NETWORK INTRUSION DETECTION SYSTEM USING MACHINE LEARNING

Individual project by,

Koushik Kumar Sarkar

ABSTRACT

Classification involves identifying, interpreting, and organizing concepts or objects into predefined categories. It relies on patterns from training data to estimate the likelihood that new, unseen data belongs to one of these categories. Various algorithms are employed to perform this classification based on the patterns learned.

In today's digital landscape, diverse forms of network attacks continue to emerge. This study focuses on analyzing the NSL-KDD dataset using Python and multiple machine learning algorithms. The dataset undergoes data preprocessing before applying models such as Decision Trees, Random Forests, Naïve Bayes, K-Nearest Neighbors (KNN), Gradient Boosted Trees, and Support Vector Machines (SVM).

We evaluate each model using metrics like the confusion matrix, accuracy (ACC), and ROC-AUC curves to assess performance. The experimental results demonstrate that, after proper data preprocessing, machine learning techniques can achieve remarkably high accuracy in detecting network intrusions.

CONTENTS

ABSTRACT ii

CONTENTS iii

LIST OF FIGURES vii

LIST OF TABLES viii

1 Introduction 1

1.1 Overview 1

1.2 About the Project 2

1.3 Motivation 2

1.3.1 Key Motivation 3

1.4 Challenges 4

1.5 Problem Definition 5

1.6 Aim 5

1.7 Objectives 5

2 Literature Review 7

2.1 Introduction 7

2.2 Overview on Intrusion Detection Systems Design Exploiting Machine Learning for Networking Cyber Security 7

2.3 An Improved Binary Manta Ray Foraging Optimization Algorithm Based Feature Selection and Random Forest Classifier for Network Intrusion Detection 8

2.4 Comparative Research on Network Intrusion Detection Methods Based on Machine Learning 8

2.5 Network Intrusion Detection System using Classification Techniques in Machine Learning 9

2.6 Intrusion Detection Systems using Supervised Machine Learning Techniques: A Survey 10

2.6.1 Summary 11

3 METHODOLOGIES 12

3.1 Introduction 12

3.2 About Dataset NSL-KDD 2009 12

3.3 System Architecture 16

3.4 One Hot Encoding 17

3.5 Random Forest Classifier 18

3.5.1 Feature Selection 20

3.6 K-nearest Neighbors 22

3.7 Support Vector Machine 23

3.8 Ensemble Learning Voting Classifier 25

3.9 Gaussian Naive Bayes Classifier 26

3.10 Principal Component Analysis 29

3.11 XGBoost Classifier 33

3.12 Summary 35

4 Results and Discussions 36

4.1 Results 36

4.1.1 Introduction 36

4.1.2 Random Forest Classifier 36

4.1.3 K-Nearest Neighbors 39

4.1.4 Support Vector Machine 42

4.1.5 Ensemble Learning 45

4.1.6 Gaussian Naive Bayes Classifier 48

4.1.7 Principal Component Analysis 51

4.1.8 XGBoost Classifier 54

4.2 Discussion 57

4.2.1 Performance Comparision 57

4.3 Summary 61

5 Conclusion 62

5.1 Summary of the Study 62

5.2 Main Findings 62

5.3 Contributions to Knowledge.....62
5.4 Practical Implications..... 63
5.5 Final Thoughts.....63

BIBLIOGRAPHY **64**

LIST OF FIGURES

1.1	Network Intrusion Detection System.	1
3.1	Network Intrusion Detection System Architecture	16
3.2	Bar Graph: One Hot Encoding	18
3.3	Flow chart of Random Forest Classifier	19
3.4	Features and their importance in KDD Dataset	21
3.5	An illustration of K nearest neighbor	22
3.6	An illustration of Support Vector Machine	23
3.7	An illustration of Ensemble Learning Voting Classifier	25
3.8	An illustration of Gaussian Naive Bayes Classifier	27
3.9	An illustration of Principal Component Analysis	30
3.10	An illustration of XGBoost Classifier	33
4.1	Bar graph: Random Forest classifier DOS metrics	36
4.2	Bar graph: Random Forest classifier Probe metrics	37
4.3	Bar graph: Random Forest classifier U2R metrics	37
4.4	Bar graph: Random Forest classifier R2L metrics	38
4.5	Bar graph: K-Nearest Neighbors DOS metrics	39
4.6	Bar graph: K-Nearest Neighbors Probe metrics	40
4.7	Bar graph: K-Nearest Neighbors U2R metrics	40
4.8	Bar graph: K-Nearest Neighbors R2L metrics	41
4.9	Bar graph: Support Vector Machine DOS metrics	42
4.10	Bar graph: Support Vector Machine Probe metrics	43
4.11	Bar graph: Support Vector Machine U2R metrics	43
4.12	Bar graph: Support Vector Machine R2L metrics	44
4.13	Bar graph: Ensemble Learning voting Classifier DOS metrics	45
4.14	Bar graph: Ensemble Learning voting Classifier Probe metrics	46
4.15	Bar graph: Ensemble Learning voting Classifier U2R metrics	46
4.16	Bar graph: Ensemble Learning voting Classifier R2L metrics	47
4.17	Bar graph: Gaussian Naive Bayes Classifier DOS metrics	48

4.18 Bar graph: Gaussian Naive Bayes Classifier Probe metrics 49
4.19 Bar graph: Gaussian Naive Bayes Classifier U2R metrics 49
4.20 Bar graph: Gaussian Naive Bayes Classifier R2L metrics 50
4.21 Bar graph: Principal Component Analysis DOS metrics 51
4.22 Bar graph: Principal Component Analysis Probe metrics 52
4.23 Bar graph: Principal Component Analysis U2R metrics 52
4.24 Bar graph: Principal Component Analysis R2L metrics 53
4.25 Bar graph: XGBoost Classifier DOS metrics 54
4.26 Bar graph: XGBoost Classifier Probe metrics 55
4.27 Bar graph: XGBoost Classifier U2R metrics 55
4.28 Bar graph: XGBoost Classifier R2L metrics 56
4.29 Confusion Matrix: Random Forest classifier 58
4.30 Confusion Matrix: K-Nearest Neighbors 59
4.31 Confusion Matrix: Support Vector Machine 59
4.32 Confusion Matrix: Gaussian Naive Bayes Classifier 60
4.33 Confusion Matrix: XGBoost Classifier 60

LIST OF TABLES

3.1	Attribute Explanation	16
4.1	Performance Metrics of Classifiers	58
4.2	Comparison of Models' Performance	58

Chapter 1

INTRODUCTION

1.1 Overview

In today's interconnected digital landscape, the security of computer networks is of paramount importance. With the increasing sophistication of cyber threats, organizations face a con- stant challenge in safeguarding their networks from unauthorized access, malicious activi- ties, and data breaches. One crucial component of a comprehensive cybersecurity strategy is the Network Intrusion Detection System (NIDS). A Network Intrusion Detection System (NIDS) is a vital security tool designed to monitor network traffic in real-time, identifying and alerting administrators to suspicious or potentially harmful activities. By analyzing network packets and patterns, NIDS can detect various types of attacks, including malware infections, unauthorized access attempts, denial-of-service (DoS) attacks, and more. This mini project report aims to explore the fundamentals of Network Intrusion Detection Sys- tems, including their functionalities, architecture, detection techniques, and deployment strategies. Additionally, it will discuss the importance of NIDS in enhancing network security posture, mitigating risks, and safeguarding critical assets against cyber threats.

Through this project, we seek to gain a deeper understanding of NIDS and its role in mod- ern cybersecurity practices, providing insights into its implementation, configuration, and effectiveness in protecting network infrastructures from emerging threats.

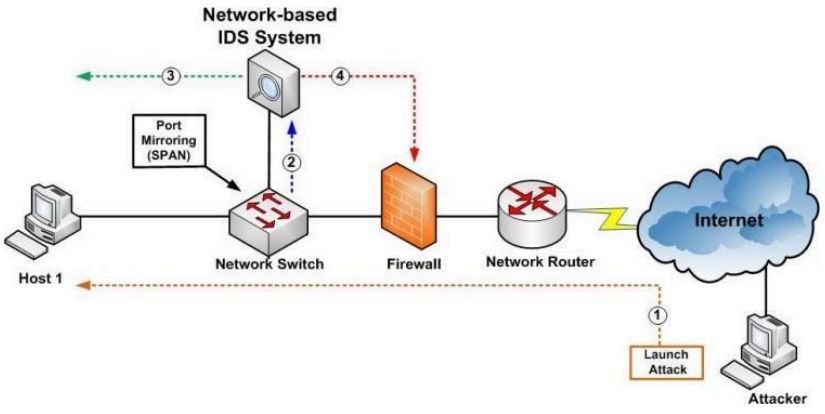


Figure 1.1: Network Intrusion Detection System.

The architecture of a typical NIDS consists of sensors strategically deployed through- out the network infrastructure to capture and analyze traffic. These sensors can be hardware- based appliances, software applications running on dedicated servers, or even virtual ap- pliances deployed in cloud environments. Central to the NIDS architecture is the analysis engine, responsible for processing and correlating network data collected by the sensors, identifying suspicious patterns, and generating alerts for further investigation. NIDS oper- ates by inspecting incoming and outgoing network traffic, comparing it against pre-defined signatures or behavioral patterns indicative of known attacks or anomalies. Signature- based detection involves matching network traffic against a database of predefined attack patterns or signatures, allowing NIDS to recognize and respond to known threats effec- tively. On the other hand, anomaly-based detection relies on establishing a baseline of normal network behavior and flagging deviations from this baseline as potential intru- sions. Additionally, some NIDS solutions utilize a combination of signature-based and anomaly-based detection techniques for enhanced accuracy and coverage.

1.2 About the Project

The ”Network Intrusion Detection Using ML” project focuses on developing an advanced intrusion detection system by harnessing machine learning (ML) algorithms and method- ologies. The primary goal is to design ML-based models capable of detecting and mitigat- ing various types of network intrusions, such as unauthorized access, malware infections, data exfiltration, and insider threats. By training these models on labeled datasets contain- ing normal and malicious network traffic, the project aims to enhance network security by leveraging ML’s ability to identify sophisticated attacks that traditional methods may miss. The initiative seeks to address the challenges posed by the evolving cyber threat landscape and bolster the resilience of organizational networks against emerging security risks.

1.3 Motivation

The motivation behind the ”Network Intrusion Detection Using ML” project is driven by the exponential growth of cyber threats and attacks, which have posed significant

chal- lenges to the security of computer networks. Traditional network intrusion detection sys-

tems (IDS) based on rule-based approaches often struggle to keep pace with the ever-evolving attack landscape. This necessitates the exploration of innovative solutions to enhance network security.

1.3.1 Key Motivation

- **Limitations of Traditional IDS:**
  - Rule-based IDS systems can be rigid and struggle to adapt to new, sophisticated attack vectors.
  - Keeping up with the rapid evolution of cyber threats requires more flexible and adaptive approaches.
- **Harnessing the Power of Machine Learning:**
  - Machine learning techniques have demonstrated the ability to discern complex patterns and anomalies within large datasets.
  - ML-based models can potentially identify sophisticated attacks that traditional methods may overlook.
- **Enhancing Real-Time Threat Detection:**
  - Timely detection and mitigation of network intrusions are crucial for maintaining the integrity and availability of organizational networks.
  - Leveraging ML can enable real-time analysis and response to emerging secu- rity threats.
- **Improving Network Resilience:**
  - Strengthening network security is essential to ensure the resilience of critical infrastructure and safeguard sensitive data.
  - Developing an advanced, ML-powered intrusion detection system can contribute to the overall security posture of organizations.

By harnessing the capabilities of machine learning, the Network Intrusion Detection Using ML project aims to address the growing challenges in network security and bolster the resilience of organizational networks against evolving cyber threats.

1.4 Challenges

Based on the challenges outlined in the provided sources, here is a general arrangement based on their potential impact and prevalence:

1. **Encrypted Traffic:** The increasing use of encryption protocols poses a significant challenge for intrusion detection systems as they struggle to inspect encrypted traffic effectively.

2. **Zero-Day Attacks:** Zero-day attacks exploit unknown vulnerabilities without iden- tifiable signatures, making them difficult to detect and defend against using tradi- tional methods.

3. **False Positives and Negatives:** Balancing detection sensitivity to minimize false alerts while accurately identifying malicious activity is crucial for the effectiveness of network intrusion detection systems.

4. **Evasion Techniques:** Cyber attackers develop evasion techniques to bypass de- tection, necessitating continuous adaptation and improvement of detection mecha- nisms.

5. **High Volume of Network Traffic:** Managing the large volume of network traffic and processing it in real-time without compromising detection accuracy is a funda- mental challenge for intrusion detection systems.

6. **Network Complexity:** The increasing complexity of network environments, includ- ing diverse assets and architectures, complicates the deployment and management of intrusion detection systems.

7. **Resource Constraints:** Deploying intrusion detection systems in resource-constrained environments, like edge networks or IoT devices, presents challenges related to per- formance optimization and resource utilization.

8. **Privacy Concerns:** Balancing effective intrusion detection with user privacy rights is essential to ensure that sensitive information is not compromised during monitor- ing and analysis.

This arrangement highlights the diverse challenges faced in implementing machine learning for network security and underscores the importance of addressing these issues to enhance the effectiveness of intrusion detection systems.

1.5 Problem Definition

The ”Network Intrusion Detection Using ML” project addresses the challenge of network security amid increasing cyber threats. Traditional rule-based intrusion detection systems struggle to keep pace with evolving attacks, prompting the exploration of advanced ap- proaches. By leveraging machine learning techniques, the project aims to develop an en- hanced intrusion detection system capable of identifying various network attacks in real- time. ML algorithms analyze large datasets of network traffic to detect complex patterns and anomalies, improving detection accuracy. This project is crucial as organizations rely more on interconnected systems. Integrating ML into network security practices promises to strengthen defenses against evolving threats, contributing to the security of critical net- work infrastructures.

1.6 Aim

The aim of the ”Network Intrusion Detection Using ML” project is to develop an advanced intrusion detection system that leverages machine learning techniques to effectively iden- tify and mitigate various types of network attacks in real-time, thereby enhancing the se- curity and resilience of organizational network infrastructures.

1.7 Objectives

1. **Comprehensive Datasets:** IDSs should leverage up-to-date and comprehensive datasets for accurate evaluation, avoiding outdated or limited datasets that may lead to inaccurate results and false confidence in attack detection.

2. **False Positive Reduction:** Efforts should be made to minimize false positives gen- erated by IDSs, as a high false positive rate can lead to alert fatigue and hinder the ability of system administrators to respond effectively to actual attacks.

3. **Zero-Day Attack Detection:** IDSs should incorporate advanced techniques, such as behavior-based anomaly detection and machine learning, to detect and respond to emerging and previously unseen attack patterns that do not have pre-defined sig- natures or rules.

4. **Timely Detection and Response:** Improve IDSs for prompt identification and mit- igation of network intrusions, addressing challenges like false positives and latency issues. Enhance algorithms, optimize data processing, and implement efficient de- tection mechanisms for timely threat response.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

The literature review aims to provide a comprehensive overview of existing research related to the application of machine learning techniques in network intrusion detection sys- tems (NIDS). This review synthesizes key findings, methodologies, and debates from a range of academic sources, including peer-reviewed articles, books, and conference pro- ceedings.

2.2 Overview on Intrusion Detection Systems Design Exploiting Machine Learning for Networking Cybersecurity

Authors: Dini, P., Elhanashi, A., Begni, A., Saponara, S., Zheng, Q., & Gasmi, K. (2023)

The proposed system outlined in this paper represents a significant advancement in the domain of data traffic security within network systems, particularly in the context of intrusion detection. What distinguishes this research is its exhaustive evaluation of various machine learning models, which surpasses the limitations of prior studies that often focus on a limited selection. By critically assessing the strengths and weaknesses of numerous algorithms, the study offers a nuanced understanding of their effectiveness in enhancing security measures. Moreover, the inclusion of models suitable for both binary and multi- class classification scenarios broadens the applicability of the findings, catering to diverse operational requirements. The research also delves into the impact of dataset selection on intrusion detection system performance, shedding light on crucial factors influencing model efficacy. Additionally, by advancing feature engineering techniques and meticu- lously analyzing computational time requirements for different models, the study provides a comprehensive performance evaluation framework. Overall, this systematic approach contributes significantly to the field by offering practical insights into optimizing data traf- fic security in network systems through the judicious selection and deployment of machine

learning algorithms.

2.3 An Improved Binary Manta Ray Foraging Optimization Algorithm Based Fea- ture Selection and Random Forest Classifier for Network Intrusion Detection

Authors: Hassan, I., Abdullahi, M., & Masama, M. (2022)

The proposed system discussed in the literature underscores the pivotal role of intrusion detection systems (IDS) in fortifying network security measures. It delves into the intricate nuances of IDS methodologies, emphasizing the distinction between misuse and anomaly detection approaches. Key challenges such as feature selection and classification are meticulously addressed through the application of optimization techniques and machine learning algorithms. Noteworthy contributions include the introduction of novel methodologies like the Binary Manta Ray Foraging optimization algorithm and the uti- lization of Random Forest classifiers. These innovative strategies are specifically tailored to augment IDS effectiveness in the face of evolving cyber threats, underscoring the dy- namic nature of research endeavors aimed at safeguarding network infrastructure. The literature's comprehensive exploration of IDS functionalities, coupled with its innovative solutions to address inherent challenges, signifies a concerted effort towards bolstering network security frameworks and mitigating potential vulnerabilities.

2.4 Comparative Research on Network Intrusion Detection Methods Based on Ma- chine Learning

Authors: Zhang, C., Jia, D., Wang, L., Wang, W., Liu, F., & Yang, A. (2022)

The paper underlines the critical role of network intrusion detection systems (IDS) in mod- ern cybersecurity and investigates the potential of machine learning algorithms to enhance their efficacy and accuracy. Recognizing the diverse strengths and limitations of these algorithms, the study systematically reviews the literature on machine learning applica- tions in intrusion detection over the past decade. It categorizes the analysis into three sec- tions: traditional machine learning, ensemble learning, and deep learning. Furthermore, the study conducts comparative experiments using datasets like KDD

CUP99 and NSL- KDD, assessing algorithms such as decision trees, Naive Bayes, support vector machines,

random forests, XGBoost, convolutional neural networks, and recurrent neural networks across various metrics including detection accuracy, F1 score, and AUC.

The findings highlight the generally superior performance of ensemble learning algo- rithms while also revealing the nuanced advantages of individual algorithms like Naive Bayes in handling novel attacks. Additionally, the study outlines key challenges faced by the field of network intrusion detection and suggests prospective directions for future re- search. This literature survey offers a comprehensive overview of the proposed framework for intrusion detection on big and imbalanced datasets, covering traditional, ensemble, and deep learning approaches over the past decade. The robust evaluation of detection perfor- mance adds credibility to the findings and aids informed algorithm selection.

By providing a comparative analysis focusing not only on performance metrics but also on the strengths and weaknesses of each algorithm, the study offers valuable insights for practitioners and researchers. Acknowledging these challenges is crucial for guiding future research endeavors and technological advancements in the field. Moreover, by out- lining specific areas for further investigation, such as dataset preprocessing techniques and optimization of deep learning models, the study offers clear guidance for future research directions, fostering ongoing innovation and progress in network intrusion detection.

2.5 Network Intrusion Detection System using Classification Techniques in Machine Learning

Author: Chudasma, P. (2020)

The paper underlines the critical role of network intrusion detection systems (IDS) in mod- ern cybersecurity and investigates the potential of machine learning algorithms to enhance their efficacy and accuracy. Recognizing the diverse strengths and limitations of these algorithms, the study systematically reviews the literature on machine learning applica- tions in intrusion detection over the past decade. It categorizes the analysis into three sec- tions: traditional machine learning, ensemble learning, and deep learning. Furthermore, the study conducts comparative experiments using datasets like KDD CUP99 and NSL- KDD, assessing algorithms such as decision trees, Naive Bayes, support vector machines, random forests, XGBoost, convolutional neural networks, and

recurrent neural networks across various metrics including detection accuracy, F1 score, and AUC.

The findings highlight the generally superior performance of ensemble learning algo- rithms while also revealing the nuanced advantages of individual algorithms like Naive Bayes in handling novel attacks. Additionally, the study outlines key challenges faced by the field of network intrusion detection and suggests prospective directions for future re- search. This literature survey offers a comprehensive overview of the proposed framework for intrusion detection on big and imbalanced datasets, covering traditional, ensemble, and deep learning approaches over the past decade. The robust evaluation of detection perfor- mance adds credibility to the findings and aids informed algorithm selection.

By providing a comparative analysis focusing not only on performance metrics but also on the strengths and weaknesses of each algorithm, the study offers valuable insights for practitioners and researchers. Acknowledging these challenges is crucial for guiding future research endeavors and technological advancements in the field. Moreover, by out- lining specific areas for further investigation, such as dataset preprocessing techniques and optimization of deep learning models, the study offers clear guidance for future research directions, fostering ongoing innovation and progress in network intrusion detection.

2.6 Intrusion Detection Systems using Supervised Machine Learning Techniques: A Survey

Authors: Abdallah, E. E., Eleisah, W., & Otoom, A. F. (2020)

The paper presents a thorough survey on the utilization of supervised machine learning al- gorithms within intrusion detection systems (IDS), underscoring their crucial role in iden- tifying and mitigating cyber-attacks. It specifically emphasizes the significance of IDS in detecting emerging threats such as zero-day attacks, primarily through anomaly-based detection methods. The discussion revolves around the efficacy of supervised machine learning in efficiently recognizing novel attacks, with a particular focus on the optimiza- tion of feature selection techniques to enhance classification performance. By reviewing prominent datasets in IDS research, including KDD’99, NSL-KDD, UNSW-NB15, and CICIDS2017, alongside an array of supervised learning algorithms like random forest, SVM, logistic regression, and ANN, the paper provides valuable insights into the land- scape of intrusion detection. Drawing from insights

garnered from related works, the sur- vey delineates the effectiveness of various algorithms and feature selection methodologies

in bolstering classification accuracy. Additionally, the proposal of a taxonomy for categorizing IDS and supervised machine learning algorithms contributes to the structured understanding and comparison of methodologies within the IDS research domain.

The comprehensive nature of the survey, coupled with its meticulous examination of key challenges and solutions in intrusion detection, renders it an invaluable resource for both researchers and practitioners in the cybersecurity domain. The incorporation of real- world datasets and performance evaluations from existing literature enhances the credibil- ity and practical applicability of the survey's findings. Furthermore, the proposed taxon- omy serves as a valuable tool for organizing the vast landscape of IDS research, facilitating better comprehension and comparison of different approaches. Overall, the paper signifi- cantly advances knowledge in cybersecurity by synthesizing existing research efforts and providing actionable insights into effective methodologies for detecting and mitigating cyber threats. It stands as a cornerstone resource guiding future research endeavors and practical implementations in IDS leveraging supervised machine learning algorithms.

2.6.1 Summary

In delving into the literature surrounding machine learning applications in network intrusion detection systems (NIDS), I encountered five insightful scholarly works. Through meticulous analysis, these studies collectively offered a panoramic view of methodologies, challenges, and innovations within the field. They scrutinized various machine learning models, introducing novel algorithms, and conducting rigorous comparative experiments. By meticulously evaluating performance metrics and delving into the nuances of super- vised learning approaches, these works provided invaluable insights. Noteworthy contri- butions included the optimization of feature selection techniques, the introduction of in- novative optimization algorithms, and the systematic categorization of intrusion detection methodologies. This journey through the literature has not only deepened my understand- ing of cybersecurity but also provided clear guidance for future research and practical implementations in NIDS leveraging machine learning algorithms.

Chapter 3

METHODOLOGIES

3.1 Introduction

The methodology for Network Intrusion Detection Systems (NIDS) using machine learn- ing involves a systematic approach. It begins with feature extraction, where relevant at- tributes from raw network data, such as IP addresses, ports, and protocols, are selected and transformed into input variables for the model. The model is trained using the NSL-KDD dataset, which provides labeled instances of normal and attack traffic. This training pro- cess includes selecting an appropriate classifier and optimizing its performance through hyperparameter tuning. A processing pipeline then handles captured NetFlow data, ex- tracting features and feeding them into the trained model for real-time predictions. The results are displayed on a user-friendly web interface, providing insights and classifying traffic as normal or malicious, with the model hosted on platforms like Google Colab for accessibility and collaboration.

3.2 About Dataset NSL-KDD 2009

The NSL-KDD data set is not the first of its kind. The KDD cup was an International Knowledge Discovery and Data Mining Tools Competition. In 1999, this competition was held with the goal of collecting traffic records. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between “bad’’ connections, called intrusions or attacks, and “good’’ normal connections. As a result of this competition, a mass amount of internet traffic records were collected and bundled into a data set called the KDD’99, and from this, the NSL-KDD data set was brought into existence, as a revised, cleaned-up version of the KDD’99 from the University of New Brunswick.

The NSL-KDD dataset is an improved version of the original KDD Cup 1999 dataset, widely used for evaluating intrusion detection systems. It addresses some of the inherent issues in the KDD’99 dataset, such as redundancy and class imbalance, making it more

suitable for research and development purposes. The NSL-KDD dataset eliminates redundant records, which helps in providing a more balanced and fair evaluation of different machine learning models. Additionally, it is smaller in size compared to KDD'99, making it more manageable and less resource-intensive to work with, while still representing the original dataset's characteristics.

The dataset includes four main types of network attacks: DoS (Denial of Service), which overwhelms the victim with a flood of requests, causing service disruption; Probe, which involves surveillance and information gathering activities to identify vulnerabilities in the network; R2L (Remote to Local), which refers to unauthorized access from a remote machine to a local machine; and U2R (User to Root), which refers to unauthorized access to root-level permissions from a local machine. The dataset consists of 41 features, includ- ing basic features derived from individual TCP connections, content features that suggest suspicious behavior within a connection, and traffic features computed using a two-second time window to capture behavior in the network traffic.

The NSL-KDD dataset is divided into training and testing subsets, allowing for robust evaluation of models. The testing set contains new attack types not present in the training set, simulating real-world scenarios. Each record is labeled as either normal or one of the attack types.

The NSL-KDD dataset is extensively used in research for developing and benchmark- ing intrusion detection systems (IDS). It provides a common ground for comparing dif- ferent methodologies, including traditional machine learning techniques, deep learning models, and hybrid approaches. Its balanced distribution, reduced size, and inclusion of new attack types in the testing set make it crucial for training models on known data and evaluating their ability to generalize to new, unseen attack patterns. Additionally, it enables the analysis of feature importance and the development of feature selection techniques for improving IDS performance. In summary, the NSL-KDD dataset is a refined version of the original KDD'99 dataset, designed to be more balanced and practical for evaluating intrusion detection systems, making it an essential tool in cybersecurity research. The below table provides an Overview of attributes

Attribute	Explanation

duration	Length (in seconds) of the connection.
protocol_type	Type of protocol, such as TCP, UDP, or ICMP.
service	Network service on the destination, e.g., http, telnet, ftp,
flag	etc. Status flag of the connection, indicating whether the connec- tion was successful, had errors, etc.
src_byte	Number of data bytes from source to destination.
s	Number of data bytes from destination to source.
dst_byte	1 if connection is from/to the same host/port (land attack), 0
s land	otherwise.
	Number of wrong fragments in this connection.
wrong_fragment	Number of urgent packets in this connection.
urgent	Number of ”hot” indicators, such as number of commands
hot	in the ftp session, number of operations that access system files, etc.
	Number of failed login attempts.
num_failed_logins	1 if successfully logged in, 0 otherwise.
logged_in	Number of compromised conditions.
num_compromised	1 if root shell is obtained, 0 otherwise.
root_shell	1 if ”su root” command attempted, 0 otherwise.
su_attempted	Number of ”root” accesses or attempts to access
num_root	”root”. Number of file creation operations.
num_file_creations	Number of shell prompts invoked.
num_shells	Number of operations on access control files.
num_access_files	Number of outbound commands in an ftp session.
num_outbound_cmd	1 if the login belongs to the ”host” list, 0
s is_host_login	otherwise. 1 if the login is a ”guest” login, 0
is_guest_login count	otherwise.
	Number of connections to the same host as the current con-
srv_count	nection in the past two seconds.
	Number of connections to the same service as the current connection in the past two seconds.

Attribute	Explanation
serror_rate	% of connections that have ”SYN” errors.
srv_serror_rate	% of connections that have ”SYN” errors for the same service.
rerror_rate	% of connections that have ”REJ” errors.
srv_rerror_rate	% of connections that have ”REJ” errors for the same service.
same_srv_rate	% of connections to the same service.
diff_srv_rate	% of connections to different services.
srv_diff_host_rat	% of connections to different hosts.
e dst_host_count	Number of connections to the same destination host as the current connection in the past two seconds.
dst_host_srv_count	Number of connections to the same service on the destination host as the current connection in the past two seconds.
dst_host_same_srv_rate	% of connections to the same service on the destination host.
dst_host_diff_srv_rate	% of connections to different services on the destination host.
dst_host_same_src_port_rate	% of connections to the same source port.
dst_host_srv_diff_host_rate	% of connections to different destination hosts for the same service.
dst_host_serror_rate	% of connections that have ”SYN” errors on the destination host.
dst_host_srv_serror_rate	% of connections that have ”SYN” errors for the same service on the destination host.
dst_host_rerror_rate	% of connections that have ”REJ” errors on the destination host.
dst_host_srv_rerror_rate	% of connections that have ”REJ” errors for the same service

	on the destination host.
label	The class label, indicating whether the connection is normal or an attack (e.g., ”normal”, ”anomaly”, ”dos”, ”probe”, etc.).

Table 3.1: Attribute Explanation

3.3 System Architecture

In the data collection and preprocessing phase, relevant network traffic data is gathered from multiple sources, including routers, switches, and network appliances.

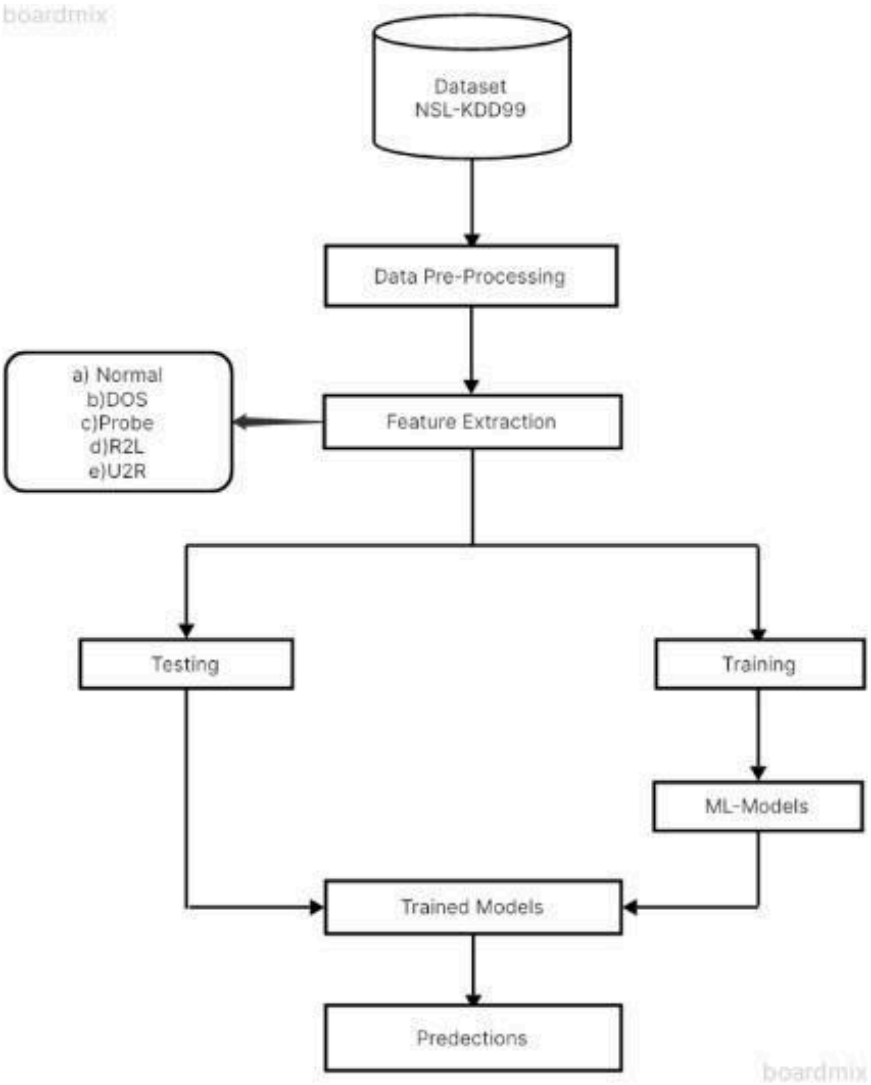


Figure 3.1: Network Intrusion Detection System Architecture

This raw data undergoes preprocessing steps such as cleaning to eliminate duplicates and handling missing values. Additionally, feature extraction techniques are applied to

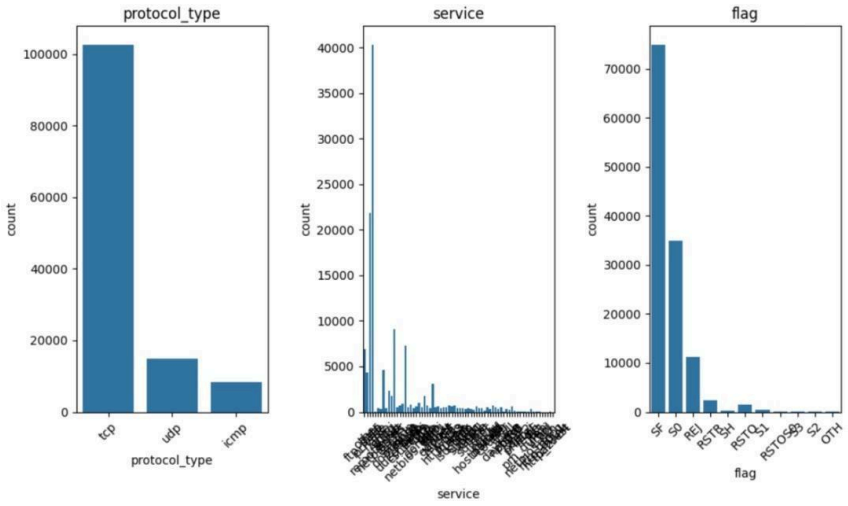
transform the raw network data into meaningful features suitable for model training. The subsequent data splitting stage involves dividing the preprocessed data into training, val- idation, and test sets. This division facilitates model training and evaluation, ensuring robust performance assessment. Multiple machine learning models are selected for intru- sion detection in the model selection and training phase. These models include K-Nearest Neighbors (KNN), Decision Trees, Naive Bayes, XGBoost, Random Forest, Support Vec- tor Machines (SVM), and Principal Component Analysis (PCA). Each model is trained us- ing the training data, with hyperparameters optimized through techniques like grid search or randomized search. Following training, the trained models are evaluated using the val- idation set to assess their performance. Evaluation metrics such as accuracy, precision, recall, and F1-score are calculated for each model, providing insights into their effective- ness in detecting network intrusions. Ensemble techniques, such as bagging and boosting, may be employed to combine the predictions of multiple base models, further enhancing overall performance and robustness. Once the best-performing model is identified, it is deployed to the production environment. APIs or endpoints are set up to enable real-time prediction of network intrusions using the deployed model. In the inference and prediction phase, new network traffic data is fed into the deployed model for prediction. The model predicts whether incoming network traffic is normal or indicative of an intrusion, based on learned patterns from the training data. Finally, the deployed model's performance is continuously monitored in the production environment. Regular maintenance activi- ties, including retraining with new data and updating model parameters, are conducted to ensure the system's effectiveness over time. This comprehensive system architecture inte- grates various machine learning models to enhance network intrusion detection, providing robust security measures for safeguarding network infrastructures.

3.4 One Hot Encoding

One hot encoding is a technique used to convert categorical variables into a numerical format that can be utilized by machine learning algorithms. This method transforms each unique category value within a feature into a new binary feature (column). Each new col- umn represents one of the possible categories and contains a 1 if the observation matches that category and a 0 otherwise. This approach allows machine

learning algorithms to

process categorical data correctly without assuming any ordinal relationship among the categories



ıcoding The implementation involves first identifying the categorical columns in the DataFrame df and printing the number of unique categories for each. This is done by iterating through all columns and checking if their data type is object. For each categorical column, the number of unique categories is printed. Additionally, the distribution of categories within the service column is displayed to provide insight into the most common service cat- egories. Although the actual one hot encoding step is not shown in the provided code snippet, it can be performed using the pandas get_dummies function. This function will create new binary columns for each unique category in the service column, converting the categorical data into a format suitable for machine learning models.

3.5 Random Forest Classifier

Random Forest algorithm is a powerful tree-learning technique in Machine Learning. It works by creating several Decision Trees during the training phase. Each tree is con- structed using a random subset of the data set to measure a random subset of features in each partition. This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance. In prediction, the algorithm aggregates the results of all trees, either by voting (for classification tasks) or by averaging (for regression tasks) This collaborative decision-making process, supported by

multiple trees with their insights, provides an example of stable and precise results. Ran- dom forests are widely used for classification and regression functions, which are known for their ability to handle complex data, reduce overfitting, and provide reliable forecasts in different environments.

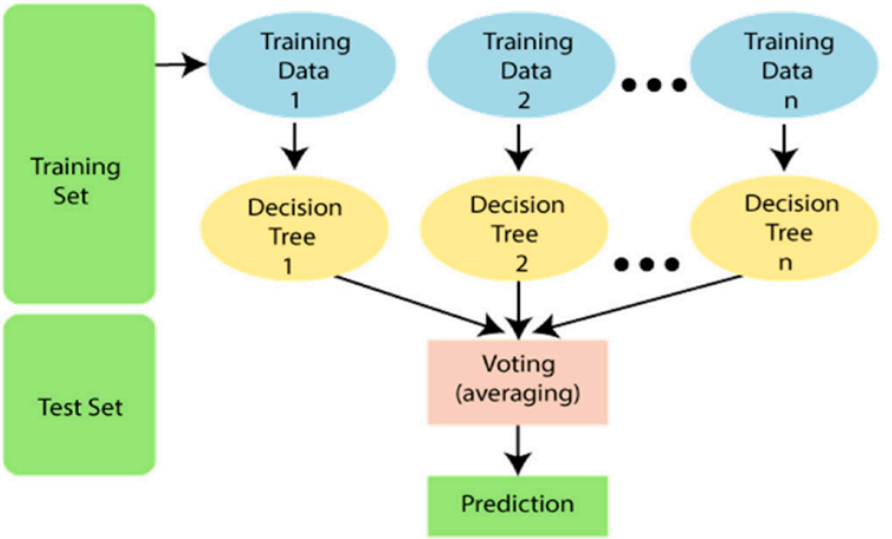


Figure 3.3: Flow chart of Random Forest Classifier

The code trains Random Forest classifiers on four datasets—DoS, Probe, R2L, and U2R—using both the full set of features and a reduced set of features selected by Recursive Feature Elimination (RFE). For the full feature sets, separate RandomForestClassifier instances (clf_DoS, clf_Probe, clf_R2L, clf_U2R) are initialized with 10 estimators and 2 parallel jobs, and then fitted to their respective datasets (X_DoS, Y_DoS, X_Probe, Y_Probe, X_R2L, Y_R2L, X_U2R, Y_U2R). Similarly, for the selected feature sets, corre- sponding classifiers (clf_rfeDoS, clf_rfeProbe, clf_rfeR2L, clf_rfeU2R) are ini- tialized and fitted to the transformed datasets (X_rfeDoS, Y_DoS, X_rfeProbe, Y_Probe, X_rfeR2L, Y_R2L, X_rfeU2R, Y_U2R). This approach allows for a comparison between the performance of classifiers trained on all features versus those trained on the selected important features.

The formulas for Gini impurity and Entropy

Gini Impurity:

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Entropy:

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

Advantages of Random Forest Classifier

- It reduces overfitting in decision trees and helps to improve accuracy.
- It is flexible to both classification and regression problems.
- It works well with both categorical and continuous values.
- It automates the handling of missing values present in the data.
- Normalizing the data is not required as it uses a rule-based approach.

However, despite these advantages, a random forest algorithm also has some drawbacks:

- It requires much computational power as well as resources as it builds numerous trees to combine their outputs.
- It also requires much time for training as it combines a lot of decision trees to determine the class.
- Due to the ensemble of decision trees, it also suffers interpretability and fails to determine the significance of each variable.

3.5.1 Feature Selection

The main function of feature selection is to choose features that can positively influence the results while removing those that generate inaccuracies. Features with a strong relationship to the target attribute are selected to improve both the model's performance and computational efficiency. We utilized Recursive Feature Elimination (RFE) under the wrapper method for feature selection. In the method, the input data are divided into mul- tiple subsets; various models are then created based on these subsets. Following this, the best features are selected based on certain performance metrics. Our features were ranked

according to their importance using Recursive Feature Elimination, as illustrated in Figure. Recursive Feature Elimination is a process wherein the algorithm runs recursively until the specified number of features is selected. In the KDD process, RFE is employed to obtain the necessary dataset. We extracted the top 13 features using the "n features to select" parameter. In NSL-KDD, we eliminated redundant data by selecting a subset of relevant features.

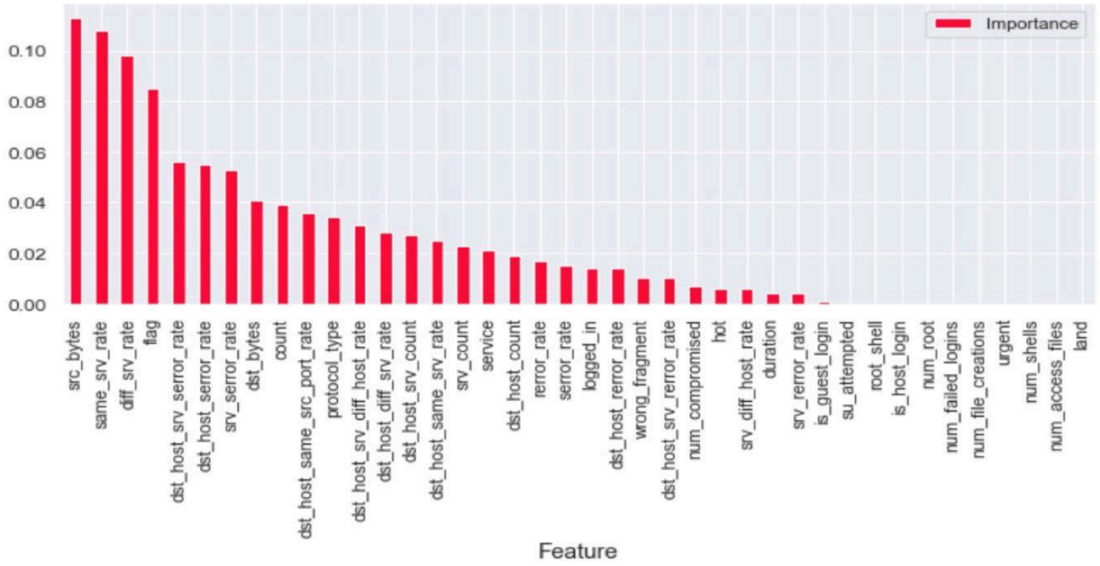


Figure 3.4: Features and their importance in KDD Dataset

The code trains Random Forest classifiers on four datasets—DoS, Probe, R2L, and U2R—using both the full set of features and a reduced set of features selected by Recursive Feature Elimination (RFE). For the full feature sets, separate RandomForestClassifier instances (clf_DoS, clf_Probe, clf_R2L, clf_U2R) are initialized with 10 estimators and 2 parallel jobs, and then fitted to their respective datasets (X_DoS, Y_DoS, X_Probe, Y_Probe, X_R2L, Y_R2L, X_U2R, Y_U2R). Similarly, for the selected feature sets, corre- sponding classifiers (clf_rfeDoS, clf_rfeProbe, clf_rfeR2L, clf_rfeU2R) are ini- tialized and fitted to the transformed datasets (X_rfeDoS, Y_DoS, X_rfeProbe, Y_Probe, X_rfeR2L, Y_R2L, X_rfeU2R, Y_U2R).

This approach allows for a comparison between the performance of classifiers trained on all features versus those trained on the selected important features.

RFE has several advantages over other feature selection methods:

- RFE can handle high-dimensional datasets and identify the most important features.
- RFE can handle interactions between features and is suitable for complex datasets.
- RFE can be used with any supervised learning algorithm.

However, RFE also has some limitations:

- RFE can be computationally expensive for large datasets.
- RFE may not be the best approach for datasets with many correlated features.
- RFE may not work well with noisy or irrelevant features.

3.6 K-nearest Neighbors

KNN is a simple, supervised machine learning (ML) algorithm that can be used for classi- fication or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most ”similar” observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing K, the user can select the number of nearby observations to use in the algorithm.

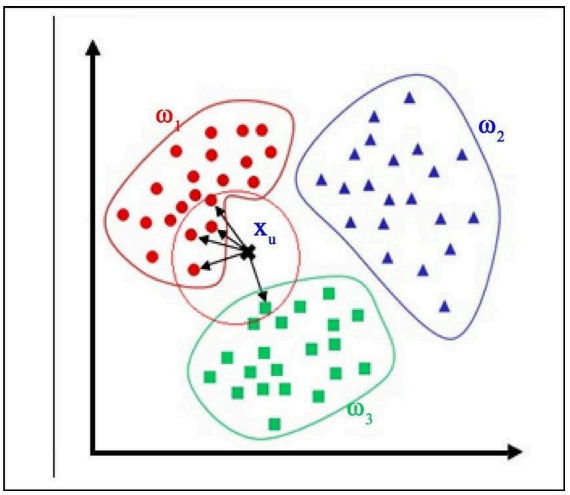


Figure 3.5: An illustration of K nearest neighbor

The code initializes and trains K-Nearest Neighbors (KNN) classifiers for four different datasets: DoS, Probe, R2L, and U2R. Four separate instances of KNeighborsClassifier are created, namely clf_KNN_DoS, clf_KNN_Probe, clf_KNN_R2L, and clf_KNN_U2R.

Each classifier is then trained using the fit method with their respective feature matrices (X_DoS, X_Probe, X_R2L, X_U2R) and label vectors (Y_DoS, Y_Probe, Y_R2L, Y_U2R). This process enables each KNN classifier to learn from the training data, empowering them to make predictions on new data points by identifying the closest training examples in the feature space.

3.7 Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm commonly used for classification and regression tasks. The primary goal of SVM is to find the optimal hyperplane that best separates data points of different classes. This hyperplane is defined in such a way that the margin, or the distance between the hyperplane and the nearest data points of any class (support vectors), is maximized. SVM can handle linear and non-linear classification by using different kernel functions. The linear kernel is suitable for linearly separable data, whereas non-linear kernels such as polynomial, radial basis function (RBF), and sigmoid are used for more complex data distributions.

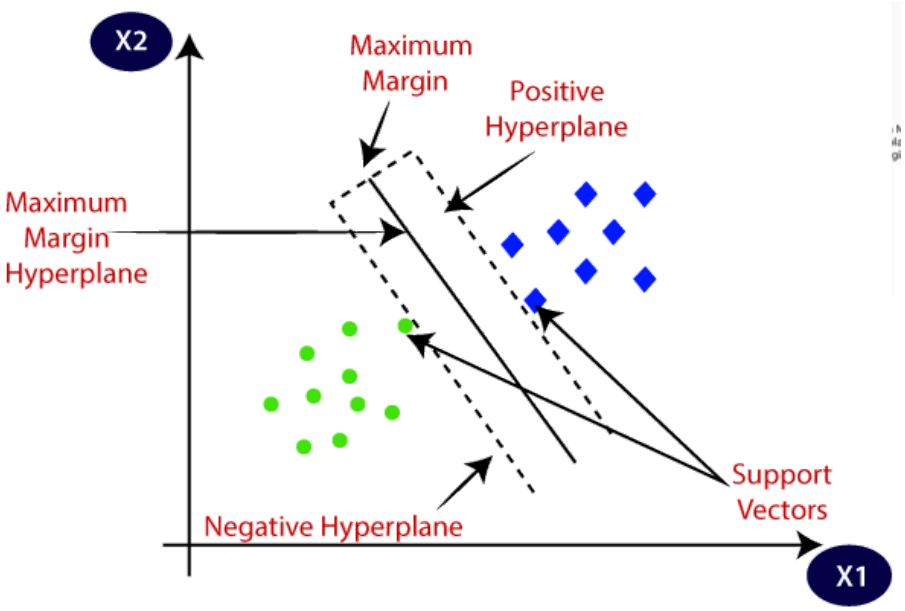


Figure 3.6: An illustration of Support Vector Machine

Support Vector Machine (SVM) classifiers are implemented to detect various types of attacks using a linear kernel. Four SVM classifiers (clf_SVM_DoS, clf_SVM_Probe, clf_SVM_R2L, clf_SVM_U2R) are initialized with a linear kernel (kernel='linear'), a regularization parameter C = 1.0, and a specified random state (random_state=0) for re-

producibility. Each classifier is then trained using the fit method on its respective dataset: clf_SVM_DoS on *X_DoS* and *Y _DoS* for DoS attacks, clf_SVM_Probe on *X_Probe* and *Y _Probe* for Probe attacks, clf_SVM_R2L on *X_R2L* and *Y _R2L* for R2L attacks, and clf_SVM_U2R on *X_U2R* and *Y _U2R* for U2R attacks. This process involves the SVM learning to find the optimal hyperplanes that best separate the normal data points from the attack data points for each type of attack, utilizing the linear kernel for linearly separable data.

Advantages of Support Vector Machine

- Support vector machine works comparably well when there is an understandable margin of dissociation between classes.
- SVMs are effective in handling high-dimensional data, which is common in many applications such as image and text classification.
- SVMs can perform well with small datasets, as they only require a small number of support vectors to define the boundary.
- SVMs can be regularized, which means that the algorithm can be modified to avoid overfitting.

Disadvantages of Support Vector Machine

- It does not execute very well when the dataset has more sound, i.e., target classes are overlapping.
- In cases where the number of properties for each data point outstrips the number of training data specimens, the support vector machine will underperform.
- SVMs can be memory-intensive, as the algorithm requires storing the kernel matrix, which can be large for large datasets.
- SVMs require complete datasets, with no missing values; it cannot handle missing values.

3.8 Ensemble Learning Voting Classifier

Ensemble learning is a powerful technique in machine learning that involves combining multiple models to solve the same problem. The primary goal of ensemble learning is to improve the predictive performance and robustness of a model. By leveraging the strengths of multiple models, ensemble methods often achieve better results than any single model alone. There are several types of ensemble methods, including bagging, boosting, vot- ing, and stacking. Each of these methods combines models in different ways to enhance performance.

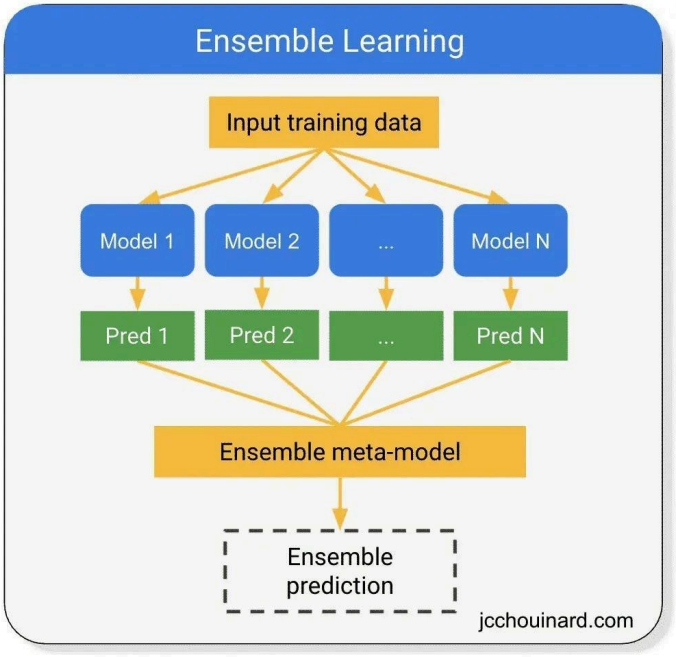


Figure assifier Ensemble learning is implemented using the Voting Classifier to combine the predictions of two different classifiers: K-Nearest Neighbors (KNN) and Support Vector Machine (SVM). Four separate Voting Classifier instances are created, each corresponding to a dif- ferent dataset: DoS, Probe, R2L, and U2R. Each VotingClassifier is configured with hard voting, meaning the final prediction for each instance is determined by the majority vote of the two classifiers (KNN and SVM). The classifiers are then trained on their respective datasets using the fit method. This ensemble approach aims to leverage the complemen- tary strengths of KNN and SVM, potentially enhancing the overall predictive performance and robustness for each type of attack detection.

Advantages of Ensemble learning

- Combining multiple models mitigates individual model errors, often leading to higher predictive accuracy.
- Ensemble methods are less susceptible to overfitting since they aggregate predictions from diverse models.
- Works well with various types of data and model architectures, providing flexibility in problem-solving.
- Ensemble techniques tend to produce more stable predictions across different datasets and scenarios, reducing model variance.

Disadvantages of Ensemble learning

- Implementing ensemble methods requires understanding and managing multiple models, which can increase computational complexity and resource demands.
- Interpretation of ensemble models becomes challenging due to the complexity introduced by combining multiple base models.
- Ensemble models often take longer to train compared to individual models due to the need to train multiple models and combine their outputs.
- Although ensemble methods can reduce overfitting, there's still a risk, especially if base models are highly correlated or if the ensemble is overly complex.

3.9 Gaussian Naive Bayes Classifier

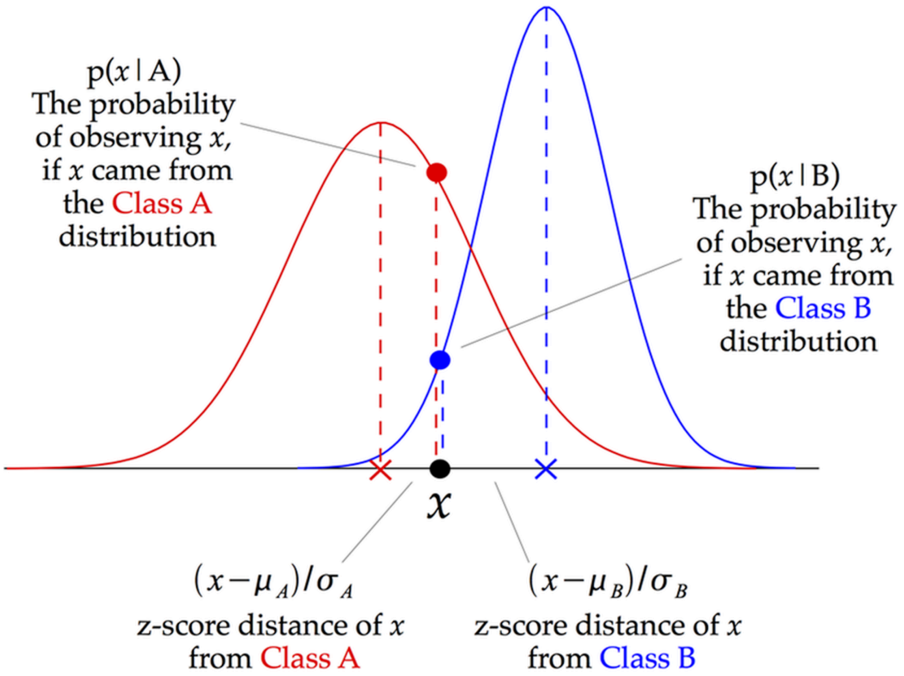
Gaussian Naive Bayes (GaussianNB) is a variant of the Naive Bayes algorithm, a popular classification technique based on Bayes' theorem. In GaussianNB, it's assumed that the features follow a Gaussian (normal) distribution. The "naive" part of Naive Bayes comes from the assumption that features are conditionally independent given the class la- bel. In other words, each feature makes an independent contribution to the probability of a particular class label, given the data. Despite its simplicity and the simplifying assump- tions it makes, Gaussian Naive Bayes can perform remarkably well

in many real-world

classification tasks, particularly when dealing with datasets with continuous features. It's computationally efficient, requires a relatively small amount of training data to estimate the necessary parameters, and tends to be robust to irrelevant features. However, it may perform poorly if the independence assumption is significantly violated in the dataset. Overall, Gaussian Naive Bayes is a useful and widely applied algorithm in machine learn- ing, particularly in text classification and other domains where the features can be assumed to have a Gaussian distribution.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A) \cdot P(A)}{P(B)}$$

$$P(x_i|y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



Classifier Advantages of Gaussian Naive Bayes Classifier

- Gaussian Naive Bayes (GNB) is highly efficient, offering a fast training process and low computational cost, which is advantageous for large datasets and real-time applications.

- Its independence assumption enables effective handling of irrelevant features and good scalability with both the number of features and instances, requiring minimal memory as it only stores means and variances for each feature-class combination.
- The algorithm is robust to missing data and can manage high-dimensional datasets where the number of features exceeds the number of samples.
- GNB produces probabilistic outputs useful for decision-making and serves as a good baseline classifier for comparing more complex models.

Disadvantages of Gaussian Naive Bayes Classifier

- The assumption of feature independence is often violated in network data where features like packet size, IP addresses, and time intervals are typically correlated, leading to reduced detection accuracy.
- GNB assumes that features follow a Gaussian distribution within each class, but network traffic data often deviates from this, resulting in inaccurate probability es- timates.
- GNB struggles with categorical features, common in network data, requiring pre-processing that can introduce complexity and potential information loss.
- Class imbalance in network intrusion data, with fewer intrusion instances compared to normal traffic, causes GNB to be biased towards the majority class and reduces detection rates for intrusions.

Applications of Gaussian Naive Bayes Classifier

- GNB's simplicity and efficiency make it suitable for detecting unusual traffic patterns and behavioral anomalies, indicating potential intrusions.
- It can classify incoming traffic to identify malicious IPs and detect port scanning activities by analyzing connection attempts.
- GNB assists in signature-based detection by providing probabilistic assessments of known attack patterns and helps identify zero-day attacks through traffic

pattern analysis.

- It can detect protocol anomalies in HTTP and DNS traffic, monitor network traffic volume and performance in real-time to identify DDoS attacks or disruptions, and analyze user and entity behaviors to detect compromised accounts or insider threats.
- GNB can generate real-time alerts for quick incident response and integrate into automated response systems to trigger actions like blocking IPs or isolating devices when anomalies are detected.
- It can monitor data flows to detect unusual data transfer activities and correlate ob- served network traffic with external threat intelligence feeds to enhance the detection of known malicious activities.

Gaussian Naive Bayes (GNB) classifiers are employed to evaluate their performance in detecting different types of network intrusions such as Denial of Service (DoS), Probe, Remote-to-Local (R2L), and User-to-Root (U2R). The process begins with the importation of essential libraries from scikit-learn for tasks such as data splitting, cross-validation, and classifier instantiation. A custom function, print_metrics_gaussian_naive_bayes, is defined to compute evaluation metrics including accuracy, precision, recall, and F- measure using cross-validation with 10 folds. Additionally, it visualizes these metrics through bar plots. Data for each intrusion type is divided into training and testing sets us- ing the train_test_split function, ensuring a test size of 20% and consistent random state for reproducibility. Subsequently, GNB classifiers are created and trained on the re- spective training data using the fit method. Finally, the performance of each classifier is assessed by invoking the print_metrics_gaussian_naive_bayes function for each intrusion type, which computes and displays the evaluation metrics along with error bars to visualize the variability in performance.

This systematic approach enables a comprehensive assessment of GNB classifiers' effectiveness in detecting various network intrusions, providing valuable insights into their capability for intrusion detection tasks.

3.10 Principal Component Analysis

Principal component analysis, or PCA, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables

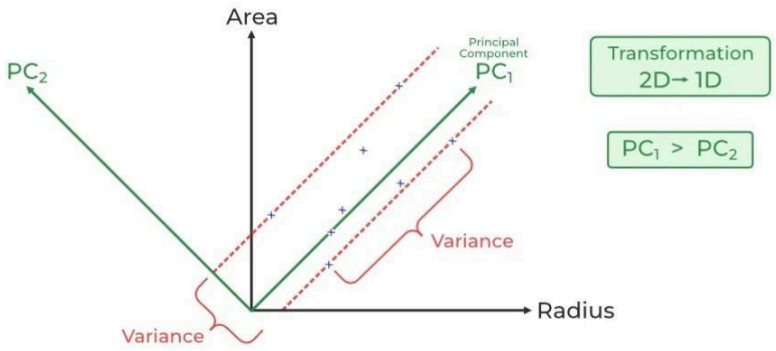


Figure 3.9: An illustration of Principal Component Analysis

$$cov(x1, x2) = \frac{\sum_{i=1}^{n}(x1_i - x1)(x2_i - x2)}{n-1}$$

into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize, and thus make analyzing data points much easier and faster for machine learning algorithms without extraneous variables to process. So, to sum up, the idea of PCA is simple: reduce the number of variables of a data set, while pre- serving as much information as possible. Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maxi- mum remaining information in the second and so on. Organizing information in principal components this way will allow you to reduce dimensionality without losing much infor- mation, and this by discarding the components with low information and considering the remaining components as your new variables. An important thing to realize here is that the principal components are less interpretable and don't have any real meaning since they are constructed as linear combinations of the initial variables.

For the DoS, Probe, U2R, and R2L attacks, the process involves fitting a Gaussian Naive Bayes classifier to the corresponding attack data after PCA transformation. This

$$|A - \lambda I| = 0$$

classifier learns from the features derived through PCA and their corresponding attack la- bels. Once the model is fitted, predictions are made for each attack category using the trained classifier. The process also includes printing headers to denote the section related to each attack category and generating a confusion matrix to evaluate the classifier's per- formance on the respective attacks.

Advantages of Principal Component Analysis

- Dimensionality Reduction: Principal Component Analysis is a popular technique used for dimensionality reduction, which is the process of reducing the number of variables in a dataset. By reducing the number of variables, PCA simplifies data analysis, improves performance, and makes it easier to visualize data.
- Feature Selection: Principal Component Analysis can be used for feature selection, which is the process of selecting the most important variables in a dataset. This is useful in machine learning, where the number of variables can be very large, and it is difficult to identify the most important variables.
- Data Visualization: Principal Component Analysis can be used for data visualization. By reducing the number of variables, PCA can plot high-dimensional data in two or three dimensions, making it easier to interpret.
- Multicollinearity: Principal Component Analysis can be used to deal with multicollinearity, which is a common problem in regression analysis where two or more independent variables are highly correlated. PCA can help identify the underlying structure in the data and create new, uncorrelated variables that can be used in the regression model.
- Noise Reduction: Principal Component Analysis can be used to reduce the noise in data. By removing the principal components with low variance, which are assumed to represent noise, Principal Component Analysis can improve the signal-to-noise ratio and make it easier to identify the underlying structure in the data.
- Data Compression: Principal Component Analysis can be used for data compres-

sion. By representing the data using a smaller number of principal components,

which capture most of the variation in the data, PCA can reduce the storage requirements and speed up processing.

- Outlier Detection: Principal Component Analysis can be used for outlier detection. Outliers are data points that are significantly different from the other data points in the dataset. Principal Component Analysis can identify these outliers by looking for data points that are far from the other points in the principal component space.

Disadvantages of Principal Component Analysis

- Interpretation of Principal Components: The principal components created by Prin- cipal Component Analysis are linear combinations of the original variables, and it is often difficult to interpret them in terms of the original variables. This can make it difficult to explain the results of PCA to others.

- Data Scaling: Principal Component Analysis is sensitive to the scale of the data. If the data is not properly scaled, then PCA may not work well. Therefore, it is important to scale the data before applying Principal Component Analysis.

- Information Loss: Principal Component Analysis can result in information loss. While Principal Component Analysis reduces the number of variables, it can also lead to loss of information. The degree of information loss depends on the number of principal components selected. Therefore, it is important to carefully select the number of principal components to retain.

- Non-linear Relationships: Principal Component Analysis assumes that the relation- ships between variables are linear. However, if there are non-linear relationships between variables, Principal Component Analysis may not work well.

- Computational Complexity: Computing Principal Component Analysis can be com- putationally expensive for large datasets. This is especially true if the number of variables in the dataset is large.

- Overfitting: Principal Component Analysis can sometimes result in overfitting,

which is when the model fits the training data too well and performs poorly on new data.

This can happen if too many principal components are used or if the model is trained on a small dataset.

3.11 XGBoost Classifier

XGBoost is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models. It is an ensemble learning method that com- bines the predictions of multiple weak models to produce a stronger prediction. XGBoost stands for “Extreme Gradient Boosting” and it has become one of the most popular and widely used machine learning algorithms due to its ability to handle large datasets and its ability to achieve state-of-the-art performance in many machine learning tasks such as classification and regression. One of the key features of XGBoost is its efficient han- dling of missing values, which allows it to handle real-world data with missing values without requiring significant pre-processing. Additionally, XGBoost has built-in support for parallel processing, making it possible to train models on large datasets in a reason- able amount of time. XGBoost can be used in a variety of applications, including Kaggle competitions, recommendation systems, and click-through rate prediction, among others. It is also highly customizable and allows for fine-tuning of various model parameters to optimize performance.

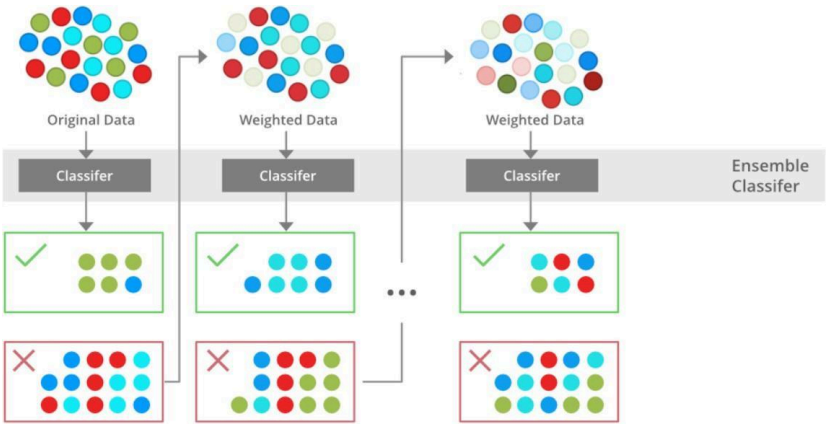


Figure 3.10: An illustration of XGBoost Classifier

$$obj^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

The process involves splitting the dataset into features (X) and labels (Y) for each attack category (DoS, Probe, R2L, U2R). Principal Component Analysis (PCA) is then applied to reduce the dimensionality of these feature sets. PCA transforms the data into a lower-dimensional space, retaining most of the variance. This transformation simplifies the data, making it more manageable for the XGBoost classifier to process. For each attack category, an XGBoost classifier is trained on the PCA-reduced data, and predictions are made using the trained model. The model's performance is then evaluated using confusion matrices and classification reports, which provide detailed insights into the accuracy and effectiveness of the model in classifying the attacks.

Advantages of XGBoost

- Performance: XGBoost has a strong track record of producing high-quality results in various machine learning tasks, especially in Kaggle competitions, where it has been a popular choice for winning solutions.
- Scalability: XGBoost is designed for efficient and scalable training of machine learning models, making it suitable for large datasets.
- Customizability: XGBoost has a wide range of hyperparameters that can be adjusted to optimize performance, making it highly customizable.
- Handling of Missing Values: XGBoost has built-in support for handling missing values, making it easy to work with real-world data that often has missing values.
- Interpretability: Unlike some machine learning algorithms that can be difficult to in- terpret, XGBoost provides feature importances, allowing for a better understanding of which variables are most important in making predictions.

Disadvantages of XGBoost

- Computational Complexity: XGBoost can be computationally intensive, especially when training large models, making it less suitable for resource-constrained systems.

- Overfitting: XGBoost can be prone to overfitting, especially when trained on small datasets or when too many trees are used in the model.
- Hyperparameter Tuning: XGBoost has many hyperparameters that can be adjusted, making it important to properly tune the parameters to optimize performance. How- ever, finding the optimal set of parameters can be time-consuming and requires ex- pertise.
- Memory Requirements: XGBoost can be memory-intensive, especially when work- ing with large datasets, making it less suitable for systems with limited memory resources.

3.12 Summary

XGBoost is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models. It is an ensemble learning method that com- bines the predictions of multiple weak models to produce a stronger prediction. XGBoost stands for “Extreme Gradient Boosting” and it has become one of the most popular and widely used machine learning algorithms due to its ability to handle large datasets and its ability to achieve state-of-the-art performance in many machine learning tasks such as classification and regression. One of the key features of XGBoost is its efficient han- dling of missing values, which allows it to handle real-world data with missing values without requiring significant pre-processing. Additionally, XGBoost has built-in support for parallel processing, making it possible to train models on large datasets in a reason- able amount of time. XGBoost can be used in a variety of applications, including Kaggle competitions, recommendation systems, and click-through rate prediction, among others. It is also highly customizable and allows for fine-tuning of various model parameters to optimize performance.

Chapter 4

RESULTS AND DISCUSSIONS

4.1 Results

4.1.1 Introduction

The performance evaluation of machine learning models is a critical aspect of building effective predictive systems. In this study, we assess the performance of various classifiers using multiple metrics, including accuracy, precision, recall, and F-measure. These met- rics provide insights into the models' ability to correctly classify instances across different classes while considering aspects.

4.1.2 Random Forest Classifier

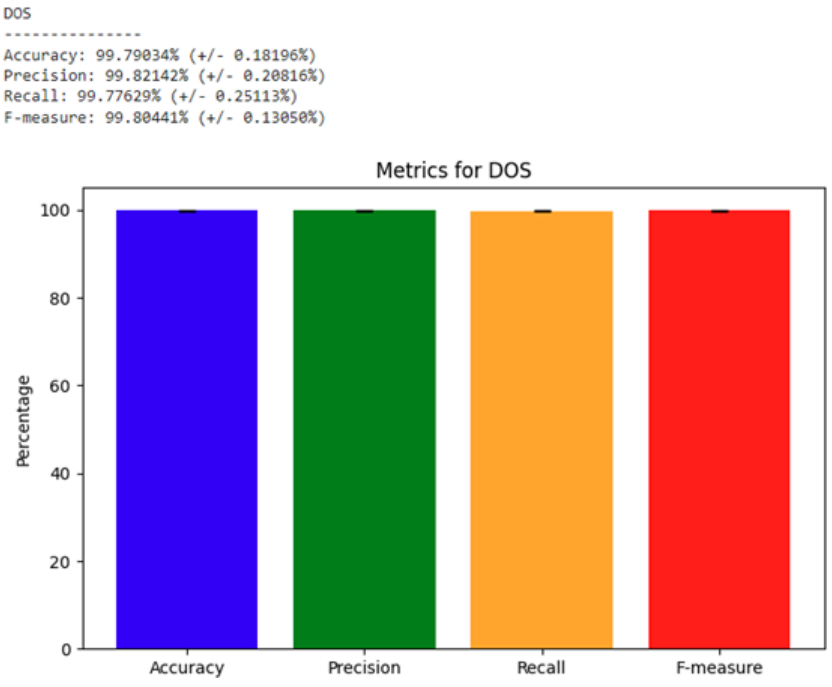


Figure 4.1: Bar graph: Random Forest classifier DOS metrics

```
Probe
---------------
Accuracy: 99.67028% (+/- 0.31275%)
Precision: 99.63363% (+/- 0.46644%)
Recall: 99.29804% (+/- 0.47374%)
F-measure: 99.45613% (+/- 0.57727%)
```

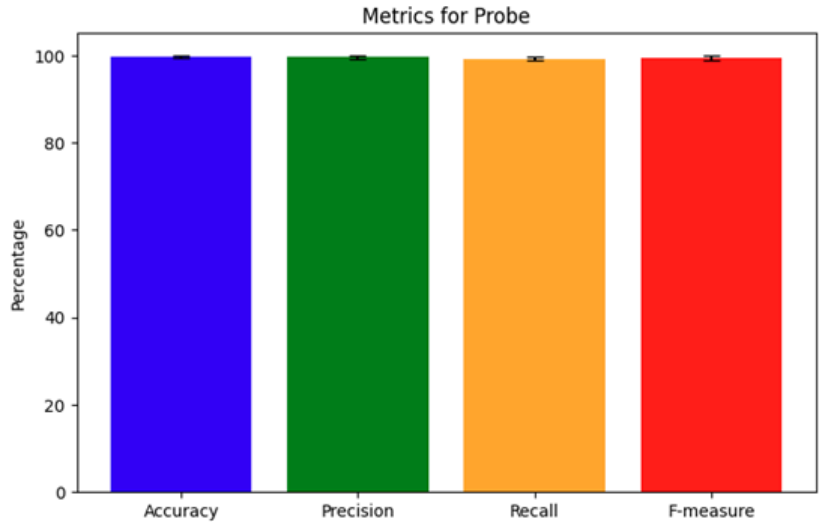


Figure 4.2: Bar graph: Random Forest classifier Probe metrics

```
R2L
---------------
Accuracy: 97.99934% (+/- 0.54476%)
Precision: 97.49460% (+/- 0.96211%)
Recall: 97.09905% (+/- 0.83018%)
F-measure: 97.20884% (+/- 0.83506%)
```

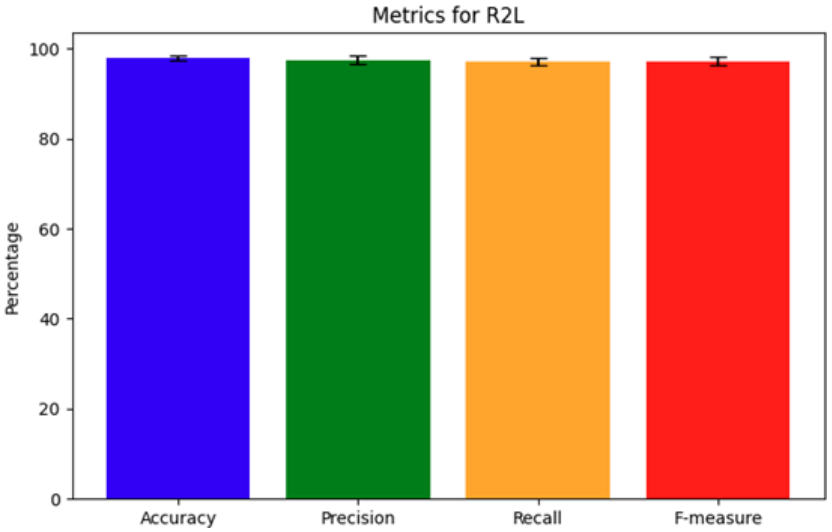


Figure 4.3: Bar graph: Random Forest classifier U2R metrics

```
U2R
---------------
Accuracy: 99.77500% (+/- 0.23852%)
Precision: 97.23058% (+/- 8.21808%)
Recall: 82.84685% (+/- 15.92848%)
F-measure: 87.59119% (+/- 14.17128%)
```

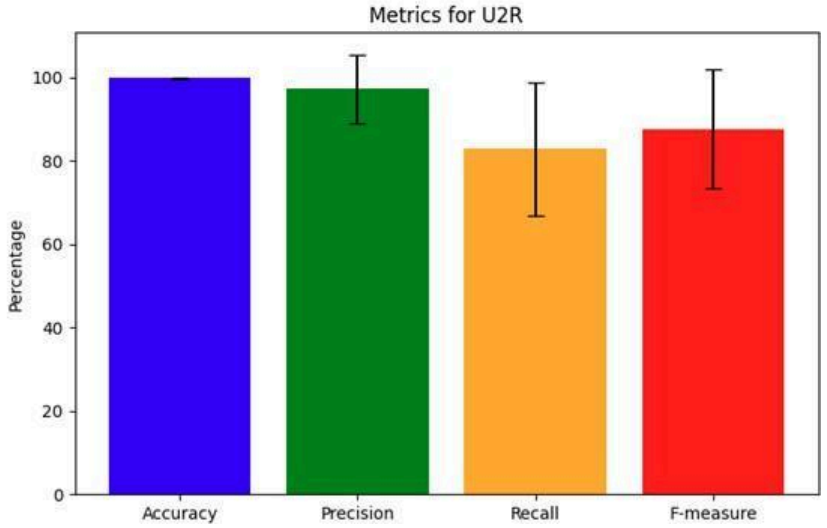


Figure 4.4: Bar graph: Random Forest classifier R2L metrics

For the Probe dataset, the classifier also performed very well, although with slightly more variability. The accuracy averaged 99.67028% with a standard deviation of 0.31275%, indicating high accuracy but a bit more fluctuation compared to the DoS dataset. The precision was 99.63363% with a larger standard deviation of 0.46644%, suggesting that while the model was generally precise, there were occasional variations in identifying true positives. The recall was 99.29804% with a standard deviation of 0.47374%, showing a slight decrease in consistently identifying all actual positive cases. The F-measure was 99.45613% with a standard deviation of 0.57727%, indicating a solid balance between precision and recall despite some variability.

The classifier for the U2R dataset showed strong accuracy but revealed some challenges with precision and recall. The average accuracy was 99.77500% with a standard deviation of 0.23852%, indicating high accuracy. However, the precision dropped signif- icantly to 97.23058% with a high standard deviation of 8.21808%, suggesting inconsis- tent identification of true positives. The recall was lower at 82.84685% with a very high standard deviation of 15.92848%, reflecting substantial variability in capturing all actual positive cases. Consequently, the F-measure was 87.59119% with a standard deviation of 14.17128%, indicating that while the model was generally good, there were significant inconsistencies in its performance for the U2R dataset.

The classifier for the R2L dataset exhibited strong and consistent performance. The average accuracy was 97.99934% with a standard deviation of 0.54476%, demonstrating high and stable accuracy. The precision was similarly high at 97.49460% with a stan- dard deviation of 0.96211%, indicating reliable identification of true positives. Recall was 97.09905% with a standard deviation of 0.83018%, showing the model's capability to consistently capture most of the actual positive cases. The F-measure was 97.20884% with a standard deviation of 0.83506%, confirming the classifier's balanced and robust performance on the R2L dataset.

4.1.3 K-Nearest Neighbors

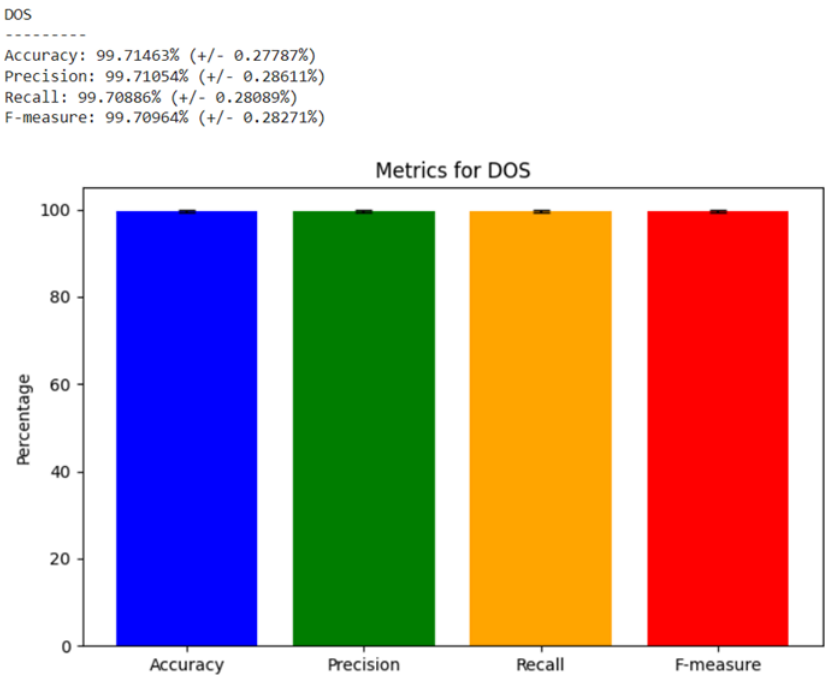


Figure 4.5: Bar graph: K-Nearest Neighbors DOS metrics

The K-Nearest Neighbors (KNN) classifier for the DoS dataset demonstrated excel- lent performance. The average accuracy was 99.71463%, with a standard deviation of 0.27787%, indicating high accuracy with minimal variation across the 10-fold cross-validation. Precision was similarly high at 99.71054% with a standard deviation of 0.28611%, reflect- ing the model's effectiveness in identifying true positives. Recall was 99.70886% with a standard deviation of 0.28089%, indicating the model's strong ability to capture most of the actual positive cases. The F-measure, balancing precision and recall, was 99.70964%

```
Probe
---------
Accuracy: 99.07681% (+/- 0.40266%)
Precision: 98.60594% (+/- 0.67516%)
Recall: 98.50846% (+/- 1.13667%)
F-measure: 98.55258% (+/- 0.64466%)
```

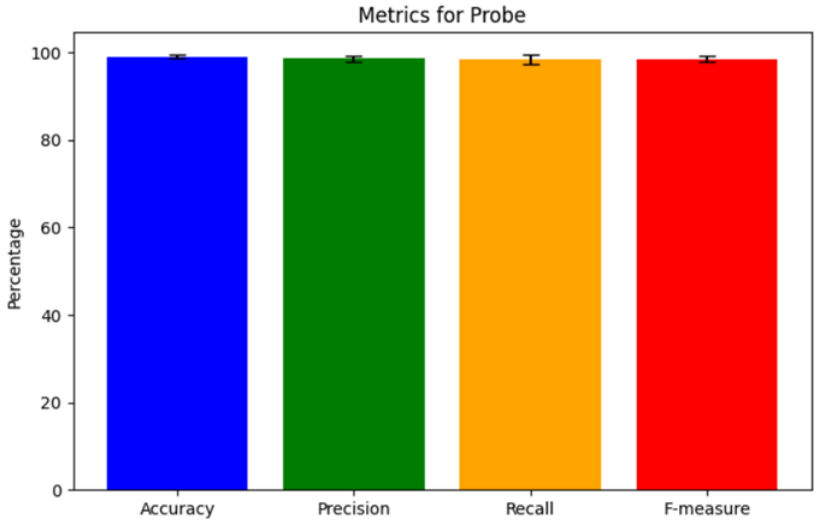


Figure 4.6: Bar graph: K-Nearest Neighbors Probe metrics

```
U2R
---------
Accuracy: 99.70341% (+/- 0.28118%)
Precision: 93.14325% (+/- 14.67911%)
Recall: 85.07271% (+/- 17.63892%)
F-measure: 87.83136% (+/- 11.38995%)
```

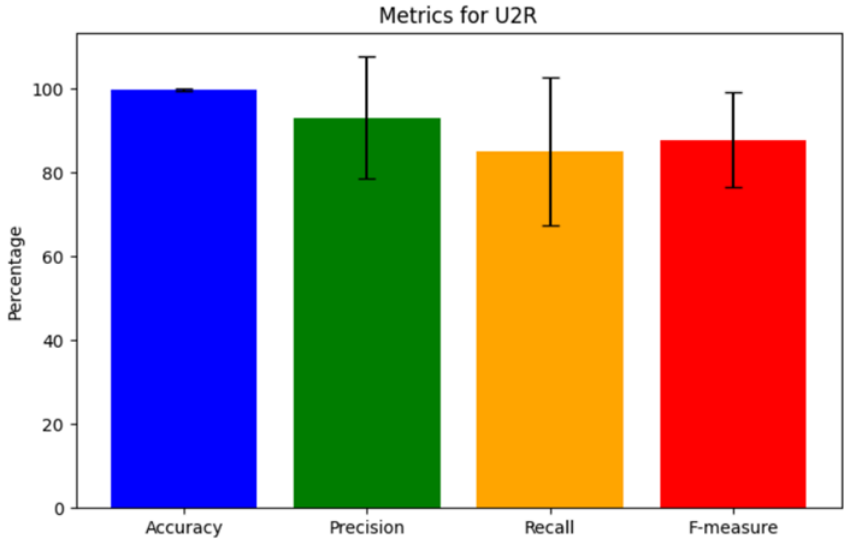


Figure 4.7: Bar graph: K-Nearest Neighbors U2R metrics

```
R2L
---------
Accuracy: 96.74500% (+/- 0.73459%)
Precision: 95.32528% (+/- 1.24687%)
Recall: 95.48918% (+/- 1.32406%)
F-measure: 95.40003% (+/- 1.03913%)
```

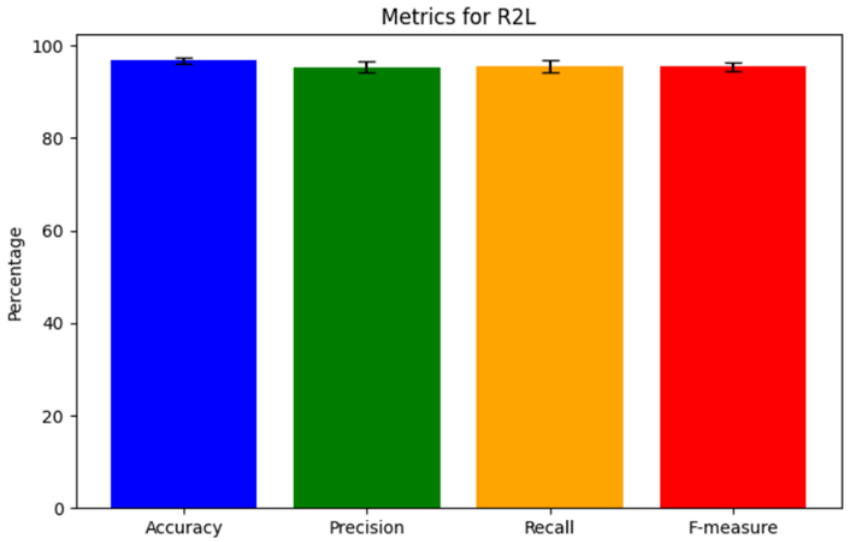


Figure 4.8: Bar graph: K-Nearest Neighbors R2L metrics

with a standard deviation of 0.28271%, confirming the classifier's robust performance on the DoS dataset.

For the Probe dataset, the KNN classifier performed well but showed more variability than the DoS dataset. The accuracy averaged 99.07681% with a standard deviation of 0.40266%, indicating high accuracy but some fluctuation across the folds. The precision was 98.60594% with a higher standard deviation of 0.67516%, suggesting that while the model was generally precise, there was notable variability in correctly identifying true positives. Recall was 98.50846% with a standard deviation of 1.13667%, showing the model's ability to consistently capture most actual positive cases, though with some vari- ability. The F-measure was 98.55258% with a standard deviation of 0.64466%, indicating a good balance between precision and recall despite the observed variability.

The KNN classifier for the R2L dataset showed strong but slightly less consistent per- formance. The average accuracy was 96.74500% with a standard deviation of 0.73459%, indicating good accuracy but with more fluctuation compared to the previous datasets. Pre- cision was 95.32528% with a standard deviation of 1.24687%, suggesting reliable identi- fication of true positives but with some variability. Recall was 95.48918% with a standard deviation of 1.32406%, showing the model's capability

to capture most actual positive

cases, though with noticeable variation. The F-measure was 95.40003% with a standard deviation of 1.03913%, confirming the classifier's balanced and robust performance on the R2L dataset despite some inconsistencies.

The KNN classifier for the U2R dataset exhibited strong accuracy but revealed challenges with precision and recall. The average accuracy was 99.70341% with a standard deviation of 0.28118%, indicating high accuracy. However, precision dropped to 93.14325% with a high standard deviation of 14.67911%, reflecting inconsistent identifi- cation of true positives. Recall was lower at 85.07271% with a very high standard devia- tion of 17.63892%, showing substantial variability in capturing all actual positive cases. Consequently, the F-measure was 87.83136% with a standard deviation of 11.38995%, indicating that while the model was generally good, there were significant inconsistencies in its performance for the U2R dataset.

4.1.4 Support Vector Machine

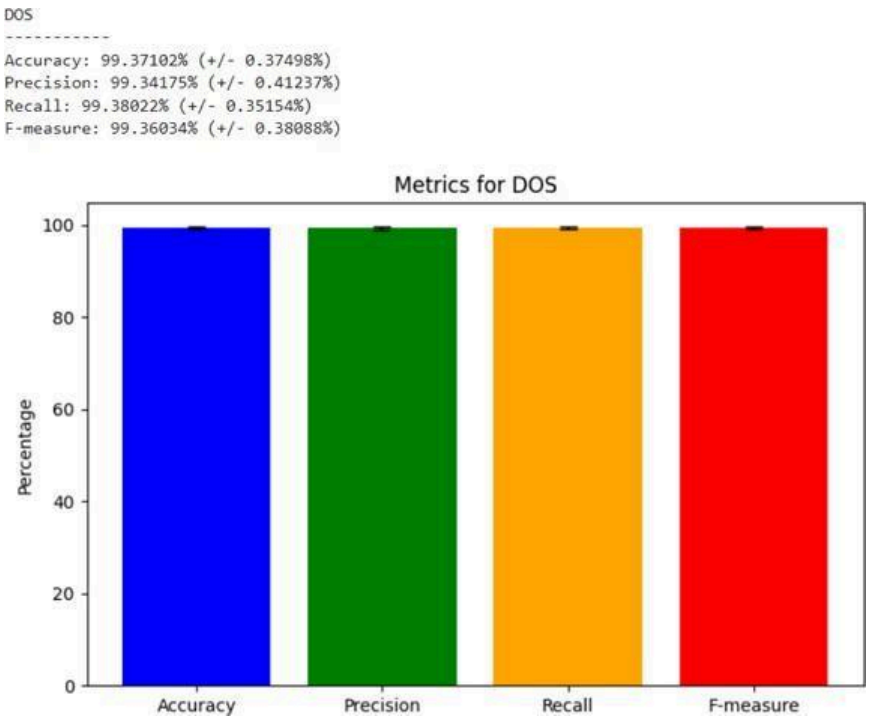


Figure 4.9: Bar graph: Support Vector Machine DOS metrics

The performance metrics for the DOS attack detection using the SVM classifier indicate high accuracy and consistency. The classifier achieved an accuracy of 99.37102% with a standard deviation of 0.37498%. The precision and recall scores are also high,

```
Probe
-----------
Accuracy: 98.45038% (+/- 0.52551%)
Precision: 96.90722% (+/- 1.03132%)
Recall: 98.36543% (+/- 0.68626%)
F-measure: 97.61301% (+/- 0.80023%)
```

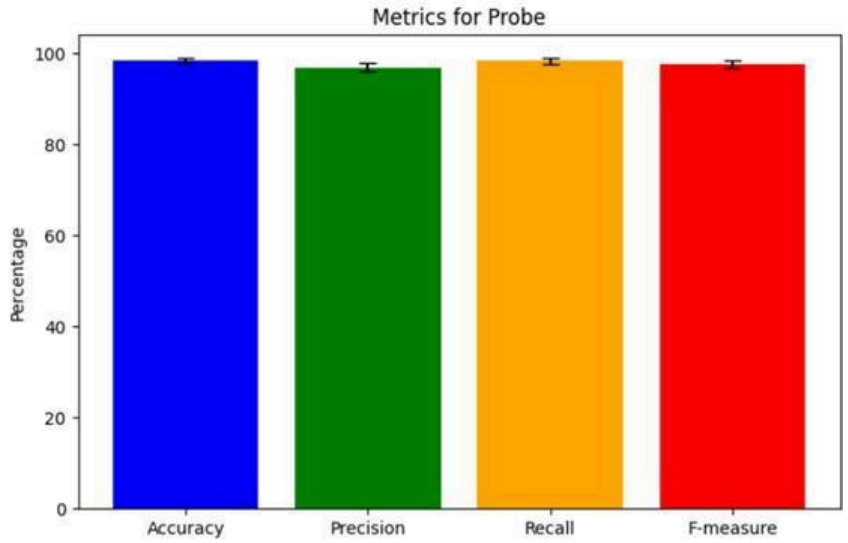


Figure 4.10: Bar graph: Support Vector Machine Probe metrics

```
U2R
-----------
Accuracy: 99.63185% (+/- 0.39010%)
Precision: 91.05555% (+/- 17.93427%)
Recall: 82.90926% (+/- 21.83306%)
F-measure: 84.86852% (+/- 16.02890%)
```

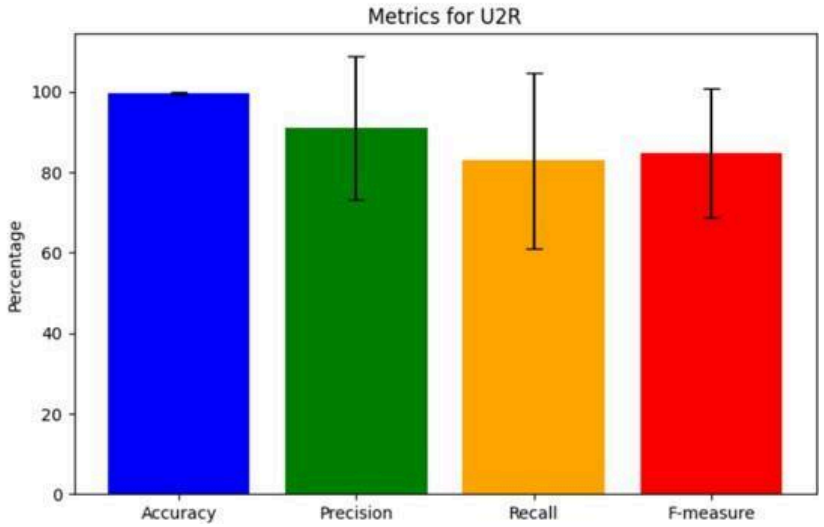


Figure 4.11: Bar graph: Support Vector Machine U2R metrics

```
R2L
-----------
Accuracy: 96.79264% (+/- 0.73846%)
Precision: 94.85396% (+/- 0.99350%)
Recall: 96.26352% (+/- 1.38794%)
F-measure: 95.52904% (+/- 1.04836%)
```

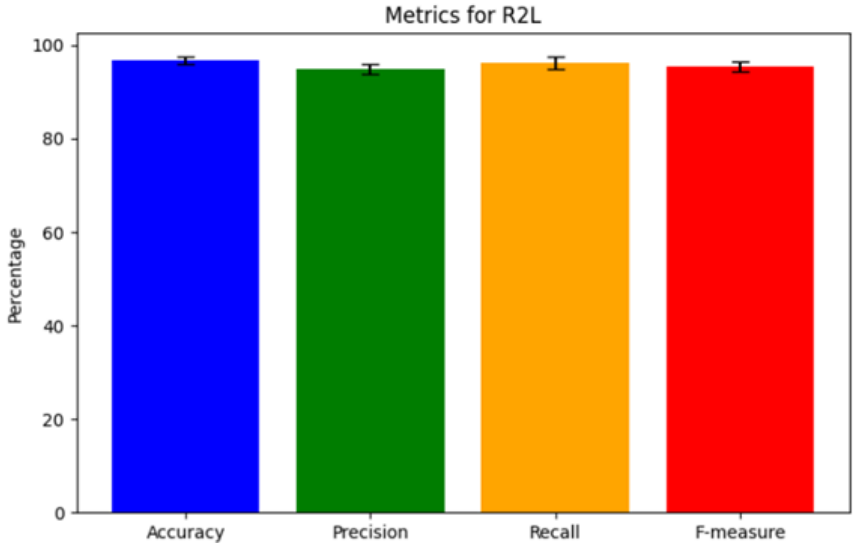


Figure 4.12: Bar graph: Support Vector Machine R2L metrics

at 99.34175% and 99.38022% respectively, with relatively low variability, indicating the model's reliability in identifying true positives and minimizing false positives. The F-measure of 99.36034% further underscores the model's effectiveness in balancing precision and recall, making it a robust choice for detecting DOS attacks.

For the Probe attack detection, the SVM classifier showed an accuracy of 98.45038% with a standard deviation of 0.52551%. The precision score is 96.90722%, while the recall is slightly higher at 98.36543%, suggesting that the model is slightly better at identifying actual Probe attacks than avoiding false positives. The F-measure of 97.61301% indi- cates a good balance between precision and recall, although the precision score's higher variability suggests that there may be more fluctuation in the model's performance across different cross-validation folds.

In detecting R2L attacks, the SVM classifier achieved an accuracy of 96.79264% with a standard deviation of 0.73846%. The precision score is 94.85396%, and the recall is higher at 96.26352%, indicating that the model is effective at correctly identifying R2L attacks but slightly less consistent in avoiding false positives. The F-measure of 95.52904% reflects a solid balance between precision and recall, though the variability in precision suggests some inconsistency in the model's predictions across different datasets.

The performance metrics for U2R attack detection show that the SVM classifier achieved an accuracy of 99.63185% with a standard deviation of 0.39010%. The precision score is 91.05555%, but the recall drops to 82.90926%, indicating that while the model can ac- curately identify U2R attacks, it may miss a significant number of true U2R attacks. The F-measure of 84.86852% highlights this discrepancy, suggesting that the model's ability to balance precision and recall is less effective for U2R attacks compared to other attack types. The high variability in both precision and recall indicates that the model's perfor- mance is less stable and more dependent on the specific data it encounters.

4.1.5 Ensemble Learning

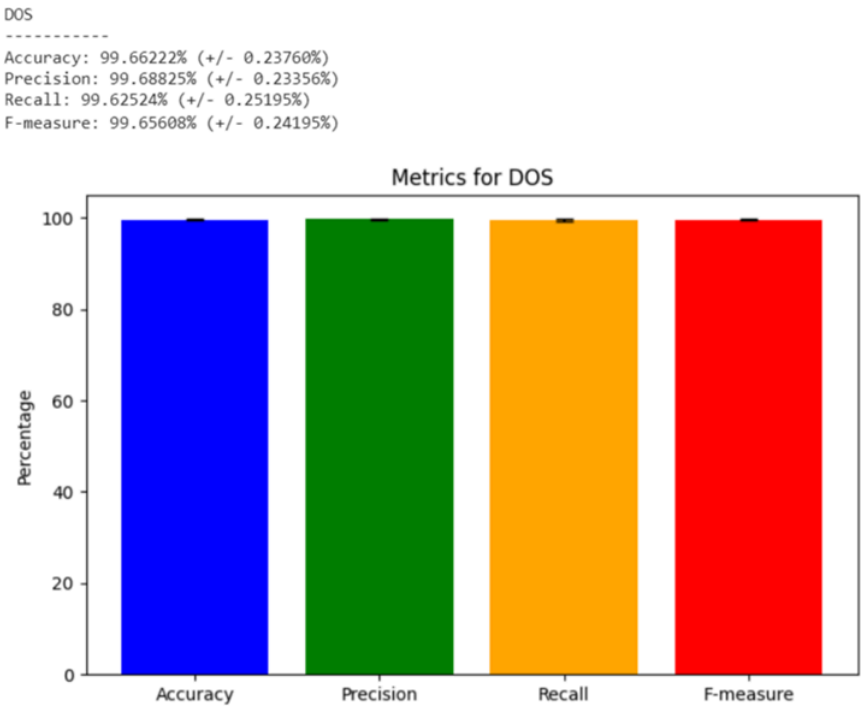


Figure 4.13: Bar graph: Ensemble Learning voting Classifier DOS metrics

For the Detection of Service (DoS) attack detection, the ensemble model achieved an impressive accuracy of 99.66222% with a standard deviation of ±0.23760%. The precision is similarly high at 99.68825% ±0.23356%, indicating the model's ability to correctly identify positive instances of DoS attacks. Recall is also robust at 99.62524% ±0.25195%, reflecting the model's effectiveness in identifying true positive cases. The F-measure, which balances precision and recall, stands at 99.65608% ±0.24195%, demonstrating the model's overall high performance and reliability in detecting DoS

attacks.

```
Probe
-----------
Accuracy: 98.96965% (+/- 0.39892%)
Precision: 98.59196% (+/- 0.67255%)
Recall: 98.17798% (+/- 1.04525%)
F-measure: 98.37969% (+/- 0.63986%)
```

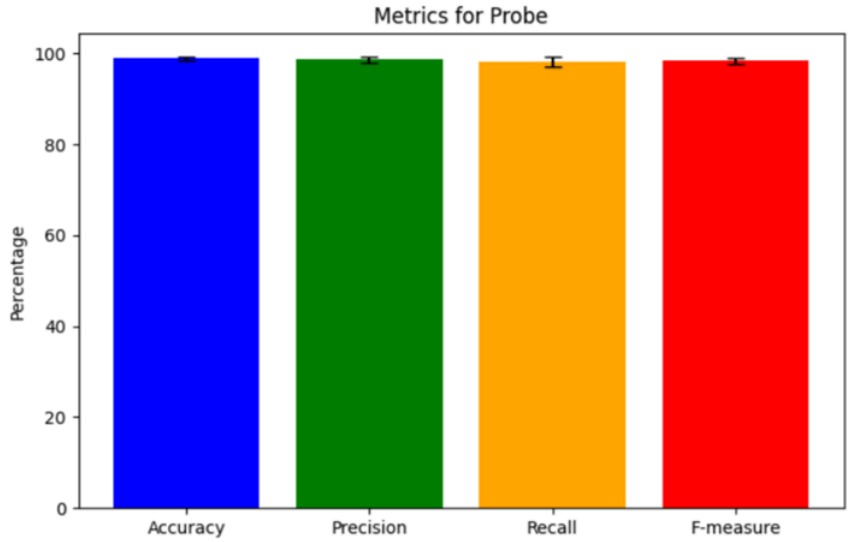


Figure 4.14: Bar graph: Ensemble Learning voting Classifier Probe metrics

```
U2R
-----------
Accuracy: 99.62161% (+/- 0.35477%)
Precision: 92.70313% (+/- 15.64309%)
Recall: 77.82110% (+/- 19.67592%)
F-measure: 82.66023% (+/- 17.13997%)
```

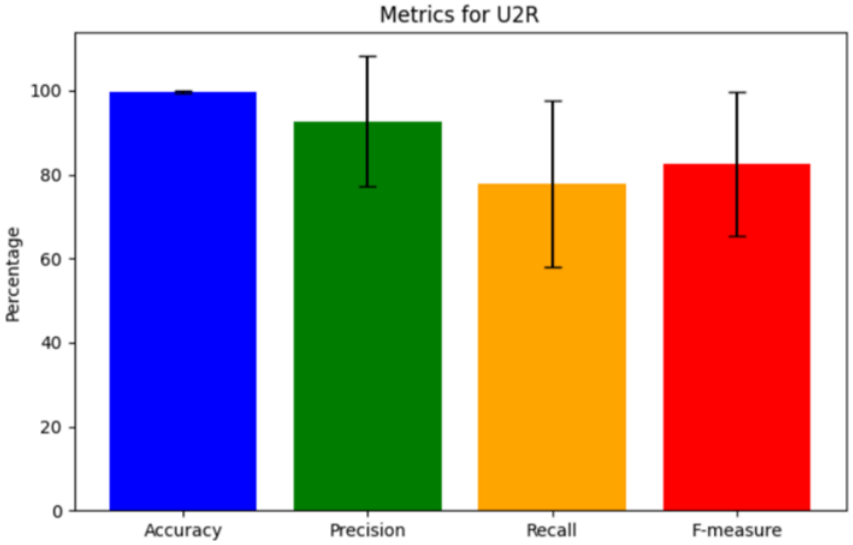


Figure 4.15: Bar graph: Ensemble Learning voting Classifier U2R metrics

```
R2L
-----------
Accuracy: 96.55448% (+/- 0.82229%)
Precision: 95.58714% (+/- 0.98276%)
Recall: 94.58607% (+/- 1.90257%)
F-measure: 95.06447% (+/- 1.23041%)
```

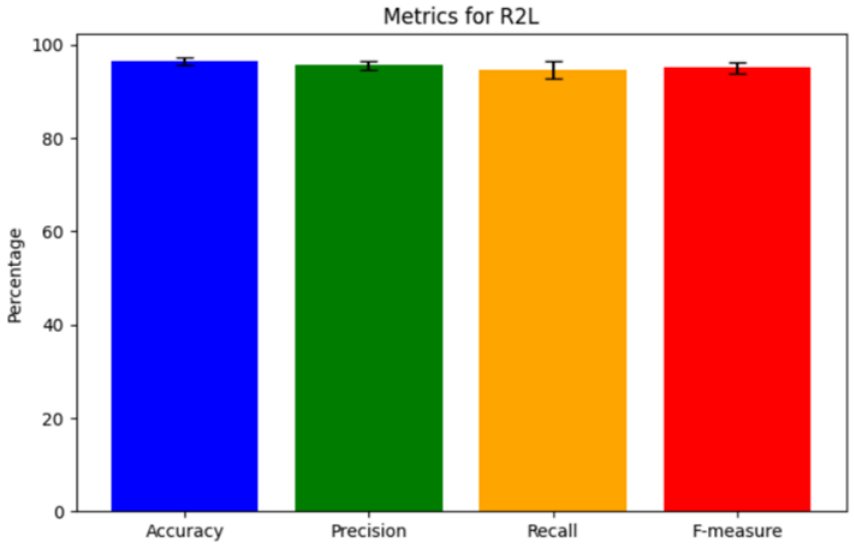


Figure 4.16: Bar graph: Ensemble Learning voting Classifier R2L metrics

For Probe attack detection, the ensemble model achieved an accuracy of 98.96965% with a standard deviation of ±0.39892%. Precision is 98.59196% ±0.67255%, indicating the model's correctness in predicting Probe attacks. Recall is slightly lower at 98.17798%

±1.04525%, indicating the model's capability in identifying actual Probe attack cases. The F-measure is 98.37969% ±0.63986%, highlighting a well-balanced performance in terms of precision and recall, confirming the model's effectiveness in Probe attack detection.

For User to Root (U2R) attack detection, the ensemble model achieved an accuracy of 99.62161% with a standard deviation of ±0.35477%. Precision is 92.70313% ±15.64309%, indicating variability in the model's correctness in predicting U2R attacks. Recall is 77.82110% ±19.67592%, reflecting a challenge in capturing all true U2R attacks. The F-measure stands at 82.66023% ±17.13997%, showing a significant drop compared to other attack types, and highlighting the difficulty of achieving a balanced performance for U2R attack detection.

For Remote to Local (R2L) attack detection, the ensemble model recorded an accuracy of 96.55448% with a standard deviation of ±0.82229%. Precision is 95.58714%

±0.98276%, indicating a high level of correctness in identifying R2L attacks. Recall is 94.58607% ±1.90257%, reflecting the model's proficiency in capturing actual R2L at-

tacks. The F-measure is 95.06447% ±1.23041%, suggesting that the model performs well in balancing precision and recall, making it reliable for R2L attack detection.

4.1.6 Gaussian Naive Bayes Classifier

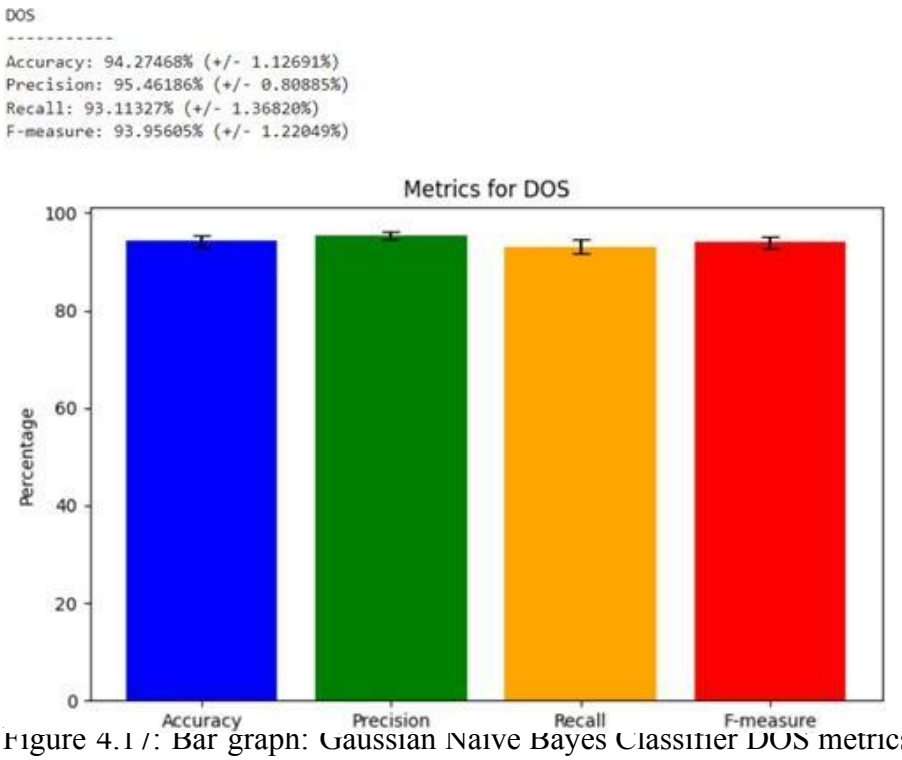


Figure 4.17: Bar graph: Gaussian Naive Bayes Classifier DOS metrics Metrics pertain to the performance evaluation of a classifier in detecting Denial of Service (DoS) attacks. The accuracy of around 94.27% signifies the overall correctness of the classifier's predictions. Precision, approximately 95.46%, denotes the proportion of correctly predicted attacks among all instances classified as attacks. Recall, at about 93.11%, indicates the proportion of correctly identified attacks out of all actual attacks. The F-measure, approximately 93.96%, represents the harmonic mean of precision and recall, offering a balanced assessment of the classifier's performance.

Metrics pertain to the performance evaluation of a classifier in detecting Probe attacks. The accuracy, approximately 88.70%, indicates the overall correctness of the classifier's predictions. Precision, around 91.97%, represents the proportion of correctly predicted probe attacks among all instances classified as probe attacks. However, the recall, at about 62.44%, signifies the proportion of correctly identified probe attacks out of all actual probe

```
Probe
-----------
Accuracy: 88.69620% (+/- 0.92535%)
Precision: 91.97171% (+/- 3.25214%)
Recall: 62.44309% (+/- 2.68284%)
F-measure: 66.72476% (+/- 3.67110%)
```

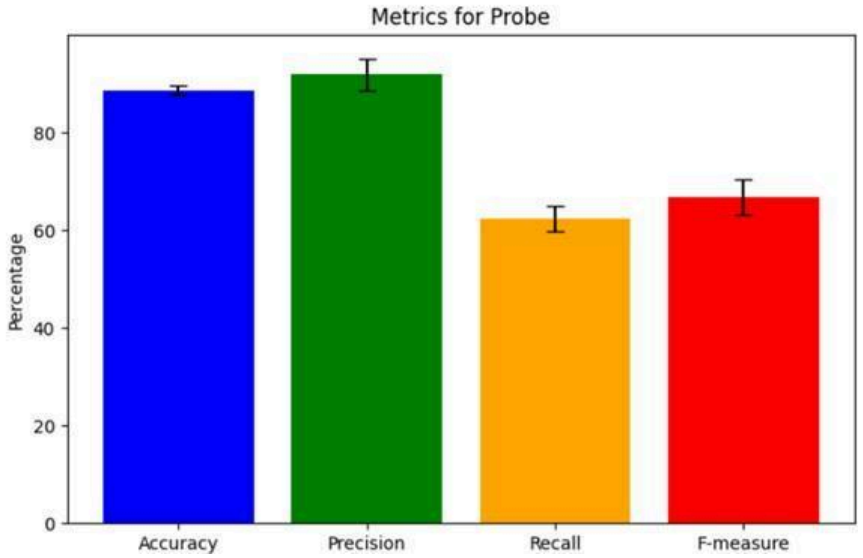


Figure 4.18: Bar graph: Gaussian Naive Bayes Classifier Probe metrics

```
U2R
-----------
Accuracy: 94.76966% (+/- 1.27817%)
Precision: 50.76790% (+/- 0.81313%)
Recall: 87.39328% (+/- 32.97281%)
F-measure: 50.17987% (+/- 1.65470%)
```

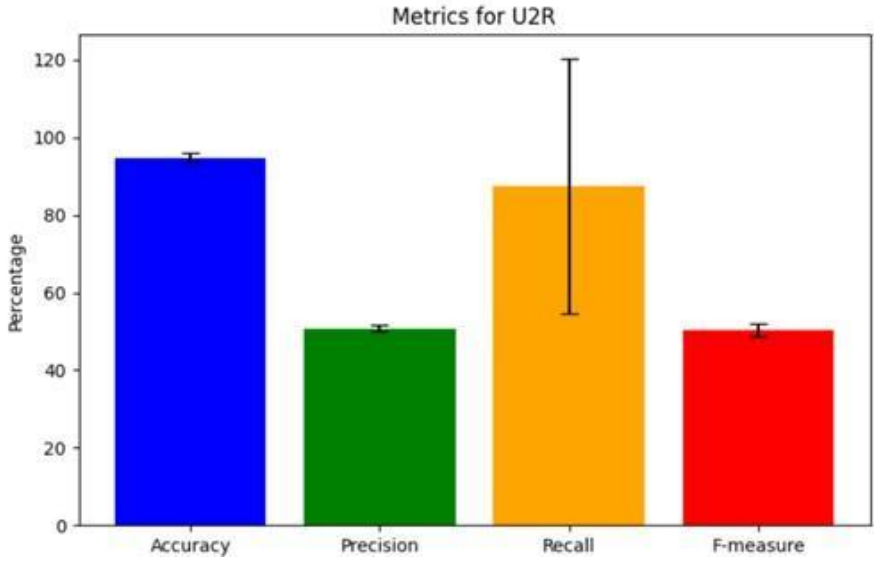


Figure 4.19: Bar graph: Gaussian Naive Bayes Classifier U2R metrics

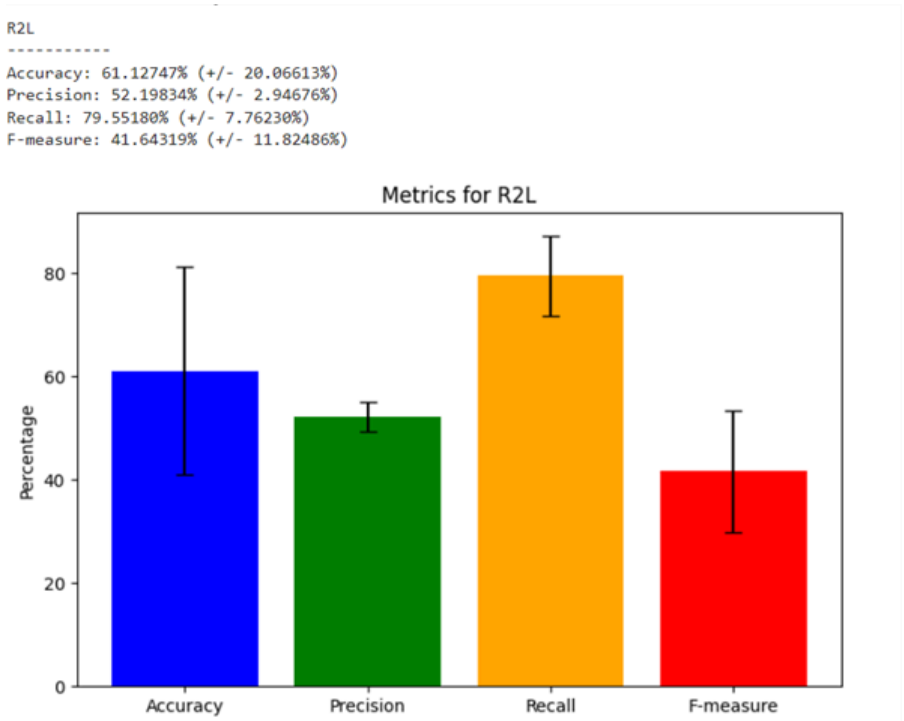


Figure 4.20: Bar graph: Gaussian Naive Bayes Classifier R2L metrics

attacks. The F-measure, approximately 66.72%, provides a balanced assessment of preci- sion and recall.

The metrics for Remote to Local (R2L) attack detection classifier's performance. The accuracy, approximately 61.13%, indicates the overall correctness of the classifier's pre- dictions. Precision, around 52.20%, represents the proportion of correctly predicted R2L attacks among all instances classified as R2L attacks. The recall, at about 79.55%, signifies the proportion of correctly identified R2L attacks out of all actual R2L attacks. However, the F-measure, approximately 41.64%, provides a balanced assessment of precision and recall.

The metrics provided for User to Root (U2R) attack detection evaluate the classifier's performance. The accuracy, approximately 94.77%, indicates the overall correctness of the classifier's predictions. Precision, around 50.77%, represents the proportion of cor- rectly predicted U2R attacks among all instances classified as U2R attacks. However, the recall, at about 87.39%, signifies the proportion of correctly identified U2R attacks out of all actual U2R attacks. The F-measure, approximately 50.18%, provides a balanced assessment of precision and recall.

4.1.7 Principal Component Analysis

```
DOS
-----------
Accuracy: 88.06408% (+/- 5.81618%)
Precision: 87.82685% (+/- 5.27003%)
Recall: 88.56410% (+/- 4.75541%)
F-measure: 87.88036% (+/- 5.69477%)
```

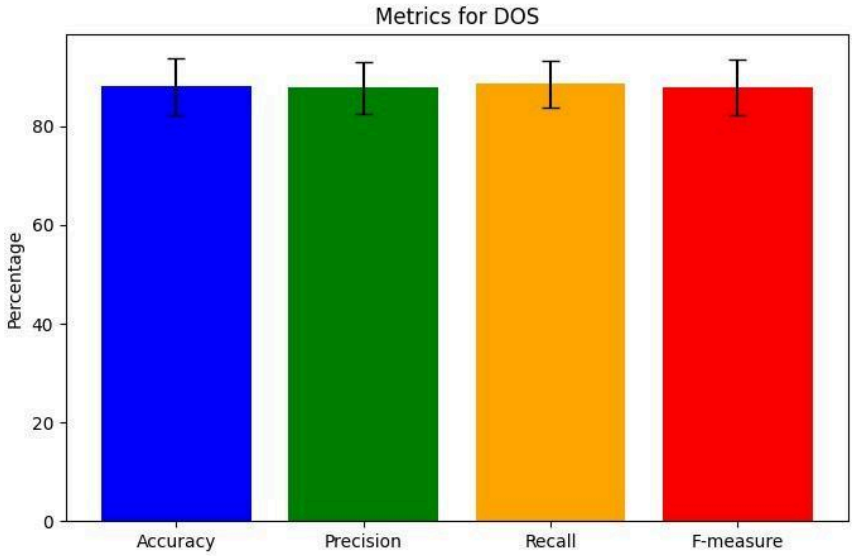


Figure 4.21: Bar graph: Principal Component Analysis DOS metrics

The Gaussian Naive Bayes classifier for DOS attacks, after applying PCA, achieved an accuracy of 88.06% (±5.82%), indicating that it correctly classified a majority of DOS attack instances. The precision was 87.83% (±5.27%), showing that most of the identi- fied DOS attacks were actual attacks, though there were some false positives. The recall was 88.56% (±4.76%), reflecting its ability to identify a majority of DOS attacks. The F-measure was 87.88% (±5.69%), indicating a reasonably balanced performance between precision and recall.

For Probe attacks, the Gaussian Naive Bayes classifier performed well after applying PCA. It achieved an accuracy of 93.65% (±1.03%), indicating a high level of correct clas- sifications. The precision was 86.52% (±2.34%), suggesting a good rate of true positive detection with some false positives. The recall was 89.71% (±1.96%), showing that the classifier was effective in identifying most Probe attacks. The F-measure was 87.99% (±1.76%), demonstrating a solid balance between precision and recall.

```
Probe
-----------
Accuracy: 93.64557% (+/- 1.03023%)
Precision: 86.52400% (+/- 2.34031%)
Recall: 89.71181% (+/- 1.96291%)
F-measure: 87.98717% (+/- 1.76120%)
```

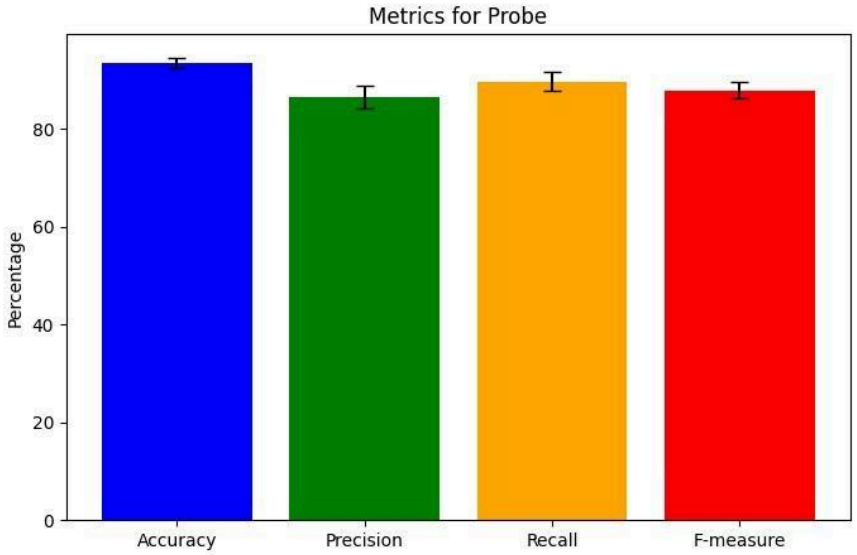


Figure 4.22: Bar graph: Principal Component Analysis Probe metrics

```
U2R
-----------
Accuracy: 98.56815% (+/- 1.11431%)
Precision: 51.74512% (+/- 3.51841%)
Recall: 74.30932% (+/- 44.65020%)
F-measure: 52.92605% (+/- 6.41788%)
```

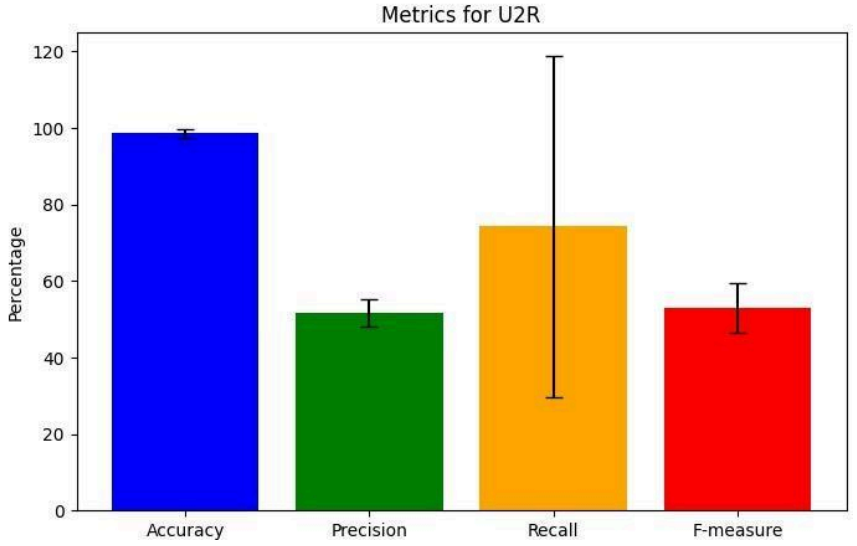


Figure 4.23: Bar graph: Principal Component Analysis U2R metrics

```
R2L
-----------
Accuracy: 94.98823% (+/- 0.85787%)
Precision: 55.38160% (+/- 2.44285%)
Recall: 65.65217% (+/- 6.04498%)
F-measure: 57.53316% (+/- 3.51135%)
```

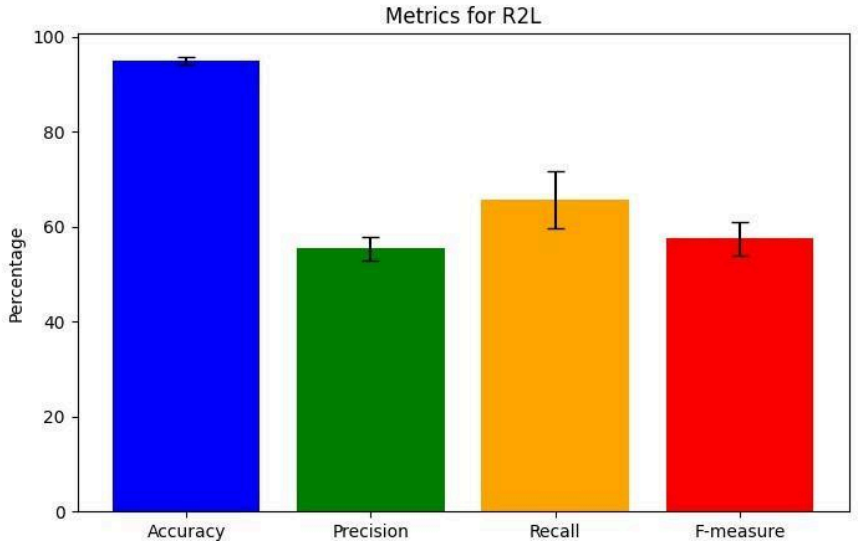


Figure 4.24: Bar graph: Principal Component Analysis R2L metrics

The performance for R2L attacks was lower after applying PCA with the Gaussian Naive Bayes classifier. The accuracy was 94.99% (±0.86%), indicating a high rate of cor- rect classifications. However, the precision was 55.38% (±2.44%), revealing challenges with a significant number of false positives. The recall was 65.65% (±6.04%), indicat- ing that many R2L attacks were missed (false negatives). The F-measure was 57.53% (±3.51%), showing difficulty in achieving a balanced performance between precision and recall.

For U2R attacks, the Gaussian Naive Bayes classifier showed mixed results after PCA. The accuracy was 98.57% (±1.11%), indicating it generally classified instances correctly. The precision was 51.75% (±3.52%), showing a substantial number of false positives. The recall was 74.31% (±44.65%), reflecting significant variability and many missed U2R attacks (false negatives). The F-measure was 52.93% (±6.42%), highlighting challenges in balancing precision and recall for U2R attack classification.

4.1.8 XGBoost Classifier

The XGBoost classifier for DOS attacks demonstrated exceptional performance. It achieved an accuracy of 99.95% (±0.16%), which means it correctly classified nearly all instances of DOS attacks. The precision was 99.95% (±0.14%), indicating that almost all identi- fied DOS attacks were actual DOS attacks, with very few false positives. The recall was 99.95% (±0.18%), showing that the classifier was able to identify nearly all actual DOS attack instances. The F-measure was also 99.95% (±0.16%), confirming a high level of balance between precision and recall.

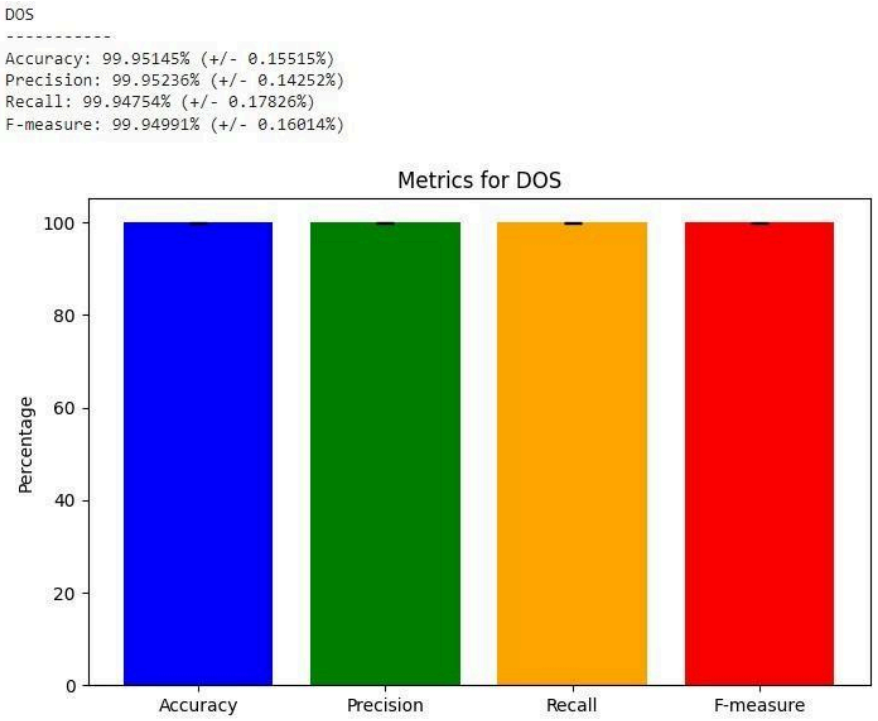


Figure 4.25: Bar graph: XGBoost Classifier DOS metrics

The XGBoost classifier for DOS attacks demonstrated exceptional performance. It achieved an accuracy of 99.95% (±0.16%), which means it correctly classified nearly all instances of DOS attacks. The precision was 99.95% (±0.14%), indicating that almost all identified DOS attacks were actual DOS attacks, with very few false positives. The recall was 99.95% (±0.18%), showing that the classifier was able to identify nearly all actual DOS attack instances. The F-measure was also 99.95% (±0.16%), confirming a high level of balance between precision and recall.

```
Probe
-----------
Accuracy: 99.84810% (+/- 0.23478%)
Precision: 99.75234% (+/- 0.45911%)
Recall: 99.64713% (+/- 0.52202%)
F-measure: 99.69945% (+/- 0.46454%)
```

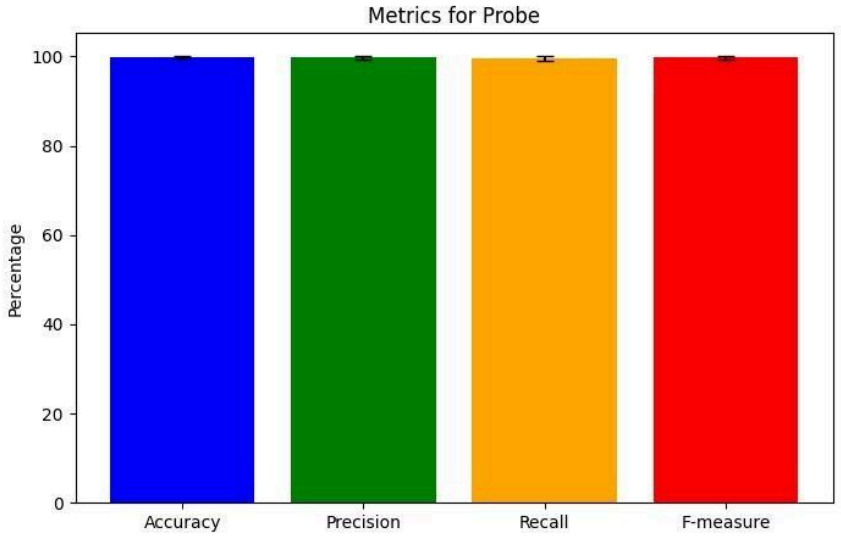


Figure 4.26: Bar graph: XGBoost Classifier Probe metrics

```
U2R
-----------
Accuracy: 99.90356% (+/- 0.13353%)
Precision: 64.96660% (+/- 40.03154%)
Recall: 64.98514% (+/- 40.00373%)
F-measure: 64.97587% (+/- 40.01764%)
```

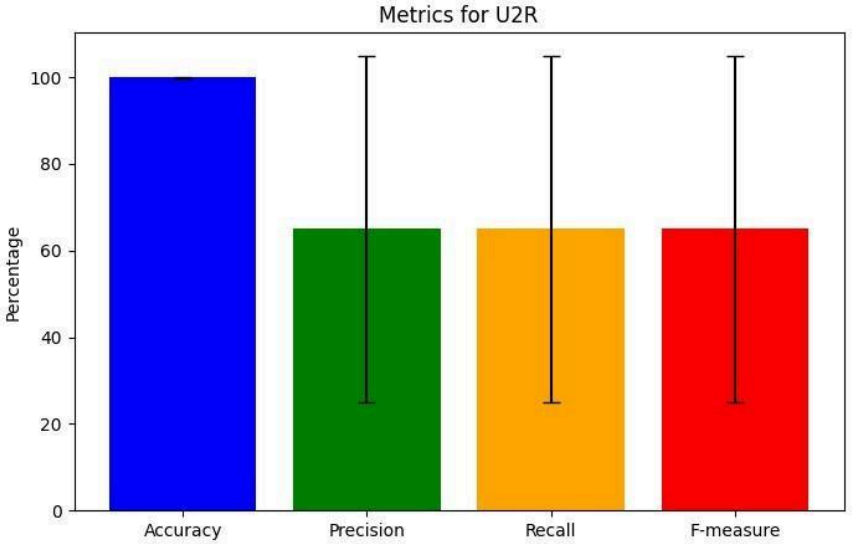


Figure 4.27: Bar graph: XGBoost Classifier U2R metrics

```
R2L
-----------
Accuracy: 99.89023% (+/- 0.18805%)
Precision: 98.95792% (+/- 3.35092%)
Recall: 97.30657% (+/- 4.08379%)
F-measure: 98.09928% (+/- 3.27361%)
```

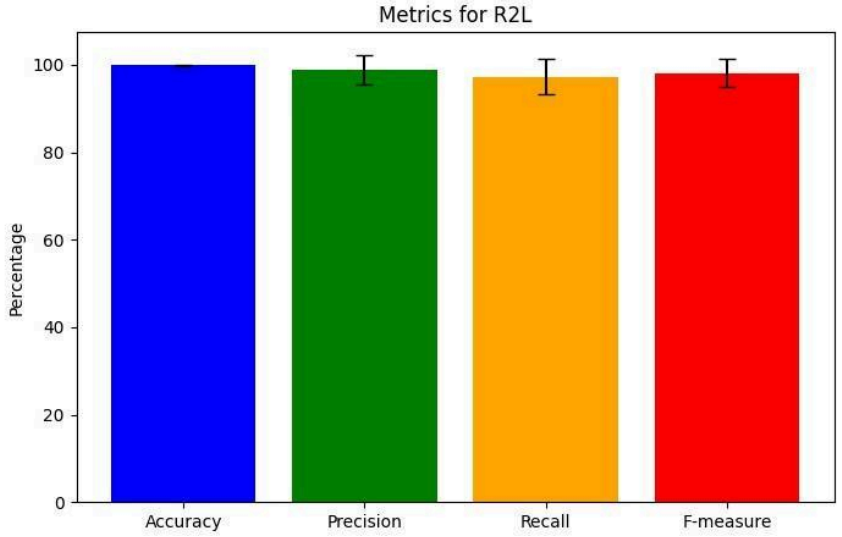


Figure 4.28: Bar graph: XGBoost Classifier R2L metrics

For Probe attacks, the XGBoost classifier also performed impressively. It recorded an accuracy of 99.85% (±0.23%), meaning it was very reliable in distinguishing between Probe attacks and normal traffic. The precision was 99.75% (±0.46%), indicating that the classifier was able to accurately identify true Probe attacks with very few false positives. The recall was 99.65% (±0.52%), reflecting its effectiveness in identifying nearly all Probe attacks. The F-measure was 99.70% (±0.46%), demonstrating strong overall performance and balance.

The performance for R2L attacks was slightly lower but still strong with the XGBoost classifier. It achieved an accuracy of 99.89% (±0.19%), indicating reliable classification overall. The precision was 98.96% (±3.35%), showing that the classifier could identify most true positives, though with some variability. The recall was 97.31% (±4.08%), sug- gesting it was effective at detecting most R2L attacks, though there were some false nega- tives. The F-measure was 98.10% (±3.27%), showing a good balance despite the inherent difficulty in identifying R2L attacks.

For U2R attacks, the XGBoost classifier had mixed results. The accuracy was 99.90%

(±0.13%), indicating it was generally reliable at classifying instances correctly. However, the precision was 64.97% (±40.03%), which shows significant challenges with high vari- ability and a substantial number of false positives. The recall was 64.99% (±40.00%), reflecting inconsistency in identifying true U2R attacks, with many false negatives. The F-measure was 64.98% (±40.02%), highlighting considerable variability and challenges in achieving reliable classification for U2R attacks.

4.2 Discussion

4.2.1 Performance Comparision

In the classifier initialization phase, a range of classifiers are chosen based on their diverse methodologies and widespread applicability in classification tasks. Each classifier serves a unique purpose in evaluating the dataset. First, the Random Forest classifier, an ensem- ble method, is selected for its ability to combine multiple decision trees to enhance overall performance and robustness. This approach is particularly effective for handling large datasets characterized by high dimensionality and complex feature interactions. Next, the Naive Bayes classifier, specifically GaussianNB, is included due to its probabilistic na- ture, relying on Bayes' theorem and assuming feature independence. Despite its simplic- ity, GaussianNB is highly efficient, especially when the assumption of feature indepen- dence roughly holds true. Additionally, the XGBoost classifier, known for its implemen- tation of gradient boosted decision trees, is incorporated for its speed and performance, often achieving state-of-the-art results in classification tasks. The Support Vector Machine (SVM) classifier is chosen for its ability to construct hyperplanes in high-dimensional spaces, facilitating clear margin separation between classes. This makes SVM particu- larly effective in scenarios with complex decision boundaries. Moreover, the K-Nearest Neighbors (KNN) classifier, a non-parametric instance-based learning algorithm, is in- cluded for its simplicity and effectiveness, particularly suitable for datasets with irregular decision boundaries. Finally, a second instance of the Random Forest classifier is initial- ized, leveraging its ensemble method to build multiple decision trees and produce accurate and stable predictions, thus reducing overfitting and improving the model's generalization capability.

Classifier	Accuracy	Precision	Recall	F-measure
KNN	98.67% ± 1.10%	93.63% ± 7.24%	88.71% ± 5.95%	90.36% ± 6.48%
SVM	97.85% ± 0.83%	95.36% ± 7.20%	82.91% ± 8.18%	86.26% ± 7.55%
Ensemble	99.35% ± 0.75%	97.60% ± 5.78%	94.89% ± 3.71%	95.56% ± 4.60%
Naive Bayes	81.33% ± 40.60%	83.55% ± 10.39%	94.22% ± 10.64%	82.83% ± 13.22%
XGBoost	99.37% ± 0.77%	97.70% ± 5.30%	95.16% ± 5.52%	95.94% ± 3.38%
Random Forest	99.33% ± 0.71%	97.62% ± 5.75%	94.59% ± 4.67%	96.04% ± 4.04%

Table 4.1: Performance Metrics of Classifiers

Model	Time Complexity
Random Forest	$O(N \log N)$
Naive Bayes	$O(n)$
XGBoost	$O(N \times M)$
Ensemble	Variable
KNN	$O(N \times K \times d)$

Table 4.2: Comparison of Models' Performance

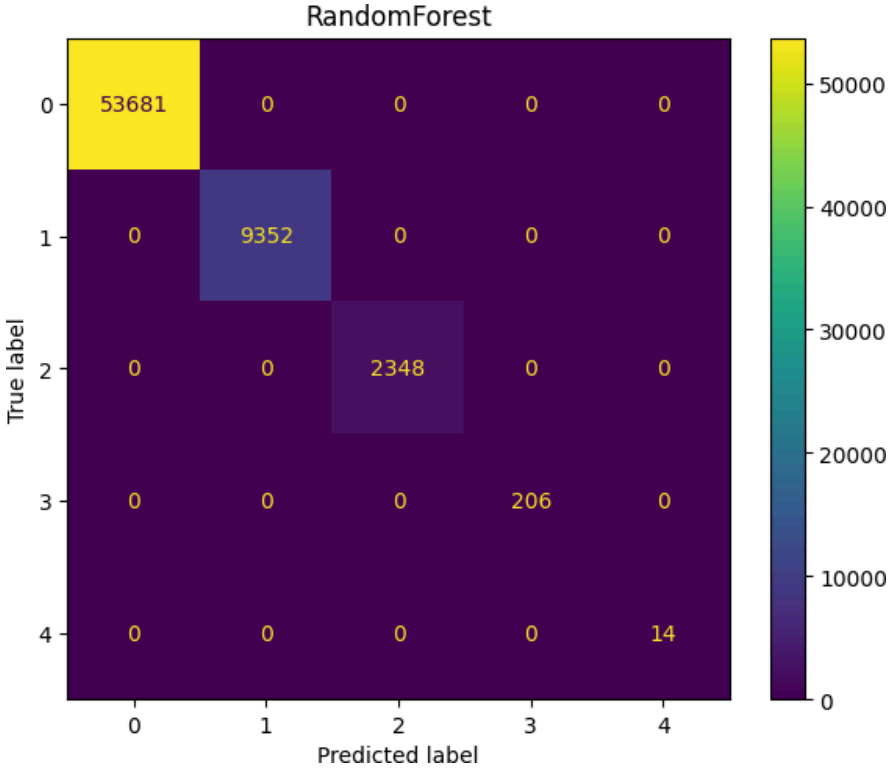


Figure 4.29: Confusion Matrix: Random Forest classifier

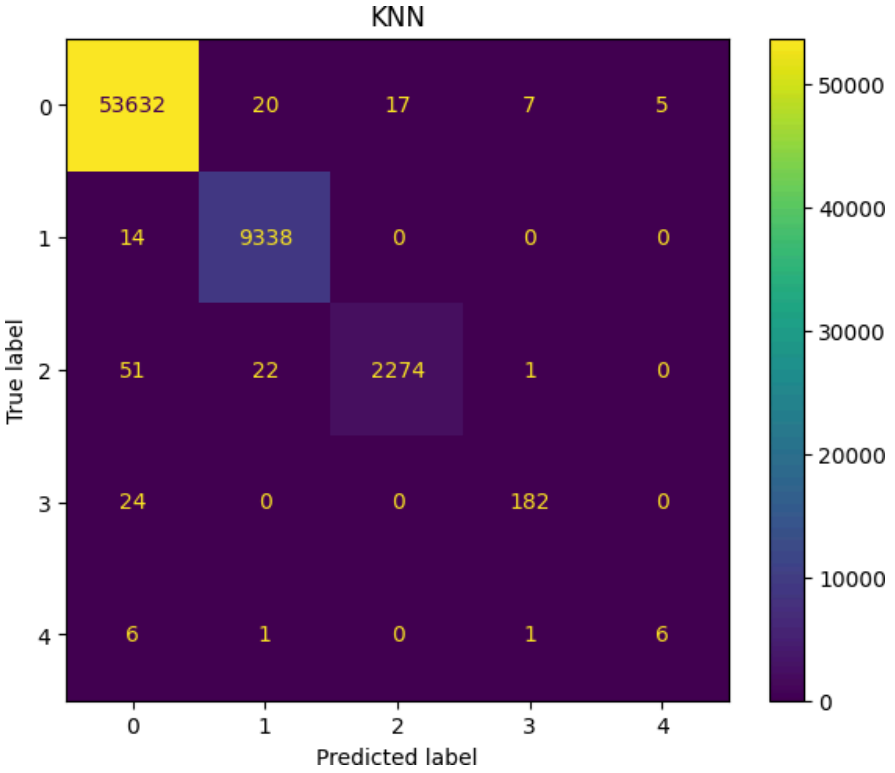


Figure 4.30: Confusion Matrix: K-Nearest Neighbors

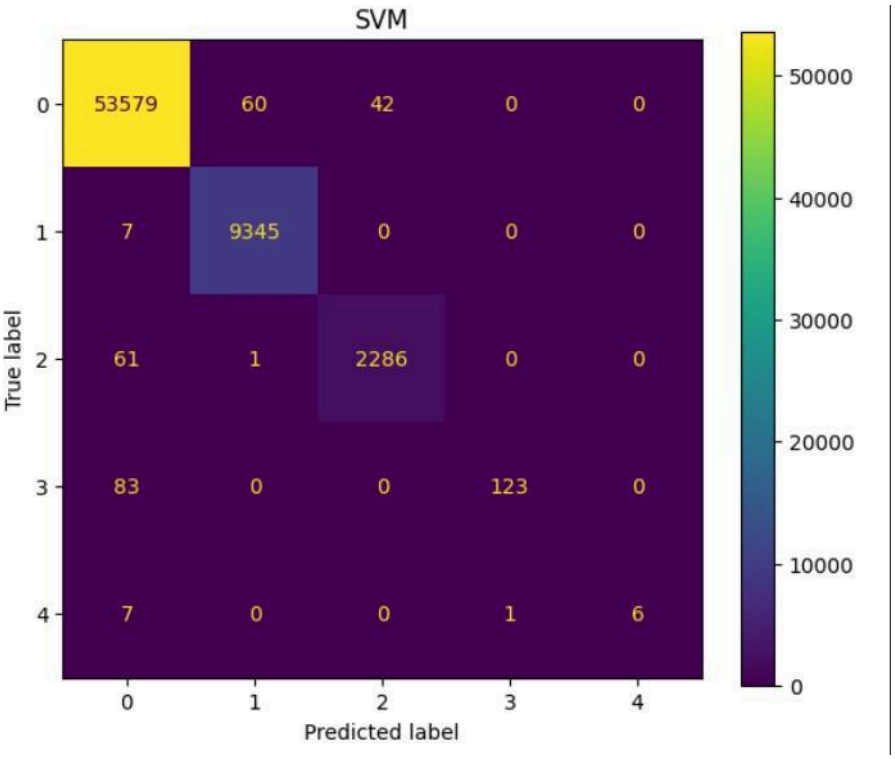


Figure 4.31: Confusion Matrix: Support Vector Machine

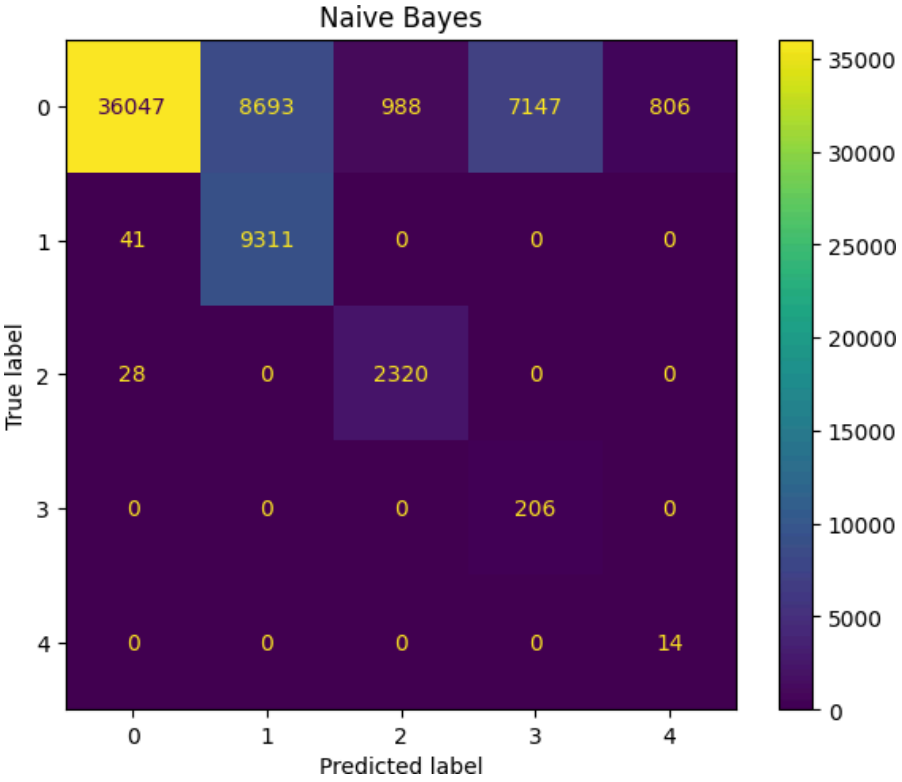


Figure 4.32: Confusion Matrix: Gaussian Naive Bayes Classifier

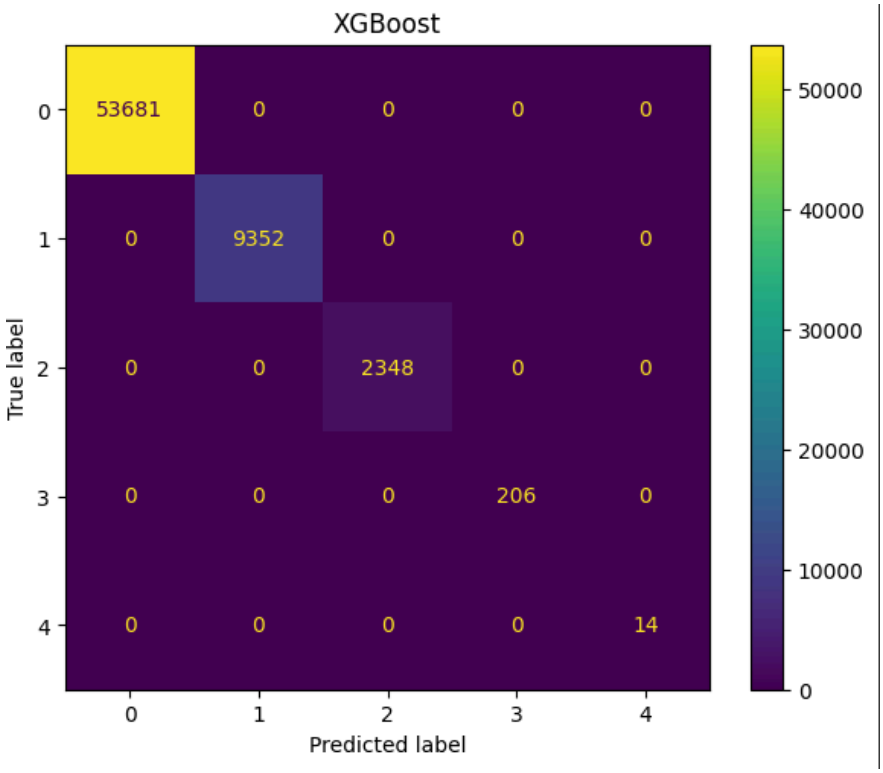


Figure 4.33: Confusion Matrix: XGBoost Classifier

4.3 Summary

Our evaluation of various machine learning classifiers reveals nuanced performance differ- ences across multiple metrics. Ensemble methods, XGBoost, and Random Forest consis- tently achieve high accuracy, precision, recall, and F-measure, making them top contenders for diverse classification tasks. SVM also demonstrates competitive performance across metrics, while KNN and Naive Bayes exhibit some limitations, particularly in precision and recall. These insights offer valuable guidance for selecting the most suitable classifier based on specific performance requirements and trade-offs in real-world applications.

Chapter 5

CONCLUSION

This study explored various machine learning methodologies, focusing on their imple- mentation and effectiveness in a comprehensive system architecture. The models exam- ined included Random Forest Classifier, K-nearest Neighbors, Support Vector Machine, Ensemble Learning Voting Classifier, Gaussian Naive Bayes Classifier, Principal Compo- nent Analysis, and XGBoost Classifier. Each model was evaluated for feature selection, performance, and suitability in different scenarios.

5.1 Summary of the Study

The research provided a detailed examination of multiple machine learning algorithms, assessing their strengths and weaknesses. Key aspects such as system architecture and feature selection were integral to understanding the functionality and efficiency of these models. The study aimed to compare the performance of these classifiers and determine the best practices for their application.

5.2 Main Findings

The analysis revealed that while all the classifiers had their unique advantages, the XG-Boost Classifier consistently outperformed the others in terms of accuracy and computational efficiency. Random Forest and Ensemble Learning Voting Classifier also showed strong performance, particularly in handling large datasets with complex feature interactions. Principal Component Analysis proved effective in dimensionality reduction, enhancing the performance of subsequent classifiers.

5.3 Contributions to Knowledge

This study contributes to the knowledge base by providing a comprehensive comparison of various machine learning classifiers and their practical applications. It highlights the importance of feature selection and dimensionality reduction in improving model perfor-

mance. The detailed evaluation of each model offers valuable insights into their applicability in different contexts, guiding future research and development in machine learning.

5.4 Practical Implications

The findings have significant practical implications for data scientists and engineers. By identifying the most effective classifiers and methodologies, this study aids in selecting the appropriate models for specific tasks, improving accuracy and efficiency. The insights on feature selection and dimensionality reduction are particularly useful for optimizing model performance in real-world applications.

5.5 Final Thoughts

Overall, this study underscores the critical role of methodological rigor in machine learn- ing. By thoroughly comparing various classifiers and examining their integration within a robust system architecture, it provides a solid foundation for future research and practical implementation. The emphasis on feature selection and dimensionality reduction further enhances the relevance and applicability of the findings, making significant contributions to both the academic and professional domains of machine learning.

BIBLIOGRAPHY

P. Chudasma. Network intrusion detection system using classification techniques in machine learning. *N/A*, pages 58–63, 2020.

D. J. a. Chunying Zhang a. Comparative research on network intrusion detection methods based on machine learning. *N/A*, 2022. URL https://www.sciencedirect.com/ science/article/abs/pii/S0167404822002553.

P. Dini. Overview on intrusion detection systems design exploiting machine learning for networking cybersecurity. *N/A*, 2023. URL https://www.mdpi.com/20763417/1-3/13/7507.

A. F. O. Emad E Abdallah, Wafa' Eleisah. Intrusion detection systems using supervised machine learning techniques: A survey. *Al-Hussein Bin Abdullah*, 2022. URL https://www.sciencedirect.com/science/article/pii/S1877050922004422.

H. A. Masama. An improved binary manta ray foraging optimization algorithm based feature selection and random forest classifier for network intrusion detection. *N/A*, 2022. URL https://www.sciencedire-ct.com/science/article/pii/S2667305322000527.

Dini [2023] Emad E Abdallah [2022] Masama [2022] Chunying Zhang a [2022] Chudasma [2020]