Classifying Imbalanced Data for DDoS Attack Detection

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

By

Amal A. Alghamdi

Approved by:

Major Professor: Frederick Sheldon, PhD.

Committee Members: Xiaogang Ma, Ph.D.; Jia Song, Ph.D.

Department Administrator: Terry Soule

December, 2021

Abstract

In the first quarter of 2021, researchers witnessed over 2.8 million Distributed Denial of Service (DDoS) attacks —a 32% increase from the same period in 2020, as reported by Info-Security magazine on May 18, 2021. The magazine also noted that the number of attacks against educational institutions has increased by 41% over the past three quarters. DDoS has become a serious issue for many organizations and individuals. The evolution of networks has ushered in a level of complexity that is the enemy of security. Currently, attacks are more prevalent and at the same time more noticeable due to the variety of features that exist on networks, a consequence of the constant escalation between attackers and defenders. Machine learning algorithms (MLAs) have become a tool to help thicken the layers of defense. To be effective, MLAs must be trained in ways that provide high confidence for detection and prevention, which boils down to precision and accuracy (i.e., low false positives and/or high true positives). This work has developed a setup for establishing a measured intrusion detection system (IDS) that can help to better understand and identify the various unique features of a network to better prevent DoS and DDoS attacks from being successful.

The goal is to develop models that can predict (i.e., classify) with high precision and accurately identify different types of DoS/DDoS attacks with low false positive/negative rates. In addition to dealing with the multiclass classification and extremely imbalanced problems, the derived model leverages two feature selection techniques to reduce the number of features in the dataset and help improve the model's execution time, thereby reducing the IDS complexity. A combination of under-sampling combined with adjusting weights was applied to handle the imbalance problem. The extracted data was evaluated using supervised MLAs, including Random Forest, Decision tree, Naive Bayes, Logistic regression, and ensemble methods. Ensemble methods using supervised outcomes aim to improve the overall performance of the classification. The experiments utilized the popular benchmark NSL-KDD and CICIDS2017 datasets. Random Forest achieved the best performance results, decreasing by 37% the training and testing time. In addition to solving the imbalance problem caused by feature selection, it increased accuracy 6.25% and FPR 21%. The random forest model has achieved 99% accuracy and 0.0001 for the False-Positive rate. Furthermore, using this setup, we can detect minor classes with more than 80% accuracy.

Acknowledgment

First and foremost, I would like to thank God for letting me through all difficulties that I have experienced day by day. Praises and thanks for his showers of blessings during my life.

I would also like to express my gratitude to my primary supervisor, Frederick Sheldon, who gave me the golden opportunity to do this wonderful project and guided me throughout this project. His guidance and advice carried me through my entire journey. Really grateful for his patience, motivation, vision, energy, and immense knowledge.

Besides my advisor, I would also like to thank my committee members Doctor Marshall Xiaogang and Doctor Jia Song for letting my defense be an enjoyable moment, and for their brilliant comments and suggestions, thanks to you doctors.

Last but not the least, I would like to thank my family: my parents Ali and Thahaba Alghamdi, for giving birth to me in the first place and for their love, prayers, and caring for educating and preparing me for my future. I would also like to give special thanks to my husband Rayan for his continuous support, love, and understanding when undertaking my research and writing my project.

Dedication

Dedicated to my Family for their faith and their advice, bless them. Dedicated especially to my amazing husband Rayan and my son Tayem; you are my life, my happiness, and everything. May Allah keep you all safe and happy. Remember that I will always love you.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Overview

Security has become an integral part of any organization's operations. Cybersecurity is very important for the safety of the organization and the public. It is a significant issue that requires implementing a strategy focused on securing cyberspace. The increasing popularity of these technologies has raised the level of attacks by criminal elements. The ability to easily exploit existing limitations of the Internet has become a major concern for organizations. Developing new countermeasures will help to mitigate cybercrime and will go a long way in preventing attacks.

An intrusion detection system is hardware or software that monitors the network traffic for suspicious or abnormal behavior. An anomaly-based detection approach is more prevalent than signature-based detection in detecting network threats. Traditional IDS still is not capable of detecting unknown attacks. In contrast, a distributed denial of service attack (DDoS) is a cyber-attack that attacks the network's resources. Usually, it overloads the bandwidth and prevents the intended users from accessing the network. DDoS is a distributed denial-of-service attack that uses TCP and UDP packets to flood a server with traffic. A DDoS attack is different from DoS because it uses multiple unique IP addresses to perform its operation. The attacks affect over a hundred Internet companies.

Machine learning is a field of artificial intelligence that has had promising results in detecting cyber-attacks such as DDoS. In machine learning, supervised methods classify anomalous data and distinguish anomalous and normal data from a tagged dataset. Unsupervised methods can only distinguish normal data from anomalous data. Ensemble Feature Selection Method (FS) is another technique that can improve the detection of DDoS attacks by selecting the most important and relevant feature. Combining different supervised predictions by ensemble methods such as voting will increase the accuracy and reduce false alarms. Ensemble learning works more accurately than a single classifier. The performance evaluation metric is called the confusion metric. The simple calculation of probability called the ROC-AUC curve is based on the true positive rate against the false-positive rate, which will help evaluate the model performance. By combining all methods, a new model will be produced to handle imbalanced multiclass classification and detect types of DDoS attacks. The model

will increase accuracy, reduce false alarms, reduce training time, deal with imbalanced data, and detect minor attack types.

1.2 Related Work

Due to the increasing number of attacks on computer systems, the demand for computer security has also grown. This is why various firms are focusing on developing effective Intrusion Detection Systems (IDSs). A distributed denial of service attack is carried out by an attacker to disrupt the operation of a computer system. It can be initiated by exploiting a vulnerability in the network. In this field, various techniques have been surveyed to minimize the malicious actions within end systems and networks. Some of the studies prove that the use of network-based systems and host-based systems can improve the detection of attacks [4].

Rajput and Thakkar's research provides an overview of network intrusion detection and countermeasure selection (NICE) graph models to describe the various countermeasures that are used to prevent and detect attacks in the cloud environment. Machine learning and deep learning techniques are some of the techniques that are being used to improve the anomaly detection using IDS [4].

Some authors like Sharmila et al. [3] illustrate several taxonomies for anomaly detection that have been developed, based on six criteria to classify IDSs: alert, architecture, environment, time of detection, processing, and data source. In Vasilomanolakis et al.'s research [5], the concept of distributed intrusion detection systems (CIDSs) was introduced as associated with requirements when deployed in large environments. The various types of attacks that can be exploited against IDSs were discussed. The CIDSs are mainly considered as communication architecture models that can be either centralized or decentralized. For each class, a further refined taxonomy and a detailed discussion of representative approaches are provided.

Recent work by Riza'ain et al. [7] introduces the concept of DDoS attack and its characteristics, along with discussing various techniques that are used to detect the attack. Their research provides the most accurate and up-to-date analysis and evaluation for each detection and prediction. It also provides a trusted source for information related to DDoS attack detection. Fifty-three articles from different libraries such as Science Direct, IEEE Xplore, Springer and Web of Science contributed to this taxonomy. About 30% used machine learning approaches in their detection system and had significant success.

In another study in detecting DoS attacks, Rohan et al. [8] applied five different ML classifiers for DoS attack traffic where the data was collected from the internet of things device network. The five tested algorithms are K-nearest neighbors algorithm (KNN), Support vector machine with the linear kernel (LSVM), Decision tree using Gini impurity scores (DT), Random Forest using Gini impurity scores (RF), and Neural Network (NN). The results revealed each classifier had higher accuracy at 99% and this could help other researchers in the field of machine learning anomaly detection.

Some authors focus on detecting DDoS attacks using different machine learning methods such as entropy-based anomaly detection [10], neural network-based detection [11], and deep learning [12], which successfully mitigated denial of service attack in IoT and cloud system. Some researchers focus on studying the impact of imbalanced data on the performance of ML-Based DDoS detection systems. The class imbalance problem is a major issue that affects the performance of various machine learning techniques. Liang et al.'s paper [13] presents an analysis of the impact of this issue on the performance of various ML-based techniques. The results suggest that the issue should not be underestimated when it comes to identifying and suppressing DDoS attacks.

To get an effective result by using SMOTE resampling before the train the model, Soe et al. [14], proposed a solution by designing an effective model to detect DDoS attacks in internet of things (IoT) system. They used a modern botnet attack dataset called Bot-IoT that contains a small number of benign samples with a large number of DDoS attack samples. Their work is focusing on solving the imbalanced data by using SMOTE oversampling before training the Artificial Neural Network (ANN) model. The outcome result shows that the model was effective to detect minor class with 99% reliability only using one hiding layer node and one output node in the ANN model.

Some proposed solutions focused on generating a new dataset by extracting the data from others' datasets to create a dataset that contains up-to-date DDoS attacks the same as Prasad et al.'s research [15]. They generated new dataset by combining CSE-CIC-IDS2018-AWS, CICIDS2017, and CIC DoS dataset. They proposed a method to detect DDoS attacks by using a gradient boosting algorithm. The results show that SGB performed better than most of the competing algorithms, with 100% misclassifying for both balanced and imbalanced data set with and without feature selection methods.

1.3 Thesis Objective

An IDS is designed to protect systems from different types of cyber-attacks. Its monitoring techniques should constantly be updated and should be capable of detecting unseen attacks. The rapid emergence and evolution of techniques and strategies for detecting DDoS attacks have greatly impacted the detecting system. Many of the techniques and approaches used to detect DDoS attacks have been refined and tested within a short time. However, due to the continuous growth of these techniques and approaches, they are no longer useful. Several experiments focus on detecting multi-class attacks, and others focus on binary classification of DDoS attacks. As discussed in Section 1.1, imbalanced data is still the major problem in many research works for binary classification and multi-class classification. Dividing the dataset into percentages (e.g. 80% of the dataset are attack and 20% are benign to increase the accuracy) can lead to the loss of some important information and behavior.

We proposed a supervised-based ensemble model, aiming to improve the accuracy of the model to detect multi-imbalanced classes along with detect DDoS attacks. This experiment focuses on:

- Providing a generic intrusion detection model that can be applied in multiclass classification.
- Designing a model that can deal with imbalanced data problems using resampling method
- Reducing training time of Machine learning models by minimizing dimensionality with Feature selection.
- Detecting DDoS attacks along with reducing false alarms.
- Comparing supervised ML models in terms of designing an accurate detection model.

To achieve these goals, we designed a model using two ensembles that predict for each sample by combining four supervised kinds of machine learning. Two feature selections were combined and applied to the data to select the most relevant features. In this experiment, two intrusion datasets, NSL-KDD [16] and CICIDS2017 [19], are used. The proposed work and their datasets will be explained in detail throughout Chapters 3 and 4.

1.4 Process Flow of Thesis

The proposed approach is divided into three stages. The first stage is preprocessing the data. In preprocessing data, Label Encoder will be applied to convert the non-numerical labels to numerical labels. After that, a combination of two feature selections is presented for the classification of IDS. Removing the irrelevant features can positively affect model performance. Feature reduction will be used to remove the irrelevant features, consisting of two different steps. First, Correlation will be applied to calculate the correlation between features for more data reduction. The higher correlation between two features will lead to removing one of the features because both features will contribute or affect similarly on the prediction process. The selected feature will be passed to Chi-square to calculate the relevance between feature and target. Then, the best 15 Chi-square scores will be selected and used to train the supervised ML model. In the second stage, supervised machine learning will be applied such as Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), and Naive Bayes (NB). At the third stage, voting ensemble learning will be performed with soft voting to make the prediction for each sample in addition to train the stacking ensemble. Finally, a confusion matrix will be used to evaluate the model and compared the second stage result to the third stage result. A ROC-AUC curve will be used to illustrate the prediction process. Figure 1.1 illustrates the process of proposed model.

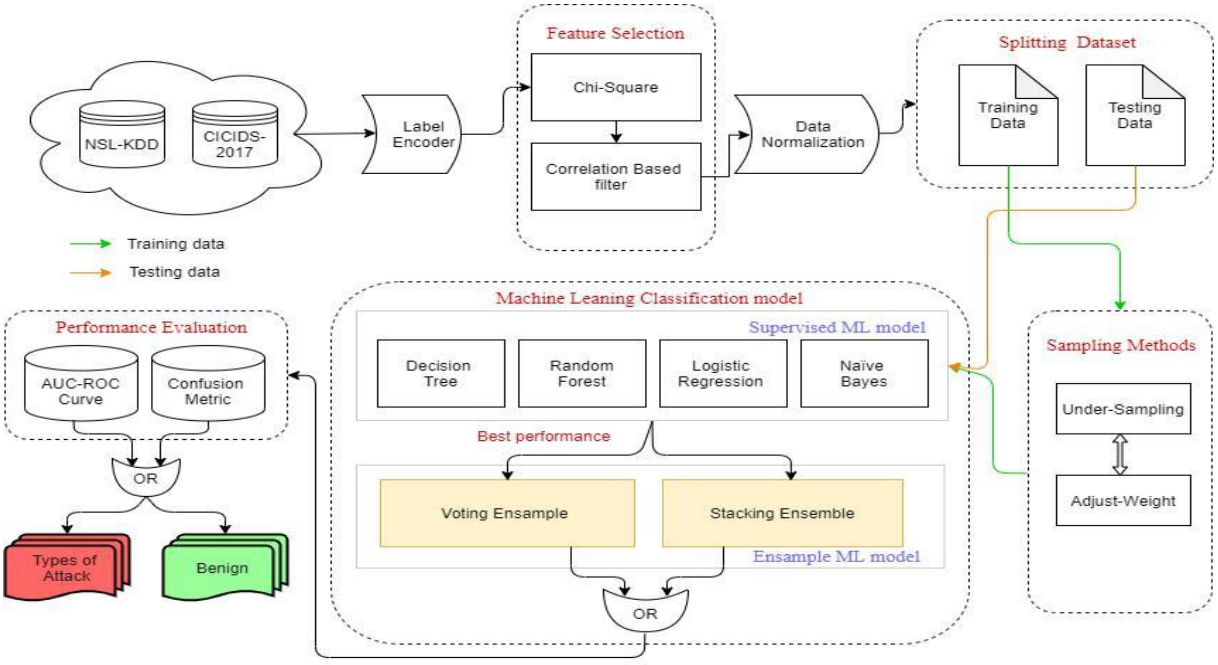

Figure 1.1 Process Flow of Thesis

1.5 Thesis Roadmap

Chapter 2 gives a brief discussion of the background on intrusion detection system IDS, distribution denial of service DDoS, and types of machine learning approach, feature selection, and performance evaluation. It also summarizes the ensemble approaches in detail and highlights some shortcomings of resampling techniques. The datasets that are used in the proposed methods are discussed in depth in Chapter 3. Two interpretable feature selections are applied and their explanation will be included in Chapter 3 along with resampling methods and data preprocessing. Ensemble frameworks using supervised classifiers are shown in Chapter 4 with their experimentation details. Finally, in Chapter 5, we conclude our thesis and demonstrate future directions in order to extend this thesis.

Chapter 2: Background

To better understanding the research problem, this chapter presents detailed background about the Intrusion Detection System (IDS), Denial of Service and Distribution Denial of Service Attacks (DoS/DDoS), Machine Learning (ML), and Ensample with Resampling methods.

2.1 Intrusion Detection System (IDs)

2.1.1 *What is IDS?*

Intrusion refers to various sorts of unauthorized activities that cause a potential threat to the availability, confidentiality, or integrity of information. An intrusion detection system (IDS) refers to a hardware system or software system that clearly identifies malicious behaviors on computer systems. It is a security mechanism that is used to protect the network against various types of attacks and malicious activities. It is deployed in combination with other security mechanisms such as firewalls and access control to secure the network. IDS is helpful to maintain system security and identify unusual or malicious behavior in network traffic. It aids in achieving high protection against harmful behavior. The intrusion detection process starts with monitoring network traffic and collecting data for analysis. It then identifies and investigates unusual behavior and patterns. It consists of various steps that are necessary to identify and prevent intrusions. IDSs can be placed anywhere in a network to monitor a system's activity. It can also be deployed inside an offline host to monitor its activities [1].

2.1.2 *Types of IDS*

Due to the complexity of network configurations, there are several types of IDS technologies that exist. Each type has its advantages and disadvantages. IDS can be classified based on detection techniques, data resources, and functionality [2] [3].

2.1.2.1 IDS Based on Detection Techniques

There are two main types of intrusion detection systems (IDS) that can be used to identify and prevent unauthorized activities. The first type is known as an Anomaly-based Intrusion Detection System (AIDS) and the second type is known as Signature-based Intrusion Detection System (SIDS).

a. *Anomaly-based IDS* (AIDS): This is a type of intrusion detection system that uses techniques that are not designed to identify suspicious activities. It comes to cover the limitations of SIDS [1]. AIDS contains two development phases, the training and

testing phase. In the training phase, usual user behavior is recorded and used to train the model. A new dataset is used in the test phase to evaluate the model and identify the hidden intrusions [2].

b. *Signature-based IDS* (SIDS): SIDS is an intrusion detection system that can only detect attacks that have been previously classified as malware. It requires frequent updates and cannot prevent all attacks that have not previously been identified [2].

The essential difference between SIDS and AIDS is that AIDS can identify zero-day attacks and new attacks [1].

2.1.2.2 IDS Based on Data Resources

IDS can be categorized as Network-based IDS (NIDS) and Host-based IDS (HIDS) based on data resources [2].

a. *Host-based IDS (HIDS)*: HIDS is a single computer security system that can monitor the security of a system or a computer from both internal and external attacks. HIDS collects the data from the host system and audits sources such as firewall logs and database logs. When a certain activity is detected by HIDS, it is immediately reported to the appropriate authorities. HIDS does not require land bandwidth and it requires less training than NIDS.

b. *Network-based IDS (NIDS):* NIDS collects the data from network traffic by capture packets and analyzes the capture information of the packet. NIDS uses advanced techniques to identify and prevent attacks on the network. It can identify and report the abnormal behavior of the network. It can also send an alert to the administrator when abnormal behavior is observed. NIDS is easily managed through a centralized environment and centrally managed. It is built-in to support cross-platform environments. But it requires more training and the failure rate is higher than HIDS.

Both IDSs are installed on the workstation and are not connected to the network. During host compromise, a NIDS can be disabled by attackers. The main difference between a NIDS and a HID is that the latter is installed on the workstation while the former is on the network.

2.1.2.2 IDS Based on Functionality

a. Intrusion Detection System (IDS) is a type of security system that monitors network activities and triggers alarms. It can also be used for monitoring, detecting, and preventing unauthorized activities on a network
b. Intrusion Prevention System (IPS) is another type of security system. IPS is designed to prevent attacks before they can enter a network. It uses a combination of packet analysis and network profiling to identify the patterns of attacks and block them.
c. Intrusion detection and prevention systems (IDPS) are designed to prevent and detect unauthorized access to an organization's network. It works by classifying and protecting the packet data before it is processed. The systems are equipped with various components, such as pre-processing, classification, and prevention. This feature can be used for various purposes such as phishing, spam, and unauthorized access.

2.1.3 IDS Requirement

Some factors need to be considered while we build an IDS model. Here are some of the requirements for building a IDSs [2]:

Accuracy: the accuracy of an IDS is determined by the percentage of attacks that were successfully detected and the percentage of false negatives. Both percentage factors need to be taken into account to calculate the accuracy of an IDS.

Scalability: requires that the IDS's performance increases linearly with the addition of new resources. It should not contain bottlenecks or special purpose objects.

Resilience: in the event of a CIDS failure, it should still be able to maintain its availability and integrity. This is especially true in the event of internal attacks, where malicious components could cause damage. It should also be resilient to attacks from internal and external components.

Minimal overhead: the overhead refers to the effort involved in generating and communicating intrusion alerts must be minimal. Also, the signaling inside the IDS should be as minimal as possible.

Privacy: in a collaborative environment, an exchange of alerts may contain sensitive information that should not be shared or disclosed to all parties involved.

Self-configuration: is the ability to modify a system's configuration without requiring an administrator to do so. This feature allows the system to self-configure itself. It avoids the need for an administrator to enter configuration.

Interoperability: is the ability of an IDS system to work seamlessly with other systems in the same network. This can be achieved through the utilization of various standardized formats such as the Intrusion Detection Message Exchange Format (IDMEF).

2.2 Denial of Service Attack (DoS)/Distribution DoS

DoS is a malicious attempt to overwhelm the target with a flood of Internet traffic for disruption or make services unavailable to users. The most common method of a DoS attack happens when the attacker floods the webserver with communication. In this kind of DoS attack, the attacker sends some requests to the target computer, overloading it with communication. These maintenance requests are illegal and have fabricated return addresses that inform the computer when it attempts to certify the requestor. As the junk requests are processed incessantly, the computer is overcome, which causes the DoS shape to legitimate requestors.

The first known DoS attack was carried out by a 15-year-old hacker in February 2000, which targeted several e-commerce sites such as Amazon and eBay by using a series of distributed denial of service attacks. The attacks used various techniques to overwhelm the servers of the various Internet companies. The FBI estimated that the cost of the attacks was over $1 billion. A distributed denial of service (DDoS) attack happens when multiple systems flood a targeted system with enough bandwidth to render it unusable. The difference between DoS and DDoS is that DoS floods a server with TCP and UDP packets from single machine, but a DDoS attack usually uses more than a single unique IP address or machine [27]. DDoS attacks have increased recently and become a serious issue for organizations and the government.

The well-known DDoS attack that targeted GitHub was executed on February 28, 2018, and caused a massive influx of traffic rate of 1.3TBps at a rate of 126.9 million per second. It was carried out using a database caching method known as Memcached. The attack lasted for 10 minutes and the platform was down for 5 minutes. It was stopped within this timeframe only

due to the platform's DDoS protection. One week was the time taken for recovery. Designing new countermeasures will aid in reducing the flow of cybercrime. The latest and largest DDoS attack occurred in the first quarter of 2020. Amazon Web Services (AWS) detected a distributed denial-of-services attack with a volume of 2.3 TBps. This event was 44% larger than the previous record-breaking event detected by AWS. The architecture and the types of DDoS attacks will be explained in this section.

2.1.1 DDoS Attack Architecture

A distributed denial of service (DDoS) attack consists of four components: the attacker, the target, the handlers, and bots. The goal of the attack is to disrupt the services of the target. The attacker and the target are both systems that can be controlled through bots and controllers. Attacks usually require a large volume of traffic to crash a website or network. From a single computer it is usually difficult for attackers to generate large amounts of traffic. To do this, they often use botnets (hundreds or thousands of internet-connected computers). These computers are infected with malware and controlled by the attacker. The attacker uses controllers to infect many botnets and this allows attacker to execute a DDoS attack on target by installing threats or instructions to the bots about how or when to attack the target to cripple it. In most cases, the attacker creates a botnet with high-rate traffic and then infects other systems to make more bots with its malicious code. This method increases the attack's power and makes it possible to make any target down within a short time. These methods make it basically impossible to detect the original source to prevent the attack. Fig 1 shows the architecture of DDoS.

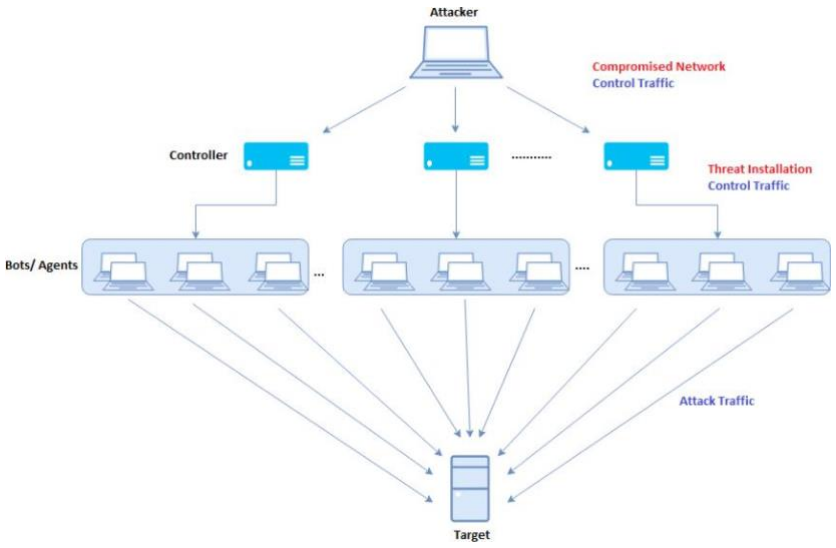

Figure 2.1 DDoS Attack Architecture

2.1.2 DDoS attack types

DoS and DDoS attacks can be divided into three types [30]:

- Volumetric attacks: also known as floods, these are a type of DDoS that use UDP or ICMP to overwhelm a site. The goal is to inflict as much bandwidth as possible. It is measured in bits per second (Bps) and allows attackers to launch massive DDoS attacks, which can reach levels of up to terabits per second.
- A Protocol attack: this is a type of network attack that uses a combination of various techniques to cause physical and virtual server resources to shut. Instead of attacking higher level resources, protocol attacks seek to exploit weaknesses in the protocols used to connect to the Internet. They usually involve the exploitation of the normal behavior of the protocols. A protocol attack can be initiated by sending multiple packets to a single server and it measured in packets per second (Pps). SYN floods, fragmented packet attacks, Ping of Death, Smurf DDoS are some types of protocol attacks.
- Application layer attack: is a type of attack that targets web servers and web application platforms. The attackers try to crash the server and make it unusable to the users. Attacks are initiated by sending HTTP requests to a web server. These attacks can execute arbitrary code or expose known application vulnerabilities. They can also abuse higher-layer protocols (HTTPS and SNMP).

Here are descriptions of some of the most common types of DDoS attacks:

SYN flood: is a volumetric attack that floods a server with new connection requests and causes a server to ignore or stop processing new connection requests. In a three-way handshake:

A client sends an SYN (synchronize) packet to a website to request a new connection. The server responds with an SYN-ACK (synchronize-acknowledge) packet.

The server waits for the response with ACK (acknowledge) packet from the client. In an SYN flood attack, the attacker sends more SYN requests to the server without responding with the ACK packet. This causes the server to overload but cannot respond to new requests [26]. Figure 2.2 shows the complete three handshake and incomplete SYN flood.

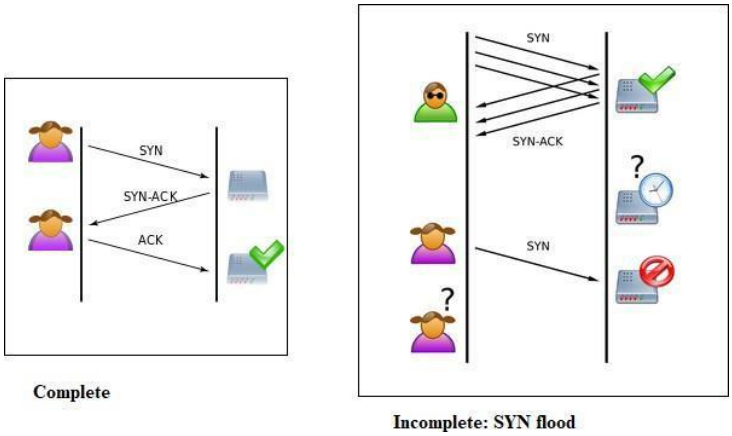

Figure 2.2 Complete and Incomplete Connection of Three Handshake

UDP flood: is a volumetric and protocol attack that tries to abuse the normal behavior of the UDP protocol, which has no handshake mechanism. It does not create a session and cannot verify the sender's IP address. The attacker sends a large number of UDP packets with forged IP addresses to various ports on the victim's server. Once the server gets overwhelmed, it can no longer respond to the requests [26].

HTTP GET and POST floods: HTTP is a protocol that enables people to communicate with web servers. It is used to establish a connection. HTTP GET and HTTP POST are two commonly used methods to obtain data from a resource. An attacker can easily take down a website by sending HTTP or POST requests to it continuously. These attacks are difficult to detect because they mimic legitimate HTTP requests. They appear to be happening even though they are not [25].

A ping of death attack: This is a type of attack that uses several or malicious pings to a computer. The maximum frame size of a packet over an Ethernet network is locked to 1500 bytes. In this case, the fragments are split into several IP packets. The Data Link Layer typically has limits to the frame size that are applicable to it. In this case, the recipient hosts partition the IP packet into fragments and reassemble them into a complete packet. In a Ping of Death scenario, an attacker can cause a packet to overflow its memory buffers by loading a packet with a size larger than 65,535 bytes. This can cause a denial of service due to memory overflow [26].

2.3 Machine Learning

Machine learning (ML) refers to the process of extracting interesting data patterns from massive amounts of data, leading to the recognition of unusual behaviors. Supervised, unsupervised, and semi-supervised learning are the three types of Machine learning. Several algorithms and techniques have been applied to discover the attack patterns from intrusion datasets (e.g. neural networks, decision trees, and nearest neighbor methods). This section will briefly explain the types of machine learning and specifically focus on the supervised classification algorithms.

2.3.1 Supervised Machine learning classification:

Supervised learning uses the target in the training process and helps in the prediction process. In supervised machine learning, the algorithm is trained to predict the correct class of a given data. It can then predict the label based on its previous predictions. Through the training process, the model can learn the relationship between samples and their labels. With a sufficient amount of data and over time, the model can then observe a good label for any new given data.

Classification and regression are the two types of supervised machine learning [38]. The classification model is employed when the target variable is a category, for instance, classified as normal or breast cancer diseases. Logistic Regression (LR), Naïve Bayes (NB), Decision Tree (DT), Random Forest (RF) are used on the proposed classification model. Here is a brief summary of each algorithm [39]:

Decision Tree: Decision tree is a non-parametric supervised learning method that works by building a tree with multiple branching trees where each leaf shows a decision. The algorithms collect information about the subject and apply rules for the purpose of decision-making. Both classification and regression can be used in decision tree methods. The goal of the decision tree is to create a training model that can lead to an accurate decision by classifying data according to predefined attributes.

Random Forest: Random Forest contains a great number of individual decision trees that operate as an ensemble method. It is used for classification and regression. In the random forest, each tree predicts a class and the class with the most votes is final model's prediction. Random forest is the modification of bagging which are mainly used for building

large collections of de-correlated trees. Sadly, bagging can have poor predictive power and suffers from tree correlation.

Logistic Regression: Logistic Regression was mainly used in the biological sciences during the early twentieth century. It is also used in social science applications. Logistic Regression is a kind of parametric classification model that can classify linearly separable classes. It is used to explain data and calculating the probability of a class to demonstrate the relationship between the dependent variables. The idea behind logistic regression is similar to linear regression, but instead of using a line to fit into the giving data, it uses a curve to fit the data such as sigmoid. The best coefficients are those that minimize the error that the model produces and predict a value close to 1 for the default and normal classes.

Naive Bayes: Naive Bayes is a set of supervised learning algorithms used for classification tasks based on the idea of dependent probability. The naive Bayes classifier is a simple and widely used probabilistic classifier. It uses the Bayes theorem to define the probability of an event after taking into account the conditions of prior knowledge of certain conditions associated with its events. Naive Bayes has several types, such as Gaussian Naive Bayes, Multinomial Naive Bayes, Complement Naive Bayes, and Bernoulli Naive Bayes. Gaussian Naive Bayes will be used in the proposed model and will be explained later in Chapter 4.

2.3.2 Unsupervised Machine Learning:

Unsupervised learning is an advanced form of machine learning that uses algorithms to analyze and cluster large amounts of unlabeled data. It is a technique that tries to identify previously unknown patterns in data. It is an ideal tool because of its ability to discover hidden patterns and differences in information. It is hard to directly apply on classification and regression algorithms because of the unlabeled data making it difficult to train the model as normally do in supervised machine learning. Most of the time, discovering the pattern may fail to provide the expected results. Since there is no way to know what the goals are and what the results should be, it is very difficult to measure the accuracy of the outputs, which makes supervised machine learning more suitable to real-world problems [38].

2.3.3 Semi-Supervised Learning

In semi-supervised learning, the learning is performed by means of supervised and unsupervised methods. The goal is to minimize the need for labeling and eliminate the need for skilled human experts. Semi-supervised learning algorithms can learn the structure of a data set by analyzing both labeled and unlabeled data. In semi-supervised learning, a combination of a small amount of labeled data and a large amount of unlabeled data will be used during training process. With the help of labeled data, the model can accurately classify the unlabeled data. Some of these algorithms include label propagation, generative methods, and hidden Markov models [38].

2.4 Ensample Learning

Each model in machine learning makes predictions for each sample. Ensemble learning is a method that combines these predictions to form a more robust classifier in the ML community. The advantage of ensembles is that they can perform better than a single classifier by increasing accuracy and stability. This technique is used to minimize error that is caused by noise, variance, and bias. It has been used in the field of cyber security to detect Intrusion, distribution denial of service, malware, and fraud. Ensemble methods can be categorized under basic and advanced methods. Both categories will be explained in detail in this section.

2.4.1 Basic Ensemble Techniques

Voting Ensemble is the easiest ensemble method. It is used by adding together the predictions of multiple classifiers. It then uses soft voting or hard voting to make the decision or prediction. This voting method is also a parallel ensemble method because there are no dependencies between the results of base methods. All base models will be separately trained and then combined into voting methods. The voting ensemble method can be used for regression and classification. In the hard voting the model will predict based on the majority vote that comes from the base classifiers. In soft voting, the prediction will be based on the average probability calculated using all the base model predictions. Voting ensemble will be better to use when all base models have a good performance. It has some limitations, however, as it treats all the base models equally, and not all models can have good performance. So, these ensemble methods sometimes cannot perform better than single methods [40].

2.4.2 Advanced Ensemble Techniques

The Advanced ensemble method is a sequential method where each base model depends on the previous one. With any poor result, the second model will correct the error of the previous model by improving the weight. In advanced ensemble, the two main ways to ensemble the classifiers are homogeneous and heterogeneous. Homogeneous refers to the methods that combine similar types of classifiers or use single base leaner such as bagging and boosting methods, while heterogeneous refers to the method that combines different types of classifiers or multi-base leaners such as stacking. Bagging, stacking, and boosting are some types of ensemble learning. Here is a brief description [31] [32]:

Stacking: stacking is an ensemble leaner method that combines multi-classifiers via the Meta model. Stacking contains two stages called level 0 and level 1, where level 0 is the level that contains the multi different models that will train the entire dataset and push the prediction to level 1. In level 1 the Meta model same as the base model will combine the prediction as a feature and trained the Meta model. The algorithm used in level 0 can reduce either bias or variance [32].

Bagging: is called Bootstrap Aggregation. It is a parallel method, similar to majority voting, where each model is trained alone and then combined to make a final prediction. The only difference is that in bagging a different subset of data will be used to train multiple models and aggregate them for the final decision. It runs faster than using a whole dataset and it can reduce the overfitting and variance. Bagged decision trees and extra trees are some examples of bagging.

Boosting: boosting is a sequential method that adjusts a weight from the previous model. Boosting is designed to reduce bias in the result. For example, if the sample is classified incorrectly, then the weight of that sample will be increased in the next iteration. However, if the sample is classified correctly, then the weight of that sample will be decreased in the next iteration until it achieves the best result with low bias. The parameter turning is important in boosting because the model may face overfitting in the training process and parameters can help to reduce it. AdaBoost, Gradient Boosting Machine (GBM), XGBoost, and CatBoost are some examples of boosting.

2.5 Feature Selection

Feature Selection is a process that selects the most relevant features that contribute primarily to the model predictions or output. It is an integral part of machine learning because the irrelevant features can negatively impact the training process, which may lead to poor prediction. In each model, cleaning data and feature selection should be the first step to help the model to accurately predict the attacks. Feature selection can improve the performance of the model by reducing overfitting, improving accuracy, and reducing training time. Less data with important content can reduce the training time and reduce the complexity of the model, in addition to reducing overfitting by removing the noisy data. All this leads to increased accuracy and better predictions. This section will explain briefly the three types of feature selection, which are Filter, Wrapper, and Embedded methods. The correlation feature selection and Chi-square feature selection will also be explained in detail.

2.5.1 Filter Methods

Filter methods compute the properties of features by using univariate statistical techniques instead of using cross-validation. It is faster and cheaper than wrapper methods in terms of using high dimensionality data. In filter selection, features are selected according to their statistical tests' correlation with an outcome variable instead of using machine learning algorithms. Several methods were proposed, including Chi-square, which is applied to categorical variables, and correlation methods that were applied to continuous variables [37].

1) Chi-Square:

The test determines the relationship between two or more random features. It tells how much difference exists between these two features. If the result of the test is very small, it means that the observed data fit very well and both features have a relationship. Otherwise, the observed data do not fit and have no relationship [33]. The O and E in the formula (1) are the observed and the expected values [33].

$$X_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

2) Pearson's Correlation:

Pearson's Correlation is a statistical equation that calculates the linear correlation between two features referred as x and y. The value of Pearson's (r) can be 1, 0, or -1 [34].

Where:

+1 → positive correlation

0 → no linear correlation

-1 → negative correlation

$$r = \frac{\sum(x_i - \bar{x})^2 \, (y_i - \bar{y})^2}{\sqrt{\sum(x_i - \bar{x})^2 \, \sum(y_i - \bar{y})^2}}$$

Where

r = correlation coefficient

x_i = values of the x-variable in a sample

$\bar{x}$ = mean of the values of the x-variable

y_i = values of the y-variable in a sample

$\bar{y}$ = mean of the values of the y-variable

High correlation will be nearest or equal 1 and low correlation will result nearest to -1. The higher correlation between two features will lead to removing one of the features because both features will be contributed or affected similarly [34].

2.5.2 Wrapper Methods

In a wrapper method, a set of features will be selected to train a model. After that, a decision will be made based on the inferences drawn from the previous model and it will be decided which features to add or remove. Wrapper methods are typically used to solve a search problem. They are very expensive to implement and are typically not used for large-scale problems. Forward feature selection, backward feature elimination, and recursive feature elimination are common examples in wrapper feature selection. The filter method is a statistical method that is measured the relevance between variables based on correlation, but wrapper methods use cross-validation in their measurement by actually training a model. Compared to wrapper methods, filter methods are much faster and do not require training [37].

2.5.3 Embedded Method

The Embedded method combines the advantages of both the wrapper and filter methods. It has its own built-in features selection methods. This method performs the feature selection process during the construction of a machine learning algorithm. A learning algorithm performs feature selection and classification/regression at the same time. It uses its own variable selection process. They take into account the interaction between features like wrapper methods and filter methods. They are faster and more accurate than filter methods and are typically less prone to over-fitting [37].

2.6 Resampling Methods

In designing a multiclass classification or regression model, it should be treated with an imbalanced dataset problem, especially on a supervised model. The accuracy may appear high, but the model can be biased and only detect the majority class. It cannot be robust in dealing with a minority class and that's why designing a model within an imbalanced data could be tough. Resampling is an efficient method for dealing with highly imbalanced datasets [36].

Under-sampling: This is a widely used technique to remove samples from a dataset that is heavily imbalanced. This method is used to remove samples from the majority class and make it equal to the number of minority class in order to create balanced data. The major disadvantage is potentially losing some important information [23].

Over-sampling: Over-sampling works by taking advantage of the diversity of the sample and then adding more examples to the minority class. Oversizing the number of minority class to be equal to the number of majority class. The major disadvantage leads to an increase in overfitting by duplicating an existing sample and slowing down the training time [36].

Adjust weight: Adjusting weight could be a simple solution to avoid the problems of both over and under sampling methods. This concept proposes to weigh the loss of various samples differently depending on the class they belong to. It tries to assign high weight to the minority class and small weight to the majority class. It could be used to balance classes weighted to automatically adjust weight for classes. This method is truly effective when dealing with large dataset, because oversampling could be expensive and requires more memory space. So, weight can help estimators to train and learn without losing information or facing a lots of noise and overfitting [35].

2.7 Performance Evaluation

Different evaluation processes have been applied to evaluate the model and display results as a curve using an AUC-ROC curve. Confusion metric and AUC-ROC were used to evaluate the proposed model. This section will explain both in detail.

2.7.1 Confusion Metric

A confusion metric is a performance evaluation metric used to measure the detected class based on given predicted and real value. It is used for binary or multiclass types. False-positive, false negative, true positive, and true negative are used to evaluate the model. Table 2.1 illustrates how the confusion metric calculates the rates for class attack 2. False positive and false negative rates are important because they provide information when the model is classifying samples erroneously [28].

Table 2.1 Confusion Matric Explanation

		predict class				
		Normal	Attack 1	Attack 2	Attack 3	Attack 4
Real class	Normal	TN		FP	TN	
	Attack 1					
	Attack 2	FN		TP	FN	
	Attack 3	TN		FP	TN	
	Attack 4					

To test the capability of the detection model, accuracy, precision, recall, F-measure, FPR, and specificity are used to evaluate the performance of the model. The mathematical equations of these measures are as follows [4]:

Where:

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

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

$$\text{TPR/ Sensitivity/ Recall (R)} = \frac{TP}{TP+FN}$$

$$\text{F - Measure} = \frac{2(P*R)}{P+R}$$

$$\text{False-Positive Rate (FPR)} = \frac{FP}{FP+TN}$$

$$\text{Specificity} = \frac{TN}{TN+FP}$$

$$FPR = \frac{FP}{TN+FP} \text{ or } 1 - \text{Specificity}$$

True-Positive (TP): the number of attacks accurately identified as malware.

Ture-Negative (TN): the number of normal accurately identified as benign.

False-Positive (FP): the number of normal inaccurately identified as malware.

False-Negative (FN): the number of attacks inaccurately identified as benign.

2.7.2 Receiver operating characteristic -AUC (Roc-AUC Curve)

The ROC-AUC curve, where AUC is short of the area under the curve and ROC short of receiver operating characteristic, is a simple calculation that plots the curve of the probability of the true-positive rate against the false-positive rate for multiclass data. The former shows the percentage of observations that are correctly predicted to be positive while the latter shows the percentage of observations that are correctly predicted to be negative. It is also a performance measure that shows how much a model can improve its ability to classify classes. The Roc curve also exposes how much control the model has over distinguishing between normal with different attacks. An excellent model has AUC near the 1 which shows it has high separability. A model has AUC close to 0 which shows it has the worst separability. Fig I, Fig II, and Fig III illustrate how to evaluate the model from the Roc curve [6]. For each class in one model with different colors will be plotted to compare the differences. The curve result will be explained later in chapter 5.

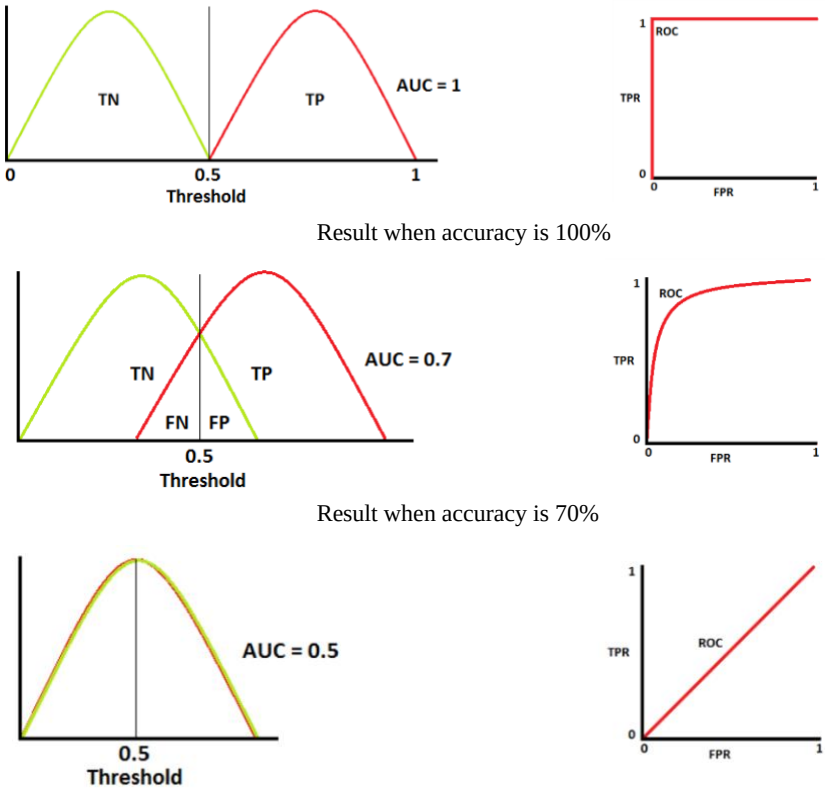

Figure 2.3 Maps the Roc Curve in Different Accuracy Rate [29]

2.8 Chapter Summary

This chapter covered the basic information necessary to understand the rest of the research. It illustrated the Intrusion Detection System (IDS) types and requirements in addition to Denial of service and Distribution denial of service attack (DoS/DDoS). It also explained supervised Machine learning (ML) types, ensemble learning models, and Feature selection types with resampling methods. Finally, the performance evaluation such as ROC-AUC and Confusion metric were explained in depth. Chapter 3 will cover the dataset and the preprocessing steps that were conducted on the research.

Chapter 3: Dataset and Data Preprocessing

This section reviews the selected datasets in addition to explaining their pre-processing approaches. Both NSL-KDD and CICIDS-2017 datasets were used in the proposed model, later explained in Chapter 4. The proposed model was applied on the entire dataset and an extracted subset that only contained DoS/DDoS attacks for each dataset. The data will be explained in detail. In the preprocessing phases, the data transformation, normalization, scaling, and feature selection approaches will be discussed.

3.1 Dataset

The IDS datasets have been available since 1998. The first dataset called DARPA was produced in 1998 and includes not real-world network traffic. Another dataset called KDD'99 is an update of the DARPA dataset. Several other datasets have been created across time. Sharafaldin et al.'s research [16] illustrates and evaluates eleven openly available IDS datasets along with their gaps and types of attacks. In the data collecting stage, creating new data is challenging and time-consuming. Simultaneously, creating a dataset can be valuable, useful, and frequently used by several researchers. Using an existing dataset may be helpful in terms of comparing the results with previous studies. Two datasets were used in our research, namely NSL-KDD (the updated version on KDD'99) and the cicids2017 dataset. This section provides the details of these two datasets [17].

3.1.1 NSL-KDD

NSL-KDD Analyze intrusion detection has become very valuable recently. Much research focused on diverse datasets to increase accuracy and to reduce the false-positive rate. KDD cup 99 dataset is an updated version of DAPRA 98. Because of some issues in the KDD cup 99 version that affect the performance of the systems and reduce the accuracy of anomaly detection approaches, the NSL-KDD dataset was proposed. The NSL-KDD contains all the KDD-cup 99 samples, which solves the gap in the KDD-cup 99 version. NSLKDD is used to apply several machine learning algorithms in order to detect malware attacks. The main advantages of the new NSL-KDD dataset are the following:

- The training set has not included any redundant records that can lead to biased results.
- The test set has no duplicate samples which will produce more trust prediction rates.
- The number of samples in the dataset are sensible, which makes the dataset low-cost to run the experiments.

This section covers a brief explanation of the NSLKDD dataset. This selected dataset provides a beneficial analysis for different machine learning techniques based on intrusion detection.

1) Attack Types

The records in the NSL-KDD dataset are classified as normal or one of the four grouped attacks containing 37 different kinds. Table 3.1 maps the four attacks with types and examples. The four main classes are [18]:

a. DoS: Denial of service is a malicious attempt to overwhelming the target with a flood of Internet traffic for disruption or make services unavailable to users (for example, SYN flooding).
b. U2R: unauthorized access to victim device aiming to gain the root privileges by exposing some machine vulnerability such as buffer overflow attacks.
c. Probe: another kind of attack that aims to gain information about the remote victim such as port scanning.
d. R2L: illegal entrance to a remote machine that gains local access to the target machine (for example, guessing the password).

Table 3.1 Maps the Attack Classes with Its Types [1]

Class	**Types**	**Example**
DOS	Back, Land, Neptune, Pod, Smurf, Teardrop, Mailbomb, Processtable, Udpstorm, Apache2, Worm	SYN flooding
Probe	Satan, IPsweep, Nmap, Portsweep, Mscan, Saint	Port-scanning
R2L	Guess_password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Named, warezclient, spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail,	Password-guessing
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps	Buffer- overflow

2) Features

In the NSL-KDD dataset, each record reveals different traffic features with 43 attributes plus an assigned label classifying each record as either normal or attack. In addition, another label categorized the 37 types of attack in four groups ranged from 0 to 5. The features of the dataset hold three types of data: Nominal, Numeric, and Binary. The features 'protocol_type',

'service', 'attack, and 'flag' have nominal values while the features 'logged in', 'root_shell', 'su_attempted', 'num_root' and 'num_file_creations' contain binary values, and the other features are numeric types. Table 3.2 illustrates the number for each DoS attack class in the training and test set. Table 3.3 maps the numbers of records for each class for both training and test set where 80% for training and 20% for test.

Table 3.2 Number of record for each DoS attack

DoS attack class	**Training set**	**Test set**	**Total**
Normal	61,617	15,436	77,053
Neptune	36,708	9161	45,869
Smurf	2651	660	3311
Back	1044	271	1315
Teardrop	735	169	904
Apache2	607	130	737
Processtable	531	154	685
Mailbomb	230	63	293
Pod	204	38	242
Land	-	-	25
Worm	-	-	2
Udpstorm	-	-	2
Total	**104,327**	**26,082**	**130,438**

Table 3.3 Number of record for each NSL-KDD attack types

Attacks class	**Training set**	**Test set**	**Total**
Normal	61,697	15,356	77,053
DOS	42,685	10,700	53,385
Probe	11,219	2858	14,077
R2L	3179	761	3880
U2R	91	28	119
Total	**118,811**	**29,703**	**148,514**

3.1.2 CICIDS2017

The Canadian Internet Incident Dataset 2017 (CICIDS2017), which was created by the University of New Brunswick's Institute for Cybersecurity, contains the most up-to-date records on several types of cyberattacks. The dataset contains real network traffic of over 2,830,108 records where 471,454 of them are the malicious records. It was analyzed using real traces of the traffic. This unique dataset features an up-to-date attack list and is capable of handling different types of attacks comparison with other datasets such as UNSW-NB15 [21]. The attacks include Botnet, Web Attack, Infiltration Attack, DoS Attack, Brute Force Attack, HeartBleed Attack, and Distributed DoS (DDoS) Attack. This section provides an overview of the dataset, including traffic analysis conducted using CICFlowMeter, which is a tool that enables network traffic analysis. It includes the results of the labeled flows based on the timestamp, source, and destination IPs, source and destination ports, protocols, and attacks. The data was collected in 5 days from Monday morning, July 3, 2017 until Friday July 7, 2017 afternoon. Table 3.4 illustrates the dataset files including their types of attacks.

Table 3.4 CICIDS-2017 Dataset Summary [20]

File Name	Type of Traffic	Number of Record
Monday-WorkingHours.pcap_ISCX.csv	Benign	529,918
Tuesday-WorkingHours.pcap_ISCX.csv	Benign SSH-Patator FTP-Patator	432,074 5,897 7,938
Wednesday-workingHours.pcap_ISCX.csv	Benign DoS Hulk DoS GoldenEye DoS Slowlories DoS Slowhttptest Heartbleed	440,031 231,073 10,293 5,796 5499 11
Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv	Benign Web attack- Brute Force Web Attack-Sql injection Web Attack-XSS	168,186 1,507 21 652
Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv	Benign Infiltration	288,566 36
Friday-WorkingHours-Morning.pcap_ISCX.csv	Benign Bot	189,067 1966
Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv	Benign Port-scan	127,537 158,930
Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv	Benign DDoS	97,718 128,027
Total Instance/Record	**-**	**2,830,743**

1) Attack Types

The CICIDS2017 dataset contains different types of DoS and DDoS. In our experiment, the two file *Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv* and *Wednesday-workingHours.pcap_ISCX.csv* were merged to generate one file that included random selected types of denial-of-service attack (DoS) with distributed denial of service attack (DDoS). Table 3.5 explains each type of Both DoS and DDoS attacks.

Table 3.5 CICIDS2017 DDos attacks type's description [21]

Traffic Type	**Size**	**Description**
Benign	2,358,036	Normal traffic behavior
DoS Hulk	231,073	The attacker uses the HULK tool to generate massive amounts of DoS traffic that can be obfuscated or unique. This method can also bypass server caching engines.
DDoS	41,835	The attacker uses multiple machines that operate together to attack one victim machine
DoS GoldenEye	10,293	The attacker uses the GoldenEye tool to perform a denial of service attack
DoS Slow Loris	5796	The attacker uses the Slow Loris tool to execute a denial of service attack
DoS Slow HTTP Test	5499	The attacker exploits the HTTP Get request to exceed the number of HTTP connections allowed on a server, preventing other clients from accessing and giving the attacker the opportunity to open multiple HTTP connections to the same server

2) Features

A total of 79 features were extracted and taken from the information present in the pcap file (e.g. time stamp, source and destination ports, source and destination IPs, protocols and attack) [21]. The best common features for DoS detection are the related features of the Flow Inter arrival time (IAT), such as Min, Mean, Max and also the Flow Duration. Finally, for DDoS attack, also some related features to IAT have been selected along with average packet size and backward packet length [16]. The feature reduction will be explained later in Section 3.3. The number of samples of DoS/DDoS for both training and testing data is explained in Table 3.5 for each type.

3.2 Data Preprocessing

Data cleansing, feature reduction, and resampling are the three phases of data preprocessing. In the first step, the data is cleansed by converting string instance to number using the label encoder in addition to deleting some columns that are not required in the training process. For example, some datasets include three-columns: one for binary labeled data (normal/attack), a column for the multi-class label, and the last column contains the name of attacks. Based on the purpose of the research, the column will be selected and the others will be deleted. In the second step, Chi-2 and correlation-based filter selection will be used to reduce the dimension of the data. After that, the dataset will split into 80% for the training set and 20%

for the testing set. The data will be normalized to help the model read the structure of data in the same way. In the last step, resampling techniques will be used to handle imbalanced data.

Dataset cleansing: in the NSL-KDD two data columns were deleted. The two-column *attack map* and *attack* that contain binary labeled (attack/normal) and general types of NSL-KDD, which make these two columns out of the target and not necessarily in the experiments. This is because in our experiments we are trying to test models only on DoS and DDoS attack in NSL-KDD and CICIDS2017.

Label encoder: used to convert each instance into values arrange between 0 to 1. It can also be used to convert non-numerical to numerical values. In our research, a label encoder was used to convert the non-numerical instances to numerical instances [24].

MinMax Scaler: is a normalization method used to rescale data by estimate minimum and maximum values. The goal is to make data more understandable to the model. The Sklearn.preprocessing library contains *MinMaxScaler* that was used in the experiment preprocessing step [22].

$$y = (x - \min) / (\max - \min)$$

Where:

Y is the normalized result. X is a giving value.

Min and Max are minimum and maximum values in the dataset.

3.3 Feature Reduction

The Feature selection was done for each dataset separately. Two methods were selected to remove irrelevant features with the goal of reducing training time and increasing model performance. The correlation-based feature selection will be applied first to calculate the correlation between features. The higher correlation between two features will lead to removing one of the features because both features will contribute or affect similarly. Next, the result of Correlation methods will be passed to the Chi-square to calculate the relevance between each feature and the target. Then, the highest 15 feature scores will be selected based on k highest scores. Finally, the residual features will be used to train the model. Table 3.6 explains the

number of the feature after each feature selection method, along with their original number of features for both datasets.

Table 3.6 Reduced feature sets for The Two dataset

Dataset	**Methods name**	**Feature Count**	**Features**
NSL-KDD DoS	All set	42	All feature
	Chi-2	20	1, 5, 6, 8, 23, 24, 25, 26, 27, 28, 29, 31, 32, 33, 34, 36, 38, 39, 40, 41
	Correlation	20	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 30, 31, 32, 34, 35, 36, 37, 39, 41, 43
	Both	15	1, 3, 5, 8, 23, 24, 29, 30, 31, 32, 34, 36, 37, 39, 41
CICIDS2017 DoS/DDos	All set	79	All feature
	Chi-2	13	2, 15, 16, 17, 18, 19, 21 22, 23, 37, 38, 42, 43
	Correlation	15	1, 10, 15, 20, 25, 29, 30, 37, 38, 66, 72, 73, 74, 76, 79
	Both	15	1, 10, 15, 20, 25, 29, 30, 37, 38, 66, 72, 73, 74, 76, 79

3.4 Under Sampling and Weight Assign

Due to the nature of the class data, most machine learning algorithms are biased and many algorithms are not very useful when dealing with imbalanced data. The imbalance problem refers to the dataset that contains different variance numbers of classes. This problem affects classification performance. It can lead to a serious bias against the majority class. To solve this problem, two resampling methods, Neighborhood Cleaning Rule (NCL) and adjusting weight, were used in the preprocessing level in order to avoid biased data [23].

In the preprocessing stages, Neighborhood Cleaning Rule (NCL) was used to deal with the imbalance problem for both majority and minority classes on the selected dataset. NCL used the Edited Nearest Neighbor Rule (ENN), which is an under-sampling method based on the concept of nearest neighbor (NN) [23]. It is removing any instance from the majority class that belongs to a different class than most of the selected number of neighbors. NCL uses the same concept of ENN in removing from majority class but it also deals with minority instance by removing the nearest class that belongs to the majority. In our research, Neighborhood Cleaning Rule (NCL) was done by select 10 of Neighbor and 0.9 for threshold cleaning. After that, the outcomes were used in the next step to adjust the weight for each class. Because under sampling

the majority class in extremely imbalanced data is not helpful and NCL will not make the number of classes equal or even close, the weight was adjusted by using set different weights for each class. This method works by giving different weights to minority and majority classes. The goal is to give more weight to the minority groups that most likely will classify as the minority classes and set a small weight for the majority group. This helps the model in the training process to treat classes equally and focus more on minority classes to avoid biased prediction. Furthermore, it will increase the accuracy by accurately predict minor classes. Table 3.7 and Table 3.8 represent the number of samples before and after the resampling technique along with their adjusted weight.

Table 3.7 Number of CICIDS-DDoS Sampling After Applying Sampling Methods

Resampling Methods		CICIDS2017 – DDoS/DoS						
		Benign	DoS Hulk	DDoS	DoS Golden Eye	DoS Slow Loris	DoS-Slow HTTP	Heart bleed
NCL	Before	351,925	184,768	102,557	8260	4686	4379	9
	After	350.741	184.369	102,396	8153	4627	4337	9
Adjust Weight		0.26	0.50	0.91	11.47	20.21	21.56	1039.0

Table 3.8 Number of NSLKDD-DDoS Sampling After Applying Sampling Methods

Resampling Methods		NSL-KDD - DoS									
		Benign	Neptune	Smurf	Back	Teardrop	Apache	Process table	Mail bomb	Pod	land
NCL	Before	61,673	36,711	2606	1048	724	587	554	229	195	20
	After	61,402	36,688	2509	1040	718	585	547	224	158	19
Adjust Weight		0.16	0.28	4.14	9.9	14.4	17.7	18.9	46.3	56.1	519.5

3.5 Chapter Summary

As shown above, this chapter gave brief information about the two popular benchmark NSL-KDD and CICIDS2017 datasets that were utilized in the experiments. It also explained the approaches that are used to handle the data such as using Label encoder and MinMax methods to normalize and scale data. In addition to that, it explained how we remove irrelevant features using Correlation and Chi-2 feature reduction. Because many ML algorithms are biased and not useful when dealing with imbalanced data, this chapter discussed the resampling methods that were used in the proposed framework to handle these issues. Chapter 4 will discuss and compare benchmark testing and results along with the feature reduction results.

Chapter 4: Benchmark Testing and Results

4.1 Introduction

The proposed framework is built based on the Scikit-learn library [41]. The proposed approach is separated into three steps: data preprocessing and reduction as described in Chapter 3, supervised machine learning methods, and ensemble classifier. In preprocessing data, Label Encoder has been applied to convert the non-numerical labels to numerical labels. After that, two feature selections are used to remove the irrelevant features for a positive effect on the time. Chi-square and Pearson's correlation have been used and tested on both datasets. Section 2.5 chapter 2 will explain the results for each test. Finally, the under-sampling model weight will be used to give weight for each class that has been applied to solve the problem of imbalanced data that will be caused by some types of feature selection. This chapter will explain the supervised machine learning experiment using Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), and Naive Bayes (NB) models. In addition, the soft voting ensemble and stacking ensemble will be performed to make a prediction for each sample by combining the best results from the previous steps. Finally, each model will evaluate using a confusion matrix and ROC-AUC described in Section 2.7 Chapter 2. In this chapter, two experiments have been done on two benchmark datasets, CICIDS2017 DoS/DDoS attacks and NSL-KDD DoS attacks, for both supervised and ensemble and a detailed explanation for the results aiming to evaluate the model. Table 4.1 explains the hyper-parameter for each model used in our experiments. The same hyper-parameter is used in the two dataset training process.

Table 4.1 Hyper-parameter for Supervised Model and Ensemble Classifier

Classification type	**Classifier Name**	**Abbreviated Name**	**Hyper-parameter**
Supervised	Logistic Regression	LR	solver= 'liblinear' , C= 10, class_weight=class_weights
	Decision Tree	DT	criterion="entropy" , class_weight=class_weights
	Random Forest	RF	class_weight=class_weights
	Naïve Bayes	NB	Default
Ensemble	Stacking	Ens-Stack	estimators=[(‘Best model performance’)], final_estimator=RandomForest, cv=3
	Voting	Ens-Vote	estimators=[(Best model performance’)], voting='soft'

4.2 Feature Selection

Among all the feature selections, two methods were chosen based on the dataset's value types. The Chi-square and Correlation have been applied separately on the NSL-KDD dataset as the first step. Then, both features have been integrated and applied, with the correlation applied first and the chi-square second for more feature reduction. Both outcomes' results will be compared before and after applying feature selections by using four supervised ML models. The comparison focused on how the feature selections have affected training time, model accuracy, and False-positive rate. Feature reduction may reduce training time but sometimes it has a negative impact on accuracy and False-Positive rate; that is because removing sufficient features to reduce training time will not help the models to learn enough from features with short of time.

In our experiment, the correlation method provided the best performance that reduced training time without affected model accuracy and FPR. In contrast, applying both correlation and chi-square reduced more training time, but the accuracy and chi-square were affected. Table 4.2 represents the results of CICIDS2017 before and after applying Chi-square and Correlation methods. From Figure 4.1, we can notice that there is a big improvement in the time after we used both methods together. Based on the results from Table 4.2, we can see that after applying correlation the time decreased 8% than before and 46% decreased after applying both methods. About the accuracy and false-positive rate, there are rare changes after applying correlation versus a huge decrease after applying Correlation and Chi-square methods.

Table 4.2 Feature Selection Comparisons for CICIDS2017 Dataset

ML model	**Before**			**correlation**			**Correlation + Chi-2**		
	ACC	FPR	Time	ACC	FPR	Time	ACC	FPR	Time
LR	0.98	0.002	269.0	0.98	0.002	255.6	0.91	0.01	128.7
DT	0.99	0.0002	34.09	0.99	0.0001	29.12	0.74	0.09	15.58
RF	0.99	0.0001	309.12	0.99	0.0001	302.1	0.80	0.03	281.9
NB	0.79	0.03	2.87	0.79	0.03	2.56	0.54	0.08	0.839

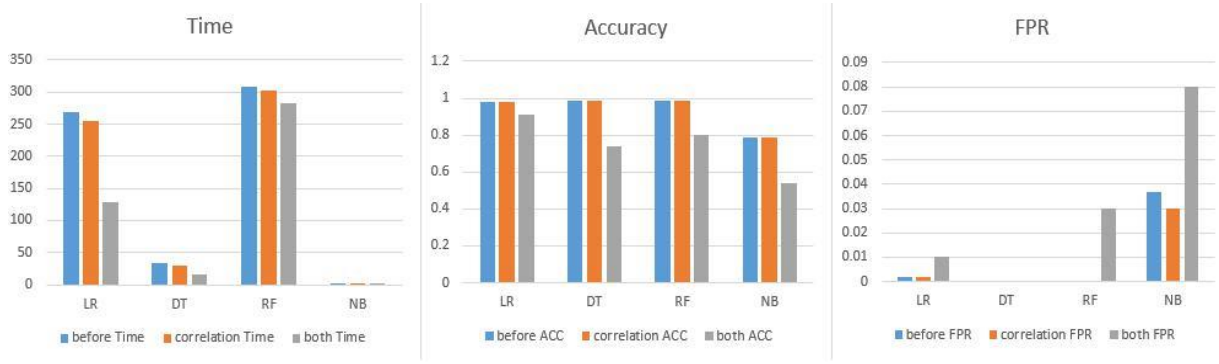

Figure 4.1 Feature Selection Performance Comparisons for CICIDS2017

Same as the previous dataset, each feature selection has been applied separately and a collection of two applied next. Table 4.3 represents the results of the NSL-KDD DoS before and after applying feature selection methods and Figure 4.2 represent the data in more readable way. In this experiment, the feature selection has affected the accuracy, FPR, and time differently for each ML model. For instance, after applying both methods, the Decision Tree has been affected by increasing accuracy and reducing FPR and time. About the time and results from Table 4.3, the time was reduced by 16% after applying correlation and by 42% after applying both methods (Correlation and Chi-square).

Table 4.3 Feature Selection Comparisons for NSL-KDD Dataset

ML model	**Before**			**correlation**			**Both**		
	ACC	FPR	Time	ACC	FPR	Time	ACC	FPR	Time
LR	0.99	0.0003	11.52	0.99	0.0008	8.39	0.97	0.004	6.31
DT	0.72	0.20	0.53	0.72	0.02	0.43	0.99	0.0008	0.29
RF	0.99	0.0001	8.15	0.99	0.0001	8.02	0.99	0.0001	7.52
NB	0.95	0.010	0.38	0.95	0.010	0.30	0.94	0.006	0.11

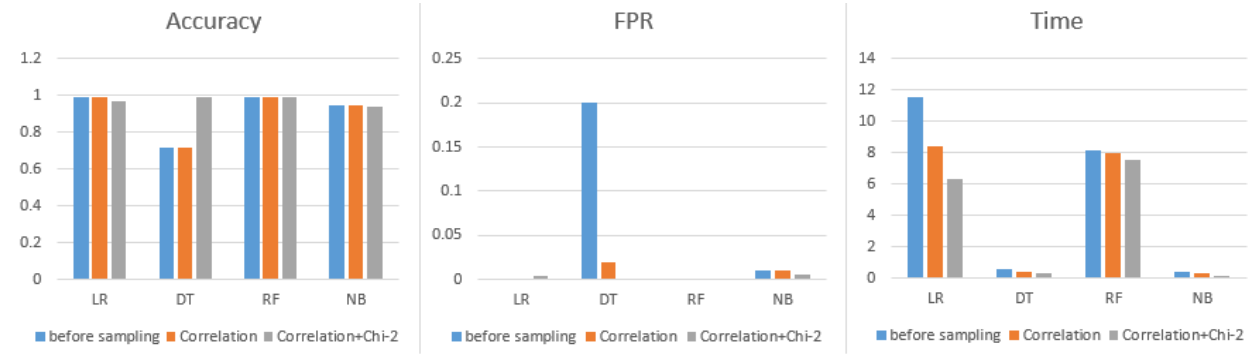

Figure 4.2 Feature Selection Performance Comparisons for NSL-KDD

The aim of feature reduction in our experiment is to reduce training time only. Based on the huge difference in time before and after applying both feature selections, the time was hugely reduced training time for both datasets. The combination of Correlation and Chi-square was selected to build the IDS model. However, this method produced another problem called imbalanced data. To solve this problem, a combination of two sampling methods have been used to reduce the number of sampling and adjust the weight. Section 0.0 will illustrate the process and the results of sampling methods. To conclude, feature selections reduced training time by 44% from the original training time.

4.3 Sampling performance

Sampling methods have been used to solve the problem of imbalanced data caused by feature selections. Neighborhood Rule Cleaning is the method that was used in this stage to remove some samples based on the concept of nearest neighbor (NN). This method was selected because it deals with minority instances by removing the nearest class that belongs to the majority. At this point, the minority class will not be affected much by this method. Removing some samples from the majority class that is misclassified by their 10-nearest neighbors will help models to avoid overlap between samples and avoid overfitting. This method still needs some help from other methods because the number of samples from each class did not get close. Adjusting weight for each class has been used to create a balance between classes when the model is trained. Table 4.4 and Figure 4.3 explain how the sampling methods help to improve the models' performance after applying feature selection on the CICIDS2017 dataset. Based on the results from Table 4.4, these two sampling methods have reduced FPR by 55% and 34% time, in addition, to increase accuracy by 12%.

Table 4.4 Supervised models performance comparisons after sampling method for CICIDS2017

ML model	**Feature selections**			**Sampling methods**		
	ACC	FPR	Time	ACC	FPR	Time
LR	0.91	0.017	128.7	0.94	0.012	73.29
DT	0.74	0.09	15.58	0.88	0.03	11.33
RF	0.80	0.033	281.9	1.00	0.0005	172.17
NB	0.54	0.084	0.839	0.64	0.058	0.58

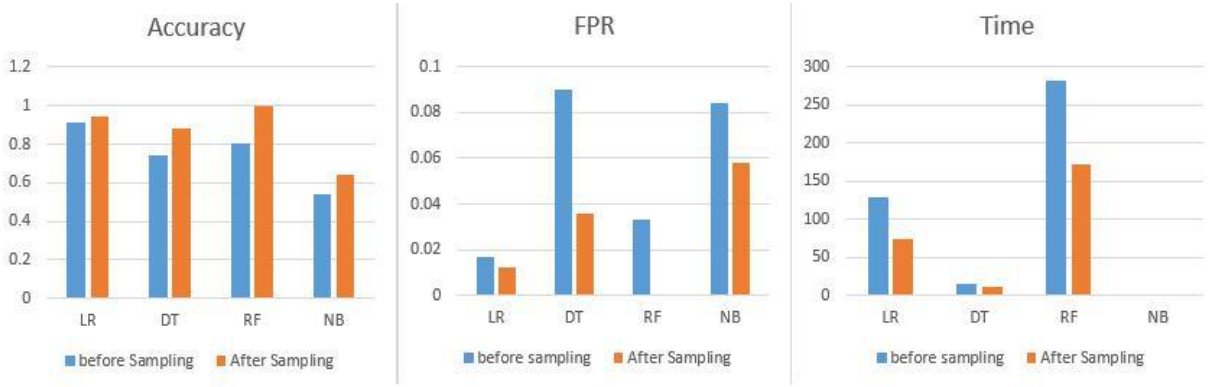

Figure 4.3 Supervised ML performance after sampling methods for CICIDS2017

Regarding the NSL-KDD dataset and from the results of Table 4.5, the accuracy improved rarely especially on Naive Bayes and Random Forest. For FPR in Figure 4.4, the FPR was improved by 20% specifically on Decision Tree and Naive Bayes. The time also improved by 8% for all models.

Table 4.5 Supervised models performance comparisons after sampling methods for NSL-KDD

ML model	**Feature selections**			**Sampling**		
	ACC	FPR	Time	ACC	FPR	Time
LR	0.97	0.004	6.31	0.97	0.004	5.76
DT	0.99	0.0008	0.29	0.99	0.0003	0.27
RF	0.99	0.0001	7.52	1.00	0.0001	6.98
NB	0.94	0.006	0.11	0.95	0.005	0.10

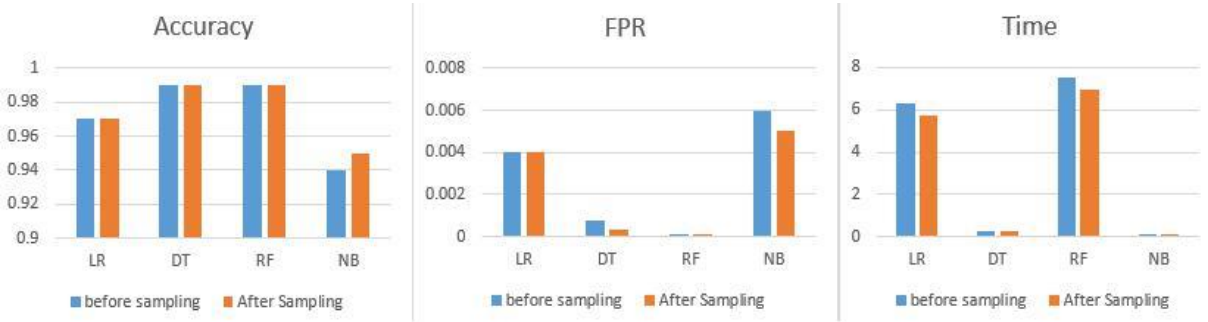

Figure 4.4 Supervised ML performance after sampling methods for NSL-KDD

Finally, good performance resulted from CICIDS2017 because the number of samples that the models were trained and tested on is 820 thousand samples, as compared with NSL-KDD, which contains 130 thousand samples.

4.4 Results and Discussion

This section represents the best models' performance for both supervised ML models and Ensemble methods. In the Ensemble methods, soft voting is used to vote for each sample by calculating the average of predictions from multiple classifiers. About the stacking, multi-classifiers will be combined via the Meta-model. Stacking contains two stages called level 0 and level 1. Where level 0 is the level that contains the best-supervised ML models. In level 1 the Meta model, the same as the base model, will combine the prediction as a feature and train the Meta-model, which is a random forest in our experiment. Tables 4.6 and 4.7 illustrate all the models' performance for CICIDS2017 and NSL-KDD datasets.

Table 4.6 Supervised models and Ensemble models performance for CICIDS2017

Class	Accuracy	FPR	ROC-AUC (Macro)	ROC-AUC (Micro)	Time
LR	0.94	0.012	0.98	0.99	73.29
DT	0.88	0.036	0.93	0.93	11.33
RF	**1.00**	**0.0001**	**1.00**	**1.00**	**172.17**
NB	0.64	0.058	0.93	0.93	0.58
Stack	**0.99**	**0.0003**	**1.00**	**1.00**	**250.12**
Voting	0.99	0.002	1.00	1.00	72.77

Table 4.7 Supervised models and Ensemble models performance for NSL-KDD

Class	Accuracy	FPR	ROC-AUC (Macro)	ROC-AUC (Micro)	Time
LR	0.97	0.004	1.00	1.00	5.76
DT	0.99	0.0003	0.89	0.99	0.27
RF	**1.00**	**0.0001**	**1.00**	**1.00**	**6.98**
NB	0.95	0.005	0.99	1.00	0.10
Stack	**1.00**	**0.0002**	**0.99**	**1.00**	**56.6**
Voting	0.99	0.0004	1.00	1.00	13.30

With the minor classes Table, 4.8 and 4.9 represent the Precision, Recall, and F-1 for Random forest models for CICIDS2017 and NSL-KDD datasets. The model is robust in detecting the minor class such as in Table 4.8, where Heartbleed only has 2 samples in the test set. The same concept in Table 4.9 represents the types of NSL-KDD, where land only has 5 samples in the test set.

Table 4.8 Confusion Metric results for each class of CICIDS2017

Class	**RF**		
	Precision	Recall	F1
Benign	1.00	1.00	1.00
DoS Hulk	1.00	0.99	0.99
DDoS	0.99	0.99	0.99
DoS GoldenEye	1.00	1.00	1.00
DoS Slowhttptest	0.98	0.98	0.98
DoS slowloris	1.00	1.00	1.00
Heartbleed	**1.00**	**1.00**	**1.00**

Table 4.9 Confusion Metric results for each class of NSL-KDD

Class	**RF**		
	Precision	Recall	F- 1
Apache	1.00	0.99	0.99
Back	1.00	1.00	1.00
Land	**0.80**	**0.80**	**0.80**
Mailbomb	1.00	1.00	1.00
Neptune	1.00	1.00	1.00
Normal	1.00	1.00	1.00
Pod	0.94	0.98	0.96
Processtable	1.00	1.00	1.00
Smurf	1.00	1.00	1.00
teardrop	0.98	0.99	0.99

4.5 Chapter Summary

In our proposed model, Random Forest achieved the best performance results. The model decreased 37% of the training and test time, in addition to solving the imbalance problem caused by feature selection, and increased accuracy 6.25% accuracy and FPR 21%. The random forest model has achieved 99% accuracy and 0.0001 for the False-Positive rate. Furthermore, it can detect minor classes with more than 80% accuracy.

Chapter 5: Conclusion

Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are a type of cyber-attack that can disrupt a server's normal operation. Typically, these use a network of computers to execute a DDoS. Even the most prominent service providers like Facebook, Amazon, and Instagram have experienced service disruption due to DDoS. Therefore, Intrusion detection systems (IDSs) should be used to build a system that should be able to identify and prevent an attack using artificial intelligence. Recently, deep learning and machine learning show promising results in solving these types of complicated problems. Machine learning is an AI tool that can provide solid intelligence when used by IDSs. The ability of machine learning to detect patterns of attacks makes it an ideal tool for cybersecurity. This thesis project aimed to design a model that uses the advantages of the feature selection process to reduce time and improve the overall performance of such a system. Further, we have addressed the problem of imbalanced data, characteristic of DoS and DDoS attacks, that demonstrates improved overall performance compared to other studies described in the literature.

5.1 Summary

To summarize our results from this research, we have designed a model using four supervised learning methods namely, Decision tree, Random Forest, Logistic Regression, and Naive Bayes, in addition to using two ensemble methods (i.e., stacking and voting). Two popular benchmarks called NSL-KDD and CICIDS2017 have been used as offline data to train the model, specifically the DoS and DDoS that have been extracted from both datasets. Both of the NSL-KDD DoS and CICIDS2017 DoS/DDoS attacks have been normalized and split into a training set and testing set. Two feature selections (i.e., Chi-square and correlation) and two resampling methods (i.e., Neighborhood Cleaning Rule and Adjust Weight) have been applied on both datasets. After that, the supervised models have been derived and the best results were used to train the ensemble model. Comparing all results, Random Forest achieved the best performance results. The model decreased 37% of the training and test time, in addition to solving the imbalanced problem caused by feature selection and increased accuracy 6.25% and FPR 21%. The Random Forest model has achieved 99% accuracy and 0.0001 for the False-Positive rate. Furthermore, it can detect minor classes with better than 80% accuracy. The result has met our goals of designing a model that can detect DoS and DDoS attacks with high

accuracy, low false rates, and in less computational time in addition to dealing with extremely imbalanced data.

5.2 Challenges

Many interesting and difficult problems have been addressed in this thesis. Designing a detection model that can effectively detect various types of DoS and DDoS attacks with extremely imbalanced data was the biggest issue faced during this journey. IDSs require a robust model that can accurately detect minor types with low false alarms in a short period of time. Time was the most important aspect considered in designing the model. To reduce the time, feature selection was the second problem faced. Choosing the correct feature helped to improve time to detection. Regarding the accuracy and false alarm rates, with the help of resampling techniques many practices such as random over-sampling and random under-sampling were tested and did not show an improvement with the false alarm rate. Another issue is that more samples can cause the model to exacerbate the overfitting problems and its side effects (i.e., higher false alarm rates, lower accuracy, and precision).

5.3 Future Work

In the near future, we will test the same model but we will use an unsupervised model instead of a supervised model. The best outcomes will be integrated with the decision tree to build a semi-supervised model. This model will be used in real-world traffic to train the unlabeled data that will be used along with the supervised offline labeled data. Different feature selection and resampling techniques will be studied toward developing a faster and more accurate system.

References

[1] A. Thakkar and R. Lohiya, "Attack classification using feature selection techniques: a comparative study," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 1, pp. 1249-1266, 2020. Available: 10.1007/s12652-020-02167-9.

[2] D. A. Effendy, K. Kusrini, and S. Sudarmawan, "Classification of intrusion detection system (IDS) based on computer network," in *2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, 2017.

[3] S. KishorWagh, V. K. Pachghare, and S. R. Kolhe, "Survey on intrusion detection system using machine learning techniques," *Int. J. Comput. Appl.*, vol. 78, no. 16, pp. 30–37, 2013.

[4] D. Rajput and A. Thakkar, "A survey on different network intrusion detection systems and Counter Measure," in *Emerging Research in Computing, Information, Communication and Applications*, Singapore: Springer Singapore, 2019, pp. 497–506.

[5] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Taxonomy and survey of collaborative intrusion detection," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–33, 2015.

[6] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861-874, 2006. Available: 10.1016/j.patrec.2005.10.010.

[7] A. Riza', A. Yusof, N. I. Udzir, and A. Selamat, "Systematic literature review and taxonomy for DDoS attack detection and prediction," *Int. J. Digit. Enterp. Technol.*, vol.
1, no. 3, p. 292, 2019.

[8] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer Internet of Things devices," *arXiv* [cs.CR], 2018.

[9] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed DDoS detection mechanism in software-defined networking," in *2015 IEEE Trustcom/BigDataSE/ISPA*, 2015.

[10] A. S. S. Navaz, V. Sangeetha, and C. Prabhadevi, "Entropy based Anomaly detection System to prevent DDoS attacks in cloud," *arXiv* [cs.CR], 2013.

[11] J. Li, Y. Liu, and L. Gu, "DDoS attack detection based on neural network," in *2010 2nd*

International Symposium on Aware Computing, 2010.

[12] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 2, no. 1, pp. 41–50, 2018.

[13] X. Liang and T. Znati, "An empirical study of intelligent approaches to DDoS detection in large scale networks," in *2019 International Conference on Computing, Networking and Communications (ICNC)*, 2019.

[14] Y. N. Soe, P. I. Santosa, and R. Hartanto, "DDoS attack detection based on simple ANN with SMOTE for IoT environment," in *2019 Fourth International Conference on*

Informatics and Computing (ICIC), 2019.

[15] M. D. Prasad, P. Babu, and C. Amarnath, "Machine learning DDoS detection using stochastic gradient boosting," *Int. J. Comput. Sci. Eng.*, vol. 7, no. 4, pp. 157–166, 2019.

[16] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, 2018.

[17] "NSL-KDD," Unb.ca. [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html. [Accessed: 29-Jun-2021].

[18] H. Gunes Kayacik, A. Nur Zincir-Heywood, and M. I. Heywood, "A hierarchical SOM-based intrusion detection system," *Eng. Appl. Artif. Intell.*, vol. 20, no. 4, pp. 439– 451, 2007.

[19] R. Sahani, Shatabdinalini, C. Rout, J. Chandrakanta Badajena, A. K. Jena, and H. Das,

"Classification of intrusion detection using data mining techniques," in *Advances in Intelligent Systems and Computing*, Singapore: Springer Singapore, 2018, pp. 753–764.

[20] Kurniabudi, D. Stiawan, Darmawijoyo, M. Y. Bin Idris, A. M. Bamhdi, and R. Budiarto,

"CICIDS-2017 dataset feature analysis with information gain for anomaly detection," *IEEE Access*, vol. 8, pp. 132911–132921, 2020.

[21] R. Abdulhammed, H. Musafer, A. Alessa, M. Faezipour, and A. Abuzneid, "Features dimensionality reduction approaches for Machine Learning based network intrusion detection," *Electronics* (Basel), vol. 8, no. 3, p. 322, 2019.

[22] "sklearn.preprocessing.MinMaxScaler — scikit-learn 0.24.2 documentation," Scikit-learn.org. [Online]. Available: https://scikit learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. [Accessed: 29-Jun-2021].

[23] "Using Under-Sampling Techniques for Extremely Imbalanced Data", *Medium*, 2021. [Online]. Available: https://medium.com/dataman-in-ai/sampling-techniques-for-extremely-imbalanced-data-part-i-under-sampling-a8dbc3d8d6d8. [Accessed: 28-Jun-2021].

[24] "sklearn.preprocessing.LabelEncoder—scikit-learn 0.24.2 documentation," Scikit-learn.org, 2021. [Online].Available:https://scikitlearn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html. [Accessed: 28- Jun- 2021].

[25] T. Mahjabin, Y. Xiao, G. Sun and W. Jiang, "A survey of distributed denial-of-service attack, prevention, and mitigation techniques," *International Journal of Distributed Sensor Networks*, vol. 13, no. 12, p. 155014771774146, 2017. Available: 10.1177/1550147717741463.

[26] K. N. Mallikarjunan, K. Muthupriya, and S. M. Shalinie, "A survey of distributed denial of service attack," in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, 2016.

[27] "Top 4 famous DDoS attacks in history," Indusface.com, 03-Sep-2019. [Online]. Available: https://www.indusface.com/blog/famous-ddos-attacks/. [Accessed: 30-Jun-2021].

[28] S. Loukas, "Multi-class Classification: Extracting performance metrics from the confusion matrix," *Towards Data Science*, 19-Jun-2020. [Online]. Available:

https://towardsdatascience.com/multi-class-classification-extracting-performance-metrics-from-the-confusion-matrix-b379b427a872. [Accessed: 30-Jun-2021].

[29] S. Narkhede, "Understanding AUC - ROC Curve - Towards Data Science," *Towards Data Science,* 26-Jun-2018. [Online]. Available: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5. [Accessed: 30-Jun-2021].

[30] L. Babarinde, Patrick Wall, and G. McKeever, "DDoS Attack Types & Mitigation Methods," Imperva.com. [Online]. Available: https://www.imperva.com/learn/ddos/ddos-attacks/. [Accessed: 30-Jun-2021].

[31] S. Sun, "A survey of multi-view machine learning," *Neural Comput. Appl.*, vol. 23, no.
7–8, pp. 2031–2038, 2013.

[32] H. Rajadurai and U. D. Gandhi, "A stacked ensemble learning model for intrusion detection in wireless network," *Neural Comput. Appl.*, 2020.

[33] "Chi-square statistic: How to calculate it / distribution," Statisticshowto.com, 04-Dec-2020. [Online]. Available: https://www.statisticshowto.com/probability-and-statistics/chi-square/. [Accessed: 30-Jun-2021].

[34] "Python - Pearson correlation test between two variables - GeeksforGeeks," Geeksforgeeks.org, 06-Apr-2020. [Online]. Available: https://www.geeksforgeeks.org/python-pearson-correlation-test-between-two-variables/. [Accessed: 30-Jun-2021].

[35] Procrastinator, "How to dealing with imbalanced classes in machine learning," Analyticsvidhya.com, 06-Oct-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/. [Accessed: 30-Jun-2021].

[36] R. Mohammed, J. Rawashdeh, and M. Abdullah, "Machine learning with oversampling and undersampling techniques: Overview study and experimental results," in *2020*
11th International Conference on Information and Communication Systems (ICICS), 2020.

[37] Aman, “Feature selection techniques in machine learning,” Analyticsvidhya.com, 10-Oct-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/. [Accessed: 30-Jun-2021].

[38] H. Heidenreich, “What are the types of machine learning? - Towards Data Science,” TowardsDataScience,04-Dec-2018.[Online].Available: https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f. [Accessed: 30-Jun-2021].

[39] O. F.y et al., “Supervised machine learning algorithms: Classification and comparison,” *Int. J. Comput. Trends Technol.*, vol. 48, no. 3, pp. 128–138, 2017.

[40] N. Bhati and M. Khari, "A new ensemble based approach for intrusion detection system using voting,” *Journal of Intelligent & Fuzzy Systems*, pp. 1-11, 2021. Available: 10.3233/jifs-189764.

[41] G. Varoquaux, L. Buitinck, G. Louppe, O. Grisel, F. Pedregosa, and A. Mueller, “Scikit-learn: Machine learning without learning the machinery,” *GetMob. Mob. Comput. Commun.*, vol. 19, no. 1, pp. 29–33, 2015.

Appendix A

This section will provide detailed results about each step that has been done in this thesis. Appendix A has been divided into three sections. Section A.1 explained the results for each supervised machine learning method before and after applying feature selection. The result after each feature selection method has been compared to select the best results. Section A.2 illustrated the results for the NSL-KDD dataset after applying the correlation method and applying correlation with the Chi-2 method. This section also shows the performance for each supervised method along with each ensemble method by using the ROC-AUC curve. The same order has been done for section A.3. This section focused on the results of the CICIDS-2017 dataset in the same order as the previous section.

A.1 Feature Selection Detailed Results

Table A.1 Feature Selection Comparisons for NSL-KDD

ML model	Before			correlation			Chi-2		
	Acc	**FPR**	**Time**	**Acc**	**FPR**	**Time**	**Acc**	**FPR**	**Time**
LR	0.99	0.0003	11.52	0.99	0.0008	8.39	0.98	0.003	8.27
DT	0.72	0.20	0.53	0.72	0.02	0.43	0.99	0.0008	0.41
RF	0.99	0.0001	8.15	0.99	0.0001	8.02	0.99	0.0001	8.28
NB	0.95	0.010	0.38	0.95	0.010	0.30	0.93	0.006	0.14

Table A.2 Feature Selection Comparisons for CICIDS2017

ML model	Before			correlation			Chi-2		
	Acc	**FPR**	**Time**	**Acc**	**FPR**	**Time**	**Acc**	**FPR**	**Time**
LR	0.98	0.0027	269.0	0.98	0.0027	255.6	0.91	0.017	128.09
DT	0.99	0.0002	34.09	0.99	0.0001	29.12	0.75	0.05	15.64
RF	0.99	0.0001	309.12	0.99	0.0001	302.1	0.80	0.033	284.4
NB	0.79	0.037	2.87	0.79	0.037	2.56	0.54	0.084	0.862

A.2 NSL-KDD Dataset Results

Table A.3 Supervised Model performance after applying Correlation based feature selection – NSL-KDD

Class	LR			DT			RF			NB		
	Precision	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**	**Precision**	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**
Apache	0.96	0.98	0.97	0.98	0.63	0.76	1.00	0.99	0.99	0.95	0.99	0.97
Back	0.99	1.00	1.00	1.00	0.99	0.99	1.00	1.00	1.00	1.00	0.99	0.99
Land	0.80	0.80	0.80	0.00	0.00	0.00	1.00	0.80	0.89	0.00	0.00	0.00
Mailbomb	0.94	0.70	0.80	0.97	0.92	0.94	1.00	1.00	1.00	0.00	0.00	0.00
Neptune	1.00	1.00	1.00	1.00	0.22	0.37	1.00	1.00	1.00	0.98	1.00	0.99
Normal	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.94	0.99	0.96
Pod	0.92	0.98	0.95	1.00	1.00	1.00	0.98	1.00	0.99	0.00	0.00	0.00
Processtable	0.92	0.92	0.92	0.02	0.99	0.04	1.00	1.00	1.00	0.97	1.00	0.98
Smurf	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00
teardrop	0.96	1.00	0.98	1.00	0.99	1.00	1.00	0.99	1.00	0.00	0.00	0.00

Table A.4 Ensemble Classifier after applying Correlation Based Feature Selection - NSL-KDD

Class	Voting			Stacking		
	Precision	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**
Apache	1.00	0.99	0.99	1.00	0.99	1.00
Back	1.00	1.00	1.00	1.00	1.00	1.00
Land	0.75	0.60	0.67	1.00	0.80	0.89
Mailbomb	1.00	1.00	1.00	1.00	1.00	1.00
Neptune	1.00	1.00	1.00	1.00	1.00	1.00
Normal	1.00	1.00	1.00	1.00	1.00	1.00
Pod	0.98	0.98	0.98	0.98	1.00	0.99
Processtable	0.98	1.00	0.99	1.00	1.00	1.00
Smurf	1.00	1.00	1.00	1.00	1.00	1.00
teardrop	0.97	0.99	0.98	1.00	0.99	1.00

Table A.5 Supervised models and Ensemble Classifier Performance – NSL-KDD

Class	Accuracy	FPR	ROC-AUC (Macro)	ROC-AUC (Micro)	Time
LR	0.99	0.0008	1.00	1.00	8.29
DT	0.72	0.028	0.87	0.85	0.41
RF	1.00	0.0001	1.00	1.00	6.78
NB	0.95	0.010	0.98	0.74	0.21
Stack	1.00	0.0001	1.00	1.00	66.32
Voting	1.00	0.0002	1.00	1.00	14.87

Table A.6 Supervised Model performance after applying Correlation and Chi-2 based feature selection – NSL-KDD

Class	LR			DT			RF			NB		
	Precision	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**	**Precision**	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**
Apache	0.97	0.93	0.95	1.00	0.99	0.99	1.00	0.99	0.99	0.91	0.99	0.95
Back	0.98	1.00	0.99	0.88	0.08	0.14	1.00	1.00	1.00	1.00	0.98	0.99
Land	0.60	0.60	0.60	0.50	0.80	0.62	0.80	0.80	0.80	0.03	1.00	0.07
Mailbomb	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00
Neptune	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99
Normal	0.98	0.99	0.98	0.98	1.00	0.99	1.00	1.00	1.00	0.99	0.94	0.96
Pod	0.91	0.66	0.77	0.94	0.98	0.96	0.94	0.98	0.96	0.94	0.98	0.96
Processtable	0.95	0.74	0.83	1.00	0.98	0.99	1.00	1.00	1.00	0.90	0.97	0.93
Smurf	0.76	0.75	0.76	1.00	1.00	1.00	1.00	1.00	1.00	0.44	1.00	0.62
teardrop	0.93	1.00	0.96	0.97	0.99	0.98	0.98	0.99	0.99	0.96	1.00	0.98

Table A.7 Ensemble Classifier after applying Correlation and Chi-2- NSL-KDD

Class	Voting			Stacking		
	Precision	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**
Apache	1.00	0.98	0.99	1.00	0.99	0.99
Back	1.00	1.00	1.00	1.00	1.00	1.00
Land	0.60	0.60	0.60	0.80	0.80	0.80
Mailbomb	1.00	0.97	0.98	1.00	1.00	1.00
Neptune	1.00	1.00	1.00	1.00	1.00	1.00
Normal	1.00	1.00	1.00	1.00	1.00	1.00
Pod	0.94	0.98	0.96	0.94	0.98	0.96
Processtable	0.98	0.97	0.98	1.00	0.99	1.00
Smurf	1.00	0.99	1.00	1.00	1.00	1.00
teardrop	0.97	0.99	0.98	0.98	0.99	0.99

Table A.8 Supervised models and Ensemble Classifier Performance– NSL-KDD

Class	Accuracy	FPR	ROC-AUC (Macro)	ROC-AUC (Micro)	Time
LR	0.97	0.004	1.00	1.00	5.76
DT	0.99	0.0003	0.89	0.99	0.27
RF	1.00	0.0001	1.00	1.00	6.98
NB	0.95	0.005	0.99	1.00	0.10
Stack	1.00	0.0002	0.99	1.00	56.6
Voting	0.99	0.0004	1.00	1.00	13.30

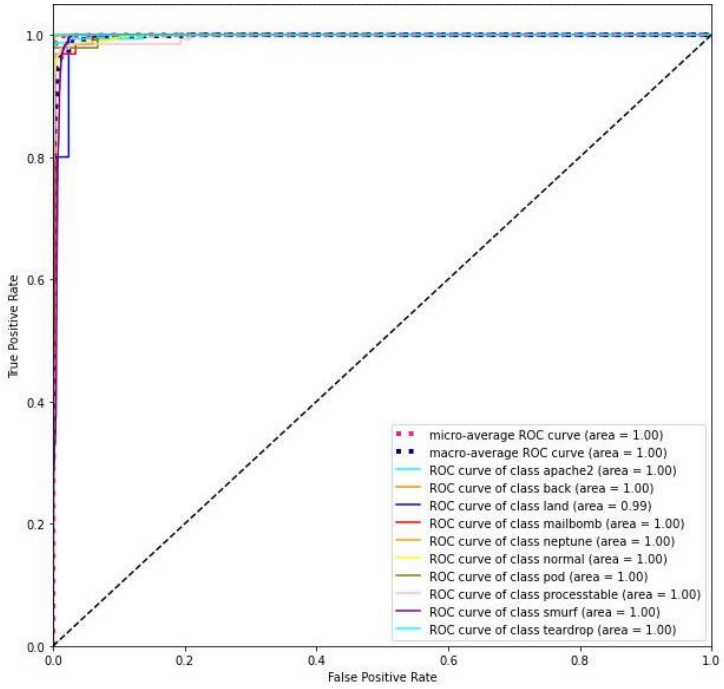

Figure A.1 ROC-AUC curve for Logistic Regression model – NSL-KDD

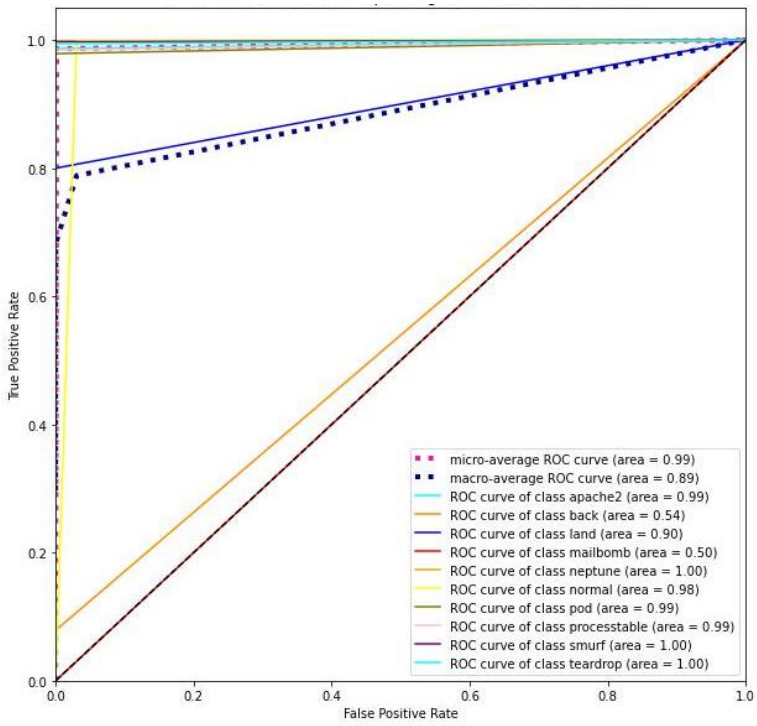

Figure A.2 ROC-AUC curve for Decision Tree model – NSL-KDD

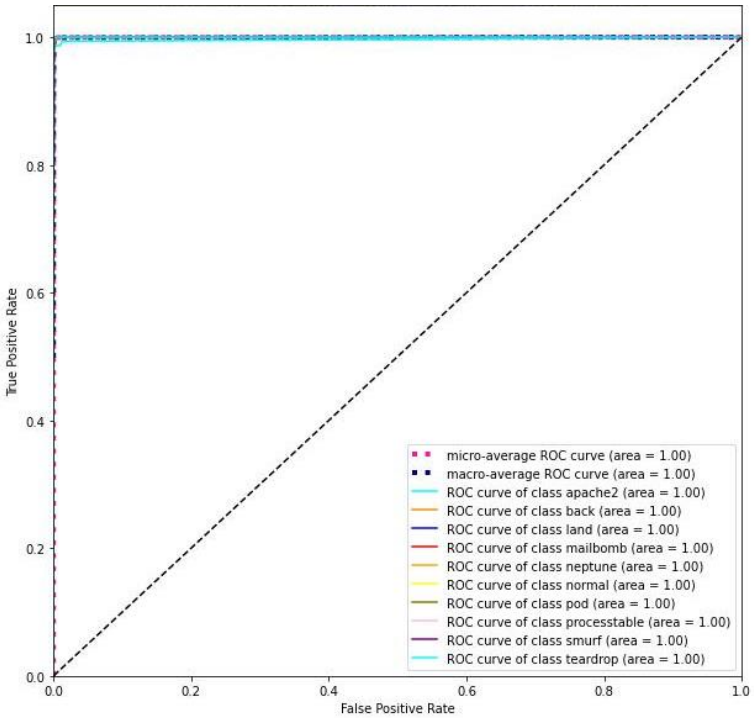

Figure A.3 ROC-AUC curve for Random Forest model – NSL-KDD

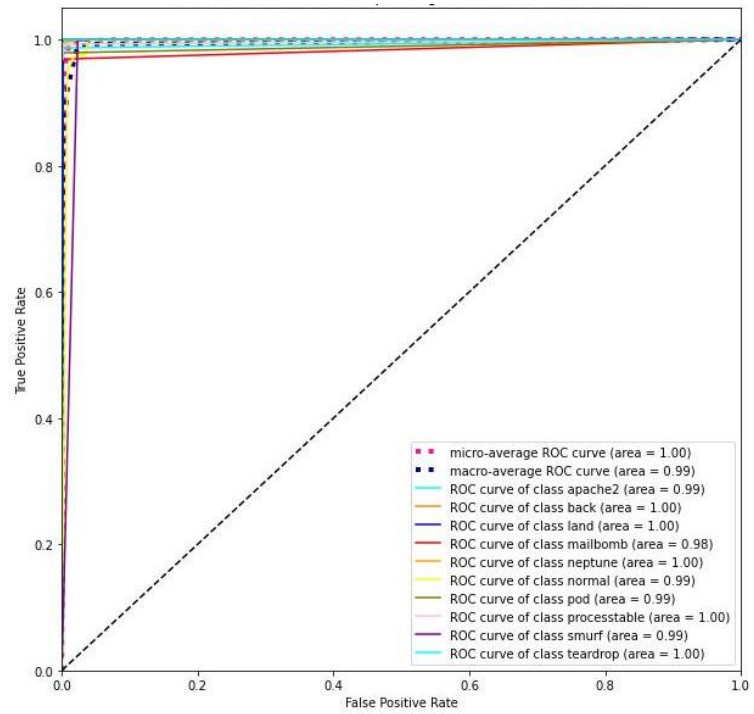

Figure A.4 ROC-AUC curve for Naive Bayes model – NSL-KDD

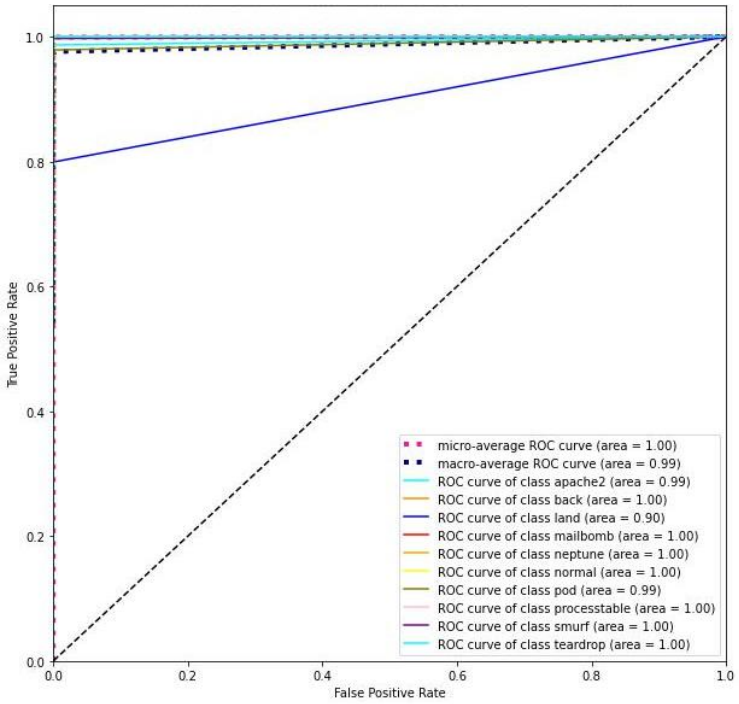

Figure A.5 ROC-AUC curve for Stacking Ensemble model – NSL-KDD

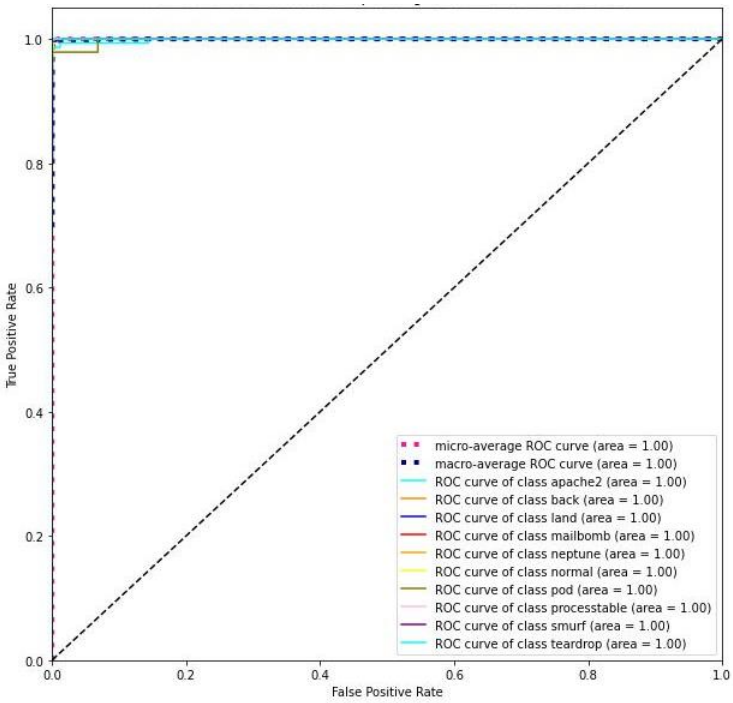

Figure A.6 ROC-AUC curve for Voting Ensemble model – NSL-KDD

A.3 CICIDS2017 Dataset Results

Table A.9 Supervised Model performance after applying Correlation based feature selection – CICIDS2017

Class	RF			DT			LR			NB		
	Precision	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**	**Precision**	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**
Benign	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.99	0.97	0.92	0.94
DoS Hulk	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00
DDoS	1.00	1.00	1.00	0.99	0.96	0.96	0.99	0.95	0.97	0.54	0.65	0.59
DoS GoldenEye	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.99	0.99	0.75	0.99	0.85
DoS Slowhttptest	1.00	0.99	0.99	0.99	0.99	0.99	0.95	0.96	0.96	0.06	0.66	0.11
DoS slowloris	0.99	1.00	0.99	0.99	0.99	0.99	0.96	0.84	0.90	0.21	0.96	0.34
Heartbleed	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00

Table A.10 Ensemble Classifier after applying Correlation Based Feature Selection – CICIDS2017

Class	Voting			Stacking		
	Precision	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**
Benign	1.00	1.00	1.00	1.00	1.00	1.00
DoS Hulk	1.00	1.00	1.00	1.00	1.00	1.00
DDoS	1.00	0.97	0.98	1.00	0.99	1.00
DoS GoldenEye	1.00	1.00	1.00	1.00	1.00	1.00
DoS Slowhttptest	0.99	0.99	0.99	1.00	0.99	0.99
DoS slowloris	0.99	1.00	0.99	0.99	1.00	1.00
Heartbleed	1.00	1.00	1.00	1.00	1.00	1.00

Table A.11 Supervised models and Ensemble Classifier Performance – CICIDS2017

Class	Accuracy	FPR	ROC-AUC (Macro)	ROC-AUC (Micro)	Time
LR	0.99	0.0027	1.00	1.00	255.6
DT	1.00	0.0001	1.00	1.00	32.67
RF	1.00	0.0001	1.00	1.00	305.19
NB	0.79	0.037	0.88	0.83	2.733
Stack	1.00	0.0001	1.00	1.00	5.45
Voting	0.99	0.0002	1.00	1.00	3.02

Table A.12 Supervised Model performance after applying Correlation and Chi-2 based feature selection – CICIDS2017

Class	RF			DT			LR			NB		
	Precision	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**	**Precision**	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**
Benign	1.00	1.00	1.00	0.82	1.00	0.90	0.97	0.93	0.95	0.99	0.53	0.69
DoS Hulk	1.00	1.00	1.00	0.99	1.00	0.99	0.98	0.99	0.98	0.36	0.99	0.53
DDoS	0.99	0.99	0.99	1.00	0.98	0.99	0.34	0.66	0.45	0.07	0.13	0.09
DoS GoldenEye	1.00	1.00	1.00	0.99	0.58	0.74	0.95	0.95	0.95	0.92	0.68	0.78
DoS Slowhttptest	0.98	0.94	0.96	0.99	0.88	0.93	0.55	0.73	0.63	0.27	0.56	0.37
DoS slowloris	1.00	0.97	0.98	0.99	0.88	0.93	0.30	0.51	0.37	0.13	0.81	0.23
Heartbleed	1.00	1.00	1.00	1.00	1.00	1.00	0.20	0.50	0.29	1.00	1.00	1.00

Table A.13 Ensemble Classifier after applying Correlation and Chi-2 – CICIDS2017

Class	Voting			Stacking		
	Precision	**Recall**	**F- 1**	**Precision**	**Recall**	**F-1**
Benign	0.99	1.00	0.99	1.00	1.00	1.00
DoS Hulk	0.99	1.00	1.00	0.99	1.00	1.00
DDoS	1.00	0.98	0.99	0.99	0.99	0.99
DoS GoldenEye	1.00	0.98	0.99	1.00	1.00	1.00
DoS Slowhttptest	0.99	0.88	0.93	0.98	0.92	0.95
DoS slowloris	0.99	0.88	0.93	1.00	0.97	0.98
Heartbleed	1.00	1.00	1.00	1.00	1.00	1.00

Table A.14 Supervised models and Ensemble Classifier Performance after Correlation and Chi-2 – CICIDS2017

Class	Accuracy	FPR	ROC-AUC (Macro)	ROC-AUC (Micro)	Time
LR	0.94	0.012	0.98	0.99	73.29
DT	0.88	0.036	0.93	0.93	11.33
RF	1.00	0.0005	1.00	1.00	172.17
NB	0.64	0.058	0.93	0.93	0.58
Stack	0.99	0.0007	1.00	1.00	5.12
Voting	0.99	0.002	1.00	1.00	2.77

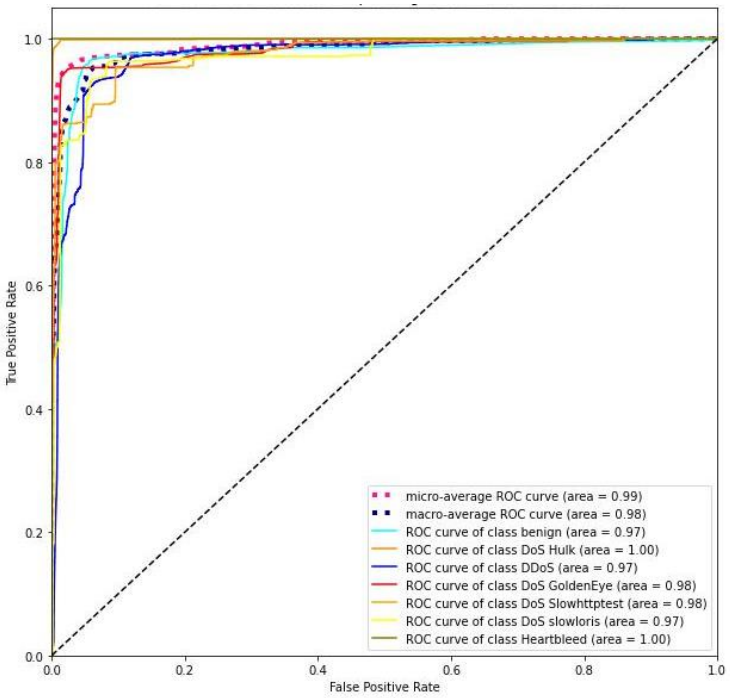

Figure A.7 ROC-AUC curve for Logistic Regression model – CICIDS2017

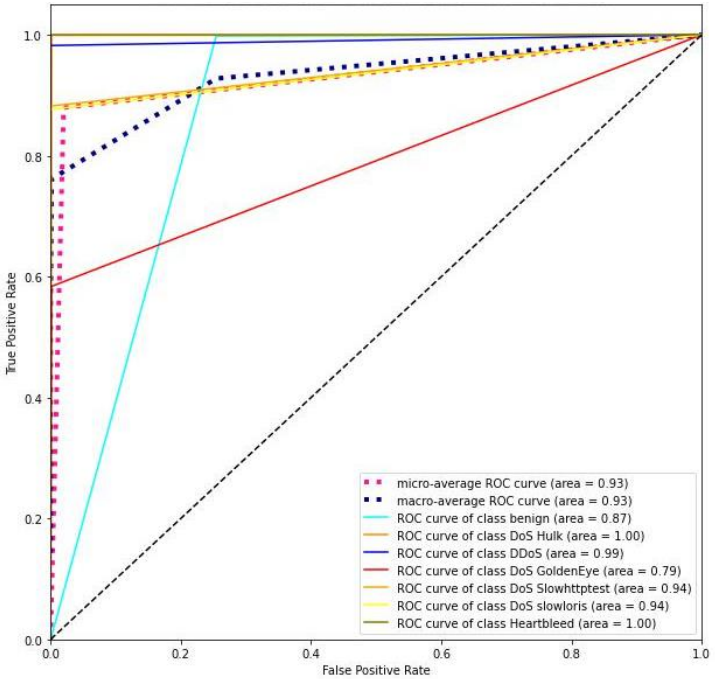

Figure A.8 ROC-AUC curve for Decision Tree model – CICIDS2017

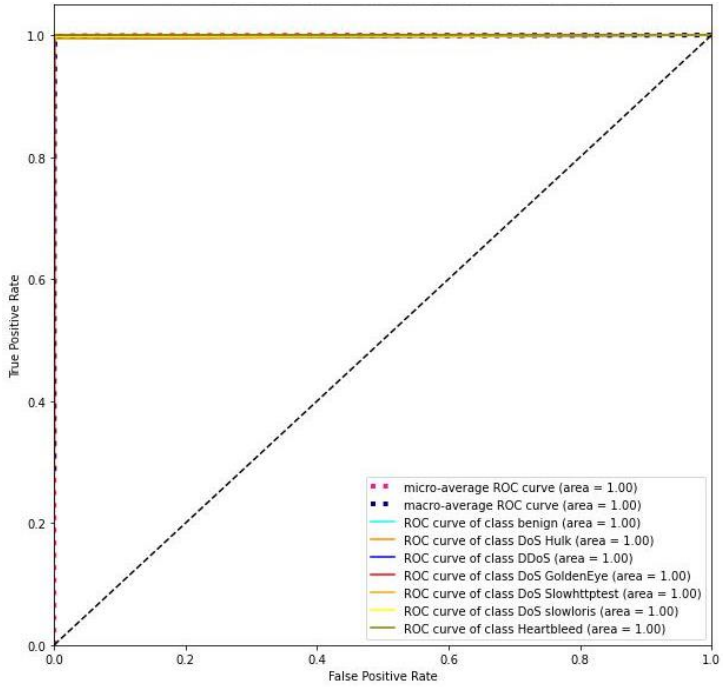

Figure A.9 ROC-AUC curve for Random Forest model – CICIDS2017

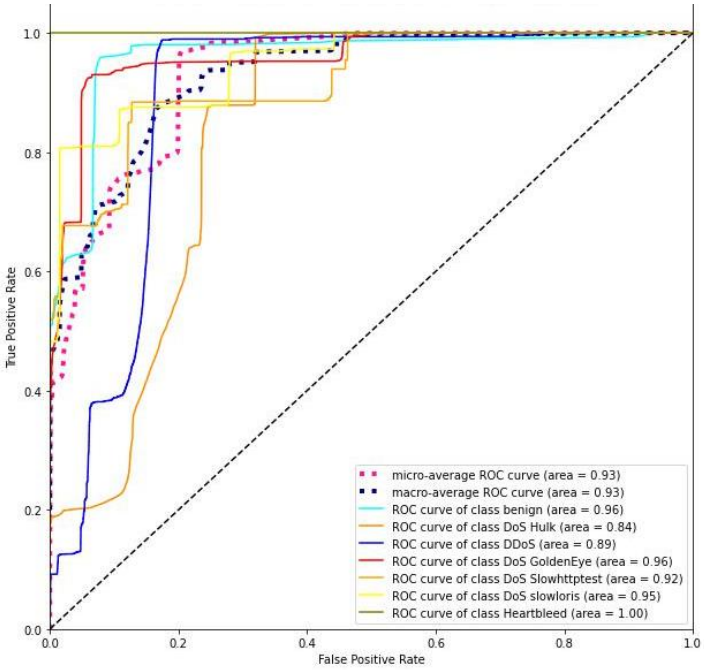

Figure A.10 ROC-AUC curve for Naive Bayes model – CICIDS2017

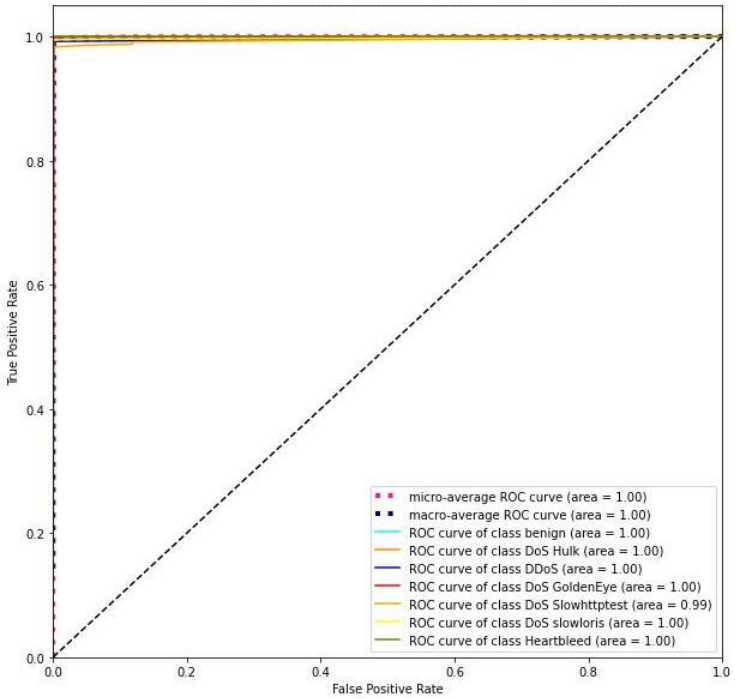

Figure A.9 ROC-AUC curve for Stacking Ensemble model – CICIDS2017

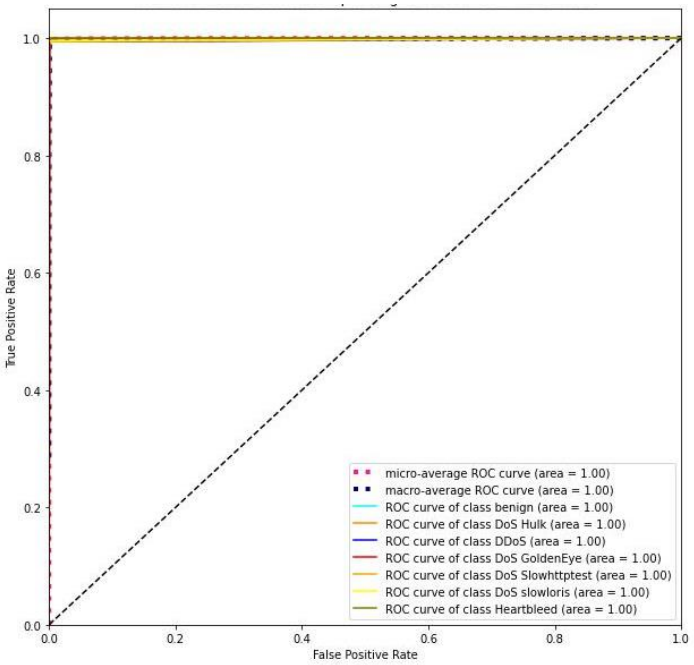

Figure A.10 ROC-AUC curve for Voting Ensemble model – CICIDS2017

Appendix B

Appendix B contains two main sections B.1 and B.2. In section B.1, the code that has been applied in the NSL-KDD dataset is illustrated along with its comments for more understanding. Each step is illustrated in the comments above its code with the symbol # before it. Same as in section B.2, the code that has been applied on CICIDS-2017 is explained with its comments.

B.1 NSL-KDD Dataset Detailed Codes

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, f1_score, precision_score,recall_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.preprocessing import MinMaxScaler
from itertools import cycle
from sklearn.ensemble import StackingClassifier
import time
import keras

#open NSL-KDD Dataset file
file_path = 'NSLKDD_Dos.xlsx'
dataset = pd.read_excel(file_path)
dataset.head()

#Set column names
columns = ([ ‘name of columns’])
dataset.columns = columns
```

```
#print class attack with their number of sample
print('Label distribution:')
dataset[' Label'].value_counts()

# divided dataset to x for data and y for target
X = dataset.iloc[:,0:44]
y = dataset.iloc[:, -4].values

#plot the number of samples for each class of attack
labels, counts = np.unique(y, return_counts=True)
plt.figure(figsize=(13, 8))
plt.bar(labels, counts, align='center')
plt.gca().set_xticks(labels)
plt.show()

#print the object feature to know which feature will convert to numeric
s = (dataset.dtypes == 'object')
object_cols = list(s[s].index)
print("Categorical variables:")
print(object_cols)

#convert non numeric to numeric value for all features
label_df = dataset.select_dtypes(include=['object']).copy()
encoder = preprocessing.LabelEncoder()
for col in columns:
    label_df[col] = encoder.fit_transform(dataset[col])
label_df.head(5)

#Re-divided the to x for data and y for target
X = label_df.iloc[:,0:43]
y = label_df.iloc[:, 3].values
X = X.drop([ 'attack'], axis=1)

#feature selection using chi2
bestfeatures = SelectKBest(score_func=chi2, k=2)
fit = bestfeatures.fit(X, y)
feat_importances = pd.Series(fit.scores_, index=X.columns)
topFatures = feat_importances.nlargest(25).copy().index.values
print("TOP 25 Features (Best to worst) :\n")
print(topFatures)

#create new x with the selected feature and pass it to the correlation
X = X[['dst_bytes','src_bytes', 'duration', 'srv_count' ,'count',
```

```
 'dst_host_srv_count', 'dst_host_srv_serror_rate' ,'dst_host_serror_rate',
 'serror_rate' ,'srv_serror_rate', 'same_srv_rate' ,'dst_host_same_srv_rate',
 'dst_host_same_src_port_rate' ,'dst_host_count', 'rerror_rate',
 'srv_rerror_rate' ,'dst_host_srv_rerror_rate', 'dst_host_rerror_rate',
 'srv_diff_host_rate', 'wrong_fragment', 'dst_host_srv_diff_host_rate',
 'diff_srv_rate' ,'service', 'dst_host_diff_srv_rate' ,'land']]

#Applying Correlation based feature selection
def identify_correlated(df, threshold):
    #A function to identify highly correlated features.
    # Compute correlation matrix with absolute values
    matrix = df.corr().abs()
    # Create a boolean mask
    mask = np.triu(np.ones_like(matrix, dtype=bool))
    # Subset the matrix
    reduced_matrix = matrix.mask(mask)
    # Find cols that meet the threshold
    to_drop = [c for c in reduced_matrix.columns if
               any(reduced_matrix[c] > threshold)]
    plt.figure(figsize=(15,15))
    sns.heatmap(matrix)
    plt.show()
    return to_drop
to_drop = identify_correlated(X, threshold=.9)
len(to_drop)
#the heatmap correlation is not good option for large feature

# Drop the high correlated cols and print the selected one
data_reduced = X.drop(to_drop, axis=1)
X = data_reduced
print(type(X))
X.head()

#spilt the dataset to train and test set
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=42 )
y_train_onehot = keras.utils.to_categorical(y_train)

#scale the data using MinMaxScaler / data normalization
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

### Under-sampling + adjusted weight
#print number of test set and train set for each class before applying Sampling methods
from collections import Counter
counter = Counter(y_train)
```

```
print(counter)
counter1 = Counter(y_test)
print(counter1)
labels = ['apache2' , 'back'  ,'mailbomb' , 'neptune' , 'normal' ,'pod' ,'processtable', 'smurf' ,
'teardrop']

#Applying UnderSampling Methods - NeighbourhoodCleaningRule
from imblearn.under_sampling import NeighbourhoodCleaningRule
undersample = NeighbourhoodCleaningRule(n_neighbors=10, threshold_cleaning=0.9)
x_train, y_train = undersample.fit_resample(x_train, y_train)

#Adjusting weight
from sklearn.utils import class_weight
class_weights = dict(enumerate(class_weight.compute_class_weight('balanced',
                                        np.unique(y_train),y_train)))
print(class_weights)


### evaluation metrix
#used to calculate the overall accuracy and precision for each class
def myEvaluation(y, y_pred):
    # calculate the overall acc
    correct_preds = np.sum(y == y_pred, axis=0)
    acc = correct_preds / y.shape[0]
    # calculate the precision for each class
    cm = confusion_matrix(y , y_pred)
    #print(cm)
    precision = np.array([])
    for i in range(y_train_onehot.shape[1]):
        first_part = cm[i][i]
        #print('i : ' ,i , 'first_part = ' , first_part)
        sec_part = 0
        for j in range(y_train_onehot.shape[1]):
            sec_part = sec_part + cm[j][i]
        result = first_part / sec_part
        #print('the presision of class', i , 'is = ' , result)
        precision = np.append(precision,result)
    # return acc and precision array
    return acc ,precision


'''This function is used from this website
https://towardsdatascience.com/multi-class-classification-extracting-
performance-metrics-from-the-confusion-matrix-b379b427a872
to evluate the models '''
```

```
from statistics import *
def met_eval(cnf_matrix):
    FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
    FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
    TP = np.diag(cnf_matrix)
    TN = cnf_matrix.sum() - (FP + FN + TP)
    FP = FP.astype(float)
    FN = FN.astype(float)
    TP = TP.astype(float)
    TN = TN.astype(float)
    # Sensitivity, hit rate, recall, or true positive rate
    TPR = TP/(TP+FN)
    # Specificity or true negative rate
    PPV = TP/(TP+FP)
    # false positive rate
    FPR = FP/(FP+TN)
    # Overall accuracy for each class
    ACC = (TP+TN)/(TP+FP+FN+TN)
    print(labels)
    print('ACC : ', ACC)
    print('Sensitivity : ', TPR)
    print('FPR : ', mean(FPR))
    print('Precision :', PPV)


### ROC-AUC curve
y_test_onehot = keras.utils.to_categorical(y_test)
y = label_binarize(y_test_onehot, classes=[0, 1,2 ,3 ,4,5,6 ,7,8,9,10 ])
n_classes = y.shape[1]
def ROC(n_classes,y_test_onehot, pred_prob):
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_onehot[:, i], pred_prob[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])
    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test_onehot.ravel(),pred_prob.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
    return roc_auc,fpr,tpr

def plot_classes(n_classes , fpr ,tpr):
    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
```

```
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
    # Finally average it and compute AUC
    mean_tpr /= n_classes
    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])
    # Plot all ROC curves
    plt.figure(figsize=(10, 10))
    plt.plot(fpr["micro"], tpr["micro"],
             label='micro-average ROC curve (area = {0:0.2f})'
                   ''.format(roc_auc["micro"]),
             color='deeppink', linestyle=':', linewidth=4)

    plt.plot(fpr["macro"], tpr["macro"],
             label='macro-average ROC curve (area = {0:0.2f})'
                   ''.format(roc_auc["macro"]),
             color='navy', linestyle=':', linewidth=4)

    colors = cycle(['aqua', 'darkorange', 'blue', 'red', 'orange', 'yellow' , 'olive' , 'pink' , 'purple'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color,
                 label='ROC curve of class {0} (area = {1:0.2f})'
                 ''.format(labels[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Some extension of Receiver operating characteristic to multi-class')
    plt.legend(loc="lower right")
    plt.show()

###Supervised/stacking/voting --machine learning models
#Logistic Regression
LogReg_clf = LogisticRegression(solver= 'liblinear' , C= 10 )
start = time.time()
LogReg_clf.fit(x_train, y_train)
LogReg_pred = LogReg_clf.predict(x_test)
LogReg_pred_prob = LogReg_clf.predict_proba(x_test)
LogReg_pred_score = accuracy_score(LogReg_pred, y_test)
print('LogReg_pred_score :', LogReg_pred_score)
acc , precision= myEvaluation(y_test, LogReg_pred)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
```

```
print(classification_report(y_test, LogReg_pred))
cnf_matrix_lr = confusion_matrix(y_test, LogReg_pred)
class_names = labels
disp = plot_confusion_matrix(LogReg_clf, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_lr)
end = time.time()
print(end - start, 'seconds')
#print Roc accuracy for each class of attack
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, LogReg_pred_prob)
plot_classes(n_classes , fpr ,tpr)

#Decision Tree
DTree_clf = DecisionTreeClassifier(criterion="entropy" )
DTree_clf.fit(x_train, y_train)
start = time.time()
DTree_pred = DTree_clf.predict(x_test)
end = time.time/()
DTree_pred_prob = DTree_clf.predict_proba(x_test)
DTtree_pred_score = accuracy_score(DTree_pred, y_test)
print('DTtree_pred_score :', DTtree_pred_score)
acc , precision= myEvaluation(y_test, DTree_pred)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
print(classification_report(y_test, DTree_pred))
cnf_matrix_dt = confusion_matrix(y_test, DTree_pred)
class_names = labels
disp = plot_confusion_matrix(DTree_clf, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_dt)
print(end - start, 'seconds')
#Print Roc curve for decision tree
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, DTree_pred_prob)
plot_classes(n_classes , fpr ,tpr)

#Random Forest
RF_clf = RandomForestClassifier()
RF_clf.fit( x_train, y_train)
RF_pred = RF_clf.predict(x_test)
end = time.time()
RF_pred_prob = RF_clf.predict_proba(x_test)
RF_pred_score = accuracy_score(RF_pred, y_test)
```

```
print('RF_pred_score :', RF_pred_score)
acc , precision= myEvaluation(y_test, RF_pred)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
cnf_matrix_rf = confusion_matrix(y_test, RF_pred)
print(classification_report(y_test, RF_pred))
class_names = labels
disp = plot_confusion_matrix(RF_clf, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_rf)
print(end - start, 'seconds')
#Roc Curve for random forest results
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, RF_pred_prob)
plot_classes(n_classes , fpr ,tpr)

#Naive Bayes
NB_clf = GaussianNB()
NB_clf.fit( x_train, y_train)
start = time.time()
NB_pred = NB_clf.predict(x_test)
end = time.time()
NB_pred_prob = NB_clf.predict_proba(x_test)
NB_pred_score = accuracy_score(NB_pred, y_test)
print('NB_pred_score :', NB_pred_score)
acc , precision= myEvaluation(y_test, NB_pred)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
print(classification_report(y_test, NB_pred))
cnf_matrix_nb = confusion_matrix(y_test, NB_pred)
class_names = labels
disp = plot_confusion_matrix(NB_clf, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_nb)
print(end - start, 'seconds')
#Roc Curve for NB reslts
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, NB_pred_prob)
plot_classes(n_classes , fpr ,tpr)

# Stacking ensamble classifier
# define meta learner model
level1 = RandomForestClassifier()
# define the stacking ensemble
```

```
model = StackingClassifier(estimators=[('DTree', DTree_clf),('RF' , RF_clf)],
final_estimator=level1, cv=3)
# fit the model on all available data
model.fit(x_train, y_train)
start = time.time()
y_preds = model.predict(x_test)
end = time.time()
model_pred_prob = model.predict_proba(x_test)
stacking_pred_score = accuracy_score(y_preds, y_test)
print('voting_pred_score :', stacking_pred_score)
acc , precision= myEvaluation(y_test, y_preds)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
print(classification_report(y_test, y_preds))
cnf_matrix_vot = confusion_matrix(y_test, y_preds)
class_names = labels
disp = plot_confusion_matrix(model, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_vot)
print(end - start, 'seconds')
#Plot Roc Curve for stacking ensamble classifier
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, model_pred_prob)
plot_classes(n_classes , fpr ,tpr)

#Applying Voting ensamble classifier
voting_clf = VotingClassifier(estimators=[('DTree', DTree_clf), ('RF' , RF_clf)], voting='soft')
voting_clf.fit(x_train, y_train)
start = time.time()
preds = voting_clf.predict(x_test)
end = time.time()
voting_pred_prob = voting_clf.predict_proba(x_test)
voting_pred_score = accuracy_score(preds, y_test)
print('voting_pred_score :', voting_pred_score)
acc , precision= myEvaluation(y_test, preds)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
print(classification_report(y_test, preds))
cnf_matrix_vot = confusion_matrix(y_test, preds)
class_names = labels
disp = plot_confusion_matrix(voting_clf, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_vot)
```

```
print(end - start, 'seconds')
#Plot Roc curve for voting ensamble classifier
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, voting_pred_prob)
plot_classes(n_classes , fpr ,tpr)
```

B.2 CICIDS-2017 Detailed Codes

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.feature_selection import mutual_info_regression, mutual_info_classif
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, f1_score, precision_score,recall_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.preprocessing import MinMaxScaler
from itertools import cycle
from sklearn.ensemble import StackingClassifier
from sklearn.metrics import log_loss
import time
import keras

#open Dos Dataset file and DDos Dataset file
df1 = pd.read_csv("cicids/Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv")
df8=pd.read_csv("cicids/Wednesday-workingHours.pcap_ISCX.csv")

#Combined Both dataset with deleting BENIGN case from first dataset
data = df1.loc[df1[" Label"] != 'BENIGN']
dataset = pd.concat([df8,data]).reset_index(drop=True)

#print class attack with their number of sample
print('Label distribution:')
dataset[' Label'].value_counts()
```

```
# divided dataset to x for data and y for target
X = dataset.iloc[:,0:78]
y = dataset.iloc[:, -1].values



#plot the number of samples for each class of attack
labels, counts = np.unique(y, return_counts=True)
plt.figure(figsize=(13, 8))
plt.bar(labels, counts, align='center')
plt.gca().set_xticks(labels)
plt.show()

#print the object feature to know which feature will convert to numeric
s = (dataset.dtypes == 'object')
object_cols = list(s[s].index)
print("Categorical variables:")
print(object_cols)

#convert non numeric to numeric value for all features
label_df = dataset.select_dtypes(include=['object']).copy()
encoder = preprocessing.LabelEncoder()
for col in columns:
   label_df[col] = encoder.fit_transform(dataset[col])
label_df.head(5)

#Re-divided the to x for data and y for target
X = label_df.iloc[:,0:79]
y = label_df.iloc[:, 0].values
X = X.drop([ ' Label'], axis=1)

#feature selection using chi2
bestfeatures = SelectKBest(score_func=chi2, k=2)
fit = bestfeatures.fit(X, y)
feat_importances = pd.Series(fit.scores_, index=X.columns)
topFatures = feat_importances.nlargest(25).copy().index.values
print("TOP 25 Features (Best to worst) :\n")
print(topFatures)

#create new x with the selected feature and pass it to the correlation
X = X[[' Bwd Packets/s' ,' Flow IAT Mean' ,' Flow Duration', ' Flow IAT Std',
 'Flow Bytes/s' ,' Packet Length Std' ,' Packet Length Variance',
 ' Fwd IAT Mean' ,' Fwd IAT Std', ' Flow Packets/s' ,' Flow IAT Max',
 'Fwd Packets/s' ,' Fwd IAT Max' ,'Fwd IAT Total' ,' Bwd Packet Length Std',
 ' Bwd IAT Std' ,' Idle Min' ,' Packet Length Mean' ,' Average Packet Size',
 'Idle Mean', ' Bwd Packet Length Mean', ' Avg Bwd Segment Size',
 ' Bwd IAT Mean', ' Idle Max' ,'Bwd IAT Total']]
```

```
#Applying Correlation based feature selection
def identify_correlated(df, threshold):
    #A function to identify highly correlated features.
    # Compute correlation matrix with absolute values
    matrix = df.corr().abs()
    # Create a boolean mask
    mask = np.triu(np.ones_like(matrix, dtype=bool))
    # Subset the matrix
    reduced_matrix = matrix.mask(mask)
    # Find cols that meet the threshold
    to_drop = [c for c in reduced_matrix.columns if
               any(reduced_matrix[c] > threshold)]
    plt.figure(figsize=(15,15))
    sns.heatmap(matrix)
    plt.show()
    return to_drop
to_drop = identify_correlated(X, threshold=.9)
len(to_drop)
#the heatmap correlation is not good option for large feature

# Drop the high correlated cols and print the selected one
data_reduced = X.drop(to_drop, axis=1)
X = data_reduced
print(type(X))
X.head()

#spilt the dataset to train and test set
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=42 )
y_train_onehot = keras.utils.to_categorical(y_train)

#scale the data using MinMaxScaler / data normalization
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

### Under-sampling + adjusted weight
#print number of test set and train set for each class before applying Sampling methods
from collections import Counter
counter = Counter(y_train)
print(counter)
counter1 = Counter(y_test)
print(counter1)
labels = ['BENIGN' , 'DDoS' , 'DoS GoldenEye' , 'DoS slowloris' , 'DoS Slowhttptest']

#Applying UnderSampling Methods - NeighbourhoodCleaningRule
```

```
from imblearn.under_sampling import NeighbourhoodCleaningRule
undersample = NeighbourhoodCleaningRule(n_neighbors=10, threshold_cleaning=0.9)
x_train, y_train = undersample.fit_resample(x_train, y_train)

#Adjusting weight
from sklearn.utils import class_weight
class_weights = dict(enumerate(class_weight.compute_class_weight('balanced',
                                        np.unique(y_train),y_train)))
print(class_weights)


### evaluation metrix
#used to calculate the overall accuracy and precision for each class
def myEvaluation(y, y_pred):
    # calculate the overall acc
    correct_preds = np.sum(y == y_pred, axis=0)
    acc = correct_preds / y.shape[0]
    # calculate the precision for each class
    cm = confusion_matrix(y , y_pred)
    #print(cm)
    precision = np.array([])
    for i in range(y_train_onehot.shape[1]):
        first_part = cm[i][i]
        #print('i : ' ,i , 'first_part = ' , first_part)
        sec_part = 0
        for j in range(y_train_onehot.shape[1]):
            sec_part = sec_part + cm[j][i]
        result = first_part / sec_part
        #print('the presision of class', i , 'is = ' , result)
        precision = np.append(precision,result)
    # return acc and precision array
    return acc ,precision


'''This function is used from this website
https://towardsdatascience.com/multi-class-classification-extracting-
performance-metrics-from-the-confusion-matrix-b379b427a872
to evluate the models '''

from statistics import *
def met_eval(cnf_matrix):
    FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
    FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
    TP = np.diag(cnf_matrix)
    TN = cnf_matrix.sum() - (FP + FN + TP)
    FP = FP.astype(float)
```

```
    FN = FN.astype(float)
    TP = TP.astype(float)
    TN = TN.astype(float)
    # Sensitivity, hit rate, recall, or true positive rate
    TPR = TP/(TP+FN)
    # Specificity or true negative rate
    PPV = TP/(TP+FP)
    # false positive rate
    FPR = FP/(FP+TN)
    # Overall accuracy for each class
    ACC = (TP+TN)/(TP+FP+FN+TN)
    print(labels)
    print('ACC : ', ACC)
    print('Sensitivity : ', TPR)
    print('FPR : ', mean(FPR))
    print('Precision :', PPV)


### ROC-AUC curve
y_test_onehot = keras.utils.to_categorical(y_test)
y = label_binarize(y_test_onehot, classes=[0, 1,2 ,3 ,4,5,6  ])
n_classes = y.shape[1]
def ROC(n_classes,y_test_onehot, pred_prob):
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_onehot[:, i], pred_prob[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])
    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test_onehot.ravel(),pred_prob.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
    return roc_auc,fpr,tpr

def plot_classes(n_classes , fpr ,tpr):
    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
    # Finally average it and compute AUC
    mean_tpr /= n_classes
    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])
```

```
    # Plot all ROC curves
    plt.figure(figsize=(10, 10))
    plt.plot(fpr["micro"], tpr["micro"],
             label='micro-average ROC curve (area = {0:0.2f})'
                   ".format(roc_auc["micro"]),
             color='deeppink', linestyle=':', linewidth=4)

    plt.plot(fpr["macro"], tpr["macro"],
             label='macro-average ROC curve (area = {0:0.2f})'
                   ".format(roc_auc["macro"]),
             color='navy', linestyle=':', linewidth=4)

    colors = cycle(['aqua', 'darkorange', 'blue', 'red', 'orange', 'yellow' , 'olive' , 'pink' , 'purple'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color,
                 label='ROC curve of class {0} (area = {1:0.2f})'
                 ".format(labels[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Some extension of Receiver operating characteristic to multi-class')
    plt.legend(loc="lower right")
    plt.show()

###Supervised/stacking/voting --machine learning models
#Logistic Regression
LogReg_clf = LogisticRegression(solver= 'liblinear' , C= 10 )
start = time.time()
LogReg_clf.fit(x_train, y_train)
LogReg_pred = LogReg_clf.predict(x_test)
LogReg_pred_prob = LogReg_clf.predict_proba(x_test)
LogReg_pred_score = accuracy_score(LogReg_pred, y_test)
print('LogReg_pred_score :', LogReg_pred_score)
acc , precision= myEvaluation(y_test, LogReg_pred)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
print(classification_report(y_test, LogReg_pred))
cnf_matrix_lr = confusion_matrix(y_test, LogReg_pred)
class_names = labels
disp = plot_confusion_matrix(LogReg_clf, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
```

```
met_eval(cnf_matrix_lr)
end = time.time()
print(end - start, 'seconds')
#print Roc accuracy for each class of attack
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, LogReg_pred_prob)
plot_classes(n_classes , fpr ,tpr)

#Decision Tree
DTree_clf = DecisionTreeClassifier(criterion="entropy" )
DTree_clf.fit(x_train, y_train)
start = time.time()
DTree_pred = DTree_clf.predict(x_test)
end = time.time/()
DTree_pred_prob = DTree_clf.predict_proba(x_test)
DTtree_pred_score = accuracy_score(DTree_pred, y_test)
print('DTtree_pred_score :', DTtree_pred_score)
acc , precision= myEvaluation(y_test, DTree_pred)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
print(classification_report(y_test, DTree_pred))
cnf_matrix_dt = confusion_matrix(y_test, DTree_pred)
class_names = labels
disp = plot_confusion_matrix(DTree_clf, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_dt)
print(end - start, 'seconds')
#Print Roc curve for decision tree
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, DTree_pred_prob)
plot_classes(n_classes , fpr ,tpr)

#Random Forest
RF_clf = RandomForestClassifier()
RF_clf.fit( x_train, y_train)
RF_pred = RF_clf.predict(x_test)
end = time.time()
RF_pred_prob = RF_clf.predict_proba(x_test)
RF_pred_score = accuracy_score(RF_pred, y_test)
print('RF_pred_score :', RF_pred_score)
acc , precision= myEvaluation(y_test, RF_pred)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
cnf_matrix_rf = confusion_matrix(y_test, RF_pred)
print(classification_report(y_test, RF_pred))
class_names = labels
```

```
disp = plot_confusion_matrix(RF_clf, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_rf)
print(end - start, 'seconds')
#Roc Curve for random forest results
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, RF_pred_prob)
plot_classes(n_classes , fpr ,tpr)

#Naive Bayes
NB_clf = GaussianNB()
NB_clf.fit( x_train, y_train)
start = time.time()
NB_pred = NB_clf.predict(x_test)
end = time.time()
NB_pred_prob = NB_clf.predict_proba(x_test)
NB_pred_score = accuracy_score(NB_pred, y_test)
print('NB_pred_score :', NB_pred_score)
acc , precision= myEvaluation(y_test, NB_pred)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
print(classification_report(y_test, NB_pred))
cnf_matrix_nb = confusion_matrix(y_test, NB_pred)
class_names = labels
disp = plot_confusion_matrix(NB_clf, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_nb)
print(end - start, 'seconds')
#Roc Curve for NB reslts
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, NB_pred_prob)
plot_classes(n_classes , fpr ,tpr)

# Stacking ensamble classifier
# define meta learner model
level1 = RandomForestClassifier()
# define the stacking ensemble
model = StackingClassifier(estimators=[('DTree', DTree_clf),('RF' , RF_clf)],
final_estimator=level1, cv=3)
# fit the model on all available data
model.fit(x_train, y_train)
start = time.time()
y_preds = model.predict(x_test)
end = time.time()
```

```
model_pred_prob = model.predict_proba(x_test)
stacking_pred_score = accuracy_score(y_preds, y_test)
print('voting_pred_score :', stacking_pred_score)
acc , precision= myEvaluation(y_test, y_preds)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
print(classification_report(y_test, y_preds))
cnf_matrix_vot = confusion_matrix(y_test, y_preds)
class_names = labels
disp = plot_confusion_matrix(model, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_vot)
print(end - start, 'seconds')
#Plot Roc Curve for stacking ensamble classifier
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, model_pred_prob)
plot_classes(n_classes , fpr ,tpr)

#Applying Voting ensamble classifier
voting_clf = VotingClassifier(estimators=[('DTree', DTree_clf), ('RF' , RF_clf)], voting='soft')
voting_clf.fit(x_train, y_train)
start = time.time()
preds = voting_clf.predict(x_test)
end = time.time()
voting_pred_prob = voting_clf.predict_proba(x_test)
voting_pred_score = accuracy_score(preds, y_test)
print('voting_pred_score :', voting_pred_score)
acc , precision= myEvaluation(y_test, preds)
print('my accuracy:    {:.2f}'.format(acc))
print('my precision:', precision)
print(classification_report(y_test, preds))
cnf_matrix_vot = confusion_matrix(y_test, preds)
class_names = labels
disp = plot_confusion_matrix(voting_clf, x_test, y_test,
                             display_labels=class_names,
                             cmap=plt.cm.Blues)
plt.show()
met_eval(cnf_matrix_vot)
print(end - start, 'seconds')
#Plot Roc curve for voting ensamble classifier
roc_auc,fpr,tpr = ROC(n_classes,y_test_onehot, voting_pred_prob)
plot_classes(n_classes , fpr ,tpr)
```