Data Science Meets Cybersecurity

by Kelvin Simatwo

Student ID: F134712

Loughborough University

21COP328: Data Science Project

Supervisor: Dr Asma Adnane

SUBMITTED ON 26th AUGUST 2022

Abstract

Preventing cyber attacks from happening has become a priority for many companies and industries today. Data science techniques such as machine learning (ML) have revolutionized the detection of cyber attacks, especially in anomaly detection. Finding the best dataset and ML algorithm to use in anomaly detection has been a research area explored by many authors. Most authors find different results depending on the type of the dataset and whether ML or Deep Learning (DL) techniques were used. Several authors use standard evaluation metrics such as accuracy, recall, precision and F1 scores to evaluate ML models, but fail to include model training and testing times as metrics. The main aim of this project is to conduct a comparison study across several datasets and several ML algorithms, while using the standard evaluation metrics as well as model training and testing times to evaluate ML models and fill in the gap in model evaluation. In this project, the Decision Tree classifier was found to perform best in multi-class datasets while the Naive Bayes classifier was found to perform best in binary datasets.

Contents

List of Figures **v**

List of Tables **1**

1 Introduction **2**

1.1 Overview 2

1.2 Project aim and objectives 4

2 Literature Review **5**

2.1 Introduction 5

2.1.1 Cyber Attacks 5

2.1.2 Anomaly Detection 6

2.2 Machine Learning in Cyber Security 7

2.2.1 Datasets 7

2.2.2 Machine Learning Algorithms 9

2.3 Gaps 10

3 Methodology **11**

3.1 Introduction 11

3.2 Datasets 11

3.2.1 NSL-KDD Dataset 11

3.2.2 CICIDS2017 Dataset 13

3.2.3 CICDDoS2019 Dataset 14
3.3 Data Cleaning 15
3.4 Data Normalization and Feature Engineering 19
3.5 Model Training and Testing 22
3.5.1 CatBoost Model 22
3.5.2 KNN Model 23
3.5.3 Stochastic Gradient Descent Model 25
3.5.4 SVM Model 25
3.5.5 AdaBoost Model 26
3.5.6 Decision Trees Model 27
3.5.7 Random Forests Model 28
3.5.8 Naive Bayes Model 29
3.5.9 MLP Model 31
3.5.10 Model Fitting and Evaluation 32
3.6 Coding Details 35
3.7 Summary 35

4 Results **37**
4.1 Introduction 37
4.2 CICIDS2017 37
4.2.1 Evaluation Metrics 37
4.2.2 Confusion Matrices 39
4.2.3 Observations 39
4.3 CICDDoS2019 41
4.3.1 Evaluation metrics 42
4.3.2 Confusion Matrices 43
4.3.3 Observations 43
4.4 NSL-KDD 46

4.4.1 Evaluation Metrics . . . 46
4.4.2 Confusion Matrices and ROC Curves . . . 49
4.4.3 Observations . . . 49

5 Discussion **58**

5.1 Machine Learning Algorithms . . . 58
5.2 Datasets . . . 60

6 Conclusion **62**

Bibliography **64**

List of Figures

2.1 Types of Cyber Attacks According to Sun et al. [1] 6

2.2 Types of Cyber Attacks According to Note and Ali [2] 6

3.1 Diagram representing the methodology process 12

3.2 Diagram representing the attacks in CICDDoS2019 14

3.3 Attacks across NSL-KDD, CICIDS2017 and CICDDoS2019 Datasets. 16

3.4 Attack Categories across CICIDS2017 and CICDDoS2019 Datasets. 18

3.5 Attack categories across NSL-KDD, CICIDS2017 and CICDDoS2019 Datasets after sampling. 19

3.6 Feature Engineering. 21

3.7 Finding the best value of K 24

3.8 Comparison of kernel accuracies in the SVM models across datasets 26

3.9 Decision Tree 27

3.10 Comparison of accuracies while varying tree depth in the Decision Tree models across datasets 28

3.11 Comparison of accuracies while varying maximum depth in the Random Forest models across datasets 30

3.12 A Multilayer Perceptron Network with two hidden layers. 32

3.13 Confusion Matrix Structure 34

4.1 Confusion Matrices for the CICIDS2017 Machine Learning Algorithms . . . 40

4.2 Confusion Matrices for the CICIDS2017 Machine Learning Algorithms . . . 41
4.3 Confusion Matrices for the CICDDoS2019 Machine Learning Algorithms . . 44
4.4 Confusion Matrices for the CICDDoS2019 Machine Learning Algorithms . . 45
4.5 Confusion Matrices for the NSL-KDD ML Algorithms (KDDTrain+.TXT) . 50
4.6 Confusion Matrices for the NSL-KDD ML Algorithms (KDDTrain+.TXT) . 51
4.7 Confusion Matrices for the NSL-KDD ML Algorithms (KDDTest+.TXT) . . 52
4.8 Confusion Matrices for the NSL-KDD ML Algorithms (KDDTest+.TXT) . . 53
4.9 ROC Curves for the NSL-KDD ML Algorithms (KDDTrain+.TXT) 54
4.10 ROC Curves for the NSL-KDD ML Algorithms (KDDTrain+.TXT) 55
4.11 ROC Curves for the NSL-KDD ML Algorithms (KDDTest+.TXT) 56
4.12 ROC Curves for the NSL-KDD ML Algorithms (KDDTest+.TXT) 57

List of Tables

2.1 Comparison of Datasets Used Across Different Papers 8

2.2 Comparison of Algorithms and Metrics Used Across Different Papers 9

3.1 NSL-KDD Attack Categories [3] 13

3.2 CICIDS2017 Labels [4] 13

3.3 CICDDoS Attack Examples [5] 15

3.4 Attack Classes in Each Dataset 17

4.1 Model training duration for CICIDS2017 dataset 38

4.2 Model testing duration for CICIDS2017 dataset 38

4.3 Model evaluation scores for CICIDS2017 dataset 39

4.4 Model training duration for CICDDoS2019 dataset 42

4.5 Model testing duration for CICDDoS2019 dataset 42

4.6 Evaluation metrics for CICDDoS2019 dataset 43

4.7 Model training duration for NSL-KDD dataset (KDDTrain+.TXT) 46

4.8 Model testing duration for NSL-KDD dataset (KDDTrain+.TXT) 47

4.9 Model testing duration for NSL-KDD dataset (KDDTest+.TXT) 47

4.10 Evaluation Metrics for NSL-KDD dataset (KDDTrain+.TXT) 48

4.11 Evaluation Metrics for NSL-KDD dataset (KDDTest+.TXT) 48

Chapter 1

Introduction

1.1 Overview

Cyber attacks have been increasing exponentially in recent years [6]. This is most likely due to the increased pace at which technology is evolving. Most industries today are adopting digitalization and there is an increased dependence on the Internet of Things (IoT) in today's world [6]. Consequently, cyber attacks such as zero day attacks, denial of service, unauthorized access, malware attacks, among others, are ever on the increase. To combat cyber attacks, cyber security has been in place since the 1970s. Cybersecurity can be defined as a set of technologies and processes used to protect computer systems, networks, programs and data from cyber attacks. Cybersecurity aims at preserving the confidentiality, integrity and availability of information linked with computer systems [6].

Traditionally, cyber security defence strategies such as firewalls, anti-viruses and user authentication have been effective to an extent. However, Sarker et al. [6] remarks that these defence strategies are not effective with regard to today's need in cyber security. Intrusion detection resolves the issues in traditional cyber security strategies by analyzing important cyber security data in several key points along a computer network or system [6]. An intrusion detection system (IDS) is a device or software that identifies unauthorized use, alteration or destruction of computer systems [7]. Based on methodologies, IDSs can be categorized as either signature-based or anomaly-based [6]. A signature-based IDS uses predefined rules or patterns to identify an attack, limiting it to known attacks only [6]. On the other hand, anomaly-based IDSs model normal system behaviour, and identify deviations from the normal behaviour as potential cyber attacks [7].

Data science methodologies such as machine learning (ML) have found numerous applications in cyber security, particularly in anomaly-detection. This is because ML algorithms can be trained to identify normal behaviour in computer systems and can therefore identify anomalies as cyber attacks. Several cybersecurity datasets have been made to facilitate the anomaly detection process. Most of these datasets are created using network traffic generators that simulate both normal and attack traffic. Examples of these datasets are the KDD Cup 99, NSL-KDD, CICIDS2017, CICDDoS2019, among others.

The main challenge facing anomaly detection is the high rate of false alarms [7]. Some anomalies may not be cyber attacks, and it is important that ML models can distinguish between benign anomalies and malicious anomalies. The main research problem to be tackled in this paper involves finding out the best ML algorithm and the best dataset that can be used to reduce false alarm rates and ensure high accuracies in cyber attack detection. Deep learning (DL) algorithms such as Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) have provided high accuracies in anomaly detection [8]. However, graphics processing units (GPUs) have been used mostly with DL algorithms to increase their speed, thus causing DL algorithms to have a high computational cost [8][9]. Therefore, this project's scope will focus on classic ML algorithms that have lower computational costs, as well as one deep learning algorithm (multi-layer perceptron) that also has a lower computational cost.

1.2 Project aim and objectives

The main aim of this project is to conduct a comparison study of different machine learning algorithms using several datasets and evaluation metrics. This study will help determine the best algorithm and dataset for use in anomaly detection in a cyber security context.

The best ML algorithm would need to be efficient in terms of speed and accuracy when predicting cyber attacks. The following are therefore the objectives of this project:

- The main objective of the project is to investigate which ML algorithm and dataset is the best for the detection of cyber attacks. Using several cyber security datasets from the Canadian Institute of Cyber Security, different machine learning algorithms will be trained, tested and evaluated on how best they detect cyber attacks. The algorithms will be compared against each other using several evaluation metrics such as accuracy, recall, precision, F1 and area under the curve (AUC) scores.

- The second objective of the project will be to investigate which features are the greatest indicator of cyber attacks. This will be investigated through feature engineering.

- The final objective of the project will be to investigate which machine learning model has the best computational efficiency. This will be done by measuring the training and testing times of the models and comparing them to each other.

Chapter 2

Literature Review

2.1 Introduction

The research question to be investigated is 'What is the best ML algorithm and dataset for anomaly detection in a cyber security context, with regards to speed and accuracy?'. In order to answer this question, it is first essential to understand what anomaly detection in a cyber security context is.

2.1.1 Cyber Attacks

According to Note and Ali [2], cyber attacks are attacks within the system of a computer made to jeopardize the confidentiality, integrity and availability of the data within the computer. Sarker et al. [6] agrees with this definition by elaborating that cyber security is a set of defense strategies that preserve the confidentiality, integrity and availability of information within computer systems. The confidentiality of a computer system relates to the prevention of unauthorized access of information by unauthorized persons or entities [6]. An example of a breach of confidentiality in a computer system is when an individual or system obtains a password from another user without their permission. The integrity of a computer system relates to how well information can be protected against unauthorized alteration or destruction while the availability of a system relates to how easy it is for an authorized entity to access their own information. An example of a breach of integrity is when an unauthorized individual gains access to a computer system and modifies the stored data, while an example of a breach of availability is the inaccessibility of data because of its deletion [2].

There are over 20 types types of cyber attacks and the categorization of these cyber attacks vary from author to author. Sun et al. [1] simplifies the categories into 4 types of attacks, while Note and Ali [2] categorize them into 6 types, as shown in Figures 2.1 and 2.2 respectively.

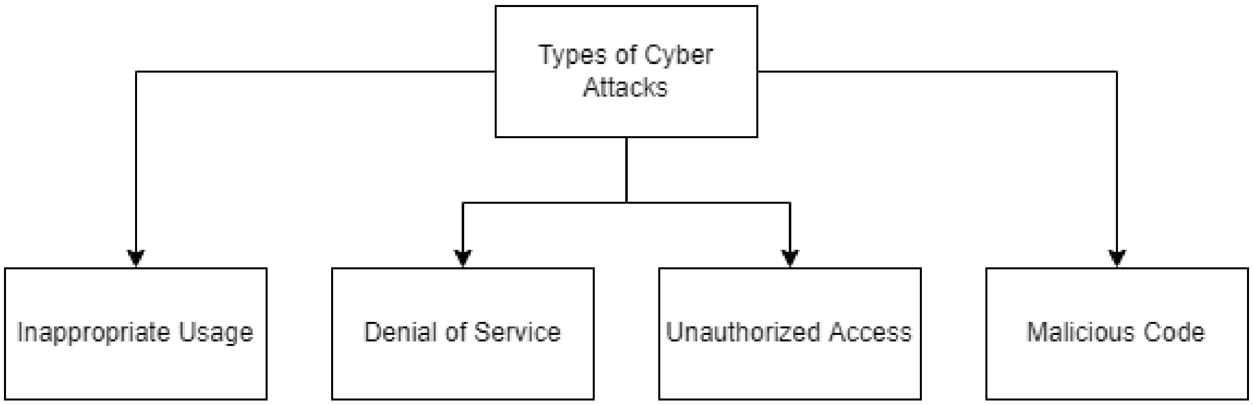


Figure 2.1: Types of Cyber Attacks According to Sun et al. [1]

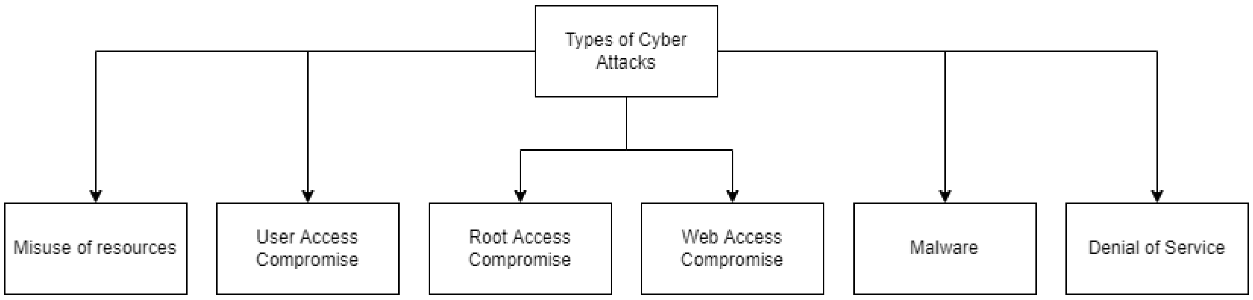


Figure 2.2: Types of Cyber Attacks According to Note and Ali [2]

2.1.2 Anomaly Detection

According to Khraisat et al. [10], in order to protect computer systems from attacks that compromise the confidentiality, integrity and availability of information, an intrusion detection system (IDS) is necessary. An IDS is a software or hardware system that identifies malicious network traffic. IDS systems are much more efficient in detecting cyber attacks than a traditional firewall [10]. According to Khraisat et al., there are two types of IDS systems: signature-based IDS and anomaly-based IDS [10]. Note and Ali [2] explain that an anomaly-based IDS is much better than a signature-based IDS. Companies that use a signature-based IDS often rely on past signatures they have already created to identify malicious software each time a cyber attack occurs. However, cyber attackers can change some bit of code and bypass the signature-based IDS, making this a huge flaw. Conversely, an anomaly-based IDS is trained using both benign and malicious software. This means that even if an attacker changed some bit of code, the anomaly-based IDS would still detect the attack because it knows how normal network traffic looks like. Multiple authors agree that indeed an anomaly-based IDS is best for future predictions of new attacks [2][6][10].

2.2 Machine Learning in Cyber Security

Sarker et al. defines machine learning (ML) to be a branch of artificial intelligence that relates with computational statistics, which enables computers to learn from data and make predictions based on the data provided [6]. ML is used in several fields such as finance, health, agriculture, transport, among others. In a cyber security context, ML is especially useful in anomaly detection when applied in an anomaly-based IDS [8].

2.2.1 Datasets

A cybersecurity dataset is required in order to train ML models to detect anomalies and hence cyber attacks. There are many datasets currently available for this purpose and different papers employ the use of different datasets when investigating the effectiveness of different ML algorithms in predicting cyber attacks, as shown in Table 2.1. A popular dataset amongst different authors is the NSL-KDD [11]. Note and Ali [2] emphasize that it is important to use well structured, good quality and free datasets, and among the ones used in their paper is the NSL-KDD dataset. Kilincer et al. [12] confirm this and further elaborate by mentioning that the NSL-KDD dataset was developed as an improvement to the original KDD CUP 99 dataset [13]. This improvement was made by removing redundant records as well as dividing the dataset into separate training and testing files [14].

Paper	Year	Datasets
Note and Ali [2]	2022	KDD Cup 99, NSL-KDD, Kyoto, UNSW-NB15
Rashid et al. [15]	2022	Windows 10, Windows 7, UNSW-NB15, NSL-KDD, KDD Cup 99
Elmrabit et al. [8]	2020	UNSW-NB15, CICIDS2017, ICS Cyber Attack
Kilincer et al. [12]	2021	CICIDS2018, NSL-KDD, ISCX 2012, CIDDS-001
Patil et al. [16]	2019	UNSW-NB15, CICIDS2017
Khammassi and Krichen [17]	2020	CICIDS2017, NSL-KDD, UNSW-NB15
Gottwalt et al. [18]	2019	UNSW-NB15
Kasongo et al. [19]	2020	UNSW-NB15, AWID
Verma et al. [20]	2018	CIDDS-001

Table 2.1: Comparison of Datasets Used Across Different Papers

Another popular dataset used for anomaly detection is the UNSW-NB15 [21]. Note and Ali [2] uphold the dataset as being easy to use and of good quality. However, Khammassi and Krichen [17] criticize the dataset by implying it has fewer features than the rest of the datasets. In comparison to the CICIDS2017 [22] dataset, the UNSW-NB15 has 44 features while the CICIDS2017 has 79 features. Nevertheless, Khammassi and Krichen [17] use the dataset in anomaly detection and they present their model results with the NSL-KDD and the CICIDS2017 datasets.

Not many papers use the fairly recent CICDDoS2019 dataset [23]. Chartuni and Marquez [24] explore the dataset and use it in multiclass classification. These authors describe the dataset as ideal for the detection of Distributed Denial of Service (DDoS) attacks. Note and Ali [2] also present the use of the Kyoto dataset [25]. The authors remark that it contains high levels of accuracy, despite containing 24 features, which is lower than most intrusion-detection datasets.

2.2.2 Machine Learning Algorithms

Most of the papers reviewed ML algorithms such as Gaussian Naive Bayes (NB), Decision Trees (DT), Random Forests (RF), Support Vector Machines (SVM), Logistic Regression (LR), K-Nearest Neighbours (KNN) and Multi-layer Perceptron (MLP), as shown in Table 2.2. Algorithms such as Adaptive Boosting (AdaB), Gradient Boosting(GB), Stochastic Gradient Descent (SGD), Multi-class Classifier (MCC), Simple Logistic Regression (SLR), Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN) and Deep Neural Networks (DNN) feature in the literature as well.

Paper	Year	Algorithms	Metrics
Note and Ali [2]	2022	NB, DT, SGD, RF, SVM, LR, MLP, GB	Accuracy, Time
Rashid et al. [15]	2022	NB, SVM, J48, RF, LR, KNN, RT, MCC, SLR, MLP	True Positive Rate, Time
Elmrabit et al. [8]	2020	LR, NB, KNN, DT, AdaB, RF, CNN, CNN-LSTM, LSTM, GRU, RNN, DNN	Accuracy, Precision, Recall, F1
Kilincer et al. [12]	2021	SVM, DT, KNN	Accuracy, Precision, Recall, Geometric Mean, F1
Patil et al. [16]	2019	RF	Accuracy, False Positive Rate
Khammassi and Krichen [17]	2020	C4.5, RF, NB	Accuracy
Gottwalt et al. [18]	2019	CorrCorr	Accuracy, Accuracy, Detection Rate, False Positive Rate
Kasongo et al. [19]	2020	FFDNN	Accuracy
Verma et al. [20]	2018	kNN, kmeans	Accuracy

Table 2.2: Comparison of Algorithms and Metrics Used Across Different Papers

Feature selection is an important part in machine learning model development. Rashid et al. [15] defines feature selection as the process of reducing the dimensions of a dataset by selecting a few features from a dataset. Sun et al. [1] expound on this by explaining that the selection of features directly affects the performance of a machine learning model. Sarker et al. [6] explain that deep learning algorithms such as MLP, CNN and LSTM eliminate the need for feature selection. However, deep learning algorithms require much more computational power for training models, compared to classical ML models [6]. Note and Ali [2] as well as Rashid et al. [15] confirm this in their results where deep learning algorithms have taken significantly longer time during training.

As evident in Table 2.2, most authors use accuracy as well as other metrics to evaluate performance of machine learning models. Elmrabit et al. [8] compared the accuracy of different algorithms using the datasets as listed in Table 2.1, and found that the best machine learning algorithm was Random Forests. Closely following random forest in terms of accuracy scores was the Decision Tree. Similarly, Note and Ali [2] found that Random Forests provided the highest accuracy overall, but training and testing times were not consistent depending on the size of the dataset. Elmrabit et al. [8] found that the Naive Bayes classifier had the poorest performance overall. Note and Ali [2] also found that the Naive Bayes classifier performed poorly across the datasets they used.

2.3 Gaps

Key papers in anomaly detection using ML have been discussed. Most authors use evaluation metrics such as accuracy, F1, precision and recall to evaluate ML models, as shown in Table 2.2. Fewer authors use training and testing duration as an evaluation metric, therefore creating a gap in research. In addition, newer datasets such as CICDDoS2019 have not been explored as much, as evident in Table 2.1. In addition, not many articles present which features greatly affect model performance, hence this remains unclear for several datasets. These gaps are therefore explored in this project.

Chapter 3

Methodology

3.1 Introduction

In order to investigate each of the objectives described in the Introduction chapter, it is necessary to design a logical methodology that will address each of the objectives. In this project, several recent datasets were chosen and a standard machine learning (ML) pipeline was followed, as illustrated in Figure 3.1. The programming language used to make the ML pipeline is Python [26] through the use of a Google Colaboratory Jupyter Notebook [27].

3.2 Datasets

The first step involved in the methodology was choosing cyber security datasets that would be used for anomaly detection. Three datasets were chosen, namely the NSL-KDD [11], CICIDS2017 [22] and the CICDDoS2019 [23] datasets.

3.2.1 NSL-KDD Dataset

The NSL-KDD dataset is an improved version of the KDD Cup 99 Dataset [13]. As discussed in section 2.2.1, the NSL-KDD dataset is well-structured and of good quality. This is the

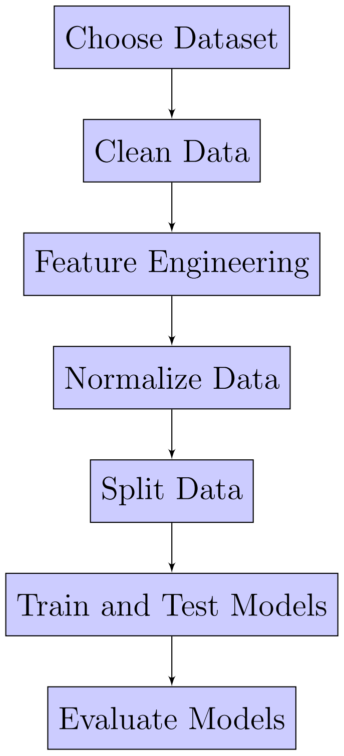


Figure 3.1: Diagram representing the methodology process

main reason why it has been chosen for use in this paper. The improvements were made by removing redundant values in the dataset [3]. The dataset is provided within a folder that contains 8 files. Out of the 8 files, 2 were chosen to be used in this project. The first file is the KDDTrain+.TXT which contains the full NSL-KDD training set in comma separated value (CSV) format. The second file is the KDDTest+.TXT which contains the full NSL-KDD test set in CSV format. The dataset contains 41 attributes, including the attack type and the difficulty level in predicting an attack, giving a total of 43 attributes.

The attack classes in the dataset are grouped into four categories, namely Denial of Service (DoS), Probing, U2R and R2L [3]. DoS attacks seek to make the machine or network inaccessible to its users while probing attacks invade a user's privacy by getting information about them. U2R attacks involve unauthorized access to a local user while R2L attacks involve unauthorized access to a machine remotely [3]. Examples of each attack type are detailed in Table 3.1.

The NSL-KDD dataset was downloaded from the Canadian Institute of Cyber Security

Category	Attacks
DoS	Back, Land, Neptune, Pod, Smurf, Teardrop, Apache2, Udpstorm, Process table, worm
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint
U2R	Buffer overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps
R2L	Guess Password, Ftp write, Imap, Phf, Multi-hop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named

Table 3.1: NSL-KDD Attack Categories [3]

(CIC) website [11] onto Google Drive [28]. Using Python's Pandas library [29], the KDD-Train+.TXT and the KDD-Test+.TXT files were loaded as dataframes on a new Jupyter Notebook.

3.2.2 CICIDS2017 Dataset

The CICIDS2017 dataset consists of eight CSV files containing features and attacks collected over a period of five days, as detailed in Table 3.2 [4].

Day	Label
Monday	Benign
Tuesday	Brute Force, FTP-Patator, SSH-Patator
Wednesday	Denial of Service, Heartbleed, DoS-Slowloris, DoS-Slowhttptest, Hulk, GoldenEye
Thursday	Web Attacks, Infiltration
Friday	DDoS LOIT, Botnet ARES, PortScans

Table 3.2: CICIDS2017 Labels [4]

Brute Force attacks are used for obtaining passwords and uncovering hidden pages. Heartbleed attacks involve the vulnerability of Open Secure Sockets Layer (Open SSL) protocols. Botnet attacks involve the use of multiple interconnected devices to steal data. Similar to the DoS attack, the Distributed DoS (DDoS) attack uses multiple systems to strain the resources of a computer user, rendering them inaccessible. Web attacks involve the use of SQL injections, script injections and brute force over http to access a user's data. On the other hand, infiltration attacks occur when a vulnerable software is used to create a back door that a cyber attacker could use to perform various attacks [4].

This dataset was selected to be used in this project because of its many features. The CICIDS2017 has 79 features, as compared to the 44 features present in the UNSW-NB15 discussed in section 2.2.1. However, the CICIDS2017 dataset has some shortcomings such as large volume of data, high class imbalance and missing values [30]. These shortcomings were addressed during the data cleaning.

The CICIDS2017 dataset was downloaded from the CIC website and loaded onto Google Drive. Using the Pandas library on a clean Jupyter Notebook, the eight CSV files were merged into one dataframe.

3.2.3 CICDDoS2019 Dataset

The CICDDoS2019 dataset consists of several CSV files, which contain attacks as illustrated in Figure 3.2.

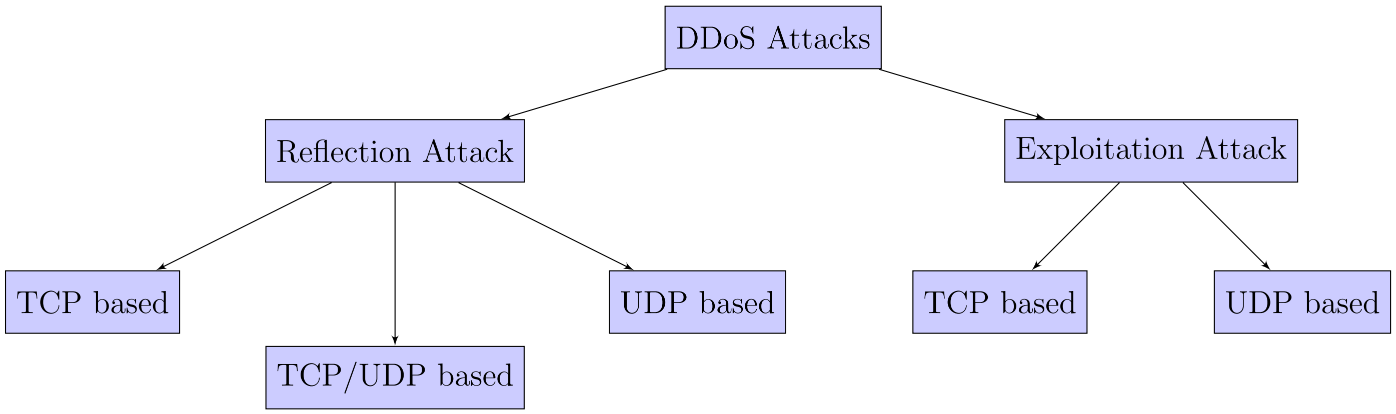


Figure 3.2: Diagram representing the attacks in CICDDoS2019

Both reflection based as well as exploitation based DDoS attacks involve the use of a third party component by the cyber attacker. In both categories, the cyber attacker sends packets to the reflector servers with the source internet protocol (IP) address set to the victim's IP address. Examples of each type of category are detailed in Table 3.3.

Category	Attacks
Reflection TCP Based Attack	MSSQL, SSDP
Reflection TCP/UDP Based Attack	DNS, LDAP, NETBIOS, SNMP, PORTMAP
Reflection UDP Based Attack	CharGen, NTP, TFTP
Exploitation TCP Based Attack	SYN Flood
Exploitation TCP Based Attack	UDP Flood, UDP-Lag

Table 3.3: CICDDoS Attack Examples [5]

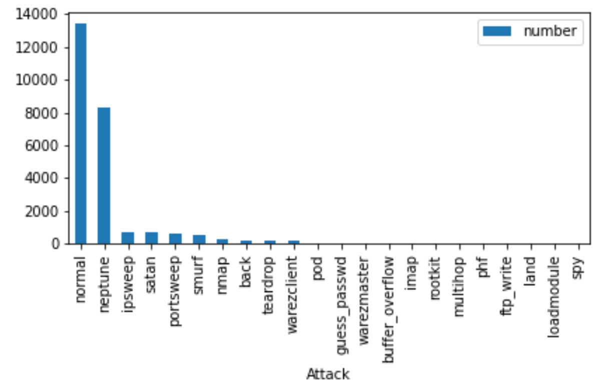
The main reason the CICDDoS2019 was selected as part of this project is because it is rarely used in literature, as explained in section 2.2.1. The dataset was downloaded from the CIC website and loaded onto Google Drive. Similar to the CICIDS2017 dataset, the CICDDoS2019 files were merged into one dataframe using Pandas library on a clean Jupyter Notebook.

3.3 Data Cleaning

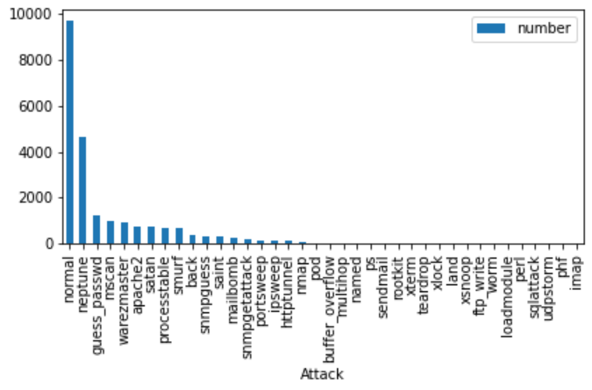
After loading each of the datasets onto a dataframe, it was then necessary to clean the dataframes. First, the column names were cleaned using a 'clean_columns()' command from a Python library called Skimpy [31]. The command changes all upper case letter to lower case and replaces any space in between column names with an underscore. The purpose of cleaning all column names is to put them in a standard format for easier reference. This command was applied to the CICIDS2017 and CICDDoS2019 datasets. The NSL-KDD dataset was left as it was, since the column names were already cleaned beforehand.

The next stage of cleaning the data was checking the number of unique classes in the attack feature for each dataset. As shown in Figure 3.3, there are many attacks in each dataset.

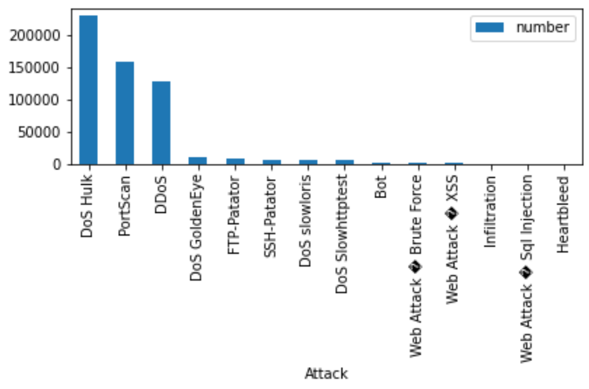
In order to reduce the attack classes, and to fulfill the objectives in Chapter 1, the attack classes were grouped into fewer categories as detailed in Table 3.4. The NSL-KDD attacks were all grouped into one category named 'attack'. This is so that the machine



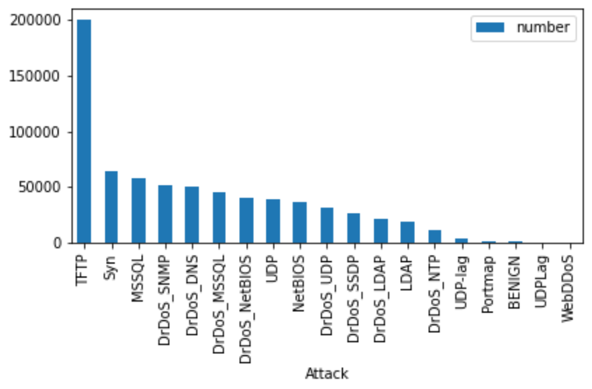
(a) Attacks in NSL-KDD Training Set



(b) Attacks in NSL-KDD Test Set



(c) Attacks in CICIDS2017



(d) Attacks in CICDDoS2019

Figure 3.3: Attacks across NSL-KDD, CICIDS2017 and CICDDoS2019 Datasets.

learning algorithms developed would classify the NSL-KDD attacks as a binary classification problem. On the other hand, the CICIDS2017 and the CICDDoS2019 datasets had more attack categories so that they would be evaluated as a multi-class classification problem. The attacks in the CICIDS2017 dataset were grouped according to categories recommended by Panigrahi and Borah [30]. The attacks in the CICDDoS2019 dataset were grouped into categories reflecting Figure 3.2.

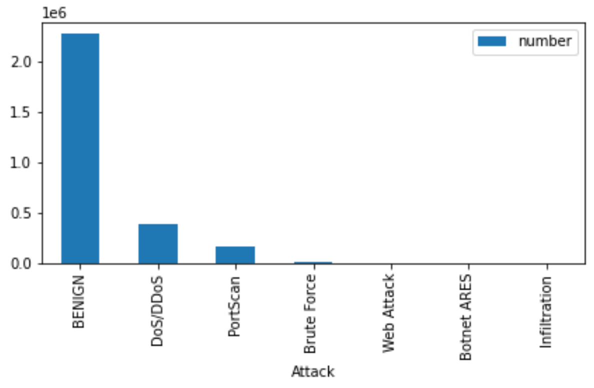
Dataset	Categories
NSL-KDD	Normal, Attack
CICIDS2017	Benign, Botnet ARES, Brute Force, DoS/DDoS, PortScan, Web Attack
CICDDoS2019	Benign, TCP Reflection Attack, UDP Reflection Attack, TCP/UDP Reflection Attack, TCP Exploitation Attack, UDP Exploitation Attack, WebDDoS

Table 3.4: Attack Classes in Each Dataset

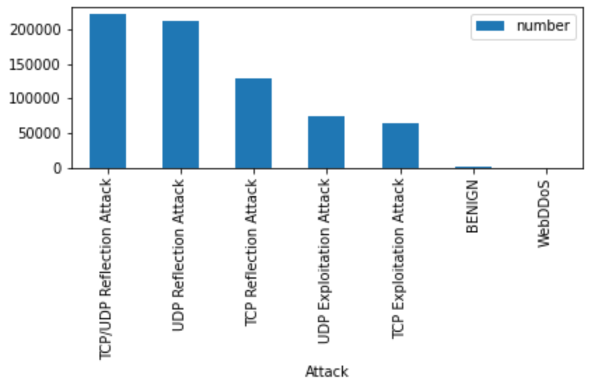
Next, all infinite values across the datasets were converted to not a number (NaN) values. This is because there is no command to remove infinity values directly. All NaN values were then removed from the dataset using Numpy's [32] 'dropna' function. The NSL-KDD dataset did not have any NaN values, implying that the dataset was cleaned beforehand. Conversely, the CICIDS2017 and the CICDDoS2019 datasets had 5,734 and 43,440 NaN values respectively.

The last step in cleaning the dataframes was conducting sampling. As evidenced in Figure 3.3, the attack classes were imbalanced and after placing them in the categories shown in Table 3.4, the attack groups in the CICIDS2017 and CICDDoS2019 datasets were still imbalanced as shown in Figure 3.4. The NSL-KDD dataset had fairly balanced classes as illustrated in Figure 3.5, therefore no sampling was performed on it.

As shown in Figure 3.4 (a), the benign values were over 2 million while the infiltration values were exactly 36. Similarly, the WebDDos and benign value counts in Figure 3.4 (b) were 2 and 1114 respectively, whereas the TCP/UDP reflection attacks were over 200,000.



(a) Attack Categories in CICIDS2017

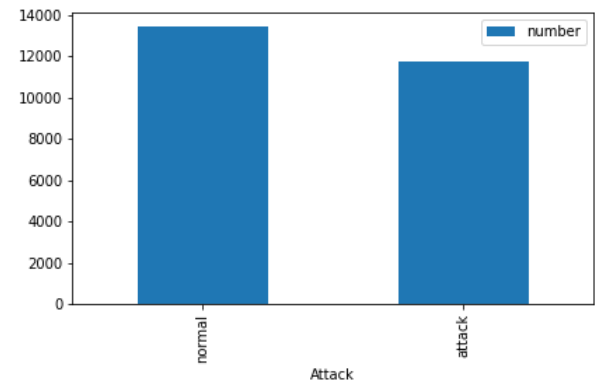


(b) Attack Categories in CICDDoS2019

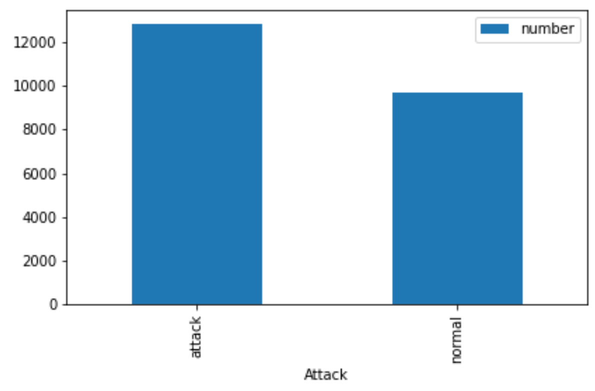
Figure 3.4: Attack Categories across CICIDS2017 and CICDDoS2019 Datasets.

The method of sampling chosen to balance the dataset was random under sampling. Random under sampling involves removing instances of the majority classes until they balance with the minority class [33]. This method of sampling was chosen because it is quick and easy to implement. However, since the number of infiltration attacks and WebDDoS attacks were too few, that is, 36 and 2 respectively, they were removed from the CICIDS2017 and the CICDDoS2019 datasets respectively. Consequently, the minority class in the CICIDS2017 dataset was the Botnet ARES class having 1966 values. Similarly, the minority class in the CICDDoS2019 dataset was the benign class having 1114 values.

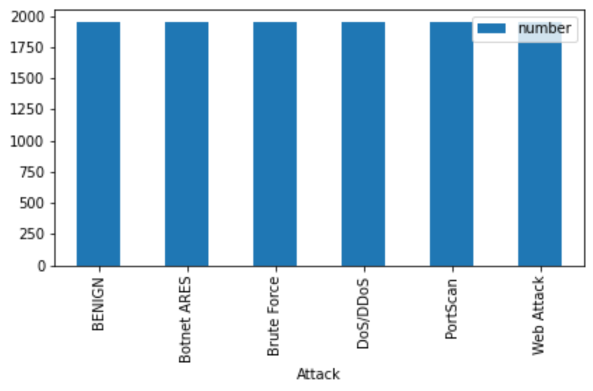
One main disadvantage of the Random Under Sampling method is that important data may be lost. However, since the minority class had a sufficiently large value in both datasets, this disadvantage was overlooked. Instead, the Random Under Sampling method introduced a new advantage. This advantage was that the dataframe would be smaller, enabling faster computation during model training and testing. Another method of sampling, SMOTE (Synthetic Minority Over-Sampling Technique) was considered for use. SMOTE involves oversampling the minority classes by adding synthetically made data points [33]. This would have been advantageous, since no attack would have been removed from the dataset. However, due to the increase in data points, the model training and testing would have been computationally inefficient as a result of higher testing and training times. The challenge of



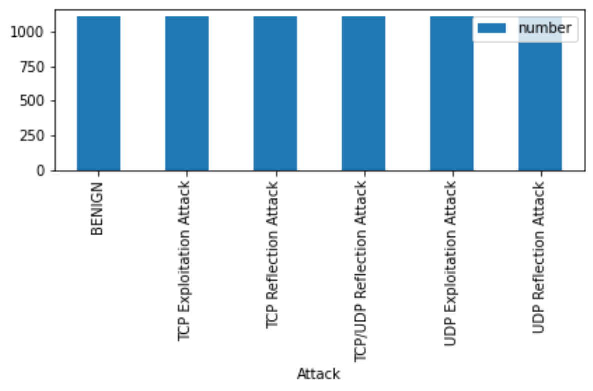
(a) Attack Categories in NSL-KDD Training Set



(b) Attack Categories in NSL-KDD Test Set



(c) Attack Categories in CICIDS2017



(d) Attack Categories in CICDDoS2019

Figure 3.5: Attack categories across NSL-KDD, CICIDS2017 and CICDDoS2019 Datasets after sampling.

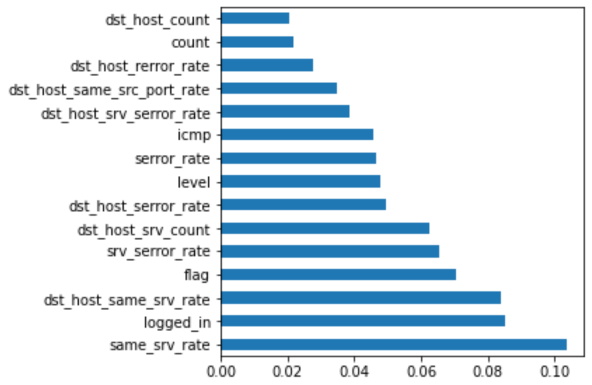
insufficient random access memory (RAM) in the Google Colaboratory Jupyter Notebook also arose when SMOTE was used. Consequently, Random Under Sampling was chosen as the most appropriate sampling technique. Figures 3.5(c) and 3.5(d) shows the classes across the CICIDS2017 and CICDDoS2019 datasets after sampling was carried out.

3.4 Data Normalization and Feature Engineering

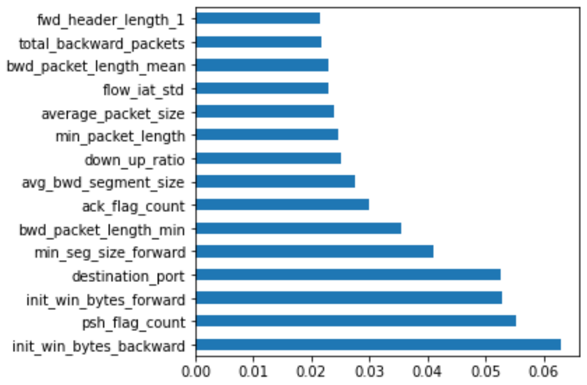
After cleaning and balancing the dataframes, it was then necessary to transform the data to ensure optimal machine learning performance. Data normalization involves transforming the numerical attributes of a dataset into a common range so that excessively large values cannot dominate the machine learning model [34]. In order for data normalization to take place, all predictor attributes must be in numerical format. This was the case in the CICIDS2017 dataset, as all predictor features were of integer data type. However, in the NSL-KDD and the CICDDoS datasets, there were some predictor features which were of object data type.

In order to change features that are of an object type to an integer type, it is necessary to encode them. The two popular types of encoding are label encoding and one-hot encoding. Label encoding involves assigning a numeric value to each unique label, while one-hot encoding involves the creation of new features for each unique category of an attribute [35]. In one-hot encoding, each column contains a '1' or a '0' depending on whether the feature is associated with the created column. In the NSL-KDD dataset, the features that were of an object data type were protocol type, service and flag. The number of unique values were then obtained for each feature. Protocol type, service and flag had 3, 66 and 11 distinct values respectively. One-hot encoding was applied to the protocol-type feature since only 2 more columns would be created. However, since the service and flag features had a high number of distinct values, label encoding was applied in order to prevent the creation of too many columns. This is because too many columns would increase the computational time of the machine learning process, thus making it inefficient.

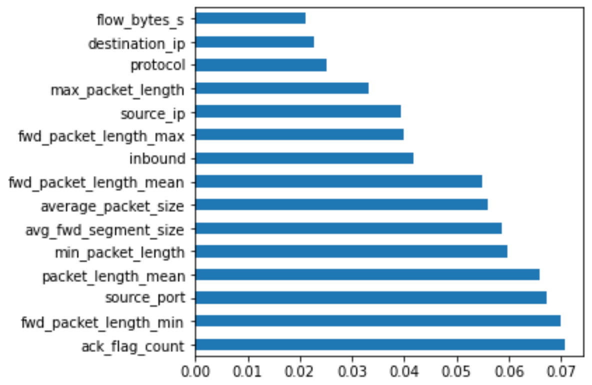
Similarly, the source IP, destination IP and similar http features in the CICDDoS2019 dataset were of object type and had 144, 259 and 26 distinct values respectively. As a result, label encoding was applied, since one-hot encoding would increase the number of columns. It was then necessary to select the best features in order to reduce the overall number of columns. This is because redundant and irrelevant features may cause inefficiencies during machine learning model training [34]. The Extra Trees classifier from the scikit-learn library was used to calculate feature importance. The Extra Trees classifier is made up of multiple decision trees [36]. Each tree is linked to random samples as well as random features, therefore a single tree cannot have access to all data. Feature importance is then calculated as a reduction in node impurity weight during the training process. The mathematical



(a) Most Important Features in the NSL-KDD dataset



(b) Most important features in the CICIDS2017 dataset



(c) Most important features in the CICDDoS2019 dataset

Figure 3.6: Feature Engineering.

criterion used to calculate feature importance is the Gini Index [36]. A higher value of impurity indicates a higher feature importance. Once the Extra Trees classifier was run, the 15 most important features were selected as the predictor features to be used in the model training and testing process. These features are as shown in Figure 3.6. It is important to note that feature engineering was not carried out for the multi-layer perceptron (MLP) model, since most deep learning algorithms do not require feature selection, as discussed in section 2.2.2.

Finally, the predictor features were then standardized using Scikit-learn's [37] standard scaler function. This form of normalization uses the mean and standard deviation to re-scale the data. The features that result have a mean of 0 and unit variance [34]. Each instance of the data, x_i is transformed into x'_i as follows:

$$x'_i = \frac{x_i - \mu_i}{\sigma_i} \tag{3.1}$$

where μ and σ represent the mean and standard deviation of the ith feature respectively [34].

3.5 Model Training and Testing

After data normalization and feature engineering, it was then necessary to train the resultant datasets on ML algorithms. For each dataset, the cleaned and normalized data frame was split into training data and testing data. The training data comprised 67 percent of the data frame while the testing data comprised 33 percent of the data frame. This was done using Python's 'train_test_split' function. The data was automatically shuffled during splitting to ensure that the order of the data had no effect on the model performance. In addition, a random state of 42 was selected for the function to ensure that the results were reproducible. The ML algorithms selected for model building and evaluation were classification algorithms such as Categorical Boosting (CatBoost), K-Nearest Neighbour (KNN), Stochastic Gradient Descent, Support Vector Machines (SVM), Adaptive boosting (AdaBoost), Decision Trees, Random Forests, Naive Bayes and Multi-layer Perceptron (MLP).

3.5.1 CatBoost Model

CatBoost is a form of gradient boosting that uses binary decision trees as base predictors [38]. Given a dataset with samples $S = \{(X_j, y_j)\}_{j=1,...,m}$, where $X_j = (x_j^1, x_j^2, ..., x_j^n)$ is a vector of n features and target feature $y_j \in \mathbb{R}$, then the samples (X_j, y_j) are distributed according to a distribution $p(\cdot, \cdot)$ [38]. The aim of CatBoost's learning task is to train a function $H : \mathbb{R}^n \to \mathbb{R}$ which minimizes loss as follows:

$$L(H) := \mathbb{E}L(y, H(X)) \tag{3.2}$$

where $L(\cdot,\cdot)$ is a smooth loss function and (X, y) is testing data sampled from S [38].

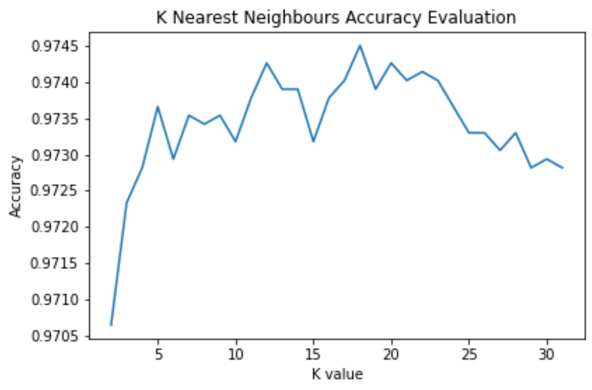
In this project, the parameters of the CatBoost algorithm included the number of iterations and the learning rate. These were set to 100 and 0.03 respectively.

3.5.2 KNN Model

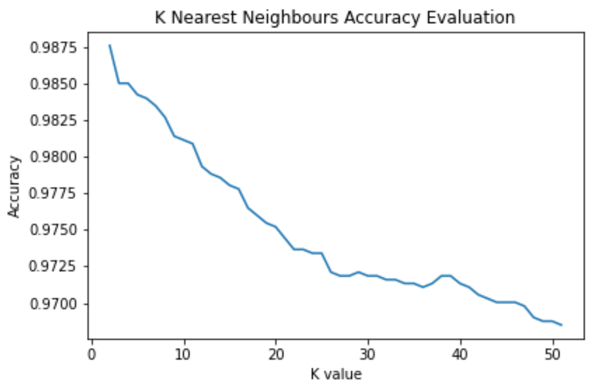
The K-Nearest Neighbour (KNN) algorithm is one of the most popular machine learning algorithms used because of its simplicity. It is especially used in classification and regression tasks [2]. The KNN algorithm is based on a Euclidean distance function that measures the distance between a test point and all the examples from the training data. It then looks for examples from the training data that are nearest to the test point and chooses the most nearest point in the case of classification. Given two features, $x = [x_1, x_2, x_3, ..., x_k]$ and $y = [y_1, y_2, y_3, ..., y_k]$ where k is the dimensional space, then the distance, D, is calculated as follows [8]:

$$D(x, y) = \sqrt{\sum_{i=1}^{k}(y_i - x_i)^2} \tag{3.3}$$

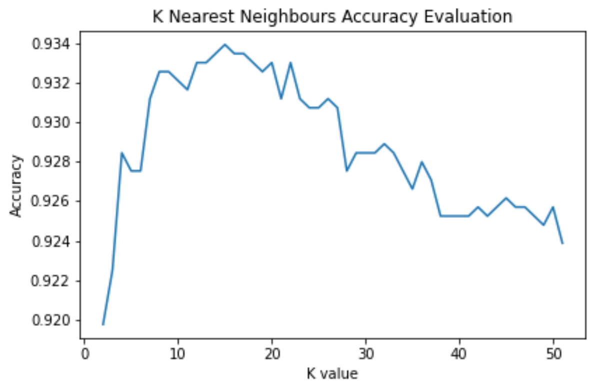
In order to optimize the KNN algorithm, a suitable value of K must be chosen by running the algorithm multiple times until the K value that gives the highest accuracy is found. This was done for the NSL-KDD, CICIDS2017 and the CICDDoS2019 datasets as illustrated in Figure 3.7. The best values of K were 18, 2 and 15 in the NSL-KDD, CICIDS2017 and CICDDoS2019 datasests respectively. These were selected to be used during model training and testing.



(a) Finding the best value of K in NSL-KDD



(b) Finding the best value of K in CICIDS2017



(c) Finding the best value of K in CICDDoS2019

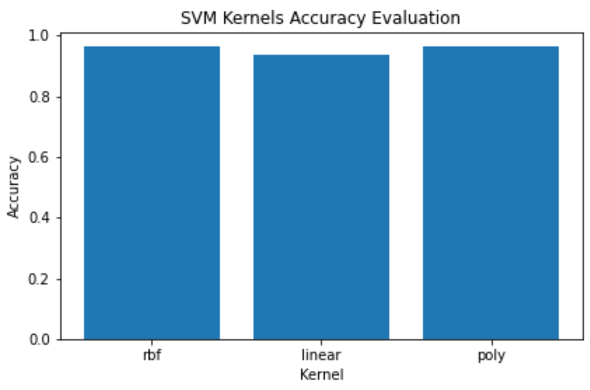
Figure 3.7: Finding the best value of K

3.5.3 Stochastic Gradient Descent Model

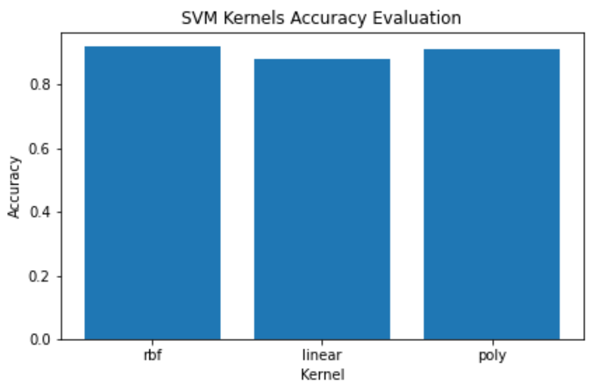
The Stochastic Gradient Descent algorithm is an optimization algorithm that finds the minimum value of a target function when given a set of input variables. The algorithm involves the calculation of the gradient of the target function with respect to the input variables. A learning rate, which is also known as a step size, is used to regulate the rate of change of each input variable with respect to the gradient. The loss function, which is the target function that is being minimized, is computed after every sample within a period of time. The model is then updated with schedules of decreasing strength [2]. The main advantage of Stochastic Gradient Descent is that it picks a random point in the dataset during each computation, hence saving time. However, the algorithm does poorly on large datasets. In this project, the Stochastic Gradient Descent classifier was trained using 1000 iterations.

3.5.4 SVM Model

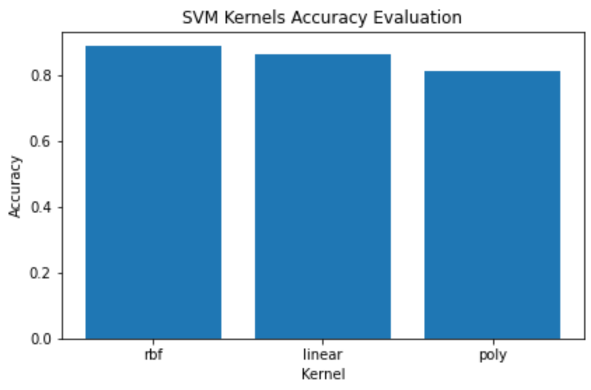
The Support Vector Machine (SVM) algorithm works by dividing different classes in a dataset using hyperplanes. Linear SVM algorithms divide classes using a straight hyperplane [2]. This is especially useful with linear data. Conversely, non-linear SVM algorithms are used with non-linear data. Non-linear data needs to be transformed to higher dimensions in order to find appropriate hyperplanes to separate classes. However, calculations involved with higher dimension data is computationally expensive and inefficient. This problem, however, is solved by the use of kernel functions that simplify the calculation complexity of higher dimensions. The most popular kernel functions used in SVM algorithms are the radial basis function (RBF) kernel, the linear kernel and the polynomial kernel [39]. In this project, different kernel types were chosen and their accuracies in the model were compared. The kernel with the highest accuracy for each dataset was used to develop the SVM model. The polynomial kernel was selected in the SVM model for the NSL-KDD dataset as it had the highest accuracy. In contrast, the RBF kernel was chosen for the SVM models in the



(a) Kernel Accuracies in NSL-KDD



(b) Kernel Accuracies in CICIDS2017



(c) Kernel Accuracies in CICDDoS2019

Figure 3.8: Comparison of kernel accuracies in the SVM models across datasets

CICIDS2017 dataset and the CICDDoS2019 dataset as it had the highest accuracies in both. This is illustrated in Figure 3.8.

3.5.5 AdaBoost Model

The Adaptive Boosting (AdaBoost algorithm) is an ensemble classifier that uses an iterative approach to build strong classifiers from weak ones. The AdaBoost classifier works by fitting a copy of a classifier on a dataset and updates the dataset, fitting copies of the same classifier each time while balancing the inaccurate data points [40]. The AdaBoost classifier gives emphasis to the wrongly classified points in the weak classifier and increases the weights of the wrongly classified data to prevent future errors. In this project, a decision tree with a maximum depth of 7 was chosen to be the base classifier for the AdaBoost model. The random state was chosen as 1 to ensure reproducibility.

3.5.6 Decision Trees Model

Decision Trees are a type of machine learning algorithm that uses a top-down recursive division structure as shown in Figure 3.9. Decision trees consist of root nodes, decision nodes and terminal nodes. The root node represents the start of the problem, that is, the homogeneous sample. The root node splits into branches that can consist of either decision nodes or terminal nodes. Decision nodes are nodes that can be split into further sub-nodes, while terminal or leaf nodes are the final nodes that do not split. Starting from the root node, each decision node finds a feature in its training sample set to test the sample set and splits into further nodes until a specific termination condition is met [41].

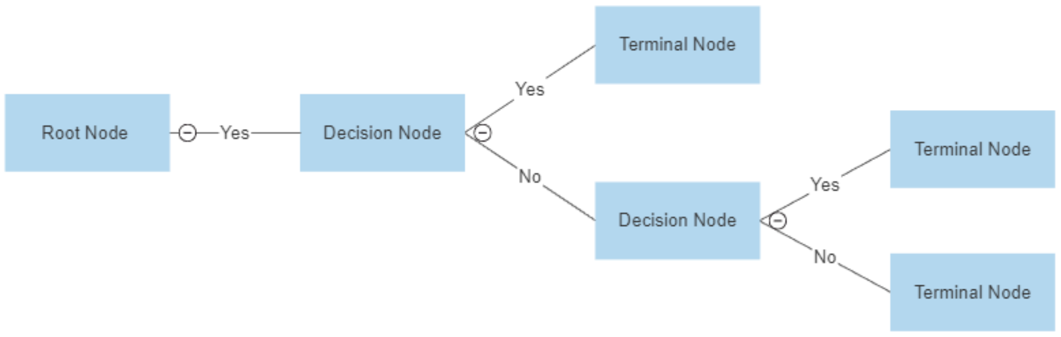
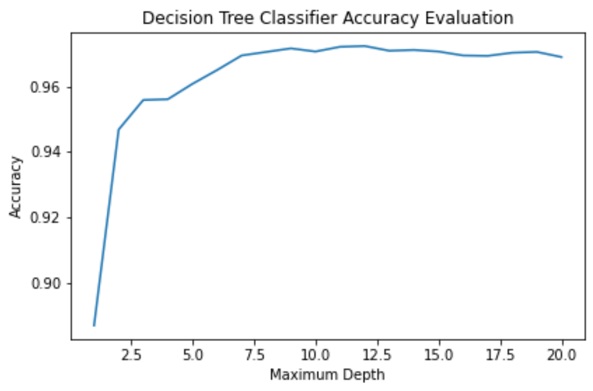
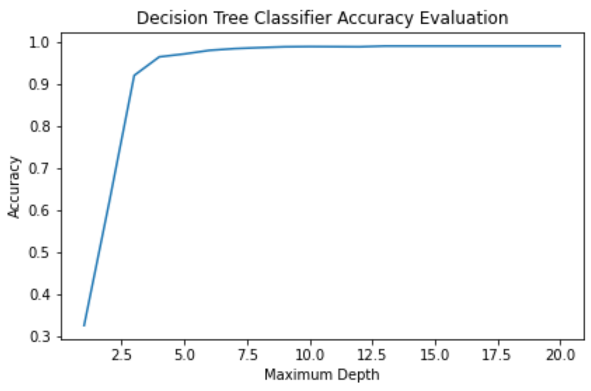


Figure 3.9: Decision Tree

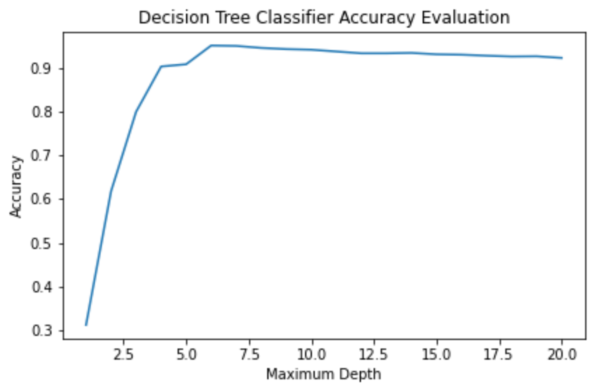
Constructing a decision tree involves two steps, that is, tree building and pruning. Tree building involves using part of the training data to split root nodes into terminal nodes using a specified depth. Pruning involves checking the decision tree obtained in the tree building stage and correcting errors. This stage removes unnecessary and redundant parts of the tree until a correct decision tree is built [41]. In this project, the maximum depths of decision tree models were varied until a suitable decision tree with the highest accuracy was



(a) Tree Depth Accuracies in NSL-KDD



(b) Tree Depth Accuracies in CICIDS2017



(c) Tree Depth Accuracies in CICDDoS2019

Figure 3.10: Comparison of accuracies while varying tree depth in the Decision Tree models across datasets

obtained. This was done for the NSL-KDD, CICIDS2017 and the CICDDoS2019 datasets as illustrated in Figure 3.10. The values of tree depth chosen were 12, 13 and 6 for the NSL-KDD, CICIDS2017 and the CICDDoS2019 datasets respectively. It is evident in Figure 3.10 that decision tree accuracy increases with increase in depth up to a certain critical depth. Past this critical depth, the accuracy remains fairly constant.

3.5.7 Random Forests Model

A random forest classifier is a combination of several decision trees trained in parallel through bootstrapping. Bootstrapping ensures that various samples of a training set are trained using decision trees in parallel, so that each decision tree in the random forest is distinct. The effect of this is that the overall variance of the random forest classifier is reduced, making it a powerful classification technique [2]. The main advantage of the random forest classifier over the decision tree classifier is that feature scaling does not need to be carried out while using the random forest classifier. In addition, the random forest classifier is more robust to noise present in the training set [42].

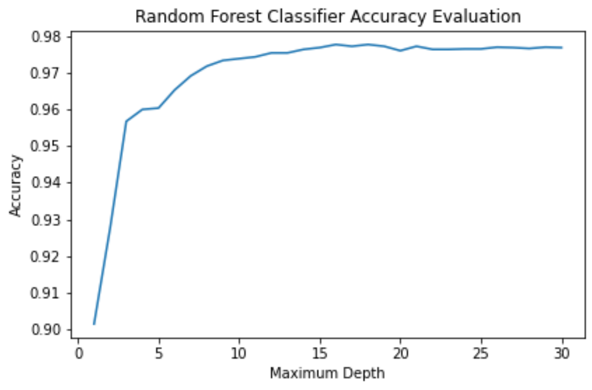
Similar to the decision tree algorithm, the maximum depth parameter was varied and different instances of the model was run, as illustrated in Figure 3.11. The maximum depth that produced that highest accuracy was then chosen. Values of maximum depth were selected as 16, 15 and 6 in the NSL-KDD, CICIDS2017 and CICDDoS2019 datasets respectively. It is worth noting that the accuracy increased up to a critical value of maximum depth and then remained fairly constant in the random forest models across the three datasets, as illustrated in Figure 3.11.

3.5.8 Naive Bayes Model

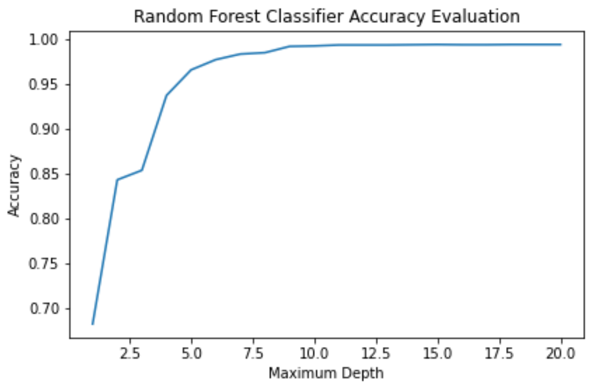
Bayes' theorem is represented as follows:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} \tag{3.4}$$

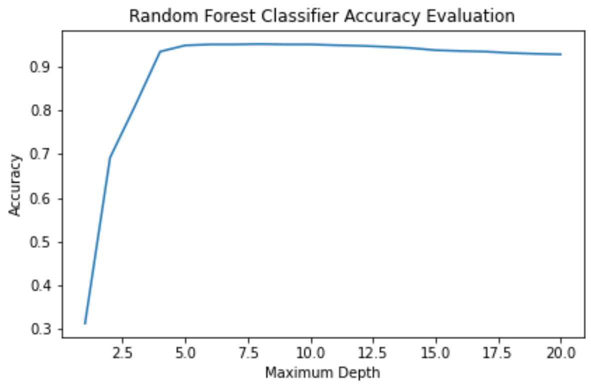
where A and B are events, $P(A \mid B)$ is the probability of A given B is true, $P(B \mid A)$ is the probability of B given A is true and $P(A)$ and $P(B)$ are the probabilities of A and B respectively. This theorem enables one to calculate conditional probabilities. The Naive Bayes machine learning algorithm, on the other hand, uses Bayes' theorem to calculate probabilities. It assumes that the features in a given dataset are strongly independent. This is a disadvantage, as most features in a real world dataset are dependent on each other. The main advantage of the Naive Bayes classifier is that it is easy to build, it is relatively quick in performance compared to other machine learning algorithms and it works well with



(a) Random Forest Depth Accuracies in NSL-KDD



(b) Random Forest Depth Accuracies in CI-CIDS2017



(c) Random Forest Depth Accuracies in CICD-DoS2019

Figure 3.11: Comparison of accuracies while varying maximum depth in the Random Forest models across datasets

relatively small datasets [43]. In this project, a Gaussian Naive Bayes classifier was trained and tested using data from the NSL-KDD, CICIDS2017 and the CICDDoS2019 dataset.

3.5.9 MLP Model

The perceptron is a simple neural network aimed at binary classification. It is a linear classifier, aimed at separating two classes using a straight line. The perceptron algorithm produces a single output based on its inputs and a linear combination of their weights. This is represented as follows:

$$y = \varphi(\sum_{i=1}^{n}(w_i - x_i) + b) \tag{3.5}$$

where w is the vector of weights, x is the input vector, b is the bias and φ is the non-linear activation function [44]. Due to its limited mapping capability, a single perceptron is not useful on its own for predictions. It was therefore necessary to merge perceptron layers to form a multilayer perceptron (MLP). An MLP machine learning model is made up of an input layer that receives a signal, an output layer that gives a result based on the inputs, and a number of layers in between the input and output layer that comprises of neurons that make computations, as illustrated in Figure 3.12.

The number of neurons in the input layer correspond to the number of features in the dataset, while the number of neurons in the output layer correspond to the number of classes to be predicted [45]. Consequently, unlike other machine learning algorithms, the MLP classifier does not require feature engineering. Each hidden layer in a MLP network consists of weights and as a result, the MLP learning process involves changing the weights between layers to obtain a negligible difference between the actual output and the desired output [45]. Since the MLP architecture is based on artificial neural networks, the MLP classifier is a deep learning algorithm. In this project, 3 hidden layers were used for the MLP classifier. The first layer consisted of 150 neurons, the second layer consisted of 100 neurons, while the third layer was composed of 50 neurons. The maximum iterations were

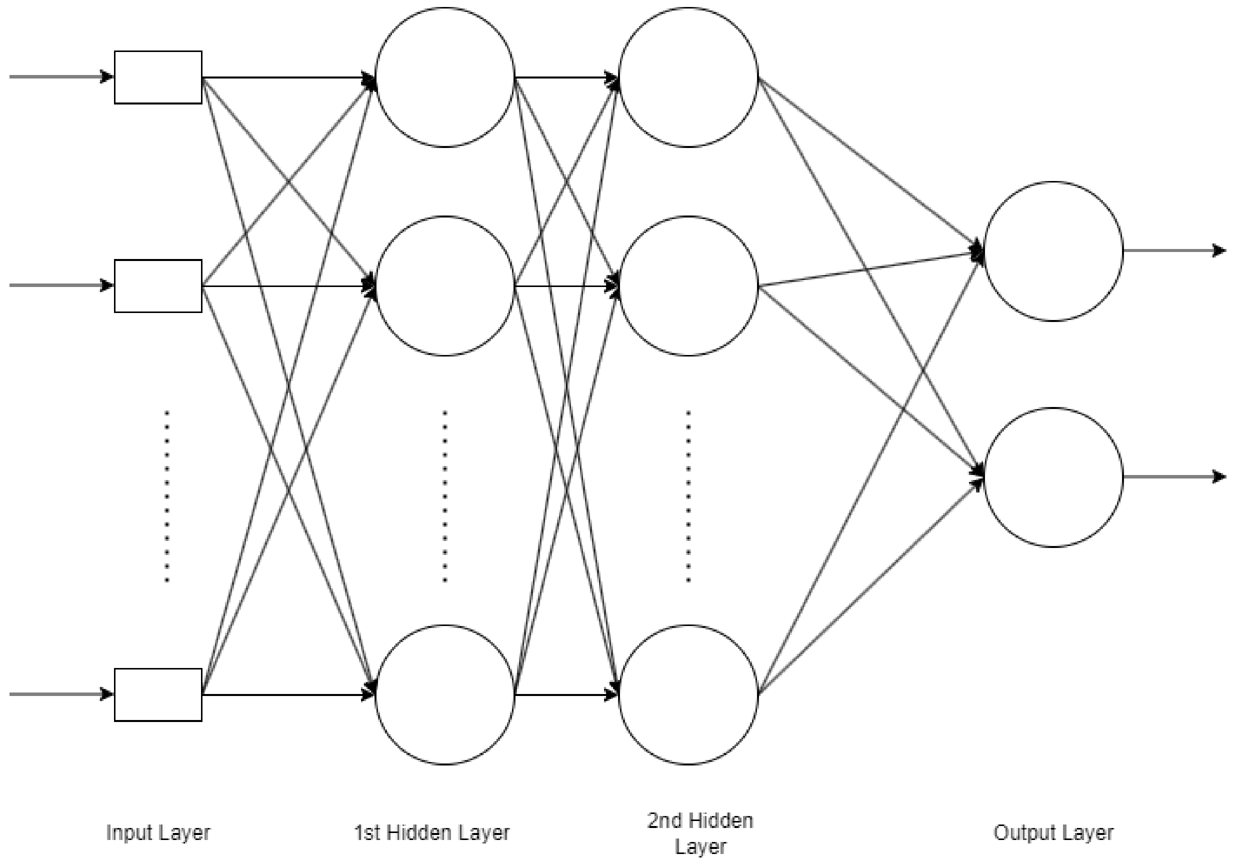


Figure 3.12: A Multilayer Perceptron Network with two hidden layers.

set at 300. The optimizer chosen for the MLP model was an adam optimizer. The model was then trained and tested using data from the NSL-KDD, CICIDS2017 and CICDDoS2019 datasets.

3.5.10 Model Fitting and Evaluation

Each of the machine learning algorithms discussed in this section was then used to fit the data from the NSL-KDD, CICIDS2017 and CICDDoS2019 datasets. Several model performance metrics were then used to evaluate the performance of each machine learning model. These metrics include accuracy, F1 score, precision, recall, training time, testing time and a confusion matrices. For the NSL-KDD's binary classification models, area under the curve (AUC) scores and Receiver Operating Characteristic (ROC) curves were used in addition to the aforementioned metrics. The accuracy of a model describes the total percentage of correct classifications. Precision defines how close the measurements are to each other [8]. Before defining the remaining metrics, it is important to understand what true positives, true negatives, false positives and false negatives mean. A true positive (TP) is an instance where a model correctly predicts the positive class. A true negative (TN) is where the model correctly predicts the negative class. It then follows that a false positive (FP) is an instance when a model incorrectly predicts the positive class. Similarly, a false negative (FN) occurs when a model incorrectly predicts a negative class. With these definitions, accuracy and precision can be defined as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.6}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.7}$$

Recall can be defined as the percentage of actual positives that were correctly identified and is mathematically defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{3.8}$$

F1 scores are a combination of precision and recall scores and they take the harmonic mean of precision and recall scores as follows:

$$F1 = \frac{2(P \times R)}{P + R} \tag{3.9}$$

The F1 score's main purpose is to act as a means of comparison between two classifiers. A confusion matrix is a table that visualizes the true negative, true positive, false negative and false positive scores. It is as illustrated in Figure 3.13

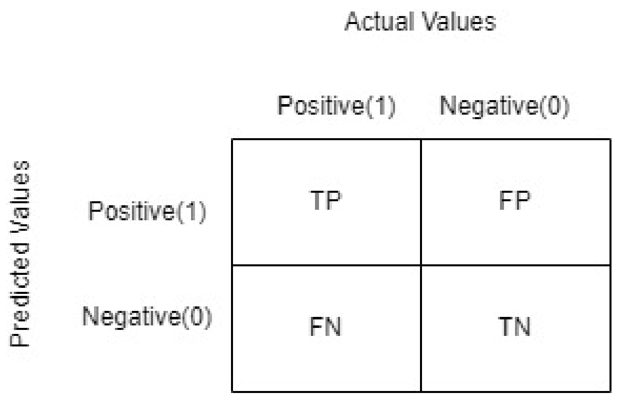


Figure 3.13: Confusion Matrix Structure

AUC scores as well as ROC Curves are used especially in binary classifications. The AUC score is the area under the ROC curve and as long as the AUC score is higher, then the machine learning model has performed better.

The evaluation metrics presented in this section were used across the binary NSL-KDD models, as well as in the multi-class CICIDS2017 and CICDDoS2019 models. The training and testing duration of the machine learning models were measured three times and the average was obtained. Python's time library was used, and the specific function used was 'time.time()'. This function returns a float number that represents time in seconds. This function was called at the beginning of the training code and saved to a variable. The function was then called at the end of the training code and saved as another variable. The difference between the two variables was the training time. A similar procedure was followed during model testing. At the beginning of model testing stage, the function was used to capture that point in time and saved in a variable. At the end of the model testing, the same function was used and the value was saved in a separate variable. The difference between the two variables was the testing time.

It is important to note that in the NSL-KDD dataset, the training and testing of machine learning models was performed using the KDDTrain+.TXT file. However, to simulate real-world applications of the models, the models obtained using the KDDTrain+.TXT file were used to test data present in the KDDTest+.TXT file. The same evaluation metrics described in this section were applied to the KDDTest+.TXT file.

3.6 Coding Details

As mentioned previously in this chapter, the coding environment used was a Google Colaboratory Jupyter Notebook [27]. The dataset files were downloaded onto Google Drive [28]. The coding language used for the machine learning pipeline was Python. For data cleaning, the Pandas [29], Numpy [32] and Skimpy [31] libraries were used. Pandas was used to load the data, clean the data, and manipulate dataframes. Numpy was used for numerical manipulations and Skimpy was used for data cleaning, particularly column names. The normalization of data, model training and testing all used the Scikit-learn library [37]. All visualizations were made using the Matplotlib library [46]. The processor used for the Jupyter Notebook was a CPU (central processing unit) and the maximum Random Access Memory (RAM) available was 25GB.

3.7 Summary

The methodology chapter describes the machine learning process, as illustrated in Figure 3.1. A description of the datasets chosen, that is, the NSL-KDD, CICIDS2017 and the CICDDoS2019, was provided. The reasons for the selection of each of the datasets were also explained. The data cleaning process that involved merging data files, removing infinite and NaN values, and appropriate sampling methods was explained. The process of data normalization and feature engineering was then highlighted, where the standard scaler, one-hot encoding and label encoding techniques were highlighted as data normalization methods. The Extra Trees Classifier was explained as a means of feature engineering, from which suitable features were extracted. Next, the machine learning algorithms selected for use in this project were explained. These include CatBoost, KNN, Stochastic Gradient Descent, SVM, AdaBoost, Decision Trees, Random Forests, Naive Bayes and MLP machine learning algorithms. The evaluation metrics used in this project were then described. These include accuracy, precision, recall, F1 and AUC scores. Confusion matrices, training time, testing time and ROC curves were also highlighted as evaluation metrics used.

Chapter 4

Results

This chapter presents the results obtained from the Methodology. The results mainly consist of tables and figures.

4.1 Introduction

The results presented in this section include the evaluation metric scores described in section 3.5.10. These include the training time, testing time, accuracy score, precision, recall, F1 score and confusion matrices for each model in both the multi-class evaluated CICIDS2017 and CICDDoS2019 datasets. In addition to the aforementioned evaluation metrics, AUC scores and ROC curves are presented for the binary-class evaluated NSL-KDD dataset.

4.2 CICIDS2017

This section presents the results related to the CICIDS2017 dataset.

4.2.1 Evaluation Metrics

The training and testing duration of several ML models are presented in Tables 4.1 and 4.2 respectively. The accuracy, precision, recall and F1 scores are recorded in Table 4.3.

Algorithm	Training Time 1	Training Time 2	Training Time 3	Average Training Time
CatBoost	1.06	1.098	1.015	1.058
KNN	0.024	0.022	0.02	0.022
SGD	0.101	0.099	0.108	0.103
SVM	2.432	3.151	3.222	2.935
AdaBoost	2.215	3.17	2.714	2.700
Decision Tree	0.025	0.038	0.025	0.029
Random Forest	0.555	0.597	0.559	0.570
Naïve Bayes	0.014	0.016	0.018	0.016
MLP	18.35	20.844	17.757	18.984

Table 4.1: Model training duration for CICIDS2017 dataset

Algorithm	Test Time 1	Test Time 2	Test Time 3	Average Test Time
CatBoost	0.016	0.017	0.013	0.015
KNN	0.089	0.065	0.084	0.079
SGD	0.005	0.002	0.002	0.003
SVM	0.481	0.529	0.533	0.514
AdaBoost	0.133	0.134	0.127	0.131
Decision Tree	0.002	0.002	0.002	0.002
Random Forest	0.07	0.068	0.067	0.068
Naïve Bayes	0.005	0.005	0.005	0.005
MLP	0.025	0.024	0.0235	0.024

Table 4.2: Model testing duration for CICIDS2017 dataset

Algorithm	Accuracy	Precision	Recall	F1
CatBoost	0.97	0.98	0.97	0.97
KNN	0.99	0.99	0.99	0.99
SGD	0.82	0.83	0.82	0.81
SVM	0.92	0.93	0.92	0.92
AdaBoost	0.98	0.98	0.98	0.98
Decision Tree	0.99	0.99	0.99	0.99
Random Forest	0.99	0.99	0.99	0.99
Naïve Bayes	0.86	0.86	0.86	0.85
MLP	0.98	0.98	0.98	0.98

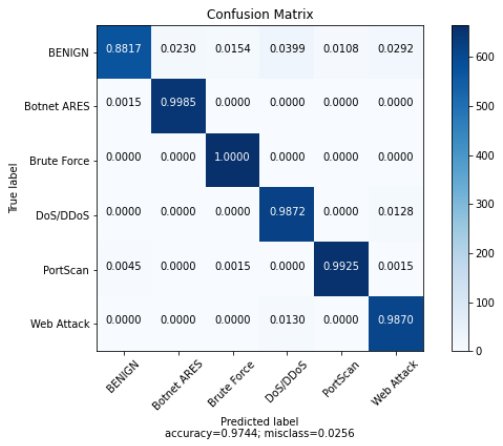
Table 4.3: Model evaluation scores for CICIDS2017 dataset

4.2.2 Confusion Matrices

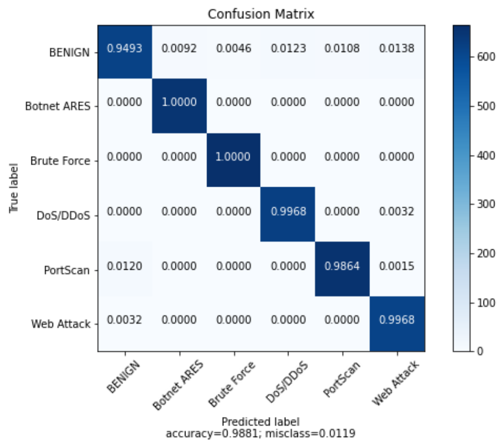
The confusion matrices for the various ML models are illustrated in Figures 4.1 and 4.2.

4.2.3 Observations

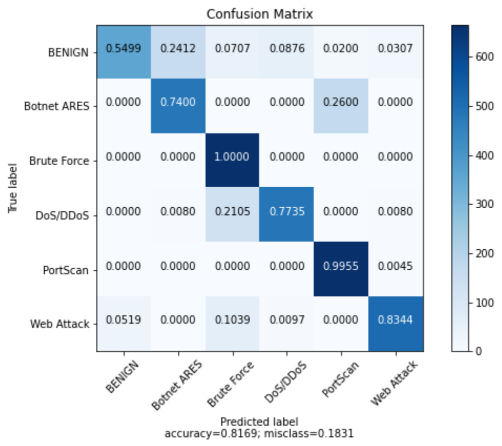
From Table 4.3, it is observed that the Random Forest, Decision Tree and KNN classifiers have the highest evaluation metric scores. Accuracy, precision, recall and F1 scores for these three models are 99%. They are closely followed by AdaBoost, MLP and CatBoost classifiers which have accuracies of 98%, 98% and 97% respectively. The Stochastic Gradient Descent (SGD) and the Naive Bayes models did poorly in comparison with the rest of the classifiers, with accuracies of 82% and 86% respectively. Out of the most accurate models, KNN and Decision Tree models had the lowest average training times, as illustrated in 4.1. The Decision Tree classifier had a lower average testing time compared to KNN, as evident in Table 4.2. This makes the Decision Tree the best classifier overall in this case. With regards to the confusion matrices, SGD, SVM and Naive Bayes had the highest misclassification



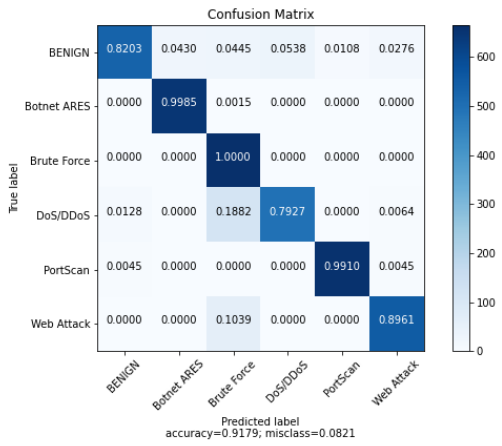
(a) CatBoost Confusion Matrix



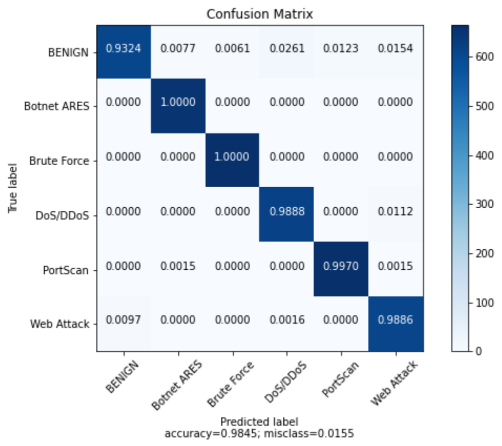
(b) KNN Confusion Matrix



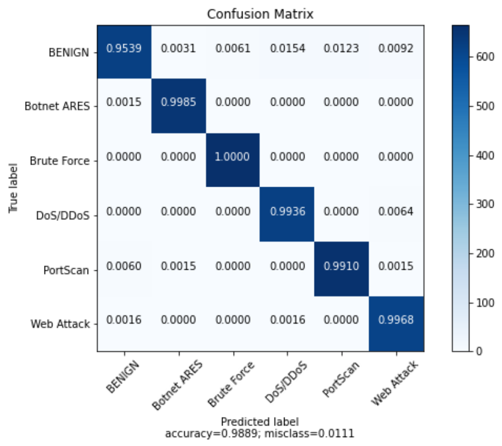
(c) SGD Confusion Matrix



(d) SVM Confusion Matrix

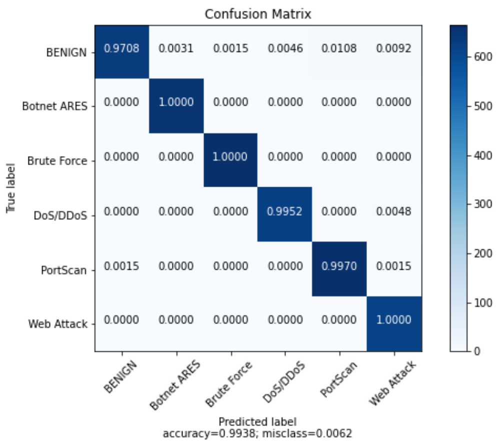


(e) AdaBoost Confusion Matrix

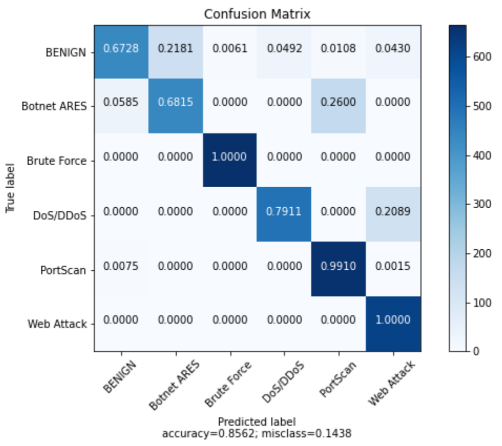


(f) Decision Trees Confusion Matrix

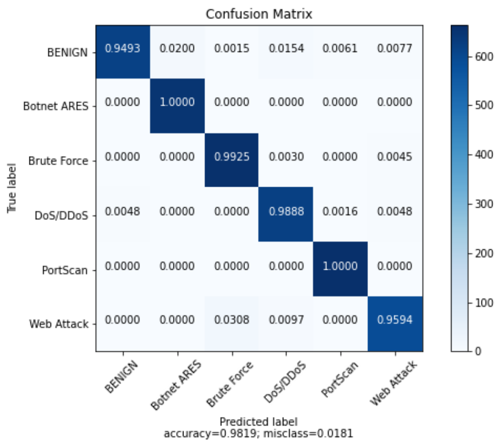
Figure 4.1: Confusion Matrices for the CICIDS2017 Machine Learning Algorithms



(a) Random Forest Confusion Matrix



(b) Naive Bayes Confusion Matrix



(c) MLP Confusion Matrix

Figure 4.2: Confusion Matrices for the CICIDS2017 Machine Learning Algorithms

percentages, as illustrated in Figures 4.1 and 4.2. Since SGD and Naive Bayes had the lowest evaluation metric scores and highest misclassification percentages of 18.31% and 14.38%, it follows that they would be the poorest classifiers in comparison to the rest of the ML algorithms used to model the CICIDS2017 dataset.

4.3 CICDDoS2019

This section presents the results associated with the CICDDoS2019 dataset.

4.3.1 Evaluation metrics

The model training and testing times are presented in Tables 4.4 and 4.5 respectively. The accuracy, precision, recall and F1 scores are presented in Table 4.6.

Algorithm	Training Time 1	Training Time 2	Training Time 3	Average Training Time
CatBoost	1.909	2.067	1.777	1.918
KNN	0.02	0.022	0.018	0.020
SGD	0.116	0.126	0.087	0.110
SVM	1.344	1.296	1.33	1.323
AdaBoost	2.546	2.731	2.545	2.607
Decision Tree	0.025	0.023	0.027	0.025
Random Forest	0.451	0.439	0.423	0.438
Naïve Bayes	0.01	0.01	0.012	0.011
MLP	12.928	13.295	15.465	13.896

Table 4.4: Model training duration for CICDDoS2019 dataset

Algorithm	Test Time 1	Test Time 2	Test Time 3	Average Test Time
CatBoost	0.026	0.021	0.025	0.024
KNN	0.039	0.031	0.042	0.037
SGD	0.001	0.001	0.002	0.001
SVM	0.286	0.279	0.311	0.292
AdaBoost	0.088	0.087	0.086	0.087
Decision Tree	0.001	0.002	0.002	0.002
Random Forest	0.044	0.041	0.039	0.041
Naïve Bayes	0.003	0.003	0.004	0.003
MLP	0.013	0.014	0.015	0.014

Table 4.5: Model testing duration for CICDDoS2019 dataset

Algorithm	Accuracy	Precision	Recall	F1
CatBoost	0.95	0.95	0.95	0.95
KNN	0.93	0.94	0.93	0.93
SGD	0.88	0.88	0.88	0.88
SVM	0.89	0.89	0.89	0.88
AdaBoost	0.95	0.95	0.95	0.95
Decision Tree	0.95	0.95	0.95	0.95
Random Forest	0.95	0.95	0.95	0.95
Naïve Bayes	0.76	0.77	0.76	0.74
MLP	0.94	0.95	0.94	0.94

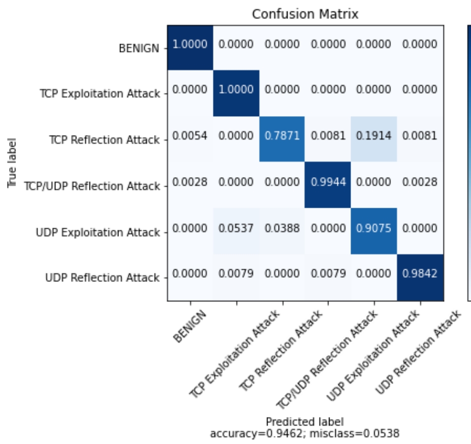
Table 4.6: Evaluation metrics for CICDDoS2019 dataset

4.3.2 Confusion Matrices

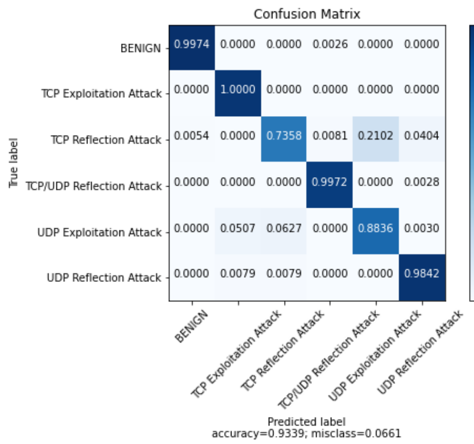
The confusion matrices are illustrated in Figures 4.3 and 4.4.

4.3.3 Observations

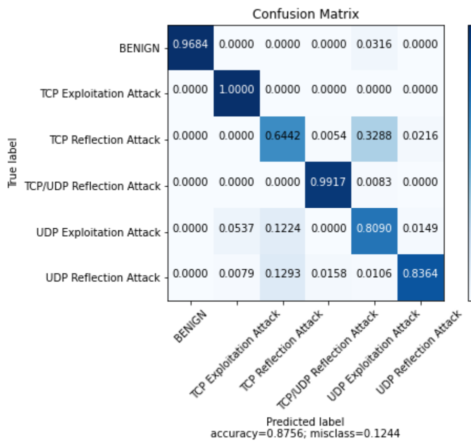
The AdaBoost, Decision Tree, Random Forest and Catboost classifiers were the best models all with accuracy, precision, recall and F1 scores of 95% as illustrated in Table 4.6. Their performance was followed closely by the MLP and KNN classifiers, with accuracies of 94% and 93% respectively. The classifier with the lowest accuracy score was the Naive Bayes classifier, having an accuracy of 76%. In terms of average training time, the Decision Tree and KNN models had the lowest average training times, as shown in Table 4.4. Although the KNN classifier had a slightly faster average training time than the Decision Tree classifier, the Decision tree classifier had a faster average testing time compared to the KNN classifier, as evidenced in Table 4.5. The MLP classifier had the slowest average training time. With regards to the confusion matrices, the Naive Bayes had the highest percentage



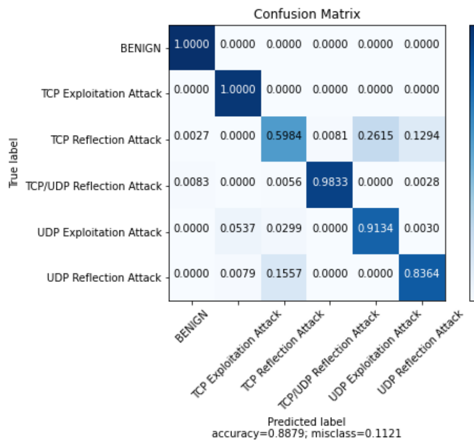
(a) CatBoost Confusion Matrix



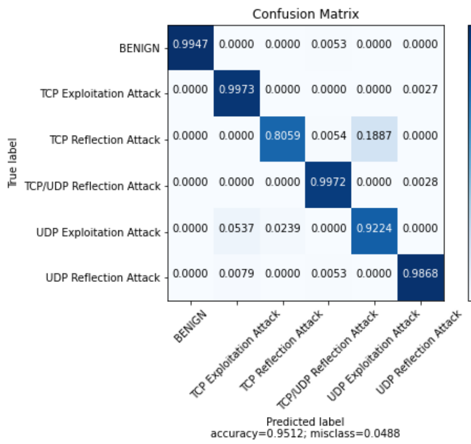
(b) KNN Confusion Matrix



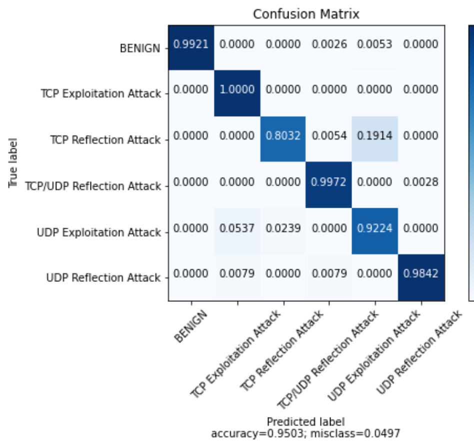
(c) SGD Confusion Matrix



(d) SVM Confusion Matrix

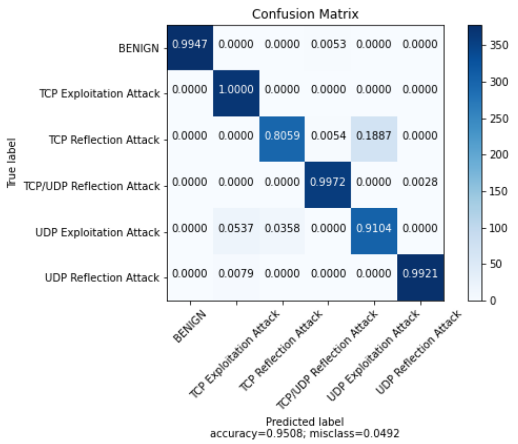


(e) AdaBoost Confusion Matrix

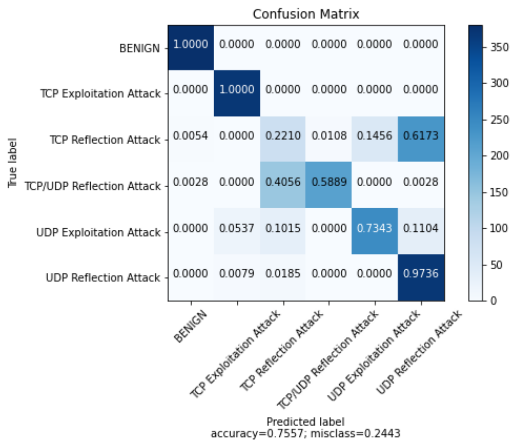


(f) Decision Trees Confusion Matrix

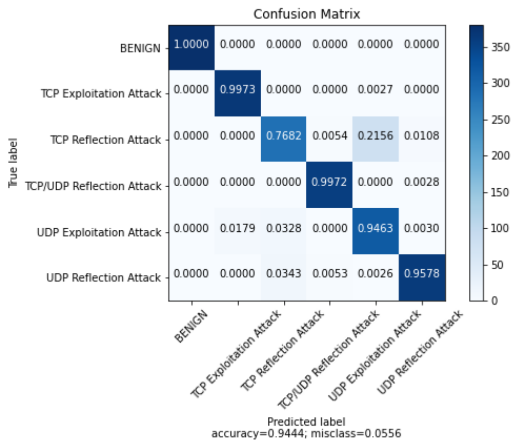
Figure 4.3: Confusion Matrices for the CICDDoS2019 Machine Learning Algorithms



(a) Random Forest Confusion Matrix



(b) Naive Bayes Confusion Matrix



(c) MLP Confusion Matrix

Figure 4.4: Confusion Matrices for the CICDDoS2019 Machine Learning Algorithms

of misclassification at 24.43% as illustrated in Figure 4.4(b). The SGD and SVM also had high percentages of misclassification at 12.44% and 11.21% respectively, as illustrated in Figures 4.3(c) and 4.3(d). Overall, with regards to the evaluation metrics and the confusion matrices, the Decision Tree and KNN classifiers performed the best, while the Naive Bayes classifier performed the poorest. In spite of being the poorest classifier, the Naive Bayes had relatively fast average training and testing times.

4.4 NSL-KDD

This section presents the results associated with the NSL-KDD dataset.

4.4.1 Evaluation Metrics

The training and testing times for the NSL-KDD dataset, with regards to the KDDTrain+.TXT file, are presented in Tables 4.7 and 4.8. The model testing time with regards to the KDDTest+.TXT file is presented in Table 4.9. The accuracy, precision, recall, F1 scores for the NSL-KDD dataset with regards to the KDDTrain+.TXT and KDDTest+.TXT files are illustrated in Tables 4.10 and 4.11 respectively.

Algorithm	Training Time 1	Training Time 2	Training Time 3	Average Training Time
CatBoost	1.547	0.557	0.584	0.896
KNN	0.074	0.076	0.167	0.106
SGD	0.059	0.082	0.11	0.084
SVM	9.812	9.455	9.445	9.571
AdaBoost	4.17	4.096	9.145	5.804
Decision Tree	0.045	0.063	0.123	0.077
Random Forest	0.941	0.889	1.028	0.953
Naïve Bayes	0.043	0.035	0.05	0.043
MLP	12.587	21.769	11.596	15.317

Table 4.7: Model training duration for NSL-KDD dataset (KDDTrain+.TXT)

Algorithm	Test Time 1	Test Time 2	Test Time 3	Average Test Time
CatBoost	0.07	0.032	0.037	0.046
KNN	1.099	1.336	1.354	1.263
SGD	0.002	0.002	0.005	0.003
SVM	0.452	0.445	0.885	0.594
AdaBoost	0.137	0.133	0.219	0.163
Decision Tree	0.002	0.002	0.004	0.003
Random Forest	0.108	0.111	0.142	0.120
Naïve Bayes	0.004	0.004	0.003	0.004
MLP	0.046	0.079	0.043	0.056

Table 4.8: Model testing duration for NSL-KDD dataset (KDDTrain+.TXT)

Algorithm	Test Time 1	Test Time 2	Test Time 3	Average Test Time
CatBoost	0.016	0.03	0.022	0.023
KNN	2.375	2.367	2.394	2.379
SGD	0.003	0.002	0.007	0.004
SVM	1.217	1.202	1.222	1.214
AdaBoost	0.352	0.328	0.344	0.341
Decision Tree	0.005	0.005	0.003	0.004
Random Forest	0.209	0.206	0.197	0.204
Naïve Bayes	0.006	0.006	0.005	0.006
MLP	0.121	0.109	0.111	0.114

Table 4.9: Model testing duration for NSL-KDD dataset (KDDTest+.TXT)

Algorithm	Accuracy	Precision	Recall	F1	AUC
CatBoost	0.97	0.97	0.97	0.97	0.99
KNN	0.97	0.97	0.97	0.97	0.99
SGD	0.94	0.94	0.94	0.94	-
SVM	0.96	0.96	0.96	0.96	0.99
AdaBoost	0.97	0.97	0.97	0.97	0.97
Decision Tree	0.97	0.97	0.97	0.97	0.99
Random Forest	0.98	0.98	0.98	0.98	0.99
Naïve Bayes	0.93	0.93	0.93	0.93	0.97
MLP	0.83	0.83	0.83	0.83	0.83

Table 4.10: Evaluation Metrics for NSL-KDD dataset (KDDTrain+.TXT)

Algorithm	Accuracy	Precision	Recall	F1	AUC
CatBoost	0.76	0.81	0.76	0.76	0.90
KNN	0.79	0.84	0.79	0.79	0.82
SGD	0.77	0.83	0.77	0.77	-
SVM	0.77	0.83	0.77	0.77	0.88
AdaBoost	0.79	0.82	0.79	0.80	0.81
Decision Tree	0.76	0.81	0.76	0.76	0.80
Random Forest	0.76	0.81	0.76	0.76	0.90
Naïve Bayes	0.80	0.84	0.80	0.80	0.91
MLP	0.67	0.70	0.67	0.67	0.70

Table 4.11: Evaluation Metrics for NSL-KDD dataset (KDDTest+.TXT)

4.4.2 Confusion Matrices and ROC Curves

The confusion matrices associated with the KDDTrain+.TXT file are illustrated in Figures 4.5 and 4.6. The confusion matrices associated with the KDDTest+.TXT file are illustrated in Figures 4.7 and 4.8. The ROC curves associated with the KDDTrain+.TXT file are illustrated in Figures 4.9 and 4.10. The ROC curves associated with the KDDTest+.TXT file are illustrated in Figures 4.11 and 4.12.

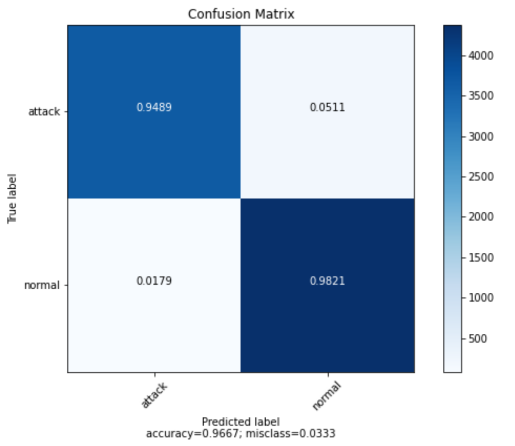
4.4.3 Observations

KDDTrain+.TXT

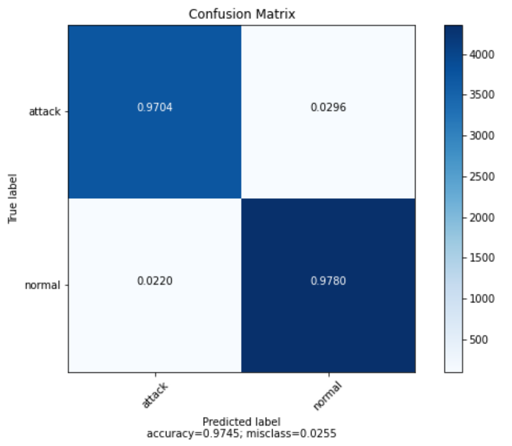
With regards to the Train+.TXT file, the model with the highest accuracy, precision, recall, F1 and AUC scores was the Random Forest model, as evident in Table 4.10. The Random forest classifier had an accuracy of 98%, followed by the Decision Tree, AdaBoost, KNN and CatBoost models with accuracies of 97%. It is worth noting that the SVM classifier had a high accuracy of 96%. Conversely, the MLP classifier had the lowest accuracy at 83%. The Naive Bayes, Decision Tree, SGD and KNN models had the fastest average training times, as shown in Table 4.7. The Decision Tree, SGD and Naive Bayes classifiers had the fastest average testing times, but the KNN classifier had the slowest average testing time, as shown in Table 4.8. In terms of confusion matrices, it is worth noting that the KNN classifier had a slightly lower misclassification percentage compared to the Decision Tree classifier, as illustrated in Figure 4.5. The best performing classifier with accuracy, training speed and testing speed taken into consideration would be the Decision Tree.

KDDTest+.TXT

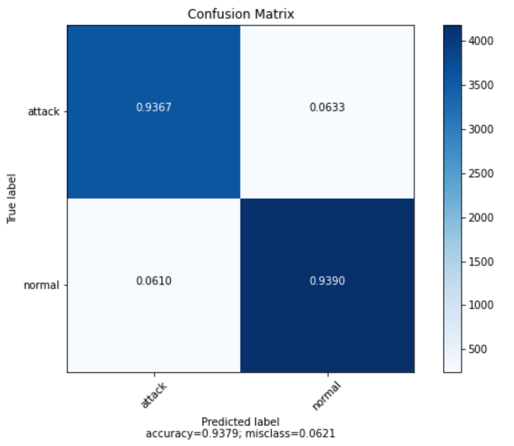
With regards to the KDDTest+.TXT file which was used to test the ML models developed using the KDDTrain+.TXT file, the model with the highest evaluation metric scores was the Naive Bayes classifier, as presented in Table 4.11. The Naive Bayes classifier had an



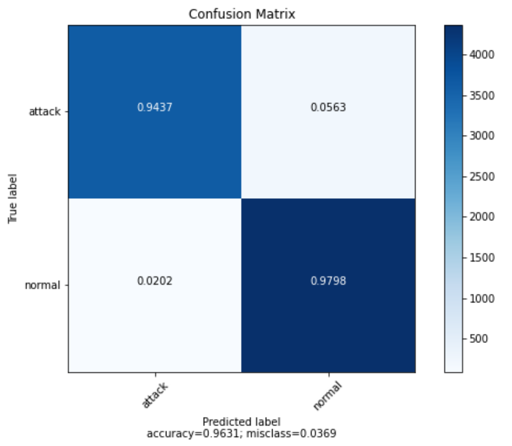
(a) CatBoost Confusion Matrix



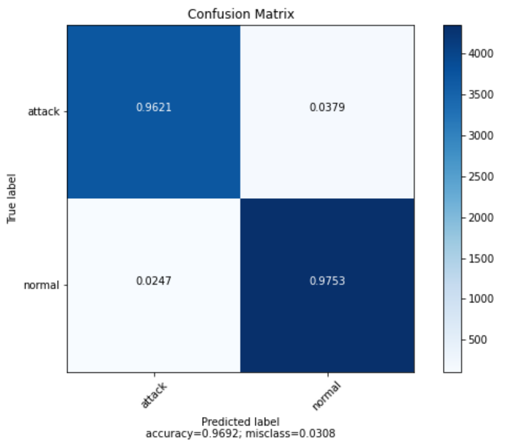
(b) KNN Confusion Matrix



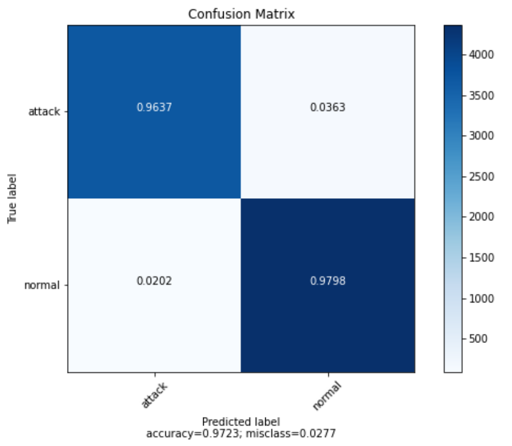
(c) SGD Confusion Matrix



(d) SVM Confusion Matrix

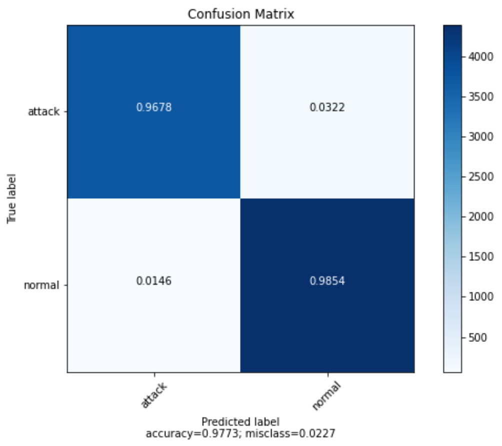


(e) AdaBoost Confusion Matrix

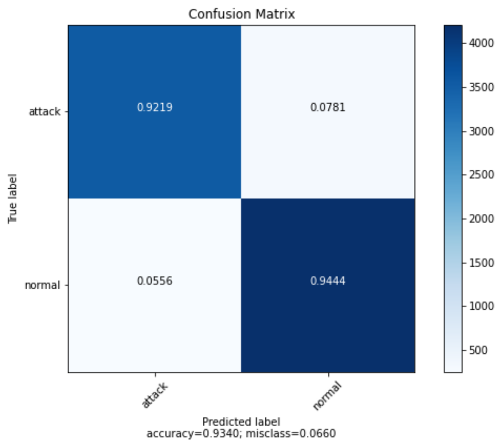


(f) Decision Trees Confusion Matrix

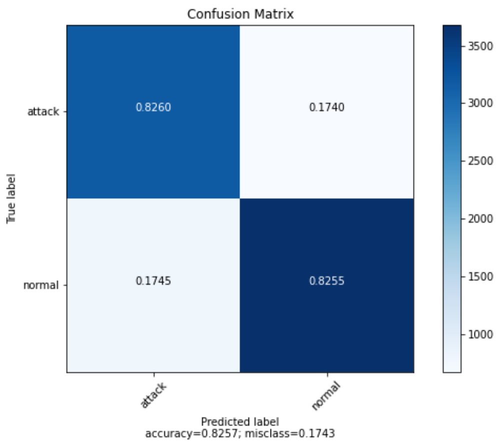
Figure 4.5: Confusion Matrices for the NSL-KDD ML Algorithms (KDDTrain+.TXT)



(a) Random Forest Confusion Matrix



(b) Naive Bayes Confusion Matrix

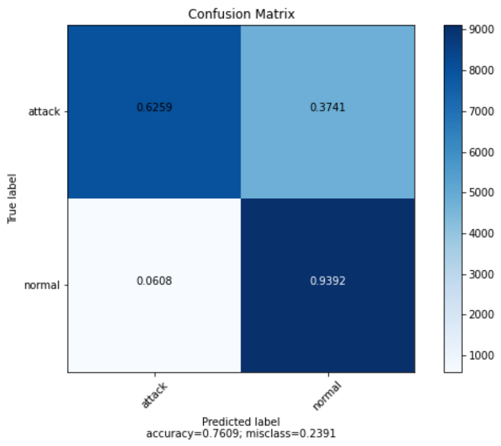


(c) MLP Confusion Matrix

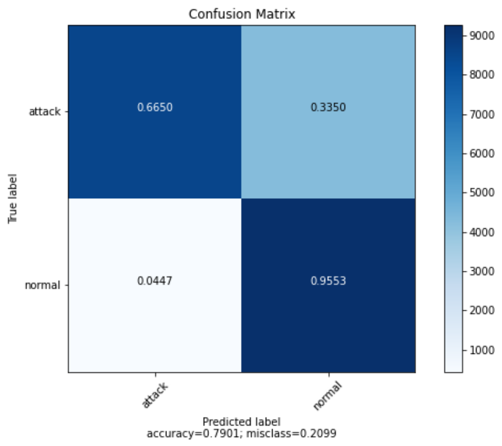
Figure 4.6: Confusion Matrices for the NSL-KDD ML Algorithms (KDDTrain+.TXT)

accuracy of 80%. This was followed by the AdaBoost and KNN classifiers with accuracies of 79% each. The MLP classifier had the lowest accuracy at 67%. In terms of average testing time, the SGD and Decision Tree classifiers had the fastest testing time closely followed by the Naive Bayes classifier as illustrated in Table 4.9. The KNN classifier had the slowest average testing time.

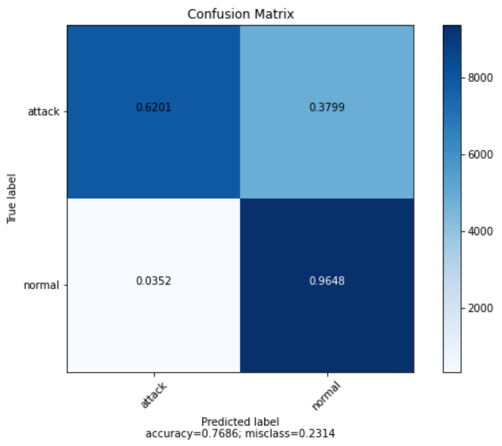
In terms of confusion matrices, the Naive Bayes classifier had the lowest misclassification percentage, as illustrated in Figure 4.8(b). It is also worth noting that the AdaBoost classifier had the lowest percentage of false positives, as illustrated in Figure 4.7(e). The models with the highest AUC scores were CatBoost, Random Forests and Naive Bayes, and this



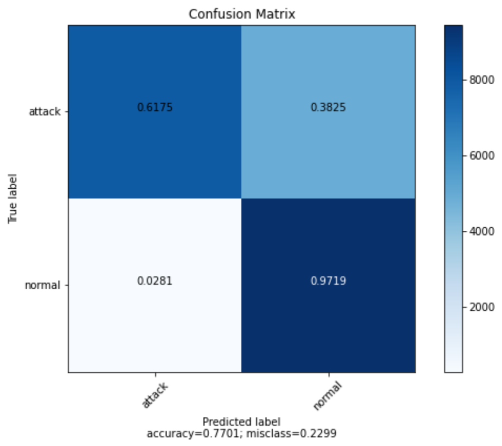
(a) CatBoost Confusion Matrix



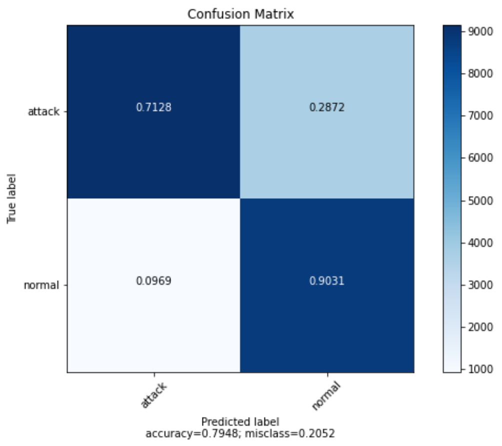
(b) KNN Confusion Matrix



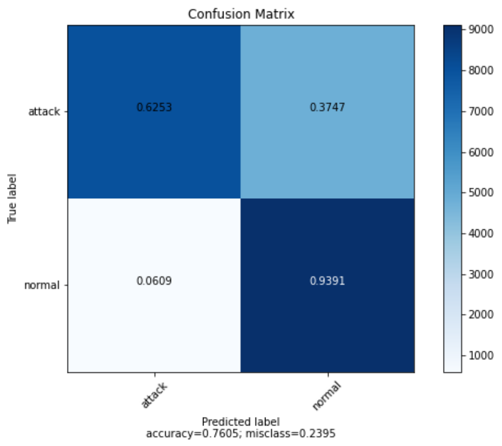
(c) SGD Confusion Matrix



(d) SVM Confusion Matrix

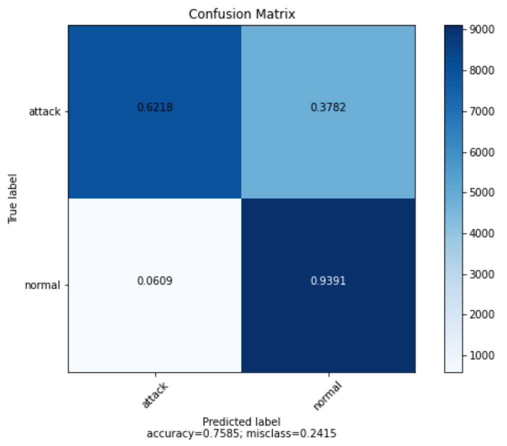


(e) AdaBoost Confusion Matrix

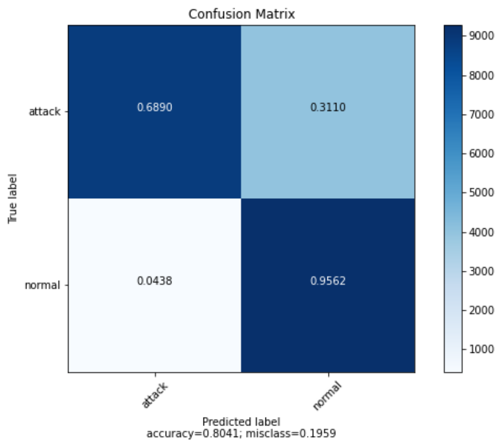


(f) Decision Trees Confusion Matrix

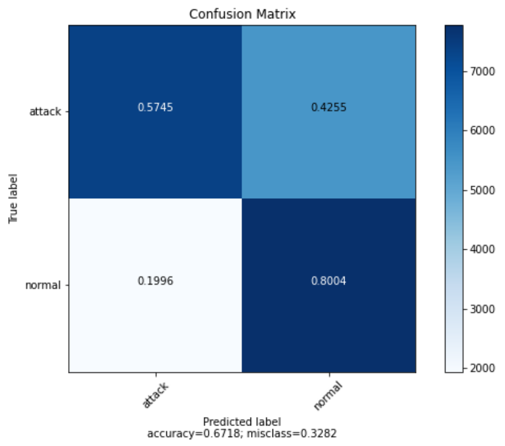
Figure 4.7: Confusion Matrices for the NSL-KDD ML Algorithms (KDDTest+.TXT)



(a) Random Forest Confusion Matrix



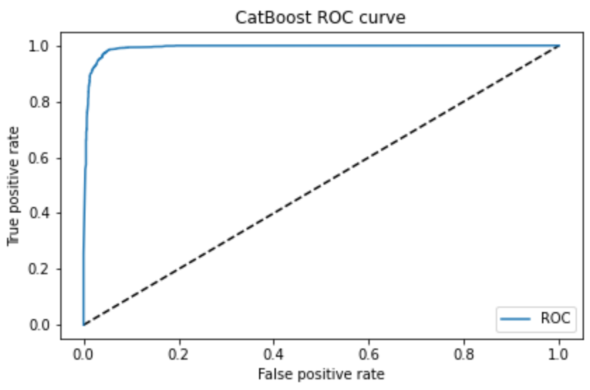
(b) Naive Bayes Confusion Matrix



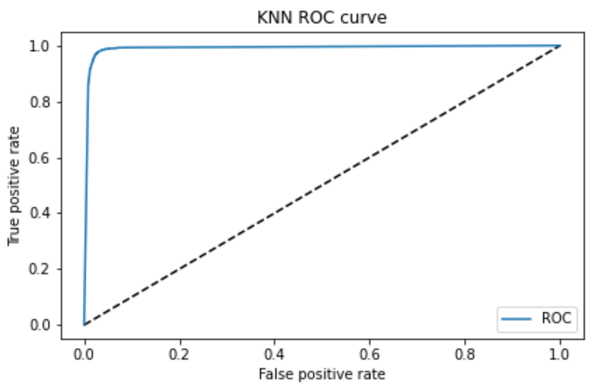
(c) MLP Confusion Matrix

Figure 4.8: Confusion Matrices for the NSL-KDD ML Algorithms (KDDTest+.TXT)

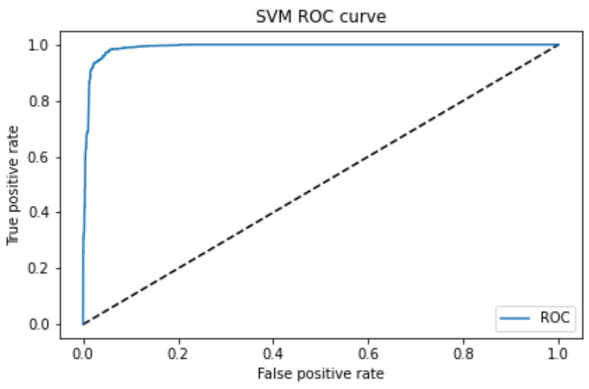
corresponded to better ROC curves as illustrated in Figures 4.11 and 4.12.



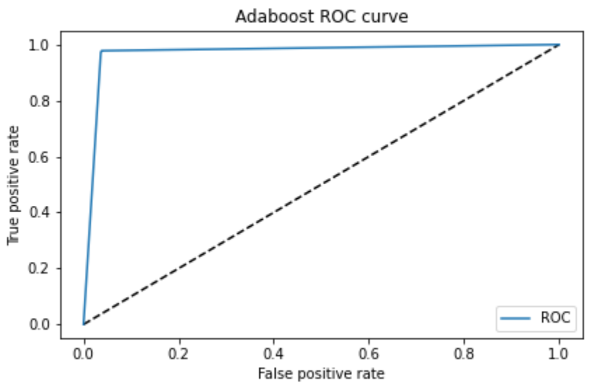
(a) CatBoost ROC Curve



(b) KNN ROC Curve

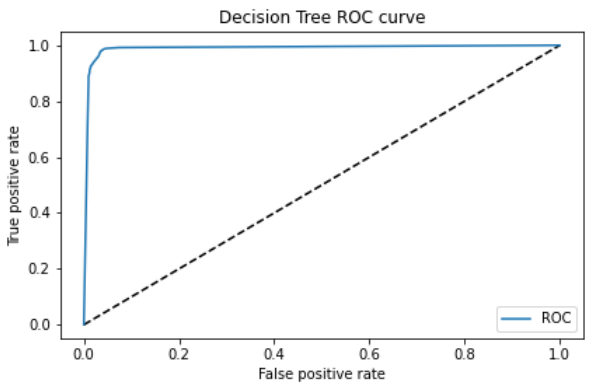


(c) SVM ROC Curve

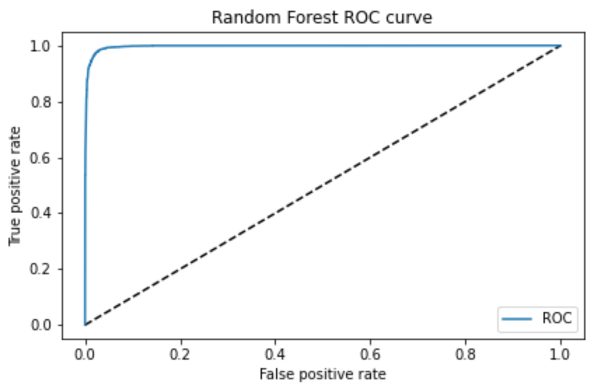


(d) AdaBoost ROC Curve

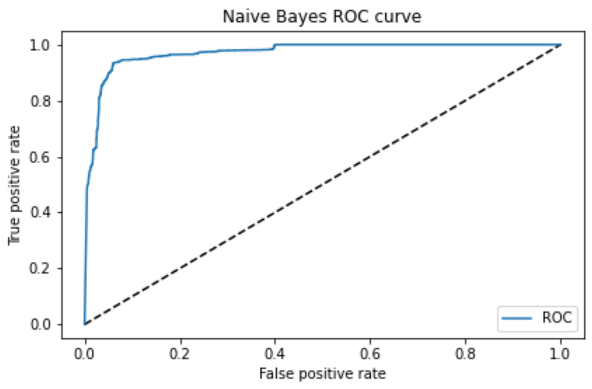
Figure 4.9: ROC Curves for the NSL-KDD ML Algorithms (KDDTrain+.TXT)



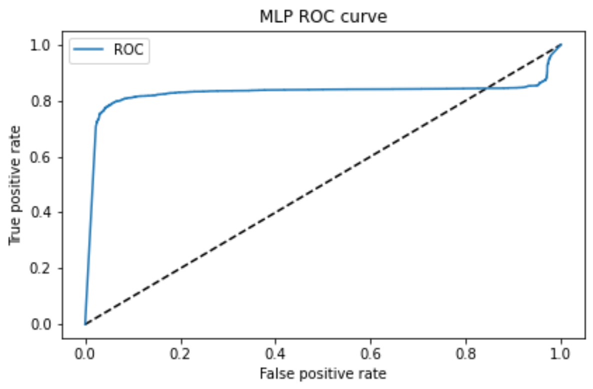
(a) Decision Trees ROC Curve



(b) Random Forest ROC Curve

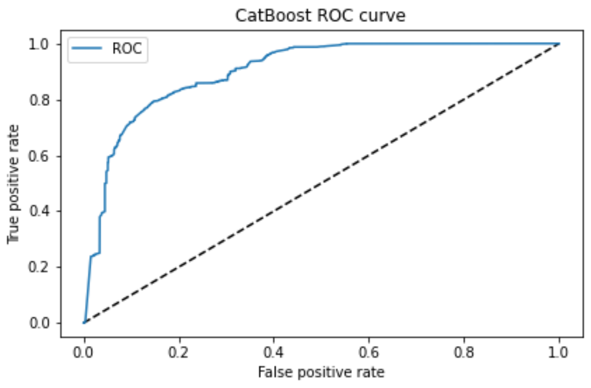


(c) Naive Bayes ROC Curve

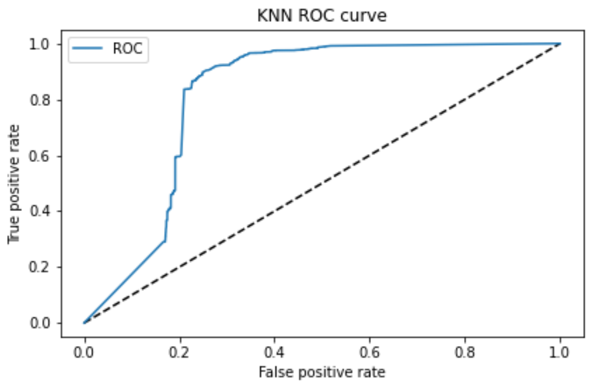


(d) MLP ROC Curve

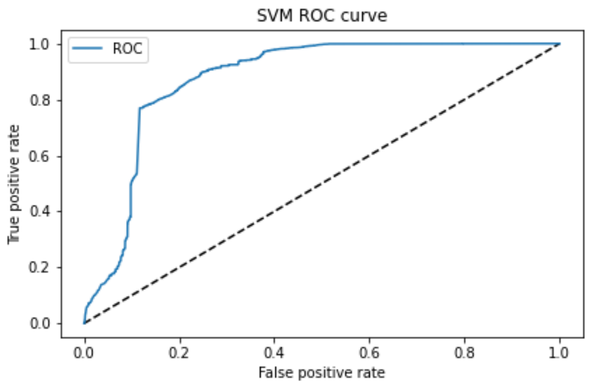
Figure 4.10: ROC Curves for the NSL-KDD ML Algorithms (KDDTrain+.TXT)



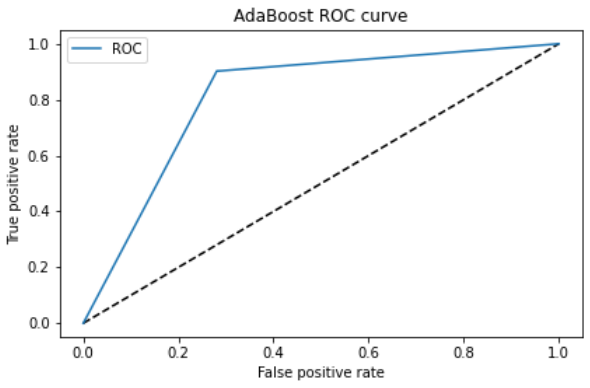
(a) CatBoost ROC Curve



(b) KNN ROC Curve

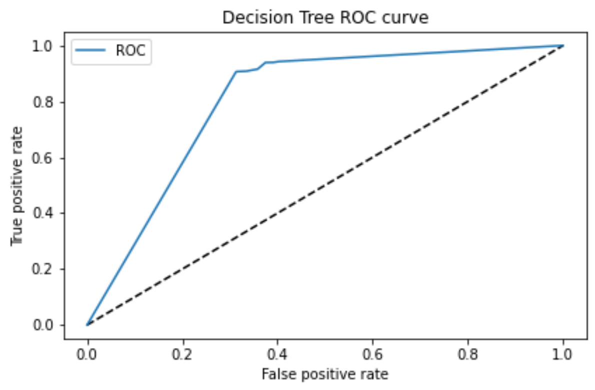


(c) SVM ROC Curve

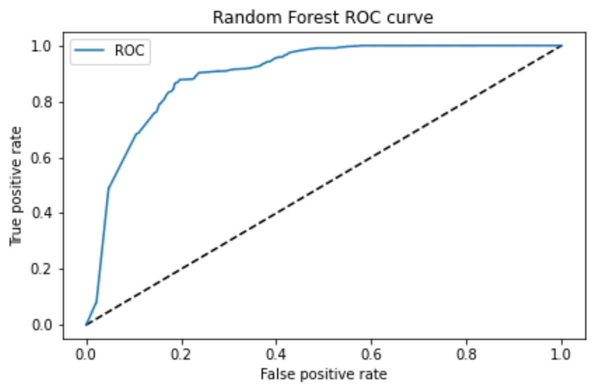


(d) AdaBoost ROC Curve

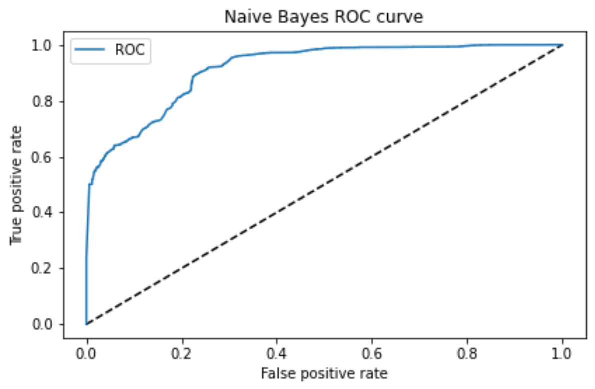
Figure 4.11: ROC Curves for the NSL-KDD ML Algorithms (KDDTest+.TXT)



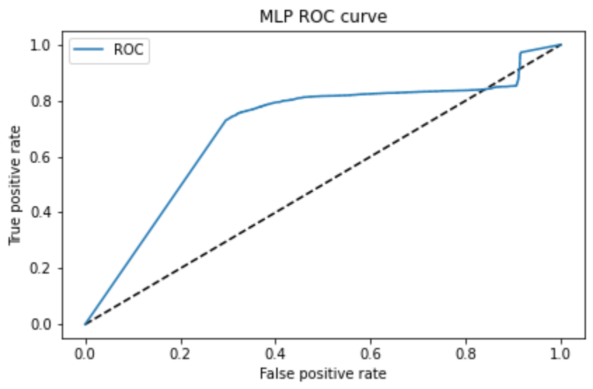
(a) Decision Trees ROC Curve



(b) Random Forest ROC Curve



(c) Naive Bayes ROC Curve



(d) MLP ROC Curve

Figure 4.12: ROC Curves for the NSL-KDD ML Algorithms (KDDTest+.TXT)

Chapter 5

Discussion

5.1 Machine Learning Algorithms

The Decision Tree classifier is the best classifier with regards to both accuracy and speed in the CICIDS2017 and CICDDoS2019 datasets. It also performs relatively well in the NSL-KDD dataset. The Decision Tree classifier achieves high accuracies because decision trees are effective at capturing non-linear relationships between features [47]. They are also faster possibly because of their minimal computational complexity. Their computational complexity is dependent upon the depth of the trees, and in this project the depths of the trees were all below 14, as discussed in section 3.5.6 of this paper. The KNN classifier, though an accurate classifier, was computationally slower in terms of model testing, but computationally faster in terms of model training. The reason for the difference in testing and training times is because KNN is an instance based classifier. Consequently, the training of a KNN model only involves recording where the features are located, but the testing involves calculating the distance of the test features from the training features. The extra calculation step involved in model testing makes the KNN computationally slower when compared to the Decision Tree classifier.

The Naive Bayes classifier performed poorly on the CICIDS2017 and CICDDoS2019 datasets. As described in section 3.5.8 of this paper, the Naive Bayes algorithm assumes that the features in a dataset are strongly independent. Most likely, the features in the CICIDS2017 and the CICDDoS2019 datasets were dependent upon each other, thus making the model perform poorly. This confirms the observation made by Elmrabit et al. [8] which was discussed in the Literature Review in section 2.2.2. Elmrabit et al. [8] found that the Naive Bayes model had the worst performance. As previously mentioned in section 2.2.2 the Naive Bayes had the worst performance in the NSL-KDD dataset according to Note and Ali [2]. However, in this project, the Naive Bayes classifier had the best performance in the NSL-KDD test dataset associated with the KDDTest+.TXT file, as discussed in section 4.4.3 of this paper. The difference in results could be because previous studies have used the algorithm in a multi-class classification context, while in this project, the algorithm was used in a binary classification context. Since the features were either 'attack' or 'normal', the distinction between them may have increased feature independence, thus making the Naive Bayes classifier perform better compared to the CICIDS2017 and CICDDoS2019 datasets. Across all datasets, the Naive Bayes classifier is computationally fast in terms of training and testing time. This is expected, because the algorithm calculates simple probabilities and assumes independence of features, thus making it computationally inexpensive.

The Support Vector Machine (SVM) and Stochastic Gradient Descent (SGD) models had higher percentages of misclassification in comparison to other ML models in the CICIDS2017 and CICDDoS2019 datasets. However, both classifiers had lower percentages of misclassification in the NSL-KDD dataset. This could be due to the difference in the nature of the datasets. The CICIDS2017 and CICDDoS2019 datasets were pre-processed to a multi-class dataset while the NSL-KDD was pre-processed to a binary dataset. The nature of SVM algorithms is that they separate two classes using a boundary. They are therefore usually ideal for binary classifiers but can be used as well in a multi-class context by combining several SVM algorithms [48]. The algorithm's bias towards being a binary classifier may explain why the SVM classifier performs better with the binary NSL-KDD dataset in comparison with the multi-class CICIDS2017 and CICDDoS2019. Since the SGD's loss function was set to 'hinge', which is a linear SVM, then the same reason can be given as to why the SGD performed better in binary classification compared to multi-class classification. The hinge loss function also explains why there is no AUC score or ROC curve associated with the SGD classifier in the results. There are no probability estimates available for the hinge loss function, hence an ROC curve and AUC score that are associated with probabilities could not be obtained. It is also worth noting that the SVM classifier had a relatively worse training time in the NSL-KDD dataset in comparison to the other two datasets. This is because the NSL-KDD dataset was significantly larger than the other datasets because it was not sampled unlike the CICIDS2017 and the CICDDoS2019. The SVM algorithm is much faster in smaller datasets in comparison to larger datasets.

The MLP algorithm, in terms of evaluation metric scores, had one of the best performances in both the CICIDS2017 dataset and the CICDDoS2019 datasets. However, it had the worst performance in the NSL-KDD dataset. This may be due to over-fitting, as the model's ROC curve during testing associated with the KDDTest+.TXT file, shown in Figure 4.12(d), has a much smaller area under the curve in comparison with the ROC curve associated with the KDDTrain+.TXT file, as shown in Figure 4.10(d). The large difference in ROC curves may indicate over-fitting, possibly due to the lack of feature engineering for the MLP model, as discussed in section 3.4 of this paper. This may also explain why the MLP had the slowest training time in comparison to the rest of the classifiers.

5.2 Datasets

The CICIDS2017 dataset had classifier accuracies that ranged between 82% and 99%. The CICDDoS2019 dataset had accuracies that ranged between 76% and 95%. The NSL-KDD dataset had accuracies that ranged from 83% and 98% with the KDDTrain+.TXT file, and accuracies that ranged 67% and 80% with the KDDTest+.TXT file. Even though the CICIDS2017 and CICDDoS2019 datasets are both used in a multi-class classification context, it is difficult to compare their performance. Even though the CICIDS2017 dataset had higher accuracies, the types of attacks presented in both datasets are of significantly different types, hence comparing the two datasets is futile. In terms of the NSL-KDD dataset, classifiers that used test data associated with the KDDTrain+.TXT file performed better than classifiers that used test data from the KDDTest+.TXT file. This is because the KDDTest+.TXT file had extra attacks that were not present in the KDDTrain+.TXT file in order to simulate real world attacks. With regards to feature engineering across the datasets, the most important features that are chosen for model training depend on the dataset chosen, due to the different attack types present in each dataset. This is evidenced by the diverse features shown in Figure 3.6.

Chapter 6

Conclusion

The project's main aim was to conduct a comparison study of several ML algorithms using several cybersecurity datasets and evaluation metrics. The ML algorithms chosen and evaluated in this project include CatBoost, KNN, SGD, SVM, AdaBoost, Decision Trees, Random Forests, Naive Bayes and NLP. The methodology undertaken in this project involved the following of a standard machine learning pipeline and this includes data selection, data pre-processing and cleaning, feature selection, model training, model testing and finally model evaluation. The main finding was that the Decision Tree classifier performed the best in terms of both accuracy and speed in detecting cyber attacks in a multi-class context using the CICIDS2017 and CICDDoS2019 datasets. The Naive Bayes classifier performed the worst in detecting attacks in a multi-class context. However, the Naive Bayes classifier performed the best in detecting attacks in a binary context using the NSL-KDD dataset. It was also discovered that the best features to be selected for model training depend on the selected dataset. Finally, selecting the best dataset to use for predicting cyber attacks was difficult, since the datasets chosen in this project each had different types of attacks, therefore a comparison across datasets was deemed futile. Future work would involve selecting more datasets that have similar attacks, and conducting a comparison study between the datasets with similar attacks. Future work would also involve incorporating datasets with zero day attacks into the comparison study, since the datasets used in this project did not contain zero day attacks, yet it is one of the biggest threats to organizations today [49].

Bibliography

[1] N. Sun, J. Zhang, S. Member, P. Rimba, S. Gao, L. Yu Zhang, and Y. Xiang, "Data-Driven Cybersecurity Incident Prediction: A Survey; Data-Driven Cybersecurity Incident Prediction: A Survey," *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, vol. 21, no. 2, 2019. doi: 10.1109/COMST.2018.2885561. [Online]. Available: http://www.ieee.org/publications_standards/publications/rights/index.html

[2] J. Note and M. Ali, "Comparative Analysis of Intrusion Detection System Using Machine Learning and Deep Learning Algorithms," *Annals of Emerging Technologies in Computing*, vol. 6, no. 3, pp. 19–36, 7 2022. doi: 10.33166/AETiC.2022.03.003

[3] L. Dhanabal and S. P. Shantharajah, "A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, 2015. doi: 10.17148/IJARCCE.2015.4696

[4] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, vol. 2018-January. SciTePress, 2018. doi: 10.5220/0006639801080116. ISBN 9789897582820 pp. 108–116.

[5] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing Realistic

Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST).* IEEE, 10 2019. doi: 10.1109/CCST.2019.8888419. ISBN 978-1-7281-1576-4 pp. 1–8.

[6] I. H. Sarker, A. S. M. Kayes, S. Badsha, H. Alqahtani, P. Watters, and A. Ng, "Cybersecurity data science: an overview from machine learning perspective," *Journal of Big Data*, vol. 7, no. 1, p. 41, 12 2020. doi: 10.1186/s40537-020-00318-5

[7] A. L. Buczak and E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 2, pp. 1153–1176, 4 2016. doi: 10.1109/COMST.2015.2494502

[8] N. Elmrabit, F. Zhou, F. Li, and H. Zhou, "Evaluation of Machine Learning Algorithms for Anomaly Detection," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security).* IEEE, 6 2020. doi: 10.1109/CyberSecurity49315.2020.9138871. ISBN 978-1-7281-6428-1 pp. 1–8.

[9] P. Dixit and S. Silakari, "Deep Learning Algorithms for Cybersecurity Applications: A Technological and Status Review," *Computer Science Review*, vol. 39, p. 100317, 2 2021. doi: 10.1016/j.cosrev.2020.100317

[10] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 20, 12 2019. doi: 10.1186/s42400-019-0038-7

[11] "NSL-KDD — Datasets — Research — Canadian Institute for Cybersecurity — UNB." [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html

[12] I. F. Kilincer, F. Ertam, and A. Sengur, "Machine learning methods for cyber security intrusion detection: Datasets and comparative study," *Computer Networks*, vol. 188, p. 107840, 4 2021. doi: 10.1016/j.comnet.2021.107840

[13] "KDD Cup 1999 Data." [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[14] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, 12 2009. doi: 10.1109/CISDA.2009.5356528. ISBN 9781424437641

[15] A. N. M. B. Rashid, M. Ahmed, L. F. Sikos, and P. Haskell-Dowland, "Anomaly Detection in Cybersecurity Datasets via Cooperative Co-evolution-based Feature Selection," *ACM Transactions on Management Information Systems*, vol. 13, no. 3, pp. 1–39, 9 2022. doi: 10.1145/3495165

[16] R. Patil, H. Dudeja, and C. Modi, "Designing an efficient security framework for detecting intrusions in virtual network of cloud computing," *Computers & Security*, vol. 85, pp. 402–422, 8 2019. doi: 10.1016/j.cose.2019.05.016

[17] C. Khammassi and S. Krichen, "A NSGA2-LR wrapper approach for feature selection in network intrusion detection," *Computer Networks*, vol. 172, p. 107183, 5 2020. doi: 10.1016/j.comnet.2020.107183

[18] F. Gottwalt, E. Chang, and T. Dillon, "CorrCorr: A feature selection method for multivariate correlation network anomaly detection techniques," *Computers & Security*, vol. 83, pp. 234–245, 6 2019. doi: 10.1016/j.cose.2019.02.008

[19] S. M. Kasongo and Y. Sun, "A deep learning method with wrapper based feature extraction for wireless intrusion detection system," *Computers & Security*, vol. 92, p. 101752, 5 2020. doi: 10.1016/j.cose.2020.101752

[20] A. Verma and V. Ranga, "Statistical analysis of CIDDS-001 dataset for Network Intrusion Detection Systems using Distance-based Machine Learning," *Procedia Computer Science*, vol. 125, pp. 709–716, 2018. doi: 10.1016/j.procs.2017.12.091

[21] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," *2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings*, 12 2015. doi: 10.1109/MILCIS.2015.7348942

[22] "IDS 2017 — Datasets — Research — Canadian Institute for Cybersecurity — UNB." [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html

[23] "DDoS 2019 — Datasets — Research — Canadian Institute for Cybersecurity — UNB." [Online]. Available: https://www.unb.ca/cic/datasets/ddos-2019.html

[24] A. Chartuni and J. Márquez, "Multi-classifier of ddos attacks in computer networks built on neural networks," *Applied Sciences (Switzerland)*, vol. 11, no. 22, 11 2021. doi: 10.3390/app112210609

[25] J. Song, H. Takakura, and Y. Okabe, "Benchmark Data." [Online]. Available: http://www.secure-ware.com/contents/product/ashula.html

[26] "Welcome to Python.org." [Online]. Available: https://www.python.org/

[27] "Making the Most of your Colab Subscription - Colaboratory." [Online]. Available: https://colab.research.google.com/?utm_source=scs-index

[28] "Personal cloud storage and file-sharing platform – Google." [Online]. Available: https://www.google.co.uk/intl/en-GB/drive/

[29] "pandas - Python Data Analysis Library." [Online]. Available: https://pandas.pydata.org/

[30] R. Panigrahi and S. Borah, "A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems WEKA Result Reader-A Smart Tool for Reading and Summarizing WEKA Simulator Files View project Intrusion detection system View project A detailed analysis of CICIDS2017 dataset for

designing Intrusion Detection Systems," Tech. Rep. 3, 2018. [Online]. Available: https://www.researchgate.net/publication/329045441

[31] "skimpy · PyPI." [Online]. Available: https://pypi.org/project/skimpy/

[32] "NumPy." [Online]. Available: https://numpy.org/

[33] S. Mishra, "Handling Imbalanced Data: SMOTE vs. Random Undersampling," *International Research Journal of Engineering and Technology*, 2017. [Online]. Available: www.irjet.net

[34] D. Singh and B. Singh, "Investigating the impact of data normalization on classification performance," *Applied Soft Computing*, vol. 97, 12 2020. doi: 10.1016/j.asoc.2019.105524

[35] N. Sharma*, H. V. Bhandari, N. S. Yadav, and H. V. J. Shroff, "Optimization of IDS using Filter-Based Feature Selection and Machine Learning Algorithms," *International Journal of Innovative Technology and Exploring Engineering*, vol. 10, no. 2, pp. 96–102, 12 2020. doi: 10.35940/ijitee.B8278.1210220. [Online]. Available: https://www.ijitee.org/portfolio-item/B82781210220/

[36] A. R. Kharwar and D. V. Thakor, "An Ensemble Approach for Feature Selection and Classification in Intrusion Detection Using Extra-Tree Algorithm," *International Journal of Information Security and Privacy*, vol. 16, no. 1, pp. 1–21, 1 2022. doi: 10.4018/i-jisp.2022010113

[37] "scikit-learn: machine learning in Python — scikit-learn 1.1.2 documentation." [Online]. Available: https://scikit-learn.org/stable/

[38] M. M. Muhammed, A. A. Ibrahim, R. L. Ridwan, R. O. Abdulaziz, and G. A. Saheed, "Comparison of the CatBoost Classifier with other Machine Learning Methods,"

Tech. Rep. 11, 2020. [Online]. Available: https://www.researchgate.net/publication/348277609

[39] A. Patle and D. S. Chouhan, "SVM kernel functions for classification," in *2013 International Conference on Advances in Technology and Engineering (ICATE).* IEEE, 1 2013. doi: 10.1109/ICAdTE.2013.6524743. ISBN 978-1-4673-5619-0 pp. 1–9.

[40] S. S. Azmi and S. Baliga, "An Overview of Boosting Decision Tree Algorithms utilizing AdaBoost and XGBoost Boosting strategies," *International Research Journal of Engineering and Technology*, 2020. [Online]. Available: www.irjet.net

[41] Z. Zhang, Z. Zhao, and D. S. Yeom, "Decision Tree Algorithm-Based Model and Computer Simulation for Evaluating the Effectiveness of Physical Education in Universities," *Complexity*, vol. 2020, 2020. doi: 10.1155/2020/8868793

[42] S. Misra and H. Li, "Noninvasive fracture characterization based on the classification of sonic wave travel times," in *Machine Learning for Subsurface Characterization.* Elsevier, 2020, pp. 243–287.

[43] N. Moustafa and J. Slay, "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set," *Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 4 2016. doi: 10.1080/19393555.2015.1125974

[44] Honkela Antti, "Multilayer perceptrons," 2001. [Online]. Available: http://users.ics.aalto.fi/ahonkela/dippa/node41.html

[45] H. Ramchoun, M. Amine, J. Idrissi, Y. Ghanou, and M. Ettaouil, "Multilayer Perceptron: Architecture Optimization and Training," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 4, no. 1, p. 26, 2016. doi: 10.9781/ijimai.2016.415

[46] "Matplotlib — Visualization with Python." [Online]. Available: https://matplotlib.org/

[47] L. Auret and C. Aldrich, "Interpretation of nonlinear relationships between process variables by use of random forests," *Minerals Engineering*, vol. 35, pp. 27–42, 8 2012. doi: 10.1016/j.mineng.2012.05.008

[48] G. J. J. Van Den Burg and P. J. F. Groenen, "GenSVM: A Generalized Multiclass Support Vector Machine," Tech. Rep., 2016. [Online]. Available: http://www.stat.osu.edu/

[49] M. Inzimam, C. Yongle, and Z. Zhang, "An Efficient Approach towards Assessment of Zero-day Attacks," *International Journal of Computer Applications*, vol. 177, no. 26, pp. 975–8887, 2019.