

A project report
on
**A Hybrid Approach for Intrusion Detection
System Using K-Means and Random Forest**

Submitted in partial fulfilment of the requirements for the award of the degree

Of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

BY

RAHIM BAIG

160317733046

Under the guidance of

Mr. MOHD MUNAWER

Assistant professor

Department of CSE, **DCET**



Department of computer science engineering

DECCAN COLLEGE OF ENGINEERING AND TECHNOLOGY

(Affiliated to Osmania University)

Hyderabad 2021

CERTIFICATE

This is to certify that the project report on “A Hybrid Approach for Intrusion Detection System Using K-Means and Random Forest” being submitted by

Mr. Rahim Baig

160317733046

in partial fulfilment for the award of the Degree of Bachelor of Engineering in Computer Science by the Osmania University is a record of Bonafide work carried out by them under my guidance and supervision.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Internal Guide

Mr. Mohd Munawer

Assistant Professor

Dept. of CSE, DCET.

Head of the Department

Dr. Syed Raziuddin

Dept. of CSE, DCET

External Examine

DECLARATION

This is to certify that the work reported in the present project entitled “A Hybrid Approach for Intrusion Detection System Using K-Means and Random Forest” is a record of work done by me in the Department of Computer Science & Engineering, Deccan College of Engineering and Technology, Osmania University. The reports are based on the project work done entirely by me and not copied from any other source.

RAHIM BAIG

(160317733046)

ACKNOWLEDGEMENT

I am thankful to PRINCIPAL **Mr. DR. M. A. MALIK** for providing excellent infrastructure and a nice atmosphere for completing this project successfully.

I am thankful to Head of the department Mr. **DR. SYED RAZIUDDIN** for providing the necessary facilities during the execution of our project work.

This project would not have been a success without my internal guide. So, I would extend my deep sense of gratitude to my internal guide **Mr. Mohd Munawer**, for the effort he took in guiding me in all the stages of completion of our project work. I also thank For his valuable suggestions, advice, guidance and constructive ideas in each and every step, which was indeed a great need towards the successful completion of the project.

I convey My heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, I would like to take this opportunity to thank My family for their support through the work. I sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work

ABSTRACT

In the modern computer world, use of the internet is increasing day by day. However, the increasing use of the internet creates some security issues. These days, such new type of security attacks occurs everyday and it is not easy to detect and prevent those attacks effectively. One common method of attack involves sending large amount of request to site or server and server will be unable to handle such huge requests and site will be offline for many day . This type of attack is called distributed denial of service (PCPV) attack, which act as a major security threat to internet services and most critical attack for cyber security world. Detection and prevention of Distributed Potential Corporate Privacy Violation (PCPV) becomes a crucial process for the commercial organizations that uses the internet. Different approaches have been adopted to process traffic information collected by a monitoring stations (Routers and Servers) to distinguish malicious traffic such as PCPV attack from normal traffic in Intrusion Detection Systems (IDS). In general, Machine learning techniques can be designed and implemented with the intrusion systems to protect the organizations from malicious traffic. Specifically, supervised clustering techniques allow to effectively distinguish the normal traffic from malicious traffic with good accuracy. In this paper, machine learning algorithms are used to detect PCPV attacks collected from “KDDcup 99 Dataset”.pre-processing and feature selection technique is used on the dataset to enhance the performance of the classifiers and reduce the detection time. The classification algorithms such as Random Forest and clustering algorithm Kmeans is applied on the training dataset and the implementation of the algorithm is done using Google Colab tool. The performance comparison of algorithms is shown using confusion matrix and it is found that Random Forest is more efficient in detection of PCPV attack .The proposed method can be used as PCPV defence system.

LIST OF TABLES

Seq No.	Name Of The Table	Page No
3.5.1	Performance of IDS with K-Means	13
3.5.2	Performance of IDS with K-Means	14
3.5.3	Performance of IDS with RF(Unsupervised)	15
3.5.4	Performance of IDS with RF	16
3.5.5	Performance of IDS with K-Means and RF (Hybrid IDS)	15

LIST OF FIGURES

Seq No.	Name Of The Figure	Page No.
1.1	Various categories of ML techniques	1
3.1	Existing system	7
3.2.1	Flow chart of K-Means algorithm	7
3.2.2	Random forest example	8
3.2.3	Functioning of RF classifier	9
3.3.1	Outline of IDS	10
3.3.2	IDS with RF classifier	11
3.3.3	Hybrid IDS with RF and K-Means clustering	11
3.4.1	Confusion matrix	12
3.5.1	Performance of IDS with K-Means (Unsupervised Learning)	13
3.5.2	Performance of IDS with K-Means (Unsupervised Learning)	14
3.5.3	Performance of IDS with RF (Supervised Learning)	15
3.5.4	Performance of IDS with RF (Supervised Learning)	16
3.5.5	Performance of Hybrid IDS with K-Means and RF (Supervised Learning)	17
4.2.1	SDLC life cycle	19
5.3.1	system Architecture	23
6.2.1	Use Case Diagram	25

6.3.1	Class Diagram	26
6.4.1	Activity Diagram	27
6.5.1	Sequence Diagram	28
6.6.1	ER Diagram	29
6.7.1	DFD Diagram	30
7.1.1	Random Forest Example	32
7.1.4	Difference of Kmeans Plotting Before and After	34
7.1.4.1	Number of Clusters	35
7.1.4.2	Centroids	35
7.1.4.3	Assignment of Clusters	36
7.1.4.4	Final Clustered Output	36
8.2.1	Result Shows the Misc Activity	63
8.2.2	Result Without Misc Activity	64
8.2.3	Elbow Plotting Diagram	64
8.2.4	Features in the Dataset	65
8.2.5	Different Kinds Of Attacks and There Counts	65
8.2.6	Correlation	66
8.2.7	HeatMap	67
8.2.8	Predicted Values Of First 10 Rows	67
8.2.9	Confusion Matrix	68
8.2.10	Scores of Precision, F1 Score, Recall, Support	69

LIST OF ABBREVIATIONS

KDD	Knowledge discovery in Databases
Dos	Denial Of Service
R2L	Local Remote Attack
RF	Random Forest
TP/TN	True Positive/True Negative
SDLC	Software Development Life Cycle
API	Application Programing Interface
NIDS	Network Intrusion Detection System

TABLE OF CONTENT

CONTENT	PAGE NO.
DECLARATION	I
ACKNOWLEDGEMENT	II
ABSTRACT	III
LIST OF TABLE	IV
LIST OF FIGURES	V
LIST OF ABBREVIATIONS	VI

CHAPTER 1 INTRODUCTION

1.1	Introduction	1
1.2	Problem statements	2
1.3	Objective	3
1.4	Scope	3

CHAPTER 2 LITERATURE SURVEY

2.1	literature survey	4
2.2	KDD CUP 99 Dataset	4

CHAPTER 3 PROPOSED SYSTEM

3.1	Proposed System	6
3.2	Potential Models Used	7
3.3	Methodology For IDS	10
3.4	Confusion Matrix	12
3.5	Experimental Setup	13

CHAPTER 4 SYSTEM ANALYSIS

4.1	Introduction	18
-----	--------------	----

	4.2	Software development cycle	18
CHAPTER 5		SYSTEM SPECIFICATION	
	5.1	Software Requirements	22
	5.2	Hardware Requirements	22
	5.3	System Architecture	22
CHAPTER 6		SYSTEM DESIGN	
	6.1	About UML Diagrams	24
	6.2	Use case Diagram	25
	6.3	Class Diagram	26
	6.4	Activity Diagram	27
	6.5	Sequence Diagram	28
	6.6	ER Diagram	29
	6.7	Data Flow Diagram	30
CHAPTER 7		IMPLIMENTATION	
	7.1	Algorithm	31
	7.1.1	Random Forest	31
	7.1.2	Simple Decision Tree	32
	7.1.3	Random Forest Classifier	33
	7.1.4	Kmeans Clustering	33
	7.2	Sample Code	38
CHAPTER 8		TESTING	
	8.1	Testing Methodology	60
	8.2	Output Screens	63
	8.2.1	Misc Activity in Kmeans	63

	8.2.2	Without Misc Activity	63
	8.2.3	Elbow Plotting	64
	8.2.4	Feature Selection	65
	8.2.5	Count Of Different Attacks	65
	8.2.6	Correlation	66
	8.2.7	HeatMap	66
	8.2.8	Predicted Values	67
	8.2.9	Confusion Matrix	68
CHAPTER 9		CONCLUSION AND FUTURE WORK	70
CHAPTER 10		REFERENCES	71

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Machine learning is widely used for discovering business intelligence from data of different domains. In fact, organizations solve many problems with machine learning techniques that are part of AI. There are different kinds of techniques as shown in Figure 1. Broadly, they can be classified into supervised learning (training is needed), unsupervised learning (no explicit training), semi-supervised learning (have qualities of supervised and non-supervised) and reinforcement learning. In this project supervised learning and unsupervised learning models are used to realize a hybrid IDS. The IDS is made up of K-Means clustering and Random Forest (RF) classification algorithm. In most of the applications, classification accuracy can be improved with certain pre-processing techniques and clustering as well.

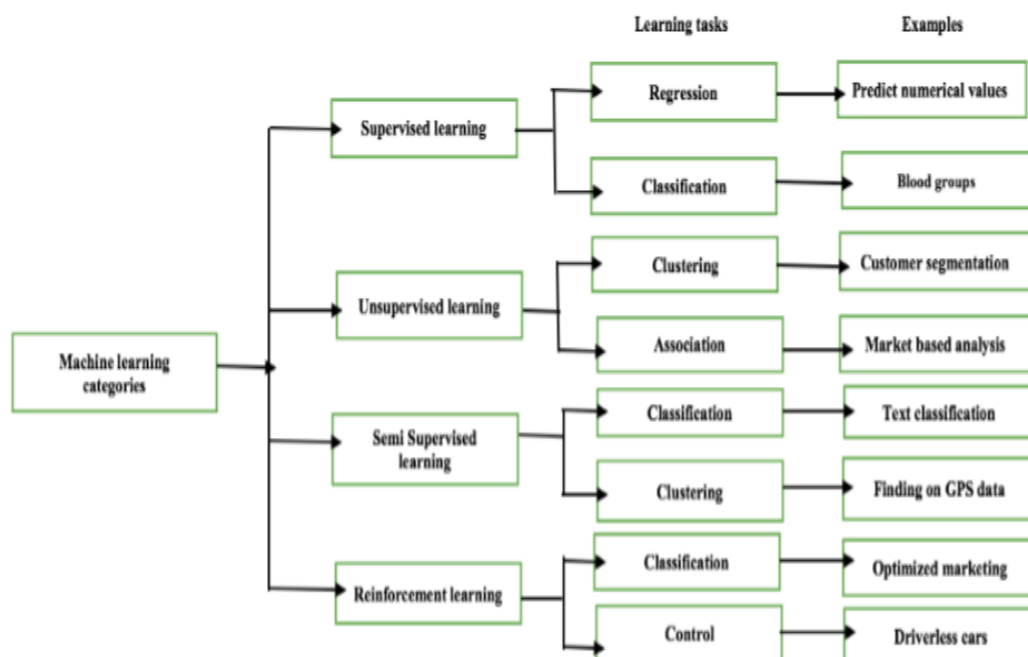


Figure 1.1: Various categories of ML techniques

As presented in Figure 1, there are many examples of applications for which different learning models are used. In this project K-Means clustering, RF classification and attribute ratio based feature selection are used. A hybrid IDE is realized using

aforementioned techniques. The contributions in this project include the usage of K-Means and RF for making an IDS that is supported by attribute ratio which is a good feature selection method. If feature selection is not used, the IDS may show poor performance. Our contributions are as follows.

1. An IDS is built with K-Means clustering with attribute ratio based feature selection.
2. An IDS is built with RF classification with attribute ratio based feature selection.
3. A Hybrid IDS is realized by combining K-Means and RF for better results with different datasets.

The remainder of the report is structured as follows. Section 2 reviews literature. Section 3 provides details of the K-Means and RF. Section 4 presents the proposed system. Section 5 provides performance metrics. Section 6 covers experimental setup and dataset details. Section 7 presents experimental results. Section 8 concludes the paper along with directions for future work.

1.2 Problem Statement

The existing system is able to detect the problem but it will tell them once the attack is happened already .Therefore, **A Hybrid Approach for Intrusion Detection System Using K-Means and Random Forest** will let you know the intrusion is happening at the system right when it happens and we can be saved from the intrusion happening into our systems

1.3 OBJECTIVE

The objective of the project is to design a web-application which can accurately classify intrusion. The system will take URL of website as input and then process it using Clustering and by using machine learning techniques, accurately classify INTURSION. Obviously, it is difficult to achieve 100% accuracy while classifying Intrusion, however our aim is to get an accuracy which is as close as possible to the ideal value (i.e.100%) The system should also be able to process the input and classify Intrusion as quickly as possible. However, the speed of processing could sometimes depend upon the processing capability of the machine the web-application is being run on. The project

should be simple and elegant with a user-friendly interface. Users should be able to access the web-application to identify Intrusion easily.

1.4 SCOPE

The scope of our project is to detect INTRUSION from online web servers using machine learning. Our intrusion detection makes use of ML techniques to detect intrusion in websites. Our project has major impact on small scale websites because they cannot afford to pay the high subscriptions to the web servers .

CHAPTER 2

LITERATURE SURVEY

2.1 Literature Survey

Different algorithms in ML are used for security analysis. Many IDS strived to reduce false alarm rate as explored in [1]. In [2] an IDS is realized with clustering technique and also a classifier known as KNN. K-Means is used for intrusion detection in [3] which is widely used as a clustering technique in the data mining domain. The combination of K-Means and SVM are used in [4] for developing an IDS with extreme learning procedure. RBF kernel function and K-Means are used in [5] for developing an IDS with feature selection as well. The C4.5 and K-Means algorithms are studied for IDS in [6] while a Network IDS (NIDS) is developed in [7] based on weighted k-Means and RF.

Apache Spark is used in [8] for an effective IDS towards NIDS for cyber security. There are many classification techniques used for IDS as reviewed in [9]. A supervised tree based mechanism is employed in [10] while RF is used for NIDS in [11]. Botnet based attacks are explored and an IDS is made to detect such attacks using RF in [12]. AdaBoost and Artificial Bee Colony combination is employed in [13] for NIDS. Various data mining approaches are used in [14] for realizing an IDS. A hybrid mode for IDS with feature selection is considered in [15] while the combination of SVM and GA (Genetic Algorithm) are used for IDS in [16]. Intrusion detection for Wireless Sensor Network (WSN) [17], the combination of anomaly detection and misuse detection are explored in [18] while a hybrid IDS for power system is made in [19]. From the literature, it is understood that there is need for hybrid IDS with feature selection for higher level of accuracy and least false alarm rate.

2.2 KDD CUP 99 DATASET

In 1998, DARPA in concert with Lincoln Laboratory at MIT launched the DARPA 1998 dataset for evaluating IDS [36]. The DARPA 1998 dataset contains seven weeks of training and also two weeks of testing data. In total, there are 38 attacks in training data as well as in testing data. The refined version of DARPA dataset which contains only network data (i.e. Tcpdump data) is termed as KDD dataset [37]. The Third

International Knowledge Discovery and Data Mining Tools Competition were held in colligation with KDD-99, the Fifth International Conference on Knowledge Discovery and Data Mining. KDD dataset is a dataset employed for this Third International Knowledge Discovery and Data Mining Tools Competition. KDD training dataset consists of relatively 4,900,000 single connection vectors where each single connection vectors consists of 41 features and is marked as either normal or an attack, with exactly one particular attack type [38]. These features had all forms of continuous and symbolic with extensively varying ranges falling in four categories:

- In a connection, the first category consists of the intrinsic features which comprises of the fundamental features of each individual TCP connections. Some of the features for each individual TCP connections are duration of the connection, the type of the protocol (TCP, UDP, etc.) and network service (http, telnet, etc.).
- The content features suggested by domain knowledge are used to assess the payload of the original TCP packets, such as the number of failed login attempts.
- Within a connection, the same host features observe the recognized connections that have the same destination host as present connection in past two seconds and the statistics related to the protocol behavior, service, etc are estimated.
- The similar same service features scrutinize the connections that have the same service as the current connection in past two seconds.

A variety of attacks incorporated in the dataset fall into following four major categories:

- Potential Corporate Privacy Violations:** A Potential Corporate Privacy Violation is an attack where the attacker constructs some computing or memory resource fully occupied or unavailable to manage legitimate requirements, or reject legitimate users right to use a machine.
- User to Root Attacks:** User to Root exploits are a category of exploits where the attacker initiate by accessing a normal user account on the system (possibly achieved by tracking down the passwords, a dictionary attack, or social engineering) and take advantage of some susceptibility to achieve root access to the system.
- Remote to User Attacks:** A Remote to User attack takes place when an attacker who has the capability to send packets to a machine over a network but does not have an account on that machine, makes use of some vulnerability to achieve local access as a user of that machine.
- Probes:** Probing is a category of attacks where an attacker examines a network to collect information or discover well-known vulnerabilities.

CHAPTER 3

PROPOSED SYSTEM

3.1 Proposed System

The workflow of the proposed method is setup , starting with data collection (KDD-99 Dataset), Pre-Processing: Training and testing dataset, building model and result analysis

KDD cup 1999 dataset Collection In 1998, the DARPA Intrusion Detection Assessment Program was prepared and managed by MIT Lincoln Labs. Its purpose was to study and evaluate intrusion detection research. Standard data sets include various simulation intrusions in military network environments. The connection to the dataset includes a sequence of TCP packets beginning and ending at a well-defined time between the source IP address and the destination IP address using a well-defined protocol. Each connection is categorized as a normal or specific type of attack. Data sets are categorized into five sub-sets: denial-of-service attacks, local or remote network attacks, user / root attacks, sample attacks, and generic data. Each record is classified as normal or attack with exactly one type of attack. They are categorized as follows:

- Denial of service (DoS) Denial of Service (DOS) allows a legitimate user to gain access to the machine by creating too much or too much computer resources or memory for an attacker to handle legitimate requests.
- R2L (Local Remote Attack (User)) Local Remote Attack (R2L) is a type of attack in which an attacker can send packets to a computer over the network and then exploit a vulnerability in the computer to illegally attack local access. On the machine.
- Root User Attack (U2R) Root User Attack (U2R) is the attack class that an attacker first accesses a regular user account on a system. The vulnerability could be exploited to gain root access to the system.
- Monitoring (monitoring and other discovery) detection is an attack type in which an attacker scans the network for known information or vulnerabilities. An

attacker with a map of systems and services available on the network will use the information to detect attacks.

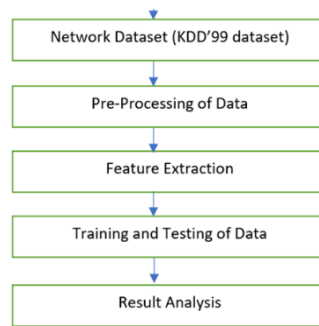


Fig 3.1: Existing system

3.2 POTENTIAL MODELS USED

This section provides popular machine learning techniques used for realization of IDS in this project. They are known as K-Means clustering and Random Forest.

3.2.1 K-Means

K-Means is one of the clustering algorithms that is widely used to solve real time problems. It is explored in [1], [2], [3], [4], [5] and [6] for different purposes. It can be used along with supervised learning method as pre-processing technique as well.

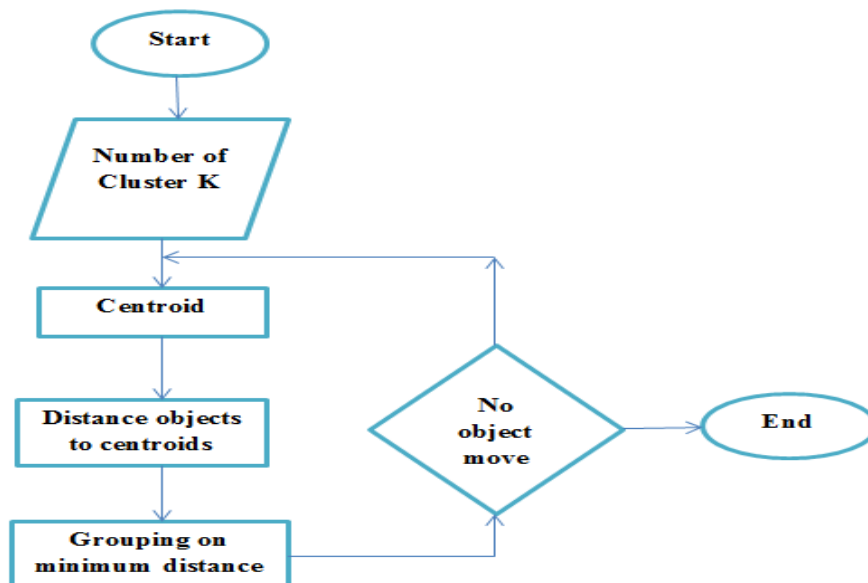


Figure 3.2.1: Flow chart of K-Means algorithm

As presented in Figure 2, the K-Means algorithm takes a dataset as input and divides the objects into different clusters. Each cluster will have objects that are similar to each other. Based on the number clusters (K), it creates initial clusters and then as the iterative process is going on, the cluster centroids are updated and finally it results in grouping of objects into k number of clusters.

3.2.2 Random Forest

It is the widely used supervised learning method. It needs training data in order to perform its detection process. In this project, it is used to detect intrusions. Based on the knowledge of training it can label unlabelled instances. It makes use of number of decision trees. Each tree is known as a learner. When multiple trees are combined they become a random forest.

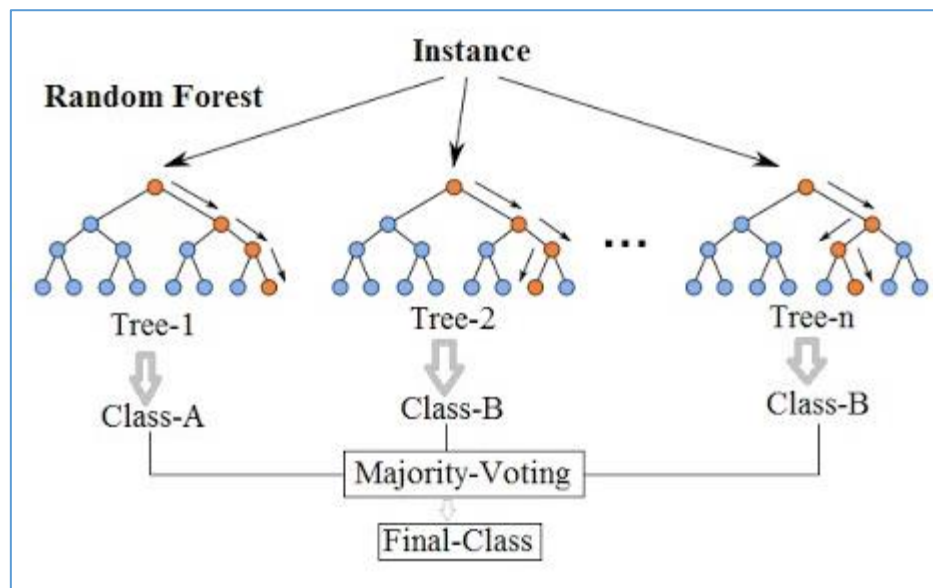


Figure 3.2.2: Random forest example

As shown in Figure 3.2.2, there are many decision trees generated. Each tree is related to a class. There will be selection of final class based on the concept of majority voting.

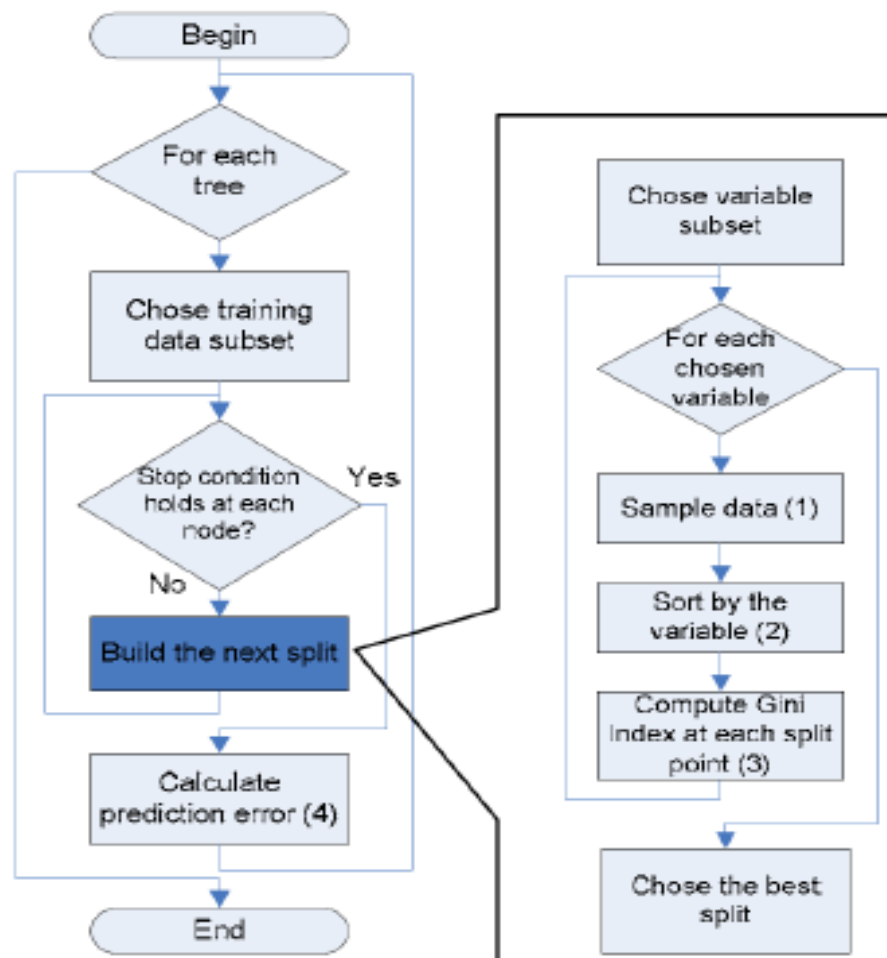


Figure 3.2.3: Functioning of RF classifier

As presented in Figure 4 training data is chosen for each tree. There is stop condition that is verified every time. After computation of the prediction error, the iterative process ends. When stop condition fails, it moves on to the next split followed by many actions in an iterative process. RF provides many benefits and its accuracy is higher for many real world datasets. It can also work with large volumes of data. It also can handle thousands of dimensions in the data besides having ability to deal with missing data. It can reuse generated forests and balance errors besides locating outliers and supporting discovery of hidden interactions. However, it has some issues like overfitting in case of certain datasets and it also shows bias towards dimensions that exhibit more levels.

3.3 METHODOLOGY FOR INTRUSION DETECTION SYSTEM

The methodology used for building an intrusion detection system based on machine learning algorithms is described here. Figure 5 shows outline of the IDS. It takes data as input and performs intrusion detection process. It can distinguish between the malicious and benign instances. The dataset used for the experiments is NSL-KDD taken from [20].

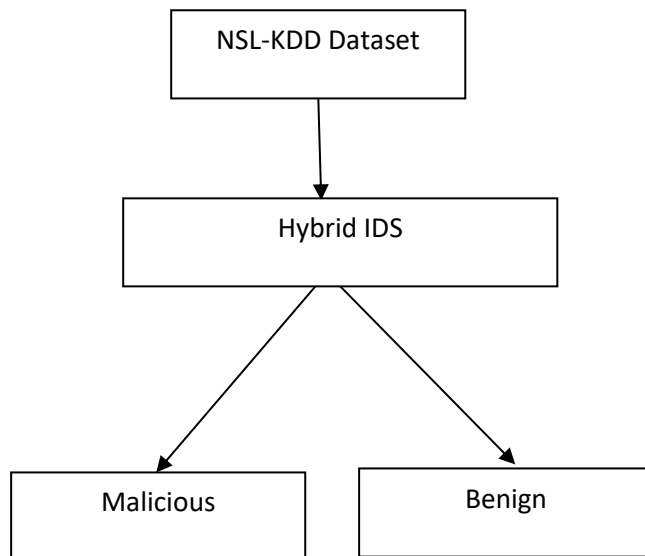


Figure 3.3.1: Outline of IDS

The intrusion detection system is provided with more details in Figure 3.3.2. It consists of many functions like pre-processing of data to have training and testing sets, feature extraction, feature selection and intrusion classification. Feature extraction is the process of obtaining all the features from the NSL-KDD dataset. Once all the features are extracted, there is need for feature selection. Feature selection is the process in which some features that have higher weight in contributing towards detecting class labels. There are many feature selection methods such as filter and wrapper methods. One filter method is employed in this project. It is known as attribute ratio method. According to this method the extracted features are subjected to an iterative process in which each attribute is verified to know whether it is having any contribution towards detecting class label. Once the features are identified, the selected features are provided to the classifier for training. In other words, the training data with chosen features are provided to classifier. Here the classifier used is RF algorithm. It takes the selected features as part of training set and learns from the data. As the training data contains

class labels, the algorithm learns from it and makes a model. This model is known as knowledge model that is used for prediction of class labels for unlabelled data. Once training is completed, in the testing phase, the RF classifier will be able to show the class labels for unlabelled instances. Thus the proposed IDS is able to identify intrusions.

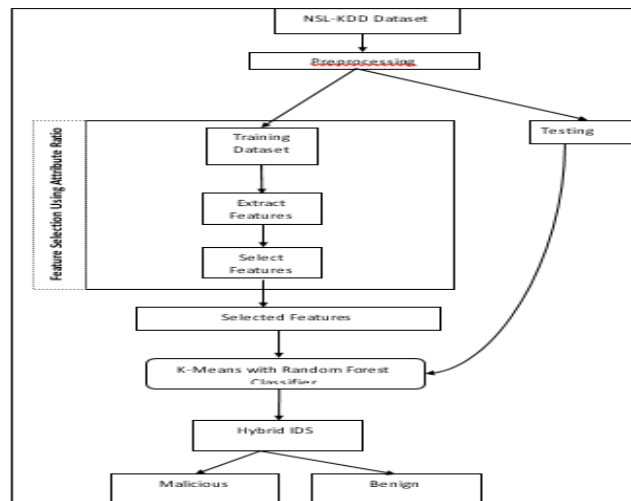


Figure 3.3.2: IDS with RF classifier

RF classifier is used as part of the proposed IDS. However, it is understood from the literature that when clustering is preceded by the classification, it will be able to improve performance as it becomes hybrid IDS. This is illustrated in Figure 3.3.3.

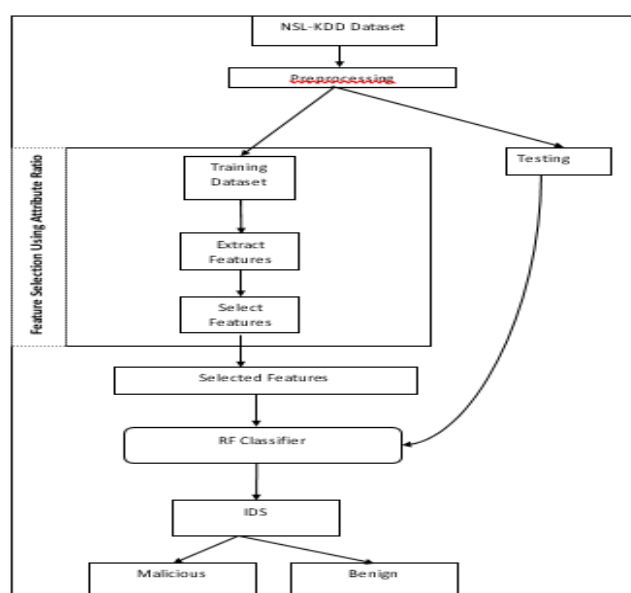


Figure 3.3.3: Hybrid IDS with RF and K-Means clustering

The approach of the hybrid IDS is similar to that of the one illustrated in Figure 3.3.3. The difference is that the hybrid IDS is made up of both clustering and classification techniques. The clustering is made with K-Means while the classification is made with RF. However, the feature selection and other functions remain the same.

3.4. CONFUSION METRIX

In this project, many performance metrics are used for evaluating the IDS developed using machine learning algorithms. Confusion matrix is the basis for the metrics.

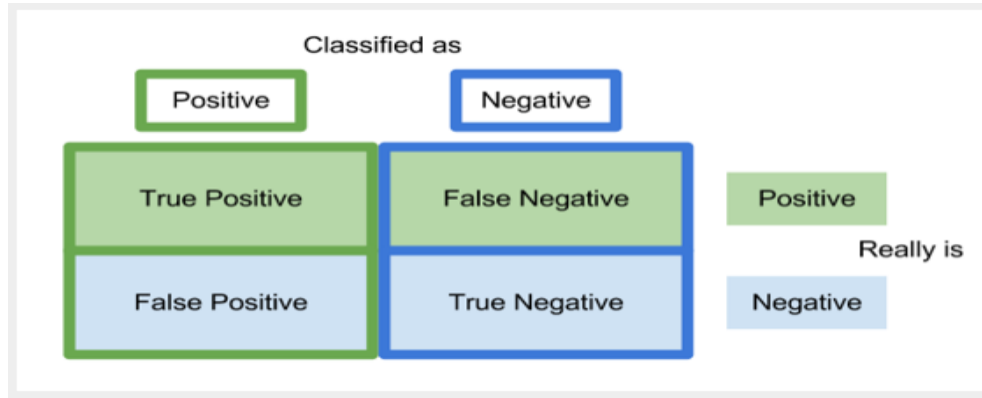


Figure 3.4.1: Confusion matrix

As presented in Figure 3.4.1, based on the correct and wrong predictions of intrusions, the initial measures like True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN) are used for deriving other metrics like precision, recall, F1 Score and accuracy as provided in Eq. 1, Eq. 2, Eq. 3 and Eq. 4.

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3)$$

$$F1 \text{ score} = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \quad (4)$$

F1-Score is the measure which reflects the accuracy of the IDS. This measure is the harmonic mean of two measures like precision and recall.

3.5 EXPERIMENTAL SETUP

The environment used for coding is Anaconda which is Python based data science platform with many IDEs available. The language used is Python, machine learning toolkit used in Scikit-learn and the distributed programming framework used is known as PySpark. Dataset used in this project is NSL-KDD which is acquired from [20].

3.5.1 EXPERIMENTAL RESULTS

This section provides experimental results when IDS is used with K-Means and RF individually and also when they are combined with a hybrid IDS.

Probability Threshold	Precision	Recall	F1-Score	Accuracy
0.05	1	0.99	0.99	0.99
0.005	0.91	0.86	0.89	0.9

Table 3.5.1: Shows the performance of IDS with K-Means

As presented in Table 1, the performance of IDS with K-Means is provided against different probability thresholds like 0.05 and 0.005.

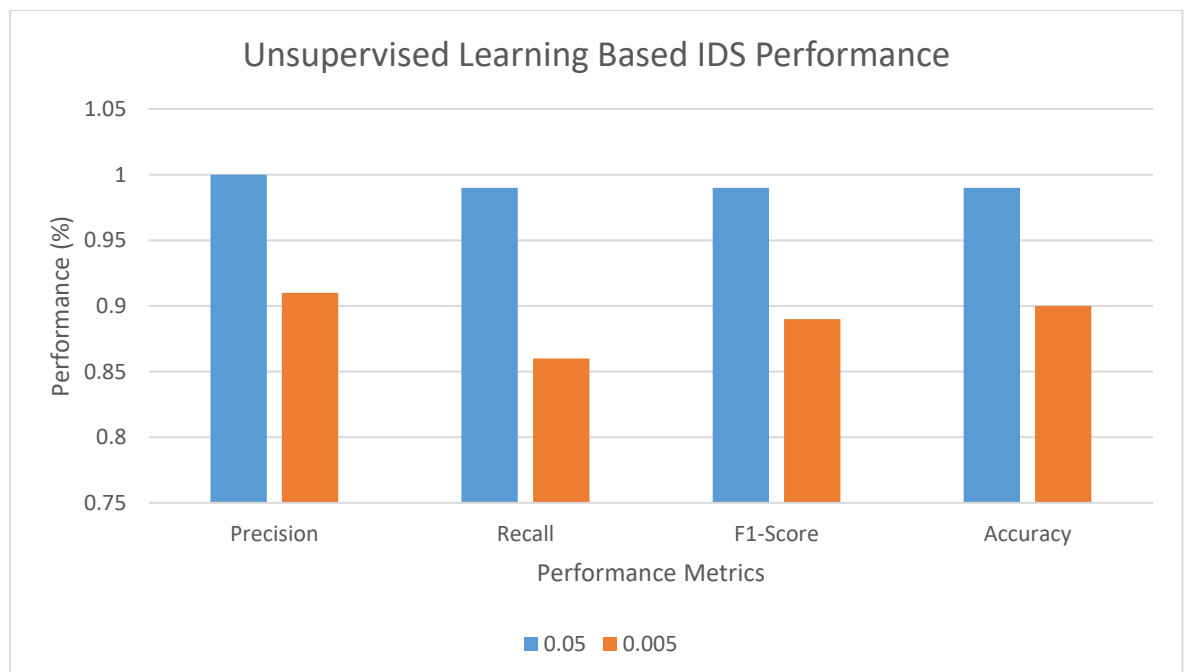


Figure 3.5.1: Performance of IDS with K-Means (Unsupervised Learning)

As presented in Figure 3.5.1, the performance metrics like accuracy, F1-score, recall and precision are provided in horizontal axis. The performance of the IDS when probability is set at 0.05 and 0.005 is presented in vertical axis. The performance of the IDS differs when probability threshold is changed. More probability threshold has higher performance when compared with the low probability threshold.

Probability Threshold	Detection rate	False alarm rate
0.05	0.997	0.149
0.005	0.91	0.86

Table 3.5.2: Shows the performance of IDS with K-Means

As presented in Table 2, the performance of IDS with K-Means is provided against different probability thresholds like 0.05 and 0.005.

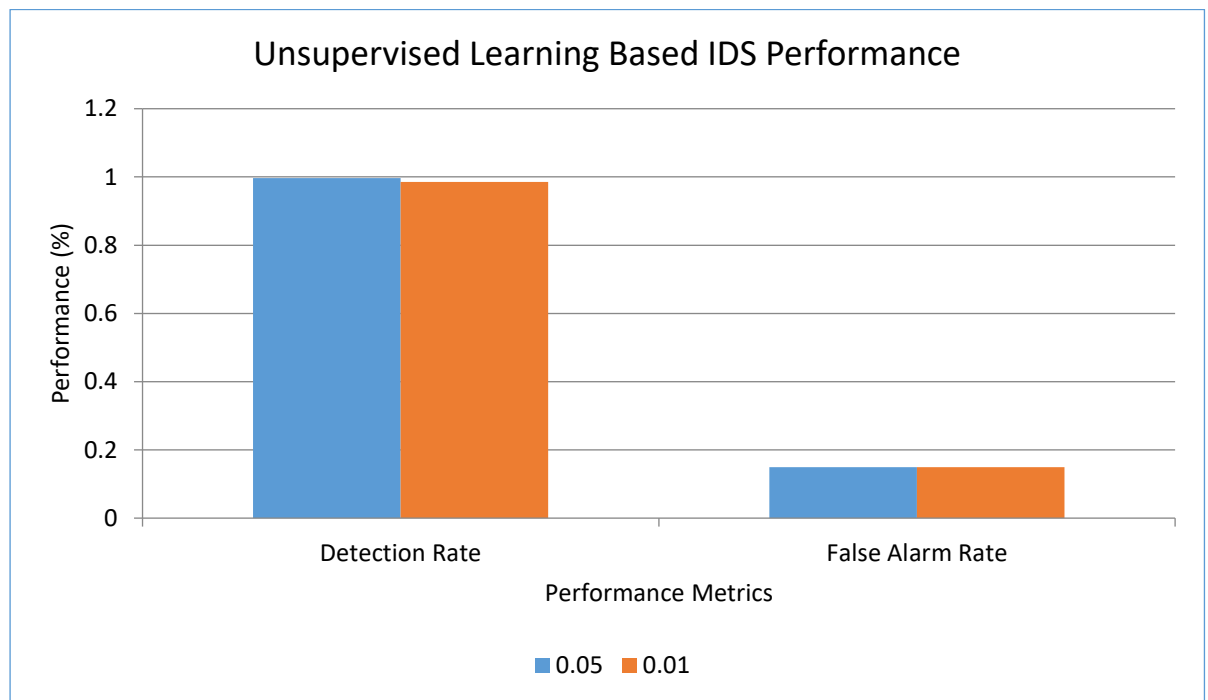


Figure 3.5.2: Performance of IDS with K-Means (Unsupervised Learning)

As presented in Figure 3.5.2, the performance metrics like detection rate and false positive rate are provided in horizontal axis. The performance of the IDS when probability is set at 0.05 and 0.01 is presented in vertical axis. The performance of the IDS differs when probability threshold is changed. However, less different is found in performance when there is less change in the probability threshold.

Probability Threshold	Precision	Recall	F1-Score	Accuracy
0.05	1	0.99	0.99	0.99
0.005	1	0.98	0.99	0.987

Table 3.5.3: Shows the performance of IDS with RF

As presented in Table 3, the performance of IDS with RF is provided against different probability thresholds like 0.05 and 0.005.

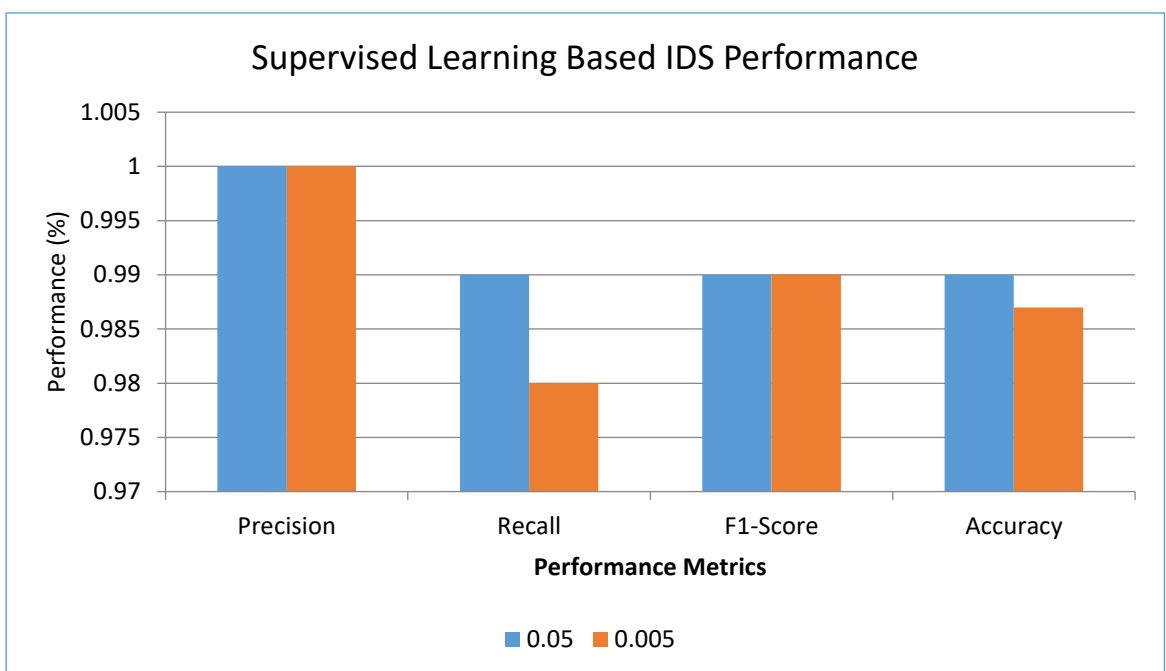


Figure 3.5.3: Performance of IDS with RF (Supervised Learning)

As presented in Figure 3.5.3, the performance metrics like accuracy, F1-score, recall and precision are provided in horizontal axis. The performance of the IDS when probability is set at 0.05 and 0.005 is presented in vertical axis. The performance of the IDS differs when probability threshold is changed.

Probability Threshold	detection rate	False alarm rate
0.05	0.999	0.009
0.005	0.9353	0.1384

Table 3.5.4: Shows the performance of IDS with RF

As presented in Table 4, the performance of IDS with RF is provided against different probability thresholds like 0.05 and 0.005.

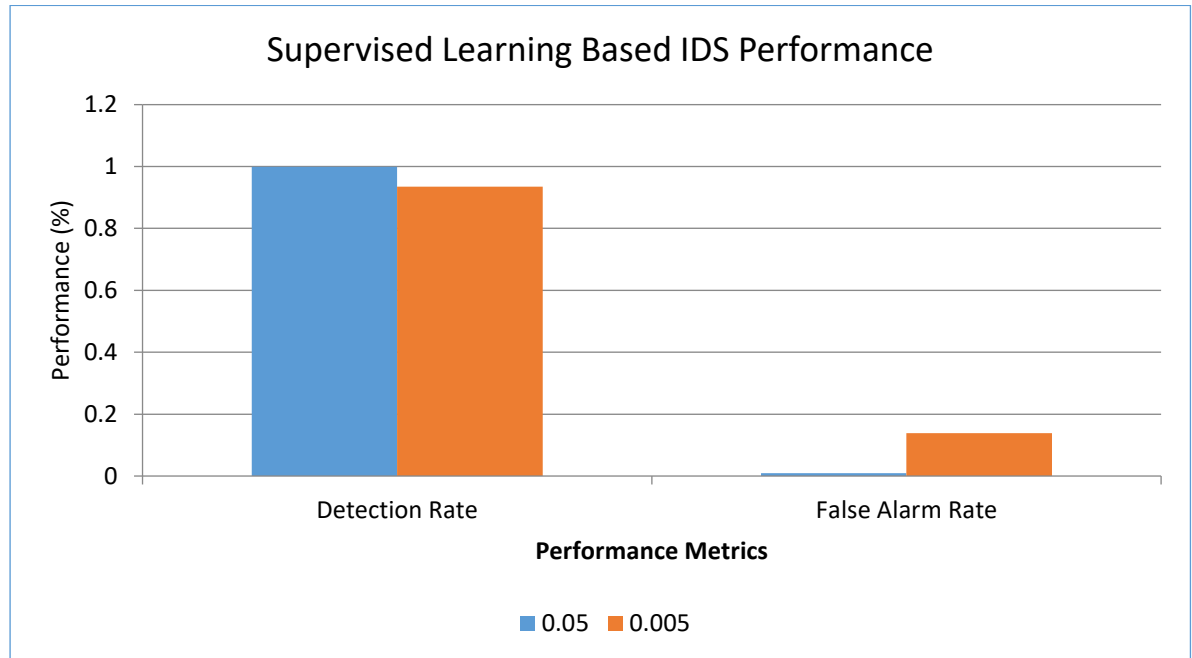


Figure 3.5.4: Performance of IDS with RF (Supervised Learning)

As presented in Figure 3.5.4, the performance metrics like detection rate and false positive rate are provided in horizontal axis. The performance of the IDS when probability is set at 0.05 and 0.005 is presented in vertical axis. The performance of the IDS differs when probability threshold is changed. However, less different is found in performance when there is less change in the probability threshold.

IDS	Detection Rate	False Alarm Rate	F1 Score
Hybrid IDS	0.99	0.14	0.94

Table 3.5.5: Shows the performance of IDS with K-Means and RF (Hybrid IDS)

As presented in Table 5, the performance of IDS with K-Means and RF is provided. The performance is shown in terms of detection rate, false alarm rate and F1-score.

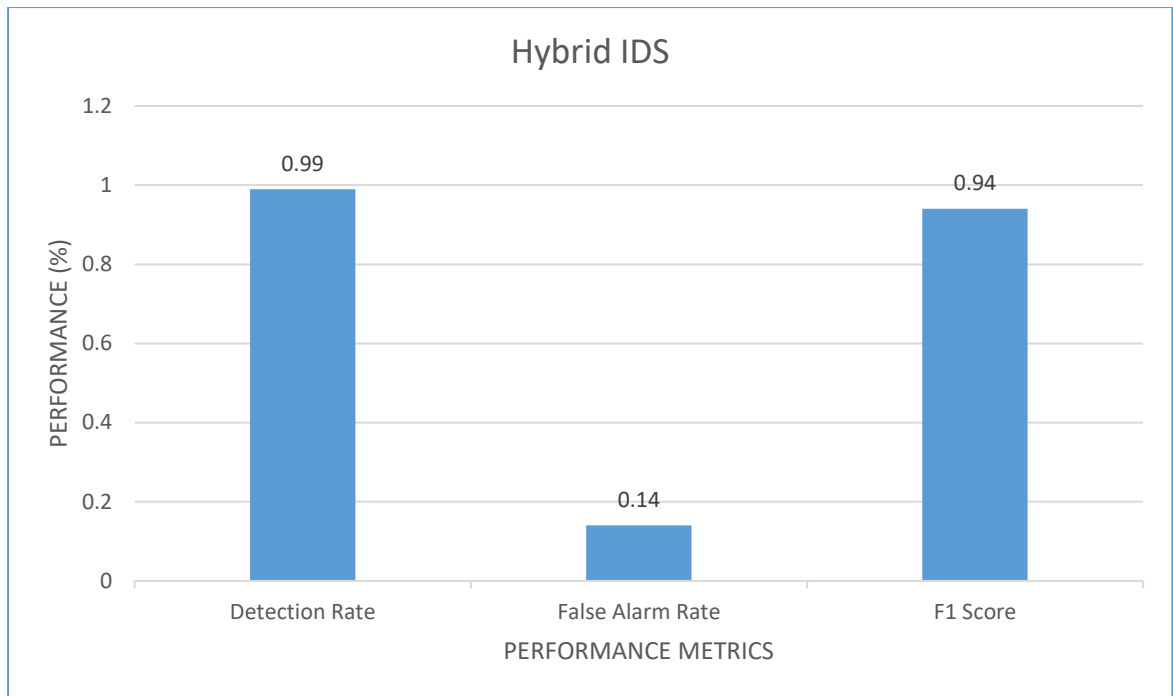


Figure 3.5.5: Performance of Hybrid IDS with K-Means and RF (Supervised Learning)

As presented in Figure 3.5.5, the performance metrics like detection rate, F1-score and false positive rate are provided in horizontal axis. The performance of the IDS is presented in vertical axis. The hybrid IDS showed 99% detection rate and 14% false alarm rate besides 94% F1-score.

Chapter 4

System Analysis

4.1 Introduction

System analysis is the process of gathering and interpreting facts, diagnosing problems and using the information to recommend improvements on the system. System analysis is a problem solving activity that requires intensive communication between the system users and system developers. System analysis or study is an important phase of any system development process. The system is studied to the minutest detail and analyzed. The system analyst plays the role of an interrogator and dwells deep into the working of the present system. The system is viewed as a whole and the inputs to the system are identified. The outputs from the organization are traced through the various processing that the inputs phase through in the organization. A detailed study of these processes must be made by various techniques like Interviews, Questionnaires etc. The data collected by these sources must be scrutinized to arrive to a conclusion. The conclusion is an understanding of how the system functions. This system is called the existing system. Now, the existing system is subjected to close study and the problem areas are identified. The designer now functions as a problem solver and tries to sort out the difficulties that the enterprise faces. The solutions are given as a proposal. The proposal is then weighed with the existing system analytically and the best one is selected. The proposal is presented to the user for an endorsement by the user. The proposal is reviewed on user request and suitable changes are made. This loop ends as soon as the user is satisfied with the proposal.

4.2 Software Development Life Cycle

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality soft wares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates. The following figure is a graphical representation of the various stages of a typical SDLC.

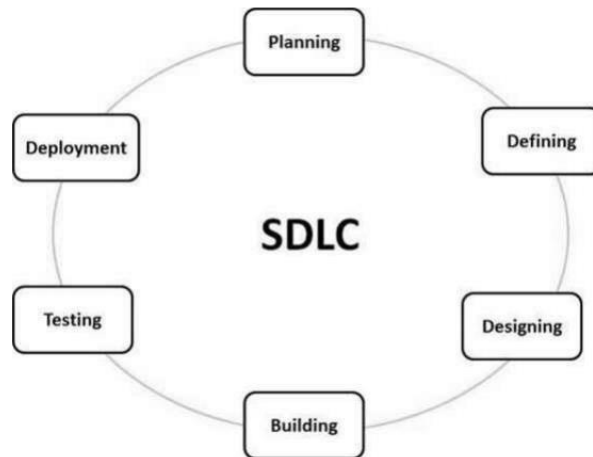


Figure 4.2.1:SDLC life cycle

4.2.1 Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas. Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.¹⁵

4.2.2 Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle.

4.2.3 Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually

more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification. This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product. A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

4.2.4 Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle. Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.¹⁶

4.2.5 Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

4.2.6 Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing). Then based on the feedback, the product may be released as it is or with suggested

enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base

Chapter 5

System Specification

5.1 Software Requirements

The software requirements for the development for this project are: -

1. Python (Minimum version 3.7.x)
2. Anaconda Package/Google Colab
3. Google Chrome
4. Operating systems: Windows or later, macOS, and Linux

5.2 Hardware Requirements

The minimum hardware requirements for the development of this project are as follows,

1. Physical server or virtual machine.
2. CPU: 2 x 64-bit, 2.8 GHz, 8.00 GT/s CPUs or better.
3. Memory: minimum RAM size of 32 GB, or 16 GB RAM with
4. 1600 MHz DDR3 installed, for a typical installation with 50 regular users
5. Storage: Recommended minimum of 100 GB, or 300 GB
6. Internet access to download the files

5.3 System architecture

Regardless of the type of IDS there are a few common components that typically constitute IDS:

- Traffic Collector - The component is responsible for gathering activity and event data for analysis. On a host-based ID this will typically include metrics such as inbound and outbound traffic and log and audit file activity recorded by the operating system. A NIDS will analyze the segment of the network by pulling the data off.
- Analysis Engine - The analysis engine analyzes the data that the traffic collector gathers. In case of a knowledge-based IDS comparison of data is done with a

signature database. A behaviourbased IDS, compares it against normal behaviour information gathered over time to see if the current behaviour deviates from the norm.

- Signature Database - Used in knowledge-based systems, the signature database is an amalgamation of signatures known to be associated with suspicious and malicious activities. A knowledge based IDS is only as good as its database.
- Management and Reporting Interface - A management interface providing a mechanism by which system administrators may manage the system and receive alerts when intrusions are detected

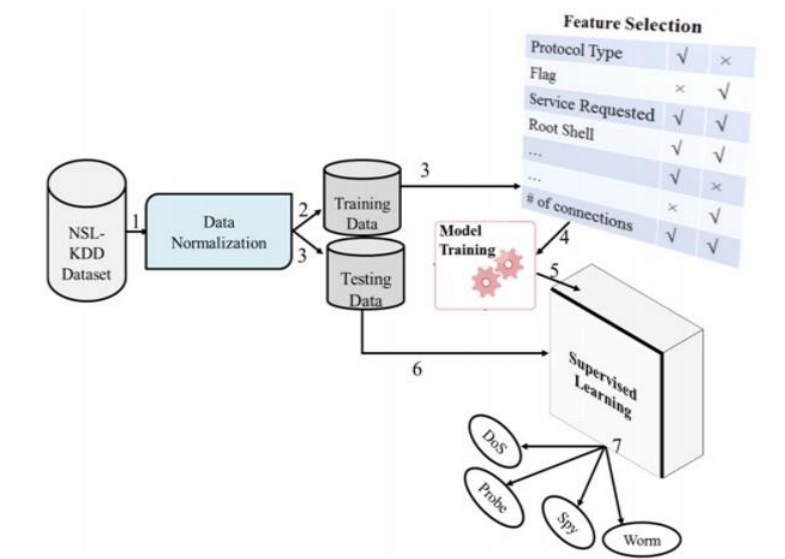


Figure 5.3.1: system Architecture

Chapter 6

System Design

6.1 About UML Diagrams

UML, short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. It is analogous to the blueprints used in other fields, and consists of different types of diagrams. In the aggregate, UML diagrams describe the boundary, structure, and the behavior of the system and the objects within it. UML is not a programming language but there are tools that can be used to generate code in various languages using UML diagrams. UML has a direct relation with object-oriented analysis and design.

6.2 Use Case Diagram

Use Case diagrams are used to analyze the system's high-level requirements. Use case diagrams give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact. A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.²⁰

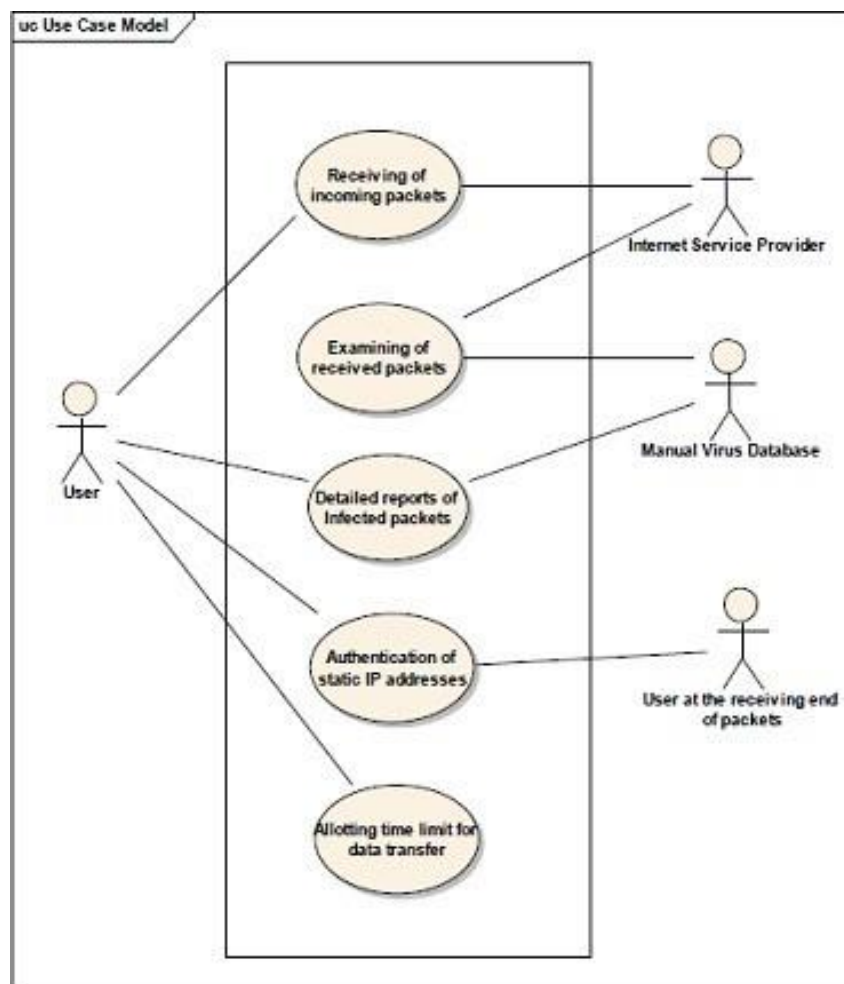


Fig 6.2.1: Use Case Diagram

6.3 Class Diagram

Class diagrams are the main building block of any object-oriented solution. It shows the classes in a system, attributes, and operations of each class and the relationship between each class. In most modelling tools, a class has three parts. Name at the top, attributes in the middle and operations or methods at the bottom. In a large system with many related classes, classes are grouped together to create class diagrams.

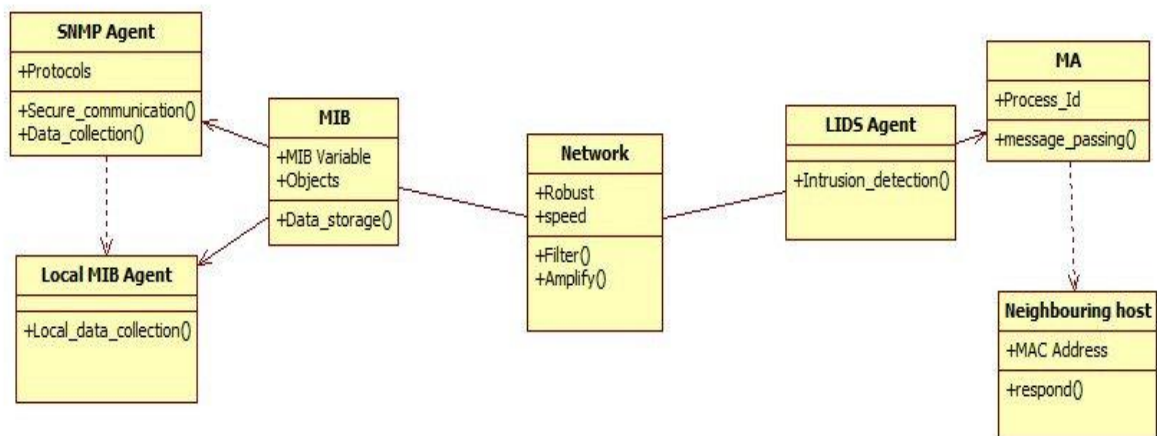


Fig 6.3.1: Class Diagram

6.4 Activity Diagram

Activity diagrams represent workflows in a graphical way. They can be used to describe the business workflow or the operational workflow of any component in a system. Sometimes activity diagrams are used as an alternative to State machine diagrams.

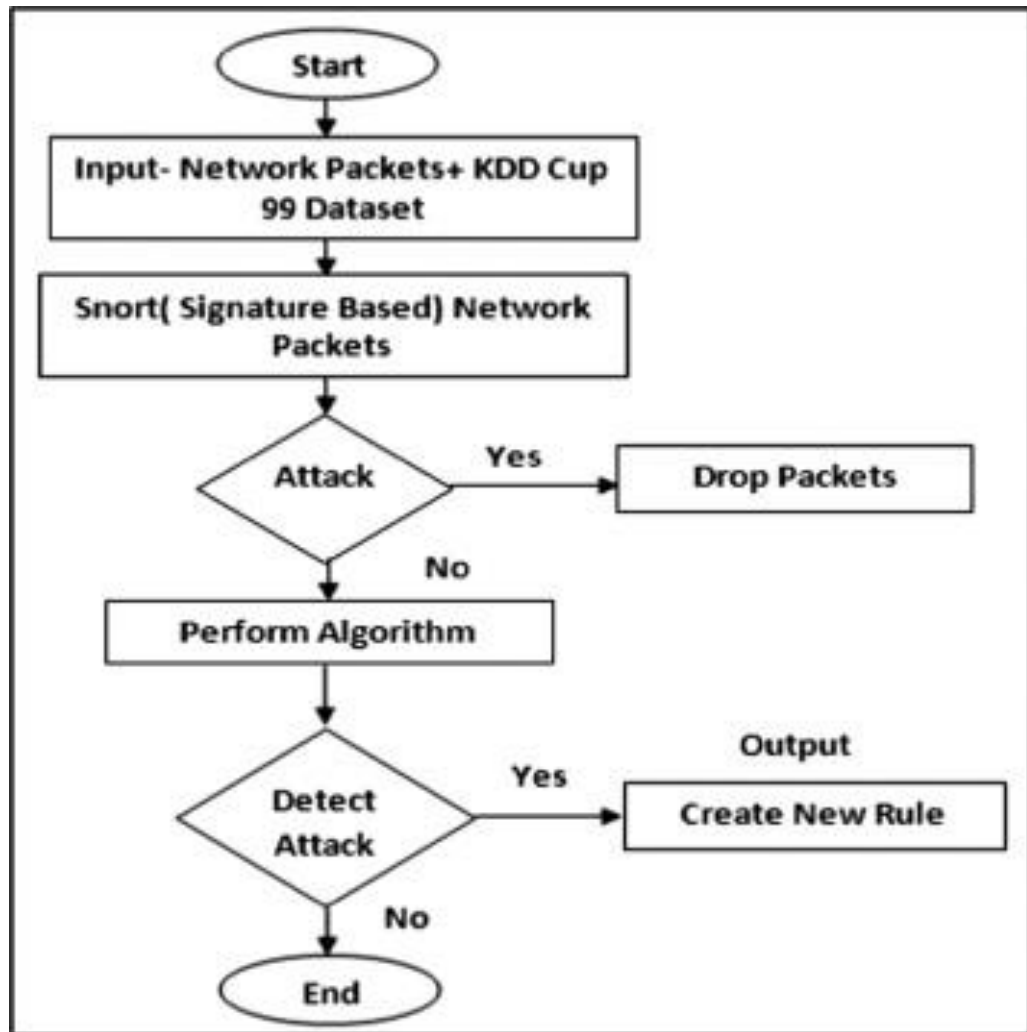


Figure 6.4.1: Activity Diagram

6.5 Sequence Diagram

Sequence diagrams are probably the most important UML diagrams among not only the computer science community but also as design-level models for business application²² development. Lately, they have become popular in depicting business processes, because of their visually self-explanatory nature. As the name suggests, sequence diagrams describe the sequence of messages and interactions that happen between actors and objects. Actors or objects can be active only when needed or when another object wants to communicate with them. All communication is represented in a chronological manner. As the name suggests, structural diagrams are used to depict the structure of a system. More specifically, it is used in software development to represent the architecture of the system and how the different components are interconnected (not how they behave or communicate, simply where they stand).

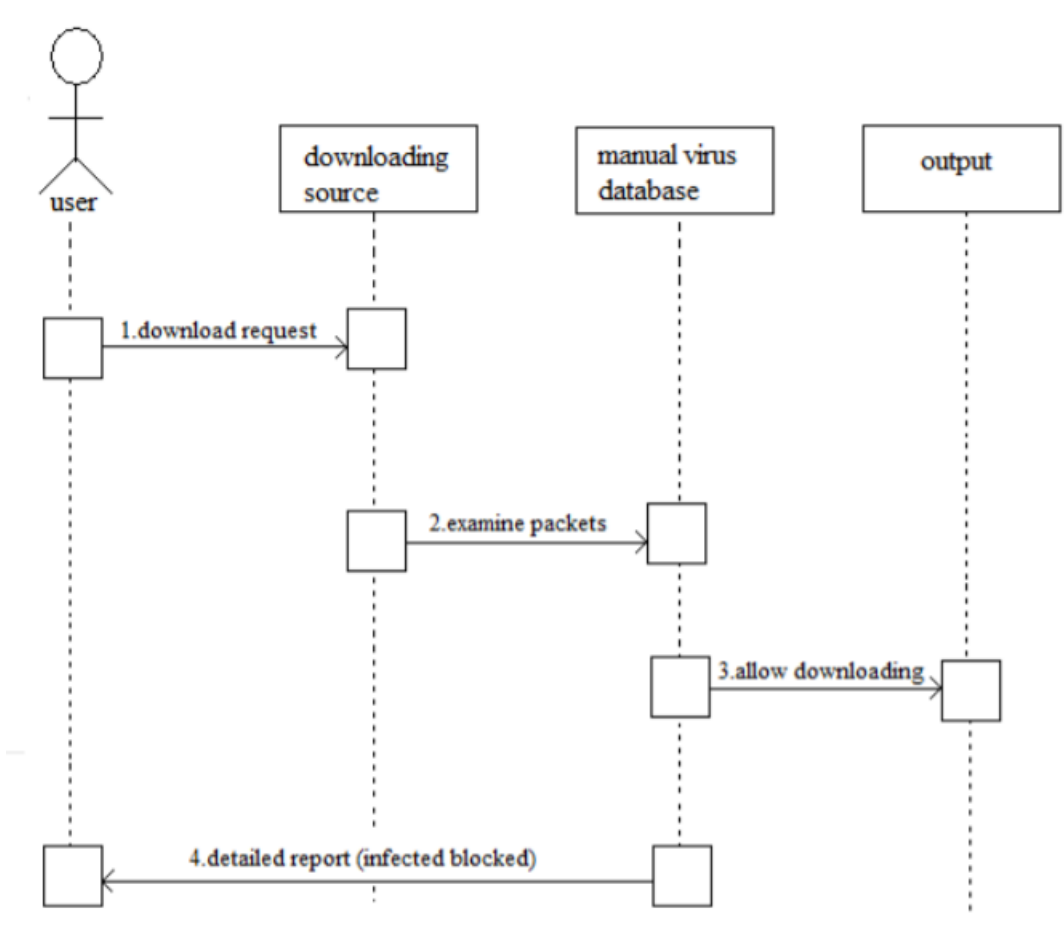


Fig 6.5.1: Sequence Diagram

6.6 ER Diagram

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In other words, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database.²³

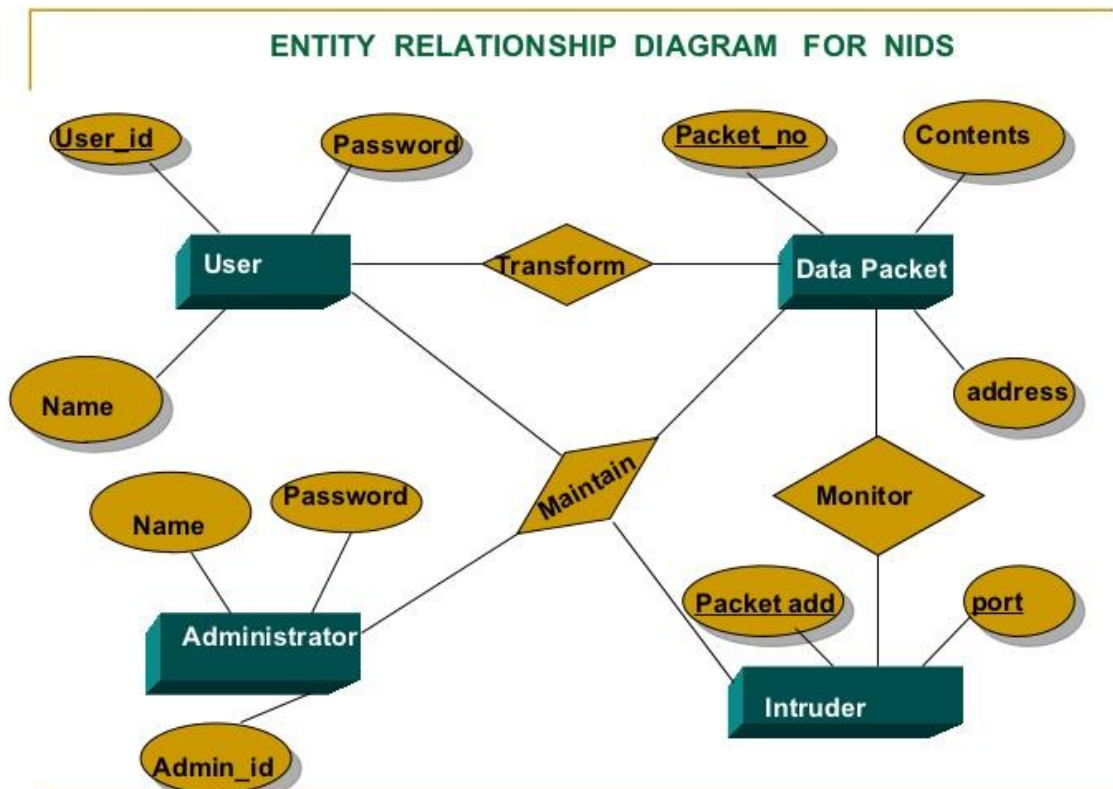


Fig 6.6.1: ER Diagram.

6.7 Data-Flow Diagram

A data-flow diagram is a way of representing a flow of data through a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself.

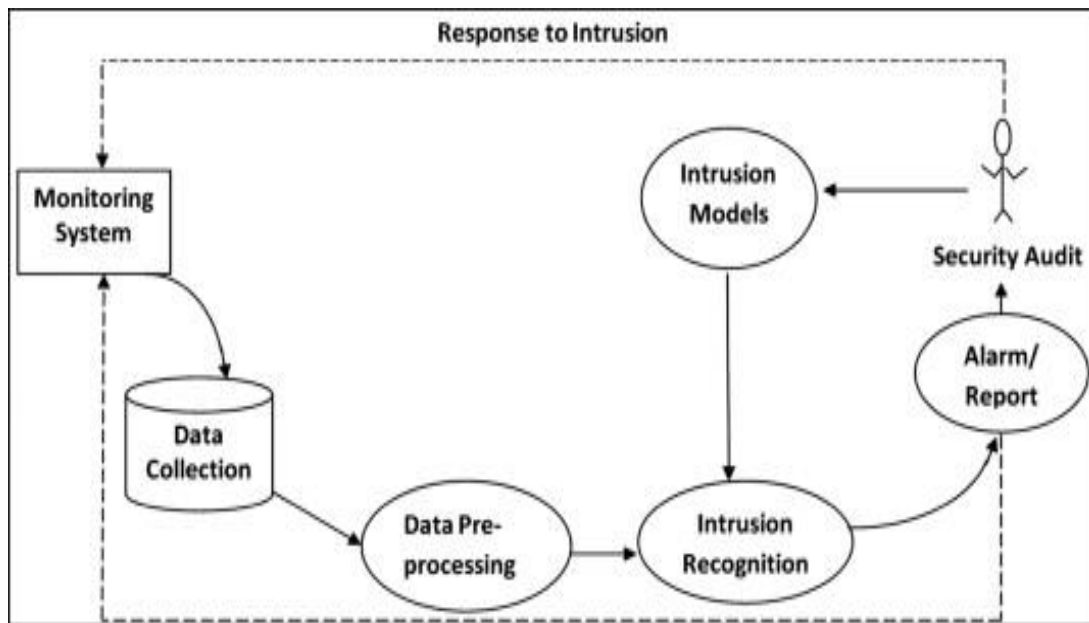


Fig 6.7.1: DFD Diagram

CHAPTER 7

IMPLEMENTATION

7.1 Algorithms

7.1.1 Random forest

A big part of machine learning is classification — we want to know what class (a.k.a. group) an observation belongs to. The ability to precisely classify observations is extremely valuable for various business applications like predicting whether a particular user will buy a product or forecasting whether a given loan will default or not.

Data science provides a plethora of classification algorithms such as logistic regression, support vector machine, naive Bayes classifier, and decision trees. But near the top of the classifier hierarchy is the random forest classifier (there is also the random forest regressor but that is a topic for another day).

In this post, we will examine how basic decision trees work, how individual decisions trees are combined to make a random forest, and ultimately discover why random forests are so good at what they do.

Decision Trees

Let's quickly go over decision trees as they are the building blocks of the random forest model. Fortunately, they are pretty intuitive. I'd be willing to bet that most people have used a decision tree, knowingly or not, at some point in their lives.

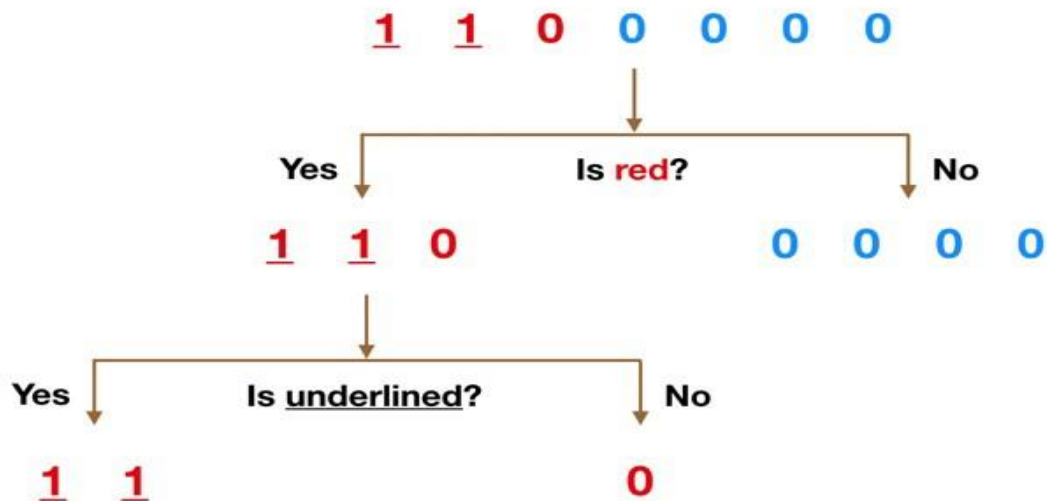


Fig 7.1.1: Random Forest Example

7.1.2 Simple Decision Tree

It's probably much easier to understand how a decision tree works through an example. Imagine that our dataset consists of the numbers at the top of the figure to the left. We have two 1s and five 0s (1s and 0s are our classes) and desire to separate the classes using their features. The features are color (red vs. blue) and whether the observation is underlined or not. So how can we do this?

Color seems like a pretty obvious feature to split by as all but one of the 0s are blue. So we can use the question, "Is it red?" to split our first node. You can think of a node in a tree as the point where the path splits into two — observations that meet the criteria go down the Yes branch and ones that don't go down the No branch.

The No branch (the blues) is all 0s now so we are done there, but our Yes branch can still be split further. Now we can use the second feature and ask, "Is it underlined?" to make a second split.

The two 1s that are underlined go down the Yes subbranch and the 0 that is not underlined goes down the right subbranch and we are all done. Our decision tree was able to use the two features to split up the data perfectly. Victory!

Obviously in real life our data will not be this clean but the logic that a decision tree employs remains the same. At each node, it will ask —

What feature will allow me to split the observations at hand in a way that the resulting groups are as different from each other as possible (and the members of each resulting subgroup are as similar to each other as possible)?

7.1.3 The Random Forest Classifier

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below).

Visualization of a Random Forest Model Making a Prediction

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

7.1.4 K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if $K=2$, there will be two clusters, and for $K=3$, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

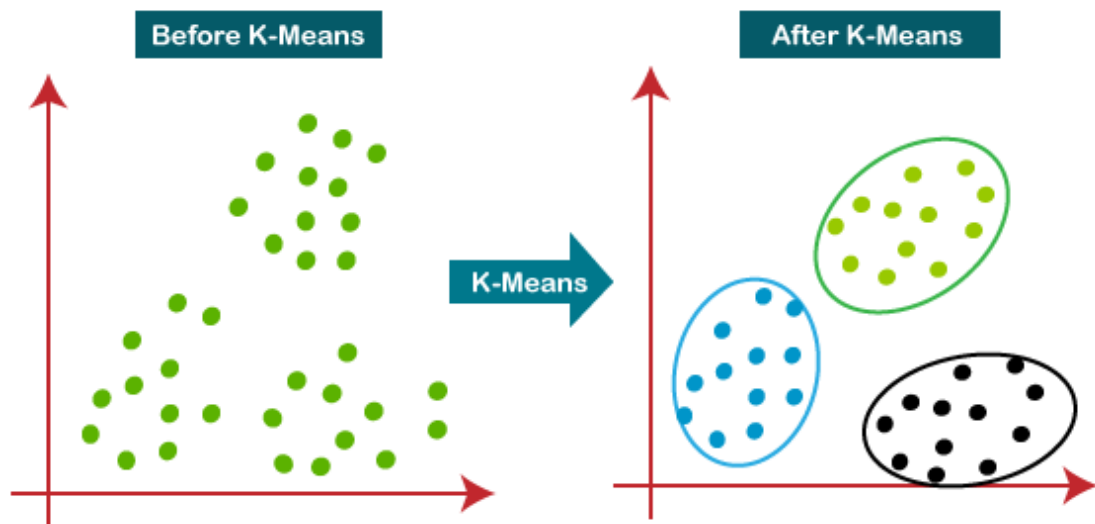


Fig 7.1.4: Difference of Kmeans Plotting Before and After

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

Determines the best value for K center points or centroids by an iterative process.

Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:

How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

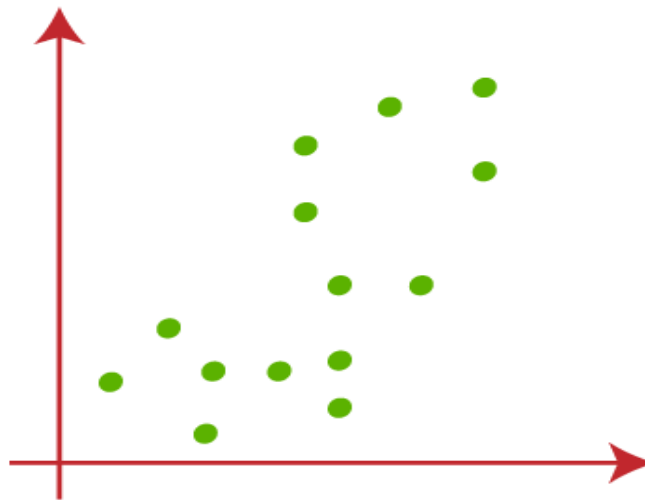


Fig 7.1.4.1: Number of Clusters

Step-2: Select random K points or centroids. (It can be other from the input dataset).

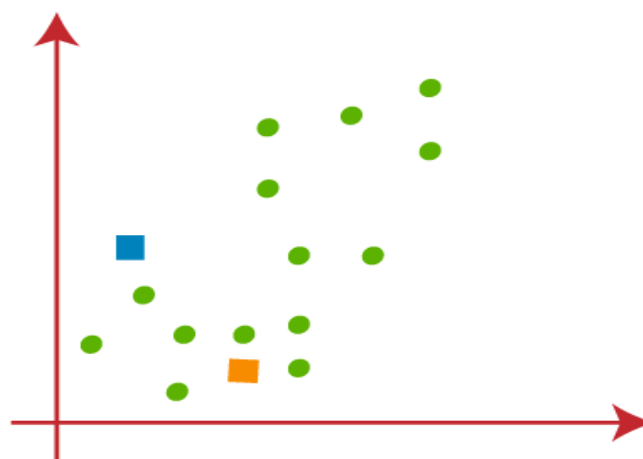


Fig 7.1.4.2: Centroids

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

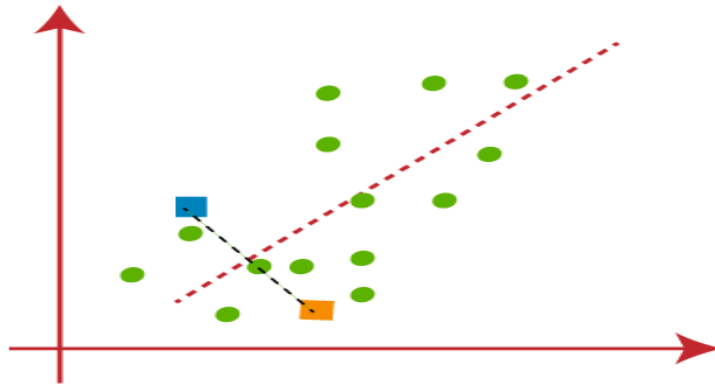


Fig 7.1.4.3: Assignment of Clusters

Step-4: Calculate the variance and place a new centroid of each cluster. Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

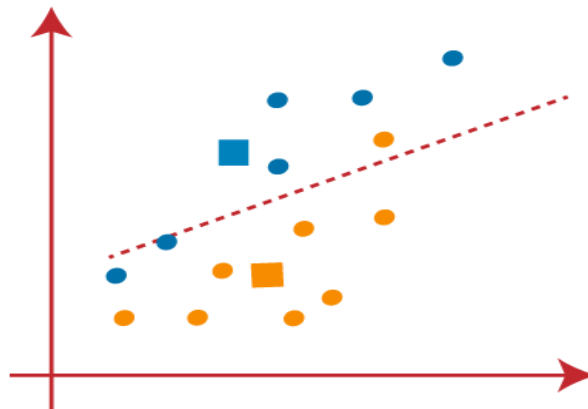


Fig 7.1.4.4: Final Clustered Output

Let's understand the above steps by considering the visual plots:

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

Let's take number k of clusters, i.e., $K=2$, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.

We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset. Consider the below image:

Now we will assign each data point of the scatter plot to its closest K -point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:

From the above image, it is clear that points left side of the line is near to the K_1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.

As we need to find the closest cluster, so we will repeat the process by choosing a new centroid. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:

Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:

From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.

As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K -points.

We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:

As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:

We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:

As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:

7.2 Sample Code

//Mounting the Code To Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

//Grouping of the dataset

```
import numpy as npy
import pandas as pds
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pandas as pd
```

```
dataFrame_1 =
pds.read_csv('/content/drive/MyDrive/intrusion_detetcion_system/Dataset_1.csv')
```

```
grouped_sIPs = dataFrame_1.groupby('sourceIP').count()
grouped_dIPs = dataFrame_1.groupby('destIP').count()
grouped_clfs = dataFrame_1.groupby('classification').count()
```

//Couting the Dataset distinct elements

```
distict_sIPs_count = grouped_sIPs.shape[0]
distict_dIPs_count = grouped_dIPs.shape[0]
distict_clfs_count = grouped_clfs.shape[0]
```

```
print("Distict Source IP addresses are: " + str(distict_sIPs_count))
print("Distict Destination IP addresses are: " + str(distict_dIPs_count))
print("Distict Classifications are: " + str(distict_clfs_count))
```

//Plotting of the Dataset

```
sIP_dict = dataFrame_1.groupby('sourceIP').groups
```

```

dIP_dict = dataframe_1.groupby('destIP').groups

distict_sIPs_keys = list(sIP_dict.keys()) # Distinct Source IPs list
distict_dIPs_keys = list(dIP_dict.keys()) # Distinct Destination IPs list


distict_sIPs_values = list(map(lambda x: x.size, sIP_dict.values())) # Distinct Source
IPs Frequency of appearance
distict_dIPs_values = list(map(lambda x: x.size, dIP_dict.values())) # Distinct
Destination IPs Frequency of appearance


sIPs_df = pd.DataFrame(list(zip(distict_sIPs_keys, distict_sIPs_values)),
                          columns=['SourceIPs', 'Frequency'])

dIPs_df = pd.DataFrame(list(zip(distict_dIPs_keys, distict_dIPs_values)),
                        columns=['DestinationIPs', 'Frequency'])


sIPs_df['IndexColumn'] = sIPs_df.index
dIPs_df['IndexColumn'] = dIPs_df.index


# Dataframe with distinct Source IPs and their Frequency of appearance
print(sIPs_df)


# Dataframe with distinct Destination IPs and their Frequency of appearance
print(dIPs_df)


plt.hist(sIPs_df['Frequency'], bins=12) # Histogram plot of 12 bins
plt.show()


plt.hist(dIPs_df['Frequency'], bins=10)
plt.show()


//Colouring the Distinct elements
color_source = ("green")

```

```
color_destin = ("red")
```

```
plt.scatter(sIPs_df['IndexColumn'], sIPs_df['Frequency'], c=color_source, alpha=0.5)  
plt.show()
```

```
plt.scatter(dIPs_df['IndexColumn'], dIPs_df['Frequency'], alpha=0.5)  
plt.show()
```

// K-means: Finding the optimal number of clusters with Elbow Plot

```
wcss_s = []  
for i in range(1, 11):  
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,  
random_state=0)  
    kmeans.fit(sIPs_df[['IndexColumn', 'Frequency']])  
    wcss_s.append(kmeans.inertia_)  
plt.plot(range(1, 11), wcss_s, 'gx-')  
plt.title('Elbow Method')  
plt.xlabel('Number of clusters for Source IPs')  
plt.ylabel('Distortion')  
plt.show()
```

```
wcss_d = []  
for i in range(1, 11):  
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,  
random_state=0)  
    kmeans.fit(dIPs_df[['IndexColumn', 'Frequency']])  
    wcss_d.append(kmeans.inertia_)  
plt.plot(range(1, 11), wcss_d, 'x-')  
plt.title('Elbow Method')  
plt.xlabel('Number of clusters for Destination IPs')  
plt.ylabel('Distortion')  
plt.show()
```

//K-means clustering with 4 clusters for both Source and Destination IPs

```

kmeans = KMeans(n_clusters=4, init='k-means++')
pred_y = kmeans.fit_predict(sIPs_df[['IndexColumn', 'Frequency']])
plt.scatter(sIPs_df['IndexColumn'], sIPs_df['Frequency'], c=kmeans.labels_,
            cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[0], kmeans.cluster_centers_[1],
            s=sIPs_df.shape[0], c='black')
plt.show()

```

```

kmeans = KMeans(n_clusters=4, init='k-means++')
pred_y = kmeans.fit_predict(dIPs_df[['IndexColumn', 'Frequency']])
plt.scatter(dIPs_df['IndexColumn'], dIPs_df['Frequency'], c=kmeans.labels_,
            cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[0], kmeans.cluster_centers_[1],
            s=dIPs_df.shape[0], c='black')
plt.show()

```

//Hierarchical Clustering of Source IPs

```
import numpy as np
```

```

from scipy.cluster.hierarchy import dendrogram, linkage
x= np.array(sIPs_df[['IndexColumn', 'Frequency']])
linked = linkage(x, 'single')
labelList = range(0, len(x))
plt.figure(figsize=(10, 7))
dendrogram(linked,labels=labelList)
plt.show()

```

Gaussian Mixture Clustering of Source IPs

"Gaussian Mixture Clustering of SourceIPs"

```
from sklearn.mixture import GaussianMixture as GMM
```

```
gmm = GMM(n_components=3).fit(x)
```

```
print(gmm.means_)
```

```

print(gmm.covariances_)
print(gmm.weights_)
labels = gmm.predict(x)
plt.scatter(x[:, 0], x[:, 1], c=labels, s=40, cmap='viridis')

```

//Hierarchial Clustering of Destination IPs

```

from scipy.cluster.hierarchy import dendrogram, linkage
y= np.array(dIPs_df[['IndexColumn', 'Frequency']])
linked = linkage(y, 'single')
labelList = range(0, len(y))
plt.figure(figsize=(10, 7))
dendrogram(linked,labels=labelList)
plt.show()

```

//Gaussian Mixture Clustering of Destination

```

from sklearn.mixture import GaussianMixture as GMM
gmm = GMM(n_components=3).fit(y)
print(gmm.means_)
print(gmm.covariances_)
print(gmm.weights_)
labels = gmm.predict(y)
plt.scatter(y[:, 0], y[:, 1], c=labels, s=40, cmap='viridis')

```

//Ploting the clusters with distinct values

```

import seaborn
seaborn.set(style='ticks')

```

```

def clusterSourceIPs (row):
    if row['Frequency'] <21:
        cluster = 1
    elif row['Frequency'] >20 and row['Frequency'] <201:

```

```

        cluster = 2
    elif row['Frequency'] >200 and row['Frequency'] <401:
        cluster = 3
    else:
        cluster = 4
    return cluster

def clusterDestinationIPs (row):
    if row['Frequency'] <41:
        cluster = 1
    elif row['Frequency'] >40 and row['Frequency'] <101:
        cluster = 2
    elif row['Frequency'] >100 and row['Frequency'] <401:
        cluster = 3
    else:
        cluster = 4
    return cluster

sIPs_df['ClusterLabel'] = sIPs_df.apply(clusterSourceIPs, axis=1) # CReate a new
column with cluster labels for SourceIPs
dIPs_df['ClusterLabel'] = dIPs_df.apply(clusterDestinationIPs, axis=1) # CReate a
new column with cluster labels for DestinationIPs

# Plot the clusters now.....
cluster_range= [1,2,3,4]
sp = seaborn.FacetGrid(data=sIPs_df[['IndexColumn','Frequency','ClusterLabel']],
hue='ClusterLabel', hue_order=cluster_range, aspect=1.61)
sp.map(plt.scatter, 'IndexColumn', 'Frequency').add_legend()
plt.show()

dp = seaborn.FacetGrid(data=dIPs_df[['IndexColumn','Frequency','ClusterLabel']],
hue='ClusterLabel', hue_order=cluster_range, aspect=1.61)
dp.map(plt.scatter, 'IndexColumn', 'Frequency').add_legend()

```

```
plt.show()
```

//Create a dataframe of actual data merged with corresponding Source

```
def createLabelForSource(row):
    ip = row['sourceIP']
    indexVal = sIPs_df[sIPs_df.SourceIPs == ip].index.item();
    """print(indexVal)"""
    """print(ip.head())"""
    """print(sIPs_df.at[indexVal,'ClusterLabel'])"""
    return sIPs_df.at[indexVal,'ClusterLabel']

def createLabelForDestination(row):
    ip = row['destIP']
    indexVal = dIPs_df[dIPs_df.DestinationIPs == ip].index.item();
    """print(indexVal)"""
    """print(ip.head())"""
    """print(sIPs_df.at[indexVal,'ClusterLabel'])"""
    return dIPs_df.at[indexVal,'ClusterLabel']

newDF = dataframe_1[['sourceIP', 'destIP', 'classification']]
"""ddf = newDF[:3]"""

"""newDF.drop(['NewLabel'])"""
and Destination Labels for each IP
newDF['SourceLabel'] = newDF.apply(createLabelForSource, axis=1)
newDF['DestinationLabel'] = newDF.apply(createLabelForDestination, axis=1)
print(newDF)
```

//Create new columns for Source clusters and destination clusters.

```
copiedDF = dataframe_1.copy()

copiedDF['SourceLabel'] = copiedDF.apply(createLabelForSource, axis=1)
```



```
copiedDF['DestinationLabel'] = copiedDF.apply(createLabelForDestination, axis=1)
copiedDF['IndexColumn'] = copiedDF.index
```

```
print(copiedDF.head())
```

//Source and Destination Clusters Relationship

```
copiedDF[copiedDF['SourceLabel'] ==
1].plot('IndexColumn',y=['SourceLabel','DestinationLabel'])
plt.show()
```

```
copiedDF[copiedDF['SourceLabel'] ==
2].plot('IndexColumn',y=['SourceLabel','DestinationLabel'])
plt.show()
```

```
copiedDF[copiedDF['SourceLabel'] ==
3].plot('IndexColumn',y=['SourceLabel','DestinationLabel'])
plt.show()
```

// Printing the Accuracy of the DATASET

```
def train_test_split(df, test_size):

    if isinstance(test_size, float):
        test_size = round(test_size * len(df))

    indices = df.index.tolist()
    test_indices = random.sample(population=indices, k=test_size)

    test_df = df.loc[test_indices]
    train_df = df.drop(test_indices)

    return train_df, test_df
```

```

def check_purity(data):

    label_column = data[:, -1]
    unique_classes = np.unique(label_column)

    if len(unique_classes) == 1:
        return True
    else:
        return False

def classify_data(data):

    label_column = data[:, -1]
    unique_classes, counts_unique_classes = np.unique(label_column,
return_counts=True)

    index = counts_unique_classes.argmax()
    classification = unique_classes[index]

    return classification

def get_potential_splits(data):

    potential_splits = { }
    _, n_columns = data.shape
    for column_index in range(n_columns - 1):
        values = data[:, column_index]
        unique_values = np.unique(values)

        potential_splits[column_index] = unique_values

    return potential_splits

```

```

def split_data(data, split_column, split_value):

    split_column_values = data[:, split_column]

    type_of_feature = FEATURE_TYPES[split_column]
    if type_of_feature == "continuous":
        data_below = data[split_column_values <= split_value]
        data_above = data[split_column_values > split_value]

    # feature is categorical
    else:
        data_below = data[split_column_values == split_value]
        data_above = data[split_column_values != split_value]

    return data_below, data_above

def calculate_entropy(data):

    label_column = data[:, -1]
    _, counts = np.unique(label_column, return_counts=True)

    probabilities = counts / counts.sum()
    entropy = sum(probabilities * -np.log2(probabilities))

    return entropy

def calculate_overall_entropy(data_below, data_above):

    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n

    overall_entropy = (p_data_below * calculate_entropy(data_below)
                       + p_data_above * calculate_entropy(data_above))

```

```

return overall_entropy

def determine_best_split(data, potential_splits):

    overall_entropy = 9999
    for column_index in potential_splits:
        for value in potential_splits[column_index]:
            data_below, data_above = split_data(data, split_column=column_index,
split_value=value)
            current_overall_entropy = calculate_overall_entropy(data_below, data_above)

            if current_overall_entropy <= overall_entropy:
                overall_entropy = current_overall_entropy
                best_split_column = column_index
                best_split_value = value

    return best_split_column, best_split_value

def determine_type_of_feature(df):

    feature_types = []
    n_unique_values_treshold = 15
    for feature in df.columns:
        if feature != "label":
            unique_values = df[feature].unique()
            example_value = unique_values[0]

            if (isinstance(example_value, str)) or (len(unique_values) <=
n_unique_values_treshold):
                feature_types.append("categorical")
            else:
                feature_types.append("continuous")

```

```
return feature_types
```

```
def decision_tree_algorithm(df, counter=0, min_samples=2, max_depth=5):
```

```
    # data preparations
```

```
    if counter == 0:
```

```
        global COLUMN_HEADERS, FEATURE_TYPES
```

```
        COLUMN_HEADERS = df.columns
```

```
        FEATURE_TYPES = determine_type_of_feature(df)
```

```
        data = df.values
```

```
    else:
```

```
        data = df
```

```
    # base cases
```

```
    if (check_purity(data)) or (len(data) < min_samples) or (counter == max_depth):
```

```
        classification = classify_data(data)
```

```
    return classification
```

```
    # recursive part
```

```
    else:
```

```
        counter += 1
```

```
    # helper functions
```

```
    potential_splits = get_potential_splits(data)
```

```
    split_column, split_value = determine_best_split(data, potential_splits)
```

```
    data_below, data_above = split_data(data, split_column, split_value)
```

```
    # check for empty data
```

```
    if len(data_below) == 0 or len(data_above) == 0:
```

```
        classification = classify_data(data)
```

```
    return classification
```

```

# determine question
feature_name = COLUMN_HEADERS[split_column]
type_of_feature = FEATURE_TYPES[split_column]
if type_of_feature == "continuous":
    question = "{} <= {}".format(feature_name, split_value)

# feature is categorical
else:
    question = "{} = {}".format(feature_name, split_value)

# instantiate sub-tree
sub_tree = {question: []}

# find answers (recursion)
yes_answer = decision_tree_algorithm(data_below, counter, min_samples,
max_depth)
no_answer = decision_tree_algorithm(data_above, counter, min_samples,
max_depth)

# If the answers are the same, then there is no point in asking the question.
# This could happen when the data is classified even though it is not pure
# yet (min_samples or max_depth base case).
# if yes_answer == no_answer:
    #sub_tree = yes_answer
#else:
    sub_tree[question].append(yes_answer)
    sub_tree[question].append(no_answer)

return sub_tree

def classify_example(example, tree):
    question = list(tree.keys())[0]
    feature_name, comparison_operator, value = question.split(" ")

```

```

# ask question
if comparison_operator == "<=": # feature is continuous
    if example[feature_name] <= float(value):
        answer = tree[question][0]
    else:
        answer = tree[question][1]

# feature is categorical
else:
    if str(example[feature_name]) == value:
        answer = tree[question][0]
    else:
        answer = tree[question][1]

# base case
if not isinstance(answer, dict):
    return answer

# recursive part
else:
    residual_tree = answer
    return classify_example(example, residual_tree)

def calculate_accuracy(df, tree):

    df["classification"] = df.apply(classify_example, axis=1, args=(tree,))
    df["classification_correct"] = df["classification"] == df["label"]

    accuracy = df["classification_correct"].mean()

    return accuracy

df = df[['sourceIP', 'destIP', 'SourceLabel', 'DestinationLabel', 'classification']]

```

```
print(df)
```

```
df["label"] = df.classification  
df = df.drop(["classification"], axis=1)  
print(df.head())
```

```
//Printing the accuracy
```

```
import random  
from pprint import pprint
```

```
random.seed(0)
```

```
train_df, test_df = train_test_split(df, 0.2)  
tree = decision_tree_algorithm(train_df, max_depth=3)  
accuracy = calculate_accuracy(test_df, tree)
```

```
pprint(tree, width=50)  
accuracy
```

```
//Dataframe with second dataset.
```

```
dataFrame_2 =  
pds.read_csv('/content/drive/MyDrive/intrusion_detetcion_system/Dataset_2.csv')
```

```
sIP_dict_2 = dataFrame_2.groupby('sourceIP').groups  
dIP_dict_2 = dataFrame_2.groupby('destIP').groups
```

```
distict_sIPs_keys_2 = list(sIP_dict_2.keys()) # Distinct Source IPs list  
distict_dIPs_keys_2 = list(dIP_dict_2.keys()) # Distinct Destination IPs list
```



```

distinct_sIPs_values_2 = list(map(lambda x: x.size, sIP_dict_2.values())) # Distinct
Source IPs Frequency of appearance
distinct_dIPs_values_2 = list(map(lambda x: x.size, dIP_dict_2.values())) # Distinct
Destination IPs Frequency of appearance

sIPs_df_2 = pd.DataFrame(list(zip(distinct_sIPs_keys_2, distinct_sIPs_values_2)),
                           columns=['SourceIPs', 'Frequency'])

dIPs_df_2 = pd.DataFrame(list(zip(distinct_dIPs_keys_2, distinct_dIPs_values_2)),
                           columns=['DestinationIPs', 'Frequency'])

sIPs_df_2['IndexColumn'] = sIPs_df_2.index
dIPs_df_2['IndexColumn'] = dIPs_df_2.index

sIPs_df_2['ClusterLabel'] = sIPs_df_2.apply(clusterSourceIPs, axis=1)
dIPs_df_2['ClusterLabel'] = dIPs_df_2.apply(clusterDestinationIPs, axis=1)

def createLabelForSource_2(row):
    ip = row['sourceIP']
    indexVal = sIPs_df_2[sIPs_df_2.SourceIPs == ip].index.item()
    return sIPs_df_2.at[indexVal, 'ClusterLabel']

def createLabelForDestination_2(row):
    ip = row['destIP']
    indexVal = dIPs_df_2[dIPs_df_2.DestinationIPs == ip].index.item()
    return dIPs_df_2.at[indexVal, 'ClusterLabel']

newDF_2 = dataframe_2[['sourceIP', 'destIP', 'classification']]
newDF_2['SourceLabel'] = newDF_2.apply(createLabelForSource_2, axis=1)
newDF_2['DestinationLabel'] = newDF_2.apply(createLabelForDestination_2,
axis=1)
print(newDF_2)

```

```

df_2 = newDF_2.copy()
"print (df_2)"
df_2 = df_2[['sourceIP', 'destIP', 'SourceLabel', 'DestinationLabel', 'classification']]
"print(df)"
df_2["label"] = df_2.classification
df_2 = df_2.drop(["classification"], axis=1)
print(df_2.head())

```

//Printing the accuracy of the 2nd dataset

```

train_df_2, test_df_2 = train_test_split(df_2, 0.2)
tree_2 = decision_tree_algorithm(train_df_2, max_depth=3)
accuracy_2 = calculate_accuracy(test_df_2, tree)

```

```

pprint(tree_2, width=50)
accuracy_2

```

```

# -*- coding: utf-8 -*-
"""random_forest_Weka.ipynb

```

Automatically generated by Colaboratory.

Original file is located at

```

https://colab.research.google.com/drive/16OxUdon3CrmrTJZhMzb5EaVJ3vd04jnW
"""

```

```

import tensorflow as tf
import pandas as pd

```

```

from google.colab import drive
drive.mount('/content/drive')

```

```

df = pd.read_csv('/content/drive/MyDrive/weka.csv')

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# might be needed depending on your version of Jupyter
# %matplotlib inline

df.info()

df.head()

df.corr()

df.isnull().sum().sum()

df[' Label'].unique()

len(df)

df.select_dtypes(['object']).columns

df.shape

df.describe()

df.transpose()

sns.countplot(x=' Label',data=df);

# converting each Label attacks names into digits

```

```
df['label'] = pd.factorize(df[' Label'])[0]
```

```
df.head()
```

```
df = df.drop(' Label', axis=1)
```

```
df.head()
```

```
"""# Feature Selection """
```

```
df.replace([np.inf, -np.inf], np.nan).dropna(axis=1)
```

```
# Instead of dropping rows which contain any nulls and infinite numbers,
```

```
# it is more succinct to the reverse the logic of that and instead
```

```
#return the rows where all cells are finite numbers.
```

```
#The numpy isfinite function does this and the '.all(1)' will only return a
```

```
#TRUE if all cells in row are finite.
```

```
df = df[np.isfinite(df).all(1)]
```

```
len(df)
```

```
from sklearn.model_selection import train_test_split
```

```
X = df.drop('label',axis=1).values
```

```
y = df['label'].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
```

```
random_state=101)
```

```
X.shape
```

```
y.shape
```

```
"""Remove Constant, Quasi Constant and Duplicate features"""
```

```
from sklearn.feature_selection import VarianceThreshold
```

```
constant_filter = VarianceThreshold(threshold=0.01)
```

```
constant_filter.fit(X_train)
```

```
X_train_filter = constant_filter.transform(X_train)
```

```
X_test_filter = constant_filter.transform(X_test)
```

```
X_train_filter.shape, X_test_filter.shape
```

```
"""Remove duplicate features"""
```

```
X_train_T = X_train_filter.T # taking the transpose
```

```
X_test_T = X_test_filter.T
```

```
X_train_T = pd.DataFrame(X_train_T)
```

```
X_test_T = pd.DataFrame(X_test_T)
```

```
X_train_T.duplicated().sum()
```

```
duplicate_features = X_train_T.duplicated()
```

```
features_to_keep = [not index for index in duplicate_features]
```

```
X_train_unique = X_train_T[features_to_keep].T
```

```
X_test_unique = X_test_T[features_to_keep].T
```

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler().fit(X_train_unique)
```

```
X_train_unique = scaler.transform(X_train_unique)
```

```
X_test_unique = scaler.transform(X_test_unique)
```

```
X_train_unique = pd.DataFrame(X_train_unique)
```

```
X_test_unique = pd.DataFrame(X_test_unique)
```

```
X_train_unique.shape, X_test_unique.shape
```

```
"""Pearson Correlated features removal"""
```

```
corrmat = X_train_unique.corr()
```

```
plt.figure(figsize=(12,4))
```

```
sns.heatmap(corrmat)
```

```
corrmat.shape
```

```
# Finding the correlated features
```

```
def get_correlation(data, threshold):
```

```
    corr_col = set() # makes a set of unrepeated data
```

```
    corrmat = data.corr() # getting a corr matrix
```

```
    for i in range(len(corrmat.columns)):
```

```
        for j in range(i): # corr -> (i,j) = (1,0), (1,2), ..., (2,0).....
```

```
            if abs(corrmat.iloc[i,j]) > threshold:
```

```
                colname = corrmat.columns[i]
```

```
                corr_col.add(colname)
```

```
    return corr_col
```

```
corr_features = get_correlation(X_train_unique, 0.70)
```

```
print("Correlated features:", len(set(corr_features)))
```

```
corr_features
```

```
X_train_uncorr = X_train_unique.drop(labels=corr_features, axis=1)
```

```
X_test_uncorr = X_test_unique.drop(labels=corr_features, axis=1)
```

```
X_train_uncorr.shape, X_train_unique.shape
```

```
"""# Random Forest Classifier"""
```

```

from sklearn.ensemble import RandomForestClassifier

# Instantiate model with 100 decision trees
rc = RandomForestClassifier(n_estimators = 100, random_state = 101, n_jobs=-1)
# Train the model on training data
rc.fit(X_train_uncorr, y_train)

predictions = rc.predict(X_test_uncorr)

predictions

# viewing the predicted probabilities of first 10 rows of test
rc.predict_proba(X_test_uncorr)[0:10]

# confusion matrix
pd.crosstab(y_test, predictions, rownames=['Actual Attack'], colnames=['Predicted
Attack'] )
from sklearn.metrics import classification_report

print(classification_report(y_test, predictions))

```

CHAPTER 8

TESTING

8.1 Testing Methodology

8.1.1 Testing Strategies and Methodologies

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. Testing is the debugging program is one of the most critical aspects of the computer programming triggers, without programming that works, the system would never produce an output of which it was designed. Testing is best performed when user development is asked to assist in identifying all errors and bugs. The sample data are used for testing. It is not quantity but quality of the data used the matters of testing. Testing is aimed at ensuring that the system was accurately and efficiently before live operation commands.

- Testing objectives

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time. Stating formally, we can say, testing is a process of executing a program with intent of finding an error.

- 1) A successful test is one that uncovers an as yet undiscovered error.
- 2) A good test case is one that has probability of finding an error, if it exists.
- 3) The test is inadequate to detect possibly present errors.
- 4) The software more or less confirms to the quality and reliable standards.

- Levels of Testing

In order to uncover present in different phases we have the concept of levels of testing. Tests are grouped together based on where they are added in SDLC or the by the level of detailing they contain. In general, there are four levels of testing: unit testing, integration testing, system testing, and acceptance testing. The purpose of Levels of testing is to make software testing systematic and easily identify all possible test cases at a particular level.

There are many different testing levels which help to check behaviour and performance for software testing. These testing levels are designed to recognize missing areas and reconciliation between the development lifecycle states. In SDLC models there are characterized phases such as requirement gathering, analysis, design, coding or execution, testing, and deployment.

8.1.2 Code testing

This examines the logic of the program. For example, the logic for updating various sample data and with the sample files and directories were tested and verified.

Executing this specification starting what the program should do and how it should performed under various conditions. Test cases for various situation and combination of conditions in all the modules are tested.- 69 -

8.1.3 Unit testing

In the unit testing we test each module individually and integrate with the overall system. Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results. Each Module can be tested using the following two Strategies:

➤ Black Box Testing

➤ White Box Testing

8.1.4 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot “see” into it. Black Box Testing we just focus on inputs and output of the software system without

bothering about internal knowledge of the software program. The above Black Box can be any software system you want to test. For example: an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

8.1.5 White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

White Box Testing is the testing of a software solution's internal coding and infrastructure. It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability. White box testing is also known as clear, open, structural, and glass box testing.

It is one of two parts of the "box testing" approach of software testing. Its counterpart, black box testing, involves testing from an external or end-user type perspective. On the other hand, White box testing is based on the inner workings of an application and revolves around internal testing. The term "white box" was used because of the seethrough box concept. The clear box or white box name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "black box testing" symbolizes not being able to see the innerworkings of the software so that only the end-user experience can be tested.

White box testing involves the testing of the software code for the following:

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object and function on an individual basis

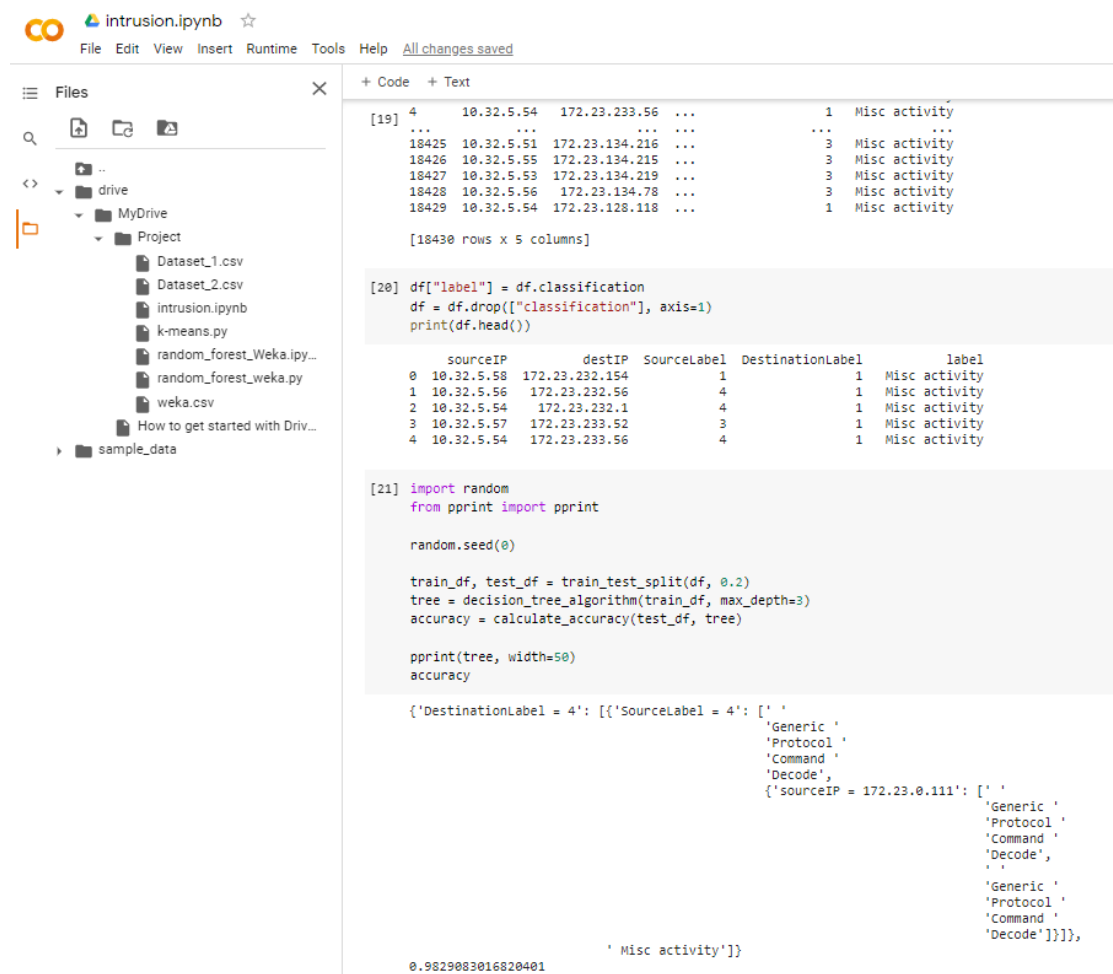
The testing can be done at system, integration and unit levels of software development.

One of the basic goals of white box testing is to verify a working flow for an application.

8.2 OUTPUT SCREENS

8.2.1 Misc Activity in KMEANS

In the below Figure you can see that if there is a Misc Activity in the given dataset, the output shows a accuracy value of 0.98(Fig 7.2.1) which says that there is some Intrusion in the System activity



```
[19] 4      10.32.5.54      172.23.233.56      ...      1      Misc activity
...      ...      ...      ...      ...
18425  10.32.5.51      172.23.134.216      ...      3      Misc activity
18426  10.32.5.55      172.23.134.215      ...      3      Misc activity
18427  10.32.5.53      172.23.134.219      ...      3      Misc activity
18428  10.32.5.56      172.23.134.78      ...      3      Misc activity
18429  10.32.5.54      172.23.128.118      ...      1      Misc activity

[18430 rows x 5 columns]

[20] df["label"] = df.classification
df = df.drop(["classification"], axis=1)
print(df.head())

      sourceIP      destIP  SourceLabel  DestinationLabel      label
0  10.32.5.58      172.23.232.154      1      1      Misc activity
1  10.32.5.56      172.23.232.56      4      1      Misc activity
2  10.32.5.54      172.23.232.1      4      1      Misc activity
3  10.32.5.57      172.23.233.52      3      1      Misc activity
4  10.32.5.54      172.23.233.56      4      1      Misc activity

[21] import random
from pprint import pprint

random.seed(0)

train_df, test_df = train_test_split(df, 0.2)
tree = decision_tree_algorithm(train_df, max_depth=3)
accuracy = calculate_accuracy(test_df, tree)

pprint(tree, width=50)
accuracy

{'DestinationLabel = 4': [{'SourceLabel = 4': ['Generic ',
'Protocol ',
'Command ',
'Decode',
],
'sourceIP = 172.23.0.111': ['Generic ',
'Protocol ',
'Command ',
'Decode']]]},
'Misc activity']]

0.9829083016820401
```

Fig 8.2.1: Result Shows the Misc Activity

8.2.2 Without Misc Activity

Here we can see that with the misc activity in the given Dataset is removed, so the Accuracy of the system have changed to 0.60(Fig 7.2.2) which says it is Clean without any Intrusion

```

intrusion.ipynb
File Edit View Insert Runtime Tools Help All changes saved

Files
drive
MyDrive
Project
Dataset_1.csv
Dataset_2.csv
intrusion.ipynb
k-means.py
random_forest_Weka.py
random_forest_weka.py
weka.csv
How to get started with Driv...
sample_data

[25] 32642 10.32.5.56 172.23.233.2 ... 4 1

[32643 rows x 5 columns]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

[26] df_2 = newDF_2.copy()
      ''print(df_2)''
      df_2 = df_2[['sourceIP', 'destIP', 'SourceLabel', 'DestinationLabel', 'classification']]
      ''print(df)''
      df_2["label"] = df_2.classification
      df_2 = df_2.drop(["classification"], axis=1)
      print(df_2.head())

      sourceIP ... label
0 172.23.1.101 ... Generic Protocol Command Decode
1 172.23.1.101 ... Generic Protocol Command Decode
2 172.23.1.101 ... Generic Protocol Command Decode
3 172.23.1.101 ... Generic Protocol Command Decode
4 172.23.0.212 ... Generic Protocol Command Decode

[5 rows x 5 columns]

train_df_2, test_df_2 = train_test_split(df_2, 0.2)
tree_2 = decision_tree_algorithm(train_df_2, max_depth=3)
accuracy_2 = calculate_accuracy(test_df_2, tree)

pprint(tree_2, width=50)
accuracy_2

{'SourceLabel = 2': [{'sourceIP = 172.23.231.69': ['Attempted', 'Information', 'Leak'],
                    {'sourceIP = 172.23.232.4': ['Attempted', 'Information', 'Leak',
                    'Attempted', 'Information', 'Leak']}]},
                    'Generic Protocol Command Decode']}
0.6079032011027723

```

Fig 8.2.2: Result Without Misc Activity

8.2.3 Elbow Plotting

Using the "elbow" or "knee of a curve" as a cutoff point is a common heuristic

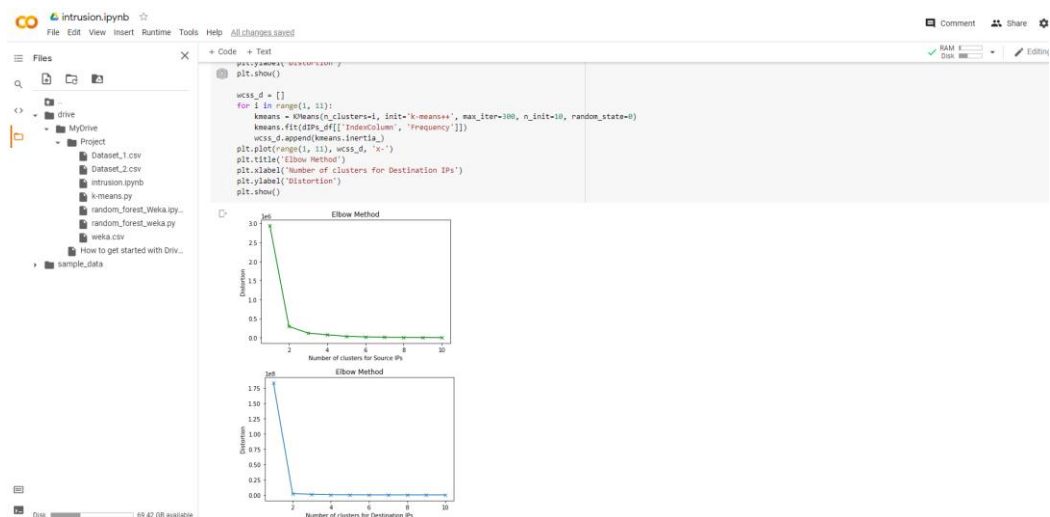


Fig 8.2.3: Elbow Plotting Diagram

8.2.4 Feature Selection in Random Forest

When building a machine learning model in real-life, it's almost rare that all the variables in the dataset are useful to build a model. Adding redundant variables reduces the generalization capability of the model and may also reduce the overall accuracy of a classifier. Furthermore adding more and more variables to a model increases the overall complexity of the model. As per the Law of Parsimony of 'Occam's Razor', the best explanation to a problem is that which involves the fewest possible assumptions. Thus, feature selection becomes an indispensable part of building machine learning models.

The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The code cell contains a line of Python code: `[13] df.replace([np.inf, -np.inf], np.nan).dropna(inplace=True)`. Below the code, a large table of data is displayed, representing the dataset. The table has 74 columns, including 'Destination Port', 'Flow Duration', 'Total Fwd Packets', 'Total Length of Fwd Packets', and various packet length statistics (e.g., 'Fwd Packet Length Min', 'Fwd Packet Length Max', etc.). The first few rows of data are visible, showing values for these metrics across different samples.

Fig 8.2.4: Features in the Dataset

8.2.5 Count Of Different Attacks

Here the Fig Shows The Count of attacks of different kinds of attacks we inputted in the dataset

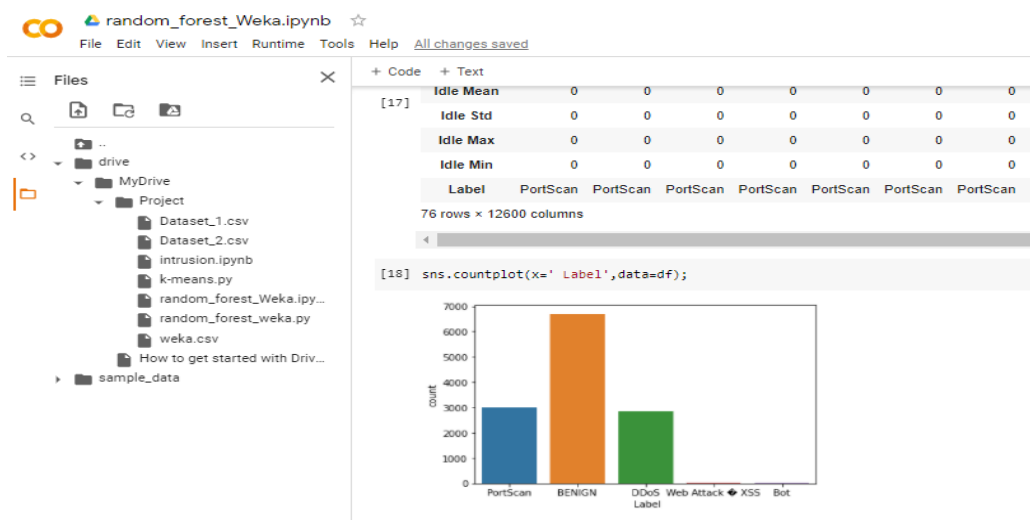


Fig 8.2.5: Different Kinds Of Attacks and There Counts

8.2.6 Correlations

In statistics, correlation or dependence is any statistical relationship, whether causal or not, between two random variables or bivariate data. In the broadest sense correlation is any statistical association, though it commonly refers to the degree to which a pair of variables are linearly related. Familiar examples of dependent phenomena include the correlation between the height of parents and their offspring, and the correlation between the price of a good and the quantity the consumers are willing to purchase, as it is depicted in the so-called demand curve.

Correlations are useful because they can indicate a predictive relationship that can be exploited in practice. For example, an electrical utility may produce less power on a mild day based on the correlation between electricity demand and weather. In this example, there is a causal relationship, because extreme weather causes people to use more electricity for heating or cooling. However, in general, the presence of a correlation is not sufficient to infer the presence of a causal relationship (i.e., correlation does not imply causation).

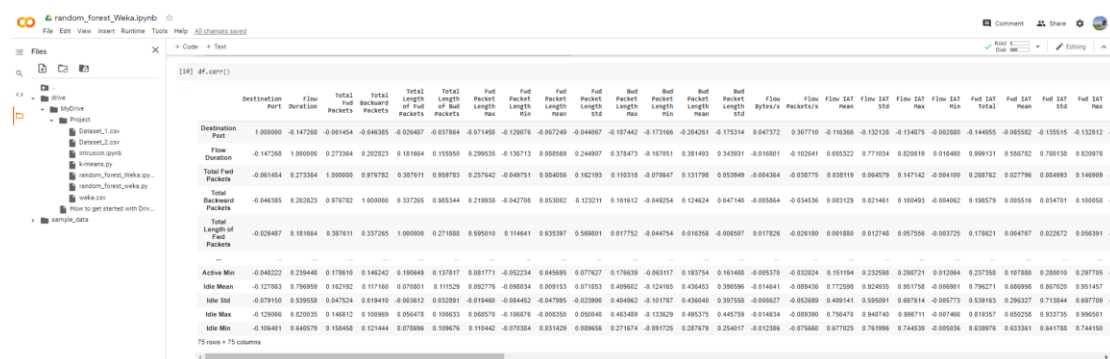


Fig 8.2.6:Correlation

8.2.7 HeatMap

A heatmap is a graphical representation where individual values of a matrix are represented as colors. A heatmap is very useful in visualizing the concentration of values between two dimensions of a matrix. This helps in finding patterns and gives a perspective of depth.

Let's start off by creating a basic heatmap between two dimensions. We'll create a 10 x 6 matrix of random values and visualize it as a heatmap

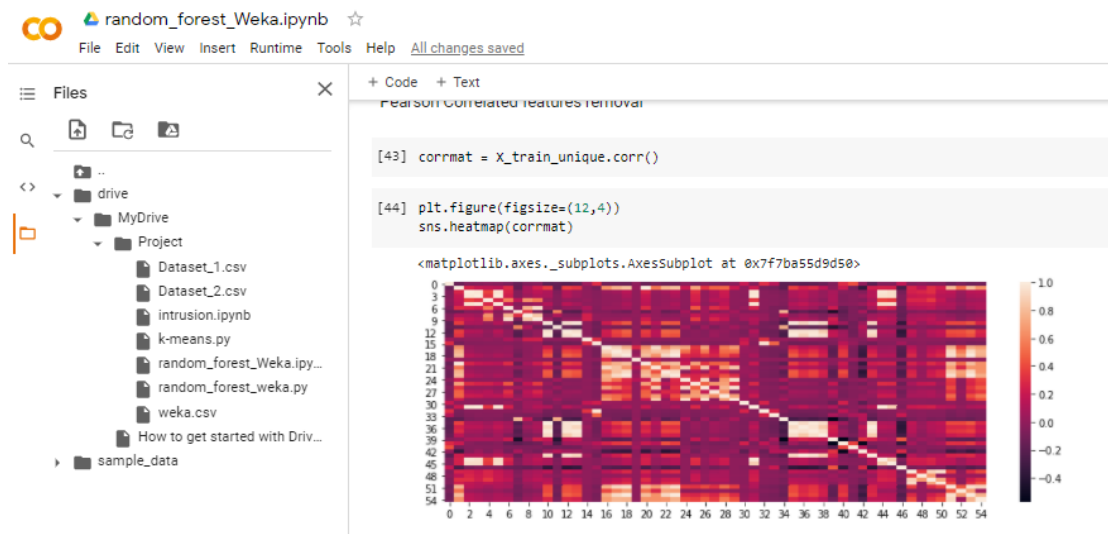


Fig 7.2.7:HeatMap

8.2.8 Predicted Values

```
[54] # viewing the predicted probabilities of first 10 rows of test
      rc.predict_proba(X_test_uncorr)[0:10]

array([[0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.]])
```

Fig 8.2.8: Predicted Values Of First 10 Rows

8.2.9 Confusion Matrix

A Confusion matrix is an $N \times N$ matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares

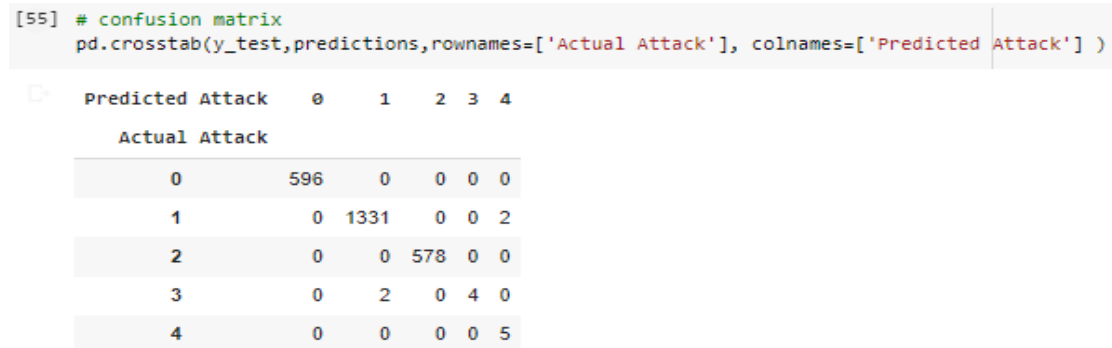


Fig 7.2.9: Confusion Matrix

the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

8.2.10 Precision, F1 Score, Recall, Support

Precision says that, In pattern recognition, information retrieval and classification (machine learning), precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that were retrieved.

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

In an imbalanced classification problem with two classes, recall is calculated as the number of true positives divided by the total number of true positives and false negatives. The result is a value between 0.0 for no recall and 1.0 for full or perfect

REcall.

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

```
[56] from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	596
1	1.00	1.00	1.00	1333
2	1.00	1.00	1.00	578
3	1.00	0.67	0.80	6
4	0.71	1.00	0.83	5
accuracy			1.00	2518
macro avg	0.94	0.93	0.93	2518
weighted avg	1.00	1.00	1.00	2518

Fig 8.2.10: Scores of Precision, F1 Score, Recall, Support

CHAPTER 9

CONCLUSION AND FUTURE WORK

Hybrid IDS is developed using machine learning approaches. It combines Random Forest classification and K-Means clustering. This will use both misuse detection and anomaly detection for improving performance of the IDS. These algorithms are evaluated for the four categories of attacks based on accuracy, false-alarm-rate, and detection-rate etc. The four kinds of intrusions considered are Denial of Service (DoS), trying to have unauthorized access such as guessing password (R2L), unauthorized access such as violating privileges (U2R) and probing (such as port scanning and surveillance). Experiments are made with NSL-KDD dataset. The IDS is supported by feature selection method known as attribute ratio. Anaconda data science platform issued to develop IDS. The results showed that the hybrid IDS has around 99% detection rate.

CHAPTER 10

REFERENCES

- [1] Om, H., & Kundu, A. (2012). *A hybrid system for reducing the false alarm rate of anomaly intrusion detection system. 2012 1st International Conference on Recent Advances in Information Technology (RAIT)*. P1-6.
- [2] Lin, W.-C., Ke, S.-W., & Tsai, C.-F. (2015). *CANN: An intrusion detection system based on combining cluster centres and nearest neighbours. Knowledge-Based Systems*, 78, 13–21.
- [3] Maglaras, L. A., & Jiang, J. (2014). *OCSVM model combined with K-means recursive clustering for intrusion detection in SCADA systems. 10th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*. P1-2.
- [4] Al-Yaseen, W. L., Othman, Z. A., & Nazri, M. Z. A. (2017). *Multi-level hybrid support vector machine and extreme learning machine based on modified K-means for intrusion detection system. Expert Systems with Applications*, 67, 296–303.
- [5] Ravale, U., Marathe, N., & Padiya, P. (2015). *Feature Selection Based Hybrid Anomaly Intrusion Detection System Using K Means and RBF Kernel Function. Procedia Computer Science*, 45, 428–435.
- [6] Muniyandi, A. P., Rajeswari, R., & Rajaram, R. (2012). *Network Anomaly Detection by Cascading K-Means Clustering and C4.5 Decision Tree algorithm. Procedia Engineering*, 30, 174–182.
- [7] Elbasiony, R. M., Sallam, E. A., Eltobely, T. E., & Fahmy, M. M. (2013). *A hybrid network intrusion detection framework based on random forests and weighted k-means. Ain Shams Engineering Journal*, 4(4), 753–762.
- [8] Gupta, G. P., & Kulariya, M. (2016). *A Framework for Fast and Efficient Cyber Security Network Intrusion Detection Using Apache Spark. Procedia Computer Science*, 93, 824–831.

- [9] Chauhan, H., Kumar, V., Pundir, S., & Pilli, E. S. (2013). *A Comparative Study of Classification Techniques for Intrusion Detection. 2013 International Symposium on Computational and Business Intelligence*. P1-4.
- [10] Thaseen, S., & Kumar, C. A. (2013). *An analysis of supervised tree based classifiers for intrusion detection system. 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*. P1-6.
- [11] Farnaaz, N., & Jabbar, M. A. (2016). *Random Forest Modeling for Network Intrusion Detection System. Procedia Computer Science*, 89, 213–217.
- [12] Singh, K., Guntuku, S. C., Thakur, A., & Hota, C. (2014). *Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests. Information Sciences*, 278, 488–497.
- [13] Mazini, M., Shirazi, B., & Mahdavi, I. (2018). *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*. p1-30.
- [14] Nadiammai, G. V., & Hemalatha, M. (2014). *Effective approach toward Intrusion Detection System using data mining techniques. Egyptian Informatics Journal*, 15(1), 37–50.
- [15] Aljawarneh, S., Aldwairi, M., & Yassein, M. B. (2018). *Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model. Journal of Computational Science*, 25, 152–160.
- [16] Aslahi-Shahri, B. M., Rahmani, R., Chizari, M., Maralani, A., Eslami, M., Golkar, M. J., & Ebrahimi, A. (2015). *A hybrid method consisting of GA and SVM for intrusion detection system. Neural Computing and Applications*, 27(6), 1669–1676.
- [17] Maleh, Y., Ezzati, A., Qasmaoui, Y., & Mbida, M. (2015). *A Global Hybrid Intrusion Detection System for Wireless Sensor Networks. Procedia Computer Science*, 52, 1047–1052.

- [18] Kim, G., Lee, S., & Kim, S. (2014). *A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. Expert Systems with Applications, 41(4), 1690–1700.*
- [19] Pan, S., Morris, T., & Adhikari, U. (2015). *Developing a Hybrid Intrusion Detection System Using Data Mining for Power Systems. IEEE Transactions on Smart Grid, 6(6), 3104–3113.*
- [20] NSL-KDD Dataset. Retrieved from <https://www.unb.ca/cic/datasets/index.html>