NETWORK INTRUSION DETECTION SYSTEM USING SINGLE LEVEL MULTI MODEL DECISION TREES

CONTENTS

ABSTRACT

An intrusion detection system(IDS) is basically a software application that monitors a network or systems for malicious activity or policy violations. Any intrusion activity or violation of the protocols is reported to an authorized person or an administrator. Intrusion detection helps to detect a malicious activity which may cause any harm or damage to the important data or services. We have made an Intrusion detection using Machine Learning which is able to detect attacks if they are performed on the network or the system. It analyzes the various network parameters like the flag, srv_count , etc and on the basis of these values it predicts whether an attack is being performed or not. The number of false positives have been reduced to a large extent and the accuracy for correct prediction is also very good.

INTRODUCTION

Intrusion detection system allows packets to pass and then based on the performance or behavior of those packets in the system the IDS stops the further incoming packets. An advantage of an Intrusion detection system is that it is very fast and hence only a very less number of infected packets are able to enter the system and the entry of suck packets is stopped very soon thus saving the system from much damage. In the today's world where everyone is connected to the Internet and everything is being done online and there is a lot of communication between the devices where the data packets are sent from one PC to another, several types of attacks have evolved which not only can harm the private data of organization but can also lead to the disruption of services like in the case of Denial Of Service attack. So a system which can detect such malicious packets and protect the system from losses is the need of the hour. Intrusion detection System is a must for any company

or organization dealing with general to protect its internal systems and data from the attacker. The IDS that we have made here protects the systems from four different kinds of attacks by detecting them with a good accuracy. The thing that we have focussed a lot is on decreasing the number of false positives since a higher number of false positives will continuously disrupt the normal working of the company in-spite of there not being any danger or attacks. This allows a company to carry on its normal routine work without any unnecessary disruptions or stoppages.

OBJECTIVES

- To make an Intrusion detection system using Machine Learning which is able to detect attacks if they are performed on the network or the system.
- To detect a malicious activity which may cause any harm or damage to the important data or services.
- To make a system which will analyze the various network parameters like the flag, srv_count , etc and on the basis of these values it will predict whether an attack is being performed or not.
- To reduce the number of false positives to a large extent and increase the accuracy for correct prediction.

LITERATURE STUDY

[1] Machine learning and deep learning methods for intrusion detection systems: recent developments and challenges

By: Geeta Kocher, Gulshan Kumar

Year: 2021

The paper used a systematic review methodology for both ML and DL methods in intrusion detection, which ensured a structured and methodical approach to the research, using quantitative methods to compare different approaches and understand their strengths and weaknesses. The authors analyzed benchmark datasets and performance evaluation measures, compared experimental results, and identified current research challenges to provide clues for future research. The methodology used in the paper appears rigorous and thorough, enhancing the reliability and validity of the findings.

[2] Federated Learning for intrusion detection system: Concepts, challenges and future directions

By: Shaashwat Agrawal, Sagnik Sarkar, Ons Aouedi, Gokul Yenduri, Kandaraj Piamrat, Mamoun Alazab, Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu

Year: 2022

The paper aims to present a comprehensive review of the use of Federated Learning (FL) in Intrusion Detection Systems (IDS) and to identify the limitations of this approach. The authors discuss the challenges associated with the deployment of IDS

due to the growing complexity and heterogeneity of network infrastructures caused by the rise in usage of smart devices, such as mobile phones, wearable devices, and autonomous vehicles. They also explain the advantages of FL as a privacy-preserving, decentralized learning technique that trains models locally and transfers parameters to the centralized server, rather than transmitting data. The paper reviews the implementation of FL in various aspects of anomaly detection and identifies the challenges and issues of using FL for IDS.

[3] Accuracy of Machine Learning Algorithms in Detecting DoS Attacks Type

By: Noureldien Yousif

Year: 2017

Inspected the exhibition of seven regulated AI calculations in identifying the DoS assaults utilizing NSLKDD dataset. They utilized 10-crease cross approval in tests and assess the strategies to affirm that methods will accomplish on undetected information. Their outcomes showed that the Random Committee was the best calculation for identifying smurf assault with an exactness of 98.6161%.

[4] An effective intrusion detection framework based on SVM with feature augmentation

By: Wang

Year: 2017

They proposed a SVM based interruption identification procedure that considers pre handling information using changing over the typical qualities by the logarithms of the negligible thickness proportions that abuses the order data that is remembered for each element. This resulted in information that has top caliber and compact which

thus accomplished a superior recognition execution as well as lessening the preparation time required for the SVM identification model .

[5] A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks

By: Yuefei Zhu, Jinlong Fei

Year: 2017

They have investigated how to show an IDS dependent on a profound learning approach utilizing repetitive neural organizations (RNN-IDS) in view of its capability of removing better portrayals for the information and making better models. They preprocessed the dataset utilizing Numericalization procedure on the grounds that the information worth of RNN-IDS ought to be a numeric network. The outcomes showed that RNNIDS has high precision rate and location rate but a low positive rate contrasted with conventional grouping techniques.

[6] Intrusion detection model using fusion of chisquare feature selection and multi class SVM

By: Ikram and Cherukuri

Year: 2017

The researchers proposed an ID model utilizing Chi-Square trait choice and multi-class support vector machine (SVM). The principle thought behind this model is to develop a multi class SVM which has not been received for IDS so far to diminish the preparation and testing time and increment the individual grouping precision of the organization assaults

[7] Intelligent network intrusion detection using alternating decision trees

By: Jabbar and Samreen

Year: 2016

They have introduced a novel methodology for ID utilizing exchanging choice trees (ADT) to characterize the different kinds of assaults while it is normally utilized for parallel characterization issues. The outcomes showed that their proposed model delivered a higher location rate and decreased the bogus caution rate in order of IDS assaults.

[8] Performance of Machine Learning-Based Multi-Model Voting Ensemble Methods for Network Threat Detection in Agriculture 4.0

By: Nikolaos Peppes, Emmanouil Daskalakis, Theodoros Alexakis, Evgenia Adamopoulou and Konstantinos Demestichas

Year: 2021

Machine learning-based multi-model voting ensemble methods are being used for network threat detection in Agriculture 4.0. These methods involve combining the predictions of multiple machine learning models to improve the accuracy of the system. Studies have shown that such methods are effective for detecting anomalous behavior in network traffic and identifying potential threats.

EXISTING MODEL

There are several models of Intrusion detection systems which exist. There are rule based IDS and anomaly based IDS. Rule based IDS have some predefined rules and signatures based on which it will try to detect the attacks. Anomaly based IDS usually detect the attacks based on statistical models which detect deviation from normal network activity which helps in detecting new and unknown attacks.

There are several demerits associated with these existing models, rule based IDS has high false positive rate, if rules are not defined accurately then there might be false alarms and may not detect the malicious activity accurately. Normal behavior can change over time, as anomaly based detection is based on statistical models, the change in normal behavior can be misidentified by the system and might be considered as malicious behavior.

PROPOSED MODEL

The essential objective is to plan an arrangement detecting intrusions within the system inside the framework with the number of features inside the dataset. In view of the information from past papers distributed, we can tell that a development of highlights in the dataset is subsidiary to the Intrusion Detection System. We need to scale back the dimensionality of the dataset to assemble an improved classifier in a legitimate measure of time. The methodology we will utilize has a sum of 4 phases : In the primary stage, we choose the huge highlights for each class utilizing highlight selection. In the following we join the different highlights, with the goal that the last bunch of highlights are ideal and important for each assault class. The third stage is for building a classifier. Here, the ideal highlights found in the past stage are sent as

a contribution to the classifier. In the last stage, we test the model by utilizing a test dataset.

Attacks against which the IDS will provide protection:

1) U2R : U2R attack means unauthorized access to local root privileges. This is the type of attack where the attacker attempts to illegally obtain root privileges by actually legally accessing a local machine by using some vulnerability in the victim's system to his advantage.

2) R2L : Remote to local attack is launched by an attacker to gain unauthorized access to a victim machine in the entire network. Here the attacker gains access to the victim's device by gaining the root access. It is similar to the U2R attack.

3) DoS : A denial-of-service attack (DoS attack) is a cyber-attack in which the attacker seeks to make a machine or network resource unavailable to its intended users by disrupting services of a host connected to the Internet by sending millions and millions of packets to the server responsible for the service thus crashing the server.

4) Probing : This is also a cyber attack where the attacker tries to steal sensitive information present in the victim's system. Many of the similar attacks have been classified into these four categories to extend this project to detect more attacks.

ATTACK TYPE	ATTACK NAME
DOS	Neptune, Smurf, Pod, Teardrop, Land, Back
Probe	Port-Sweep, IP-sweep, Nmap, Satan
R2L	Guess-password, Ftp-write, Imap, Phf, Multihop, spy, warezclient, Warezmaster
U2R	Buffer-overflow, Load-module, Perl, Rootkit

MODULES USED

Feature selection

Here we will utilize Information Gain (IG) so that we can choose the relevant features from the dataset. It is determined for each and every class independently. The classes are ranked as per the information gain such that if the value is less than a threshold value, feature will be eliminated.We partition the preparation dataset into 4 datasets. The preparation dataset is partitioned into 4 datasets so that each dataset comprises records having a place with a similar attack class alongside a portion of the records of the first dataset. Then the datasets for each attack class are sent independently as info into the technique used to compute the attack class. Obtaining the best features Here we will be using recursive feature elimination technique to eliminate the lower ranked features so that we have only more relevant and higher ranked features which decreases the computation and increases the accuracy.

Developing a classifier

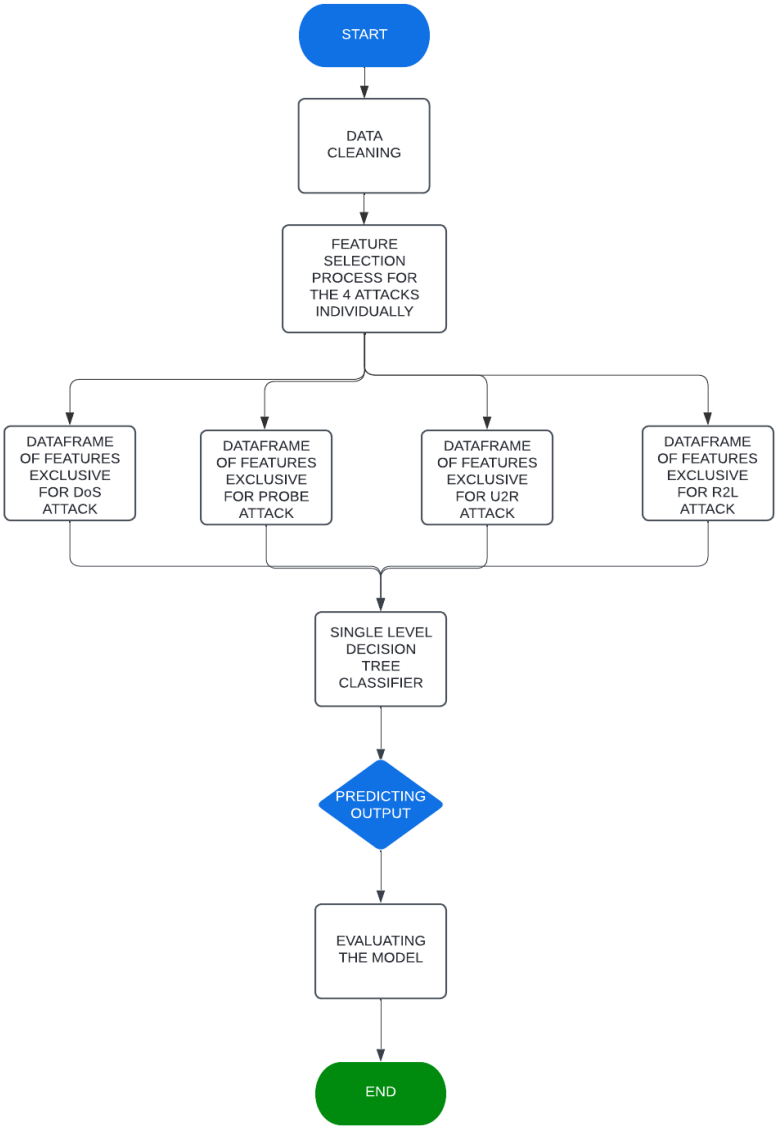
A supervised machine learning model is being used here which is used to classify a label data into a particular class. The classification algorithm that we have used in our project is decision tree classifier Decision tree is an algorithm which takes

decisions at each node of the tree and is widely used for regression and classification. We have chosen decision trees since they can be trained very easily. It is more productive than most of the classification algorithms in ML like KNearest Neighbours in most of the cases.

Dataset Description:

We are using the NSL-KDD dataset for the modeling of the IDS system. The KDD data set is a well known benchmark in the research of Intrusion Detection techniques. However, NSL-KDD is a new version of the KDD'99 data set. This is an effective benchmark data set to help researchers compare different intrusion detection methods. It beats some inborn issues in the first KDD dataset: repetitive records in the preparation set have been eliminated so that the classifiers produce unprejudiced outcomes.

ARCHITECTURE MODEL



OUTCOMES OF THE PROPOSED MODEL

Intrusion detection System is a must for any company or organization dealing with general to protect its internal systems and data from the attacker.

The essential outcome is to plan an arrangement detecting intrusions within the system inside the framework with the number of features inside the dataset.

The IDS that we intend to make protects the systems from four different kinds of attack (U2R, R2L, DoS, Probing) by detecting them with a good accuracy.
The outcome that we are expecting from this project is to create single level decision tree classifiers for every attack type ie. DoS, U2R, R2L and Probe and make it fine tuned so that on passing an input, it predicts whether such an attack will take place or not.
We intend to focus a lot on decreasing the number of false positives since a higher number of false positives will continuously disrupt the normal working of the company in-spite of there not being any danger or attacks. This allows a company to carry on its normal routine work without any unnecessary disruptions or stoppages.

MERITS OF THE MODEL

- Detection of attacks more efficiently:

  The IDS is made using machine learning algorithms like decision trees which can efficiently detect the attacks performed on the system or network.Which helps in identifying and preventing the malicious activity quickly and thereby protecting the system from potential harm.

- Capable of monitoring the system in real-time:

  The system is capable of monitoring the network in the realtime which helps in the identification of attacks and responding to them immediately. By which we can stop the attacks before any damage is done.

- Reduced number of false positives:

The main and important merit of the project is that the number of false positives is reduced using this system, which makes sure that only actual attacks are detected and false alarms are not given.

- Increased accuracy:

  Many network parameters like flag,srv_count are considered and analyzed to predict if an attack is performed or not, which helps in increasing the accuracy and attacks are predicted accurately.

DEMERITS OF THE MODEL

- Detection of limited number of attacks:

  The IDS which is made in this project focuses mainly on 4 types of attacks U2R, R2L, Dos and probing. So the system may not detect all types of attacks.

- Reliance on network parameters:

  This project has heavy reliability on network parameters to detect the attacks which may not always be accurate and sometimes an attack may be undetected due to unreliable network parameters.

CODE AND OUTPUT

In [ ]:
```
import pandas as pd
import numpy as np
import sys
import sklearn
print(pd.__version__)
print(np.__version__)
print(sys.version)
print(sklearn.__version__)
```

```
1.3.5
1.21.6
3.8.10 (default, Nov 14 2022, 12:59:47)
[GCC 9.4.0]
1.0.2
```

In [ ]:
```
# attach the column names to the dataset
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"]

# KDDTrain+_2.csv & KDDTest+_2.csv are the datafiles without the last column about the difficulty score
# these have already been removed.
df = pd.read_csv("KDDTrain+_2.csv", header=None, names = col_names)
df_test = pd.read_csv("KDDTest+_2.csv", header=None, names = col_names)

# shape, this gives the dimensions of the dataset
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)
```

Sample view of the training dataset

In [ ]:
```
# first five rows
df.head(5)
```

Out[4]:

	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_diff_sr
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	

5 rows × 42 columns

Label Distribution of the Training set and the Test set

In [ ]:
```
print('Label distribution Training set:')
print(df['label'].value_counts())
print()
print('Label distribution Test set:')
print(df_test['label'].value_counts())
```

```
Label distribution Training set:
normal              67343
neptune             41214
satan                3633
ipsweep              3599
portsweep            2931
smurf                2646
nmap                 1493
back                  956
teardrop              892
warezclient           890
pod                   201
guess_passwd           53
buffer_overflow        30
warezmaster            20
land                   18
imap                   11
rootkit                10
loadmodule              9
ftp_write               8
multihop                7
phf                     4
perl                    3
spy                     2
Name: label, dtype: int64

Label distribution Test set:
normal              9711
neptune             4657
guess_passwd        1231
mscan                996
warezmaster          944
apache2              737
satan                735
processtable         685
smurf                665
back                 359
snmpguess            331
saint                319
mailbomb             293
snmpgetattack        178
portsweep            157
ipsweep              141
httptunnel           133
nmap                  73
pod                   41
buffer_overflow       20
multihop              18
named                 17
ps                    15
sendmail              14
rootkit               13
xterm                 13
teardrop              12
xlock                  9
land                   7
xsnoop                 4
ftp_write              3
worm                   2
loadmodule             2
perl                   2
sqlattack              2
udpstorm               2
phf                    2
imap                   1
Name: label, dtype: int64
```

Here we identify categorical features

In [ ]:
```
# colums that are categorical and not binary yet: protocol_type (column 2), service (column 3), flag (column 4).
# explore categorical features
print('Training set:')
for col_name in df.columns:
    if df[col_name].dtypes == 'object' :
        unique_cat = len(df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))

#see how distributed the feature service is, it is evenly distributed and therefore we need to make dummies for all.
print()
print('Distribution of categories in service:')
print(df['service'].value_counts().sort_values(ascending=False).head())
```

```
Training set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 70 categories
Feature 'flag' has 11 categories
Feature 'label' has 23 categories

Distribution of categories in service:
http        40338
private     21853
domain_u     9043
smtp         7313
ftp_data     6860
Name: service, dtype: int64
```

In [ ]:
```
# Test set
print('Test set:')
for col_name in df_test.columns:
    if df_test[col_name].dtypes == 'object' :
        unique_cat = len(df_test[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))
```

```
Test set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 64 categories
Feature 'flag' has 11 categories
Feature 'label' has 38 categories
```

LabelEncoder

Here we insert the categorical features into a 2D numpy array

In [ ]:
```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
categorical_columns=['protocol_type', 'service', 'flag']
# insert code to get a list of categorical columns into a variable, categorical_columns
categorical_columns=['protocol_type', 'service', 'flag']
 # Get the categorical values into a 2D numpy array
df_categorical_values = df[categorical_columns]
testdf_categorical_values = df_test[categorical_columns]
df_categorical_values.head()
```

Out[8]:

	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

Now we make column names for dummies

In [ ]:
```
# protocol type
unique_protocol=sorted(df.protocol_type.unique())
string1 = 'Protocol_type_'
unique_protocol2=[string1 + x for x in unique_protocol]
# service
unique_service=sorted(df.service.unique())
string2 = 'service_'
unique_service2=[string2 + x for x in unique_service]
# flag
unique_flag=sorted(df.flag.unique())
string3 = 'flag_'
unique_flag2=[string3 + x for x in unique_flag]
# put together
dumcols=unique_protocol2 + unique_service2 + unique_flag2
print(dumcols)

#test set
unique_service_test=sorted(df_test.service.unique())
unique_service2_test=[string2 + x for x in unique_service_test]
testdumcols=unique_protocol2 + unique_service2_test + unique_flag2
print(testdumcols)
```

```
['Protocol_type_icmp', 'Protocol_type_tcp', 'Protocol_type_udp', 'service_IRC', 'service_X11', 'service_Z39_50', 'service_aol', 'se
rvice_auth', 'service_bgp', 'service_courier', 'service_csnet_ns', 'service_ctf', 'service_daytime', 'service_discard', 'service_do
main', 'service_domain_u', 'service_echo', 'service_eco_i', 'service_ecr_i', 'service_efs', 'service_exec', 'service_finger', 'serv
ice_ftp', 'service_ftp_data', 'service_gopher', 'service_harvest', 'service_hostnames', 'service_http', 'service_http_2784', 'servi
ce_http_443', 'service_http_8001', 'service_imap4', 'service_iso_tsap', 'service_klogin', 'service_kshell', 'service_ldap', 'servic
e_link', 'service_login', 'service_mtp', 'service_name', 'service_netbios_dgm', 'service_netbios_ns', 'service_netbios_ssn', 'servi
ce_netstat', 'service_nnsp', 'service_nntp', 'service_ntp_u', 'service_other', 'service_pm_dump', 'service_pop_2', 'service_pop_3',
'service_printer', 'service_private', 'service_red_i', 'service_remote_job', 'service_rje', 'service_shell', 'service_smtp', 'servi
ce_sql_net', 'service_ssh', 'service_sunrpc', 'service_supdup', 'service_systat', 'service_telnet', 'service_tftp_u', 'service_tim_
i', 'service_time', 'service_urh_i', 'service_urp_i', 'service_uucp', 'service_uucp_path', 'service_vmnet', 'service_whois', 'flag_
OTH', 'flag_REJ', 'flag_RSTO', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0', 'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH']
['Protocol_type_icmp', 'Protocol_type_tcp', 'Protocol_type_udp', 'service_IRC', 'service_X11', 'service_Z39_50', 'service_auth', 's
ervice_bgp', 'service_courier', 'service_csnet_ns', 'service_ctf', 'service_daytime', 'service_discard', 'service_domain', 'service
_domain_u', 'service_echo', 'service_eco_i', 'service_ecr_i', 'service_efs', 'service_exec', 'service_finger', 'service_ftp', 'serv
ice_ftp_data', 'service_gopher', 'service_hostnames', 'service_http', 'service_http_443', 'service_imap4', 'service_iso_tsap', 'ser
vice_klogin', 'service_kshell', 'service_ldap', 'service_link', 'service_login', 'service_mtp', 'service_name', 'service_netbios_dg
m', 'service_netbios_ns', 'service_netbios_ssn', 'service_netstat', 'service_nnsp', 'service_nntp', 'service_ntp_u', 'service_othe
r', 'service_pm_dump', 'service_pop_2', 'service_pop_3', 'service_printer', 'service_private', 'service_remote_job', 'service_rje',
'service_shell', 'service_smtp', 'service_sql_net', 'service_ssh', 'service_sunrpc', 'service_supdup', 'service_systat', 'service_t
elnet', 'service_tftp_u', 'service_tim_i', 'service_time', 'service_urp_i', 'service_uucp', 'service_uucp_path', 'service_vmnet',
'service_whois', 'flag_OTH', 'flag_REJ', 'flag_RSTO', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0', 'flag_S1', 'flag_S2', 'flag_S3', 'flag
_SF', 'flag_SH']
```

Here we transform categorical features into numbers using LabelEncoder()

In [ ]:
```
df_categorical_values_enc=df_categorical_values.apply(LabelEncoder().fit_transform)
print(df_categorical_values_enc.head())
# test set
testdf_categorical_values_enc=testdf_categorical_values.apply(LabelEncoder().fit_transform)
```

```
   protocol_type  service  flag
0              1       20     9
1              2       44     9
2              1       49     5
3              1       24     9
4              1       24     9
```

One-Hot-Encoding

In [ ]:
```
enc = OneHotEncoder()
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[11]:

	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

Now we add 6 missing categories from the training set to the test set

In [ ]:
```
trainservice=df['service'].tolist()
testservice= df_test['service'].tolist()
difference=list(set(trainservice) - set(testservice))
string = 'service_'
difference=[string + x for x in difference]
difference
```

Out[12]:
```
['service_red_i',
 'service_urh_i',
 'service_http_2784',
 'service_http_8001',
 'service_aol',
 'service_harvest']
```

In [ ]:
```
for col in difference:
    testdf_cat_data[col] = 0

testdf_cat_data.shape
```

Out[13]:
```
(22544, 84)
```

Here we join the encoded categorical dataframe with the non-categorical dataframe

In [ ]:
```
newdf=df.join(df_cat_data)
newdf.drop('flag', axis=1, inplace=True)
newdf.drop('protocol_type', axis=1, inplace=True)
newdf.drop('service', axis=1, inplace=True)
# test data
newdf_test=df_test.join(testdf_cat_data)
newdf_test.drop('flag', axis=1, inplace=True)
newdf_test.drop('protocol_type', axis=1, inplace=True)
newdf_test.drop('service', axis=1, inplace=True)
print(newdf.shape)
print(newdf_test.shape)
```

```
(125973, 123)
(22544, 123)
```

Now we split Dataset into 4 datasets for every attack category

In [ ]:
```
# take label column
labeldf=newdf['label']
labeldf_test=newdf_test['label']
# change the label column
newlabeldf=labeldf.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbomb': 1, '
                            'ipsweep' : 2,'nmap' : 2,'portsweep' : 2,'satan' : 2,'mscan' : 2,'saint' : 2
                            ,'ftp_write': 3,'guess_passwd': 3,'imap': 3,'multihop': 3,'phf': 3,'spy': 3,'warezclient': 3,'warezmaster
                            'buffer_overflow': 4,'loadmodule': 4,'perl': 4,'rootkit': 4,'ps': 4,'sqlattack': 4,'xterm': 4})
newlabeldf_test=labeldf_test.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailb
                            'ipsweep' : 2,'nmap' : 2,'portsweep' : 2,'satan' : 2,'mscan' : 2,'saint' : 2
                            ,'ftp_write': 3,'guess_passwd': 3,'imap': 3,'multihop': 3,'phf': 3,'spy': 3,'warezclient': 3,'warezmaster
                            'buffer_overflow': 4,'loadmodule': 4,'perl': 4,'rootkit': 4,'ps': 4,'sqlattack': 4,'xterm': 4})
# put the new label column back
newdf['label'] = newlabeldf
newdf_test['label'] = newlabeldf_test
print(newdf['label'].head())
```

```
0    0
1    0
2    1
3    0
4    0
Name: label, dtype: int64
```

In [ ]:
```
to_drop_DoS = [2,3,4]
to_drop_Probe = [1,3,4]
to_drop_R2L = [1,2,4]
to_drop_U2R = [1,2,3]

#TRAIN
DoS_df=newdf[~newdf['label'].isin(to_drop_DoS)];
Probe_df=newdf[~newdf['label'].isin(to_drop_Probe)];
R2L_df=newdf[~newdf['label'].isin(to_drop_R2L)];
U2R_df=newdf[~newdf['label'].isin(to_drop_U2R)];

#test
DoS_df_test=newdf_test[~newdf_test['label'].isin(to_drop_DoS)];
Probe_df_test=newdf_test[~newdf_test['label'].isin(to_drop_Probe)];
R2L_df_test=newdf_test[~newdf_test['label'].isin(to_drop_R2L)];
U2R_df_test=newdf_test[~newdf_test['label'].isin(to_drop_U2R)];
print('Train:')
print('Dimensions of DoS:' ,DoS_df.shape)
print('Dimensions of Probe:' ,Probe_df.shape)
print('Dimensions of R2L:' ,R2L_df.shape)
print('Dimensions of U2R:' ,U2R_df.shape)
print('Test:')
print('Dimensions of DoS:' ,DoS_df_test.shape)
print('Dimensions of Probe:' ,Probe_df_test.shape)
print('Dimensions of R2L:' ,R2L_df_test.shape)
print('Dimensions of U2R:' ,U2R_df_test.shape)
```

```
Train:
Dimensions of DoS: (113270, 123)
Dimensions of Probe: (78999, 123)
Dimensions of R2L: (68338, 123)
Dimensions of U2R: (67395, 123)
Test:
Dimensions of DoS: (17171, 123)
Dimensions of Probe: (12132, 123)
Dimensions of R2L: (12596, 123)
Dimensions of U2R: (9778, 123)
```

Step 2: Feature Scaling

In [ ]:
```
# Split dataframes into X & Y
# assign X as a dataframe of feautures and Y as a series of outcome 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
```

```
<ipython-input-17-c950fdcb5610>:3: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argu
ment 'labels' will be keyword-only
  X_DoS = DoS_df.drop('label',1)
<ipython-input-17-c950fdcb5610>:5: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argu
ment 'labels' will be keyword-only
  X_Probe = Probe_df.drop('label',1)
<ipython-input-17-c950fdcb5610>:7: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argu
ment 'labels' will be keyword-only
  X_R2L = R2L_df.drop('label',1)
<ipython-input-17-c950fdcb5610>:9: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argu
ment 'labels' will be keyword-only
  X_U2R = U2R_df.drop('label',1)
<ipython-input-17-c950fdcb5610>:12: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the arg
ument 'labels' will be keyword-only
  X_DoS_test = DoS_df_test.drop('label',1)
<ipython-input-17-c950fdcb5610>:14: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the arg
ument 'labels' will be keyword-only
  X_Probe_test = Probe_df_test.drop('label',1)
<ipython-input-17-c950fdcb5610>:16: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the arg
ument 'labels' will be keyword-only
  X_R2L_test = R2L_df_test.drop('label',1)
<ipython-input-17-c950fdcb5610>:18: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the arg
ument 'labels' will be keyword-only
  X_U2R_test = U2R_df_test.drop('label',1)
```

Here we save a list of feature names for later use (it is the same for every attack category).

In [ ]:
```
colNames=list(X_DoS)
colNames_test=list(X_DoS_test)
```

We use StandardScaler() to scale the dataframes

In [ ]:
```
from sklearn import preprocessing
scaler1 = preprocessing.StandardScaler().fit(X_DoS)
X_DoS=scaler1.transform(X_DoS)
scaler2 = preprocessing.StandardScaler().fit(X_Probe)
X_Probe=scaler2.transform(X_Probe)
scaler3 = preprocessing.StandardScaler().fit(X_R2L)
X_R2L=scaler3.transform(X_R2L)
scaler4 = preprocessing.StandardScaler().fit(X_U2R)
X_U2R=scaler4.transform(X_U2R)
# test data
scaler5 = preprocessing.StandardScaler().fit(X_DoS_test)
X_DoS_test=scaler5.transform(X_DoS_test)
scaler6 = preprocessing.StandardScaler().fit(X_Probe_test)
X_Probe_test=scaler6.transform(X_Probe_test)
scaler7 = preprocessing.StandardScaler().fit(X_R2L_test)
X_R2L_test=scaler7.transform(X_R2L_test)
scaler8 = preprocessing.StandardScaler().fit(X_U2R_test)
X_U2R_test=scaler8.transform(X_U2R_test)
```

In [ ]:
```
print(X_DoS.std(axis=0))
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1.
 1. 1.]
```

In [ ]:
```
X_Probe.std(axis=0);
X_R2L.std(axis=0);
X_U2R.std(axis=0);
```

Step 3: Feature Selection

1. Univariate Feature Selection using ANOVA F-test

In [ ]:
```
from sklearn.feature_selection import SelectPercentile, f_classif
np.seterr(divide='ignore', invalid='ignore');
selector=SelectPercentile(f_classif, percentile=10)
X_newDoS = selector.fit_transform(X_DoS,Y_DoS)
X_newDoS.shape
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_selection/_univariate_selection.py:112: UserWarning: Features [ 16  44  63
66  68  86 114] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
```

Out[22]:
```
(113270, 13)
```

Get the features that were selected: DoS

In [ ]:
```
true=selector.get_support()
newcolindex_DoS=[i for i, x in enumerate(true) if x]
newcolname_DoS=list( colNames[i] for i in newcolindex_DoS )
newcolname_DoS
```

Out[23]:
```
['logged_in',
 'count',
 'serror_rate',
 'srv_serror_rate',
 'same_srv_rate',
 'dst_host_count',
 'dst_host_srv_count',
 'dst_host_same_srv_rate',
 'dst_host_serror_rate',
 'dst_host_srv_serror_rate',
 'service_http',
 'flag_S0',
 'flag_SF']
```

In [ ]:
```
X_newProbe = selector.fit_transform(X_Probe,Y_Probe)
X_newProbe.shape
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_selection/_univariate_selection.py:112: UserWarning: Features [ 4 16] are co
nstant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
```

Out[24]:
```
(78999, 13)
```

Get the features that were selected: Probe

In [ ]:
```
true=selector.get_support()
newcolindex_Probe=[i for i, x in enumerate(true) if x]
newcolname_Probe=list( colNames[i] for i in newcolindex_Probe )
newcolname_Probe
```

Out[25]:
```
['logged_in',
 'rerror_rate',
 'srv_rerror_rate',
 'dst_host_srv_count',
 'dst_host_diff_srv_rate',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'dst_host_rerror_rate',
 'dst_host_srv_rerror_rate',
 'Protocol_type_icmp',
 'service_eco_i',
 'service_private',
 'flag_SF']
```

In [ ]:
```
X_newR2L = selector.fit_transform(X_R2L,Y_R2L)
X_newR2L.shape
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_selection/_univariate_selection.py:112: UserWarning: Features [  4  16  43
44  46  47  48  49  50  51  54  57  58  62  63  64  66  67
  68  70  71  72  73  74  76  77  78  79  80  81  82  83  86  87  89  92
  93  96  98  99 100 107 108 109 110 114] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
```

Out[26]:
```
(68338, 13)
```

Get the features that were selected: R2L

In [ ]:
```
true=selector.get_support()
newcolindex_R2L=[i for i, x in enumerate(true) if x]
newcolname_R2L=list( colNames[i] for i in newcolindex_R2L)
newcolname_R2L
```

Out[27]:
```
['src_bytes',
 'dst_bytes',
 'hot',
 'num_failed_logins',
 'is_guest_login',
 'dst_host_srv_count',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'service_ftp',
 'service_ftp_data',
 'service_http',
 'service_imap4',
 'flag_RSTO']
```

In [ ]:
```
X_newU2R = selector.fit_transform(X_U2R,Y_U2R)
X_newU2R.shape
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_selection/_univariate_selection.py:112: UserWarning: Features [  4  16  43
44  46  47  48  49  50  51  54  57  58  62  63  64  66  67
  68  70  71  72  73  74  75  76  77  78  79  80  81  82  83  86  87  89
  92  93  96  98  99 100 107 108 109 110 114] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
```

Out[28]:
```
(67395, 13)
```

Get the features that were selected: U2R

In [ ]:
```
true=selector.get_support()
newcolindex_U2R=[i for i, x in enumerate(true) if x]
newcolname_U2R=list( colNames[i] for i in newcolindex_U2R)
newcolname_U2R
```

Out[29]:
```
['urgent',
 'hot',
 'root_shell',
 'num_file_creations',
 'num_shells',
 'srv_diff_host_rate',
 'dst_host_count',
 'dst_host_srv_count',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'service_ftp_data',
 'service_http',
 'service_telnet']
```

Summary of features selected by Univariate Feature Selection

In [ ]:
```
print('Features selected for DoS:',newcolname_DoS)
print()
print('Features selected for Probe:',newcolname_Probe)
print()
print('Features selected for R2L:',newcolname_R2L)
print()
print('Features selected for U2R:',newcolname_U2R)
```

```
Features selected for DoS: ['logged_in', 'count', 'serror_rate', 'srv_serror_rate', 'same_srv_rate', 'dst_host_count', 'dst_host_sr
v_count', 'dst_host_same_srv_rate', 'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'service_http', 'flag_S0', 'flag_SF']

Features selected for Probe: ['logged_in', 'rerror_rate', 'srv_rerror_rate', 'dst_host_srv_count', 'dst_host_diff_srv_rate', 'dst_h
ost_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate', 'Protocol_type_icmp',
'service_eco_i', 'service_private', 'flag_SF']

Features selected for R2L: ['src_bytes', 'dst_bytes', 'hot', 'num_failed_logins', 'is_guest_login', 'dst_host_srv_count', 'dst_host
_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'service_ftp', 'service_ftp_data', 'service_http', 'service_imap4', 'flag_RST
O']

Features selected for U2R: ['urgent', 'hot', 'root_shell', 'num_file_creations', 'num_shells', 'srv_diff_host_rate', 'dst_host_coun
t', 'dst_host_srv_count', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'service_ftp_data', 'service_http', 'servic
e_telnet']
```

2. Recursive Feature Elimination for feature ranking

In [ ]:
```
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
# Create a decision tree classifier. By convention, clf means 'classifier'
clf = DecisionTreeClassifier(random_state=0)

#rank all features, i.e continue the elimination until the last one
rfe = RFE(clf, n_features_to_select=1)
```

In [ ]:
```
Y_DoS=Y_DoS.astype('int')
rfe.fit(X_newDoS, Y_DoS)
print ("DoS Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newcolname_DoS)))
```

```
DoS Features sorted by their rank:
[(1, 'same_srv_rate'), (2, 'count'), (3, 'flag_SF'), (4, 'dst_host_serror_rate'), (5, 'dst_host_same_srv_rate'), (6, 'dst_host_srv_
count'), (7, 'dst_host_count'), (8, 'logged_in'), (9, 'serror_rate'), (10, 'dst_host_srv_serror_rate'), (11, 'srv_serror_rate'), (1
2, 'service_http'), (13, 'flag_S0')]
```

In [ ]:
```
Y_Probe=Y_Probe.astype('int')

rfe.fit(X_newProbe, Y_Probe)
print ("Probe Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newcolname_Probe)))
```

```
Probe Features sorted by their rank:
[(1, 'dst_host_same_src_port_rate'), (2, 'dst_host_srv_count'), (3, 'dst_host_rerror_rate'), (4, 'service_private'), (5, 'logged_i
n'), (6, 'dst_host_diff_srv_rate'), (7, 'dst_host_srv_diff_host_rate'), (8, 'flag_SF'), (9, 'service_eco_i'), (10, 'rerror_rate'),
(11, 'Protocol_type_icmp'), (12, 'dst_host_srv_rerror_rate'), (13, 'srv_rerror_rate')]
```

In [ ]:
```
Y_R2L=Y_R2L.astype('int')

rfe.fit(X_newR2L, Y_R2L)
print ("R2L Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newcolname_R2L)))
```

```
R2L Features sorted by their rank:
[(1, 'src_bytes'), (2, 'dst_bytes'), (3, 'hot'), (4, 'dst_host_srv_diff_host_rate'), (5, 'service_ftp_data'), (6, 'dst_host_same_sr
c_port_rate'), (7, 'dst_host_srv_count'), (8, 'num_failed_logins'), (9, 'service_imap4'), (10, 'is_guest_login'), (11, 'service_ft
p'), (12, 'flag_RSTO'), (13, 'service_http')]
```

In [ ]:
```
Y_U2R=Y_U2R.astype('int')
X_newU2R=X_newU2R.astype('int')
rfe.fit(X_newU2R, Y_U2R)
print ("U2R Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newcolname_U2R)))
```

```
U2R Features sorted by their rank:
[(1, 'hot'), (2, 'root_shell'), (3, 'num_shells'), (4, 'service_ftp_data'), (5, 'dst_host_count'), (6, 'dst_host_same_src_port_rat
e'), (7, 'dst_host_srv_count'), (8, 'num_file_creations'), (9, 'dst_host_srv_diff_host_rate'), (10, 'service_telnet'), (11, 'urgen
t'), (12, 'srv_diff_host_rate'), (13, 'service_http')]
```

2. Recursive Feature Elimination, we select 13 features each of 122

In [ ]:
```
from sklearn.feature_selection import RFE
clf = DecisionTreeClassifier(random_state=0)
rfe = RFE(estimator=clf, n_features_to_select=13, step=1)
Y_DoS=Y_DoS.astype('int')
rfe.fit(X_DoS, Y_DoS)
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 [ ]:
```
Y_Probe=Y_Probe.astype('int')
rfe.fit(X_Probe, Y_Probe)
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 [ ]:
```
Y_R2L=Y_R2L.astype('int')
rfe.fit(X_R2L, Y_R2L)
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 [ ]:
```
Y_U2R=Y_U2R.astype('int')
rfe.fit(X_U2R, Y_U2R)
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)
```

Summary of the features selected by RFE

In [ ]:
```
print('Features selected for DoS:',rfecolname_DoS)
print()
print('Features selected for Probe:',rfecolname_Probe)
print()
print('Features selected for R2L:',rfecolname_R2L)
print()
print('Features selected for U2R:',rfecolname_U2R)
```

```
Features selected for DoS: ['src_bytes', 'dst_bytes', 'wrong_fragment', 'num_compromised', 'same_srv_rate', 'diff_srv_rate', 'dst_h
ost_count', 'dst_host_same_srv_rate', 'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'service_ecr_i', 'flag_RSTR', 'flag_S0']

Features selected for Probe: ['src_bytes', 'dst_bytes', 'rerror_rate', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate', 'dst_hos
t_same_src_port_rate', 'dst_host_rerror_rate', 'service_finger', 'service_ftp_data', 'service_http', 'service_private', 'service_sm
tp', 'service_telnet']

Features selected for R2L: ['duration', 'src_bytes', 'dst_bytes', 'hot', 'num_failed_logins', 'num_access_files', 'dst_host_count',
'dst_host_srv_count', 'dst_host_same_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'service_ftp_data',
'service_imap4']

Features selected for U2R: ['duration', 'src_bytes', 'dst_bytes', 'hot', 'root_shell', 'num_file_creations', 'num_shells', 'srv_cou
nt', 'dst_host_count', 'dst_host_same_srv_rate', 'dst_host_srv_diff_host_rate', 'service_ftp_data', 'service_other']
```

In [ ]:
```
print(X_rfeDoS.shape)
print(X_rfeProbe.shape)
print(X_rfeR2L.shape)
print(X_rfeU2R.shape)
```

```
(113270, 13)
(78999, 13)
(68338, 13)
(67395, 13)
```

Step 4: Build the model

In [ ]:
```
# all features
clf_DoS=DecisionTreeClassifier(random_state=0)
clf_Probe=DecisionTreeClassifier(random_state=0)
clf_R2L=DecisionTreeClassifier(random_state=0)
clf_U2R=DecisionTreeClassifier(random_state=0)
clf_DoS.fit(X_DoS, Y_DoS)
clf_Probe.fit(X_Probe, Y_Probe)
clf_R2L.fit(X_R2L, Y_R2L)
clf_U2R.fit(X_U2R, Y_U2R)
```

Out[42]:
```
DecisionTreeClassifier(random_state=0)
```

In [ ]:
```
# selected features
clf_rfeDoS=DecisionTreeClassifier(random_state=0)
clf_rfeProbe=DecisionTreeClassifier(random_state=0)
clf_rfeR2L=DecisionTreeClassifier(random_state=0)
clf_rfeU2R=DecisionTreeClassifier(random_state=0)
clf_rfeDoS.fit(X_rfeDoS, Y_DoS)
clf_rfeProbe.fit(X_rfeProbe, Y_Probe)
clf_rfeR2L.fit(X_rfeR2L, Y_R2L)
clf_rfeU2R.fit(X_rfeU2R, Y_U2R)
```

Out[43]:
```
DecisionTreeClassifier(random_state=0)
```

Step 5: Prediction & Evaluation (validation)

Confusion Matrices

DoS

FUTURE WORK

In future we would like to explore other alternative feature selection like PCA or mutual information to see if they provide better performance. Another area we would like to experiment on different classifiers like perceptron, logistic regression, Naive bayes, K-nearest neighbors etc and also incorporating other additional attacks to make our proposed intrusion detection model more comprehensive in the future. And also including Ensemble models such as Adaboost or bagging may offer improved performance compared to single classifiers for future work. Additionally delving into deep learning methods, such as CNN and RNN could lead to future enhancements in the intrusion detection system's performance.

CONCLUSION

We have made an intrusion detection system using the decision tree machine learning model classifier to classify a particular attack and detect the attack. Preprocessing on the data has been done so as to decrease the computation time and analyze only the futures. Many network parameters like srv_count, etc have been taken into account which makes our IDS system to detect attacks more easily and effectively. The confusion matrices have been calculated which show that there are a very low number of false positives and very high number of true positives. Also the evaluation measures like accuracy and precision etc have been calculated and the accuracy was found to be around 99% which is very good and better than many of the other IDS systems that exist.

REFERENCES

[1] Kocher, G., & Kumar , G. (2021, June 24). *Machine learning and deep learning methods for intrusion detection systems: recent developments and challenges* . springer.

[2] Agrawal S, Sarkar S, Aouedi O, Yenduri G, Piamrat K, Alazab M, Bhattacharya S, Maddikunta PKR, Gadekallu TR. *Federated Learning for intrusion detection system: Concepts, challenges and future directions*. 2022. ResearchGate.

[3] Yousif N. *Accuracy of Machine Learning Algorithms in Detecting DoS Attacks Type*. 2017.

[4] Wang. *An effective intrusion detection framework based on SVM with feature augmentation*. 2017

[5] Yin J, Yang Z, Chen H, Li P, Wang J. *A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks*. 2017

[6] Ikram N, Cherukuri AK. *Intrusion detection model using fusion of chisquare feature selection and multi class SVM*. 2017.

[7] Jabbar S, Samreen F. *Intelligent network intrusion detection using alternating decision trees*. 2016.

[8] Peppes N, Daskalakis E, Alexakis T, Adamopoulou E, Demestichas K. *Performance of Machine Learning-Based Multi-Model Voting Ensemble Methods for Network Threat Detection in Agriculture 4.0*. 2021.