INTRUSION DETECTION IN SOFTWARE DEFINED NETWORK USING MACHINE LEARNING

Submitted in partial fulfillment of the requirements for

The award of

Bachelor of Engineering Degree in Computer Science and Engineering

By

CHINTALA PANITH KUMAR (Reg. No. 38110382)
PAMMI PAVAN KUMAR REDDY (Reg. No. 38110381)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI - 600 119

April - 2022

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI-600119

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **CHINTALA PANITH KUMAR (REG. NO. 38110382), PAMMI PAVAN KUMAR REDDY (REG. NO. 38110381)** who carried out the project entitled **"INTRUSION DETECTION IN SOFTWARE DEFINED NETWORK USING MACHINE LEARNING"** under our supervision from Nov 2021 to April 2022.

Internal Guide

Dr. KAMALESH, M.E., Ph.D.,

Head of the Department

Dr. S. VIGNESHWARI M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner **External Examiner**

DECLARATION

I **CHINTALA PANITH KUMAR (Reg. No. 38110382), PAMMI PAVAN KUMAR REDDY (Reg. No. 38110382)** hereby declare that the Project Report entitled **"INTRUSION DETECTION IN SOFTWARE DEFINED NETWORK USING MACHINE LEARNING"** done by me under the guidance of **Dr. KAMALESH, M.E., Ph.D.,** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Technology degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI **SIGNATURE OF THE CANDIDATE**

ACKNOWLEDGEMENT

We are pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., Ph.D**., Dean, School of Computing, and DR. **S. VIGNESHWARI, M.E., Ph.D**., Head of the Department, Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. KAMALESH**, M.E., Ph.D., for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the department of **COMPUTER SCIENCE AND ENGINEERING** who were helpful in many ways for the completion of the project.

ABSTRACT

Intrusion detection systems (IDSs) are currently drawing a great amount of interest as a key part of system defense. IDSs collect network traffic information from some point on the network or computer system and then use this information to secure the network. To distinguish the activities of the network traffic that the intrusion and normal is very difficult and to need much time consuming. An analyst must review all the data that large and wide to find the sequence of intrusion on the network connection. Therefore, it needs a way that can detect network intrusion to reflect the current network traffics. In this study, a novel method to find intrusion characteristic for IDS using genetic algorithm machine learning of data mining technique was proposed. Method used to generate of rules is classification by Genetic algorithm of decision tree. These rules can determine of intrusion characteristics then to implement in the genetic algorithm as prevention.so that besides detecting the existence of intrusion also can execute by doing deny of intrusion as prevention.

TABLE OF CONTENT

List Of Figures

Chapter 1

INTRODUCTION

Approaches for intrusion detection can be broadly divided into two types: misuse detection and anomaly detection. In misuse detection system, all known types of attacks (intrusions) can be detected by looking into the predefined intrusion patterns in system audit traffic. In case of anomaly detection, the system first learns a normal activity profile and then flags all system events that do not match with the already established profile. The main advantage of the misuse detection is its capability for high detection rate with a difficulty in finding the new or unforeseen attacks. The advantage of anomaly detection lies in the ability to identify the novel (or unforeseen) attacks at the expense of high false positive rate. Network monitoring-based machine learning techniques have been involved in diverse fields. Using bi-directional long-short-term-memory neural networks, a social media network monitoring system is proposed for analyzing and detecting traffic accidents.

The proposed method retrieves traffic-related information from social media (Facebook and Twitter) using query-based crawling: this process collects sentences related to any traffic events, such as jams, road closures, etc. Subsequently, several pre-processing techniques are carried out, such as steaming, tokenization, POS tagging and segmentation, in order to transform the retrieved data into structured form. Thereafter, the data are automatically labeled as 'traffic' or 'non-traffic', using a latent Dirichlet allocation (LDA) algorithm. Traffic- labeled data are analyzed into three types; positive, negative, and neutral. The output from this stage is a sentence labeled according to whether it is traffic or non-traffic, and with the polarity of that traffic sentence (positive, negative or neutral). Then, using the bag-of-words (BoW) technique, each sentence is transformed into a one-hot encoding representation in order to feed it to the Bi-directional LSTM neural network (Bi-LSTM). After the learning process, the neural networks perform multi-class classification using the softmax layer in order to classify the sentence in terms of location, traffic event and polarity types. The proposed method compares different classical machine learning and advanced deep learning approaches in terms of accuracy, F-score and other criteria.

EXISTING SYSTEM:

Today network has become an essential part of public infrastructures with the inception of public and private cloud computing. The traditional networking approach has become too complex. This complexity has resulted in a barrier for creating new and innovative services within a single data center, difficulties in interconnecting data centers, interconnection within enterprises, and bigger barrier in the continued growth of the Internet in general.

PROBLEM STATEMENT:

- To distinguish the activities of the network traffic that the intrusion and normal is very difficult and to need much time consuming.

- An analyst must review all the data that large and wide to find the sequence of intrusion on the network connection.
- It needs a way that can detect network intrusion to reflect the current network traffics.
- Combination of IDS and firewall so-called the IPS, so that besides detecting the existence of intrusion also can execute by doing deny of intrusion as prevention**.**

Chapter 2

Designing a Network Intrusion Detection System Based on Machine Learning for Software Defined Networks

Abstract:
Software-defined Networking (SDN) has recently developed and been put forward as a promising and encouraging solution for future internet architecture. Managed, the centralized and controlled network has become more flflexible and visible using SDN. On the other hand, these advantages bring us a more vulnerable environment and dangerous threats, causing network breakdowns, systems paralysis, online banking frauds and robberies. These issues have a significantly destructive impact on organizations, companies or even economies. Accuracy, high performance and real-time systems are essential to achieve this goal successfully. Extending intelligent machine learning algorithms in a network intrusion detection system (NIDS) through a software-defined network (SDN) has attracted considerable attention in the last decade. Big data availability, the diversity of data analysis techniques, and the massive improvement in the machine learning algorithms enable the building of an effective, reliable and dependable system for detecting different types of attacks that frequently target networks. This study demonstrates the use of machine learning algorithms for traffic monitoring to detect malicious behavior in the network as part of NIDS in the SDN controller. Different classical and advanced tree-based machine learning techniques, Decision Tree, Random Forest and XGBoost are chosen to demonstrate attack detection. The NSL-KDD dataset is used for training and testing the proposed methods; it is considered a benchmarking dataset for several state-of-the-art approaches in NIDS. Several advanced preprocessing techniques are performed on the dataset in order to extract the best form of the data, which produces outstanding results compared to other systems. Using just five out of 41 features of NSL-KDD, a multi-class classification task is conducted by detecting whether there is an attack and classifying the type of attack (DDoS, PROBE, R2L, and U2R), accomplishing an accuracy of 95.95%.

Introduction

A network intrusion detection system is a process for discovering the existence of malicious or unwanted packets in the network. This process is done using real-time traffic monitoring to find out if any unusual behavior is present in the network or not. Big data, powerful computation facilities, and the expansion of the network size increase the demand for the required tasks that should be carried out simultaneously in real-time. Therefore, NIDS should be careful, accurate,

and precise in monitoring, which has not been the case in the traditional methods. On the other hand, the rapid increase in the accuracy of machine learning algorithms is highly impressive. Its introduction relies on the increasing demand for improved performance on different types of network. However, software defined network (SDN) implementation of the network-based intrusion detection system (NIDS) has opened a frontier for its deployment, considering the increasing scope and typology of security risks of modern networks. The rapid growth in the volume of network data and connected devices carries inherent security risks. The adoption of technologies such as the Internet of Things (IoT), artificial intelligence (AI), and quantum computing, has increased the threat level, making network security challenging and necessitating a new paradigm in its implementation. Various attacks have overwhelmed previous approaches (classified into signature-based intrusion detection systems and anomaly-based intrusion detection systems, increasing the need for advanced, adaptable and resilient security implementation. For this reason, the traditional network design platform is being transformed into the evolving SDN implementation Monitoring data and analyzing it over time are essential to the process of predicting future events, such as risks, attacks and diseases. The more details are formed, discovered and documented through analyzing very large-scale data, the more saved resources, as well as the working environment, will remain normal without any variations. Big data analytics (BDA) research in the supply chain becomes the secret of a protector for managing and preventing risks. BDA for humanitarian supply chains can aid the donors in their decision of what is appropriate in situations such as disasters, where it can improve the response and minimize human suffering and deaths. BDA and data monitoring using machine learning can help in identifying and understanding the interrelationships between the reasons, difficulties, obstacles and barriers that guide organizations in taking the most efficient and accurate decisions in risk management processes. This could impact entire organizations and countries, producing a hugely significant improvement in the process. Network monitoring-based machine learning techniques have been involved in diverse fields. Using bi-directional long-short-term-memory neural networks, a social media network monitoring system is proposed for analyzing and detecting traffic accidents.The proposed method retrieves traffic-related information from social media (Facebook and Twitter) using query-based crawling: this process collects sentences related to any traffic events, such as jams, road closures, etc. Subsequently, several pre-processing techniques are carried out, such as steaming, tokenization, POS tagging and segmentation, in order to transform the retrieved data into structured form. Thereafter, the data are automatically labeled as ’traffic’ or ’non-traffic’, using a latent Dirichlet allocation (LDA) algorithm.Traffic- labeled data are analyzed into three types; positive, negative, and neutral. The output from this stage is a sentence labeled according to whether it is traffic or non-traffic,and with the polarity of that traffic sentence (positive, negative or neutral). Then, using the bag-of-words (BoW) technique, each sentence is transformed into a one-hot encoding representation in order to feed it to the Bi-directional LSTM neural network (Bi-LSTM). After the learning process, the neural networks perform multi-class classification using the softmax layer in order to classify the sentence in terms of location, traffic event and polarity types. The proposed method compares different classical machine learning and advanced deep learning approaches in terms of accuracy, F-score and other criteria. Many initiatives and workshops have been conducted in order to improve and develop the healthcare systems using machine learning, such as [12]. In these workshops several proposed

machine algorithms have been used, such as K Nearest-Neighbors, logistic regression, K-means clustering, Random Forest (RF) etc, together with deep learning algorithms such as CNN, RNN, fully connected layer and auto-encoder. These varieties of techniques allow the researchers to deal with several data types, such as medical imaging,history, medical notes, video data, etc. Therefore, different topics and applications are introduced, with significant performance results such as causal inference, in investigations of Covid-19, disease prediction, such as disorders and heart diseases. Using intelligent ensemble deep learning methods, healthcare monitoring is carried out for prediction of heart diseases. Real-time health status monitoring can prevent and predict any heart attacks before occurrence. For disease prediction, the proposed ensemble deep learning approach achieved a brilliant accuracy performance score of 98.5%. The proposed model takes two types of data that are transferred and saved on an online cloud database. The first is the data transferred from the sensors; these sensors have been placed in different places on the body in order to extract more than 10 different types of medical data. The second type is the daily electronic medical records from doctors, which includes various types of data, such as smoking history, family diseases, etc. The features are fused using the feature fusion Framingham Risk factors technique, which executes two tasks at a time, fusing the data together, and then extracting a fused and informative feature from this data. Then different pre-processing techniques are used to transform the data into a structured and well-prepared form, such as normalization, missing values filtering and feature weighting. Subsequently, an ensemble deep learning algorithm starts which learns from the data in order to predict whether a heart disease will occur or the threat is absent. IDS refers to a mechanism capable of identifying or detecting intrusive activities. In a broader view, this encompasses all the processes used in the discovery of unauthorized uses of network devices or computers. This is achieved through software designed specifically to detect unusual or abnormal activities. IDS can be classified according to several surveys and sources in the literature into four types (HIDS, NIDS, WIDS, NBA). NIDS is an inline or passive-based intrusion detection technique. The scope of its detection targets network and host levels. The only architecture that fits and works with NIDS is the managed network. The advantage of using NIDS is that it costs less and is quicker in response, since there is no need to maintain sensor programming at the host level. The performance of monitoring the traffic is close to real-time; NIDS can detect attacks as they occur. However, it has the following limited features. It does not indicate if such attacks are successful or not: it has restricted visibility inside the host machine. There is also no effective way to analyze encrypted network traffic to detect the type of attack. Moreover, NIDS may have difficulty capturing all packets in a large or busy network. Thus, it may fail to recognize an attack launched during a period of high traffic. SDN provides a novel means of network implementation, stimulating the development of a new type of network security application. It adopts the concept of programmable networks through the deployment of logically centralized management. The network deployment and configuration are virtualized to simplify complex processes, such as orchestration, network optimization, and traffic engineering. It creates a scalable architecture that allows sufficient and reliable services based on certain types of traffic. The global view approach to a network enhances flow-level control of the underlying layers. Implementing NIDS over SDN becomes a major effective security defense mechanism for detecting network attacks from the network entry point. NIDS has been implemented and investigated for decades to achieve optimal efficiency. It represents an application or device for

monitoring network traffic for suspicious or malicious activity with policy violations. Such activities include malware attacks, untrustworthy users, security breaches, and DDoS. NIDS focuses on identifying anomalous network traffic or behavior; its efficiency means that network anomaly is adequately implemented as part of the security implementation. Since it is nearly impossible to prevent threats and attacks, NIDS will ensure early detection and mitigation. However, the advancement in NIDS has not instilled sufficient confidence among practitioners, since most solutions still use less capable, signature-based techniques. This study aims to increase the focus on several points:

- choosing the right algorithm for the right tasks depends on the data types, size and network behavior and needs.
- Implementing the optimized development process by preparing and selecting the benchmark dataset in order to build a promising system in NIDS.
- Analyzing the data, finding, shaping, and engineering the important features, using several preprocessing techniques by stacking them together with an intelligent order to find the best accuracy with the lowest amount of data representation and size.
- Proposing an integration and complete development process using those algorithms and techniques from the selection of dataset to the evaluation of the algorithms using a different metric. Which can be extended to other NIDS applications.

This study enhances the implementation of NIDS by deploying different machine learning algorithms over SDN. Tree-based machine learning algorithms (XGBoost, randomforest (RF), and decision tree (DT)) were implemented to enhance the monitoring and accuracy performance of NIDS. The proposed method will be trained on network traffic packet data, collected from large-scale resources and servers called NSL-KDD dataset to perform two tasks at a time by detecting whether there is an attack or not and classifying the type of attack. This study enhances the implementation of NIDS by deploying machine learning over SDN. Tree-based machine learning algorithms (XGBoost, random forest (RF), and decision tree (DT)) are proposed to enhance NIDS. The proposed method will be trained on network traffic packet data, collected from large-scale resources and servers, called the NSL-KDD dataset to perform two tasks at a time by detecting whether there is an attack or not and classifying the type of attack.

Background and Related Work:

Integrating machine learning algorithms into SDN has attracted significant attention.

In, a solution was proposed that solved the issues in KDD Cup 99 by performing an extensive experimental study, using the NSL-KDD dataset to achieve the best accuracy in intrusion detection. The experimental study was conducted on five popular and efficient machine learning algorithms (RF, J48, SVM, CART, and Naïve Bayes). The correlation feature selection algorithm was used to reduce the complexity of features, resulting in 13 features only in the NSL-KDD dataset. This study tests the NSL-KDD dataset's performance for real-world anomaly detection in network behavior. Five classic machine learning models RF, J48, SVM, CART, and Naïve Bayes were trained on all 41 features against the five normal types of attacks, DOS, probe, U2R, and R2L to achieve average accuracies of 97.7%, 83%, 94%, 85%, and 70% for each algorithm,

respectively. The same models were trained again using the reduced 13 features to achieve average accuracies of 98%, 85%, 95%, 86%, and 73% for each model. In, a deep neural network model was proposed to find and detect intrusions in the SDN. The NSL-KDD dataset was used to train and test the model. The neural network was constructed with five primary layers, one input layer with six inputs, three hidden layers with (12, 6, 3) neurons, and one output layer with 2D dimensions. The proposed method was trained on six features chosen from 41 features in the NSL-KDD dataset, which are basic and traffic features that can easily be obtained from the SDN environment. The proposed method calculates the accuracy, precision and recall, achieving an F1-score of 0.75. A second evaluation was conducted on seven classic machine learning models (RF, NB, NB Tree, J48, DT, MLP, and SVM) proposed in and the model achieved sixth place out of eight. The same author extended the approach using a gated recurrent unit neural network (GRU-RNN) for SDN anomaly detection, achieving accuracy up to 89%. In addition, the Min-Max normalization technique is used for feature scaling to improve and boost the learning process. The SVM classifier, integrated with the principal component analysis (PCA) algorithm, was used for an intrusion detection application. The NSL-KDD dataset is used in this approach to train and optimize the model for detecting abnormal patterns. A Min-Max normalization technique was proposed to solve the diversity data scale ranges with the lowest misclassification errors. The PCA algorithm is selected as a statistical technique to reduce the NSL-KDD dataset's complexity, reducing the number of trainable parameters that needed to be learned. The nonlinear radial basis function kernel was chosen for SVM optimization. Detection rate (DR), false alarm rate (FAR), and correlation coefficient metrics were chosen to evaluate the proposed model, with an overall average accuracy of 95% using 31 features in the dataset. In [32], an extreme gradient-boosting (XGBoost) classifier was used to distinguish between two attacks, i.e., normal and DoS. The detection method was analyzed and conducted over POX SDN, as a controller, which is an SDN open-source platform for prototyping and developing a technique based on SDN. Mininet was used to emulate the network topology to simulate real-time SDN-based cloud detection. Logistic regression was selected as a learning algorithm, with a regularization term penalty to prevent overfitting. The XGBoost term was added and combined with the logistic regression algorithm to boost the computations by constructing structure trees. The dataset used in this approach was KDD Cup 1999, while 400 K samples were selected for constructing the training set. Two types of normalization techniques were used; one with a logarithmic-based technique and one with a Min-Max-based technique. The average overall accuracy for XGBoost, compared to RF and SVM, was 98%, 96%, 97% respectively. Based on DDoS attack characteristics, a detection system was simulated with the Mininet and flfloodlight platform using the SVM algorithm [5]. The proposed method categorizes the characteristics into six tuples, which are calculated from the packet network. These characteristics are the speed of the source IP (SSIP), the speed of the source port, the standard deviation of flflow packets, the deviation of flflow bytes (SDFB), the speed of flow entries, and the ratio of pair-flflow. Based on the calculated statistics from the SVM classifier's six characteristics, the current network state is normal or attack. Attack flow (AF), DR, and FAR were chosen to achieve an average accuracy of 95%. In TSDL a model with two stages of deep neural networks was designed and proposed for NIDS, using a stacked auto-encoder, integrated with softmax in the output layer as a classifier. TSDL was designed and implemented for Multi-class classification of attack detection. Down-sampling and other preprocessing techniques were

performed over different datasets in order to improve the detection rate, as well as the monitoring efficiency. The detection accuracy for UNSW-NB15 was 89.134%. Different models of neural networks, such as variational auto-encoder, seq2seq structures using Long-ShortTerm-Memory (LSTM) and fully connected networks were proposed in [34] for NIDS. The proposed approach was designed and implemented to differentiate between normal and attack packets in the network, using several datasets, such as NSL-KDD, UNSW NB15, KYOTO-HONEYPOT, and MAWILAB. A variety of preprocessing techniques have been used, such as one-hot-encoding, normalization, etc., for data preparation, feature manipulation and selection and smooth training in neural networks. Those factors are designed mainly, but not only, to enable the neural networks to learn complex features from different scopes of a single packet. Using 4 hidden layers, a deep neural network model [35] was illustrated and implemented on KDD cup99 for monitoring intrusion attacks. Feature scaling and encoding were used for data preprocessing and lower data usage. More than 50 features were used to perform this task on different datasets. Therefore, complex hardware GPUs were used in order to handle this huge number of features with lower training time. A supervised [36] adversarial auto-encoder neural network was proposed for NIDS. It combined GANS and a variational auto-encoder. GANS consists of two different neural networks competing with each other, known as the generator and the discriminator. The result of the competition is to minimize the objective function as much as possible, using the Jensen-Shannon minimization algorithm. The generator tries to generate fake data packets, while the discriminator determined whether this data is real or fake; in other words, it checks if that packet is an attack or normal. In addition, the proposed method integrates the regularization penalty with the model structure for overfitting control behavior. The results were reasonable in the detection rate of U2RL and R2L but lower in others. Multi-channel deep learning of features for NIDS was presented in [37], using AE involving CNN, two fully connected layers and the output to the softmax classifier. The evaluation is done over three different datasets; KDD cup99, UNSW-NB15 and CICIDS, with an average accuracy of 94%. The proposed model provides effective results; however, the structure and the characteristics of the attack were not highlighted clearly. The proposed method enhances the implementation of NIDS by deploying machine learning over SDN. It introduces a machine learning algorithm for network monitoring within the NIDS implementation on the central controller of the SDN. In this paper, enhanced tree-based machine learning algorithms are proposed for anomaly detection. Using only five features, a multi-class classification task is conducted by detecting whether there is an attack or not and classifying the type of attack.

3. Proposed Method

In this section, we discuss and explain each component and its role in the NIDS architecture. As shown in Figure 1, the SDN architecture can be divided into three main layers, as follows:

System Architecture Layers

NIDS component architecture is constructed in three main parts as follows:

- The infrastructure layer consists of two main parts: hardware and software components. The hardware components are devices such as routers and switches. The software components are those components that interface with the hardware, such as Open Flow switches.

- The control layer is an intelligent network controller, such as an SDN controller. The control layer is the layer responsible for regulating actions and traffic data management by establishing or denying every network flow.
- The application layer is the one that performs all network management tasks. These tasks can be performed using an SDN controller and NIDS.

Proposed NIDS Scenario
Attacks are created by an attacker and delivered through the internet. NIDS is deployed over the SDN controller. As NIDS listens to the network and actively compares all traffic against predefined attack signatures, it detects the attacker's scanning attempts. It sends an alert to administrators through its control, and the connections will be blocked due to specific rules in the firewall or routers.

Evaluation
This section presents a generalized flowchart of the proposed method. The dataset,pre-processing techniques, and proposed machine learning algorithms will be presented and discussed.

Generalized Block Diagram
In this subsection, a generalized block diagram is presented and discussed. As shown, the NSL-KDD dataset is used. Data analysis, feature engineering, and other preprocessing techniques are conducted to train the model, using the best hyperparameters, with only five features. Tree-based algorithms are used for the multi-class classification task. The processed data enter the algorithm and are classified as to whether they constitute an attack or are normal; then, the type of attack will be analyzed to see which category it belongs to, and action is taken accordingly.

Dataset Overview
The KDD Cup is the leading data mining competition in the world. The NSL-KDD dataset was proposed to solve many issues represented in the KDD Cup 1999 dataset.
Many researchers have used the NSL-KDD dataset to develop and evaluate the NIDS problem. The dataset includes all types of attacks. The dataset has 41 features, categorized into three main types (basic feature, content-based, and traffic-based features) and labeled as either normal or attack, with the attack type precisely categorized. The categories can be classified into four main groups, with a brief description of each attack type and its impact.

Training Features
As stated in the previous subsection, the dataset has 41 features labeled as either normal or attack with the precise attack category. After experimental trials, five features were selected out of the 41 features in the NSL-KDD dataset, which have the most impact and effect on algorithm-learning performance. Presents the selected five features with a brief description.

Results and Discussion

To evaluate the performance of NIDS in terms of accuracy (AC), different metrics were used; precision (P), recall (R), and F-measure (F). These metrics can be calculated using confusion matrix parameters: true positive (the number of anomalous instances that are correctly classified); false positive (the number of normal instances that are incorrectly classified as anomalous); true negative (the number of normal instances that are correctly classified); and false negative (the number of anomalous instances that are incorrectly classified as normal). A good NIDS must achieve high DR and FAR. Accuracy (AC): This is the percentage of correctly classified network activities. Precision (P): The percentage of predicted anomalous instances that are actual anomalous instances; the higher P, the lower FAR. . Recall (R): the percentage of predicted attack instances versus all attack instances presented. F-measure (F): measuring the performance of NIDS using the harmonic mean of the P and R. We aim to achieve a high F-score.We compare XGBoost against the other two tree- based methods, RF and DT. Using the test set, which includes the four types of attacks as discussed. Three different evaluation metrics are computed; F-score, precision and recall.XGBoost ranked first in the evaluation, with an F1-score of 95.55%, while RF and DT achieved 94.6% and 94.5%, respectively. For precision, XGBoost outperformed RF and DT with a score of 92%, while RF and DT scored 90% and 90.2% respectively. Finally, for Recall, our proposed method with XGBoost proves its stability with a score of 98% while for RF and DT, the results were 82%, and 85%, respectively. From these results, the proposed model with XGBoost performs with high precision and high recall, which means that the classifier returns accurate results and high precision, while, at the same time, returning a majority of all positive results (it's an attack and the classifier detects that it's an attack), which means high recall.

Finally, we evaluate the proposed method using an accuracy analysis against seven classical machine learning algorithms, in addition to the deep neural network. The proposed method achieves an accuracy of 95.55%, while the second-best accuracy performance is 82.02 for the NB tree, showing a significant difference between the accuracy of our proposed method and the other approaches. This evaluation confirms that the proposed method is accurate and robust, even compared against other algorithms. This shows how the unambiguous steps in our approach are reliable, effective and authoritative.We conclude that the proposed method achieves a verifiable result using several techniques. For the precise literature and comparison, we carefully chose the NSL-KDD data set, which is considered one of the most powerful benchmark datasets. Several procedures of data statistics, cleaning and verification are performed on the dataset, which are very important in order to produce a smooth learning process with no obstacles, such as over- or under-fitting issues. This stage ensures that the proposed model has unified data and increases the value of data, which helps in decision-making. Feature normalization and selection clarifies the path for clear selection and intelligent preferences, using only 5 features. Subsequently, more detailed exploration and various comparisons are carried out, based on three machine learning algorithms, i.e., DT, RF, and XGBoost, in order to test their performance with different criteria and then select the best performing algorithm for our task. This shows that the selection is dependably proven and technically verified.

Conclusions and Future Work

NIDS in SDN-based machine learning algorithms has attracted significant attention in the last two decades because of the datasets and various algorithms proposed in machine learning, using only limited features for better detection of anomalies better and more efficient network security. In this study, the benchmarking dataset NSL-KDD is used for training and testing. Feature normalization, feature selection and data preprocessing techniques are used in order to improve and optimize the algorithm's performance for accurate prediction, as well as to facilitate a smooth training process with minimal time and resources. To select the appropriate algorithm, we compare three classical tree-based machine learning algorithms; Random Forest, Decision Trees and XGBoost. We examine them using a variety of evaluation metrics to find the disadvantages and advantages of using one or more. Using six different evaluation metrics, the proposed XGBoost model outperformed more than seven algorithms used in NIDS. The proposed method focused on detecting anomalies and protecting the SDN platform from attacks in real-time scenarios. The proposed methods performed two tasks simultaneously; to detect if there is an attack or not, and to determine the type of attack (Dos, probe, U2R, R2L). In future studies, more evaluation metrics will be carried out. We plan to implement the approach using several deep neural network algorithms, such as Auto-Encoder, Generative Adversarial Networks, and Recurrent neural networks, such as GRU and LSTM.These techniques have been proven in the literature to allow convenient anomaly detection approaches in NIDS applications. Also, we plan to compare these algorithms against each other and integrate one or more neural network architectures to extract more details of how we can implement an efficient anomaly detection system in NIDS, with lower consumption of time and resources. In addition, for a more solid basis for comparison, several benchmarking cyber security datasets, such as NSL-KDD, UNSW-NB15, CIC-IDS2017 will be conducted, in order to make sure that the selection of the proposed algorithm is not biased in any situation. These various datasets are generated in different environments and conditions, so more complex features will be available, more generalized attacks will be covered and the accuracy of the proposed algorithm will significantly increase, which could lead to a state-of-the art approach.

A Deep Learning Approach for Network Intrusion DetectionSystem

Introduction:

Network Intrusion Detection Systems (NIDSs) are essential tools for the network system administrators to detect various security breaches inside an organization's network. An NIDS monitors and analyzes the network traffic entering into or exiting from the network devices of an organization and raises alarms if an intrusion is observed. Based on the methods of intrusion detection, NIDSs are categorized into two classes: i) signature (misuse) based NIDS (SNIDS), and ii) anomaly detection based NIDS (ADNIDS). In SNIDS, e.g. Snort, attack signatures are preinstalled in the NIDS. A pattern matching is performed for the traffic against the installed signatures to detect an intrusion in the network. In contrast, an ADNIDS classifies network traffic as an intrusion when it observes a deviation from the normal traffic pattern. SNIDS is effective in the detection of . known attacks and shows high detection accuracy with less false-alarm rates. However, its performance suffers during detection of unknown or new attacks due to the limitation of attack signatures that can be installed beforehand in an IDS. ADNIDS, on the other hand, is well-suited for the detection of unknown and new attacks. Although ADNIDS produces

high false-positive rates, its theoretical potential in the identification of novel attacks has caused its wide acceptance among the research community. There are primarily two challenges that arise while developing an efficient and flexible NIDS for unknown future attacks. First, proper feature selections from the network traffic dataset for anomaly detection is difficult. The features selected for one class of attack may not work well for other categories of attacks due to continuously changing and evolving attack scenarios. Second, unavailability of labeled traffic dataset from real networks for developing an NIDS. Immense efforts are required to produce such a labeled dataset from the raw network traffic traces collected over a period or in real-time. Additionally, to preserve the confidentiality of the internal organizational network structure as well as the privacy of various users, network administrators are reluctant towards reporting any intrusion that might have occurred in their networks. Various machine learning techniques have been used to develop ADNIDSs, such as Artificial Neural Networks (ANN), Support Vector Machines (SVM), Naive-Bayesian (NB), Random Forests (RF), and Self-Organized Maps (SOM). The NIDSs are developed as classifiers to differentiate the normal traffic from the anomalous traffic. Many NIDSs perform a feature selection task to extract a subset of relevant features from the traffic dataset to enhance classification results. Feature selection helps in the elimination of the possibility of incorrect training through the removal of redundant features and noises. Recently, deep learning based methods have been successfully applied in audio, image, and speech processing applications. These methods aim to learn a good feature representation from a large amount of unlabeled data and subsequently apply these learned features on a limited amount of labeled data in a supervised classification. The labeled and unlabeled data may come from different distributions. However, they must be relevant to each other. It is envisioned that the deep learning based approaches can help to overcome the challenges of developing an efficient NIDS. We can collect unlabeled network traffic data from different network sources and a good feature representation from these datasets using deep learning techniques can be obtained. These features can, then, be applied for supervised classification to a small, but labeled traffic dataset consisting of normal as well as anomalous traffic records. The traffic data for labeled dataset can be collected in a confined, isolated and private network environment. With this motivation, we use self-taught learning, a deep learning technique based on sparse auto encoder and soft-max regression, to develop an NIDS. We verify the usability of the self-taught learning based NIDS by applying on NSL-KDD intrusion dataset, an improved version of the benchmark dataset for various NIDS evaluations - KDD Cup 99. We provide a comparison of our current work with other techniques as well. Towards this end, our paper is organized into four sections. In Section 2, we discuss a few closely related work. Section 3 presents an overview of self-taught learning and the NSL-KDD dataset. We discuss our results and comparative analysis in Section 4 and finally conclude our paper with future work direction in Section 5. 2.

RELATED WORK :

This section presents various recent accomplishments in this area. It should be noted that we only discuss the work that have used the NSL-KDD dataset for their performance bench marking. Therefore, any dataset referred from this point forward should be considered as NSL-KDD. This approach allows a more accurate comparison of work with other found in the literature. Another limitation is the use of training data for both training and testing by most work. Finally, we discuss

a few deep learning based approaches that have been tried so far for similar kind of work. One of the earliest work found in literature used ANN with enhanced resilient back-propagation for the design of such an IDS. This work used only the training dataset for training (70%), validation (15%) and testing (15%). As expected, use of unlabeled data for testing resulted in a reduction of performance. A more recent work used J48 decision tree classifier with 10-fold cross-validation for testing on the training dataset. This work used a reduced feature set of 22 features instead of the full set of 41 features. A similar work evaluated various popular supervised tree-based classifiers and found that Random Tree model performed best with the highest degree of accuracy along with a reduced false alarm rate. Many 2-level classification approaches have also been proposed. One such work used Discriminative Multinomial Naive Bayes (DMNB) as a base classifier and Nominal-to Binary supervised filtering at the second level along with 10-fold cross validation for testing. This work was further extended to use Ensembles of Balanced Nested Dichotomies (END) at the first level and Random Forest at the second level. As expected, this enhancement resulted in an improved detection rate and a lower false positive rate. Another 2-level implementation used principal component analysis (PCA) for the feature set reduction and then SVM (using Radial Basis Function) for final classification, resulted in a high detection accuracy with only the training dataset and full 41 features set. A reduction in features set to 23 resulted in even better detection accuracy in some of the attack classes, but the overall performance was reduced. The authors improved their work by using information gain to rank the features and then a behaviorbased feature selection to reduce the feature set to 20. This resulted in an improvement in reported accuracy using the training dataset. The second category to look at, used both the training and test dataset. An initial attempt in this category used fuzzy classification with genetic algorithm and resulted in a detection accuracy of 80%+ with a low false positive rate. Another important work used unsupervised clustering algorithms and found that the performance using only the training data was reduced drastically when test data was also used . A similar implementation using the k-point algorithm resulted in a slightly better detection accuracy and lower false positive rate, using both training and test datasets. Another less popular technique, OPF (optimumpath forest) which uses graph partitioning for feature classification, was found to demonstrate a high detection accuracy within one-third of the time compared to SVMRBF method. We observed a deep learning approach with Deep Belief Network (DBN) as a feature selector and SVM as a classifier in. This approach resulted in an accuracy of 92.84% when applied on training data. Our current work could be easily compared to this work due to the enhancement of approach over this work and use of both the training and test dataset in our work. A similar, however, semi-supervised learning approach has been used in. The authors used real-world trace for training and evaluated their approach on real-world and KDD Cup 99 traces. Our approach is different from them in the sense that we use NSL-KDD dataset to find deep learning applicability in NIDS implementation. Moreover, the feature learning task is completely unsupervised and based on sparse autoencoder in our approach. We recently observed a sparse autoencoder based deep learning approach for network traffic identification in. The authors performed TCP based unknown protocols identification in their work instead of network intrusion detection

Self-Taught Learning:

Self-taught Learning (STL) is a deep learning approach that consists of two stages for the classification. First, a good feature representation is learnt from a large collection of unlabeled data, xu, termed as Unsupervised Feature Learning (UFL). In the second stage, this learnt representation is applied to labeled data, xl, and used for the classification task. Although the unlabeled and labeled data may come from different distributions, there must be relevance among them. Figure 1 shows the architecture diagram of STL. There are different approaches used for UFL, such as Sparse Auto encoder, Restricted Boltzmann Machine (RBM), K-Means Clustering, and Gaussian Mixtures. We use sparse auto encoder based feature learning for our work due to its relatively easier implementation and good performance. A sparse autoencoder is a neural network consists of an input, a hidden, and an output layers. The input and output layers contain N nodes, and the hidden layer contains K nodes. The target values at the output layer are set equal to the input values, i.e., ˆxi = xi The sparse auto encoder network finds the optimal values for weight matrices, W ∈ <K×N and V ∈ <N×K, and bias vectors, b1 ∈ <K×1 and b2 ∈ <N×1 , using back-propagation algorithm while trying to learn the approximation of the identity function, i.e., output ˆx similar to x [18]. Sigmoid function, g(z) = 1 1+e−z , is used for the activation, hW,b of the nodes in the hidden and output layers: hW,b(x) = g(W x + b) (1) J = 1 2m Xm i=1 kxi − xˆik 2 + λ 2 (X k,n W2 + X n,k V 2 + X k b1 2 + X n b2 2) + β XK j=1 KL(ρkρˆj) (2) The cost function to be minimized in sparse auto encoder using back-propagation is represented by Eq. (2). The first term is the average of sum-of-square error terms for all m input data. The second term is a weight decay term, with λ as weight decay parameter, to avoid the over-fitting in training. The last term in the equation is sparsity penalty term that puts a constraint into the hidden layer to maintain a low average activation values, and expressed as KullbackLeibler (KL) divergence shown in Eq. (3): KL(ρkρˆj) = ρlog ρ ρˆj + (1 − ρ)log 1 − ρ 1 − ρˆj (3) where ρ is a sparsity constraint parameter ranges from 0 to 1 and β controls the sparsity penalty term. The KL(ρkρˆj) attains a minimum value when ρ = ˆρj , where ˆρj denotes the average activation value of hidden unit j over all training inputs x. Once we learn optimal values for W and b1 by applying the sparse autoencoder on unlabeled data, xu, we evaluate the feature representation a = hW,b1 (xl) for the labeled data, (xl, y). We use this new features representation, a, with the labels vector, y, for the classification task in the second stage. We use soft-max regression for the classification task

NSL-KDD Dataset:

As discussed earlier, we used NSL-KDD dataset in our work. The dataset is an improved and reduced version of the KDD Cup 99 dataset . The KDD Cup dataset was prepared using the network traffic captured by 1998 DARPA IDS evaluation program. The network traffic includes normal and different kinds of attack traffic, such as DoS, Probing, user-to-root (U2R), and root-to-local (R2L). The network traffic for training was collected for seven weeks followed by two weeks of traffic collection for testing in raw tcpdump format. The test data contains many attacks that were not injected during the training data collection phase to make the intrusion detection task realistic. It is believed that most of the novel attacks can be derived from the known attacks. Finally, the training and test data were processed into the datasets of five million and two million TCP/IP connection records, respectively. The KDD Cup dataset has been widely used as a benchmark dataset for many years in the evaluation of NIDS. One of the major drawback with the dataset is that it contains an enormous amount of redundant records both in the training and

test data. It was observed that almost 78% and 75% records are redundant in the training and test dataset, respectively. This redundancy makes the learning algorithms biased towards the frequent attack records and leads to poor classification results for the infrequent, but harmful records. The training and test data were classified with the minimum accuracy of 98% and 86% respectively using a very simple machine learning algorithm. It made the comparison task difficult for various IDSs based on different learning algorithms. NSL-KDD was proposed to overcome the limitation of KDD Cup dataset. The dataset is derived from the KDD Cup dataset. It improved the previous dataset in two ways. First, it eliminated all the redundant records from the training and test data. Second, it partitioned all the records in the KDD Cup dataset into various difficulty levels based on the number of learning algorithms that can correctly classify the records. Further, it selected the records by random sampling of the distinct records from different difficulty levels in a fraction that is inversely proportional to their fractions in the distinct records. The multi-steps processing of KDD Cup dataset made the total records statistics reasonable in the NSL-KDD dataset. Moreover, these enhancements made the evaluation of various machine learning techniques realistic. Each record in the NSL-KDD dataset consists of 41 features1 and is labeled with either normal or a particular kind of attack. These features include basic features derived directly from a TCP/IP connection, traffic features accumulated in a window interval, either time, e.g. two seconds, or a number of connections, and content features extracted from the application layer data of connections. Out of 41 features, three are nominal, four are binary, and remaining 34 are continuous. The training data contains 23 traffic classes that include 22 classes of attack and one normal class. The test data contains 38 traffic classes that include 21 attack classes from the training data, 16 novel attacks, and one normal class. These attacks are also grouped into four categories based on the purpose they serve. These categories are DoS, Probing, U2R, and R2L. Table-1 shows the statistics of records for the training and test data for normal and different attack classes.

NIDS Implementation:

As discussed in the previous section, the dataset contains different kinds of attributes with different values. We pre process the dataset before applying self-taught learning on it. Nominal attributes are converted into discrete attributes using 1-to-n encoding. In addition, there is one attribute, num outbound cmds, in the dataset whose value is always 0 for all the records in the training and test data. We eliminated this attribute from the dataset. The total number of attributes become 121 after performing the steps mentioned above. The values in the output layer during the feature learning phase is computed by the sigmoid function that gives values between 0 and 1. Since the output layer values are identical to the input layer values in this phase, it results in normalization of the values at the input layer in the range of. To obtain this, we perform max-min normalization on the new attributes list. With the new attributes, we use the NSL-KDD training data without labels for feature learning using sparse autoencoder for the first stage of self-taught learning. In the second stage, we apply the newly learned features representation on the training data itself for the classification using soft-max regression. In our implementation, both the unlabeled and labeled data for feature learning and classifier training come from the same source, i.e., NSL-KDD training data.

Performance Evaluation:
We implemented the NIDS for three different types of classification: a) Normal and anomaly (2-class), b) Normal and four different attack categories (5-class), and c) Normal and 22 different attacks (23-class). We have evaluated classification accuracy for all types. However, precision, recall, and f-measure values are evaluated in the case of 2-class and 5- class classification. We have computed the weighted values of these metrics in the case of 5-class classification.

Results and Discussion:
As discussed in Section 2, there are two approaches applied for the evaluation of NIDSs. In the most widely used approach, the training data is used for both training and testing either using n-fold cross-validation or splitting the training data into training, cross-validation, and test sets. NIDSs based on this approach achieved very high accuracy and less false-alarm rates. The second approach uses the training and test data separately for the training and testing. Since the training and test data were collected in different environments, the accuracy obtained using the second approach is not as high as in the first approach. Therefore, we emphasize on the results of the second approach in our work for accurate evaluation of NIDS. However, we present the results of the first approach as well for completeness. We describe our NIDS implementation before discussing the results.

Conclusion:
We proposed a deep learning based approach for developing an efficient and flexible NIDS. A sparse auto encoder and soft-max regression based NIDS was implemented. We used the benchmark network intrusion dataset - NSL-KDD to evaluate anomaly detection accuracy. We observed that the proposed NIDS performed very well compared to previously implemented NIDSs for the normal/anomaly detection when evaluated on the test data. The performance can be further enhanced by applying techniques such as Stacked Auto encoder, an extension of sparse auto encoder in deep belief nets, for unsupervised feature learning, and NB-Tree, Random Tree, or J48 for further classification. It was noted that the latter techniques performed well when applied directly on the dataset. In future, we plan to implement a real-time NIDS for actual networks using deep learning technique. Additionally, on-the-go feature learning on raw network traffic headers instead of derived features can be another high impact research in this area.

Intrusion Preventing System using Intrusion Detection System Decision Tree Data Mining

Abstract:
Problem statement: To distinguish the activities of the network traffic that the intrusion and normal is very difficult and to need much time consuming. An analyst must review all the data that large and wide to find the sequence of intrusion on the network connection. Therefore, it needs a way that can detect network intrusion to reflect the current network traffics.

Approach: In this study, a novel method to find intrusion characteristic for IDS using decision tree machine learning of data mining technique was proposed. Method used to generate of rules is classification by ID3 algorithm of decision tree.
Results: These rules can determine of intrusion characteristics then to implement in the firewall policy rules as prevention.
Conclusion: Combination of IDS and firewall so-called the IPS, so
that besides detecting the existence of intrusion also can execute by doing deny of intrusion as prevention.

INTRODUCTION

With the global Internet connection, network security has gained significant attention in research and industrial communities. Due to the increasing threat of network attacks, firewalls have become important elements of the security policy is generally. Firewall can be allow or deny access network packet, but firewall cannot detect intrusion or attack, so to need intrusion detection and then implemented to firewall is access control systems as prevention. Intrusion detection are also considered as a complementary solution to firewall technology by recognizing attacks against the network that are missed by the firewall. Firewall and IDS represent an old stuff terminology in the field of IT security. Firewall is good for protection a system and network and can minimization risk of attack to network. IDS can detect existence intrusion or attack. The joining ability of IDS and firewalls, that is socalled IPS. That is a functioning tool to detect intrusion and then denying by firewall for prevention. For each type of network traffics, there are one or more different rules. Every network packet, which arrives at firewall, must be check against defined rules until a matching rule found. The packet will be then allow or banned access to the network, depending on the action specified in the matching rule. Each rule identifies specific type of network traffic. Characteristics to reflect the current of network traffics can observe from network traffic logs as human pattern recognize. This Study focus on some methods to prevention from attempt intrusion to find intrusion characteristics in the network traffic as IDS then implementation to firewall policy rules as prevention. To find rules of intrusion characteristics using decision tree machine learning data mining. Method used to generate of rules is classification by ID3 algorithm of decision tree. It is an efficient and optimized to make the rules filtering in firewall.

Theoretical background:

Intrusion Detection System (IDS): Intrusion detection can be performed manually or automatically. Manual intrusion detection might take place by examining log files or other evidence for signs of intrusions, including network traffic. A system that performs automated intrusion detection is called an Intrusion Detection System (IDS). IDS play a vital role in ensuring the security of modern computer installations. Such systems are need in order to detect hostile activity and to respond appropriately. As networks continue to expand and become more exposed to a diversity of sources, both hostile and benign, IDS need to be able to deal with a large and ever-increasing flow of alerts and events. Therefore, automatic procedures for detecting and responding to intrusion are becoming increasingly essential.

Firewall rules:

A firewall security policy is a list of ordered filtering rules that define the actions performed on packets that satisfy specific conditions. Before to develop rules filtering by using packet filter, anything have to be considered beforehand how far demarcation which will be applied, because more and more demarcation applied hence increases the search time and space requirements of the packet filtering process[1] and consequences to make downhill performance progressively. This matter because every incoming network packet and go out the network checked beforehand by rules alternately until matching rule found in firewall. Firewall rules can limit to access the connection of pursuant to parameter: source IP, destination IP, source port, destination port, protocol and others[8,10] . Following example of firewall rules in Fig. 1. Firewall rule of above explaining to enhance the order by the end of chain (A) for the traffic of incoming to firewall (INPUT) by source IP address (-s) 203.230.206.5 with the type protocol (-p) tcp to destination IP address (-d) 10.10.15.7 and destination port (--dport) 80 hence done by action (-j) dropped (DROP) by firewall.

Log files:

Log files can give an idea about what the different parts of system are doing. Logs can show what is going right and what is going wrong. Log files can provide a useful profile activity. From a security standpoint, it is crucial to be able to distinguish normal activity from the activity of someone to attack server or network.

Log files are useful for three reasons:

• Log files help with troubleshooting system problems and understanding what is happening on the system

• Logs serve as an early warning for both system and security events

• Logs can be indispensable in reconstructing events, whether determined an intrusion has occurred and performing the follow-up forensic investigation or just profiling normal activity

Decision tree of data mining:

Decision tree is a technique in classification method of data mining for learning patterns from data and using these patterns for classification. Decision tree are structures used to classify and data and with and common and attributes and as Each decision tree represents a rule, which categorizes data according to these attributes.

Where each node (nonleaf node) denotes a test on an attribute, each branch represent an outcome of the test and each leaf node or terminal node holds a class label. The topmost node in a tree is the root node. A decision tree classifier is one of the most widely need supervised learning methods used for data exploration. It is easy to interpret and can be represented as if-then-else rules. This classifier works well on noisy data. A decision tree aids in data exploration in the following manner:

• It reduces a volume of data by transformation into a more compact form that preserves the essential characteristics and provides an accurate summary

• It discovers whether the data contains well separated classes of objects, such that the classes can be interpreted meaningfully in the context of a substantive theory

• It maps data in the form the leaves to its root. This may used to predict the outcome for a new data or Query.

MATERIALS AND METHODS:

This research using decision tree a technique of data mining machine learning to find the intrusion characteristics for intrusion detection. Algorithm is used ID3 to construct Decision tree. Network traffic logs as data training that describes the human behavior
in network traffics as intrusive activities and normal activities. The results of decision tree training will get rules of intrusion characteristics then these rules to implement in the firewall rules as prevention. Determining occurrence of intrusion or normal
activities at network traffic log can be conducted with two way of that is:

• Observe manually activities network traffic in log files. Example, application software of log files is syslog, syslog_ng, tcpdump and others. Pattern
found to see intrusion through log seen modestly, for example there are some times trying to access using login or password failed, trying port scan, abundant ping, delivery of abundant package by repeat

• Using software as a means of assists functioning as Network Intrusion Detection System (NIDS) able to determine intrusion activities or normal activities, for example snort software

RESULTS:

Collect and extract log files of intrusive activities and normal activities become five of parameter as attributes and belongs to a class ‘Yes’ or ‘No’ of intrusive for the data training of decision tree. The parameter is IP address source, IP address destination,
port source, port destination and protocol . Applying Decision Tree to Find Intrusion Characteristic: Suppose train a decision tree.

Implementation to firewall rules:

The examples of extract rule of tree decision representing characteristic of intrusion earn implementation into firewall rules .Do not forget to every rule there is a TCP protocol. Firewall policy rules above representing preventive action, where every network packet with criteria like rules firewall above will DROP.

CONCLUSION:

Network traffic logs to describe patterns of behavior in network traffic accident with intrusive or normal activity. Decision tree technique is good for the intrusion characteristic of the network traffic logs for IDS and implemented in the firewall as prevention. The both of this combination is called IPS. The other hand, this technique is also good efficiency and optimize rule for the firewall rules such as avoid redundancy.

A Deep Learning Approach to Network Intrusion Detection

Abstract:
Network Intrusion Detection Systems (NIDS) play a crucial role in defending computer networks. However, there are concerns regarding the feasibility and sustainability of current approaches when faced with the demands of modern networks. More specifically, these concerns relate to the increasing levels of required human interaction and the decreasing levels of detection accuracy. This paper presents a novel deep learning technique for intrusion detection, which addresses these concerns. We detail our proposed non-symmetric deep auto-encoder (NDAE) for unsupervised feature learning. Furthermore, we also propose our novel deep learning classification model constructed using stacked NDAEs. Our proposed classifier has been implemented in GPU-enabled TensorFlow and evaluated using the benchmark KDD Cup '99 and NSL-KDD datasets. Promising results have been obtained from our model thus far, demonstrating improvements over existing approaches and the strong potential for use in modern NIDSs.

INTRODUCTION:

one of the major challenges in network security is the provision of a robust and effective Network Intrusion Detection System (NIDS). Despite the significant advances in NIDS technology, the majority of solutions still operate using less-capable signature-based techniques, as opposed to anomaly detection techniques. There are several reasons for this reluctance to switch, including the high false error
rate (and associated costs), difficulty in obtaining reliable training data, longevity of training data and behavioural dynamics of the system. The current situation will reach a point whereby reliance on such techniques leads to ineffective and inaccurate detection. The specifics of this challenge are to create a widely-accepted anomaly detection technique capable of overcoming limitations induced by the ongoing
changes occurring in modern networks. We are concerned with three main limitations, which contribute to this network security challenge. The first is the drastic growth in the volume of network data, which is set to continue. This growth can be predominantly attributed to increasing levels of connectivity, the popularity of the
Internet of Things and the extensive adoption of cloud based services. Dealing with these volumes requires techniques that can analyse data in an increasingly rapid, efficient and effective manner. The second cause is the in-depth monitoring and granularity required to improve effectiveness and accuracy. NIDS analysis needs to be more detailed and contextually-aware, which means shifting away from ab
stract and high-level observations. For example, behavioural changes need to be easily attributable to specific elements of a network, e.g. individual users, operating system versions or protocols. The final cause is the number of different
traversing through modern networks. This is possibly the most significant challenge and introduces high-levels of difficulty and complexity when attempting to differentiate between normal and abnormal behaviour. It increases the difficulty in establishing an accurate norm and widens the scope for potential exploitation or

zero-day attacks. In recent years, one of the main focuses within NIDS research has been the application of machine learning and shallow learning techniques such as Naive Bayes, Decision Trees and Support Vector Machines (SVM) [1]. By enlarge,
the application of these techniques has offered improvements in detection accuracy. However, there are limitations with these techniques, such as the comparatively high level of human expert interaction required; expert knowledge is needed to process data e.g. identifying useful data and patterns. Not only is this a labour intensive and expensive process but it is also error prone [2]. Similarly, a vast quantity of training data is required for operation (with associated time overheads), which can become challenging in a heterogeneous and dynamic environment. To address the above limitations, a research area currently receiving substantial interest across multiple domains is that of deep learning. This is an advanced subset of machine learning, which can overcome some of the limitations of shallow learning. Thus far, initial deep learning research has demonstrated that its superior layer-wise feature learning can better or at least match the performance of shallow learning techniques. It is capable of facilitating a deeper analysis of network data and faster identification of any anomalies.In this paper, we propose a novel deep learning model to enable NIDS operation within modern networks. The model we propose is a combination of deep and shallow learning, capable of correctly analysing a wide-range of network traffic. More specifically, we combine the power of stacking our proposed non-symmetric deep auto-encoder (NDAE) (deep-learning) and the accuracy and speed of Random Forest (RF)(shallow learning). We have practically evaluated our model using GPU-enabled TensorFlow and obtained promising results from analysing the KDD Cup ’99 and NSL-KDD datasets. We are aware of the limitations of these datasets but they remain widely-used benchmarks amongst similar works, enabling us to draw direct comparisons. This paper offers the following novel contributions:

• A new NDAE technique for unsupervised feature learning, which unlike typical auto-encoder approaches provides non-symmetric data dimensionality reduction. Hence, our technique is able to facilitate improved classification results when compared with leading methods such as Deep Belief Networks (DBNs).

• A novel classifier model that utilises stacked NDAEs and the RF classification algorithm. By combining both deep and shallow learning techniques to exploit
their respective strengths and reduce analytical over heads. We are able to better or at least match results from similar research, whilst significantly reducing the training time. The remainder of this paper is structured as follows. Section 2 presents relevant background information. Section 3 examines existing research. Section 4 specifies our proposed solution, which is subsequently evaluated in Section 5. Section 6 discusses our findings from the evaluation. Finally the paper concludes in Section 7.

BACKGROUND

In this section, we will provide background information necessary to understand our motivations and the concepts behind the model proposed in this paper.

NIDS challenges

Network monitoring has been used extensively for the purposes of security, forensics and anomaly detection. However, recent advances have created many new obstacles for NIDSs. Some of the most pertinent issues include:

• **Volume** - The volume of data both stored and passing through networks continues to increase. It is forecast that by 2020, the amount of data in existence will top 44ZB [4]. As such, the traffic capacity of modern networks has drastically increased to facilitate the volume of traffic observed. Many modern backbone links are now operating at wirespeeds of 100Gbps or more. To contextualise this, a 100Gbps
link is capable of handling 148,809,524 packets per second [5]. Hence, to operate at wirespeed, a NIDS would need to be capable of completing the analysis of a packet within 6.72ns. Providing NIDS at such a speed is difficult and ensuring satisfactory levels of accuracy, effectiveness and efficiency also presents a significant challenge.

• **Accuracy** - To maintain the aforementioned levels of accuracy, existing techniques cannot be relied upon. Therefore, greater levels of granularity, depth and contextual understanding are required to provide a more holistic and accurate view.Unfortunately,
this comes with various financial, computational and time costs.

• **Diversity** - Recent years have seen an increase in the number of new or customised protocols being utilised in modern networks. This can be partially attributed to the number of devices with network and/or Internet connectivity. As a result, it is be
coming increasingly difficult to differentiate between normal and abnormal traffic and/or behaviours.

• **Dynamics** - Given the diversity and flflexibility of modern networks, the behaviour is dynamic and difficult to predict. In turn, this leads to difficulty in establishing a reliable behavioural norm. It also raises concerns as to the lifespan of learning models.

• **Low-frequency attacks** - These types of attacks have often thwarted previous anomaly detection techniques, including artificial intelligence approaches.The problem stems from imbalances in the training dataset, meaning that NIDS offer weaker detection precision when faced with these types of low frequency attacks.

• **Adaptability** - Modern networks have adopted many new technologies to reduce their reliance on static technologies and management styles. Therefore, there is more widespread usage of dynamic technologies such as containerisation, virtualisation
and Software Defined Networks. NIDSs will need to be able to adapt to the usage of such technologies and the side effects they bring about.

Deep Learning

Deep learning is an advanced sub-field of machine learning,which advances Machine Learning closer to Artificial Intelligence. It facilitates the modelling of complex relationships and concepts using multiple levels of representation.
Supervised and unsupervised learning algorithms are used to construct successively higher levels of abstraction, defined using the output features from lower levels.

- **Auto-encoder**

A popular technique currently utilised within deep learning research is auto-encoders, which is utilised by our proposed solution (detailed in Section 4). An auto-encoder is an unsupervised neural network-based feature extraction algorithm, which learns the best parameters required to

reconstruct its output as close to its input as possible. One of it desirable characteristics is the capability to provide more a powerful and non-linear generalisation than Principle Component Analysis (PCA). This is achieved by applying backpropagation and setting the target values to be equal to the inputs. In other words, it is trying to learn an approximation to the identity function. An auto-encoder typically has an input layer, output layer (with the same dimension as the input layer) and a hidden layer. This hidden layer normally has a smaller dimension than that of the input (known as an undercomplete or sparse auto-encoder). An example of an auto-encoder.Most researchers use auto-encoders as a nonlinear transformation to discover interesting data structures,by imposing other constraints on the network, and compare the results with those of PCA (linear transformation). These
methods are based on the encoder-decoder paradigm. The input is first transformed into a typically lower-dimensional space (encoder), and then expanded to reproduce the initial data (decoder). Once a layer is trained, its code is fed to the next, to better model highly non-linear dependencies in the input. This paradigm focuses on reducing the dimensionality of input data. To achieve this, there is a special layer
- the code layer, at the centre of the deep auto-encoder structure. This code layer is used as a compressed feature vector for classification or for combination within a stacked auto-encoder.
Deep learning can be applied to auto-encoders, whereby the hidden layers are the simple concepts and multiple hidden layers are used to provide depth, in a technique
known as a stacked auto-encoder. This increased depth can reduce computational costs and the amount of required training data, as well as yielding greater degrees of accuracy. The output from each hidden layer is used as the input for a progressively higher level. Hence, the first layer of a stacked auto-encoder usually learns first-order features in raw input. The second layer usually learns second-order features relating to patterns in the appearance of the first order features. Subsequent higher layers learn higher-order features. An illustrative example of a stacked auto-encoder. Here, the superscript numbers refer to the hidden layer identity and the subscript numbers signify the dimension for that layer.

EXISTING WORK :

Deep learning is garnering significant interest and its application is being investigated within many research domains, such as: healthcare; automotive design; manufacturing and law enforcement. There are also several existing works within the domain of NIDS. In this section, we will discuss the most current notable works.Dong and Wang undertook a literary and experimental comparison between the use of specific traditional NIDS techniques and deep learning methods [1]. The authors concluded that the deep learning-based methods offered improved detection accuracy across a range of sample sizes and traffic anomaly types. The authors also demonstrated that problems associated with imbalanced datasets can be overcome by using oversampling for which, they used the Synthetic Minority Oversampling Technique (SMOTE).Zhao presented a state-of-the-art survey of deep learning applications within machine health monitoring.They experimentally compared conventional machine learning methods against four common deep learning methods
(auto-encoders, Restricted Boltzmann Machine (RBM), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Their work concluded that deep learning methods offer

better accuracy than conventional methods. Our literature review identified several proposed deep learning methods specifically for NIDSs.

Alrawashdeh and Purdy [18] proposed using a RBM with one hidden layer to perform unsupervised feature reduction. The weights are passed to another RBM to produce a DBN. The pre-trained weights are passed into a fine tuning layer consisting of a Logistic Regression classifier (trained with 10 epochs) with multi-class soft-max. The proposed solution was evaluated using the KDD Cup '99 dataset. The authors claimed a detection rate of 97.90% and a false negative rate of 2.47%. This is an improvement over results claimed by authors of similar papers.The work by Kim *et al*. [19] aspired to specifically target advanced persistent threats. They propose a Deep Neural Network (DNN) using 100 hidden units, combined with the Rectified Linear Unit activation function and the ADAM optimiser. Their approach was implemented on a GPU using TensorFlow, and evaluated using the KDD data set. The authors claimed an average accuracy rate of 99%, and summarised that both RNN and Long Short-Term Memory (LSTM) models are needed for improving future defences.

Javaid *et al*. [20] propose a deep learning based approach to building an effective and flflexible NIDS. Their method is referred to as self-taught learning (STL), which combines a sparse auto-encoder with softmax regression. They have implemented their solution and evaluated it against the benchmark NSL-KDD dataset. The authors claim some promising levels of classification accuracy in both binary and 5-class classification. Their results show that their 5-class classification achieved an average f-score of 75.76%. Potluri and Diedrich [21] propose a method using 41 features and their DNN has 3 hidden layers (2 auto-encoders and 1 soft-max). The results obtained were mixed, those focusing on fewer classes were more accurate than those with more classes. The authors attributed this to insufficient training data for some classes.Cordero et al. [22] proposed an unsupervised method to learn models of normal network flflows. They use RNN, auto-encoder and the dropout concepts of deep learning.The exact accuracy of their proposed method evaluated is not fully disclosed.Similarly, Tang et al. [23] also propose a method to monitor network flflow data. The paper lacked details about its exact algorithms but does present an valuation using the NSL-KDD dataset, which the authors claim gave an accuracy of 75.75% using six basic features. Kang and Kang [24] proposed the use of an unsupervised DBN to train parameters to initialise the DNN, which yielded improved classification results (exact details of the approach are not clear). Their evaluation shows improved performance in terms of classification errors. Hodo *et al*. [25] have produced a comprehensive taxonomy and survey on notable NIDSs approaches that utilise deep and shallow learning. They have also aggregated some of the most pertinent results from these works. In addition, there is other relevant work, including the DDoS detection system proposed by Niyaz *et al*. [26]. They propose a deep learning-based DDoS detection system for a software defined network (SDN). Evaluation is performed using custom generated traffic traces. The authors claim to have achieved binary classification accuracy of 99.82% and 8-class classification accuracy of 95.65%. However, we feel that drawing comparisons with this paper would be unfair due to the contextual difference of the dataset. Specifically, benchmark KDD datasets cover different distinct categories of attack, whereas the dataset used in this paper focuses on subcategories of the same attack. You et al. [16] propose an automatic

security auditing tool for short messages (SMS). Their method is based upon the RNN model. The authors claimed that their evaluations resulted in an accuracy rate of 92.7%, thus improving existing classification methods (e.g. SVM and Naive Bayes). Wang et al. [27] propose an approach for detecting
malicious JavaScript. Their method uses a 3 layer SdA with linear regression. It was evaluated against other classifier techniques, showing that it had the highest true positive rate but the second best false positive rate.The work by Hou et al. [3] outlines their commercial Android malware detection framework, Deep4MalDroid. Their method involves the use of stacked auto-encoders with best accuracy resulting from 3 layers. The 10-fold cross validation was used, showing that in comparison to shallow learning, their approach offers improved detection performance.Lee et al. [28] propose a deep-learning approach to fault monitoring in semiconductor manufacturing. They use a Stacked de-noising Auto-encoder (SdA) approach to provide an unsupervised learning solution. A comparison with conventional methods has demonstrated that throughout different use cases the approach increases accuracy by up to 14%. in different use cases. They also concluded that among the SdAs analysed (1-4 layers) those with 4 layers produced the best results. The findings from our literature review have shown that despite the high detection accuracies being achieved, there is still room for improvement. Such weaknesses include the reliance on human operators, long training times, inconsistent or average accuracy levels and the heavy modification of datasets (e.g. balancing or profiling). The area is still in an infantile stage, with most researchers still experimenting on combining various algorithms (e.g. training, optimisation, activation and classification) and layering approaches to produce the most accurate and efficient solution for a specific dataset. Hence, we believe the model and work presented in this paper will be able to make a valid contribution to the current pool of knowledge.

PROPOSED METHODOLOGY

- **Non-symmetric deep auto-encoder:**

Decreasing the reliance on human operators is a crucial requirement for future-proofing NIDSs. Hence, our aim is to devise a technique capable of providing reliable unsupervised feature learning, which can improve upon the performance and accuracy of existing techniques.This paper introduces our NDAE, which is an auto
encoder featuring non-symmetrical multiple hidden layers.Fundamentally, this involves the proposed shift from the encoder-decoder paradigm (symmetric) and towards utilising just the encoder phase (non-symmetric). The reasoning behind this is that given the correct learning structure, it is be possible to reduce both computational and time overheads, with minimal impact on accuracy and efficiency. NDAE can be used as a hierarchical unsupervised feature extractor that scales well to accommodate high-dimensional inputs. It learns non-trivial features using a similar training strategy to that of a typical auto-encoder.

- **Stacked non-symmetric deep auto-encoders:**

Due to the data that we envisage this model using,we have designed the model to handle large and complex datasets (further details on this are provided in 6). Despite the 42 features present in the KDD Cup ’99 and NSL-KDD datasets being comparatively small, we maintain that it provides a benchmark indication as to the model’s capability. However, the classification power of stacked

auto encoders with a typical soft-max layer is relatively weak compared to other discriminative models including RF, KNN and SVM. Hence, we have combined the deep learning power of our stacked NDAEs with a shallow learning classifier. For our shallow learning classifer, we have decided upon using Random Forest. Current comparative research such as that by Choudhury and Bhowal, and Anbar *et al*.shows that RF is one of the best algorithms for intrusion detection. These are findings that were replicated by our own initial tests. Additionally, there are many examples of current intrusion detection research also utilising RF is basically an ensemble learning method, the principle of which is to group 'weak learners' to form a 'strong learner' . In this instance, numerous individual decision trees (the weak learners) are combined to form a forest. RF can be considered as the bagging (records are selected at random with replacement from the original data) of these un-pruned decision trees, with a random selection of features at each split. It boasts advantages such as low levels of bias, robustness to outliers and overfitting correction, all of which would be useful in a NIDS scenario. In our model, we train the RF classifier using the encoded representations learned by the stacked NDAEs to classify network traffic into normal data and known attacks. In deep learning research, the exact structure of a model dictates its success. Currently, researchers are unable to explain what makes a successful deep learning structure. The exact structure of our model has resulted from experimented with numerous structural compositions to achieve the best results.

EVALUATION & RESULTS:

Similar to most existing deep learning research, our proposed classification model (Section 4.2) was implemented using TensorFlow. All of our evaluations were performed using GPU-enabled TensorFlow running on a 64-bit Ubuntu 16.04 LTS PC with an Intel Xeon 3.60GHz processor, 16 GBRAM and an NVIDIA GTX 750 GPU.

To perform our evaluations, we have used the KDD Cup'99 and NSL-KDD datasets. Both of these datasets are considered as benchmarks within NIDS research. Furthermore,using these datasets assists in drawing comparisons with existing methods and research.Throughout this section, we will be using the metrics
defined below:

• *True Positive (TP)* - Attack data that is correctly clas
sified as an attack.

• *False Positive (FP)* - Normal data that is incorrectly
classified as an attack.

• *True Negative (TN)* - Normal data that is correctly
classified as normal.

• *False Negative (FN)* - Attack data that is incorrectly
classified as normal.

Datasets

This paper utilises the KDD Cup '99 and NSL-KDD benchmark datasets. Both of which have been used extensively in IDS research involving traffic with both normal and abnormal connections.

5.1.1 KDD Cup '99

The KDD Cup ’99 dataset was used in DARPA’s IDS evaluation program. The data consists of 4 gigabytes-worth of compressed tcpdump data resulting from 7 weeks of
network traffic. This can be processed into about 5 million connection records, each with about 100 bytes. It consists of approximately 4,900,000 single connection vectors each of which contains 41 features. These include Basic features (e.g.protocol type, packet size), Domain knowledge features (e.g.number of failed logins) and timed observation features (e.g.% of connections with SYN errors). Each vector is labelled as either normal or as an attack It is common practice to use 10% of the full size dataset, as this provides a suitable representation with reduced computational requirements. This 10% subset is produced and disseminated alongside the original dataset. In this paper, we use the 10% (herein referred to as KDD Cup ’99) subset,
which contains 494,021 training records and 311,029 testing records. The KDD Cup ’99 dataset needs pre-processing to be successfully utilised with our proposed stacked NDAE model. This is because our model operates using only numeric
values but one record in the dataset has a mixture of numeric and symbolic values, so a data transformation was needed to convert them. In addition integer values also need
normalisation as they were mixed with flfloating point values between 0 and 1, which would make learning difficult.

NSL-KDD

The newer NSL-KDD dataset, which was produced by Tavallaee *et al*. to overcome the inherent problems of the KDD ’99 data set, which are discussed in. Although, this new version of the dataset still suffers from some of the problems discussed by McHugh in and may not be a perfect representation of existing real networks. Most
current NIDS research still uses this dataset, so we believe it remains an effective benchmark to help researchers compare different methods. The NSL-KDD dataset is fundamentally the same structure as the KDD Cup ’99 dataset (i.e. it has 22 attack patterns or normal traffic, and fields for 41 features). We will be using the whole NSL-KDD dataset for our evaluations, some of the attack patterns have been high
lighted. This indicates attack patterns that contain less than 20 occurrences in the dataset. 20 is the minimum threshold required for accurate levels of training and evaluation. So, for this paper these attacks have been omitted. One of the most prominent techniques currently used within deep learning research is DBNs.One notable publication on the technique is by Alrawashdeh and Purdy, where the authors propose the use of a DBN model for NIDSs.

DISCUSSION

Our evaluations show that our proposed stacked NDAE model has produced a promising set of results.

Class KDD Cup ’99 Classification

With regards to the KDD Cup ’99 dataset evaluation, the results show that our model is able to offer an average accuracy of 97.85%. more specifically, the results show that our accuracy is better than or comparable with the work in, in 3 out of 5 classes. It is also a significant improvement on other deep learning methods such as However, it is noted that the results for “R2L” and “U2L” attack classes are anomalous. The stacked NDAE model requires greater

amounts of data to learn from. Unfortunately, due to the smaller number of training datum available, the results
achieved are less stable. Despite this, it is evident from the performance analysis that our model can offer improved precision, recall and F-score, especially for larger classes. Furthermore, our model managed to produce these comparable performance results, whilst consistently reducing the required training time by an average of 97.72%.

Class NSL-KDD Classification

With regards to the NSL-KDD dataset, we can see from the results that throughout all of the measures our model yields superior level of performance in 3 of the 5 classes. Notably, the model offered a total accuracy rate of 85.42%, which improves upon the DBN model by just under 5%. It also offered a 4.84% reduction in the false alarm rate. The results also re-emphasise the point made, that our model doesn't handle smaller classes ("R2L" and "U2R") as well. Another important factor is that the time required to train our model is drastically reduced, yielding an average time saving of 78.19% against DBN. This is of critical importance particularly for application in a NIDS.

Class NSL-KDD Classification

The results from the 13-Class classification evaluate demonstrate that our model was able to offer a 3.8% improvement on its own accuracy simply by using a more granular dataset. This supports our claim that the model is able to work more effectively with larger and complex datasets. Furthermore, the larger dataset gives a better insight into the weakness in our model. As it can be seen from the results, there is a direct correlation between the size of the training datasets for each label and the accuracy/error rates. This supports our observation that the smaller classes (in this case "back", "guess password", "tear drop" and "warez client") yield lower levels of accuracy using our model. However, it must also be noted that the larger classes
yielded consistently high rates throughout all of the performance measures.

Comparison with Related Works:

We have also compared the results from our stacked NDAE model against the results obtained from similar deep learning-based NIDSs. In, the authors claim their 5-class classification of the NSL-KDD dataset produced an f-score of 75.76%. Their recall and precision results are not listed but the bar charts show them to be around 69% and 83% respectively. Our model has produced superior results by offering f-score of
87.37%, recall of 85.42% and precision of 100.00%. Tang *et al*. claim that their Deep Neural Network (DNN) approach achieved an accuracy of 75.75% when performing a 5-class classification of the NSL-KDD dataset. This is result is lower than our achieved accuracy of 85.42%. Whilst classifying the KDD Cup '99 dataset, Kim *et al*. claim they have achieved an accuracy of accuracy of 96.93%. Also Gao *et al*. claim their deep learning DBN model achieved an accuracy of 93.49%. Both of these results
are less than the 97.85% accomplished by our model. These comparisons show that our model's results are very promising when compared to other current deep learning-based methods.

Conclusion & Future Work:

In this paper, we have discussed the problems faced by existing NIDS techniques. In response to this we have proposed our novel NDAE method for unsupervised feature
learning. We have then built upon this by proposing a novel classification model constructed from stacked NDAEs and the RF classification algorithm. We have implemented our proposed model in TensorFlow and performed extensive valuations on its capabilities. For our evaluations we have utilised the benchmark KDD Cup '99 and NSL-KDD datasets and achieved very promising results. Our results have demonstrated that our approach offers high levels of accuracy, precision and recall together with reduced training time. Most notably, we have compared our stacked NDAE model against the mainstream DBN technique. These comparisons have demonstrated that our model offers up to a 5% improvement in accuracy and train
ing time reduction of up to 98.81%. Unlike most previous work, we have evaluated the capabilities of our model based on both benchmark datasets, revealing a consistent level of classification accuracy. Although our model has achieved the above promising results, we acknowledge that it is not perfect and there is further room for improvement.In our future work, the first avenue of exploration for improvement will be to assess and extend the capability of our model to handle zero-day attacks. We will then look to expand upon our existing evaluations by utilising real world backbone network traffic to demonstrate the merits of the extended model.

Chapter 3

Existing System:

- Today network has become an essential part of public infrastructures with the inception of public and private cloud computing.
- The traditional networking approach has become too complex.
- This complexity has resulted in a barrier for creating new and innovative services within a single data center, difficulties in interconnecting data centers, interconnection within enterprises, and bigger barrier in the continued growth of the Internet in general.

Problem Statement:

- To distinguish the activities of the network traffic that the intrusion and normal is very difficult and to need much time consuming.
- An analyst must review all the data that large and wide to find the sequence of intrusion on the network connection.
- It needs a way that can detect network intrusion to reflect the current network traffics.
- Combination of IDS and firewall so-called the IPS, so that besides detecting the existence of intrusion also can execute by doing deny of intrusion as prevention.

Proposed System :

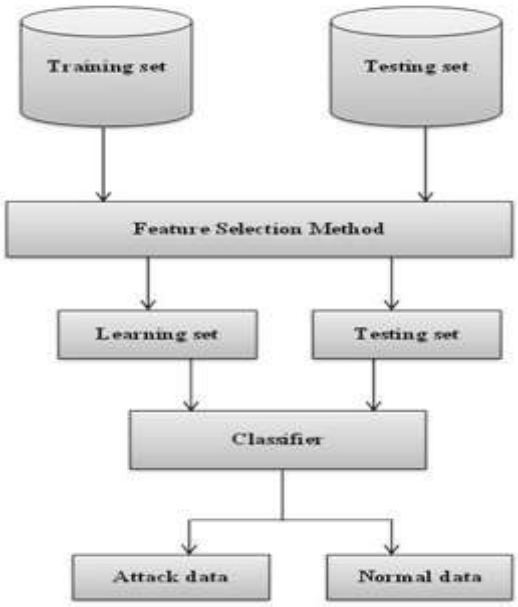
- Genetic algorithm is one of the most commonly used machine learning approach in the field of intrusion detection, which consists of its natural selection.

- Decision node represents to testing a single attribute of the given instances whereas the leaf node presents the idea about whether the output of a classifier falls in to either normal or intrusion (any of the possible attacks) category during the classification phase.
- a novel method to find intrusion characteristic for IDS using decision tree machine learning of data mining technique was proposed.Method used to generate of rules is classification by genetic algorithm of decision tree.

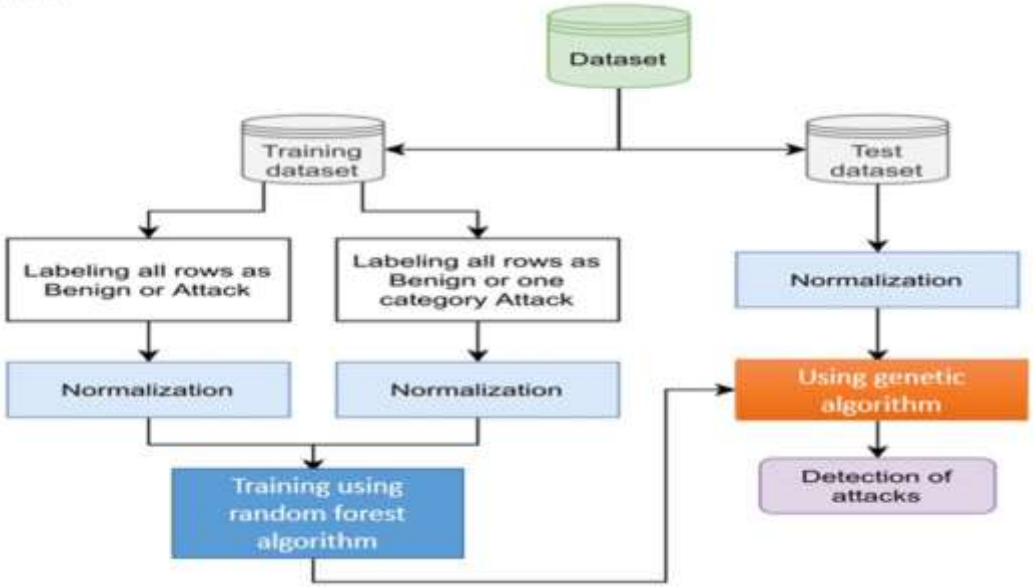
Advantages:

- Intrusion detection can be performed manually or automatically.
- IDS need to be able to deal with a large and ever-increasing flow of alerts and events.
- Using Decision trees is more essential.automatic procedures for detecting and responding to intrusion are becoming increasingly essential.

Block diagram:



Flow diagram:



Decision tree

Introduction

Till now we have learned about linear regression, logistic regression, and they were pretty hard to understand. Let's now start with Decision tree's and I assure you this is probably the easiest algorithm in Machine Learning. There's not much mathematics involved here. Since it is very easy to use and interpret it is one of the most widely used and practical methods used in Machine Learning.

What is a Decision Tree?

It is a tool that has applications spanning several different areas. Decision trees can be used for
classification as well as regression problems. The name itself suggests that it uses a flowchart like a tree structure to show the predictions that result from a series of feature-based splits. It starts with a root
node and ends with a decision made by leave. Before learning more about decision trees let's get familiar with some of the terminologies.

Root Nodes – It is the node present at the beginning of a decision tree from this node the population starts dividing according to various features.

Decision Nodes – the nodes we get after splitting the root nodes are called Decision Node

Leaf Nodes – the nodes where further splitting is not possible are called leaf nodes or terminal nodes

Sub-tree – just like a small portion of a graph is called sub-graph similarly a sub-section of this decisiontree is called sub-tree.

Pruning – is nothing but cutting down some nodes to stop overfitting.

Example of a decision tree.

Let's understand decision trees with the help of an example.
Decision trees are upside down which means the root is at the top and then this root is split into various several nodes. Decision trees are nothing but a bunch of if-else statements in layman terms. It checks if the condition is true and if it is then it goes to the next node attached to that decision.
Did you notice anything in the above flowchart? We see that if the *weather is cloudy* then we must go to play. Why didn't it split more? Why did it stop there?

To answer this question, we need to know about few more concepts like entropy, information gain, and Gini index. But in simple terms, I can say here that the output for the training dataset is always yes for cloudy weather, since there is no disorderliness here we don't need to split the node further.

The goal of machine learning is to decrease uncertainty or disorders from the dataset and for this, we use decision trees.

Now you must be thinking how do I know what should be the root node? what should be the decision node? when should I stop splitting? To decide this, there is a metric called "Entropy" which is the amount of uncertainty in the dataset.

Entropy:

Entropy is nothing but the uncertainty in our dataset or measure of disorder. Let me try to explain this with the help of an example.

Suppose you have a group of friends who decides which movie they can watch together on Sunday. There are 2 choices for movies, one is ***"Lucy"*** and the second is ***"Titanic"*** and now everyone has to tell their choice. After everyone gives their answer we see that

"Lucy" gets 4 votes and *"Titanic" gets 5 votes*. Which movie do we watch now? Isn't it hard to choose 1 movie now because the votes for both the movies are somewhat equal.

This is exactly what we call disorderness, there is an equal number of votes for both the movies, and we can't really decide which movie we should watch. It would have been much easier if the votes for "Lucy" were 8 and for "Titanic" it was 2. Here we could easily say that the majority of votes are for "Lucy" hence everyone will be watching this movie.

In a decision tree, the output is mostly "yes" or "no" The formula for Entropy is shown below:

How do Decision Trees use Entropy?

Now we know what entropy is and what is its formula, Next, we need to know that how exactly does it work in this algorithm.

Entropy basically measures the impurity of a node. Impurity is the degree of randomness; it tells how random our data is. A **pure sub-split** means that either you should be getting "yes", or you should be getting "no".

Suppose *feature 1* had 8 yes and 4 no, after the split *feature 2 get 5 yes and 2 no* whereas f*eature 3 gets 3 yes and 2 no.*

We see here the split is not pure, why? Because we can still see some negative classes in both the feature. In order to make a decision tree, we need to calculate the impurity of each split, and when the purity is 100% we make it as a leaf node. To check the impurity of feature 2 and feature 3 we will take the help for Entropy

We can clearly see from the tree itself that feature 2 has low entropy or more purity than feature 3 since feature 2 has more "yes" and it is easy to make a decision here.

Always remember that the higher the Entropy, the lower will be the purity and the higher will be the impurity.

As mentioned earlier the goal of machine learning is to decrease the uncertainty or impurity in the dataset, here by using the entropy we are getting the impurity of a feature or a particular node, we don't know if the parent entropy or the entropy of a particular node has decreased or not.

For this, we bring a new metric called "Information gain" which tells us how much the parent entropy has decreased after splitting it with some feature.

Information Gain:

Information gain measures the reduction of uncertainty given some feature and it is also a deciding factor for which attribute should be selected as a decision node or root node. It is just entropy of the full dataset – entropy of the dataset given some feature.
Let's see how our decision tree will be made using these 2 features. We'll use information gain to decide which feature should be the root node and which feature should be placed after the split.

When to stop splitting?

You must be asking this question to yourself that when do we stop growing our tree? Usually, real-world datasets have a large number of features, which will result in a large number of splits, which in turn gives a huge tree. Such trees take time to build and can lead to overfitting. That means the tree will give very good accuracy on the training dataset but will give bad accuracy in test data.

There are many ways to tackle this problem through hyperparameter tuning. We can set the maximum depth of our decision tree using the max_depth parameter. The more the value of max_depth, the more complex your tree will be. The training error will off-course decrease if we increase the max_depth value but when our test data comes into the picture, we will get a very bad accuracy. Hence you need a value thatwill not overfit as well as underfit our data and for this, you can use GridSearchCV.

Another way is to set the minimum number of samples for each spilt. It is denoted by min_samples_split. Here we specify the minimum number of samples required to do a spilt. For example, we can use a minimum of 10 samples to reach a decision. That means if a node has less than 10 samples then using this parameter, we can stop the further splitting of this node and make it a leaf node. There are more hyperparameters such as :
min_samples_leaf – represents the minimum number of samples required to be in the leaf node. The more you increase the number, the more is the possibility of overfitting.

max_features – it helps us decide what number of features to consider when

looking for the best split.To read more about these hyperparameters **Pruning**

It is another method that can help us avoid overfitting. It helps in improving the performance of the tree by cutting the nodes or sub-nodes which are not significant. There are mainly 2 ways for pruning:

(i) **Pre-pruning** – we can stop growing the tree earlier, which means we can prune/remove/cut a node if ithas low importance **while growing** the tree.
(ii) **Post-pruning** – once our **tree is built to its depth**, we can start pruning the nodes based on their significance.

Endnotes

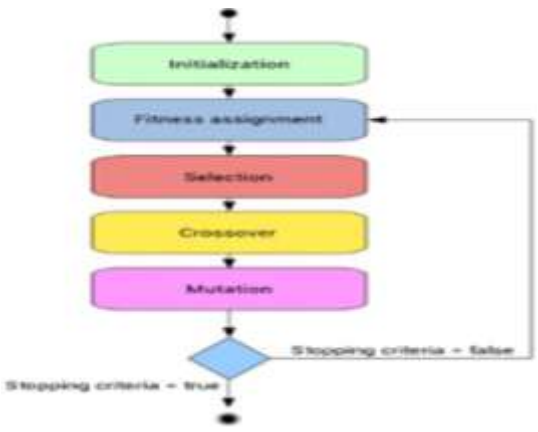
To summarize, in this article we learned about decision trees. On what basis the tree splits the nodes and how to can stop overfitting. why linear regression doesn't work in the case of classification problems. In the next article, I will explain Random forests, which is again a new technique to avoid overfitting.

Genetic Algorithm

Let's get back to the example we discussed above and summarize what we did.

1. Firstly, we defined our initial population as our countrymen.
2. We defined a function to classify whether is a person is good or bad.
3. Then we selected good people for mating to produce their off-springs.
4. And finally, these off-springs replace the bad people from the population and this process repeats.

This is how genetic algorithm actually works, which basically tries to mimic the human evolution to some extent. So to formalize a definition of a genetic algorithm, we can say that it is an optimization technique, which tries to find out such values of input so that we get the best output values or results. The working of a genetic algorithm is also derived from biology, which is as shown in the image below.



Steps Involved in Genetic Algorithm:

- Initialisation
- Fitness Function
- Selection
- Crossover
- Mutation

Application of Genetic Algorithm:
Feature Selection

Every time you participate in a data science competition, how do you select features that are important in prediction of the target variable? You always look at the feature importance of some model, and then manually decide the threshold, and select the features which have importance above that threshold.

Is there any better way to deal with this kind of situations? Actually one of the most advanced algorithms for feature selection is genetic algorithm.

The method here is completely same as the one we did with the knapsack problem.

We will again start with the population of chromosome, where each chromosome will be binary string. 1 will denote "inclusion" of feature in model and 0 will denote "exclusion" of feature in the model.

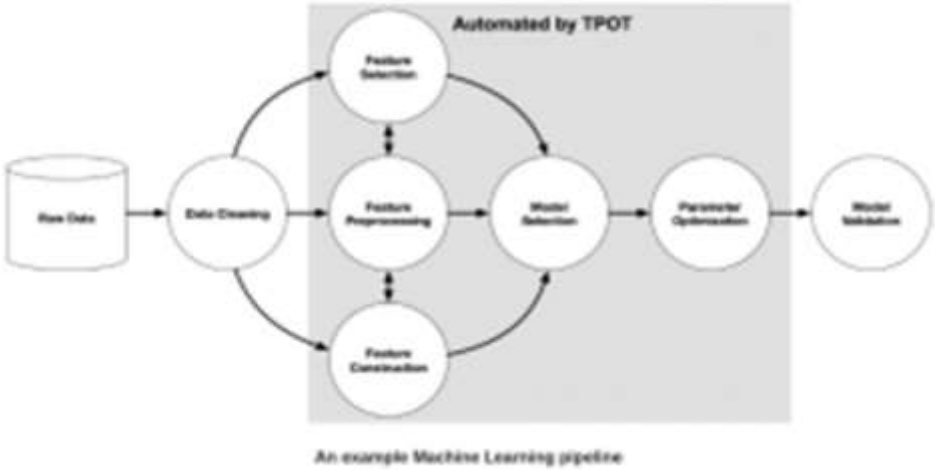
And another difference would be that the fitness function would be changed. The fitness function here will be our accuracy metric of the competition. The more accurate our set of chromosome in predicting value, the more fit it will be.

I suppose, you would now be thinking is there any use of such tough tasks. I will not answer this question now, rather let us look at the implementation of it using TPOT library and then you decide this.

Implementation using TPOT library

First, let's take a quick view on the TPOT (Tree-based Pipeline Optimisation Technique) which is build upon scikit-learn library.

A basic pipeline structure is shown in the image below.



An example Machine Learning pipeline

So the highlighted grey section in the image above is automated using TPOT. This automation is achieved using genetic algorithm.So, without going deep into this, let's directly try to implement it.For using TPOT library, you first have to install some existing python libraries on which TPOT is build. So let us quickly install them.

Applications in Real World:

- Engineering Design
- Robotics.

End Notes

I hope that now you have gain enough understanding about what genetic algorithm is and also how to implement it using TPOT library. But this knowledge is not enough, if you don't apply it somewhere. So try to implement it whether in any real world application or in a data science competition.

Hardware requirements:

- System: Pentium i3 Processor.
- Hard Disk: 500 GB.
- Monitor : 15'' LED
- Input Devices : Keyboard, Mouse
- Ram : 2 GB

Software requirements:

- Operating System: Windows 10.
- Coding Language : Python

Chapter 4

Machine learning

What are the 7 steps of machine learning?

7 Steps of Machine Learning

- Step 1: Gathering Data. ...
- Step 2: Preparing that Data. ...
- Step 3: Choosing a Model. ...
- Step 4: Training. ...
- Step 5: Evaluation. ...
- Step 6: Hyper parameter Tuning. ...
- Step 7: Prediction.

Introduction:
In this blog, we will discuss the workflow of a Machine learning project this includes all the steps required to build the proper machine learning project from scratch.
We will also go over data pre-processing, data cleaning, feature exploration and feature engineering and show the impact that it has on Machine Learning Model Performance. We will also cover a couple of the pre-modelling steps that can help to improve the model performance.
Python Libraries that would be need to achieve the task:
1. Numpy
2. Pandas
3. Sci-kit Learn
4. Matplotlib

Understanding the machine learning workflow
We can define the machine learning workflow in 3 stages.
Gathering data
Data pre-processing
Researching the model that will be best for the type of data
Training and testing the model
Evaluation
Okay but first let's start from the basics

What is the machine learning Model?
The machine learning model is nothing but a piece of code; an engineer or data scientist makes it smart through training with data. So, if you give garbage to the model, you will get garbage in return, i.e. the trained model will provide false or wrong prediction

1. Gathering Data

The process of gathering data depends on the type of project we desire to make, if we want to make an ML project that uses real-time data, then we can build an IoT system that using different sensors data. The data set can be collected from various sources such as a file, database, sensor and many other such sources but the collected data cannot be used directly for performing the analysis process as there might be a lot of missing data, extremely large values, unorganized text data or noisy data. Therefore, to solve this problem Data Preparation is done.

We can also use some free data sets which are present on the internet. Kaggle and UCI Machine learning Repository are the repositories that are used the most for making Machine learning models. Kaggle is one of the most visited websites that is used for practicing machine learning algorithms, they also host competitions in which people can participate and get to test their knowledge of machine learning.

2. Data pre-processing

Data pre-processing is one of the most important steps in machine learning. It is the most important step that helps in building machine learning models more accurately. In machine learning, there is an 80/20 rule. Every data scientist should spend 80% time for data per-processing and 20% time to actually perform the analysis.

What is data pre-processing?

Data pre-processing is a process of cleaning the raw data i.e. the data is collected in the real world and is converted to a clean data set. In other words, whenever the data is gathered from different sources it is collected in a raw format and this data isn't feasible for the analysis. Therefore, certain steps are executed to convert the data into a small clean data set, this part of the process is called as data pre-processing.

Why do we need it?

As we know that data pre-processing is a process of cleaning the raw data into clean data, so that can be used to train the model. So, we definitely need data pre-processing to achieve good results from the applied model in machine learning and deep learning projects.Most of the real-world data is messy, some of these types of data are:

1. **Missing data:** Missing data can be found when it is not continuously created or due to technical issues in the application (IOT system).
2. **Noisy data:** This type of data is also called outliners, this can occur due to human errors (human manually gathering the data) or some technical problem of the device at the time of collection of data.
3. **Inconsistent data:** This type of data might be collected due to human errors (mistakes with the name or values) or duplication of data.

Three Types of Data

1. Numeric e.g. income, age
2. Categorical e.g. gender, nationality
3. Ordinal e.g. low/medium/high

How can data pre-processing be performed?

These are some of the basic pre — processing techniques that can be used to convert raw data.

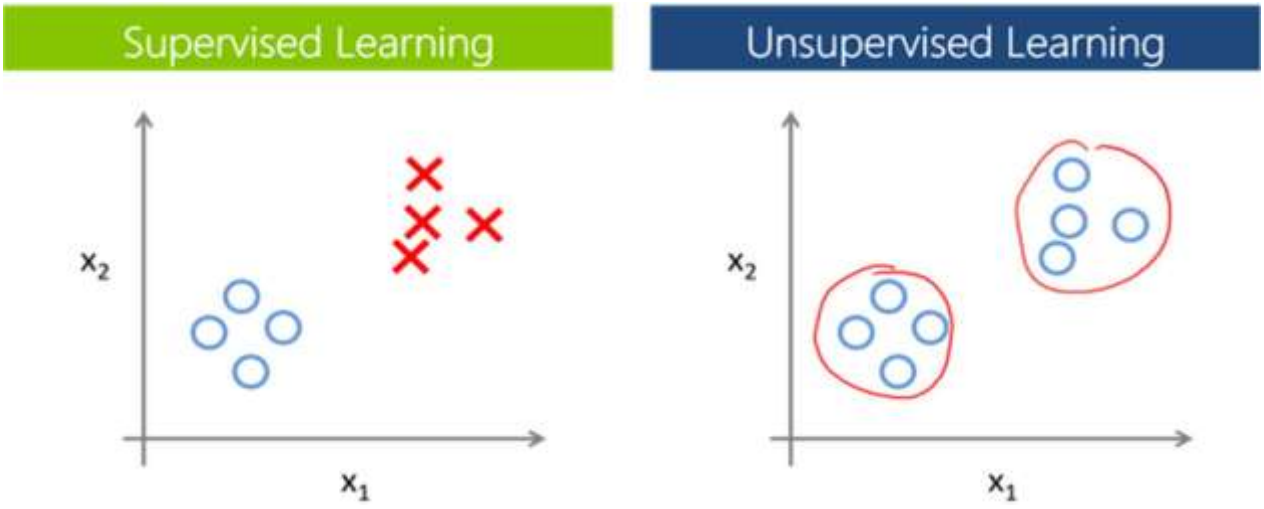
1. **Conversion of data:** As we know that Machine Learning models can only handle numeric features, hence categorical and ordinal data must be somehow converted into numeric features.

2. **Ignoring the missing values:** Whenever we encounter missing data in the data set then we can remove the row or column of data depending on our need. This method is known to be efficient but it shouldn't be performed if there are a lot of missing values in the dataset.
3. **Filling the missing values:** Whenever we encounter missing data in the data set then we can fill the missing data manually, most commonly the mean, median or highest frequency value is used.
4. **Machine learning:** If we have some missing data then we can predict what data shall be present at the empty position by using the existing data.
5. **Outliers detection:** There are some error data that might be present in our data set that deviates drastically from other observations in a data set. [Example: human weight = 800 Kg; due to mistyping of extra 0]

Researching the model that will be best for the type of data

Our main goal is to train the best performing model possible, using the pre-processed data.

MACHINE LEARNING CLASSIFICATION



Supervised Learning:

In Supervised learning, an AI system is presented with data which is labelled, which means that each data tagged with the correct label.The supervised learning is categorized into 2 other categories which are "**Classification**" and "**Regression**".

Classification:

Classification problem is when the target variable is categorical (i.e. the output could be classified into classes — it belongs to either Class A or B or something else).

A classification problem is when the output variable is a category, such as "red" or "blue" , "disease" or "no disease" or "spam" or "not spam".

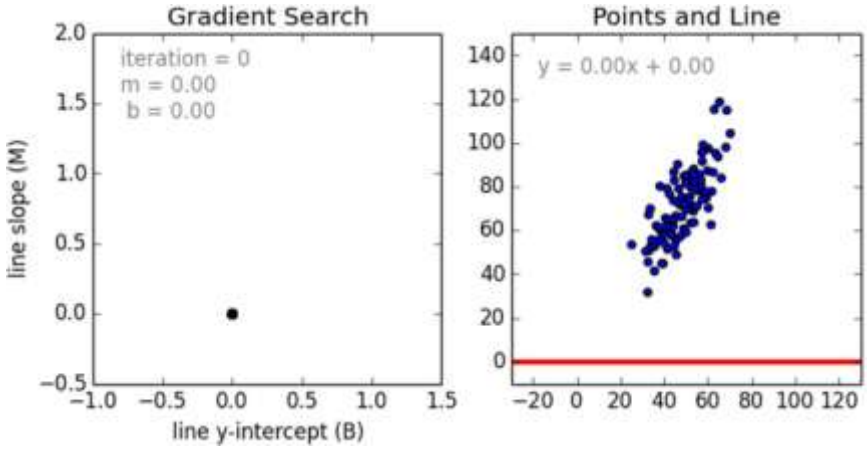
As shown in the above representation, we have 2 classes which are plotted on the graph i.e. red and blue which can be represented as 'setosa flower' and 'versicolor flower', we can image the X-axis as ther 'Sepal Width' and the Y-axis as the 'Sepal Length', so we try to create the best fit line that separates both classes of flowers.

These some most used classification algorithms.

- K-Nearest Neighbor
- Naive Bayes
- Decision Trees/Random Forest
- Support Vector Machine
- Logistic Regression

Regression:

While a Regression problem is when the target variable is continuous (i.e. the output is numeric).

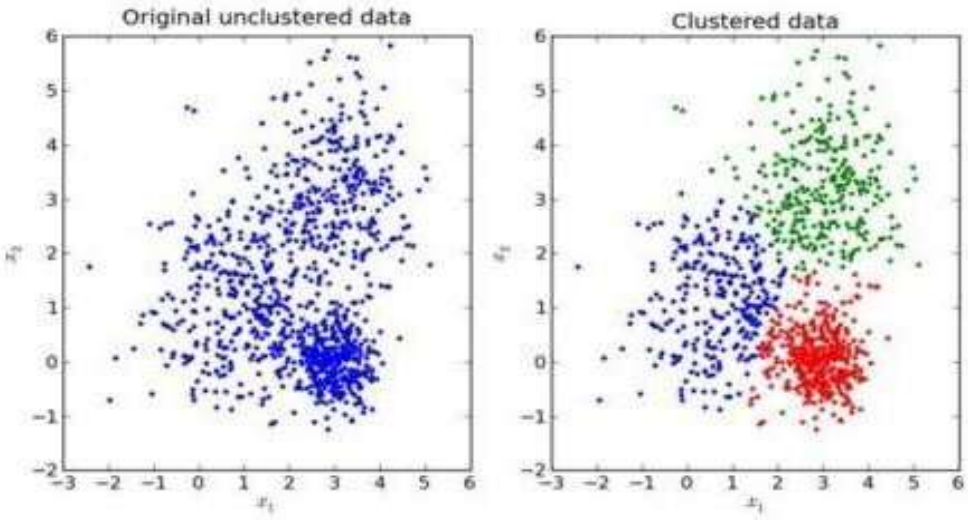


As shown in the above representation, we can imagine that the graph's X-axis is the 'Test scores' and the Y-axis represents 'IQ'. So we try to create the best fit line in the given graph so that we can use that line to predict any approximate IQ that isn't present in the given data.These some most used regression algorithms.

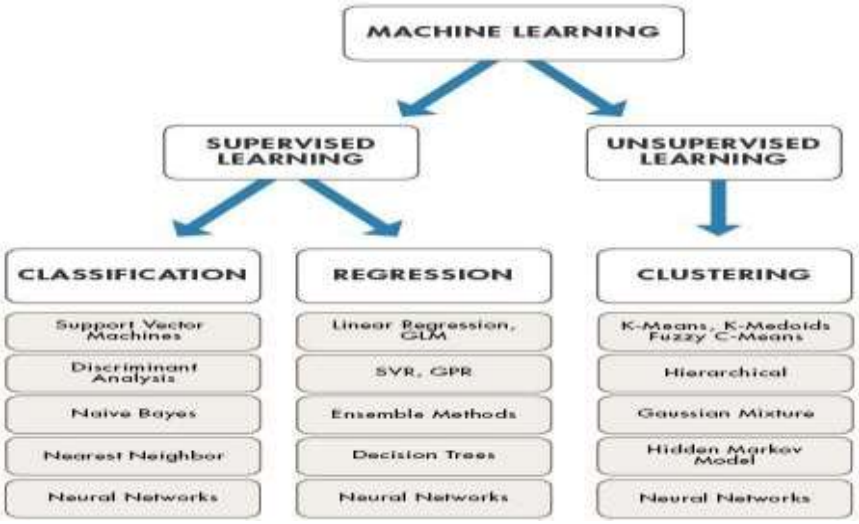
Linear Regression

Support Vector Regression

Decision Tress/Random Forest
Gaussian Progresses Regression
Ensemble Methods
Unsupervised Learning:
The unsupervised learning is categorized into 2 other categories which are "**Clustering**" and "**Association**".
Clustering:
A set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.

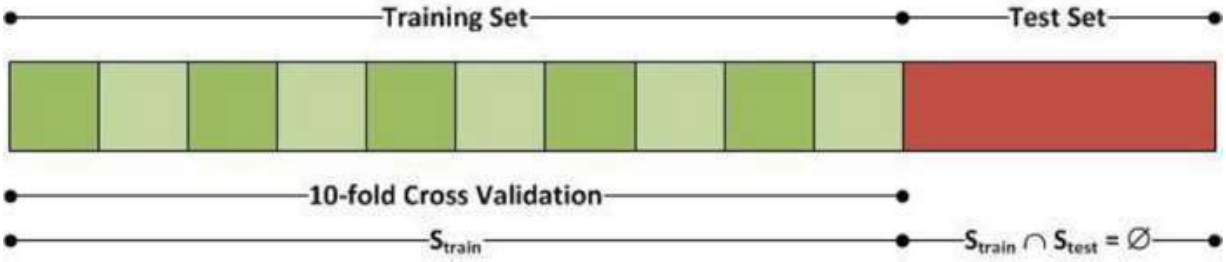


Methods used for clustering are:
Gaussian mixtures
K-Means Clustering
Boosting
Hierarchical Clustering
K-Means Clustering
Spectral Clustering
Overview of models under categories:



4. Training and testing the model on data

For training a model we initially split the model into 3 three sections which are '**Training data**' ,'**Validation data**' and '**Testing data**'.You train the classifier using '**training data set**', tune the parameters using '**validation set**' and then test the performance of your classifier on unseen '**test data set**'. An important point to note is that during training the classifier only the training and/or validation set is available. The test data set must not be used during training the classifier. The test set will only be available during testing the classifier.

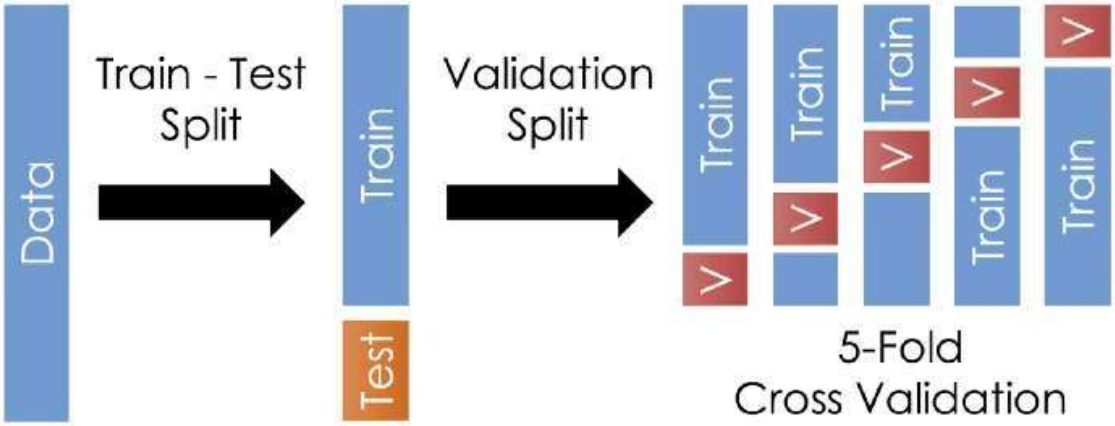


Training set:

The training set is the material through which the computer learns how to process information. Machine learning uses algorithms to perform the training part. A set of data used for learning, that is to fit the parameters of the classifier.

Validation set:

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. A set of unseen data is used from the training data to tune the parameters of a classifier.



Once the data is divided into the 3 given segments we can start the training process.

In a data set, a training set is implemented to build up a model, while a test (or validation) set is to validate the model built. Data points in the training set are excluded from the test (validation) set. Usually, a data set is divided into a training set, a validation set (some people use 'test set' instead) in each iteration, or divided into a training set, a validation set and a test set in each iteration.The model uses any one of the models that we had chosen in step 3/ point 3. Once the model is trained we can use the same trained model to predict using the testing data i.e. the unseen data. Once this is done we can develop a confusion matrix, this tells us how well our model is trained. A confusion matrix has 4 parameters, which are '**True positives'**, '**True Negatives'**, '**False Positives'** and '**False Negative'**. We prefer that we get more values in the True negatives and true positives to get a more accurate model. The size of the Confusion matrix completely depends upon the number of classes.

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

True positives : These are cases in which we predicted TRUE and our predicted output is correct.

True negatives : We predicted FALSE and our predicted output is correct.

False positives : We predicted TRUE, but the actual predicted output is FALSE.

False negatives : We predicted FALSE, but the actual predicted output is TRUE.

We can also find out the accuracy of the model using the confusion matrix.

Accuracy = (True Positives +True Negatives) / (Total number of classes)

i.e. for the above example:

Accuracy = (100 + 50) / 165 = 0.9090 (90.9% accuracy)

5. Evaluation

Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future.To

improve the model we might tune the hyper-parameters of the model and try to improve the accuracy and also looking at the confusion matrix to try to increase the number of true positives and true negatives.
Conclusion
In this blog, we have discussed the workflow a Machine learning project and gives us a basic idea of how a should the problem be tackled.

Machine Learning Work Flow

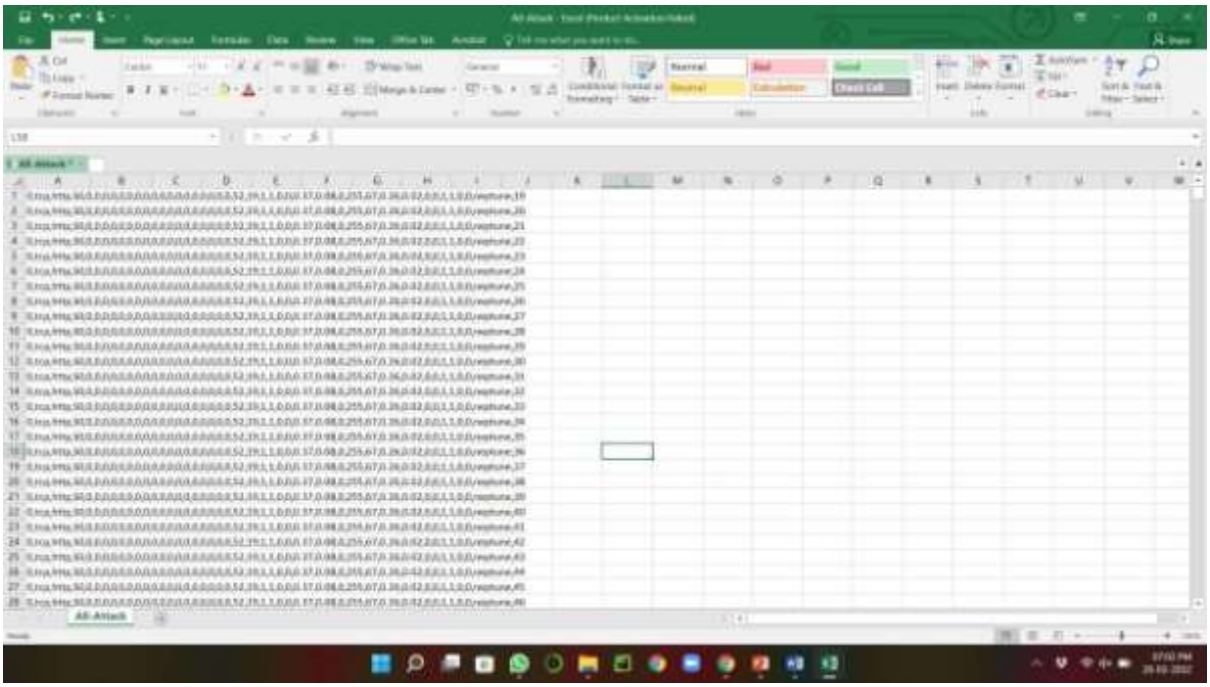
The machine learning model is nothing but a piece of code; an engineer or data scientist makes it smart through training with data. So, if you give garbage to the model, you will get garbage in return, i.e. the trained model will provide false or wrong prediction.
We can define the machine learning workflow in below.

1. Data Collection
2. Data pre-processing
3. Future Extraction
4. Model Training
5. Testing Model
6. Evaluation
7. Prediction

Data Collection

Collecting data allows you to capture a record of past events so that we can use data analysis to find recurring patterns.
KDD datasets:

The KDD data set is a well-known benchmark in the research of Intrusion Detection techniques. A lot of work is going on for the improvement of intrusion detection strategies while the research on the data used for training and testing the detection model is equally of prime concern because better data quality can improve offline intrusion detection. This paper presents the analysis of KDD data set with respect to four classes which are Basic, Content, Traffic and Host in which all data attributes can be categorized.



Data Pre-Processing

Data pre-processing is a process of cleaning the raw data i.e. the data is collected in the real world and is converted to a clean data set. In other words, whenever the data is gathered from different sources it is collected in a raw format and this data isn't feasible for the analysis. Therefore, certain steps are executed to convert the data into a small clean data set, this part of the process is called as data pre-processing.

Feature Extraction:
This is done to reduce the number of attributes in the dataset hence providing advantages like speeding up the training and accuracy improvements.

Model training:
A training model is a dataset that is used to train an ML algorithm. It consists of the sample output data and the corresponding sets of input data that have an influence on the output.

Testing model:
In this module we test the trained machine learning model using the test dataset. Quality assurance is required to make sure that the software system works according to the requirements. Were all the features implemented as agreed? Does the program behave as expected? All the parameters that you test the program against should be stated in the technical specification document.

Performance Evaluation:
In this module, we evaluate the performance of trained machine learning model using performance evaluation criteria such as F1 score, accuracy and classification error. Performance Evaluation is defined as a formal and productive procedure to measure an employee's work and results based on their job responsibilities. It is used to gauge the amount of value added by an employee in terms of increased business revenue, in comparison to industry standards and overall employee return on investment (ROI).

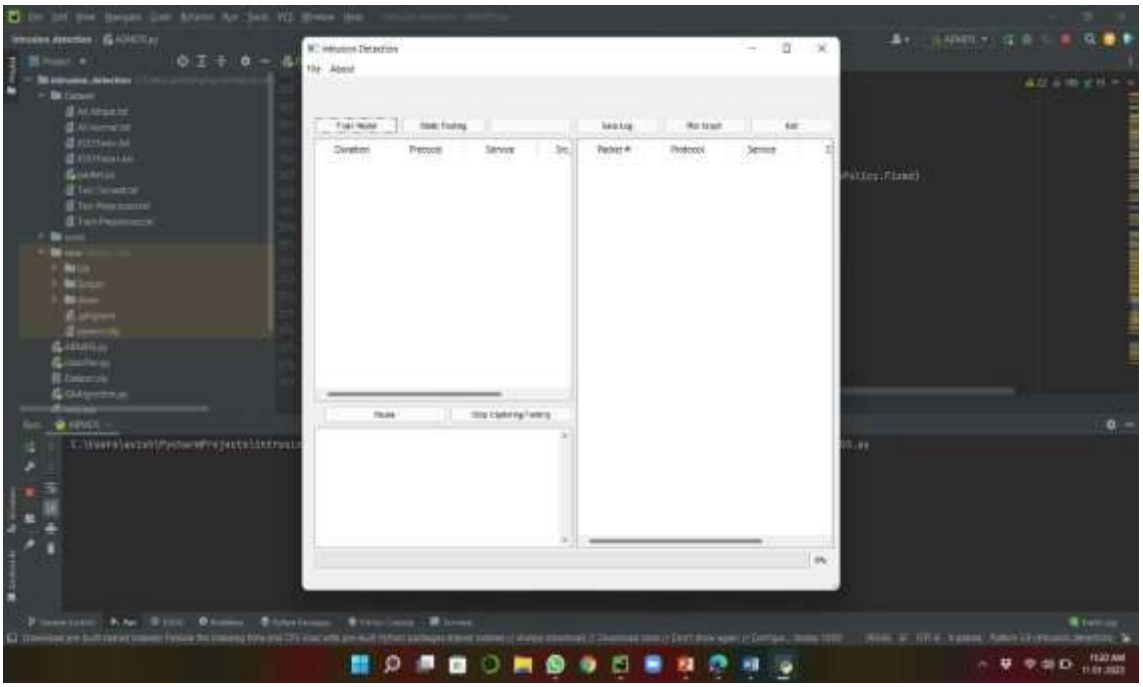
Prediction:
The algorithm will generate probable values for an unknown variable for each record in the new data, allowing the model builder to identify what that value will most likely be. The word "prediction" can be misleading. In some cases, it really does mean that you are predicting a future outcome, such as when you're using machine learning to determine the next best action in a marketing campaign.

Chapter 5

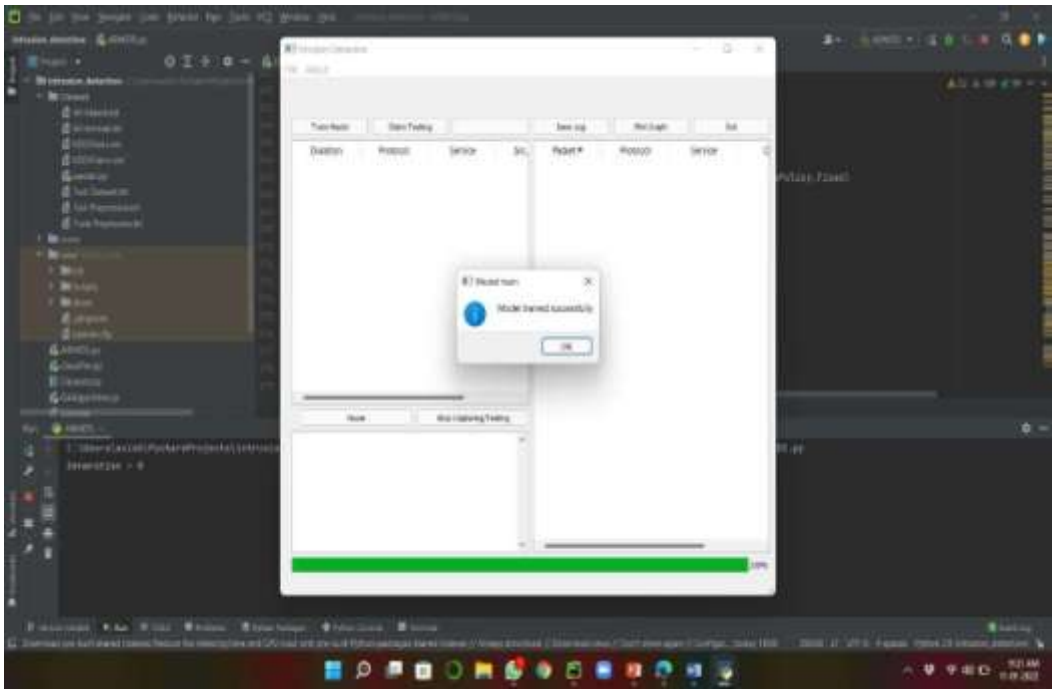
5.1 RESULT

S. No	Types of Attacks	Count
1	All Attacks	4019
2	Normal Attacks	1589

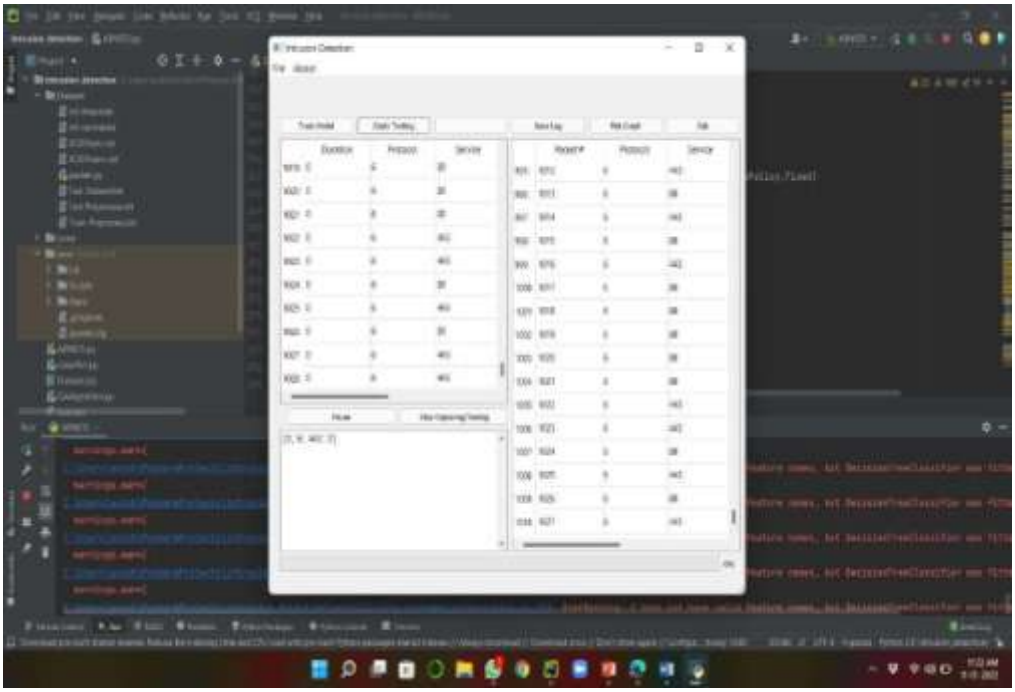
4.2 Attacks



4.2.1 Training dataset

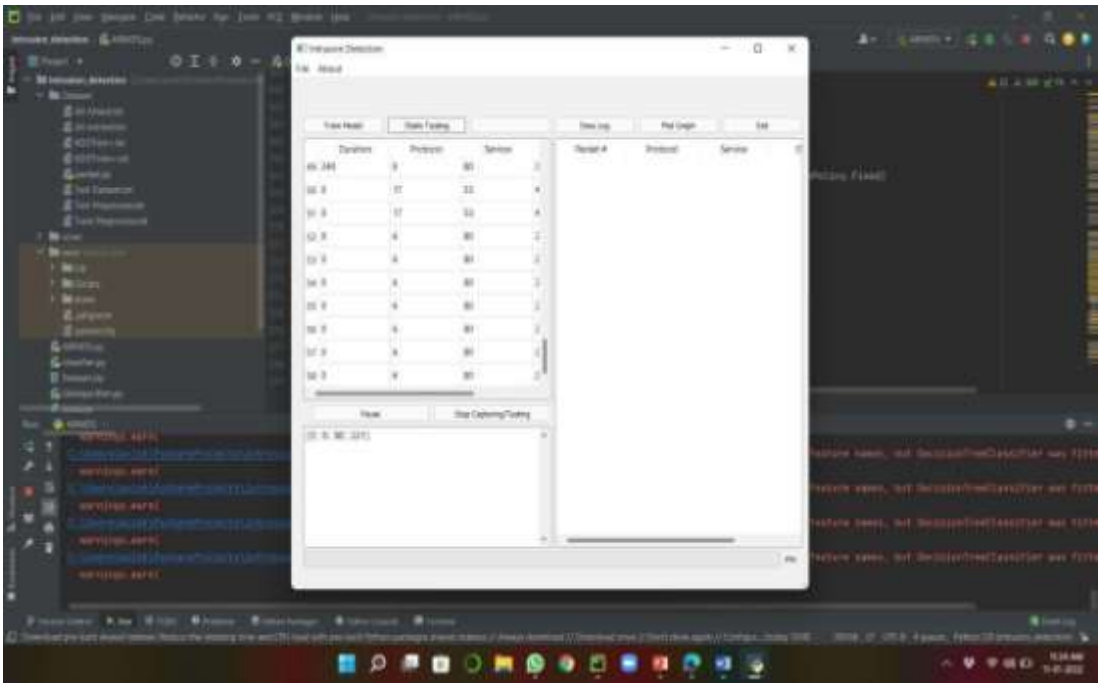


4.2.2



Testing All Attacks

All Attackks Plot graph



Normal Attacks Testing

Normal Attacks Plot Graph

5.2 Code

Dataset.py

```
import pyshark
import time
import random
class Packet:
    packet_list = list()
    def initiating_packets(self):
        self.packet_list.clear()
        capture = pyshark.LiveCapture(interface="Wi-Fi")
        for packet in capture.sniff_continuously(packet_count=25):
```

```
            try:
                if "<UDP Layer>" in str(packet.layers) and "<IP Layer>" in str(packet.layers):
                    self.packet_list.append(packet)
                elif "<TCP Layer>" in str(packet.layers) and "<IP Layer>" in str(packet.layers):
                    self.packet_list.append(packet)
            except:
                print(f"No Attribute name 'ip' {packet.layers}")
    def udp_packet_attributes(self,packet):
        attr_list = list()
        a1 = packet.ip.ttl
        a2 = packet.ip.proto
        a3 = self.__get_service(packet.udp.port, packet.udp.dstport)
        a4 = packet.ip.len
        a5 = random.randrange(0,1000)
        a6  =  self._get_land(packet,a2)
        a7 = 0
        a8, a10, a11 = self._get_count_with_same_and_diff_service_rate(packet.udp.dstport, a3)
#23, 29, 30
        a9, a12 = self.__get_srv_count_and_srv_diff_host_rate(packet.ip.dst, a3) #24, 31
        a13, a15, a16 = self._get_dst_host_count(packet.ip.dst, a3) # 32,34,35
        a14, a17, a18 = self._get_dst_host_srv_count(packet.udp.port, packet.udp.dstport,
packet.ip.dst) #33, 36, 37
        attr_list.extend((a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18))
        return self.get_all_float(attr_list)

    def tcp_packet_attributes(self,packet):
        attr_list = list()
        a1 = packet.ip.ttl #duration
        a2 = packet.ip.proto    #protocol
        a3 = self.__get_service(packet.tcp.port, packet.tcp.dstport) # service
        a4 = packet.ip.len
        a5 = random.randrange(0,1000)
        a6  =  self._get_land(packet,a2)
        a7 = packet.tcp.urgent_pointer
        a8, a10, a11 = self._get_count_with_same_and_diff_service_rate(packet.tcp.dstport, a3)
#23, 29, 30
        a9, a12 = self.__get_srv_count_and_srv_diff_host_rate(packet.ip.dst, a3) #24, 31
        a13, a15, a16 = self._get_dst_host_count(packet.ip.dst, a3) # 32,34,35
        a14, a17, a18 = self._get_dst_host_srv_count(packet.tcp.port, packet.tcp.dstport,
packet.ip.dst) #33, 36, 37
        attr_list.extend((a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18))
        return self.get_all_float(attr_list)
```

```
def __get_service(self,src_port,dst_port):
    services = [80,443,53]
    if int(src_port) in services:
        return int(src_port)
    elif int(dst_port) in services:
        return int(dst_port)
    else:
        return 53

def __get_land(self,packet, protocol):
    if int(protocol) == 6:
        if(packet.ip.dst == packet.ip.src and packet.tcp.port == packet.tcp.dstport):
            return 1
        else:
            return 0
    elif int(protocol) == 17:
        if(packet.ip.dst == packet.ip.src and packet.udp.port == packet.udp.dstport):
            return 1
        else:
            return 0


def __get_count_with_same_and_diff_service_rate(self,dst_port, service): #23, 29, 30
    count = 0
    packet_with_same_service = 0
    for p in self.packet_list:
            if "<UDP Layer>" in str(p.layers):
                if (p.udp.dstport == dst_port):
                    count+=1
                    if (self.__get_service(p.udp.port, p.udp.dstport) == service):
                        packet_with_same_service+=1
            elif "<TCP Layer>" in str(p.layers):
                if (p.tcp.dstport == dst_port):
                    count+=1
                    if (self.__get_service(p.tcp.port, p.tcp.dstport) == service):
                        packet_with_same_service+=1
    same_service_rate=0.0
    diff_service_rate = 1.0
    if not count==0:          # To avoid zero divison error
        same_service_rate = ((packet_with_same_service*100)/count)/100
        diff_service_rate = diff_service_rate-same_service_rate
    return (count, same_service_rate, diff_service_rate)
```

```
def __get_srv_count_and_srv_diff_host_rate(self,dst_ip, service): #24, 31
    diff_dst_ip = 0
    service_count = 0
    for p in self.packet_list:
            if "<UDP Layer>" in str(p.layers):
                    if (self.__get_service(p.udp.port, p.udp.dstport) == service):
                        service_count+=1
                        if not (p.ip.dst == dst_ip):          # not added
                            diff_dst_ip+=1
            elif "<TCP Layer>" in str(p.layers):
                if (self.__get_service(p.tcp.port, p.tcp.dstport) == service):
                        service_count+=1
                        if not (p.ip.dst == dst_ip):           # not added
                            diff_dst_ip+=1
    srv_diff_host_rate = 0.0
    if not(service_count == 0):
        srv_diff_host_rate = ((diff_dst_ip*100)/service_count)/100
    return (service_count, srv_diff_host_rate)

def __get_dst_host_count(self,dst_ip, service): #32, 34, 35
    same_dst_ip = 0
    same_service=0
    for p in self.packet_list:
        if(p.ip.dst == dst_ip):
            same_dst_ip+=1
            if "<UDP Layer>" in str(p.layers):
                if (self.__get_service(p.udp.port, p.udp.dstport) == service):
                        same_service+=1
            elif "<TCP Layer>" in str(p.layers):
                if (self.__get_service(p.tcp.port, p.tcp.dstport) == service):
                        same_service+=1
    dst_host_same_srv_rate = 0.0
    dst_host_diff_srv_rate = 1.0
    if not same_dst_ip==0:
        dst_host_same_srv_rate = ((same_service*100)/same_dst_ip)/100
        dst_host_diff_srv_rate = dst_host_diff_srv_rate-dst_host_same_srv_rate
    return (same_dst_ip, dst_host_same_srv_rate, dst_host_diff_srv_rate)

def __get_dst_host_srv_count(self,src_port, dst_port, dst_ip): #33, 36, 37
    dst_host_srv_count = 0
    same_src_port = 0
    diff_dst_ip = 0
    for p in self.packet_list:
```

```
            if "<UDP Layer>" in str(p.layers):
                if (p.udp.dstport == dst_port):      # same destination port
                    dst_host_srv_count+=1
                    if (p.udp.port == src_port):    # same src port
                        same_src_port+=1
                    if not (p.ip.dst == dst_ip):        # different destination Ip
                        diff_dst_ip+=1

            elif "<TCP Layer>" in str(p.layers):
                if (p.tcp.dstport == dst_port):     # same destination port
                    dst_host_srv_count+=1
                    if (p.tcp.port == src_port):     # same src port
                        same_src_port+=1
                    if not (p.ip.dst == dst_ip):        #different destination ip
                        diff_dst_ip+=1
        dst_host_same_src_port_rate = 0.0
        dst_host_srv_diff_host_rate = 0.0
        if not dst_host_srv_count==0:
            dst_host_same_src_port_rate = ((same_src_port*100)/dst_host_srv_count)/100
            dst_host_srv_diff_host_rate = ((diff_dst_ip*100)/dst_host_srv_count)/100
        return (dst_host_srv_count, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate)


    def get_all_float(self,l):

        all_float = list()
        for x in l:
            all_float.append(round(float(x),1))
        return all_float
```

GAAlogrithm.py

```
import Population
import random


class GAAlgorithm():

    def __init__(self,train_dataset, test_dataset, population_size, mutation_rate,gene_length=18):
        self.train_dataset = train_dataset
        self.test_dataset = test_dataset
        self.population_size = population_size
        self.mutation_rate = mutation_rate
```

```
        self.gene_length = int(gene_length)
        self.population = Population.Population(self.train_dataset, self.test_dataset,
self.population_size, self.gene_length)

    def initialization(self):
        self.population.initialize_population()

    def calculate_fitness(self):
        self.population.calculate_fitness()

    def selection(self):
        parents = list()
        end = int(self.population_size/2)
        no_of_parents = int(self.population_size/2)
        for x in range(no_of_parents):
            p1 = random.randint(0,end-1)
            p2 = random.randint(end,self.population_size-1)
            parents.append([p1,p2])
        return parents
    def cross_over(self,parents):
        self.population.cross_over(parents)

    def mutation(self):
        self.population.mutation(self.mutation_rate)

    def clear_population(self):
        self.population.clear_population()
```

Individual.py

```
import random
import string
import pandas
from classifier import DecisionTree

class Individual:

    chromosome = list()
    fitness = 0
    def __init__(self, train_dataset, test_dataset, gene_length=18):
        self.gene_length=int(gene_length)
```

```
        self.chromosome = [random.randint(0,1) for x in range(self.gene_length)]
        self.train_dataset = train_dataset
        self.test_dataset = test_dataset
        self.gene_length = gene_length


    def calculate_fitness(self):
        header = list(string.ascii_lowercase[0:(self.gene_length+1)])
        kdd_train = pandas.read_csv(self.train_dataset, names=header)
        kdd_test = pandas.read_csv(self.test_dataset, names=header)
        selected_index= [header[x] for x, y in enumerate(self.chromosome) if y==1]
        var_train, res_train = kdd_train[selected_index], kdd_train[header[18]]
        var_test, res_test = kdd_test[selected_index], kdd_test[header[18]]
        self.fitness = self.__get_fitness(var_train, res_train, var_test, res_test)*100

    def__get_fitness(self,var_train, res_train, var_test, res_test):
        return DecisionTree.get_fitness(var_train, res_train, var_test, res_test)
```

Packet.py

```
import pyshark
import  random
class Packet:
    packet_list = list()            #list is declare
    def initiating_packets(self):
        self.packet_list.clear()
        capture = pyshark.LiveCapture(interface="Wi-Fi")
        for packet in capture.sniff_continuously(packet_count=25):
            try:
                if "<UDP Layer>" in str(packet.layers) and "<IP Layer>" in str(packet.layers):
                    self.packet_list.append(packet)
                elif "<TCP Layer>" in str(packet.layers) and "<IP Layer>" in str(packet.layers):
                    self.packet_list.append(packet)
            except:
                print(f"No Attribute name 'ip' {packet.layers}")
    def udp_packet_attributes(self,packet):
        attr_list = list()
        a1 = packet.ip.ttl
        a2 = packet.ip.proto
        a3 = self.__get_service(packet.udp.port, packet.udp.dstport)
        a4 = packet.ip.len
        a5 = random.randrange(0,1000)
```

```
        a6 = self.__get_land(packet,a2)
        a7 = 0        # urgent pointer not exist in udp layer
        a8, a10, a11 = self.__get_count_with_same_and_diff_service_rate(packet.udp.dstport, a3)
#23, 29, 30
        a9, a12 = self.__get_srv_count_and_srv_diff_host_rate(packet.ip.dst, a3) #24, 31
        a13, a15, a16 = self.__get_dst_host_count(packet.ip.dst, a3) # 32,34,35
        a14, a17, a18 = self.__get_dst_host_srv_count(packet.udp.port, packet.udp.dstport,
packet.ip.dst) #33, 36, 37
        attr_list.extend((a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18))
        return self.get_all_float(attr_list)

    def tcp_packet_attributes(self,packet):
        attr_list = list()
        a1 = packet.ip.ttl #duration
        a2 = packet.ip.proto    #protocol
        a3 = self.__get_service(packet.tcp.port, packet.tcp.dstport) # service
        a4 = packet.ip.len #Src - byte
        a5 = random.randrange(0,1000) #dest_byte
        a6 = self.__get_land(packet,a2) #land
        a7 = packet.tcp.urgent_pointer #urgentpoint
        a8, a10, a11 = self.__get_count_with_same_and_diff_service_rate(packet.tcp.dstport, a3)
#23, 29, 30
        a9, a12 = self.__get_srv_count_and_srv_diff_host_rate(packet.ip.dst, a3) #24, 31
        a13, a15, a16 = self.__get_dst_host_count(packet.ip.dst, a3) # 32,34,35
        a14, a17, a18 = self.__get_dst_host_srv_count(packet.tcp.port, packet.tcp.dstport,
packet.ip.dst) #33, 36, 37
        attr_list.extend((a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18))
        return self.get_all_float(attr_list)      # convert every attribute to float data type

    def__get_service(self,src_port,dst_port):
        services = [80,443,53]
        if int(src_port) in services:
            return int(src_port)
        elif int(dst_port) in services:
            return int(dst_port)
        else:
            return 53

    def__get_land(self,packet, protocol):
        if int(protocol) == 6:
            if(packet.ip.dst == packet.ip.src and packet.tcp.port == packet.tcp.dstport):
                return 1
            else:
```

```
            return 0
        elif int(protocol) == 17:
            if(packet.ip.dst == packet.ip.src and packet.udp.port == packet.udp.dstport):
                return 1
            else:
                return 0


    def __get_count_with_same_and_diff_service_rate(self,dst_port, service): #23, 29, 30
        count = 0
        packet_with_same_service = 0
        for p in self.packet_list:
            if "<UDP Layer>" in str(p.layers):
                if (p.udp.dstport == dst_port):         #same destination port
                    count+=1
                    if (self.__get_service(p.udp.port, p.udp.dstport) == service): # same service
                        packet_with_same_service+=1
            elif "<TCP Layer>" in str(p.layers):
                if (p.tcp.dstport == dst_port):
                    count+=1
                    if (self.__get_service(p.tcp.port, p.tcp.dstport) == service):
                        packet_with_same_service+=1
        same_service_rate=0.0
        diff_service_rate = 1.0
        if not count==0:
            same_service_rate = ((packet_with_same_service*100)/count)/100
            diff_service_rate = diff_service_rate-same_service_rate
        return (count, same_service_rate, diff_service_rate)

    def __get_srv_count_and_srv_diff_host_rate(self,dst_ip, service): #24, 31
        diff_dst_ip = 0
        service_count = 0
        for p in self.packet_list:
            if "<UDP Layer>" in str(p.layers):
                    if (self.__get_service(p.udp.port, p.udp.dstport) == service): # same service
                        service_count+=1
                        if not (p.ip.dst == dst_ip):          # different destination ip if udp
                            diff_dst_ip+=1
            elif "<TCP Layer>" in str(p.layers):
                if (self.__get_service(p.tcp.port, p.tcp.dstport) == service):
                        service_count+=1
                        if not (p.ip.dst == dst_ip):           # # different destination ip if tcp
                            diff_dst_ip+=1
```

```
        srv_diff_host_rate = 0.0
        if not(service_count == 0):
            srv_diff_host_rate = ((diff_dst_ip*100)/service_count)/100
        return (service_count, srv_diff_host_rate)

    def __get_dst_host_count(self,dst_ip, service): #32, 34, 35
        same_dst_ip = 0
        same_service=0
        for p in self.packet_list:
            if(p.ip.dst == dst_ip): # same destination ip
                same_dst_ip+=1
                if "<UDP Layer>" in str(p.layers):
                    if (self.__get_service(p.udp.port, p.udp.dstport) == service): # same service if udp
                            same_service+=1
                elif "<TCP Layer>" in str(p.layers):
                    if (self.__get_service(p.tcp.port, p.tcp.dstport) == service): # same service if tcp
                            same_service+=1
        dst_host_same_srv_rate = 0.0
        dst_host_diff_srv_rate = 1.0
        if not same_dst_ip==0:
            dst_host_same_srv_rate = ((same_service*100)/same_dst_ip)/100
            dst_host_diff_srv_rate = dst_host_diff_srv_rate-dst_host_same_srv_rate
        return (same_dst_ip, dst_host_same_srv_rate, dst_host_diff_srv_rate)

    def __get_dst_host_srv_count(self,src_port, dst_port, dst_ip): #33, 36, 37
        dst_host_srv_count = 0
        same_src_port = 0
        diff_dst_ip = 0
        for p in self.packet_list:
            if "<UDP Layer>" in str(p.layers):
                if (p.udp.dstport == dst_port):      # same destination port
                    dst_host_srv_count+=1
                    if (p.udp.port == src_port):    # same src port
                        same_src_port+=1
                    if not (p.ip.dst == dst_ip):        # different destination Ip
                        diff_dst_ip+=1

            elif "<TCP Layer>" in str(p.layers):
                if (p.tcp.dstport == dst_port):     # same destination port
                    dst_host_srv_count+=1
                    if (p.tcp.port == src_port):     # same src port
                        same_src_port+=1
                    if not (p.ip.dst == dst_ip):        #different destination ip
```

```
                diff_dst_ip+=1
        dst_host_same_src_port_rate = 0.0
        dst_host_srv_diff_host_rate = 0.0
        if not dst_host_srv_count==0:
            dst_host_same_src_port_rate = ((same_src_port*100)/dst_host_srv_count)/100
            dst_host_srv_diff_host_rate = ((diff_dst_ip*100)/dst_host_srv_count)/100
        return (dst_host_srv_count, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate)


    def get_all_float(self,l):

        all_float = list()
        for x in l:
            all_float.append(round(float(x),1))
        return all_float
```

ABNIDS.py

```
# Change testing panel to avoid segmentation fault
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtGui import QIcon, QPixmap
from PyQt5.QtWidgets import
qApp,QFileDialog,QMessageBox,QMainWindow,QDialog,QDialogButtonBox,QVBoxLayout,
QHeaderView, QMessageBox
import os
import time
import pyshark
import matplotlib.pyplot as plt
import threading
import packet as pack
import GAAlgorithm
import Preprocess as data
import classifier


class Ui_MainWindow(object):
    def __init__(self):
        self.tree_classifier = classifier.DecisionTree()
        self.packet = pack.Packet()
        self.trained = False
        self.stop = False
        self.threadActive = False
```

```
        self.pause = False
    def plot_graph(self):
        x = ['Normal','DoS','Prob']
        normal,dos,prob = self.tree_classifier.get_class_count()
        y = [normal,dos,prob]
        plt.bar(x,y,width=0.3,label="BARCHART")
        plt.xlabel('Classes')
        plt.ylabel('Count')
        plt.title('Graph Plotting')
        plt.legend()
        plt.show()

    def train_model(self):
        try:
            train_dataset, train_dataset_type = QFileDialog.getOpenFileName(MainWindow,
"Select Training Dataset","","All Files (*);;CSV Files (*.csv)")
            if train_dataset:
                os.chdir(os.path.dirname(train_dataset))
                test_dataset, test_dataset_type = QFileDialog.getOpenFileName(MainWindow,
"Select Testing Dataset","","All Files (*);;CSV Files (*.csv)")
            if train_dataset and test_dataset:
                generation  = 0
                train_dataset = data.Dataset.refine_dataset(train_dataset, "Train Preprocess.txt")

                test_dataset = data.Dataset.refine_dataset(test_dataset, "Test Preprocess.txt")
                #Start Genetic Algorithm
                ga =
GAAlgorithm.GAAlgorithm(train_dataset,test_dataset,population_size=5,mutation_rate=65)
                ga.initialization() # if error occur due to invalid dataset population needs to be clear to
avoid append of new population
                ga.calculate_fitness()
                while(ga.population.max_fitness<93 and generation<1):
                    print(f"Generation = {generation}")
                    generation+=1
                    parents = ga.selection()
                    ga.cross_over(parents)
                    ga.mutation()
                    ga.calculate_fitness()
                max_fitest = ga.population.max_fittest
                max_fitness = round(ga.population.max_fitness,1)
                self.tree_classifier.train_classifier(train_dataset,max_fitest)
                self.trained = True
                ga.clear_population()
```

```
            self.progressBar.setProperty("value", 100)
            self.showdialog('Model train',f'Model trained successfully',1)

        except:
            try:
                ga.clear_population()
            except:
                print("Err 00")
            finally:
                self.showdialog('Model train','Model trained unsuccessfully',2)


    def static_testing(self):
        if self.isModelTrained():
            if (self.threadActive):
                self.showdialog('Warning','Please stop currently testing',3)
            else:
                test_dataset, train_dataset_type = QFileDialog.getOpenFileName(MainWindow,
"Select Testing Dataset","","All Files (*);;CSV Files (*.csv)")
                if test_dataset:
                    try:
                        test_dataset = data.Dataset.refine_dataset(test_dataset, "Test Dataset.txt")
                        t1 = threading.Thread(target=self.static_testing_thread, name = 'Static testing',
args=(test_dataset,))
                        t1.start()
                        self.threadActive = True
                    except:
                        self.showdialog('Error','Invalid Dataset',2)
        else:
            self.showdialog('Warning','Model not trained',3)

    def static_testing_thread(self,dataset):
        row = 0
        self.reset_all_content()
        with open(dataset,"r") as file:
            for line in file.readlines():
                try:
                    line = line.split(',')
                    result, result_type = self.tree_classifier.test_dataset(line)
                    self.insert_data(line,result,result_type,row)

                    row+=1
                    if self.pause:
```

```
                while(self.pause):
                    pass
            if self.isStop():
                self.stop=False
                break
            time.sleep(0.05)
        except:
            print("Err")
    self.threadActive = False


def realtime_testing(self):
    if self.isModelTrained():
        if (self.threadActive):
            self.showdialog('Warning','Please stop currently testing',3)
        else:
            t2 = threading.Thread(target=self.realtime_testing_thread, name = 'Realtime testing')
            t2.start()
            self.threadActive = True
    else:
        self.showdialog('Warning','Model not trained',3)
def realtime_testing_thread(self):
    self.reset_all_content()
    self.packet.initiating_packets()
    t1 = time.time()
    attr_list  = list()
    capture = pyshark.LiveCapture(interface='Wi-Fi')
    row = 0
    try:
        for p in capture.sniff_continuously():
            try:
                if "<UDP Layer>" in str(p.layers) and "<IP Layer>" in str(p.layers):
                    attr_list = self.packet.udp_packet_attributes(p)
                    result, result_type = self.tree_classifier.test_dataset(attr_list)
                    self.insert_data(attr_list,result,result_type,row)
                    print(attr_list)
                    row+=1
                elif "<TCP Layer>" in str(p.layers) and "<IP Layer>" in str(p.layers):
                    attr_list = self.packet.tcp_packet_attributes(p)
                    result, result_type = self.tree_classifier.test_dataset(attr_list)
                    self.insert_data(attr_list,result,result_type,row)
                    print(attr_list)
                    row+=1
```

```
                if (time.time()-t1) > 5 and not self.isStop: # 5Seconds
                    print("Updateing List")
                    self.packet.initiating_packets()
                    t1 = time.time()
                if self.pause:
                        while(self.pause):
                            pass
                if self.isStop():
                    self.stop=False
                    break
            except :
                print("Err")
        except :
            print("Error in looooooop")

    def pause_resume(self):
        if self.pause:
            self.pause = False
            self.btn_start.setText("Pause")
        else:
            self.pause = True
            self.btn_start.setText("Resume")


    def save_log_file(self):
        log = self.tree_classifier.get_log()
        url = QFileDialog.getSaveFileName(None, 'Save Log', 'untitled', "Text file (*.txt);;All
Files (*)")
        if url[0]:
            try:
                name = url[1]
                url = url[0]
                with open(url, 'w') as file:
                    file.write(log)
                self.showdialog('Saved',f'File saved as {url}',1)
            except:
                self.showdialog('Error','File not saved',2)

    def stop_capturing_testing(self):
        if self.pause:
            self.pause = False
            self.btn_start.setText('Pause')
        if not self.stop:
```

```
            self.stop = True
        if self.threadActive:
            self.threadActive = False
    def reset_all_content(self):
        if self.pause:
            self.pause = False
            self.btn_start.setText('Pause')
        self.stop=False
        self.tree_classifier.reset_class_count()
        self.panel_capturing.clearContents()
        self.panel_capturing.setRowCount(0)
        self.panel_result.clearContents()
        self.panel_result.setRowCount(0)
        self.panel_testing.clear()


    def insert_data(self,line,result,result_type,row):
        self.panel_capturing.insertRow(row)
        for column, item in enumerate(line[0:4:1]):
            self.panel_capturing.setItem(row,column,QtWidgets.QTableWidgetItem(str(item)))
            self.panel_capturing.scrollToBottom()
        self.panel_testing.clear()
        self.panel_testing.addItem(str(line[0:4:1]))
        if not result==0:
            result_row = self.panel_result.rowCount()
            self.panel_result.insertRow(result_row)
            x = [row+1, line[1], line[2], result_type]
            for column, item  in enumerate(x):

self.panel_result.setItem(result_row,column,QtWidgets.QTableWidgetItem(str(item)))
                self.panel_result.scrollToBottom()

    def clickexit(self):
        buttonReply = QMessageBox.question(MainWindow, 'Exit', "Are ou sure to exit?",
QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
        if buttonReply == QMessageBox.Yes:
            if self.threadActive:
                self.pause = False
                self.stop = True
            qApp.quit()
        else:
            print('No clicked.')
```

```
    def isStop(self):
        return self.stop
    def showdialog(self,title,text, icon_type):
        msg = QMessageBox()
        if icon_type==1:
            msg.setIcon(QMessageBox.Information)
        elif icon_type==2:
            msg.setIcon(QMessageBox.Critical)
        elif icon_type==3:
            msg.setIcon(QMessageBox.Warning)
        msg.setText(text)
        msg.setWindowTitle(title)
        msg.setStandardButtons(QMessageBox.Ok)
        msg.buttonClicked.connect(self.msgbtn)
        retval = msg.exec_()

    def msgbtn(self):
        self.progressBar.setProperty("value", 0)
    def isModelTrained(self):
        return self.trained
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        path = os.path.dirname(os.path.abspath(__file__))
        MainWindow.setWindowIcon(QtGui.QIcon(os.path.join(path,'icon.png')))
        MainWindow.resize(908, 844)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())
        MainWindow.setSizePolicy(sizePolicy)
        MainWindow.setIconSize(QtCore.QSize(30, 30))
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)
        self.gridLayout.setObjectName("gridLayout")
        spacerItem = QtWidgets.QSpacerItem(10, 10, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.gridLayout.addItem(spacerItem, 1, 0, 1, 1)
        spacerItem1 = QtWidgets.QSpacerItem(20, 20, QtWidgets.QSizePolicy.Minimum,
```

```
QtWidgets.QSizePolicy.Maximum)
        self.gridLayout.addItem(spacerItem1, 4, 1, 1, 1)
        spacerItem2 = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Fixed)
        self.gridLayout.addItem(spacerItem2, 6, 1, 1, 1)
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        spacerItem3 = QtWidgets.QSpacerItem(15, 10, QtWidgets.QSizePolicy.Ignored,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_2.addItem(spacerItem3)
        self.btn_start = QtWidgets.QPushButton(self.centralwidget)

        self.btn_start.setObjectName("btn_start")
        self.btn_start.setText('Pause')
        self.btn_start.clicked.connect(self.pause_resume)
        self.horizontalLayout_2.addWidget(self.btn_start)

        # ###########################################################
        self.btn_pause = QtWidgets.QPushButton(self.centralwidget)
        self.btn_pause.setText("Stop Capturing/Testing")

        self.btn_pause.setObjectName("btn_pause")
        self.btn_pause.clicked.connect(self.stop_capturing_testing)
        self.horizontalLayout_2.addWidget(self.btn_pause)
        self.gridLayout.addLayout(self.horizontalLayout_2, 8, 1, 1, 1)
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        # ###########################################################
        self.btn_modeltrain = QtWidgets.QPushButton(self.centralwidget)
        self.btn_modeltrain.setText("Train Model")

        self.btn_modeltrain.setObjectName("btn_modeltrain")
        self.btn_modeltrain.clicked.connect(self.train_model)
        self.horizontalLayout.addWidget(self.btn_modeltrain)
        # ###########################################################
        self.btn_statictesting = QtWidgets.QPushButton(self.centralwidget)
        self.btn_statictesting.setText("Static Testing")

        self.btn_statictesting.setObjectName("btn_statictesting")
        self.btn_statictesting.clicked.connect(self.static_testing)
        self.horizontalLayout.addWidget(self.btn_statictesting)
        # ###########################################################
        self.btn_realtimetesting = QtWidgets.QPushButton(self.centralwidget)
```

```
        self.btn_realtimetesting.setText("")


        self.btn_realtimetesting.setObjectName("")
        self.btn_realtimetesting.clicked.connect(self.realtime_testing)
        self.horizontalLayout.addWidget(self.btn_realtimetesting)

        # ###########################################################
        self.btn_savelog = QtWidgets.QPushButton(self.centralwidget)
        self.btn_savelog.setText("Save Log")
        icon5 = QtGui.QIcon()

        self.btn_savelog.setObjectName("btn_savelog")
        self.btn_savelog.clicked.connect(self.save_log_file)
        self.horizontalLayout.addWidget(self.btn_savelog)

        # ###########################################################
        self.btn_graph = QtWidgets.QPushButton(self.centralwidget)
        self.btn_graph.setText("Plot Graph")

        self.btn_graph.setObjectName("btn_graph")
        self.btn_graph.clicked.connect(self.plot_graph)
        self.horizontalLayout.addWidget(self.btn_graph)

        # ###########################################################
        self.btn_exit = QtWidgets.QPushButton(self.centralwidget)
        self.btn_exit.setText("Exit")


        self.btn_exit.setObjectName("btn_exit")
        self.btn_exit.clicked.connect(self.clickexit)
        self.horizontalLayout.addWidget(self.btn_exit)
        # ###########################################################
        self.gridLayout.addLayout(self.horizontalLayout, 3, 1, 1, 2)
        spacerItem4 = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Fixed)
        self.gridLayout.addItem(spacerItem4, 8, 1, 1, 1)
        spacerItem5 = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Fixed)
        self.gridLayout.addItem(spacerItem5, 0, 1, 1, 1)
        self.panel_capturing = QtWidgets.QTableWidget(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Preferred)
```

```
        sizePolicy.setHorizontalStretch(10)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.panel_capturing.sizePolicy().hasHeightForWidth())
        self.panel_capturing.setSizePolicy(sizePolicy)
        self.panel_capturing.setRowCount(0)
        self.panel_capturing.setColumnCount(4)
        self.panel_capturing.setObjectName("panel_capturing")
        item = QtWidgets.QTableWidgetItem()
        self.panel_capturing.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.panel_capturing.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.panel_capturing.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.panel_capturing.setHorizontalHeaderItem(3, item)
        self.gridLayout.addWidget(self.panel_capturing, 4, 1, 4, 1)
        self.label = QtWidgets.QLabel(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.label.sizePolicy().hasHeightForWidth())
        self.label.setSizePolicy(sizePolicy)
        self.label.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.label.setAutoFillBackground(False)
        self.label.setText("")
        path = os.path.dirname(os.path.abspath(__file__))
        path = path + r'\icons'
        self.label.setPixmap(QtGui.QPixmap(os.path.join(path,'logo.jpg')))
        self.label.setScaledContents(True)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setObjectName("label")
        self.gridLayout.addWidget(self.label, 1, 1, 1, 1)
        spacerItem6 = QtWidgets.QSpacerItem(10, 20, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Fixed)
        self.gridLayout.addItem(spacerItem6, 2, 1, 1, 1)
        self.panel_testing = QtWidgets.QListWidget(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.panel_testing.sizePolicy().hasHeightForWidth())
        self.panel_testing.setSizePolicy(sizePolicy)
```

```
        self.panel_testing.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
        self.panel_testing.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)
        self.panel_testing.setObjectName("panel_testing")
        self.gridLayout.addWidget(self.panel_testing, 9, 1, 1, 1)
        self.progressBar = QtWidgets.QProgressBar(self.centralwidget)
        self.progressBar.setProperty("value", 0)
        self.progressBar.setObjectName("progressBar")
        self.gridLayout.addWidget(self.progressBar, 10, 1, 1, 2)
        #_____________________________________________#

        self.panel_result = QtWidgets.QTableWidget(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(10)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.panel_result.sizePolicy().hasHeightForWidth())
        self.panel_result.setSizePolicy(sizePolicy)
        self.panel_result.setRowCount(0)
        self.panel_result.setColumnCount(4)
        self.panel_result.setObjectName("panel_result")
        item = QtWidgets.QTableWidgetItem()
        self.panel_result.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.panel_result.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.panel_result.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.panel_result.setHorizontalHeaderItem(3, item)
        self.gridLayout.addWidget(self.panel_result, 4,2,6,1)
        #_____________________________________________#
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 908, 26))
        self.menubar.setObjectName("menubar")
        self.menuFile = QtWidgets.QMenu(self.menubar)
        self.menuFile.setObjectName("menuFile")
        self.menuAbout = QtWidgets.QMenu(self.menubar)
        self.menuAbout.setObjectName("menuAbout")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.actionNew = QtWidgets.QAction(MainWindow)
```

```
        self.actionNew.setObjectName("actionNew")
        self.actionOpen = QtWidgets.QAction(MainWindow)
        self.actionOpen.setObjectName("actionOpen")
        self.actionExit = QtWidgets.QAction(MainWindow)
        self.actionExit.setObjectName("actionExit")
        self.actionHelp = QtWidgets.QAction(MainWindow)
        self.actionHelp.setObjectName("actionHelp")
        self.menuFile.addAction(self.actionNew)
        self.menuFile.addAction(self.actionOpen)
        self.menuFile.addSeparator()
        self.menuFile.addAction(self.actionExit)
        self.actionExit.triggered.connect(qApp.quit)
        self.menuAbout.addAction(self.actionHelp)
        self.menubar.addAction(self.menuFile.menuAction())
        self.menubar.addAction(self.menuAbout.menuAction())

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Intrusion Detection"))
        self.btn_start.setStatusTip(_translate("MainWindow", "Pause/Resume"))
        self.btn_pause.setStatusTip(_translate("MainWindow", "Stop"))
        self.btn_modeltrain.setStatusTip(_translate("MainWindow", "Train Model"))
        self.btn_statictesting.setToolTip(_translate("MainWindow", "Stactic Testing"))
        self.btn_statictesting.setStatusTip(_translate("MainWindow", "Static Testing"))

        self.btn_savelog.setToolTip(_translate("MainWindow", "Real Time Capturing"))
        self.btn_savelog.setStatusTip(_translate("MainWindow", "Real Time Capturing"))
        self.btn_graph.setStatusTip(_translate("MainWindow", "Graph"))
        self.btn_exit.setStatusTip(_translate("MainWindow", "Exit"))
        item = self.panel_capturing.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Duration"))
        item = self.panel_capturing.horizontalHeaderItem(1)
        item.setText(_translate("MainWindow", "Protocol"))
        item = self.panel_capturing.horizontalHeaderItem(2)
        item.setText(_translate("MainWindow", "Service"))
        item = self.panel_capturing.horizontalHeaderItem(3)
        item.setText(_translate("MainWindow", "Src_Bytes"))
        #_____________________________________#
        item = self.panel_result.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Packet #"))
```

```
        item = self.panel_result.horizontalHeaderItem(1)
        item.setText(_translate("MainWindow", "Protocol"))
        item = self.panel_result.horizontalHeaderItem(2)
        item.setText(_translate("MainWindow", "Service"))
        item = self.panel_result.horizontalHeaderItem(3)
        item.setText(_translate("MainWindow", "Class"))
        #_____________________________________#
        self.menuFile.setTitle(_translate("MainWindow", "File"))
        self.menuAbout.setTitle(_translate("MainWindow", "About"))
        self.actionNew.setText(_translate("MainWindow", "New"))
        self.actionOpen.setText(_translate("MainWindow", "Open"))
        self.actionExit.setText(_translate("MainWindow", "Exit"))
        self.actionHelp.setText(_translate("MainWindow", "Help"))


if__name__== "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Conclusion:

Network traffic logs to describe patterns of behavior in network traffic accident with intrusive or normal activity. Decision tree technique is good for the intrusion characteristic of the network traffic logs for IDS and implemented in the genetic algorithm as prevention.The other hand, this technique is also good efficiency and optimize rule for the firewall rules such as avoid redundancy.

REFERENCES:

[1] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN," Future Generation Computer Systems, vol. 111, pp. 763-779, 2020, doi: 10.1016/j.future.2019.10.015.
[2]"Software Defined Networking Definition." https://www.opennetworking.org/sdn-definition/ (accessed March, 2, 2020).

[3] S. Garg, K. Kaur, N. Kumar, and J. J. P. C. Rodrigues, "Hybrid Deep-Learning-Based Anomaly Detection Scheme for Suspicious Flow Detection in SDN: A Social Multimedia Perspective," IEEE Transactions on Multimedia, vol. 21, no. 3, pp. 566-578, 2019, doi: 10.1109/tmm.2019.2893549.
[4] M. Nobakht, V. Sivaraman, and R. Boreli, "A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow," presented at the 2016 11th International Conference on Availability, Reliability and Security (ARES), 2016.
[5] M. S. Elsayed, N. Le-Khac, S. Dev, and A. D. Jurcut, "Machine Learning Techniques for Detecting Attacks in SDN," in 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), 19-20 Oct. 2019 2019, pp. 277-281, doi: 10.1109/ICCSNT47585.2019.8962519.

Publication

INTRUSION DETECTION IN SOFTWARE DEFINED NETWORK USING MACHINE LEARNING

Submission date: 02-Mar-2022 09:59AM (UTC-0600)

Submission ID: 1649798424

File name: INTRUSION_DETECTION_IN_SOFTWARE_DEFINED_NETWORK_USING_MACHINE_LEARNING_1.docx (365.3K)

Word count: 2100

Character count: 11888

INTRUSION DETECTION IN SOFTWARE DEFINED NETWORK USING MACHINE LEARNING

***ABSTARCT*:**

In recent decades, researchers have been working on improving intrusion detection systems (IDS). Computer networks can be detected as potentially harmful by using Genetic Algorithms (GAs). A series of classification rules has been generated based on various features of connection data, such as duration and types of connections. Detection of data originating from attackers or originals is easily done in this proposed work. This proposed work is to detect the data which is received from attacker or non-attacker node.

Keywords- Intrusion detection, K-Nearest Neighbor, Naive Bayes, Decision Trees, Support Vector Machine, Prediction

I INTRODUCTION

A system for monitoring network traffic alerts administrators to suspicious activity when it is detected. Data breaches can be detected using intrusion detection systems (IDS). In this software, harmful activity or policy breaches are scanned for on a network or system. An administrator or a SIEM system is used to collect data about malicious endeavors or violations. Using alarm filtering, the SIEM identifies malicious and erroneous entries by analyzing multiple data sources. A DoS attack shuts down a computer or network by attempting to disrupt its functioning and make it unavailable to its intended users. The DoS attack accomplishes this by sending a flood of traffic to a target or triggering a crash as a result of information sent there.

Probing attacks are an invasive method for bypassing security measures by observing the physical silicon implementation of a chip. As an invasive attack, one directly accesses the internal wires and connections of a targeted device and extracts sensitive information.

This proposed work is focused on lowering risk, identifying errors, increasing network efficiency, providing insight into threat levels, and changing user behavior. By creating tree-like representations for representations of classes and attributes, GAs are aimed at predicting variables' values. An attack on a network is an unauthorized action that is performed on the digital assets of an organization. Network attacks are usually carried out by malicious parties to alter, destroy, or steal private data. During a DDoS attack, attackers flood targets such as computers, smartphones, and internet-connected devices with traffic from exploited computers. Attacks that probe chips are an invasive method of circumventing security measures by paying attention to how the physical silicon is implemented. Invasive attacks include gaining access to an electronic device's internal wiring and connections and extracting sensitive information. The process of distinguishing between intrusions and normal network traffic is time consuming and difficult.

- 1 To identify the sequence of intrusion on the network connection, an analyst must review large and diverse amounts of data.
- 1 For network traffic to be reflected, it must be able to detect network intrusion.
- The combination of an intrusion detection system and a firewall is known as an intrusion prevention system. It does not only detect intrusions, but also prevent them after they have been detected.

In this proposed system, the attacker is identified by using genetic algorithm.

II LITERATURE REVIEW

Software-defined Networking (SDNs) have as of late been created as a feasible and promising answer for the eventual fate of the Internet. Networks are made due, incorporated, and observed and adjusted utilizing SDN. These advantages, then again, bring us ecological dangers, for example, network crashes, framework incapacities, internet banking misrepresentation, and robbery. These issues can detrimentally affect families, organizations, and the economy. Truth, superior execution, and the genuine framework are fundamental to accomplishing this objective. In the previous decade, software-defined networks (SDNs) have been greatly enhancing network intrusion detection systems (NIDSs) by utilizing intelligent AI algorithms. The accessibility of data, the distinction in information investigation, and the many advances in AI calculations assist us with making a superior, more dependable, and solid framework for distinguishing the various sorts of organization assaults. The review was essential for the NIDS SDN survey. Network Intrusion Detection Systems (NIDSs) are a significant device for network framework overseers to decide network security. NIDS screens and examines approaching and active calls from family network gadgets and cautions assuming that entrance is identified. As far as access control, NIDS is separated into two classifications: I) NIDS (SNIDS) based mark

(abuse), and ii) NIDS (ADNIDS) based secrecy location. The helpful plan is made of against slip vehicle to permit admittance to the organization. Interestingly, ADNIDS permits network traffic to stream in when it is going to split away from typical traffic. However, ADNIDS is critical to be familiar with obscure and new assaults. In spite of the fact that ADNIDS estimates its adequacy well, its capacity to identify new assaults has prompted its far and wide acknowledgment. There are two issues that function admirably in the advancement of NIDS: gentle and direct assaults. Above all else, the strategy for choosing the right traffic information from the informational index line is hard to distinguish peculiarities. Because of steady vacillations and changes, the capacities chose at a similar assault level may not be reasonable for other assault classes. Second, there is an absence of a bunch of traffic information from the genuine line of NIDS improvement. It requires a ton of work to separate a bunch of genuine or ongoing recorded information from the crude line of the gathered way.

With worldwide availability, network security has become more associated with innovative work. As the quantity of assaults builds, the firewall has turned into a significant security strategy issue overall. Firewalls can be permitted or denied over the organization, however since firewalls can't be recognized or assaulted, signing in and applying to a firewall is a method for controlling how you forestall it. Access location Firewall innovation is viewed as an extra answer for identify interruptions in an organization without a firewall. Firewalls and IDS address the old as far as data innovation security. A firewall is great for ensuring frameworks and networks and lessens the danger of organization assaults. IDS can identify endurance or assault. Capacity to interface IDS and firewalls called IPS. That is the fair thing to do, and it should end there. There is at least one distinct standard for every retailer. Each organization parcel that arrives at the firewall should be tried by characterized rules until an appropriate rule is found. Under current law, bundles will be permitted or restricted from arriving at the line. Every law determines a particular kind of vehicle. The points of interest of how the pipeline will be sold should be visible from the lines of vehicles from people's perspective. This review plans to try not to attempt to sign in to look for Internet-based substance, like IDS, and afterward implementing firewall rules like impeding. Need to find out about our information mining machine security strategy. The technique used to make the standard is to rank the ID3 calculation by tree endorsement. It's a decent and great practice to implement firewalls.

The Network Access System (NIDS) assumes a significant part in ensuring PC organizations. Be that as it may, there are worries about the accessibility and maintainability of current innovation to meet present day network necessities. Specifically, these worries are connected with the increment in individuals' level of correspondence and the lessening in their level of information. This paper presents new top to bottom examination techniques to comprehend and resolve these issues. We plainly characterize non-standard encoder (NDAE) prerequisites for the investigation of uncontrolled items. Furthermore, we suggest a top to bottom investigation of the classes made utilizing the NDAE. Our proposals were carried out in GPU-TensorFlow and assessed utilizing the KDD Cup '99 scale and the NSL-KDD informational index. The main limitation of existing survey is less accuracy for DoS attack and Probe attack.

III Proposed System

Monitoring systems for intrusion (IDS) detect suspicious activity in network traffic and alert users accordingly. An antimalware program is a piece of software that scans a network or system to detect harmful activities. Implementing firewall rules and an intrusion detection system (IDS) are commonly used to prevent DoS at the network or infrastructure level. IDSs block traffic from suspicious sources once an attack is detected. Cyber attackers often fail to get past this approach, because it's simple enough and effective. Detection systems for collaborative intrusions now face probe-response attacks. Probes are attacks that are designed to be detected by their target and to produce recognizable "fingerprints" in their documentation. Using Genetic Algorithm identify the attacks in an efficient manner. The different modules in the proposed system is discussed below:

Dataset collection:

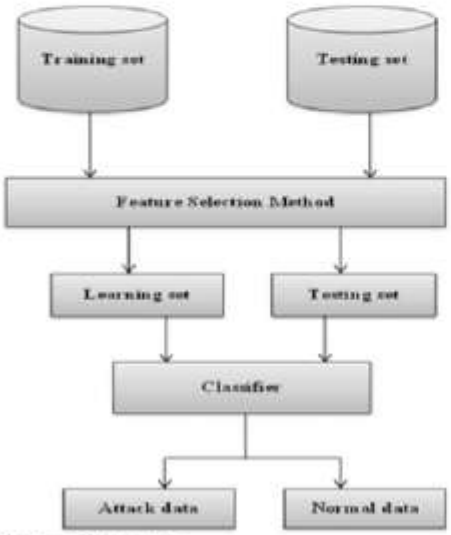
By analyzing data, you can uncover recurring patterns by keeping a record of past events.

KDD datasets:

In the research of Intrusion Detection techniques, the KDD data set is widely respected.

Data cleaning:

Every machine learning project begins with data cleaning. As part of this module, we clean data to prepare it for analysis. Incorrect or incomplete data, duplicates, or incorrectly formatted data can be deleted or modified.



Feature Extraction:

This is done to reduce the number of attributes in the dataset hence providing advantages like speeding up the training and accuracy improvements.

Model training:

An ML algorithm is trained using a training model. Along with each set of input data, the output data is included to influence the result.

Testing model:

In this module we test the trained machine learning model using the test dataset

Genetic Algorithm:

Natural selection and genetics are the fundamental principles of genetic algorithms. Each individual signifies a solution of the problem to be solved after GA creates a population of initial individuals. Chromosomes contain predetermined numbers of genes and are unique to each individual. As a numerical representation of a rule's adaptation to a particular environment, a fitness function measures a rule's quality. A population of random individuals is created at the beginning. As time passes, the population evolves while gradually improving the individual's quality as measured by their fitness value. Selection, crossover, and mutation are three of the three basic genetic operators used in every generation.

Decision Tree Algorithm:

With supervised learning techniques, people tend to use Decision Trees when they need to solve Classification problems, and Regression Trees when they need to solve Regression problems. Tree-structured classifiers represent a dataset's various characteristics, the rules are the branches and the results are the leaves. The Decision Node and Leaf Node are both parts of a Decision tree. Leaf nodes produce the output of decisions made by decision nodes, but they do not have any branches. Based on the features of the given dataset, decisions are made or tests are performed.

The decision tree algorithm classifies the attacker data and normal data after training and computation of this algorithm.

Algorithm:

1. In the given dataset, select the target attribute
2. Calculate the Information Gain for target attribute

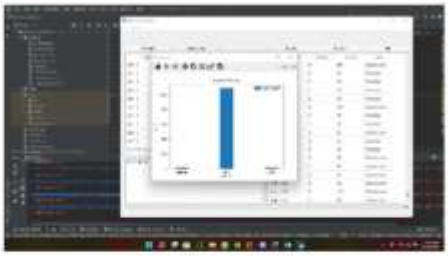
$$TA = -\frac{A}{A+T} log_2 \frac{A}{A+T} - \frac{T}{A+T} log_2 \frac{T}{A+T}$$

3. Calculate Entropy
   Entropy=TA*Probability
4. $Ent(Attr) = \sum \frac{Ai+Ti}{A+T} I(AiTi)$
5. Calculate Gain=TA-Ent(Attr)
   In this dataset

IV Results and Discussion

The experimental analysis is done by evaluating IDS dataset (KDD).It has 41 characteristics and class label. For this experiment only Probe attack and DoS attack is taken for implementation. The subclasses for DoS attacks is land ,apache2,smurf,mailbomb,teardrop.

For Probe attack the subclasses are Ipsweep, portsweep,satan and saint.The training instances for DoS attack is 1500 and testing instances is 1800. For probe attack 1050 instances and testing instances as 1700.



The proposed work Accuracy is calculated as

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

The TP-True Positive, TN-True Negative, FP-False Positive and FN-False Negative. The accuracy by using decision tree is 89% accuracy.

CONCLUSION:

Detours depict personal conduct standards that happen during street mishaps and typical exercises. The tree managing method is the most ideal to the working of the IDS access street and is executed in the hereditary calculation of avoidance. Then again, this innovation functions admirably and maintains a strategic distance from over-the-top guidelines, like firewalls.

REFERENCES:

[1] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN," Future Generation Computer Systems, vol. 111, pp. 763-779, 2020.

[2] S. Garg, K. Kaur, N. Kumar, and J. J. P. C. Rodrigues, "Hybrid Deep-Learning-Based Anomaly Detection Scheme for Suspicious Flow Detection in SDN: A Social Multimedia Perspective," IEEE Transactions on Multimedia, vol. 21, no. 3, pp. 566-578, 2019.

[3] M. Nobakht, V. Sivaraman, and R. Boreli, "A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow," presented at the 2016 11th International Conference on Availability, Reliability and Security (ARES), 2016.

[4] M. S. Elsayed, N. Le-Khac, S. Dev, and A. D. Jurcut, "Machine Learning Techniques for Detecting Attacks in SDN," in 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), 19-20 Oct. 2019, pp. 277-281.

INTRUSION DETECTION IN SOFTWARE DEFINED NETWORK USING MACHINE LEARNING

ORIGINALITY REPORT

2%	1%	1%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Aston University Student Paper	1%
2	opus.lib.uts.edu.au Internet Source	1%
3	Peng Cui. "A Tighter Analysis of Set Cover Greedy Algorithm for Test Set", Lecture Notes in Computer Science, 2007 Publication	<1%
4	Bambang Susilo, Riri Fitri Sari. "Intrusion Detection in Software Defined Network Using Deep Learning Approach", 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), 2021 Publication	<1%
5	thesai.org Internet Source	

<1%

Exclude quotes	On	Exclude matches	Off

Detection of Attacks (DoS, Probe) Using Genetic Algorithm

A. Venkata Srinadh Reddy[1], B. Prasanth Reddy[2], L. Sujihelen[3]
[1, 2, 3] Department of Computer Science and Engineering
Sathyabama Institute of Science and Technology, Chennai.

***ABSTARCT*:**

The entrance framework (IDS) is right now exceptionally fascinating as a significant piece of framework security. The IDS gathers traffic data from the line or framework and afterward involves it for better security. Assaults are typically truly challenging and tedious to isolate street exercises. To screen the organization association, the examiner should survey all data, enormous and wide. Subsequently, an organization search strategy is expected to decide the recurrence of traffic. In this review, another strategy for looking for IDS identifiers was created utilizing a technique for concentrating on information mining procedures from a calculation machine. The technique used to set the principles is to sort the choice tree and calculation. These guidelines can be utilized to decide the idea of the assault and afterward apply it to the hereditary calculation for avoidance, so that as well as distinguishing the assault, it is feasible to find ways to forestall the assault and deny the assault.

Keywords- Intrusion detection*, *K-Nearest Neighbor*, *Naive Bayes*, *Decision Trees, Support Vector Machine, Prediction

INTRODUCTION

Input techniques can be partitioned into two kinds: misconstruing and deformity location. A wide range of known (irresistible) assaults can be distinguished by evaluating the normal interruption pace of the framework for checking the means of misconception. In the case of something surprising occurs, the framework initially learns the ordinary profile and afterward records every one of the components of the framework that don't match the set up profile. The principle advantage of discovery is the maltreatment of the capacity to identify new or surprising assaults at high rates, making it hard to distinguish.

The upside of having the option to identify uncommon things is the capacity to recognize new (or startling) assaults that convey many advantages. Procedures dependent on innovation pipelines utilized in different ventures. We give general data to the investigation of traffic data and for the location of street mishaps utilizing the significant distance-course of-the-street

The proposed technique utilizes tests dependent on the issue of eliminating traffic data via online media (Facebook and Twitter): this movement gathers sentences connected with all traffic exercises, for example, traffic stops or street terminations. The quantity of starting handling strategies is presently executed. breathing, signal presentation, POS signal, partition, and so forth to change the data acquired in the inherent structure. The information is then consequently shown as "traffic" or "traffic" utilizing the latent Dirichlet allocation (LDA) calculation. Vehicle enrollment data is isolated into three kinds; great, terrible and impartial. The response to this classification is the expression enraptured (positive, negative, or unbiased) as for street sentences, contingent upon whether or not it is traffic. The bag-of-words (BoW) is presently used to change each sentence over to a solitary hot code to take care of bi-directional LSTM organizations (Bi-LSTM). In the wake of preparing, a multi-stage muscle network utilizes softmax to arrange sentences as indicated by area, vehicle experience, and sort of polarization. The proposed strategy contrasts the preparation of various machines and the high-level preparing techniques as far as precision, F scores, and different standards.

LITERATURE REVIEW

Designing a Network Intrusion Detection System Based on Machine Learning for Software Defined Networks

Software-defined Networking (SDNs) have as of late been created as a feasible and promising answer for the eventual fate of the Internet. Networks are made due, incorporated, and observed and adjusted utilizing SDN. These advantages, then again, bring us ecological dangers, for example, network crashes, framework incapacities, internet banking misrepresentation, and robbery. These issues can detrimentally affect families, organizations, and the economy. Truth, superior execution, and the genuine framework are fundamental to accomplishing this objective. The extension of wise AI calculations into the network intrusion detection system (NIDS) through a software-defined network (SDN) has been extremely invigorating over the previous decade. The accessibility of data, the distinction in information investigation, and the many advances in AI calculations assist us with making a superior, more dependable, and solid framework for distinguishing the various sorts of organization assaults. The review was essential for the NIDS SDN survey.

A Deep Learning Approach for Network Intrusion Detection System

Network Intrusion Detection Systems (NIDSs) are a significant device for network framework overseers to decide network security. NIDS screens and examines approaching and active calls from family network gadgets and cautions assuming that entrance is identified. As far as access control, NIDS is separated into two classifications: I) NIDS (SNIDS) based mark (abuse), and ii) NIDS (ADNIDS) based secrecy location. SNIDS and Drinking put assault marks first in NIDS. The helpful plan is made of against slip vehicle to permit admittance to the organization. Interestingly, ADNIDS permits network traffic to stream in when it is going to split away from typical traffic. Significant in characterizing SNIDS. notable, notable assault, non-salvage assault. Nonetheless, its unmistakable makes it extremely challenging to distinguish obscure or new assaults on the grounds that the marks of pre-introduced assaults on the IDS are decreased. However, ADNIDS is critical to be familiar with obscure and new assaults. In spite of the fact that ADNIDS estimates its

adequacy well, its capacity to identify new assaults has prompted its far and wide acknowledgment. There are two issues that function admirably in the advancement of NIDS: gentle and direct assaults. Above all else, the strategy for choosing the right traffic information from the informational index line is hard to distinguish peculiarities. Because of steady vacillations and changes, the capacities chose at a similar assault level may not be reasonable for other assault classes. Second, there is an absence of a bunch of traffic information from the genuine line of NIDS improvement. It requires a ton of work to separate a bunch of genuine or ongoing recorded information from the crude line of the gathered way.

Intrusion Preventing System using Intrusion Detection System Decision Tree Data Mining

With worldwide availability, network security has become more associated with innovative work. As the quantity of assaults builds, the firewall has turned into a significant security strategy issue overall. Firewalls can be permitted or denied over the organization, however since firewalls can't be recognized or assaulted, signing in and applying to a firewall is a method for controlling how you forestall it. Access location Firewall innovation is viewed as an extra answer for identify interruptions in an organization without a firewall. Firewalls and IDS address the old as far as data innovation security. A firewall is great for ensuring frameworks and networks and lessens the danger of organization assaults. IDS can identify endurance or assault. Capacity to interface IDS and firewalls called IPS. That is the fair thing to do, and it should end there. There are at least one distinct standard for every retailer. Each organization parcel that arrives at the firewall should be tried by characterized rules until an appropriate rule is found. Under current law, bundles will be permitted or restricted from arriving at the line. Every law determines a particular kind of vehicle. The points of interest of how the pipeline will be sold should be visible from the lines of vehicles from people's perspective. This review plans to try not to attempt to sign in to look for Internet-based substance, like IDS, and afterward implementing firewall rules like impeding. Need to find out about our information mining machine security strategy. The technique used to make the standard is to rank the ID3 calculation by tree endorsement. It's a decent and great practice to implement firewalls.

A Deep Learning Approach to Network Intrusion Detection

The Network Access System (NIDS) assumes a significant part in ensuring PC organizations. Be that as it may, there are worries about the accessibility and maintainability of current innovation to meet present day network necessities. Specifically, these worries are connected with the increment in individuals' level of correspondence and the lessening in their level of information. This paper presents new top to bottom examination techniques to comprehend and resolve these issues. We plainly characterize non-standard encoder (NDAE) prerequisites for the investigation of uncontrolled items. Furthermore, we suggest a top to bottom investigation of the classes made utilizing the NDAE. Our proposals were carried out in GPU-TensorFlow and assessed utilizing the KDD Cup '99 scale and the NSL-KDD informational index.

EXISTING SYSTEM:

- Today, pipelines have turned into a significant piece of public foundation and the computation of public or private mists.
- Techniques Traditional organization network has turned into a test.
- These troubles have forestalled the foundation of new and forward-thinking administrations in a similar organization, making it hard to associate organizations, business associations, and the Internet overall.

Problem Statement:

- Attacks are truly challenging, typical, and tedious to isolate street exercises.
- Utilizes Analysts need to think about enormous and wide-going data to screen the seriousness of pipelines.
- Technique The strategy used to recognize the pipelines is expected to decide the progression of traffic.
- Associating a firewall to an IDS, otherwise called an IDS, can distinguish an assault, however can likewise keep it from assaulting.

Proposed System:

- Hereditary Algorithms are one of the most generally utilized techniques for AI as far as availability.
- Cold The choice sheet looks at the test to one of the qualities of a specific case, while the leaf shows the possibility of whether the result is in the ordinary or typical period of the assault (potentially a potential assault).
- Strategy A better approach to observe IDS tokens utilizing an authentication tree. A strategy for AI has been given. The technique utilized in lawmaking is to sort the choice tree and calculation.

Advantages:

- Attack location should be possible physically or consequently.
- IDS should have the option to adapt to the hours of development and exposure.
- It is vital to utilize a choice tree. Understanding programmed assaults and how to react is turning out to be progressively significant.

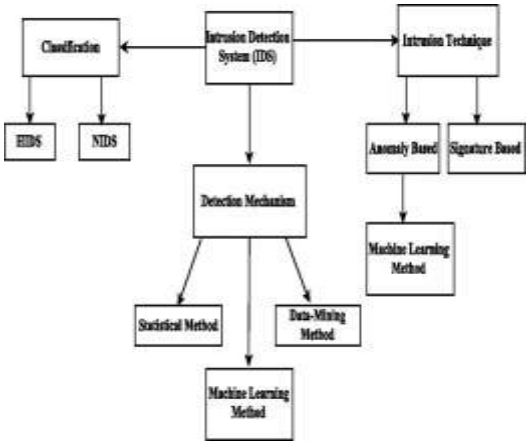
HARDWARE REQUIREMENTS:

- System : Pentium i3 Processor.
- HDD : 500 GB.
- Screen : 15'' LED
- Devices : Keyboard, Mouse
- Random Access Memory: 2 GB

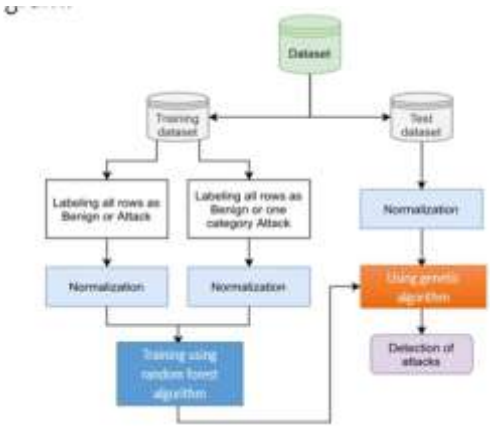
SOFTWARE REQUIREMENTS:

- Software : Windows 10.
- Language : Python

BLOCK DIAGRAM:



FLOW DIAGRAM:



The absolute most broadly utilized calculations.

- K-Neighbor
- Blameless Bays
- Choice tree/Natural woodland
- Support for vector machines
- Intercession

Decision tree

Introduction

Up until this point, we have figured out how to go this way and that, and it has been hard to comprehend. Presently how about we start with "Tree Decision", I guarantee you it very well may be a straightforward calculation in Machine Learning. There aren't so numerous here. It is one of the most broadly utilized and commonsense strategies for AI since it is not difficult to utilize and clarify.

What is a Decision Tree?

It is an instrument with applications running in better places. The testament tree can be utilized in similar class as obsolete issues. The actual name recommends that it utilizes plans, for example, trees to show prescience from the request in which things are isolated. It begins at the root and finishes with the choice to get away. Before we study the choice tree, how about we investigate a few words.
Root Nodes The top of this hub is toward the start of the choice tree, and the public starts to isolate it as indicated by different elements.
Decision Nodes - The gatherings we see subsequent to isolating the root are called Resolutions
Leaf Nodes - an indivisible head called a leaf or leaf
Sub-tree - 33% of the sub-tree plan, a large portion of the exactness of the sub-tree.
Pruning - There is nothing to do except for remove the head to quit trying too hard.

MODULES:

- Dataset collection
- Data Cleaning
- Feature Extraction
- Model training
- Testing model
- Performance Evaluation
- Prediction

Dataset collection:

Informational index assortment:
Information assortment can assist you with tracking down ways of following previous occasions utilizing information examination to record them. This permits you to foresee the way and make prescient models utilizing AI devices to anticipate future changes. Since the prescient model is just pretty much as great as the data acquired, the most effective way to gather information is to further develop execution. The data ought to be faultless (garbage, open air squander) and ought to incorporate data about the work you are doing. For instance, a non-performing advance may not profit from the sum got, yet may profit from gas costs over the long run. In this module, we gather data from the kaggle data set. These figures contain data on yearly contrasts.

Data cleaning:
Data cleanliness is a significant piece of all AI exercises. The data cleanliness of this module is expected for the arrangement of information for the annihilation and transformation of wrong, inadequate, deluding or misdirecting data. You can utilize it to look for data. Discover what cleaning you can do.

Feature Extraction:

This is done to lessen the quantity of capacities in the informational index, which will accelerate preparing and increment proficiency.
In AI, picture acknowledgment, and picture handling, mining starts at the front line of estimated, useful data (ascribes) pointed toward guaranteeing, adjusting, following, and normalizing data, and now and again prompting more prominent clearness. Take out the properties related with aspect decrease
On the off chance that the calculation's feedback is excessively enormous, it won't be handled, and assuming it is suspected to be excessively huge (like estimating one foot and meter, or rehashing the picture displayed in pixels), it tends to be switched. properties (likewise called vector properties).
Characterize the initial segment, called highlight choice. The chose things ought to contain data about the data got so they can fill the ideal role utilizing this portrayal rather than complete data.

Model training:

An illustration of this preparation is the informational collection used to prepare the ML calculation. It comprises of significant info definitions that influence information inspecting and yield.

The preparation model is utilized to utilize the information through the result and result change calculations. The aftereffects of this connection will be utilized to alter the layout.

This strategy for assault is designated "matching model". Information preparing definition or informational collection approval is significant for demonstrating.

Plan language preparing is a method for giving data about the ML calculation and assist with deciding and become familiar with the best significance of every one of its highlights. There are many kinds of AI, the majority of which are controlled and uncontrolled.

Testing model:

In this module, we test an AI machine planned utilizing research information

Quality protection is needed to make the product framework work appropriately. All chances settled upon? Does the program fill in true to form? All program testing standards should be remembered for the specialized detail.

What's more, programming testing can uncover every one of the defects and shortcomings that have happened during improvement. Once the application is delivered, you don't need your clients to come to your home together. Various kinds of tests just take into account recognition of blunders during activity.

Performance Evaluation:

In this module, we audit the presentation of an AI framework utilizing execution assessment measures, for example, F1 scores, exactness, and arrangement mistake.

At the point when the model performs inadequately, we change the AI to further develop execution.

Execution examination is characterized as a norm and productive method for estimating representative execution dependent on worker obligations. It is utilized to gauge the worth of representatives by expanding their business pay contrasted with industry and all out venture (ROI).

All associations that have taken in the specialty of "mutual benefit" depend on the presentation of their workers dependent on an exhibition examination framework to continually survey and assess the presentation of its representatives.

In a perfect world, workers are evaluated yearly upon the arrival of the occasion, in view of advancement or compensation increment.

Execution examination plays an immediate part to play in giving input to workers to all the more likely comprehend their principles.

Prediction:

Consistency "alludes to the outcomes subsequent to preparing the calculation on the historical backdrop of the set and carrying out it when you expect the chance of a specific outcome, for example, deciding whether the client will remain for 30 days.

The worth-based calculation can be changed for each new thing composed, permitting the author to decide the worth that is destined to be.

"Speculation" can be misdirecting. Now and then, this implies foreseeing the future, like utilizing a machine to decide the following game-plan.

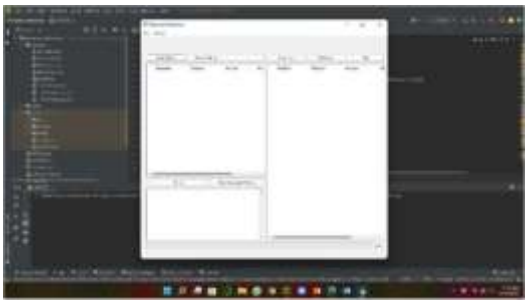
In different cases, "prescience" is connected, for instance, in the event that the item has as of now been created.

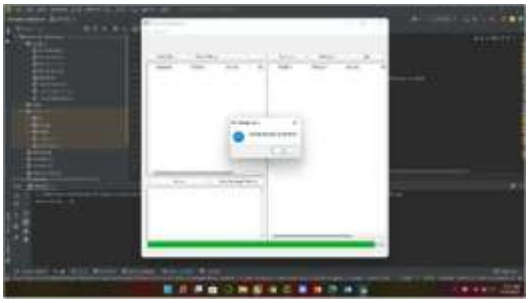
For this situation, the move has as of now been made, however it will assist you with giving input on whether it is satisfactory and to make a proper move.

In this module, we utilize an organized, AI technique to decide whether the patient will respond to a portion of the inquiries.

RESULT:

Train and Test the dataset





After completion of training and testing the KDD dataset, now dataset which contain the attacks undergoes for static testing. After completion of testing it shows the attacks data in plot graph.

Now, we can static

testing of the normal dataset for attacks

classification

CONCLUSION:

Detours depict personal conduct standards that happen during street mishaps and typical exercises. The tree managing method is the most ideal to the working of the IDS access street and is executed in the hereditary calculation of avoidance. Then again, this innovation functions admirably and maintains a strategic distance from over-the-top guidelines, like firewalls.

REFERENCES:

[1] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN," Future Generation Computer Systems, vol. 111, pp. 763-779, 2020, doi: 10.1016/j.future.2019.10.015.

[2]"Software Defined Networking Definition." https://www.opennetworking.org/sdn-definition/ (accessed March, 2, 2020).

[3] S. Garg, K. Kaur, N. Kumar, and J. J. P. C. Rodrigues, "Hybrid Deep-Learning-Based Anomaly Detection Scheme for Suspicious Flow Detection in SDN: A Social Multimedia Perspective," IEEE Transactions on Multimedia, vol. 21, no. 3, pp. 566-578, 2019, doi: 10.1109/tmm.2019.2893549.

[4] M. Nobakht, V. Sivaraman, and R. Boreli, "A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow," presented at the 2016 11th

International Conference on Availability, Reliability and Security (ARES), 2016.

[5] M. S. Elsayed, N. Le-Khac, S. Dev, and A. D. Jurcut, "Machine Learning Techniques for Detecting Attacks in SDN," in 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), 19-20 Oct. 2019 2019, pp. 277-281, doi: 10.1109/ICCSNT47585.2019.8962519.