs an accuracy of 99.62% with relatively low training of 12.7620 seconds and testing times of 0.1042 seconds, important to its efficiency. The precision 99.98% for class 0 and 98.38% for class 1, recall 99.69% for class 0 and 99.90% for class 1, and F1-scores 99.84% for class 0 and 99.25% for class 1. XGBoost provides a good balance between performance and computational efficiency, making it a practical choice for many applications in intrusion detection.

The Stacking (DT + KNN + RF + XGBoost) method achieves an accuracy of 99.76% and excellent performance metrics for both classes. Precision 99.96% for class 0 and 99.56% for class 1, recall 99.74% for class 0 and 99.82% for class 1, and F1-scores 99.85% for class 0 and 99.69% for class 1 are very high, indicating superior performance. However,

it has the highest training 515.8013 seconds, and testing times 188.5048 seconds, indicating significant computational costs. Stacking (DT + KNN + RF + XGBoost) is ideal for scenarios where the utmost accuracy is required, and computational resources and time are not limiting factors.

In conclusion, while Stacking (DT + KNN + RF + XGBoost) and Random Forest offer the best performance metrics, their high computational cost should be considered. XGBoost stands out for its balance of high accuracy and efficiency, making it suitable for practical applications. Decision Tree provides a good trade-off between simplicity and performance, whereas KNN, due to its inefficiency, might be less suitable for large-scale or real-time tasks.

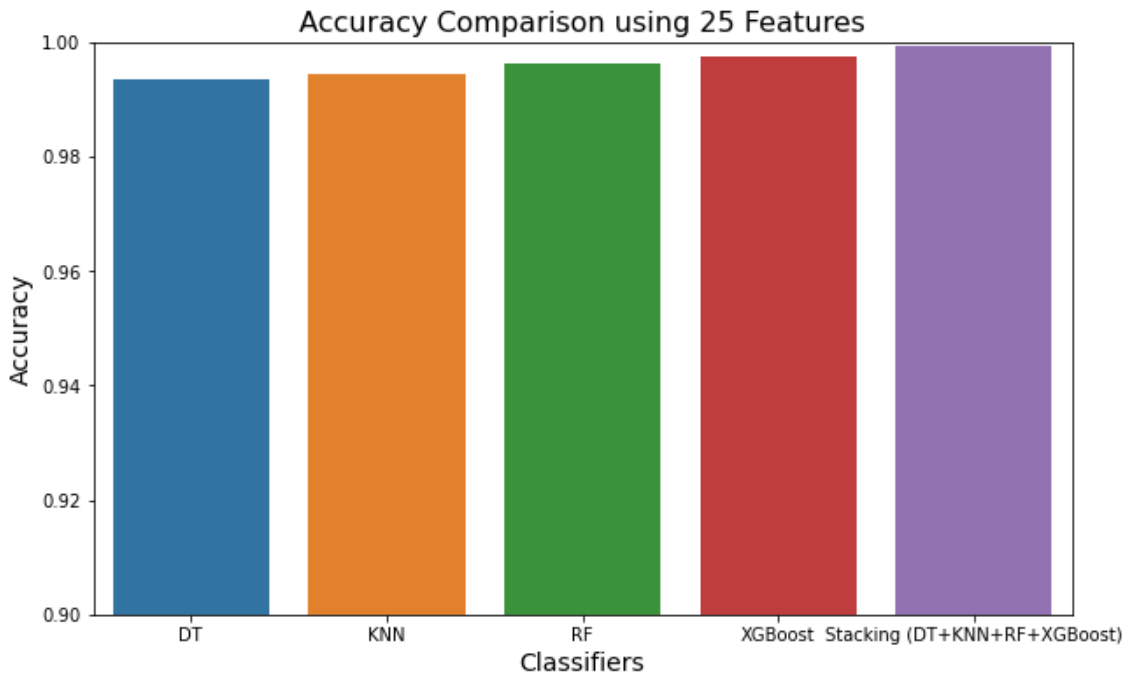


Figure 4. 12 Accuracy Comparison of Reduced Features

In the histogram chart in Figure 4.12 above, we present an analysis of the performance of DT, KNN, RF, XGBoost, and Stacking (DT + KNN + RF + XGBoost) algorithms in terms of accuracy when the number of features is reduced. From the chart, it is evident that all the algorithms performed exceptionally well with reduced features, achieving accuracy above 99%.

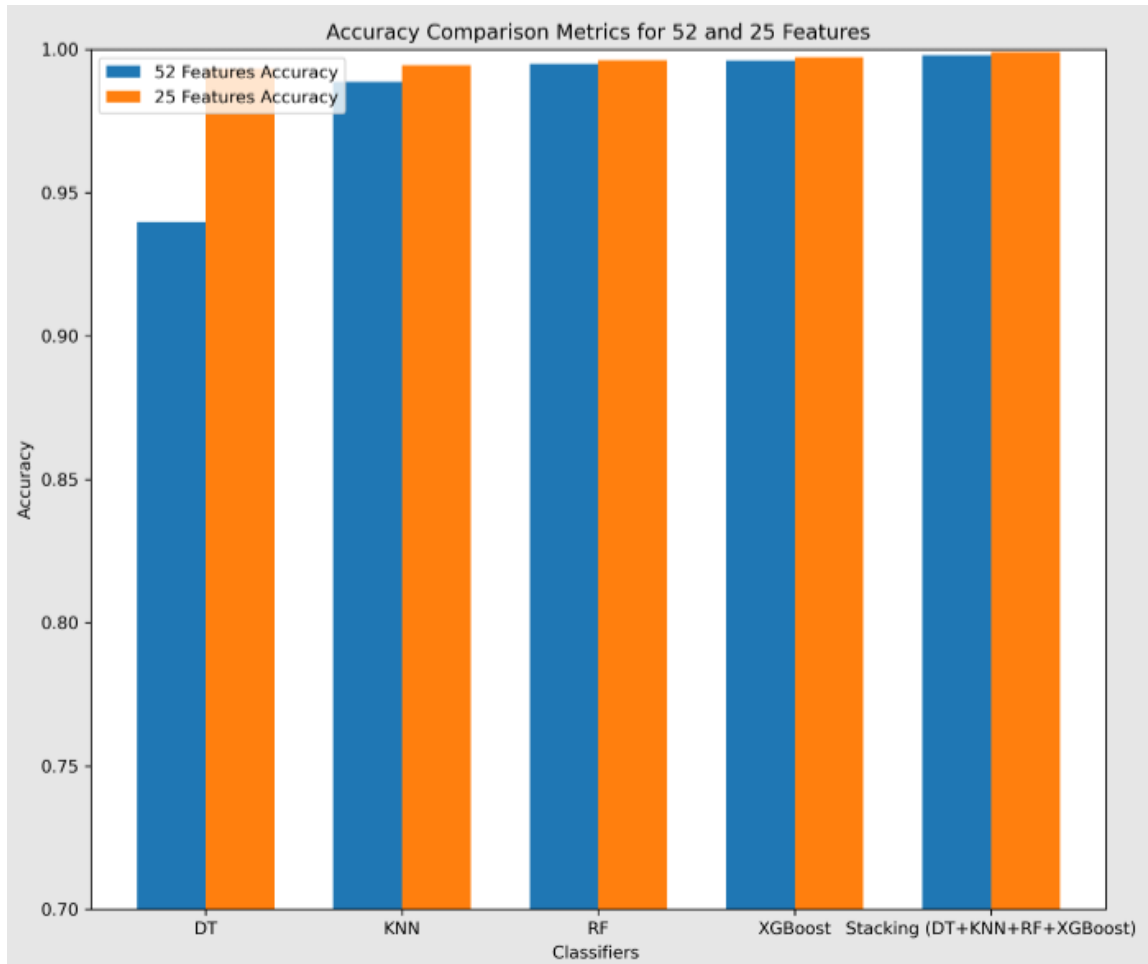


Figure 4. 13 Accuracy Comparison of the 25 and 52 Features

Figure 4.13 shows a comparison of classifier accuracies using 52 Features and 25 Features for DT, KNN, RF, XGBoost, and a Stacking (DT+ KNN +RF + XGBoost).

The results indicate that reducing the number of features can lead to an improvement in classifier performance, particularly for the DT and KNN classifiers. This improvement could be attributed to the reduction of noise and the simplification of the model, which allows the classifiers to generalize better to the test data.

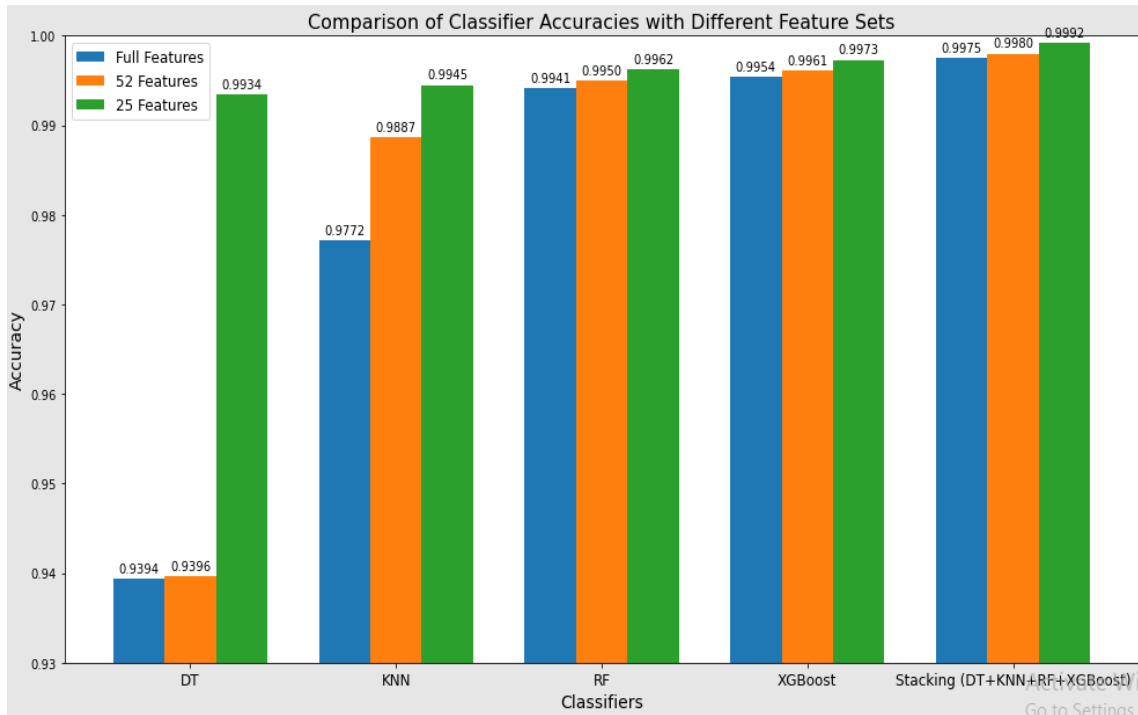


Figure 4. 14 Accuracy Comparison of the 25, 52 and full (78) Features

Figure 4.14 shows a comparison of classifier accuracies using Full Features (78), 52 Features, and 25 Features for DT, KNN, RF, XGBoost, and a Stacking (DT+ KNN +RF + XGBoost).

The results indicate that reducing the number of features can lead to an improvement in classifier performance, particularly for the DT and KNN classifiers. This improvement could be attributed to the reduction of noise and the simplification of the model, which allows the classifiers to generalize better to the test data.

For the RF and XGBoost classifiers, the accuracy remains consistently high across all feature sets, suggesting that these ensemble methods are robust to changes in the number of features. However, they still show slight improvements with feature reduction, indicating that even these models benefit from feature selection.

The Stacking ((DT+ KNN +RF + XGBoost)) ensemble method outperforms all individual classifiers, demonstrating the effectiveness of combining multiple models to

leverage their strengths. The highest accuracy of 0.9992 is achieved with 25 features, which underscores the importance of feature selection in improving the performance of complex models.

In conclusion, our study highlights the critical role of feature selection in intrusion detection systems demonstrating that reducing the feature set to the most informative ones can significantly enhance the performance of various algorithms, particularly DT and KNN, while still benefiting robust RF, XGBoost, and Stacking (DT + KNN + RF + XGBoost).

4.5.7 Sample Validation results

The following figures 4.15, 4.16 and 4.17 display sample cross-validation for our experiments using 3 folds.

```
Cross-Validation Scores (DT): [0.99405133 0.99452324 0.99446696]  
Mean Cross-Validation Score (DT): 0.9943
```

Figure 4. 15 Cross-validation for DT

Figure 4.15 shows DT classifier has been assessed through a three-fold cross-validation procedure, where the dataset is divided into three equally sized folds. The model is trained on two folds and validated on the remaining fold, and this process is repeated three times, each time with a different fold as the validation set. The scores across the three folds indicate high and consistent performance, with an average accuracy of approximately 99.43%. This suggests that the DT model is performing well on the given dataset and is likely to generalize effectively to unseen data.

```
Cross-Validation Scores (RF): [0.99491289 0.9956489 0.99541511]  
Mean Cross-Validation Score (RF): 0.9953
```

Figure 4. 16 Cross-validation for RF

Figure 4.16 shows the RF classifier has undergone a three-fold cross-validation evaluation, ensuring that each fold of the dataset is used once as a validation set while the remaining folds serve as the training set. The **mean cross-validation score** of 99.53% is an average of the accuracy scores across all folds. This high average accuracy suggests

that the RF classifier is highly effective and performs consistently well across different partitions of the data.

Cross-Validation Scores (Stacking): [0.99892521 0.99909856 0.99835747]
 Mean Cross-Validation Score (Stacking): 0.9988

Figure 4. 17 Cross-validation for stacking (DT+ KNN + RF + XGBoost)

Figure 4.17 shows stacking (DT + KNN + RF + XGBoost) classifier has undergone a three-fold cross-validation evaluation, ensuring that each fold of the dataset is used once as a validation set while the remaining folds serve as the training set. The mean cross-validation score of 99.88% is an average of the accuracy scores across all folds. This high average accuracy suggests that the stacking classifier is highly effective and performs consistently well across different partitions of the data.

Table 4. 8 Comparison of training and testing time

Algorithm	Full Features(78)		Reduced Features(25)	
	Training Time(s)	Testing Time(s)	Training Time(s)	Testing Time(s)
DT	12.5006	0.0312	11.8113	0.02
KNN	131.8696	40.0321	115.2157	29.8378
RF	488.2859	1.357	309.3665	1.1232
XGBoost	22.2843	0.1269	12.762	0.1042
Stacking (DT + KNN + RF + XGBoost)	595.968	302.6123	515.8013	188.5048

The above table 4.8 compares the training and testing times (in seconds) for Decision Tree (DT), K-Nearest Neighbors (KNN), Random Forest (RF), XGBoost, and Stacking (DT + KNN + RF + XGBoost) algorithms using both full features and reduced features.

For the Decision Tree algorithm, the training time decreased from 12.5006 seconds with full features to 11.8113 seconds with reduced features, while the testing time reduced from 0.0312 seconds to 0.02 seconds. The K-Nearest Neighbors (KNN) algorithm exhibited a significant reduction in both training and testing times, with training time

decreasing from 131.8696 seconds to 115.2157 seconds, and testing time decreasing from 40.0321 seconds to 29.8378 seconds when using reduced features.

The Random Forest algorithm showed a substantial decrease in training time, from 488.2859 seconds with full features to 309.3665 seconds with reduced features, along with a modest reduction in testing time, from 1.357 seconds to 1.1232 seconds. XGBoost demonstrated a noticeable reduction in both training and testing times, with training time decreasing from 22.2843 seconds to 12.762 seconds and testing time from 0.1269 seconds to 0.1042 seconds when the number of features was reduced.

The Stacking (DT + KNN + RF + XGBoost) method saw a significant decrease in both training and testing times, with training time reducing from 595.968 seconds to 515.8013 seconds, and testing time from 302.6123 seconds to 188.5048 seconds with reduced features.

Overall, reducing the number of features from 78 to 25 resulted in lower training and testing times across all evaluated algorithms. The impact was most pronounced in the K-Nearest Neighbors and Stacking (DT + KNN + RF + XGBoost) algorithms, where the reduction in the number of features led to substantial time savings. Decision Tree and XGBoost also benefited from reduced feature sets, but the changes were less dramatic compared to KNN and Stacking (DT + KNN + RF + XGBoost). Random Forest showed a significant decrease in training time, though the reduction in testing time was relatively modest. In summary, feature reduction improves computational efficiency for all algorithms, making it a valuable technique for optimizing model performance and resource utilization.

4.6 Answer to the research questions

RQ#1: What are the features selected through the Hybrid Feature Selection method?

Through the Hybrid Feature Selection method, the following features were identified as the most relevant for the model:

- Destination Port, Total Fwd Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Mean, Bwd Packet

Length Mean, Bwd Packet Length Std, Flow IAT Max, Fwd IAT Max, Fwd Header Length, Bwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Mean, Packet Length Std, Packet Length Variance, Average Packet Size, Avg Bwd Segment Size, Fwd Header Length, Subflow Fwd Packets, SubflowFwd Bytes, Subflow Bwd Bytes, Init_Win_bytes_forward, Init_Win_bytes_backward

RQ#2: Does implementing a Hybrid Feature Selection method improve the performance of the model?

Based on our experiments implementation of a Hybrid Feature Selection method improves the performance of the model in accuracy as well as computational time.

- **DT:** Accuracy increased from 93.94% to 99.34%.
- **KNN:** Accuracy increased from 97.72% to 99.45%.
- **RF:** Accuracy increased from 99.41% to 99.62%.
- **XGBoost:** Accuracy increased from 99.54% to 99.73%.
- **Stacking (DT + KNN + RF + XGBoost):** Accuracy increased from 99.75% to 99.92%.

The Hybrid Feature Selection method significantly improves the performance of intrusion detection models by enhancing both their accuracy and computational efficiency. By selecting the most relevant features, the method ensures that models are not only more accurate but also faster to train and test, making them more practical for real-world applications.

RQ#3: Does a combining classifier algorithm enhance the performance of the proposed model?

The combining classifier algorithm using Stacking (DT + KNN + RF + XGBoost) demonstrates a significant enhancement in the performance of our proposed model

- The Stacking (DT + KNN + RF + XGBoost) classifier achieves an accuracy of 99.75% with the full feature set, improving the performance of all individual base classifiers (DT, KNN, RF, and XGBoost).

- With the reduced features, the Stacking (DT + KNN + RF + XGBoost) classifier further improves its accuracy to 99.92%, indicating that combining classifiers enhances performance even with fewer features.

CHAPTER FIVE

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

This study proposed both Hybrid Feature Selection methods by combining Mutual Information (MI) and RFE, and combining classifier algorithms using Stacking (DT + KNN + RF + XGBoost) for intrusion detection systems significantly enhances the performance of the proposed model. This feature selection approach successfully solves the problem which is raised by the high dimensional dataset in IDS; the experiment was conducted using the CICIDS 2017 dataset, which contains 78 features. These 78 features were reduced to 25 features by the proposed feature selection technique. Initially, we carried out the experiments using the proposed ML approaches over the full features of the CICIDS 2017 dataset. Then we conduct the experiments using the reduced features that were generated by the MI-RFE feature selection algorithm proposed in this work. The experimental results demonstrated that using reduced (optimal) features has improved the accuracy and computation time of the proposed method. The proposed method improved the accuracy of Decision Trees (DT), K-nearest neighbors (KNN), Random Forests (RF), and XGBoost, with accuracy increases ranging from moderate to substantial. In particular, 99.75% accuracy with full features was achieved by the Stacking (DT + KNN + RF + XGBoost) classifier, which combines base classifiers, and 99.92% accuracy with reduced features. By combining classifiers and choosing the most pertinent features, this dual approach makes the most of each method's advantages and produces an intrusion detection model that is both more accurate and effective.

5.2 Recommendations

- The dataset used for this thesis was from publicly available data. For future research, we recommend testing the proposed model using local datasets.
- We recommend testing the proposed method using other intrusion detection datasets.

REFERENCES

- Aghdam, M. H., & Kabiri, P. (2016). Feature Selection for Intrusion Detection System Using Ant Colony Optimization. https://doi.org/10.1007/978-3-319-32213-1_27
- Ahmim, A., Maglaras, L., Ferrag, M. A., Derdour, M., & Janicke, H. (2018). A Novel Hierarchical Intrusion Detection System based on Decision Tree and Rules-based Models (arXiv:1812.09059). arXiv. <http://arxiv.org/abs/1812.09059>
- Ahsan, R., Shi, W., & Corriveau, J. (2022). Network intrusion detection using machine learning approaches: Addressing data imbalance. *IET Cyber-Physical Systems: Theory & Applications*, 7(1), 30–39. <https://doi.org/10.1049/cps2.12013>
- Al Lail, M., Garcia, A., & Olivo, S. (2023). Machine Learning for Network Intrusion Detection—A Comparative Study. *Future Internet*, 15(7), 243. <https://doi.org/10.3390/fi15070243>
- Alalhareth, M., & Hong, S.-C. (2023). An Improved Mutual Information Feature Selection Technique for Intrusion Detection Systems in the Internet of Medical Things. *Sensors*, 23(10), 4971. <https://doi.org/10.3390/s23104971>
- Almasoudy, F. H., Al-Yaseen, W. L., & Idrees, A. K. (2020). Differential Evolution Wrapper Feature Selection for Intrusion Detection System. *Procedia Computer Science*, 167, 1230–1239. <https://doi.org/10.1016/j.procs.2020.03.438>
- Alom, M. Z., & Taha, T. M. (2017). Network intrusion detection for cyber security using unsupervised deep learning approaches. 2017 IEEE National Aerospace and Electronics Conference (NAECON), 63–69. <https://doi.org/10.1109/NAECON.2017.8268746>

- Al-rimy, B. A. S., Maarof, M. A., Alazab, M., Shaid, S. Z. M., Ghaleb, F. A., Almalawi, A., Ali, A. M., & Al-Hadhrami, T. (2021). Redundancy Coefficient Gradual Up-weighting-based Mutual Information Feature Selection technique for Cryptoransomware early detection. *Future Generation Computer Systems*, 115, 641–658. <https://doi.org/10.1016/j.future.2020.10.002>
- Ambikavathi, C., & Srivatsa, S. K. (2020). Predictor Selection and Attack Classification using Random Forest for Intrusion Detection. 79. <http://nopr.niscpr.res.in/handle/123456789/54710>
- Ambusaidi, M. A., He, X., Nanda, P., & Tan, Z. (2016). Building an Intrusion Detection System Using a Filter-Based Feature Selection Algorithm. *IEEE Transactions on Computers*, 65(10), 2986–2998. <https://doi.org/10.1109/TC.2016.2519914>
- Arif Ali, Z., H. Abduljabbar, Z., A. Tahir, H., Bibo Sallow, A., & Almufti, S. M. (2023). eXtreme Gradient Boosting Algorithm with Machine Learning: A Review. *Academic Journal of Nawroz University*, 12(2), 320–334. <https://doi.org/10.25007/ajnu.v12n2a1612>
- Asir, D., Appavu, S., & Jebamalar, E. (2016). Literature Review on Feature Selection Methods for High-Dimensional Data. *International Journal of Computer Applications*, 136(1), 9–17. <https://doi.org/10.5120/ijca2016908317>
- Assistant Professor, Department of Information technology, Bishop Heber College, Affiliated to Bharathidasan University, Tiruchirappalli, 620 024, Tamil Nadu, India, Usha, P., & Anuradha, M. P. (2023). Feature Selection Techniques in Learning Algorithms to Predict Truthful Data. *Indian Journal Of Science And Technology*, 16(10), 744–755. <https://doi.org/10.17485/IJST/v16i10.2102>

- Berman, D., Buczak, A., Chavis, J., & Corbett, C. (2019). A Survey of Deep Learning Methods for Cyber Security. *Information*, 10(4), 122.
<https://doi.org/10.3390/info10040122>
- Dhal, P., & Azad, C. (2022). A comprehensive survey on feature selection in the various fields of machine learning. *Applied Intelligence*, 52(4), 4543–4581.
<https://doi.org/10.1007/s10489-021-02550-9>
- Fan, Z., Sohail, S., Sabrina, F., & Gu, X. (2024). Sampling-Based Machine Learning Models for Intrusion Detection in Imbalanced Dataset. *Electronics*, 13(10), 1878.
<https://doi.org/10.3390/electronics13101878>
- Faysal, J. A., Mostafa, S. T., Tamanna, J. S., Mumenin, K. M., Arifin, Md. M., Awal, Md. A., Shome, A., & Mostafa, S. S. (2022). XGB-RF: A Hybrid Machine Learning Approach for IoT Intrusion Detection. *Telecom*, 3(1), 52–69.
<https://doi.org/10.3390/telecom3010003>
- Gharib, A., Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2016). An Evaluation Framework for Intrusion Detection Dataset. 2016 International Conference on Information Science and Security (ICISS), 1–6.
<https://doi.org/10.1109/ICISSEC.2016.7885840>
- Gupta, B. B., Perez, G. M., Agrawal, D. P., & Gupta, D. (Eds.). (2020). *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*. Springer International Publishing. <https://doi.org/10.1007/978-3-030-22277-2>
- Gupta, A. R. bhai, & Agrawal, J. (2020). A Comprehensive Survey on Various Machine Learning Methods used for Intrusion Detection System. 2020 IEEE 9th

- International Conference on Communication Systems and Network Technologies (CSNT), 282–289. <https://doi.org/10.1109/CSNT48778.2020.9115764>
- Habeeb, M. S., & Babu, T. R. (2024). A Two-Phase Feature Selection Technique using Information Gain and XGBoost-RFE for NIDS. *International Journal of Intelligent Systems and Applications in Engineering*.
- Hafeez, M. A., Rashid, M., Tariq, H., Abideen, Z. U., Alotaibi, S. S., & Sinky, M. H. (2021). Performance Improvement of Decision Tree: A Robust Classifier Using Tabu Search Algorithm. *Applied Sciences*, 11(15), 6728. <https://doi.org/10.3390/app11156728>
- Hancer, E., Xue, B., & Zhang, M. (2020). A survey on feature selection approaches for clustering. *Artificial Intelligence Review*, 53(6), 4519–4545. <https://doi.org/10.1007/s10462-019-09800-w>
- Humayun, M., Niazi, M., Jhanjhi, N., Alshayeb, M., & Mahmood, S. (2020). Cyber Security Threats and Vulnerabilities: A Systematic Mapping Study. *Arabian Journal for Science and Engineering*, 45(4), 3171–3189. <https://doi.org/10.1007/s13369-019-04319-2>
- Jabali, V. K. (2017). Taxonomy of Feature selection in Intrusion Detection System. 15.
- Jhaveri, S., Khedkar, I., Kantharia, Y., & Jaswal, S. (2019). Success Prediction using Random Forest, CatBoost, XGBoost and AdaBoost for Kickstarter Campaigns. 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), 1170–1173. <https://doi.org/10.1109/ICCMC.2019.8819828>

- Jing, D., & Chen, H.-B. (2019). SVM Based Network Intrusion Detection for the UNSW-NB15 Dataset. 2019 IEEE 13th International Conference on ASIC (ASICON), 1–4. <https://doi.org/10.1109/ASICON47005.2019.8983598>
- Jose, S., Malathi, D., Reddy, B., & Jayaseeli, D. (2018). A Survey on Anomaly Based Host Intrusion Detection System. *Journal of Physics: Conference Series*, 1000, 012049. <https://doi.org/10.1088/1742-6596/1000/1/012049>
- Kasongo, S. M., & Sun, Y. (2020a). Performance Analysis of Intrusion Detection Systems Using a Feature Selection Method on the UNSW-NB15 Dataset. *Journal of Big Data*, 7(1), 105. <https://doi.org/10.1186/s40537-020-00379-6>
- Kasongo, S. M., & Sun, Y. (2020b). Performance Analysis of Intrusion Detection Systems Using a Feature Selection Method on the UNSW-NB15 Dataset. *Journal of Big Data*, 7(1), 105. <https://doi.org/10.1186/s40537-020-00379-6>
- Khan, N. M., Madhav C, N., Negi, A., & Thaseen, I. S. (2020). Analysis on Improving the Performance of Machine Learning Models Using Feature Selection Technique. In A. Abraham, A. K. Cherukuri, P. Melin, & N. Gandhi (Eds.), *Intelligent Systems Design and Applications* (Vol. 941, pp. 69–77). Springer International Publishing. https://doi.org/10.1007/978-3-030-16660-1_7
- Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity*, 2(1), 20. <https://doi.org/10.1186/s42400-019-0038-7>
- Kou, G., Yang, P., Peng, Y., Xiao, F., Chen, Y., & Alsaadi, F. E. (2020). Evaluation of feature selection methods for text classification with small datasets using multiple

- criteria decision-making methods. *Applied Soft Computing*, 86, 105836.
<https://doi.org/10.1016/j.asoc.2019.105836>
- Krstinić, D., Braović, M., Šerić, L., & Božić-Štulić, D. (2020). Multi-label Classifier Performance Evaluation with Confusion Matrix. *Computer Science & Information Technology*, 01–14. <https://doi.org/10.5121/csit.2020.100801>
- Kumar, V. (2014). Feature Selection: A literature Review. *The Smart Computing Review*, 4(3). <https://doi.org/10.6029/smartcr.2014.03.007>
- Liu, H., & Lang, B. (2019a). Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey. *Applied Sciences*, 9(20), 4396.
<https://doi.org/10.3390/app9204396>
- Liu, H., & Lang, B. (2019b). Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey. *Applied Sciences*, 9(20), Article 20.
<https://doi.org/10.3390/app9204396>
- Mahesh, B. (2018). Machine Learning Algorithms—A Review. 9(1), 7.
- Maseer, Z. K., Yusof, R., Bahaman, N., Mostafa, S. A., & Foozy, C. F. M. (2021). Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset. *IEEE Access*, 9, 22351–22370.
<https://doi.org/10.1109/ACCESS.2021.3056614>
- Mebawondu, J. O., Alowolodu, O. D., Mebawondu, J. O., & Adetunmbi, A. O. (2020). Network intrusion detection system using supervised learning paradigm. *Scientific African*, 9, e00497. <https://doi.org/10.1016/j.sciaf.2020.e00497>

- Nazir, A., & Khan, R. A. (2021). A novel combinatorial optimization based feature selection method for network intrusion detection. *Computers & Security*, 102, 102164. <https://doi.org/10.1016/j.cose.2020.102164>
- Panda, M., Abraham, A., & Patra, M. R. (2012). A Hybrid Intelligent Approach for Network Intrusion Detection. *Procedia Engineering*, 30, 1–9. <https://doi.org/10.1016/j.proeng.2012.01.827>
- Priscilla, C. V., & Prabha, D. P. (2021). A two-phase feature selection technique using mutual information and XGB-RFE for credit card fraud detection. *International Journal of Advanced Technology and Engineering Exploration*, 8(85). <https://doi.org/10.19101/IJATEE.2021.874615>
- Python_for_Data_Analysis.pdf. (n.d.).
- Rajadurai, H., & Gandhi, U. D. (2022). A stacked ensemble learning model for intrusion detection in wireless network. *Neural Computing and Applications*, 34(18), 15387–15395. <https://doi.org/10.1007/s00521-020-04986-5>
- Saranya, T., Sridevi, S., Deisy, C., Chung, T. D., & Khan, M. K. A. A. (2020). Performance Analysis of Machine Learning Algorithms in Intrusion Detection System: A Review. *Procedia Computer Science*, 171, 1251–1260. <https://doi.org/10.1016/j.procs.2020.04.133>
- Sarker, I. H., Kayes, A. S. M., Badsha, S., Alqahtani, H., Watters, P., & Ng, A. (2020). Cybersecurity data science: An overview from machine learning perspective. *Journal of Big Data*, 7(1), 41. <https://doi.org/10.1186/s40537-020-00318-5>
- Sharafaldin, I., Habibi Lashkari, A., & Ghorbani, A. A. (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:

- Proceedings of the 4th International Conference on Information Systems Security and Privacy, 108–116. <https://doi.org/10.5220/0006639801080116>
- Shaukat, K., Luo, S., Varadharajan, V., Hameed, I. A., & Xu, M. (2020). A Survey on Machine Learning Techniques for Cyber Security in the Last Decade. *IEEE Access*, 8, 222310–222354. <https://doi.org/10.1109/ACCESS.2020.3041951>
- Sofiane Maza, & Mohamed Touahria. (2018). Feature Selection Algorithms in Intrusion Detection System: A Survey. *KSII Transactions on Internet and Information Systems*, 12(10). <https://doi.org/10.3837/tiis.2018.10.024>
- Taher, K. A., Mohammed Yasin Jisan, B., & Rahman, Md. M. (2019). Network Intrusion Detection using Supervised Machine Learning Technique with Feature Selection. 2019 International Conference on Robotics,Electrical and Signal Processing Techniques (ICREST), 643–646. <https://doi.org/10.1109/ICREST.2019.8644161>
- Thakkar, A., & Lohiya, R. (2021). Attack classification using feature selection techniques: A comparative study. *Journal of Ambient Intelligence and Humanized Computing*, 12(1), 1249–1266. <https://doi.org/10.1007/s12652-020-02167-9>
- Thakkar, A., & Lohiya, R. (2022). A survey on intrusion detection system: Feature selection, model, performance measures, application perspective, challenges, and future research directions. *Artificial Intelligence Review*, 55(1), 453–563. <https://doi.org/10.1007/s10462-021-10037-9>
- Umar, M. A., Chen, Z., & Liu, Y. (2021). A Hybrid Intrusion Detection with Decision Tree for Feature Selection. *Information & Security: An International Journal*. <https://doi.org/10.11610/isij.4901>

Umar, M. A., Zhanfang, C., & Liu, Y. (n.d.). A Hybrid Intrusion Detection with Decision Tree for Feature Selection. 16.

Valenzuela, O., Rojas, I., Herrera, L. J., Guillén, A., Rojas, F., Marquez, L., & Pasadas, M. (n.d.). FEATURE SELECTION USING MUTUAL INFORMATION AND NEURAL NETWORKS.

Venkatesh, B., & Anuradha, J. (2019). A Review of Feature Selection and Its Methods. *Cybernetics and Information Technologies*, 19(1), 3–26.
<https://doi.org/10.2478/cait-2019-0001>

Yin, Y., Jang-Jaccard, J., Xu, W., Singh, A., Zhu, J., Sabrina, F., & Kwak, J. (2023). IGRF-RFE: A hybrid feature selection method for MLP-based network intrusion detection on UNSW-NB15 dataset. *Journal of Big Data*, 10(1), 15.
<https://doi.org/10.1186/s40537-023-00694-8>