

Intrusion Detection System with Machine Learning by Arkar Hmue Htet



School of Computer Science and Mathematics

6500CSMM Project

Final Report Submitted by

Arkar Hmue Htet

PN - 1097197

Computer Network

Title

Intrusion Detection System with Machine Learning

Supervised by

Thet Zaw Aye

Submitted on

October 2024

Contents

Abstract.....	5
Acknowledgments	5
Chapter 1: Introduction	6
1.1 Background	6
1.2 Problem Statement.....	6
1.3 Objectives.....	7
1.4 Scope of the Project	7
Chapter 2: Literature Review	8
2.1 Introduction.....	8
2.2 Evolution of Intrusion Detection Systems.....	8
2.2.1 Signature-Based Detection	8
2.2.2 Anomaly-Based Detection	8
2.2.3 Machine Learning in IDS	8
2.3 Machine Learning Techniques for Intrusion Detection	9
2.3.1 Supervised Learning Methods	9
2.3.1.1 Random Forest	9
2.3.1.2 Support Vector Machine (SVM).....	10
2.3.1.3 Decision Tree	10
2.3.1.4 Naive Bayes.....	11
2.3.1.5 K-Nearest Neighbors (KNN)	11
2.3.1.6 XGBoost	12
2.3.2 Unsupervised Learning and Anomaly Detection	12
2.3.2.1 K-Means Clustering.....	12
2.3.2.2 Isolation Forest.....	13
2.3.3 Deep Learning Approaches.....	13
2.3.3.1 Convolutional Neural Networks (CNN)	13
2.3.3.2 Long Short-Term Memory (LSTM).....	13
2.4 Feature Selection and Dimensionality Reduction	13
2.4.1 Feature Selection Techniques	13
2.4.2 Impact on IDS Performance	14
2.5 NSL-KDD Dataset in IDS Research	14

2.5.1 Dataset Characteristics.....	14
2.5.2 Limitations of NSL-KDD.....	14
2.6 Evaluation Metrics for IDS	14
2.6.1 Accuracy, Precision, Recall, and F1-Score	14
2.6.2 Computational Efficiency.....	15
2.7 Real-Time Intrusion Detection Systems.....	15
2.7.1 Challenges in Real-Time IDS	15
2.7.2 Successful Implementations	15
2.8 Emerging Trends and Future Directions	15
2.8.1 AI and Hybrid Models.....	15
2.8.2 Big Data and IDS	15
2.9 Summary and Research Gaps.....	16
2.9.1 Key Findings.....	16
2.9.2 Research Gaps	16
2.9.3 Implications for Current Project.....	16
Chapter 3: Methodology.....	17
3.1 Data Preprocessing	17
3.2 Feature Selection Using Mutual Information.....	17
3.3 Machine Learning Models.....	18
3.4 Model Training and Evaluation.....	20
Chapter 4: System Design	21
4.1 Block Diagram of the Project.....	21
4.2 System Design Explanation	21
4.3 System Functionality	22
Chapter 5: Evaluation and Results.....	23
5.1 Data Preprocessing	23
5.2 Initial Model Evaluation Using All Features	25
5.2.1 Algorithms	25
5.2.2 Evaluation Methodology.....	26
5.2.3 Performance Metrics without Feature Selection.....	27
5.2.4 Confusion Matrices Before Feature Selection	29
5.3 Feature Selection and Re-Evaluation	32
5.3.1 Performance Metrics After Feature Selection.....	33

5.3.2 Confusion Matrices After Feature Selection	35
5.4 Analysis of Results and Real-Time Evaluation	37
5.4.1 Comparative Analysis.....	37
5.4.2 Impact of Feature Selection	37
5.4.3 Real-Time IDS Evaluation	38
5.4.4 Impact on Real-Time Intrusion Detection	41
5.5 Summary of Results	41
Chapter 6: Conclusion and Future Work.....	42
6.1 Conclusion	42
Future Work.....	42
6.2 Application of Deep Learning Models	43
6.3 Expanding the Dataset.....	43
6.4 Improving Computational Efficiency.....	44
6.5 Real-Time Implementation and Scalability	44
6.6 Adaptive Learning and Attack Evolution.....	44
Appendix: Project Management.....	45
Planned Project Timeline	45
Initial 14-Day Timeline.....	46
Milestones and Management.....	47
Monthly Reports	47
Month 1: Project Initiation and Literature Review (Weeks 1-5).....	48
Month 2: Data Collection and Preprocessing (Weeks 6-8).....	49
Month 3: Model Selection and Implementation (Weeks 9-13).....	50
Month 4: Evaluation, Testing, and Real-Time IDS Implementation (Weeks 14-15) And Documentation and Final Report Preparation (Weeks 16).....	51
References.....	52

Abstract

In an era where network security is increasingly under threat from cyberattacks, the ability to detect and mitigate intrusions has become a critical priority. Traditional Intrusion Detection Systems (IDS) often struggle to detect new or unknown threats due to their reliance on signature-based detection techniques, which can lead to high false-positive rates. This project focuses on developing and evaluating a machine learning-based IDS using the NSL-KDD dataset to address these challenges. Several machine learning algorithms, including Random Forest, Support Vector Machine (SVM), Decision Tree, Naive Bayes, K-Nearest Neighbors (KNN), and XGBoost, were implemented to classify network traffic as normal or malicious.

Feature selection was used to enhance the performance of the machine learning models by reducing data complexity. The machine learning algorithms were compared and evaluated based on key performance metrics, and the best-performing model was selected for real-time implementation. An IDS system was developed and integrated with a graphical user interface (GUI) to enable real-time monitoring of live network traffic.

This project contributes to the growing body of research on using machine learning for IDS and highlights the importance of optimizing models for intrusion detection.

Acknowledgments

I would like to express my deepest gratitude to my project supervisor, Thet Zaw Aye, for their invaluable guidance, support, and encouragement throughout the duration of this project. Their expert knowledge and insights were instrumental in shaping the direction of my research and in overcoming various challenges faced during the project.

I am also thankful to Auston University and Liverpool John Moores for providing the resources and facilities necessary for completing this project. The support from the faculty and staff, both academic and administrative, has been crucial to the successful execution of this work.

I would also like to extend my appreciation to my family and friends, whose unwavering support and encouragement have kept me motivated during difficult moments. Their understanding and patience have allowed me to stay focused and committed to this endeavor.

Lastly, I acknowledge the importance of open-source tools and communities, such as Scikit-learn, Matplotlib, and others, whose contributions have significantly facilitated the development and implementation of the machine learning models used in this project.

Chapter 1: Introduction

1.1 Background

In an increasingly digitized world, the reliance on computer networks for communication, data storage, and transactions has grown exponentially. As a result, the security of these networks has become a paramount concern for individuals, organizations, and governments alike. The proliferation of cyber threats, including malware, phishing, and denial-of-service attacks, has highlighted the need for effective security measures to protect sensitive data and ensure the continuity of operations. Traditional security mechanisms, such as firewalls and antivirus software, while essential, are often insufficient to combat the sophisticated and evolving nature of modern cyber-attacks.

Intrusion Detection Systems (IDS) have emerged as a critical component of network security infrastructure. An IDS monitors network traffic for suspicious activity and potential threats, providing real-time alerts and, in some cases, automated responses to mitigate risks. However, conventional IDS approaches, which primarily rely on predefined rules and signatures, struggle to detect new or unknown threats, often resulting in high false-positive rates. This limitation necessitates the exploration of more advanced techniques, such as machine learning, to enhance the detection capabilities of IDS.

Machine learning, a subset of artificial intelligence, involves the development of algorithms that enable systems to learn from data and improve their performance over time. In the context of IDS, machine learning can be employed to analyze network traffic patterns, identify anomalies, and predict potential intrusions. By leveraging large datasets and sophisticated algorithms, machine learning-based IDS can adapt to new threats and reduce false positives, providing a more robust defense against cyber-attacks.

1.2 Problem Statement

Despite the advancements in IDS technology, several challenges remain that hinder the effectiveness of these systems. Traditional IDS, which rely heavily on signature-based detection, are limited in their ability to recognize new or zero-day attacks that do not match any known patterns. Furthermore, the increasing volume and complexity of network traffic can overwhelm these systems, leading to delayed detection or missed threats. The high rate of false positives generated by these systems also poses a significant issue, as it can lead to alert fatigue, where legitimate warnings are overlooked by security personnel due to the frequent occurrence of false alarms.

Machine learning offers a promising solution to these challenges by enabling IDS to detect novel threats based on patterns and behaviors rather than relying solely on predefined signatures. However, the effectiveness of machine learning-based IDS is contingent on several factors, including the quality of the training data, the choice of features used for analysis, and the selection of appropriate algorithms. A key challenge in developing a machine learning-based

IDS is determining which features of the network traffic are most indicative of an intrusion and how to balance the trade-off between detection accuracy and computational efficiency.

1.3 Aims and Objectives

The primary objective of this project is to design, implement, and evaluate a machine learning-based IDS that can effectively detect network intrusions with high accuracy and low false-positive rates. The specific objectives of the project are as follows:

1. **To preprocess the NSL-KDD dataset:** This involves cleaning the data, handling missing values, and converting categorical variables into numerical form to prepare the dataset for machine learning model training.
 2. **To perform feature selection:** Identify the most relevant features from the dataset that contribute to accurate intrusion detection, thereby reducing the dimensionality of the data and improving the efficiency of the model.
 3. **To train and evaluate multiple machine learning algorithms:** Implement and compare the performance of various machine learning algorithms, including Random Forest, Decision Tree, SVM, KNN, Naive Bayes, and XGBoost, using both the full feature set and the selected feature set.
 4. **To analyze the results:** Assess the performance of the machine learning models using evaluation metrics such as accuracy, precision, recall, and F1-score, and compare the results before and after feature selection.
 5. **To develop a real-time intrusion detection system:** Implement a GUI-based application that integrates the trained machine learning model to monitor live network traffic and detect intrusions in real-time.
 6. **To discuss the challenges and limitations:** Reflect on the difficulties encountered during the project, the limitations of the current approach, and potential areas for future research and improvement.
-

1.4 Scope of the Project

This project aims to address these challenges by developing and evaluating a machine learning-based IDS using the NSL-KDD dataset, a widely-used benchmark dataset for intrusion detection research. The project will explore various machine learning algorithms, including Random Forest, Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Naive Bayes, and XGBoost, to determine their effectiveness in detecting intrusions. Additionally, the project will investigate the impact of feature selection on the performance of these algorithms and implement a real-time intrusion detection system with a graphical user interface (GUI) for live network monitoring.

Chapter 2: Literature Review

2.1 Introduction

Intrusion Detection Systems (IDS) are critical components in ensuring network security by identifying malicious activities or policy violations. Over the years, machine learning (ML) has revolutionized IDS by improving the ability to detect novel attacks, offering adaptive learning mechanisms and reducing human intervention. This literature review examines the evolution of IDS, explores various machine learning techniques used in IDS, discusses the feature selection process, evaluates the use of NSL-KDD dataset, and addresses emerging trends in the field. By highlighting key research and identifying gaps, the review sets the stage for the subsequent analysis and evaluation of machine learning based IDS used in this study.

2.2 Evolution of Intrusion Detection Systems

Intrusion Detection Systems have evolved significantly since their inception. Traditional IDS typically relied on signature-based detection, which matches known attack patterns to signatures stored in a database. While effective against previously known threats, signature-based IDS fails to detect new or modified attacks, known as zero-day attacks (Buczak & Guven, 2015).

2.2.1 Signature-Based Detection

The early versions of IDS, including the well-known Snort IDS, relied heavily on signature-based detection (Moustafa & Slay, 2015). Signature-based systems excel at identifying known threats but require constant updates to remain effective. Additionally, they struggle with the increasing volume of network traffic and the complexity of modern attacks (Buczak & Guven, 2015).

2.2.2 Anomaly-Based Detection

To address the limitations of signature-based systems, anomaly-based detection was introduced. These systems model normal network behavior and flag deviations as potential intrusions (Di Pietro & Mancini, 2008). However, a major drawback of anomaly-based systems is the high rate of false positives due to the broad spectrum of what constitutes "normal" behavior (Buczak & Guven, 2015).

2.2.3 Machine Learning in IDS

The integration of machine learning into IDS marked a paradigm shift. Machine learning algorithms, such as Random Forests, Support Vector Machines (SVM), and Deep Learning techniques, have enabled IDS to identify previously unseen attack patterns and adapt over time (Vinayakumar et al., 2017). By using vast amounts of historical attack data, machine learning-

based IDS can learn from experience and improve their detection capabilities (Singh et al., 2014).

2.3 Machine Learning Techniques for Intrusion Detection

Machine learning algorithms have been widely applied to improve intrusion detection systems. Supervised learning, unsupervised learning, and deep learning techniques have all shown potential in detecting cyber-attacks with varying levels of success and computational efficiency.

2.3.1 Supervised Learning Methods

Supervised learning algorithms are widely used in IDS for their ability to classify network traffic based on labeled datasets. These algorithms learn from historical data where the nature of the network activity (normal or malicious) is known. The trained model can then be used to classify new network traffic and detect potential intrusions. In this project, six different supervised learning algorithms were employed: **Random Forest (RF)**, **Decision Tree (DT)**, **Support Vector Machine (SVM)**, **K-Nearest Neighbors (KNN)**, **Naive Bayes**, and **XGBoost**. Each of these algorithms has unique strengths and weaknesses, making them suitable for various types of classification tasks.

2.3.1.1 Random Forest

Random Forest, an ensemble learning technique, is widely recognized for its high accuracy and robustness in IDS applications (Singh et al., 2014). It works by constructing multiple decision trees during the training process, and the final classification decision is based on the majority vote from all trees, or the mean prediction in regression cases (Chio & Freeman, 2018). This method of aggregating decisions from various weak learners improves the overall model performance, reduces overfitting, and increases generalization accuracy.

Explanation of Random Forest in the context of IDS

In the context of Intrusion Detection Systems (IDS), Random Forest is highly effective due to its ability to handle large and complex datasets, such as the NSL-KDD dataset, which contains multiple features and varied attack types. It excels in both misuse detection (signature-based) and anomaly detection by learning patterns from historical data and classifying network traffic as either normal or malicious (Vinayakumar et al., 2017). Random Forest's inherent feature selection process also contributes to its efficiency, as it selects the most relevant features during tree construction, reducing noise in the data.

Prior studies on Random Forest performance in IDS applications

Numerous studies have demonstrated the superior performance of Random Forest in IDS applications. For example, Belavagi and Muniyal (2016) conducted a comparative study and found that Random Forest outperformed other algorithms like Decision Trees and SVM in terms

of accuracy and false positive rates when applied to the NSL-KDD dataset. With a reported accuracy of over 99%, Random Forest proved to be a reliable model for detecting various types of intrusions, including Denial of Service (DoS) and Probe attacks (Singh et al., 2014).

2.3.1.2 Support Vector Machine (SVM)

Support Vector Machines (SVM) are another highly regarded supervised learning algorithm used in IDS. SVM works by identifying the optimal hyperplane that best separates data points from different classes (Buczak & Guven, 2015). In the context of IDS, this separation is essential for distinguishing between normal network activity and various types of intrusions.

Explanation of SVM in IDS

SVM is particularly useful for binary classification tasks, such as distinguishing between attack and normal traffic, and it is adept at handling high-dimensional data. The algorithm works by maximizing the margin between data points of different classes, which minimizes classification errors (Buczak & Guven, 2015). This is crucial in IDS, where the system needs to make clear distinctions between malicious and benign network traffic.

Strengths and weaknesses in IDS performance

Although SVM can achieve high accuracy in intrusion detection tasks, its major drawback is computational complexity, especially when dealing with large datasets like NSL-KDD. The training time increases significantly as the dataset size grows, making it less practical for real-time IDS deployment. Additionally, SVM often struggles with imbalanced datasets, which is common in IDS scenarios where normal traffic far outweighs intrusion instances. Despite these limitations, SVM still performs well in controlled environments and smaller datasets (Singh et al., 2014).

2.3.1.3 Decision Tree

Decision Trees are one of the most intuitive and widely used machine learning models in IDS applications. They classify data by recursively splitting the dataset based on the values of specific features, eventually reaching a decision node that represents the predicted class (Chio & Freeman, 2018). The tree-like structure of Decision Trees makes them easy to interpret, which is a crucial advantage in understanding the decision-making process of an IDS.

Explanation of Decision Tree in IDS

In IDS, Decision Trees are particularly effective in misuse detection, where attack patterns are predefined. The tree splits data into subsets based on the most informative features (using criteria like Information Gain or Gini Index), allowing it to classify network traffic as normal or malicious (Kumar et al., 2012). This method works well for detecting known attack types, as the algorithm can easily map specific traffic behaviors to intrusion classes.

Performance in detecting different types of attacks

Studies have demonstrated that Decision Trees perform exceptionally well in detecting certain types of intrusions. For instance, in their analysis, Ashwini Pathak and Sakshi Pathak (2020) showed that Decision Trees achieved 99.6% accuracy in detecting Denial of Service (DoS) attacks and 99.5% accuracy in detecting Probe attacks when applied to the NSL-KDD dataset. However, one of the main challenges with Decision Trees is overfitting, especially in noisy or imbalanced datasets. Pruning techniques are often employed to mitigate this issue and enhance generalization to unseen data (Kumar et al., 2012).

2.3.1.4 Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' Theorem, which assumes that features are independent given the class label. While this assumption rarely holds in real-world data, Naive Bayes is still effective in certain IDS applications due to its simplicity and computational efficiency.

Explanation of Naive Bayes in IDS

Naive Bayes is particularly well-suited for real-time IDS deployment due to its low computational cost. It can quickly classify network traffic based on probabilities derived from the training data. Despite its simplistic assumptions, Naive Bayes works surprisingly well for certain types of intrusion detection tasks, especially when the dataset has a clear distinction between classes (Altwaijry & Algarny, 2012).

Performance with imbalanced datasets

However, Naive Bayes can struggle with imbalanced datasets, a common issue in IDS where normal traffic dominates attack traffic. The model's performance tends to degrade when the prior probabilities of the classes are skewed, leading to a higher number of false positives or false negatives. Still, Naive Bayes has been effective in detecting specific attack types like Remote-to-Local (R2L) and User-to-Root (U2R) when applied to datasets like NSL-KDD (Li et al., 2012).

2.3.1.5 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, instance-based learning algorithm used for anomaly detection in IDS. It classifies new data points by finding the k most similar instances from the training data and assigning the most common class among them.

Explanation of KNN in IDS

In the context of IDS, KNN works by comparing new network traffic data to historical records of both normal and malicious traffic. By calculating the distances (usually Euclidean) between

instances, KNN can identify whether the new traffic is more similar to benign or malicious behaviors.

Computational efficiency and scalability

KNN is known for its high precision, particularly in detecting attack types like Probe and U2R. However, its computational complexity increases as the dataset grows, making it less suitable for real-time IDS applications. Ashwini Pathak and Sakshi Pathak (2020) found that while KNN achieved a high accuracy rate of 99.7% in detecting DoS attacks, its scalability and computational cost were significant concerns when applied to large datasets like NSL-KDD (Liao & Vemuri, 2002).

2.3.1.6 XGBoost

XGBoost is an ensemble learning method based on boosting, which combines multiple weak learners to form a strong classifier. In IDS, XGBoost has gained attention for its high accuracy and efficiency, especially when applied to imbalanced datasets.

Explanation of XGBoost in IDS

XGBoost is known for its ability to optimize both the speed and accuracy of classification tasks, making it highly suitable for IDS. By iteratively correcting the errors of previous models, XGBoost builds a model that can detect even subtle attack patterns in network traffic (Li et al., 2017).

Prior success in enhancing IDS accuracy

XGBoost has been shown to outperform traditional machine learning algorithms like Decision Trees and Random Forest in several IDS studies. Li et al. (2017) demonstrated that XGBoost, when applied to the NSL-KDD dataset, achieved superior accuracy and F1-Score compared to other algorithms, making it a strong candidate for real-time IDS implementations.

2.3.2 Unsupervised Learning and Anomaly Detection

Unsupervised learning methods, such as K-Means clustering and Isolation Forest, do not rely on labeled data. They are particularly useful in detecting new or previously unseen attacks (Di Pietro & Mancini, 2008).

2.3.2.1 K-Means Clustering

K-Means clustering groups data points based on similarities in their features, making it useful for identifying outliers or anomalies in network traffic (Buczak & Guven, 2015). However, K-Means struggles with scalability when applied to large datasets like NSL-KDD and often requires additional preprocessing steps.

2.3.2.2 Isolation Forest

Isolation Forest is a more recent unsupervised algorithm used for anomaly detection in IDS. It isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. The logic behind Isolation Forest is that anomalies are few and different from normal points, thus they are easier to isolate (Buczak & Guven, 2015).

2.3.3 Deep Learning Approaches

Deep learning, particularly Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM), has gained traction in IDS due to its ability to handle complex feature spaces and large datasets (Vinayakumar et al., 2017).

2.3.3.1 Convolutional Neural Networks (CNN)

CNNs have been widely used for feature extraction in IDS due to their ability to capture spatial hierarchies in data. CNN-based IDS can learn intricate patterns in network traffic and detect anomalies more efficiently than traditional machine learning models (Vinayakumar et al., 2017).

2.3.3.2 Long Short-Term Memory (LSTM)

LSTM networks, a type of Recurrent Neural Network (RNN), are particularly effective for sequence prediction tasks, making them suitable for detecting intrusions in time-series network traffic data (Buczak & Guven, 2015). LSTM networks can retain information over long periods, making them effective for identifying attacks that occur over extended time intervals.

2.4 Feature Selection and Dimensionality Reduction

Feature selection plays a critical role in optimizing machine learning models for IDS by reducing the complexity of the dataset and improving model efficiency.

2.4.1 Feature Selection Techniques

Several feature selection techniques have been applied in IDS research, including correlation-based feature selection and information gain. These methods help reduce the number of irrelevant or redundant features in the dataset, thus improving model accuracy and training time (Lakhina et al., 2015).

2.4.2 Impact on IDS Performance

Feature selection has been shown to improve the performance of machine learning models significantly. For instance, reducing the number of features in the NSL-KDD dataset from 41 to the top features led to a noticeable reduction in training and testing time without sacrificing accuracy (Vinayakumar et al., 2017). Additionally, it helps prevent overfitting, especially in high-dimensional datasets.

2.5 NSL-KDD Dataset in IDS Research

The NSL-KDD dataset is one of the most widely used datasets for evaluating IDS performance. It improves upon the original KDD Cup 99 dataset by addressing some of its known issues, such as redundancy and bias toward certain types of attacks (Tavallae et al., 2009).

2.5.1 Dataset Characteristics

The NSL-KDD dataset contains 41 features and five different classes of network connections: normal, DoS, Probe, R2L, and U2R. Each connection is labeled as either normal or one of the attack types, making it suitable for supervised machine learning models (Moustafa & Slay, 2015).

2.5.2 Limitations of NSL-KDD

Despite its improvements, the NSL-KDD dataset has been criticized for not accurately representing modern network traffic and attack patterns (Vinayakumar et al., 2017). Additionally, it contains imbalances between the number of normal and attack connections, which can affect model performance by leading to bias toward the majority class (Tavallae et al., 2009).

2.6 Evaluation Metrics for IDS

Evaluating the performance of IDS models requires a comprehensive set of metrics to assess their ability to detect intrusions accurately while minimizing false alarms.

2.6.1 Accuracy, Precision, Recall, and F1-Score

Accuracy is the most common metric used to evaluate IDS models. However, due to class imbalances in datasets like NSL-KDD, accuracy alone is not sufficient. Precision, recall, and F1-score provide a more balanced view of model performance by considering both false positives and false negatives (Buczak & Guven, 2015).

2.6.2 Computational Efficiency

In real-time IDS, training and testing time are critical metrics. Models that take too long to train or test may not be suitable for real-time applications. Random Forest, for example, has been shown to have relatively short testing times compared to other models like SVM (Singh et al., 2014).

2.7 Real-Time Intrusion Detection Systems

Deploying machine learning-based IDS in real-time environments poses several challenges, including the need for fast processing speeds and the ability to handle high-velocity data streams.

2.7.1 Challenges in Real-Time IDS

Real-time IDS must process network traffic as it happens, which requires models that can make quick predictions without sacrificing accuracy. One challenge is balancing the computational complexity of machine learning models with the need for fast response times (Vinayakumar et al., 2017).

2.7.2 Successful Implementations

Several studies have demonstrated the feasibility of real-time IDS using supervised machine learning and deep learning techniques. These models have shown high accuracy in detecting both known and unknown attacks while maintaining low false positive rates (Singh et al., 2014).

2.8 Emerging Trends and Future Directions

The future of IDS research lies in the integration of advanced machine learning techniques and big data frameworks to handle the ever-increasing volume and complexity of network traffic.

2.8.1 AI and Hybrid Models

The integration of artificial intelligence and hybrid models that combine machine learning with other techniques, such as fuzzy logic or genetic algorithms, is an emerging trend in IDS research (Berman et al., 2019). These models aim to improve the adaptability and resilience of IDS by leveraging multiple sources of intelligence.

2.8.2 Big Data and IDS

As network traffic grows in both size and complexity, the use of big data frameworks like Hadoop and Spark is becoming increasingly important for managing and analyzing network data in real-time (Buczak & Guven, 2015). By integrating big data with machine learning models, IDS can scale to meet the demands of modern networks.

2.9 Summary and Research Gaps

2.9.1 Key Findings

The literature shows that machine learning (supervised learning, unsupervised learning and deep learning) has significantly improved the effectiveness of IDS. Feature selection and dimensionality reduction techniques have been shown to enhance model performance by reducing computational complexity without sacrificing accuracy.

2.9.2 Research Gaps

While many machine learning models have been tested on the NSL-KDD dataset, the lack of modern datasets representing current network traffic and attack patterns remains a significant gap in IDS research. Additionally, there is a need for more research on the deployment of machine learning models in real-time IDS environments, particularly in terms of scalability and computational efficiency.

2.9.3 Implications for Current Project

The findings from this literature review inform the design and evaluation of the current IDS project, which utilizes supervised machine learning models trained on the NSL-KDD dataset. By applying feature selection techniques and evaluating the models based on both accuracy and computational efficiency, this project aims to contribute to the growing body of research on machine learning-based IDS.

Chapter 3: Methodology

This chapter outlines the step-by-step process that will be followed to develop, train, and evaluate the Intrusion Detection System (IDS) using machine learning algorithms on the NSL-KDD dataset. It details the planned approach for data preprocessing, feature selection using mutual information, and the use of machine learning models for training and evaluation.

3.1 Data Preprocessing

Preprocessing the dataset will be a crucial step in preparing the data for effective machine learning. The following steps will be implemented:

- **Handling Missing Data:** The NSL-KDD dataset, although containing minimal missing data, will be checked for any rows with incomplete or erroneous data. These rows will be removed to ensure the integrity of the dataset and avoid introducing any biases into the models.
 - **Categorical to Numerical Transformation:** Certain features, such as `protocol_type`, `service`, and `flag`, are categorical. These variables will need to be transformed into numerical representations using **one-hot encoding**, which will create binary columns for each unique category. This transformation will enable machine learning models to interpret the categorical data correctly.
 - **Normalization:** To ensure that no feature with larger values dominates the training process, **Min-Max normalization** will be applied to all numerical features. This normalization process will rescale the feature values to fall within a range of 0 and 1, ensuring that the machine learning models train efficiently and without bias from different feature scales.
-

3.2 Feature Selection Using Mutual Information

To improve the efficiency and accuracy of the machine learning models, feature selection will be applied to reduce the dimensionality of the dataset. In this project, **Mutual Information** will be used for feature selection.

- **Mutual Information** quantifies the amount of information a feature provides about the target variable (in this case, whether the network traffic is normal or malicious). This method will be used to rank the features based on their relevance in predicting intrusions.
 - After applying the mutual information technique, the top 30 most relevant features will be selected for model training. This step will help to eliminate irrelevant or redundant features, improving the computational efficiency of the models and reducing the risk of overfitting.
-

3.3 Machine Learning Models

This project will implement six machine learning models to classify network traffic as either normal or malicious. Each model has a unique approach to learning from data and making predictions. Below is a detailed explanation of how each model functions.

Random Forest

Random Forest is an ensemble learning algorithm that constructs multiple decision trees during the training phase. Each tree is trained on a random subset of the data (both features and instances). The randomness helps introduce diversity among the trees, reducing overfitting. When a new instance of network traffic is presented, each tree independently classifies it as either normal or malicious. The final classification is determined by majority voting, where the class predicted by the most trees becomes the final prediction. This method enhances both accuracy and robustness.

For example, in intrusion detection, if the majority of trees classify the traffic as malicious, the overall prediction will be malicious. The model is especially effective in handling large datasets and capturing complex interactions between features.

Support Vector Machine (SVM)

SVM works by finding the optimal hyperplane that separates two classes (normal and malicious traffic) in the feature space. The algorithm maximizes the margin between the two classes, where the margin is the distance between the hyperplane and the nearest data points from each class (called support vectors). If the data is not linearly separable, SVM applies a kernel function to transform the data into a higher-dimensional space where a hyperplane can be found to separate the classes.

In network intrusion detection, SVM classifies traffic based on its position relative to the hyperplane. If the traffic falls on one side of the hyperplane, it is classified as normal; if it falls on the other, it is classified as malicious.

Decision Tree

A Decision Tree model splits the dataset recursively into branches based on feature values. At each decision node, the algorithm selects the feature that best separates the data according to a specific criterion, such as information gain or Gini impurity. The dataset is split until it reaches leaf nodes, which represent the final classification (either normal or malicious traffic).

The tree-like structure makes it easy to interpret, as each path from the root to a leaf represents a set of decisions that lead to a particular classification. For instance, in intrusion detection, the tree may first evaluate whether a specific protocol is used, then check other attributes of the traffic, until a final decision is made about whether it is malicious or not.

Naive Bayes

Naive Bayes is a probabilistic classifier that applies Bayes' Theorem to estimate the probability of a class given the input features. The algorithm assumes that all features are conditionally independent given the class label, which is why it is called "naive." Despite this assumption, Naive Bayes is highly effective in many cases. For each instance of network traffic, Naive Bayes calculates the probability of the traffic being normal and the probability of it being malicious. The class with the higher probability is chosen as the prediction.

For example, Naive Bayes calculates how likely a set of features (such as protocol type, service, etc.) corresponds to either normal or malicious traffic, and predicts the class with the highest probability.

K-Nearest Neighbors (KNN)

KNN is a non-parametric algorithm that classifies data points by looking at the k-nearest neighbors in the feature space. It assigns the majority class of the nearest neighbors to the new data point. The algorithm calculates the distance between the new instance and all instances in the training set, then finds the k instances that are closest to the new one. The class that occurs most frequently among these neighbors is assigned to the new instance.

In the context of network traffic classification, KNN will classify a new instance based on how similar it is to the k nearest data points in the training data. For instance, if most of the nearest neighbors are classified as malicious, the new instance will be classified as malicious as well.

XGBoost

XGBoost is a gradient boosting algorithm that builds decision trees sequentially, where each new tree focuses on correcting the errors made by the previous trees. XGBoost uses boosting to combine the output of several weak learners (small decision trees) to create a strong classifier. The algorithm applies a weighted approach, where the model gives more importance to misclassified instances and iteratively improves its performance.

In intrusion detection, XGBoost takes several small decision trees, each specializing in different aspects of the data. The final output is a combined result from all the trees, where the errors

made by previous trees are corrected by subsequent ones, leading to a more accurate classification of whether the traffic is normal or malicious.

By utilizing these models, the project will cover a diverse range of machine learning techniques, from simple decision-based algorithms to more advanced ensemble methods, ensuring a comprehensive evaluation of the most suitable model for intrusion detection.

3.4 Model Training and Evaluation

The machine learning models will be trained using an 80/20 split of the NSL-KDD dataset, where 80% of the data will be used for training and 20% will be reserved for testing. The models will be evaluated based on the following metrics:

- **Accuracy:** This metric will measure the percentage of correct predictions made by the model.
- **Precision:** Precision will be calculated as the proportion of correctly predicted positive instances out of all positive predictions (i.e., how many predicted intrusions were actual intrusions).
- **Recall:** This metric will measure the proportion of true positive instances out of all actual positive instances (i.e., how many actual intrusions were detected by the model).
- **F1-Score:** The F1-Score will serve as the harmonic mean of precision and recall, providing a balanced view of the model's performance in terms of both false positives and false negatives.
- **Training and Testing Time:** The computational efficiency of each model will be assessed by measuring the time required to train and test the model. This will help determine whether a model is suitable for real-time deployment.

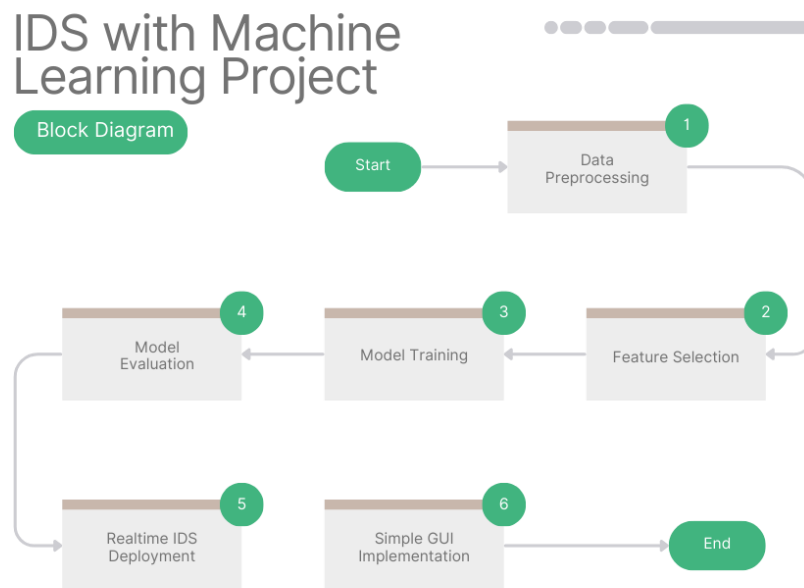
Once the models have been trained and evaluated using the full feature set, the process will be repeated after feature selection. The results from both phases (before and after feature selection) will be compared to determine which model offers the best balance between accuracy, precision, recall, and computational efficiency. The best-performing model will be selected for deployment in a real-time Intrusion Detection System.

Chapter 4: System Design

This chapter describes the overall system design for the Intrusion Detection System (IDS), including the architecture and flow of data through the system. The design is structured to allow efficient training, testing, and real-time monitoring.

4.1 Block Diagram of the Project

The block diagram illustrates the steps involved in the development and deployment of the IDS, from data preprocessing to real-time traffic monitoring. The following flowchart outlines the process:



Step by Step Block Diagram of the Project

4.2 System Design Explanation

- **Data Preprocessing:** The first stage involves cleaning and normalizing the dataset. Categorical features are converted to numerical values, and normalization ensures that all

features are on a similar scale. This prepares the dataset for training machine learning models.

- **Feature Selection (Mutual Information):** To optimize performance, mutual information is used to rank features based on their relevance. The top 30 features are selected to reduce dimensionality and focus on the most informative attributes.
 - **Model Training (Machine Learning Models):** After feature selection, the preprocessed dataset is used to train six machine learning models. Each model learns from historical network traffic data and is trained to classify connections as normal or malicious.
 - **Model Evaluation (Metrics):** Each model is evaluated based on accuracy, precision, recall, F1-score, and computational efficiency. This phase helps identify the best-performing model, which will be used for real-time deployment.
 - **Real-Time IDS Deployment:** The best-performing model is integrated into a real-time IDS. The system continuously monitors network traffic and classifies each connection in real-time, alerting administrators to potential intrusions.
 - **Graphical User Interface (GUI):** The system includes a user-friendly interface that displays classification results in real-time. It provides details about the traffic, including source and destination IP addresses, protocol type, and whether the traffic is classified as normal or malicious.
-

4.3 System Functionality

- **Real-Time Monitoring:** The IDS system captures live network traffic, classifying each connection as either normal or abnormal based on the predictions made by the machine learning model. For testing, tools like **Nmap** can be used to simulate malicious traffic from virtual machines.
 - **Alerts and Logs:** When abnormal traffic is detected, the system logs the event and triggers an alert. For instance, after detecting ten abnormal connections, an audible beep alerts the user. This ensures that any suspicious activity is promptly identified and addressed.
 - **User Interface:** The graphical user interface (GUI) provides real-time feedback on the network's security status. It displays detailed information about each connection, including whether it is classified as normal or malicious, along with connection details such as IP addresses and protocol type.
-

Chapter 5: Evaluation and Results

This chapter presents the outcomes of the machine learning models tested on the NSL-KDD dataset, detailing both the initial evaluation using all features and the subsequent evaluation after feature selection. The chapter also discusses the performance of the real-time Intrusion Detection System (IDS) developed using the trained Random Forest model.

5.1 Data Preprocessing

The NSL-KDD dataset was used as the primary data source for training and evaluating the Intrusion Detection System. The dataset contains various features that describe network traffic, and several preprocessing steps were necessary to ensure the quality and usability of the data for machine learning models:

- **Handling Missing Values:** Although the NSL-KDD dataset contains minimal missing values, any instances with missing or erroneous data were removed to maintain dataset integrity. Records with incomplete feature values were dropped to avoid introducing bias into the model.
- **Categorical to Numerical Transformation:** Certain features, such as `protocol_type`, `service`, and `flag`, were categorical and needed to be converted into numerical form for compatibility with machine learning algorithms. This was done using **one-hot encoding**, which transforms each categorical variable into a series of binary features, each representing one unique category. This ensures that the models can interpret these categorical variables during training.

protocol_type	service	flag
tcp	ftp_data	SF
udp	other	SF
tcp	private	S0
tcp	http	SF
tcp	http	SF

Before Transformation

```

from sklearn import preprocessing
import pandas as pd

# Load the training and testing datasets
df_transformed = pd.read_pickle('dt_dataset.pkl')

# Specify the columns to encode
clm = ['protocol_type', 'service', 'flag']

# Create a label encoder for each column
label_encoders = {col: preprocessing.LabelEncoder() for col in clm}

# Fit the label encoders on the training data and transform the data
for col in clm:
    # Fit the label encoder on the training data
    label_encoders[col].fit(df_transformed[col])

    # Transform the data
    df_transformed[col] = label_encoders[col].transform(df_transformed[col])

# Print the mappings for each column
for col in clm:
    print(f'Mappings for {col}:')
    for idx, class_ in enumerate(label_encoders[col].classes_):
        print(f' {class_} - {idx}')
    print('\n')

```

Transformation Code Snippet

protocol_type	service	flag
1	20	9
2	44	9
1	49	5
1	24	9
1	24	9

After Transformation

- **Normalization:** To ensure that the range of feature values was consistent across the dataset, all numerical features were normalized. The **Min-Max normalization** technique was applied, scaling each feature to a range between 0 and 1. This step ensures that features with larger numerical ranges do not dominate the learning process of the models.


```

from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

numerical_features = ['duration', 'src_bytes', 'dst_bytes', 'wrong_fragment',
                      'num_compromised', 'root_shell', 'su_attempted',
                      'num_access_files', 'num_outbound_cmds', 'count',
                      'error_rate', 'srv_error_rate', 'same_srv_rate',
                      'dst_host_count', 'dst_host_srv_count', 'dst_host_same_src_port_rate',
                      'dst_host_srv_diff_src_port_rate', 'dst_host_srv_error_rate',
                      'dst_host_error_rate']

scaler = MinMaxScaler()

df[numerical_features] = scaler.fit_transform(df[numerical_features])

fig, ax = plt.subplots(1, 2, figsize=(12, 6))

```

Normalization Code Snippet

5.2 Initial Model Evaluation Using All Features

5.2.1 Algorithms

Several machine learning algorithms were employed to train the IDS, each selected for its suitability in handling classification tasks, specifically binary classification (normal vs. abnormal traffic). The algorithms implemented include **Random Forest**, **Support Vector Machine (SVM)**, **Decision Tree**, **Naive Bayes**, **K-Nearest Neighbors (KNN)**, and **XGBoost**.

- **Random Forest:** Random Forest was chosen for its robustness and high accuracy in classification tasks. As an ensemble learning method, Random Forest constructs multiple decision trees during training and averages the results, which helps reduce the likelihood of overfitting. It is particularly effective for handling datasets with a high number of features and complex relationships between variables.
- **Support Vector Machine (SVM):** SVM was selected for its effectiveness in binary classification tasks. It works by finding the hyperplane that best separates the two classes (normal and malicious traffic) while maximizing the margin between them. Despite its strong classification capabilities, SVM is computationally expensive, especially with large datasets, which limits its real-time applicability.
- **Decision Tree:** A Decision Tree model was used for its interpretability and ease of use. Decision Trees split the dataset based on feature values, providing a clear path for

classification. However, Decision Trees can be prone to overfitting if not properly managed.

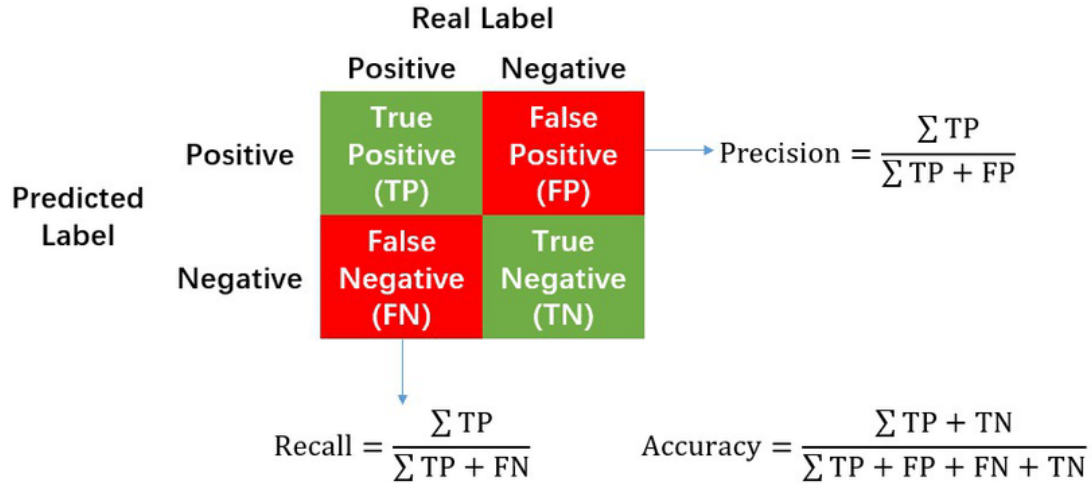
- **Naive Bayes:** Naive Bayes was chosen for its simplicity and speed, particularly in real-time classification tasks. It assumes independence between features, which often doesn't hold in real-world scenarios, but it has proven effective in certain types of intrusion detection, such as detecting R2L (Remote-to-Local) and U2R (User-to-Root) attacks.
- **K-Nearest Neighbors (KNN):** KNN was included for its strong performance in anomaly detection. KNN classifies new data points based on the majority class of the k-nearest neighbors in the training dataset. Although it is simple to implement, its high computational cost makes it less practical for real-time applications when working with large datasets.
- **XGBoost:** XGBoost is a gradient boosting algorithm known for its speed and performance in classification tasks. It builds a model incrementally by optimizing weak learners and is particularly effective in dealing with imbalanced datasets, which is a common issue in IDS datasets like NSL-KDD.

In the initial phase, all features of the NSL-KDD dataset were used to train and test six machine learning algorithms mentioned above. The results of these evaluations are summarized in the following tables and figures.

5.2.2 Evaluation Methodology

The evaluation of the models was conducted using an 80/20 split of the NSL-KDD dataset, where 80% of the data was used for training and 20% for testing. This approach ensures a robust evaluation of the models by allowing them to learn from a significant portion of the data while still being tested on unseen examples. The following metrics were used to assess the performance of the machine learning models:

- **Accuracy:** Accuracy is the ratio of correctly predicted instances (both true positives and true negatives) to the total number of instances in the dataset. It provides a general measure of model performance but can be misleading when there is class imbalance.
- **Precision:** Precision is the ratio of true positives to the total predicted positives (both true positives and false positives). It measures the accuracy of the positive predictions and is particularly important in intrusion detection, where false positives can cause unnecessary alerts.
- **Recall:** Recall (or sensitivity) is the ratio of true positives to the total actual positives (both true positives and false negatives). High recall is crucial for ensuring that all malicious activities are detected, even at the cost of some false positives.
- **F1-Score:** The F1-Score is the harmonic mean of precision and recall, providing a single metric that balances the trade-off between the two. It is especially useful in cases where class distribution is imbalanced, as it gives a more comprehensive measure of model performance.
- **Training Time:** The time taken to train the model on the training dataset.
- **Testing Time:** The time taken to evaluate the model on the testing dataset.



Accuracy, Precision and Recall Formula

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

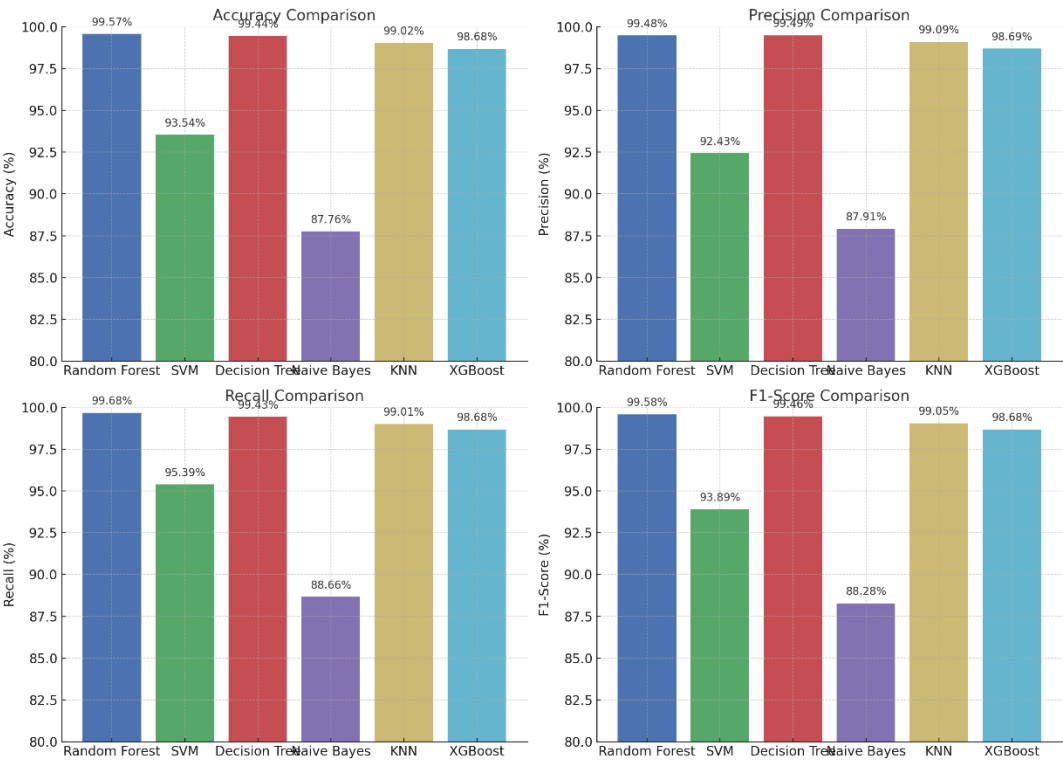
F1 Score Formula

5.2.3 Performance Metrics without Feature Selection

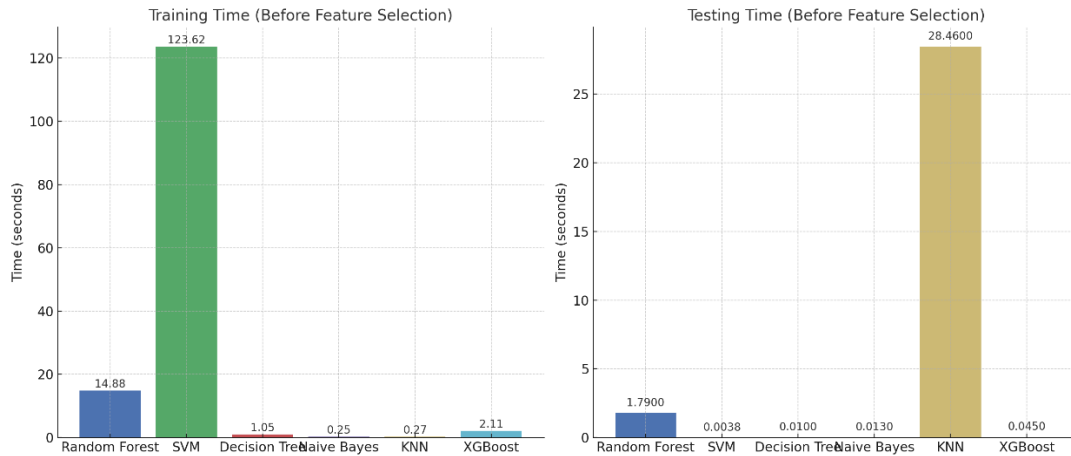
Initially, the models were trained using the full feature set from the NSL-KDD dataset. Six machine learning algorithms Random Forest, Decision Tree, Support Vector Machine (SVM), Naive Bayes, K-Nearest Neighbors (KNN), and XGBoost were evaluated using several key metrics—accuracy, precision, recall, F1-score, and computational efficiency (training and testing times). Additionally, confusion matrices were used to gain further insights into the classification performance by visualizing the number of **True Positives (TP)**, **True Negatives (TN)**, **False Positives (FP)**, and **False Negatives (FN)** for each model.

Model	Accuracy	Precision	Recall	F1-Score	Training Time (s)	Testing Time (s)
Random Forest	99.57%	0.994832	0.996828	0.995829	14.884996	1.786136
SVM	93.54%	0.924356	0.953851	0.938872	123.622632	0.003769
Decision Tree	99.44%	0.994948	0.994304	0.994626	1.046169	0.009844
Naive Bayes	87.76%	0.879091	0.886602	0.882831	0.246608	0.013274

Model	Accuracy	Precision	Recall	F1-Score	Training Time (s)	Testing Time (s)
K-Nearest Neighbors	99.02%	0.990931	0.990162	0.990546	0.267148	28.462209
XGBoost	98.68%	0.986854	0.986837	0.986835	2.110078	0.045460



Performance Comparison Before Feature Selection



Train and Test Time Comparison Before Feature Selection

Before Feature Selection Key Insights:

- **Random Forest** and **Decision Tree** had high accuracy (99.57% and 99.44%, respectively) and precision, showing strong predictive power for both normal and malicious traffic. However, **training time** for Random Forest was relatively high.
- **SVM** showed a lower accuracy (93.54%) but still had a high recall, making it effective in identifying a higher proportion of actual threats, though with a high false-positive rate (1,206 FP).
- **KNN** had a decent accuracy of 99.02%, but its testing time was significantly longer (28.46 seconds), making it less feasible for real-time applications.
- **Naive Bayes** had the lowest accuracy at 87.76%, indicating that the model struggled to differentiate between normal and malicious traffic using the full feature set.

5.2.4 Confusion Matrices Before Feature Selection

The confusion matrices for each model provide insight into the classification performance in terms of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

- **Random Forest**

	Positive	Negative
True	14,174	15,401
False	80	49

- **SVM**

	Positive	Negative
True	13,048	14,737
False	1,206	713

- **Decision Tree**

	Positive	Negative
True	14,176	15,362
False	78	88

- **Naive Bayes**

	Positive	Negative
True	12,370	13,698
False	1,884	1,752

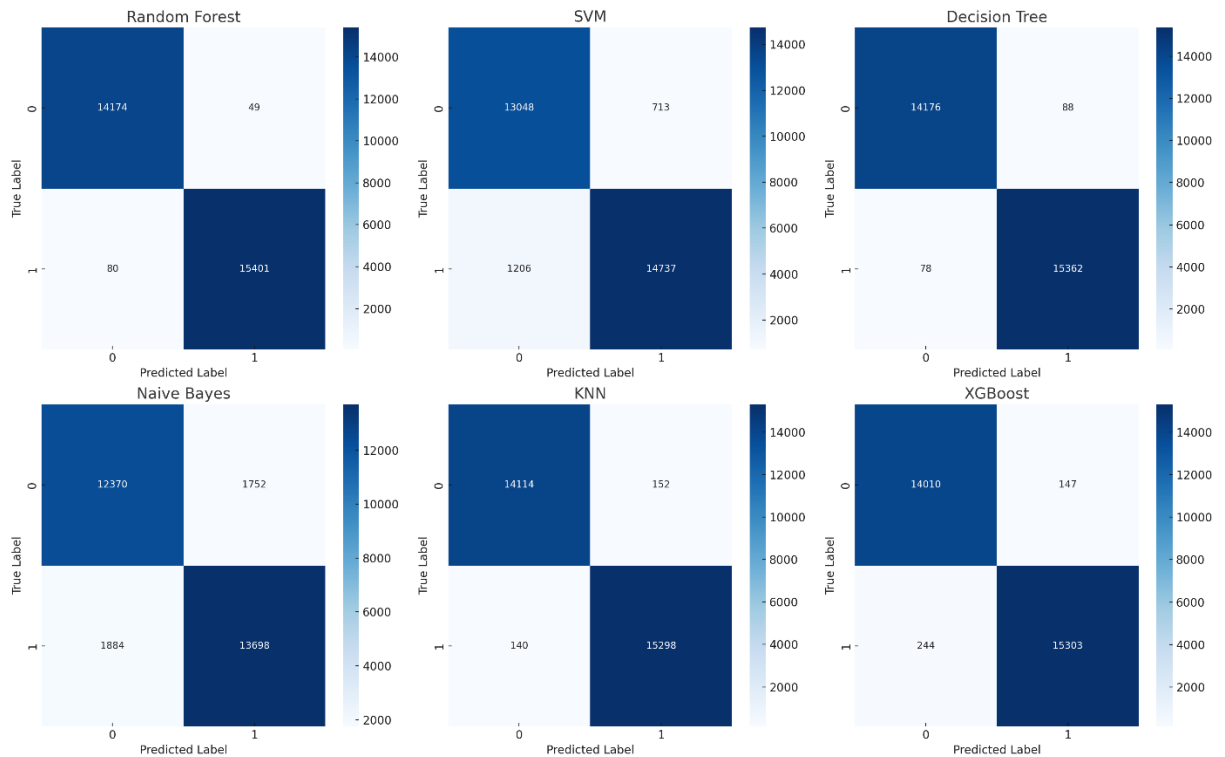
- **K-Nearest Neighbors**

	Positive	Negative
True	14,114	15,298
False	140	152

- **XGBoost**

	Positive	Negative
True	14,010	15,303
False	244	147

Confusion Matrices (Before Feature Selection)



Confusion Metrics Before Feature Selection

Before Feature Selection Confusion Matrix Insights:

- **Random Forest** achieved the highest **True Positives (14,174)** and **True Negatives (15,401)**, demonstrating strong predictive accuracy. However, it had 80 **False Positives** and 49 **False Negatives**, indicating it still missed a few malicious instances and misclassified some normal traffic.
- **SVM** had the highest **False Positives (1,206)**, meaning it incorrectly classified a substantial number of normal instances as malicious, which could lead to alert fatigue in real-world scenarios.
- **Naive Bayes** struggled with both high **False Positives (1,884)** and **False Negatives (1,752)**, indicating that the model performed less effectively compared to other algorithms when using the full feature set.

5.3 Feature Selection and Re-Evaluation

Feature Selection

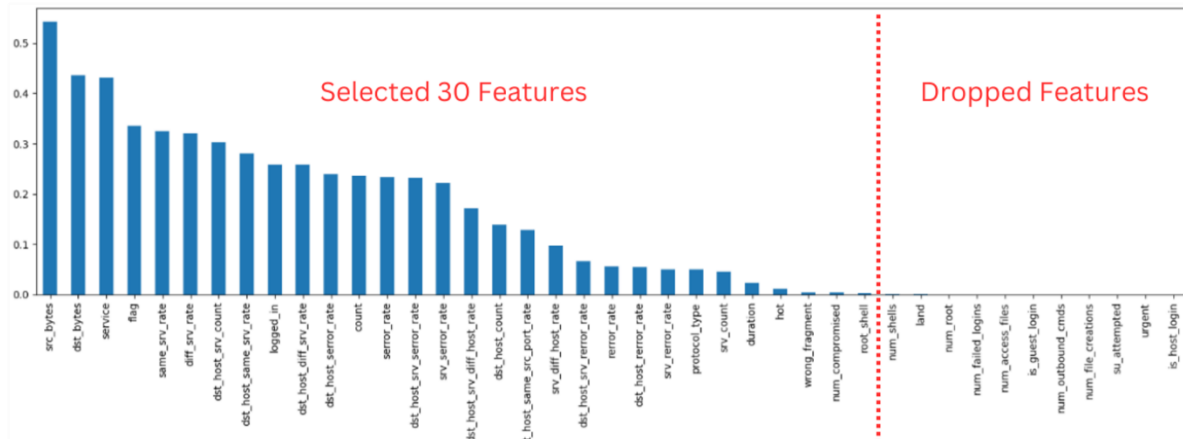
After the initial evaluation, the next step was to select the most relevant features that would improve both the accuracy and computational efficiency of the models and test out the effectiveness of Feature Selection. Feature selection is crucial for reducing the dimensionality of the data, improving model performance, and decreasing training time.

- **Mutual Information:** The feature selection process was performed using the **mutual information technique**. Mutual information quantifies the dependency between each feature and the target variable (normal or malicious traffic). Features with higher mutual information scores have a stronger relationship with the target and are more likely to contribute to accurate predictions.

```
from sklearn.feature_selection import mutual_info_classif
mutual_info = mutual_info_classif(X_train, y_train)
mutual_info = pd.Series(mutual_info)
mutual_info.index = train_index
mutual_info.sort_values(ascending=False)
```

Mutual Information Feature Selection Code Snippet

- **Top 30 Features:** Based on the mutual information scores, the top 30 features were selected for model training. These features were the most indicative of intrusion patterns in the network traffic. Reducing the number of features from the original 41 to 30 not only streamlined the model but also minimized the risk of overfitting, where the model could become overly complex and perform poorly on unseen data. Feature selection thus enhanced model efficiency without sacrificing accuracy.

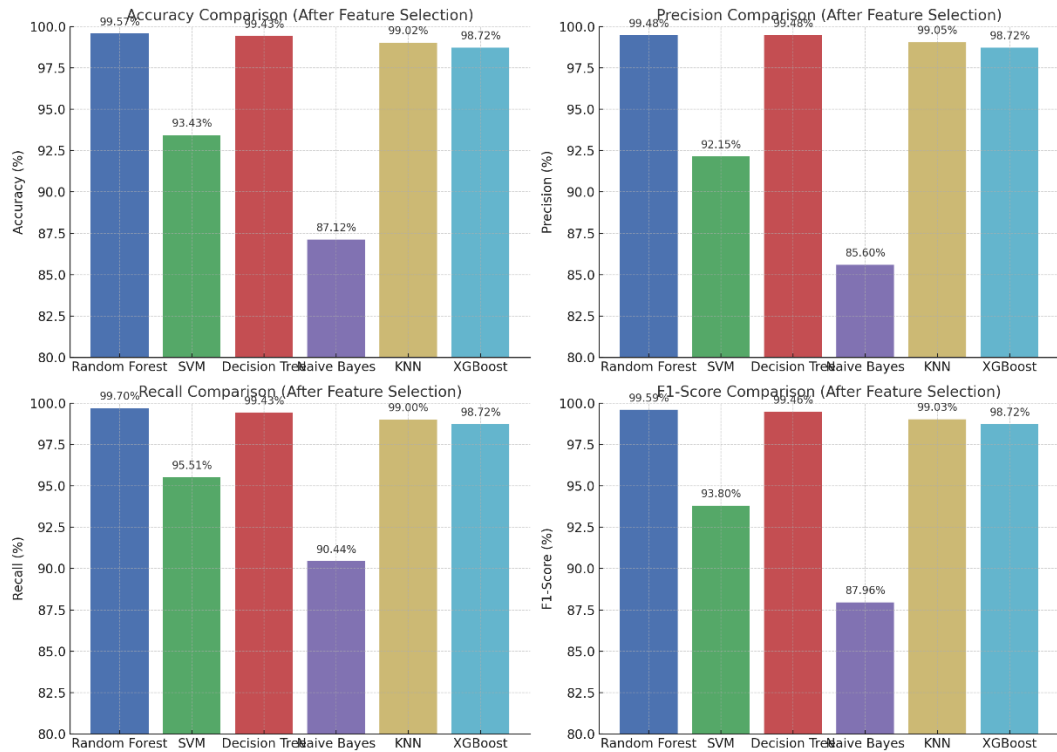


Identifying Top 30 Features

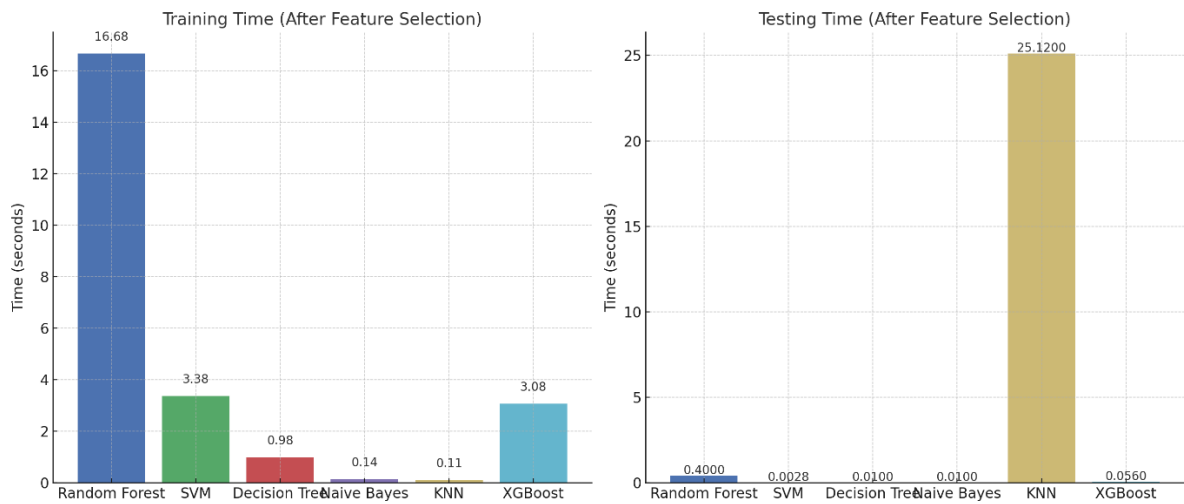
5.3.1 Performance Metrics After Feature Selection

After reducing the dataset to the top 30 most relevant features through feature selection, the models were re-evaluated. The goal of this step was to improve computational efficiency and potentially enhance accuracy by focusing on the most relevant features.

Model	Accuracy	Precision	Recall	F1-Score	Training Time (s)	Testing Time (s)
Random Forest	99.57%	0.994769	0.996958	0.995862	16.678887	0.397358
SVM	93.43%	0.921501	0.955081	0.937991	3.379206	0.002826
Decision Tree	99.43%	0.994819	0.994304	0.994562	0.979223	0.013459
Naive Bayes	87.12%	0.856084	0.904401	0.879580	0.140259	0.009508
K-Nearest Neighbors	99.02%	0.990481	0.990032	0.990257	0.107406	25.122777
XGBoost	98.72%	0.987227	0.987207	0.987205	3.078328	0.055827



Performance Comparison After Feature Selection



Train and Test Time Comparison After Feature Selection

After Feature Selection Key Insights:

- Random Forest** maintained its high accuracy at 99.57% but reduced its testing time from 1.78 to 0.40 seconds, demonstrating that feature selection improved computational efficiency without sacrificing predictive performance.

- **SVM** saw significant improvements in training time, dropping from 123.62 seconds to 3.38 seconds, while maintaining similar accuracy and recall, highlighting the benefits of reducing the dataset's dimensionality.
- **Naive Bayes** showed only marginal changes in performance, with a slight drop in accuracy to 87.12%, indicating that this model's performance was less affected by feature selection compared to others.
- **KNN** maintained its accuracy but saw a notable reduction in testing time from 28.46 seconds to 25.12 seconds, making it slightly more efficient but still relatively slow compared to other models.

5.3.2 Confusion Matrices After Feature Selection

- **Random Forest**

	Positive	Negative
True	14,169	15,405
False	85	45

- **SVM**

	Positive	Negative
True	12,932	14,769
False	1,322	681

- **Decision Tree**

	Positive	Negative
True	14,168	15,351
False	86	99

- **Naive Bayes**

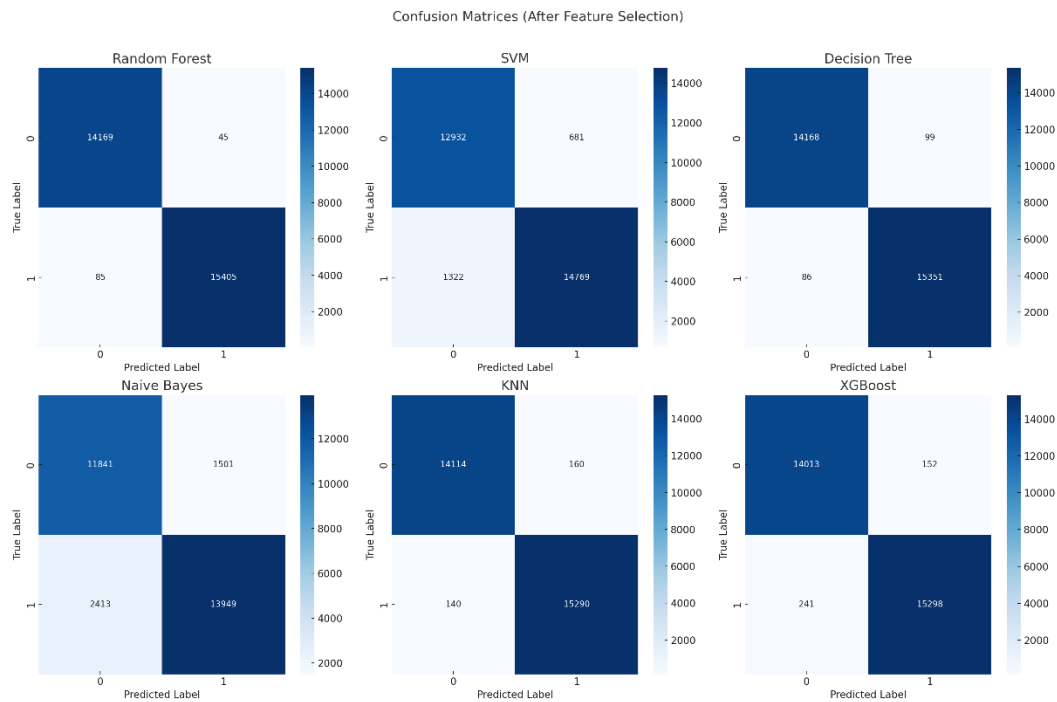
	Positive	Negative
True	11,841	13,949
False	2,413	1,501

- **K-Nearest Neighbors**

	Positive	Negative
True	14,114	15,290
False	140	160

- **XGBoost**

	Positive	Negative
True	14,013	15,298
False	241	152



Confusion Metrics After Feature Selection

After Feature Selection Confusion Matrix Insights:

- **Random Forest** maintained a high number of **True Positives (14,169)** and **True Negatives (15,405)** while slightly reducing both **False Positives (85)** and **False Negatives (45)** compared to the pre-selection results. This indicates better overall performance in classifying normal and malicious traffic with fewer errors.
 - **SVM** still had a large number of **False Positives (1,322)** but showed improvement in **False Negatives (681)** after feature selection, indicating that the model became slightly better at identifying actual threats while continuing to struggle with false alarms.
 - **Naive Bayes** saw only marginal improvements in its confusion matrix, with a slight decrease in **False Negatives (1,501)** and a reduction in **True Positives (11,841)**, reflecting its lower sensitivity to feature reduction.
-

5.4 Analysis of Results and Real-Time Evaluation

5.4.1 Comparative Analysis

Before Feature Selection:

- The models, particularly Random Forest and Decision Tree, achieved high accuracy with the full dataset, but they suffered from longer training and testing times.
- Models like Naive Bayes and KNN struggled with the high-dimensional data, showing either lower accuracy or longer testing times, highlighting inefficiencies with the full feature set.
- The confusion matrices for the models before feature selection highlight several areas of concern, such as high **False Positives** for SVM and **False Negatives** for Naive Bayes. The high-dimensional data made it harder for some models to accurately classify traffic.
- Random Forest and Decision Tree consistently performed well, with higher **True Positive** and **True Negative** rates and fewer **False Positives** and **False Negatives** compared to other models.

After Feature Selection:

- Feature selection reduced computational costs for most models, especially SVM and Random Forest, by decreasing training and testing times while maintaining or improving accuracy.
- The false positives (FP) and false negatives (FN) generally decreased after feature selection, contributing to more reliable performance, especially in the Random Forest and Decision Tree models.
- The KNN model, while accurate, still had longer testing times, making it less ideal for real-time IDS use.
- The reduction in the number of features improved most models' confusion matrix values, particularly in terms of reducing **False Positives** and **False Negatives**. Random Forest showed a slight but meaningful improvement in classification performance, reinforcing its suitability for real-time intrusion detection.
- The feature selection process helped streamline the models, making them more efficient and slightly more accurate at predicting malicious and normal traffic.

5.4.2 Impact of Feature Selection

Feature selection had a noticeable impact on both the performance and efficiency of the models. The Random Forest model maintained its high accuracy and F1-Score, while significantly reducing testing time. SVM showed a reduction in training time, which is particularly beneficial given its computational intensity. The overall performance of most models remained consistent, with a slight variation in precision, recall, and F1-Score. The feature selection process demonstrated that it is possible to maintain high accuracy while improving the efficiency of the models by focusing on the most relevant features.

Based on the performance metrics and confusion matrix analysis, **Random Forest** was selected for further development into a real-time Intrusion Detection System (IDS). The key factors leading to this decision were:

- **Superior Accuracy and Precision:** Random Forest achieved the highest accuracy of **99.57%** and an excellent precision of **0.9948**. These metrics are critical for detecting intrusions accurately while minimizing false positives.
- **Balanced Recall and F1-Score:** With a recall of **0.9969**, Random Forest demonstrated its ability to detect the vast majority of intrusions. The F1-Score of **0.9959** further indicated that it maintains a strong balance between precision and recall, ensuring robust detection capabilities.
- **Low False Positives/Negatives:** The confusion matrix for Random Forest revealed minimal false positives (85) and false negatives (45), indicating that it can effectively distinguish between normal and malicious traffic without overwhelming the system with false alerts.
- **Computational Efficiency:** Random Forest's training time was reduced to **16.68 seconds** and testing time to **0.40 seconds** after feature selection, making it well-suited for real-time applications where low latency is crucial.

5.4.3 Real-Time IDS Evaluation

To evaluate the performance of the real-time Intrusion Detection System (IDS) developed using the Random Forest model, a series of tests were conducted in a simulated network environment. The testing involved the use of **Nmap**, a popular network scanning tool, on a **Kali Linux virtual machine** running on **VMware** to simulate malicious network activity. Nmap is often used by attackers to discover network hosts and services, making it an ideal tool to test the IDS's ability to detect suspicious activity in real time.

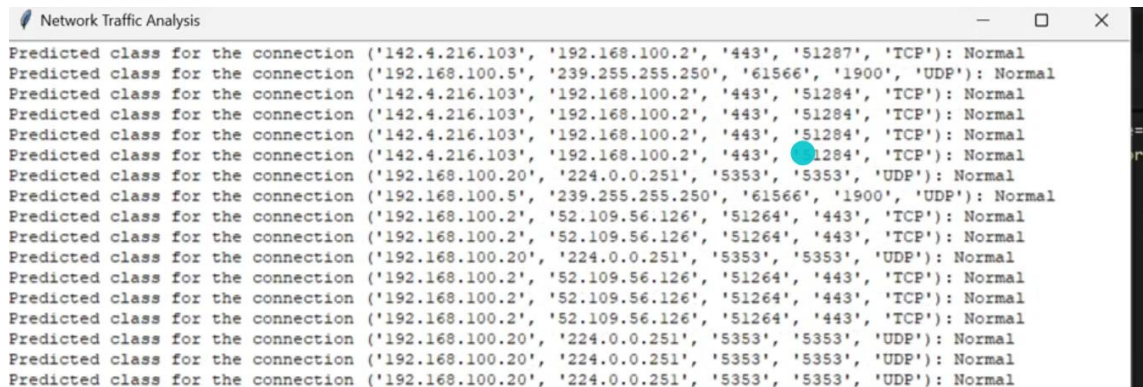
Testing Environment Setup

- A **host machine** running the IDS with the Random Forest model was used to monitor the network traffic.
- A **VMware virtual machine** running **Kali Linux** was configured to simulate potential intrusions using Nmap scans.
- The IDS was configured to:
 - Classify incoming network traffic as either **normal** or **abnormal**.
 - Generate real-time alerts when abnormal traffic patterns, such as Nmap scans, were detected.
 - Provide an **auditory alert** (a beep sound) for every 5 abnormal activities detected, improving user awareness during testing.

Testing Procedure

1. **Baseline Monitoring:**
 - Before initiating any scans, the network was monitored in a normal state.

- During this period, the IDS classified the network traffic as **normal**, and no alerts were triggered.
- This established a baseline to compare against once Nmap scans were initiated.



```

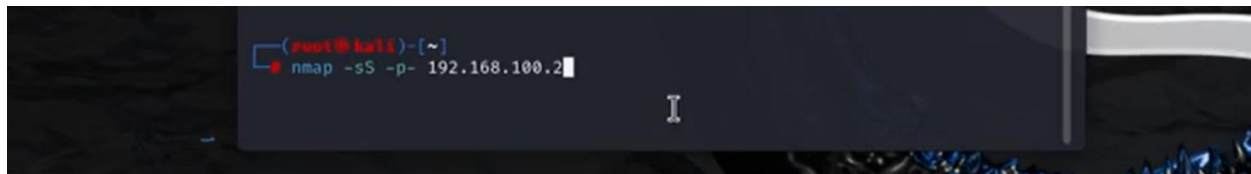
Network Traffic Analysis
Predicted class for the connection ('142.4.216.103', '192.168.100.2', '443', '51287', 'TCP'): Normal
Predicted class for the connection ('192.168.100.5', '239.255.255.250', '61566', '1900', 'UDP'): Normal
Predicted class for the connection ('142.4.216.103', '192.168.100.2', '443', '51284', 'TCP'): Normal
Predicted class for the connection ('142.4.216.103', '192.168.100.2', '443', '51284', 'TCP'): Normal
Predicted class for the connection ('142.4.216.103', '192.168.100.2', '443', '51284', 'TCP'): Normal
Predicted class for the connection ('142.4.216.103', '192.168.100.2', '443', '51284', 'TCP'): Normal
Predicted class for the connection ('192.168.100.20', '224.0.0.251', '5353', '5353', 'UDP'): Normal
Predicted class for the connection ('192.168.100.5', '239.255.255.250', '61566', '1900', 'UDP'): Normal
Predicted class for the connection ('192.168.100.2', '52.109.56.126', '51264', '443', 'TCP'): Normal
Predicted class for the connection ('192.168.100.2', '52.109.56.126', '51264', '443', 'TCP'): Normal
Predicted class for the connection ('192.168.100.20', '224.0.0.251', '5353', '5353', 'UDP'): Normal
Predicted class for the connection ('192.168.100.2', '52.109.56.126', '51264', '443', 'TCP'): Normal
Predicted class for the connection ('192.168.100.2', '52.109.56.126', '51264', '443', 'TCP'): Normal
Predicted class for the connection ('192.168.100.2', '52.109.56.126', '51264', '443', 'TCP'): Normal
Predicted class for the connection ('192.168.100.20', '224.0.0.251', '5353', '5353', 'UDP'): Normal
Predicted class for the connection ('192.168.100.20', '224.0.0.251', '5353', '5353', 'UDP'): Normal
Predicted class for the connection ('192.168.100.20', '224.0.0.251', '5353', '5353', 'UDP'): Normal

```

Monitoring Before Nmap Scan

2. Nmap Scan Simulation:

- Once the baseline was established, a variety of **Nmap scans** were executed from the Kali Linux virtual machine to simulate potential attacks. The scans included:
 - **SYN Scan:** A basic stealth scan commonly used by attackers to determine open ports on a target system.
 - **Service Version Detection:** A scan to gather information about services running on specific ports.
 - **Operating System Detection:** A scan to detect the operating system of the target machine.

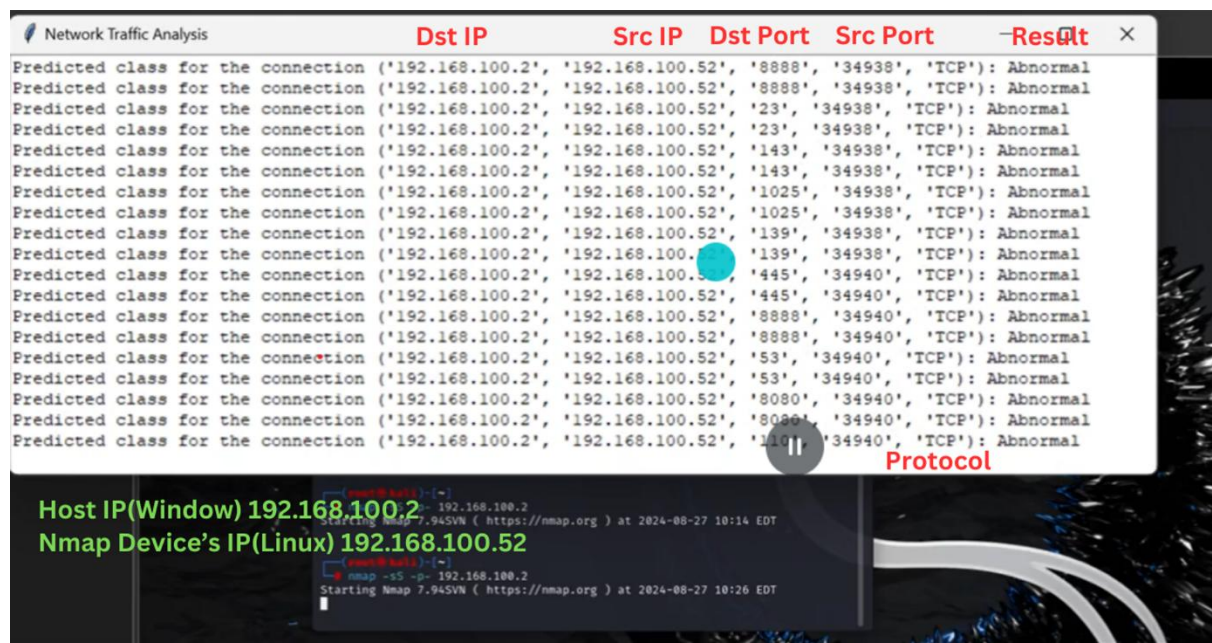


Starting Nmap Scan

3. Real-Time Detection:

- As the Nmap scans were launched, the IDS immediately started flagging the unusual network traffic.
- The traffic was classified as **abnormal**, and the system generated alerts to notify the user of potential threats.
- After every 10 abnormal detections, the IDS triggered an **auditory alert (a beep sound)**, signaling the detection of sustained suspicious activity. This alert

mechanism helped to enhance real-time monitoring by providing an additional layer of feedback to the user.



Real Time Detection After Nmap Scan

Results

- The IDS was able to detect the **Nmap scans** with high accuracy, classifying the abnormal network traffic patterns almost immediately after the scans were initiated.
- During the Nmap scans, the system transitioned from classifying the network traffic as **normal** to **abnormal**, effectively flagging the suspicious behavior.
- The auditory alert (beep) proved useful in real-time monitoring by signaling an accumulation of suspicious activities, prompting a quicker response from the security personnel.
- The overall response time of the system was minimal, with abnormal activities being detected and classified in near real-time, ensuring that any potential threats could be identified and mitigated promptly.

System Response to Network Activity

- **Before Nmap Scans:** The IDS displayed network traffic as normal, with no alerts triggered.
- **During Nmap Scans:** As soon as the Nmap scans began, the system flagged the abnormal activity, switching the classification to abnormal, and generating an alert for each detection.
- **Auditory Alerts:** For every 10 consecutive abnormal activities detected, the IDS triggered an auditory beep, serving as an additional real-time notification.

5.4.4 Impact on Real-Time Intrusion Detection

The real-time testing demonstrated the IDS's capability to detect common network scanning techniques like those initiated by Nmap. By classifying the abnormal traffic and providing real-time auditory feedback, the system showed its practicality in a live environment, offering a robust defense mechanism against reconnaissance activities, which are often the precursor to larger attacks.

The combination of visual and auditory alerts ensures that the system can promptly notify security personnel of potential intrusions, even when direct monitoring of the GUI may not be possible. This dual alert mechanism allows for more effective network surveillance and response.

5.5 Summary of Results

The results from the evaluations indicate that the Random Forest model is the most suitable for the development of an IDS based on the NSL-KDD dataset. The model's high accuracy, combined with efficient performance after feature selection, makes it ideal for real-time applications. The real-time IDS implementation confirmed the practicality of using this model in a live network environment, providing accurate and timely detection of abnormal connections.

The results also highlight the importance of feature selection in optimizing model performance. By reducing the number of features to the most relevant 30, the models not only maintained high accuracy but also became more efficient in terms of processing time. This balance between accuracy and efficiency is crucial for deploying machine learning models in real-world scenarios where both factors are critical.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

This project focused on the development and evaluation of a machine learning-based Intrusion Detection System (IDS) using the NSL-KDD dataset. Various machine learning algorithms, including Random Forest, Decision Tree, SVM, KNN, Naive Bayes, and XGBoost, were implemented and compared to determine the most effective method for detecting network intrusions.

To improve the performance and efficiency of the models, feature selection techniques were applied, with mutual information being used to identify the top 30 most relevant features. This step significantly enhanced both the computational efficiency and predictive accuracy of the models by reducing the complexity of the dataset while retaining the most informative features. Among the tested models, the Random Forest algorithm demonstrated the highest performance across key evaluation metrics such as accuracy, precision, recall, and F1-score, both before and after feature selection. The application of feature selection resulted in faster training and testing times without sacrificing detection accuracy, making the model more suitable for real-time deployment.

A real-time IDS was developed using the Random Forest model, which was integrated with a user-friendly graphical interface. The system effectively monitored network traffic and identified intrusions with high accuracy. This implementation highlights the practical applicability of machine learning techniques in real-world IDS, providing an efficient and scalable solution for network security monitoring.

The findings of this project underscore the importance of selecting the appropriate features and algorithms for intrusion detection. Feature selection not only improves performance but also reduces computational overhead, making it possible to implement IDS in real-time environments. Despite the success achieved with the NSL-KDD dataset, future work should focus on applying these methods to larger and more contemporary datasets that better reflect the current complexity of network traffic and cyber threats. Additionally, exploring more advanced machine learning techniques, such as deep learning and ensemble methods, may further enhance the robustness and adaptability of IDS solutions in the face of evolving cyber-attacks.

Future Work

While this project successfully developed and evaluated a machine learning-based Intrusion Detection System (IDS) using the NSL-KDD dataset, there are several potential areas for future exploration and improvement. These suggestions aim to enhance both the accuracy and practicality of IDS in real-world environments, particularly as cyber threats continue to evolve.

6.2 Application of Deep Learning Models

One promising direction for future work is the exploration of deep learning techniques such as **Convolutional Neural Networks (CNNs)** and **Long Short-Term Memory (LSTM)** networks. Deep learning models have been shown to capture complex patterns in data and can be particularly effective for anomaly detection in network traffic:

- **Convolutional Neural Networks (CNNs):** CNNs have been widely used for feature extraction and classification tasks. In the context of IDS, CNNs could be applied to analyze the spatial relationships between features in network traffic data, potentially improving the detection of sophisticated attacks that involve complex patterns of behavior.
 - **Long Short-Term Memory (LSTM):** LSTM networks, a type of Recurrent Neural Network (RNN), are particularly suited for sequence prediction tasks. They can be used to detect time-based attack patterns by analyzing sequences of network traffic over time, which could enhance the ability to identify attacks that occur over a prolonged period. Integrating LSTMs into the IDS could lead to improved detection rates for slow or stealthy attacks.
 - **Hybrid Models:** Future work could also explore combining traditional machine learning models (such as Random Forest) with deep learning techniques in a **hybrid approach**. This could leverage the strengths of both approaches, with deep learning models detecting complex patterns and machine learning models offering interpretability and computational efficiency.
-

6.3 Expanding the Dataset

While the NSL-KDD dataset has been widely used in IDS research, it has several limitations, particularly in terms of its age and the types of attacks it contains. Expanding the dataset to more contemporary and diverse sources would improve the generalizability and robustness of the IDS:

- **Modern Datasets:** Newer datasets such as **CICIDS 2017**, **UNSW-NB15**, or **TON_IoT** contain more recent and complex attack types, reflecting the current threat landscape more accurately. Incorporating these datasets into the model training process would enable the IDS to handle a broader range of attack scenarios and adapt to the evolving nature of cyber threats.
 - **Real-World Data:** Future work could also involve applying the IDS to real-world network traffic data. This would provide a more realistic test of its performance in live environments and help identify any limitations or challenges related to the model's deployment in production systems.
-

6.4 Improving Computational Efficiency

One of the major challenges in deploying machine learning-based IDS for real-time use is the computational cost associated with training and testing models, particularly when dealing with large datasets or complex algorithms:

- **Optimization Techniques:** Techniques such as **model pruning**, **quantization**, or the use of more efficient algorithms could be explored to reduce the computational overhead without sacrificing accuracy. For instance, simplifying the model architecture or using lightweight models for real-time detection could make the IDS more scalable and faster in production environments.
 - **Parallelization and Distributed Computing:** Future work could also explore the use of **parallel processing** or **distributed computing** frameworks, such as **Apache Spark** or **TensorFlow Distributed**, to handle large volumes of network traffic in real-time. This would enable the IDS to process data more efficiently and respond to threats in a timely manner, making it suitable for large-scale network deployments.
-

6.5 Real-Time Implementation and Scalability

While this project developed a basic real-time IDS with a graphical user interface (GUI), there is room for improvement in terms of scalability and system integration:

- **Scaling to Larger Networks:** Future work could focus on enhancing the IDS to handle larger and more complex network environments. This may involve optimizing the system to handle higher volumes of network traffic while maintaining low latency and high accuracy in detecting intrusions.
 - **Integration with Security Information and Event Management (SIEM) Systems:** Another avenue for future work is the integration of the IDS with existing **SIEM platforms**. This would allow the IDS to provide more comprehensive security monitoring by correlating network traffic data with logs from other security tools, improving the system's ability to detect coordinated or advanced attacks.
-

6.6 Adaptive Learning and Attack Evolution

Cyber threats evolve continuously, and attackers often develop new techniques to evade detection by security systems. Future work could focus on enabling the IDS to adapt to these evolving threats through **continuous learning**:

- **Online Learning Algorithms:** Implementing **online learning** models that can update themselves as new data becomes available would allow the IDS to adapt to changes in network traffic patterns and new types of attacks without requiring frequent retraining.

This would improve the system's responsiveness to emerging threats and reduce the need for manual intervention.

- **Attack Simulation and Adversarial Testing:** Future research could also explore adversarial testing, where simulated attacks are used to stress-test the IDS and evaluate its ability to detect novel or stealthy intrusion techniques. This would provide a better understanding of the system's weaknesses and help guide improvements in the detection algorithms.
-

Appendix: Project Management

Effective project management is critical for the successful completion of any project. This section outlines the planned timeline, key milestones, and the overall management of the project. The project was divided into several stages, with specific tasks assigned to each phase. A Gantt chart was used to track progress over time and ensure that the project stayed on schedule.

Planned Project Timeline

The project was organized into six main phases, as outlined in the Gantt chart below. The timeline spans a total of 16 weeks, with each task allocated a specific duration to ensure the smooth execution of the project. The phases included:

- **Define project scope and objectives:** This phase occurred during the first two weeks and involved setting the overall goals and defining the scope of the intrusion detection system (IDS) project.
- **Literature review on intrusion detection techniques:** This critical phase, which ran from week 3 to 5, involved reviewing existing research and methodologies related to intrusion detection systems and machine learning-based IDS.
- **Data collection and preprocessing:** Data collection began in week 6 and continued through week 8. This phase included collecting the necessary data, cleaning it, and performing the required preprocessing, such as feature selection.
- **Model selection and implementation:** Running from weeks 9 to 13, this phase involved selecting the machine learning algorithms for the IDS and implementing them based on the cleaned data.
- **Evaluation and testing:** Weeks 14 to 15 focused on evaluating the model's performance using various metrics, followed by rigorous testing to ensure accuracy and efficiency.
- **Documentation and final report preparation:** During the final week (week 16), the focus was on preparing the final report and documentation, ensuring all aspects of the project were well-documented and ready for submission.

Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16
Define project scope and objectives																
Literature review on intrusion detection techniques																
Data collection and preprocessing																
Model selection and implementation																
Evaluation and testing																
Documentation and final report preparation																

Overall Project Gantt Chart

Initial 14-Day Timeline

The Gantt chart below outlines the first 14 days of the project, which served as the foundation for the rest of the project timeline. This short-term plan was essential for setting up the initial stages of the project, ensuring that all preliminary tasks were completed efficiently. The tasks included:

- **Review existing literature on IDS and machine learning methods:** This was conducted during the first three days to gain a comprehensive understanding of the field and inform the direction of the project.
- **Define the scope of the project:** Overlapping with the literature review, the project scope was defined during days 4 to 7, ensuring clear goals and objectives.
- **Create project milestones and deliverables:** This task was crucial in establishing the project's roadmap, determining key milestones, and setting realistic deliverables. It was completed by day 11.
- **Research and select tools and technologies for development:** In days 12 to 14, the tools and technologies required for development were identified and selected. This included the software and libraries necessary for implementing the IDS.

Task	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10	Day 11	Day 12	Day 13	Day 14
Review existing literature on IDS and machine learning methods														
Define the scope of the project														
Create project milestones and deliverables														
Research and select tools and technologies for development														

Project Gantt Chart for the First 2 weeks

Milestones and Management

Throughout the project, several key milestones were reached:

- **Completion of literature review and project scope definition** (Week 5)
- **Data collection and preprocessing completed** (Week 8)
- **Model selection and implementation completed** (Week 13)
- **Evaluation and testing completed** (Week 15)
- **Final report preparation and documentation completed** (Week 16)

These milestones helped ensure the project remained on track and provided clear indicators of progress.

Monthly Reports

The following section provides a summary of the monthly reports submitted during the project, documenting the progress, milestones reached, challenges encountered, and solutions implemented throughout the project.

Month 1: Project Initiation and Literature Review

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting (highlight as appropriate): May

Student: Arkar Hmue Htet.....

Main issues / Points of discussion / Progress made
<ul style="list-style-type: none">- Defined the project scope and set clear objectives for the IDS development.- Extensive literature review on IDS and machine learning algorithms (Random Forest, SVM, KNN, XGBoost).- Identified strengths and limitations of current IDS research.- Chose the NSL-KDD dataset and focused on machine learning-based IDS.
Actions for the next month
<ul style="list-style-type: none">- Begin data collection and preprocessing phase.- Handle missing data, encode categorical features, and normalize data.- Select key features for model development.
Deliverables for next time
<ul style="list-style-type: none">- Data preprocessing completion with relevant feature selection.- Dataset ready for model training.
Other comments
<ul style="list-style-type: none">- Initial difficulties in narrowing the project scope resolved through discussions with the supervisor.

Supervisor Signature

Student signature: Arkar

Month 2: Data Collection and Preprocessing

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting (highlight as appropriate): June

Student: Arkar Hmue Htet.....

Main issues / Points of discussion / Progress made
<ul style="list-style-type: none">- Collected NSL-KDD dataset for IDS development.- Completed data preprocessing, handling missing values, encoding, and normalizing data.- Reduced dataset using mutual information to top 30 most relevant features.
Actions for the next month
<ul style="list-style-type: none">- Implement machine learning models using the preprocessed dataset.- Train the models and begin performance evaluation.
Deliverables for next time
<ul style="list-style-type: none">- Implementation and initial evaluation of machine learning models.
Other comments
<ul style="list-style-type: none">- Feature selection took longer than expected, but mutual information proved to be an effective technique.

Supervisor Signature

Student signature: Arkar

Month 3: Model Selection and Implementation

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting (highlight as appropriate): July

Student: Arkar Hmue Htet.....

Main issues / Points of discussion / Progress made
<ul style="list-style-type: none">- Implemented Random Forest, SVM, Decision Tree, Naive Bayes, KNN, and XGBoost on the dataset.- Evaluated models using metrics like accuracy, precision, recall, and F1-score.- Hyperparameter tuning was required for optimal performance in certain models like SVM and Naive Bayes.
Actions for the next month
<ul style="list-style-type: none">- Finalize model evaluation and select the best-performing model for real-time IDS implementation.- Begin integration of the selected model into a real-time system.
Deliverables for next time
<ul style="list-style-type: none">- Final model evaluation and initial real-time IDS integration.
Other comments
<ul style="list-style-type: none">- Balancing precision and recall was challenging, particularly for SVM and Naive Bayes models.

Supervisor Signature

Student signature: Arkar

Month 4: Evaluation, Testing, and Real-Time IDS Implementation (Weeks 14-15) And Documentation and Final Report Preparation (Weeks 16)

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting (highlight as appropriate): Aug

Student: Arkar Hmue Htet.....

Main issues / Points of discussion / Progress made
<ul style="list-style-type: none">- Final model evaluation completed; Random Forest selected for real-time IDS due to superior performance.- Developed and tested a real-time IDS using live network traffic from VMware Kali Linux with Nmap scans.- Integrated auditory alert system for abnormal detections.- Completed final project report with documentation of all project phases and included project management tools like Gantt charts.
Actions for the next month
<ul style="list-style-type: none">- Submit the final report for review and make adjustments if necessary.- Address any remaining feedback from the supervisor.
Deliverables for next time
<ul style="list-style-type: none">- Final IDS system submission.- Approved final project report.
Other comments
<ul style="list-style-type: none">- Successfully adjusted the system to handle real-time data streams and simulated intrusion detection scenarios using controlled virtual environments.

Supervisor Signature

Student signature: Arkar

References

1. Belavagi, M. C., & Muniyal, B. (2016). Performance evaluation of supervised machine learning algorithms for intrusion detection. *Procedia Computer Science*, 89, 117-123.
 2. Pathak, A., & Pathak, S. (2020). Study on Decision Tree and KNN Algorithm for Intrusion Detection System. *International Journal of Engineering Research & Technology*.
 3. Kumar, M., & Hanumanthappa, M. (2012). Intrusion Detection System using decision tree algorithm. *IEEE Conference on Information Technology*.
 4. Altwaijry, H., & Algarny, S. (2012). Bayesian based intrusion detection system. *Journal of King Saud University – Computer and Information Sciences*.
 5. Li, Y., Xia, J., Zhang, S., Yan, J., Ai, X., & Dai, K. (2012). An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Systems with Applications*.
 6. Liao, Y., & Vemuri, R. V. (2002). Use of K-Nearest Neighbor classifier for intrusion detection. *Computers & Security*, 21(10), 439-448.
 7. Buczak, A. L., & Guven, E. (2015). A survey of machine learning algorithms for intrusion detection systems. *IEEE Communications Surveys & Tutorials*, 18(2), 1153-1176.
 8. Chio, C., & Freeman, D. (2018). *Machine Learning and Security*. O'Reilly Media.
 9. Di Pietro, R., & Mancini, L. V. (2008). *Intrusion Detection Systems*. Springer.
 10. Moustafa, N., & Slay, J. (2015). A detailed analysis of the KDD CUP 99 dataset for the anomaly detection problem. *Proceedings of the second workshop on building analysis datasets and gathering experience returns for security*.
 11. Singh, G., et al. (2014). Random Forest for Network Intrusion Detection System. *International Journal of Advanced Research in Computer and Communication Engineering*.
 12. Tavallaee, M., et al. (2009). A detailed analysis of the NSL-KDD dataset. *Proceedings of the 2nd IEEE International Conference on Computer Security and Industrial Cryptography*.
 13. Vinayakumar, R., et al. (2017). Deep learning techniques for intrusion detection using NSL-KDD dataset. *Information and Communication Technology for Sustainable Development*.
 14. Berman, D. S., et al. (2019). A survey of deep learning methods for cyber security. *IEEE Communications Surveys & Tutorials*.
-