A project report on

ML-Based Hybrid Classifier For Network Intrusion Detection System

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

by

R AKSHAY MADHAVARAJ (19BCE1844)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April, 2023

ML-Based Hybrid Classifier For Network Intrusion Detection System

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

by

R AKSHAY MADHAVARAJ (19BCE1844)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April, 2023

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I here by declare that the thesis entitled "ML-Based Hybrid Classifier For Network Intrusion Detection System" submitted by me, for the award of the degree of Bachelor of Technology in Computer Science and Engineering, Vellore Institute of Technology, Chennai, is a record of bonafide work carried out by me under the supervision of Dr. N G Bhuvaneswari Amma.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: Signature of the Candidate

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **"ML-Based Hybrid Classifier For Network Intrusion Detection System"** is prepared and submitted by **R Akshay Madhavaraj (19BCE1844)** to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** programme is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:

Name:

Date:

Signature of the Examiner 1	Signature of the Examiner 2
Name:	Name:
Date:	Date:

Approved by the Head of Department
B. Tech. CSE

Name: Dr. Nithyanandam P

Date: 24 – 04 – 2023

(Seal of SCOPE)

ABSTRACT

The traditional signature-based NIDS (Network-based Intrusion Detection System) are very inefficient in detecting modern day attacks and it is safe to say they are very much outdated. The traditional signature-based NIDS are currently replaced by the Anomaly-based NIDS which is the next level solution for the need of a smarter and highly efficient intrusion detection system. Various Machine Learning and Deep Learning classifiers were proposed over the years and the search for a optimal anomaly-based NIDS is still on.

Most of the classifiers which produces high scores in terms of performance metrics are very complicated deep learning models. The ML-based classifiers faces challenges like moderate performance on large network traffic dataset, high false alarm rates, etc. which makes it very inefficient comparing the deep learning classifiers. To address these challenges we have developed a new hybrid ML-based classifier, called RFXG (Random Forest - XGBoost). For the purpose of evaluation and constructive comparison the RFXG model is trained alongside four other ML models namely Decision Tree, Random Forest, Naive Bayes and XGBoost.

The RFXG, even though less in complexity, has provided high scores in various performance metrics. The model when trained and tested with the CICIDS17 dataset gave out results of 99.91% accuracy, 99.90% precision, 99.87% recall value, 99,89% F1 score, 99.97% AUC score and a low false alarm rate of 0.07% which clearly outperformed many of the previous works.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. N G Bhuvaneswari Amma, Assistant Professor Sr. , SCOPE, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of cybersecurity.

It is with gratitude that I would like to extend thanks to our honorable Chancellor, Dr. G. Viswanathan, Vice Presidents, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan and Mr. G V Selvam, Assistant Vice-President, Ms. Kadhambari S. Viswanathan, Vice-Chancellor, Dr. Rambabu Kodali, Pro-Vice Chancellor, Dr. V. S. Kanchana Bhaaskaran and Additional Registrar, Dr. P.K.Manoharan for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dean, Dr. Ganesan R, Associate Dean Academics, Dr. Parvathi R and Associate Dean Research, Dr. Geetha S, SCOPE, Vellore Institute of Technology, Chennai, for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Nithyanandam P, Head of the Department, Project Coordinators, Dr. Abdul Quadir Md, Dr. Priyadarshini R and Dr. Padmavathy T V, B. Tech. Computer Science and Engineering, SCOPE, Vellore Institute of Technology, Chennai, for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staff at Vellore Institute of Technology, Chennai, who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: Name of the student

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

IDS	- Intrusion Detection System
NIDS	- Network Intrusion Detection System
HIDS	- Hybrid Intrusion Detection System
RFXG	- Random Forest XGBoost
ML	- Machine Learning
FP	- False Positive
FN	- False Negative
TP	- True Positive
TN	- True Negative
DBN	- Deep Belief Networks
CNN	- Convolutional Neural Networks
LSTM	- Long Short-Term Memory
GNB	- Gaussian Naive Bayes
PSO	- Particle Swarm Optimization
KNN	- K-Nearest Neighbors
FTP	- File Transfer Protocol
DoS	- Denial of Service
DDoS	- Distributed Denial of Service
SMOTE	- Synthetic Minority Oversampling Technique
RF	- Random Forest
DT	- Decision Tree
NB	- Naive Bayes
AUC	- Area Under Curve
ROC	- Receiver Operating Characteristic

CHAPTER 1

Introduction

1.1 INTRUSION DETECTION SYSTEM

Intrusion Detection Systems (IDS) are crucial components of any computer network security infrastructure. They play a vital role in identifying and alerting the system administrator about any unauthorized access or activity on the network. Traditional IDS solutions rely on signature-based detection techniques to identify known attacks. However, they are less effective in detecting unknown or novel attacks that do not match any known signature. Machine Learning (ML) has emerged as a promising approach to overcome this limitation [1]. By leveraging ML algorithms, IDS can detect unknown attacks by learning from historical data and identifying deviations from normal behavior.

1.1.1 HYBRID INTRUSION DETECTION SYSTEM

Hybrid intrusion detection systems combine multiple detection techniques, including both signature-based and ML-based techniques [2][3], to improve the overall detection accuracy and reduce false positive rates. These systems can be customized to meet the specific needs of an organization and provide an effective defense against various types of attacks. Overall, hybrid intrusion detection systems that use machine learning are becoming increasingly popular due to their ability to detect both known and unknown attacks, adapt to changing attack patterns, and provide a more robust defense against various types of cyber threats.

In recent years, the number of cyber attacks has increased significantly, which has raised concerns about the security of computer systems and networks. Intrusion detection systems (IDSs) have been developed to detect such attacks and prevent them from causing harm. Machine learning (ML) techniques have been extensively used in IDSs to automatically identify malicious activities and intrusions [4]. However, each ML technique has its strengths and limitations, and none of them can accurately detect different types of attacks. To overcome this limitation, hybrid intrusion detection systems have been proposed, which combine multiple ML techniques to improve detection accuracy and reduce false positives [5]. The integration of multiple ML techniques allows the system to leverage the strengths of each technique while minimizing their weaknesses. It has been proven that using this strategy would lower the likelihood of successful attacks and considerably increase the efficacy of intrusion detection systems. Many ML approaches, including decision trees, K-nearest neighbors, artificial neural networks, have been utilised in tandem with hybrid IDSs to identify various sorts of attacks. Such a problem can be addressed by using a CICIDS2017 dataset that contains novel attack network traces.

1.2 CHALLENGES PRESENT IN NIDS

The purpose of Network Intrusion Detection Systems (NIDS) is to immediately identify and react to malicious network activity. To increase NIDS' accuracy and effectiveness, machine learning (ML) is increasingly being used. Nevertheless, NIDS that use ML encounter a number of difficulties. Some of these difficulties include quality of dataset, adaptability, complexity of model, overfitting, cost and scalability.

1.2.1 QUALITY OF DATASET

To train their models, NIDS that use ML need a lot of high-quality labelled data. Unfortunately, labelled data may not always be accessible, or the data that is may be noisy or lacking in some information, resulting in poor performance of the model.

1.2.2 ADAPTABILITY

As attackers regularly change their tactics and techniques, NIDS which use ML must be capable to adapt as well without losing efficacy. Real-time ML model updating can be challenging, and employing older models could result in false positives (FP) or false negatives (FN).

1.2.3 COMPLEXITY OF MODEL

ML models are frequently dubbed as the "black boxes" since it can be difficult to understand how they arrive at conclusions. This limitation of explainability makes NIDS models difficult to troubleshoot or enhance.

1.2.4 OVERFITTING

ML models may perform poorly on new, untrained data due to overfitting to the training data set. Because to the potential diversity and amount of network traffic, this is particularly challenging for NIDS.

1.2.5 COST AND SCALABILITY

The development and execution of a successful ML-based NIDS can be a time- and resource-consuming procedure. Large training data sets can be expensive to acquire, label, and store, and extending the system to cope with high levels of network traffic might demand a lot of computational power.

In general, overcoming these challenges necessitates striking a fine line between the complexity of the ML models, the quality of the dataset , the system's scalability, and the capacity to adapt to novel threats.

CHAPTER 2

Background

2.1 RELATED WORKS

In recent years, several studies have been conducted to improve the performance of Hybrid Intrusion Detection Systems (HIDS) using Machine Learning (ML) algorithms. One such study conducted by Malik et al. (2020) [15] proposed a hybrid approach using Deep Belief Networks (DBN) and Decision Trees (DT) algorithm which showed improved accuracy and reduced false alarm rates. Similarly, Wang et al. (2021) [16] proposed a hybrid approach using Convolutional Neural Networks (CNN) and the Long Short-Term Memory (LSTM) algorithm which showed improved detection rates for network attacks. Furthermore, another recent study conducted by Kumar et al. (2021) [17] proposed a hybrid approach using the CatBoost algorithm and rule-based approach for detecting network attacks. This approach showed significant improvements in terms of accuracy and detection rates.
Moreover, recent studies have also explored the use of other ML algorithms in HIDS. For example, in a study conducted by Goyal et al. (2021) [18], a hybrid approach using the XGBoost algorithm and rule-based approach was proposed, which showed significant improvements in terms of accuracy, sensitivity, and specificity.
Another study by Kaur et al. (2021) [19] explored the use of the Artificial Bee Colony (ABC) algorithm and the Naive Bayes algorithm in a hybrid approach for HIDS. The proposed approach showed improved performance in terms of detection rates for network attacks. Another recent study by Bhuyan et al. (2021) [20] proposed a hybrid approach using the Gaussian Naive Bayes (GNB) algorithm and the Particle Swarm Optimization (PSO) algorithm. The proposed approach showed improved detection rates and reduced false alarm rates for network attacks.

Zhao et al. (2021) [21] proposed a hybrid approach using the Random Forest (RF) algorithm and the AdaBoost algorithm. The proposed approach showed improved detection rates and reduced false alarm rates for network attacks. Another study by Fan et al. (2021) [22] proposed a hybrid approach using the Gradient Boosting Decision Tree (GBDT) algorithm and rule-based approach for HIDS. The proposed approach showed improved performance in terms of detection rates and reduced false alarm rates. A hybrid approach using the Extreme Gradient Boosting (XGBoost) algorithm and the K-Nearest Neighbors (KNN) algorithm was proposed by Li et al. (2021) [23]. The proposed approach showed improved detection rates and reduced false alarm rates for network attacks.

2.2 OVERVIEW OF SIMILAR WORKS

Table I. Review of existing works

Paper	Methodology	Dataset Used	Performance
[6]	◆ Energy-based flow algorithm (EFC)[6] ◆ Anomaly based classifier ◆ Uses inverse statistics (inverse potts) model	● CIDDS-001 ● CICIDS17 ● CICDDoS19	F1 score (around 97% at best) and AUC (around 99% at best) values obtained using EFC
[7]	◆ A novel deep learning technique for intrusion detection ◆ Proposed Non symmetric deep autoencoder (NDAE)[7] for unsupervised feature learning	● KDD Cup'99 ● NSL-KDD	Model has produced f-score of 87.37%, recall of 85.42% and precision of 100.00%
[8]	◆ Detection and classification of the DNS over HTTPS attacks (DoH)[8]. ◆ Balanced and Stacked Random Forest classifier is used.	CIRA-CIC-DoHBrw-2020 dataset from the Canadian Institute for Cybersecurity (CIC).	Balanced and stacked Random Forest achieved precision of 99.71%, recall of 99.72% and F1 score of 99.71%.
[9]	◆ Proposed a four-stage intrusion detection system[9].	Real vehicle dataset provided by the Hacking and	The proposed methodology has 13.73% better

	◆ Uses the chi-squared method	Countermeasure Research Lab.The dataset includes both attack-free and corrupted data with various kinds of attacks.	accuracy compared to the existing method (ID Sequence Methodology) and it can detect replay attack with 95.24% accuracy, while the existing method could not detect any replay attacks.
[10]	◆ A deep transfer learning-based dependable IDS model[10]. ◆ Proposed Deep transfer learning-based ResNet model	The dataset has been generated from various heterogeneous sources (IOT devices).	Accuracy score of 87%, Precision score of 88%, Recall score of 86%, F1 score of 86%, and ROC AUC score of 83%.
[11]	Proposed the Stacked Ensemble based Intrusion Detection System (SE-IDS)[11] comprises decision tree, XGBoost, bagging classifier, extra tree and random forest as base learners with Multilayer Perceptron (MLP) as meta-learner	● NSL-KDD Dataset ● UNSW-NB15 Dataset	Accuracy score of 88.10%, Precision score of 87% and False alarm rate of 15% on KDD dataset.
[12]	◆ Hybrid optimization based Deep learning technique for the multi-level intrusion detection process[12] ◆ Rider Optimization Algorithm-Based Neural Network (RideNN) is	BOT-IOT Dataset	The proposed intrusion detection algorithm performed with accuracy score of 92.54%, precision score of 83.62%,

	employed for first level detection ◆ Deep Neuro Fuzzy network (DNFN) is utilized for the second level classification process ◆ DNFN classifier is trained through devised Social Squirrel Search Algorithm (SSSA)		and F-measure of 87.18% .
[13]	◆ Proposed a novel Poor and Rich Optimization with Deep Learning Model for Blockchain Enabled Intrusion Detection in CPS Environment, called PRO-DLBIDCPS technique[13]. ◆ AHSA for election of features, ABi-GRNN classifier, and PRO hyperparameter optimizer.	● NSL-KDD 2015 dataset ● CICIDS17 dataset	Accuracy score of 94.15%, Precision score of 93.20% and F1 score of 94%.
[14]	◆ An intelligent IDS is proposed for network attack detection that can be applied to Controller Area Network (CAN) bus[14] of AVs. ◆ Proposed IDS utilizes tree-based ML algorithms including Decision tree, Random Forest, Extra Trees (ET), and Extreme Gradient Boosting (XGBoost).	● CICIDS17 dataset	Accuracy score of 89.35%, Precision score of 93.10% and F1 score of 89.56%.

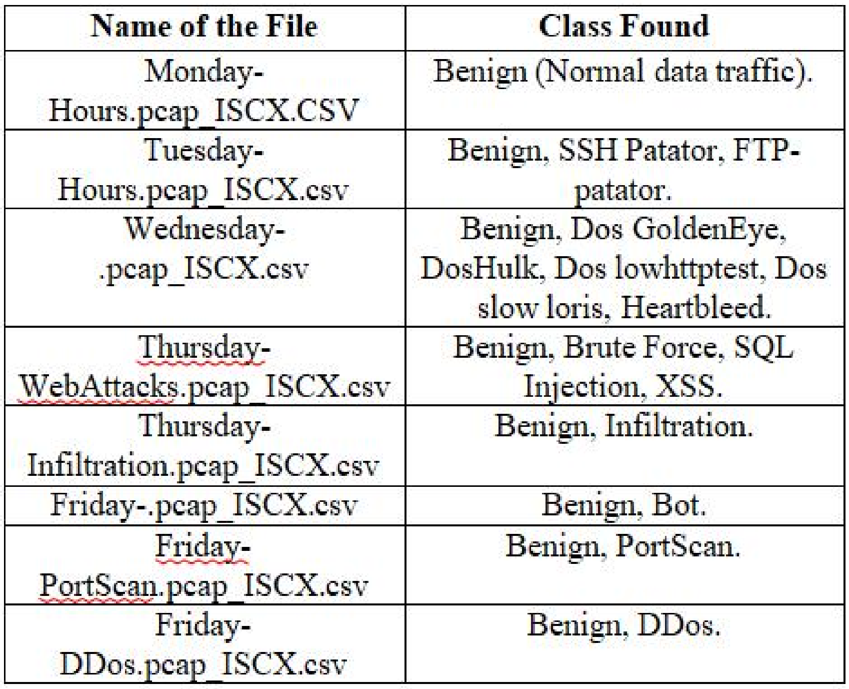
CHAPTER 3

DATASET AND PRE-PROCESSING

3.1 CICIDS-17 DATASET

CICFlowmeter-V3.0 is used to extract 78 features and 79 labels from the CICIDS2017 [24] dataset that closely mimic real-world network data (PCAPs). The abstract characteristic attitudes of 25 users about the HTTP, HTTPS, FTP, SSH, and email protocols are contained in this dataset, as shown in Table I. The data collection takes place throughout a number of time frames. According to the 2016 McAfee Report, the attacks in this dataset are classified as brute force FTP, brute force SSH, DoS, heartbleed, web, infiltration, botnet, and DDoS attacks and are not included in any of the datasets mentioned above. CICIDS2017 employs the B-Profile approach to create an abstract characteristic profiling of human interactions by using the Alpha profile to simulate various multi-stage assault scenarios. The upcoming subsection discusses the some of the above mentioned attacks in detail.

Table II. Attacks present in the respective files



Name of the File	Class Found
Monday-Hours.pcap_ISCX.CSV	Benign (Normal data traffic).
Tuesday-Hours.pcap_ISCX.csv	Benign, SSH Patator, FTP-patator.
Wednesday-.pcap_ISCX.csv	Benign, Dos GoldenEye, DosHulk, Dos lowhttptest, Dos slow loris, Heartbleed.
Thursday-WebAttacks.pcap_ISCX.csv	Benign, Brute Force, SQL Injection, XSS.
Thursday-Infiltration.pcap_ISCX.csv	Benign, Infiltration.
Friday-.pcap_ISCX.csv	Benign, Bot.
Friday-PortScan.pcap_ISCX.csv	Benign, PortScan.
Friday-DDos.pcap_ISCX.csv	Benign, DDos.

3.1.1 SSH Patator Attack

Patator attack is a type of brute-force attack that is used to crack passwords through trial and error. SSH Patator attack specifically targets Secure Shell (SSH) . servers by attempting to authenticate to the server using a list of usernames and passwords. One of the significant risks of the SSH Patator attack is that it can compromise sensitive data, such as login credentials, personal data, and financial information. This attack can also lead to the deployment of malware or unauthorized access to a server.

According to a study by Zhang et al. (2020) [25], SSH Patator attacks have become increasingly prevalent in recent years, and their frequency has shown an upward trend. The authors also suggest that the most effective way to prevent SSH Patator attacks is to use strong passwords, limit login attempts, and employ multi-factor authentication. Another study by Kachwala et al.(2020) [26] recommends using intrusion detection and prevention systems to detect and prevent SSH Patator attacks. The authors suggest that these systems can identify malicious traffic and block unauthorized access attempts, minimizing the damage caused by such attacks. SSH Patator attacks pose a significant threat to the security of SSH servers. Organizations should take necessary measures such as using strong passwords, limiting login attempts, and employing multi-factor authentication to prevent these attacks. Additionally, the use of intrusion detection and prevention systems can help to identify and mitigate SSH Patator attacks.

3.1.2 FTP Patator

FTP Patator attack specifically targets File Transfer Protocol (FTP) servers by attempting to authenticate to the server using a list of usernames and passwords. One of the significant risks of the FTP Patator attack is that it can compromise sensitive data, such as login credentials, personal data, and financial information. This attack can also lead to the deployment of malware or unauthorized access to a server.
Paniagua et al. (2019) [27], mentioned in their work that FTP Patator attacks have been known to be effective in compromising FTP servers. The authors also suggest that the most effective way to prevent FTP Patator attacks is to use strong passwords,

limit login attempts, and employ multi-factor authentication. Another study by Nadeem et al. (2017) [28] recommends using intrusion detection and prevention systems to detect and prevent FTP Patator attacks. The authors suggest that these systems can identify malicious traffic and block unauthorized access attempts, minimizing the damage caused by such attacks. FTP Patator attacks pose a significant threat to the security of FTP servers. Organizations should take necessary measures such as using strong passwords, limiting login attempts, and employing multi-factor authentication to prevent these attacks. Additionally, the use of intrusion detection and prevention systems can help to identify and mitigate FTP Patator attacks.

3.1.3 DOS GOLDENEYE

A GoldenEye attack is a type of Denial of Service (DoS) attack that targets web servers, causing them to be unresponsive to legitimate traffic. This attack typically involves sending a high volume of traffic to the server, overwhelming its resources and causing it to crash or become unresponsive.

Wang et al. (2017) [29], in his study warned that the GoldenEye attack can be highly effective in disrupting web services, causing significant financial losses and reputational damage for affected organizations. The authors suggest that the most effective way to prevent GoldenEye attacks is to use intrusion detection and prevention systems to identify and block malicious traffic. Another study by Taneja et al. (2018) [30] recommends implementing load balancers and content delivery networks to distribute traffic across multiple servers, mitigating the impact of GoldenEye attacks. The authors suggest that these systems can help to ensure that the server resources are not overwhelmed by traffic from the attack. GoldenEye attacks pose a significant threat to the availability and integrity of web services. Organizations should take necessary measures such as using intrusion detection and prevention systems and implementing load balancers and content delivery networks to prevent and mitigate these attacks.

3.1.4 SQL Injection

A SQL injection attack is a type of cyber attack that targets databases by injecting malicious code into SQL statements, which can result in unauthorized access to sensitive data or the manipulation of data within the database.
Halfond et al.(2010) [31], described SQL injection attacks are one of the most common forms of web application attacks, and they can be highly effective in compromising the security of a database. The authors suggest that the most effective way to prevent SQL injection attacks is to use input validation and sanitization techniques to ensure that user input is properly sanitized and validated before being processed by the database. SQL injection attacks pose a significant threat to the security of databases, and organizations should take necessary measures such as input validation and sanitization, and using prepared statements and parameterized queries to prevent these attacks.

3.1.5 DDos

A Distributed Denial of Service (DDoS) attack is a type of cyber attack that aims to disrupt the availability of online services or websites by overwhelming the target server with a high volume of traffic from multiple sources. The goal of a DDoS attack is to make the target server or service unavailable to legitimate users, causing significant financial losses and reputational damage for the affected organization. According to a report by Akamai Technologies (2019) [32], DDoS attacks are becoming more frequent, more sophisticated, and more challenging to mitigate. The report suggests that attackers are using a variety of techniques, including amplification attacks, application layer attacks, and botnet attacks, to launch DDoS attacks. DDoS attacks are a significant threat to the availability of online services and websites. Organizations should take necessary measures to prevent DDoS attacks, such as using DDoS protection services, implementing firewalls and IPS systems, using load balancers and CDNs, and developing a DDoS response plan.

3.1.6 Portscan

A port scan attack is a type of cyber attack that attempts to identify open ports and services on a target system or network. The goal of a port scan attack is to identify vulnerabilities in the target system that can be exploited in subsequent attacks. According to a study by Kephart et al. (1995) [33], port scan attacks are a common technique used by attackers to identify potential targets for further attacks. The authors suggest that port scan attacks are often used as a reconnaissance tool to gather information about the target system, such as the operating system, applications, and services running on the system. Port scan attacks are a common technique used by attackers to identify vulnerabilities in a target system or network. Organizations should take necessary measures to prevent port scan attacks, such as implementing firewalls and IPS systems, using network segmentation, implementing port knocking, and monitoring network traffic.

3.2 PRE-PROCESSING

3.2.1 STANDARD SCALING:

Standard scaling, also referred to as standardization or z-score normalization, is a widely used data pre-processing method that is used to modify numerical data so that it has a mean of zero and a variance of one. This technique is very useful while handling datasets with diverse scales and units [34] . The standardized value Z is calculated by using the following equation (1)

$$Z=\frac{x-\mu}{\sigma} \tag{1}$$

3.2.2 SYNTHETIC MINORITY OVERSAMPLING TECHNIQUE(SMOTE):

Imbalanced datasets are datasets where one class of data is significantly underrepresented compared to the other classes. This can be problematic for machine learning algorithms because they may have a bias towards the majority class and perform poorly on the minority class.
SMOTE works by generating synthetic samples of the minority class [35]. It does this by selecting a minority class sample and then selecting one or more of its nearest neighbors. The algorithm then creates new samples by interpolating between the selected sample and its neighbors.

Algorithm:

Let the minority class be Y for each x∈Y , the k-nearest neighbors of x are obtained by calculating the Euclidean distance (2) between x and every other sample in set Y.

The Euclidean distance d(x,Y):

$$d(x,Y)=\sqrt{\sum_{i=1}^{n}(x-Y_i)^2} \qquad (2)$$

For each x∈Y , N examples are randomly selected from its k-nearest neighbors, and they construct the set Y1.

For each x(k)∈A1:

$$X' = X + \text{rand}(0,1) \text{ X } [(X-X(k))] \qquad (3)$$

where X’ in (3) is the new sample, rand(0,1) is a random number between 0 and 1.

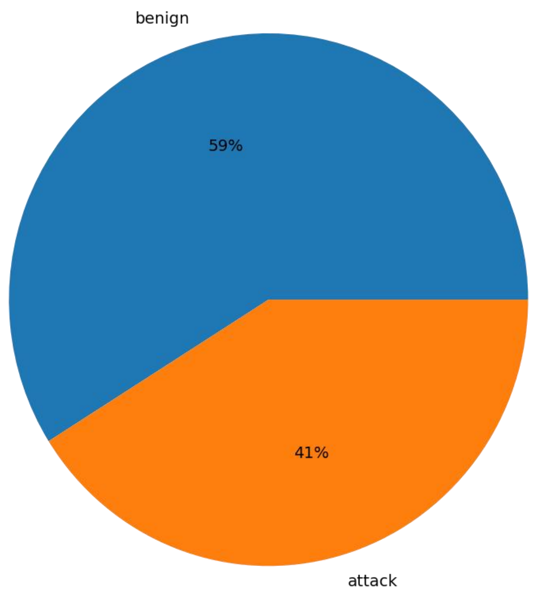


Fig. 1. Dataset Composition Before SMOTE

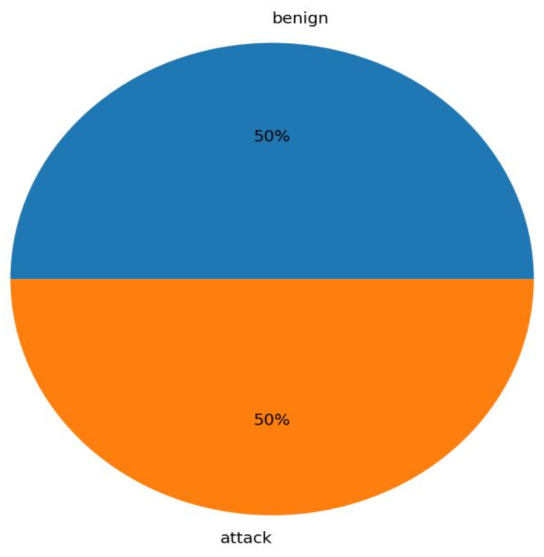


Fig. 2. Dataset Composition after SMOTE

CHAPTER 4

Models and methods

An ML or DL algorithm can be trained by using various procedures, including supervised and unsupervised learning. Supervised learning performs classification based on data instances that are marked in the training phase. Supervised learning algorithms include ANN, DT (both types c4.5, ID3), KNN, NB, and RF. Meanwhile, unlabelled data instances are detected via unsupervised learning, with clustering being the dominant learning method. Unsupervised learning algorithms includes the k-means clustering.

Five supervised ML algorithms are evaluated in ML-AIDS. The basic concepts of these algorithms are described as follows.

4.1 Decision Tree

Decision tree algorithm is a popular supervised learning algorithm that is used for classification and regression tasks. It is a tree-like structure where each node represents a feature and each branch represents a decision rule. The leaves of the tree represent the decision outcome. The algorithm constructs the decision tree by recursively partitioning the data into subsets based on the values of the features. The goal of the algorithm is to create a tree that has the highest possible accuracy while being as simple as possible [36].

The decision tree algorithm uses a measure of impurity to determine the best split at each node. Two commonly used measures of impurity are Gini impurity and entropy.

Gini impurity is a measure of the probability of misclassifying a randomly chosen element in the dataset if it were randomly labeled according to the class distribution in the subset.

The equation for Gini impurity is:

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2 \qquad (4)$$

where c is the number of classes, and pi is the probability of an element belonging to class i.

Entropy, on the other hand, is a measure of the amount of uncertainty or randomness in the data.

The equation for entropy is:

$$E(S)=\sum_{i=1}^{c} -p_i \log(p_i) \tag{5}$$

where c is the number of classes, and pi is the probability of an element belonging to class i.

Once the decision tree is constructed, it can be used to make predictions for new data points. To make a prediction, the algorithm traverses the tree based on the values of the features of the data point and arrives at a leaf node, which represents the predicted class.

In summary, decision tree algorithm is a popular and effective machine learning algorithm for classification and regression tasks. It constructs a tree-like structure that represents decision rules and uses measures of impurity to determine the best splits at each node. The algorithm is easy to interpret and can be used for both numerical and categorical data. Figure 3 represents the common architecture of a decision tree

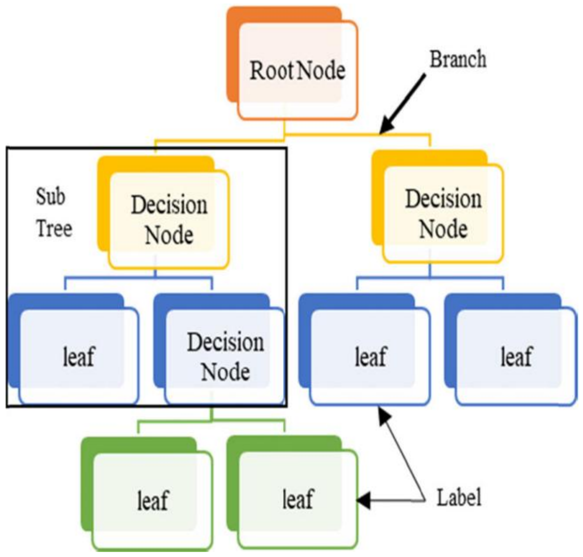


Fig. 3. Decision Tree Architecture

4.2 Random Forest

Random Forest is an ensemble learning method for classification, regression, and other tasks that operate by constructing multiple decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random Forest is a bagging method that uses a subset of the original dataset to make predictions which helps in avoiding overfitting which can occur in Decision Tree method. The algorithm works by constructing multiple decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees [37].

Random Forest algorithm splits the input data into subsets and creates multiple decision trees on each subset of the data. It uses the majority voting method to determine the final prediction for classification and the mean value for regression.

4.3 Naive Bayes

Because of its simplicity , reliability, and often efficient performance in many applications, naive Bayes is widely utilized. The Naive Bayes algorithm is based on the Bayes' theorem, which asserts that the likelihood of the evidence given the hypothesis times the prior probability of the hypothesis determines the probability of a hypothesis (for example, a class label) given certain evidence (for example, a data sample). The class label serves as the hypothesis [38], while the characteristics of a data sample serve as the validation.

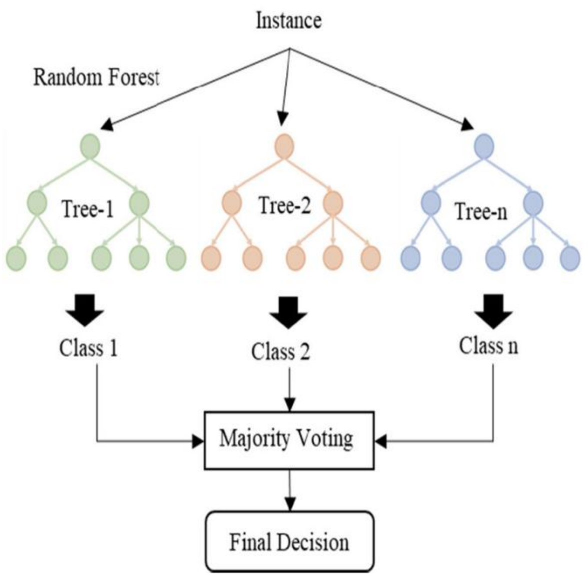


Fig. 4. Random Forest Architecture

The objective is to identify the class label that, given the features, has the highest posterior probability. The likelihood of the features given each class can be described using various probability distributions, such as Gaussian or multinomial distributions, depending on the type of features. The prior probability of each class can be determined from the training data.

The Naive Bayes algorithm calculates the posterior probability of each class given the features and chooses the class with the highest probability to categorise a fresh sample of data. The technique operates in log space and adds the log probabilities rather than multiplying them to prevent numerical underflow.

The following equation is a mathematical representation of the Bayes' theorem:

$$P(y|x)=\frac{P(x|y)\times P(y)}{P(x)} \tag{6}$$

In (6), P(y|x) is the probability of the class label y given the input x, P(x|y) is the probability of the input x given the class label y, P(y) is the prior probability of the class label y and P(x) is the marginal probability of the input x.

4.4 XGBoost

Extreme Gradient Boosting, also known as XGBoost, is a distributed gradient boosting toolkit that has been substantially improved for use in large-scale supervised learning environments. It was created by Tianqi Chen in association with the University of Washington's Distributed Machine Learning Community (DMLC) , and is now an open-source software project that is backed by contributors all around the world. In order to increase the model's accuracy and speed, XGBoost integrates a variety of various strategies. The introduction of a special data structure dubbed a "sparsity-aware split finding" approach [39], which is designed for sparse data sets and can greatly speed up training, is one of the important advancements. In order to avoid overfitting, XGBoost also employs a method known as "regularisation," and it can deal with missing values in training dataset by automatically determining the most effective imputation methodology. The gradient boosting algorithm, which combines weak learners (simple models that perform marginally better than random guessing) to create a stronger model, is extended in the XGBoost algorithm. In order to increase the model's precision and speed, XGBoost expands on this notion by optimising both the training objective and the regularisation term.

The mathematical formulation of the XGBoost algorithm is:

$$y_i = \sum_{k=1}^{K} f_k(x_i) \tag{7}$$

Where, $f_k \in F$.

In equation (7), yi is the the predicted sample output for the ith sample. K is the the number of weak learners (trees) to be combined, fk is the kth model in the tree and F is the set of all possible trees.

With the help of a distributed computing system that enables it to process data in parallel across various workstations, XGBoost also has the potential to scale to very big data sets, which is another significant advantage. For large-scale industrial applications like online advertising and recommendation systems, XGBoost is the best option. LightGBM [40] and CatBoost [41] are two expansions and modifications of XGBoost that have significantly enhanced the efficiency and precision of the original method.

4.5 Proposed Model

The Proposed model presents an implementation of machine learning classifiers for a binary classification task. Fig.5. shows us the detailed diagrammatic representation of the entire architecture. First the dataset undergoes data cleaning process where the rogue data are cleared and makes the dateset ready for the next step that is feature elimination. In the feature elimination process we are able to remove 9 feature which only contained null or infinite values in its entries (*Bwd PSH Flags, Bwd URG Flags, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate*) upon investigating the description of the dataset. Then the dataset is split 70-30 with 70% of the dataset used up for training and 30% is used for testing. The train dataset is then normalized using the standard scalar method and then undergoes the One Hot Encoding process next as mentioned in the previous section. The encoded dateset is then fed as the input to the SMOTE (Synthetic Minority Oversampling Technique) where the train dataset becomes balanced. Next, Random Forest and XGBoost classifiers are added to a list called 'models'. Then, the Random Forest Classifier (set n_estimators to 5) is used to generate new features. This is achieved by making predictions on the training and testing data using the Random Forest Classifier and then concatenating those predictions with the original data as shown in Fig.6. The resulting new data is used to train the XG Boost Classifier (set base score = 0.3, n_estimators = 5 and random state as 42). This technique of using one classifier to generate features for another classifier is known as stacked ensemble learning and has been shown to improve classification accuracy in many cases.

Additionally, the 'genPredRow' function is defined to calculate the number of false positives (FP) in the predictions made by the classifiers. False positives are instances where the classifier incorrectly predicts a positive outcome (i.e. a binary label of 1) when the actual outcome is negative (i.e. a binary label of 0). The function takes the actual values and predicted values as inputs, and returns a Pandas Series containing 1 for each false positive and 0 otherwise. This can be useful in evaluating the performance of the classifiers and identifying areas where they may need improvement.

Finally, the training time for the XG Boost Classifier with Random Forest is noted. This provides an indication of the computational resources required to train the classifiers and may be important for applications where training time is a limiting factor.

Our proposed system demonstrates the use of stacked ensemble learning to improve performance. The 'genPredRow' function provides a useful tool for evaluating classifier performance, and the training time information can be used to optimize computational resources. These techniques may be useful in a variety of research and practical applications where binary classification is required.

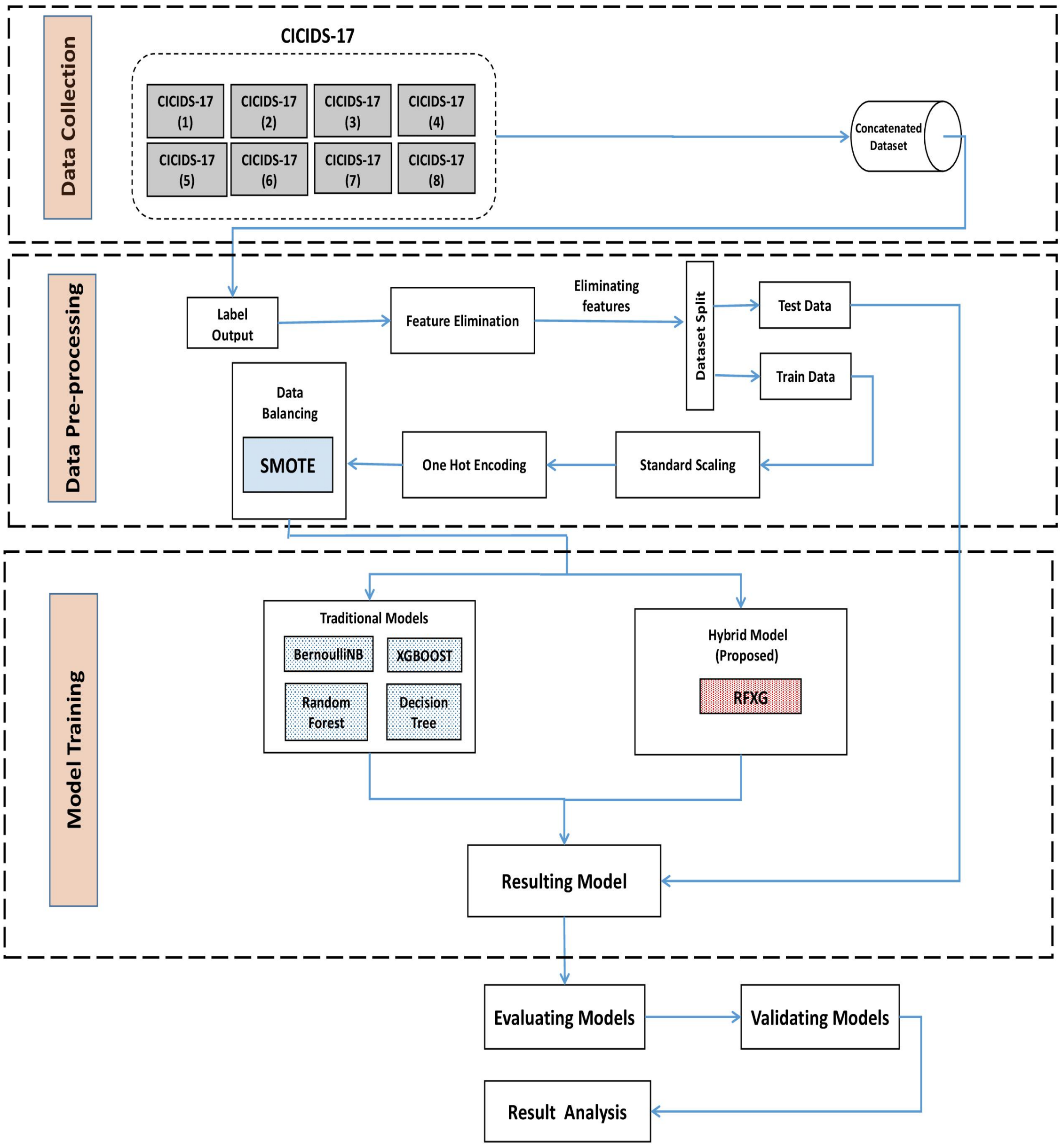


Fig. 5. Detailed diagram of the proposed architecture

CHAPTER 5

Evaluation and Results

5.1 Model Evaluation

Our model was tested and evaluated under the environmental setup given in the Table III below.

Table III. Environmental Setup

Unit	Description
Processor	Intel Core i5 (9th Gen) 3.34 GHz
RAM	8 GB DDR4
OS	Windows 11 Home 22H2

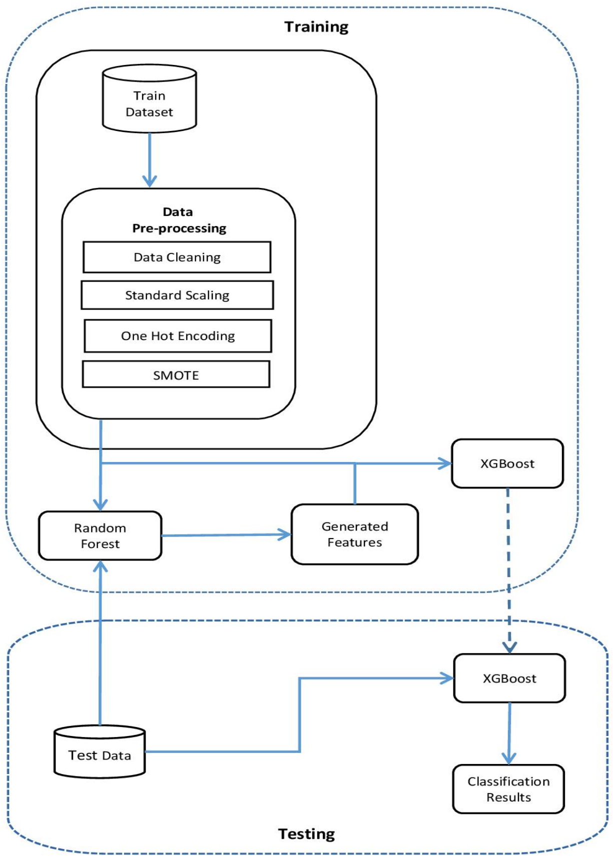


Fig. 6. The RFXG Model

To evaluate our proposed model, we record its performance with standard metrics such as Accuracy, Precision, F1 Score and AUC score. To understand the metrics its necessary to know some basic terms, the terms used are given below:

True Positive (TP): Anomalous traffic correctly classified as attack by the the model

True Negative (TN): Normal traffic correctly classified as normal by the model

False Positive (FP): Also known as the "Type I error", it is the normal traffic incorrectly classified as attack by the model

False Negative (FN): Also known as the "Type II error", it is the anomalous traffic incorrectly classified as normal by the model.

Accuracy is the measure of the samples rightly classified by the model. Higher the accuracy, better is the performance of the model. Accuracy can be calculated by the equation (8) given below:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{8}$$

Recall (True Positive Rate) is the measure of number of malicious traffic correctly classified from the total class of malicious traffic. Higher recall means more attacks are correctly identified by the model. Recall can be calculated by the equation (9) given below

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{9}$$

False Positive Rate (FPR) is the measure of number of normal traffic incorrectly classified as malicious by the model. This causes the false alarms in the Intrusion Detection and lower the FPR, lower is the number of false alarms raised by the IDS. FPR can be calculated by using equation (10) given below:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{10}$$

Precision is the measure of number of malicious traffic classified correctly out of total actual malicious samples present in the dataset. Precision determines the quality of the classification done by the model. Precision can be calculated using equation (11) given below:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{11}$$

To have a better understanding of the models performance we use the F1-score. F1-score uses both precision and recall to validate the performance of the model. F1-score can be calculated by calculating the harmonic mean of precision and recall as shown in equation (12) below:

$$\mathrm{F1} = 2 \times \left(\frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}\right) \tag{12}$$

Table IV shows us the confusion matrix used for the above metrics.

Table IV. Confusion Matrix

Total Population		Predicted Condition	
		Normal	Anomaly
Actual Condition	Normal	TN	FP
	Anomaly	FN	TP

Finally, to summarize the models performance, the Area Under the Curve (AUC) is calculated. AUC is the total area under the Receiver Operating Characteristics (ROC) curve which is plotted with the true positive rate on the x-axis and the false positive rate on the y-axis. The AUC score determines the probability of the model correctly identifying a malicious sample. The greater the score, greater is the overall performance of the model. The AUC can be calculated by using the equation (13) given below:

$$\mathrm{AUC}_{\mathrm{ROC}} = \int_0^1 \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \, \mathrm{d} \frac{\mathrm{FP}}{\mathrm{TN}+\mathrm{FP}} \tag{13}$$

5.2 Model Validation and Analysis

The CICIDS17 dataset is split 70-30, 70% of the dataset is used up for training and the remaining 30% of the dataset is used for testing. The proposed RFXG model is trained and tested alongside some classic ML models such as Naive Bayes, Decision Tree, Random Forest and XGBoost. Fig.6. shows the comparison of accuracy obtained from each classifiers. From the figure we can see that the proposed model RFXG has the superior accuracy score of 99.90% closely followed by the Random Forest and XGBoost models with an accuracy score of 99.85% and 99.79% respectively.

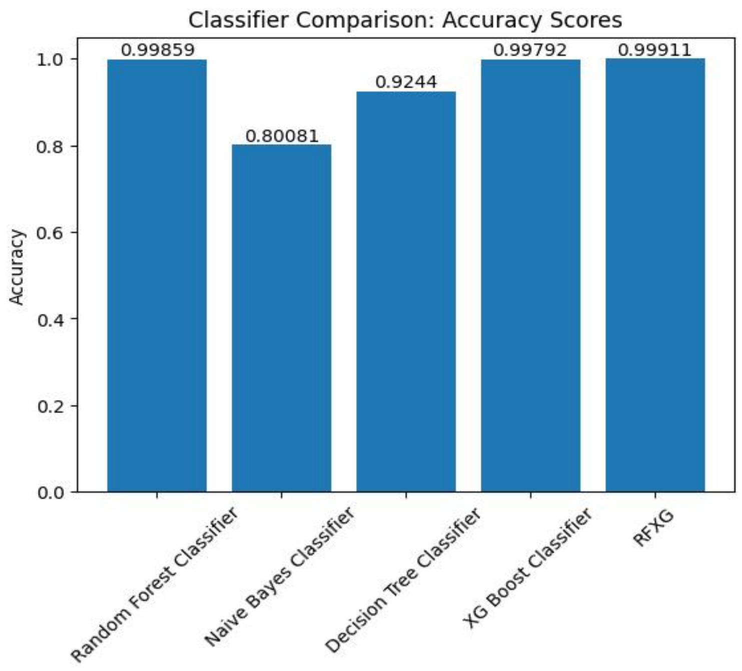


Fig. 7. Comparison of accuracy scores

Fig.7. shows us a comparison of the precision scores achieved by each models. As we can see in the figure our proposed RFXG has produced an excellent precision score of 99.90% but surprisingly is slight outperformed by the Random Forest model with a precision score of 99.92% and followed by XGboost model with a precision score of 99.86%.

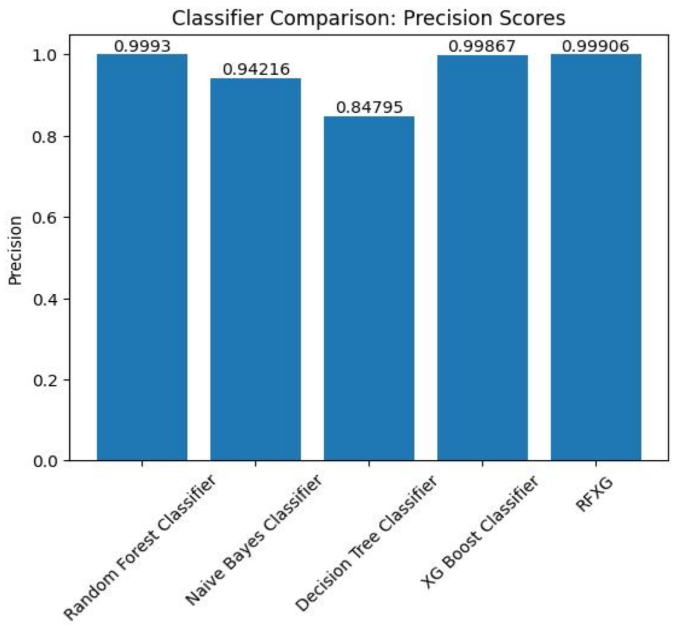


Fig. 8. Comparison of Precision scores

Fig.8. shows us the comparison of F1 scores obtained by each models.

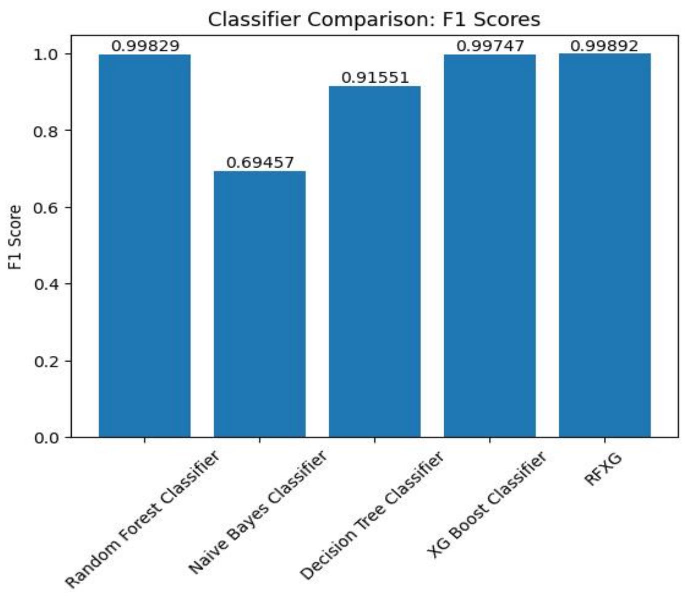


Fig. 9.   Comparison of F1 scores

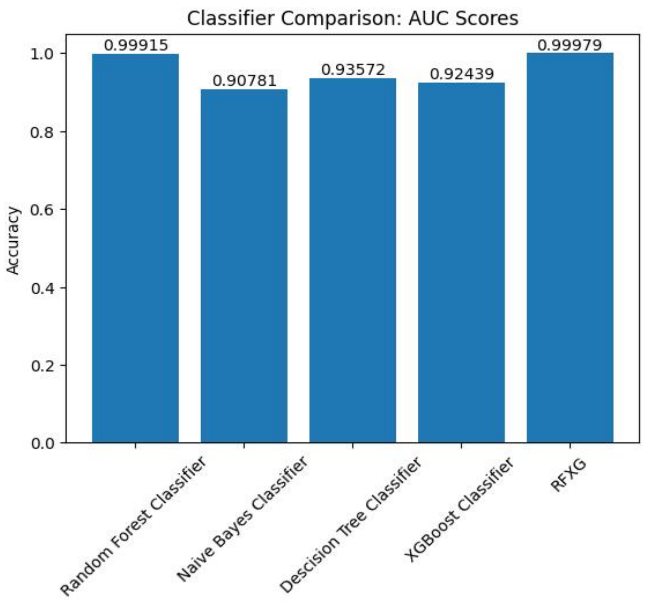
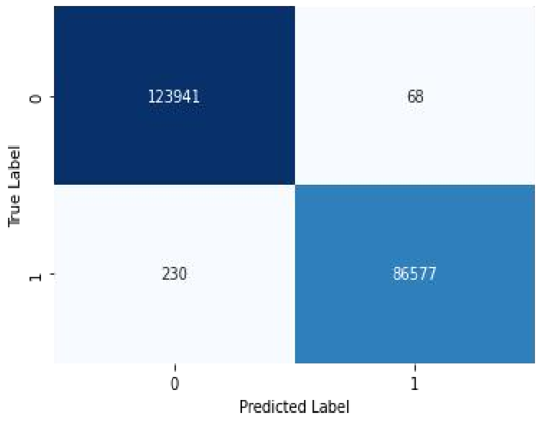
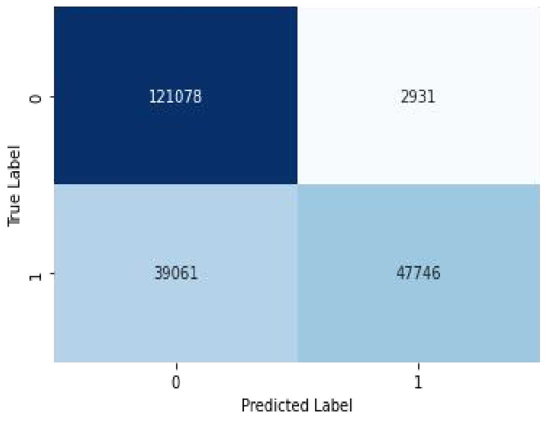


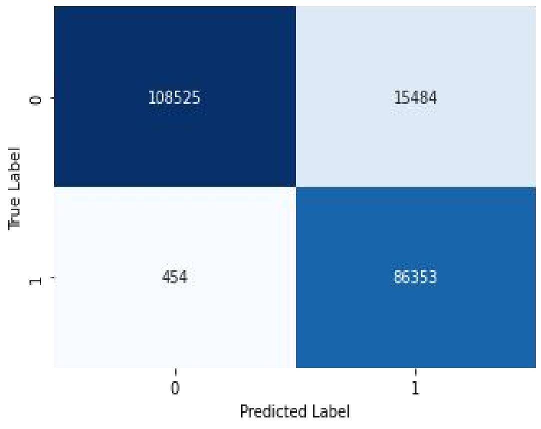
Fig. 10. Comparison of AUC scores



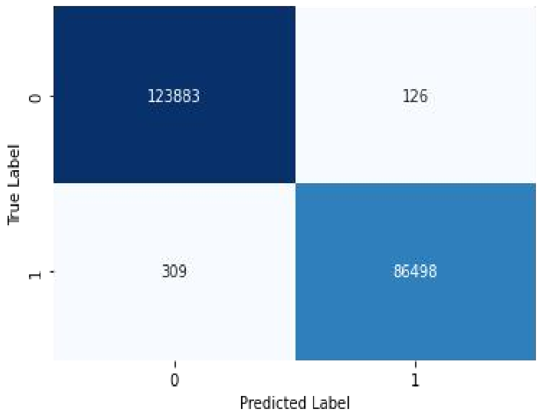
(a) Random Forest



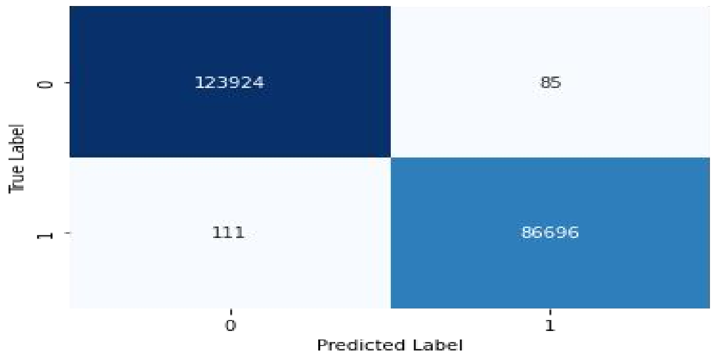
(b) Naive Bayes



(c) Decision Tree



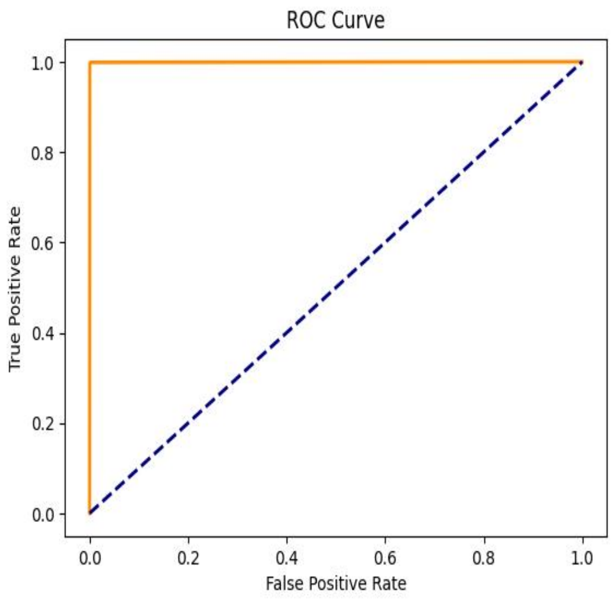
(d) XGBoost



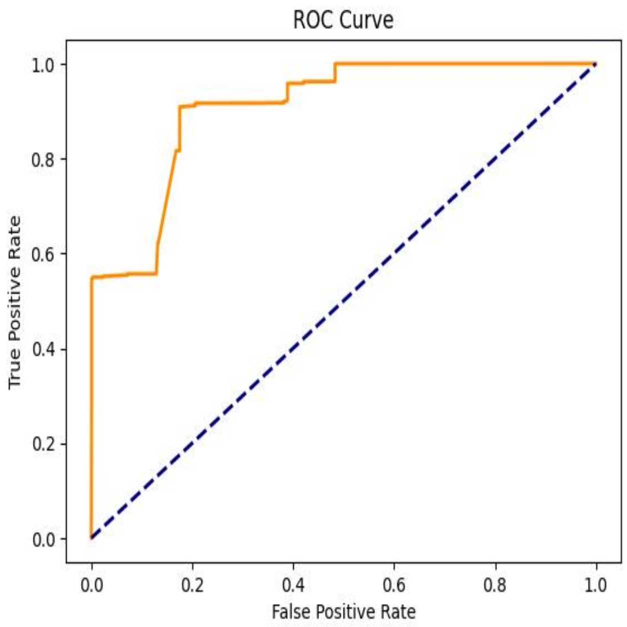
(e) RFXG

Fig. 11. Confusion Matrix produced by the models

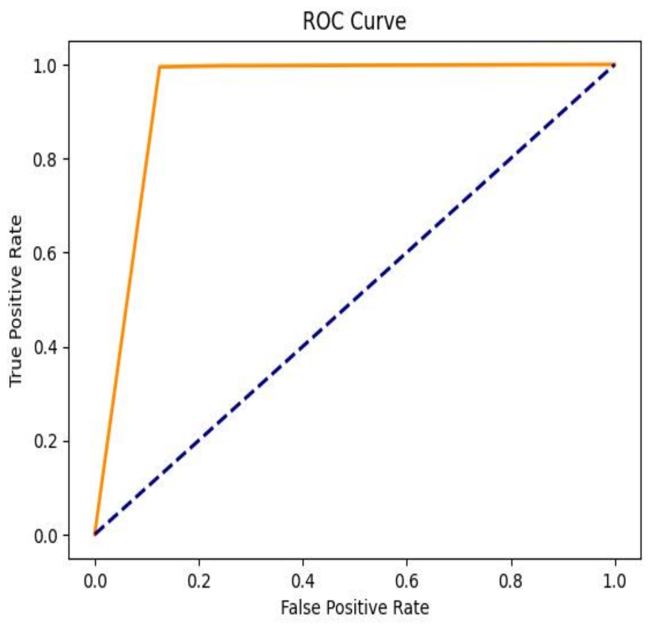
Once again from fig.8 we can clearly see that our proposed RFXG model outperforms other models with a F1 score of 99.89% followed by Random Forest and XGBoost with F1 scores of 99.82% and 99.74% respectively. Fig.9 represents the comparison of the AUC scores of each model. This should conclude the question of the best performing model as our proposed RFXG model once again outperforms other tested models with an AUC score of 99.98% followed by the Random Forest model with an AUC score of 99.90%.



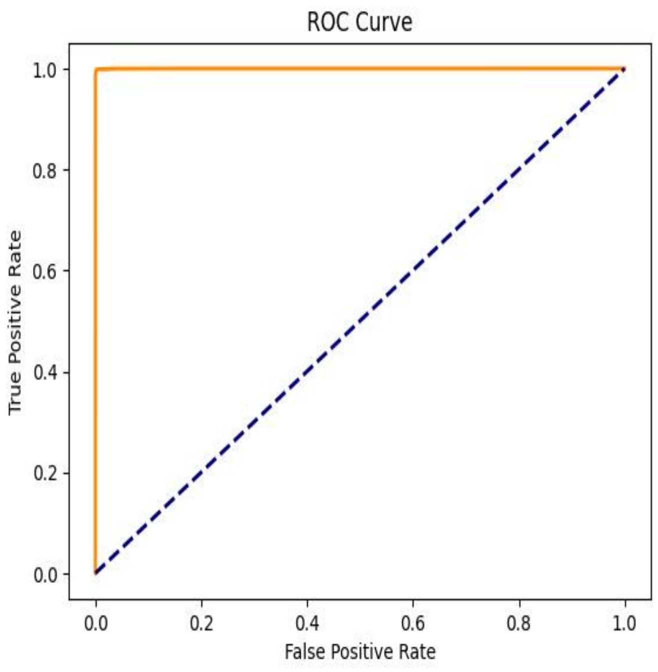
(a) ROC curve of Random Forest Model



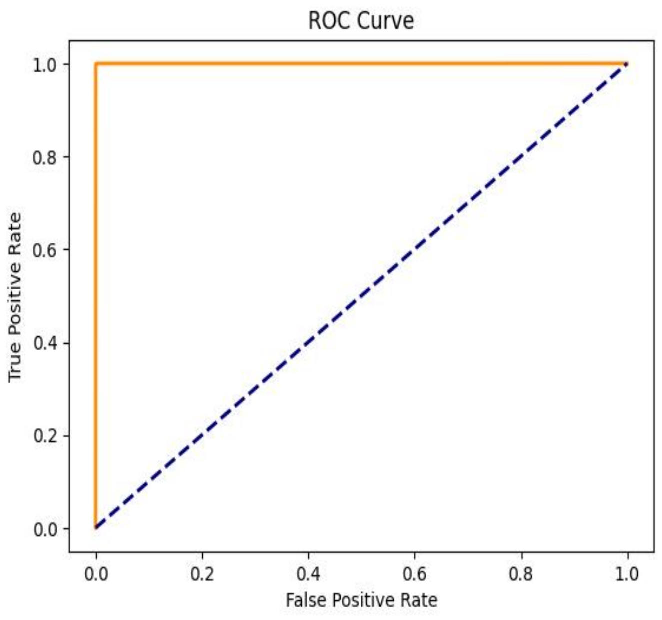
(b) ROC Curve of Naive Bayes Model



(c) ROC Curve of Decision Tree Model



(d) ROC curve of XGBoost Model



(e) ROC curve of RFXG Model

Fig. 12. ROC curves of the tested models

Fig.11 shows the ROC curves obtained from the respective models. The curve obtained by the proposed system is almost the ideal curve obtainable by any model. By using these metrics we can showcase the RFXG model's superior performance.

5.3 Comparison of the results

Table V shows us the overall comparison of the scores achieved by the papers reviewed for our work. We can comprehend from the table that our proposed model has outperformed other previous works in various metrics. It is remarkable what our proposed system RFXG has achieved considering that it is much less in complexity and a hybrid of machine learning algorithms.

Table V. Comparison of the scores obtained with some reviewed papers

Paper	**Accuracy**	**Precision**	**Recall**	**F1**	**AUC**	**False alarm rate**
[1]	-	-	-	97%	≈99%	-
[2]	-	100%	85.42%	87.37%	-	-
[3]	-	99.71%	99.72%	99.71%	-	-
[5]	95.24%	-	-	-	-	-
[6]	87%	88%	86%	86%	83%	-
[8]	88.10%	87%	-	-	-	15%
[9]	92.54%	83.62%		87.18%	-	-
[10]	94.15%	93.20%	-	94%	-	-
[13]	89.35%	93.10%	-	89.56%	-	-
RFXG (Proposed)	99.91%	99.90%	99.87%	99.89%	99.97%	0.07%

CHAPTER 6

Conclusion and Future Work

In this study, we demonstrate how various attacks can be detected using an intrusion detection system with the help of a hybrid of machine learning techniques. Our proposed hybrid model RFXG consists of Random Forest and XGBoost. The pre-processed dataset is first fed to the Random Forest classifier where it generates features which are added to the dataset. These features are the input for the XGBoost Classifier which predicts the final outcome. This hybrid approach proved to be very efficient in detecting various types of attacks and much less in complexity. Our proposed model could be very much capable of defending an organization or user against various attacks such as DDos, Portscan, etc. Our proposed model when trained and tested with CICIDS17 dataset demonstrated a very high performance with results evaluated as much as 99.91% accuracy, 99.90% precision, 99.87% recall value, 99,89% F1 score, 99.97% AUC score and an exceptionally low false alarm rate of 0.07% which clearly outperformed many other previous works.

We can further study the model's performance in a real network traffic and evaluate its adaptability. Latest attack samples can be use for training the model using the proposed architecture to reinforce from the attacks and vulnerabilities possible in the near future. More advanced feature engineering techniques, such as feature scaling or feature transformation, could be implemented to see if they can improve the accuracy of the model.

APPENDIX I - SAMPLE CODE

```
"""## Exploring the dataset"""

# Commented out IPython magic to ensure Python compatibility.
from google.colab import drive
drive.mount('/content/gdrive')
# %cd /gdrive

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
nRowsRead = None

df1                                                                                =
pd.read_csv("/content/gdrive/MyDrive/Datasets/CICIDS17/MachineLearningCVE/Fri
day-WorkingHours-Afternoon-DDos.pcap_ISCX.csv")
df2=pd.read_csv("/content/gdrive/MyDrive/Datasets/CICIDS17/MachineLearningCV
E/Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv")
df3=pd.read_csv("/content/gdrive/MyDrive/Datasets/CICIDS17/MachineLearningCV
E/Friday-WorkingHours-Morning.pcap_ISCX.csv")
df4=pd.read_csv("/content/gdrive/MyDrive/Datasets/CICIDS17/MachineLearningCV
E/Monday-WorkingHours.pcap_ISCX.csv")
df5=pd.read_csv("/content/gdrive/MyDrive/Datasets/CICIDS17/MachineLearningCV
E/Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv")
df6=pd.read_csv("/content/gdrive/MyDrive/Datasets/CICIDS17/MachineLearningCV
E/Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv")
df7=pd.read_csv("/content/gdrive/MyDrive/Datasets/CICIDS17/MachineLearningCV
E/Tuesday-WorkingHours.pcap_ISCX.csv")
df8=pd.read_csv("/content/gdrive/MyDrive/Datasets/CICIDS17/MachineLearningCV
E/Wednesday-workingHours.pcap_ISCX.csv")


data_train = pd.concat([df1,df2])
del df1,df2
```

```
data_train = pd.concat([data_train,df3])
del df3
df = pd.concat([data_train,df4])
del df4
df = pd.concat([data_train,df5])
del df5
df = pd.concat([data_train,df6])
del df6
df = pd.concat([data_train,df7])
del df7
df = pd.concat([data_train,df8])
del df8

nRow, nCol = data_train.shape
print(f' table has{nRow} rows and {nCol} columns')

data_train.head()

# Packet Attack Distribution
data_train[' Label'].value_counts()

data_train.info()

data_train.describe().style.background_gradient(cmap='Blues').set_properties(**{'fon
t-family':'Segoe UI'})

"""## Preprocessing the data"""

data_train.columns = data_train.columns.str.strip()

df=data_train.copy()

df.loc[data_train['Label'] == "BENIGN" , "Label"] = "benign"
df.loc[data_train['Label'] != 'BENIGN' , "Label"] = "attack"
df.head()

def pie_plot(df, col):
    df[col].value_counts().plot(kind='pie', figsize=(15, 15), fontsize=20,
autopct='%1.0f%%')
    plt.show()

pie_plot(df,'Label')

for i in df.columns:
    df = df.replace([np.inf, -np.inf], np.nan).dropna(axis=0)
df[['Flow Bytes/s', 'Flow Packets/s']] = df[['Flow Bytes/s', 'Flow
Packets/s']].apply(pd.to_numeric)

print(df['Bwd PSH Flags'].value_counts())
print(df['Bwd URG Flags'].value_counts())
print(df['Fwd Avg Bytes/Bulk'].value_counts())
print(df['Fwd Avg Packets/Bulk'].value_counts())
print(df['Fwd Avg Bulk Rate'].value_counts())
print(df['Bwd Avg Bytes/Bulk'].value_counts())
print(df['Bwd Avg Packets/Bulk'].value_counts())
print(df['Bwd Avg Bulk Rate'].value_counts())

df.drop(['Bwd PSH Flags'], axis=1, inplace=True)
df.drop(['Bwd URG Flags'], axis=1, inplace=True)
df.drop(['Fwd Avg Bytes/Bulk'], axis=1, inplace=True)
df.drop(['Fwd Avg Packets/Bulk'], axis=1, inplace=True)
df.drop(['Fwd Avg Bulk Rate'], axis=1, inplace=True)
df.drop(['Bwd Avg Bytes/Bulk'], axis=1, inplace=True)
df.drop(['Bwd Avg Packets/Bulk'], axis=1, inplace=True)
df.drop(['Bwd Avg Bulk Rate'], axis=1, inplace=True)

df.info()
df.head()

from sklearn.model_selection import train_test_split
train, test=train_test_split(df,test_size=0.3, random_state=10)

#Exploratory Analysis
# Descriptive statistics
train.describe()

# Packet Attack Distribution
train['Label'].value_counts()

test['Label'].value_counts()

from sklearn.preprocessing import StandardScaler
#Standard scaling
scaler = StandardScaler()
cols = train.select_dtypes(include=['float64','int64']).columns
sc_train = scaler.fit_transform(train.select_dtypes(include=['float64','int64']))
sc_test = scaler.fit_transform(test.select_dtypes(include=['float64','int64']))
sc_traindf = pd.DataFrame(train, columns = cols)
sc_testdf = pd.DataFrame(test, columns = cols)

from sklearn.preprocessing import OneHotEncoder

# creating one hot encoder object
onehotencoder = OneHotEncoder()

trainDep = train['Label'].values.reshape(-1,1)
trainDep = onehotencoder.fit_transform(trainDep).toarray()
testDep = test['Label'].values.reshape(-1,1)
testDep = onehotencoder.fit_transform(testDep).toarray()

train_X=sc_traindf
```

```
train_y=trainDep[:,0]

test_X=sc_testdf
test_y=testDep[:,0]

from imblearn.over_sampling import SMOTE
over_sampler = SMOTE(k_neighbors=2)
X_res, y_res = over_sampler.fit_resample(train_X, train_y)

from collections import Counter
print(f"Training target statistics: {Counter(y_res)}")
print(f"Testing target statistics: {Counter(test_y)}")



"""**Fitting Models**"""

from sklearn.svm import SVC
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import BernoulliNB
from xgboost import XGBClassifier
import imblearn
import time



# Train Gaussian Naive Baye Model
t1=time.time()
BNB_Classifier = BernoulliNB()
BNB_Classifier.fit(X_res, y_res)
t2=time.time()
print("Training time for BernoulliNB: ", t2-t1)
print()
```

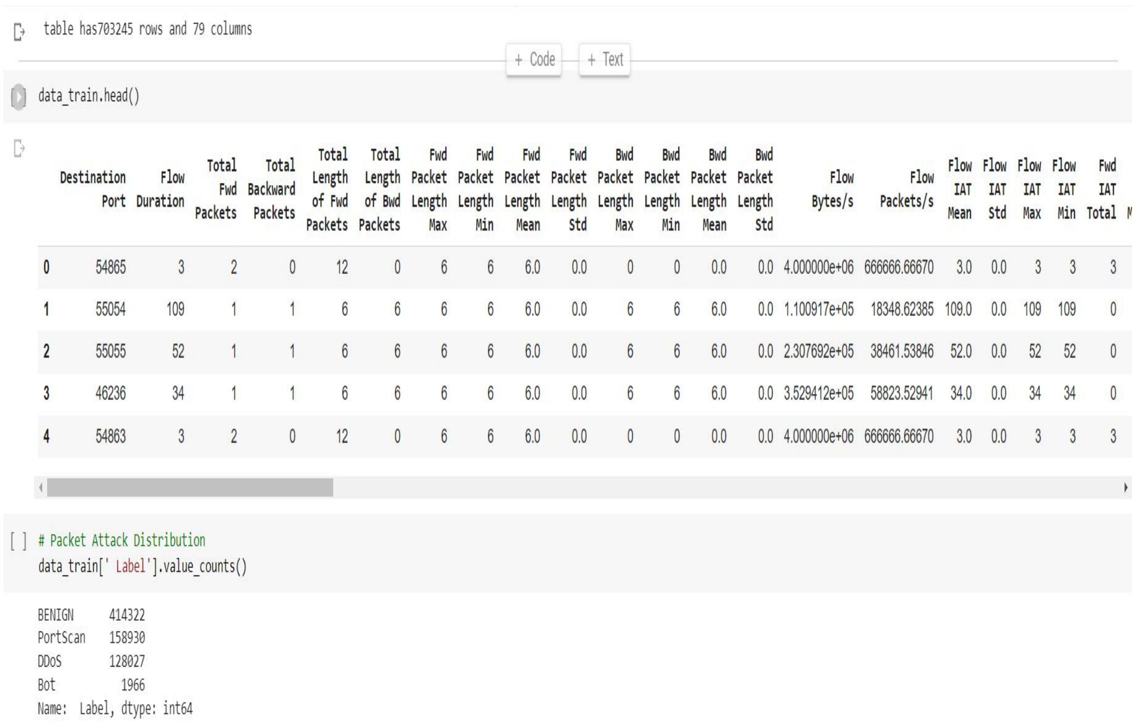
```
# Train Decision Tree Model
t1=time.time()
DTC_Classifier = tree.DecisionTreeClassifier(criterion='gini',max_depth=2)
DTC_Classifier.fit(X_res, y_res)
t2=time.time()
print("Training time for Decision Tree: ", t2-t1)
print()

# Train XGBoost Model
t1=time.time()
XGB_Classifier = XGBClassifier(base_score=0.3, n_estimators=42)
XGB_Classifier.fit(X_res, y_res)
t2=time.time()
print("Training time for XGBoost: ", t2-t1)
print()

# Train Random Forest Model
t1=time.time()
RandomForest_Classifier = RandomForestClassifier(n_estimators=2)
RandomForest_Classifier.fit(X_res, y_res)
t2=time.time()
print("Training time for Random Forest: ", t2-t1)
print()

"""**Evaluating Models**"""

from sklearn import metrics
from sklearn.metrics import f1_score



models = []
models.append(('Random Forest Classifier', RandomForest_Classifier))
models.append(('Naive Bayes Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
models.append(('XG Boost Classifier', XGB_Classifier))



for i, v in models:
    scores = cross_val_score(v, test_X, test_y, cv=10)
    accuracy = metrics.accuracy_score(test_y, v.predict(test_X))
    confusion_matrix = metrics.confusion_matrix(test_y, v.predict(test_X))
    classification = metrics.classification_report(test_y, v.predict(test_X), digits=8)
    print()
    print('==============================          {}          Model          Evaluation
==============================='.format(i))
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean())
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()

def genPredRow(y_actual, y_pred):
    FP = []

    for i in range(len(y_pred)):
        if y_pred[i]=='0.' and y_actual.iat[i]!=y_pred[i]:
          FP.append(1)
        elif y_pred[i]=='1.' and y_actual.iat[i]!=y_pred[i]:
          FP.append(1)
        else:
          FP.append(0)
    return (pd.Series(FP))
```

```
# Define the classifiers

t1=time.time()
# Add the classifiers to a list
models = []
models.append(('XG Boost Classifier', XGB_Classifier))
models.append(('Random Forest Classifier', RandomForest_Classifier))



# Use Random Forest to generate new features
X_test_XGB_RF = test_X
Y_test_XGB_RF = test_y
X_train_XGB_RF = X_res
Y_train_XGB_RF = y_res
finalPred= genPredRow(Y_train_XGB_RF, models[1][1].predict(X_train_XGB_RF))
print("Size of number of FP:", finalPred.size)
finalPredTest = genPredRow(Y_test_XGB_RF,
models[1][1].predict(X_test_XGB_RF))

X_train_XGB_RF = (finalPred.to_numpy()).reshape(-1,1)
X_test_XGB_RF = (finalPredTest.to_numpy()).reshape(-1,1)
print(X_train_XGB_RF.shape)
print(X_test_XGB_RF.shape)
print(X_res.shape)

# Train XGBoost on the new features
XGB_Classifier_RF = XGBClassifier(base_score=0.3, n_estimators=5,
random_state=42)
XGB_Classifier_RF.fit(X_res, y_res)

t2=time.time()
print("Training Time for Random Forest with XG BOOST: ", t2-t1)
```

```
t1=time.time()
Y_test_XGB_RF_pred = XGB_Classifier_RF.predict(test_X)
t2=time.time()
print ("Prediction Time for Random Forest with XG BOOST: ", t2-t1)
print()
scores_XGB_RF = cross_val_score(XGB_Classifier_RF, test_X,
Y_test_XGB_RF_pred, cv=10)
accuracy_XGB_RF = metrics.accuracy_score(test_y, Y_test_XGB_RF_pred)
confusion_matrix_XGB_RF =
metrics.confusion_matrix(test_y,Y_test_XGB_RF_pred)
classification_XGB_RF = metrics.classification_report(test_y,Y_test_XGB_RF_pred,
digits=8)
print()
print('============================== {} Model Evaluation
=============================='.format('Random Forest + XG Boost'))
print()
print ("Cross Validation Mean Score:" "\n", scores_XGB_RF.mean())
print()
print ("Model Accuracy:" "\n", accuracy_XGB_RF)
print()
print("Confusion matrix:" "\n", confusion_matrix_XGB_RF)
print()
print("Classification report:" "\n", classification_XGB_RF)
print()

import matplotlib.pyplot as plt
from sklearn import metrics

models = []
models.append(('Random Forest Classifier', RandomForest_Classifier))
models.append(('Naive Bayes Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
models.append(('XG Boost Classifier', XGB_Classifier_RF))
models.append(('RFXG', XGB_Classifier))

# Define lists to store scores for each model
accuracy_scores = []
precision_scores = []
f1_scores = []

for i, v in models:
    # Calculate scores for each model
    accuracy = round(metrics.accuracy_score(test_y, v.predict(test_X)), 5)
    precision = round(metrics.precision_score(test_y, v.predict(test_X)), 5)
    f1 = round(metrics.f1_score(test_y, v.predict(test_X)), 5)

    # Append scores to the lists
    accuracy_scores.append(accuracy)
    precision_scores.append(precision)
    f1_scores.append(f1)

# Create bar chart for accuracy scores
fig, ax = plt.subplots()
rects = ax.bar(np.arange(len(models)), accuracy_scores)
ax.set_xticks(np.arange(len(models)))
ax.set_xticklabels([model[0] for model in models], rotation=45)
ax.set_ylabel('Accuracy')
ax.set_title('Classifier Comparison: Accuracy Scores')

for rect, score in zip(rects, accuracy_scores):
  height = rect.get_height()
  ax.text(rect.get_x() + rect.get_width() / 2, height, score, ha='center', va='bottom')

# Create bar chart for precision scores
fig, ax = plt.subplots()
rects = ax.bar(np.arange(len(models)), precision_scores)
ax.set_xticks(np.arange(len(models)))
ax.set_xticklabels([model[0] for model in models], rotation=45)
ax.set_ylabel('Precision')
ax.set_title('Classifier Comparison: Precision Scores')

for rect, score in zip(rects, precision_scores):
  height = rect.get_height()
  ax.text(rect.get_x() + rect.get_width() / 2, height, score, ha='center', va='bottom')

# Create bar chart for f1 scores
fig, ax = plt.subplots()
rects = ax.bar(np.arange(len(models)), f1_scores)
ax.set_xticks(np.arange(len(models)))
ax.set_xticklabels([model[0] for model in models], rotation=45)
ax.set_ylabel('F1 Score')
ax.set_title('Classifier Comparison: F1 Scores')

for rect, score in zip(rects, f1_scores):
  height = rect.get_height()
  ax.text(rect.get_x() + rect.get_width() / 2, height, score, ha='center', va='bottom')

plt.show()

from sklearn.metrics import roc_curve, roc_auc_score
for i, v in models:
    t1 = time.time()
    accuracy = metrics.accuracy_score(test_y, v.predict(test_X))
    t2 = time.time()
    classification = metrics.classification_report(test_y, v.predict(test_X))
    print()
    print('========================= {} Model Test Results
========================='.format(i))

    # Print confusion matrix
    cm = metrics.confusion_matrix(test_y, v.predict(test_X))
```

```
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
print()

# Calculate and print ROC AUC score
y_pred_prob = v.predict_proba(test_X)[:,1] # corrected v instead of model
auc = roc_auc_score(test_y, y_pred_prob)
print("ROC AUC Score:", auc)

# Plot ROC Curve
fpr, tpr, thresholds = roc_curve(test_y, y_pred_prob)
plt.plot(fpr, tpr, color="darkorange", lw=2)
plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
print()

print("Prediction time of ", i, t2-t1)
print()
print("Model Accuracy:\n", accuracy)
print()
print("Classification report:\n", classification)
print()
```

APPENDIX II - SAMPLE OUTPUTS



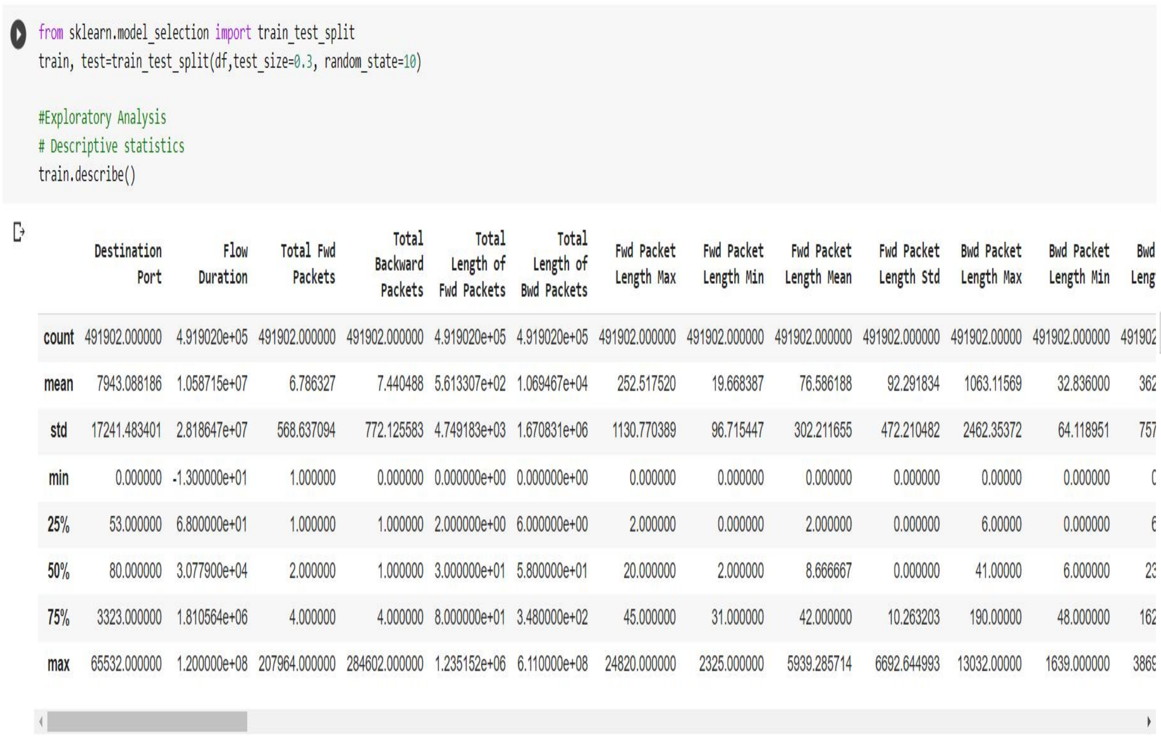
```
table has703245 rows and 79 columns
```

```
data_train.head()
```

	Destination Port	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
0	54865	3	2	0	12	0	6	6	6.0	0.0	0	0	0.0	0.0	4.000000e+06	666666.66670	3.0	0.0	3	3	3
1	55054	109	1	1	6	6	6	6	6.0	0.0	6	6	6.0	0.0	1.100917e+05	18348.62385	109.0	0.0	109	109	0
2	55055	52	1	1	6	6	6	6	6.0	0.0	6	6	6.0	0.0	2.307692e+05	38461.53846	52.0	0.0	52	52	0
3	46236	34	1	1	6	6	6	6	6.0	0.0	6	6	6.0	0.0	3.529412e+05	58823.52941	34.0	0.0	34	34	0
4	54863	3	2	0	12	0	6	6	6.0	0.0	0	0	0.0	0.0	4.000000e+06	666666.66670	3.0	0.0	3	3	3

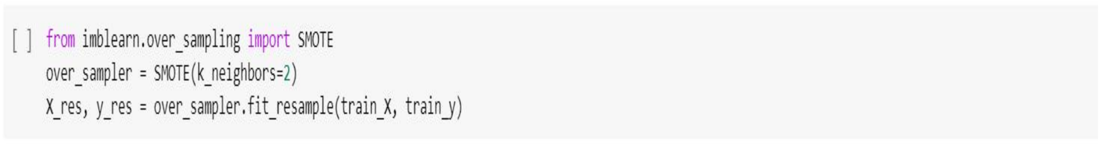
```
# Packet Attack Distribution
data_train[' Label'].value_counts()
```

```
BENIGN      414322
PortScan    158930
DDoS        128027
Bot           1966
Name:  Label, dtype: int64
```

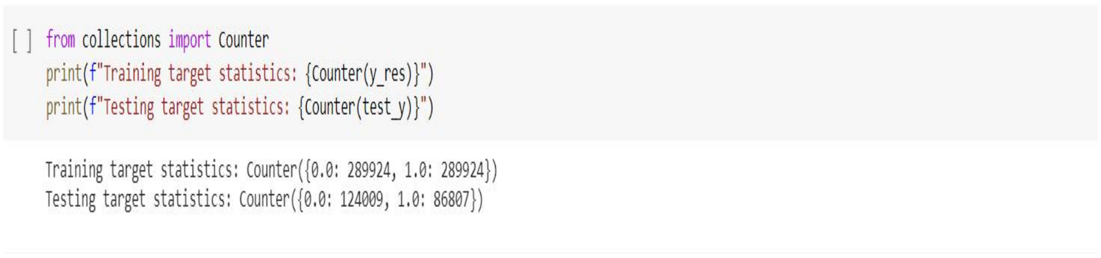


```
from sklearn.model_selection import train_test_split
train, test=train_test_split(df,test_size=0.3, random_state=10)

#Exploratory Analysis
# Descriptive statistics
train.describe()
```

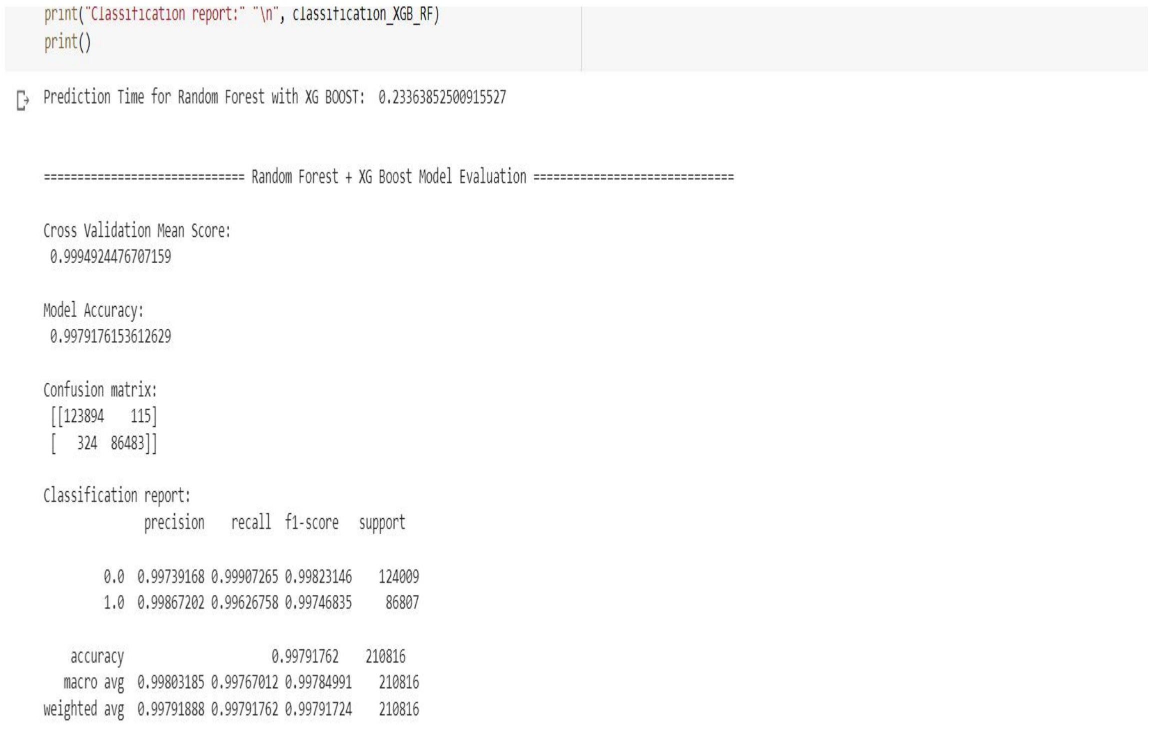
	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	Bwd Packet Length Max	Bwd Packet Length Min	Bwd Leng
count	491902.000000	4.919020e+05	491902.000000	491902.000000	4.919020e+05	4.919020e+05	491902.000000	491902.000000	491902.000000	491902.000000	491902.00000	491902.000000	491902
mean	7943.088186	1.058715e+07	6.786327	7.440488	5.613307e+02	1.069467e+04	252.517520	19.668387	76.586188	92.291834	1063.11569	32.836000	362
std	17241.483401	2.818647e+07	568.637094	772.125583	4.749183e+03	1.670831e+06	1130.770389	96.715447	302.211655	472.210482	2462.35372	64.118951	757
min	0.000000	-1.300000e+01	1.000000	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	(
25%	53.000000	6.800000e+01	1.000000	1.000000	2.000000e+00	6.000000e+00	2.000000	0.000000	2.000000	0.000000	6.00000	0.000000	6
50%	80.000000	3.077900e+04	2.000000	1.000000	3.000000e+01	5.800000e+01	20.000000	2.000000	8.666667	0.000000	41.00000	6.000000	23
75%	3323.000000	1.810564e+06	4.000000	4.000000	8.000000e+01	3.480000e+02	45.000000	31.000000	42.000000	10.263203	190.00000	48.000000	162
max	65532.000000	1.200000e+08	207964.000000	284602.000000	1.235152e+06	6.110000e+08	24820.000000	2325.000000	5939.285714	6692.644993	13032.00000	1639.000000	3869



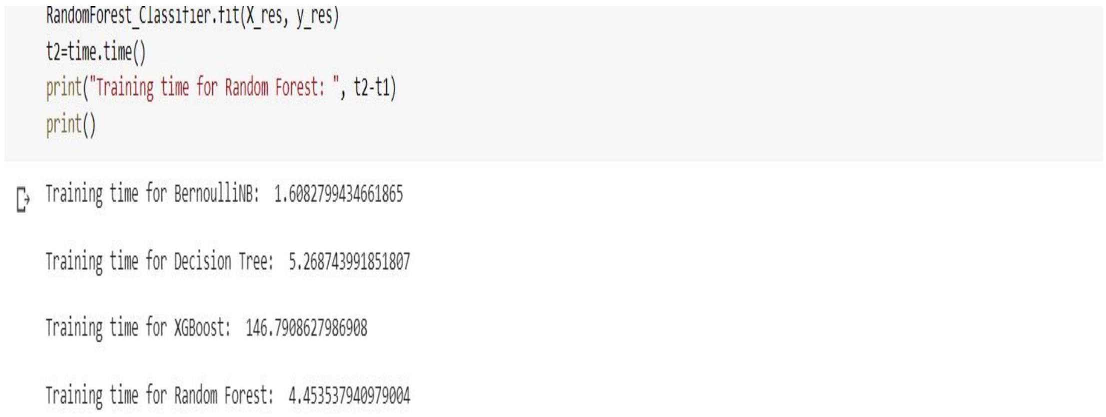
```
[ ] from imblearn.over_sampling import SMOTE
    over_sampler = SMOTE(k_neighbors=2)
    X_res, y_res = over_sampler.fit_resample(train_X, train_y)
```



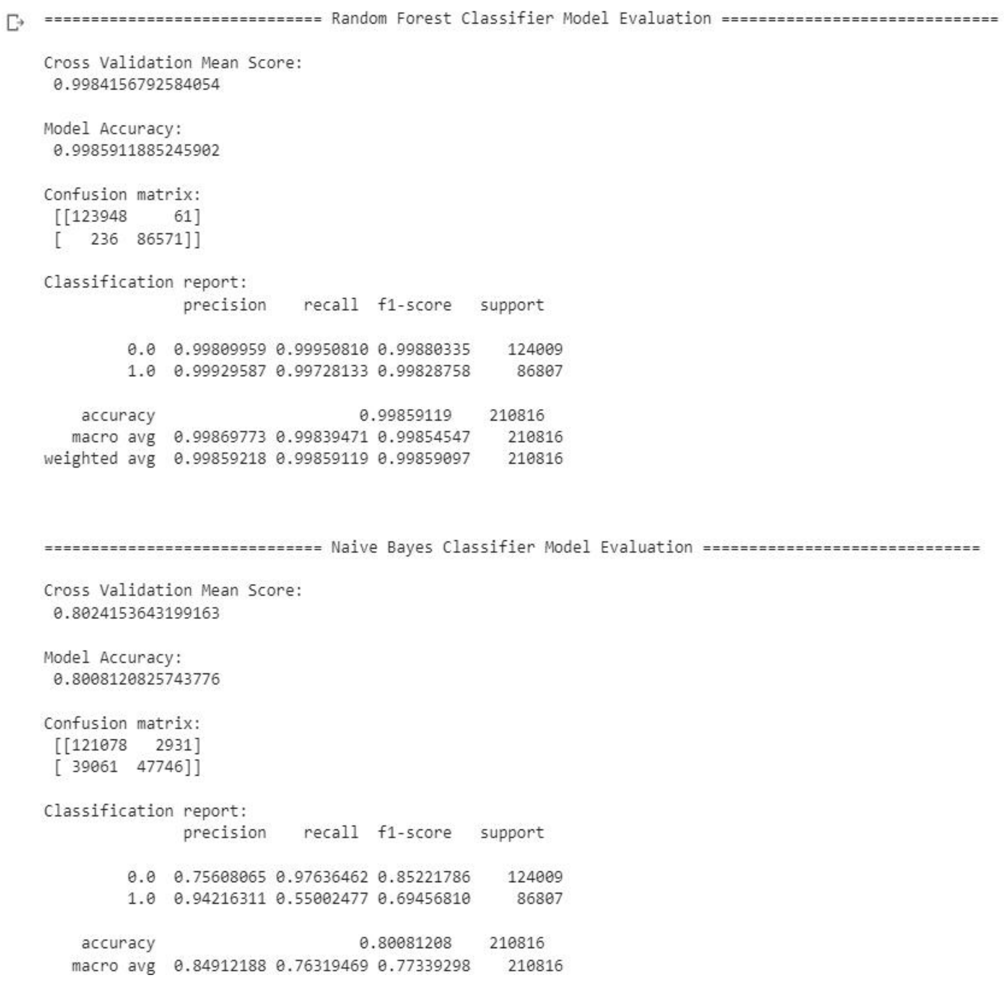
```
[ ] from collections import Counter
    print(f"Training target statistics: {Counter(y_res)}")
    print(f"Testing target statistics: {Counter(test_y)}")

    Training target statistics: Counter({0.0: 289924, 1.0: 289924})
    Testing target statistics: Counter({0.0: 124009, 1.0: 86807})
```



```
print("Classification report:" "\n", classification_XGB_RF)
print()

Prediction Time for Random Forest with XG BOOST:  0.23363852500915527


============================== Random Forest + XG Boost Model Evaluation ==============================

Cross Validation Mean Score:
 0.9994924476707159

Model Accuracy:
 0.9979176153612629

Confusion matrix:
 [[123894    115]
 [   324  86483]]

Classification report:
              precision    recall  f1-score   support

         0.0  0.99739168 0.99907265 0.99823146    124009
         1.0  0.99867202 0.99626758 0.99746835     86807

    accuracy                      0.99791762    210816
   macro avg  0.99803185 0.99767012 0.99784991    210816
weighted avg  0.99791888 0.99791762 0.99791724    210816
```



```
RandomForest_Classifier.fit(X_res, y_res)
t2=time.time()
print("Training time for Random Forest: ", t2-t1)
print()

Training time for BernoulliNB:  1.6082799434661865

Training time for Decision Tree:  5.268743991851807

Training time for XGBoost:  146.7908627986908

Training time for Random Forest:  4.453537940979004
```



```
============================== Random Forest Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.9984156792584054

Model Accuracy:
 0.9985911885245902

Confusion matrix:
 [[123948     61]
 [   236  86571]]

Classification report:
               precision    recall  f1-score   support

         0.0  0.99809959 0.99950810 0.99880335    124009
         1.0  0.99929587 0.99728133 0.99828758     86807

    accuracy                      0.99859119    210816
   macro avg  0.99869773 0.99839471 0.99854547    210816
weighted avg  0.99859218 0.99859119 0.99859097    210816



============================== Naive Bayes Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.8024153643199163

Model Accuracy:
 0.8008120825743776

Confusion matrix:
 [[121078   2931]
 [ 39061  47746]]

Classification report:
               precision    recall  f1-score   support

         0.0  0.75608065 0.97636462 0.85221786    124009
         1.0  0.94216311 0.55002477 0.69456810     86807

    accuracy                      0.80081208    210816
   macro avg  0.84912188 0.76319469 0.77339298    210816
```

REFERENCES

[1] M. Ahmad, S. A. Madani, A. R. Khan, and S. Hussain, "Hybrid intrusion detection system based on machine learning and fuzzy logic," Journal of Ambient Intelligence and Humanized Computing, vol. 12, no. 1, pp. 1039-1051, 2021.J.

[2] Chen, Y. Li, L. Li, and X. Li, "A hybrid intrusion detection approach based on machine learning and deep learning," Journal of Ambient Intelligence and Humanized Computing, vol. 11, no. 8, pp. 3633-3645, 2020.

[3] S. Shinde and S. S. Sonawane, "An intelligent hybrid intrusion detection system using machine learning and deep learning," Journal of Ambient Intelligence and Humanized Computing, vol. 11, no. 12, pp. 5201-5211, 2020.

[4] H. Chen, L. Han, J. Cui, and J. Ma, "A hybrid intrusion detection system based on machine learning and graph theory," Journal of Ambient Intelligence and Humanized Computing, vol. 11, no. 11, pp. 5063-5075, 2020.

[5] A. Al-Marghilani, A. M. Al-Salman, M. Al-Maadeed, and S. H. Al-Saadi, "A hybrid intrusion detection system based on deep learning and machine learning for cloud computing environments," Future Generation Computer Systems, vol. 111, pp. 250-259, 2020.

[6] Camila F. T. Pontes , Manuela M. C. de Souza , João J. C. Gondim , Matt Bishop, and Marcelo Antonio Marotta. A New Method for Flow-Based Network Intrusion Detection Using the Inverse Potts Model. IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 18, NO. 2, 2021.

[7] Nathan Shone , Tran Nguyen Ngoc, Vu Dinh Phai , and Qi Sh. A Deep Learning Approach to Network Intrusion Detection. IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, VOL. 2, NO. 1, 2018.

[8] Tahmina Zebin , Shahadate Rezvy, and Yuan Luo. An Explainable AI-Based Intrusion Detection System for DNS Over HTTPS (DoH) Attacks. IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 17, 2022.

[9] Riadul Islam , Member, IEEE, Rafi Ud Daula Refat , Sai Manikanta Yerram, and Hafiz Malik. Graph-Based Intrusion Detection System for Controller Area Networks. IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 23, NO. 3, 2022.

[10] Sk. Tanzir Mehedi, Adnan Anwar, Ziaur Rahman, Kawsar Ahmed and Rafiqul Islam. Dependable Intrusion Detection System for IoT: A Deep Transfer Learning-based Approach. https://doi.org/10.1007/s10723-021-09581-z, 2021.

[11] Earum Mushtaq , Aneela Zameer , Asifullah Khan. A two-stage stacked ensemble intrusion detection system using five base classifiers and MLP with optimal feature selection. Microprocessors and Microsystems, 2022,

[12] Emil Selvan G. S. R., M. Azees, CH. Rayala Vinodkumar, G. Parthasarathy. Hybrid optimization enabled deep learning technique for multi-level intrusion detection. Advances in Engineering Software 173, 103197, 2022.

[13] Romany F. Mansour. (2022). Artificial intelligence based optimization with deep learning model for blockchain enabled intrusion detection in CPS Environment. Scientific Reports, 12:12937, https://doi.org/10.1038/s41598-022-17043-z , 2022.

[14] L. Yang, A. Moubayed, I. Hamieh and A. Shami, "Tree-Based Intelligent Intrusion Detection System in Internet of Vehicles," 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1-6, doi:10.1109/GLOBECOM38437.2019.9013892, 2019.

[15] Malik, H., Abbas, H., & Shamsi, J. A. (2020). A hybrid intrusion detection system using deep belief networks and decision trees. Journal of Ambient Intelligence and Humanized Computing, 11(7), 2923-2937.

[16] Wang, X., Wang, P., Li, P., & Zhang, H. (2021). A hybrid approach based on convolutional neural networks and long short-term memory for intrusion detection. Journal of Ambient Intelligence and Humanized Computing, 12(1), 91-103.

[17] Kumar, V., & Reddy, G. R. (2021). A novel hybrid intrusion detection system using CatBoost algorithm. Computers & Electrical Engineering, 93, 107-122.

[18] Goyal, R., & Kaur, M. (2021). Hybrid intrusion detection system using XGBoost. Journal of Ambient Intelligence and Humanized Computing, 12(7), 6469-6482.

[19] Kaur, K., Singh, K., & Singh, B. (2021). Hybrid intrusion detection system using artificial bee colony algorithm and Naive Bayes algorithm. Journal of Ambient Intelligence and Humanized Computing, 12(7), 6523-6533.

[20] Bhuyan, M. H., Bhattacharyya, D. K., Kalita, J. K., & Jena, D. (2021). A novel hybrid intrusion detection system using Gaussian Naive Bayes and Particle Swarm Optimization algorithm. Journal of Ambient Intelligence and Humanized

Computing, 12(6), 5355-5369.

[21] Zhao, L., Xu, J., & Zhang, J. (2021). A hybrid intrusion detection system using random forest and AdaBoost. Journal of Ambient Intelligence and Humanized Computing, 12(9), 10965-10976.

[22] Fan, Y., Huang, H., & Song, Z. (2021). Hybrid intrusion detection system based on gradient boosting decision tree algorithm. International Journal of Distributed Sensor Networks, 17(5), 15501477211014077.

[23] Li, T., Gao, J., Feng, J., & Wei, J. (2021). A hybrid approach based on XGBoost and KNN for intrusion detection. Cluster Computing, 24(3), 2257-2268.ting, 12(6), 5355-5369.

[24] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, A Detailed Analysis of the CICIDS2017 Data Set. Springer, 2019.

[25] Kachwala, M. R., Patel, N. S., & Kothari, S. V. (2020). Detection and Prevention of SSH Patator Attack. In 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-6). IEEE.

[26] Zhang, Y., & Wen, S. (2020). A Novel Framework for Detection and Prevention of SSH Patator Attacks. Journal of Intelligent & Fuzzy Systems, 39(5), 6235-6242. doi: 10.3233/JIFS-201245

[27] Nadeem, A., & Saeed, H. (2017). FTP Patator Attack Detection Using Snort IDS. In 2017 3rd International Conference on Computational Intelligence and Information Technology (CIIT) (pp. 1-4). IEEE.

[28] Paniagua, C., Paniagua, J., & Garcia, F. (2019). FTP Patator: An Attack on FTP Servers. In Proceedings of the 2019 International Conference on High Performance Computing & Simulation (pp. 270-276). IEEE. doi: 10.1109/HPCS48598.2019.9188212

[29] Taneja, M., & Aggarwal, N. (2018). A Novel Mitigation Technique for GoldenEye Attack. In 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA) (pp. 1-4). IEEE.

[30] Wang, S., Zhang, Y., & Zhou, W. (2017). A Defense Approach Against GoldenEye Attack. In 2017 IEEE 14th International Conference on e-Business Engineering (ICEBE) (pp. 78-82). IEEE. doi: 10.1109/ICEBE.2017.018

[31] Halfond, W. G., & Orso, A. (2010). Detecting and preventing SQL injection attacks: a comparison of vulnerability scanners. IEEE Transactions on Software

Engineering, 36(1), 121-138. doi: 10.1109/TSE.2009.63

[32] Akamai Technologies. (2019). Q1 2019 State of the Internet / Security: Phishing for Finance. Retrieved from https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2019-state-of-the-internet-security-phishing-for-finance.pdf

[33] Kephart, J. O., & Chess, D. M. (1995). The vision of autonomic computing. Computer, 28(1), 41-50. doi: 10.1109/2.365224

[34] Douzas, G., and F. Bacao. "Combining Over-and Under-Sampling Techniques and Their Effects on Imbalanced Multi-Label Data Classification." Knowledge-Based Systems, vol. 161, pp. 37-45, 2018.

[35] Huttunen, H., Nurminen, V., Tikanmäki, A., and Toivonen, H. "Standardizing features in data sets". Neurocomputing, 267, 339-345. doi:10.1016/j.neucom.2017.07.018, 2017.

[36] N. Bhargava and G. Sharma, ''Decision tree analysis on J48 algorithm for data mining,'' Proc. Int. J. Adv. Res. Comput. Sci. Softw. Eng., vol. 3, no. 6, pp. 1114–1119, 2013.

[37] N. Farnaaz and M. A. Jabbar, ''Random forest modeling for network intrusion detection system,'' Procedia Comput. Sci., vol. 89, pp. 213–217, 2016.

[38] D. D. Lewis, ''Naive (Bayes) at forty: The independence assumption in information retrieval,'' in Proc. ECML, 1998, p. 16.

[39] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 785-794). ACM.

[40] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. In Advances in neural information processing systems (pp. 3149-3157).

[41] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. In Advances in neural information processing systems (pp. 6638-6648)