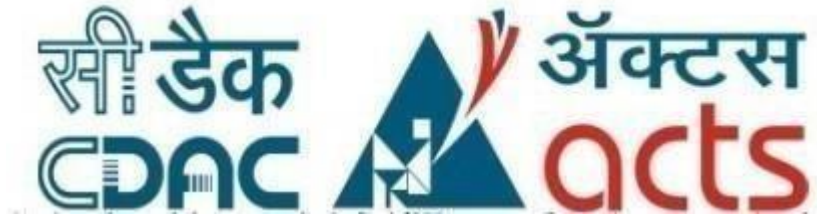


Project Report
On
Network Anomaly Detection System



Submitted
In partial fulfilment
For the award of the Degree of

PG-Diploma in Big Data Analytics

(C-DAC, ACTS (Pune))

Guided By:

Mr. Nagendra Kumar Vishwakarma

Submitted By:

Hyunh Trung Tru (230340125061)

Nikhilesh Bhalerao(230340125032)

Anup Painuly (230340125007)

Ankit Malviya (23034012005)

Centre for Development of Advanced Computing

(C-DAC), ACTS (Pune- 411008)

Acknowledgement

This is to acknowledge our indebtedness to our Project Guide, **Mr. Nagendra Kumar Vishwakarma**, C-DAC ACTS, Pune for his constant guidance and helpful suggestion for preparing this project **Network Anomaly Detection System**. We express our deep gratitude towards him for inspiration, personal involvement, constructive criticism that he provided us along with technical guidance during the course of this project.

We take this opportunity to thank Head of the department **Mr. Gaur Sunder** for providing us such a great infrastructure and environment for our overall development.

We express sincere thanks to **Mrs. Namrata Ailawar**, Process Owner, for their kind cooperation and extendible support towards the completion of our project.

It is our great pleasure in expressing sincere and deep gratitude towards **Mrs. Risha P R (Program Head)** and **Ms. Gayatri Pandit** (Course Coordinator, PG-DBDA) for their valuable guidance and constant support throughout this work and help to pursue additional studies.

Also, our warm thanks to **C-DAC ACTS Pune**, which provided us this opportunity to carry out, this prestigious Project and enhance our learning in various technical fields.

Hyunh Trung Tru (230340125061)

Nikhilesh Bhalerao(230340125032)

Anup Painuly (230340125007)

Ankit Malviya (23034012005)

ABSTRACT

This project introduces a Network Anomaly Detection System (NADS) designed to enhance cybersecurity by identifying and mitigating abnormal activities within computer networks. Leveraging machine learning techniques, particularly deep learning models, the NADS processes and analyzes network traffic patterns to differentiate between normal and anomalous behaviors. Through comprehensive preprocessing, feature extraction, and model training, the system achieves effective anomaly detection. Real-time evaluation of network behavior against learned patterns enables the NADS to promptly raise alerts upon detecting potential threats, ensuring proactive responses to security breaches. The project's rigorous evaluation demonstrates the system's accuracy, efficiency, and adaptability, reaffirming its significance in safeguarding modern network infrastructures.

Table of Contents

S. No	Title	Page No.
	Front Page	I
	Acknowledgement	II
	Abstract	II
	Table of Contents	I
		IV
1	Introduction	01-02
1.1	Introduction	01
1.2	Objective and Specifications	02
2	Literature Review	03-04
3	Methodology/ Techniques	05-10
3.1	Approach and Methodology/ Techniques	05
3.2	Dataset	08
3.3	Model Description	09
4	Implementation	11-13
4.1	Implementation	11
5	Results	15-16
5.1	Results	15
6	Conclusion	17
6.1	Conclusion	17
7	References	18
7.1	References	18

List of Figure

Figure 1:Increase in Internet users over the years.....	7
Figure 2 Project Flow Chart.....	13
Figure 3 Naive Bayes Classifier.....	15
Figure 4Multi Layer Perceptron.....	16
Figure 5 Random Forest Classifier	17
Figure 6 Ada Boost Algorithm.....	17
Figure 7 KNN Algorithm.....	18
Figure 8 Attack vs Benign Percentage	24
Figure 9 Attack labels - High number group	25
Figure 10 Attacks labels - Medium number group	25
Figure 11 Attacks labels- Small number group	26
Figure 12 Outliers per Attack types	26
Figure 13 Correlation Matrix	27
Figure 14 Flow Duration by Label.....	27
Figure 15Weight Importance Pie Chart	29
Figure 16 Feature Importance Bar graph	29
Figure 17 Front-End.....	33
Figure 18 Deployment on Local Host.....	34
Figure 19 Deployment on Private Server.....	34
Figure 20 EC2 Instance.....	35
Figure 21 Deployment on AWS	35

List of Tables

Table 1 Distribution of attack by types.....	24
Table 2 According Attack and Benign Labels Feature Importance Weight List.....	28
Table 3 Performance Comparison of Different Machine Learning Algorithm	36

Chapter 1

Introduction

1.1 Motivation

Every day millions of people and hundreds of thousands of institutions communicate with each other over the Internet. In the past two decades, while the number of people using the Internet has increased very fast, today this number has exceeded 4 billion and this increase is continuing rapidly

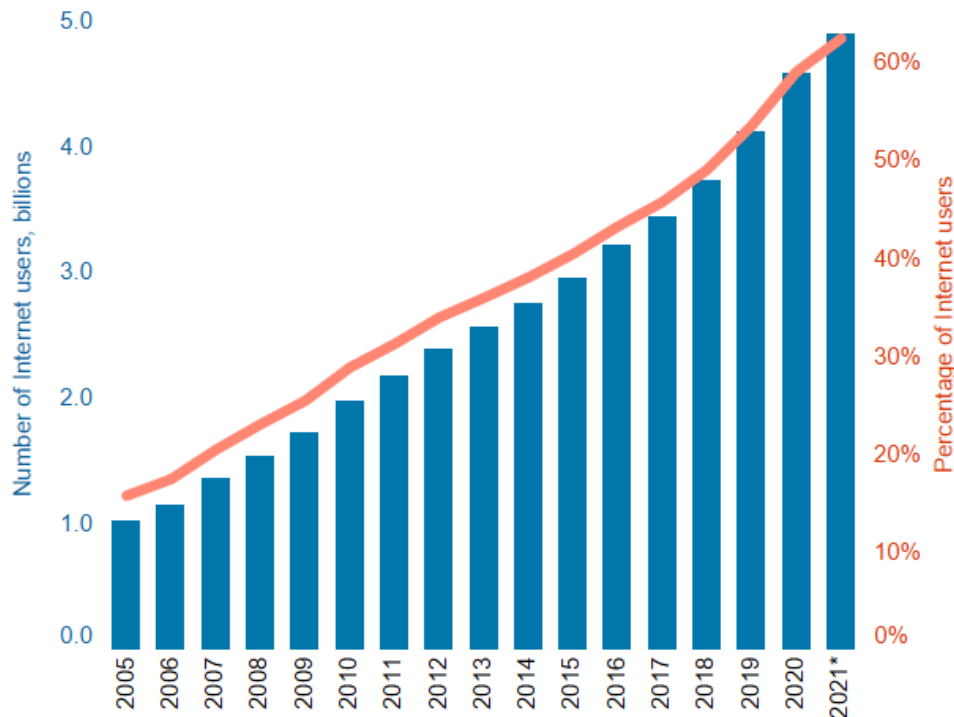


Figure 1: Increase in Internet users over the years

Parallel to these developments, the number of attacks made on the Internet is increasing day by day. Against these attacks, there are two basic methods used to detect the attacks in order to ensure information security; identification based on signature, and detection based on anomaly.

Signature-based methods use the database they created to detect attacks. This method is quite successful, but the databases need to be constantly updated and new attack information processed. Moreover, even if the databases are up-to-date, they are vulnerable to the zero-day (previously unseen) attacks. Since these attacks are not in the database, they cannot prevent these attacks. The anomaly-based approach focuses on detecting unusual network behaviors by examining network flow. This method, which has been successful in detecting attacks that it has not encountered before, so is effective against zero-day attacks.

In addition, more than half of today's internet usage is encrypted using SSL / TLS (Secure Sockets Layer / Transport Layer Security) protocols, and this rate is increasing day by day. Because of the inability to observe the contents of the encrypted internet

stream, signature- based methods do not work effectively on this type of data. However, the anomaly-based approach analyses data by its general properties such as size, connection time, and number of packets. So, it does not need to see the message content and it can also do the analysis of encrypted protocols. Due to all these advantages, the anomaly-base detection method is being used intensively to detect and prevent network attacks.

In this study, it was aimed to contribute to the literature by developing a system that detects network anomaly quickly and effectively by means of machine learning methods

1.2 Goals and Objectives

1.2.1 Goals

The goals that are aimed to achieve at the end of this study are as follows:

- Examination of machine learning algorithms that can be used to detect network anomalies.
- To detect network attacks in a fast and effective way by studying network anomaly with machine learning methods.
- To determine the success level of the study by comparing the results obtained in it with the studies previously conducted in this area.

1.2.2 Objectives

The objectives that are aimed to achieve at the end of this study are as follows:

- To examine the previous work done in the field by doing extensive field research.
- Selecting the appropriate dataset by performing comprehensive research on the alternatives to the dataset.
- Choosing suitable algorithms by conducting extensive research on machine learning algorithms.
- Deciding on right algorithms by performing exhaustive research on machine learning methods.
- Selecting the appropriate software platform.
- Choosing the suitable hardware/equipment platform.
- Deciding on the right evaluation criteria.
- Choosing the benchmark studies to be compared during the evaluation phase.

Chapter 2

Literature Review

A comprehensive literature review on Anomaly Detection Systems (ADS) reveals the evolution of techniques, methodologies, and applications in the field of network security and beyond. This review summarizes key research trends, challenges, and advancements in the realm of anomaly detection.

Early studies in anomaly detection focused on statistical methods, which utilized predefined thresholds or probability distributions to identify deviations from expected behavior. However, these methods struggled to adapt to the increasing complexity of modern networks and the diversity of potential anomalies.

Thanks to the advent of the internet, there is more data available to companies now more than ever before. But with the increase in data, there has also been an increase in security threats to business such as cyber attacks.

In simple, straightforward situations, it is possible to separate anomalous data from normal data by use of data visualization. However, as one scales up to higher numbers of variables, the exercise becomes more and more complicated. Manual thresholds in such cases do not offer a viable, scalable solution to anomaly detection. This is where anomaly detection algorithms in machine learning come in.

Machine learning helps companies manage the vast amounts of data at their disposal as well as analyze transactions in real time. By identifying differences between data points, anomaly detection opens up interesting opportunities for companies. On one hand, it minimizes potential risks for business operators while on the other, it maximizes revenue potential. Moreover, it can help companies adapt to changing conditions rapidly.

The advent of machine learning brought a paradigm shift in anomaly detection, with a surge in research leveraging supervised, unsupervised, and semi-supervised learning techniques. Support Vector Machines (SVMs) emerged as a popular choice for classification, while clustering algorithms like k-means found use in unsupervised scenarios. Hybrid methods combined the strengths of multiple techniques to improve accuracy.

The evolution of deep learning marked a significant turning point. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and more recently, Transformers, demonstrated remarkable success in capturing intricate patterns in data, including network traffic. Autoencoders, a class of unsupervised neural networks, garnered attention for their ability to learn compact representations of normal data, enabling anomaly detection by identifying deviations from these representations.

Feature engineering remained crucial, as the performance of machine learning

algorithms relied heavily on the quality of input features. Studies increasingly focused on feature selection and extraction techniques tailored to network data, such as traffic flows, packet timing, and payload characteristics. Additionally, researchers delved into the challenges of handling imbalanced datasets, where anomalies are rare compared to normal instances.

The integration of domain knowledge with data-driven techniques gained traction. Context-aware anomaly detection, where system-specific characteristics were considered, enhanced detection accuracy and reduced false positives. Moreover, researchers began to explore the applicability of anomaly detection beyond network security, such as in healthcare (intrusion detection in medical devices), finance (fraud detection), and manufacturing (predictive maintenance).

Evaluation metrics evolved to address the nuances of anomaly detection, considering true positives, false positives, true negatives, false negatives, precision, recall, and F1-score. Public datasets like KDD Cup 1999, NSL-KDD, and UNSW-NB15 became benchmarks for testing the efficacy of new algorithms.

Open challenges in anomaly detection include coping with adversarial attacks designed to evade detection, real-time processing of high-speed network data, reducing false positive rates while maintaining high detection accuracy, and the interpretability of deep learning models. Researchers are also exploring the fusion of different data sources and the incorporation of human feedback into the anomaly detection loop.

In summary, the literature review showcases the diverse landscape of anomaly detection research, from traditional statistical methods to cutting-edge deep learning and interdisciplinary applications. With challenges spanning privacy, adaptability, real-time processing, and interpretability, the field continues to evolve, driven by the evolving threat landscape and the demand for robust and effective anomaly detection solutions. The literature review highlights the evolution from traditional statistical methods to advanced machine learning and deep learning techniques in anomaly detection systems. The field continues to progress as it tackles challenges posed by complex network behaviors, emerging threat vectors, and diverse application domains.

Chapter 3

Methodology and Techniques

3.1 Methodology

Machine learning algorithms designed for anomaly detection operate by analyzing data points in a sequential manner, continually defining and updating the concept of 'normal behavior' through statistical evaluations of the available data. Throughout this processing, a series of events unfold:

1.Data Set Selection: The choice of dataset significantly influences the performance of anomaly detection algorithms. Ideally, the dataset should encompass a representative range of normal behaviors and potential anomalies.

2.Data Preprocessing: Raw data collected from networks often contains noise, missing values, and irrelevant attributes that can impede effective analysis. Preprocessing involves tasks such as data cleaning, handling missing values, and removing redundant attributes. Additionally, normalizing or scaling data is essential to ensure that all attributes contribute equally to the analysis.

3.Feature Extraction: Extracting relevant features is pivotal in providing the anomaly detection model with informative input. Feature extraction aims to distill the most pertinent information from the raw data while minimizing dimensionality. Techniques such as Principal Component Analysis (PCA), Fourier transforms, and time-series decomposition can help capture essential patterns.

4.Model Creation and Learning: The system initiates by constructing a model based on discerned patterns within the provided data. This model serves as a representation of typical behavior within the dataset, capturing trends, correlations, and fluctuations.

5.Prediction Generation: Utilizing the established model, the system extrapolates an anticipated value for the forthcoming data point in the sequence. This prediction is derived from the model's comprehension of the prevailing data patterns and serves as a benchmark for what is deemed usual in the context of the dataset.

6.Anomaly Detection: When the observed data point materially diverges from the predicted value, a notable disparity is identified. This deviation acts as a signal to the system, indicating that the observed data point could potentially be an anomaly, signifying an event or behavior that deviates from the established norm.

7.Flagging and Preliminary Assessment: The system then marks the observed data point as a candidate anomaly, bringing it to the attention of subsequent analysis stages. This initial flagging is indicative of a possible irregularity but requires further examination to ascertain its validity as a genuine anomaly.

This iterative process illustrates how machine learning-based anomaly detection systems progressively fine-tune their understanding of 'normal' behavior and respond to deviations that may indicate anomalies. By employing predictive modeling, statistical assessments, and comprehensive analysis of relationships within the data, these systems enhance their ability to accurately identify anomalies and differentiate them from expected patterns. This dynamic approach contributes to a more robust and adaptive anomaly detection mechanism that remains effective in diverse and evolving contexts.

3.2 Dataset

The dataset contains benign and the most up-to-date common attacks, which resembles the true real-world data (PCAPs). It also includes the results of the network traffic analysis using CICFlowMeter with labeled flows based on the time stamp, source, and destination IPs, source and destination ports, protocols and attack (CSV files). Also available is the extracted features definition.

The CICIDS data set has the following advantages over the other datasets mentioned above:

- The obtained data is the real-world data; was obtained from a testbed consisting of real computers.
- Data streams are collected from computers with the up-to-date operating system. There is operating system diversity (Mac, Windows, and Linux) between both attacker and victim computers.
- Data sets are labelled. In order to apply the machine learning methods, the feature extraction, which is a critical step, was applied and 85 features (see Appendix A for the feature list) were obtained.
- Both raw data (pcap files - captured network packets files) and processed data (CSV files- comma-separated data files) are available to work on.
- In the course of deciding which attack to take place, the McAfee security report was used, so there is a wide and up-to-date assortment of attacks.
- It is more abundant than other data sets in terms of protocols used. It also includes the HTTPS (Hypertext Transfer Protocol Secure) protocol in addition to FTP, HTTP, SSH and e-mail protocols.

The dataset consists of:

- Seven csv files containing data collected over a period of time
- Each csv consists of 85 unique columns which are discussed in detail in Appendix A
- On whole dataset consists of 3119345 rows

3.3 Model Description

Python and Jupyter notebook were used to create this application. Before running the files, it must be ensured that Python and the following libraries are installed.

Library	Task
Sklearn	Machine Learning Library
Numpy	Mathematical Operations
Pandas	Data Analysis Tools
Matplotlib	Graphics and Visuality
TensorFlow	Neural Network Library

The implementation phase consists of 5 steps,

- 1- Pre-processing
- 2- Statistics/EDA
- 3- Feature Selection
- 4- Machine Learning Implementation
- 5- Model Evaluation and Model Selection
- 6- Model Deployment

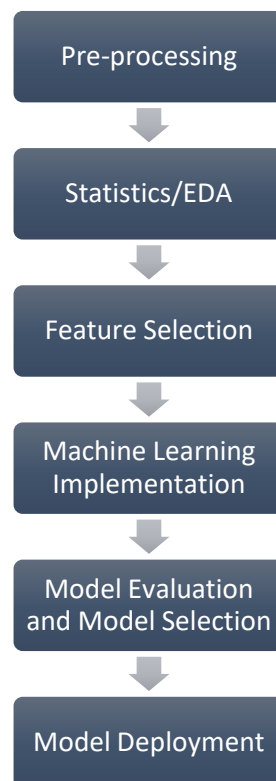


Figure 2 Project Flow Chart

1.Pre-processing:

Pre-processing involves cleaning, transforming, and organizing raw data to prepare it for analysis. It includes tasks like handling missing values, removing duplicates, and addressing outliers. Pre-processing ensures that data is in a suitable format for further analysis, reducing noise and improving the accuracy of models.

2.Statistics/EDA (Exploratory Data Analysis):

EDA involves visualizing and summarizing data to gain insights into its distribution, central tendencies, and relationships between variables. Descriptive statistics and data visualizations help identify patterns, trends, and anomalies. EDA guides subsequent analysis by revealing the nature of the data.

3.Feature Selection:

Feature selection aims to identify the most relevant attributes that contribute to the predictive power of a model. It helps prevent overfitting, reduces computation complexity, and improves model interpretability. Selecting the right features enhances the model's performance by focusing on meaningful information.

4.Machine Learning Implementation:

This phase involves implementing machine learning algorithms on pre-processed data. It includes tasks like selecting appropriate algorithms, training models using training data, tuning hyperparameters, and validating model performance. The goal is to build predictive models that can accurately generalize from the data.

5.Model Evaluation and Model Selection:

Model evaluation assesses the performance of trained models using metrics like accuracy, precision, recall, F1-score, and more. Cross-validation techniques help estimate how well a model will perform on unseen data. Model selection involves comparing different algorithms and variations to choose the best-performing one.

6.Model Deployment:

Model deployment is the process of making trained models operational and accessible for real-world use. It involves integrating the model into production systems, setting up APIs, and ensuring scalability and reliability. Deployed models generate predictions or classifications on new data, contributing to decision-making processes.

Each of these steps contributes to the overall data analysis and machine learning workflow, ensuring that the insights gained from data are translated into effective models that provide value to various applications and domains.

3.4 Machine Models:

The machine learning models utilized for the training purpose in this project are as follow:

- **Naive Bayes**
- **QDA**
- **MLP**
- **Random Forest**
- **XGBoost**
- **AdaBoost**
- **K-Nearest Neighbors**

Naive Bayes

Naive Bayes is a probabilistic machine learning algorithm based on Bayes' theorem. It assumes that the features are conditionally independent given the class label, even if this assumption isn't always met. Despite its simplicity, Naive Bayes is effective for text classification and sentiment analysis. It's particularly useful when dealing with a large number of features or dimensions.

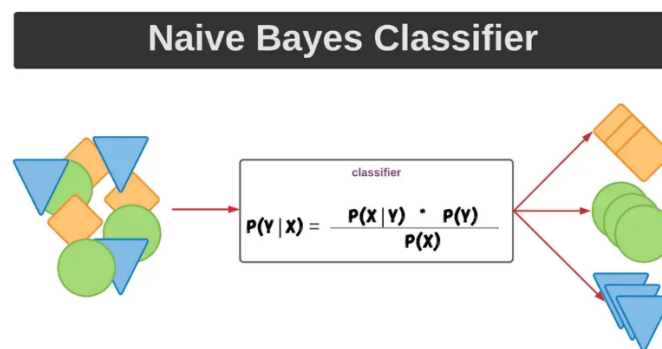


Figure 3 Naive Bayes Classifier

The algorithm calculates the probability of an instance belonging to a class based on the conditional probabilities of its features given that class. It selects the class with the highest probability as the predicted class for the instance. Naive Bayes can handle categorical and continuous data, and it's known for its efficiency and speed.

Quadratic Discriminant Analysis (QDA)

Quadratic Discriminant Analysis (QDA) is a classification algorithm that extends Linear Discriminant Analysis (LDA) to handle non-linear decision boundaries. Unlike LDA, QDA assumes that each class has its own covariance matrix, which allows it to capture more complex relationships between features and classes. QDA works well when the covariance structures of different classes are distinct.

QDA computes the posterior probability of an instance belonging to each class and assigns it to the class with the highest probability. It's particularly useful when the class distribution isn't well-separated and there's a need to capture the intricacies of data distributions.

Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron (MLP) is a type of artificial neural network that consists of multiple layers of interconnected neurons. It's a versatile algorithm used for various tasks, including classification, regression, and even more complex tasks like image recognition. MLP can learn complex non-linear relationships within data.

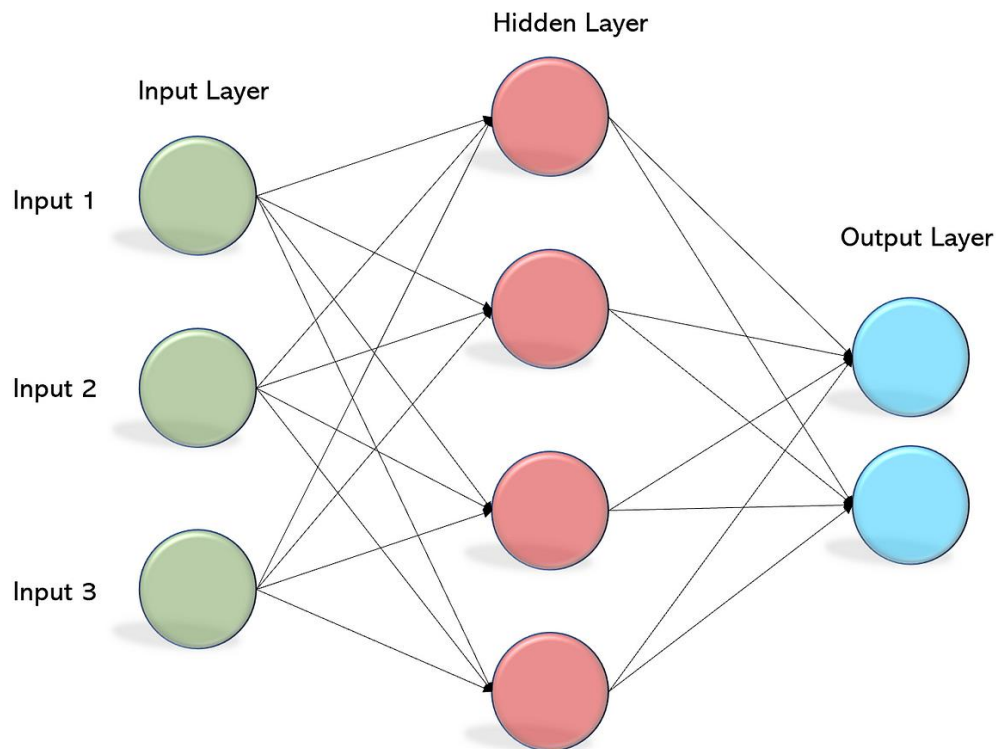


Figure 4 Multi Layer Perceptron

The network consists of an input layer, one or more hidden layers, and an output layer. Each neuron in a layer is connected to all neurons in the previous and subsequent layers. The network uses activation functions to introduce non-linearity and learns weights through backpropagation.

Random Forest

Random Forest is an ensemble learning algorithm that builds multiple decision trees during training and combines their outputs for better predictive accuracy and generalization. Each decision tree is trained on a random subset of the data and a random subset of features. Random Forest is effective for both classification and regression tasks.

During prediction, each decision tree in the forest makes a prediction, and the mode (for classification) or average (for regression) of these predictions becomes the final prediction. Random Forest handles overfitting well and is less sensitive to noise in the data compared to individual decision trees.

Random Forest Classifier

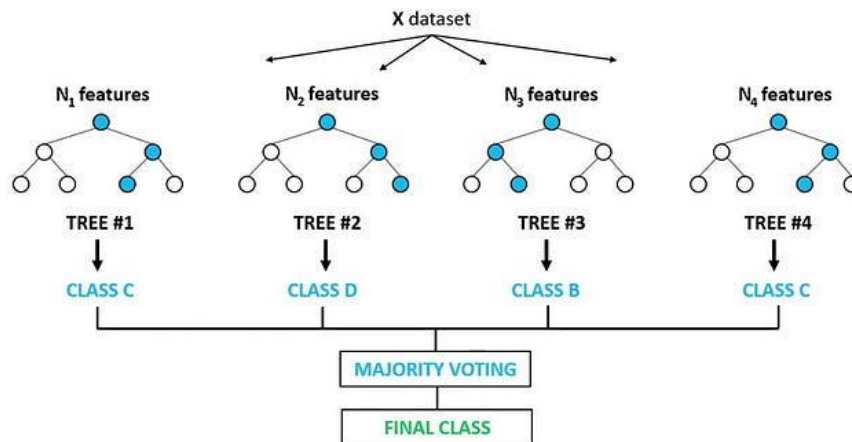


Figure 5 Random Forest Classifier

AdaBoost (Adaptive Boosting)

AdaBoost (Adaptive Boosting) is an ensemble learning algorithm that focuses on improving the accuracy of weak learners by iteratively assigning more weight to misclassified instances. Weak learners are models that perform slightly better than random guessing. AdaBoost combines the outputs of these weak learners to create a strong learner.

In each iteration, AdaBoost gives more weight to misclassified instances and trains a new weak learner. The algorithm assigns a weight to each weak learner's prediction based on its accuracy. The final prediction is a weighted combination of the weak learners' predictions.

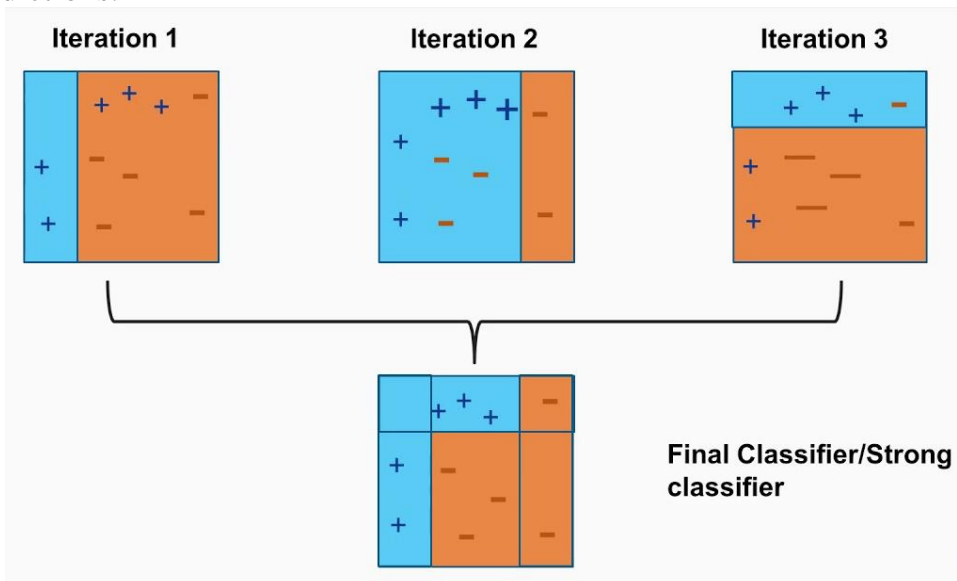


Figure 6 Ada Boost Algorithm

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a non-parametric classification algorithm that makes predictions based on the majority class among its k-nearest neighbors. KNN is simple and intuitive, suitable for both classification and regression tasks. It works well when data is locally clustered and decision boundaries are non-linear.

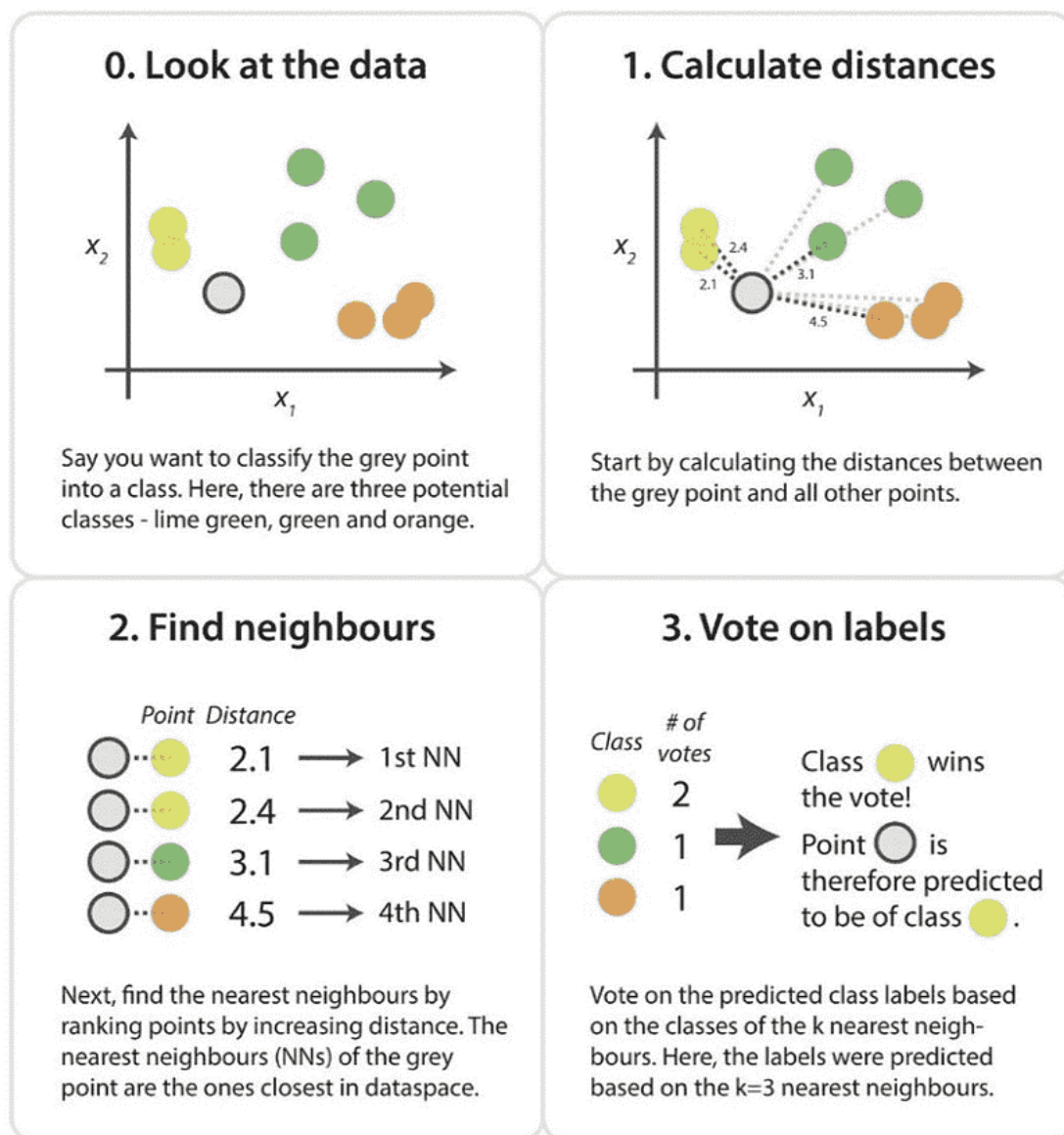


Figure 7 KNN Algorithm

Given a new instance, KNN identifies its k-nearest neighbors in the training data based on a distance metric (e.g., Euclidean distance). The prediction is then determined by the majority class among these neighbors. The choice of k impacts the algorithm's bias-variance trade-off.

XGBoost:

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm that has gained widespread popularity for its effectiveness in predictive modeling and structured data analysis. It is an ensemble learning method that belongs to the gradient boosting family of algorithms.

Key features of XGBoost:

Gradient Boosting Framework: XGBoost builds a predictive model in a stage-wise manner by combining the predictions of multiple weak learners (usually decision trees). Each subsequent learner is trained to correct the errors of the previous ones, leading to a more accurate final prediction.

Regularization: XGBoost incorporates regularization techniques such as L1 (Lasso) and L2 (Ridge) regularization into the model to prevent overfitting. This helps improve the model's generalization to new data.

Customizable Objective Function: It allows users to define their own optimization objectives based on their specific problem. This flexibility is beneficial when dealing with diverse types of data and prediction tasks.

Handling Missing Values: XGBoost can automatically handle missing values in the dataset during training and prediction, reducing the need for extensive data preprocessing.

Feature Importance: The algorithm provides insights into feature importance, allowing users to identify which features contribute the most to the model's predictions. This aids in feature selection and understanding the underlying patterns in the data.

Parallel Processing: XGBoost efficiently utilizes parallel processing and optimization techniques, making it faster than many other boosting algorithms.

Cross-Validation: The built-in cross-validation capabilities of XGBoost help tune hyperparameters effectively, leading to improved model performance.

Wide Applicability: XGBoost is versatile and can be applied to various types of machine learning tasks, including classification, regression, ranking, and more.

Due to its exceptional performance and versatility, XGBoost has been widely adopted and used in numerous data science competitions, real-world applications, and research projects. It has become a standard tool in the toolkit of machine learning practitioners for building accurate and robust predictive models.

3.5 Performance Evaluation Methods:

The results of this study are evaluated according to four criteria, namely accuracy, precision, f-measure, and recall. All these criteria take a value between 0 and 1. When it approaches 1, the performance increases, while when it approaches 0, it decreases.

Accuracy: The ratio of successfully categorized data to total data

$$\text{Accuracy} = \frac{TN+TP}{FP+TN+TP+FN} \quad (3.1)$$

Recall (Sensitivity): The ratio of data classified as an attack to all attack data

$$\text{Recall} = \frac{TP}{TP+FN} \quad (3.2)$$

Precision: The ratio of successful classified data as the attack to all data classified as the attack

$$\text{Precision} = \frac{TP}{FP+TP} \quad (3.3)$$

F-measure (F-score/F1-score): The harmonic-mean of sensitivity and precision. This concept is used to express the overall success. So, in this study, when analyzing the results, it will be focused, especially on the F1 Score.

$$\text{F-measure} = \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}} \quad (3.4)$$

In calculating these four items, the four values summarized below are used:

- **TP:** True Positive (Correct Detection) The attack data classified as attack
- **FP:** False Positive (Type-1 Error) The benign data classified as attack.
- **FN:** False Negative (Type-2 Error) The attack data classified as benign.
- **TN:** True Negative (Correct Rejection) The benign data classified as benign

This distribution is presented by visualizing Confusion matrix in figure, also.

True Class	
Predicted Class	Actual Positive
	Actual Negative
Actual Positive	TP True Positive (Desired)
Actual Negative	FN False Negative (Undesired)
Actual Positive	FP False Positive (Undesired)
Actual Negative	TN True Negative (Desired)

In addition to these 4 measures, it was included in this list, considering that the processing time is also an important factor in selecting algorithms, although it is not considered a success criterion.

Receiver Operating Characteristic (ROC):

The Receiver Operating Characteristic (ROC) curve is a fundamental tool for evaluating and visualizing the performance of binary classification models. It provides insights into a model's ability to distinguish between the two classes by plotting the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) at various decision thresholds.

At the heart of the ROC curve lies the concept of sensitivity and specificity:

Sensitivity (True Positive Rate, TPR): Sensitivity measures the proportion of actual positive instances that the model correctly classifies as positive. It reflects the model's ability to identify the positive cases within the dataset.

Specificity (True Negative Rate, TNR): Specificity measures the proportion of actual negative instances that the model correctly classifies as negative. It indicates the model's proficiency in correctly identifying the negative cases.

As the classification threshold changes, the balance between sensitivity and specificity shifts. A low threshold leads to classifying more instances as positive, raising sensitivity but potentially increasing false positives, thereby reducing specificity. Conversely, a high threshold can boost specificity while potentially lowering sensitivity.

The ROC curve is created by plotting various points corresponding to different threshold settings. The resulting curve showcases the model's performance spectrum as it navigates this sensitivity-specificity trade-off. In an ideal scenario, where the model perfectly distinguishes between classes, the ROC curve would closely hug the top-left corner, signifying high sensitivity and low false positives.

The Area Under the Curve (AUC) is a crucial metric associated with the ROC curve. It summarizes the model's overall performance across all possible thresholds. A higher AUC value indicates superior discrimination capabilities and a better overall classifier. In fact, an AUC of 0.5 implies that the model's performance is equivalent to random guessing, while an AUC of 1.0 represents a flawless classifier.

The ROC curve's utility extends beyond performance measurement:

- **Model Comparison:** When comparing multiple models, the one with a higher AUC tends to be superior, as it consistently maintains a better balance between sensitivity and specificity.
- **Threshold Selection:** Different applications might prioritize sensitivity or specificity. The ROC curve helps in selecting the optimal threshold that aligns with the desired trade-off.
- **Imbalanced Data:** In cases where class distribution is imbalanced, the ROC curve provides insights into the model's performance that might not be evident through metrics like accuracy. This is because it focuses on the true positive and false positive rates, which are sensitive to class distribution.
- **Diagnostic Test Evaluation:** ROC analysis is widely used in medicine to evaluate the performance of diagnostic tests, where sensitivity and specificity are critical for accurate disease detection.
- **Limitations:** While the ROC curve is a versatile tool, it may not be suitable for all scenarios. In cases where the cost of false positives and false negatives is significantly different, other evaluation metrics like precision-recall curves might provide better insights.

In conclusion, the ROC curve is a foundational tool in binary classification evaluation. Its graphical representation of sensitivity and specificity trade-offs, along with the associated AUC metric, offers a comprehensive understanding of a model's performance. It aids in comparing models, selecting thresholds, and handling imbalanced data, contributing to effective decision-making in various domains.

Chapter 4

Implementation

4.1. Pre-processing:

This step provides code which is intended to preprocess the CICIDS2017 dataset, which contains cybersecurity data related to network traffic and intrusions. The dataset consists of multiple CSV files, each representing different scenarios. The goal of this code is to clean and prepare the data for analysis by addressing specific issues and converting categorical features into numeric form. Here's an explanation of the main steps in the code:

- **CSV File Handling:**
 - The code first defines the list of CSV file names to be processed. These filenames correspond to different days and types of network traffic, including both normal and attack scenarios.
- **Handling Empty Entries and Unrecognized Characters:**
 - The code reads each line of the CSV files and addresses two main issues:
 - Some entries are empty or meaningless, represented by a series of commas without actual data. These entries are eliminated from the data.
 - Unrecognized characters, specifically "–" (Unicode code: 8211), are replaced with a hyphen "-" (Unicode code: 45). This change is done to ensure compatibility with Python-Panda's library.
- **Handling Infinity and NaN Values:**
 - The code replaces "inf", "Infinity", and "NaN" with appropriate numeric values (0) to ensure consistent data representation.
- **Converting Non-Numeric Features:**
 - The code identifies columns with non-numeric (string and categorical) properties. These features are stored in the string_features list.
- **Removing Unnecessary Columns:**
 - Column 61 ("Fwd Header Length") is removed due to a mistake in the code where it was rewritten erroneously.
- **Merging CSV Files:**
 - The cleaned and pre-processed data from each CSV file is merged into a single file named "all_data.csv". This consolidated file is easier to work with for further analysis.

The code for preprocessing can be referred from Appendix C.

4.2. EDA/Statistics:

In this step, program requires the presence of the all_data.csv file in the same directory. The purpose of the program is to provide statistical insights about the data contained in the dataset.

Due to the varying sizes of the data, the program generates graphics in three separate groups, allowing all data to be visualized effectively:

- Big Group: Labels with more than 11,000 occurrences.
- Medium Group: Labels with occurrences between 600 and 11,000.
- Small Group: Labels with fewer than 600 occurrences.

The final set of graphics provides the attack rates and normal behavior rates.

The program generates visualizations and statistics for the dataset based on the label groupings. It aims to provide insights into the distribution and behavior of the data.

Label	count
BENIGN	2203723
DoS Hulk	231073
PortScan	158930
DDoS	41835
DoS GoldenEye	10293
FTP-Patator	7938
SSH-Patator	5897
DoS slowloris	5796
DoS Slowhttptest	5499
Bot	1966
Infiltration	36
Heartbleed	11

Table 1 Distribution of attack by types

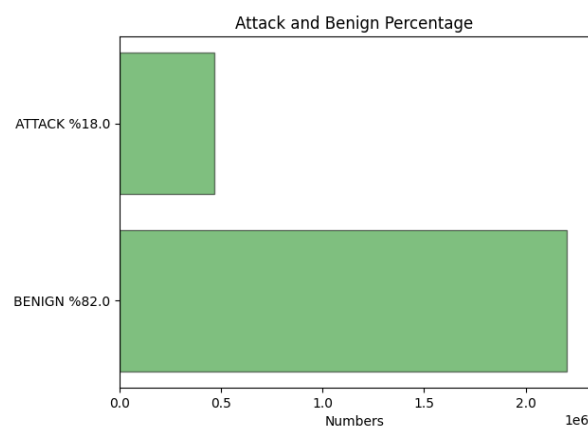


Figure 8 Attack vs Benign Percentage

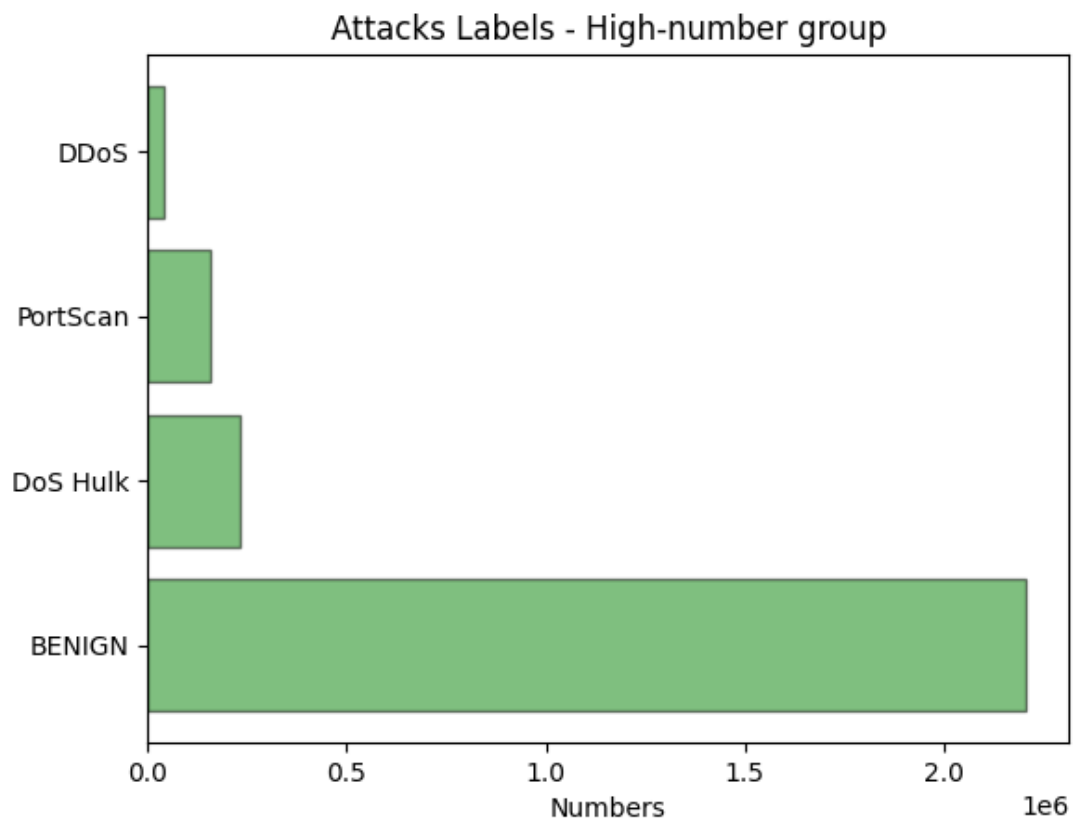


Figure 9 Attack labels - High number group

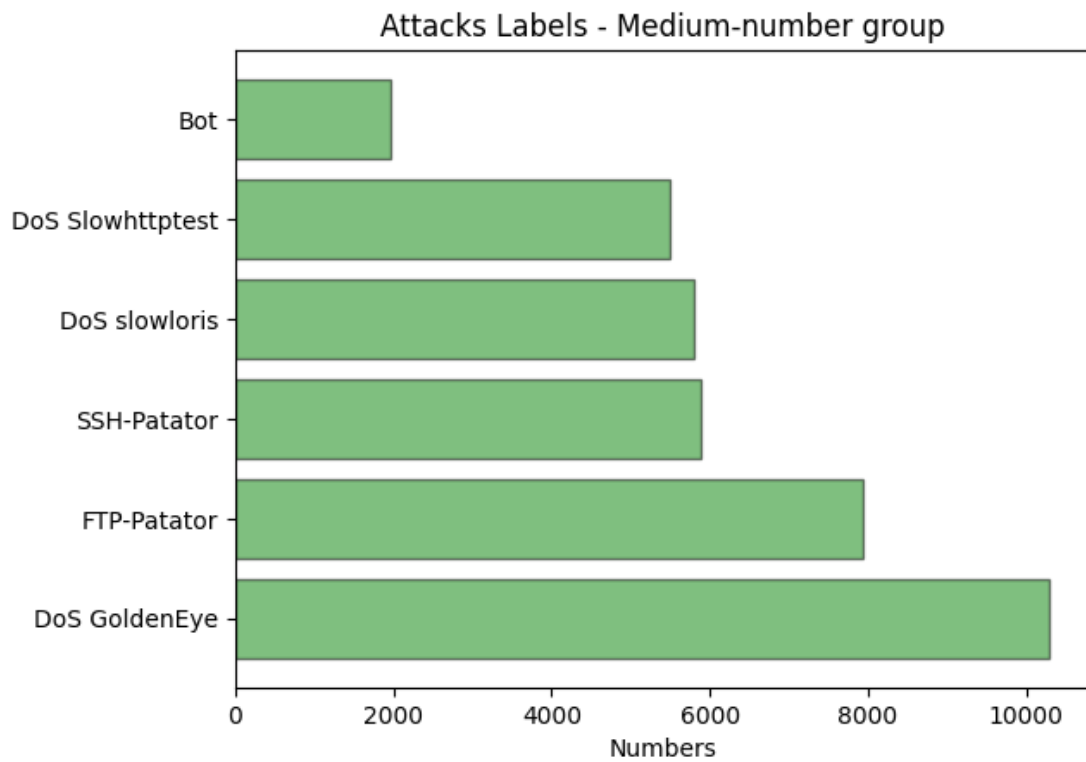


Figure 10 Attacks labels - Medium number group

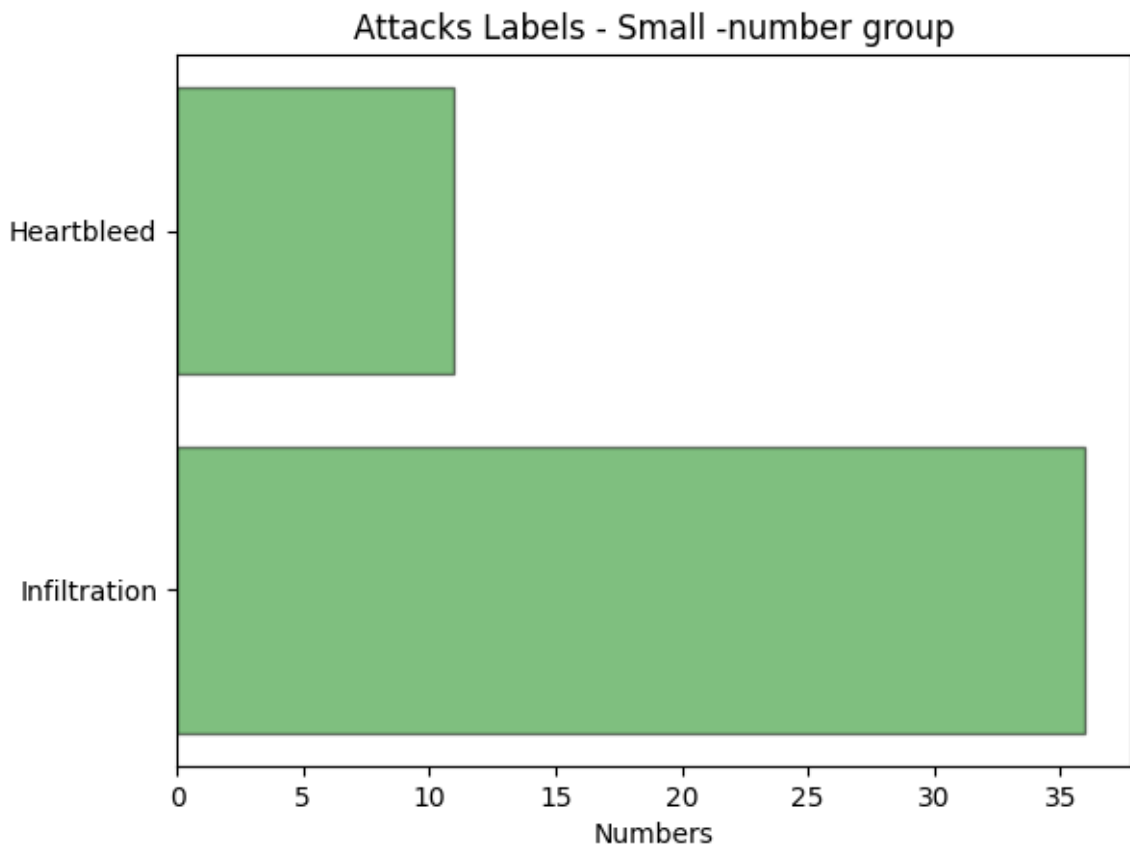


Figure 11 Attacks labels- Small number group

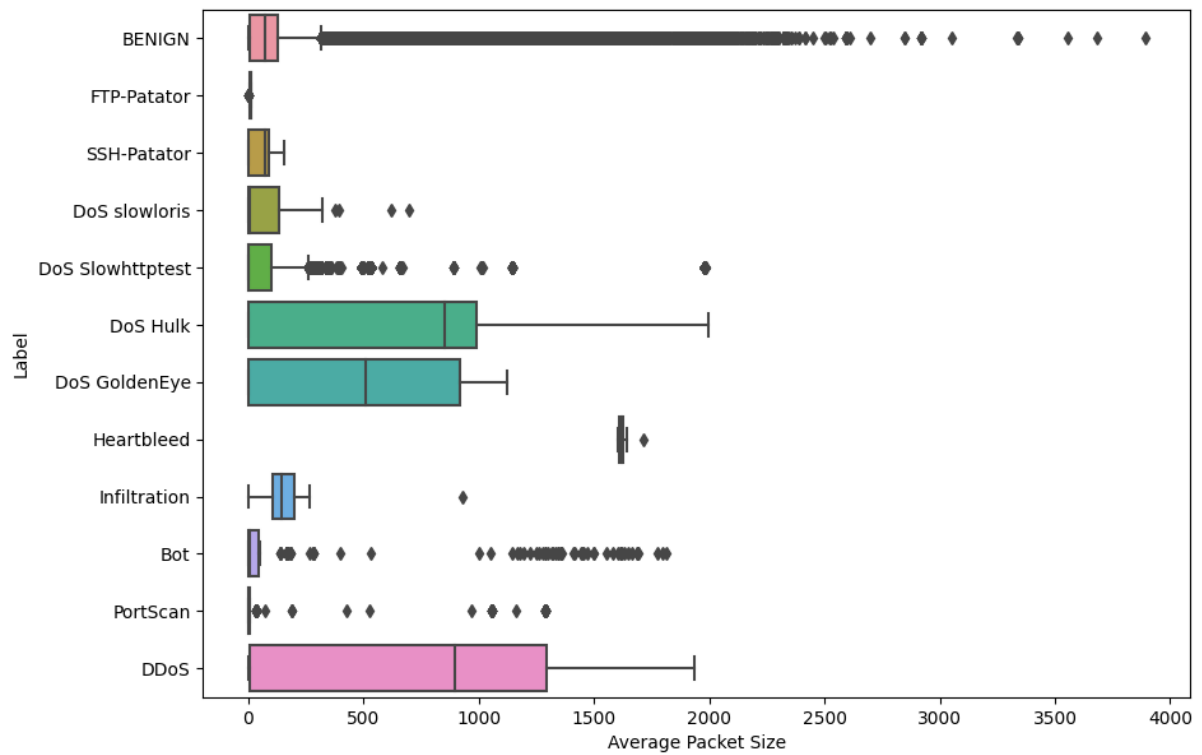


Figure 12 Outliers per Attack types

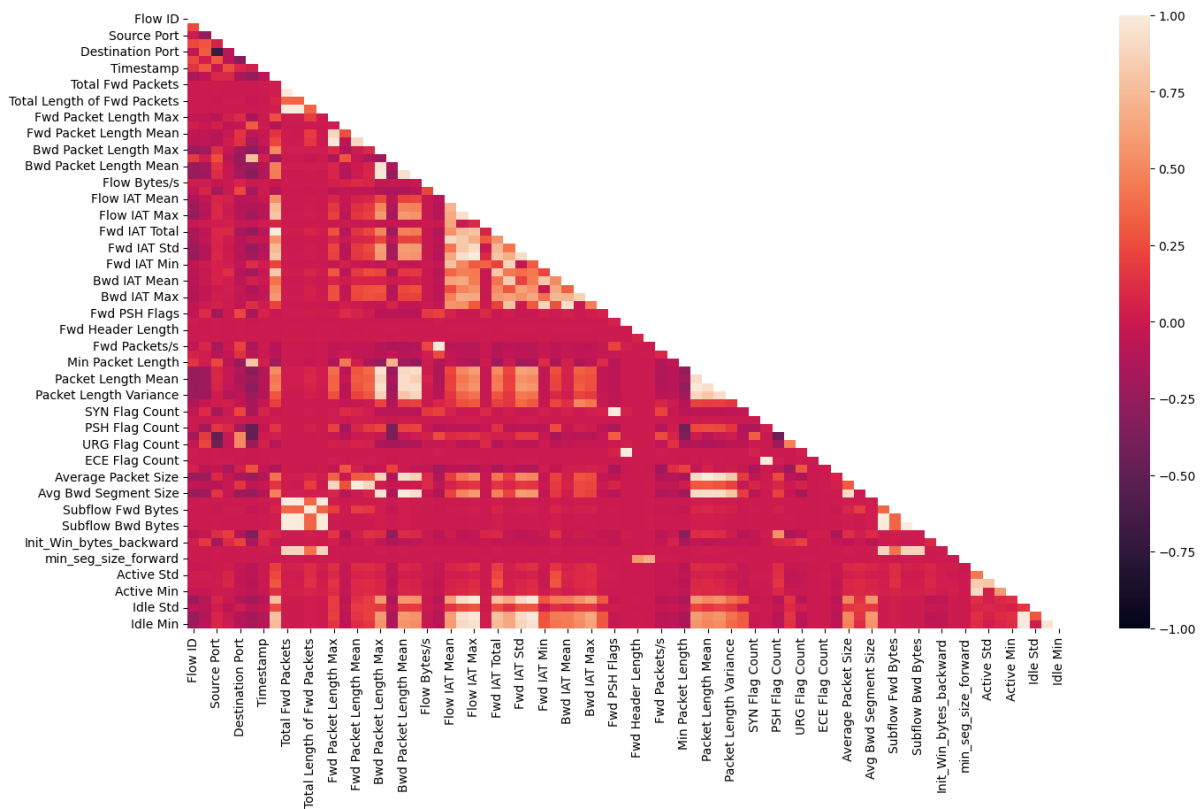


Figure 13 Correlation Matrix

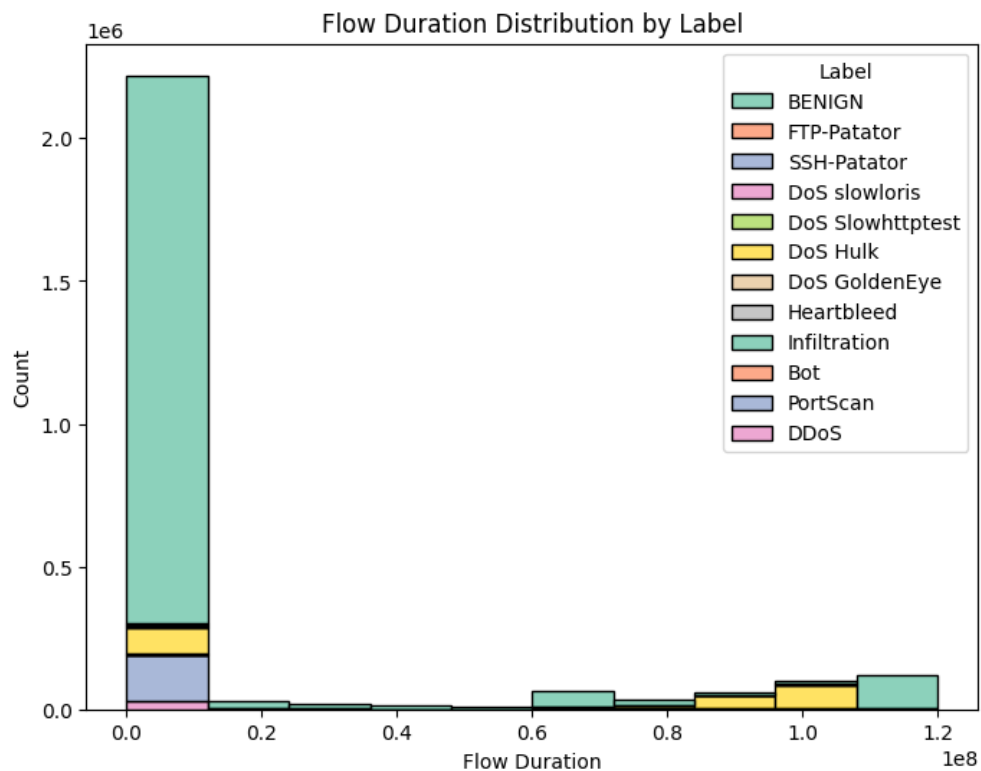


Figure 14 Flow Duration by Label

The code for EDA can be referred from Appendix D

Note: This program assumes that the **pandas**, **matplotlib**, **seaborn** check the imports section and make sure other required libraries are installed.

4.3 Feature Extraction

In this section, the features in the dataset are evaluated to determine which features are important to define which attack. The list of features and a detailed description are provided in the Appendix A.

The approach in feature selection is to apply the Random Forest Regressor operation to the whole dataset by collecting all attack types under a single label; “attack”. So, the data in this file contains only the attack and benign tags.

The result after using Random Forest Regressor is as follow:

Feature Name	Weight Importance
Bwd Packet Length Std	0.255541
Flow Bytes/s	0.183915
Total Length of Fwd Packets	0.119303
Fwd Packet Length Std	0.067645
Flow IAT Std	0.009947
Flow IAT Min	0.004074
Fwd IAT Total	0.003286
Flow Duration	0.002430
Total Length of Bwd Packets	0.002312
Flow IAT Mean	0.001901
Flow IAT Max	0.001648
Bwd Packet Length Max	0.000787
Fwd Packet Length Mean	0.000767
Fwd Packet Length Min	0.000586
Bwd Packet Length Mean	0.000206
Total Backward Packets	0.000153
Flow Packets/s	0.000147
Fwd Packet Length Max	0.000108
Total Fwd Packets	0.000079
Bwd Packet Length Min	0.000067

Table 2 According Attack and Benign Labels Feature Importance Weight List

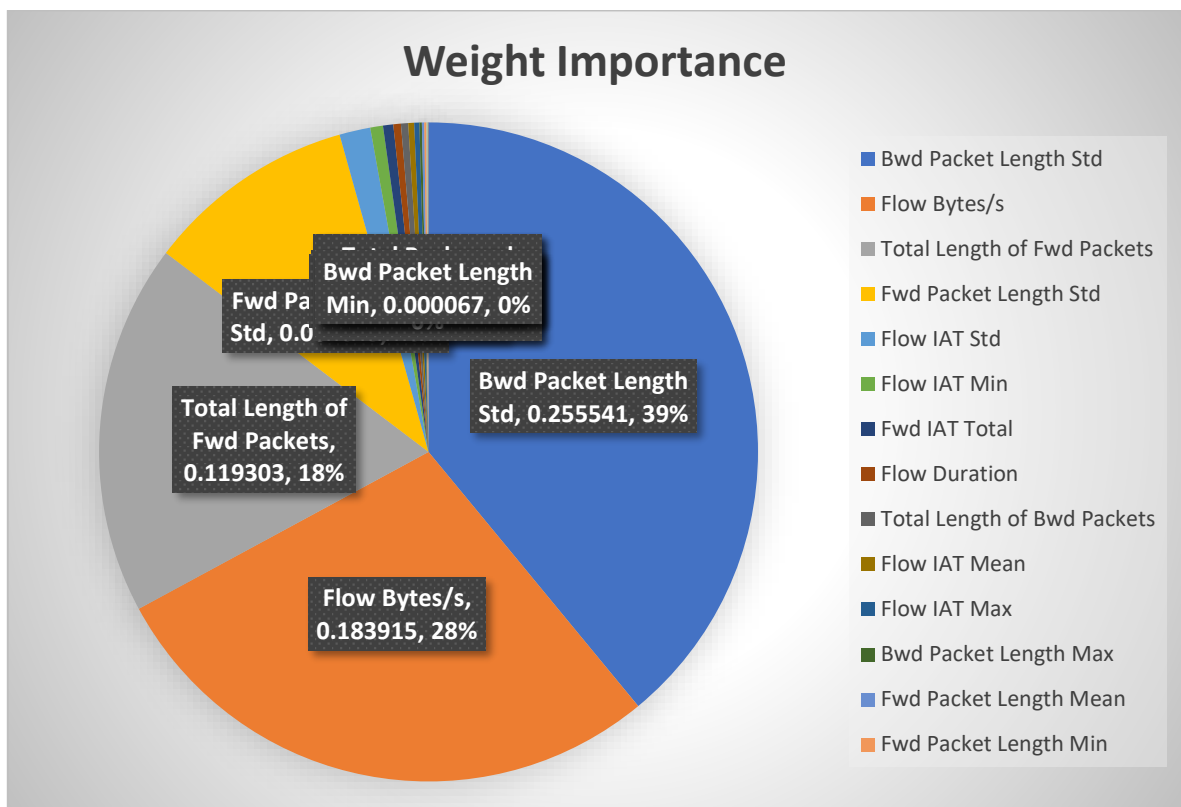


Figure 15 Weight Importance Pie Chart

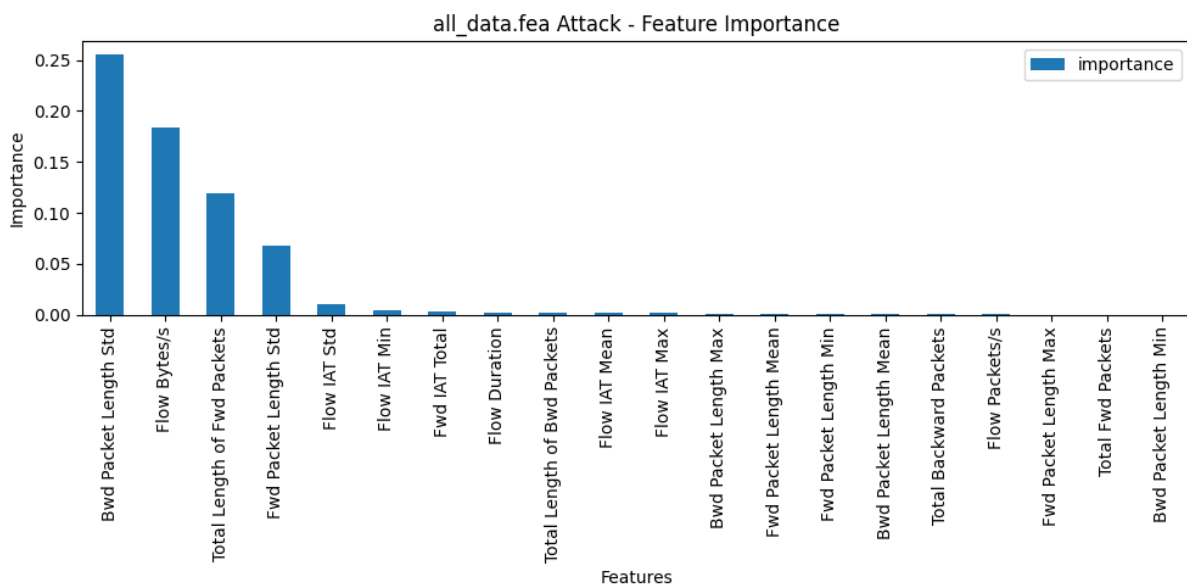


Figure 16 Feature Importance Bar graph

On inspection of the result provided by Random Forest Regressor following 7 features are selected for model training:

- Bwd Packet Length Std
- Flow Bytes/s
- Total Length of Fwd Packets
- Fwd Packet Length Std
- Flow IAT Std
- Flow IAT Min
- Fwd IAT Total

The code for feature selection/extraction can be referred from Appendix E.

4.4 Machine Learning Algorithm Implementation

The approach has been used to apply machine learning algorithms to the dataset. In this method, the feature selected in the Feature selection section are used. The dataframe contain 30% attack and 70% benign data. The seven machine learning methods are applied to dataframe, resulting in a separate outcome for each model. With this method, it is aimed to observe the effectiveness and performance of different machine learning methods on different attack types. The different machine learning models and parameters which are used are as follow:

1. Navie Bayes:
2. QDA
3. MLP
4. Random Forest
5. XGBoost
6. Adaboost
7. KNN

Parameters:

Each model was configured with specific parameters to optimize its performance during training and testing.

- **Naive Bayes:**
 - Default parameters for Gaussian Naive Bayes were used.
 - No specific parameters were provided for Gaussian Naive Bayes. The model assumes that the features are normally distributed.
- **QDA:**
 - No specific parameters were provided, using the default settings.
 - QDA models the class-specific covariance matrices, which influence its decision boundaries.
- **XGBoost:**
 - `n_estimators=100`, `learning_rate=0.01`, `objective='binary:logistic'`.
 - `n_estimators`: Number of boosting rounds or decision trees in the ensemble (100 in this case).
 - `learning_rate`: The step size at which the model adjusts based on misclassified instances (0.01).

- objective: The learning objective, set to 'binary:logistic' for binary classification.
- **Random Forest:**
 - max_depth=5, n_estimators=100, max_features=2, n_jobs=-1.
 - max_depth: Maximum depth of the decision trees (5) to prevent overfitting.
 - n_estimators: Number of decision trees in the ensemble (100).
 - max_features: Maximum number of features considered for splitting (2).
 - n_jobs: Number of CPU cores used for parallel processing (-1 for all available cores).
- **AdaBoost:**
 - Default parameters for AdaBoostClassifier were used.
 - The model adapts its boosting algorithm based on the weak learners' performance.
- **MLP:**
 - hidden_layer_sizes=(13,13,13), max_iter=500.
 - hidden_layer_sizes: Number of neurons in each hidden layer (13 neurons in three layers).
 - max_iter: Maximum number of iterations for convergence during training (500).
- **Nearest Neighbors:**
 - n_neighbors=3, n_jobs=-1.
 - n_neighbors: Number of neighbors considered for prediction (3) based on proximity.
 - n_jobs: Number of CPU cores used for parallel processing (-1 for all available cores).

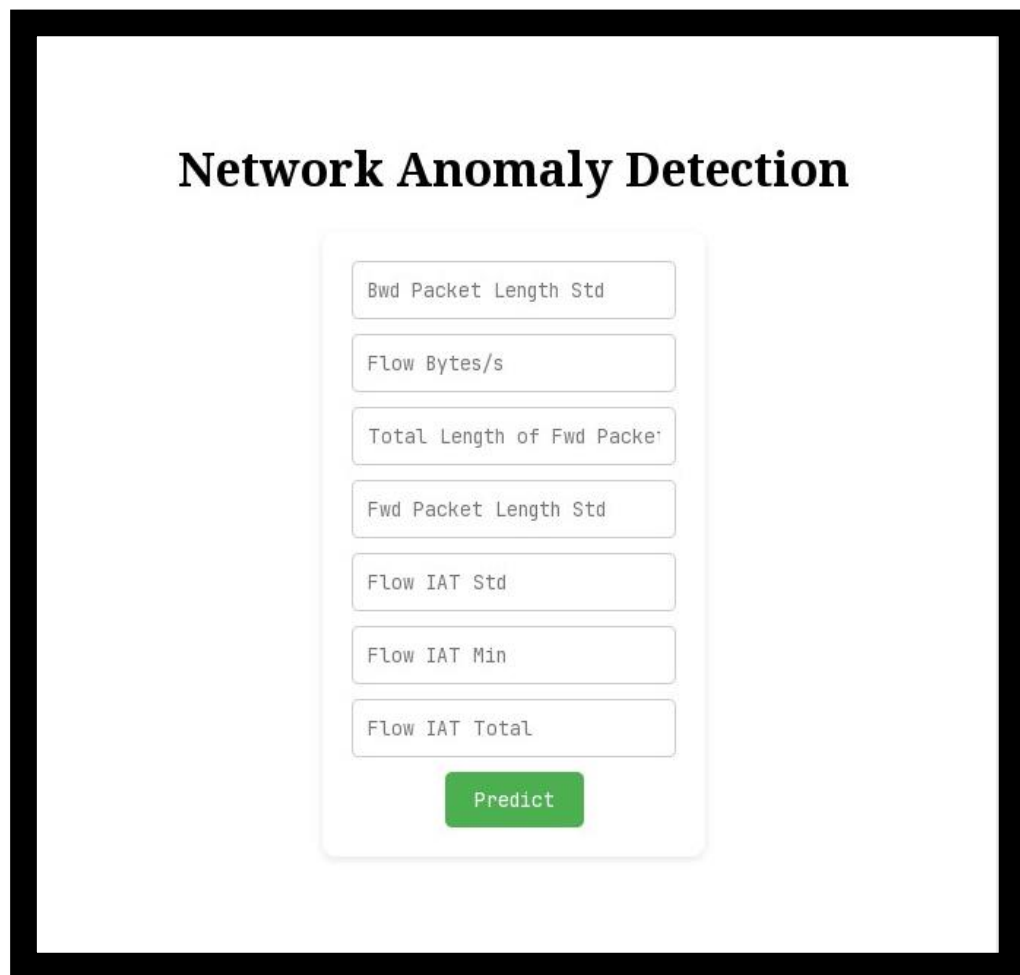
The code for machine learning algorithm implementation can be referred from Appendix F

4.5 Deployment:

Model deployment is the process of making trained machine learning models accessible and operational for real-world use. Deployed models allow automated and efficient execution of predictions on new data, enabling timely decision-making. For this project we used “Pickle Library” of Python to save and export the best performing model i.e., XGBoost.

Front-End

Frontend development plays a crucial role in delivering user-friendly and visually appealing web applications that meet both functional and aesthetic requirements. For this project we used “Flask Library” of Python to build front-end.



The screenshot displays a web application titled "Network Anomaly Detection". It features a central form with seven input fields for network metrics: "Bwd Packet Length Std", "Flow Bytes/s", "Total Length of Fwd Packets", "Fwd Packet Length Std", "Flow IAT Std", "Flow IAT Min", and "Flow IAT Total". Below these fields is a green "Predict" button. The entire form is enclosed in a light gray rounded rectangle, which is itself centered within a larger white area with a thick black border.

Figure 17 Front-End

Deployment on Local Host:

Deployment on localhost involves hosting a web application or service on a local server for development or testing purposes. It enables developers to simulate real-world conditions, test functionality, and debug code before deploying to a live server, enhancing the development process and ensuring quality.

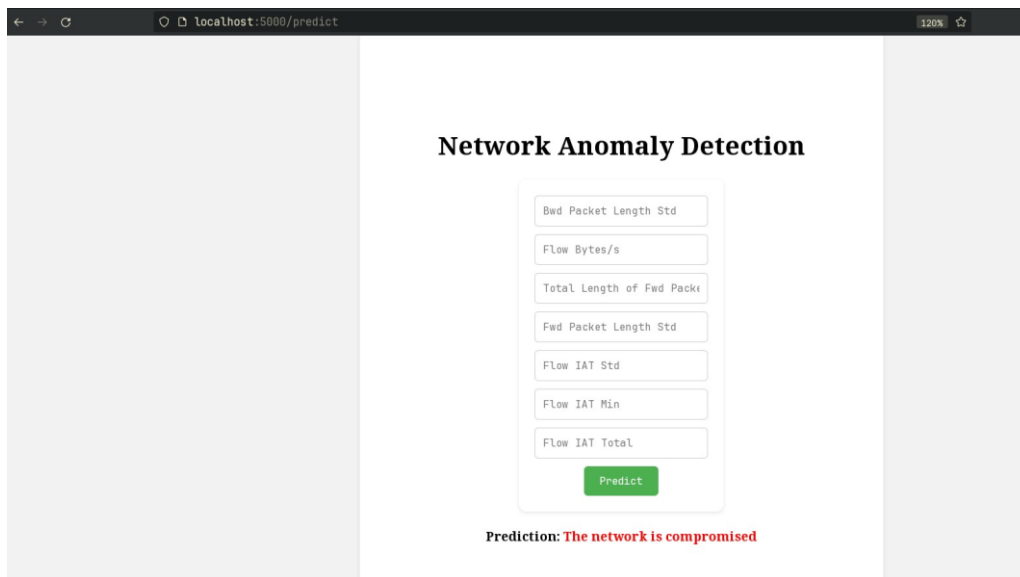


Figure 18 Deployment on Local Host

Deployment on Private Server:

Deployment on a private server involves hosting applications on a dedicated server within an organization's network. This offers control over security, data privacy, and customization, making it suitable for sensitive or specialized projects where public hosting is not preferred. We deployed this project on Windows server for hosting.

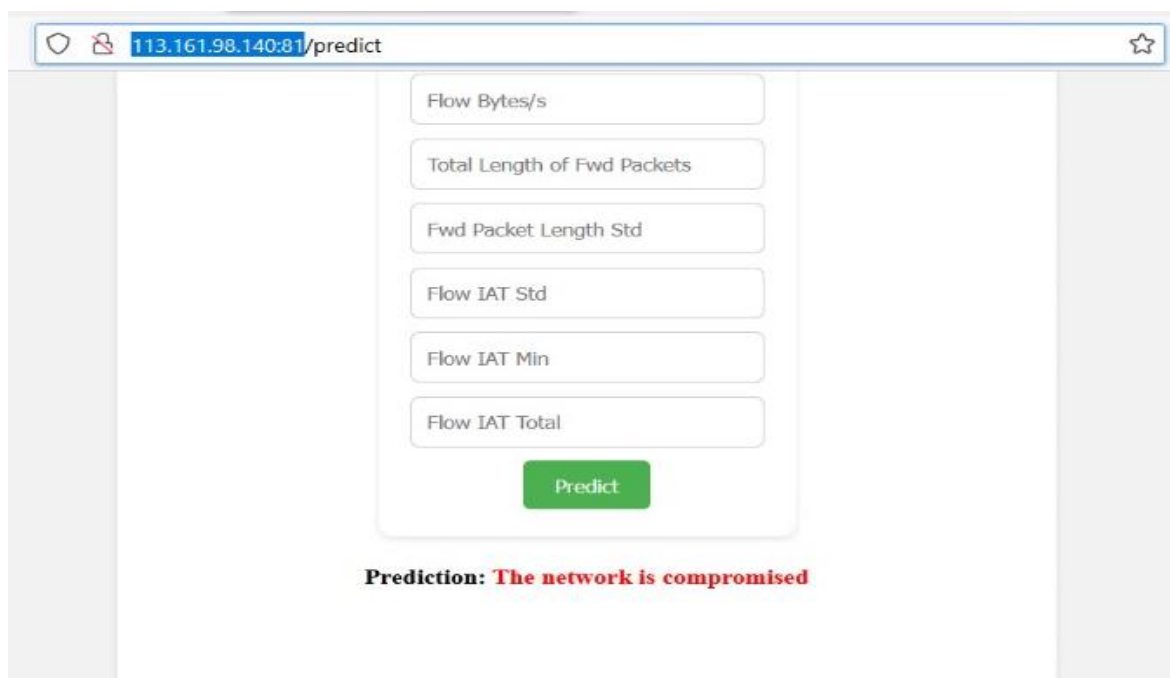


Figure 19 Deployment on Private Server

Deployment on AWS:

Deployment on Amazon Web Services (AWS) involves hosting applications on Amazon's cloud infrastructure. AWS offers a range of services such as EC2 for virtual servers, S3 for storage, and Lambda for serverless computing. It provides scalability, reliability, and global accessibility, reducing hardware management overhead. Developers can deploy, manage, and scale applications easily, catering to various workloads. AWS's pay-as-you-go pricing model and extensive service ecosystem make it a popular choice for startups, enterprises, and individuals seeking efficient and cost-effective cloud deployment solutions.

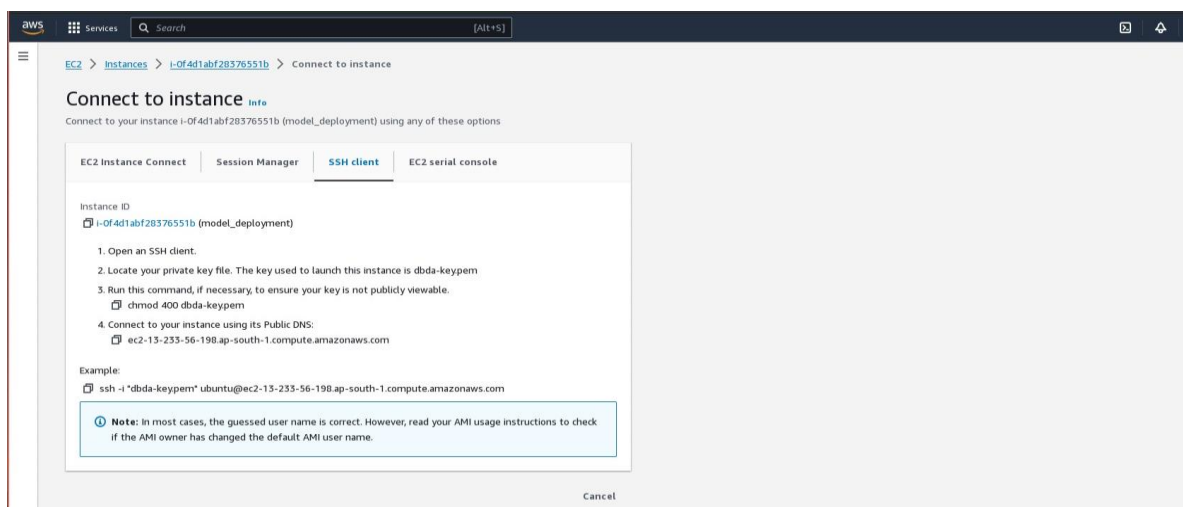


Figure 20 EC2 Instance

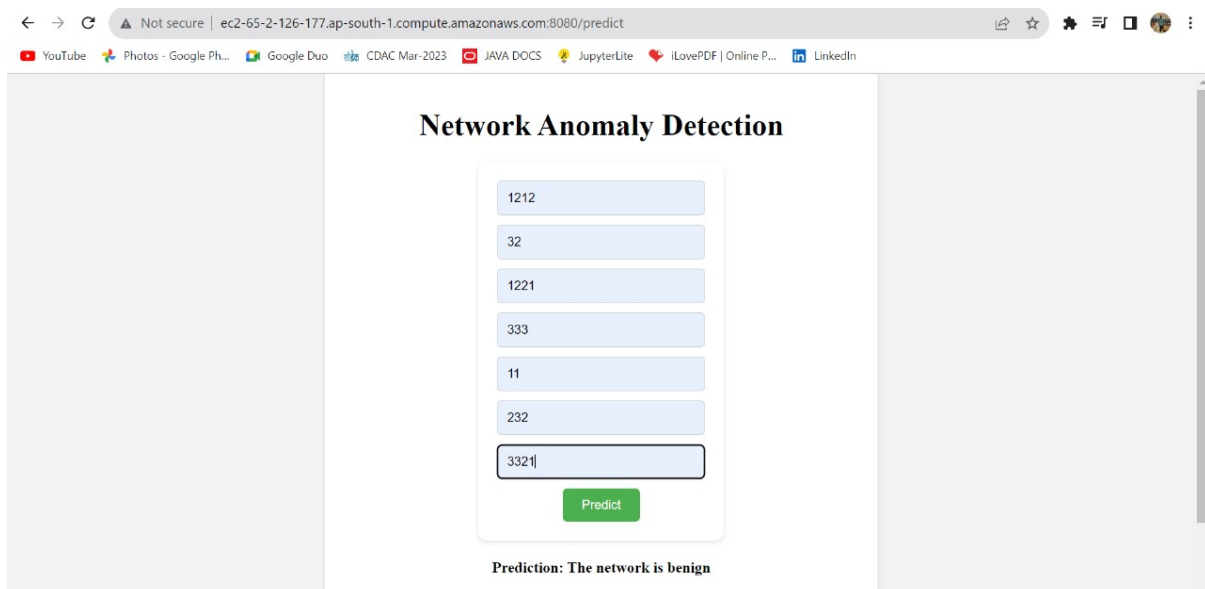


Figure 21 Deployment on AWS

Chapter 5

Results

In this section, the final results of the implementation are compared. The ROC was determined as the main evaluation criterion. However, in cases where equality prevailed in the results, Accuracy, Recall, Precision and F-1 Score were assessed respectively. Therefore, the timing of the execution was not included in the evaluation, considering that it may be misleading, even though it is shared

Machine Learning Algorithm	Accuracy	Precision	Recall	F-1 Score
Naïve Bayes	0.81	0.66	0.63	0.64
QDA	0.86	0.79	0.68	0.71
MLP	0.83	0.75	0.54	0.54
Random Forest	0.95	0.97	0.85	0.90
XGBoost	0.96	0.95	0.93	0.94
Adaboost	0.94	0.94	0.86	0.89
KNN	0.96	0.94	0.94	0.94

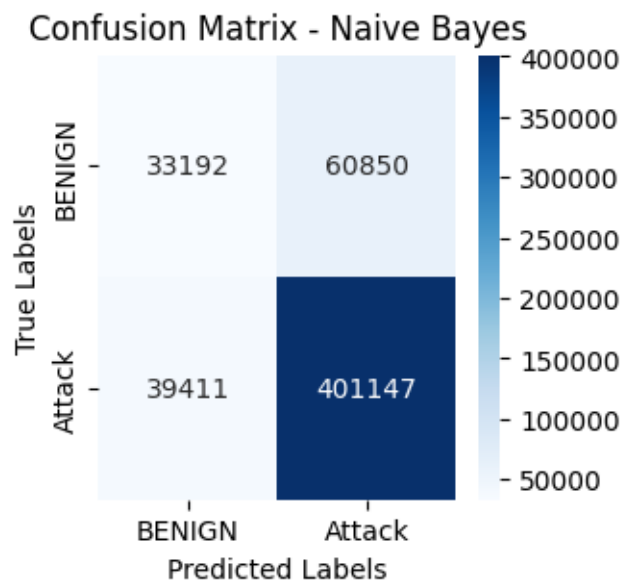
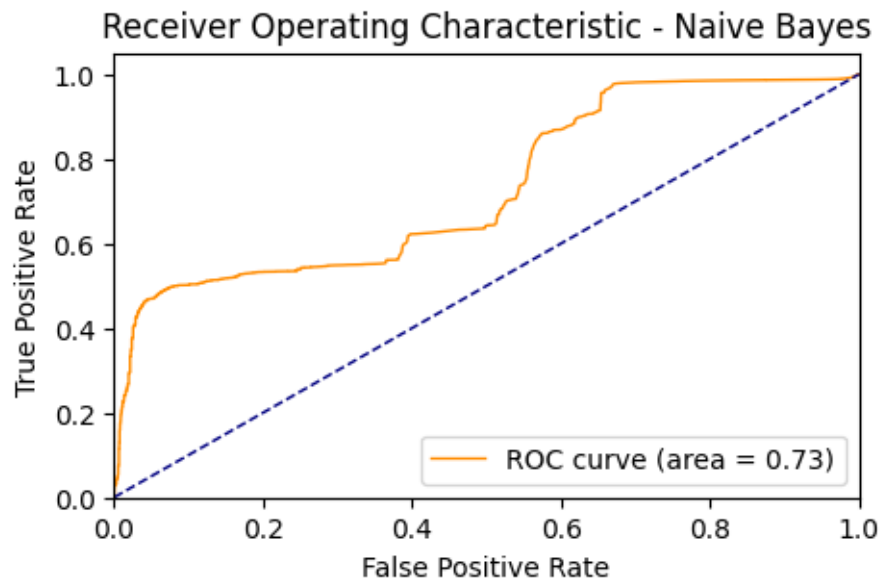
Table 3 Performance Comparison of Different Machine Learning Algorithm

From table we can infer that XGBoost displays remarkable consistency, XGBoost attains an **Accuracy of 0.96**. It excels in both **Precision (0.95)** and **Recall (0.93)**, contributing to an impressive **F-1 Score of 0.94**. These scores reflect its strong performance across the entire spectrum of metrics, making it a reliable choice for effective intrusion detection. Also, Random Forest with an impressive **Accuracy of 0.95**, Random Forest delivers exceptional results. It not only achieves high Precision and Recall but also boasts an impressive **F-1 Score of 0.90**, indicating its ability to balance precision and recall effectively. On the other hand, despite achieving an overall **Accuracy of 0.83**, MLP falls behind in terms of **Recall (0.54)** and **F-1 Score (0.54)**. This suggests that while the algorithm can correctly classify a significant portion of instances, it struggles to identify a notable number of actual intrusion cases. Improving its ability to detect intrusions is crucial to enhance its performance.

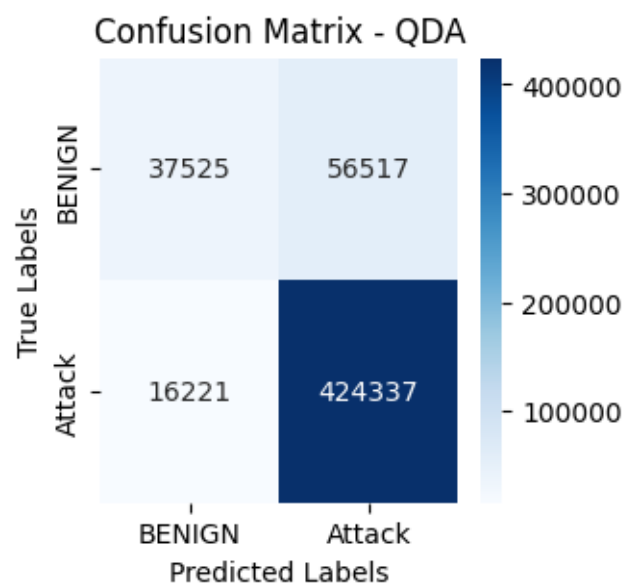
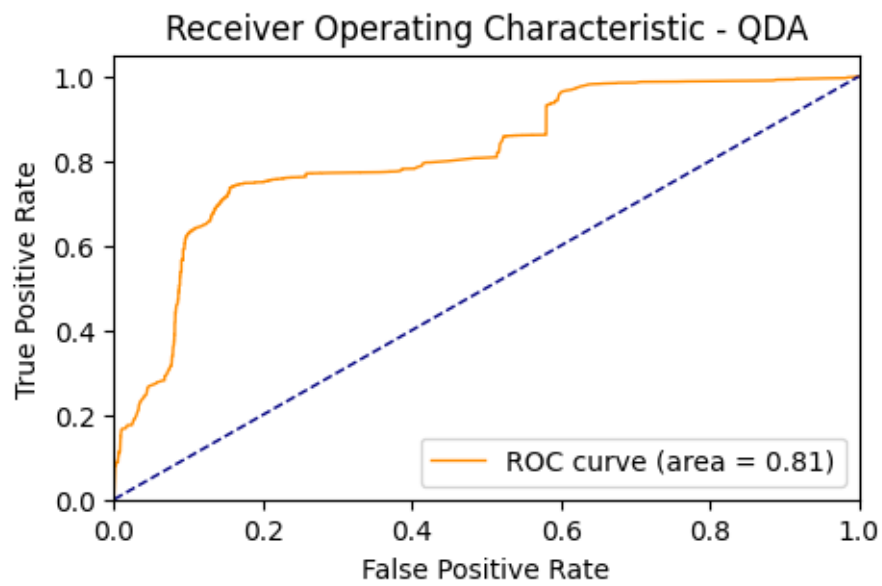
In summary, the evaluation provides valuable insights into the strengths and weaknesses of each machine learning algorithm for intrusion detection. While all algorithms exhibit varying degrees of performance, Random Forest, XGBoost, and KNN emerge as top contenders, consistently delivering high Accuracy and well-balanced Precision, Recall, and F-1 Score. These algorithms showcase their effectiveness in handling complex intrusion scenarios and mitigating both false positives and false negatives.

On considering ROC, following were the results of different algorithm:

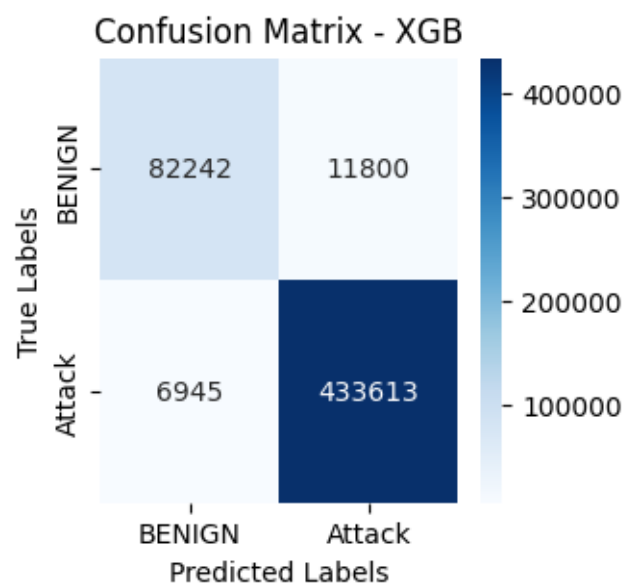
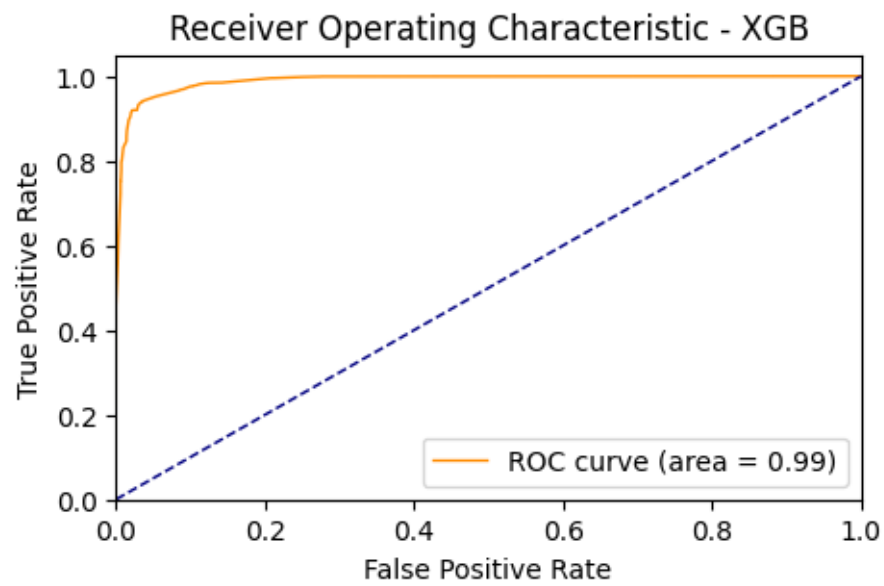
Naïve Bayes:



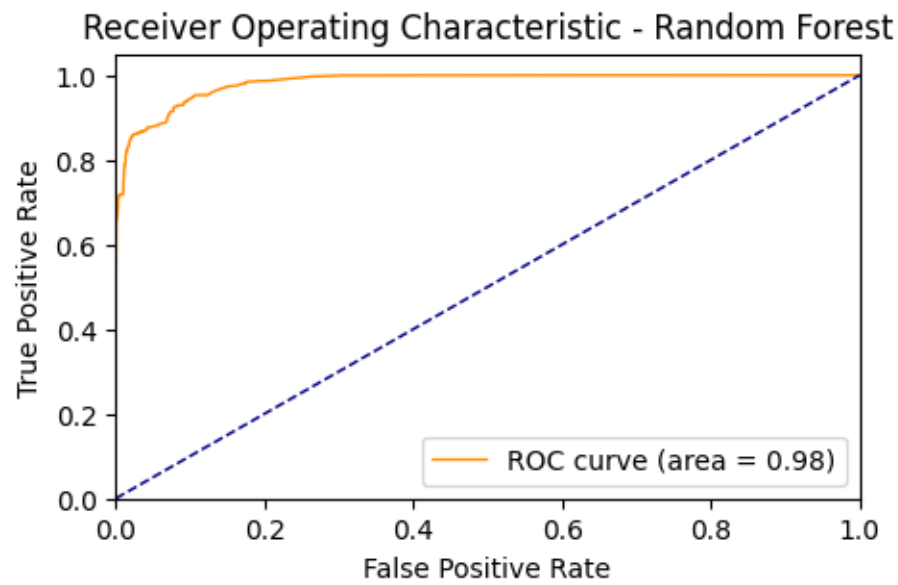
QDA:



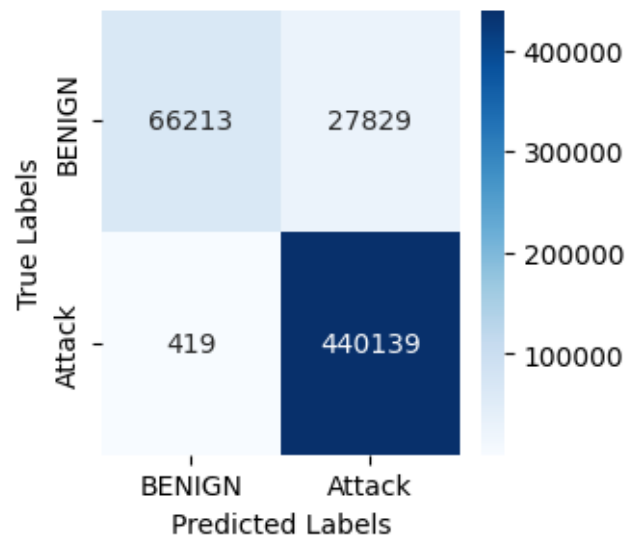
XGBoost:



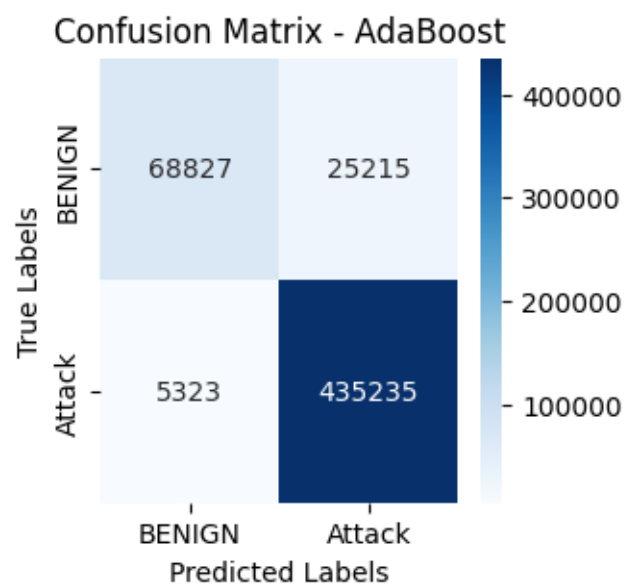
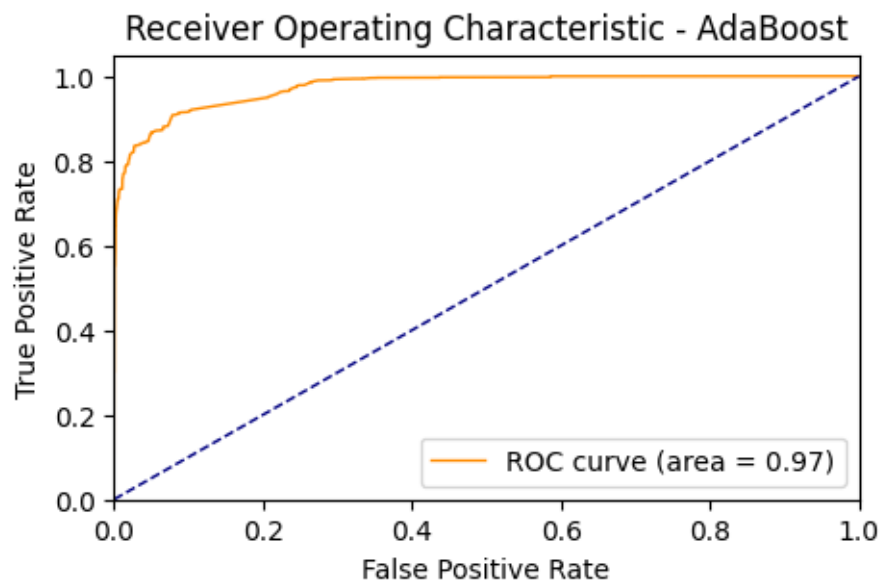
Random Forest:



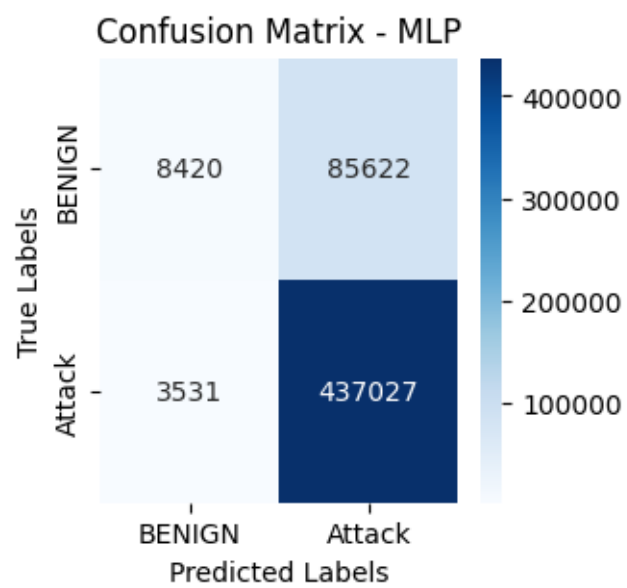
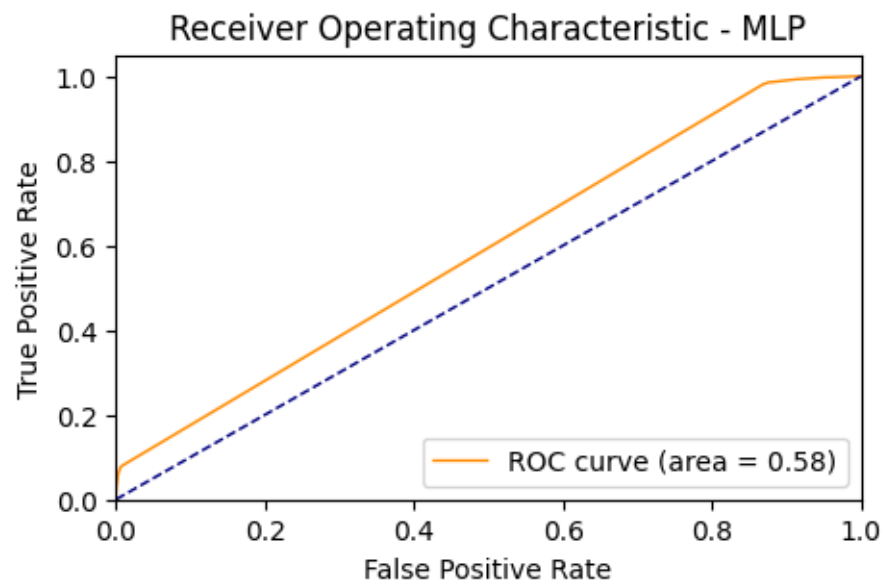
Confusion Matrix - Random Forest



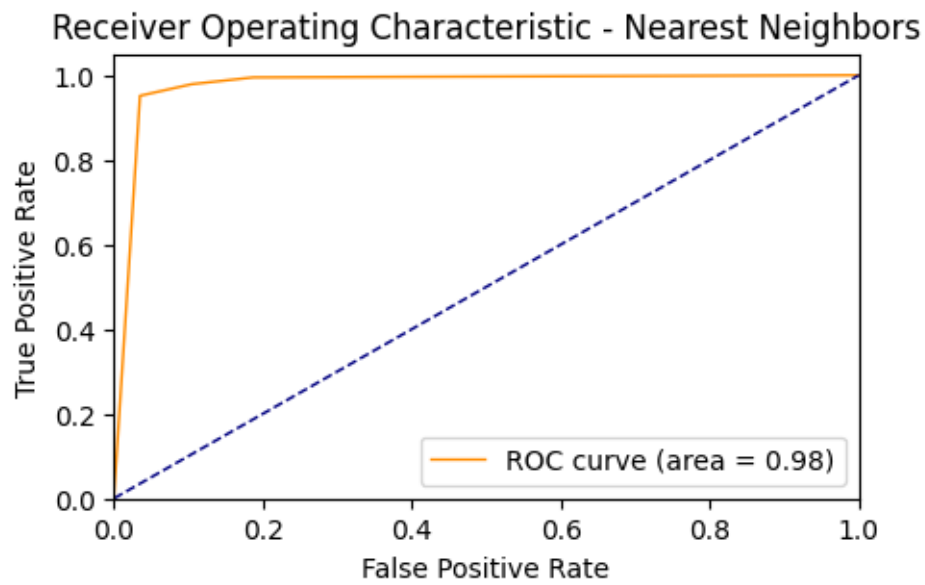
AdaBoost:



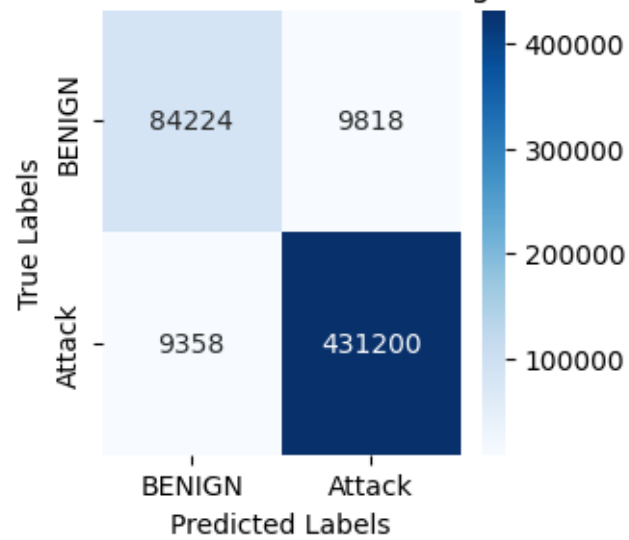
MLP:



KNN:



Confusion Matrix - Nearest Neighbors



Comparing the ROC results we get the following table:

Machine Learning Algorithm	ROC-AUC
Naïve Bayes	0.73
QDA	0.81
MLP	0.58
Random Forest	0.98
XGBoost	0.99
AdaBoost	0.97
KNN	0.98

In this study, we conducted a comprehensive evaluation of various machine learning algorithms using the Receiver Operating Characteristic (ROC) curve as a performance metric. The ROC curve offers valuable insights into the discriminatory power of each algorithm across different threshold settings, shedding light on their strengths and weaknesses.

Upon examining the results presented in the ROC table, several key observations emerge. The evaluated algorithms exhibit a wide spectrum of performance, as indicated by their respective ROC values. This suggests that the algorithms differ significantly in their ability to distinguish between the binary classes.

Notably, Random Forest, XGBoost, and AdaBoost showcase particularly high ROC values of 0.98, 0.99, and 0.97 respectively. These algorithms demonstrate robust discriminatory capabilities and are well-suited for applications where accurate class separation is crucial.

Naïve Bayes and Quadratic Discriminant Analysis (QDA) demonstrate moderate performance, with ROC values of 0.73 and 0.81 respectively. While their discriminatory power is somewhat lower, they still provide viable options depending on the specific context of the application

On the other hand, Multilayer Perceptron (MLP) presents a relatively lower ROC value of 0.58, suggesting room for improvement in its classification capabilities. Further optimization or model adjustments might be needed to enhance its performance.

In conclusion, The ROC curve serves as an effective benchmark for algorithm comparison, enabling practitioners to gauge the relative performance of different algorithms within the same framework.

Chapter 6

Conclusion

6.1 Conclusion

- In conclusion, the Network Anomaly Detection System (NADS) developed in this project presents a robust and adaptive solution for bolstering network security in the face of escalating cyber threats.
- By harnessing the power of deep learning and advanced data analysis techniques, the NADS showcases its ability to accurately differentiate between normal and anomalous network behaviors.
- The project's successful implementation and comprehensive evaluation underscore the NADS's effectiveness in threat detection, with its minimal false positive rate and efficient response mechanism.
- As network landscapes continue to evolve, the NADS stands as a critical asset, contributing significantly to the protection of sensitive information and the uninterrupted operation of computer networks in an increasingly interconnected digital world.

6.2 Future Enhancement

- **Attack Categorization:** Exploring machine learning techniques to allow the NADS to categorize novel anomalies, making it more effective.
- **Realtime Learning Data:** In this study, a data set consisting of CSV files containing features obtained from the network flow was used as the training and test data. Unfortunately, this method is not practically viable in real systems. However, this problem can be solved by adding a module that catches real network data and makes it workable with the machine learning algorithm.
- **Unsupervised Learning:** Explore unsupervised learning techniques to allow the NADS to discover novel anomalies without relying solely on labeled data, making it more adaptable to new attack vectors.
- **Real-time Adaptability:** Develop techniques to enable the NADS to adapt in real-time to emerging threats and evolving network behaviors, ensuring its continued efficacy in a dynamic environment.
- **Multi-Modal Data Fusion:** Investigate methods to integrate data from multiple sources, such as logs, network flow data, and user behavior, to create a holistic view of network activities and anomalies.

Chapter 7

References

- [1] Canadian Institute for Cybersecurity Dataset:
<https://www.unb.ca/cic/datasets/ids-2017.html>
- [2] K. Kostas, "Anomaly Detection in Networks Using Machine Learning," Research Proposal, 23 Mar 2018, 2018.
- [3] VENUGOPAL DEP "Anomaly Detection using different methods"
Available: <https://www.kaggle.com/code/adepvenugopal/anomaly-detection-using-different-methods>
- [4] Network Datasets: Computer Network Traffic from workstation IPs
Available: <https://www.kaggle.com/code/adepvenugopal/network-datasets/input>
- [5] "1998 DARPA Intrusion Detection Evaluation Data Set," Lincoln Laboratory, Massachusetts Institute of Technology, [Online]. Available: <https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-data-set>.
- [6] Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "An evaluation framework for intrusion detection dataset," in Information Science and Security (ICISS), 2016 International Conference on, 2016, pp. 1-6: IEEE.
- [7] "KDD Cup 1999 Data," University of California, Irvine, [Online].
Available: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>. [Accessed 05 Aug 2018].
- [8] Github-Sanjay Sane: <https://github.com/sanjayssane/Machine-Learning>
- [9] Keras API Docs: <https://keras.io/api/>

Appendix A.

The List of The Features and Explanations

This list includes the features and explanations contained in the CICIDS dataset.

This list has been taken from: <http://www.netflowmeter.ca/netflowmeter.html>

No	Feature Name	Feature Description
1	Flow ID	Flow ID
2	Source IP	Source IP
3	Source Port	Source Port
4	Destination IP	Destination IP
5	Destination Port	Destination Port
6	Protocol	Protocol
7	Timestamp	Timestamp
8	Flow Duration	Duration of the flow in Microsecond
9	Total Fwd Packets	Total packets in the forward direction
10	Total Backward Packets	Total packets in the backward direction
11	Total Length of Fwd Packets	Total size of packet in forward direction
12	Total Length of Bwd Packets	Total size of packet in backward direction
13	Fwd Packet Length Max	Maximum size of packet in forward direction
14	Fwd Packet Length Min	Minimum size of packet in forward direction
15	Fwd Packet Length Mean	Mean size of packet in forward direction
16	Fwd Packet Length Std	Standard deviation size of packet in forward direction
17	Bwd Packet Length Max	Maximum size of packet in backward direction
18	Bwd Packet Length Min	Minimum size of packet in backward direction
19	Bwd Packet Length Mean	Mean size of packet in backward direction
20	Bwd Packet Length Std	Standard deviation size of packet in backward direction
21	Flow Bytes/s	Number of flow bytes per second
22	Flow Packets/s	Number of flow packets per second
23	Flow IAT Mean	Mean length of a flow
24	Flow IAT Std	Standard deviation length of a flow
25	Flow IAT Max	Maximum length of a flow
26	Flow IAT Min	Minimum length of a flow
27	Fwd IAT Total	Total time between two packets sent in the forward direction
28	Fwd IAT Mean	Mean time between two packets sent in the forward direction
29	Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
30	Fwd IAT Max	Maximum time between two packets sent in the forward direction
31	Fwd IAT Min	Minimum time between two packets sent in the forward direction
32	Bwd IAT Total	Total time between two packets sent in the backward direction
33	Bwd IAT Mean	Mean time between two packets sent in the backward direction
34	Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
35	Bwd IAT Max	Maximum time between two packets sent in the backward direction
36	Bwd IAT Min	Minimum time between two packets sent in the backward direction
37	Fwd PSH Flags	Number of packets with PUSH
38	Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
39	Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
40	Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
41	Fwd Header Length	Total bytes used for headers in the forward direction
42	Bwd Header Length	Total bytes used for headers in the backward direction
43	Fwd Packets/s	Number of forward packets per second
44	Bwd Packets/s	Number of backward packets per second

No	Feature Name	Feature Description
45	Min Packet Length	Minimum inter-arrival time of packet
46	Max Packet Length	Maximum inter-arrival time of packet
47	Packet Length Mean	Mean inter-arrival time of packet
48	Packet Length Std	Standard deviation inter-arrival time of packet
49	Packet Length Variance	Packet Length Variance
50	FIN Flag Count	Number of packets with FIN
51	SYN Flag Count	Number of packets with SYN
52	RST Flag Count	Number of packets with RST
53	PSH Flag Count	Number of packets with PUSH
54	ACK Flag Count	Number of packets with ACK
55	URG Flag Count	Number of packets with URG
56	CWE Flag Count	Number of packets with CWE
57	ECE Flag Count	Number of packets with ECE
58	Down/Up Ratio	Download and upload ratio
59	Average Packet Size	Average size of packet
60	Avg Fwd Segment Size	Average size observed in the forward direction
61	Avg Bwd Segment Size	Average size observed in the backward direction
62	Fwd Avg Bytes/Bulk	Average number of bytes bulk rate in the forward direction
63	Fwd Avg Packets/Bulk	Average number of packets bulk rate in the forward direction
64	Bwd Avg Bulk Rate	Average number of bulk rate in the backward direction
65	Bwd Avg Bytes/Bulk	Average number of bytes bulk rate in the backward direction
66	Bwd Avg Packets/Bulk	Average number of packets bulk rate in the backward direction
67	Bwd Avg Bulk Rate	Average number of bulk rate in the backward direction
68	Subflow Fwd Packets	The average number of packets in a sub flow in the forward direction
69	Subflow Fwd Bytes	The average number of bytes in a sub flow in the forward direction
70	Subflow Bwd Packets	The average number of packets in a sub flow in the backward direction
71	Subflow Bwd Bytes	The average number of bytes in a sub flow in the backward direction
72	Init_Win_bytes_forward	The total number of bytes sent in initial window in the forward direction
73	Init_Win_bytes_backward	The total number of bytes sent in initial window in the backward direction
74	act_data_pkt_fwd	Count of packets with at least 1 byte of TCP data payload in the forward direction
75	min_seg_size_forward	Minimum segment size observed in the forward direction
76	Active Mean	Mean time a flow was active before becoming idle
77	Active Std	Standard deviation time a flow was active before becoming idle
78	Active Max	Maximum time a flow was active before becoming idle
79	Active Min	Minimum time a flow was active before becoming idle
80	Idle Mean	Mean time a flow was idle before becoming active
81	Idle Std	Standard deviation time a flow was idle before becoming active
82	Idle Max	Maximum time a flow was idle before becoming active
83	Idle Min	Minimum time a flow was idle before becoming active
84	Label	Label
85	External IP	External IP

Appendix B.

The List of Attacks

1.DDoS (Distributed Denial of Service): DDoS attacks involve overwhelming a target system or network with an excessive amount of traffic from multiple sources. The goal is to render the target unavailable to its intended users, causing disruption and service downtime.

2.PortScan: Port scanning is a reconnaissance technique where attackers scan a target's computer network to identify open ports and services. This information can be exploited to find potential vulnerabilities for further attacks.

3.Bot: A bot, short for "robot," is a software application that operates autonomously, often maliciously. Bots can be used to automate various tasks, including cyberattacks like DDoS, data theft, and spreading malware.

4.Infiltration: Infiltration refers to unauthorized access into a computer system or network with malicious intent. Attackers gain entry by exploiting vulnerabilities, weak security controls, or human errors.

5. Web Attack – Brute Force: A brute force attack involves systematically trying all possible combinations of passwords until the correct one is found. In the context of web attacks, it's used to gain unauthorized access to accounts or systems.

6. Web Attack – XSS (Cross-Site Scripting): XSS is a vulnerability that allows attackers to inject malicious scripts into web applications, which are then executed by unsuspecting users. This can lead to the theft of user data or session hijacking.

7. Web Attack – SQL Injection: SQL injection occurs when attackers manipulate input fields on a website to inject malicious SQL queries. Successful attacks can lead to unauthorized access to databases and data theft.

8. FTP-Patator: FTP Patator is a type of attack where attackers use automated tools to systematically guess usernames and passwords to gain unauthorized access to FTP (File Transfer Protocol) servers.

9.SSH-Patator: Similar to FTP Patator, SSH Patator involves automated attempts to guess SSH (Secure Shell) login credentials to gain unauthorized access to remote systems.

10.DoS (Denial of Service) – Slowloris: Slowloris is a type of DoS attack where an attacker sends partial HTTP requests to a target server and keeps the connections open by sending small, delayed packets. This ties up server resources and prevents legitimate connections.

11.DoS – Slowhttptest: Slowhttptest is another variant of DoS attack where the attacker sends incomplete HTTP requests to a server, keeping connections open to exhaust server resources.

12.DoS – Hulk: Hulk is a DoS attack that floods a target server with a large number of HTTP GET or POST requests, overwhelming its capacity and causing it to become unresponsive.

13.DoS – GoldenEye: GoldenEye is a DoS attack tool that targets web servers with multiple attack vectors, including SYN flood and HTTP flood, to cause service disruption.

14.Heartbleed: Heartbleed is a security vulnerability in the OpenSSL cryptography library. Attackers can exploit this bug to access sensitive information, such as private keys and user data, from the memory of a vulnerable server.

Appendix C.

Code For Pre-Processing

8/29/23, 1:23 AM

01_preprocessing.ipynb - Colaboratory

```
## CICIDS2017 csv files are required for the operation of the program.
## These files must be located under the "CSVs" folder in the same directory as the program.

## The purpose of this program is to clear the csv files containing CICIDS2017 data from errors.
## the faults observed are:
## 1- 288602 of the entries in the file "Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv" are empty / meaningless.
##      (e.g. ",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,")
##
## 2- In the original csv files, while describing the Web Attack types such as Brute Force, XSS, Sql Injection, the character used is n
##      by the Python-Pandas library and leads to the error.
##      this character ("n", Unicode code:8211) has been changed with another character ("=", Unicode code:45) to correct the e
##
## After the error correction, all the csv files were made into a single file (all_date.csv) to make it easier to process.

import pandas as pd
import os
from sklearn import preprocessing
import time
import pyarrow
seconds = time.time()
%matplotlib inline

print("This process may take 5 to 10 minutes, depending on the performance of your computer.\n\n\n")
number="0123456789"
# CSV files names:
csv_files=["Monday-WorkingHours.pcap_ISCX",
           "Tuesday-WorkingHours.pcap_ISCX",
           "Wednesday-workingHours.pcap_ISCX",
           "Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX",
           "Thursday-WorkingHours-Afternoon-Infiltration.pcap_ISCX",
           "Friday-WorkingHours-Morning.pcap_ISCX",
           "Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX",
           "Friday-WorkingHours-Afternoon-DDos.pcap_ISCX",]

# Headers of column
main_labels=["Flow ID","Source IP","Source Port","Destination IP","Destination Port","Protocol","Timestamp","Flow Duration","Total Fwd Packet
"Total Backward Packets","Total Length of Fwd Packets","Total Length of Bwd Packets","Fwd Packet Length Max","Fwd Packet Length Min",
"Fwd Packet Length Mean","Fwd Packet Length Std","Bwd Packet Length Max","Bwd Packet Length Min","Bwd Packet Length Mean","Bwd Packet Leng
"Flow Bytes/s","Flow Packets/s","Flow IAT Mean","Flow IAT Std","Flow IAT Max","Flow IAT Min","Fwd IAT Total","Fwd IAT Mean","Fwd IAT Std",
"Fwd IAT Min","Bwd IAT Total","Bwd IAT Mean","Bwd IAT Std","Bwd IAT Max","Bwd IAT Min","Fwd PSH Flags","Bwd PSH Flags","Fwd URG Flags","Bw
"Fwd Header Length","Bwd Header Length","Fwd Packets/s","Bwd Packets/s","Min Packet Length","Max Packet Length","Packet Length Mean","Pack
"Packet Length Variance","FIN Flag Count","SYN Flag Count","RST Flag Count","PSH Flag Count","ACK Flag Count","URG Flag Count","CWE Flag C
"ECE Flag Count","Down/Up Ratio","Average Packet Size","Avg Fwd Segment Size","Avg Bwd Segment Size","faulty-Fwd Header Length","Fwd Avg B
"Fwd Avg Packets/Bulk","Fwd Avg Bulk Rate","Bwd Avg Bytes/Bulk","Bwd Avg Packets/Bulk","Bwd Avg Bulk Rate","Subflow Fwd Packets","Subflow
"Subflow Bwd Packets","Subflow Bwd Bytes","Init_Min_bytes_forward","Init_Min_bytes_backward","act_data_pkt_fwd",
"min_seg_size_forward","Active Mean","Active Std","Active Max","Active Min","Idle Mean","Idle Std","Idle Max","Idle Min","Label","External

main_labels2=main_labels
main_labels=( ",".join( i for i in main_labels ) )
main_labels=main_labels+"\n"
flag=True
for i in range(len(csv_files)):
    ths = open(str(i)+".csv", "w")
    ths.write(main_labels)
    with open("./CSVs/"+csv_files[i]+".csv", "r") as file:
        while True:
            try:
                line=file.readline()
                if line[0] in number:# this line eliminates the headers of CSV files and incomplete streams .
                    if "n" in str(line): ## if there is "n" character ("n", Unicode code:8211) in the flow , it will be chanced with "="
                        line=(str(line).replace("n","="))
                        line=(str(line).replace("inf","0"))
                        line=(str(line).replace("Infinity","0"))
                        line=(str(line).replace("NaN","0"))

                        ths.write(str(line))
                    else:
                        continue
            except:
                break
    ths.close()
```

```

df=df.fillna(0)

string_features=["Flow Bytes/s","Flow Packets/s"]
for ii in string_features: #Some data in the "Flow Bytes / s" and "Flow Packets / s" columns are not numeric. Fixing this bug in this loop
    df[ii]=df[ii].replace('Infinity', -1)
    df[ii]=df[ii].replace('NaN', 0)
    number_or_not=[]
    for iii in df[ii]:
        try:
            k=int(float(iii))
            number_or_not.append(int(k))
        except:
            number_or_not.append(iii)
    df[ii]=number_or_not

string_features=[]
for j in main_labels2: # In this section, non-numeric (string and / or categorical) properties (columns) are detected.
    if df[j].dtype=="object":
        string_features.append(j)
try:
    string_features.remove('Label')#The "Label" property was removed from the list. Because it has to remain "categorical" for using with
except:
    print("error!")
labelencoder_X = preprocessing.LabelEncoder()

for ii in string_features: ## In this loop, non-numeric (string and/or categorical) properties converted to numeric features.
    try:
        df[ii]=labelencoder_X.fit_transform(df[ii])
    except:
        df[ii]=df[ii].replace('Infinity', -1)
df=df.drop(main_labels2[61], axis=1) ## Column 61 is deleted because it is unnecessary, column 41 ("Fwd Header Length" feature) had be mi

##All CSV files are merged into a single file.
if flag:
    df.to_csv('all_data.csv',index = False)
    flag=False
else:
    df.to_csv('all_data.csv',index = False,header=False,mode="a")
os.remove(str(i)+".csv")
print("The pre-processing phase of the ",csv_files[i]," file is completed.\n")

print("Total operation time: = ",time.time()- seconds ,"seconds")

 This process may take 5 to 10 minutes, depending on the performance of your computer.

```

The pre-processing phase of the Monday-WorkingHours.pcap_ISCX file is completed.

The pre-processing phase of the Tuesday-WorkingHours.pcap_ISCX file is completed.

The pre-processing phase of the Wednesday-WorkingHours.pcap_ISCX file is completed.

The pre-processing phase of the Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX file is completed.

The pre-processing phase of the Thursday-WorkingHours-Afternoon-Infiltration.pcap_ISCX file is completed.

The pre-processing phase of the Friday-WorkingHours-Morning.pcap_ISCX file is completed.

The pre-processing phase of the Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX file is completed.

The pre-processing phase of the Friday-WorkingHours-Afternoon-DDos.pcap_ISCX file is completed.

Mission accomplished!

Total operation time: = 157.36881017684937 seconds

Appendix D.

Code For EDA/Statistics

8/29/23, 1:29 AM

02_statistics.ipynb - Colaboratory

Imports

```
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
import time
import seaborn as sns
seconds = time.time()
```

Data Analysis Program

This program requires the presence of the `all_data.csv` file in the same directory. The purpose of the program is to provide statistical insights about the data contained in the dataset.

Due to the varying sizes of the data, the program generates graphics in three separate groups, allowing all data to be visualized effectively:

- **Big Group:** Labels with more than 11,000 occurrences.
- **Medium Group:** Labels with occurrences between 600 and 11,000.
- **Small Group:** Labels with fewer than 600 occurrences.

The final set of graphics provides the attack rates and normal behavior rates.

Instructions

1. Ensure the `all_data.csv` file is present in the program's directory.
2. Run the program to generate graphics and statistics.

Program Output

The program generates visualizations and statistics for the dataset based on the label groupings. It aims to provide insights into the distribution and behavior of the data.

Note: This program assumes that the `pandas`, `matplotlib`, `seaborn` check the imports section and make sure other required libraries are installed.

```
# graph creation function
def graph(objects,performance,x_label,y_label):
    y_pos = np.arange(len(objects))
    plt.barh(y_pos, performance, align='center', alpha=0.5, color='g', edgecolor='black')
    plt.yticks(y_pos, objects)
    plt.xlabel(x_label)
    plt.title(y_label)
    plt.show()

df=pd.read_feather('all_data.feather')
df = df[['Label']]
label_count = df.iloc[:,0].value_counts()
label_count_df = pd.DataFrame(label_count)
a=(df.iloc[:,0].value_counts())

key=a.keys()
values=a.values
small_labels=[]
small_values=[]
big_labels=[]
big_values=[]
medium_labels=[]
medium_values=[]
attack=0
benign=0
label_count_df.style.background_gradient()
```

```

## In this section, the attacks are grouped under 3 groups,
## so that all values can be seen on the graph.
for i in range(0,len(values)):
    if values[i]>1000:
        big_labels.append(str(key[i]))
        big_values.append(values[i])
    elif values[i]<600:
        small_labels.append(str(key[i]))
        small_values.append(values[i])
    else:
        medium_labels.append(str(key[i]))
        medium_values.append(values[i])

    if str(key[i])=="BENIGN":
        benign+=values[i]
    else:
        attack+=values[i]

key =[benign,attack]

#functions are called to create a chartes
labels=["BENIGN %"+str(round(benign/(benign+attack),2)*100),
        "ATTACK %"+str(round(attack/(benign+attack),2)*100)]
graph(big_labels,big_values,"Numbers","Attacks Labels - High-number group")
graph(medium_labels,medium_values,"Numbers","Attacks Labels - Medium-number group")
graph(small_labels,small_values,"Numbers","Attacks Labels - Small -number group")
graph(labels,key,"Numbers","Attack and Benign Percentage")

print("mission accomplished!")
print("Total operation time: = ",time.time()- seconds ,"seconds")

```

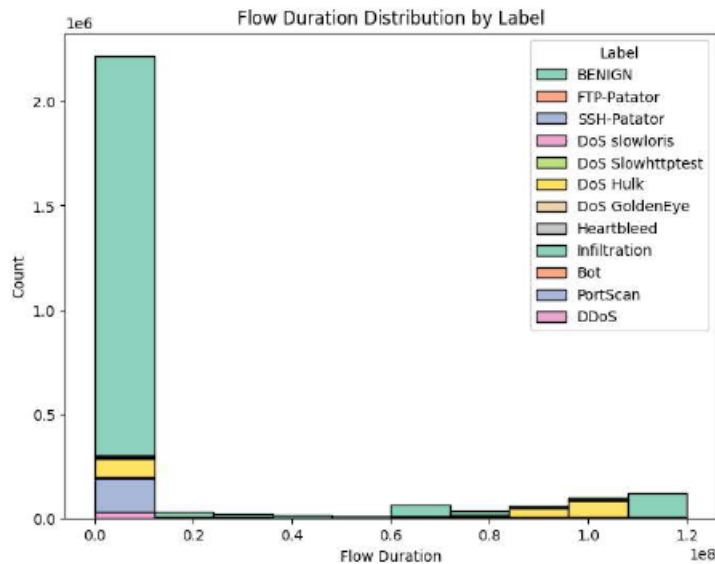
Outliers in a box plot are data points that fall significantly outside the overall distribution of the data. When you see more outliers for the "Benign" label, it could suggest the following:

Higher Variability: The "Benign" label may have higher variability or a wider spread of average packet sizes compared to the "Attack" label. This could be due to a range of factors, such as diverse types of benign network traffic or a broader range of sources.

dat.columns

```
Index(['Flow ID', 'Source IP', 'Source Port', 'Destination IP',
      'Destination Port', 'Protocol', 'Timestamp', 'Flow Duration',
      'Total Fwd Packets', 'Total Backward Packets',
      'Total Length of Fwd Packets', 'Total Length of Bwd Packets',
      'Fwd Packet Length Max', 'Fwd Packet Length Min',
      'Fwd Packet Length Mean', 'Fwd Packet Length Std',
      'Bwd Packet Length Max', 'Bwd Packet Length Min',
      'Bwd Packet Length Mean', 'Bwd Packet Length Std', 'Flow Bytes/s',
      'Flow Packets/s', 'Flow IAT Mean', 'Flow IAT Std', 'Flow IAT Max',
      'Flow IAT Min', 'Fwd IAT Total', 'Fwd IAT Mean', 'Fwd IAT Std',
      'Fwd IAT Max', 'Fwd IAT Min', 'Bwd IAT Total', 'Bwd IAT Mean',
      'Bwd IAT Std', 'Bwd IAT Max', 'Bwd IAT Min', 'Fwd PSH Flags',
      'Bwd PSH Flags', 'Fwd URG Flags', 'Bwd URG Flags', 'Fwd Header Length',
      'Bwd Header Length', 'Fwd Packets/s', 'Bwd Packets/s',
      'Min Packet Length', 'Max Packet Length', 'Packet Length Mean',
      'Packet Length Std', 'Packet Length Variance', 'FIN Flag Count',
      'SYN Flag Count', 'RST Flag Count', 'PSH Flag Count', 'ACK Flag Count',
      'URG Flag Count', 'CWE Flag Count', 'ECE Flag Count', 'Down/Up Ratio',
      'Average Packet Size', 'Avg Fwd Segment Size', 'Avg Bwd Segment Size',
      'Fwd Avg Bytes/Bulk', 'Fwd Avg Packets/Bulk', 'Fwd Avg Bulk Rate',
      'Bwd Avg Bytes/Bulk', 'Bwd Avg Packets/Bulk', 'Bwd Avg Bulk Rate',
      'Subflow Fwd Packets', 'Subflow Fwd Bytes', 'Subflow Bwd Packets',
      'Subflow Bwd Bytes', 'Init_win_bytes_forward',
      'Init_win_bytes_backward', 'act_data_pkt_fwd', 'min_seg_size_forward',
      'Active Mean', 'Active Std', 'Active Max', 'Active Min', 'Idle Mean',
      'Idle Std', 'Idle Max', 'Idle Min', 'Label', 'External IP'],
      dtype='object')
```

```
# Flow Duration Distribution (Histogram)
plt.figure(figsize=(8, 6))
sns.histplot(data=dat, x='Flow Duration', bins=10, hue='Label', multiple='stack', palette='Set2')
plt.title('Flow Duration Distribution by Label')
plt.xlabel('Flow Duration')
plt.ylabel('Count')
plt.show()
```



Appendix E.

Code For Feature Selection

8/29/23, 1:34 AM

04_2_feature_selection_for_all_data.ipynb - Colaboratory

```
## "all_data.csv" file is required for the operation of the program.
## "all_data.csv" file must be located in the same directory as the program.

## the purpose of this code is to determine which features to use in the machine learning phase.
## for this purpose, the importance weights of the attacks are calculated.
## this calculation was made using sklearn=RandomForestRegressor.

## the some codes parts used for calculation and graphing are taken from the following site.
## http://scikit-learn.org/stable/auto\_examples/ensemble/plot\_forest\_importances.html

import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, KFold, StratifiedKFold
import sklearn as sk
import time
seconds = time.time()
```

▼ GridSearch

```
def folder(f_name): #this function creates a folder named "feare_pics" in the program directory.
    try:
        if not os.path.exists(f_name):
            os.makedirs(f_name)
    except OSError:
        print ("The folder could not be created!")

# CSV files names:
csv_files=["all_data.feather"]# It creates a list of file names in the "attacks" folder.

# Headers of column
main_labels=["Flow Duration","Total Fwd Packets", "Total Backward Packets","Total Length of Fwd Packets","Total Length of Bwd Packets","Fwd Packet Length Mean","Fwd Packet Length Std","Bwd Packet Length Max","Bwd Packet Length Min","Bwd Packet Length Mean","Bwd Packet Length Std","Flow Bytes/s","Flow Packets/s","Flow IAT Mean","Flow IAT Std","Flow IAT Max","Flow IAT Min","Fwd IAT Total","Fwd IAT Mean","Fwd IAT Std","Fwd IAT Min","Bwd IAT Total","Bwd IAT Mean","Bwd IAT Std","Bwd IAT Max","Bwd IAT Min","Fwd PSH Flags","Bwd PSH Flags","Fwd URG Flags","Bwd URG Flags","Fwd Header Length","Bwd Header Length","Fwd Packets/s","Bwd Packets/s","Min Packet Length","Max Packet Length","Packet Length Mean","Packet Length Variance","FIN Flag Count","SYN Flag Count","RST Flag Count","PSH Flag Count","ACK Flag Count","URG Flag Count","CWE Flag Count","ECE Flag Count","Down/Up Ratio","Average Packet Size","Avg Fwd Segment Size","Fwd Avg Segment Size","Fwd Avg Bytes/Bulk","Fwd Avg Packets/Bulk","Fwd Avg Bulk Rate","Bwd Avg Bytes/Bulk","Bwd Avg Packets/Bulk","Bwd Avg Bulk Rate","Subflow Fwd Packets","Subflow Bwd Packets","Subflow Bwd Bytes","Init_Win_bytes_forward","Init_Win_bytes_backward","act_data_pkt_fwd","min_seg_size_forward","Active Mean","Active Std","Active Max","Active Min","Idle Mean","Idle Std","Idle Max","Idle Min","Label"]

ths = open("importance_list_all_data.csv", "w")
folder("./feare_pics/")
for j in csv_files:
    df=pd.read_feather(j)
    df = df[main_labels]
    df=df.fillna(0)
    attack_or_not=[]
    for i in df["Label"]:#it changes the normal label to "1" and the attack tag to "0" for use in the machine learning algorithm
        if i == "BENIGN":
            attack_or_not.append(1)
        else:
            attack_or_not.append(0)
    df["Label"]=attack_or_not

y = df["Label"].values
del df["Label"]
X = df.values

X = np.float32(X)
X[np.isnan(X)] = 0
X[np.isinf(X)] = 0
kfold = KFold(n_splits=5, shuffle=True, random_state=0)
rf = sk.ensemble.RandomForestRegressor()
```


8/29/23, 1:34 AM

04_2_feature_selection_for_all_data.ipynb - Colaboratory

```

params = {
    'max_depth':[None, 3, 4, 5, 6, 7, 8, 9, 10],
    'min_samples_split':[2, 5, 10, 20],
    'min_samples_leaf':[1, 5, 10, 20],
    'n_estimators':[50, 100, 150, 250],
    'n_jobs':[-1]
}

gcv = RandomizedSearchCV(rf, cv=kfold, verbose=3, param_distributions=params)
gcv.fit(X,y)

#computing the feature importances
# forest = sk.ensemble.RandomForestRegressor(max_depth=10, n_estimators=150, random_state=0, n_jobs=-1, verbose=3)
# forest.fit(X, y)
# importances = forest.feature_importances_
importances = gcv.feature_importances_

# std = np.std([tree.feature_importances_ for tree in forest.estimators_],axis=0)
std = np.std([tree.feature_importances_ for tree in rf.estimators_],axis=0)

indices = np.argsort(importances)[::-1]
refclasscol=list(df.columns.values)
import_bar = pd.DataFrame({'Features':refclasscol[0:20], 'importance':importances[0:20]})
import_bar = import_bar.sort_values('importance',ascending=False).set_index('Features')
plt.rcParams['figure.figsize'] = (10, 5)
import_bar.plot.bar();
#printing the feature importances
count=0
fea_ture=j[0:-4]+="-["
for i in import_bar.index:
    fea_ture=fea_ture+"\""+str(i)+"\"","
    count+=1
    if count==5:
        fea_ture=fea_ture[0:-1]+"]"
        break
print(j[0:-4],"importance list:")
print(j[0:-4],"n",import_bar.head(20),"n\n\n")
print(fea_ture)
plt.title(j[0:-4]+" Attack - Feature Importance")
plt.xlabel('Importance')
#plt.savefig("./feature_pics/"+j[0:-4]+".pdf",bbox_inches='tight', papertype = 'a4', orientation = 'portrait', format = 'pdf')
#ths.write(( fea_ture ) )
plt.tight_layout()
plt.show()
print("-----\n\n\n\n\n")

print("mission accomplished!")
print("Total operation time: = ",time.time()- seconds ,"seconds")
ths.close()

Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV 1/5] END max_depth=3, min_samples_leaf=20, min_samples_split=10, n_estimators=50, n_jobs=-1, score=0.690 total time= 3.7min
[CV 2/5] END max_depth=3, min_samples_leaf=20, min_samples_split=10, n_estimators=50, n_jobs=-1, score=0.688 total time= 3.7min
[CV 3/5] END max_depth=3, min_samples_leaf=20, min_samples_split=10, n_estimators=50, n_jobs=-1, score=0.689 total time= 3.6min
[CV 4/5] END max_depth=3, min_samples_leaf=20, min_samples_split=10, n_estimators=50, n_jobs=-1, score=0.689 total time= 3.6min
[CV 5/5] END max_depth=3, min_samples_leaf=20, min_samples_split=10, n_estimators=50, n_jobs=-1, score=0.687 total time= 3.7min
[CV 1/5] END max_depth=9, min_samples_leaf=5, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.911 total time=19.4min
[CV 2/5] END max_depth=9, min_samples_leaf=5, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.909 total time=19.4min
[CV 3/5] END max_depth=9, min_samples_leaf=5, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.909 total time=19.3min
[CV 4/5] END max_depth=9, min_samples_leaf=5, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.910 total time=20.4min
[CV 5/5] END max_depth=9, min_samples_leaf=5, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.910 total time=24.0min
[CV 1/5] END max_depth=None, min_samples_leaf=20, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.945 total time=46.0min
[CV 2/5] END max_depth=None, min_samples_leaf=20, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.943 total time=43.3min
[CV 3/5] END max_depth=None, min_samples_leaf=20, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.944 total time=41.8min
[CV 4/5] END max_depth=None, min_samples_leaf=20, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.944 total time=43.5min
[CV 5/5] END max_depth=None, min_samples_leaf=20, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.943 total time=43.1min
[CV 1/5] END max_depth=10, min_samples_leaf=1, min_samples_split=20, n_estimators=250, n_jobs=-1, score=0.922 total time=52.4min
[CV 2/5] END max_depth=10, min_samples_leaf=1, min_samples_split=20, n_estimators=250, n_jobs=-1, score=0.922 total time=52.4min
[CV 3/5] END max_depth=10, min_samples_leaf=1, min_samples_split=20, n_estimators=250, n_jobs=-1, score=0.922 total time=52.0min
[CV 4/5] END max_depth=10, min_samples_leaf=1, min_samples_split=20, n_estimators=250, n_jobs=-1, score=0.922 total time=55.5min
[CV 5/5] END max_depth=10, min_samples_leaf=1, min_samples_split=20, n_estimators=250, n_jobs=-1, score=0.923 total time=58.8min
[CV 1/5] END max_depth=5, min_samples_leaf=20, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.796 total time=11.5min
[CV 2/5] END max_depth=5, min_samples_leaf=20, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.794 total time=11.6min
[CV 3/5] END max_depth=5, min_samples_leaf=20, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.795 total time=11.5min
[CV 4/5] END max_depth=5, min_samples_leaf=20, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.796 total time=11.5min
[CV 5/5] END max_depth=5, min_samples_leaf=20, min_samples_split=20, n_estimators=100, n_jobs=-1, score=0.793 total time=11.4min
[CV 1/5] END max_depth=3, min_samples_leaf=10, min_samples_split=10, n_estimators=50, n_jobs=-1, score=0.690 total time= 3.7min
[CV 2/5] END max_depth=3, min_samples_leaf=10, min_samples_split=10, n_estimators=50, n_jobs=-1, score=0.688 total time= 3.7min
[CV 3/5] END max_depth=3, min_samples_leaf=10, min_samples_split=10, n_estimators=50, n_jobs=-1, score=0.689 total time= 3.7min
[CV 4/5] END max_depth=3, min_samples_leaf=10, min_samples_split=10, n_estimators=50, n_jobs=-1, score=0.689 total time= 3.7min

```

▼ Random forest

```
def folder(f_name): #this function creates a folder named "feare_pics" in the program directory.
    try:
        if not os.path.exists(f_name):
            os.makedirs(f_name)
    except OSError:
        print ("The folder could not be created!")

# CSV files names:
csv_files=["all_data.feather"]# It creates a list of file names in the "attacks" folder.

# Headers of column
main_labels=["Flow Duration","Total Fwd Packets", "Total Backward Packets","Total Length of Fwd Packets","Total Length of Bwd Packets","Fwd
    "Fwd Packet Length Mean","Fwd Packet Length Std","Bwd Packet Length Max","Bwd Packet Length Min","Bwd Packet Length Mean","Bwd Packet Leng
    "Flow Bytes/s","Flow Packets/s","Flow IAT Mean","Flow IAT Std","Flow IAT Max","Flow IAT Min","Fwd IAT Total","Fwd IAT Mean","Fwd IAT Std",
    "Fwd IAT Min","Bwd IAT Total","Bwd IAT Mean","Bwd IAT Std","Bwd IAT Max","Bwd IAT Min","Fwd PSH Flags","Bwd PSH Flags","Fwd URG Flags","Bw
    "Fwd Header Length","Bwd Header Length","Fwd Packets/s","Bwd Packets/s","Min Packet Length","Max Packet Length","Packet Length Mean","Pack
    "Packet Length Variance","FIN Flag Count","SYN Flag Count","RST Flag Count","PSH Flag Count","ACK Flag Count","URG Flag Count","CWE Flag C
    "ECE Flag Count","Down/Up Ratio","Average Packet Size","Avg Fwd Segment Size","Avg Bwd Segment Size","Fwd Avg Bytes/Bulk",
    "Fwd Avg Packets/Bulk","Fwd Avg Bulk Rate","Bwd Avg Bytes/Bulk","Bwd Avg Packets/Bulk","Bwd Avg Bulk Rate","Subflow Fwd Packets","Subflow
    "Subflow Bwd Packets","Subflow Bwd Bytes","Init_Win_bytes_forward","Init_Win_bytes_backward","act_data_pkt_fwd",
    "min_seg_size_forward","Active Mean","Active Std","Active Max","Active Min",
    "Idle Mean","Idle Std","Idle Max", "Idle Min","Label"]

ths = open("importance_list_all_data.csv", "w")
folder("./feare_pics/")
for j in csv_files:
    df=pd.read_feather(j)
    df = df[main_labels]
    df=df.fillna(0)
    attack_or_not=[]
    for i in df["Label"]:#it changes the normal label to "1" and the attack tag to "0" for use in the machine learning algorithm
        if i == "BENIGN":
            attack_or_not.append(1)
        else:
            attack_or_not.append(0)
    df["Label"]=attack_or_not

    y = df["Label"].values
    del df["Label"]
    X = df.values

    X = np.float32(X)
    X[np.isnan(X)] = 0
    X[np.isinf(X)] = 0

    #computing the feature importances
    forest = sk.ensemble.RandomForestRegressor(max_depth=None, min_samples_leaf=20, min_samples_split=20, n_estimators=100, random_state=0, n
    forest.fit(X, y)
    importances = forest.feature_importances_
    importances = forest.feature_importances_
    std = np.std([tree.feature_importances_ for tree in forest.estimators_],axis=0)
    indices = np.argsort(importances)[::-1]
    refclasscol=list(df.columns.values)
    impor_bars = pd.DataFrame({'Features':refclasscol[0:20],'importance':importances[0:20]})
    impor_bars = impor_bars.sort_values('importance',ascending=False).set_index('Features')
    plt.rcParams['figure.figsize'] = (10, 5)
    impor_bars.plot.bar();

    #printing the feature importances
    count=0
    fea_ture=j[0:-4]+"="
    for i in impor_bars.index:
        fea_ture=fea_ture+"\n"+str(i)+"\n",
        count+=1
    if count==5:

8/29/23, 1:34 AM 04_2_feature_selection_for_all_data.ipynb - Colaboratory

        fea_ture=fea_ture[0:-1]+"`"
        break
    print(j[0:-4],"importance list:")
    print(j[0:-4],"\n",impor_bars.head(20),"\n\n")
    print(fea_ture)
    plt.title(j[0:-4]+" Attack - Feature Importance")
    plt.ylabel('Importance')
    plt.tight_layout()
    plt.show()
    print("-----\n\n\n\n\n")

print("Total operation time: = ",time.time()- seconds , "seconds")
ths.close()
```

Appendix F.

Code For Machine Learning Algorithm Implementation

▼ Required Files

To run this program, you need an "all_data.csv" file. The "all_data.csv" file should be located in the same directory as the program.

Program Purpose

The purpose of this program is to apply various machine learning algorithms to a dataset and observe their performance. The algorithms used in this program are:

- Naive Bayes
- QDA (Quadratic Discriminant Analysis)
- Random Forest
- ID3 (Iterative Dichotomiser 3)
- AdaBoost
- MLP (Multi-Layer Perceptron)
- Nearest Neighbors

The program's output includes the following information for each algorithm:

- File name
- Machine learning algorithm name
- Accuracy
- Precision
- Recall
- F1-score
- Time taken

Additionally, the program will generate a CSV file containing the results and a folder containing graphics.

Attribution

Some portions of the code used for calculations and graphing have been adapted from the [scikit-learn website](https://scikit-learn.org/).

```
from sklearn import metrics
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import average_precision_score, confusion_matrix, roc_curve, roc_auc_score, auc
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline
import os
import pandas as pd
import csv
import time
import warnings
import math
import pickle
warnings.filterwarnings("ignore")
```

▼ test saving best model

```

result="./results/results_3.csv" #a CSV file is named in which the results are saved.
csv_files=["all_data.csv"]# CSV files names: #The names of the dataset files (csv_files).
path=""
repetition=10

def folder(f_name): #this function creates a folder named "results" and "result_graph_1" in the program directory.
    try:
        if not os.path.exists(f_name):
            os.makedirs(f_name)
    except OSError:
        print ("The folder could not be created!")

folder_name="./results/"
folder(folder_name)
folder_name="./results/result_graph_3/"
folder(folder_name)

#The machine learning algorithms to be used are defined in a dictionary (ml_list).
ml_list={
    "Naive Bayes":GaussianNB(),
    "QDA":QDA(),
    "XGB":XGBClassifier(n_estimators=100, learning_rate=0.01, objective='binary:logistic'),
    "Random Forest":RandomForestClassifier(max_depth=5, n_estimators=100, max_features=2, n_jobs=-1),
    "ID3":DecisionTreeClassifier(max_depth=5,criterion="entropy"),
    "AdaBoost":AdaBoostClassifier(),
    "MLP":MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500),
    "Nearest Neighbors":KNeighborsClassifier(3, n_jobs=-1)}

#list of all columns to be imported
# the 7 features with the highest importance weight selected by the file "04_2_feature_selection_for_attack_files.py" are used here. (+ Label
features={"all_data":["Bwd Packet Length Std", "Flow Bytes/s", "Total Length of Fwd Packets", "Fwd Packet Length Std",
    "Flow IAT Std", "Flow IAT Min", "Fwd IAT Total","Label"]}

seconds=time.time()#time stamp for all processing time

with open(result, "w", newline="",encoding="utf-8") as f:#a CSV file is created to save the results obtained.
    wrt = csv.writer(f)
    wrt.writerow(["File","ML algorithm","accuracy","Precision", "Recall" , "F1-score","Time"])

best_model = {
    'algorithm': None,
    'roc_auc': 0.0, # Initialize with a low value
    'model': None, # Store the trained model object
}

for j in csv_files: #this loop runs on the list containing the filenames.Operations are repeated for all attack files
    print ('%-15s %-15s  %-12s %-12s %-12s %-12s %-12s %-12s' % ("File","ML algorithm","accuracy","Precision", "Recall" , "F1-score","Time",
    feature_list=list(features[j[0:-4]]))
    df=pd.read_csv(path+j,usecols=feature_list)#read an attack file.
    df=df.fillna(0)
    attack_or_not=[]
    for i in df["Label"]: #it changes the normal label to "1" and the attack tag to "0" for use in the machine learning algorithm
        if i == "BENIGN":
            attack_or_not.append(1)
        else:
            attack_or_not.append(0)
    df["Label"]=attack_or_not

    y = df["Label"] #this section separates the label and the data into two separate pieces, as Label=y Data=X
    del df["Label"]
    feature_list.remove('Label')
    X = df[feature_list]

    for ii in ml_list: #this loop runs on the list containing the machine learning algorithm names. Operations are repeated for all the 7 alg
        precision=[]
        recall=[]
        fi=[]
        accuracy=[]

```

```

t_time=[]
roc_auc = []
for i in range(repetition): # This loop allows cross-validation and machine learning algorithm to be repeated 10 times
    second=time.time()#time stamp for processing time

    # cross-validation
    X_train, X_test, y_train, y_test = train_test_split(X, y,# data (X) and labels (y) are divided into 2 parts to be sent to the ma
        test_size = 0.20, random_state = repetition)# So, in total there are 4 tracks: training data(X_train), training tag (y_train

    #machine learning algorithm is applied in this section
    clf = ml_list[ii]#choose algorithm from ml_list dictionary
    clf.fit(X_train, y_train)
    predict =clf.predict(X_test)

    #makes "classification report" and assigns the precision, f-measure, and recall values.s.

    f1=f1_score(y_test, predict, average='macro')
    pr=precision_score(y_test, predict, average='macro')
    rc=recall_score(y_test, predict, average='macro')

    precision.append(float(pr))
    recall.append(float(rc))
    f1.append(float(f1))
    accuracy.append(clf.score(X_test, y_test))
    t_time.append(float((time.time()-second)) )

    y_scores = clf.predict_proba(X_test)[:, 1] # Use predicted probabilities for positive class
    fpr, tpr, _ = roc_curve(y_test, y_scores)
    roc_auc.append(auc(fpr, tpr))

# Plot ROC curve
plt.figure(figsize=(5,3))
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % np.mean(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - ' + str(ii))
plt.legend(loc="lower right")
plt.show()

cm = confusion_matrix(y_test, predict)

# Plot confusion matrix
plt.figure(figsize=(3, 3))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["BENIGN", "Attack"], yticklabels=["BENIGN", "Attack"])
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title(f"Confusion Matrix - {ii}")
plt.show()

print ('%-15s %-15s %-12s %-12s %-12s %-12s %-12s %-12s' % (j[0:-4],ii,str(round(np.mean(accuracy),2)),str(round(np.mean(precision),
    str(round(np.mean(recall),2)),str(round(np.mean(f1),2)),str(round(np.mean(t_time),4)),str(round(np.mean(roc_auc),4))))#the result
if np.mean(roc_auc) > best_model['roc_auc']:
    best_model['algorithm'] = ii
    best_model['roc_auc'] = np.mean(roc_auc)
    best_model['model'] = clf

with open(result, "a", newline="",encoding="utf-8") as f: # all the values found are saved in the opened file.
    wrt = csv.writer(f)
    for i in range(0,len(t_time)):
        wrt.writerow([j[0:-4],ii,accuracy[i],precision[i],recall[i],f1[i],t_time[i]])#file name, algorithm name, precision, recall an

if best_model['model'] is not None:
    best_model_filename = 'best_model.pkl'
    with open(best_model_filename, 'wb') as model_file:
        pickle.dump(best_model['model'], model_file)
    print(f'Best model ({best_model["algorithm"]}) saved as {best_model_filename}')
else:
    print('No best model found.')

print("Total operation time: = ",time.time()- seconds ,"seconds")

```