Intrusion Detection System with Machine Learning Algorithms and Comparison Analysis

A PROJECT REPORT

Submittedby

JAIN KARAN ANAND

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

ARTIFICIAL INTELIGENCE & MACHINE LEARNING

Chandigarh University

BONAFIDE CERTIFICATE

Certified that this project report **"Intrusion Detection System with Machine Learning Algorithms and Comparison Analysis"**is the bonafide work of "**JAIN KARAN ANAND"** who carried out the project work under my/our supervision.

<<Signature of the HoD>>

SIGNATURE

<<Name of the Head of the Department>>

HEAD OF THE DEPARTMENT

<<Department>>

<<Signature of the Supervisor>>

SIGNATURE

Prof. Siddharth Kumar

SUPERVISOR

Project Head

AIT - CSE

Submitted for the project viva-voce examination held on _

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

The incremental increase in the usage of technology has led to an increase in the amount of data that is being processed over the Internet significantly over the time period. With the huge amount of data that is being flown over the Internet, comes the scenario of providing security to the data, and this is where an Intrusion Detection System (IDS) comes into the picture and helps in detecting any virtual security threats. Intrusion Detection System (IDS) is a system that monitors and analyzes data to detect any intrusion in the system or network. Intruders find different ways to penetrate into a network. The IDS which is being proposed is being implemented using latest technologies such as Machine Learning Algorithms to classify the attacks and detecting them whenever an attack happens and also to find which machine learning algorithm is best suitable for identifying the attack.

Keywords:

Intrusion, Intrusion Detection System, Denial of service, User to Root attacks, Remote to User attacks, Local Area Network, Principal Component Analysis, Support Vector Machine, Random Forest, Decision Tree, KNN Algorithm, Logistic Regression, Alerts, False Positives, False Negatives.

LIST OF FIGURES

FIGURE NUMBER	FIGURE NAME	PAGE NUMBER
3.1.1.1	System Architecture	16
3.1.2.1.1	Data Visualization for Intrusion Detection Rate	18
3.1.2.2.1	Flowchart of Logistic Regression Algorithm	23
3.1.2.2.2	Flowchart of Decision tree Algorithm	25
3.1.2.2.3	Flowchart of Random Forest Algorithm	26
3.1.2.2.4	Flowchart of KNN algorithm	28
4.1.1.1	Use Case Diagram for the Proposed System	30
4.1.2.1	Class Diagram for the Proposed System	31
4.1.3.1	Sequence Diagram of the Proposed System	32
4.1.4.1	Activity Diagram for the Proposed System	34
4.1.5.1	State Chart Diagram for the Proposed System	36

4.1.6.1	Component Diagram for the Proposed System	38
4.1.7.1	Deployment Diagram for the Proposed System	39
5.3.1	Scatter Plot of the Data Set	62
5.3.2	Data Histograms of the Data Set	63
5.3.3	Output of the Program in Command Prompt	64
5.3.4	MSE Values	64
5.3.5	MAE Values	65
5.3.6	RMSE Values	65
5.3.7	Accuracy Rate of the Algorithm	66
5.3.8	Detection Rate of the Algorithm	66
5.3.9	Running Snort by Configuring Rules	67
5.3.10	Snort Checking for Alerts	67
5.4.1	Accuracy Rate Comparison of the Algorithms	68

LIST OF TABLES

TABLE NUMBER	TABLE NAME	PAGE NUMBER
3.1.2.1.1	Basic Features of individual TCP Connections	19
3.1.2.1.2	Content Features within a connection suggested by domain knowledge	20
3.1.2.1.3	Traffic features computed using a two second time window	21
5.4.1	Experimental Analysis Result of the Proposed System	68

TABLE OF CONTENTS

Abstract **i**

List of Figures **ii**

List of Tables **iv**

1. INTRODUCTION 1

1.1 Introduction 1

1.2 Motivation for the work 5

1.3 Problem Statement 5

2. LITERATURE SURVEY 7

2.1 Network Intrusion Detection System Based On Machine Learning Algorithm 7

2.2 Detecting web based Ddos attack using attack using mapreduce operations in cloud computing environment 8

2.3 A Pattern Recognition Scheme for Distributed Denial of Service (DDOS) Attacks in Wireless 9

2.4 Analyzing Log Files for Post-Mortem Intrusion Detection 9

2.5 Network Traffic Analysis and Intrusion Detection using Packet Analyzer 11

2.6 Existing System 12

3. METHODOLOGY 13

3.1 Proposed System 13

3.1.1 Architecture of the System 16

3.1.2 Implementation of the System 17

3.1.2.1 DataSet Details 17

3.1.2.2 Implementation of Machine Learning Algorithms 21

4. DESIGN 29

4.1 UML Diagrams 29

4.1.1 Use Case Diagram 30

4.1.2 Class Diagram 31

4.1.3 Sequence Diagram 31
4.1.4 Activity Diagram 33
4.1.5 State Chart Diagram 35
4.1.6 Component Diagram 36
4.1.7 Deployment Diagram 38
5. EXPERIMENTAL ANALYSIS AND RESULTS 40
5.1 System Configurations 40
5.1.1 Software Requirements 40
5.1.2 Hardware Requirements 46
5.2 Source Code 47
5.3 Screen Shots 62
5.4 Experimental Analysis/Testing 68
6. CONCLUSION AND FUTURE SCOPE 69
6.1 Conclusion 69
6.2 Future Scope 69
REFERENCES 70

1. INTRODUCTION

1.1 Introduction

An Intrusion Detection System (IDS) is a device or software application that monitors a network or systems for malicious activity or policy violations. Any intrusion activity or violation is typically reported either to an administrator or collected centrally using a security information and event management (SIEM) system. A SIEM system combines outputs from multiple sources and uses alarm filtering techniques to distinguish malicious activity from false alarms.

Although Intrusion Detection Systems monitor networks for potentially malicious activity, they are also disposed to false alarms. Hence, organizations need to fine-tune their IDS products when they first install them. It means properly setting up the intrusion detection systems to recognize what normal traffic on the network looks like as compared to malicious activity.

Classification of Intrusion Detection System

IDS are classified into 5 types

1. **Network Intrusion Detection System (NIDS)**

   Network intrusion detection systems (NIDS) are set up at a planned point within the network to examine traffic from all devices on the network. It performs an observation of passing traffic on the entire subnet and matches the traffic that is passed on the subnets to the collection of known attacks. Once an attack is identified or abnormal behavior is observed, the alert can be sent to the administrator. An example of an NIDS is installing it on the subnet where firewalls are located in order to see if someone is trying crack the firewall.

2. **Host Intrusion Detection System (HIDS)**

   Host intrusion detection systems (HIDS) run on independent hosts or devices on the network. A HIDS monitors the incoming and outgoing packets from the device only and will alert the administrator if suspicious or malicious activity is detected. It takes a snapshot of existing system files and compares it with the previous snapshot. If the analytical system files

were edited or deleted, an alert is sent to the administrator to investigate. An example of HIDS usage can be seen on mission critical machines, which are not expected to change their layout.

3. **Protocol-based Intrusion Detection System (PIDS)**

Protocol-based intrusion detection system (PIDS) comprises of a system or agent that would consistently resides at the front end of a server, controlling and interpreting the protocol between a user/device and the server. It is trying to secure the web server by regularly monitoring the HTTPS protocol stream and accept the related HTTP protocol. As HTTPS is un-encrypted and before instantly entering its web presentation layer then this system would need to reside in this interface, between to use the HTTPS.

4. **Application Protocol-based Intrusion Detection System (APIDS)**

Application Protocol-based Intrusion Detection System (APIDS) is a system or agent that generally resides within a group of servers. It identifies the intrusions by monitoring and interpreting the communication on application specific protocols. For example, this would monitor the SQL protocol explicit to the middleware as it transacts with the database in the web server.

5. **Hybrid Intrusion Detection System**

Hybrid intrusion detection system is made by the combination of two or more approaches of the intrusion detection system. In the hybrid intrusion detection system, host agent or system data is combined with network information to develop a complete view of the network system. Hybrid intrusion detection system is more effective in comparison to the other intrusion detection system. Prelude is an example of Hybrid IDS.

Detection Method of IDS

1. **Signature-based Method**

Signature-based IDS detects the attacks on the basis of the specific patterns such as number of bytes or number of 1's or number of 0's in the network traffic. It also detects on the basis of the already known malicious instruction sequence that is used by the malware. The detected patterns in the IDS are known as signatures.

Signature-based IDS can easily detect the attacks whose pattern (signature) already exists in system but it is quite difficult to detect the new malware attacks as their pattern (signature) is not known.

2. Anomaly-based Method

Anomaly-based IDS was introduced to detect the unknown malware attacks as new malware are developed rapidly. In anomaly-based IDS there is use of machine learning to create a trustful activity model and anything coming is compared with that model and it is declared suspicious if it is not found in model. Machine learning based method has a better generalized property in comparison to signature-based IDS as these models can be trained according to the applications and hardware configurations.

Comparison of IDS with Firewalls

IDS and firewall both are related to the network security but an IDS differs from a firewall as a firewall looks outwardly for intrusions in order to stop them from happening. Firewalls restrict access between networks to prevent intrusion and if an attack is from inside the network it don't signal. An IDS describes a suspected intrusion once it has happened and then signals an alarm.

Need for IDS

Building a reliable network is a very difficult task considering all different possible types of attacks. Nowadays, computer networks and their services are widely used in industry, business, and all arenas of life. Security personnel and everyone who has a responsibility for providing protection for a network and its users, have serious concerns about intruder attacks.

Network administrators and security officers try to provide a protected environment for user's accounts, network resources, personal files and passwords. Attackers may behave in two ways to carry out their attacks on networks; one of these ways is to make a network service unavailable for users or violating personal information. Denial of service (DoS) is one of the most frequent cases representing attacks on network resources and making network services unavailable for their users. There are many types of DoS attacks, and every type has it is own behavior on consuming network resources to achieve the intruder's aim, which is to render the network unavailable for its users. Remote to user (R2L) is one type of computer network attacks, in which an intruder sends set of packets to another computer or server over a network where he/she does not have permission to access as a local user. User to root attacks (U2R) is a second type of attack where the intruder tries to access the network resources as a normal user, and after several attempts, the intruder becomes as a full access user. Probing is a third type of attack in

which the intruder scans network devices to determine weakness in topology design or some opened ports and then use them in the future for illegal access to personal information. There are many examples that represent probing over a network, such as nmap, portsweep, ipsweep.

IDS becomes an essential part for building computer network to capture these kinds of attacks in early stages, because IDS works against all intruder attacks. IDS uses classification techniques to make decision about every packet pass through the network whether it is a normal packet or an attack (i.e. DOS, U2R, R2L, PROBE) packet. Software to detect network intrusions protects a computer network from unauthorized users, including perhaps insiders. The intrusion detector learning task is to build a predictive model (i.e. a classifier) capable of distinguishing between ``bad'' connections, called intrusions or attacks, and ``good'' normal connections.

Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. They operated the LAN as if it were a true Air Force environment, but peppered it with multiple attacks.The raw training data was about four gigabytes of compressed binary TCP dump data from seven weeks of network traffic. This was processed into about five million connection records. Similarly, the two weeks of test data yielded around two million connection records.

A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol. Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes.

Different classes of Attacks

- Denial of Service (DoS)

  An attacker tries to prevent legitimate users from using a service. For example, SYN flood, Smurf and teardrop.

- User to Root (U2R)

An attacker has local access to the victim machine and tries to gain super-user privilege. For example, buffer overflow attacks.

- Remote to Local (R2L)

An attacker tries to gain access to victim machine without having an account on it. For example, password guessing attack.

- Probe

An attacker tries to gain information about the target host. For example, port-scan and ping-sweep.

1.2 Motivation for the Work

Motivation for the work is to propose a security system, which detects malicious behaviors launched toward a system at SC level. The IDS uses data mining approaches namely Decision Tree, Random Forest, Logistic Regression and KNN is used to identify attack. The attack features are learned by the machine learning algorithm.

The contributions of proposed work are: 1) Identifying attack class by applying machine learning algorithm, 2) Identifying which algorithm is best suitable for IDS problem to effectively resist insider attack.

Intrusion detection system uses classification techniques to make decision about every packet pass through the network whether it is a normal packet or an attack. Our objective is to classify the attack into multiple attack types namely DOS, U2R, R2L, PROBE packet.

1.3 Problem Statement

Intrusion detection begins where the firewall ends. Preventing unauthorized entry is best, but not always possible. It is important that the system is reliable and accurate and secure. Intrusion detection is defined as real-time monitoring and analysis of network activity and data for potential Vulnerabilities and attacks in progress.

One major limitation of current intrusion detection system (IDS) technologies is the requirement to filter false alarms. IDS is defined as a system that tries to detect and alert of attempted intrusions into a system or a network. IDSs are classified into two major approaches. Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices. Intrusion prevention is the process of performing intrusion detection and attempting to stop detected possible incidents.

Intrusion Detection and Prevention Systems (IDPS) are primarily focused on identifying possible incidents, logging information about them, attempting to stop them, and reporting them to security administrators. In addition, organizations use IDPSs for other purposes, such as identifying problems with security policies, documenting existing threats and deterring individuals from violating security policies. IDPSs have become a necessary addition to the security infrastructure of nearly every organization. Thus there is necessary to propose a strong detection mechanism to identify attacks.

2. LITERATURE SURVEY

2.1 Network Intrusion Detection System Based On Machine Learning Algorithms

Vipin, Das & Vijaya, Pathak & Sattvik, Sharma & , Sreevathsan & , MVVNS.Srikanth & Kumar T, Gireesh. (2010), International Journal of Computer Science & Information Technology.

Network and system security is of paramount importance in the present data communication environment. Hackers and intruders can create many successful attempts to cause the crash of the networks and web services by unauthorized intrusion. New threats and associated solutions to prevent these threats are emerging together with the secured system evolution. Intrusion Detection Systems (IDS) are one of these solutions. The main function of Intrusion Detection System is to protect the resources from threats. It analyzes and predicts the behaviours of users, and then these behaviours will be considered an attack or a normal behaviour. We use Rough Set Theory (RST) and Support Vector Machine (SVM) to detect network intrusions. First, packets are captured from the network, RST is used to pre-process the data and reduce the dimensions. The features selected by RST will be sent to SVM model to learn and test respectively. The method is effective to decrease the space density of data. The experiments compare the results with Principal Component Analysis (PCA) and show RST and SVM schema could reduce the false positive rate and increase the accuracy.

Keywords: IDS, RST, SVM, PCA

Techniques Used

Support Vector Machine (SVM) to detect network intrusions. They proposed an intrusion detection method using an SVM based system on a RST to reduce the number of features from 41 to 29. We also compared the performance of Rough set theory (RST) with PCA.

2.2 Detecting web based Ddos attack using mapreduce operations in cloud computing environment

Choi, J & Choi, Chang & Ko, Byeongkyu & Choi, D & Kim, P. (2013), Journal of Internet Services and Information Security. 3. 28-37.

A distributed denial of service attacks are the most serious factor among network security risks in cloud computing environment. This study proposes a method of integration between HTTP GET flooding among DDOS attacks and MapReduce processing for a fast attack detection in cloud computing environment. This method is possible to ensure the availability of the target system for accurate and reliable detection based on HTTP GET flooding. In experiments, the processing time for performance evaluation compares a pattern detection of attack features with the Snort detection. The proposed method is better than Snort detection method in experiment results because processing time of proposed method is shorter with increasing congestion.

Keywords: DDoS Attack, HTTP GET Flooding Attack, Web Security, MapReduce

Techniques used

They proposed integration between HTTP GET flooding among DDOS attacks and MapReduce processing for a fast attack detection in cloud computing environment. This method is possible to ensure the availability of the target system for accurate and reliable detection based on HTTP GET flooding.

Detection of DDoS attack analyzes to check the normal state of each network. Next step is a definition of parameter for network attack analysis. Final Step is a definition of threshold by parameter of normal state. Analyzed parameter is classified CPU usage, Load, packet size, information distribution of packet header, protocol distribution for classification distribution of network service, the maximum value, minimum value of traffic, monitoring of flow using spoofing address and so on. The traffic analysis through combination of parameters can be improved.

2.3 A Pattern Recognition Scheme for Distributed Denial of Service (DDoS) Attacks in Wireless

Sensor Networks, Baig, Zubair & Baqer, M & Khan, Asad. (2006), 1050 - 1054. 10.1109/ICPR.2006.147.

This paper defines distinct attack patterns depicting Distributed Denial of Service (DDoS) attacks against target nodes within wireless sensor networks for three most commonly used network topologies. A Graph Neuron (GN)-based, decentralized pattern recognition scheme is proposed for attack detection. The scheme does analysis of internal traffic flow of the network for DDoS attack patterns. We stipulate that the attack patterns depend on both the current energy levels, as well as the energy consumption rates of individual target nodes. The results of varying pattern update rates on the pattern recognition accuracies for the three network topologies are included in the end to test the effectiveness of our implementation.

Techniques used

The Graph Neuron (GN) is an in-network, distributed, pattern recognition algorithm which can form an associative memory overlay on the physical sensor network by interconnecting sensor nodes in a graph-like structure called the GN array. The Graph Neuron (GN) as a light-weight pattern recognition application was used to detect DDoS attack patterns inWSNs.

2.4 Analyzing Log Files for Post-mortem Intrusion Detection

Gamboa, Karen & Monroy, Raúl & Trejo, Luis & Aguirre Bermúdez, Eduardo & Mex-Perera, Carlos. (2012), IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews). 42. 10.1109/TSMCC.2012.2217325.

Upon an intrusion, security staff must analyze the IT system that has been compromised, in order to determine how the attacker gained access to it, and what he did afterward. Usually,

this analysis reveals that the attacker has run an exploit that takes advantage of a system vulnerability. Pinpointing, in a given log file, the execution of one such an exploit, if any, is very valuable for computer security. This is both because it speeds up the process of gathering evidence of the intrusion, and because it helps taking measures to prevent a further intrusion, e.g., by building and applying an appropriate attack signature for intrusion detection system maintenance. This problem, which we call postmortem intrusion detection, is fairly complex, given both the overwhelming length of a standard log file, and the difficulty of identifying exactly where the intrusion has occurred. A novel approach proposed for postmortem intrusion detection, which factors out repetitive behavior, thus, speeding up the process of locating the execution of an exploit, if any. Central to our intrusion detection mechanism is a classifier, which separates abnormal behavior from normal one. This classifier is built upon a method that combines a hidden Markov model with k-means.

Keywords

Anomaly, hidden Markov model (HMM), host-based intrusion detection, postmortem intrusion detection, sequitur.

Techniques used

Postmortem intrusion detection is of interest to two important information security activities. First, it is crucial to the development and maintenance of an attack-signature database, a central component of most commercial IDS's and, second, it might be crucial to a successful subsequent computer forensics analysis, because it enables the computer forensics scientist to focus only on gathering system activity related to an intrusion. Postmortem intrusion detection is very valuable for computer postmortems, because it speeds up the process of gathering evidence of an intrusion. In addition, it speeds up the process of building an attack signature.

2.5 Network Traffic Analysis and Intrusion Detection Using Packet Sniffer

Qadeer, Mohammed & Iqbal, Arshad & Zahid, Mohammad & Siddiqui, Misbahur, Communication Software and Networks, International Conference on. 313-317. 10.1109/ICCSN.2010.104.

Computer software that can intercept and log traffic passing over a digital network or part of a network is better known as packet sniffer. The sniffer captures these packets by setting the NIC card in the promiscuous mode and eventually decodes them. The decoded information can be used in any way depending upon the intention of the person concerned who decodes the data. Depending on the network structure one can sniff all or just parts of the traffic from a single machine within the network. However, there are some methods to avoid traffic narrowing by switches to gain access to traffic from other systems on the network. Packet sniffer and its working is considered, development of the tool on Linux platform and its use for Intrusion Detection. It also discusses ways to detect the presence of such software on the network and to handle them in an efficient way. Focus has also been laid to analyze the bottleneck scenario arising in the network, using this self developed packet sniffer. Before the development of this indigenous software, minute observation has been made on the working behavior of already existing sniffer software such as wireshark (formerly known as ethereal), tcpdump, and snort, which serve as the base for the development of our sniffer software. For the capture of the packets, a library known as libpcap has been used. The development of such software gives a chance to the developer to incorporate the additional features that are not in the existing one.

Keywords

Packet capture, traffic analysis, libpcap, network monitoring, NIC, promiscuous mode, Berkeley Packet Filter, Network analyzer, packet sniffer, intrusion detection.

Techniques used

Address Resolution Protocol (ARP) is used for packet sniffer has been designed as a solution to many network problems. Sniffers are very hard to detect due to its passiveness but there is always a way, and some of them.

2.6 Existing System

Computer forensics science, which views computer systems as crime scenes, aims to identify, preserve, recover, analyze, and present facts and opinions on information collected for a security event. It analyzes what attackers have done such as spreading computer viruses, malwares, and malicious codes and conducting DDoS attacks.

A developer used a self-developed packet sniffer to collect network packets with which to discriminate network attacks with the help of network states and packet distribution. O' Shaughnessy and Gray acquired network intrusion and attack patterns from system log files. These files contain traces of computer misuse. It means that, from synthetically generated log files, these traces or patterns of misuse can be more accurately reproduced. Wu and Banzhaf overviewed research progress of applying methods of computational intelligence, including artificial neural networks, fuzzy systems, evolutionary computation, artificial immune systems, and swarm intelligence, to detect malicious behaviors. The authors systematically summarized and compared different intrusion detection methods, thus allowing us to clearly view those existing research challenges. These aforementioned techniques and applications truly contribute to network security.

In existing work, a security system, which collects forensic features for users at command level rather than at SC level, by invoking data mining and forensic techniques, was developed. Moreover, if attackers use many sessions to issue attacks, e.g., multistage attacks, or launch DDoS attacks, then it is not easy for that system to identify attack patterns. Hu et al. presented an intelligent lightweight IDS that utilizes a forensic technique to profile user behaviors and a data mining technique to carry out cooperative attacks. The authors claimed that the system could detect intrusions effectively and efficiently in real time. However, they did not mention the SC filter.

Giffin et al. provided another example of integrating computer forensics with a knowledge-based system. The system adopts a predefined model, which, allowing SC-sequences to be normally executed, is employed by a detection system to restrict program execution to ensure the security of the protected system. This is helpful in detecting applications that issue a

series of malicious SCs and identifying attack sequences having been collected in knowledge bases. When an undetected attack is presented, the system frequently finds the attack sequence in 2 s as its computation overhead. Fiore et al. explored the effectiveness of a detection approach based on machine learning using the Discriminative Restricted Boltzmann Machine to combine the expressive power of generative models with good classification accuracy capabilities to infer part of its knowledge from incomplete training data so that the network anomaly detection scheme can provide an adequate degree of protection from both external and internal menaces. Faisal et al. analyzed the possibility of using data stream mining to enhance the security of advanced metering infrastructure through an IDS. The advanced metering infrastructure, which is one of the most crucial components of smart card, serves as a bridge for providing bidirectional information flow between the user domain and the utility domain. The authors treat an IDS as a second-line security measure after the first line of primary advanced metering infrastructure security techniques such as encryption, authorization, and authentication.

Disadvantages

Most intrusion detection techniques focus on how to find malicious network behaviors and acquire the characteristics of attack packets, i.e., attack patterns, based on the histories recorded in log files. Real time attack detection at incoming rate is still a challenging.

Existing techniques cannot easily authenticate remote-login users and detect specific types of intrusions, e.g., when an unauthorized user logs in to a system with a valid user ID and password.

3. METHODOLOGY

3.1 Proposed System

Proposed smart intrusion detection system (IDS) is viewed as an effective solution for network security and protection against external threats. However, the existing IDS often has a lower detection rate under new attacks and has a high overhead when working with audit data, and thus machine learning methods have been widely applied in intrusion detection.

In our proposed method, Decision Tree, Logistic regression, Random Forest and KNN is developed as learning methods in solving the classification problem of pattern recognition and intrusion identification.

Compared with other classification algorithms, Decision Tree, Logistic regression, Random Forest and KNN can better solve the problems of small samples, nonlinearity and high dimensionality.

Advantages

- High accuracy on detection rate.
- High True positive rate.

True Positive

A legitimate attack which triggers to produce an alarm.

False Positive

An event signaling to produce an alarm when no attack has taken place.

False Negative

When no alarm is raised when an attack has taken place.

True Negative

An event when no attack has taken place and no detection is made.

$$\text{Accuracy} = \frac{\text{True positives} + \text{False negatives}}{\text{Total number of samples}}$$

IDS are typically evaluated based on the following standard performance measures:

- True Positive Rate (TPR)

  It is calculated as the ratio between the number of correctly predicted attacks and the total number of attacks. If all intrusions are detected then the TPR is 1 which is extremely rare for an IDS. TPR is also called Detection Rate (DR) or the Sensitivity.

- False Positive Rate (FPR)

  It is calculated as the ratio between the number of normal instances incorrectly classified as an attack and the total number of normal instances.

- False Negative Rate (FNR)

  False Negative means when a detector fails to identify an anomaly and classifies it as normal.

- Classification Rate (CR) or Accuracy

  The CR measures how accurate the IDS is in detecting normal or anomalous traffic behavior. It is described as the percentage of all those correctly predicted instances to all the instances.

3.1.1 Architecture of the System

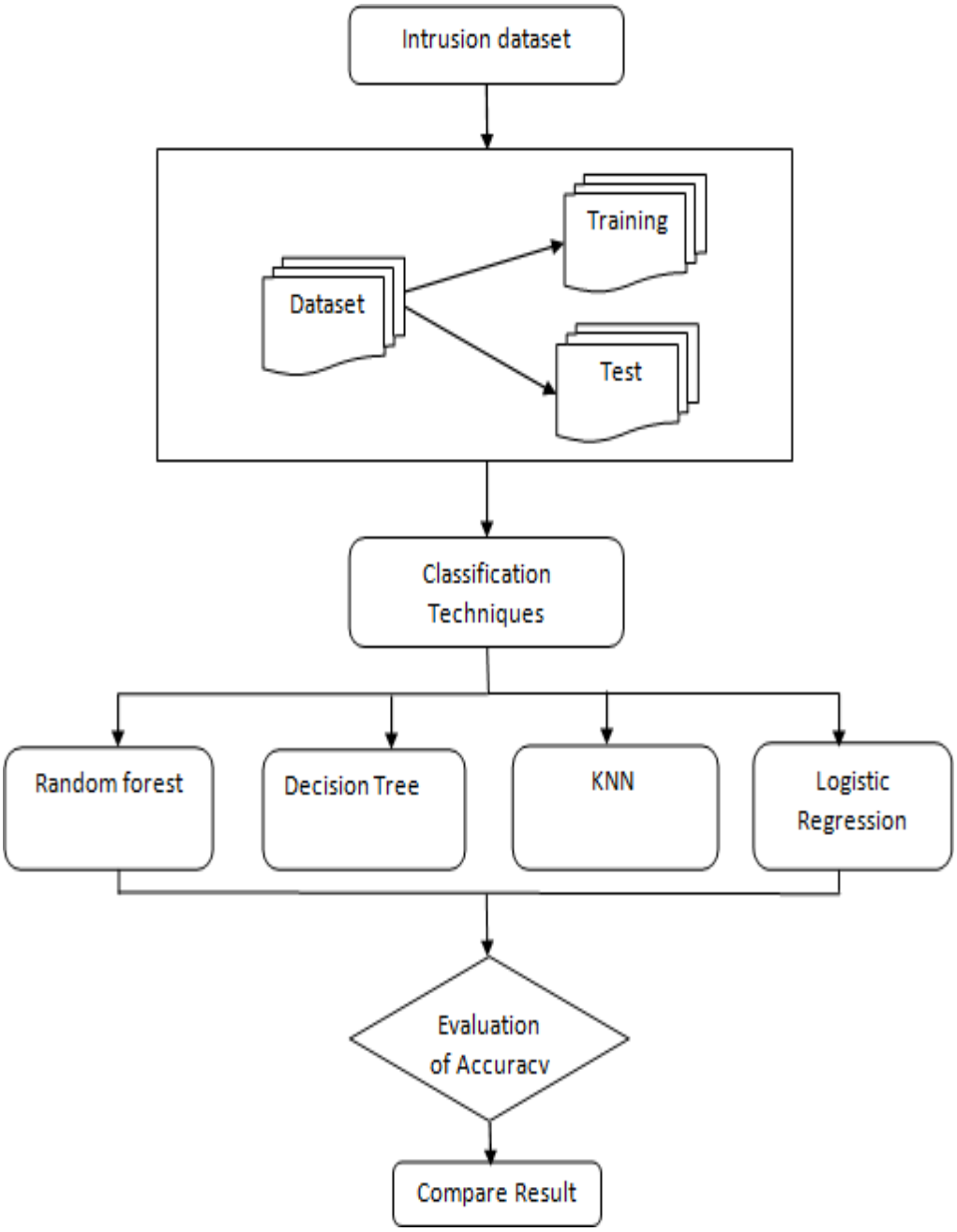


Fig 3.1.1.1 System Architecture

3.1.2 Implementation of the System

The proposed work is implemented in Python 3.6.4 with libraries scikit-learn, pandas, matplotlib and other mandatory libraries. We downloaded dataset from kdd.ics.uci.edu. The data downloaded contains train set and test set separately with four different classes of intrusions. The traindataset considered as train set and testdataset considered as test set.

We have collected the dataset for the intrusion detection system with following details from KDD dataset and we applied machine learning algorithm such as decision tree, regression, random forest and KNN.

3.1.2.1 DataSet Details

- **Data collection**

The data collection process involves the selection of quality data for analysis. Here we used KDD intrusion dataset taken from uci.edu for machine learning implementation. The job of a data analyst is to find ways and sources of collecting relevant and comprehensive data, interpreting it, and analyzing results with the help of statistical techniques.

- **Data visualization**

A large amount of information represented in graphic form is easier to understand and analyze. Some companies specify that a data analyst must know how to create slides, diagrams, charts, and templates. In our approach, the detection rates of intrusion are shown as data visualization part.

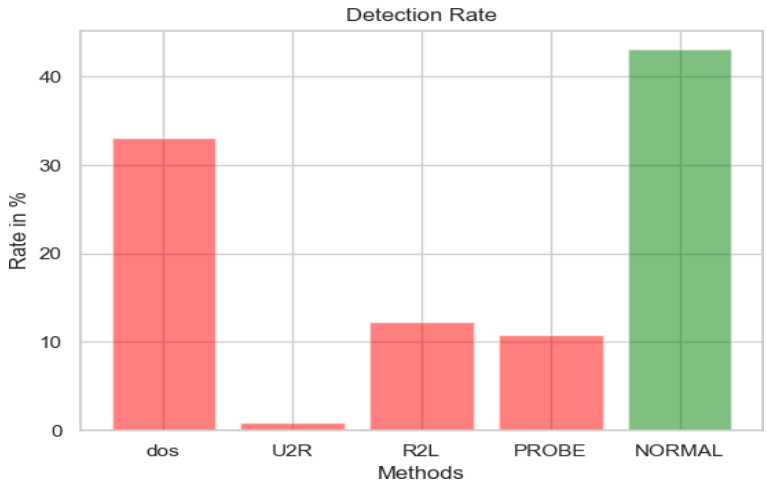


Figure 3.1.2.1.1: Data Visualization of Intrusion Detection rate

- **Data preprocessing**

The purpose of preprocessing is to convert raw data into a form that fits machine learning. Structured and clean data allows a data scientist to get more precise results from an applied machine learning model. The technique includes data formatting, cleaning, and sampling.

- **Dataset splitting**

A dataset used for machine learning should be partitioned into three subsets — training, test, and validation sets.

Training set: A data scientist uses a training set to train a model and define its optimal parameters it has to learn from data.

Test set: A test set is needed for an evaluation of the trained model and its capability for generalization. The latter means a model's ability to identify patterns in new unseen data after having been trained over a training data. It's crucial to use different subsets for training and testing to avoid model overfitting, which is the incapacity for generalization we mentioned above.

- **Model training**

After a data scientist has preprocessed the collected data and split it into train and test can proceed with a model training. This process entails “feeding” the algorithm with training data. An algorithm will process data and output a model that is able to find a target value (attribute) in new data an answer you want to get with predictive analysis. The purpose of model training is to develop a model.

- **Model evaluation and testing**

The goal of this step is to develop the simplest model which is able to formulate a target value fast and well enough. A data scientist can achieve this goal through model tuning. That’s the optimization of model parameters to achieve an algorithm’s best performance.

It is important to note that the test data is not from the same probability distribution as the training data, and it includes specific attack types not in the training data. This makes the task more realistic. Some intrusion experts believe that most novel attacks are variants of known attacks and the "signature" of known attacks can be sufficient to catch novel variants. The datasets contain a total of 24 training attack types, with an additional 14 types in the test data only. The dataset variable names are described below:

Feature name	***Description***	***type***
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong" fragments	Continuous

Table 3.1.2.1.1: Basic features of individual TCP connections.

Feature name	*Description*	*Type*
hot	number of ``hot'' indicators	Continuous
num_failed_logins	number of failed login attempts	Continuous
logged_in	1 if successfully logged in; 0 otherwise	Discrete
num_compromised	number of ``compromised'' conditions	Continuous
root_shell	1 if root shell is obtained; 0 otherwise	Discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	Discrete
num_root	number of ``root'' accesses	Continuous
num_file_creations	number of file creation operations	Continuous
num_shells	number of shell prompts	Continuous
num_access_files	number of operations on access control files	Continuous
num_outbound_cmds	number of outbound commands in an ftp session	Continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	Discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	Discrete

Table 3.1.2.1.2: Content features within a connection suggested by domain knowledge.

Feature name	*Description*	*Type*
feature name	*description*	*Type*
count	number of connections to the same host as the current connection in the past two seconds	Continuous
	Note: The following features refer to these same-host connections.	

serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Table 3.1.2.1.3: Traffic features computed using a two-second time window

3.1.2.2 Implementation of Machine Learning Algorithms

Intrusion Detection is considered with four different machine learning algorithms such as Decision Tree, Logistic Regression, KNN and Random Forest. The predicted value is compared for the predicted accuracy and error values.

A. Logistic Regression Algorithm

Logistic Regression is basically a predictive model analysis technique where the output (target) variables are discrete values for a given set of features or input (X). It is very powerful yet simple supervised classification algorithm in machine learning. Around 60% of the world's classification problems can be solved by using the logistic regression algorithm.

Logistic Regression is one of the most common algorithms used for binary classification. It predicts the probability of occurrence of a binary outcome using a logit function. It is a special case of linear regression as it predicts the probabilities of outcome using log function.

In simple, Linear Regression predicts scores on one variable from the scores on a second variable. The variable that predicted is called Criterion Variable. The variable base for

predictions on is called predictor variable. When there is only one predictor variable, the prediction method is called simple regression.

We use the activation function (sigmoid) to convert the outcome into categorical value. There are many examples where we can use Logistic Regression for example, it can be used for Fraud Detection, Spam Detection, Cancer Detection, etc.

Sigmoid Function

It is a mathematical function having a characteristic that can take any real value and map it to between o to 1 shaped like letter 'S'. The Sigmoid Function is also called Logistic Function.

The Sigmoid Function is more than 0.5 then we classify that label as Class 1 or Positive Class and if it is less than 0.5 then we can classify it to Negative Class or label as Class 0.

Pros:

- Fast
- No tuning required
- Highly interpretable
- Well-understood

Cons:

- Unlikely to produce the best predictive accuracy
- Presumes a linear relationship between the features and response
- If the relationship is highly non-linear as with many scenarios, linear relationship will not effectively model the relationship and its prediction would not be accurate.

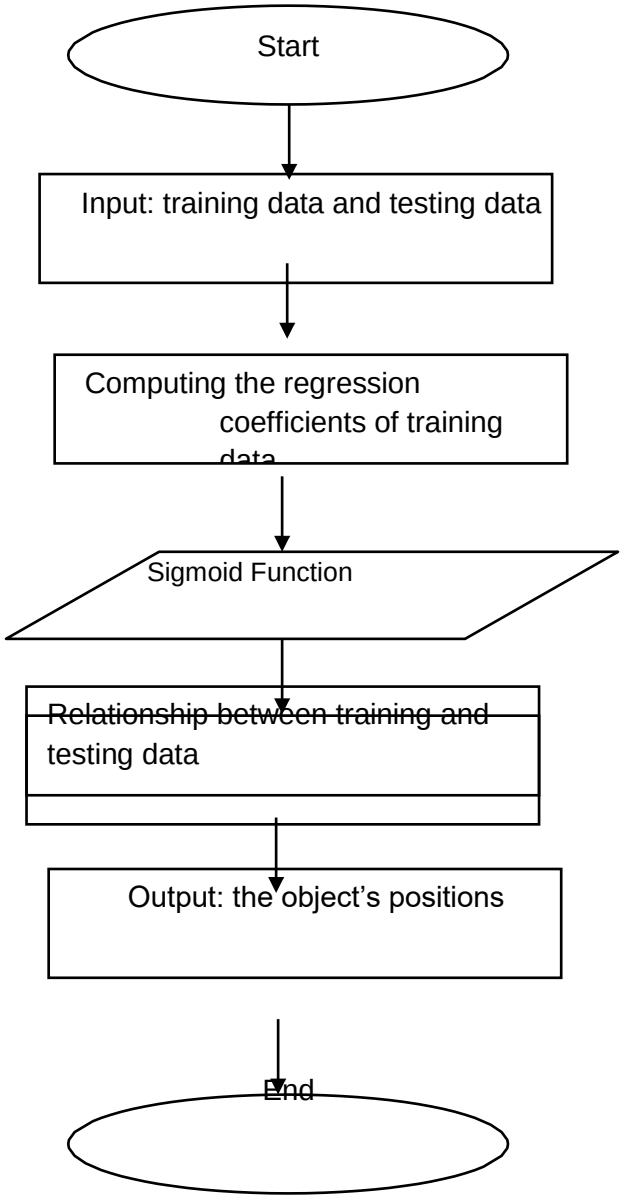


Figure 3.1.2.2.1: Flow chart of Logistic Regression algorithm

B. Decision Tree Algorithm

Decision tree is a type of supervised learning algorithm that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables. In decision tree internal node represents a test on the attribute, branch depicts the outcome and leaf represents decision made after computing attribute.

The general motive of using Decision Tree is to create a training model which can be used to predict class or a value of target variables by learning decision rules inferred from prior data (training data).

The understanding level of Decision Tree algorithm is so easy compared with other classification algorithms. The Decision Tree algorithm tries to solve the problem, by using tree representation. Each internal node of the tree corresponds to an attribute, and each leaf node corresponds to a class label.

Decision Tree works in following manner

- Place the best attribute of the dataset at the root of the tree.
- Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
- Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

In decision trees, for predicting a class label for a record we start from the root of the tree. Then compare the values of the root attribute with record's attribute. On the basis of comparison, follow the branch corresponding to that value and jump to the next node.

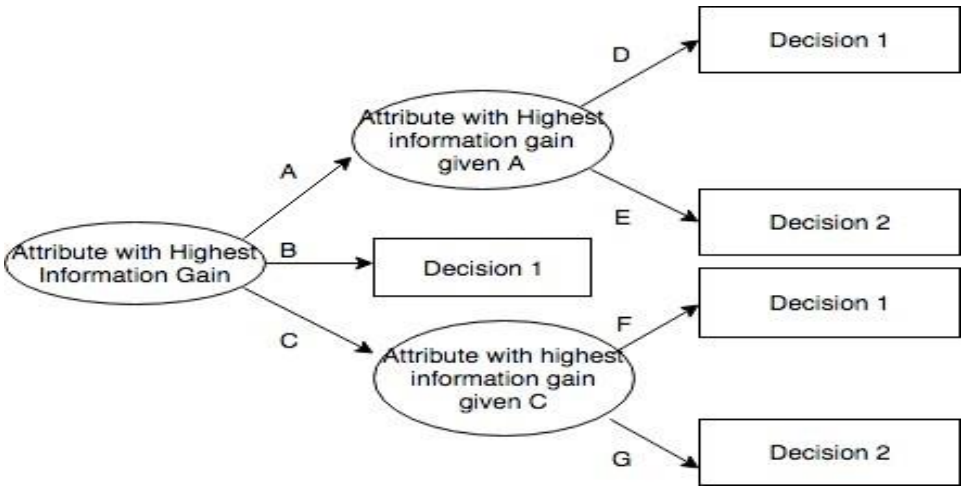


Figure 3.1.2.2.2: Flow chart Decision Tree algorithm

DecisionTreeClassifier is a class capable of performing multi-class classification on a dataset. As with other classifiers, DecisionTreeClassifier takes as input two arrays: an array X, sparse or dense, of size [n_samples, n_features] holding the training samples, and an array Y of integer values, size [n_samples], holding the class labels for the training samples.

Pros

- Prone to overfitting
    - If you have a lot of features
    - Stop growth of tree at the appropriate time
- You can build bigger classifiers out of this.

Cons

- It involves higher time to train the model.
- It is relatively expensive as complexity and time taken is more.

C. Random Forest Algorithm

Given there are n cases in the training dataset. From these n cases, sub-samples are chosen at random with replacement. These random sub-samples chosen from the training dataset are used to build individual trees.

- Assuming there are k variables for input, a number m is chosen such that m < k. m variables are selected randomly out of k variables at each node. The split which is the best of these m variables is chosen to split the node. The value of m is kept unchanged while the forest is grown.
- Each tree is grown as large as possible without pruning.
- The class of the new object is predicted based upon the majority of votes received from the combination of all the decision trees.

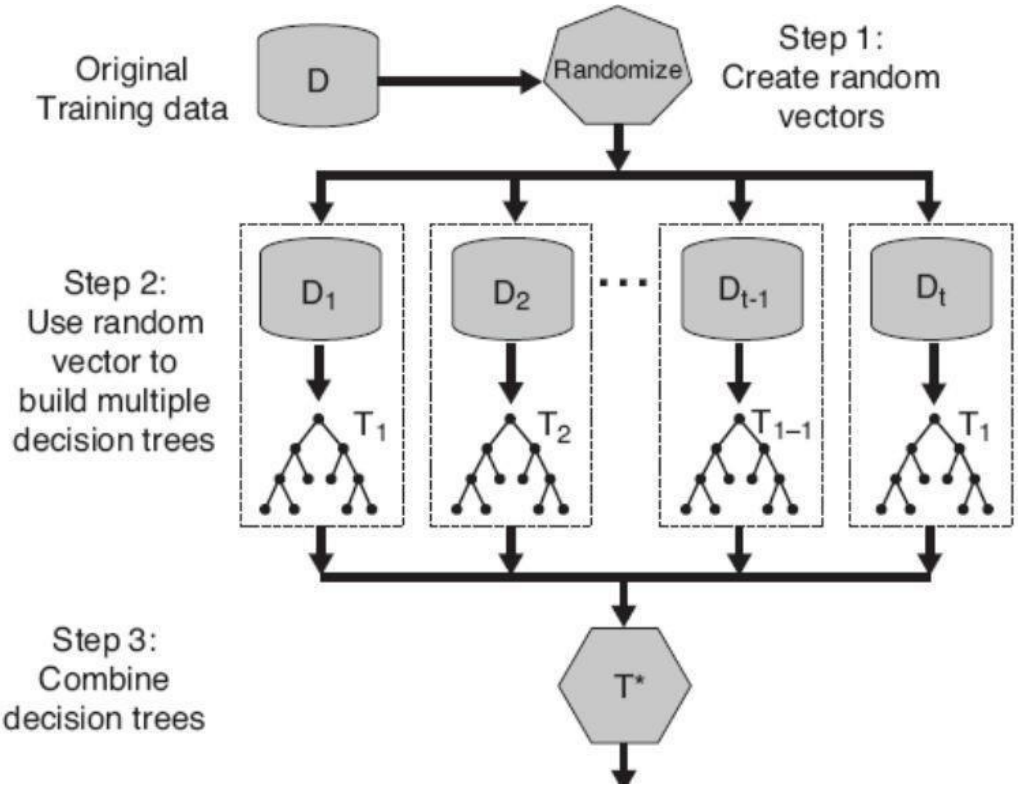


Figure 3.1.2.2.3: Flow chart of Random Forest Algorithm

D. K Nearest Neighbor (KNN) Algorithm

The K Nearest Neighbors (KNN) algorithm measures the distance between a query scenario and a set of scenarios in the data set. We can compute the distance between two scenarios using some distance function d(x,y), where x,y are scenarios composed of N features, such that $x=\{x_1,\ldots,x_N\}$, $y=\{y_1,\ldots,y_N\}$.

The Two distance functions are:

Absolute distance measuring:

$$d_A(x,y) = \sum_{i=1}^{N} |x_i - y_i|$$

Euclidian distance measuring:

$$d_E(x,y) = \sum_{i=1}^{N} \sqrt{x_i^2 - y_i^2}$$

The model for KNN is the entire training dataset. When a prediction is required for a unseen data instance, the KNN algorithm will search through the training dataset for the k-most similar instances. The prediction attribute of the most similar instances is summarized and returned as the prediction for the unseen instance.

The similarity measure is dependent on the type of data. For real-valued data, the Euclidean distance can be used. Other types of data such as categorical or binary data, Hamming distance can be used.

Instance-based algorithms are those algorithms that model the problem using data instances (or rows) in order to make predictive decisions. The KNN algorithm is an extreme form of instance-based methods because all training observations are retained as part of the model.

It is a competitive learning algorithm, because it internally uses competition between model elements (data instances) in order to make a predictive decision. The objective similarity measure between data instances causes each data instance to compete to “win” or be most similar

to a given unseen data instance and contribute to a prediction.

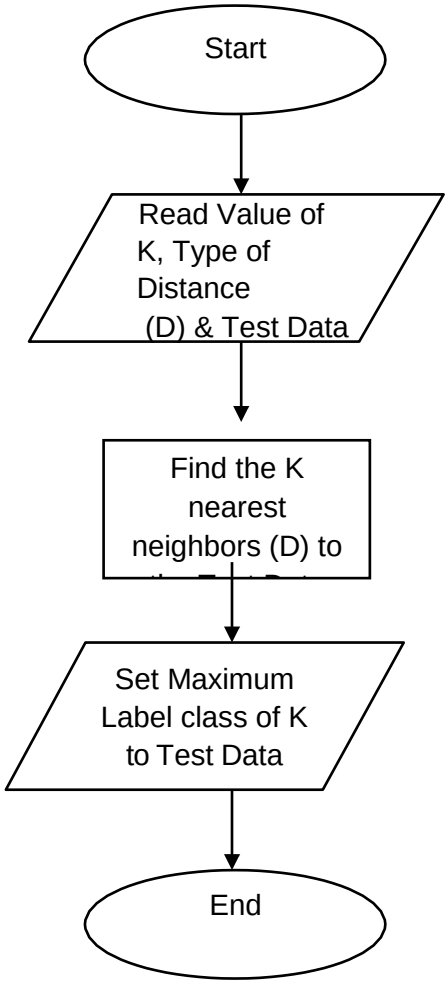


Figure 3.1.2.2.4: Flow chart of KNN Algorithm

4. DESIGN

4.1 UML Diagrams

Unified Modeling Language (UML) is a general purpose modelling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

UML is not a programming language, it is rather a visual language. We use UML diagrams to portray the behavior and structure of a system. UML helps software engineers, businessmen and system architects with modelling, design and analysis. The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997. Its been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.

UML is linked with object oriented design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

1. Structural Diagrams – Capture static aspects or structure of a system. Structural Diagrams include:
   Component Diagrams,
   Object Diagrams,
   Class Diagrams and
   Deployment Diagrams.
2. Behavior Diagrams – Capture dynamic aspects or behavior of the system. Behavior diagrams include:
   Use Case Diagrams,
   State Diagrams,
   Activity Diagrams and
   Interaction Diagrams.

Advantages of UML

- Most-Used and Flexible

UML is a highly recognized and understood platform for software design. It is a standard notation among software developers. You can safely assume that most software professionals will be at least acquainted with, if not well-versed in, UML diagrams, thus making it the go-to alternative to explain software design models.

4.1.1 Use Case Diagram

Use Case Diagram captures the system's functionality and requirements by using actors and use cases. Use Cases model the services, tasks, function that a system needs to perform. Use cases represent high-level functionalities and how a user will handle the system. Use-cases are the core concepts of Unified Modelling language modeling.

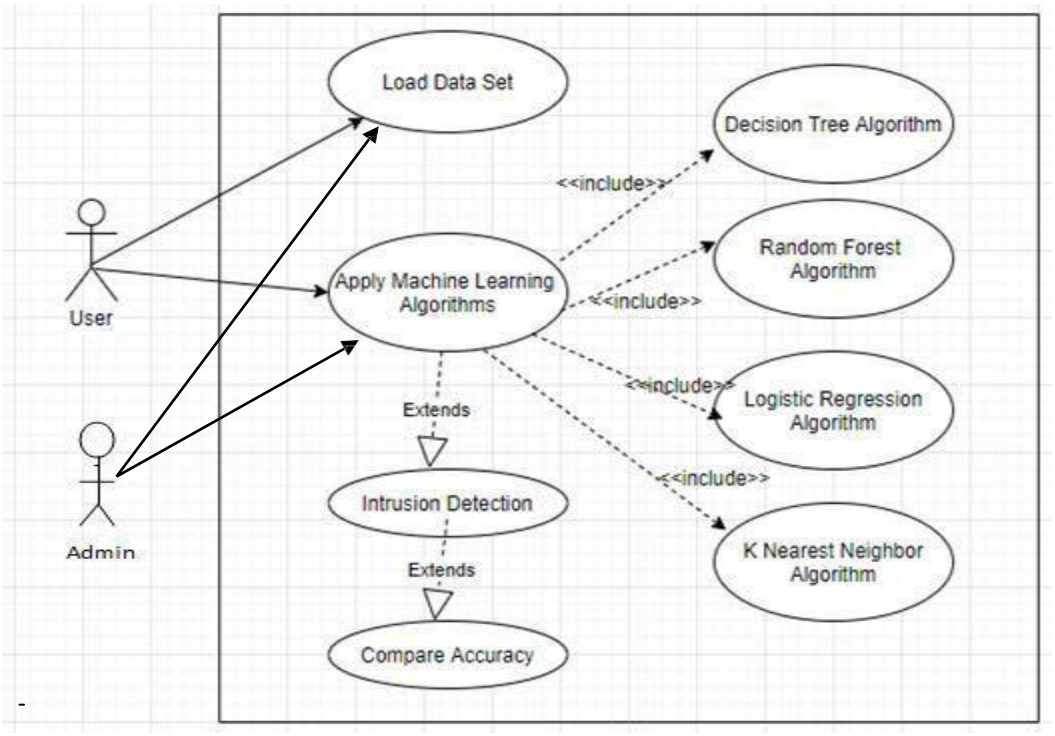


Fig 4.1.1.1 Use Case Diagram for the Proposed System

4.1.2 Class Diagram

Class diagram model class structure and contents using design elements such as classes, packages and objects. Class diagram describes 3 perspectives when designing a system-Conceptual, Specification, Implementation. Classes are composed of three things: name, attributes and operations. Class diagrams also display relations such as containment, inheritance, associations etc. The association relationship is most common relationship in a class diagram. The association shows the relationship between instances of classes.

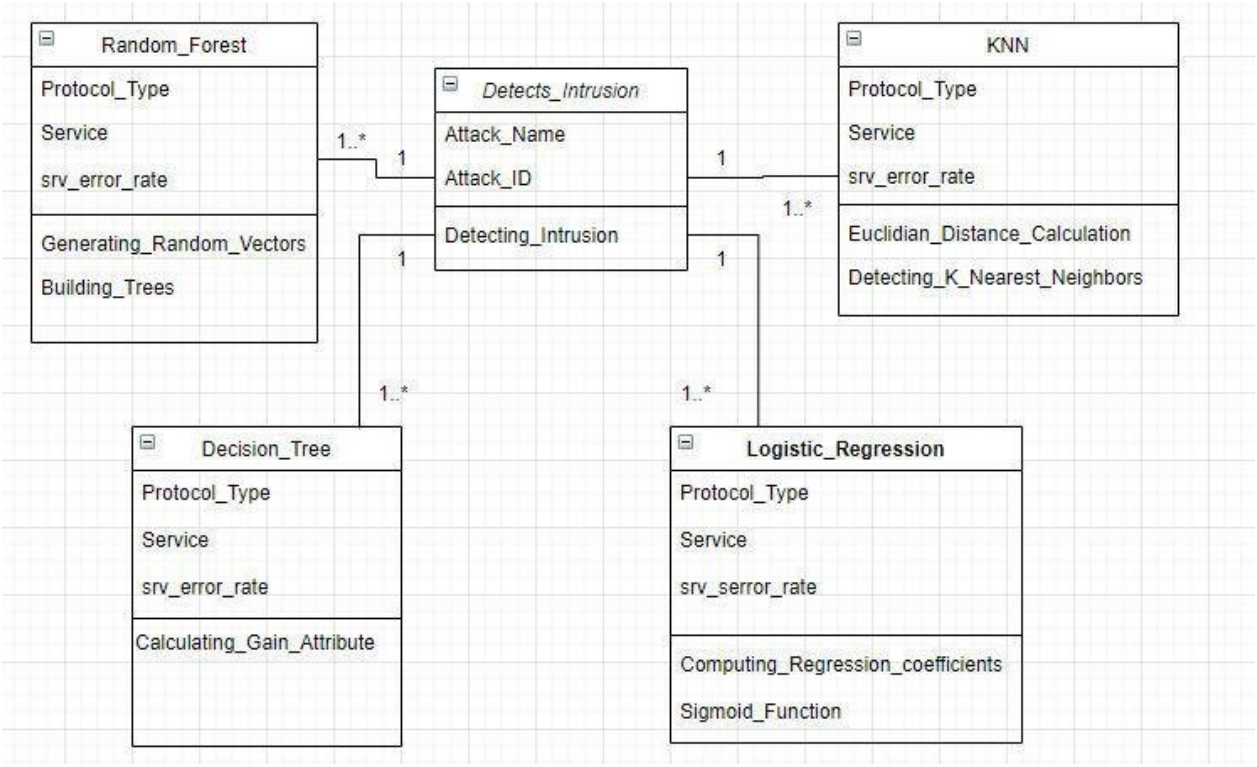


Fig 4.1.2.1 Class Diagram for IDS

4.1.3 Sequence Diagram

Sequence Diagrams display the time sequence of the objects participating in the interaction. This consists of the vertical dimension (time) and horizontal dimension (different objects).

Object: Object can be viewed as an entity at a particular point in time with a specific value and as a holder of identity that has different values over time.

Actor: An Actor represents a coherent set of roles that users of a system play when interacting with the use cases of the system.

Message: A Message is sending of a signal from one sender object to other receiver objects.

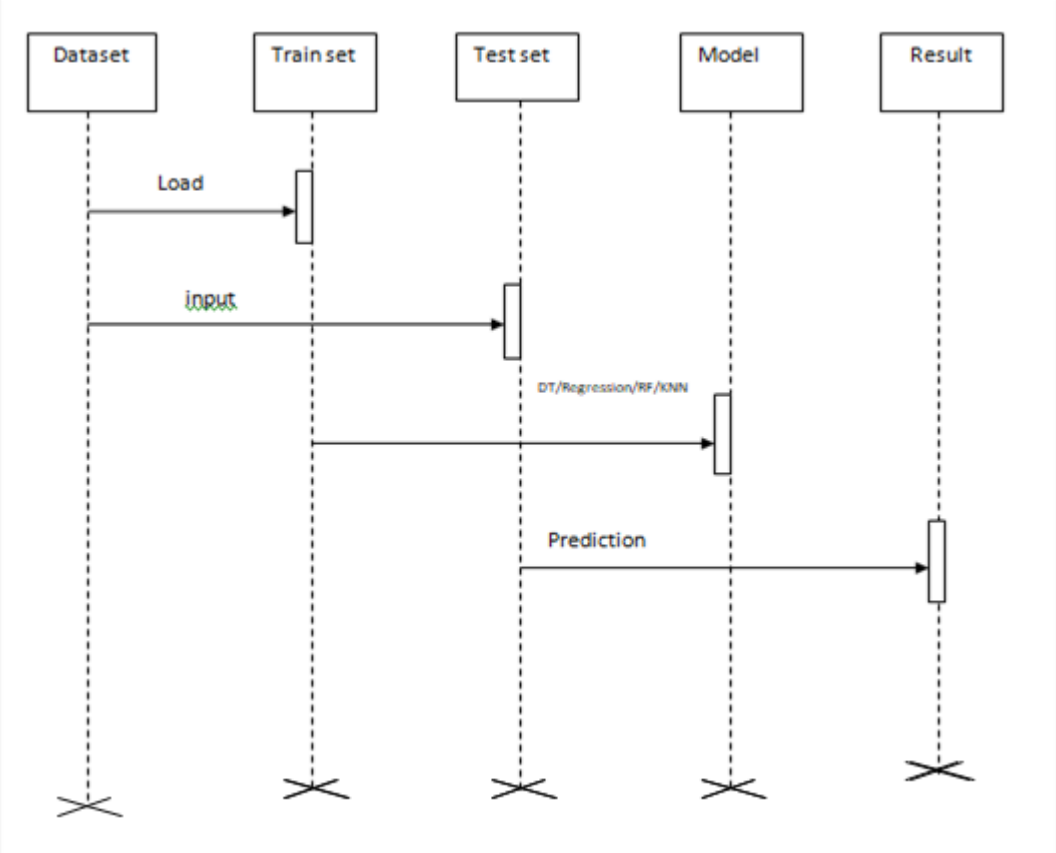


Fig 4.1.3.1 Sequence Diagram of the Proposed System

4.1.4 Activity Diagram

An activity diagram shows the flow from activity to activity. An activity is a going non-atomic execution within a state machine. An activity results in some action, results in a change of state or return of a value.

Activity Diagram commonly contains:

- Activity states and action states.
- Transitions.
- Objects, it may contain nodes and constraint .

Activity states and action states: An executable atomic computation is called action state, which cannot be decomposed. Activity state is non-atomic, decomposable and takes some duration to execute.

Transition: It is a path from one state to the next state, represented as simple directed line.

Branching: When an alternate path exists, branching arises which is represented by open diamond. It has a incoming transition, two or more outgoing transitions.

Forking and Joining: The synchronization bar when split one flow into two or more flows is called fork. When two or more flows are combined at synchronization bar, the bar is called join.

Swim Lanes: Group work flow is called swim lanes. All groups are portioned by vertical solid lines. Each swim lane specifies locus of activities and has a unique name. Each swim lane is implemented by one or more classes. Transition may occur between objects across swim lanes.

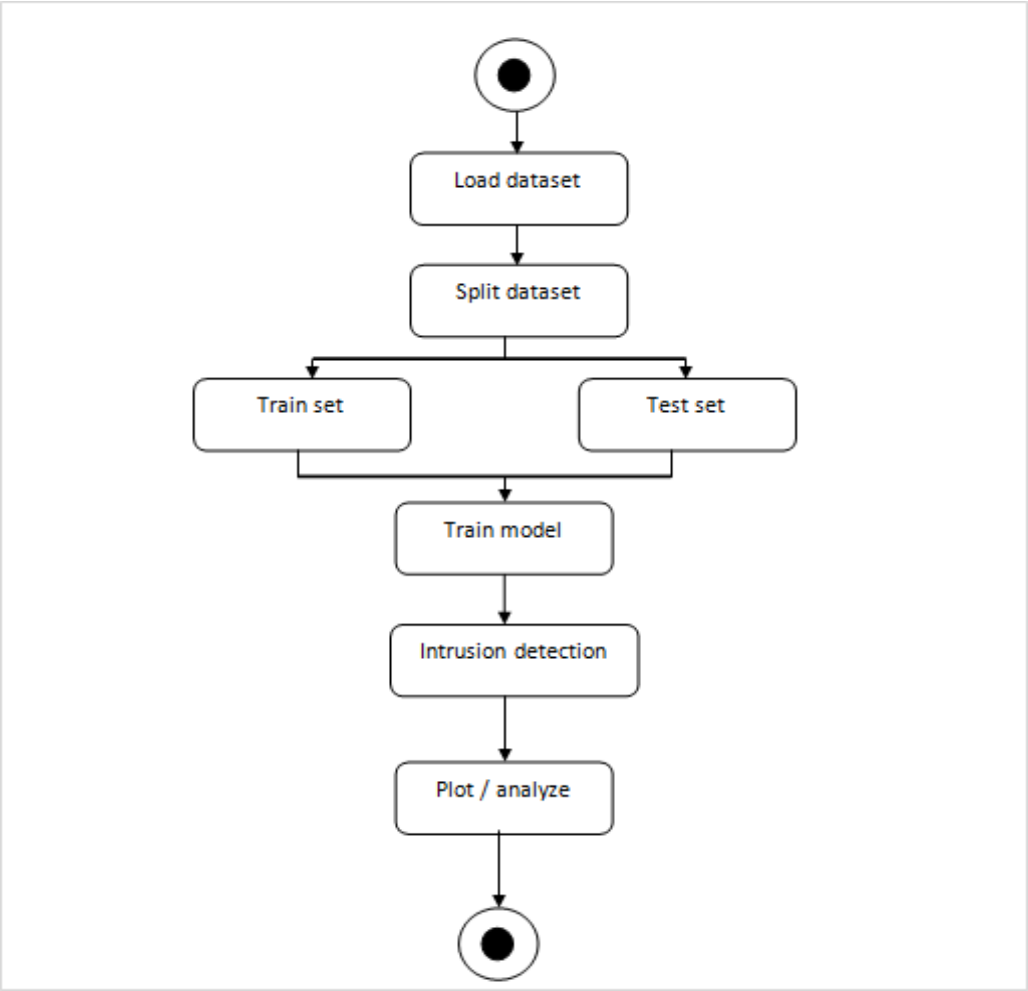


Fig 4.1.4.1 Activity Diagram of the Proposed System

4.1.5 State Chart Diagram

A state chart diagram describes a state machine which shows the behavior of classes. It shows the actual changes in state not processes or commands that create those changes and is the dynamic behavior of objects overtime by modeling the lifecycle of objects of each class.

It describes how an object is changing from one state to another state. There are mainly two states in state chart diagram:

- Initial state
- Final state

Some of the components of State Chart Diagram are:

State: It is a condition or situation in lifecycle of an object during which it satisfies same condition or performs some activity or waits for some event.

Transition: It is a relationship between two states indicating that object in first state performs some action and enters into the nextstate.

Event: An event is specification of significant occurrence that has a location in time and space.

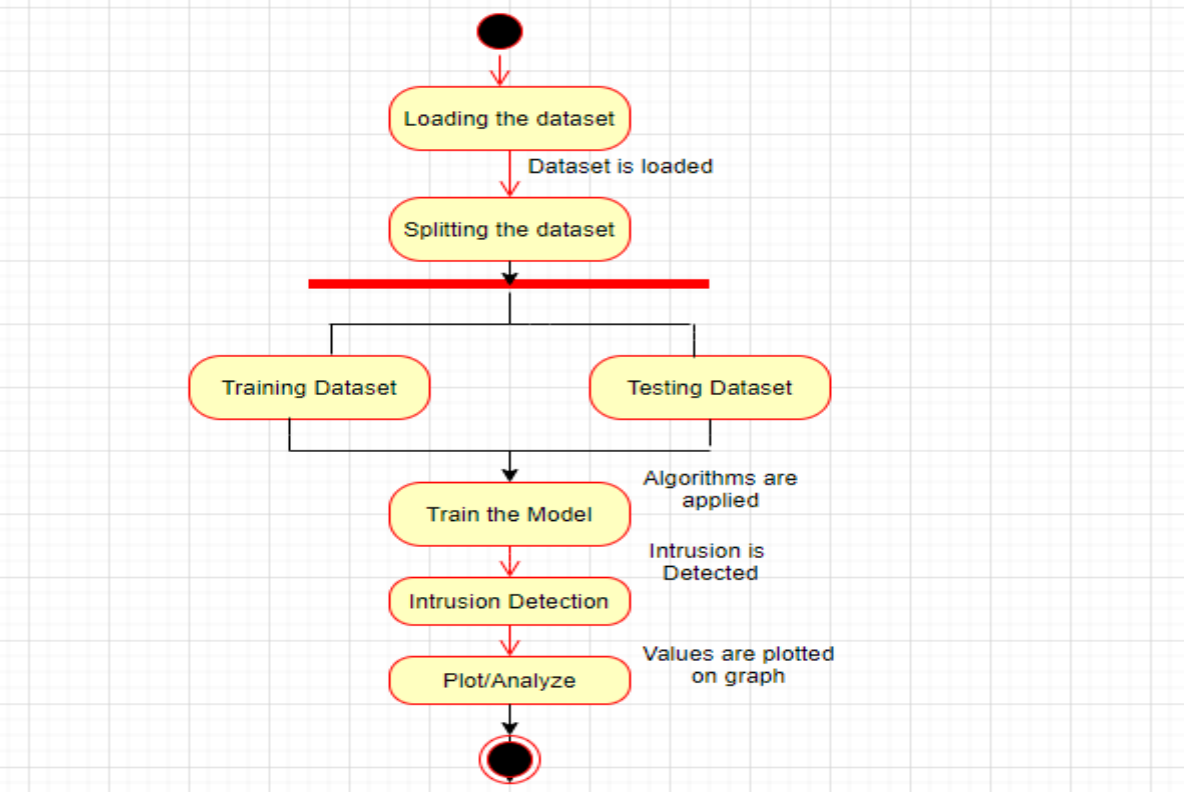


Fig 4.1.5.1 State Chart Diagram for the Proposed System

4.1.6 Component Diagram

Component Diagram describes the organization and wiring of the Physical Components in a system. It can be presented to key project state holders and implementation staff.

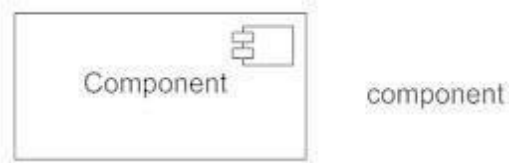
Component Diagram can be used:

- To model the components of a system.
- To model the database schema.
- To model the executability of an application.
- To model the system source code.

Component Diagram Symbols and Notations are as follows:

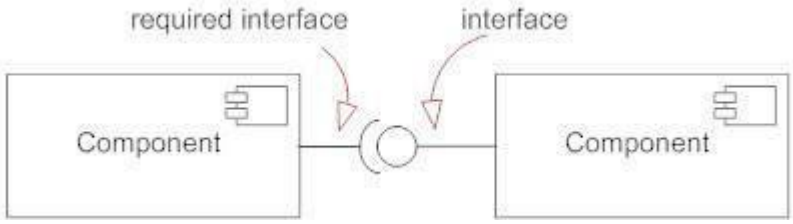
Component:

A component is a logical unit block of the system, a slightly higher abstraction than classes. It is represented as a rectangle with a smaller rectangle in the upper right corner with tabs or the word written above the name of the component to help distinguish it from a class.



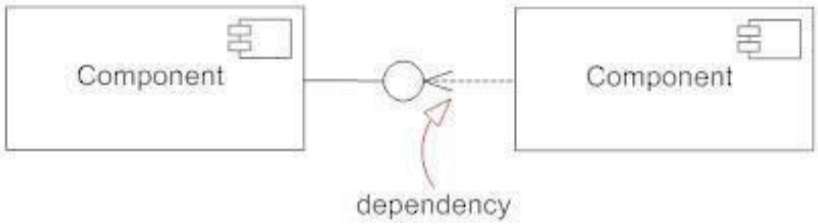
Interface:

An interface (small circle or semi-circle on a stick) describes a group of operations used (required) or created (provided) by components. A full circle represents an interface created or provided by the component. A semi-circle represents a required interface, like a person's input.



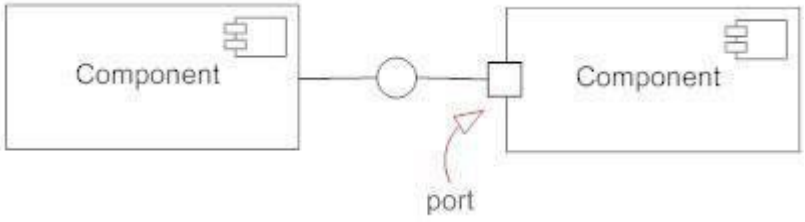
Dependencies:

Draw dependencies among components using dashed arrows.



Port:

Ports are represented using a square along the edge of the system or a component. A port is often used to help expose required and provided interfaces of a component.



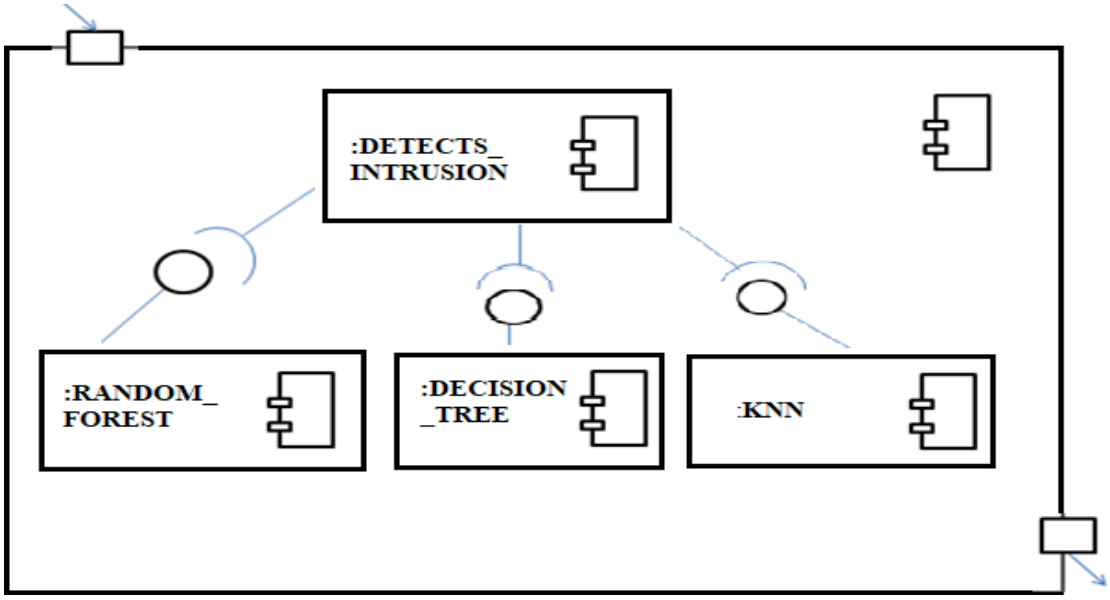


Fig 4.1.6.1 Component Diagram of the Proposed System

4.1.7 Deployment Diagram

Deployment Diagram is used to visualize the Topology of the physical component of a system, where the software components are deployed.

A Deployment Diagram consists of Nodes and their relationships. Deployment Diagram is used to show how they are deployed in hardware.

These are useful for System Engineers. An efficient Deployment Diagram is necessary as it controls the performance, maintainability and portability.

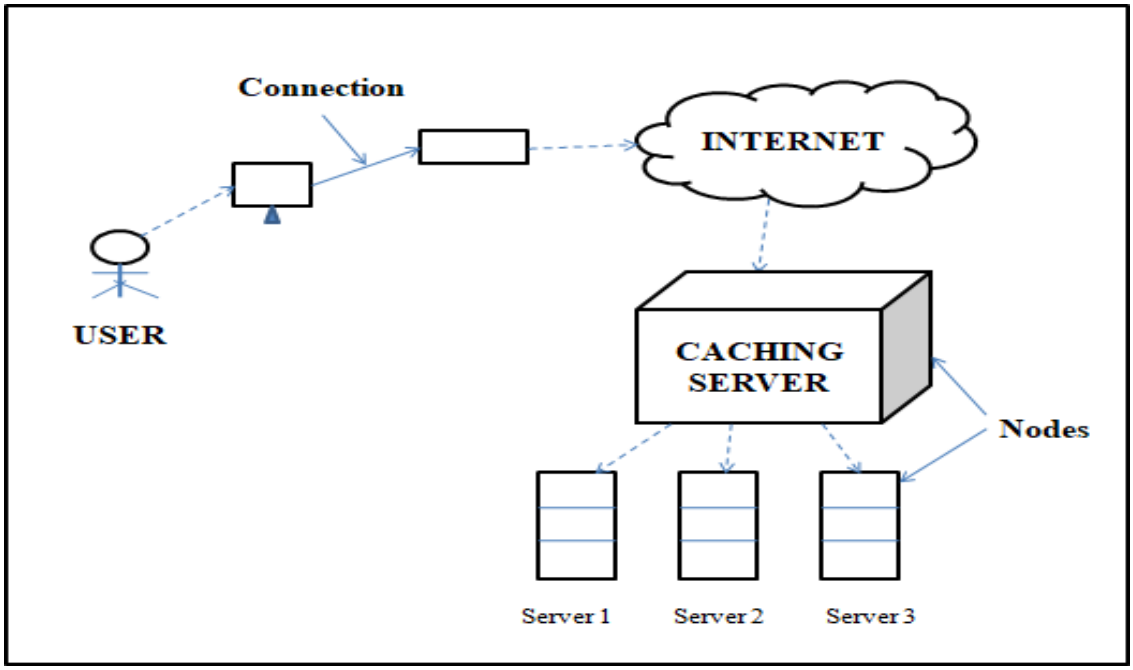


Fig 4.1.7.1 Deployment Diagram for the Proposed System

5. EXPERIMENTAL ANALYSIS AND RESULT

5.1 System Configuration

The system configuration includes Software and Hardware requirement, which are provided below

5.1.1 Software Requirements

Operating System : Windows 7 or higher

Programming : Python 3.6 and related libraries

- **Python**

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

- **Scikit-learn**

Scikit-learn is the simple and efficient tools for data mining and data analysis. Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector

machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Some popular groups of models provided by scikit-learn include:

- Clustering: for grouping unlabeled data such as KMeans.
- Cross Validation: for estimating the performance of supervised models on unseen data.
- Datasets: for test datasets and for generating datasets with specific properties for investigating model behavior.
- Dimensionality Reduction: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
- Ensemble methods: for combining the predictions of multiple supervised models.
- Feature extraction: for defining attributes in image and text data.
- Feature selection: for identifying meaningful attributes from which to create supervised models.
- Parameter Tuning: for getting the most out of supervised models.
- Manifold Learning: For summarizing and depicting complex multi-dimensional data.
- Supervised Models: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

- **Snort**

Snort is a network based intrusion detection system which is written in C programming language. It was developed in 1998 by Martin Roesch. Now it is developed by Cisco. It is a free open source software. It can also be used as a packet sniffer to monitor the system in real time. The network admin can use it to watch all the incoming packets and find the ones which are dangerous to the system. It is based on library packet capture tool. The rules are fairly easy to

create and implement and it can be deployed in any kind on operating system and any kind of network environment. The main reason of popularity of this IDS over other is that it is a free to use software and also open source because of which any user can able to use it as the way he want.

Features:

- Real-time traffic monitor.
- Packet logging.
- Analysis of protocol.
- Content matching.
- OS fingerprinting.
- Can be installed in any network environment.
- Creates logs.
- Open Source.
- Rules are easy to implement.

Snort's open source network-based intrusion detection/prevention system (IDS/IPS) has the ability to perform real-time traffic analysis and packet logging on Internet Protocol (IP) networks. Snort performs protocol analysis, content searching and matching.

The program can also be used to detect probes or attacks, including, but not limited to, operating system fingerprinting attempts, semantic URL attacks, buffer overflows, server message block probes, and stealth port scans.

Snort can be configured in three main modes:

1. Sniffer, 2. Packet Logger and 3. Network Intrusion Detection.

Sniffer Mode

The program will read network packets and display them on the console.

Packet Logger mode

In packet logger mode, the program will log packets to the disk.

Network Intrusion Detection System Mode

In intrusion detection mode, the program will monitor network traffic and analyze it against a rule set defined by the user. The program will then perform a specific action based on what has been identified.

Important Commands in Snort

- Configuring Snort into Sniffer Mode

  ./snort –v → Prints TCP/IP/UDP/ICMP packet headers to the Screen.

  ./snort –vd → Instructs Snort to display the packet data as well as the headers.

- Configuring Snort into Packet Logger Mode

  ./snort –dev –l ./name_of_the_log_directory → Records the packets to the disk.

  ./snort –l ./name_of_ the_log_directory –b → Binary Mode logs the packets into tcpdump format to a single binary file in the logging directory.

- Configuring Snort into Network Intrusion Detection Mode

  ./snort –dev –l ./log –h 192.168.1.0/24 –c snort.conf (snort.conf is the snort configuration file) → This command will apply the rules configured in the snort.conf file to each packet to decide if an action based upon the rulr type in the file should be taken. If we don't specify any output directory for the program, it will default to /var/log/snort

Some options for NIDS mode of Snort are as follows:

-A fast → Fast Alert Mode. Writes the alert in simple format with a timestamp, alert message, source and destination IPs/ports.

-A full → Full Alert Mode. This is the default alert mode and will be used automatically if you do not specify a mode.

-A none → Turns off alerting.

-A console → Sends "fast-style" alerts to Console (Screen).

-A unsock → Sends alerts to a UNIX socket that another program can listen on.

- High performance configuration in Snort

  ./snort –b –A fast –c snort.conf → This command will log packets in tcpdump format and produce minimal alerts.

Snort Modes

Snort can operate in three different modes namely- inline, passive and inline-test.

- Inline: When Snort is in Inline mode, it acts as an IPS (Intrusion Prevention System) allowing drop rules to trigger.

  snort –Q

  config policy_mode:inline

- Passsive: When Snort is in Passive mode, it acts as an IDS (Intrusion Detection System). Drop rules are not loaded.

  config policy_mode:tap

- Inline-Test: Inline-Test mode stimulates the inline mode of Snort, allowing evaluation of inline behavior without affecting traffic. The drop rules will be loaded and will be triggered as a Wdrop (Would Drop) alert.

  snort –enable-inline-test

  config policy_mode:inline_test

Generating Alerts

Snort can generate alerts as follows:

alert icmp any any -> any any (msg: ”Ping Detected”; sid:1001466; rev:1)

This rule makes alerts on each and every icmp packet as well as shows source and destination port addresses.

- **Npcap**

Npcap is the Nmap Project's packet sniffing (and sending) library for Windows. It is based on the discontinued Winpcap library, but with improved speed, portability, security, and efficiency. In particular, Npcap offers:

- **WinPcap for Windows 10**: Npcap works on Windows 7 and later by making use of the new NDIS 6 Light-Weight Filter (LWF) API. It's faster than the deprecated NDIS 5 API, which Microsoft could remove at any time. Also, the driver is signed with our EV certificate and countersigned by Microsoft, so it works even with the stricter driver signing requirements in Windows 10 1607.
- **Extra Security**: Npcap can (optionally) be restricted so that only Administrators can sniff packets. If a non-Admin user tries to utilize Npcap through software such as Nmap or Wireshark, the user will have to pass a User Account Control (UAC) dialog to utilize the driver. This is conceptually similar to UNIX, where root access is generally required to capture packets. We've also enabled the Windows ASLR and DEP security features and signed the driver, DLLs, and executables to prevent tampering.
- **Loopback Packet Capture**: Npcap is able to sniff loopback packets (transmissions between services on the same machine) by using the Windows Filtering Platform (WFP). After installation, Npcap will create an adapter named Npcap Loopback Adapter for you. If you are a Wireshark user, choose this adapter to capture, you will see all loopback traffic the same way as other non-loopback adapters. Try it by typing in commands like “ping 127.0.0.1” (IPv4) or “ping ::1” (IPv6).
- **Loopback Packet Injection**: Npcap is also able to send loopback packets using the Winsock Kernel (WSK) technique. User-level software such as Nping can just send the packets out using Npcap Loopback Adapter just like any other adapter. Npcap then does the magic of removing the packet's Ethernet header and injecting the payload into the Windows TCP/IP stack.
- **Libpcap API**: Npcap uses the excellent Libpcap library, enabling Windows applications to use a portable packet capturing API that is also supported on Linux and Mac OS X. While WinPcap was based on LibPcap 1.0.0 from 2009, Npcap includes the latest

Libpcap release along with improvements that we also contribute back upstream to Libpcap.

- **WinPcap compatibility**: For applications that don't yet make use of Npcap's advanced features, Npcap can be installed in "WinPcap Compatible Mode." This will replace any existing WinPcap installation. If compatibility mode is not selected, Npcap can coexist alongside WinPcap; applications which only know about WinPcap will continue using that, while other applications can choose to use the newer and faster Npcap driver instead.

5.1.2 Hardware Requirements

Processor	: Any Processor above 500 MHz.
Ram	: 4 GB
Hard Disk	: 4 GB
Input device	: Standard Keyboard and Mouse.
Output device	: VGA and High Resolution Monitor.

5.2 Source Code

```
import pandas as pd

import matplotlib as plt

import numpy as np

from sklearn import linear_model

#from sklearn import cross_validation

from scipy.stats import norm


from sklearn.linear_model import LogisticRegression

from sklearn.svm import LinearSVC

from sklearn import preprocessing

from sklearn import ensemble

from sklearn.svm import SVC

from sklearn import svm

from sklearn.svm import LinearSVC

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import f1_score

from sklearn.neighbors import KNeighborsClassifier


#from sklearn import cross_validation


from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import neighbors
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from random import seed
from random import randrange
from csv import reader
import csv
import numpy as np
import pandas as pd
from pandas import read_csv
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

```
mse=[]
mae=[]
rsq=[]
rmse=[]
acy=[]
np.random.seed(0)
data = pd.read_csv('KDDTrain+.csv')
print(data.head())

names=list(data.columns)

correlations = data.corr()
# plot correlation matrix
fig = plt.figure()
fig.canvas.set_window_title('Correlation Matrix')
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
```

```
ax.set_yticklabels(names)
#fig.savefig('Correlation Matrix.png')
#scatterplot
#scatter_matrix(data)
#
plt.show()


ncols=3
plt.clf()
f = plt.figure(1)
f.suptitle(" Data Histograms", fontsize=12)
vlist = list(data.columns)
nrows = len(vlist) // ncols
if len(vlist) % ncols > 0:
        nrows += 1
for i, var in enumerate(vlist):
        plt.subplot(nrows, ncols, i+1)
        plt.hist(data[var].values, bins=15)
        plt.title(var, fontsize=10)
        plt.tick_params(labelbottom='off', labelleft='off')
plt.tight_layout()
plt.subplots_adjust(top=0.88)
plt.show()
```

```
def process(X_train,X_test,y_train,y_test):

        #X_train, X_test, y_train, y_test = train_test_split(x1, y1)

        model3=LogisticRegression()

        model3.fit(X_train,y_train)

        y = model3.predict(X_test)

        print("MSE VALUE FOR LogisticRegression IS %f " %
mean_squared_error(y_test,y))

        print("MAE VALUE FOR LogisticRegression IS %f " %
mean_absolute_error(y_test,y))

        print("R-SQUARED VALUE FOR LogisticRegression IS %f " %
r2_score(y_test,y))

        rms = np.sqrt(mean_squared_error(y_test,y))

        print("RMSE VALUE FOR LogisticRegression IS %f " % rms)

        ac=accuracy_score(y_test,y) * 100

        print ("ACCURACY VALUE LogisticRegression IS %f" %ac)

        print("------------------------------------------------------------------")

        mse.append(mean_squared_error(y_test,y))

        mae.append(mean_absolute_error(y_test,y))

        rsq.append(r2_score(y_test,y))

        rmse.append(rms)

        acy.append(ac)

        result2=open("results/resultLogisticRegression.csv","w")

        result2.write("ID,Predicted Value" + "\n")

        for j in range(len(y)):
```

```
            result2.write(str(j+1) + "," + str(y[j]) + "\n")
        result2.close()
        model4 = DecisionTreeClassifier()
        model4.fit(X_train, y_train)
        y = model4.predict(X_test)
        print("MSE VALUE FOR DecisionTree IS %f " %
mean_squared_error(y_test,y))
        print("MAE VALUE FOR DecisionTree IS %f " %
mean_absolute_error(y_test,y))
        print("R-SQUARED VALUE FOR DecisionTree IS %f " % r2_score(y_test,y))
        rms = np.sqrt(mean_squared_error(y_test,y))
        print("RMSE VALUE FOR DecisionTree IS %f " % rms)
        ac=accuracy_score(y_test,y) * 100
        print ("ACCURACY VALUE DecisionTree IS %f" % ac)
        print("------------------------------------------------------------------")
        mse.append(mean_squared_error(y_test,y))
        mae.append(mean_absolute_error(y_test,y))
        rsq.append(r2_score(y_test,y))
        rmse.append(rms)
        acy.append(ac)
        result2=open("results/resultDecisionTree.csv","w")
        result2.write("ID,Predicted Value" + "\n")
        for j in range(len(y)):
            result2.write(str(j+1) + "," + str(y[j]) + "\n")
```

```
        result2.close()


        model5 = RandomForestClassifier()

        model5.fit(X_train, y_train)

        y = model5.predict(X_test)


        print("-----------------------------------------------------------------------")

        print("MSE VALUE FOR Random Forest IS %f "  %
mean_squared_error(y_test,y))

        print("MAE VALUE FOR Random Forest IS %f " %
mean_absolute_error(y_test,y))

        print("R-SQUARED VALUE FOR Random Forest IS %f " % r2_score(y_test,y))

        rms = np.sqrt(mean_squared_error(y_test,y))

        print("RMSE VALUE FOR Random Forest IS %f " % rms)

        ac=accuracy_score(y_test,y) * 100

        print ("ACCURACY VALUE Random Forest IS %f" % ac)

        print("-----------------------------------------------------------------------")

        mse.append(mean_squared_error(y_test,y))

        mae.append(mean_absolute_error(y_test,y))

        rsq.append(r2_score(y_test,y))

        rmse.append(rms)

        acy.append(ac)

        result2=open("results/resultRandomForest.csv","w")

        result2.write("ID,Predicted Value" + "\n")
```

```
for j in range(len(y)):
    result2.write(str(j+1) + "," + str(y[j]) + "\n")
result2.close()


model6 = KNeighborsClassifier(n_neighbors=5)
model6.fit(X_train, y_train)
y = model6.predict(X_test)
print("-----------------------------------------------------------------")
print("MSE VALUE FOR KNN IS %f " % mean_squared_error(y_test,y))
print("MAE VALUE FOR KNN IS %f " % mean_absolute_error(y_test,y))
print("R-SQUARED VALUE FOR KNN IS %f " % r2_score(y_test,y))
rms = np.sqrt(mean_squared_error(y_test,y))
print("RMSE VALUE FOR KNN IS %f " % rms)
ac=accuracy_score(y_test,y) * 100
print ("ACCURACY VALUE KNN IS %f" % ac)
print("-----------------------------------------------------------------")
mse.append(mean_squared_error(y_test,y))
mae.append(mean_absolute_error(y_test,y))
rsq.append(r2_score(y_test,y))
rmse.append(rms)
acy.append(ac)
result2=open("results/resultKNNresult.csv","w")
result2.write("ID,Predicted Value" + "\n")
```

```
for j in range(len(y)):
    result2.write(str(j+1) + "," + str(y[j]) + "\n")
result2.close()
al = ['Logistic Regression','DecisionTree', 'Random Forest', 'KNN']
result2=open('results/MSE.csv', 'w')
result2.write("Algorithm,MSE" + "\n")
for i in range(0,len(mse)):
    result2.write(al[i] + "," +str(mse[i]) + "\n")
result2.close()


colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#8c564b"]
explode = (0.1, 0, 0, 0, 0)

#Barplot for the dependent variable
fig = plt.figure(0)
df = pd.read_csv('results/MSE.csv')
acc = df["MSE"]
alc = df["Algorithm"]
plt.bar(alc,acc,align='center', alpha=0.5,color=colors)
plt.xlabel('Algorithm')
plt.ylabel('MSE')
plt.title("MSE Value");
fig.savefig('results/MSE.png')
```

```
plt.show()
result2=open('results/MAE.csv', 'w')
result2.write("Algorithm,MAE" + "\n")
for i in range(0,len(mae)):
    result2.write(al[i] + "," +str(mae[i]) + "\n")
result2.close()

fig = plt.figure(0)
df = pd.read_csv('results/MAE.csv')
acc = df["MAE"]
alc = df["Algorithm"]
plt.bar(alc,acc,align='center', alpha=0.5,color=colors)
plt.xlabel('Algorithm')
plt.ylabel('MAE')
plt.title('MAE Value')
fig.savefig('results/MAE.png')
plt.show()

result2=open('results/R-SQUARED.csv', 'w')
result2.write("Algorithm,R-SQUARED" + "\n")
for i in range(0,len(rsq)):
    result2.write(al[i] + "," +str(rsq[i]) + "\n")
result2.close()
```

```
fig = plt.figure(0)
df = pd.read_csv('results/R-SQUARED.csv')
acc = df["R-SQUARED"]
alc = df["Algorithm"]
colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#8c564b"]
explode = (0.1, 0, 0, 0, 0)
plt.bar(alc,acc,align='center', alpha=0.5,color=colors)
plt.xlabel('Algorithm')
plt.ylabel('R-SQUARED')
plt.title('R-SQUARED Value')
fig.savefig('results/R-SQUARED.png')
plt.show()

result2=open('results/RMSE.csv', 'w')
result2.write("Algorithm,RMSE" + "\n")
for i in range(0,len(rmse)):
    result2.write(al[i] + "," +str(rmse[i]) + "\n")
result2.close()

fig = plt.figure(0)
df = pd.read_csv('results/RMSE.csv')
acc = df["RMSE"]
alc = df["Algorithm"]
```

```
plt.bar(alc, acc, align='center', alpha=0.5,color=colors)
plt.xlabel('Algorithm')
plt.ylabel('RMSE')
plt.title('RMSE Value')
fig.savefig('results/RMSE.png')
plt.show()

result2=open('results/Accuracy.csv', 'w')
result2.write("Algorithm,Accuracy" + "\n")
for i in range(0,len(acy)):
    result2.write(al[i] + "," +str(acy[i]) + "\n")
result2.close()

fig = plt.figure(0)
df = pd.read_csv('results/Accuracy.csv')
acc = df["Accuracy"]
alc = df["Algorithm"]
plt.bar(alc, acc, align='center', alpha=0.5,color=colors)
plt.xlabel('Algorithm')
plt.ylabel('Accuracy')
plt.title('Accuracy Value')
fig.savefig('results/Accuracy.png')
plt.show()
```

```
dataset=pd.read_csv('KDDTrain+.csv').values
dataset1=pd.read_csv('KDDTest+.csv').values
X_train = dataset[:,0:41]
y_train = dataset[:,41]
X_test = dataset1[:,0:41]
y_test = dataset1[:,41]
dos=[]
u2r=[]
r2l=[]
probe=[]
normal= []
unknown=[]



for c in y_test:
        print(c)
        if c==1.0:
                dos.append(c)
        if c==2.0:
                u2r.append(c)
        if c==3.0:
                r2l.append(c)
```

```
        if c==4.0:
                probe.append(c)
        if c==5.0:
                normal.append(c)
#X_train, X_test, y_train, y_test
process(X_train,X_test,y_train,y_test)
tot=len(dos)+len(u2r)+len(r2l)+len(probe)+len(normal)
d=(len(dos)/tot)*100
u=(len(u2r)/tot)*100
r=(len(r2l)/tot)*100
p=(len(probe)/tot)*100
n=(len(normal)/tot)*100
print(len(dos))
print(len(u2r))
print(len(r2l))
print(len(probe))
print(len(normal))
print(d)
print(u)
print(r)
print(p)
print(n)
colors = ["#FF0000", "#FF0000", "#FF0000", "#FF0000", "#008000"]
```

```
explode = (0.1, 0, 0, 0, 0)

alc = ["dos","U2R","R2L","PROBE","NORMAL"]

acc = [d,u,r,p,n]

fig = plt.figure(0)

plt.bar(alc, acc, align='center', alpha=0.5,color=colors)

plt.xlabel('Methods')

plt.ylabel('Rate in %')

plt.title('Detection Rate')

plt.show()
```

5.3 Screen Shots

The following figure shows the scatter plot of whole train dataset

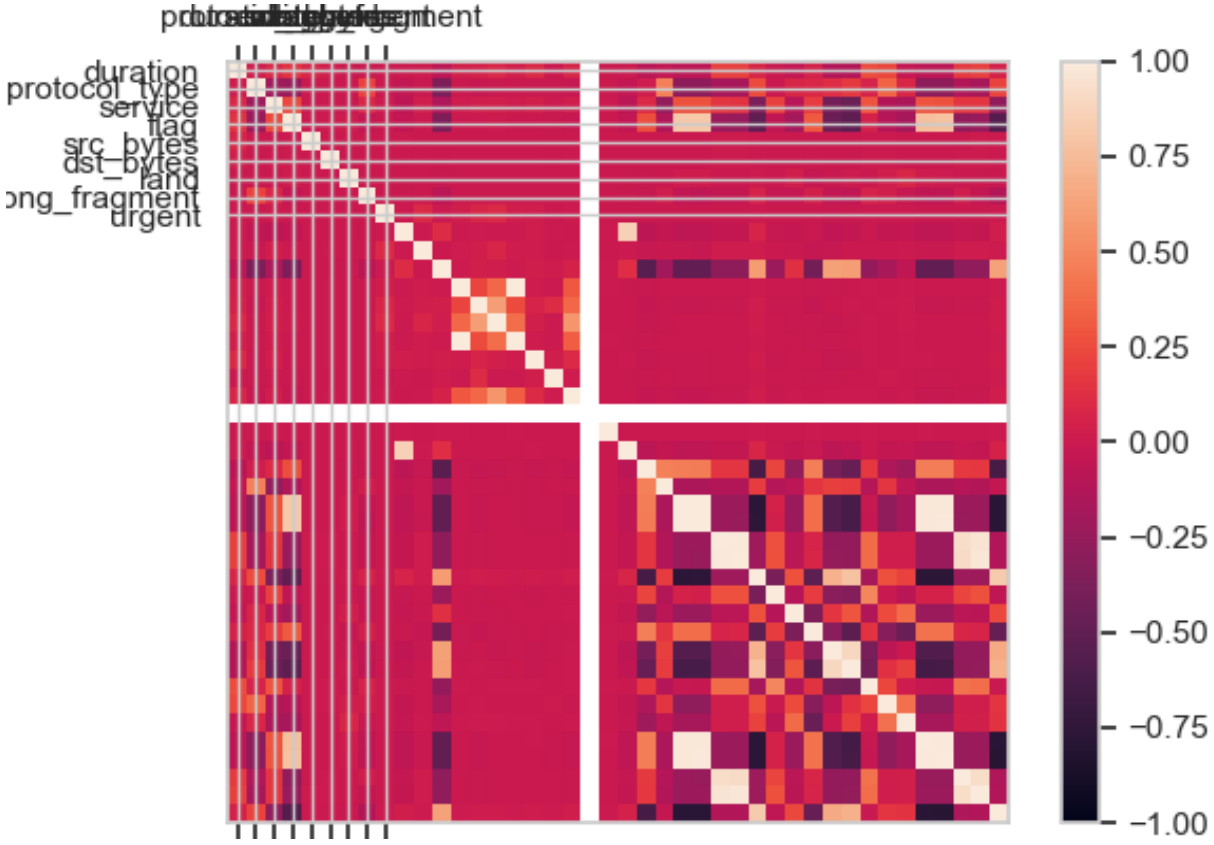


Fig 5.3.1 Scatter Plot of the Data Set

The following figure shows the Histogram data visualization of whole train dataset

Fig 5.3.2 Data Histograms of the Data Set

The following figure shows the execution of algorithm in command prompt

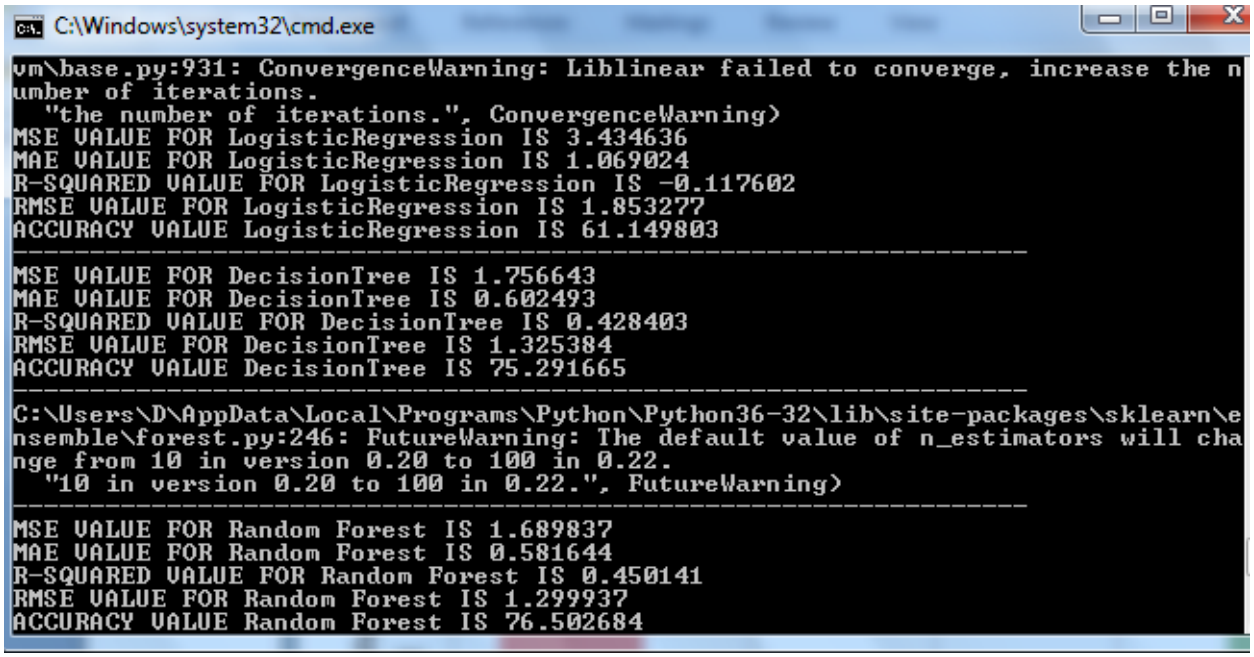


Fig 5.3.3 Output of the Program in Command Prompt

The following screens shows the MSE error value

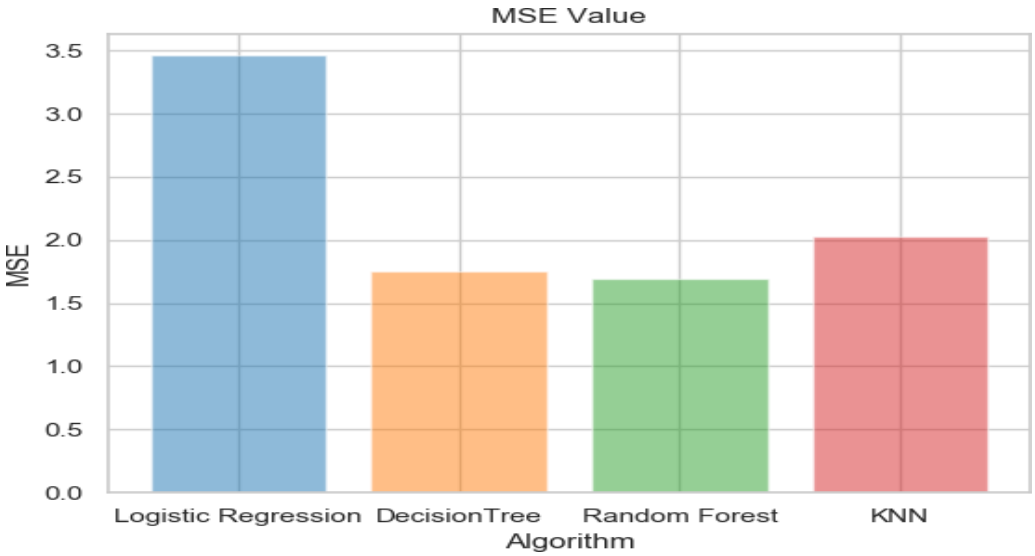


Fig 5.3.4 MSE Values

The following screens shows the MAE error value

Fig 5.3.5 MAE values

The following screens shows the RMSE error value

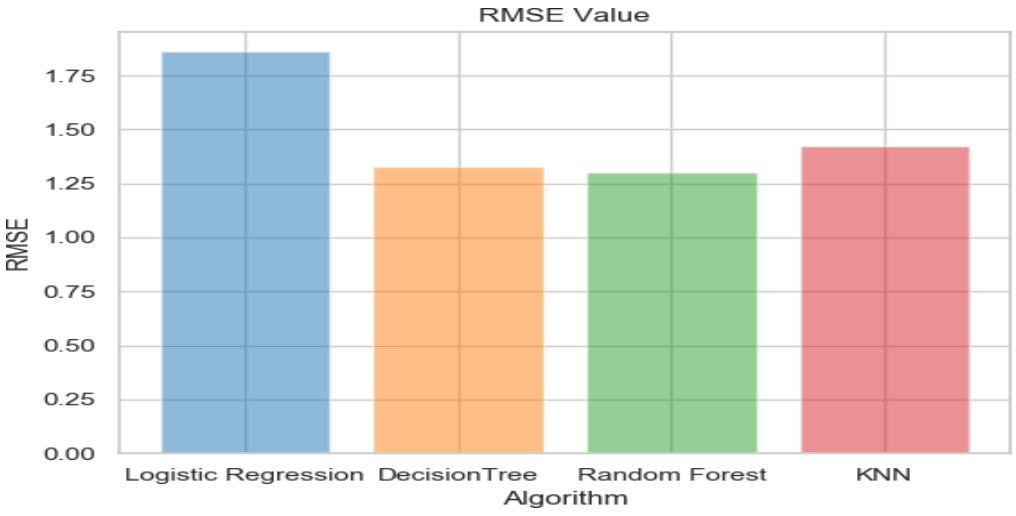


Fig 5.3.6 RMSE values

The following screen shows the Accuracy comparison of intrusion detection prediction using machine learning algorithms.

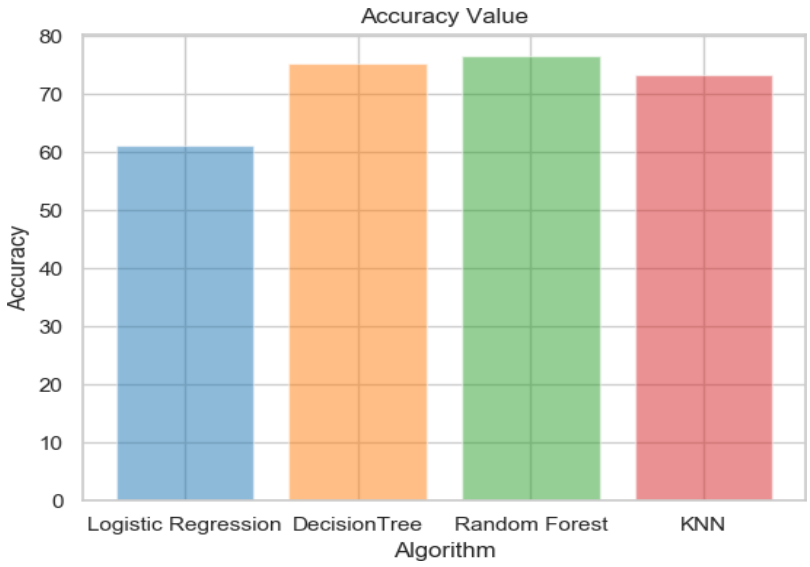


Fig 5.3.7 Accuracy Rate of the Algorithms

The following screen shows the detection rate of different class of intrusion

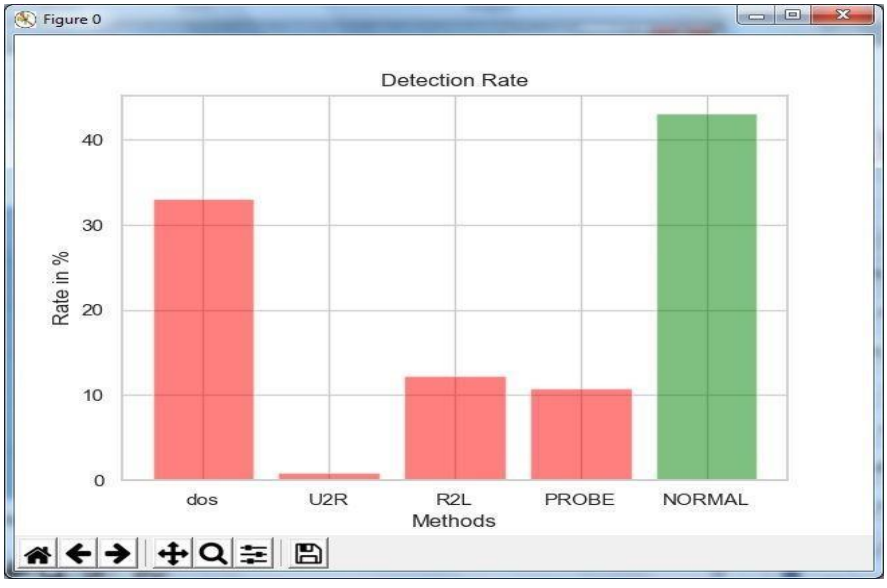


Fig 5.3.8 Detection Rate of the Attacks

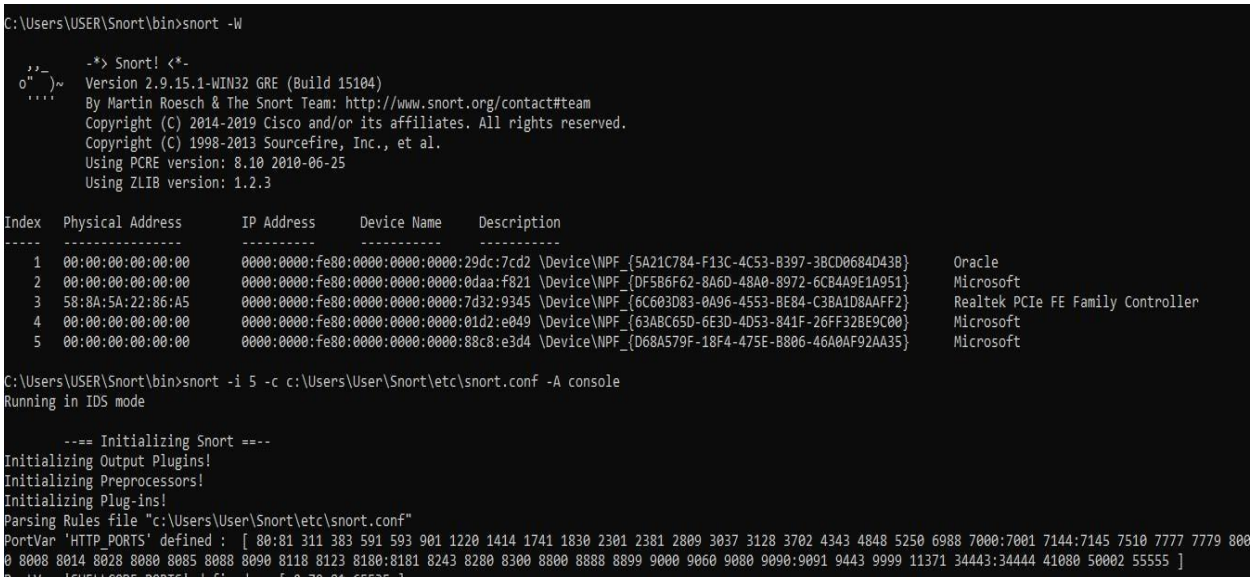


Fig 5.3.9 Running Snort by configuring rules

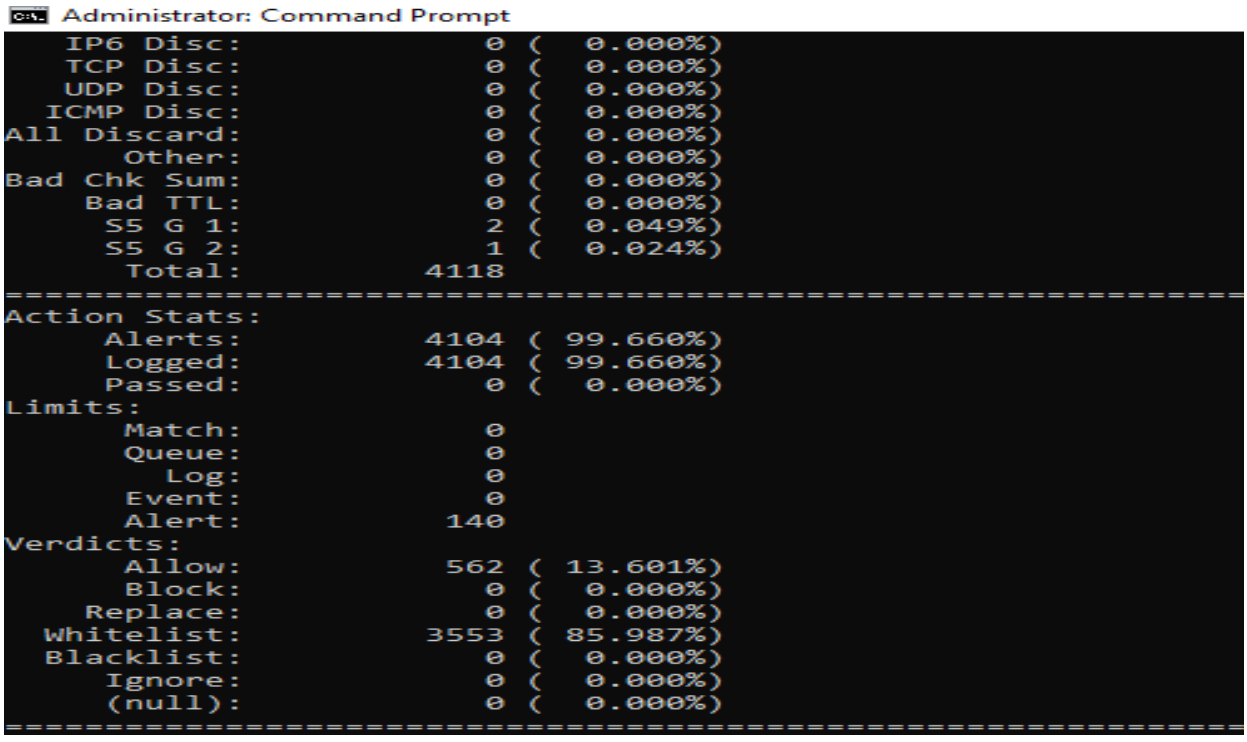


Fig 5.3.10 Snort Checking for Alerts

5.4 Experimental Analysis/ Testing

The proposed work is implemented in Python 3.6.4 with libraries scikit-learn, pandas, matplotlib and other mandatory libraries. The training dataset of KDD contains 125973 rows. The test dataset contains 22543. Machine learning algorithm is applied such as decision tree, Logistic regression and Random forest. We used these machine learning algorithm and indentified intrusion. The result shows that intrusion detection is efficient using Random Forest algorithm. Random forest achieves 76.50% accuracy, Decision Tree achieves around 75.29% accuracy, Logistic Regression achieves 61.14% accuracy.

The following table shows the accuracy arrived in our experimental analysis.

Algorithm	Accuracy (%)
Logistic regression	61.14
Decision Tree	75.29
Random Forest	76.50
KNN	73.22

Table 5.4.1 Experimental Analysis Result of the proposed system

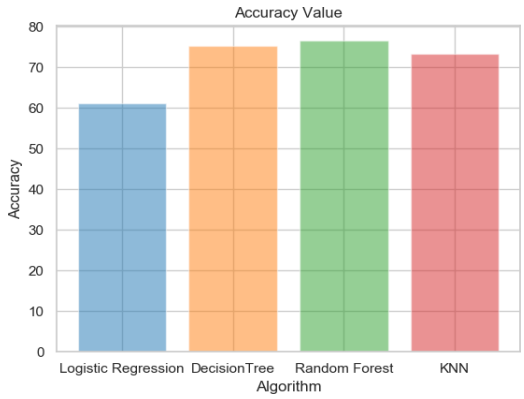


Fig 5.4.1 Accuracy Rate Comparison of the Algorithms

6. CONCLUSION & FUTURE SCOPE

6.1 CONCLUSION

The main aim of Intrusion Detection System is to detect the attacks and malicious activities that occur within a network and to reduce the rate of false positives. By using the machine learning algorithms, the output of the IDS would be accurate, advanced and reliable. This system also shows the accuracy rate of the attacks that have been detected by the different machine learning algorithms that have been implemented. The incremental increase in the use of technology has led to huge amount of data that needs to be processed and stored securely for the users. Security is a major aspect for any user. If a system is secure, we can highly ensure user's privacy is high. The more secure the system, the more reliable it is. If an Intrusion Detection System is capable of providing good security for user's data, we can say that the developed Intrusion Detection System is good.

6.2 FUTURE SCOPE

The system that has been proposed can be made more reliable and efficient by implementing other machine learning algorithms along with the ones that already have been implemented so that intrusion can be detected easily. Also the other types of attacks can also be classified as the classes of intrusion to identify more attacks and provide more security and reliability. Thus further development of the system can help to increase the detection rate and lower the false positive rates.

REFERENCES

[1] A B. Athira, V. Pathari, “Standardisation and Classification of Alerts Generated by Intrusion Detection Systems”, IJCI, International Journal on Cybernetics & Informatics, Vol 5 Issue 2, 2016.

[2] Johansson Daniel, Andersson Par, “Intrusion Detection Systems with Correlation Capabilities”

[3] Yasm Curt, “Prelude as a Hybrid IDS Framework”, March, 2009

[4] Kumar Vinod, Sangwan Prakash Om, “Signature Based Intrusion Detection System Using SNORT”, IJCAIT, International Journal of Computer Applications & Information Technology, Vol. I, Issue III, November 2012.

[5] Singh Deepak Kumar, Gupta Jitendra Kumar, “An approach for Anomaly based Intrusion detection System using SNORT“, IJSER, International Journal of Scientific & Engineering Research, Volume 4, Issue 9, September 2013.

[6] S, Vijayarani, and Maria Sylviaa S. “Intrusion Detection System – A Study”, IJSPTM, International Journal of Security, Privacy and Trust Management ,Vol 4, Issue 1, pp. 31–44, February 2015.

[7] Yang Guangming, Chen Dongming, Xu Jian, Zhu Zhiliang, “Research of Intrusion Detection System Based on Vulnerability Scanner”, ICACC, Advanced ComputerControl, March 2010.

[8] Chakraborty Nilotpal, “Intrusion Detection System and Intrusion Prevention System: A Comparative Study”, IJCBR, International Journal of Computing and Business Research , Volume 4 Issue 2, May 2013.

[9] Kothari Pravin, “Intrusion Detection Interoperability and Standardization”, February, 2002.

[10] TIMOFTE Jack, “Intrusion Detection using Open Source Tools”, Revista Informatica Economica nr.2(46), pp. 75-79, 2008.