VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Information Security Analysis and Audit (CSE3501)

PROJECT TITLE:

NETWORK INTRUSION DETECTION USING MACHINE LEARNING AND DEEP LEARNING

TEAM MEMBERS:

VINIT KUMAR SINGH (20BCE2841)
SUBHAM KUMAR(20BCE0101)
MANN PANDYA(20BCE2527)

SUBMITTED TO :

PROF. Rajarajan G

Index

Abstract

Intrusion Detection System (IDS) defined as a Device or software application which monitors the network or system activities and finds if there is any malicious activity occur. Outstanding growth and usage of internet raises concerns about how to communicate and protect the digital information safely. In today's world hackers use different types of attacks for getting the valuable information. Many of the intrusion detection techniques, methods and algorithms help to detect those several attacks. The main objective of this project is to detect types of attacks, different tools and techniques, That tool are capable to detect and prevent the intrusion from the intruder. In our project we are using the IDS-2018 Dataset and NSL KDD dataset to test our models usingdeep learning algorithms and some machine learning algorithms.

Keywords: Machine learning , Deep learning, NSL KDD, IDS 2018,Feature extraction, Data preprocessing

Introduction

Intrusion detection plays a vital role in the network defense process by aiming security administrators in forewarning them about malicious behaviors such as intrusions, attacks, and malware. Research in IDS has, hence, flourished over the years. An Intrusion detection system orIDS is a system developed to monitor for suspicious activity and issues alerts when such activity is discovered. The primary aim of IDS is to detect anomalous activities, but some systems are also able to take action against these intrusions like blocking traffic from the suspicious IP address. An IDS can also be used to help analyze the quantity and types of attacks. Organizations can use this data to change their security systems or implement more effectivesystems. An IDS can also help organisations to identify bugs or problems with their network device configurations.

Objectives

General Objective

The project aims at adapting an intrusion detection system appropriate emphasizing the importance of these to network security. It points to the need for integration of various intrusion prevention mechanisms so as to harden the intrusion detection system. This work takes into account the limitation of resources available in providing

network security.

Specific Objectives

The specific objectives of the research has been to:

- ✓ Investigate Network Intrusion Detection Systems(IDS).
- ✓ Design an effective intrusion detection system.
- ✓ Implementation of an effective scalable intrusion detection system.
- ✓ Validation of an intrusion detection system

Literature Studies:

(Vinit Kumar Singh (20BCE2841))

(1) Deep Learning for Network Anomalies Detection

Problem:

Internet of Things, Software-Defined networks, and grid computing, introduce unprecedented security threats. These challenges entail innovative solutions.

Technique used and why it is used:

In this paper, semi-supervised DL based detection framework for discovering the network abnormalities. It shows the prospect oF DL in this field.

Advantages:

Network anomalies detection is an area where DL can improve the detection precision.

Comparison :

There are comparisions with the semi-supervised deep learning approach and Their results showed comparable or better performance to methods like PCA and kernel PCA.

Limitations:

In networking intrusion applications, the network traffic usually have a limited number of features, while DL shines with more data.

(2) Intrusion Detection System Using an Optimized Framework Based on Datamining Techniques

Problem:

Unauthorized use, exploitation or damage to network resources and systems. There are 4 attack types: Denial of Service Attack (DoS), User to Root Attack (U2R), Remote to Local Attack (R2L), Probing Attack (Probe).

Techniques Used and Why it is used:

An optimized framework for network attack detection is presented using data mining techniques.

Advantages:

It detects network intrusion with higher precision. . One of the most important qualities of the suggested framework is its applicability to both supervised and unsupervised methods

Comparsion

Demonstrating an improvement compared with the new ensemble clustering (NEC) method.

(3) Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches

Problem:

With the increased use of IoT infrastructure in every domain, threats and attacks in these infrastructures are also growing commensurately.

Technique used and why it is used:

Machine learning (ML) algorithms that have been used here and comparative analysis is done between various algorithm to find which is the best.

Advantage:

The system obtained 99.4% test accuracy for Decision Tree, Random Forest, and ANN. Though these techniques have the same accuracy, other metrics prove that Random Forest performs comparatively better.

Comparison:

Compared to the papers this paper provides much detailed description of the dataset. It also provides a clear explanation of the dataset preprocessing steps. .

Limitations:

It does not assure that in the case of the big data and other unknown problems RF will perform this way.

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

Problem:

Software-defined Networking (SDN) brings vulnerable environment and dangerous threats, causing network breakdowns,systems paralysis, online banking frauds and robberies

Technique used and why it is used:

XGBoost Model is proposed for anomaly detection. The proposed method enhances the implementation of NIDS by deploying machine learning over SDN.

Advantage:

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).

Comparison:

XGBoost algorithm with the selected five features achieved a high accuracy score.Also it has evaluate the proposed method using an accuracy analysis against seven classical machine learning algorithms,

(Shubham Kumar (20BCE0101))

5) An Empirical Study on Network Anomaly Detection using Convolutional NeuralNetwork

Problem:

Deep learning has been widely applied to network anomaly detection to improve performance.

Technique used and why it is used:

In this study, they evaluated three simple CNN models with different internal depths for network anomaly detection, to see how the structural depths impact the performance.

Comparison :

There are comparisons with this paper [1] on the following techniques FCN, VAE, Seq2Seq-LSTM. The author had observed that the evaluated CNN models occasionally outperform the VAE models, but do not work better than the other deep learning models based on FCN and Seq2Seq-LSTM

Limitations of the study:

From this work, they learned that simply adding more layers to a CNN structure is not beneficial and does not improve performance.

6) AD-IoT: Anomaly Detection of IoT Cyberattacks in Smart City Using MachineLearning

Problem:

In order to mitigate the cyberattacks, the developers need to enhance new techniques for detecting infected IoT devices.

Technique used and why it is used:

They proposed a detection method called **Anomaly Detection[1]IoT (AD-IoT) system,** which is an intelligent anomaly detection based on Random Forest machine learning algorithm. The proposed solution can effectively detect compromised IoT devices at distributed fog nodes as illustrated in fig.(2).

They supposed that this method works to monitor the network traffic that passes through each fog node, as fog nodes are nearest to IoT sensors, rather than detection on the huge amount of the city's cloud storage to identify among normal and abnormal behaviors. After detecting attacks in the fog level, it should alert the security cloud services to inform them to analyze and update their system.

Advantage :

The proposed AD-IoT can significantly detect malicious behavior .

Comparison :

To evaluate the model, they utilized modern dataset to illustrate the model's accuracy. Their findings showed that the AD-IoT can effectively achieve highest classification accuracy of 99.34% with lowest false positive rate.

Limitations of the study:

This proposed design does not yet finalized the machine learning model and not yet account for urban life network designs.

7) MnasNet: Platform-Aware Neural Architecture Search for Mobile

Problem:

Designing convolutional neural networks (CNN) for mobile devices is challenging because mobile models need to be small and fast, yet still accurate. Although significant efforts have been dedicated to design and improve mobile CNNs on all dimensions, it is very difficult to manually balance these trade-offs when there are so many architectural possibilities to consider.

Technique used and why it is used:

In this paper, they propose an automated

which explicitly incorporate model latency into the main objective so that the search can identify a model that achieves a good trade-off between accuracy and latency.

Advantage :

Unlike previous work, where latency is considered via another, often inaccurate proxy (e.g., FLOPS), their approach directly measures real-world inference latency by executing the model on mobile phones. To further strike the right balance between flexibility and search space size, they proposed a novel factorized hierarchical search space that encourages layer diversity throughout the network.

Comparison :

Experimental results show that our approach consistently outperforms state-of-the-art mobile CNN models across multiple vision tasks. Their MnasNet also achieves better mAP quality than MobileNets for COCO object detection.

8) ANAMOLY DETECTION FOR SAFETY MONITORING

Problem:

In crowded scene abnormal event detection is a major issue. Many existing methods are there. Abnormal events are those which cannot be well represented. For example , if a flight is hijacked or it is damaged , it is due to some abnormal activities. Abnormal activities may occur due to human intervention or due to some weather conditions.

Technique used and why it is used:

In this system they are using **abnormal detector** to detect the events..In this complexity is high in video events due to the presence of noise. By using mixture of Gaussian interference can be avoided.

A flowchart for the proposed algorithm for abnormal event detection is mentioned in paper[4]. They capture the image from the running video. And then the images are captured into segments. By using kalman filter, segmentation process is performed. It produces better quality of images. The captured images are sent to (IOT), if they find any abnormalities. Using IOT the abnormal events are sent to the corresponding person mail id.

Advantage :

In this project they have detected abnormal activities. It is not only applicable to aircraft monitoring system, also possible in crowded places or crowded cities. We are using anomaly detector technique to overcome the abnormal events. Here GMM plays a major role.

Limitations of the study:

The scope for future work will be using some other SC based algorithms to generate the detector set. Hybrid approach with detector set using GA can upgrade the advanced genetic operators. This can be done without any human intervention.

(Mann Pandya (20BCE2527))

9) Learning Transferable Architectures for Scalable Image Recognition

Problem:

Developing neural network image classification models often requires significant architecture engineering.

Techniques Used and Why it is used:

Our approach is inspired by the recently proposed Neural Architecture Search (NAS) framework , which uses a reinforcement learning search method to optimize architecture configurations.

Advantages of the techniques implemented:

it is much faster than searching for an entire network architecture and the cell itself is more likely to generalize to other problems.The accuracy of the resulting model exceeds all human-designed models – ranging from models designed for mobile applications to computationally-heavy models designed to achieve the most accurate results.

Comparison :

NASNet achieves, among the published works, state-of-the-art accuracy of 82.7% top-1 and 96.2% top-5. This result amounts to a 1 arXiv:1707.07012v4 [cs.CV] 11 Apr 2018 1.2% improvement in top-1 accuracy than the best humaninvented architectures while having 9 billion fewer FLOPS. On CIFAR-10 itself, NASNet achieves 2.4% error rate, which is also state-of-the-art.

Limitations:

10.)***The Role of the Weibull Distribution in Internet Traffic Modeling***

Problem:

Identifying a parsimonious structural model for Internet traffic is still an open research problem.

Techniques Used and Why it is used:

This paper highlights the important role played by the two parameter Weibull distribution in Internet traffic modeling.

Advantages of the techniques implemented:

It has was concluded that the Scaling phenomenon observed at small time scales is related to the arrival process of packets with in flows. They highlighted the need for simple renewal process modeling as compared to complex processes such as multifractals to model traffic data.

Comparison :

The Weibull shape parameter values of packet, flow and session inter-arrivals increases (due to high multiplexing) at the core ISP level compared to the corresponding values at the access networks. This means that the tail of these random variables becomes less heavy and thus traffic burstiness decreases (or traffic tends to Short Range Dependent) as it moves from access to core network.

Limitations:

11.) *MnasNet: Platform-Aware Neural Architecture Search for Mobile*

Problem:

Designing convolutional neural networks (CNN) for mobile devices is challenging because mobile models need to be small and fast, yet still accurate.

Techniques Used and Why it is used:

In this paper, it's propose an automated mobile neural architecture search (MNAS) approach

Comparison :

On the ImageNet classification task, the MnasNet achieves 75.2% top-1 accuracy with 78ms latency on a Pixel phone, which is 1.8× faster than MobileNetV2 [29] with 0.5% higher accuracy and 2.3× faster than NASNet [36] with 1.2% higher accuracy.

Limitations:

Designing a resource-constrained mobile model is challenging: one has to carefully balance accuracy and resource-efficiency, resulting in a significantly large design space.

12.) *AD-IoT: Anomaly Detection of IoT Cyberattacks Smart City Using Machine Leaming*

Problem:

In recent years, the wide adoption of the modern Internet of Things (IoT) paradigm has led to the invention of smart cities.. The network traffic of a smart city via IoT systems is growing exponentially and introducing new cybersecurity challenges

Techniques Used and Why it is used:

Anomaly detection based on Random Forest machine learning algorithm.

Advantages of the techniques implemented:

As of now, no prior researches have examined this method AD-IoT which differs from previous studies. Most of the mentioned existing IDS approaches do not investigate the smart city. Furthermore, this model learns from normal traffic by training machine learning algo rithm such as RF algorithm to detect any malicious behaviour in the future

Comparison :

findings show that the AD-IoT can effectively achieve highest classification accuracy of 99.34% with lowest false positive rate.

(13) ***Machine Learning based Intrusion Detection Framework using Recursive Feature Elimination Method.*** **(a)**Huge amount of data is collected from the network.Because of this the computational time and cost is high.In order to overcomethis Feature selection or Feature elimination techniques are used.In the proposed system Recursive feature elimination method is used.

(b) Recursive feature elimination-It is the process of removing redundant and irrelevant data from the dataset.

Classification techniques-1) LDA:Linear Discriminant Analysis (LDA) used to getThe linear combination of features for dimensionality reduction before classification.

2) AdaBoost:Adaptive Boosting which classifier to strong classifier there by achieving better accuracy..3) SVM-Radial:It is maximizes the margin between different classes of training data. 4)Random forest:It creates decision trees on data samples using bagging and feature randomness and gives the prediction.

System architecture-fig 1 page no. 2

(c) By using feature elimination technique a lot of redundant data is removed and the computational cost and time are reduced.This paper perform a comparative analysis among classification models.Among this models Adaboost gives the better results.So the proposed system is capable of detecting the traffic types with better correctness when measure up to the other existing systems.

(d) There are comparision done with the classifications technique LDA,SVMs,random forest and Adaboosting algorithm and the best one is chosen.(Table 2 page no.4)

In fig2 Accuracy by cross validation graph is given which shows the performance of thesystem .

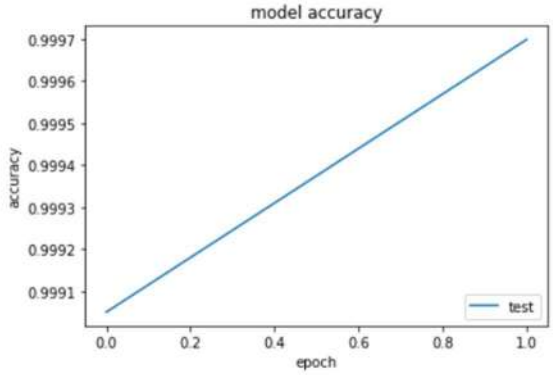
Existing System/ Model

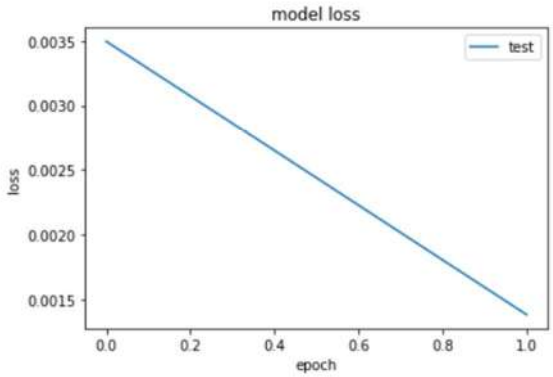
- ✓ CIC IDS- Dataset 2018 by Canadian Institute of Technology.
- ✓ Machine Learning algorithm classifier Random Forest Classifier, Essemble Learning classifier, Naïve Bayes classifier.
- ✓ Semi-supervised DL based detection framework for discovering the network abnormalities. It shows the prospect of DL in this field.

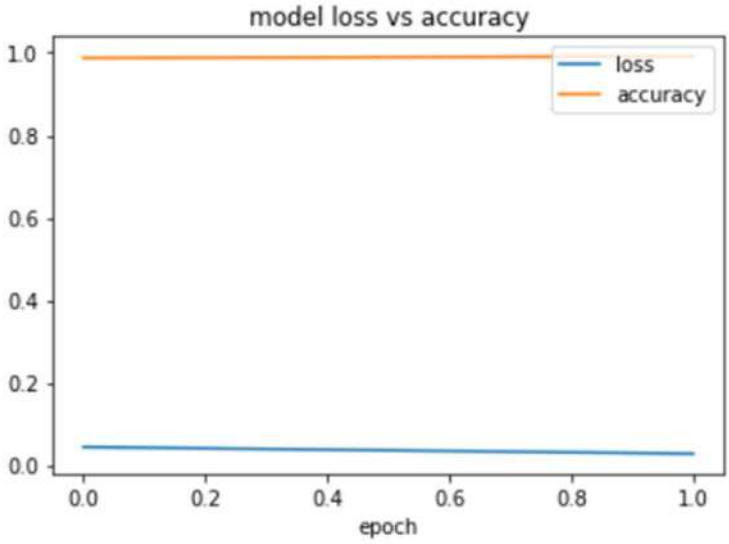
Proposed Model

Implementing an Intrusion and Anomaly Detection System using various Machine Learning and Deep Learning binary and multi-class thresholding frameworks like KNNs, Random Forest, Linear SVM, Random forest and others and determining the model accuracy in each case. The implementation design provides a framework for the modelling of effective intrusion detection systems. Integration of intrusion detection systems with a line of intrusion prevention mechanisms will greatly improve on the system performance. Moreover, it also ensures high system scalability since devices can easily be added or removed from the system without affecting its overall performance. Configurations not being tied onto a single devise reduces the chances of having a single point of failure. The system is also effective since it looks at both external attacks and internal attacks that make up one of the most dangerous threats to network security.

Outcomes of Proposed Model







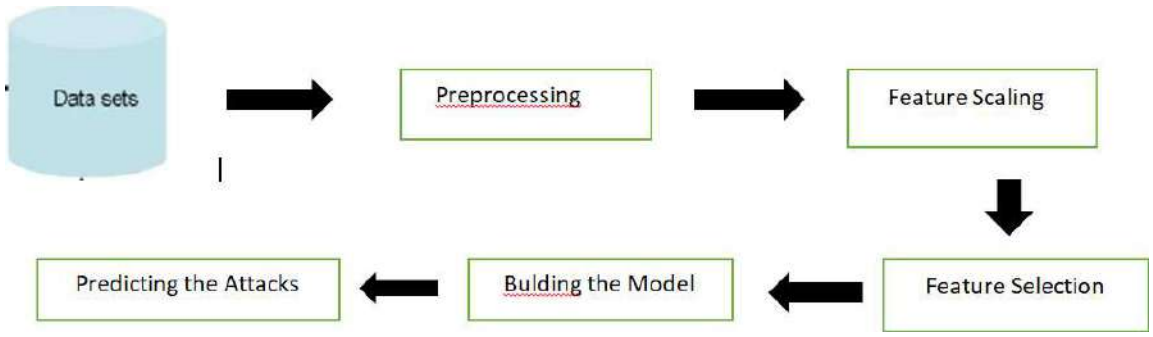
Merits

- ✓ Able to find the accuracy of model.
- ✓ Integration of intrusion detection systems with a line of intrusion prevention mechanisms will greatly improve on the system performance.
- ✓ Configurations not being tied onto a single devise reduces the chances of having a single point of failure.
- ✓ The system is also effective since it looks at both external attacks and internal attacks that make up one of the most dangerous threats to network security.

Demerits

- ✓ System will not able do to prevention kind of work within the Intrusion detection system.
- ✓ It will not detect some Social Engineering attacks like phishing , spamming etc.

HIGH LEVEL DIAGRAM



Algorithm/PseudoCode

Random Forest Algorithm :

A random forest algorithm consists of many decision trees. The 'forest' generated bythe random forest algorithm is trained through bagging or bootstrap aggregating.
Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms.It predicts by taking the average or mean of the output fromvarious trees.

Pseudocode:

1) Randomly select "k" features from total "m" features.
   Where k << m
2) Among the "k" features, calculate the node "d" using the best split point.
3) Split the node into daughter nodes using the best split.
4) Repeat 1 to 3 steps until "l" number of nodes has been reached.
5) Build forest by repeating steps 1 to 4 for "n" number times to create "n" number of trees.

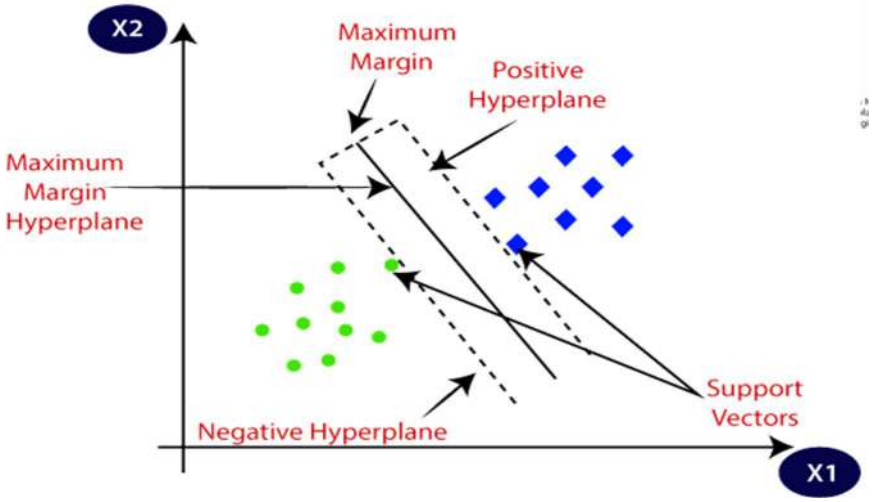
Random forest has nearly the same hyperparameters as a decision tree or a bagging classifierRandom forest adds additional randomness to the model, while
growing the trees. Instead of searching for the most important feature while splittinga node, it searches for the best feature among a random subset of features

Ensemble Algorithm-

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the (classification, prediction, function approximation, etc.) performance of a model, or reduce the likelihood of an unfortunate selection of a poor one. Other applications of ensemble learning include assigning a confidence to the decision made by the model, selecting optimal (or near optimal) features, data fusion, incremental learning, nonstationary learning and error-correcting.

Support Vector Machine(SVM) algorithm

Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features we have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.
Support Vectors are simply the coordinates of individual observation.



KNN Algorithm(K nearest neighbours) :

KNN is based on Supervised Learning technique. It assumes the similarity between the new case/data and available cases and puts the new case into the category that is most similar to the available categories.
The algorithm stores all the available data and classifies a new data point based on the similarity. When new data appears, KNN will classify them considering the greatest categorical count in k number of neighbors.
It can be more effective if the training data is large and can handle noisy data effectively

Algorithm

- Select the number K of the neighbors. We can take k=sqrt(n) for optimal results. Calculate the Euclidean distance of k number of neighbors
- Take the K nearest neighbors as per the calculated Euclidean distance.
- Among these k neighbors, count the number of the data points in each category. Assign the new data points to that category for which the number of the neighbor is maximum.

Implementation

A security mechanism can be implemented using an **Intrusion Detection System (IDS)** which can be described as a collection of software or hardware devices able to collect, analyze and detect any unwanted, suspicious or malicious traffic either on a particular computer host or network. Therefore to achieve its task, an IDS
should use some statistical or mathematical method to read and interpret the information it collects and subsequently reports anymalicious activity to the network administrator.

Using Machine Learning Approach

A data clean process can require a tremendous human effort, which is an extensive time consuming and expensive. A machine learning approach and data mining technique which is the application of machine learning methods to large databases are widely known and used to reduce or eliminate the need of a human interaction.

Machine learning helps to optimize performance criterion using example data or past experience using a computer program, models are defined with some parameters, and learning is the execution of the programming computer to optimize the parameters of the model using training data. The model can be predictive to make predictions in the future, or descriptive to gain knowledge from data.

To perform a predictive or descriptive task, machine learning generally use two main techniques:

Classification and Clustering

All categories are described below:

1) **Basic features**: It contains all features which derived from TCP/IP connection such as Protocol_type, Service, duration etc.

2) **Content features:** Those features use domain knowledge to access the payload of the original TCP packets (e.g. host, num_root, is_guest_login and etc.)

3) **Host-based traffic features**: all attacks which span longer than 2 second intervals that have the same destination host as the current connection are accessed using these features (e.g. dst_host_count,dst_host_srv_count and etc.)

NSL KDD dataset new :-

1) Elimination of redundant records in the training set will help our classifier to be unbiased towards more frequent records.

2) No presence of duplicate records in the test set, therefore, the classifier performance will not be biased by the techniques which have better detection rates on the frequent records.

3) The training and test set contains both a reasonable number of instances which is affordable for experiments on the entire set without the need to randomly choose a small portion.

Proposed Method

Step 1: Data Cleaning and Preprocessing

Step 2: Features scaling

Step3: Features Selection

Step 4: Building the Model

Sample Code

Loading Dataset

```
In [30]: import pandas as pd
         import numpy as np
         import sys
         import sklearn
         import io
         import random

In [31]: train_url = "NSL_KDD_Train.csv"
         test_url = "NSL_KDD_Test.csv"

In [32]: col_names = ["duration","protocol_type","service","flag","src_bytes",
             "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
             "logged_in","num_compromised","root_shell","su_attempted","num_root",
             "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
             "is_host_login","is_guest_login","count","srv_count","serror_rate",
             "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
             "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
             "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
             "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
             "dst_host_rerror_rate","dst_host_srv_rerror_rate","label"]


         df = pd.read_csv(train_url,header=None, names = col_names)

         df_test = pd.read_csv(test_url, header=None, names = col_names)

         print('Dimensions of the Training set:',df.shape)
         print('Dimensions of the Test set:',df_test.shape)

         Dimensions of the Training set: (125973, 42)
         Dimensions of the Test set: (22544, 42)
```

```
In [33]: df.head(5)
```

Out[33]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_d
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	

Label Encoder:-

Insert categorical features into a 2D numpy array

```
In [37]: from sklearn.preprocessing import LabelEncoder,OneHotEncoder
categorical_columns=['protocol_type', 'service', 'flag']

df_categorical_values = df[categorical_columns]
testdf_categorical_values = df_test[categorical_columns]

df_categorical_values.head()
```

Out[37]:

	protocol_type	service	flag
0	tcp	ftp_data	SF
1	udp	other	SF
2	tcp	private	S0
3	tcp	http	SF
4	tcp	http	SF

One-Hot-Encoding

One-Hot-Encoding (one-of-K) is used to to transform all categorical features into binary features. Requirement for One-Hot-encoding: "The input to this transformer should be a matrix of integers, denoting the values taken on by categorical (discrete) features. The output will be a sparse matrix where each column corresponds to one possible value of one feature. It is assumed that input features take on values in the range [0, n_values)."

Therefore the features first need to be transformed with LabelEncoder, to transform every category to a number.

```
In [40]: enc = OneHotEncoder(categories='auto')
df_categorical_values_encenc = enc.fit_transform(df_categorical_values_enc)
df_cat_data = pd.DataFrame(df_categorical_values_encenc.toarray(),columns=dumcols)


# test set
testdf_categorical_values_encenc = enc.fit_transform(testdf_categorical_values_enc)
testdf_cat_data = pd.DataFrame(testdf_categorical_values_encenc.toarray(),columns=testdumcols)

df_cat_data.head()
```

Out[40]:

	Protocol_type_icmp	Protocol_type_tcp	Protocol_type_udp	service_IRC	service_X11	service_Z39_50	service_aol	service_auth	service_bgp	service_courier
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 84 columns

Feature Scaling

In [17]:
```
# Split dataframes into X & Y
# X Properties , Y result variables

X_DoS = DoS_df.drop('label',1)
Y_DoS = DoS_df.label

X_Probe = Probe_df.drop('label',1)
Y_Probe = Probe_df.label

X_R2L = R2L_df.drop('label',1)
Y_R2L = R2L_df.label

X_U2R = U2R_df.drop('label',1)
Y_U2R = U2R_df.label

# test set
X_DoS_test = DoS_df_test.drop('label',1)
Y_DoS_test = DoS_df_test.label

X_Probe_test = Probe_df_test.drop('label',1)
Y_Probe_test = Probe_df_test.label

X_R2L_test = R2L_df_test.drop('label',1)
Y_R2L_test = R2L_df_test.label

X_U2R_test = U2R_df_test.drop('label',1)
Y_U2R_test = U2R_df_test.label
```

Feature Selection :

In [49]:
```
from sklearn.feature_selection import RFE
rfe.fit(X_DoS, Y_DoS.astype(int))
X_rfeDoS=rfe.transform(X_DoS)
true=rfe.support_
rfecolindex_DoS=[i for i, x in enumerate(true) if x]
rfecolname_DoS=list(colNames[i] for i in rfecolindex_DoS)
```

In [57]:
```
rfe.fit(X_Probe, Y_Probe.astype(int))
X_rfeProbe=rfe.transform(X_Probe)
true=rfe.support_
rfecolindex_Probe=[i for i, x in enumerate(true) if x]
rfecolname_Probe=list(colNames[i] for i in rfecolindex_Probe)
```

In [51]:
```
rfe.fit(X_R2L, Y_R2L.astype(int))
X_rfeR2L=rfe.transform(X_R2L)
true=rfe.support_
rfecolindex_R2L=[i for i, x in enumerate(true) if x]
rfecolname_R2L=list(colNames[i] for i in rfecolindex_R2L)
```

In [52]:
```
rfe.fit(X_U2R, Y_U2R.astype(int))
X_rfeU2R=rfe.transform(X_U2R)
true=rfe.support_
rfecolindex_U2R=[i for i, x in enumerate(true) if x]
rfecolname_U2R=list(colNames[i] for i in rfecolindex_U2R)
```

Random Forest Implementation:

```
Classifier is trained for all features and for reduced features, for later comparison.

The classifier model itself is stored in the clf variable.
```

In [27]:
```
# all features
clf_DoS=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_Probe=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_R2L=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_U2R=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_U2R.fit(X_U2R, Y_U2R.astype(int))
```

Out[27]:
```
RandomForestClassifier(n_estimators=10, n_jobs=2)
```

In [28]:
```
# selected features
clf_rfeDoS=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeProbe=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeR2L=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeU2R=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeDoS.fit(X_rfeDoS, Y_DoS.astype(int))
clf_rfeProbe.fit(X_rfeProbe, Y_Probe.astype(int))
clf_rfeR2L.fit(X_rfeR2L, Y_R2L.astype(int))
clf_rfeU2R.fit(X_rfeU2R, Y_U2R.astype(int))
```

Out[28]:
```
RandomForestClassifier(n_estimators=10, n_jobs=2)
```

Prediction:

In [61]:
```
# Apply the classifier we trained to the test data (which it has never seen before)
clf_DoS.predict(X_DoS_test)
```

Out[61]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

Evaluation of the Model using Cross validation:

```
Cross Validation : Accuracy, Precision, Recall, F-measure
```

```
DoS
```

In [56]:
```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99825 (+/- 0.00221)
Precision: 0.99933 (+/- 0.00180)
Recall: 0.99611 (+/- 0.00515)
F-measure: 0.99792 (+/- 0.00175)
```

```
Probe
```

In [36]:
```
accuracy = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99687 (+/- 0.00336)
Precision: 0.99633 (+/- 0.00519)
Recall: 0.99401 (+/- 0.00567)
F-measure: 0.99521 (+/- 0.00480)
```

```
U2R
```

In [37]:
```
accuracy = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99795 (+/- 0.00204)
Precision: 0.96287 (+/- 0.09230)
Recall: 0.83546 (+/- 0.20267)
```

R2L

In [62]:
```
accuracy = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.98031 (+/- 0.00610)
Precision: 0.97310 (+/- 0.00986)
Recall: 0.96863 (+/- 0.00880)
F-measure: 0.97298 (+/- 0.00717)
```

Evaluation of the Model using Cross validation including RFE:

DoS

In [44]:
```
accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99785 (+/- 0.00221)
Precision: 0.99799 (+/- 0.00343)
Recall: 0.99718 (+/- 0.00471)
F-measure: 0.99765 (+/- 0.00249)
```

Probe

In [45]:
```
accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99316 (+/- 0.00490)
Precision: 0.99137 (+/- 0.00900)
Recall: 0.98942 (+/- 0.00761)
F-measure: 0.98991 (+/- 0.00740)
```

R2L

In [46]:
```
accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.97690 (+/- 0.00753)
Precision: 0.97031 (+/- 0.01331)
Recall: 0.96276 (+/- 0.01468)
F-measure: 0.96738 (+/- 0.01301)
```

U2R

In [47]:
```
accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99724 (+/- 0.00330)
Precision: 0.94440 (+/- 0.11209)
Recall: 0.82097 (+/- 0.13210)
F-measure: 0.87602 (+/- 0.11516)
```

Confusion Matrix:

```
[ ] Y_DoS_pred=clf_DoS.predict(X_DoS_test)

    # Create confusion matrix
    pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	1
Actual attacks		
0	9661	50
1	4803	2657

Probe

```
Y_Probe_pred=clf_Probe.predict(X_Probe_test)
# Create confusion matrix

pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	2
Actual attacks		
0	9067	644
2	897	1524

R2L

```
[ ] Y_R2L_pred=clf_R2L.predict(X_R2L_test)
    # Create confusion matrix
    pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	3
Actual attacks		
0	9711	0
3	2854	31

U2R

```
[ ] Y_U2R_pred=clf_U2R.predict(X_U2R_test)
    # Create confusion matrix
    pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0
Actual attacks	
0	9711
4	67

SVM IMPLEMENTATION

In [58]:
```
from sklearn.svm import SVC

clf_SVM_DoS=SVC(kernel='linear', C=1.0, random_state=0)
clf_SVM_Probe=SVC(kernel='linear', C=1.0, random_state=0)
clf_SVM_R2L=SVC(kernel='linear', C=1.0, random_state=0)
clf_SVM_U2R=SVC(kernel='linear', C=1.0, random_state=0)

clf_SVM_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_SVM_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_SVM_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_SVM_U2R.fit(X_U2R, Y_U2R.astype(int))
```

Out[58]:
```
SVC(kernel='linear', random_state=0)
```

DoS

In [59]:
```
Y_DoS_pred=clf_SVM_DoS.predict(X_DoS_test)

# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Out[59]:

Actual attacks \ Predicted attacks	0	1
0	9455	256
1	1359	6101

In [60]:
```
Y_Probe_pred=clf_SVM_Probe.predict(X_Probe_test)
# Create confusion matrix

pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Out[60]:

Actual attacks \ Predicted attacks	0	2
0	9576	135
2	1285	1136

CROSS VALIDATION

Probe

In [64]:
```
accuracy = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.98450 (+/- 0.00526)
Precision: 0.96907 (+/- 0.01031)
Recall: 0.98365 (+/- 0.00686)
F-measure: 0.97613 (+/- 0.00800)
```

R2L

In [65]:
```
accuracy = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.96793 (+/- 0.00738)
Precision: 0.94854 (+/- 0.00994)
Recall: 0.96264 (+/- 0.01388)
F-measure: 0.95529 (+/- 0.01048)
```

U2R

In [66]:
```
accuracy = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99632 (+/- 0.00390)
Precision: 0.91056 (+/- 0.17934)
Recall: 0.82909 (+/- 0.21833)
F-measure: 0.84869 (+/- 0.16029)
```

ENSEMBLE LEARNING IMPLEMENTATION-

Ensemble learning is a general meta approach to machine learning that seeks better predictive performance by combining the predictions from multiple models. A voting ensemble (or a "*majority voting ensemble*") is an ensemble machine learning model that combines the predictions from multiple other models. It is a technique that may be used to improve model performance, ideally achieving better performance than any single model used in the ensemble.

A voting ensemble works by combining the predictions from multiple models. It can be used for classification or regression. In the case of regression, this involves calculating the average of the predictions from the models. In the case of classification, the predictions for each label are summed and the label with the majority vote is predicted.

Ensemble Learning

```
from sklearn.ensemble import VotingClassifier

clf_voting_DoS = VotingClassifier(estimators=[('rf', clf_DoS), ('knn', clf_KNN_DoS), ('svm', clf_SVM_DoS)], voting='hard')
clf_voting_Probe = VotingClassifier(estimators=[('rf', clf_Probe), ('knn', clf_KNN_Probe), ('svm', clf_SVM_Probe)], voting='hard')
clf_voting_R2L = VotingClassifier(estimators=[('rf', clf_R2L), ('knn', clf_KNN_R2L), ('svm', clf_SVM_R2L)], voting='hard')
clf_voting_U2R = VotingClassifier(estimators=[('rf', clf_U2R), ('knn', clf_KNN_U2R), ('svm', clf_SVM_U2R)], voting='hard')

clf_voting_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_voting_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_voting_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_voting_U2R.fit(X_U2R, Y_U2R.astype(int))
```

```
VotingClassifier(estimators=[('rf',
                              RandomForestClassifier(n_estimators=10,
                                                     n_jobs=2)),
                             ('knn', KNeighborsClassifier()),
                             ('svm', SVC(kernel='linear', random_state=0))])
```

DoS

```
[ ] Y_DoS_pred=clf_voting_DoS.predict(X_DoS_test)

    # Create confusion matrix
    pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Probe

```
[ ] Y_Probe_pred=clf_voting_Probe.predict(X_Probe_test)

    # Create confusion matrix
    pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	2
Actual attacks		
0	9475	236
2	1192	1229

R2L

```
Y_R2L_pred=clf_voting_R2L.predict(X_R2L_test)

# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	3
Actual attacks		
0	9711	0
3	2884	1

U2R

```
[ ] Y_U2R_pred=clf_voting_U2R.predict(X_U2R_test)

    # Create confusion matrix
    pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0
Actual attacks	
0	9711
4	67

DoS

```
[ ] from sklearn.model_selection import cross_val_score
    from sklearn import metrics
    accuracy = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
    print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
    precision = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
    print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
    recall = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
    print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
    f = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
    print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99802 (+/- 0.00234)
Precision: 0.99866 (+/- 0.00268)
Recall: 0.99705 (+/- 0.00289)
F-measure: 0.99785 (+/- 0.00231)
```

Probe

```
accuracy = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-mesaure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99242 (+/- 0.00374)
Precision: 0.98807 (+/- 0.00693)
Recall: 0.98916 (+/- 0.00729)
F-mesaure: 0.98867 (+/- 0.00564)
```

R2L

```
[ ] accuracy = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
    print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
    precision = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
    print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
    recall = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
    print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
    f = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
    print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.97182 (+/- 0.00640)
Precision: 0.95815 (+/- 0.00942)
Recall: 0.96416 (+/- 0.01088)
F-measure: 0.95960 (+/- 0.00843)
```

U2R

```
[ ] accuracy = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
    print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
    precision = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
    print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
    recall = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
    print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
    f = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
    print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99755 (+/- 0.00277)
Precision: 0.93545 (+/- 0.14873)
Recall: 0.87935 (+/- 0.15845)
F-measure: 0.90191 (+/- 0.11285)
```

KNN IMPLEMENTATION

KNeighbors

```
from sklearn.neighbors import KNeighborsClassifier

clf_KNN_DoS=KNeighborsClassifier()
clf_KNN_Probe=KNeighborsClassifier()
clf_KNN_R2L=KNeighborsClassifier()
clf_KNN_U2R=KNeighborsClassifier()

clf_KNN_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_KNN_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_KNN_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_KNN_U2R.fit(X_U2R, Y_U2R.astype(int))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

a) For DOS

DoS

```
[ ] Y_DoS_pred=clf_KNN_DoS.predict(X_DoS_test)

    # Create confusion matrix
    pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	1
Actual attacks		
0	9422	289
1	1573	5887

b) For Probe

Probe

```
Y_Probe_pred=clf_KNN_Probe.predict(X_Probe_test)
# Create confusion matrix

pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	2
Actual attacks		
0	9437	274
2	1272	1149

c) For R2L

R2L

```
[ ] Y_R2L_pred=clf_KNN_R2L.predict(X_R2L_test)
    # Create confusion matrix
    pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Actual attacks \ Predicted attacks	0	3
0	9706	5
3	2883	2

d) For U2R

U2R

```
[ ] Y_U2R_pred=clf_KNN_U2R.predict(X_U2R_test)
    # Create confusion matrix
    pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Actual attacks \ Predicted attacks	0	4
0	9711	0
4	65	2

e) Cross Validation

Cross Validation: Accuracy, Precision, Recall, F-measure

DoS

```
[ ] from sklearn.model_selection import cross_val_score
    from sklearn import metrics
    accuracy = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
    print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
    precision = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
    print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
    recall = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
    print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
    f = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
    print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99715 (+/- 0.00278)
Precision: 0.99678 (+/- 0.00383)
Recall: 0.99665 (+/- 0.00344)
F-measure: 0.99672 (+/- 0.00320)
```

Probe

```
[ ]  accuracy = cross_val_score(clf_KNN_Probe, x_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
     print("5ccuracy: %B.5f (+/- %B.Ef)" % (accuracy.meaM(), accuracy.std() * 2))
     precision = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro ')
     print("Precision: ¥0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
     recall = cross_val_scone(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='necall_macno')
     print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
     f = cross_val_score(clf_KNN_Pnobe, X_Probe_test, Y_Probe_test, cv=10, scoring='f*_macno')
     print("F-measure: ¥0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

     Accuracy: 0.99077 (+/- 0.00403)
     Precision: 0.98606 (+/- 0.00675)
     Recall: 0.98508 (a/- 0.01137)
     F-measure: 0.98553 (+/- 0.00645)
```

R2L

```
[ ]  accuracy = c ros s_va1_score(c 1f_KNN_R2L, X_R2L_t esl , Y_R2 L_tesl , cv=10, scon ng= ' a ccura‹§' )
     print("Accuracy: w0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
     precision = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring=' precision_macro')
     print("Pnecision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
     recall = cross_val_scone(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring=' necall_macno')
     print("Recall: %0.5f (+/- %8.5f)" % (recall.mean(), necall.std() * 2))
     f = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring= 'f1_macno')
     print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

     Accuracy: B . 96737 (+/ - 0. 66736)
     Prec1s ton : 0. 953J1 (+/– B.61274)
     Recall : 6 . 95484 (+/ – B. 01326)
     F -me as ure: 0. 95389 (+/– B.
```

U2R

```
     accuracy = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=IB, scoring='accuracy')
     print("Accuracy! ^0.Sf (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2})
     precision = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision macro')
     print("Precision: %0.5f (*/- %0.5f)" % (precision.mean(), precision.std() * 2))
     recall = cross_val_scone(clf_KNE_U2R, X_U2R_test, Y_U2R_test, cv=10, sconing='necall macno')
     print("Recall: %0.5f (a/- %8.5f)" % (recall.mean(), necall.std() * 2))
     f = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_u2R_test, cv=10, scoring='fl_macro')
     print("F-measure: %0.5f (*/- %0.5f)" % (f.mean(), f.std() * 2))

     Accuracy! 0.99703 (+/- 0.0028t)
     Precision: 0.93143 (+/- 0.14679)
     Recall: 0.85073 (+/- 0.17639)
     F-measure: 0.87831 (+/- 0.11390)
```

summary report of machine learning approach

The aim of this project was to demonstrate the benefits of employing machine learning algorithms for the development of an Intrusion Detection System(IDS). The limitations, as found in the previous research conducted for the same, were eliminated to obtain better results with an efficient system.The result was a system with an accuracy of above 95 in predicting Dos,Probe ,U2R and R2L attacks. The dataset used was NSL KDD dataset whichis a new and improved dataset which included a great number of U2R and R2L attacks as compared to the old KDD dataset used in many research. The machine learning algorithm employed is also lightweight compared to many heavy computational cost algorithms . The data taken from NSL KDD dataset was preprocessed to remove redundancy and unnecessary features to provide a high accuracy model with low computational cost.

Pre-processing of data also provided better results in identifying U2R and R2L attacks which were lacking in the previous research.

Accuracy score :

Algorithms	Dos	Probe	R2L	U2R
Random forest	99.7	99.3	97.6	99.7
SVM	99.3	98.4	96.7	99.6
KNN	99.7	99	96.7	99.7
Ensemble learning	99.8	99.6	99.7	97.9

Ensemble learning gives best result when compared to all others. An ensemble reduces the spread or dispersion of the predictions and model performance.

Code Link:
https://colab.research.google.com/drive/1awrjeaA67uChkQ_2MwQ8Nz6iwaz57qE7

Neural Network

DATA PREPROCESSING:

Total:6.46 GB
Dataset used :1.00GB(approx.)

Experimental Results:

Binary:

Open in Colab

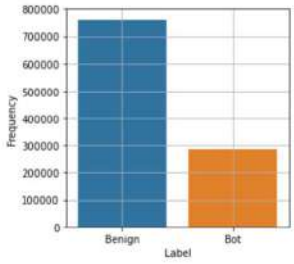
```
In [3]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

In [4]: # Loading the Cic Ids dataset dated 2nd March, 2018.
        data = pd.read_csv('03-02-2018.csv')

In [5]: data.head()
```

Out[5]:

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
0	443	6	02/03/2018 08:47:38	141385	9	7	553	3773.0	202	0	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Benign
1	49684	6	02/03/2018 08:47:38	281	2	1	38	0.0	38	0	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Benign
2	443	6	02/03/2018 08:47:40	279824	11	15	1086	10527.0	385	0	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Benign
3	443	6	02/03/2018 08:47:40	132	2	0	0	0.0	0	0	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Benign
4	443	6	02/03/2018 08:47:41	274016	9	13	1285	6141.0	517	0	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Benign

5 rows × 80 columns

In [9]:
```
plt.figure(figsize=(4,4))
sns.barplot(x=data['Label'].value_counts().index, y=data['Label'].value_counts())
plt.xlabel('Label')
plt.ylabel('Frequency')
plt.grid(True)
```



In [1]:
```
## Building the neural network
from keras.models import Sequential
from keras.layers import Dense, Dropout, Input

model = Sequential() # initializing model
model.add(Input(shape=(76,)))
model.add(Dense(64,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(units=2,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy' ,metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                4928

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 128)               8320

 dense_3 (Dense)             (None, 128)               16512

 dense_4 (Dense)             (None, 2)                 258

=================================================================
Total params: 34,178
Trainable params: 34,178
Non-trainable params: 0
_________________________________________________________________
```

In [29]:
```
## Training the dataset
history = model.fit(x_train_normal,y_train_cat  ,epochs=2)
```

```
Epoch 1/2
24576/24576 [==============================] - ETA: 0s - loss: 0.0035 - accuracy: 0.99 - 96s 4ms/step - loss: 0.0035 - accurac
y: 0.9991
Epoch 2/2
24576/24576 [==============================] - 95s 4ms/step - loss: 0.0013 - accuracy: 0.9997
```

In [30]:
```
## Testing the model
res = model.evaluate(x_test_normal, y_test_cat, verbose=1)
res
```

```
8192/8192 [==============================] - 22s 3ms/step - loss: 0.0015 - accuracy: 0.9996
```

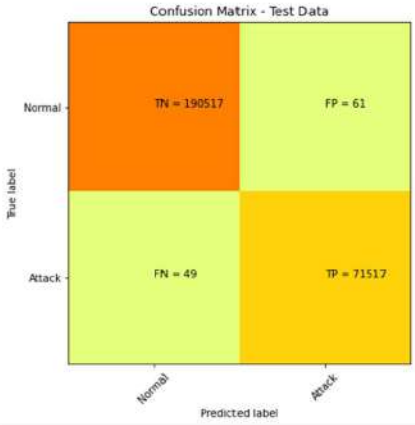
Out[30]:
```
[0.0015373515198007226, 0.9995803833007812]
```

In [31]:
```
## Comparison of true labels and predicted labels using confusion matrix
from sklearn.metrics import confusion_matrix,accuracy_score,plot_confusion_matrix

prediction =np.argmax(model.predict(x_test_normal), axis=-1)
true_labels = np.argmax(y_test_cat, axis=1)
matrix = confusion_matrix(true_labels, prediction)
matrix
```

Out[31]:
```
array([[190517,     61],
       [    49,  71517]], dtype=int64)
```

In [32]:
```
plt.clf()
plt.figure(figsize=(6, 6))
plt.imshow(matrix, cmap=plt.cm.Wistia)
classNames = ['Normal','Attack']
plt.title('Confusion Matrix - Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]

for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(matrix[i][j]))
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



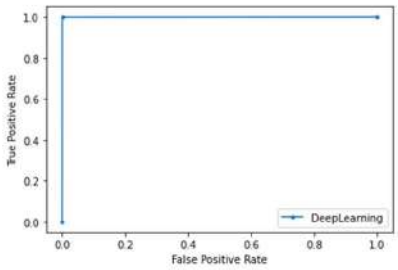
In [33]:
```
## Calculating AUC ROC score
from sklearn.metrics import roc_auc_score,roc_curve
acc = roc_auc_score(true_labels, prediction)
```

In [34]:
```
acc
```

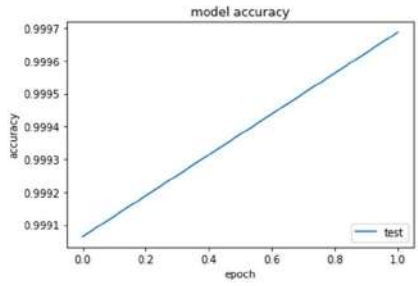
Out[34]:
```
0.9994976192058198
```

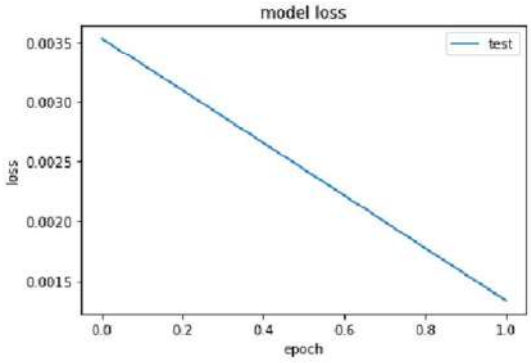
In [35]:
```
fpr, tpr, _ = roc_curve(true_labels, prediction)
```

In [36]:
```
## Plotting the Receiver operating characteristics curve
plt.plot(fpr, tpr, marker='.', label='DeepLearning')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the Legend
plt.legend()
# show the plot
plt.show()
```



In [37]:
```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['test'], loc='lower right')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['test'], loc='upper right')
plt.show()
```





MultiClass:

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

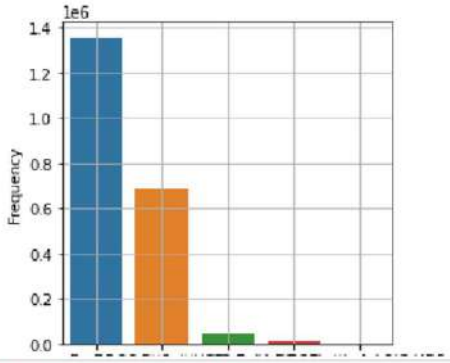
In [3]:
```
# Loading the Cic Ids dataset dated 2nd March, 2018
class dataset:
    pass
df1 = pd.read_csv("02-15-2018.csv")
df2 = pd.read_csv("02-21-2018.csv")
```

In [4]:
```
merge = [
    df1,
    df2,
]
```

In [5]:
```
class dataset:
    pass
df = pd.concat(merge)
del merge
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2097150 entries, 0 to 1048574
Data columns (total 80 columns):
 #   Column             Dtype
---  ------             -----
 0   Dst Port           int64
 1   Protocol           int64
 2   Timestamp          object
 3   Flow Duration      int64
 4   Tot Fwd Pkts       int64
 5   Tot Bwd Pkts       int64
 6   TotLen Fwd Pkts    int64
 7   TotLen Bwd Pkts    int64
 8   Fwd Pkt Len Max    int64
 9   Fwd Pkt Len Min    int64
```

In [10]:
```
plt.figure(figsize=(4,4))
sns.barplot(x=df['Label'].value_counts().index, y=df['Label'].value_counts())
plt.xlabel('Label')
plt.ylabel('Frequency')
plt.grid(True)
```



In [29]:
```
## Building the neural network
from keras.models import Sequential
from keras.layers import Dense, Dropout, Input

model = Sequential() # initializing model
model.add(Input(shape=(14,)))
model.add(Dense(64,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(units=5,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy' ,metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                960

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 128)               8320

 dense_3 (Dense)             (None, 128)               16512

 dense_4 (Dense)             (None, 5)                 645

=================================================================
Total params: 30,597
Trainable params: 30,597
Non-trainable params: 0
_________________________________________________________________
```

In [30]:
```
## Training the dataset
history = model.fit(x_train_normal,y_train_cat  ,epochs=2)
```

```
Epoch 1/2
49152/49152 [==============================] - 196s 4ms/step - loss: 0.1521 - accuracy: 0.9522
Epoch 2/2
49152/49152 [==============================] - 182s 4ms/step - loss: 0.1101 - accuracy: 0.9675
```

In [31]:
```
## Testing the model
res = model.evaluate(x_test_normal, y_test_cat, verbose=1)
res
```

```
16384/16384 [==============================] - 50s 3ms/step - loss: 0.1053 - accuracy: 0.9695
```

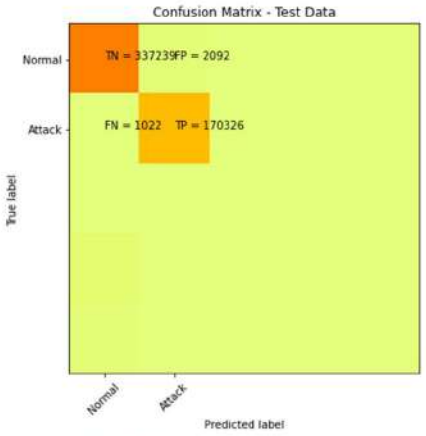
Out[31]:
```
[0.10531852394342422, 0.969451904296875]
```

In [32]:
```
## Comparison of true labels and predicted labels using confusion matrix
from sklearn.metrics import confusion_matrix,accuracy_score,plot_confusion_matrix

prediction =np.argmax(model.predict(x_test_normal), axis=-1)
true_labels = np.argmax(y_test_cat, axis=1)
matrix = confusion_matrix(true_labels, prediction)
matrix
```

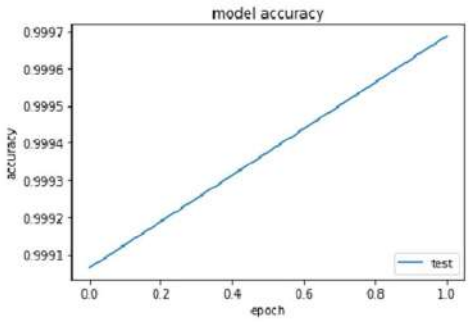
Out[32]:
```
array([[337239,   2092,     11,     60,      0],
       [  1022, 170326,      0,      0,      0],
       [     2,      0,    436,      0,      0],
       [ 10316,     27,      0,     10,      0],
       [  2479,      7,      0,      0,    261]], dtype=int64)
```

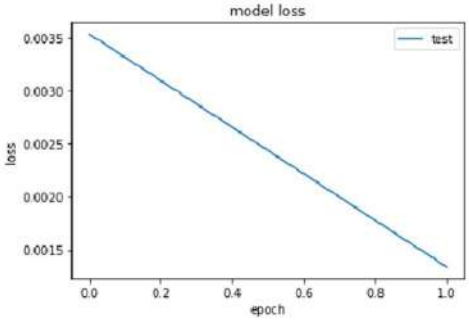
In [33]:
```
plt.clf()
plt.figure(figsize=(6, 6))
plt.imshow(matrix, cmap=plt.cm.Wistia)
classNames = ['Normal','Attack']
plt.title('Confusion Matrix - Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]

for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(matrix[i][j]))
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



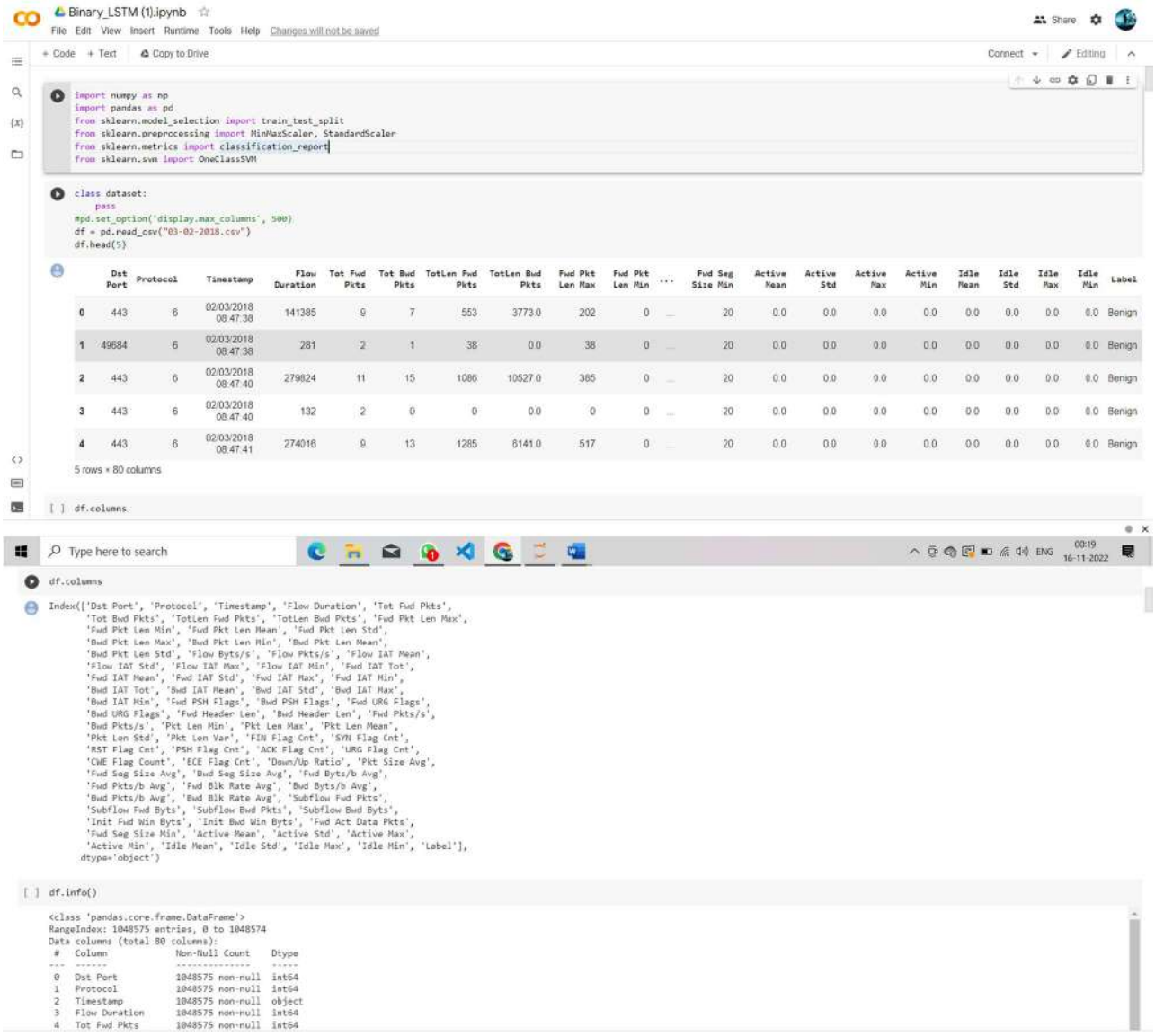

```
plt.legend(['test'], loc='lower right')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['test'], loc='upper right')
plt.show()
```





LSTM

Binary Class



```
[ ] df.shape

(1048575, 80)
```

```
df.describe()
```

	Dst Port	Protocol	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	...	Fwd Act Data Pkts	Fwd Seg Size Min	Active Mean	Active Std	Act
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	...	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048
mean	8.423429e+03	8.049848e+00	1.151421e+07	5.887282e+00	6.983180e+00	3.920838e+02	5.271275e+03	2.104312e+02	8.249200e+00	5.461640e+01	...	2.044231e+00	1.745403e+01	1.229579e+05	6.130972e+04	1.905
std	1.516176e+04	4.432629e+00	3.012113e+07	8.903317e+01	2.129580e+02	2.150779e+03	3.088768e+05	2.669497e+02	2.094972e+01	5.762840e+01	...	1.392463e+01	5.179526e+00	2.083099e+06	1.261694e+06	2.761
min	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000
25%	8.000000e+01	6.000000e+00	5.160000e+02	2.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...	0.000000e+00	2.000000e+01	0.000000e+00	0.000000e+00	0.000
50%	3.389000e+03	6.000000e+00	1.124200e+04	3.000000e+00	1.000000e+00	6.300000e+01	1.290000e+02	4.800000e+01	0.000000e+00	4.000000e+01	...	1.000000e+00	2.000000e+01	0.000000e+00	0.000000e+00	0.000
75%	8.080000e+03	6.000000e+00	2.210150e+06	7.000000e+00	5.000000e+00	3.650000e+02	5.820000e+02	3.260000e+02	0.000000e+00	1.086667e+02	...	3.000000e+00	2.000000e+01	0.000000e+00	0.000000e+00	0.000
max	6.553400e+04	1.700000e+01	1.200000e+08	4.315900e+04	6.924100e+04	1.100627e+06	1.010000e+08	1.711000e+03	1.460000e+03	1.460000e+03	...	9.262000e+03	4.400000e+01	1.110000e+08	7.490000e+07	1.110

8 rows × 78 columns

```
[ ] df.dtypes

Dst Port           int64
Protocol           int64
Timestamp         object
Flow Duration      int64
Tot Fwd Pkts       int64
                   ...
Idle Mean        float64
```


```
[ ] df['Label'].unique()

array(['Benign', 'Bot'], dtype=object)
```

```
df.isnull()
```

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1048570	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1048571	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1048572	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1048573	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1048574	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

1048575 rows × 80 columns

```
[ ] pd.Categorical(df['Flow Byts/s'])

C:\Users\potta\anaconda3\lib\site-packages\pandas\io\formats\format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
C:\Users\potta\anaconda3\lib\site-packages\pandas\io\formats\format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
```


```
[ ] pd.Categorical(df['Flow Byts/s'])

C:\Users\potta\anaconda3\lib\site-packages\pandas\io\formats\format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
C:\Users\potta\anaconda3\lib\site-packages\pandas\io\formats\format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
[30597.30523, 135231.31670, 41501.08640, 0.00000, 27100.60726, ..., 796.799143, 833.965404, 796.655744, 797.812144, 797.955823]
Length: 1048575
Categories (458104, float64): [0.000000, 0.008334, 0.008362, 0.008385, ..., 357000000.0, 394000000.0, 480000000.0, inf]

[ ] pd.Categorical(df['Flow Pkts/s'])# to check different value

C:\Users\potta\anaconda3\lib\site-packages\pandas\io\formats\format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
C:\Users\potta\anaconda3\lib\site-packages\pandas\io\formats\format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
[113.166178, 10676.156580, 92.915547, 15151.515150, 80.287282, ..., 5.524608, 5.785947, 5.494178, 5.502153, 5.537864]
Length: 1048575
Categories (455755, float64): [0.016669, 0.016671, 0.016671, 0.016671, ..., 1500000.0, 2000000.0, 3000000.0, inf]

[ ] df['Flow Pkts/s'].unique()

array([1.13166178e+02, 1.06761566e+04, 9.29155469e+01, ...,
       5.49417755e+00, 5.50215272e+00, 5.53786376e+00])

df.drop('Timestamp',axis=1,inplace=True)

[ ] df.shape

(1048575, 79)

[ ] df.drop(['Bwd PSH Flags','Fwd URG Flags','Bwd URG Flags','CWE Flag Count','Fwd Byts/b Avg','Fwd Pkts/b Avg','Fwd Blk Rate Avg','Bwd Byts/b Avg','Bwd Pkts/b Avg','Bwd Blk Rate Avg'],axis=1,inplace=True)
```

```
dataset.train = df.groupby('Label').apply(pd.DataFrame.sample, frac=0.8).reset_index(level='Label', drop=True)
dataset.test = df.drop(dataset.train.index)
dataset.label = dataset.train.Label.copy()
```

```
dataset.label.unique()
```

```
array(['Benign', 'Bot'], dtype=object)
```

```
d1 = dataset.train.replace('Benign', 0) # here converting the categorical values to numeric
d2 = d1.replace('Bot', 1)
d6_label = d2.Label.copy()
d6_label.unique()
d6_label.value_counts()
```

```
0    609907
1    228953
Name: Label, dtype: int64
```

```
d6_label.unique()
```

```
array([0, 1], dtype=int64)
```

```
dataset.test_label = dataset.test.Label.copy()
dataset.test_label.unique()
a1_label = dataset.test.Label.copy()
a1_label.unique()
```

```
array(['Benign', 'Bot'], dtype=object)
```

```
a1 = dataset.test.replace('Benign', 0)
a2 = a1.replace('Bot', 1)
a5_label = a2.Label.copy()
a5_label.unique()
```

Type here to search

ENG 00:22 16-11-2022

```
category_variables = ["Protocol"]
for cv in category_variables:
    d2[cv] = d2[cv].astype("category")
    a2[cv] = a2[cv].astype("category")

    print("Length of Categories for {} are {}".format(cv , len(d2[cv].cat.categories)))
    print("Categories for {} are {} \n".format(cv ,d2[cv].cat.categories))
```

```
Length of Categories for Protocol are 3
Categories for Protocol are Int64Index([0, 6, 17], dtype='int64')
```

```
dummy_variables_2labels = category_variables

class preprocessing:
    train_labels = pd.get_dummies(d2, columns = dummy_variables_2labels, prefix=dummy_variables_2labels)
    test_labels = pd.get_dummies(a2, columns = dummy_variables_2labels, prefix=dummy_variables_2labels)
```

```
d2.head()
```

	Dst Port	Protocol	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	...	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
418995	443	6	769419	8	9	355	4857.0	198	0	44.375000	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
239227	53	17	887	1	1	30	62.0	30	30	30.000000	...	8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
540644	53	17	1093	1	1	36	223.0	36	36	36.000000	...	8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
536874	80	6	1009	3	4	148	243.0	148	0	49.333333	...	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
734251	53	17	107464	2	2	78	110.0	39	39	39.000000	...	8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

5 rows × 69 columns

Type here to search

ENG 00:22 16-11-2022

```
[ ] preprocessing.test_labels.head()
```

	Dst Port	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	Fwd Pkt Len Std	...	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label	Protocol_0	Protocol_6	Protocol_17
10	445	234228	3	1	0	0.0	0	0	0.000000	0.000000	...	0.0	0.0	0.0	0.00000	0.0	0.0	0	0	1	0
12	49687	68899	2	0	0	0.0	0	0	0.000000	0.000000	...	0.0	0.0	0.0	0.00000	0.0	0.0	0	0	1	0
15	443	60602600	13	14	338	3264.0	207	0	26.000000	64.512273	...	295376.0	68039.0	9994401.5	23886.95478	10000000.0	9959152.0	0	0	1	0
19	443	192	3	0	77	0.0	46	0	25.666667	23.459184	...	0.0	0.0	0.0	0.00000	0.0	0.0	0	0	1	0
27	49728	247	2	1	38	0.0	38	0	19.000000	26.870058	...	0.0	0.0	0.0	0.00000	0.0	0.0	0	0	1	0

5 rows × 71 columns

```
preprocessing.train_labels.to_csv("preprocessed_train_4_new.csv")
preprocessing.test_labels.to_csv("preprocessed_test_4_new.csv")
```

+ Code + Text

```
[ ] preprocessing.train_labels.head()
```

	Dst Port	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	Fwd Pkt Len Std	...	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label	Protocol_0	Protocol_6	Protocol_17
418995	443	769419	6	9	355	4857.0	198	0	44.375000	75.864423	...	0.0	0.0	0.0	0.0	0.0	0.0	0	0	1	0
239227	53	887	1	1	30	62.0	30	30	30.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0	0	0	1
540644	53	1093	1	1	36	223.0	36	36	36.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0	0	0	1
536874	80	1009	3	4	148	243.0	148	0	49.333333	85.447840	...	0.0	0.0	0.0	0.0	0.0	0.0	0	0	1	0
734251	53	107464	2	2	78	110.0	39	39	39.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0	0	0	1

```
+ Code  + Text  Copy to Drive                                   Connect  Editing
```

```
[ ] model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

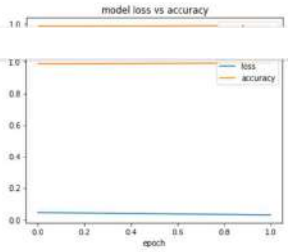
```
[ ] #we will fit the model with data that we loaded from the Keras.
    csv_logger = CSVLogger('Binlstm80.csv',separator=',', append=False)
    history= model.fit(X_train, y_train1, batch_size=batch_size, epochs=2, validation_data=(X_test, y_test1),callbacks=[csv_logger])
```

```
Epoch 1/2
16778/16778 [==============================] - 401s 23ms/step - loss: 0.0430 - accuracy: 0.9877 - val_loss: 0.0307 - val_accuracy: 0.9884
Epoch 2/2
16778/16778 [==============================] - 426s 25ms/step - loss: 0.0281 - accuracy: 0.9915 - val_loss: 0.0248 - val_accuracy: 0.9915
```

```
loss, accuracy = model.evaluate(X_test, y_test1)
print("\nLoss: %.2f, Accuracy: %.2f%%" % (loss, accuracy*100))
y_pred = model.predict(X_test)
#print(y_pred)
np.savetxt('Binlst80predicted.txt', y_pred, fmt='%01d')
```

```
6554/6554 [==============================] - 46s 7ms/step - loss: 0.0248 - accuracy: 0.9915

Loss: 0.02, Accuracy: 99.15%
```

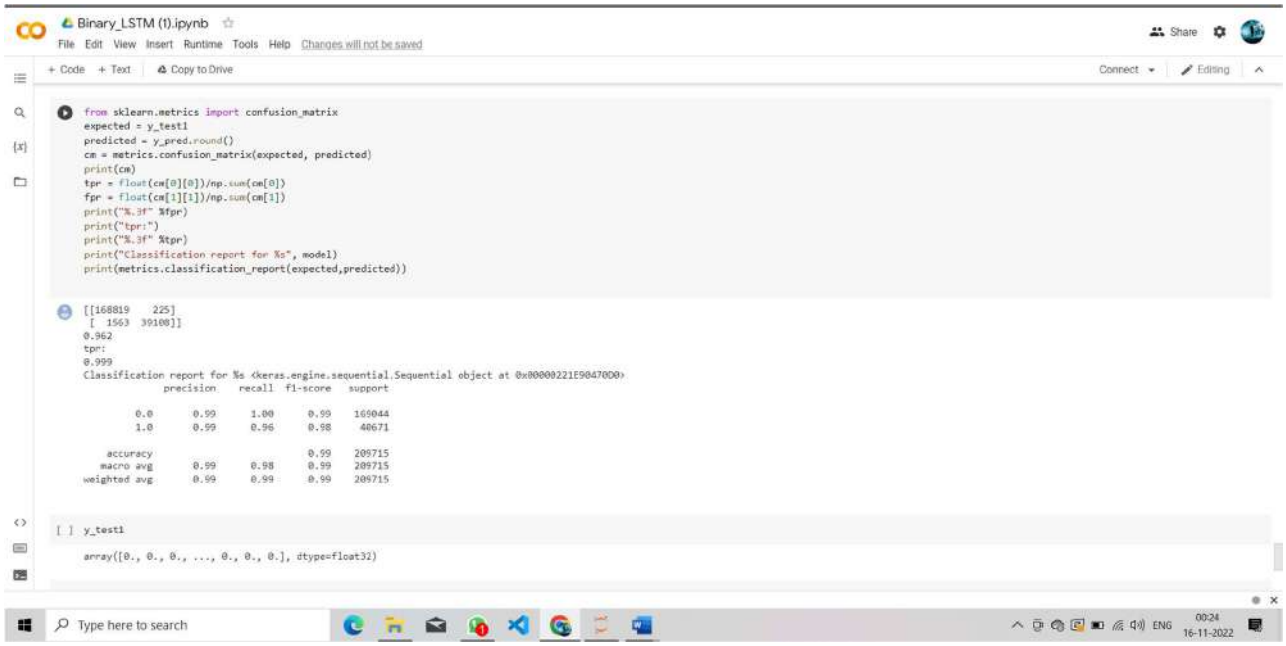
```
[ ] #we can see that the accuracy line is all time near to the one, and the loss is almost zero.
    #Thus, the model has performed well in training.
    from matplotlib import pyplot
    pyplot.plot(history.history['loss'])
    pyplot.plot(history.history['accuracy'])
    pyplot.title('model loss vs accuracy')
    pyplot.xlabel('epoch')
    pyplot.legend(['loss', 'accuracy'], loc='upper right')
    pyplot.show()
```



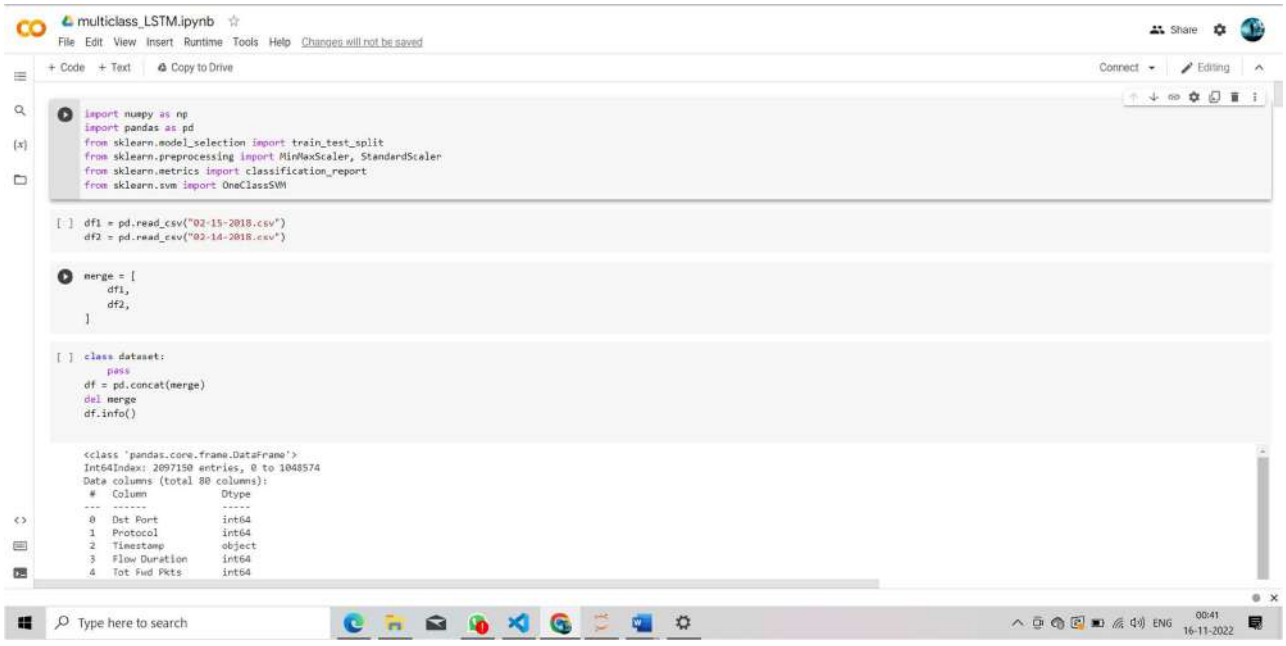

```
[ ]
```

```
from sklearn.metrics import confusion_matrix
expected = y_test1
predicted = y_pred.round()
cm = metrics.confusion_matrix(expected, predicted)
print(cm)
tpr = float(cm[0][0])/np.sum(cm[0])
fpr = float(cm[1][1])/np.sum(cm[1])
print("%.3f" %fpr)
print("tpr:")
print("%.3f" %tpr)
print("Classification report for %s", model)
print(metrics.classification_report(expected,predicted))
```

```
[[168819    225]
 [  1563  39108]]
0.962
tpr:
0.999
Classification report for %s <keras.engine.sequential.Sequential object at 0x00000221E90470D0>
              precision    recall  f1-score   support
```


Binary_LSTM (1).ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
Share
+ Code + Text Copy to Drive
Connect Editing
from sklearn.metrics import confusion_matrix
expected = y_test1
predicted = y_pred.round()
cm = metrics.confusion_matrix(expected, predicted)
print(cm)
tpr = float(cm[0][0])/np.sum(cm[0])
fpr = float(cm[1][1])/np.sum(cm[1])
print("%.3f" %fpr)
print("tpr:")
print("%.3f" %tpr)
print("Classification report for %s", model)
print(metrics.classification_report(expected,predicted))
[[168819 225]
[1563 39108]]
0.962
tpr:
0.999
Classification report for %s <keras.engine.sequential.Sequential object at 0x00000221E9047DD0>
precision recall f1-score support
0.0 0.99 1.00 0.99 169044
1.0 0.99 0.96 0.98 40671
accuracy 0.99 209715
macro avg 0.99 0.98 0.99 209715
weighted avg 0.99 0.99 0.99 209715
[ ] y_test1
array([0., 0., 0., ..., 0., 0., 0.], dtype=float32)
Type here to search
ENG
00:24
16-11-2022

Multiclass


multiclass_LSTM.ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
Share
+ Code + Text Copy to Drive
Connect Editing
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import classification_report
from sklearn.svm import OneClassSVM
[ ] df1 = pd.read_csv("02-15-2018.csv")
df2 = pd.read_csv("02-14-2018.csv")
merge = [
df1,
df2,
]
[ ] class dataset:
pass
df = pd.concat(merge)
del merge
df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2097150 entries, 0 to 1048574
Data columns (total 80 columns):
Column Dtype
--- ------ -----
0 Dst Port int64
1 Protocol int64
2 Timestamp object
3 Flow Duration int64
4 Tot Fwd Pkts int64
Type here to search
ENG
00:41
16-11-2022

```
[ ] df.head(5)
```

```
df.columns
```

```
Index(['Dst Port', 'Protocol', 'Timestamp', 'Flow Duration', 'Tot Fwd Pkts',
       'Tot Bwd Pkts', 'TotLen Fwd Pkts', 'TotLen Bwd Pkts', 'Fwd Pkt Len Max',
       'Fwd Pkt Len Min', 'Fwd Pkt Len Mean', 'Fwd Pkt Len Std',
       'Bwd Pkt Len Max', 'Bwd Pkt Len Min', 'Bwd Pkt Len Mean',
       'Bwd Pkt Len Std', 'Flow Byts/s', 'Flow Pkts/s', 'Flow IAT Mean',
       'Flow IAT Std', 'Flow IAT Max', 'Flow IAT Min', 'Fwd IAT Tot',
       'Fwd IAT Mean', 'Fwd IAT Std', 'Fwd IAT Max', 'Fwd IAT Min',
       'Bwd IAT Tot', 'Bwd IAT Mean', 'Bwd IAT Std', 'Bwd IAT Max',
       'Bwd IAT Min', 'Fwd PSH Flags', 'Bwd PSH Flags', 'Fwd URG Flags',
       'Bwd URG Flags', 'Fwd Header Len', 'Bwd Header Len', 'Fwd Pkts/s',
       'Bwd Pkts/s', 'Pkt Len Min', 'Pkt Len Max', 'Pkt Len Mean',
       'Pkt Len Std', 'Pkt Len Var', 'FIN Flag Cnt', 'SYN Flag Cnt',
       'RST Flag Cnt', 'PSH Flag Cnt', 'ACK Flag Cnt', 'URG Flag Cnt',
       'CWE Flag Count', 'ECE Flag Cnt', 'Down/Up Ratio', 'Pkt Size Avg',
       'Fwd Seg Size Avg', 'Bwd Seg Size Avg', 'Fwd Byts/b Avg',
       'Fwd Pkts/b Avg', 'Fwd Blk Rate Avg', 'Bwd Byts/b Avg',
       'Bwd Pkts/b Avg', 'Bwd Blk Rate Avg', 'Subflow Fwd Pkts',
       'Subflow Fwd Byts', 'Subflow Bwd Pkts', 'Subflow Bwd Byts',
       'Init Fwd Win Byts', 'Init Bwd Win Byts', 'Fwd Act Data Pkts',
       'Fwd Seg Size Min', 'Active Mean', 'Active Std', 'Active Max',
       'Active Min', 'Idle Mean', 'Idle Std', 'Idle Max', 'Idle Min', 'Label'],
      dtype='object')
```

```
[ ] df.shape
```

```
(2097150, 80)
```

```
[ ] df.describe()
```

```
[ ] df.dtypes
```

```
Dst Port          int64
Protocol          int64
Timestamp        object
Flow Duration     int64
Tot Fwd Pkts      int64
                  ...
Idle Mean       float64
Idle Std        float64
Idle Max          int64
Idle Min          int64
Label            object
Length: 80, dtype: object
```

```
[ ] df['Label'].unique()
```

```
array(['Benign', 'DoS attacks-GoldenEye', 'DoS attacks-Slowloris',
       'FTP-BruteForce', 'SSH-Bruteforce'], dtype=object)
```

```
df.isnull()
```

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

```
[ ] pd.Categorical(df['Flow Byts/s'])
```

```
C:\Users\potta\anaconda3\lib\site-packages\pandas\io\formats\format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
C:\Users\potta\anaconda3\lib\site-packages\pandas\io\formats\format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
C:\Users\potta\anaconda3\lib\site-packages\pandas\io\formats\format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
[0.000000, 138.117400, 117863.720074, 0.000000, 0.000000, ..., 296.544664, 0.000000, 0.000000, 0.000000, 81.277629]
Length: 2097150
Categories (948952, float64): [0.000000, 0.008347, 0.008372, 0.008378, ..., 7.880000e+08, 9.120000e+08, 1.298500e+09, inf]
```

```
[ ] pd.Categorical(df['Flow Pkts/s'])# to check different value
```

```
C:\Users\potta\anaconda3\lib\site-packages\pandas\io\formats\format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
[0.026633, 0.695806, 3683.241252, 0.026633, 0.026633, ..., 0.984544, 17094.017094, 0.785032, 0.764013, 1.721984]
Length: 2097150
Categories (873776, float64): [-0.001030, -0.000953, -0.000168, -0.000088, ..., 2500000.0, 3000000.0, 4000000.0, inf]
```

```
df['Flow Pkts/s'].unique()
```

```
array([2.66332489e-02, 6.95805540e-01, 3.68324125e+03, ...,
       7.85032415e-01, 7.64013293e-01, 1.72198367e+00])
```

+ Code + Text

```
[ ] df.drop('Timestamp',axis=1,inplace=True)
```

```
[ ] df.shape
```

```
(2097150, 79)
```

```
[ ] df.drop(['Bwd PSH Flags','Bwd URG Flags','Fwd Byts/b Avg','Fwd Pkts/b Avg','Fwd Blk Rate Avg','Bwd Byts/b Avg','Bwd Pkts/b Avg','Bwd Blk Rate Avg'],axis=1,inplace=True)
```

```
[ ] history= model.fit(X_train, y_train, batch_size=batch_size, epochs=2, validation_data=(X_test, y_test),callbacks=[csv_logger])

    Epoch 1/2
    33555/33555 [==============================] - 374s 11ms/step - loss: 0.1942 - accuracy: 0.7363 - val_loss: 0.1938 - val_accuracy: 0.7358
    Epoch 2/2
    33555/33555 [==============================] - 198s 6ms/step - loss: 0.1920 - accuracy: 0.7364 - val_loss: 0.1874 - val_accuracy: 0.7358
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print("\nLoss: %.2f, Accuracy: %.2f%%" % (loss, accuracy*100))
#y_pred = model.predict_classes(X_test)
#print(y_pred)
#np.savetxt('multiclass_Genlstm40predicted.txt',y_pred, fmt='%01d')
y_pred=model.predict(X_test)
classes_x=np.argmax(y_pred,axis=1)
```

```
2634/2634 [==============================] - 9s 3ms/step - loss: 0.1874 - accuracy: 0.7358

Loss: 0.19, Accuracy: 73.58%
```

```
[ ]
from sklearn.metrics import confusion_matrix
expected = y_test1
predicted = y_pred[:,0].round()
cm = metrics.confusion_matrix(expected, predicted)
print(cm)
tpr = float(cm[0][0])/np.sum(cm[0])
fpr = float(cm[1][1])/np.sum(cm[1])
print("%.3f" %fpr)
print("tpr:")
print("%.3f" %tpr)
print("Classification report for %s", model)
print(metrics.classification_report(expected,predicted))
```

```
[ ] [[    0 62000]
     [    0 22258]]
    1.000
    tpr:
    0.000
    Classification report for %s <keras.engine.sequential.Sequential object at 0x000002571E315FD0>
                  precision    recall  f1-score   support

             0.0       0.00      0.00      0.00     62000
             1.0       0.26      1.00      0.42     22258

        accuracy                           0.26     84258
       macro avg       0.13      0.50      0.21     84258
    weighted avg       0.07      0.26      0.11     84258

    C:\Users\potta\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Us
      _warn_prf(average, modifier, msg_start, len(result))
    C:\Users\potta\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Us
      _warn_prf(average, modifier, msg_start, len(result))
    C:\Users\potta\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Us
      _warn_prf(average, modifier, msg_start, len(result))
```

```
[ ] y_test1

    array([0., 0., 0., ..., 1., 0., 0.], dtype=float32)
```

```
[ ] y_pred

    array([[0.7368316 , 0.26316845],
           [0.7368372 , 0.26316276],
           [0.73683715, 0.26316282],
           ...,
           [0.7102651 , 0.2897349 ],
           [0.74107826, 0.25892174],
           [0.73578304, 0.26421693]], dtype=float32)
```


```
[ ] y_test1

    array([0., 0., 0., ..., 1., 0., 0.], dtype=float32)
```

```
[ ] y_pred

    array([[0.7368316 , 0.26316845],
           [0.7368372 , 0.26316276],
           [0.73683715, 0.26316282],
           ...,
           [0.7102651 , 0.2897349 ],
           [0.74107826, 0.25892174],
           [0.73578304, 0.26421693]], dtype=float32)
```


```
[ ] y_pred.round()

    array([[1., 0.],
           [1., 0.],
           [1., 0.],
           ...,
           [1., 0.],
           [1., 0.],
           [1., 0.]], dtype=float32)
```

```
[ ]
```

Comparative Study:

Binary:

	Accuracy
Neural Network	0.9996
LSTM	0.9958

Multiclass

	Accuracy
Neural Network	0.9695
LSTM	0.9589

CONCLUSION

This project has come up with recommendations that will enhance network security while providing for scalability and effectiveness, and ease of management of the system to the overall network security. The implementation design provides a framework for the modelling of effective intrusion detection systems. Integration of intrusion detection systems with a line of intrusion prevention mechanisms will greatly improve on the system performance. Moreover it also ensures high system scalability since devices can easily be added or removed from the system without affecting its overall performance. Configurations not being tied onto a single devise reduces the chances of having a single point of failure. The system is also effective since it looks at both external attacks and internal attacks that make up one of the most dangerous threats to network security.

REFERENCES:

[1] Dawoud, A., Shahristani, S., & Raun, C. (2018, December). Deep learning for network anomalies detection. In *2018 International Conference on Machine Learning and Data Engineering (iCMLDE)* (pp. 149-153). IEEE.

[2] Ariafar, E., & Kiani, R. (2017, December). Intrusion detection system using an optimized framework based on data mining techniques. In *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)* (pp. 0785-0791). IEEE.

[3] Hasan, M., Islam, M. M., Zarif, M. I. I., & Hashem, M. M. A. (2019). Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. *Internet of Things*, *7*, 100059.

[4] Alzahrani, A. O., & Alenazi, M. J. (2021). Designing a Network Intrusion Detection System Based on Machine Learning for Software Defined Networks. *Future Internet*, *13*(5), 111.

[5] Kwon, D., Natarajan, K., Suh, S. C., Kim, H., & Kim, J. (2018, July). An empirical study on network anomaly detection using convolutional neural networks. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (pp. 1595-1598). IEEE.

[6] Alrashdi, I., Alqazzaz, A., Aloufi, E., Alharthi, R., Zohdy, M., & Ming, H. (2019, January). Ad-iot: Anomaly detection of iot cyberattacks in smart city using machine learning. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)* (pp. 0305-0310). IEEE.

[7] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference onComputer Vision and Pattern Recognition* (pp. 2820-2828)

[8] Nandhini, K., Pavithra, M., Revathi, K., & Rajiv, A. (2017, March). Anamoly detection for safety monitoring. In *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)* (pp. 1-6). IEEE.

[9] H. M. Anwer, M. Farouk and A. Abdel-Hamid, "A framework for efficient network anomaly intrusion detection with features selection," 2018 9th International Conference on Information and Communication Systems (ICICS), 2018, pp. 157-162, doi: 10.1109/IACS.2018.8355459.

[10] Khraisat A., Gondal I., Vamplew P. (2018) An Anomaly Intrusion Detection System Using C5 Decision Tree Classifier. In: Ganji M., Rashidi L., Fung B., Wang C. (eds) Trends and Applications in Knowledge Discovery and Data Mining. PAKDD 2018. Lecture Notes in Computer Science, vol 11154. Springer, Cham.

[11] Ishaq, Kashif. "An Efficient Hybrid Classifier Model for Anomaly Intrusion Detection System." International Journal of Computer Science and Network Security (2018)

[12] P. Satam and S. Hariri, "WIDS: An Anomaly Based Intrusion Detection System for Wi-Fi (IEEE 802.11) Protocol," in IEEE Transactions on Network and Service Management, vol. 18, no. 1, pp. 1077-1091, March 2021, doi: 10.1109/TNSM.2020.3036138.

[13] WS, J. D. S., Parvathavarthini, B., & Arunmozhi, S. (2020, July). Machine Learning based Intrusion Detection Framework using Recursive Feature Elimination Method. In *2020 International Conference on*

System, Computation, Automation and Networking (ICSCAN) (pp. 1-4). IEEE.

[14] Divakar, S., Priyadarshini, R., & Mishra, B. K. (2020, December). A Robust Intrusion Detection System using Ensemble Machine Learning. In *2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)* (pp. 344-347). IEEE.

[15] Rokade, M. D., & Sharma, Y. K. (2021, March). MLIDS: A Machine Learning Approach for Intrusion Detection for Real Time Network Dataset. In *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)* (pp. 533-536). IEEE.

[16] Schueller, Q., Basu, K., Younas, M., Patel, M., & Ball, F. (2018, November). A hierarchical intrusion detection system using support vector machine for SDN network in cloud data center. In *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)* (pp. 1-6). IEEE.

[17] Zoph, Barret, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. "Learning transferable architectures for scalable image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697-8710. 2018.

[18Arfeen, Muhammad Asad, Krzysztof Pawlikowski, D. McNickle, and Andreas Willig. "The role of the weibull distribution in internet traffic modeling." In *Proceedings of the 2013 25th International Teletraffic Congress (ITC)*, pp. 1-8. IEEE, 2013.

[19] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 2820-2828).

[20] Alrashdi, Ibrahim, et al. "Ad-iot: Anomaly detection of iot cyberattacks in smart city using machine learning." *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2019.