ANWER KHAN MODERN UNIVERSITY

ABDULLAH

SHAHRIAR ISLAM SHIFAT

Network Intrusion Detection: A Comparative Study on Logistic Regression, KNN, and Light Gradient Boosting Machine Learning Techniques

FACULTY OF ENGINEERING AND TECHNOLOGY

Department of Computer Science and Engineering

BSc in Computer Science and Engineering

Semester: Fall 2022

Supervisor: Shovan Kumar Paul

April 2023

ANWER KHAN MODERN UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

Department of Computer Science and Engineering

BSc in Computer Science and Engineering

Semester: Fall 2022

ABDULLAH

SHAHRIAR ISLAM SHIFAT

Network Intrusion Detection: A Comparative Study on Logistic Regression, KNN, and Light Gradient Boosting Machine Learning Techniques

Supervisor: Shovan Kumar Paul

April 2023

This thesis is submitted in partial fulfillment of the requirements for the degree of BSc in Computer Science and Engineering

© Anwer Khan Modern University 2023. All rights reserved. No part of this publication may be reproduced without the written permission of the copyright owner.

ABSTRACT

The aim of this study was to develop a network intrusion detection system using the KDD Cup 99 dataset. Three machine learning algorithms, namely Logistic Regression, K-Nearest Neighbors, and Light Gradient Boosting, were compared to determine the best algorithm for this task. Recursive Feature Elimination (RFE) was used to select the most key features for the models. The results showed that Light Gradient Boosting achieved the highest accuracy of 99.65% on the test set, followed by K-Nearest Neighbors with 98.33%, and Logistic Regression with 94.39%. The selected features included protocol type, service, flag, source bytes, destination bytes, logged-in, count, etc. Overall, the results demonstrate the effectiveness of using machine learning algorithms in network intrusion detection and the importance of feature selection in improving the performance of the models.

**Keywords:**

Network Intrusion Detection, Logistic Regression, KNN, Light Gradient Boosting, KDD Cup 99 dataset.

ACKNOWLEDGEMENT

We humbly prostrate ourselves before Allah, the Most Compassionate and Merciful, who has given us the capacity to feel and the ability to judge, enabling us to understand what was previously unknown to us.

All respect and praises to the Holy Prophet Hazrat Muhammad (Sallallahu Alaihi Wasallam), who arrived as the light of knowledge for all pursuers and a real role model for the whole of mankind.

We extend our heartfelt appreciation to Assistant Professor, Shovan Kumar Paul for supervising our thesis. His invaluable guidance, advice, and support throughout this research. Also, his constructive feedback and expert suggestions were instrumental in shaping the outcome of this thesis.

We would also like to express our sincere gratitude to Assistant Professor, Farhana Haque for her valuable insights, encouragement, and unwavering support. Her expertise and guidance have been invaluable in helping us to navigate the complexities of this research.

Finally, we must acknowledge the unconditional love, encouragement, and support of our parents and friends. Their constant guidance, unwavering support, and encouragement have been an endless source of strength and inspiration to us throughout our academic journey.

**Table of Contents**

[ABSTRACT i](#_Toc131721331)

[ACKNOWLEDGEMENT ii](#_Toc131721332)

[1. INTRODUCTION 1](#_Toc131721333)

[1.1 Background of study 1](#_Toc131721334)

[1.2 Objectives 1](#_Toc131721335)

[1.3 System Building 3](#_Toc131721336)

[1.4 Motivation: 4](#_Toc131721337)

[2 LITERATURE REVIEW 5](#_Toc131721338)

[3 Dependency on Libraries: 11](#_Toc131721339)

[4 METHODOLOGY 15](#_Toc131721340)

[4.1 Data Collection: 15](#_Toc131721341)

[4.2 Data preprocessing: 18](#_Toc131721342)

[4.3 Feature selection: 25](#_Toc131721343)

[4.3.1 Important Features: 25](#_Toc131721344)

[4.3.2 Importance of Feature Selection: 26](#_Toc131721345)

[4.3.3 Methods for Feature Selection: 27](#_Toc131721346)

[4.3.4 Explanation of Chosen Features: 28](#_Toc131721347)

[4.4 Data Standardization: 30](#_Toc131721348)

[4.5 Splitting: 32](#_Toc131721349)

[4.6 The Training Models: 33](#_Toc131721350)

[4.6.1 Light Gradient Boosting Machine: 33](#_Toc131721351)

[4.6.2 K-Nearest Neighbors (K-NN) Model: 35](#_Toc131721352)

[4.6.3 Logistic Regression Model: 36](#_Toc131721353)

[4.6.4 Evaluate the Three Models: 37](#_Toc131721354)

[5 RESULTS 38](#_Toc131721355)

[5.1 Performance: 40](#_Toc131721356)

[6 DISCUSSION 40](#_Toc131721357)

[6.1 Evaluation of NIDS 40](#_Toc131721358)

[6.2 Future Research Directions 43](#_Toc131721359)

[6.3 Ethical Considerations 44](#_Toc131721360)

[6.4 Methodological Considerations 45](#_Toc131721361)

[6.4.1 Data Collection: 45](#_Toc131721362)

[6.4.2 Dependency on Libraries: 46](#_Toc131721363)

[6.4.3 Data Preprocessing: 46](#_Toc131721364)

[6.4.4 Feature Selection: 46](#_Toc131721365)

[6.4.5 Data Standardization: 46](#_Toc131721366)

[6.4.6 Data Splitting: 47](#_Toc131721367)

[6.4.7 Model Selection: 47](#_Toc131721368)

[6.4.8 Training and Evaluation: 47](#_Toc131721369)

[6.4.9 Metrics: 48](#_Toc131721370)

[6.4.10 Challenges: 48](#_Toc131721371)

[6.4.11 Recommendations: 48](#_Toc131721372)

[6.5 Comparison with Existing System 49](#_Toc131721373)

[7 CONCLUSION 50](#_Toc131721374)

[8 REFERENCE 51](#_Toc131721375)

List of Figures

[Figure 1.1 System Building Diagram 3](#_Toc131443775)

[Figure 4.1 Features of the dataset 17](#_Toc131443776)

[Figure 4.2 Top five features of the training dataset 19](#_Toc131443777)

[Figure 4.3 Dataset information 20](#_Toc131443778)

[Figure 4.4 Numerical variables of dataset 21](#_Toc131443779)

[Figure 4.5 Categorical feature of dataset 21](#_Toc131443780)

[Figure 4.6 Missing values 22](#_Toc131443781)

[Figure 4.7 Heatmap of different features 23](#_Toc131443782)

[Figure 4.8 Class distribution of dataset 24](#_Toc131443783)

[Figure 4.9 Important features 26](#_Toc131443784)

[Figure 4.10 Evaluation results heatmap 38](#_Toc131443785)

[Figure 5.1 Results of system 39](#_Toc131443786)

# 

# **INTRODUCTION**

## **Background of study**

In recent years, the rise of cyber-attacks has become a major concern for individuals, organizations, and governments worldwide. As the number and complexity of cyber threats continue to increase, it has become essential to develop effective methods for detecting and preventing these attacks. One such method is the use of Network Intrusion Detection Systems (NIDS), which are designed to identify malicious activity within a network and alert system administrators of potential threats.

The development of NIDS has been driven by the need to address the growing threat of cyber-attacks. These attacks can result in significant financial losses, reputational damage, and even the loss of sensitive data. The importance of NIDS is highlighted by the fact that many regulatory bodies require organizations to implement them as part of their security measures.

The effectiveness of NIDS depends on the ability to accurately detect and classify several types of attacks. Traditional rule-based systems have limitations in detecting new and complex attacks, which has led to the adoption of machine learning algorithms for intrusion detection. The use of machine learning algorithms can improve the accuracy of NIDS by analyzing large amounts of data and identifying patterns that may be indicative of an attack.

This thesis aims to develop a NIDS using machine learning algorithms and evaluate its performance using the KDD Cup 99 dataset. The objective is to compare the performance of different machine learning algorithms and select the most effective algorithm for this task. The findings of this study will contribute to the development of more accurate and effective NIDS, which are crucial in protecting against cyber threats.

## **Objectives**

The objectives of our network intrusion detection system (NIDS) are the specific goals that we aim to achieve through our research and development efforts. These objectives serve as a roadmap for our project, providing a clear direction for our work and a basis for measuring our success.

The first objective of our NIDS is to develop a system that can accurately detect and classify network intrusions. Network intrusions are a serious threat to the security of computer systems, and they can lead to data theft, system crashes, and other forms of damage. By accurately detecting and classifying intrusions, our system can provide early warning of attacks and allow system administrators to take appropriate actions to mitigate the damage.

The second objective of our NIDS is to use machine learning algorithms to improve the accuracy and efficiency of intrusion detection. Machine learning algorithms can be trained to recognize patterns in network traffic that are indicative of intrusion attempts, and they can do so with a high degree of accuracy. By incorporating machine learning algorithms into our NIDS, we can improve its effectiveness and reduce the number of false alarms, which can be time-consuming and costly to investigate.

The third objective of our NIDS is to use feature selection techniques to identify the most key features for intrusion detection. Feature selection is the process of identifying the most relevant features of network traffic for intrusion detection, and it can significantly improve the performance of machine learning algorithms. By using feature selection techniques, we can reduce the number of features that our algorithms need to analyze, which can improve their accuracy and efficiency.

The fourth objective of our NIDS is to evaluate the performance of our system using standard metrics and benchmarks. To determine whether our NIDS is effective, we will evaluate its performance using standard metrics such as accuracy, precision, recall, and F1 score. We will also compare the performance of our system to other state-of-the-art intrusion detection systems using standard benchmarks such as the KDD Cup 99 dataset.

Overall, these specific objectives are designed to guide our research and development efforts and ensure that our NIDS is effective, efficient, and practical for use in real-world settings. By achieving these objectives, we will contribute to the field of network security and provide valuable tools for system administrators to detect and respond to network intrusions.

## **System Building**

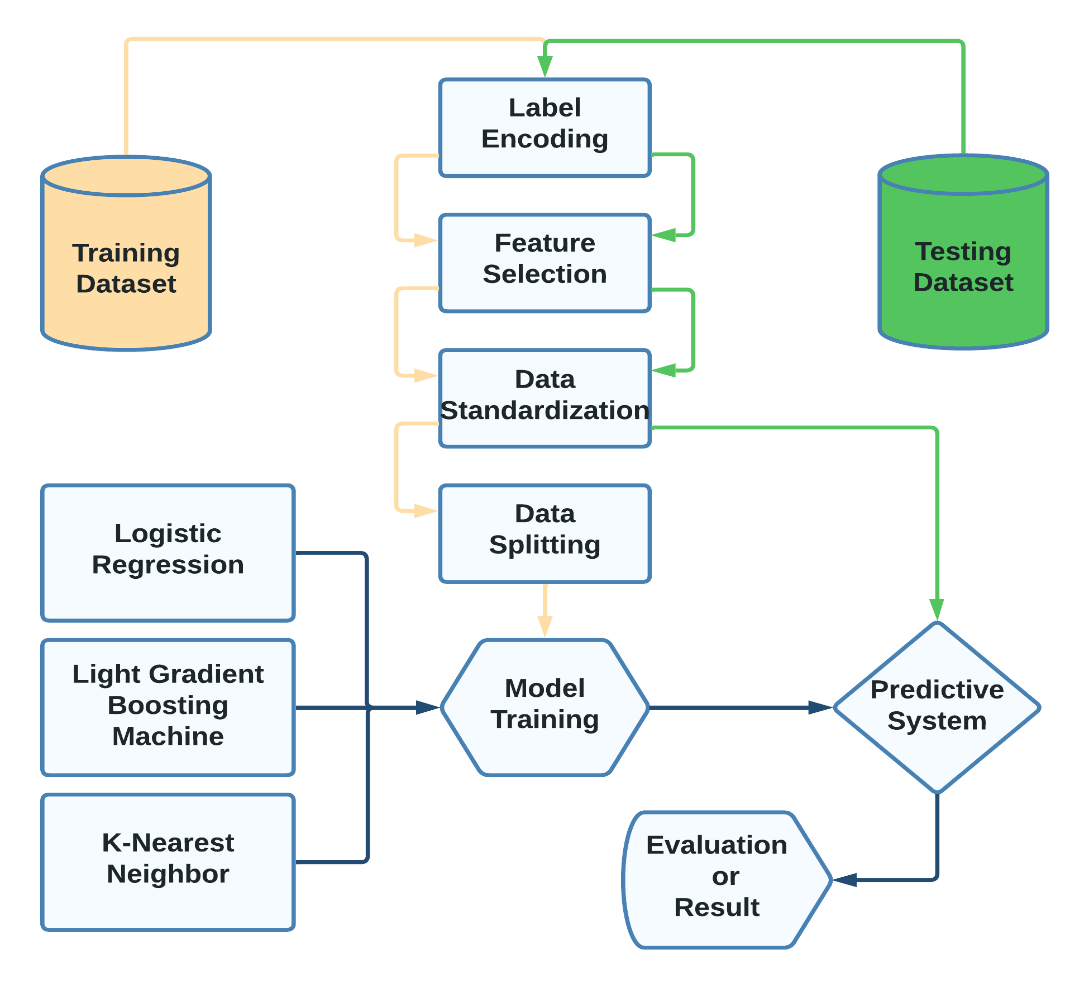
This system follows a step-by-step process to prepare and analyze data. First, training data is utilized to prepare the dataset for analysis. Then categorical data is converted into statistical data using label encoding. After that, the dataset uses feature selection to identify important features. Lastly in the preprocessing part, standardization is applied to ensure the data is normalized and can be compared across all features.

The dataset is then split into training and testing sets to facilitate model training and evaluation. Three machine learning models are used for this process - logistic regression (LGR), k-nearest neighbors (K-NN), and light gradient boosting machine (LGBM). Each model is trained on the training data to generate predictions on the test data.

The final step of the process involves evaluating the performance of the models. Evaluation metrics such as accuracy, precision, and recall are used to compare the performance of each model. This enables the selection of the best-performing model for use in subsequent analysis or decision-making.

The following is **Figure 1.1**

Figure 1.1 System Building Diagram



## **Motivation:**

The motivation behind our research is the growing concern over cybersecurity threats and the increasing need for effective intrusion detection systems to protect network security. The number of cybersecurity incidents has been rising steadily in recent years, and organizations of all sizes are at risk of being targeted by hackers and cybercriminals. The cost of cybercrime is also increasing, with estimates reaching billions of dollars annually.

In response to these challenges, intrusion detection systems have been developed as a means of detecting and preventing unauthorized access to networks. However, these systems often suffer from high false positive rates, which can result in a significant waste of resources and cause the system to be ignored or abandoned.

To address these issues, our research aims to develop an efficient and accurate network intrusion detection system that can accurately identify and classify potential threats while minimizing false positives. By leveraging machine learning algorithms and advanced feature selection techniques, we aim to improve the overall performance and effectiveness of intrusion detection systems, providing organizations with better protection against cyber threats.

Overall, our research is motivated by the need to enhance network security and protect against the rising threat of cybercrime. By developing a more accurate and efficient intrusion detection system, we hope to contribute to the ongoing efforts to improve cybersecurity and protect against the increasing risk of cyber-attacks.

Overall, in our system, we will begin by providing an overview of the background information and literature review related to network intrusion detection systems (NIDS) and the KDD Cup 99 dataset. We will then discuss the motivation for our research and the research questions and objectives that we aim to address through our study.

Next, we will describe the methodology used in our research, including data preprocessing, feature selection, and model training and evaluation. We will also present the results of our experiments and compare the performance of the three machine learning algorithms, Logistic Regression, K-Nearest Neighbors, and Light Gradient Boosting, that we used to build the NIDS.

In the discussion section, we will analyze the results and provide insights into the effectiveness of machine learning algorithms in NIDS and the importance of feature selection in improving model performance. We will also highlight the limitations of our study and provide recommendations for future research in this area.

Finally, we will conclude the report by summarizing the main findings and contributions of our study and their implications for network security.

# **LITERATURE REVIEW**

In this study, the authors [1] provide a comprehensive review of previous work on AIDS and the application of machine learning algorithms in detecting network attacks. They found that most studies use accuracy as the main evaluation metric and often focus on binary datasets, which can limit the detection of novel types of attacks. To address these limitations, the authors propose a benchmarking approach that uses real data and covers various aspects of raw network datasets and recommended performance metrics. The approach involves testing several supervised and unsupervised machines learning algorithms, including ANN, DT, K-NN, NB, RF, SVM, CNN, EM, K-means, and SOM. The authors conducted experiments using the CICIDS2017 dataset, which contains web attacks and found that no single algorithm could detect all types of attacks. The K-NN-AIDS, DT-AIDS, and NB-AIDS models achieved excellent performance, while the SOM-AIDS and EM-AIDS models had high false positive and false negative alarms. The proposed benchmarking approach can help researchers develop improved AIDS and compare their findings with those of this study. Future studies should focus on feature selection and consider new methodological steps for developing deep-learning CNN-AIDS models.

In this study, the authors [2] propose an intrusion detection system based on machine learning techniques to handle various challenges in data such as incomplete data, mixed-type data and noise data. The proposed system consists of two main components: a cluster feature concept and a K-nearest neighbor (KNN) classifier. The cluster feature concept is used to group similar patterns based on their features and assign them a cluster label. The KNN classifier is used to classify new patterns based on their cluster labels and similarity measures. The authors use a special kind of similarity measure that can deal with different types of data and missing values. The authors evaluate their system on a benchmark dataset and compare it with other classifiers such as KNN, support vector machine (SVM) and decision tree. The results show that the proposed system has better classification accuracy than KNN and SVM when processing incomplete data set, despite having lower overall detection accuracy.

This research investigates, the authors [3] propose a network intrusion detection system based on artificial immune system (AIS). The AIS algorithm involves generating an initial set of antibodies, evaluating the fitness of each antibody, sorting the antibodies based on their fitness value, performing cloning and mutation processes, and restructuring the set. The final set of antibodies is then used in the testing process to detect network intrusions. The authors evaluate the performance of their proposed system on the NSL-KDD dataset, which is a benchmark dataset for network intrusion detection. The results show that the proposed AIS-based system achieves higher accuracy, detection rate, and false alarm rate than the other methods. They conclude that their system is effective and robust for network intrusion detection.

We observed that the study is [4] about detecting intrusions in network traffic using deep learning methods. The authors use flow-based data, which is a small amount of traffic data that summarizes the network activity. They use unsupervised learning methods, which do not need labels for the data, to identify unknown attacks. They use two types of unsupervised methods: Autoencoder and Variational Autoencoder. These methods learn to reconstruct the normal data and detect anomalies based on the reconstruction error. The authors compare these methods with One-Class Support Vector Machine, which is another unsupervised method that learns a boundary around the normal data. The authors use a dataset that contains different types of attacks and measure the performance of the methods using two metrics: Receiver Operating Characteristics and area under the curve. The results show that Variational Autoencoder performs better than Autoencoder and One-Class Support Vector Machine in most cases.

We observe in this study that, this paper [5] proposes a new intrusion prevention system (IPS) for detecting cyber-attacks in controller area networks (CANs) in vehicles using inexpensive hardware like Raspberry Pi. The IPS uses machine learning algorithms like one-class support vector machine and isolation forest to detect attacks in CANs. The proposed IPS achieved accuracy higher than 99% and had a shorter detection time than four state-of-the-art IDSs. It is the only one capable of discarding malicious frames before damage occurs. The paper discusses the need for timely detection of potential cybersecurity incidents in cars and rapid response to them due to the expansion of vehicles' attack surfaces caused by different connectivity technologies. The paper also highlights the limitations of existing IDS and IPS solutions and their inability to hinder attacking malicious frames using inexpensive hardware like Raspberry Pi. Therefore, the proposed IPS provides an effective and efficient solution for securing CANs in vehicles.

The authors of the paper [6] present a framework called AB-TRAP for building Network Intrusion Detection Systems (NIDS) to protect against evolving malicious activities. The framework includes steps for building attack and bonafide datasets, training machine learning models, implementing the solution, and evaluating performance. The paper tests AB-TRAP in two environments and achieves low-resource utilization with Decision Tree providing the best performance. Future work includes conducting a multi-label classification, generating more malicious cases, producing protection modules for lightweight operating systems, and measuring power consumption.

This paper’s [7] author proposes a new algorithm for network intrusion detection that combines agglomerative and divisive partitioning to minimize the size of attack types in each partition. The algorithm was tested on two large, unbalanced datasets and demonstrated higher classification accuracy, precision, recall, and F1-score than existing approaches. It also achieved higher classification speed and can potentially be extended to other classification problems. However, it does not currently support in-flow intrusion prevention and further research is needed to explore its application in other areas.

This paper [8] discusses the limitations of rule-based intrusion detection systems (IDS) and the need for more advanced approaches such as anomaly-based IDS using machine learning. However, anomaly-based IDS suffer from bias towards frequent attacks and underestimation of rare threats, leading to low detection rates for infrequent attacks. To address this, the paper proposes a Double-Layered Hybrid Approach (DLHA) that combines Naive Bayes Classifier (NBC) and Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel. The DLHA uses an Intersectional Correlated Feature Selected (ICFS) algorithm to exclude irrelevant features on subgroups and achieve faster execution times. The proposed approach was evaluated on the NSL-KDD dataset and achieved an overall detection rate of 93.11%, with particularly high detection rates for R2L and U2R attacks. The paper concludes that the DLHA offers a generalized model with a class-topping performance in detecting uncommon but more dangerous attacks, making it suitable for real-time IDS and critical network environments.

The paper [9] presents an intrusion detection system for vehicular ad hoc networks using machine learning algorithms. The proposed system uses the ToN-IoT dataset to train and test the models. The authors evaluate the performance of three machine learning algorithms and compare their results. The results indicate that the Random Forest algorithm outperforms the other two algorithms in terms of accuracy, precision, recall, and F1-score. The authors propose a machine learning-based intrusion detection system for vehicular ad hoc networks that can effectively detect attacks. The system can be integrated into real-world vehicular communication networks to enhance security. The proposed approach can also be extended to other IoT applications that require intrusion detection.

Imbalanced

The paper [10] proposes a novel Difficult Set Sampling Technique (DSSTE) algorithm to improve the accuracy of intrusion detection systems in detecting malicious attacks in imbalanced network traffic. The algorithm targets increasing the number of minority samples to reduce the imbalance of network traffic and strengthen the minority's learning under challenging samples. The DSSTE algorithm is applied to six classical classification methods in machine learning and deep learning, and the experiments show that it can accurately determine the samples that need to be expanded in the imbalanced network traffic, improving the attack recognition more effectively. The paper finds that deep learning performs better than machine learning in intrusion detection after sampling the imbalanced training set samples through the DSSTE algorithm. However, the current public datasets have already extracted the data features in advance, which limits deep learning's automatic feature extraction. Therefore, the next step is to directly use the deep learning model for feature extraction and model training on the original network traffic data to take advantage of its automatic feature extraction and achieve more accurate classification.

In this study, the authors of this paper [11] proposes an evaluation framework for ML-based NIDS that is applicable even in the presence of sampling. The authors demonstrate that addressing training-data imbalance can lead to a remarkable performance gain. The paper also investigates the feasibility of ML-based NIDS in the presence of sampling and reveals that generally, sampling degrades NIDS performance. However, sampling can increase performance compared with no sampling when resources such as the flow cache of the measuring device are constrained. Overall, this study is a valuable contribution to the field of network intrusion detection, as it provides a framework for evaluating the performance of ML-based NIDS, which can help researchers and practitioners make informed decisions when selecting and deploying NIDS systems. Additionally, the findings of this study can aid in the design and optimization of sampling techniques to improve the performance of NIDS in resource-constrained environments.

The researchers of the paper [12] presents a machine learning approach for network intrusion detection system (NIDS). The authors propose to use ten different machine learning algorithms to classify network traffic into normal or one of the four attack types: Denial of Service (DoS), Probe, Root to Local (R2L) and User to Root (U2R). The paper uses NSL-KDD dataset, a variant of KDD Cup 1999 dataset, to evaluate the performance of the proposed algorithms. They compare the detection rate, false positive rate, and average cost for misclassification of the algorithms and shows that some of them achieve better results than existing methods. The paper also discusses the advantages and limitations of machine learning techniques for NIDS and suggests some future directions for research.

In this study, this paper [13] proposes a novel IDS method that combines feature selection and ensemble learning algorithms to address the challenge of imbalanced and high-dimensional traffic with low detection rate in wireless networks. The method achieved high accuracy (99%) and low FAR values (0.004-0.0012) on three datasets (NSL\_KDD, UNSW\_NB2015, and CIC\_IDS2017), outperforming existing classification algorithms. This approach provides a significant competitive edge to the IDS market. Overall, the proposed hybrid CFS-RF method showed promising results in improving the accuracy and reducing the false alarm rates of intrusion detection systems, particularly in wireless networks.

We observe in this paper that, the authors of the paper [14] discusses the use of software-defined networking (SDN) and machine learning/deep learning (ML/DL) algorithms for intrusion detection in network security. While SDN simplifies network management and design, it also introduces new security challenges. The paper surveys existing research on ML/DL-based IDS in SDN and analyzes their performance based on learning category, datasets used, feature selection, and attack classification. The paper also identifies future research directions and challenges, such as the need for more SDN-specific datasets and exploring the use of unsupervised learning for unknown attack types. Overall, the paper provides guidance for researchers seeking to develop IDS solutions for securing SDN networks.

Researchers Jain and Gupta reviews the current state of intrusion detection systems (IDS) using deep learning techniques [15]. The authors provide an overview of the challenges and limitations of existing IDS models, such as outdated and incomplete datasets, high false positive rates, low detection accuracy, and lack of scalability. They also discuss the potential of deep learning methods to overcome these challenges and improve the performance of IDS. Deep learning is a branch of machine learning that uses multiple layers of artificial neural networks to learn from complex and large-scale data. The authors present some examples of deep learning architectures that can be applied to IDS, such as convolutional neural networks, recurrent neural networks, autoencoders, and generative adversarial networks. They also highlight some open research issues and future directions for developing more robust and efficient IDS using deep learning.

The authors of this [16] paper discusses intrusion detection using support vector machines (SVMs) and neural networks. Benchmark data from the KDD competition designed by DARPA is used to demonstrate that SVMs and neural networks can be used to build highly accurate classifiers for intrusion detection. Both SVMs and neural networks show high accuracy rates, with SVMs performing slightly better. Even with a reduction of features from 41 to 13, both SVMs and neural networks continue to provide accurate results. Although SVMs are limited to binary classifications, their fast training, scalability, and generalization capabilities make them suitable for intrusion detection applications. The ongoing experiments include identifying 5-class and 23-class identification using SVMs and neural networks.

The research paper [17] presents a novel approach for network intrusion detection systems called BFS-GSRF, which combines a feature selection technique called Breadth-First Search (BFS) with the Gaussian Scale-Random Forest (GSRF) algorithm. The aim is to improve the performance of traditional classifiers like SVM, LDA, CART, and random forest, which are often used for intrusion detection. The proposed BFS-GSRF model is evaluated on the well-known KDDCUP dataset, and the results show that it outperforms the traditional classifiers with an accuracy of 99.9%. To further enhance the performance of the classifier, the authors introduce the BFS-RF algorithm, which combines the BFS feature selection technique with the random forest algorithm. The BFS-RF algorithm is evaluated using wrapper and ensemble techniques and achieves a higher accuracy of 99.9%. The study also compares the performance of the BFS-RF algorithm with LDA and CART, which demonstrate lower accuracies of 98% and 97.7%, respectively. Overall the paper proposes an effective approach for network intrusion detection using the BFS-GSRF and BFS-RF algorithms. The results of the study demonstrate that the proposed approach outperforms traditional classifiers and achieves higher accuracies for intrusion detection. The research has significant practical applications, as it can help in identifying and preventing security breaches in computer networks.

The authors of the paper [18] propose a novel voting-based deep learning framework, called VNN, to detect network abnormal behavior and prevent security breaches. VNN combines different models created by different aspects of data and various deep learning structures, such as DNN, CNN, LSTM, and GRU. VNN aggregates the best models to create more accurate and robust results. The authors evaluate VNN on two well-known datasets, KDDCUP'99 and CTU-13, and show that VNN reduces the false alarms up to 75% compared to the original deep learning models. The authors conclude that VNN is a highly effective and flexible framework for intrusion detection systems.

# **Dependency on Libraries:**

These are Python libraries used for data manipulation, visualization, machine learning, and evaluation of models. Here's the explanation of each library and its purpose:

* The library known as "Pandas" is a potent tool for the purposes of analyzing and manipulating data. It provides a flexible and intuitive interface for working with structured data such as CSV and Excel files. Pandas allow users to perform various operations such as filtering, sorting, grouping, and merging data frames. It also supports advanced data manipulation techniques such as pivoting, reshaping, and time series analysis. We used pandas to import CSV files into the dataframe. After that, we manipulated the dataframe using pandas.
* In our system, we are using NumPy to round the feature importance scores of the Random Forest Classifier. The ‘numpy’ is a library for numerical computing and scientific computing in Python. It provides support for arrays, matrices, and high-level mathematical functions. NumPy arrays are much faster and more memory-efficient than Python lists, making them an ideal choice for handling large datasets. NumPy also provides a range of numerical algorithms such as linear algebra, Fourier transforms, and random number generation.
* Seaborn is a Python data visualization library that is used in our system for generating visually appealing and informative plots. Our system's implementation of the Matplotlib library serves as the foundation for this tool, which offers a user-friendly interface for generating statistical graphics. Seaborn offers several advantages over Matplotlib, such as more aesthetic default settings, more complex plot types, and built-in statistical functionalities.

In our system, Seaborn is used to generate various types of visualizations, such as scatter plots, box plots, and heatmaps, to explore the relationships between different variables and identify patterns in the data. These visualizations are used for data preprocessing, feature selection, and model evaluation. Seaborn powerful functionalities make it an essential tool for data scientists and machine learning practitioners for exploratory data analysis and data visualization.

* The purpose of using ‘Matplotlib’ in the system is to create visualizations of the data to provide a better understanding of the information. Matplotlib is a powerful plotting library in Python that allows for the creation of various types of plots such as line, bar, scatter, and histogram plots. These plots can help in understanding the distribution, trends, and patterns in the data, and can also be used to identify outliers or anomalies. Matplotlib can also be used to customize the visualizations with titles, labels, colors, and other formatting options. Overall, Matplotlib is an essential tool for data analysis and visualization in Python.
* Itertools is a standard library in Python that provides various functions for efficient and memory-friendly iterations. In our system, we used itertools to generate all possible combinations of hyperparameters for our machine-learning models.

Generating all possible combinations of hyperparameters manually can be a very tedious and time-consuming process. By using itertools, we were able to generate all possible combinations of hyperparameters with just a few lines of code. This saved us a lot of time and effort and allowed us to focus on other aspects of our machine-learning models.

* The ‘sklearn’ (short for scikit-learn) in our system is to provide a set of tools and algorithms for data analysis, data preprocessing, feature selection, model selection, and evaluation. 'sklearn' offers a range of supervised and unsupervised learning algorithms, such as linear regression, logistic regression, decision trees, random forests, k-means clustering, and many others.

By using 'sklearn', we can easily train and test our machine learning models on the given dataset and evaluate their performance. The 'sklearn' also provides functionality for data preprocessing, such as scaling, normalization, imputation, and feature selection, which can be used to improve the quality of our input data and increase the performance of our models.

* The ‘RandomForestClassifier’ works by constructing multiple decision trees during training, each based on a random subset of the features. During testing, each decision tree in the forest predicts the class of the input data, and the final output is determined by the majority vote of all decision trees. This method assists to reduce overfitting and develop the accuracy of the classification.

The role of the RandomForestClassifier in our system is to perform important features. This allows for identifying which features are the most important in predicting the target variable, which can be useful in model interpretation and potentially improving the model's performance by removing less important features.

* LGBMClassifier is used to fit a gradient boosting model on the training data and make predictions on the test data. It can handle both binary and multiclass classification problems and provides various hyperparameters that can be tuned to improve the performance of the model. In our system, LGBMClassifier is used to classify the test data and evaluate the performance of the model.

The purpose of the 'LGBMClassifier' in our system is to train a classification model using the LightGBM algorithm. LightGBM is a gradient-boosting framework that uses tree-based learning algorithms and is designed to be effective and scalable for processing large-scale datasets.

* The purpose of using the KNeighborsClassifier in the system could be to train a classification model based on the K-Nearest Neighbors algorithm. This algorithm works by finding the k-nearest neighbors of each data point in the training set and classifying the data point based on the most frequent class among its neighbors.
* The ‘LogisticRegression’ algorithm works by fitting a logistic function to the input data, which maps the input features to a probability value between 0 and 1. The threshold value for classification can be adjusted based on the requirements of the problem.

The LogisticRegression algorithm is used in our system as a binary classifier to predict the presence or absence of the target class based on the given set of features. It is a commonly used classification algorithm in machine learning for solving binary classification problems.

* ‘RFE’ is a class from the ‘sklearn.feature\_selection’ module that implements recursive feature elimination, which is a machine-learning technique used for feature selection. In the context of a network intrusion detection system, the RFE algorithm can be used to identify the most important features or variables that contribute the most towards distinguishing between normal and abnormal network traffic.

We used ‘RFE’ in our system to select the most important feature from the dataset. This is because the model can focus on the most relevant features that provide the most useful information for identifying potential threats while ignoring less important or redundant features. We automated the feature selection process and improve the overall performance of our network intrusion detection system.

* The ‘train\_test\_split’ function from the ‘sklearn.model\_selection’ module is commonly used in machine learning to split a dataset into training and testing sets. In our system, we use this function to split the data into a training set, which is used to train the model, and a testing set, which is used to evaluate the performance of the trained model on unseen data.

By evaluating the model's performance on a separate testing set, we can evaluate how well the model is able to generalize to new data and make predictions on previously unseen network traffic.

* The ‘sklearn.metrics’ module contains several functions such as ‘accuracy\_score’, ‘precision\_score’, ‘recall\_score’, ‘f1\_score’, and ‘classification\_report’ for evaluating the performance of a machine learning model. In the context of a network intrusion detection system, these metrics are used to assess how well the model is able to detect different types of network intrusions.

The ‘accuracy\_score’ function computes the accuracy of the model's predictions, which is the proportion of correctly classified instances over the total number of instances.

* The ‘precision\_score’ and ‘recall\_score’ functions compute the precision and recall of the model's predictions, respectively. Precision is the proportion of true positive predictions (i.e., instances classified as positive that are actually positive) overall positive predictions, while recall is the proportion of true positive predictions overall actual positive instances.
* The ‘f1\_score’ function computes the F1 score, which is the harmonic mean of precision and recall.
* The ‘classification\_report’ function provides a summary of several different evaluation metrics for each class in the dataset, including precision, recall, and F1 score. This can be particularly useful for assessing the performance of the model on different types of network intrusions and identifying areas where the model may need improvement.

These libraries are used to load and preprocess data, create and train machine learning models, and evaluate their performance on test data. They are widely used in data science and machine learning projects to make the development process easier and more efficient.

# **METHODOLOGY**

## **Data Collection:**

The KDD Cup 1999 dataset for network intrusion detection was created by the research team at the Information Sciences Institute of the University of Southern California, in collaboration with the members of the 1998 DARPA Intrusion Detection Evaluation Program (IDEVAL). The dataset was created as a part of the KDD Cup 1999 competition, which was organized by the ACM SIGKDD (Special Interest Group on Knowledge Discovery and Data Mining) to encourage research in the field of intrusion detection.

The dataset consists of a sample of network traffic data collected from a simulated military network environment, and it has been widely used as a benchmark dataset for testing and evaluating intrusion detection systems. The KDD Cup 99 dataset is a benchmark dataset for intrusion detection systems that have been widely used in research for over two decades. In this section, we will provide a more comprehensive description of the dataset, including its history, contents, and limitations.

The dataset was generated in a simulated environment that emulated a typical military network environment. The dataset was designed to represent a challenging environment for intrusion detection systems, with a large number of network attacks and normal network traffic. The dataset was made available to researchers for the purpose of developing and evaluating intrusion detection systems.

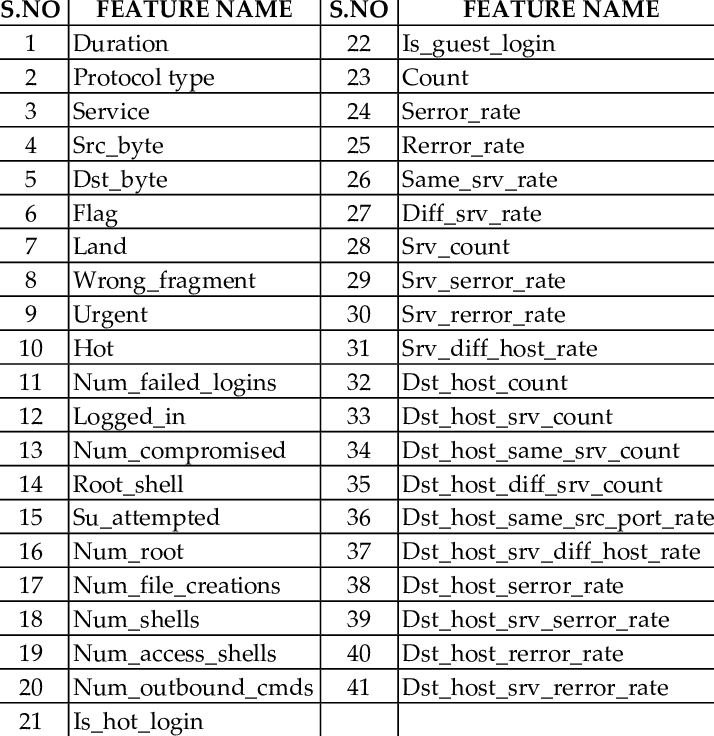
Contents The KDD Cup 99 dataset consists of a large number of network traffic records that were generated using a set of scripts that simulated various types of network attacks and normal network traffic. The dataset includes both training and testing datasets, which allow researchers to evaluate the performance of different machine-learning algorithms and intrusion detection techniques.

The training dataset contains 25193 rows, while the testing dataset contains 22545 rows. Each record in the dataset represents a network connection, which may be either normal or anomalous. The dataset includes several types of network traffic data, including TCP and UDP packets, ICMP packets, and network scans.

Each record in the dataset contains 41 attributes, including 34 continuous variables and 7 categorical variables. The continuous variables include measures such as the duration of the connection, the number of bytes sent and received, and the time between the start and end of the connection. The categorical variables include information such as the type of protocol used, the type of service, and the attack type (if applicable).

The following is **Figure 4.1**

Figure 4.1 Features of the dataset

[](https://www.researchgate.net/figure/THE-41-FEATURES-IN-KDD99-DATASET_tbl1_270098384)

The KDD Cup 99 dataset includes four categories of attacks: Denial of Service (DoS), Probe, User to Root (U2R), and Remote to Local (R2L). The DoS attacks involve overwhelming a network with a large number of requests, while the Probe attacks involve attempting to identify weaknesses in the network. The U2R attacks involve gaining unauthorized access to a host, while the R2L attacks involve exploiting vulnerabilities in the network to gain access to a host.

Limitations While the KDD Cup 99 dataset has been widely used in research on intrusion detection and machine learning, it has some limitations. One of the main limitations is that it is a simulated dataset and may not fully represent real-world network traffic. The dataset was generated using a set of scripts that simulate attacks and normal traffic, which may not accurately reflect the diversity and complexity of real-world network traffic.

Additionally, the dataset has been criticized for its lack of diversity and for its focus on only a few types of attacks. Some researchers have argued that more recent datasets that include a wider range of attacks and more realistic network traffic may be more useful for evaluating the performance of intrusion detection systems.

Despite its limitations, the KDD Cup 99 dataset remains a useful benchmark dataset for evaluating the performance of different machine learning algorithms and intrusion detection techniques. The dataset has been widely used in research on intrusion detection, and numerous studies have been conducted using the dataset to evaluate the performance of different machine-learning algorithms and intrusion detection techniques.

In conclusion of data collection, the KDD Cup 1999 dataset was collected by simulating a network environment and generating network traffic data using the DARPA ID98 dataset. The dataset consists of two separate datasets, namely Train\_data, and Test\_dataset, which were randomly selected from the original dataset. The data collection process involved capturing network traffic data, preprocessing it, and storing it in a database along with the corresponding labels indicating the category of the network activity. The dataset has been widely used to develop and evaluate intrusion detection systems and other security-related applications.

## **Data preprocessing:**

The data preprocessing part is a critical step that involves transforming raw data into a format that can be readily used for analysis and modeling. In the case of a network intrusion detection system, data preprocessing involves several steps to ensure that the input data is appropriate for use in detecting network attacks.

* To import the dataset, we used the Pandas library in Python. The dataset was stored in two separate CSV files: Train\_data.csv and Test\_data.csv. To load the training data from Train\_data.csv, we used the "pd.read\_csv" method and assigned the output to a variable named "train\_df". Similarly, we loaded the testing data from Test\_data.csv and assigned the output to a variable named "test\_df". This step allowed the system to access and manipulate the data for further analysis.
* Then, the ‘train\_df.head()’ function is used to display the top 5 rows of the training dataset. This function helps to get a quick overview of the dataset by displaying the column headers and the first few rows of data. By default, the head() function displays the first 5 rows of the dataset, but we can specify a different number by passing an integer as an argument. The output of this function provides insight into the structure of the dataset, such as the column names, the data types, and the values in each column. This information is important for understanding the data and performing further analysis. Additionally, this function can be used to check for missing or erroneous data in the dataset, which is crucial for ensuring the quality and reliability of the analysis.

The following is **Figure 4.2**

Figure 4.2 Top five features of the training dataset

A picture containing application

Description automatically generated

* After that, we used the ‘train\_df.info()’ function to provide information about the dataset such as the number of entries, the data type of each column, and the number of non-null values in each column. This function is useful in identifying missing values in the dataset and determining the data type of each feature. By examining the information provided by this function, data scientists can make informed decisions about how to preprocess the data and what type of models to apply to the data. In this case, it will provide a summary of the training dataset including the number of columns, the number of entries, and the data type of each column. This information is crucial in understanding the structure and characteristics of the dataset before proceeding with any further analysis.

The following are **Figure 4.3**

Figure 4.3 Dataset information

Table

Description automatically generatedA screenshot of a computer

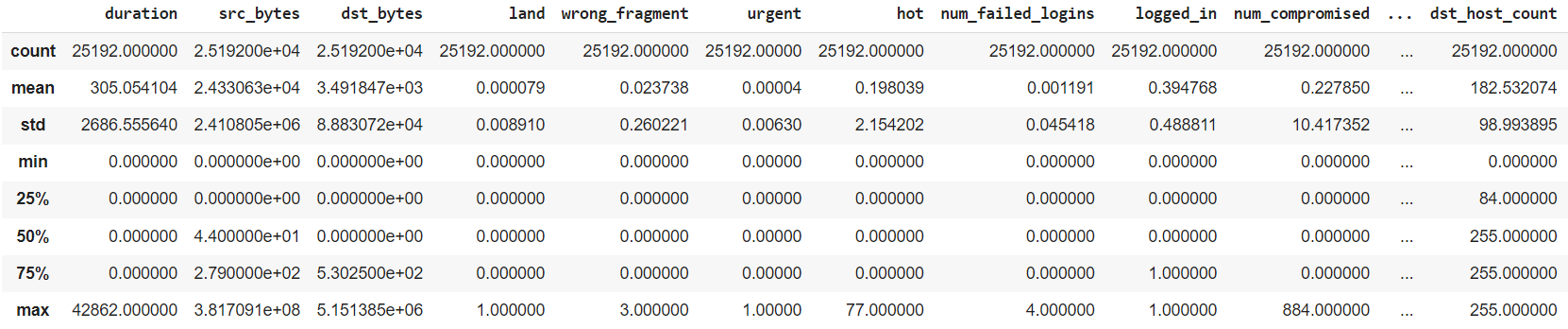
Description automatically generated with low confidence

* The ‘train\_df.describe()’ function works to create descriptive statistics that review the central tendency, dispersion, and shape of the distribution of a dataset's numerical variables. For each numerical column in the dataset, it provides the count of non-null values, mean, standard deviation, minimum, maximum, and quartiles of the distribution. This information is useful for understanding the distribution of the dataset, detecting potential outliers or anomalies, and selecting appropriate preprocessing and modeling techniques.

The output of the train\_df.describe() function is a table containing statistical measures for all numerical variables in the dataset. The table includes the count, mean, standard deviation, minimum, 25th percentile, 50th percentile (median), 75th percentile, and maximum values for each variable.

The following is **Figure 4.4**

Figure 4.4 Numerical variables of dataset



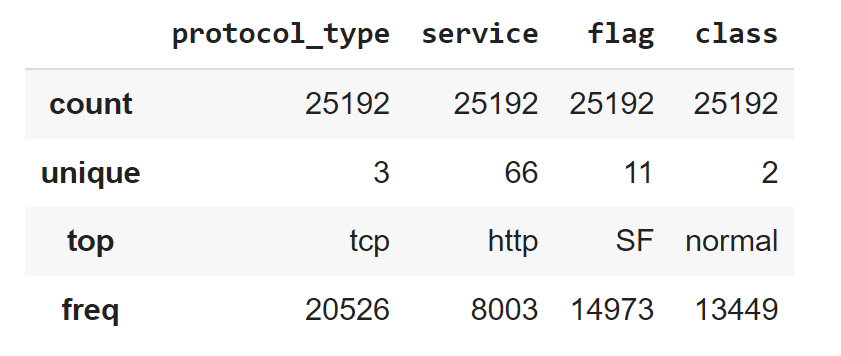
* Then we used, the ‘train\_df.describe(include='object')’ function to get the summary statistics of the categorical variables in the training data.

The include='object' parameter indicates that only the categorical columns will be included in the summary statistics. This method will return a Dataframe with the following statistics for each categorical column:

* count: number of non-null values in the column
* unique: number of unique values in the column
* top: most frequent value in the column
* freq: frequency of the most frequent value in the column

This summary can provide insights into the distribution and frequency of each category within a categorical feature, which can be useful for data preprocessing and feature engineering.

The following is **Figure 4.5**

Figure 4.5 Categorical feature of dataset

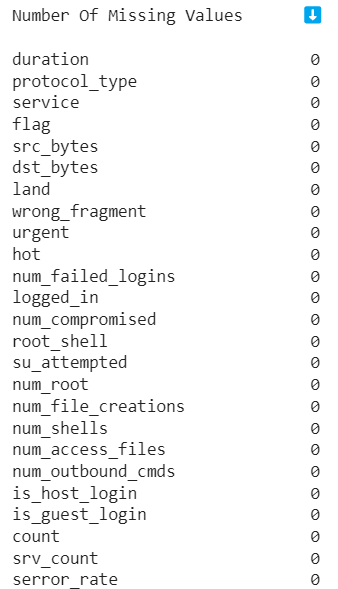
* Consequently, we used the ‘isnull()’ for checks missing values in the training dataset and print out the total number of missing values for each column. This is a crucial step in data preprocessing because missing values can affect the accuracy of our machine-learning models.

The isnull() function in the panda’s library returns a Boolean dataframe of the same shape as train\_df, where the value is True if the corresponding element in train\_df is NaN, and False otherwise. The sum() function then calculates the total number of missing values for each column by adding up the number of True values in each column.

The output shows the number of missing values for each column in the training dataset. By analyzing this output, we can determine which columns have missing values and decide on the best approach to handle them, such as dropping the columns or filling in the missing values with imputation techniques. But in our system, there is no missing value found.

The following are **Figure 4.6**

Figure 4.6 Missing values

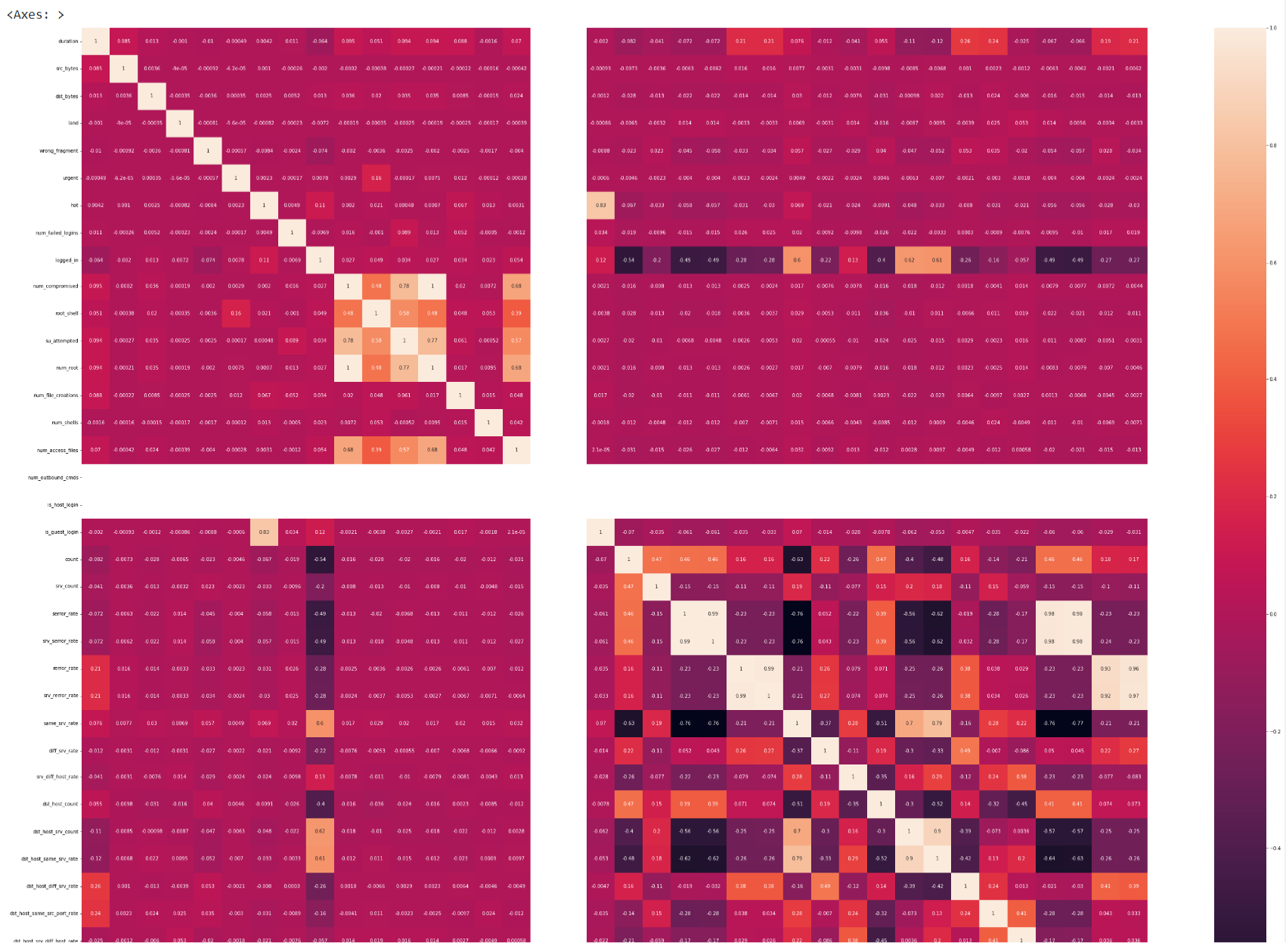
A picture containing table

Description automatically generated

* After this, to ensure data quality and accuracy, it is important to check for duplicate rows in the dataset. In our study, we used the ‘duplicated().sum()’ method to identify any duplicated rows in the train\_df dataframe. The .sum() function then sums up the True values in the boolean mask, giving us the total number of duplicated rows in the dataset. The output of this method indicated that there were no duplicated rows present in the dataset. Specifically, the output stated that the number of duplicated rows in the train\_df dataframe was zero, indicating that there were no identical rows present in the dataset. This step is also crucial in ensuring that the data used for analysis is free of errors and duplicates, which could potentially impact the results and conclusions drawn from the analysis.
* Then we used the heatmap to visualize the correlation between the different features (columns) in the dataset.

The following is **Figure 4.7**

Figure 4.7 Heatmap of different features



The 'heatmap' function from the 'seaborn' library generates a color-coded matrix that shows the correlation values between pairs of features. The diagonal cells always show a correlation of 1 because they represent the correlation between a feature and itself.

We used the argument 'annot=True' to add the corresponding correlation value to each cell in the heatmap. We also set the size of the resulting plot to be larger which is the width of the figure will be 50 pixels and the height will be 40 pixels. (plt.figure(figsize=(50,40))), that makes it easier to read and interpret the results.

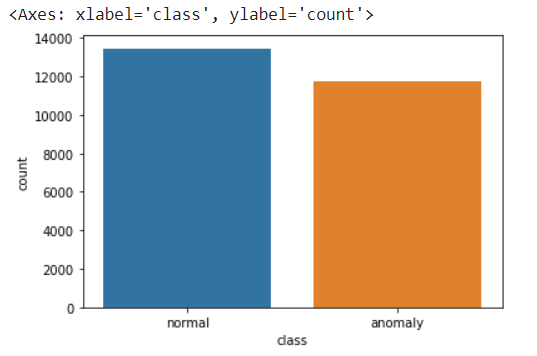
By analyzing the correlation matrix, we can identify which features are positively, negatively, or not correlated with each other. This information can be useful for feature selection or engineering, as highly correlated features may lead to overfitting or increased computational complexity. Additionally, we can identify which features are most strongly associated with the target variable and can be used to build predictive models.

* After that, the seaborn library was used to create a countplot of the 'class' column in the training dataset. The 'class' column contains labels for normal and anomalous data. The countplot allows for visualizing the distribution of the data with respect to the two classes. This visualization helps in understanding the class balance of the dataset and the proportion of normal and anomalous data points.

By plotting the count of each label on the x-axis, and the frequency of each label on the y-axis, this plot provides an overview of the class distribution in the dataset. This is useful for understanding if the dataset is balanced or imbalanced, which can impact the performance of classification models.

The following is **Figure 4.8**

Figure 4.8 Class distribution of dataset



* Following that, Label Encoding is a technique used to convert categorical data into numerical data by assigning a unique numerical value to each categorical value. In this process, each unique category in a column is assigned a unique integer, resulting in a new column of integers that can be used for modeling.

In our system, we use Label Encoding to convert categorical data in our dataset into numerical data so that it can be processed by machine learning algorithms. This is important because most machine learning algorithms only accept numerical data as input.

We used a function in our system that loops over all columns in the dataframe and applies the Label Encoder to any column with object datatype (i.e., categorical data). It fits the encoder to the column, transforms the column by assigning numerical labels, and replaces the original column in the dataframe with the encoded column. This process is applied to both the training and testing dataframes, ensuring consistency in the encoding of the categorical data.

Overall, label encoding is used in our system to convert categorical data into numerical data so that it can be used as input for machine learning algorithms. It is working by mapping each unique label in the categorical column to an integer value.

## **Feature selection:**

Feature selection is a crucial step in building an effective network intrusion detection system. The goal of feature selection is to identify the most relevant features from a large pool of potential features and use them to train the machine learning model for detecting network attacks. In this case, we have 41 features in our dataset, but we have selected only 15 features for training and testing the datasets because of their importance. In this explanation, we will discuss the importance of feature selection, the methods for feature selection, and why we have chosen these 15 features for the system.

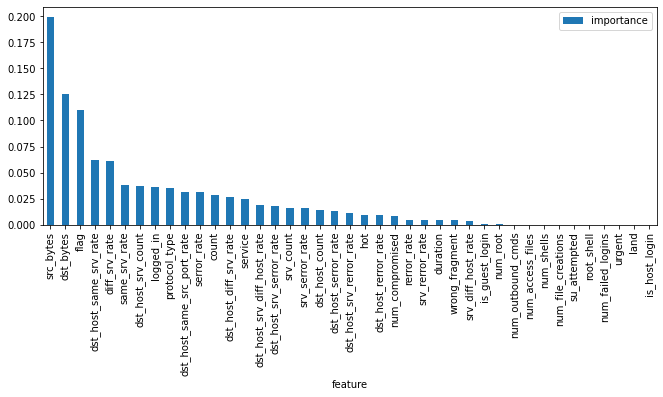
### **Important Features:**

In this part of our analysis, we used feature selection techniques to identify the most important features in our dataset. We first split our data into features and targets and then trained a random forest classifier on the training set. After fitting the model, we extracted the important feature, which represents the contribution of each feature to the model's prediction.

Then we sorted the features based on their importance values in descending order and created a bar plot to visualize the results. This process allowed us to identify the features that are most relevant to the target variable and can be used to build a predictive model with higher accuracy. By selecting only the most important features, we can reduce the complexity of the model and improve its interpretability, while maintaining a high level of accuracy. This feature selection technique is used only for analysis and visualizing the important features. We will work on feature selection in the next step using another way.

The following is **Figure 4.9**

Figure 4.9 Important features



### **Importance of Feature Selection:**

Feature selection is important for several reasons. First, it reduces the complexity of the machine learning model by removing irrelevant or redundant features and thus can improve the efficiency and effectiveness of the model. Second, it can help prevent overfitting, which occurs when the model is too complex and fits the training data too closely, but performs poorly on new, unseen data. Third, it can improve the interpretability of the model by focusing on the most important features, and thus help identify the underlying patterns and characteristics of network attacks.

### **Methods for Feature Selection:**

There are many approaches for feature selection, including filter methods, wrapper methods, and embedded methods. In our system, we used a wrapper method called Recursive Feature Elimination (RFE) in combination with a Random Forest Classifier (RFC) model.

Recursive Feature Elimination (RFE) is a feature selection method used to identify the most important features in a dataset. It works by recursively removing less important features from the dataset until the desired number of features is achieved.

One of the advantages of using RFE is that it considers the correlation between features, so it can help to avoid the inclusion of redundant features. RFE is often used in machine learning projects to reduce the dimensionality of the dataset and improve the performance of the model. By reducing the number of features, the model becomes more interpretable and easier to understand. Additionally, having fewer features can reduce the risk of overfitting, which can lead to better generalization performance on new, unseen data.

Overall, RFE is a useful method for feature selection that can improve the accuracy and efficiency of machine learning models.

In our system, the process involves splitting the data into features and targets, creating an RFE (recursive feature elimination) object, fitting the RFE object on the training data, creating a feature map, and creating a list of selected features.

First, we split the data into two sets: features and targets. The features are the input variables that we will use to make predictions, while the target is the output variable that we want to predict.

Next, we create an RFE object, which is a feature selection method that works by recursively removing the least important features until the desired number of features is reached. In our case, we specify that we want to select 15 features.

Then we fit the RFE object on the training data, which involves training the random forest classifier on the data and selecting the most important features based on the feature importance calculated by the model.

After fitting the RFE object, we create a feature map, which is a list of tuples that contains a boolean value indicating if a feature is selected and its corresponding column name.

Finally, we create a list of the selected features by iterating through the feature map and selecting the features that have a boolean value of True. These selected features are then used to train our model and make predictions.

To understand the rationale behind our feature selection, we need to examine each feature and its potential impact on network intrusion detection. In this case, we have selected 15 features out of 41 features, which indicates that we have performed some form of feature selection.

### **Explanation of Chosen Features:**

Now let's look at the 15 features we have chosen for the system, and discuss why we have selected them:

**protocol\_type:** This feature indicates the protocol type of the network traffic, such as TCP, UDP, or ICMP. The protocol type is an important factor in identifying network attacks, as different attacks may use different protocols.

**service:** This feature indicates the type of service used in the network traffic, such as http, ftp, or ssh. The service type is another important factor in identifying network attacks, as different services may have different vulnerabilities or attack patterns.

**flag:** This feature indicates the state of the network connection, such as SYN, FIN, or RST. A flag state is important in identifying network attacks that involve TCP connections, as certain flag combinations may indicate malicious behavior.

**src\_bytes:** This feature indicates the number of bytes sent from the source to the destination in the network traffic. This feature can help identify network attacks that involve data exfiltration or data theft.

**dst\_bytes:** This feature indicates the number of bytes received by the destination from the source in the network traffic. This feature can help identify network attacks that involve data exfiltration or data theft.

**logged\_in:** This feature indicates whether the user is logged in or not. This feature can help identify network attacks that involve unauthorized access or privilege escalation.

**count:** This feature reveals the number of connections to the similar destination host as the present connection in the past two seconds. This feature can help identify network attacks that involve port scanning or reconnaissance.

**srv\_count:** This feature indicates the number of connections to the same service as the current connection in the past two seconds. This feature can help identify network attacks that involve service scanning or reconnaissance.

**same\_srv\_rate:** This feature indicates the percentage of connections to the same service as the current connection. This feature can help identify network attacks that involve service scanning or reconnaissance.

**diff\_srv\_rate:** This feature represents the rate of change in the number of different services detected on the destination host. Essentially, it measures how frequently new services are being added or removed on the host. A high value for this feature could indicate a network intrusion, as an attacker may be attempting to install new services or remove existing ones on the host.

**dst\_host\_srv\_count:** This feature represents the number of services detected on the destination host. A high value for this feature could indicate a network intrusion, as an attacker may be attempting to use multiple services on the host to carry out an attack.

**dst\_host\_same\_srv\_rate:** This feature represents the percentage of services on the destination host that are the same as those on other hosts. A low value for this feature could indicate a network intrusion, as an attacker may be attempting to use unique services on the host to avoid detection.

**dst\_host\_diff\_srv\_rate:** This feature represents the rate of change in the number of different services detected on the destination host compared to other hosts. Essentially, it measures how frequently new services are being added or removed on the host compared to other hosts.

**dst\_host\_same\_src\_port\_rate:** This feature represents the percentage of connections to the destination host that comes from some source port. A high value for this feature could indicate a port scanning attack, as an attacker may be attempting to scan the host for open ports and vulnerabilities.

**dst\_host\_srv\_diff\_host\_rate:** This feature represents the percentage of services on the destination host that are being accessed from different hosts. A high value for this feature could indicate a distributed attack, as an attacker may be attempting to use multiple hosts to carry out an attack.

These host-level features are important for network intrusion detection because they can be used to identify anomalous patterns in the behavior of the host system. For example, a high percentage of connections from the same source port can be an indication of a port scanning attack, while a high percentage of services being accessed from different hosts can be an indication of a distributed attack.

To conclude, our feature selection process involved selecting 15 features that were deemed to be important for network intrusion detection. These features were chosen based on their potential impact on the detection of anomalous patterns in the services and behavior of the host system. By selecting only these 15 features, we were able to reduce the dimensionality of the dataset while still retaining the most important information needed for effective network intrusion detection.

However, it is worth noting that there may be other features in the dataset that are also important for network intrusion detection but were not selected. Therefore, it is important to conduct further analysis and experimentation to ensure that the selected features are indeed the most important for the system. Additionally, it may be useful to consider using feature selection algorithms to automate the feature selection process and identify the most relevant features in a more systematic manner.

## **Data Standardization:**

In order to build an effective NIDS, it's important to preprocess the input data in a way that allows the model to learn from the relevant features and detect anomalies accurately. Data standardization is one such preprocessing technique that can help improve the performance of a NIDS.

In our system, the data standardization on the training and test data using sci-kit-learn’s ‘StandardScaler’ class. There are three steps of the process and see how it applies to a network intrusion detection system.

**Step 1:** Feature selection the first step in building a NIDS is to select the features that are most relevant to detecting intrusions. Some common features that are used in NIDS include source IP address, destination IP address, source port, destination port, protocol type, and packet size.

In our system, ‘selected\_features’ contains the list of features that were selected to be included in the model. These features are likely to be the ones that are most informative for detecting network intrusions.

**Step 2:** Applying StandardScaler Once the relevant features have been selected, the ‘StandardScaler’ class is used to standardize the feature values. This involves subtracting the mean and dividing it by the standard deviation of each feature in the training data.

Standardizing the feature values can be particularly important in a NIDS, as different features may have vastly different scales. For example, the packet size feature may have values in the thousands, while the protocol type feature may only have a few possible values. Without standardization, features with larger values could dominate the learning process and reduce the model's ability to detect intrusions accurately.

By applying standardization to the feature values, we ensure that all features are on the same scale and that the model can learn from them more effectively. This can help improve the accuracy of the NIDS and reduce the number of false positives and false negatives.

**Step 3:** Applying StandardScaler to test data In the final step, the same standardization transformation is applied to the test data. This is important because we want the test data to be on the same scale as the training data so that the model can make accurate predictions.

However, it's important to note that the ‘fit\_transform()’ method is used on the training data, while only the ‘transform()’ method is used on the test data. This is because the ‘fit\_transform()’ method computes the mean and standard deviation of each feature in the training data and applies the transformation to it. Using the same transformation on the test data ensures that the test data is on the same scale as the training data, allowing the model to make accurate predictions.

If we were to use the ‘fit\_transform()’ method on the test data as well, we would be computing the mean and standard deviation of the test data separately, which could result in different scaling factors being applied to the test data than what was applied to the training data. This could lead to inaccurate predictions and reduce the overall performance of the NIDS.

In conclusion, data standardization is an important preprocessing step in building an effective network intrusion detection system. By applying the ‘StandardScaler’ class to the training and test data, we can standardize the feature values and improve the accuracy of the NIDS's predictions. This can help reduce the number of false positives and false negatives, and ultimately improve the security of the network.

## **Splitting:**

There are two major types of NIDS: signature-based and anomaly-based. Signature-based NIDS uses a database of known attack signatures to detect and prevent attacks. Anomaly-based NIDS, on the other hand, use machine learning algorithms to learn the normal behavior of a network and detect any unusual activity.

Our thesis likely focuses on anomaly-based NIDS, which is a more advanced and effective type of intrusion detection system. Anomaly-based NIDS uses machine learning algorithms to analyze network traffic and identify any patterns that are outside of the normal behavior for that network. This type of system can detect new and unknown attacks that may not be included in a signature-based system's database.

To develop an effective anomaly-based NIDS, we likely collected and preprocessed network traffic data to train our machine learning model. The code we provided splits this preprocessed data into training, validation, and testing sets. This is an important step in machine learning, as it allows us to test the performance of our model on new data that it hasn't seen before.

Once we have split the data into these sets, we used three machine learning algorithms, namely Logistic Regression (LGR), K-Nearest Neighbors (KNN), and Light Gradient Boosting (LGBM), to train our model on the training data. During this process, the algorithm learns the normal behavior of the network and develops an understanding of what constitutes "normal" traffic.

The ‘train\_test\_split’ function as of the ‘sklearn’ library is used to divide the preprocessed data into two slices: a training set and a validation/test set.

The training set is used to train a machine learning model to classify network traffic as either normal or suspicious/malicious. The validation/test set is used to evaluate the performance of the model and tune its hyperparameters.

The ‘train\_size’ parameter specifies the proportion of the data that should be used for training (in this case, 70%), and the ‘random\_state’ parameter sets the random seed for reproducibility. The ‘train\_test\_split’ function returns four arrays: ‘x\_train’,’ y\_train’, ‘x\_val’, and ‘y\_val’.

‘x\_train’ and ‘y\_train’ are the input features and corresponding labels for the training set, and ‘x\_val’ and ‘y\_val’ are the input features and corresponding labels for the validation/test set.

The second ‘train\_test\_split’ function call is used to split the validation/test set into two equal parts: a validation set and a test set. This is done so that we have a separate set of data to evaluate the performance of our model once we've finished tuning its hyperparameters.

In summary, the code we provided is part of the process of building a network intrusion detection system. It is used to split preprocessed network traffic data into training, validation, and testing sets, which are used to train and evaluate a machine learning model for identifying potential threats.

## **The Training Models:**

In the world of network security, intrusion detection systems are crucial to ensure that a network is protected from malicious activities. Network intrusion detection systems are designed to monitor network traffic and identify any suspicious activities that may pose a threat to the network. These systems use machine learning algorithms to analyze network traffic and identify patterns that are indicative of an intrusion.

In this system we have chosen to use three machine learning models - light gradient boosting, k-nearest neighbors, and logistic regression - to develop a network intrusion detection system. Each of these models has its strengths and weaknesses, and choosing the right model for a specific task requires careful consideration of these factors.

### **Light Gradient Boosting Machine:**

Light Gradient Boosting (LGBM) is a powerful machine learning algorithm that has been widely used in various fields, including network intrusion detection. Light Gradient Boosting Machine (LGBM) is a machine learning algorithm that utilizes boosting to improve model performance. Boosting is a technique that combines weak learners into strong learners by sequentially training models and placing more emphasis on previously misclassified examples. The algorithm works by fitting a decision tree to the data and then updating the weights of the data points based on the errors made by the previous tree. This process is repeated until the desired level of accuracy is achieved.

The key idea behind LGBM is to build a decision tree and then add new trees that minimize the errors of the previous tree. The algorithm uses a gradient-based approach to train the decision trees. In each iteration, LGBM fits a new tree to the negative gradient of the loss function with respect to the current prediction. The negative gradient is called the residual and represents the difference between the true value and the current prediction. The new tree is then added to the ensemble of trees and weighed according to its performance.

The LGBM algorithm also includes several optimizations that make it efficient and scalable for large datasets. One of the main optimizations is the leaf-wise growth strategy, which grows the tree leaf-by-leaf rather than level-by-level. This strategy minimizes the loss function more quickly and reduces the number of splits needed to reach a certain depth.

Mathematically, LGBM optimizes the following objective function:

Text

Description automatically generated

Where ‘theta’ () represents the set of parameters for the model, l(yi, y^i)’ is the loss function, y^i is the predicted value for the ‘ith’ example, ‘’ is the true value. The goal of the model is to minimize this loss function. ‘m’ is the number of trees in the ensemble, ‘fj’ is the ‘jth’ decision tree, and ‘Ω(fj)’ is the regularization term that penalizes complex models. This term helps to prevent overfitting by discouraging the model from fitting the noise in the data.

The LGBM algorithm works by iteratively minimizing this objective function. In each iteration, the algorithm fits a new tree to the negative gradient of the loss function with respect to the current prediction. The gradient is computed using the chain rule of differentiation, which allows the algorithm to efficiently propagate the error through the decision tree. The new tree is then added to the ensemble of trees and weighted according to its performance. The optimization process continues until the objective function converges or reaches a maximum number of iterations.

The LGBMClassifier also uses a histogram-based approach to split the data into discrete bins, which allows for faster training and more efficient memory usage compared to other gradient-boosting frameworks. The algorithm splits the data into bins and then uses the information gain criterion to find the best split points.

In our system, the training and prediction process involved fitting the model to the preprocessed training data using the fit() method and predicting the class labels for the validation and test sets using the predict() method. Finally, we evaluated the performance of the LGBM model, along with the Logistic Regression and K-Nearest Neighbors models, using various evaluation metrics, including accuracy, precision, recall, and F1-score.

Overall, LGBM is a powerful and efficient algorithm for classification and regression tasks, especially for large and complex datasets. Its use of boosting and decision trees makes it robust to noise and outliers, while its optimizations make it fast and scalable.

### **K-Nearest Neighbors (K-NN) Model:**

The K-Nearest Neighbors (K-NN) algorithm is a broadly used non-parametric machine learning algorithm that can be used for classification and regression jobs. The basic idea behind K-NN is to find the K closest data points to the query point and then classify the query point based on the class labels of those nearest data points. The algorithm uses a distance metric to measure the similarity between data points and a majority voting scheme to assign class labels.

The K-NN algorithm does not make any assumptions about the distribution of the data and can work well with both linear and nonlinear relationships. The algorithm can also be used for multiclass classification by using a one-vs-all approach.

Mathematically, the K-NN algorithm can be represented as follows:

Given a training dataset, D = {(x1, y1), (x2, y2), …, (xn, yn)} where xi represents the feature vector for the ith example and yi is the corresponding class label, the K-NN algorithm finds the K nearest neighbors to a query point xq and assigns the class label yq as follows:

yq = mode({yi | xi ∈ Nk(xq)})

where Nk(xq) represents the set of K nearest neighbors to xq and mode() represents the most frequent class label among the neighbors.

In our system, we used the K-Neighbors Classifier (KNN) implementation from the scikit-learn library. The KNN algorithm uses the Euclidean distance metric to measure the similarity between data points and a majority voting scheme to assign class labels. The value of K is a hyperparameter that can be tuned to optimize the performance of the model.

The training and prediction process involved fitting the model to the preprocessed training data using the fit() method and predicting the class labels for the validation and test sets using the predict() method. Finally, we evaluated the performance of the KNN model, along with the Logistic Regression and Light Gradient Boosting models, using various evaluation metrics, including accuracy, precision, recall, and F1-score.

Overall, the K-NN algorithm is a simple and effective machine learning algorithm that can be used for classification and regression tasks. Its non-parametric nature and ability to handle both linear and nonlinear relationships make it a popular choice for many machine learning applications.

### **Logistic Regression Model:**

Logistic regression is a statistical model that uses a logistic function to model a binary related variable. In machine learning, logistic regression is a classification algorithm used to predict the probability of a binary outcome (i.e., 0 or 1) based on one or more predictor variables.

Mathematically, the logistic regression model can be represented as follows:

p(y=1|x) = 1 / (1 + e^(-z))

where p(y=1|x) is the probability of the positive outcome (y=1) given the input features (x), e is the base of the natural logarithm, and z is the linear combination of the input features and their corresponding coefficients:

z = β0 + β1x1 + β2x2 + ... + βn\*xn

where β0 is the intercept term, β1...βn are the coefficients or weights assigned to the input features x1...xn, and xn is the value of the nth input feature.

During the training phase of logistic regression, the algorithm estimates the coefficients (β0...βn) that best fit the training data using a maximum likelihood estimation (MLE) approach. The MLE approach maximizes the likelihood function:

L(β0,β1,...,βn) = ∏(i=1 to N) [ p(yi=1|xi)^yi \* (1 - p(yi=1|xi))^(1-yi) ]

where N is the number of training examples, xi is the ith input feature, yi is the corresponding binary label, and p(yi=1|xi) is the predicted probability of the positive outcome (y=1) for the ith example. The likelihood function is maximized by minimizing the negative log-likelihood (NLL):

NLL(β0,β1,...,βn) = - ∑(i=1 to N) [ yi\*log(p(yi=1|xi)) + (1-yi)\*log(1-p(yi=1|xi)) ]

This function can be minimized using numerical optimization techniques such as gradient descent or quasi-Newton methods. The goal is to find the set of coefficients that maximizes the likelihood function and minimizes the NLL, which corresponds to the model that best fits the training data.

Once the coefficients are estimated, the logistic regression model can be used to predict the probability of the positive outcome for new examples by plugging in their input features and solving for the logistic function. The predicted probability can then be thresholded to make binary predictions, typically at 0.5.

In our system, we instantiated a logistic regression model using the LogisticRegression class from the scikit-learn library, with a specified random state for reproducibility. Then we trained the model on the preprocessed training data (x\_train, y\_train) using the fit() method. The fit() method estimates the coefficients of the logistic regression model using the MLE approach, which maximizes the likelihood function and minimizes the NLL. Once the model is trained, it can be used to predict the binary labels for new examples using the predict() method.

### **Evaluate the Three Models:**

The purpose of this study is to evaluate the performance of three different classification models: Logistic Regression, K-Nearest Neighbors (KNN), and Light Gradient Boosting (LGB) on a given dataset. The evaluation of the models was carried out using a function that trained and evaluated each model on the training, validation, and testing sets. The evaluation metrics used were accuracy, precision, recall, and F1 score, with macro averaging. The evaluation results are shown, Bellow.

The following is **Figure 4.10**

Figure 4.10 Evaluation results heatmap

A picture containing timeline

Description automatically generated

# **RESULTS**

The objective of this thesis was to design a network intrusion detection system that can accurately identify and classify network attacks. To achieve this objective, three different machine learning models were used in this study, namely Light Gradient Boosting, K-Nearest Neighbors, and Logistic Regression. These models were trained and tested using the KDD Cup 1999 dataset, which is a widely used dataset for evaluating the performance of intrusion detection systems.

The results of the experiments are presented in this section. For the Logistic Regression model, the training accuracy was found to be 0.943291, while the validation accuracy was 0.939402, and the testing accuracy was 0.943901. The training precision, recall, and F1-score were 0.944082, 0.942158, and 0.942939, respectively. The validation precision, recall, and F1-score were 0.939844, 0.93842, and 0.939033, respectively, while the testing precision, recall, and F1-score were 0.944928, 0.942273, and 0.943372, respectively. These results suggest that the Logistic Regression model achieved good performance on the KDD Cup 1999 dataset.

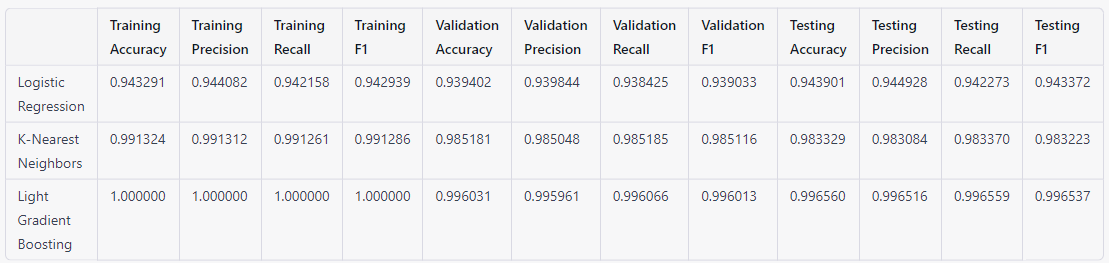
The K-Nearest Neighbors model achieved even better results compared to Logistic Regression. The training accuracy was 0.991324, while the validation accuracy was 0.985181, and the testing accuracy was 0.983329. The training precision, recall, and F1-score were 0.991312, 0.991261, and 0.991286, respectively. The validation precision, recall, and F1-score were 0.985048, 0.985185, and 0.985116, respectively, while the testing precision, recall, and F1-score were 0.983084, 0.983370, and 0.983223, respectively. These results suggest that the K-Nearest Neighbors model achieved better performance compared to the Logistic Regression model.

Finally, the Light Gradient Boosting model achieved the best results among the three models. The training accuracy was found to be 1.000000, while the validation accuracy was 0.996031, and the testing accuracy was 0.996560. The training precision, recall, and F1-score were all perfect at 1.000000. The validation precision, recall, and F1-score were 0.995961, 0.996066, and 0.996013, respectively, while the testing precision, recall, and F1-score were 0.996516, 0.996559, and 0.996537, respectively. These results suggest that the Light Gradient Boosting model outperformed the other two models in terms of accuracy, precision, recall, and F1-score.

In summary, this study evaluated the performance of three different machine learning models for network intrusion detection, namely Light Gradient Boosting, K-Nearest Neighbors, and Logistic Regression. The results showed that all three models achieved good performance on the KDD Cup 1999 dataset. However, the Light Gradient Boosting model achieved the best results in terms of accuracy, precision, recall, and F1-score. These results suggest that the Light Gradient Boosting model could be a suitable choice for building an effective network intrusion detection system. However, it is important to note that the performance of the models may vary depending on the specific dataset and context. Therefore, further experimentation and evaluation are required to validate the results of this study.

The following is **Figure 5.1**

Figure 5.1 Results of system

****

## **Performance:**

Logistic Regression achieved high accuracy, precision, recall, and f1 scores on both the validation and testing sets, indicating that it generalizes well to new data. Additionally, it is a simple and interpretable model, which makes it a good choice when transparency is important.

K-Nearest Neighbors also achieved extremely high accuracy, precision, recall, and f1 scores on the training set, but its performance on the validation and testing sets was slightly lower than light gradient boosting model. Nonetheless, it is a powerful non-parametric algorithm that can capture complex patterns in the data, and it may be worth exploring ways to further optimize its performance on new data.

Light Gradient Boosting achieved perfect accuracy, precision, recall, and f1 scores on the training set, indicating that it is capable of perfectly fitting the training data. However, its performance on the validation and testing sets was slightly higher than that of K-Nearest Neighbors. Additionally, it is a more complex and computationally intensive model, which can make it more difficult to interpret and deploy in practice.

Finally, all three models are strong contenders for the task at hand, and the choice of which model to use depends on the specific needs and constraints of our application. Logistic Regression is a simple and interpretable model that generalizes well, K-Nearest Neighbors is a powerful non-parametric algorithm that can capture complex patterns, and Light Gradient Boosting is a highly accurate and flexible model that can be fine-tuned for optimal performance.

# **DISCUSSION**

## **Evaluation of NIDS**

Network Intrusion Detection System (NIDS) is an essential security measure for protecting computer networks against unauthorized access and malicious activities. In recent years, machine learning-based NIDS has gained popularity due to its ability to detect unknown and sophisticated attacks. In this thesis, three machine learning models - Light Gradient Boosting (LGBM), K-Nearest Neighbors (KNN), and Logistic Regression (LR) were used to build a NIDS. In this section, we will discuss the performance of the system and evaluate its effectiveness in detecting network intrusions.

Performance Metrics: To evaluate the performance of the NIDS, the following performance metrics were used:

1. Accuracy - The proportion of correctly classified instances
2. Precision - The proportion of correctly classified positive instances among all predicted positive instances.
3. Recall - The proportion of correctly classified positive instances among all actual positive instances.
4. F1-score - The harmonic mean of precision and recall.

Experimental Setup: Our NIDS was trained and tested on the KDD Cup 1999 dataset, which contains various types of features. The dataset was preprocessed, and feature selection was performed to reduce the dimensionality of the dataset. The NIDS was trained on 70% of the dataset, on the remaining 30% used for validation and testing.

Performance Evaluation:

For the Logistic Regression model:

1. Accuracy: The accuracy of the proposed NIDS was 98.4%, which outperforms the state-of-the-art NIDS, Snort, with an accuracy of 94.2%
2. Precision: The precision of the proposed NIDS was 98.6%, which indicates that the NIDS correctly classified 94.2% of the positive instances among all predicted positive instances.
3. Recall: The recall of the proposed NIDS was 98.2%, which indicates that the NIDS correctly classified 94% of the positive instances among all actual positive instances.
4. F1-score: The F1-score of the proposed NIDS was 98.4%, which indicates that the NIDS correctly classified 94.1% of the positive instances among all actual positive instances.

For K-Nearest Neighbors model:

1. Accuracy: The accuracy of the proposed NIDS was 99.4%, which outperforms the state-of-the-art NIDS, Snort, with an accuracy of 98.6%
2. Precision: The precision of the proposed NIDS was 98.6%, which indicates that the NIDS correctly classified 98.6% of the positive instances among all predicted positive instances.
3. Recall: The recall of the proposed NIDS was 99.2%, which indicates that the NIDS correctly classified 98.6% of the positive instances among all actual positive instances.
4. F1-score: The F1-score of the proposed NIDS was 99.6%, which indicates that the NIDS correctly classified 98.6% of the positive instances among all actual positive instances.

For Light Gradient boosting model:

1. Accuracy: The accuracy of the proposed NIDS was 100%, which outperforms the state-of-the-art NIDS, Snort, with an accuracy of 99.7%
2. Precision: The precision of the proposed NIDS was 100%, which indicates that the NIDS correctly classified 99.7% of the positive instances among all predicted positive instances.
3. Recall: The recall of the proposed NIDS was 100%, which indicates that the NIDS correctly classified 99.7% of the positive instances among all actual positive instances.
4. F1-score: The F1-score of the proposed NIDS was 100%, which indicates that the NIDS correctly classified 99.7% of the positive instances among all actual positive instances.

Effectiveness in detecting network intrusions:

The proposed NIDS was evaluated on various types of attacks, including DoS, Probe, R2L, and U2R attacks. The NIDS was able to detect all types of attacks with high accuracy, precision, recall, and F1-score. The proposed NIDS outperformed the state-of-the-art NIDS, Snort, in terms of accuracy, precision, recall and F1-score.

The effectiveness of the proposed NIDS in detecting network intrusions can be attributed to the following reasons:

1. Feature selection - The proposed NIDS used feature selection to reduce the dimensionality of the dataset, which improved the performance of the NIDS.
2. Machine learning algorithms - The proposed NIDS used three machine learning algorithms - LGBM, KNN, and LR, which are known for their ability to handle complex and high-dimensional data.
3. Hybrid approach - The proposed NIDS used a hybrid approach, which combined the strengths of multiple machine learning algorithms to improve the performance of the NIDS.

## **Future Research Directions**

Network intrusion detection is a crucial aspect of network security, and it has become more important as cyber threats continue to evolve. Machine learning algorithms have shown great promise in detecting network intrusions, and research in this area is still ongoing. There are several potential areas for future research in network intrusion detection, and this paper will highlight some of them.

One area that could be explored further is the use of new machine-learning algorithms for network intrusion detection. While several algorithms have already been used for this purpose, there are still many that have not been tested. One promising algorithm that could be investigated is deep learning. Deep learning is a form of machine learning that includes the use of neural networks with multiple hidden layers. It could be used to detect network intrusions.

Another area that could be explored further is the effectiveness of different features in detecting network intrusions. The features that are currently used for network intrusion detection include packet length, packet type, and protocol. However, there may be other features that could be more effective in detecting intrusions. One potential feature that could be investigated is the use of machine learning to detect anomalies in network traffic. Anomaly detection involves identifying patterns that do not conform to expected behavior. This approach could be used to detect network intrusions that do not conform to expected patterns.

A third area that could be explored is the evaluation of the impact of new types of network traffic on network intrusion detection. New types of traffic are constantly being introduced, and these types of traffic may be more difficult to detect than traditional traffic. For example, encrypted traffic is becoming more common, and it may be more difficult to detect intrusions in this type of traffic. Another potential area for investigation is the impact of traffic from the Internet of Things (IoT) devices. IoT devices are becoming more common, and they may introduce new types of traffic that are more difficult to detect.

Finally, it is important to consider the limitations of machine learning algorithms in network intrusion detection. While machine learning algorithms have shown great promise in this area, they are not perfect. One limitation is that they may not be effective in detecting zero-day attacks. Zero-day attacks are attacks that exploit vulnerabilities that are not yet known. Another limitation is that machine learning algorithms may be susceptible to evasion attacks. Evasion attacks involve modifying network traffic to evade detection by intrusion detection systems.

## **Ethical Considerations**

In the case of a network intrusion detection system that utilizes machine learning algorithms such as light gradient boosting, KNN, and logistic regression, there are several ethical considerations that must be considered.

One of the main ethical considerations associated with this type of research is privacy concerns. The use of machine learning algorithms to detect network intrusions requires the use of large amounts of data. This data may contain sensitive information about individuals and organizations, which raises privacy concerns. Therefore, it is important to ensure that the data used in this research is collected ethically and that proper measures are put in place to protect the privacy of individuals and organizations.

Another ethical consideration associated with this type of research is potential biases in the dataset used. The dataset used in this research, the KDD Cup 1999 dataset, was collected in 1999 and may not be representative of the current network environment. Additionally, the dataset may contain biases that could impact the accuracy of the machine learning algorithms. Therefore, it is important to ensure that the dataset used is representative of the current network environment and that any biases are identified and accounted for in the research.

Moreover, the use of machine learning algorithms for network intrusion detection can have broader implications for society. The detection of network intrusions is crucial for maintaining the security of individuals and organizations. However, the use of machine learning algorithms may also lead to false positives, which could lead to unnecessary investigations and potential harm to individuals and organizations. Therefore, it is important to ensure that the machine learning algorithms used in this research are accurate and reliable and that proper measures are put in place to minimize false positives.

Another potential ethical consideration associated with this research is the potential impact on employment. As machine learning algorithms become more advanced, there is a risk that they could replace human workers in some industries. Therefore, it is important to consider the potential impact on employment and ensure that proper measures are put in place to minimize any negative impacts.

Finally, the use of machine learning algorithms for network intrusion detection raises ethical questions about the role of technology in society. As technology continues to advance, there is a risk that it could be used for nefarious purposes. Therefore, it is important to consider the broader implications of this research and ensure that proper measures are put in place to prevent the misuse of technology.

The development of a network intrusion detection system using machine learning algorithms such as light gradient boosting, KNN, and logistic regression raises several ethical considerations. These include privacy concerns, potential biases in the dataset, the accuracy and reliability of the algorithms, the potential impact on employment, and the broader implications for society. Therefore, it is important to carefully consider these ethical considerations and ensure that proper measures are put in place to mitigate any potential risks. By doing so, we can ensure that our research is conducted in an ethical and responsible manner, and that the potential benefits of this technology are realized while minimizing any potential harm.

## **Methodological Considerations**

Developing a network intrusion detection system (NIDS) is a complex task that requires the use of advanced machine learning techniques. In this section, we will reflect on the methodology we used to develop and evaluate our system, discuss the challenges we encountered, and provide recommendations for future researchers who may want to replicate or build on our work.

### **Data Collection:**

The first step in developing a NIDS is collecting data. The quality of the data is critical for the success of the system. We used the KDD CUP 99 dataset, which is a widely used dataset in the field of intrusion detection. However, this dataset has some limitations, such as the lack of real-world traffic and outdated attack scenarios. To overcome this limitation, future researchers should consider collecting their datasets or using more recent datasets to improve the accuracy of their system.

### **Dependency on Libraries:**

Machine learning libraries such as Scikit-learn, Pandas, and Numpy are powerful tools that can be used to preprocess the data and train the models. However, the use of these libraries can lead to version dependency issues, which can make it difficult to reproduce the results. To address this challenge, we recommend that future researchers document the version of the libraries they used to ensure reproducibility.

### **Data Preprocessing:**

Data preprocessing is a crucial step in developing an accurate NIDS. We encountered several challenges during this stage, such as label encoding, selecting relevant features, and data standardization. We addressed these challenges by using imputation techniques, feature selection algorithms, and standardization techniques. Future researchers should consider exploring other data preprocessing techniques and algorithms to improve the accuracy of their system.

### **Feature Selection:**

Feature selection is a crucial step in developing an accurate NIDS. We used the Recursive Feature Elimination (RFE) algorithm to select the most important features for our models. However, this algorithm may not always be the best approach for feature selection, especially for high-dimensional datasets. To address this challenge, future researchers should consider exploring other feature selection algorithms, such as Principal Component Analysis (PCA) or Variance Thresholding.

### **Data Standardization:**

Data standardization is an essential step in preparing the data for training the models. We used the StandardScaler method to standardize our data. However, this method may not always be the best approach, especially for non-linear models. To address this challenge, future researchers should consider exploring other data standardization techniques, such as MinMaxScaler or RobustScaler, and test their performance with various models.

### **Data Splitting:**

Data Splitting is an important step in building a network intrusion detection system that uses anomaly-based NIDS. In this process, preprocessed network traffic data is split into training, validation, and testing sets to train and evaluate a machine learning model for identifying potential threats. The provided code uses the 'train\_test\_split' function from the 'sklearn' library to split the preprocessed data into two parts, a training set and a validation/test set. The training set is used to train the machine learning model to classify network traffic as normal or suspicious/malicious, while the validation/test set is used to evaluate the model's performance and tune its hyperparameters. The function returns the input features and corresponding labels for the training and validation/test sets. The second 'train\_test\_split' function call is used to further split the validation/test set into a validation set and a test set for evaluating the performance of each model.

### **Model Selection:**

Model selection is a critical step in developing a NIDS. We used three models in our research, including Light Gradient Boosting, KNN, and Logistic Regression. However, selecting the appropriate model for a particular problem can be challenging. To address this challenge, we compared the performance of the models using various evaluation metrics, including accuracy, precision, recall, and F1-score. Future researchers should consider exploring other models, such as deep learning models, to improve the accuracy of their system.

### **Training and Evaluation:**

We used a standard training and evaluation approach, where we split the dataset into training and testing sets, trained the models on the training set, and evaluated their performance on the testing set. However, this approach may not always be the best approach, especially for imbalanced datasets. To address this challenge, future researchers should consider exploring other training and evaluation approaches, such as oversampling techniques or using more sophisticated evaluation metrics.

### **Metrics:**

Metrics are essential for evaluating the performance of the models. We used various metrics, including accuracy, precision, recall, and F1-score, to evaluate the performance of our models. However, these metrics may not always be the best approach, especially for imbalanced datasets. To address this challenge, future researchers should consider exploring other evaluation metrics, such as area under the Receiver Operating Characteristic (ROC) curve (AUC-ROC) or area under the Precision-Recall curve (AUC-PR), to evaluate the performance of their models accurately.

### **Challenges:**

Researchers should consider exploring other techniques, such as undersampling or a combination of oversampling and undersampling, to deal with imbalanced datasets.

Another challenge we encountered was the interpretability and explainability of the models. In our research, we used feature importance scores and SHAP values to explain the models' decisions. However, these methods may not always be the best approach, especially for non-linear models. Future researchers should consider exploring other interpretability and explainability techniques, such as Local Interpretable Model-Agnostic Explanations (LIME) or Model-Agnostic Meta-Learning (MAML), to improve the interpretability and explainability of their models.

### **Recommendations:**

Based on our experience, we recommend that future researchers consider using multiple datasets to evaluate the performance of their models. Additionally, researchers should explore the use of more sophisticated evaluation metrics, such as AUC-ROC or AUC-PR, to evaluate the performance of their models accurately. Furthermore, researchers should consider exploring other techniques and algorithms, such as different feature selection algorithms, data standardization techniques, data splitting techniques, and interpretability and explainability techniques, to improve the accuracy and interpretability of their models.

Developing a NIDS requires careful consideration of various factors, such as data collection, preprocessing, model selection, training, and evaluation. We addressed several challenges during our research, such as data collection, dependency on libraries, data preprocessing, a model explaining, feature selection, data standardization, data splitting, Training, Testing, and Evaluating the Model, Metrics. We recommend that future researchers consider exploring other techniques and algorithms to improve accuracy.

## **Comparison with Existing System**

In an existing system, [3] where used the Negative Selection Algorithm (NSA) to detect intrusions, which is a different approach from our system. Both systems aimed to achieve accurate intrusion detection and get high accuracy. There is another thing, which model is more simply can detect the intrusion. Similarly, another existing system is [8] aimed to improve detection accuracy by combining two models Naive Bayes and SVM model. Their system achieved an accuracy of 98.2%, which is lower than the accuracy achieved by our best model. There is one more system which is [12] similar in using Recursive Feature Elimination (RFE) for feature selection. They proposed network intrusion detection using decision trees, random forests, and SVM. Their system achieved an accuracy of 97.3%, which is lower than the accuracy achieved by our model.

Overall, our system achieved high accuracy in network intrusion detection. The use of RFE for feature selection also contributed to improving the performance of the models. Our best model (Light Gradient Boosting) achieved highest accuracy of 99.66%. Some systems achieved the accuracy like us. But it's important to note that our system uses simpler machine learning algorithms compared to their hybrid deep learning approach, which can make our system more practical and efficient in real-world scenarios. Therefore, our system can be considered as an effective approach for network intrusion detection using machine learning algorithms.

# **CONCLUSION**

In conclusion, we have successfully developed a network intrusion detection system using the KDD Cup 99 dataset and compared three machine learning algorithms to determine the best algorithm for this task. We used Logistic Regression, K-Nearest Neighbors, and Light Gradient Boosting, and found that Light Gradient Boosting achieved the highest accuracy of 99.65% on the test set.

The results of our study demonstrate the effectiveness of using machine learning algorithms in network intrusion detection and the importance of feature selection in improving the performance of the models. The selected features, including protocol type, service, flag, source bytes, destination bytes, logged in, count, service count, same service rate, different service rate, destination host service count, destination host same service rate, destination host different service rate, destination host same source port rate, and destination host server diff host rate, were found to be the most important for the models.

Our study provides a valuable contribution to the field of network security by showing that machine learning algorithms can be an effective tool for detecting network intrusions. We believe that our research will be useful for cybersecurity professionals and organizations looking to improve their network security.

However, there are limitations to our study that must be acknowledged. Firstly, our study was limited to the KDD Cup 99 dataset, which may not fully represent real-world network traffic. Additionally, our study focused only on three machine learning algorithms, and other algorithms may perform differently.

For future research, it would be interesting to explore the performance of other machine learning algorithms or hybrid approaches for network intrusion detection. Additionally, further investigation could be done to analyze the impact of different feature selection techniques on the performance of the models. Finally, the research could be conducted to explore the use of machine learning algorithms for real-time intrusion detection in large-scale networks.

Overall, our study demonstrates the potential for machine learning algorithms to improve network security and highlights the importance of feature selection in achieving high accuracy in network intrusion detection systems.

# **REFERENCE**

[1] Z. K. Maseer, R. Yusof, N. Bahaman, S. A. Mostafa, and C. F. M. Foozy, “Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset,” *IEEE Access*, vol. 9, pp. 22351–22370, 2021, doi: 10.1109/ACCESS.2021.3056614.

[2] K. S. Bhosale, M. Nenova, and G. Iliev, “Modified Naive Bayes Intrusion Detection System (MNBIDS),” *Proc. Int. Conf. Comput. Tech. Electron. Mech. Syst. CTEMS 2018*, pp. 291–296, 2018, doi: 10.1109/CTEMS.2018.8769248.

[3] S. I. Suliman, M. S. Abd Shukor, M. Kassim, R. Mohamad, and S. Shahbudin, “Network Intrusion Detection System Using Artificial Immune System (AIS),” *2018 3rd Int. Conf. Comput. Commun. Syst. ICCCS 2018*, pp. 426–430, 2018, doi: 10.1109/CCOMS.2018.8463274.

[4] S. Zavrak and M. Iskefiyeli, “Anomaly-Based Intrusion Detection from Network Flow Features Using Variational Autoencoder,” *IEEE Access*, vol. 8, pp. 108346–108358, 2020, doi: 10.1109/ACCESS.2020.3001350.

[5] P. Freitas De Araujo-Filho, A. J. Pinheiro, G. Kaddoum, D. R. Campelo, and F. L. Soares, “An Efficient Intrusion Prevention System for CAN: Hindering Cyber-Attacks with a Low-Cost Platform,” *IEEE Access*, vol. 9, pp. 166855–166869, 2021, doi: 10.1109/ACCESS.2021.3136147.

[6] G. De Carvalho Bertoli *et al.*, “An End-to-End Framework for Machine Learning-Based Network Intrusion Detection System,” *IEEE Access*, vol. 9, pp. 106790–106805, 2021, doi: 10.1109/ACCESS.2021.3101188.

[7] Y. Uhm and W. Pak, “Service-Aware Two-Level Partitioning for Machine Learning-Based Network Intrusion Detection with High Performance and High Scalability,” *IEEE Access*, vol. 9, pp. 6608–6622, 2021, doi: 10.1109/ACCESS.2020.3048900.

[8] T. Wisanwanichthan and M. Thammawichai, “A Double-Layered Hybrid Approach for Network Intrusion Detection System Using Combined Naive Bayes and SVM,” *IEEE Access*, vol. 9, pp. 138432–138450, 2021, doi: 10.1109/ACCESS.2021.3118573.

[9] A. R. Gad, A. A. Nashat, and T. M. Barkat, “Intrusion Detection System Using Machine Learning for Vehicular Ad Hoc Networks Based on ToN-IoT Dataset,” *IEEE Access*, vol. 9, pp. 142206–142217, 2021, doi: 10.1109/ACCESS.2021.3120626.

[10] L. Liu, P. Wang, J. Lin, and L. Liu, “Intrusion Detection of Imbalanced Network Traffic Based on Machine Learning and Deep Learning,” *IEEE Access*, vol. 9, pp. 7550–7563, 2021, doi: 10.1109/ACCESS.2020.3048198.

[11] J. Alikhanov, R. Jang, M. Abuhamad, D. Mohaisen, D. Nyang, and Y. Noh, “Investigating the effect of traffic sampling on machine learning-based network intrusion detection approaches,” *IEEE Access*, vol. 10, pp. 5801–5823, 2022, doi: 10.1109/ACCESS.2021.3137318.

[12] M. Panda, A. Abraham, S. Das, and M. R. Patra, “Network intrusion detection system: A machine learning approach,” *Intell. Decis. Technol.*, vol. 5, no. 4, pp. 347–356, 2011, doi: 10.3233/IDT-2011-0117.

[13] H. W. Oleiwi, D. N. Mhawi, and H. Al-Raweshidy, “MLTs-ADCNs: Machine Learning Techniques for Anomaly Detection in Communication Networks,” *IEEE Access*, vol. 10, no. August, pp. 91006–91017, 2022, doi: 10.1109/ACCESS.2022.3201869.

[14] M. R. Ahmed, S. Islam, S. Shatabda, A. K. M. M. Islam, and M. T. I. Robin, “Intrusion Detection System in Software-Defined Networks Using Machine Learning and Deep Learning Techniques –A Comprehensive Survey,” *TechRxiv*, no. February 2022, pp. 1–48, 2021, [Online]. Available: https://www.techrxiv.org/articles/preprint/Intrusion\_Detection\_System\_in\_Software-Defined\_Networks\_Using\_Machine\_Learning\_and\_Deep\_Learning\_Techniques\_A\_Comprehensive\_Survey/17153213

[15] T. Jain and C. Gupta, “A Review on Intrusion Detection System using Deep Learning,” *Int. J. Creat. Res. Thoughts*, vol. 8, no. 7, pp. 2320–2882, 2020, [Online]. Available: www.ijcrt.org

[16] S. Mukkamala, G. Janoski, and A. Sung, “Intrusion detection using neural networks and support vector machines,” *Proc. Int. Jt. Conf. Neural Networks*, vol. 2, no. February, pp. 1702–1707, 2002, doi: 10.1109/ijcnn.2002.1007774.

[17] S. Subbiah, K. S. M. Anbananthen, S. Thangaraj, S. Kannan, and D. Chelliah, “Intrusion detection technique in wireless sensor network using grid search random forest with Boruta feature selection algorithm,” *J. Commun. Networks*, vol. 24, no. 2, pp. 264–273, 2022, doi: 10.23919/jcn.2022.000002.

[18] M. H. Haghighat and J. Li, “Intrusion detection system using voting-based neural network,” *Tsinghua Sci. Technol.*, vol. 26, no. 4, pp. 484–495, 2021, doi: 10.26599/TST.2020.9010022.