ADDIS ABABA UNIVERSITY

COLLEGE OF NATURAL SCIENCES

Modeling Network Intrusion Detection System Based on Anomaly Approach Using Machine Learning Techniques

Tigist Hidru

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

March, 2020

ADDIS ABABA UNIVERSITY

COLLEGE OF NATURAL SCIENCES

Tigist Hidru

Advisor: Solomon Gizaw (PhD)

This is to certify that the thesis prepared by *Tigist Hidru*, titled: *Modeling Network Intrusion Detection System Based on Anomaly Approach Using Machine Learning Techniques* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name	**Signature**	**Date**
Advisor: Solomon Gizaw (PhD)	_____	_____
Examiner: Mulugeta Libsie (PhD)	_____	_____
Examiner: Dida Midekso (PhD)	_____	_____

Abstract

With the rapid growth use of information technology today, hacking and other unauthorized activities have dynamically increased than ever before. With the development of hardware and software, attacks are growing exponentially in type and number. Nowadays, network traffic classification has essential significance, due to the high growth of Internet users. A lot of threats are created every day by individuals and organizations to attack computer networks to steal private information and data. To protect these attacks, many organizations put into practice a broad defense such as configuring a strong firewall, authentication systems, encryption, antivirus, latest hardware and so on. Intrusion detection is another mechanism that is used to mitigate network intrusions. Many Intrusion Detection Systems have been developed for monitoring and detecting network or systems against any suspicious activity. In most of them, low detection rate, high training time, and a relatively high false alarm rate are obtained.

To overcome the problems, we proposed an approach that integrates the concepts of machine learning, big data and anomaly detection for obtaining better results with improved processing speed. The proposed system has training, validation and testing main components. In the training component, the collected training data is preprocessed and fed to the classification model. Four classification models: Random Forest, Neural Network, Logistic Regression, and Decision Tree are used and compared. In the validation component, hyperparameter tuning is done using 5-fold cross-validation with a grid search technique for each of the machine learning algorithms to find the optimal value for each hyperparameter to improve the detection rate of the models. Then, the classification models are trained using the best parameters to build the final model. Finally, the final trained model is used to classify the test data into normal or attack. All of the classification models are implemented on Apache Spark big data framework.

The experimental work is carried out using the NSL-KDD dataset which contains normal and attacks data. We split the dataset into 68% for training, 17% for validation and 15% for testing. The results show that almost all the algorithms give high prediction results. Among the algorithms, Neural Network has acquired the best result which is 99.9% accuracy, 99.8% precision, 99.7% recall, and 99.7% f1-score.

Keywords: Intrusion Detection System, Classification, Machine Learning, Neural Network, Anomaly Detection, Apache Spark

Acknowledgments

Firstly, I would like to thank God for everything. I would also like to show my sincere gratitude to my advisor Dr. Solomon Gizaw for his initial guidance and continuous support during my thesis progress.

Besides my advisor, I would like to thank Mr. Erdey Syoum for his constant help and he is always available to discuss and provide insightful comments on my work throughout my program. It was a pleasure working with him and also a wonderful learning experience.

Last but not least, I would like to thank my family and my friends: many thanks to my mother Tadelech, for her never-ending prayers and good wishes which keeps me focused towards my goal, and my elder brother Hailemariam, for always standing by me and guiding me in my academics and life in general and thanks to my classmates and friends for their support and love.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AR	Attribute Ratio
CART	Classification and Regression Tree
CFS	Correlation-based Feature Selection
CIA	Confidentiality, Integrity, and Availability
DOS	Denial of Service
DT	Decision Tree
FAR	False Alarm Rate
FN	False Negative
FP	False Positive
GB	Gradient Boosted
HIDS	Host-Based Intrusion Detection System
IDS	Intrusion Detection System
IG	Information Gain
KDD	Knowledge Discovery Dataset
LR	Logistic Regression
MI	Mutual Information
ML	Machine Learning
MLlib	Machine Learning Library
MLP	Multilayer Perceptron
NB	Naïve Bayes
NIDS	Network-Based Intrusion Detection System
NN	Neural Network
NSL-KDD	Network Services Library Knowledge Discovery Dataset
PCA	Principal Component Analysis
R2L	Remote to Local
RDD	Resilient Distributed Dataset
RF	Random Forest
SOM	Self-Organizing Map
SPLR	Space Logistic Regression
SVM	Support Vector Machine
U2R	User to Root
WEKA	Waikato Environment for Knowledge Analysis

1. INTRODUCTION

1.1 Background

The rapidly growing Internet-based technology has brought great benefits to society. The Internet transformed our lives by providing many services such as communication, possibilities to access information, business transactions involving financial data, product development and marketing, storage of sensitive company information, and other services [1]. Nowadays, the information is just a few clicks from one's web browser. The various social networking sites have made communication easier by eliminating geographic distance. Moreover, the Internet provides us online services such as online banking, online learning, and online shopping; all these activities made our lives more convenient.

Although the Internet has made our lives much more convenient, its vulnerabilities and the amount of information communicated over it generate opportunities for intruders to perform malicious activities within it. With the continuous growth of computer networks and Internet users, network traffic data have become increasingly complex for management, causing some serious challenges for safe and reliable use of the Internet by individuals and institutions. Based on the report in [2], over the years there has been a great increase in network traffic and a corresponding rise in network intrusion or cyber-attacks. Based on the Cisco reports, the size of the global Internet traffic will reach 396 exabytes per month by 2022, up from 122 exabytes per month in 2017, i.e., 4.8 zettabytes of traffic per year by the year 2022 [3]. The extensive growth in using the Internet in social networking sites (e.g., instant messaging, video conferences, twitter, etc.), health care, e-commerce, bank transactions, and many other services are being observed nowadays. These Internet applications need a suitable level of security and privacy. In addition to this, there is an increasing availability of tools and software for attacking and intruding networks.

Network intrusion or cyber-attack is an activity or set of actions that compromise the normal functioning of a computer network [4]. A lot of threats are created every day by individuals and organizations to attack computer networks to steal private information and data. To protect these attacks, many organizations put into practice a broad defense such as configuring a strong firewall, authentication systems, encryption, antivirus, and so on. Intrusion detection is another mechanism that is used to mitigate network intrusions. Although more effective methods of data processing are developed to protect against network threats, intruders constantly create new attacks with dynamic features. To

overcome these in recent years there have been various attempts to propose efficient intrusion detection systems. Intrusion detection system (IDS) is a system that monitors and detects the network or the system against any suspicious activity harming networks confidentiality, integrity and availability (CIA) [2]. This includes monitoring of unauthorized utilization of network resources by an unauthorized user and keeps them available to legitimate users. IDS can be classified into several ways and some of them are:

- Signature-based and anomaly-based IDS [5]: Signature-based IDS is an IDS approach in which detection of intrusion is based on the signatures of known attacks stored in the database and identifies suspicious data by comparing new instances with the stored signatures or patterns of attacks in the database. Anomaly-based IDS uses machine learning (ML) to create a model of normal activity, and then compare new behavior against this model. Since the models can be trained according to the application and hardware configurations, machine learning-based methods can have better generalizability in comparison to signature-based IDS. Also, anomaly-based detection can detect unknown/new attacks in the network because it depends on the rules as opposed to signature-based IDS.
- Network-based and host-based IDS [6]: Network-based IDS monitors and detects network traffic for malicious activity that attempted to break into or compromise a network. The network-based IDS analyze both inbound and outbound packets and searches for suspicious patterns and if any malicious activity is found, an alarm is generated. The IDS program is usually deployed on switches, hubs, routers, and any other point where multiple systems are networked together. Whereas, the host-based IDSs are less concerned with detecting attacks from the network rather they continually look at system logs, critical system files, and other resources that may be monitored for any suspicious activity. Host-based IDSs are usually installed in critical hosts and servers that require extra layer of protection.

Our research work focused on anomaly-based network intrusion detection system (NIDS). Anomaly approaches use a statistical method to learn the behavior of features and then to identify the attack that deviates from the normal behavior. Since the main goal of any IDS is to monitor all the data coming to the switch's interface, this approach needs, first capturing all packets that pass through the switch and then analyzing these packets to detect a deviation in behavior.

Recent studies in this area have been applied to machine learning algorithms to improve IDS by evaluating the various machine learning algorithms on a dataset to identify their performance and select the efficient one to use. In this study, the NSL-KDD dataset is selected as experimental data. An intrusion detection model is developed using random forest (RF), logistic regression (LR), neural network (NN), and decision tree (DT). For the fast processing of the machine learning algorithms, Apache Spark [7] which is a big data technology is used. Besides this, the performance of the algorithms is improved by using hyperparameter tuning and feature selection methods.

1.2 Motivation

Today, the increasing Internet traffic has resulted in increased concern about network security across businesses as attacks also dynamically increased. As a result, the existing security mechanisms of the Internet may not be able to efficiently secure the network. Hence, it is important to find solutions to easily monitor the network, so as to address them using a combination of various methods. An IDS has become an alternative solution for monitoring computer systems or computer networks for malicious activities or events. In due course, IDS applications are expected to detect intrusions that come from inbound and outbound of the network.

This work is motivated based on the issues: low detection accuracy and performance of the existing IDS. To alleviate the problem of the low performance, a fast data processing framework which is called Apache Spark is provided. This computing framework provides immediate results and eliminates delays that can be incurable for business processes. This is due to its parallel in-memory data processing. Besides, feature selection and some best parameter selection methods are provided to enhance the detection accuracy and performance of the previous works.

1.3 Statement of the Problem

Today, detection of security threats, commonly referred to as intrusion, has become a very important and critical issue in network, data and information security. A network intrusion is an activity that tries to compromise the normal functioning of a computer network [8]. Intrusion detection has become a very essential component in a computer or network security. To detect network intrusions, we need to have a mechanism called IDS, which is a method to mitigate or report these intrusions.

Based on the literature review, many intrusion detections related papers specific to machine learning-based IDSs have been developed. However, the existing IDSs still have their own shortcomings. Some of the shortcomings are, low detection rate, high training time, low processing speed, relatively high false alarm rate (FAR), and most IDSs use a signature-based approach that detects only known attacks and needs continuous updating of the database to detect new attacks in a network.

To overcome these challenges, in recent years, there have been various attempts to propose efficient IDS. But still, there is a need for further improvement in these systems. The proposed research work focused on developing an intrusion detection model with better detection rate, less training time, and improved performance by performing the training in parallel and selecting the best parameters for models that can improve the performance of the models.

1.4 Objectives

General Objective

The general objective of this research work is to design and implement a model for classification of network traffic based on the anomaly approach using machine learning techniques.

Specific Objectives

The specific objectives of this research work are to:

- Explore the latest dataset that will be used for model training and testing.
- Preprocess the dataset and identify the best features for model training and testing.
- Explore different machine learning classification algorithms and train them on the prepared training data.
- Test and analyze the results obtained.
- Select the best classification model based on the evaluation results.

1.5 Methods

- **Literature Review:** A detailed review will be done on journal articles, conference proceedings, books, dissertations, and the Web to have a deep understanding of the previous works in the study area. Different methods and techniques in network

intrusion detection will be examined. From the review of these works, appropriate techniques and methods will be selected.

- **Data Collection:** For implementing the proposed model a dataset that has the latest attack and normal data with less redundant records from the Web will be collected through analyzing different documents.
- **Tools and Development Environments:** Different free and open-source tools will be used during model development. Python programming language with Apache Spark (PySpark) will be used to implement the system. The entire implementation will be done in Anaconda environment using Spyder.
- **Testing and Evaluation:** Every trained model will be tested on new data that is unseen during the training to determine how well it classifies the samples correctly. The performance of the models will be evaluated using classification models evaluation metrics.

1.6 Scope and Limitations

This thesis is conducted to design and implement a model for classification of network traffic into normal and attack based on the anomaly approach using machine learning techniques. Four classification algorithms that are RF, DT, LR, and NN are selected from the PySpark software for the implementation and compared their result based on different classification metrics. To train and test these models, feature selection methods are utilized to determine the most discriminant features for the collected dataset. The trained models classify the input data into binary and five-classes.

One of the limitations of this research work is that only one dataset (which contains network traffic data) is used to build models that can classify the input data into normal or attack. Also, one feature selection method is applied to the dataset to select relevant features for model building and testing.

1.7 Application of Results

There are several technologies and methods to safeguard businesses' and organizations' networks from malicious activities. A network IDS is one of these mechanisms which protect a system from network-based threats by reading all packets and searches for any suspicious patterns and when threats are discovered it notifies the administrator or

excluding the source from accessing the network. The result of this study can help organizations to protect their network from external and internal attacks.

1.8 Organization of the Rest of the Thesis

This section is composed of the organization of the rest of the chapters. In Chapter 2, reviews of concepts and methods relevant and related to the proposed research are described. It discusses the various approaches to the IDS and methods used in developing the proposed system. In Chapter 3, research works related to this area including the methods used, objectives of the study, the procedures used, and the key findings of the paper are presented. Chapter 4 discusses the overall system architecture and detail description of each of the components. This chapter also described the performance evaluation metrics used for evaluating the proposed system. In Chapter 5, the programming tools and software used to develop the model, the dataset used, implementation details, and the experimental results are presented. Finally, Chapter 6 presents the conclusion, the contribution, and gives future work which the study can be extended for future.

2. LITERATURE REVIEW

In this Chapter, an overview of IDS, common approaches of IDS along with their strengths and weakness, the basic component of IDS, network attacks, general concepts to machine learning, review of some classification algorithms are discussed. Then, feature selection methods in machine learning are discussed. Finally, some classification algorithms evaluation metrics and parameter optimization metrics are given. These extensive works of literature have been reviewed to understand the problem with these works and to identify a solution in this work.

2.1 Overview of Intrusion and IDS

Intrusion is defined as malicious activity that compromises the security of a network or computer system either externally or internally without the mandatory privileges [9]. It causes loss of the CIA of a network or computer system. For instance, an intrusion may compromise the CIA of a network by gaining the root-level access and then modifying and stealing the information of the network. In today's world of ever-increasing Internet connectivity, there is an ongoing threat of intrusion, denial of service attacks, or other abuses of computer and network resources [10]. In order to overcome these problems, there are various network security measures such as firewalls, encryption, antivirus, and so on. IDS is another mechanism to secure computer systems or networks. IDSs monitor the events occurring in a computer system or network for analyzing the patterns of intrusions [11].

2.2 Categories of IDS

IDSs can be categorized into various categories based on different criteria as shown in Figure 2.1 [12]. Based on the intrusion detection approach, IDS is classified as anomaly-based and signature-based IDS; based on the source of information as host-based and network-based IDS; and based on structure (or operation) IDS can be categorized as centralized and distributed IDS. In centralized IDS, all operations are performed at a central location while in distributed IDS operations are distributed among hosts of the system. Based on the response after the detection of attacks IDS can also be classified into passive and active IDS. Passive IDS only raise an alarm and does not take any corrective actions after the detection of attacks whereas active IDS raises an alarm and can take corrective actions after detection of the attack.

The detailed discussion of anomaly-based, signature-based, network-based and host-based IDSs are presented in the next sections.

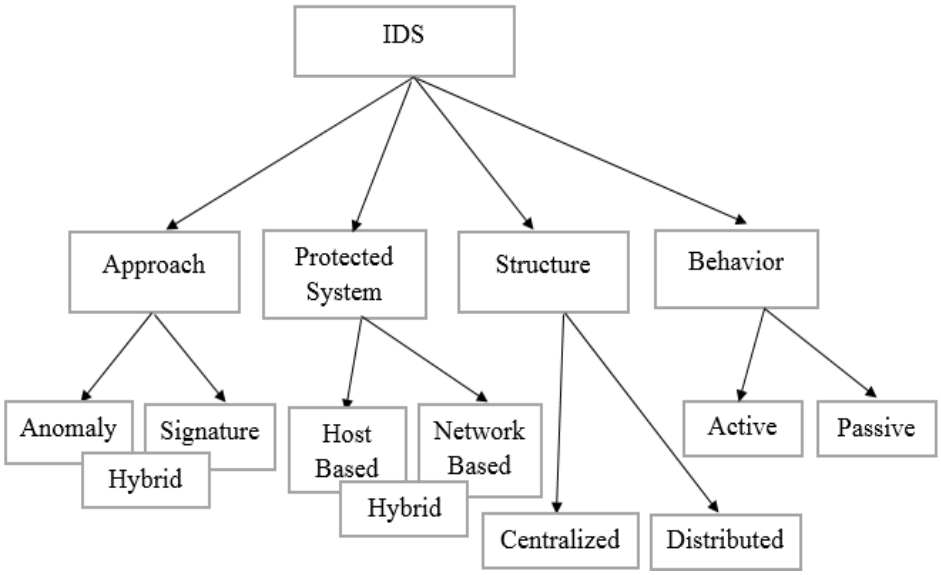

Figure 2.1: Categories of IDS from four different point of views

2.2.1 Anomaly-Based IDS

Anomalies attempt to break a computer system or a network by applying a malicious activity in order to gain access to the system or compromise it [13]. Such events must be detected in proactive manner in order to guarantee the CIA of the resources of a computer system. Anomalies may be caused due to many reasons such as malicious activities that are initiated by attackers or intruders, misconfiguration of a network or a host, overloading of networking infrastructure, and malfunctioning of network devices [14]. Therefore, detecting network attacks is an important step in preventing these dangerous events.

Anomaly-based intrusion detection is one of the approaches to IDS that enables us to detect and classify network anomalies. An anomaly detection approach makes use of profiles that describe the services and resources of each authorized user or a group that normally accesses the network and builds a model. After the model of normal profiles is created, the system can monitor users for suspicious activity by comparing the model with the detected behavior. Once deviations from the normal behavior occurred, the system generates an alarm. The major benefit of the anomaly-based detection system is its capability of detecting new attacks and high detection rate. This type of intrusion detection approach could also be feasible even if there is lack of signature patterns and also works in the condition of non-regular patterns of traffic. One of the drawbacks of anomaly-based

intrusion detection is defining the rule sets. The efficiency of the system depends on the effective implementation and testing of the rule sets on all the protocols. The various protocols can make rule defining complex. However, with proper incorporation of rules and protocols, the anomaly detection approach can perform more efficiently [15].

2.2.2 Signature-Based IDS

In contrast to anomaly detection systems, which triggers alarms based on deviations from the normal network behavior, signature detection systems, also known as misuse detection, trigger alarms based on characteristic signatures of known attacks [16]. Here, a signature means a set of characteristics such as IP numbers, TCP flags, port numbers, logons to a network and other events on the network. This approach identifies instances of attacks by comparing current activity against the signatures of known attacks stored in a database and it generates an alarm if any malicious activity is found. It is more efficient to detect attacks whose patterns are already maintained in the database. The model usually doesn't generate many false alarms since it is specifically designed to detect known attacks. However, this approach cannot detect unknown attacks that are not in the model which results in a low detection rate [17]. Another problem with this approach is that the system needs to be continuously updated in order to add other attacks. Even if there exist well-designed signatures, signature-based IDS can't detect unknown attacks due to the lack of an adequate signature database. This approach can be used by organizations that are more concerned with monitoring known attacks from intruders trying to access internal hosts from the Internet.

A few researches such as [18, 19, 20] utilized a hybrid intrusion detection (the combination of an anomaly and signature detection). This combination can achieve the goals of a high detection rate with anomaly detection and a low false positive rate with signature-based detection for IDSs. However, the challenge with this detection method is getting the components working together. For developing IDSs, a large amount of traffic data is always necessary to be collected in advance for analysis with the anomaly or signature detection approaches. Based on the collected network audit trail, signature detection techniques specify well-defined attack signatures and anomaly detection techniques establish acceptable usage profiles to differentiate intrusions and normal activities from future network traffic data.

2.2.3 Network-Based IDS (NIDS)

A network-based IDS (NIDS) is a software application that monitors and detects network traffic for malicious activity attempted to break into or compromise a network [14]. A NIDS is placed at strategic points within the network to monitor traffic to and from all devices on the network. It performs analysis of passing traffic on the entire LAN, and matches the traffic that is passed on the LANs for identification attacks. Once an attack is identified, an alert can be sent to the administrator. For example, as can be seen from Figure 2.2 [21], an IDS is installed at three LANs in the network. The IDS installed at each LAN attempts to determine unauthorized access to the LANs by analyzing traffic that passes through the network (LAN) to several hosts. The IDSs read both inbound and outbound packets and analyze the packets for suspicious patterns and if any malicious activity is found, an alarm is generated to inform administrators something is wrong to the network and then to take corrective action [22].

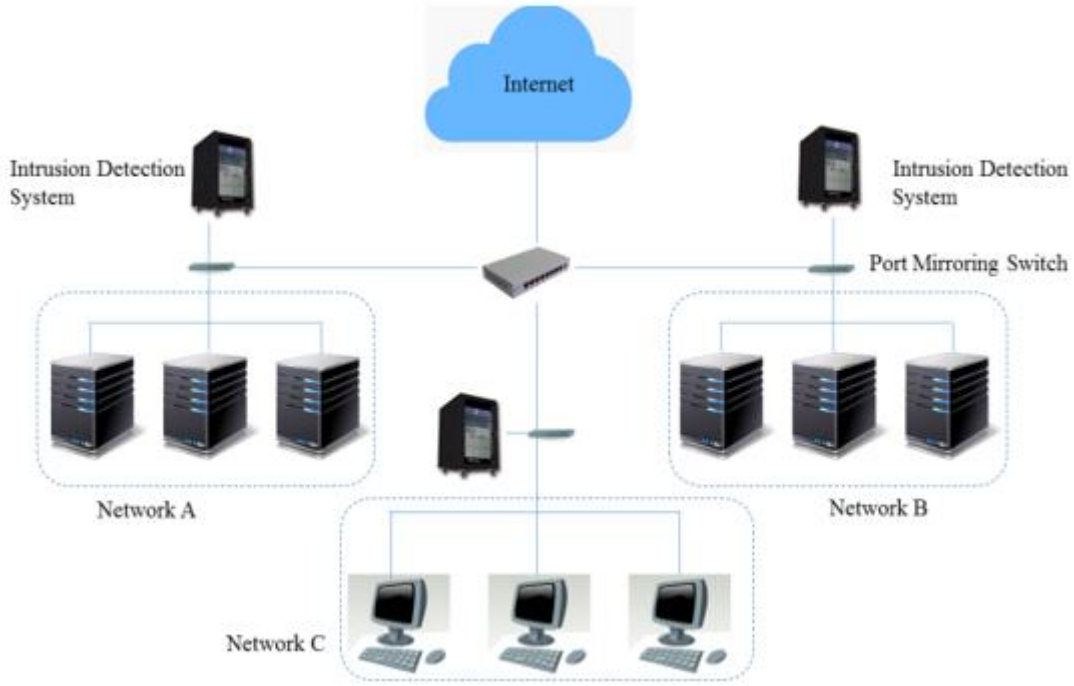

Figure 2.2: A diagram showing the NIDS

NIDS is composed of a set of components that are specialized for network use such as sensors, management servers, a command console, and a database of signatures to examine network traffic on segments [23]. The role of the sensor is to filter information and discard any irrelevant data obtained from the event set associated with the protected system, thereby detecting suspicious activities. Knowing the position where the NIDS sensors are

installed or placed on the network is important. A NIDS sensor can be placed at the critical sections: behind the firewall and before the LAN, between the firewall and the demilitarized zone or on any network segment.

2.2.4 Host-Based IDS (HIDS)

A host-based IDS (HIDS) is an IDS that operates within a computer, node or device. It mainly monitors and analyzes the internals of a computer or system to check if a computer or system is compromised by malicious activity [22]. Once a suspicious activity is detected, the HIDS can either generate an alarm or place the encountered suspicious activity in the logs. Unlike NIDS where it is designed to monitor and detect attacks flowing over the network, in HIDS a trained model or agent is placed on every host (could be web server, computer, firewall, switch, or router) to identify intrusions by analyzing system calls, application logs, file-system modifications such as password files and other host activities. Host-based IDS have the ability to detect internal attacks as well [24].

HIDS is deployed only on sensitive or critical-hosts because it is expensive to place on every host in the network. In summary, HIDS provides only data relating to the host on which it is installed, not the network as a whole. The drawback of this IDS is that installing the software on several hosts on the network takes time and can be more expensive.

NIDS and HIDS can be combined to give better performance by utilizing the strength of the approaches integrated to overcome the limitations of individual techniques. In hybrid IDS, the NIDS collects and processes network traffic in order to detect attacks and the HIDS processes the packets addressed on each host. Even though this integration is useful, getting disparate systems to work in a coordinated manner and analyzing data gathered from multiple systems is somewhat difficult [25].

2.2.5 Centralized IDS

In Centralized IDS all of the IDS activities are controlled directly by a central console with powerful computation resource and large storage/memory space. This central console must have the ability to monitor all the activities in the network, so that the deployed IDS agent can access the network activity data in real time and detect intrusions by analyzing the network activities instantaneously. The IDSs take lower monitoring and administration cost, however, this IDS is not able to detect malicious events occurring at different places at the same time [26].

2.2.6 Distributed IDS

Distributed IDS (DIDS) consists of multiple IDS over a network all of which communicates with each other or with a central server that enables network monitoring. It has capability to aggregate information from different source to detect against a network attacks such as doorknob and distributed denial of service attacks (DDoS). This IDS consists of components such as IDS agent, communication component and central analysis server [26]. An agent is a piece of software that reports attack information to the central analysis server. The use of multiple agents across a network allows the incident analysis team a broader view of the network than can be achieved with single IDS systems. In the communication component the different components of a DIDS are communicated by sending messages to obtain a global view of the system. The central analysis server is the main component that perform the main operation of the system. This allows the interactive querying of attack data for analysis as well as a useful Web interface to see the current attack status of a network. It also allows analysts to perform pre-programmed queries, such as attack aggregation, statistics gathering, to identify attack patterns and to perform incident analysis, all from a Web interface [27]. One of the advantages of this IDS is the ability to detect attack patterns across an entire corporate network, with geographic locations separating segments by time zones or even continents. It reduces computational costs. Another advantage is monitoring, analyzing, and processing of attack data is easier and speeder as compared to the centralized IDS. But, data flowing from the host and the agent may produce high network traffic overheads.

2.2.7 Passive and Active IDS

A passive IDS is a system that is configured only to monitor and analyze network traffic activity and alert an operator to potential vulnerabilities and attacks. The notification can be in an email, a text message, a pop-up window, or a notification on a central monitor. It is not capable of performing any protective or corrective functions on its own.

An active IDS (commonly known as an intrusion prevention system or IPS) is a system that is configured to automatically block suspected attacks in progress without any intervention required by an operator. IPS has the advantage of providing real-time corrective action in response to an attack but the IPS itself is susceptible to attack [28].

2.3 Components of IDS

An IDS usually consists of three main components [29].

- **Data preprocessor:** This component is responsible for collecting and providing the audit data that will be processed by the next component (analyzer). It is mainly concerned with collecting data from desired sources and converting the data into a format that is compatible with the detector. The data used for detecting intrusions can be of a user access patterns such as a sequence of commands requested at the terminal, network-level features (such as the source and destination IP addresses, types of attacks, and duration of packets) and system-level features (such as a sequence of system calls generated by processes).
- **Analyzer (Intrusion Detector):** An analyzer or intrusion detector is the main component of an IDS that analyzes data provided from the data preprocessor to detect attacks. This can include various intrusion detection techniques such as data mining, statistical-approach, machine learning and pattern matching to analyze and detect intrusions.
- **Response Engine:** This component determines how to respond when an analyzer detects an attack. The system may generate an alarm without taking any action against the source or may block the source for a certain period of time depending upon the predefined security policy of the system.

2.4 Types of Attacks

As the Internet is open for anyone, this creates opportunities for unauthorized users to perform malicious activities within it. A lot of threats are created every day by individuals and organizations to attack computer networks to steal private data and information. Attacks can be internal or external [30]. External intruders are unauthorized users of the systems they attack, whereas internal intruders are legitimate users of the organization which has permission to access the system, but do not have the privilege to root or super-user. There are many kinds of attacks to mention a few common types [31]: denial of service (DoS), a user to root (U2R), remote to local (R2L), and Probe attacks.

Denial of Service (DoS) Attack: This class of attack blocks access to certain resources (computing or memory) by making unavailable the network services so that legitimate users cannot access the resource allowed to them. The attacker creates an overloading

traffic through malicious bot to a targeted IP address and floods the network with more requests than the server can process [31].

User to Root (U2R): User to Root attack attempts to gain the systems super-user privilege by trying to access normal users' account either by sniffing or by dictionary attack. After that it steals a legitimate user's password to get super user's privilege [32].

Remote to Local (R2L): Remote to local attack is launched by an attacker to gain unauthorized access to the victim machines in the entire network. This attack is similar to U2R in that both of them try to get others' legitimate password without the legitimate users' knowledge, unlike U2R it attempts to send packets to a remote machine or a network. After gaining the password, the attacker tries to access the machine either as a root or as a normal user [33].

Probe: Probe attack aims at gaining useful information about hosts, their valid IP address, the services they offer that helps them later to identify the vulnerabilities of the system to launch an attack against the system services [33].

From the above description of the four attack categories, the DoS attack acts differently from the three attacks, where the DoS attempts to shut down a system to stop traffic flow, whereas the other three slowly penetrate the system.

2.5 Artificial Intelligence

Artificial Intelligence (AI) was born in the 1950s when a handful of pioneers from the nascent field of computer science started asking whether computers could be made to think which is still being explored today. AI is a general field that encompasses machine learning and deep learning, but it also includes many more approaches that don't involve any learning [34].

2.5.1 Machine Learning (ML)

Machine Learning (ML) was born from the theory that computers can learn without being programmed to perform specific tasks. It came into prominence perhaps in the 1990s when researchers and scientists started giving it more prominence as a sub-field of AI that perform far better compared to using fixed rule-based models requiring a lot of manual time and effort [35]. In traditional computing, input data is fed to a program to generate output. But in ML, input data and output data are fed to the ML algorithm to generate a

function or program that can be used to predict the output of input according to the learning done on the input/output dataset fed to the ML algorithm [36]. A ML system is trained rather than explicitly programmed and it needs to learn from the historical data, optimize for better computations, and also generalize themselves to provide proper results.

Nowadays, machine learning algorithms are employed for classification, regression, clustering or dimensionality reduction tasks of large datasets with high-dimension. As a result, the task of detecting network intrusions comes under ML as it involves the classification data into normal and abnormal behavior. Network traffic data is increasing; thus, ML helps to recognize the complex patterns in the given large datasets by using a learning mechanism to make decisions or predictions when new data instances are coming. ML algorithms can be classified based on the outcome of the algorithm and types of input fed during training as supervised, unsupervised, or semi-supervised learning [37].

2.5.1.1 Supervised Learning

Supervised learning is a process by which a function is derived with the help of labeled data samples also known as training data [38]. The algorithm analyzes the training data and produces a function, which will be used for input to output mapping of a new example. This learned knowledge can be then used in the future to predict an output y' for any new input data samples x' which was previously unknown or unseen during the model training process. One of the challenges of supervised ML is the labeling of a large number of data samples which is often done manually and it is time-consuming. But, if we have enough labeled data samples this type of ML produces good prediction models. A supervised learning model has two major tasks to be performed, classification and regression.

Classification: is concerned with predicting output labels or responses that are categorical in nature for input data for what the model has learned in the training phase [39]. Output labels, also known as classes or class labels, are categorical which means they are unordered and discrete values. A classification problem can be binary classification (with two classes) or multiclass classification when there are three or more classes [41]. Common algorithms for performing classification include LR, NNs, support vector machine, ensembles like RFs and gradient boosting, k-nearest neighbors, DTs, Naïve Bayes and so on.

Regression: in contrast to classification, regression is concerned with predicting the numeric value for the class label. Regression models use input data features and continuous numeric output values to learn specific relationships between the inputs and corresponding outputs [41]. With this knowledge, the regression model can predict output responses for new unseen data instances similar to classification but with continuous numeric outputs. Linear regression and multivariate regression are some of the supervised regression algorithms.

2.5.1.2 Unsupervised Learning

Unsupervised learning deals with how systems can learn to represent particular input vectors in such a way that it reflects the statistical structure or pattern of a collection of inputs [41]. In contrast to supervised learning, there are no explicit target output labels associated with each input. Here the task is to group unsorted information according to patterns, similarities, and differences without any prior training data [42]. Clustering is one of the unsupervised learning methods which helps to cluster or group data points into different groups or categories, without the availability of any output label in the input/training data.

2.5.1.3 Semi-supervised Learning

A semi-supervised learning method is another type of ML which combines supervised and unsupervised learning and is used when a smaller number of labeled data is identified for a particular application [43]. It generates a function mapping from inputs of both labeled data and unlabeled data. The goal of this learning mechanism is to classify some of the unlabeled data using the labeled set of information. The quantity of unlabeled data should be higher than the number of labeled data.

2.5.2 Some Popular Classification Algorithms

This section presents some of the common ML classification algorithms with their strengths and weaknesses.

- **Logistic Regression (LR):** The logistic regression algorithm is considered as a linear model that measures the relationship between the target variable and other variables by estimating the probability using a logistic function as shown in Eq. (1) [44]. It was first developed by statisticians to describe the properties of population growth in ecology [45]. In LR, input values (χ) are combined linearly using

weights or coefficient values to predict an output value (y). The output of LR is a value between 0 and 1. Results closer to 1 indicate that the input variable more clearly fits within the category while results closer to 0 indicate that the input variable likely does not fit within the category. The ease of mathematical interpretation of the model makes this method suitable.

$$y = \frac{e^{B0+B1\times\chi}}{1+e^{B0+B1\times\chi}} \tag{1}$$

where, Y is the predicted output, B0 is the bias or intercept term, and B1 is the coefficient for the single input value (χ).

Making predictions with a LR model is as simple as plugging in numbers into the LR equation and calculating the result. However, one challenge with this algorithm is that we cannot understand the predictions as a linear combination of the inputs.

- **Decision Trees (DT):** A DT is a supervised ML algorithm, that can be used to develop a learning model to predict a class or value of a target variable by learning the decision rule of training data [46]. This algorithm can be used to solve classification and regression problems. DTs are arranged in a hierarchal tree which is created by partitioning the training data into various groups to find homogenous subgroups [47]. Knowing which attribute to be placed at the root and at each level is the main challenge in constructing a DT. Entropy and information gain are statistical measures that are used to construct a tree in the DT. Entropy deals with calculating the homogeneity of information. The total entropy of a given dataset is obtained by first splitting the dataset based on different attributes and calculate the entropy for each branch using Eq. (2) [48] and adding each of the entropy for each branch proportionally.

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{2}$$

where E(S) is entropy of dataset, and P_i is Probability of randomly picking an element of class i.

Information gain (IG) deals with calculating the expected reduction in entropy. IG for a split is calculated by subtracting the weighted entropies of each branch from

the original entropy (entropy before split) by using Eq. (3). Attribute with maximum IG is considered to split.

$$\text{Info}(A) = E(S) - E(X) \tag{3}$$

where Info(A) is information gain of a specific attribute 'A', E(S) is entropy of the dataset given in Eq. (2), and E(X) is entropy of a specific attribute in the given dataset.

After the IG for each attribute is calculated, attributes are ranked in their order and the attribute with the largest IG is placed as the decision node in the DT. Although IG is a good measure for deciding attribute relevance, it has biasness problem when it is applied to attributes with a large number of values. One solution to this problem is using the gain ratio method. Gain ratio (GR) reduces the bias on multivalued attributes by considering the intrinsic information of a split when choosing an attribute using Eq. (4) [49]. The GR takes the number and size of branches into account when choosing an attribute. Intrinsic information is the entropy of the distribution of instances into branches (i.e., how much information do we need to tell which branch an instance belongs to).

$$\text{Gain ratio}(\text{Attribute}) = \frac{\text{Gain}(\text{Attribute})}{\text{Intrinsic\_info}(\text{Attribute})} \tag{4}$$

In general, DT algorithms are suitable for multiclass classification. But the problem with DT algorithms is overfitting [50].

- **Naïve Bayes:** Naïve Bayes is a simple and powerful algorithm for predictive modeling. It is considered as probabilistic classifier; hence, it is based on Bayes probability theorem [51]. The model comprises two types of probabilities that can be calculated directly from the training data: (i) the probability of each class and (ii) the conditional probability for each class given each x value. Once the probability is calculated, the model can be used to make predictions for new data using Bayes theorem as denoted in Eq. (5) [51].

$$P(C_i \mid O) = \frac{P(O \mid C_i)P(C_i)}{P(O)} \tag{5}$$

where $P(C_i \mid O)$ is the posterior probability of being a class label "C_i" given the attribute value" O", $P(O \mid C_i)$ is the probability of the attribute "O"

when the given class label is "C_i", P (C_i)is the prior probability of the class label "C_i", and P(O) is the prior probability of the attribute value "O".

Naïve Bayes classifier is preferable when the dataset has a small size and non-correlated or independent attributes. A dataset with many attributes would take a long time to calculate the posterior probabilities, that is P (C_i | O).

- **Support Vector Machine (SVM):** SVM works by constructing hyperplanes for linearly separable data in multidimensional space using a kernel function [52]. The hyperplanes separate the sample data into different groups. Objects are rearranged for optimal separation through the use of this kernel function. A kernel function provides a nonlinear mapping from inputs x to feature space φ(x). During mapping from input space to the feature space, the mapped object can be separated by a straight line with the new space. All the instances and target classes are represented as vectors in high-dimensional space and the SVM finds the closet two points from the two classes that support the best separating line or hyperplane as shown in Figure 2.3 [53].

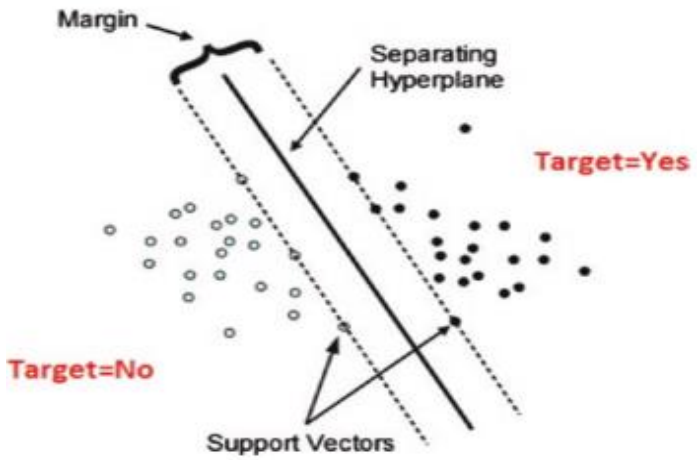

Figure 2.3: Support vector machine for linearly separable data

Although SVM was initially designed to classify two classes of linearly separable data, today it is used to classify data that are composed of more than two classes that cannot be separated linearly. In multi-class problems, datasets are classified using more than one binary SVM. SVM is capable of training very quickly and performs classification and regression tasks.

- **Random Forest:** Random forest is an ensemble classifier that fits a large number of DTs to a dataset and then combines the prediction from all trees. Ensemble methods are methods that combine multiple ML models to create more powerful models [54]. Random forest works on the voting mechanism and predicts the output class that received the maximum votes from all individual DTs [52]. It can be used for classification and regression. RF algorithm starts by randomly selecting a subset of training data and a single DT is generated with the sub-sample. The random selection of examples is repeated many times, and a single DT is generated with each sub-sample which finally forms a forest. Figure 2.4 [55], shows RF on a sample of the IDS dataset where random samples are selected from the original dataset and trees are constructed based on these sub-samples. After the forest is constructed, individual trees make a classification decision or prediction and then the final predicted class of observation is made by majority vote. One of the main advantages of RF is that it reduces the risk of overfitting.

  With RF, it is easy to build and train and often it gives very good results with less chance of overfitting than DTs. This is due to the use of a random selection on subsets of variables and its voting scheme through which decision is made [56]. RF is appropriate for high dimensional data modeling because it can handle missing values and can handle continuous, categorical and binary data.

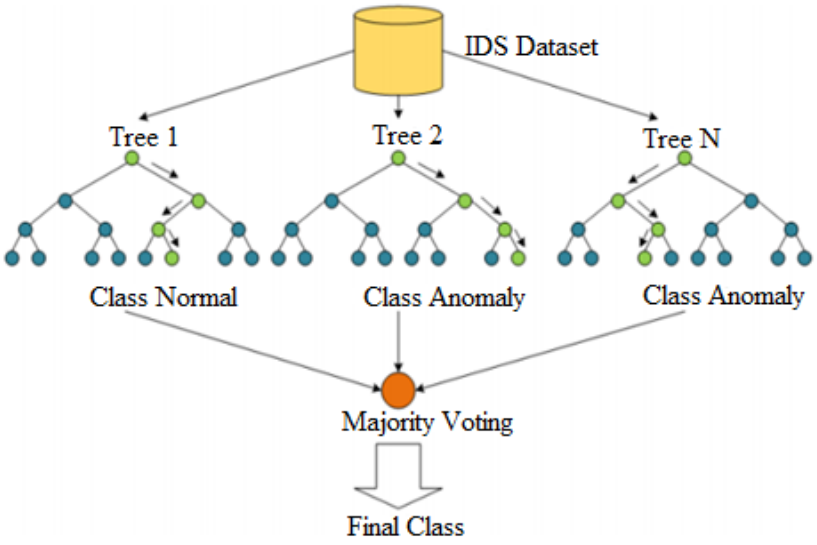

Figure 2.4: Random Forest

The ensemble scheme of RF makes it enough to overcome the problems of overfitting and hence there is no need to prune the trees. Besides, its high prediction accuracy, efficiency, and interpretability make it preferable for various types of datasets [57].

- **Neural Networks:** The first step toward artificial NNs came in 1943 when Warren McCulloch, a neurophysiologist, and a young mathematician, Walter Pitts, wrote a paper on how human neurons might work [12]. An artificial neural network is an information processing system that is inspired by the way biological nervous systems such as the brain work [58]. It is composed of a large number of highly interconnected processing elements (called neurons) working with each other to solve a specific problem. Each neuron can make decisions and feed those decisions to other neurons, organized in interconnected layers. The output of each neuron (after applying a weight parameter) is fed as an input to all of the neurons in the next layer.

  During training, the NN parameters are optimized to associate outputs with corresponding input patterns. The processing elements of a NN are organized in different layers. A typical NN has an input layer, output layer and at least one hidden layer with connection [59].

  Input Layer

  This layer is responsible for receiving information(data), signals, or features from the external environment. Usually, these inputs or patterns are normalized in order to be able to be executed by the available algorithm.

  Hidden Layers

  This layer is the collection of neurons that is responsible for extracting patterns associated with the system being analyzed. This layer performs most of the internal processing from a NN. It transforms the input into a form that can be used by the output layer or another hidden layer. Sensing patterns within received inputs, evaluating, and adjusting the weights are some of the activities performed in this layer. The number of layers needed in this layer is determined by the complexity of the task and accuracy needed by the network. Each neuron in this layer has an activation function for producing non-linear output and helps to produce better pattern detection depending on the provided data. There exist various activation functions in a NN such as hyperbolic tangent (Tanh), rectified linear unit (ReLU), sigmoid and so on. A ReLU is a powerful activation function, as it produces value

between 0 and ∞ [53]. If the input is 0 or less than 0, the output is always going to be 0, but for anything more than 0, the output is similar to the input.

Output Layer

This layer is composed of neurons and is responsible for producing and presenting the final network outputs which result from the processing performed by the neurons in the previous layers. The output layer has several neurons depending on the outputs of the problem and output values between 0 and 1.

A NN can be a single layer (with one input layer and one output layer) or multilayer (with one input layer, one or more hidden layers, and output). Single-layer NNs can only learn linear relationships and cannot perform well in a complex relationship between input and output. Multilayer NN (also known as multilayer perceptron) is composed of one or more hidden layers in addition to input and output layers. Figure 2.5 shows a multilayer NN with one input layer, two hidden layers, and one output layer. The number of neurons composing the first hidden layer is usually different from the number of neurons composing the input layer of the network. In fact, the number of hidden layers and their respective number of neurons depend on the nature and complexity of the problem being mapped by the network.

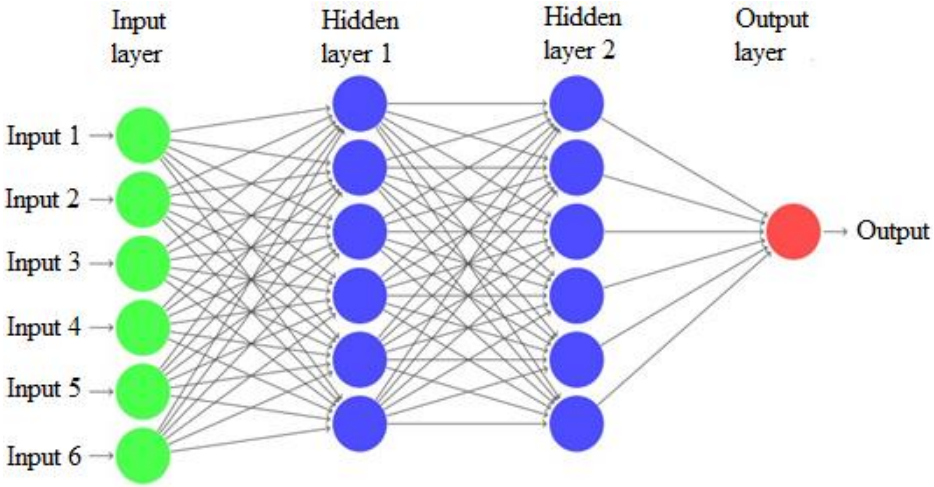

Figure 2.5: Multilayer neural network

A NN training is done by adjusting the weights. The overall training process of a NN can be classified into the following steps [53].

1. The input values are multiplied by the weights and their sum is calculated and the results are passed to the hidden layer including the bias neuron.

2. The hidden layer neurons go through the same process as step 1, using the inputs from the previous layer (input layer) and produce output.
3. The predicted output is then compared with the actual output and the error is backpropagated to the network, to adjust the weight of the connections in order to reduce the overall error of the training data and this is called backpropagation. A new output value is calculated based on updated weights by the network.

In summary, NNs are good models that can deal with large datasets and can be capable of working with noisy datasets (missing variables). However, their main challenge is that weights can be difficult to understand and the training takes a longer time than the other ML techniques.

2.6 Overfitting and Underfitting

In ML, some models are trained very well and can make correct predictions for any sample in the training data but unfortunately can't make correct predictions for other samples outside the training samples. This problem is called overfitting [50]. Overfitting happens when a model learns the details and noise in the training data which can negatively impact the performance of the model on new unseen data. Usually, nonlinear ML algorithms such as DTs, k-nearest neighbors, and SVMs have the flexibility to overfitting when learning a target function.

Overfitting can be prevented using different strategies such as cross-validation, feature selection, reducing network size in NN, dropout and early stopping [60, 61]. Cross-validation is a method to evaluate the generalization performance of ML models in a more stable and detailed way [62]. In this method, instead of splitting the dataset into a training set and test set, the data is split repeatedly and multiple models are trained. K-fold cross-validation is the most commonly used method of cross-validation, where k is a user-specified number, usually five or ten. For instance, when performing five-fold cross-validation, the data is first partitioned into five parts of approximately equal size, called folds. Next, a sequence of models is trained. The first model is trained using the first fold as the test set, and the remaining folds 2-5 as the training set. The model is built using the data in the folds 2-5, and then the accuracy is evaluated on fold 1. The second model uses fold 2 as a test set and the data in folds 1, 3, 4, and 5 as a training set. This process is repeated using the folds 3, 4, and 5 as test sets. For each of these five splits, we compute

the accuracy and finally, the highest mean accuracy is chosen. Overall, this method helps to get the best performing model to predict well on unseen data. The downside of this method is the time it takes to train all the models.

Underfitting is another issue that can occur with ML algorithms. Underfitting is the inability of the model to predict well the labels of the data it was trained on [60]. This problem usually happens when we have fewer data to build an accurate model and try to train a linear model using nonlinear data and then the model produces a lot of wrong predictions. Such problems can be minimized by training models using more data and removing irrelevant features using feature selection mechanisms. Both underfit and overfit make the model too weak and unable to be generalized to any sample. So, model tuning helps models to generalize well on predicting any sample, whether training or testing.

2.7 Feature Selection and Reduction in Machine Learning

Usually, the data used for evaluating ML models can contain redundant features, irrelevant features, weakly relevant and strongly relevant features. Some features may be optimally correlated with each other and thus only one feature is sufficient in describing the data and no need to use multiple features that work the same. In another way, some features have no correlation with the predictions required and regarded as noise. Such features don't contribute to the enhancement of the model's performance rather than negatively affect the performance of the model. So, it is better to remove these features. In addition, the longer the length of features, the more computation time is needed for training the models. Feature selection is one of the main tasks in ML which is used for dimensionality reduction by removing irrelevant and redundant features from the dataset to obtain an optimal feature subset [63]. This helps to speed up ML algorithms, improving classification accuracy, and enhancing a model's generalizability. There are several papers published on feature selection [64, 65, 66].

To classify features as relevant or irrelevant, some metrics are required to show the relevance of each feature with the output classes or how well each feature predicts the desired outputs. Here feature relevance means the ability of the feature to discriminate against the different classes. Feature selection methods are classified into supervised and unsupervised [67]. Supervised methods include filter and wrapper, and unsupervised methods include embedded.

- **Filter:** The filter approach applies feature ranking techniques for ranking features based on metrics like correlation, mutual information (MI), entropy, and so on [43]. These methods do not depend on results obtained from any model and usually check the relationship of each feature with the response variable to be predicted. Based on a threshold, the highly ranked features are selected to train and test the model. The filter methods are very fast and not time-consuming compared to other selection approaches. They are also simple in their calculations, scalable, able to avoid overfitting, and independent of the learning model. But the main drawback of this method is that it doesn't consider features dependencies. It selects each feature independently from the other features or variables in the same subset. This can affect the entire selected subset because features that are significant by themselves in increasing the learning performance are not guaranteed to be so when combined with other features. It includes popular methods such as correlation feature selection (CFS), IG, GR, attribute ratio, and principal component analysis (PCA). Nowadays, as big data with high dimensionality are evolving, the filter approach has attracted more attention than before [67].
    - **Attribute Ratio Feature Selection**

      This method chooses optimal features by calculating the attribute ratio of each feature. Attribute average and frequency for each class is used to calculate attribute ratio (AR) from numeric and binary type. AR can be calculated as:

$$AR(i) = MAX(CR(j)) \tag{6}$$

      Class ratio (CR) is attribute ratio of each class for attribute i. CR is calculated by two methods according to the type of attributes. For numeric type attributes CR can be calculated using Eq. (7).

$$CR(j) = \frac{AVG(C(j))}{AVG(total)} \tag{7}$$

      For binary type attributes CR can be calculated using Eq. (8).

$$CR(j) = \frac{Frequency(1)}{Frequency(0)} \tag{8}$$

      Then, features rank-ordering larger AR and features with a higher attribute ratio are considered as an optimal set of features and used for training and evaluating the model.

- **Wrapper:** Unlike the filter approach which relies on analyzing the general characteristics of data and evaluating features without involving any learning algorithms, wrapper methods select features from a dataset by actually training the learning algorithm itself on the data [68]. The wrapper approach interacts with the learning model as it uses the learning algorithm performance as a metric to select the best feature subsets. It also models feature dependencies, as features are not selected individually or independently from each other, and monitors how combining features together affects performance. Due to this, redundant and correlated features can be reduced in the selected subset. But this method is time-consuming as each model has to be trained multiple times and it is not important for models that take a long time to be trained.
- **Embedded:** The embedded approach combines both filter and wrapper feature selection approaches to get advantage of both approaches [67]. Such methods reduce the long consuming time by avoiding retraining the learning algorithms and maximize the performance by interacting with the learning model. The feature is selected only if it is correlated with the output class labels. This approach includes methods such as pruning and regularization. Pruning methods start by training the learning model with the complete set of features and then calculate a correlation coefficient for each feature. Such coefficients are used to rank the importance of the features according to the model used. High values for coefficients reflect a strong correlation. Regularization methods are also called penalization methods that introduce additional constraints into the optimization of a predictive algorithm (such as a regression algorithm) that bias the model toward lower complexity (fewer coefficients). Such methods include: Lasso Regressio*n (*L1) and Ridge Regression (L2). L1 adds absolute value of magnitude of coefficient as penalty term to the loss function and shrinks the less important feature's coefficient to zero thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features. While L2 regularization adds squared magnitude of coefficient as penalty term to the loss function [69].

2.8 Classification Algorithms Evaluation Metrics

The most common metrics used in most of machine learning classification applications are the accuracy, confusion matrix, precision, recall, and fl-score [70]. All the classification metrics that we mentioned here are based on actual labels and predicted labels of samples.

The actual labels of samples are already available in the dataset. However, predicted labels are obtained using the model. That means in order to evaluate a model using one of the classification metrics, it is necessary to predict labels of samples.

2.8.1 Confusion Matrix

A confusion matrix is used to evaluate the performance of the models. A confusion matrix is a table that summarizes how successful the classification model is at predicting examples of various classes [40]. One axis of the confusion matrix represents the label that the model predicted and the other axis is the actual label. Each cell in the confusion matrix represents one of TP, TN, FP, and FN outcomes of the prediction results of a model. Each column in the confusion matrix represents classified instance counts based on predictions from the model, and each row of the matrix represents instance counts based on the actual or true class labels. Tables 4.1 and 4.2 show the confusion matrix for two-class and five classifications respectively. The attack class is taken as a positive and normal class as negative.

Table 2.1: Confusion matrix for binary classification

	Predicted Class		
Actual Class		Normal	Attack
	Normal	TN	FP (False alarm)
	Attack	FN	TP

where,

⇒ True Negative (TN): number of normal instances that are actually normal and the model predicted as normal

⇒ False Positive (FP): number of normal instances that are actually normal but the model incorrectly predicted as an attack

⇒ False Negative (FN): number of attack instances that are actual attack but the model incorrectly predicted as normal

⇒ True Positive (TP): number of attack instances that are actual attack and the model correctly predicted as an attack

The confusion matrix for multiclass classification has as many rows and columns as there are different classes.

Table 2.2: Confusion matrix for the five-class classification

	Predicted Class					
Actual Class		Normal	DoS	R2L	Probe	U2R
	Normal	TN	FP	FP	FP	FP
	DoS	FN	TP	FP	FP	FP
	R2L	FN	FP	TP	FP	FP
	Probe	FN	FP	FP	TP	FP
	U2R	FN	FP	FP	FP	TP

The confusion matrix in Table 4.2, represents the TP, TN, FP, and FN of five-classes: Normal, DoS, R2L, Probe, and U2R. From this, the total number of TPs for each class is the diagonal values. The total number of FNs for a class DoS is the sum of values in the corresponding row excluding the TP. The total number of FP's for class DoS is the sum of values in the corresponding column excluding the TP. The total number of TN's for class DoS is the sum of all columns and rows excluding that class column and rows. The same procedure is applied to the other classes to find the total number of TP, TN, FP, and FN. A classification algorithm is a perfect classifier if all non-diagonal elements of the confusion matrix are zero that means FP = FN = 0.

In general, a confusion matrix is a powerful metric for accurately evaluating classification models. But it might be tedious or even impractical to analyze a confusion matrix on large class classification problems [71].

2.8.2 Accuracy

Accuracy is the percentage of correct predictions made by the model. This metric is commonly used for assessing classification models. It measures how often the classifier makes the correct predictions. It represents how the classifier predicts normal network traffic data as 'Normal' and as 'Attack' for abnormal network traffic data. This metric is useful when there are equal numbers of observations in each class and all predictions are important. Accuracy will take value in the range of [0,1]. If accuracy is equal to 1 that means all samples in the dataset are correctly classified. In contrast, if accuracy is equal to

0 that means none of the samples in the dataset is classified correctly. Accuracy is computed using Eq. (9) [72]:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{9}$$

2.8.3 Precision

Precision measures the number of actual positive cases out of all the positive cases predicted by the model based on the positive class using Eq. (10). This is also called positive predictive value. Out of the items that the classifier predicted to be true, how many are actually true? In our case, how correct enough is the classifier to recognize the normal samples? The formula for calculating precision is as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{10}$$

This metric is more important in cases where we are more concerned with finding the maximum number of positive classes.

2.8.4 Recall

A recall is the percentage of actual positive cases that the model is able to predict correctly out of the total number of positive cases. In our case, how correct is the classifier to classify all available attack instances as 'ATTACK'. The formula for calculating recall is denoted by Eq. (11).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{11}$$

2.8.5 F1- score

F1-score denoted by Eq. (12) [72] is the harmonic mean of the values of precision and recall. F1- score close to 1 means a perfect model. However, f1- score close to 0 shows a low model's predictive capability. This metric is important for unbalanced classes. F1 - score of a model is calculated as:

$$\text{F1 score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{12}$$

FPR also called false alarms or specificity; determines the total number of incorrect predictions among all negative samples in the dataset. FPR is defined by Eq. (13).

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{13}$$

2.9 Hyperparameter Tuning

Hyperparameter tuning is a technique to find hyperparameter values for a ML model that performs the best on a validation dataset for a problem [73]. Most of the time people use the terms parameter and hyperparameter interchangeably, however, they are different. Model parameters are variables that are learned during training whereas model hyperparameters are variables set by the programmer before training and helps to control the implementation of the model. For example, the weights learned during training a NN model are parameters while the maximum number of iterations, regularization parameter (regParam), and elastic net regularization parameter (elasticNetParam) needed to properly estimate the LR model; maximum depth and maximum bins of a DT; number of trees and maximum depth of a RF model; and the number of layers and nodes each layer uses in a NN are model hyperparameters. These variables need to be tuned because the best model hyperparameters for one dataset may not be the best across all datasets. There are two major ways in which we can search our parameter space for an optimal model. These two methods differ in the way we will search for them: grid search and random search.

- **Grid Search:** Grid search is the simplest of the hyperparameter tuning methods. In this method we specify the grid of values (of hyperparameters) that we want to try out and optimize to get the best parameter combinations [63]. Then we build models on each of those values (combination of multiple parameter values), we search the whole parameter grid to select the best parameters using cross-validation, and report the best parameters' combination in the whole grid. The output will be the model using the best combination from the grid. Due to its simplicity and parallelizability, this method is important when the dataset we are dealing with is of a large size. Although this method is quite simple, it suffers from one drawback that the user has to manually supply the actual parameters, which may or may not contain the most optimal parameters.
- **Randomized Search:** Random search differs from grid search in that no need to provide a discrete set of values to explore for each hyperparameters; instead we provide a statistical distribution for each hyperparameter from which values are randomly sampled and set the total number of combinations we want to try [40].

3. RELATED WORK

In this Chapter, previously conducted researches which are related to intrusion detection along with the methods, tools, and techniques used, the results gained and the challenges with these methods are presented. The following sections briefly discuss the works done using anomaly and signature detection approaches, IDSs done based on ML techniques, and the different feature selection methods applied in intrusion detection.

3.1 Anomaly and Signature-Based IDS

Qassim *et.al.* [74] developed an anomaly-based network intrusion classifier to automatically classify activities from the Internet. The authors used a packet header-based anomaly detection system for network traffic collection. From the collected network traffic, they extracted features but the methods used for feature extraction were not stated. After that, the network anomalies classifier was applied to the extracted features. They used random committee, rotation forest, RF, and random tree as ML classifiers for intrusion detection using the WEKA tool. Their report showed that they compared the performance of the algorithms on five datasets and they found random committee and random tree performed better with similar accuracy than the other algorithms. Out of the two high performed algorithms the authors report showed that random committee gained an accuracy of 96.61%, 99.70%, 98.45%, 99.09%, and 98.18% for dataset A, B, C, D, and E respectively. The proposed classification model was able to classify malicious activities having large samples effectively, but it was not effective in detecting a small number of training samples.

Karami [75] developed a system to detect anomalies accurately and to provide visualized information to end-users using ML techniques. The author's main goal was to achieve a higher detection rate and low false alarm. The author used two self-organizing map (SOM) algorithms: Fuzzy SOM and standalone SOM by using three stages. In the first stage, benign outliers were identified. In the second stage, SOM was run and in the third stage, SOM was re-run over those selected data. The author did the experiments on the Windows platform in MATLAB R2016b. The proposed model was evaluated on NSL-KDD, UNSW-NB15, AAGM, and VPN-nonVPN datasets. Finally, the experimental results showed that the proposed method provided better lattice adjustment with a few overlapped connections among neighbors. However, it failed on connections between nodes and their neighbors from the other two methods.

Sofi *et.al*. [76] have developed a system based on various ML techniques to detect and analyze distributed denial of service attacks (DDoS). They designed an anomaly-based attack detection system using four ML techniques (DT, Naïve Bayes, multilayer perceptron, and SVM). They used the following three stages: 1) collecting data from a network that contains features of four attacks namely: Smurf, UDP-Flood, HTTP-Flood, and SIDDoS and then preprocessing these data to remove irrelevant attributes, 2) prepared a dataset, and 3) developed machine learning-based models to detect DDoS attacks. Experiments were performed using a WEKA tool in Ubuntu 16.04 LTS platform. Out of the prepared dataset, 66% were used for training while the remaining 34% for testing purposes. The four ML classifiers were compared in terms of accuracy, precision, and recall. Their experimental results showed that the overall accuracy was 96.89%, 98.89%, 98.91%, and 92.31% for Naïve Bayes, DTs, multilayer perceptron, and SVM respectively. Their report showed that a multilayer perceptron achieved the highest accuracy rate. Even though the authors collected and labeled their own dataset, the dataset is restricted to only two layers of a network (network and application layer).

Kshirsagar and Joshi [77] proposed a rule-based classification model for intrusion detection using data mining frameworks. The main goal of the proposed research work was to test different rule-based classifiers for IDS and to select the best one. First, raw audit data was preprocessed into ASCII new packet information which was then summarized into connection records. Then, rules for the connection records were generated using a data mining algorithm. The authors used KDD CUP 99 dataset for evaluating the proposed model on four types of attacks namely: DoS, probe, U2R, and R2L. They conduct their experiment using a WEKA tool and the authors stated that the proposed model produced good detection accuracy on known attacks. But the work was not able to classify new attacks (attacks in the test dataset but not available in the training dataset).

3.2 IDS Based on Machine Learning

As it is already discussed in the literature review chapter, ML algorithms are well-suited to problems that are complex and difficult for human beings. The task of detecting network intrusions also comes under ML as it involves the classification data into normal and abnormal behavior.

Many researchers intend to use a big data technique to produce high speed and accurate IDS. Gupta and Kulariya [78] proposed a fast framework to detect network attacks. They used five ML techniques (LR, SVM, RF, Naïve Bayes, and gradient boosted tree) using Apache Spark to determine whether network traffic is an attack or normal. The authors used two feature selection methods, namely correlation-based feature selection (CFS) and chi-squared feature selection. The proposed framework used two real-time network traffic datasets namely KDD-CUP and NSL-KDD. Two experiments were carried out. In the first experiment, the CFS method was applied on both datasets to remove the highly correlated attributes and then they trained the classification models. In the second experiment, they applied chi-squared feature selection on both datasets and trained the classification models. Finally, the models were compared in terms of accuracy, sensitivity, specificity, and training and predicting time-based on both feature selection methods. Their first experimental results showed that the LR technique acquired the highest accuracy of 91.56% whereas SVM acquired the lowest accuracy which is 78.84%. In terms of sensitivity, NB scored highest sensitivity score of 98.62% while GB tree scored a lowest which is 89.23%; in terms of specificity GB tree scored higher than the others which is 99% and SVM acquired the lowest specificity, in terms of training and prediction time, NB took the lowest time which is 80 seconds whereas SVM took the highest time of 480 seconds on KDD-CUP dataset. According to their second experiment, the same accuracy was found for LR, SVM, and GB tree whereas the accuracy of RF and GB tree increased to 92.13% and 91.38% respectively. The major shortcoming of this approach is that the chi-squared method decreased the accuracy of the classifiers even though processing time is improved.

Tegegn Kebebaw [79] developed malware and suspicious activity detection system that protects personal computers from any kind of malware and suspicious activity. A NN was used as a ML technique. The experiment was conducted on three different computers. First, a dataset collection was performed and unwanted data was removed from the dataset. Overall performance of 82%, 75%, and 69% were found for the different machines. The study was conducted on small data and as a result low-performance result was gained.

Effendy *et.al*. [80] proposed a system that can detect network intrusions based on the NSL-KDD dataset by using feature selection methods. In this study, the dataset was normalized and discretized by the k-means method and features were selected using an information-gain algorithm and passed to the NB ML algorithm. They found that the k-means

clustering method acquired better results relative to the discretization technique of mean and standard deviation. The data after getting labeled from the k-means method was fed to the IG that uses scoring methods for normal or continuous attributes by using maximum entropy. The developed system gained a detection accuracy of 89.6% on the discretization technique.

Esmaily *et.al*. [81] developed a classification method that basically combines a machine learning-based decision algorithm and multilayer perceptron with the goal of designing an accurate IDS with a high detection rate and low FAR. The authors used artificial NNs to mitigate dataset limitations such as nonlinear and incomplete. They used the KDD-CUP99 dataset as their source dataset. They performed their work in two phases. In the first phase, the dataset was fed to both the DT-based approach and multilayer which classifies them and labels the data into attack or benign. In the second phase, the new dataset was again fed to the well-trained multilayer perceptron to evaluate the test data. The drawback of this approach is that the authors did the experiment only in small network data.

Naseer *et.al*. [82] presented a deep learning model to perform anomaly detection on network traffic data. One of the basic advantages of deep learning is its ability to learn raw input features without a need to preprocess them. The proposed approach aimed to train a deep neural network (DNN) model with raw input data. They built a model called Raw Power multi-layer DNN for network analysis. First, they built a 1D-CNN based model to detect malware at the packet level. Then, they took every packet from the labeled subset and a fixed threshold (T_s= 1300) value was assigned to each incoming packet. Once the threshold value was assigned to each packet, all packets with size larger than 1300 were reduced whereas packets with smaller sizes were added. Finally, each packet was labeled as benign or malware. Two activation functions (ReLU and max pooling) and two feed-forward layers of two hundred neurons each were applied. A binary LR algorithm was also used for classification. The model was built on Keras along with the Tensor Flow framework on the USTCTFC2016 dataset. Furthermore, the proposed model was compared with the RF algorithm by using the same input features and their result showed that RF gained better results with 70% detection and below 3% FAR. But the model performed poor on non-processed input data.

Amin and Bin [83] compared several ML techniques utilized to develop IDS. The authors did a detail discussion of the survey for intrusion classification algorithms and compared

them. They selected commonly used ensemble algorithms such as boosting, bagging, stacking, and a mixture of competing experts. The ensemble method works by training many classifiers at the same time to solve the same problem and then combining their output to improve accuracy. The ensemble classification system provides better generalization capacity than a single classifier. Based on the conducted survey, they have found that the ensemble classifier based on the majority voting strategy was an effective approach to the development of the IDS. KDD-CUP 99 dataset was used for evaluating the performance of the proposed method.

Ingre *et.al*. [84] presented tree-based intrusion detection using a CART DT to detect new or unknown attacks. A CFS method was applied to detect four attacks (DoS, probe, U2R, and R2L) with minimum resource usage. The system consisted of three steps that are preprocessing the dataset, applying feature selection method and classification of network traffic into normal or attack. In the first step, the authors selected the dataset from the NSL-KDD repository and preprocessed the data to convert the categorical attributes to numeric. In the second step, CFS subset evaluation was applied to the preprocessed dataset in order to select the best subset features. Ranked list of attributes was generated using GR feature selection method and then the CFS computes the best subset of attributes by considering the individual predictive ability of each feature and the 41 attributes of the dataset were reduced to 14 best features. Two datasets KDD-CUP 99 and NSL-KDD were used to evaluate the performance of the proposed system using the WEKA tool. Two separate DTs models were created for the dataset, one for five-class classification (normal and four attack types) and another for binary class classification (normal and attack). The overall performance of the proposed method was evaluated in terms of classification accuracy, the detection rate of each class and the false positive rate. Their experimental results showed that the overall accuracy of 83.7% was found for five-class classification and 90.2% accuracy for binary class classification. Their report mentioned that the proposed classification model was exposed to overfitting.

Tewodros Getaneh [85] investigated the practices and challenges of cyber-security to adapt and modify a cyber-security framework for selected critical infrastructures in Ethiopia. The main aim of the research was to modify the existing framework for Ethiopian Electric Power, Ethiopian Electric Utility, and Ethio Telecom infrastructures to be more efficient in identifying cyber-security threats and vulnerabilities. The author examined the existing practices and challenges of cyber-security at the mentioned three critical

infrastructures using both qualitative and quantitative research approaches. The researcher recommended the implementation of a framework should be done for the technical processes of cyber-security at the critical sections.

3.3 Feature Selection Methods in Network Traffic Classification

Feature selection is an important preprocessing stage on datasets to be used in machine learning. It reduces the dimensionality of data by removing irrelevant features and enhances the performance of the classification process. This sub-section presents some works which used feature selection methods in the IDS.

Chabathula *et.al*. [86] used the KDD-CUP99 dataset to see the effects of using principal component analysis (PCA) feature selection with ML techniques in the detection of attacks. The authors initially feed the data to PCA to reduce the higher dimension dataset to lower the dimension of datasets. Then, the new produced dataset was fed into various machine learning-based algorithms which are SVM, k-nearest neighbor, DT, RF, AdaBoost algorithm, and Naïve Bayes classifiers. They performed two experiments without PCA and with PCA. Experimental results were analyzed and compared among the classifier algorithms using detection rate and time metrics. Based on the reported result tree based algorithms (J48 and random forest) outperformed the other algorithms both in terms of accuracy and speed. They used the WEKA interface as their ML tool but increased the overhead.

Martha Teffera [87] proposed a supervised based feature selection system based on data mining algorithms. The author has used wrapper-based feature selection methods to identify relevant features from the dataset. NSL-KDD dataset was used as a source data. they reported their model is accurate in detecting network intrusions. In this work, genetic algorithm, Bayesnet, NB, k-means and DT data mining algorithms were used in both feature selection and prediction processes. First, optimal features were selected after these methods applied to the dataset. Then, the algorithms were applied to selected features to predict for an attack. Finally, the performance of the algorithms was compared on the selected features as well as with other works done using filter selection methods. The researcher used a confusion matrix performance evaluation method to show the accuracy of the classification algorithms. According to their experimental results, DT and k-means algorithms were inefficient whereas Bayesnet and NB became efficient. The Wrapper-

BayesNet method was obtained a classification accuracy of 95.3 % and FAR of 0.006. But, the challenge with this approach was its time complexity.

Kushwaha *et.al*. [65] proposed some feature selection algorithms and developed IDS using SVM. They carried feature investigation using various feature selection algorithms. In this approach, a filter method was used for feature selection and K-nearest neighbor algorithm for dimensionality reduction. They also compared the results obtained by these algorithms and ranked them according to the score obtained corresponding to each attribute. The authors used KDD-CUP 99 dataset for training and evaluation purposes. Experiments were performed on a windows platform and their results showed that mutual information (MI) feature selection was more effective in the selection of relevant features from the 41 features in the KDD-CUP99 dataset. They have used various classification algorithms and compared their accuracy and finally, they found that the SVM achieved an accuracy of 99.91% for multiclass classification.

Chae and Hyun Choi [88] proposed an attribute ratio (AR) feature selection method to select highly ranked features. A DT classifier was used for intrusion detection. NSL-KDD dataset was used for training and evaluating the classifier. The authors, first applied three feature selection methods which are CFS, IG (IG), GR and then attribute ratio feature selection method were applied on the 41 features using DT classifier. A 10-fold cross-validation technique was used to select the best set of parameters that help to improve the performance of the system. They performed two experiments: the first experiment done on the selected features and the second on full data without feature selection. The experimental results found using attribute ratio were compared among the results using the three feature selection methods and the accuracy of AR was the highest (which is 99.79%). They used the WEKA interface as their ML tool which helps them lead to increase overhead.

3.4 Summary

Based on the research works reviewed we have summarized all IDSs to identify their gaps and to suggest a solution that enhance them. A lot of research works have been done which can be applied to intrusion detection. But the related works we reviewed showed the performance, detection rate, training time, and FAR in IDS remains one of the major issues. In addition, most of the works have been done using signature-based intrusion detection approach that can only detect known attacks. Based on this observation, we

select an anomaly-based network IDS using big data processing framework (Apache Spark) to enhance the performance of intrusion detection, to minimize training time, and FAR. As, network traffic data is becoming increased, this big data processing tool helps for faster processing of network data to identify abnormal activities with in short time. Also, some parameter optimization techniques and feature selection methods are selected for developing efficient intrusion detection models.

4. THE PROPOSED NETWORK IDS

This Chapter is all about the proposed system. First of all, the proposed system architecture is given then the data collection and preparation are discussed. The data preprocessing, train-test split, validation split, algorithm training with default parameters, and grid search phases are presented next. Finally, model performance evaluation metrics for both binary and five classes are described.

4.1 System Architecture

The architecture of the proposed network IDS has different components. As shown in Figure 4.1, the system architecture has preprocessing, train-test split, validation split, training with default parameter, and grid search phases. The preprocessing phase which consists of sub-tasks: string indexing to convert string data to numeric, one hot encoding to convert string indices into 0s and 1s, feature selection to select relevant features, normalization for standardizing range of values, and vector assembling for merging features into a single feature vector. In the train-test split phase the whole preprocessed data is split into training set (for training machine learning algorithms) and testing set (for evaluating models' performance). In the validation split phase, the training set is further split into actual training and validation set. In training with default parameters phase, the machine learning algorithm is trained using the training data on their default parameters and produced a trained model. The trained model is then evaluated its performance on the separated preprocessed testing data and evaluation result 1 is generated. The grid search phase is used to find the best parameters for a model by constructing a parameter grid. In this phase first the training and validation data is passed to the grid search. Then, the parameter grid of hyperparameters and classification algorithm are specified and passed to the cross-validation. The cross-validation searches the whole parameter grid and running the training set for selecting the best parameters. Then, the models are evaluated on the validation set. The selected best parameters are then used for building final trained model which is then evaluated on the testing set. The result of this evaluation is evaluation result 2. A detailed description of each component of the system architecture is presented in the following sections.

4.1.1 Data Collection and Preparation

To develop an intrusion classification model, a huge amount of network traffic data is needed to provide enough data to the classifiers to train them properly. For this reason, a

publicly available network intrusion detection dataset is collected from the Web [89]. Even though there are many publicly available datasets for simulating network intrusions, all of them may not be helpful for developing effective IDSs. For example, the KDD CUP 99 dataset which is available at [90] is the most widely used and freely available network intrusion detection evaluation dataset. However, the major limitation of this dataset is the presence of redundant and irrelevant records in the training and testing set and using it will reduce the quality of system performance.

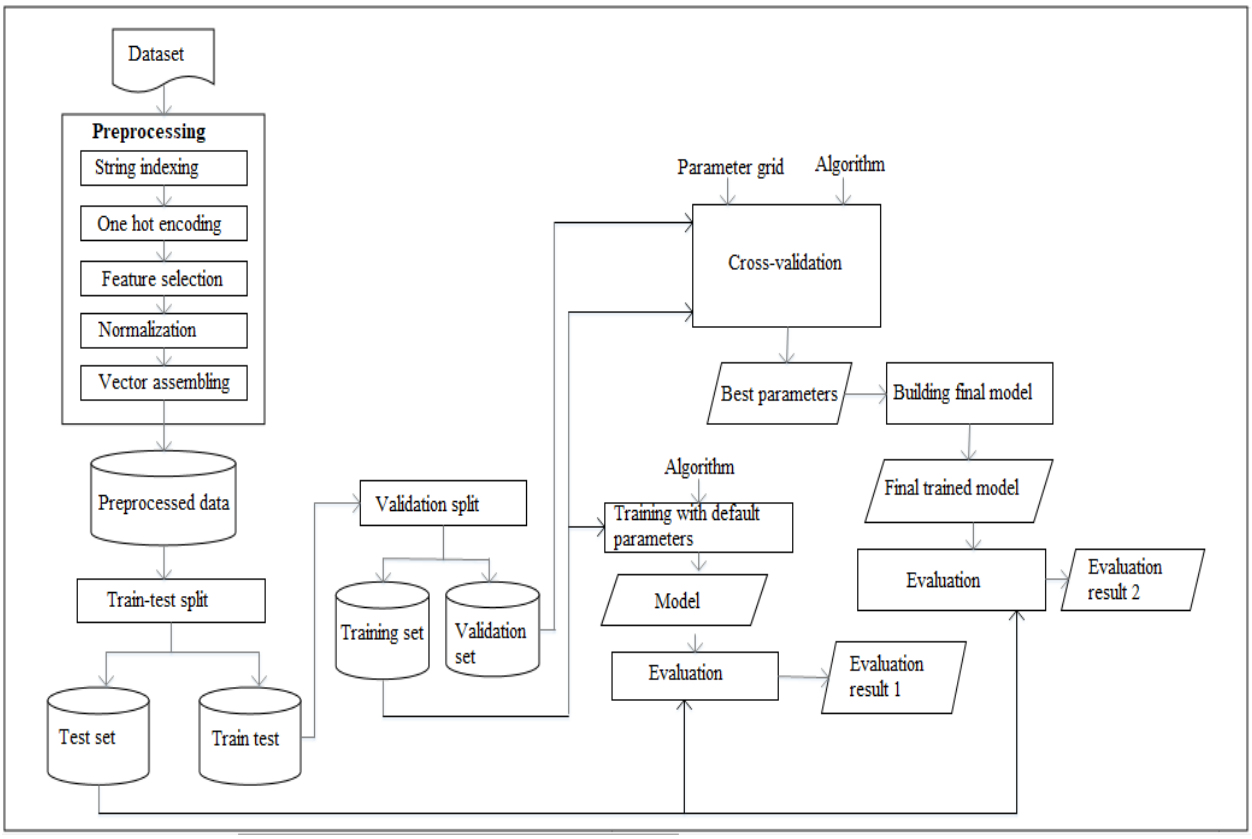

Figure 4.1: System Architecture

Furthermore, it can cause issues such as long training time, poor accuracy rate and low detection rate of a model. Even though this dataset is still used for the evaluation of network IDSs by many researchers, the evaluation results acquired by this dataset are less accurate. To overcome this, we used the latest dataset called NSL-KDD dataset [89]. This dataset is prepared by the University of New Brunswick which is the revised version of the KDD CUP 99 dataset introduced for effective and robust evaluation of machine learning-based IDSs. This dataset contains information corresponding to common types of attacks (denial of service attack, probe attack, a user to root attack, and remote to local attack) [33]. The main reason for selecting the NSL-KDD dataset is that it is the refined

version of the KDD CUP 99 which includes less redundant records in the training set as well as a testing set. As a result, the classifier will not be biased on frequent records [91]. The NSL-KDD dataset contains 42 features per record, with 41 of the features referring to the traffic input and the last feature (column) specifies the target class Annex C [92]. The target class has five categories that are one normal and four attack categories (DoS, Probe, R2L, and U2R). In addition, the dataset contains a total of 24 training attack types, with an addition of 14 new attack types in the test data. Among the 41 attributes, 32 are numeric, 6 are binary, and 3 are nominal data. Features 2, 3 and 4 are nominal, features 7, 12, 14, 15, 21, and 22 are binary and the rest features are numeric types see Annex C.

4.1.2 Data Preprocessing

The main reason why we need data preprocessing arises from the fact that redundant data and insignificant features may be unclear to the classification algorithm and this may lead to inaccurate and less generalizable models. Usually, the data collected from different sources may not be in a format that is suitable for a machine, may contain invalid and/or missed values and may be too large for a specific task. Machines need specific representations of the input data for both training and testing which is termed features before feed to a ML algorithm. In order to prepare the data in that format preprocessing tasks such as string indexing, one-hot encoding, feature selection, normalization, and vector assembling are performed on the training and testing data. Please find the attached sample of codes for string indexing, one-hot encoding, and vector assembling in Annex A. The detailed discussion of these concepts is presented in the next sections.

4.1.2.1 String Indexing

As already mentioned in Section 4.1.1, the dataset for constructing the proposed intrusion detection model contains data of three feature types: nominal, numeric, and binary features. So, before we can run a model on these, we need to change the data into the desired form that is the string or categorical data must first be changed to numeric. To convert this kind of categorical text data into model-understandable numerical data, we use the StringIndexer transformer. StringIndexer encodes categorical values of a column into indices, based on its frequency it appears in the entire input column of a data frame, such that the most frequent categorical value gets an index value of 0, 1, 2 and so on. This transformer has first to count the frequency of each of the input columns. If the input column is numeric, it will cast to string before counting its frequency. First, StringIndexer

class is imported from spark ML library (MLlib), fit and transform the given columns of data, and then replace the existing text data with the new encoded data. Here, the feature is the whole row specified to a number of columns or fields. whereas the label specifies the class of a sample either normal or attack.

4.1.2.2 One Hot Encoding

Since the output of string indexing is numeric, the model can be confused thinking that a column has data with some kind of order or hierarchy, but these numbers do not have any order. To avoid this, OneHotEncode of that column is necessary. To do this, a OneHotEncoder transformer which encodes the index of a categorical value as a binary vector is used. OneHotEncoder takes a column that has categorical data, which has been label encoded and then splits the column into multiple columns. The indices are replaced by 1s and 0s, depending on which column has what value. This encoding allows algorithms that can only operate with numerical valued features, to be applied to categorical features.

4.1.2.3 Feature Selection Using Attribute Ratio

Making use of more features to run ML algorithms tends to make models more complex and difficult to interpret. Besides this, it can often lead to the model's overfitting on the training data. This basically led models to perform very poorly on new, previously unseen data. In addition to this, having irrelevant features in a dataset can decrease the accuracy of models and takes much amount of time to train models. Therefore, we have to select an optimal number of features to train and build models that can generalize very well on the data, with less time of computation. This can be done by using feature selection techniques.

Based on the various literature reviewed, most of the feature selection methods use a complex calculation to find optimal feature subsets so that these methods cannot be efficient for large data. For this reason, in this research, AR feature selection method is chosen for its ease of calculation as compared to the reviewed feature selection methods. This feature selection method selects relevant features combination by calculating the attribute ratio of each features. After, attribute ratio of each features is calculated, features are rank ordering their higher attribute ratio value. Then features with higher attribute ratio are selected as relevant features.

4.1.2.4 Normalization

Many ML algorithms perform better or faster when features are on relatively similar scale or close to normally distributed. A dataset may contain many features that have values in different ranges. Some features may consist small floating-point values and others large integer values. Besides, some ML algorithms such as NNs cannot take large values (values which are larger than the initial values taken by the weights of a network). Therefore, to make the network learn easily, the data fed to the algorithm should take small values (in the range 0-1), all features should take values in the same range, and should be normalized each feature independently to have a mean of 0 and standard deviation of 1. In this research, a StandardScaler technique is used to standardize the features. StandardScaler standardizes a feature by removing the mean and dividing each value by the standard deviation. The reason why we select this normalization technique is because it normalizes most of the data to a small interval than the other standardizers such as MinMaxScaler, RobustScaler, and Normalizer.

Moreover, the input data fed to a machine learning algorithm may include some missing values. These missing values can confuse the classifier during learning as well as the testing process. Most ML algorithms cannot handle missing input values. So, it is better to replace all the missing values with some appropriate values. For instance, with neural networks algorithms, it's safe to input missing values as 0, and the network will learn from exposure to the data that the value 0 means missing data and will start ignoring the value.

4.1.2.5 Vector Assembling

After the categorical data is converted into a numeric and binary vector using StringIndexer and OneHotEncoder respectively the results can be of different types such as a numeric, Boolean, sparse vector or dense vector. However, many ML models are not compatible with sparse or dense vectors. In order to work with these data forms, all input features need to be merged into a single vector feature (a sequential form of data). For this purpose, we have used a VectorAssembler transformer which is available in spark's MLlib. This transformer combines individual features into single-vector features for the ML algorithms to learn.

4.1.3 Train-test Split

The main purpose of any model building process is to develop a generalizable model on the available data which will perform well on unseen data. But to estimate a model's

performance on unseen data, we need to have a separate unseen data. This is achieved by splitting our available data into training and testing sets. The training set is used for a classifier to learn what each category looks like by making predictions on the input data and then correct itself when predictions are wrong. The testing set is used to find the accuracy (predictive capability) of the model. It is only used to compare the results produced by the final model with the actual labels of the dataset It is important that the training set and testing set are independent of each other and do not overlap. This is because if we use the testing set as part of our training data, then the classifier's generalizability will be low since it has already seen the testing examples before and learned from them. We have to keep this testing set separate from the training process and use it only to evaluate the model. The size of the split can depend on the size and specifics of the dataset. Out of the preprocessed dataset, 85% is used for training and the remaining 15% is used for validation.

4.1.4 Validation Split

Although building models by splitting data into training and testing set works, when working on tuning models we need to consider some other datasets. This dataset helps to validate the model and is called validation or development set [63]. The validation set is a sample of data from the available dataset which is not seen in the training data and gives us an estimate of the model's performance in terms of tuning the hyperparameters. The dataset is already split into training and testing sets. Then we split the training set further into an actual training set and a validation set. Out of the training set, 68% is used for training and 17% for validation.

4.1.5 Training with Default Parameters

Given the training set of data, several tasks remain before starting the process to train a ML model. Particularly, it is necessary to define the architecture of the model such as the number of layers, fix some hyperparameters such as block-size, number of epochs or maximum iterations of a NN and decide whether the training starts using default parameters or tuning hyperparameters is needed to find optimal set of hyperparameter values.

The Apache Spark framework provides several predefined models for training a classification model. Four classification algorithms: DT, LR, RF, and NN are selected for training and testing of the proposed model. The main reason for the selection of these

algorithms is the ease of understanding the results of the model, the ability for both binary and multiclass classification, and applicability on large datasets. These algorithms are first trained using their default parameters. Then we evaluated their performance on the test set and results (evaluation result 1) are compared based on evaluation metrics. But, it is necessary to modify the default parameters since the model parameters must be revised depending on the number of classes in the given dataset.

4.1.6 Grid search

This phase provides an algorithm to validate the effectiveness of the model using another dataset, which was not used in the training phase. In this case, the dataset is called the validation set. This phase helps to show that the parameters derived in the training phase work based on evaluation metrics. If the results are not satisfactory, then the model must be trained again to obtain better hyperparameter values which can produce accurate models. The classification algorithms can have many parameters to train or learn from data but it is better to find the best subset of parameters for a model. In this phase both training and validation set are passed to the grid search. Then the parameter grid (hyperparameters with default values) and algorithm are specified and passed to the cross-validation. The cross-validation run on the training data and searches the whole parameter grid for finding best parameters. The developed models are validated their effectiveness on the validation data. The selected best parameters are used to build the final model. The final model is then evaluated on the testing set.

4.1.6.1 Cross-validation

In this research, a grid search with 5-fold cross-validation is used to evaluate and select the best subset of hyperparameters for the selected models. In five-fold cross-validation, the given training dataset is partitioned into five equal parts (folds). Then, only one-fold is used as a test dataset at a time while the remaining four folds (parts) are used as a training dataset to train the models (LR, DT, NN, and RF) with different hyperparameter values. This process is repeated five times until each fold serves as both testing and training datasets. Once the models are developed, we assess their performance on the validation set and select the model with the best performance as the final model. This model contains best subset of parameters which provide the most accurate models (see Annex B). Finally, the final model is evaluated on the test set to evaluate its effectiveness on unseen data.

Table 4.1 shows the default values of parameters and the values after hyperparameter tuning is performed.

Table 4.1: Default and best parameters of the classification algorithms

Algorithm	Parameters	Default values	Best parameters
DT	maxDepth	30	2
	maxBins	40	10
LR	regParam	2.0	0.01
	elasticNetParam	1.0	0.5
	maxIter	5	10
NN	maxIter	800	100
	neurons	20	35 and 30
	epochs	25	500
	block-size	128	128
	learning rate	0.00001	0.0001
	activation function	Sigmoid	ReLU
RF	numTrees	20	50
	maxDepth	10	30
	maxBins	20	40

4.2 Model Performance Evaluation

A model can be evaluated in three phases: during the training phase, validation phase, and testing phase. The performance on the test set tell us how good the model be in the real world. Evaluating a model on a dataset is usually done using classification metrics. Based on the literature review, five evaluation metrics that are: confusion matrix, accuracy,

precision, recall, and f1-score are used for evaluating the performance of the models in this research.

4.2.1 Summary

Intrusion detection is a process of detecting inappropriate, and incorrect or anomalous activity targeted at the computer system and networking devices. We propose a network IDS with integration of anomaly-based approach and machine learning techniques using Apache Spark big data processing framework. The proposed system can achieve effective detection of intrusions using the collected dataset. The dataset is preprocessed using data preprocessing methods and classified into their category based on the trained models. Feature selection and hyperparameter tuning methods are applied to the dataset. The features selected from the dataset enable the system to speed up its processing and improve its accuracy. The grid search phase is responsible to validate the effectiveness of the models using validation data, which was not used in the training phase. It helps to show that the parameters used in the training phase work better based on the evaluation metrics. If the results are not satisfactory the model can be trained again to obtain better hyperparameter values. The selected better hyperparameter values are then used to build the final model. Finally, the final model is used to predict or classify new instances into their category. Based on this evaluation result a decision can be done on the efficiency and effectiveness of the system.

5. EXPERIMENTATION AND RESULTS

In this Chapter, the experiments carried out to test the effectiveness of the proposed system is discussed. The development environment and software tools, the programming languages, and libraries used during the classification process, the dataset used, implementation details, and the performance evaluation results of the different models using various evaluation metrics are discussed.

5.1 Development Environment and Tools

Several tools and techniques are utilized for the implementation of the proposed system. All experiments for implementation of our proposed system were performed on a laptop with the following configuration: Intel(R) Core i5 CPU 2.40GHz, 4 GB RAM and the operating system platform is Ubuntu 14.04 LTS. The following tools are used to implement the proposed system.

Apache Spark

Apache Spark is used to implement the proposed system because its fast-distributed engine for large-scale data processing and ML tasks. Apache Spark is an open-source, cluster computing framework for fast data processing [7]. It was developed at UC Berkeley in 2009. Spark provides in-memory data processing and storage. Compared with another big data processing technology, for example, Hadoop MapReduce, Apache Spark performs operations 100 times faster [93]. One of the important concepts of Spark is that it provides an abstraction on the collection of objects called Resilient Distributed Dataset (RDD) which are partitioned across worker nodes and can be operated upon parallelly. The other significant property of the RDD is, they are fault-tolerant and are stored in the form of read-only immutable RDDs. Although RDDs are immutable, various functions available in Spark like a map, reduce, flatMap, filter, take, collect, and count that transform the RDDs from one state to another [94]. Spark also provides useful built-in libraries, and we have used some of them like Spark SQL, MLlib, and data frame. These APIs are developed on top of spark to provide a system that gives abstraction on the datasets we want to work. Spark SQL enables the use of SQL statements inside Spark applications which is equivalent to a table in RDBMSs and converts an RDD to a data frame. Spark's ML library (Spark MLlib) enables the development of ML applications by combining data processing

and ML algorithms implemented in a cluster of nodes. The spark data frame provides users with a platform to interact with data using Spark SQL.

Apache Spark supports many programming languages like Java, Scala, R, and Python to interact with it. But Python is selected as a programming language for this implementation because it is an open-source with varieties of free ML libraries (like pandas, Scikit-learn, seaborn, matplotlib, etc.), packages, and rich documentation for different ML tasks. In addition to this, Python provides a simple and comprehensive API with ease of code readability and maintenance. To have the advantages of both Python and Spark, PySpark which is a collaboration of Python and Spark, is used for integrating Python and Spark.

Anaconda and Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution which is a free Python distribution developed by Continuum Analytics that is made for scientific computing. It is free even for commercial use. The best thing about it is that it comes with a number of preinstalled packages that are essential for data science, math, and engineering. Anaconda Navigator allows us to launch applications and easily manage conda packages, environments, and channels without the need to use command-line commands. It works across Windows, macOS and Linux platforms [95]. For this research work Anaconda version 1.9 is used.

Spyder 3.7

Scientific Python Development Environment (Spyder) is a powerful scientific environment written in Python programming and designed for scientists, engineers and data analysts to be used for developing Python applications. It offers a unique combination of advanced editing, analysis, and debugging comprehensive development tools with the data exploration, interactive execution, deep inspection and beautiful visualization capabilities of scientific packages [96]. For this research work, the latest version Spyder 3.7 is used. In addition, the following modules are used for the implementation:

- NumPy: a library that supports large multi-dimensional arrays and matrices.
- Matplotlib: plotting library for data visualization (for producing line charts, histograms, scatter plots, and so on)
- Scikit-learn: an open-source ML library in Python, which provides useful helper functions and infrastructure.

- ML: a library that includes algorithms and statistical models.
- Py4j: a library that integrates Python with Apache Spark.

The first three are modules that come with Python and the last two are modules that come with Apache Spark.

5.2 Dataset Used

The dataset used for the experiment, that is the NSL-KDD dataset, has two categories: KDDTrain and KDDTest. The KDDTrain data is used to train classification algorithms whereas the KDDTest consists of a set of new instances that are not present in the training data that helps to evaluate the performance of a model. Table 5.1 shows the number of instances in the NSL-KDD training and testing dataset.

Table 5.1: Number of instances in the NSL-KDD training and testing dataset

Category	KDDTrain Instances	KDDTest Instances
Normal	63726	13328
DoS	44,193	9192
Probe	11692	2385
R2L	3536	213
U2R	237	15
Total	123,384	25,133

As can be seen in Table 5.1, DoS and normal classes have large instances in both training set as well as testing set whereas U2R and R2L classes have small instances on both cases. The different types of attack in both training and testing set are mapped to their respective classes or categories as follows:

- Dos: Back, land, Neptune, pod, smurf, teardrop, mailbomb, apache2, processtable, udpstorm.
- Probe: ipsweep, nmap, portsweep, satan, mscan, saint.
- R2L: ftp-write, guss_password, imap, multihop, phf, spy, warezclient, warezmaster, sendmail, named, snmpgetattack, snmpguss, xlock, xsnoop, worm.

- U2R: Buffer_overflow, loadmodule, perl, rootkit, httptunnel, ps, sqlattack, xterm.

5.3 Implementation Details

This section introduces how the experiment is conducted. The first task done after data acquisition is preprocess the acquired dataset so that the dataset is in a suitable form to be learned by machine. At the first stage, StringIndexer is applied to the training and testing dataset for converting the categorical data into the desired form which is numeric.

As can be seen in Table 5.2, the ‘labels2’ and ‘labels5’ columns are string which can’t be directly used by machines while training. The ‘labels2’ column is a binary class that contains ‘normal’ and ‘attack’ while the ‘labels5’ column is a five-class that contains ‘normal’, ‘DoS’, ‘Probe’, ‘R2L’ and ‘U2R’. After StringIndexer is applied to the two label column which is ‘labels2’, normal is replaced by 0.0 and attack is replaced by 1.0, and a new column ‘labels2_index’, which is the numeric form of the categorical data, is added while for the five-class column, normal, DoS, Probe, R2L, and U2R are replaced by 0.0, 1.0, 2.0, 3.0 and 4.0 respectively and a new column ‘labels5_index’ which is the numeric form for the five class categorical data is added as given in Table 5.2.

Table 5.2: Sample of string indexed data

labels2	labels2_index	labels5	labels5_index
normal	0.0	normal	0.0
normal	0.0	normal	0.0
attack	1.0	DoS	1.0
normal	0.0	normal	0.0
normal	0.0	normal	0.0
attack	1.0	DoS	1.0
attack	1.0	DoS	1.0
attack	1.0	DoS	1.0
attack	1.0	DoS	1.0
attack	1.0	DoS	1.0
attack	1.0	DoS	1.0
attack	1.0	DoS	1.0
normal	0.0	normal	0.0
attack	1.0	R2L	3.0
attack	1.0	DoS	1.0
attack	1.0	DoS	1.0
normal	0.0	normal	0.0
attack	1.0	Probe	2.0
normal	0.0	normal	0.0
normal	0.0	normal	0.0

only showing top 20 rows

After the categorical data are indexed, the indices are then replaced by 0s and 1s using OneHotEncoder to allow algorithms to be applied to these categorical features as shown in Table 5.3.

Table 5.3: Sample of one-hot encoded data

labels2	labels2_index	labels2_index_vec
normal	0.0	(22,[0],[1.0])
normal	0.0	(22,[0],[1.0])
attack	1.0	(22,[1],[1.0])
normal	0.0	(22,[0],[1.0])
normal	0.0	(22,[0],[1.0])
attack	1.0	(22,[1],[1.0])
attack	1.0	(22,[1],[1.0])
attack	1.0	(22,[1],[1.0])
attack	1.0	(22,[1],[1.0])
attack	1.0	(22,[1],[1.0])
attack	1.0	(22,[1],[1.0])
attack	1.0	(22,[1],[1.0])
normal	0.0	(22,[0],[1.0])
attack	9.0	(22,[9],[1.0])
attack	1.0	(22,[1],[1.0])
attack	1.0	(22,[1],[1.0])
normal	0.0	(22,[0],[1.0])
attack	3.0	(22,[3],[1.0])
normal	0.0	(22,[0],[1.0])
normal	0.0	(22,[0],[1.0])

only showing top 20 rows

After the categorical data is converted into vector form then all input features are merged into a single-vector feature (sequential data) using the VectorAssembler as shown in Table 5.4. As can be seen from Table 5.4, all the features are combined into one column and formed a new column called indexed_features. Finally, the final output of the VectorAssembler is fed to a ML algorithm to train the data.

The other task done during preparing the dataset is feature selection. After the feature selection method is performed on the 41 features, 21 features that have higher AR values are selected for training and testing models. For more detail you can see the calculated attribute ratio for each of the 41 attributes from Annex D. We conducted two different scenarios for the selected four algorithms. In scenario one, the four algorithms are trained using their default parameters and evaluated on the testing data. In the second experiment, the hyperparameter tuning technique (grid search) with a 5-fold cross-validation method is performed on the selected algorithms to find the best parameters and models are retrained using these parameters. Finally, models are evaluated using the testing data. The

dataset used for this research has 148,517 network traffic data which are used for classification into normal and attack classes.

Table 5.4: Sample of data after vector assembling

labels2	labels2_index	labels5	labels5_index	indexed_features
normal	0.0	normal	0.0	(77,[0,1,2,3,4,5,...
normal	0.0	normal	0.0	(77,[1,2,3,4,5,6,...
attack	1.0	DoS	1.0	(77,[0,1,2,3,5,6,...
normal	0.0	normal	0.0	(77,[0,1,2,3,4,5,...
normal	0.0	normal	0.0	(77,[0,1,2,3,4,5,...
attack	1.0	DoS	1.0	(77,[0,1,2,3,5,6,...
attack	1.0	DoS	1.0	(77,[0,1,2,3,5,6,...
attack	1.0	DoS	1.0	(77,[0,1,2,3,5,6,...
attack	1.0	DoS	1.0	(77,[0,1,2,3,5,6,...
attack	1.0	DoS	1.0	(77,[0,1,2,3,5,6,...
attack	1.0	DoS	1.0	(77,[0,1,2,3,5,6,...
attack	1.0	DoS	1.0	(77,[0,1,2,3,5,6,...
normal	0.0	normal	0.0	(77,[0,1,2,3,4,5,...
attack	1.0	R2L	3.0	(77,[0,1,2,3,4,5,...
attack	1.0	DoS	1.0	(77,[0,1,2,3,5,6,...
attack	1.0	DoS	1.0	(77,[0,1,2,3,5,6,...
normal	0.0	normal	0.0	(77,[0,1,2,3,4,5,...
attack	1.0	Probe	2.0	(77,[1,2,3,4,5,6,...
normal	0.0	normal	0.0	(77,[0,1,2,3,4,5,...
normal	0.0	normal	0.0	(77,[0,1,2,3,4,5,...

only showing top 20 rows

The training data contains 123,384 instances (67,343 normal and 56,041 attack). The test set contains 25,133 instances (13,328 normal and 11,805 attack). The experiment is done for both binary and five-class classification.

5.4 Performance Evaluation Results

In this Section, the experimental results on both scenarios (learning with default parameters and learning using best parameters by applying the hyperparameter technique) for the selected classification algorithms are interpreted. All the classification models are evaluated by using a 2x2 and 5x5 confusion matrix. This confusion matrix shows the number of correct and incorrect predictions made by the classification model compared to the target values in the data by calculating the number of test samples under four categories (TP, TN, FP, FN). These categories are already discussed in Chapter 2 under Section 2.8. In this section the actual results calculated based on these categories are presented for each of the models. Using default parameters, the NN has obtained 88% accuracy. While the accuracy of 80%, 79%, and 76% are found for the RF, DT, and LR respectively as shown in Tables 5.5 and 5.6. The NN classifier is the best performer compared to the other two

classifiers in both scenarios while the RF is the second-best performer. The NN achieved an overall classification accuracy of 88% and 99.9% with default parameters and using the hyperparameters technique respectively. RF obtained better classification accuracy than DT and LR in both scenarios next to NN.

To improve the accuracy of the three classifiers further, a hyperparameter tuning method is performed. Table 5.5, shows the accuracy, precision, recall, and f1-score results of the four classifiers before the hyperparameter tuning technique is applied. As can be seen in Tables 5.5 and 5.6, the accuracy of the four classifiers trained on their default parameters is lower than the accuracy gained by trained on best parameters by using the hyperparameter tuning technique.

Table 5.5: Accuracy of the four classifiers before hyperparameter tuning applied

Classifiers	Metrics			
	Accuracy	Precision	Recall	F1-score
RF	80%	98.8%	53%	69%
LR	76%	78%	52%	69%
DT	79%	97.8%	55.8%	71%
NN	88%	88.7%	88.6%	88%

Table 5.6: Accuracy of the four classifiers after hyperparameter tuning applied

Classifiers	Metrics			
	Accuracy	Precision	Recall	F1-score
RF	99.8%	99.7%	99.6%	99.5%
LR	96.8%	97.8%	95.4	96.5%
DT	99.2%	99.7%	99.3	99.2%
NN	99.9%	99.8%	99.7%	99.7%

The four selected classification algorithms are applied for both binary class in which samples are categorized into normal or attack class; and five-class classification where samples are categorized into normal, DoS, U2R, Probe, or R2L. The classification outputs of both classifications are presented in the next sections.

5.4.1 Binary-class Classification

Figure 5.1, shows the confusion matrix of RF for the two-class classification. It represents the TP, FP, FN and TN results of the model after tested on the test set. From this confusion matrix, class Normal represents negative whereas class Attack represents positive.

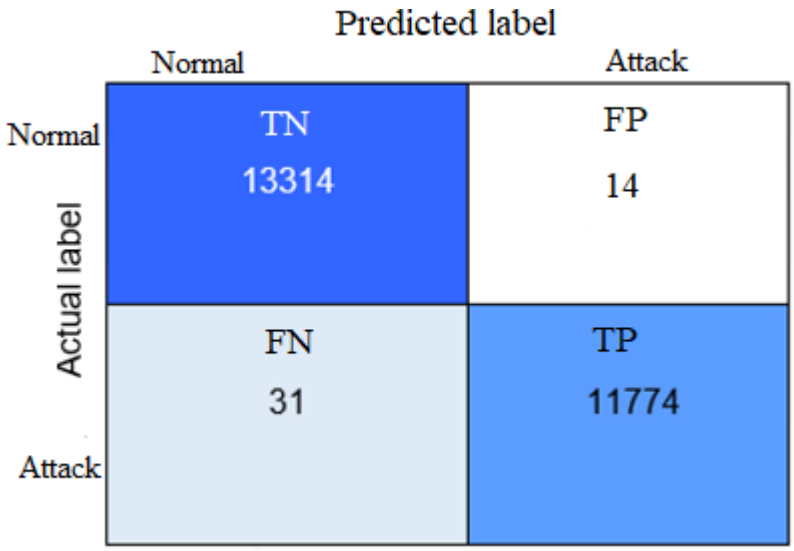

Figure 5.1: Confusion matrix for RF on the binary class classification

The confusion matrix in Figure 5.1 illustrates the predicted and actual classes of test set samples using a RF classifier for the two-class classification. The classifier obtained 11,774 true positive which is 99.7% that means among the total number of positives in the test set the model correctly classified 11,774 samples as positive. Among the total number of negative samples in the test set, 13,314 samples are correctly classified as negative by the model which is 99.89%. Regarding this count, the model gained a larger number of true positive and true negative results with minimum false positive (14) and false negative (31). Thus the model well performed on the binary class. The comparison of precision, recall, and f1-score of the four classifiers is given in Figure 5.2.

As can be seen from Figure 5.2, the DT scored 0.997 precision, 0.993 recall and 0.992 f1-score; LR obtained 0.978 precision, 0.954 recall and 0.965 f1-score; NN obtained 0.998

precision, 0.997 recall, and 0.997 f1-score; and RF classifier obtained 0.997 precision, 0.996 recall, 0.995 f1-score.

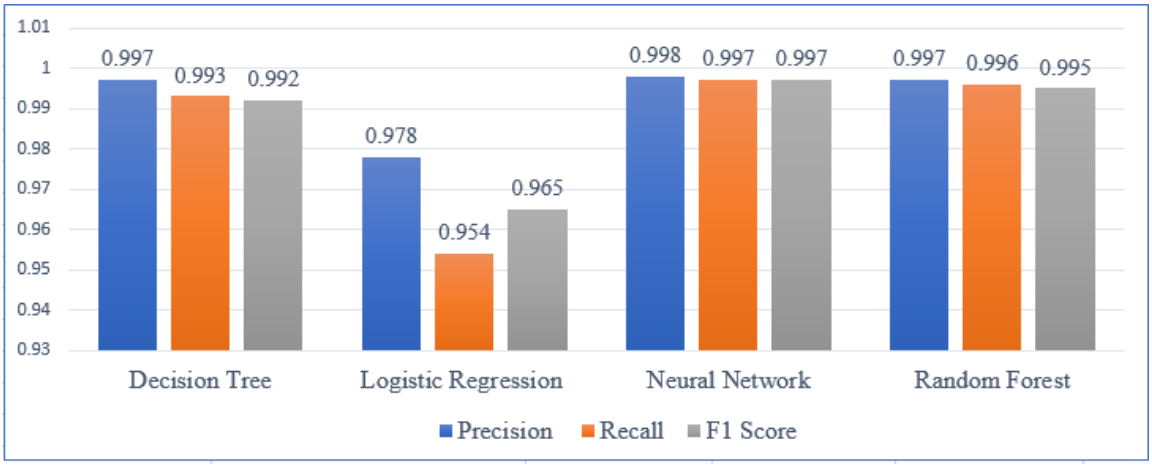

Figure 5.2: Graph comparing precision, recall, and F1-score of four classifiers

According to this graphical result, NN achieved the highest result of precision, recall, and f1-score than the other three classifiers. The RF classifier is the second-best performer, whereas the LR model is the least performer on all metrics. The different classifiers were separately evaluated regarding their performance using a confusion matrix and the confusion matrix result of these classifiers is graphically shown in Figure 5.3.

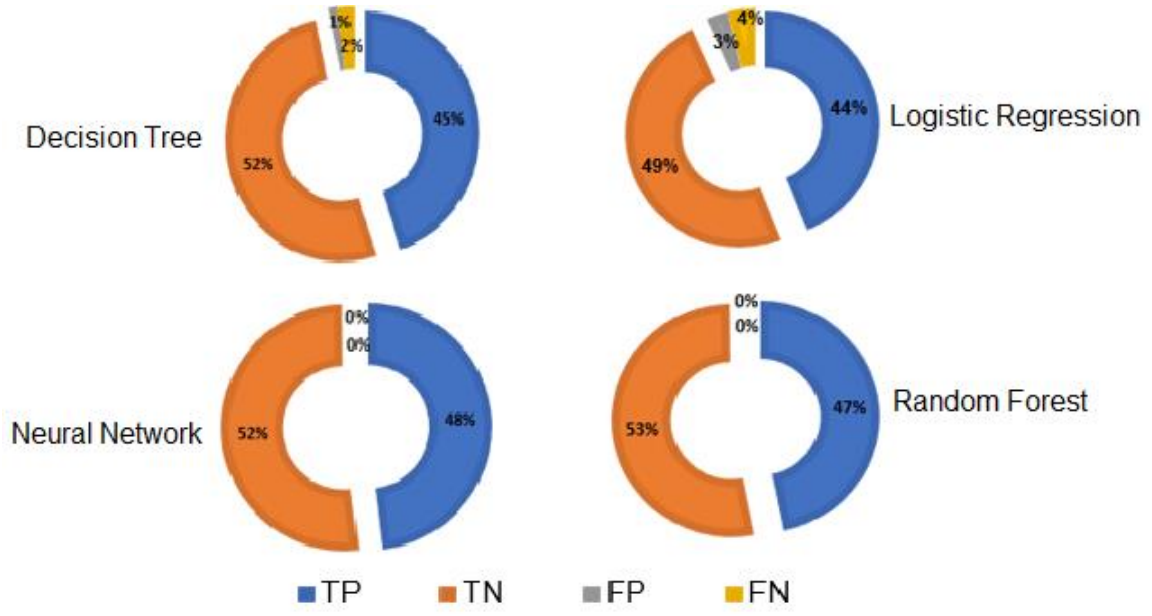

Figure 5.3: Confusion matrix result of the four models for two-class classification

The four pie charts in Figure 5.3 illustrate the coverage of test samples in relation to the predicted and actual classes of samples. All the classifiers have a greater number of true negative and true positive samples than the other categories of samples. This shows all the

classifiers are good at realizing the normal samples that they are “NORMAL” and the attack samples that they are ‘ATTACK’. The classifiers in the third and fourth pie charts that are NN and RF have the greatest coverage of true positive samples, no false positive, and false negative predicted by these two classifiers. LR has the greatest coverage of false positives than the others. This tells that LR is predicting normal samples as ‘ATTACK’ too much than the others. According to this distribution result, NN obtained better overall accuracy than the others. The RF classifier achieved a good classification performance next to NN.

In Table 5.7, a sample of test set predictions made by the RF model on the two-class is presented. It shows the actual label of the data and the prediction label made by the model. Here, the labels2 _index column contains the actual label of the samples, the prediction column shows the model prediction for each of the records in the test data. As can be seen from this figure, the model correctly classified the first 10 rows of samples to their belongings class.

Table 5.7: Sample of test set predictions using RF

indexed_features	labels2_index	labels2	prediction
(77,[0,1,2,3,5,6,...	1.0	attack	1.0
(77,[0,1,2,3,5,6,...	1.0	attack	1.0
(77,[0,1,2,3,4,5,...	0.0	normal	0.0
(77,[0,1,2,3,4,5,...	0.0	normal	0.0
(77,[0,1,2,3,5,6,...	1.0	attack	1.0
(77,[0,1,2,3,4,5,...	0.0	normal	0.0
(77,[1,2,3,4,5,6,...	0.0	normal	0.0
(77,[0,1,2,3,4,5,...	0.0	normal	0.0
(77,[0,1,2,3,5,6,...	1.0	attack	1.0
(77,[0,1,2,3,5,6,...	1.0	attack	1.0

only showing top 10 rows

5.4.2 Five-class Classification

The classification performance result for RF using the confusion matrix on the five-class is illustrated in Table 5.8. As can be seen from the table, the TP, TN, FP and FN counts of each class (Normal, DoS, Probe, R2L and U2R) are presented. The model gained the highest true positive and true negative results for each class. For the first class i.e., Normal class, 13,321 samples are correctly classified as normal out of the total normal test samples (13,328), one normal sample is wrongly classified as DoS, three normal samples are wrongly classified as Probe, two normal samples are wrongly classified as R2L, and one normal sample is wrongly classified as U2R. For the DoS class, among the total DoS test

samples, 3 DoS samples are wrongly classified as normal, 9189 DoS samples are correctly classified as DoS, and there is no incorrect classified sample as either Probe, R2L or U2R. This shows that the model has performed well in the DoS class.

Table 5.8: Confusion matrix of RF on the five-class classification

Actual label \ Predicted label	Normal	DOS	Probe	R2L	U2R
Normal	13321	1	3	2	1
DOS	3	9189	0	0	0
Probe	21	0	2364	0	0
R2L	17	0	0	195	1
U2R	6	0	0	0	9

For the Probe class, among the total Probe test samples 2,364 are correctly classified as Probe, 21 are incorrectly classified as normal, and there are no incorrect classified samples as either DoS, R2L or U2R. For the R2L class, among the total R2L test samples 195 are correctly classified as R2L, 17 are incorrectly classified as normal, and there are no incorrect classified samples as either DoS, Probe or U2R. For the U2R class, among the total U2R test samples 9 are correctly classified as U2R, 6 are incorrectly classified as normal, and there are no incorrect classified samples as either DoS, Probe or R2L.

Figure 5.4, shows the classification accuracy for the different algorithms. From this figure, the accuracy of the NN is comparatively higher than the other three classifiers. The overall accuracy of 99.9%, 99.8%, 99.2%, and 96.8% is gained by NN, RF, DT and LR respectively.

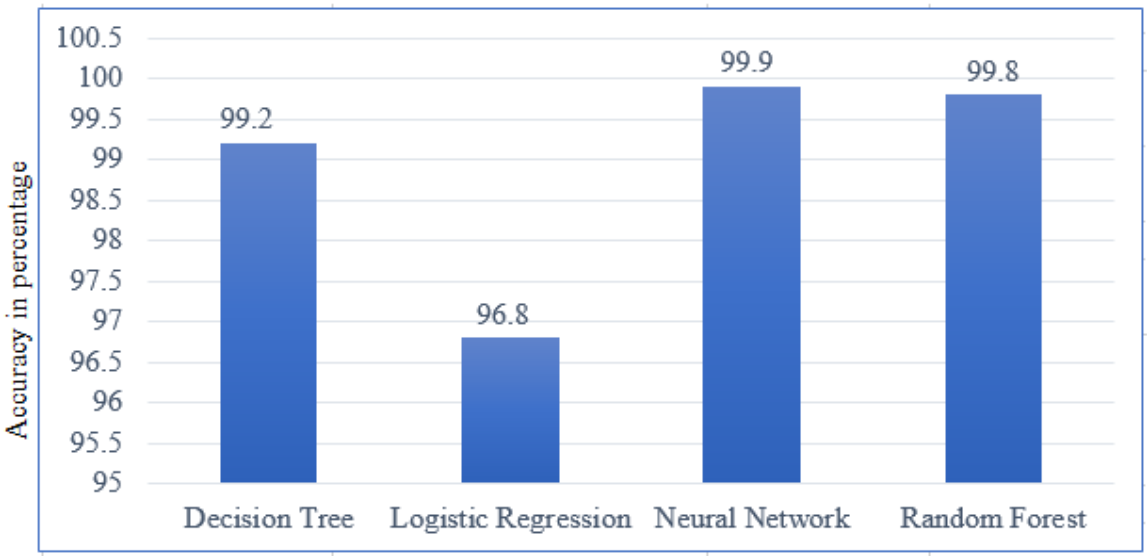

Figure 5.4: Graph Showing the percentage accuracy of four classifiers on five-class

The detection performance of the four classifiers on the five-class is presented in Table 5.9. This table shows the detection rate and FAR results of the four models for the four attack classes: DoS, Probe, R2L, and U2R. The results show that the classification models except LR performed very well in the detection of DoS attacks. Compared with the three classifiers, the LR obtained less detection rate and high FAR on the DoS attack and it didn't detect the Probe, R2L and U2R attacks. Although all the classification algorithms provide good performance results in the detection DoS attack, the least performance results were found for the U2R attack.

Table 5.9: The detection result of the four algorithms

Classifier	Metric	DoS	Probe	R2L	U2R
DT	TP(DR)	99.8%	99.2%	93.8%	84.6%
	FP(FAR)	0.00031	0.00048	0.00036	0.000079
LR	TP(DR)	97.9%	0.0	0.0	0.0
	FP(FAR)	0.157	0.0	0.0	0.0
RF	TP(DR)	99.9%	99.1%	91.5%	60%
	FP(FAR)	0.000062	0.00013	0.00008	0.000039
NN	TP(DR)	99.9%	99.4%	92.3%	61.4%
	FP(FAR)	0.000054	0.00012	0.00006	0.000033

In Table 5.10, a sample of test set prediction results made by the RF on five-class classification is given. The labels5_index column contains the actual label (class) of samples and the prediction column includes the predicted class of these samples. As can be seen from Table 5.10, the model gained good performance results on the five-class classification.

Table 5.10: Sample of predictions on the test set for the five-class classification

indexed_features	labels5_index	labels5	prediction
(77,[0,1,2,3,5,6,...]	1.0	DoS	1.0
(77,[0,1,2,3,5,6,...]	1.0	DoS	1.0
(77,[0,1,2,3,4,5,...]	0.0	normal	0.0
(77,[0,1,2,3,4,5,...]	0.0	normal	0.0
(77,[0,1,2,3,5,6,...]	1.0	DoS	1.0
(77,[0,1,2,3,4,5,...]	0.0	normal	0.0
(77,[1,2,3,4,5,6,...]	0.0	normal	0.0
(77,[0,1,2,3,4,5,...]	0.0	normal	0.0
(77,[0,1,2,3,5,6,...]	1.0	DoS	1.0
(77,[0,1,2,3,5,6,...]	1.0	DoS	1.0

only showing top 10 rows

5.5 Comparison

To compare with the existing anomaly detection techniques in references [78, 80, 87], we conduct experiments on NSL-KDD dataset. In reference [78], network anomaly detection based on NSL-KDD dataset was explored. CFS and chi-squared feature selection methods were applied to the dataset to remove the highly correlated attributes. In reference [80], the k-means method with information gain feature selection method was proposed for detecting network anomalies. In reference [87], the author proposed a supervised based feature selection method based on datamining algorithms to identify relevant features from the NSL-KDD dataset. The detection accuracy of the algorithms was compared and wrapper-Bayesnet method was obtained an accuracy of 95.3% and FAR of 0.006, while the FAR and FN rate of our approach is 0.0%, which means our approach is able to recognize the normal and attack data.

The feature selection method used in our proposed work has ease of calculation than the other methods used in the three referenced papers. The detection accuracy of the studies is also lower than our approach. Table 5.11 compares the detection accuracy of previous studies with our proposed research work. Based on these comparison results, our approach outperforms the studies in references [78, 80, 87] in network anomaly detection.

Table 5.11: Comparison with previous studies

Approach	Accuracy
CFS and chi-squared based	92.13%
k-means with IG based	89.6%
Wrapper-Bayesnet based	95.3%
Our approach	99.8%

6. CONCLUSION AND FUTURE WORKS

6.1 Conclusion

The rapid development of electronic devices and extensive dependence on Internet-based applications for both business and pleasure activities has led to an ever-increasing network or Internet communications. However, security issues have been one of the most critical problems for organizations as attackers also increased from time to time. Because of the increasing number of Internet users and complex attacks in computer networks, there should be a system that protects such security issues. ML techniques are one of the technologies that are applied to network IDSs. The main aim of this research work is to build an efficient model for the classification of network traffic into normal and attack using different ML techniques. These techniques are implemented using Apache Spark which is a fast data processing framework that includes various libraries for ML applications. The ML techniques are validated in two ways. In the first experiment, the models are trained with their default parameters and tested on a separate test set. In the second experiment, a 5-fold cross-validation with grid search techniques are applied to the ML techniques to find the optimal value for each hyperparameter to improve the accuracy of the models.

The implementation started with preprocessing of the dataset and features are selected from the preprocessed data and these selected features are used for training and testing the classifiers. During the classification, four different ML models (DT, RF, LR and NN) are created, trained, tested and compared based on their evaluation result. To accomplish this study, the NSL-KDD dataset is used to evaluate the performance of the proposed models which is taken from the University of New Brunswick. The models are implemented in Python programming language using this dataset.

Based on the evaluation results of the different models presented in Chapter 5, DT scored classification accuracy of 99.2%, precision of 99.7%, and recall of 99.3% with f1-sore of 99.2%; LR scored classification accuracy of 96.8%, precision of 97.8% and recall of 95.4 with f1-sore of 96.5 and RF scored classification accuracy of 99.8%, precision of 99.7%, and recall of 99.6% with f1-score of 99.5%. From this, we can conclude that all the models except the LR, are specifying all normal samples and good at detecting attacks and generally NN model outperforms in all evaluations and the LR is the model with least

scorers than others. The evaluation result shows the proposed approach provides effective detection of network intrusions.

6.2 Contributions

The contributions of this research work are:

- Enhancement in the detection accuracy of previous works related to network intrusion detection.
- Improve the detection performance by removing the irrelevant features from the dataset.

6.3 Future Works

This research work explores different points that can be further improved for better functioning of the system.

- Applying the system on different datasets
- Detecting network intrusions from real network using integrated ML algorithms
- Implementing the system on multiple machines
- Exploring different feature selection methods by combining our approach with other techniques

REFERENCES

[1] Joseph Migga Kizza, Computer Communications and Networks, USA: Springer, 2009.

[2] Symantec, "ISTR Internet Security Threat Report," 23 April 2018. Available at: Https://www.symantec.Com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf. Last Accessed on 25 December 2018.

[3] San Jose, "Cisco Predicts More IP Traffic in the Next Five Years Than in the History of the Internet". Available at: https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1955935, Last Accessed on 20 December 2019.

[4] Mohit Tiwari, Raj Kumar, Akash Bharti and Jai Kishan, "Intrusion Detection System," *International Journal of Technical Research and Applications,* Vol. 5, 2017.

[5] Pedro Garcia-Teodoro, JesuS Diaz-verdejo and Gabriel Macia-Fernandez, "Anomaly-Based NetworkIntrusion Detection:Techiques,Systems and Challenges," *Computer Security,* Vol. 28, 2009.

[6] Ahmad sharifi, Akram Noorollahi and Farnoosh Farokhmanesh, "Intrudion Detection and Prevention Systems and Security Issues.," *Internation Journal of Computer Science and Network Security (IJCSNS),* Vol. 14, 2014.

[7] "Apache Spark," Databricks, Available at: https://databricks.com/spark/about. Last Accessed on 7 December 2019.

[8] Ahmed, Amrita & P, "A Study of Feature Selection Methods in Intrusion Detection System: A Survey," *International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR),* Vol. 2, 2012.

[9] Tavallaee, Mahbod and Bagheri, Ebrahim and Lu, Wei and Ghorbani, Ali A, "A Detailed Analysis of the KDD CUP 99 Dataset," *in Symposium on Computational Intelligence for Security and Defense Applications,* IEEE, 2009, PP. 1--6.

[10] Potluri, Sasanka and Diedrich, Christian, "Accelerated Deep Neural Networks for Enhanced Intrusion Detection System," *IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA),* PP. 1--8, 2016.

[11] Ajay yadav," Network Design," Available at: https://resources.infosecinstitute.com/network-design-firewall-idsips/. Last Accessed on 18 November 2019.

[12] Alex Galea, Luis Capelo, Applied Deep Learning with Python, Packt Publishing, 2018.

[13] Andropov, Sergey and Guirik, Alexei and Budko, Mikhail and Budko, Marina, "Network Anomaly Detection Using Artificial Neural Networks," *IEEE*, 2017.

[14] Monowar H. Bhuyan, Dhruba K. Bhattacharyya, Jugal K. Kalita, Network Traffic Anomaly Detection and Prevention, Springer Nature, 2017.

[15] Veeramreddy Jyothsna and Koneti Munivara Prasad, "Anomaly-Based Intrusion Detection System,"Available at:https://www/Anomaly-Based Intrusion Detection System/IntechOpen.html. Last Accessed on 16 January 2020.

[16] A.Jamadar, Riyazahmed, "Network Intrusion Detection System Using Machine Learning," *Indian Journal of Science and Technology,* Vol. 11(48), 2018.

[17] Dhruba Kumar, Bhattacharyya Jugal, and Kumar Kalita, Network Anomaly Detection a Machine Learning Perspective, 2014.

[18] Maleh, Yassine and Ezzati, Abdellah and Qasmaoui, Youssef and Mbida, Mohamed, "A Global Hybrid Intrusion Detection System for Wireless Sensor Networks," *Procedia Computer Science,* Vol. 52, PP. 1047--1052, 2015.

[19] Vajiheh Hajisalem, Shahram Babaie, "A Hybrid Intrusion Detection System Based on ABC-AFS Algorithm for Misuse and Anomaly Detection," *Computer Networks,* PP. 37--50, 2018.

[20] Mehrnaz Mazini, Babak Shirazi, and Iraj Mahdavi, "Anomaly Network-Based Intrusion Detection System Using a Reliable Hybrid Artificial Bee Colony and Adaboost Algorithms," *Journal of King Saud University – Computer and Information Sciences,* 2018.

[21] Karimi, Ahmad Maroof, "Distributed Machine Learning Based Intrusion Detection System," *Masters Thesis*, 2016.

[22] Liu Hua Yeo, Xiangdong Che,, "Understanding Modern Intrusion Detection Systems: A Survey," 2016.

[23] Randy Weaver, Dawn Weaver and Dean Farwood, Guide to Network Defence and Countermeasures, US: Information Security Professionals, 2014.

[24] Pathan, Al-Sakib Khan, The State of the Art in Intrusion Prevention and Detection, CRC Press, 2016.

[25] Alazab, Ammar and Hobbs, Michael and Abawajy, Jemal and Khraisat, Ansam and Alazab, Mamoun, "Using Response Action With Intelligent Intrusion Detection and Prevention System Against Web Application Malware," *Information Management and Computer Security,* Vol. 22, PP. 431--449, 2014.

[26] Modi, Chirag and Patel, Dhiren and Borisaniya, Bhavesh and Patel, Hiren and Patel, Avi and Rajarajan, Muttukrishnan, "A Survey of Intrusion Detection Techniques in Cloud," *Journal of Network and Computer Applications*, Vol. 36, PP. 42--57, 2013.

[27] Nathan Einwechte, "An Introduction to Distributed Intrusion Detection Systems," Available at: https://www.symantec.com/connect/articles/introduction-distributed-intrusion-detection-systems. Last Accessed on 27 Feburary 2020.

[28] "Active and passive IDS,"Available at:https://www.dummies.com/computers/operating-systems/windows-xp-vista/examining-different-types-of-intrusion-detection-systems/, Last Accessed on 27 Feburary 2020.

[29] Singh, Simranjeet, "A Hybrid Intrusion Detection System Design for Computer Network Security," *International Journal of Engineering Sciences and Research and Technology*, 2018.

[30] Eric Meyer, "Defending Your Data: Securing Against Internal and External Threats,"Available at:https://inform.tmforum.org/features-and-analysis/2016/03/defending-your-data-securing-against-internal-and-external-threats/, Last Accessed on 3 January 2020.

[31] Khan, Ruzaina and Hasan, Mohammad, "Network Threats, Attacks and Security Measures: A Review," *International Journal of Advanced Research in Computer Scienc*, Vol. 8, 2017.

[32] Thomas, Ciza and Sharma, Vishwas and Balakrishnan, N, "Usefulness of DARPA Dataset for Intrusion Detection System Evaluation," *International Society for Optics and Photonics*, Vol. 6973, 2010.

[33] Gerry Saporito, "A Deeper Dive into the NSL-KDD Dataset," Available at: https://towardsdatascience.com/a-deeper-dive-into-the-nsl-kdd-data-set-15c753364657, Last Accessed on 23 December 2019.

[34] Sibanjan Das, Umit Mert Cakmak, Hands-on Authomated Machine Learning: A Beginners Guide to Building Automated Machine Learning Systems Using AutoML and Python, Packt Publishing Ltd, 2018.

[35] Garrido, a. P. J. Gabriel, OpenCV 3.x with Python by Example, Birmingham:, Packt Publishing Ltd, 2018.

[36] Rajanarayanan Thottuvaikkatumana, Apache Spark 2 for Beginners, Packt, 2016.

[37] Birmingham, Hands-On Automated Machine learning, Packt Publishing Ltd, 2018.

[38] Mark Ryan M. Talabis, D. Kaye, "Supervised Learning," Available at: https://www.sciencedirect.com/topics/computer-science/supervised-learning, Last Accessed on 3 January 2020.

[39] Dipanjan Sarkar, Raghav Bali and Tushar Sharma, Practical Machine Learning With Python, Apress, 2018.

[40] Burkov, Andriy, The Hundred-Page Machine Learning Book, 2019.

[41] Dipanjan Sarkar, Raghav Bali and Tamoghna Ghosh, Hands-On T ransfer Learning with Python, Packt Publishing, 2018.

[42] Sunshine10,"Unsupervised Learning,"Available at:

https://whatis.techtarget.com/definition/unsupervised-learning, Last Accessed on 3 January 2020.

[43] Dipanjan Sarkar, Raghav Bali , and Tushar Sharma, Practical Machine Learning with Python : A Problem-Solver's Guide to Building Real-world Intelligent Systems, Apress, 2018.

[44] Polamuri, Saimadhu, "How the Logistic Regression Model Works," Available at: https://dataaspirant.com/2017/03/02/how-logistic-regression-model-works/, Last Accessed on 6 November 2019.

[45] Brownlee, Jason, Master Machine Learning Algorithms, 2017.

[46] Saxena, Rahul, "How Decision Tree Algorithm Works," Available at: https://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works/, Last Accessed on 6 November 2019.

[47] Tagliaferri, Lisa, "An Introduction to Machine Learning," Available at: https://www.digitalocean.com/community/tutorials/an-introduction-to-machine-learning, Last Accessed on 6 November 2019.

[48] T, Sam, "Entropy: How Decision Trees Make Decisions," Available at: https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8, Last Accessed on 6 November 2019.

[49] R. Praveena Priyadarsini, M.L.Valarmathi, and S. Sivakumari, "Gain Ratio Based Feature Selection Method for Privacy Preservation," *Ctact Journal on Soft Computing,* Vol. 01, 2011.

[50] Brownlee, Jason, Master Machine Learning Algorithms: Discover How They Work and Implement them from Scrach, 2017.

[51] V Enkat N. Gudiv Ada, Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications, US: Elsevier, 2018.

[52] Lewis, N.D, Machine Learning Made Easy with R, Press, 2017.

[53] Singh, Pramod, Learn PySpark, India: Apress, 2019.

[54] Guido, Andreas C. Mueller and Sarah, Introduction to Machine Learning with Python, USA: O'Reilly Media, 2016.

[55] Rifkie Primartha, Bayu Adhi Tama, "Anomaly Detection using Random Forest: A Performance Revised," *International Conference on Data and Software Engineering (ICoDSE),* 2017.

[56] Qi, yanjun, "Random Forest for Bioinformatics," in *Ensemble Machine Learning*, US, Springer, 2012, PP. 307--323.

[57] Ali, Jehad, et al,, "Random Forests and Decision Trees," *International Journal of Computer Science Issues (IJCS)* , Vol. 9.5, 2012.

[58] Zulkernine, Mehdi Moradi and Mohammad, "A Neural Network Based System for Intrusion Detection and Classification of Attacks," *IEEE International Conference on Advances in Intelligent Systems,* 2014.

[59] Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, Silas Franco dos Reis Alves, Artificial Neural Networks :A Practical Course, Springer Nature, 2017.

[60] Torfi, Amirsina, "Cross validation,"Available at: https://machine-learning-course.readthedocs.io/en/latest/content/overview/overfitting.html, Last Accessed on 30 October 2019.

[61] Dipanjan Sarkar, Raghav Bali , and Tamoghna Ghosh, Implement Advanced Deep Learning and Neural Network Models Using Tensorflow and Keras, Packt Publishing Ltd, 2018.

[62] Andreas C. Mueller and Sarah Guido, Introduction to Machine Learning with Python a Guide for Data Scientists, USA: O'Reilly Media, 2016.

[63] Brownlee, Jason, Machine Learning Mastery with Python, 2016.

[64] Krupa Joel Chabathula, Jaidhar C.D and Ajay Kumara M.A, "Comparative Study of Principal Component Analysis Based Intrusion Detection Approach Using Machine Learning Algorithms," in *International Conference on Signal Processing, Communication and Networking (ICSCN)*, 2015.

[65] Prashant Kushwaha, Himanshu Buckchash, and Balasubramanian Raman, "Anomaly Based Intrusion Detection Using Filter Based Feature Selection on KDD CUP 99," *IEEE Region 10 Conference (TENCON),* 2017.

[66] Ammar Alazab, Michael Hobbs, Jemal Abawajy, Moutaz Alazab, "Using Feature Selection for Intrusion Detection System," *International Symposium on Communications and Information Technologies (ISCIT),* 2012.

[67] Ahmed Fawzy Gad, Practical Computer Vision Applications Using Deep Learning With CNNs, Egypt: Apress, 2018.

[68] Taylor & Francis Group, Feature Engineering for Machine Learning and Data Analytics, London (New York): CRC Press, 2018.

[69] Anuja Nagpal, "L1 and L2 Regularization Methods,"Available at: https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c, Last Accessed on 27 Feburary 2020.

[70] "Evaluating Models: Classification Metrics,"Available at: https://apple.github.io/turicreate/docs/userguide/evaluation/classification.html, Last Accessed on 9 December 2019.

[71] Hamed Habibi Aghdam, Elnaz Jahani Heravi, Guide to Convolutional Neural Networks, Springer, 2017.

[72] Pramod Singh, Machine Learning with PySpark: With Natural Language Processing and Recommender Systems, Apress, 2019.

[73] Beyeler, Michael, Machine Learning for OpenCV, Packt Publishing Ltd., 2017.

[74] Qais Saif Qassim, Abdullah Mohd Zin, and Mohd Juzaiddin Ab Aziz, "Anomalies Classification Approach for Network-based Intrusion Detection System," *International Journal of Network Security,* Vol. 18, 2016.

[75] Karami, Amin, "Ananomaly-Based Intrusion Detection Systemin Presence of Benign Outliers with Visualization Capabilities," *Expert Systems with Applications,* 2018.

[76] Irfan Sofi, Amit Mahajan, and Vibhakar Mansotra, "Machine Learning Techniques Used for the Detection and Analysis of Modern Types of DDoS Attacks," *International Research Journal of Engineering and Technology (IRJET),* Vol. 04, 2017.

[77] Vivek kshirsagar, Madhuri S.Joshi, "Rule Based Classifier Models for Intrusion Detection System," *International Journal of Computer Science and Information Technologies,* Vol. 7, 2016.

[78] Govind P Gupta, Manish Kulariya, "A Framework for Fast and Efficient Cyber Security Network Intrusion Detection Using Apache Spark," *International Conference on Advances inComputing & Communications, ICACC,* 2016.

[79] Tegegn Kebebaw, "Neural Network Based Malware and Suspicious Activities Detection Based on User Activity," *Addis Abeba, Ethiopia,* 2018.

[80] David Ahmad Effendy, Kusrini Kusrini and Sudarmawan Sudarmawan, "Classification of Intrusion Detection System (IDS) Based on Computer Network," in *International Conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE),* 2017.

[81] Jamal Esmaily, Reza Moradinezhad and Jamal Ghasemi, "Intrusion Detection System Based on Multi-Layer Perceptron Neural Networks and Decision Tree," *International Conference on Information and Knowledge Technology,* 2015.

[82] Sheraz Naseer, Yasir Saleem, Shehzad Khalid, M Khawar Bashir, Jihun Han, M Munwar Iqbal and Kijun Han, "Enhanced Network Anomaly Detection Based on Deep Neural Network," *Journal of Latx Class files,* Vol. 14, 2015.

[83] Abdulla Amin, Mamun Bin, "A Survey of Intrusion Detection Systems Based on Ensemble and Hybrid Classifiers," *Computer and security,* 2017.

[84] Bhupendra Ingre, Anamika Yadav and Atul Kumar Soni, "Decision Tree Based Intrusion Detection System for NSL-KDD Dataset and KDD Dataset," in *Conference Paper in Smart Innovation*, India, 2017.

[85] Tewodros Getaneh, "Cyber Security Practices and Challenges at Selected Critical Infrastructures In Ethiopia: Towards Tailoring Cyber Security Framework," *Addis Abeba. Ethiopia,* 2018.

[86] Krupa Joel Chabathula, Jaidhar C.D and Ajay Kumara M.A, "Comparative Study of Principal Component Analysis Based Intrusion Detection Approach Using Machine Learning Algorithms," in *International Conference on Signal Processing, Communication and Networking (ICSCN),* 2015.

[87] Martha Teffera, "A Data Mining Approach for Intrusion Detection System Using Wrapper Based Feature Selection Method," *Masters Thesis,* 2014.

[88] Hee-su Chae, Sang Hyun Choi, "Feature Selection for Efficient Intrusion Detection Using Attribute Ratio," *International Journal of Computers and Communications,* Vol. 8, 2014.

[89] Available at: https://www.unb.ca/cic/datasets/nsl.html.

[90] F, Stephen D. Bay and Dennis, "Machine Learning Repository," UCI, Available at :https://archive.ics.uci.edu/ml/machine-learning-databases/kddcup99-mld/, Last Accessed on 11 November 2019.

[91] Jamal H. Assi, Ahmed T.Sadiq, "NSL-KDD dataset Classification Using Five Classification Methods and Three Feature Selection Strategies," *Journal of Advanced Computer Science and Technology Research,* Vol. 7, 2017.

[92] L.Dhanabal, Dr. 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, No. 6, 2015.

[93] Alex Bekker, "Spark vs. Hadoop MapReduce," Databricks, Available at: https://www.scnsoft.com/blog/spark-vs-hadoop-mapreduce, Last Accessed on 7 December 2019.

[93] "Apache Spark: Lightning-fast Cluster Computing,"Available at: https://svn.apache.org/repos/asf/spark/site/index.html, Last Accessed on 7 December 2019.

[95] M. Beyeler, Machine Learning for OpenCV, Packt Publishing Ltd, 2017.

[96] Available at: https://docs.spyder-ide.org/, Last Accessed on 29 December 2019.

Annex A: Sample Code for Data Preprocessing

```
# StringIndexer Algorithm

StringIndexer(inputCol='raw_features',outputCol='indexed_fe
atures’,maxCategories=2)

#One Hot Encoding Algorithm

Encoder=OneHotEncoderEstimator(inputCols=[stringIndexer.get
OutputCol()], outputCols=[categoricalCol + \"classVec\"])

#Feature Selection

ar_dict = getAttributeRatio(train_df, numeric_cols,
binary_cols, 'labels5')

Selected_features=selectFeaturesByAR(ar_dict, 3.08),
outputCol='raw_features')

#Normalization by Using StandardScaler

train_scaler = [*binary_cols, *list(map(standardizer,
numeric_cols)), *['id', 'labels2_index', 'labels2',
'labels5_index', 'labels5']]

test_scaler = [*test_binary_cols, *list(map(standardizer,
numeric_cols)), *['id', 'labels2_index', 'labels2',
'labels5_index', 'labels5']]

#VectorAssembler Algorithm

VectorAssembler(inputCols=selectFeaturesByAR (ar_dict,
3.08), outputCol='raw_features')
```

Annex B: Sample Screenshot of Hyperparameter Tuning

(a) Sample of hyperparameter tuning with best hyperparameter values for DT

```
[([{'maxDepth': 2}, {'maxBins': 10}], 0.99253841556254441),
 ([{'maxDepth': 2}, {'maxBins': 20}], 0.9853841556254441),
 ([{'maxDepth': 2}, {'maxBins': 40}], 0.9853841556254441),
 ([{'maxDepth': 2}, {'maxBins': 80}], 0.9853841556254441),
 ([{'maxDepth': 2}, {'maxBins': 100}], 0.9853841556254441),
 ([{'maxDepth': 5}, {'maxBins': 10}], 0.9853841556254441),
 ([{'maxDepth': 5}, {'maxBins': 20}], 0.9853841556254441),
 ([{'maxDepth': 5}, {'maxBins': 40}], 0.9853841556254441),
 ([{'maxDepth': 5}, {'maxBins': 80}], 0.9853841556254441),
 ([{'maxDepth': 5}, {'maxBins': 100}], 0.9853841556254441),
 ([{'maxDepth': 10}, {'maxBins': 10}], 0.9853841556254441),
 ([{'maxDepth': 10}, {'maxBins': 20}], 0.9853841556254441),
 ([{'maxDepth': 10}, {'maxBins': 40}], 0.9853841556254441),
 ([{'maxDepth': 10}, {'maxBins': 80}], 0.9853841556254441),
 ([{'maxDepth': 10}, {'maxBins': 100}], 0.9853841556254441),
 ([{'maxDepth': 20}, {'maxBins': 10}], 0.9853841556254441),
 ([{'maxDepth': 20}, {'maxBins': 20}], 0.9853841556254441),
 ([{'maxDepth': 20}, {'maxBins': 40}], 0.9853841556254441),
 ([{'maxDepth': 20}, {'maxBins': 80}], 0.9853841556254441),
 ([{'maxDepth': 20}, {'maxBins': 100}], 0.9853841556254441),
 ([{'maxDepth': 30}, {'maxBins': 10}], 0.9853841556254441),
 ([{'maxDepth': 30}, {'maxBins': 20}], 0.9853841556254441),
 ([{'maxDepth': 30}, {'maxBins': 40}], 0.9853841556254441),
 ([{'maxDepth': 30}, {'maxBins': 80}], 0.9853841556254441),
 ([{'maxDepth': 30}, {'maxBins': 100}], 0.9853841556254441)]
best parameter value:
  ([{'maxDepth': 2}, {'maxBins': 10}], 0.99253841556254441)
```

(b) Sample of hyperparameter tuning with best values for LR

```
[([{'regParam': 0.01}, {'elasticNetParam': 0.0}, {'maxIter': 1}], 0.9040974498794389),
([{'regParam': 0.01}, {'elasticNetParam': 0.0}, {'maxIter': 5}], 0.9667593918536855),
([{'regParam': 0.01}, {'elasticNetParam': 0.0}, {'maxIter': 10}], 0.9684948564567242),
([{'regParam': 0.01}, {'elasticNetParam': 0.5}, {'maxIter': 1}], 0.8949149124952419),
([{'regParam': 0.01}, {'elasticNetParam': 0.5}, {'maxIter': 5}], 0.9643057823920793),
([{'regParam': 0.01}, {'elasticNetParam': 0.5}, {'maxIter': 10}], 0.9684981733257731),
([{'regParam': 0.01}, {'elasticNetParam': 1.0}, {'maxIter': 1}], 0.8936220899992073),
([{'regParam': 0.01}, {'elasticNetParam': 1.0}, {'maxIter': 5}], 0.9641074122958229),
([{'regParam': 0.01}, {'elasticNetParam': 1.0}, {'maxIter': 10}], 0.9683469683069705),
([{'regParam': 0.5}, {'elasticNetParam': 0.0}, {'maxIter': 1}], 0.9040974498794389),
([{'regParam': 0.5}, {'elasticNetParam': 0.0}, {'maxIter': 5}], 0.9604288592052784),
([{'regParam': 0.5}, {'elasticNetParam': 0.0}, {'maxIter': 10}], 0.9613621129569004),
([{'regParam': 0.5}, {'elasticNetParam': 0.5}, {'maxIter': 1}], 0.5),
([{'regParam': 0.5}, {'elasticNetParam': 0.5}, {'maxIter': 5}], 0.8721166741623587),
([{'regParam': 0.5}, {'elasticNetParam': 0.5}, {'maxIter': 10}], 0.8952206600908498),
([{'regParam': 0.5}, {'elasticNetParam': 1.0}, {'maxIter': 1}], 0.5),
([{'regParam': 0.5}, {'elasticNetParam': 1.0}, {'maxIter': 5}], 0.5),
([{'regParam': 0.5}, {'elasticNetParam': 1.0}, {'maxIter': 10}], 0.5),
([{'regParam': 2.0}, {'elasticNetParam': 0.0}, {'maxIter': 1}], 0.9012474753328268),
([{'regParam': 2.0}, {'elasticNetParam': 0.0}, {'maxIter': 5}], 0.9182395947357591),
([{'regParam': 2.0}, {'elasticNetParam': 0.0}, {'maxIter': 10}], 0.9176644919624433),
([{'regParam': 2.0}, {'elasticNetParam': 0.5}, {'maxIter': 1}], 0.5),
([{'regParam': 2.0}, {'elasticNetParam': 0.5}, {'maxIter': 5}], 0.5),
([{'regParam': 2.0}, {'elasticNetParam': 0.5}, {'maxIter': 10}], 0.5),
([{'regParam': 2.0}, {'elasticNetParam': 1.0}, {'maxIter': 1}], 0.5),
([{'regParam': 2.0}, {'elasticNetParam': 1.0}, {'maxIter': 5}], 0.5),
([{'regParam': 2.0}, {'elasticNetParam': 1.0}, {'maxIter': 10}], 0.5)]
Best hyperparameters: ([{'regParam': 0.01}, {'elasticNetParam': 0.5}, {'maxIter': 10}],
0.9684981733257731)
```

Annex C: The 41 Features Provided by the NSL-KDD Dataset

No	Feature name	Description	Type
1	Duration	duration of connection in second (e.g. tcp, udp, icmp)	numeric
2	Protocal_type	Connection protocol	nominal
3	Service	Dst port mapped to service (eg. Http, ftp)	nominal
4	Flag	Normal or error status flag of connection	nominal
5	src_bytes	Number of data bytes from src to dst	numeric
6	dst_bytes	bytes from dst to src	numeric
7	Land	1 if connection is from / to the same host/port: else 0	binary
8	wrong_fragment	Number of 'wrong' fragments (eg. 0,1,3)	numeric
9	Urgent	Number of urgent packets	numeric
10	Hot	number of 'hot' indicators (bio-ids features)	numeric
11	num_failed_login	number of failed login attempts	numeric
12	logged_in	1 if successfully logged in; else 0	binary
13	num_compromized	number of 'compromised' conditions	numeric
14	root_shell	1 if root shell is obtained; else 0	binary
15	su_attempted	1 if 'su root' command attempted; else 0	binary
16	num_root	number of root access	numeric

No	Feature name	Description	Type
17	num_file_creation	number of file creation operations	numeric
18	num_shells	number of shell prompts	numeric
19	num_access_files	number of operations on access control files	numeric
20	num_outbound_cmds	number of outbound commands in an ftp session	numeric
21	is_host_login	1 if login belongs to 'hot' list (e.g. Root, admin); else 0	binary
22	is_guest_login	1 if login 'guest' login (e.g. Guest, anonymous); else 0	binary
23	Count	number of connections to same host as current connection in past two seconds	numeric
24	srv_count	number of connections to same service as current connection in past two seconds	numeric
25	serror_rate	% of connections that have 'SYN' errors	numeric
26	srv_serror_rate	% of connections that have 'SYN' errors	numeric
27	rerror_rate	% of connections that have 'REJ' errors	numeric
28	srv_serror_rate	% of connections that have 'REJ' errors	numeric
29	same_srv_rate	% of connections that have same service	numeric

No	Feature name	Description	Type
30	diff_srv_rate	% of connections to different services	numeric
31	srv_diff_host_rate	% of connections to different hosts	numeric
32	dst_host_count	Count of connections having same dst host	numeric
33	dst_host_srv_count	Count of connections having same host and using same service	numeric
34	dst_host_same_srv_rate	% of connections having same host dst and using same service	numeric
35	dst_host_diff_srv_rate	% of different services on current host	numeric
36	dst_host_same_srv_port_rate	% of connections to current host having same srv port	numeric
37	dst_host_srv_diff_host_rate	% of connections to same service coming from different hosts	numeric
38	dst_host_serror_rate	% of connections to current host that have an S0 error	numeric
39	dst_host_srv_serror_rate	% of connections to current host and specified service that have an S0 error	numeric
40	dst_host_rerror_rate	% of connections to current host that have an RST error	numeric
41	dst_host_srv_rerror_rate	% of connections to the current host and specified service that have an RST error	numeric

Annex D: Features Used for Training and Testing Models

Calculated AR of the NSL-KDD dataset

Rank	Feature	Attribute Ratio	Rank	Feature	Attribute Ratio
1	protocol_type_tcp	1000.0	22	srv_diff_host_rate	3.08
2	num_shells	326.1	23	flag_S0	2.965
3	urgent	173.039	24	wrong_fragment	2.74
4	num_file_creations	62.23	25	dst_host_srv_serror_rate	2.67
5	flag_SF	51.0	26	srv_serror_rate	2.64
6	num_failed_logins	46.038	27	serror_rate	2.63
7	hot	40.77	28	dst_host_serror_rate	2.62
8	logged_in	10.569	29	num_root	2.60
9	dst_bytes	9.15	30	count	2.11
10	src_bytes	8.46	31	service_telnet	1.88
11	duration	7.225	32	dst_host_srv_count	1.645
12	dst_host_srv_diff_host_rate	5.756	33	dst_host_same_srv_rate	1.55
13	dst_host_diff_srv_rate	4.837	34	service_ftp_data	1.54
14	num_access_files	4.69	35	same_srv_rate	1.59
15	dst_host_same_src_port_rate	4.39	36	dst_host_count	1.34
16	num_compromised	4.338	37	service_http	1.29
17	diff_srv_rate	4.069	38	srv_count	1.17
18	dst_host_srv_rerror_rate	3.6679	39	root_shell	1.0
19	srv_rerror_rate	3.6677	40	service_private	0.72
20	rerror_rate	3.645	41	protocol_type_icmp	0.54
21	dst_host_rerror_rate	3.279			

Declaration

I, the undersigned, declare that this research is my original work and has not been presented for degree in any other university, and that all sources of materials used for the research have been acknowledged.

Declared by:

Name: Tigist Hidru

Signature: _____

Date: _____

Confirmed by advisor:

Name: Solomon Gizaw (PhD)

Signature: _____

Date: _____