Network Intrusion Detection System Using Single Level Multi-Model Decision Trees

By

Jatin Kumar (18BCB0072)

Sai Subramanium (18BCB0069)

Shaurya Choudhary (18BCE2113)

School of Computer Science Engineering (SCOPE)

VIT®

Vellore Institute of Technology

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

1. Abstract:

Intrusion detection has become a major concern in the field of network security and administration. Considering intrusion as a security threat, a network needs a system which protects it from known vulnerabilities and unknown vulnerabilities for efficient functioning of the network. So we are developing an Intrusion detection system which is accurate upto some extent in detecting attacks with a possible minimum number of false positives.

2. Objective and Motivation:

Intrusion is considered to compromise the integrity, confidentiality, or availability of valuable assets on the computer systems. An intrusion detection system (IDS) audits the traffic flowing in the network for suspicious activity. It then alerts the system administrator when any malicious activity is discovered in the network. The primary functions of intrusion detection systems are discovering anomalies and producing detailed reports for the intrusions discovered. An IDS is a programmable software that is developed to detect intrusions within the network. It is employed to hunt and pinpoint the intruders causing chaos within the network. The main principles of IDS are integrity, availability, confidentiality and accountability. An IDS is built using both software and hardware. It can detect highly dangerous intrusions within the network. The main purpose of IDS is to detect unauthorized packets and malicious communications that happen in computer systems and networks. The most vital ingredient for the success of intrusion detection systems is feature selection.

3. Introduction:

3.1 Purpose and Scope:

Intrusions are considered as a sequence of steps taken to compromise the integrity, confidentiality, or availability of valuable assets on the computer systems. Intruders gain unauthorized access to the resources available on the system. They use all kinds of techniques to gain access to confidential information and manipulate the data available on the system. This can sometimes damage the system and render it worthless. An **IDS** can be considered to be a blend of software and hardware units that can be used to identify and pinpoint unauthorized experiments to gain access to the network. All the network related activities can be audited by an IDS which in turn can be used to suspect the traces of invasions within the network.

The end goal of an IDS is to trigger alerts when a suspicious activity has occurred by notifying the System Administrator. Intrusion detection techniques can be classified into two types:

3.1.1 Anomaly Detection: In this kind of detection the system alerts malicious tasks by identifying deviations i.e how differently are the network activities occurring as compared to regular patterns.
3.1.1Misuse Detection: In this kind of system intrusions are detected on the basis of already known patterns i.e. previously occurred malicious activity. This method can be used to identify and pinpoint known attack patterns more accurately. An ideal IDS will monitor all the happenings within the network and then decide whether those happenings are malicious or normal. The decision is based on system availability, confidentiality and integrity of the information resources.

An Intrusion Detection System works in the following manner: Collecting Data, Selecting Features, Analysing the Data, and the Actions to be Performed.

a. Collecting Data: We need to gather reports on the traffic flowing in the network like hosts alive, protocols used and the various forms of traffic flowing.

b. Selecting Features: After collecting a huge amount of data, the next step is to pick all those required features which we want to work upon.

c. Analysing the Data: In this step the data about the features which are selected data is evaluated to help us determine if the data is unnatural or not.

d. Actions to be Performed: When a malicious attack has taken place the system administrator is alarmed or notified by the IDS. The details about the type of attack are also provided by the IDS. The IDS closes the unnecessary network ports and processes to further mitigate the attacks from happening.

4. Information Security Concepts used in our project are:

The four attack categories available within the NSL-KDD data set which we have taken are :

4.1 DOS:

This kind of attack leads to draining of the victims's resources and making it incapable in responding to legitimate requests. This is one of the 4 attack categories.
Ex: syn flooding. The suitable features from the dataset for this attack class are: **"serror_rate" and "flag_SF"**.

4.2 U2R(unauthorized access to local root privileges):

In this kind of attack, an attacker tries to obtain root/administrator privileges by taking advantage of some vulnerability within the victim's system. The attacker usually uses a traditional account to login into a victim's system. The suitable attributes from the dataset for this attack class are: **"root_shell", "service_http", and "dst_host_same_src_port_rate".**

4.3 Probing:

This kind of attack involves obtaining sensitive information present in the victim's computer/device. The suitable attributes for this attack class are: **"Protocol_type_icmp" and "dst_host_same_src_port_rate".**

4.4 R2L:

This kind of attack involves unapproved access of the victim's device by gaining root access where he/she can view the data within that device with root privileges and all this is done from a far off(remote) machine by the attacker. E.g. password bruteforce attack. The suitable features from the dataset for this attack class are:**"dst_bytes", "dst_host_srv_diff_host_rate", and "dst_host_same_src_port_rate"**.

5. Description of Our methodology:

The primary goal is to design a plan for detecting intrusions within the system with the least possible number of features within the dataset. Based on the data from previous papers published, we can tell that only a subdivision of features in the dataset are derivative to the Intrusion Detection System. We have to cut back the dimensionality of the dataset to build an improved classifier in a justifiable amount of time. The approach we are going to use has a total of 4 stages : In the first stage, we pick out the significant features for every class using feature selection.In the next we combine the various features, so that the final cluster of features are optimal and relevant for each attack class. The third stage is for building a classifier. Here, the optimal features found in the previous stage are sent as input into the classifier. In the last stage, we test the model by employing a test dataset.

6. Modules:

6.1 Feature selection:

Here we will be using Information Gain (IG) to select the subset of relevant features in this project. Information Gain often costs less and is faster. We calculate Information gain for all the attributes present in the training dataset. It is calculated for each class separately. In the next step the values of the information gain are ranked i.e. the feature with the highest information gain being at rank 1. It means that this particular feature can distinctively classify for the particular class. If the value of the Information gain is less than the fixed threshold value for a particular feature, that feature can be eliminated from the feature space.

Stage 1: We divide the training dataset into 4 datasets. The training dataset is divided into 4 datasets in such a way that each dataset consists of records belonging to the **same attack class** along with some of the records of the original dataset. This stage is performed so that the feature selection method is unbiased while selecting features for frequently occurring attacks in the dataset.

Stage 2: In this stage the datasets for each attack class are sent separately as input into the method used to calculate the information gain. The output of this method gives us the most significant features for each attack type

Stage 3: In the third stage we generate a list of ranked features for each attack class. Now we eliminate all the irrelevant features from the list in accordance with the fixed threshold values.

6.2 Combining the optimal features:

In this stage we combine the list of features generated for each attack into a single list . For some of the attack classes the highest ranks i.e. the top 4 features chosen for classification. But for some types of attack classes we can only take 1 feature since that particular feature is at the top of the rank table and the remaining features are at the very bottom of the table. So, the final set of combined optimal features can be used to entirely distinguish the attack types.

6.3 Building a classifier:

A Decision tree is an algorithm which takes decisions at each node of the tree and is widely used for regression and classification. It is a supervised learning algorithm in Machine learning where which attribute should be at which node is learnt by using a set of labeled examples. The main advantage in using decision trees is that they can be trained very easily and they can even classify non linear data. It is more productive than most of the classification algorithms in ML like K-Nearest Neighbours in most of the cases. The common measures used to select attributes at each node in Decision trees are Info gain and Gain ratio.

6.4 Evaluation:

We test the model by employing a test dataset.

6.5 Architecture:

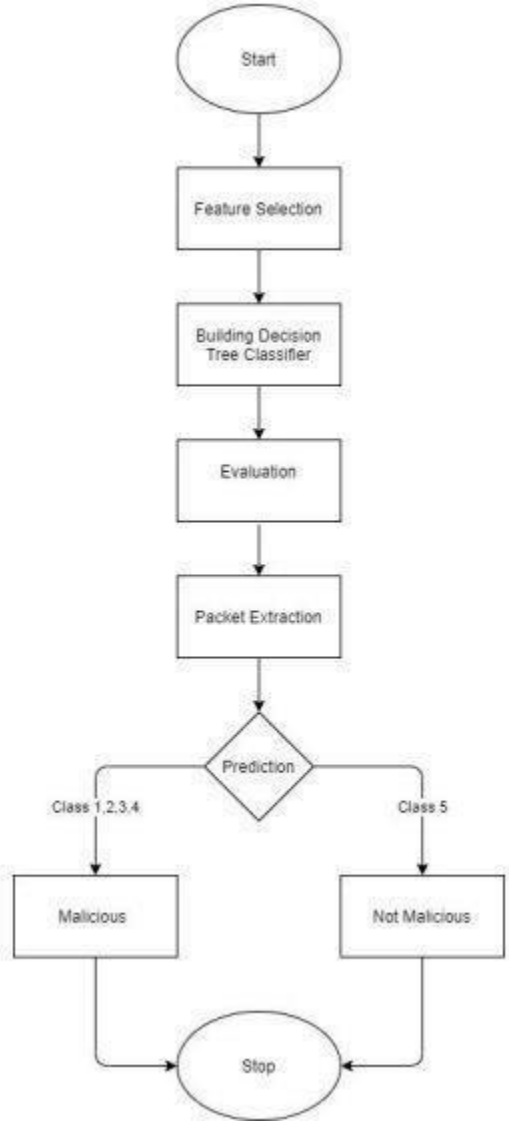

Fig 1.1

7. Code:

7.1 Data Preprocessing

7.1.1 Define Column Names

All features are made numerical using one-Hot-encoding. The features are scaled to avoid features with large values that may weigh too much in the results.

Importing neccessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import sys
import sklearn

from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn import preprocessing
from sklearn.feature_selection import SelectPercentile, f_classif
```

Define Column Names for the Dataset

In [2]:
```
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"]
```

Import Dataset and Check Dimensions

In [3]:
```
df_train = pd.read_csv("../Data/KDDTrain+_2.csv", header=None, names = col_names)
df_test = pd.read_csv("../Data/KDDTest+_2.csv", header=None, names = col_names)

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

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

Check Dataframe and description

In [4]:
```
df_train.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_c
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

7.1.2 Analyzing label distribution of training and training dataset

```
In [7]: df_test.describe()
```

Out[7]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromi
count	22544.000000	2.254400e+04	2.254400e+04	22544.000000	22544.000000	22544.000000	22544.000000	22544.000000	22544.000000	22544.000
mean	218.859076	1.039545e+04	2.056019e+03	0.000311	0.008428	0.000710	0.105394	0.021647	0.442202	0.119
std	1407.176612	4.727864e+05	2.121930e+04	0.017619	0.142599	0.036473	0.928428	0.150328	0.496659	7.269
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
50%	0.000000	5.400000e+01	4.600000e+01	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
75%	0.000000	2.870000e+02	6.010000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000
max	57715.000000	6.282565e+07	1.345927e+08	1.000000	3.000000	3.000000	101.000000	4.000000	1.000000	796.000

8 rows × 38 columns

Analysing Label Distribution of Training and Testing Dataset

```
In [8]: print('Label distribution of Training set:')
        print(df_train['label'].value_counts())
```

```
Label distribution of 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
```

```
In [5]: df_train.describe()
```

Out[5]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_comp
count	125973.00000	1.259730e+05	1.259730e+05	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	12597
mean	287.14465	4.556674e+04	1.977911e+04	0.000198	0.022687	0.000111	0.204409	0.001222	0.395736	
std	2604.51531	5.870331e+06	4.021269e+06	0.014086	0.253530	0.014366	2.149968	0.045239	0.489010	2
min	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.00000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	0.00000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	
max	42908.00000	1.379964e+09	1.309937e+09	1.000000	3.000000	3.000000	77.000000	5.000000	1.000000	747

8 rows × 38 columns

```
In [6]: df_test.head(5)
```

Out[6]:

	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
0	0	tcp	private	REJ	0	0	0	0	0	0	...	10	0.04	
1	0	tcp	private	REJ	0	0	0	0	0	0	...	1	0.00	
2	2	tcp	ftp_data	SF	12983	0	0	0	0	0	...	86	0.61	
3	0	icmp	eco_i	SF	20	0	0	0	0	0	...	57	1.00	
4	1	tcp	telnet	RSTO	0	15	0	0	0	0	...	86	0.31	

5 rows × 42 columns

```
In [7]: df_test.describe()
```

Out[7]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromi
count	22544.000000	2.254400e+04	2.254400e+04	22544.000000	22544.000000	22544.000000	22544.000000	22544.000000	22544.000000	22544.000
mean	218.859076	1.039545e+04	2.056019e+03	0.000311	0.008428	0.000710	0.105394	0.021647	0.442202	0.119

```
In [9]: print('Label distribution of Testing set:')
        print(df_test['label'].value_counts())
```

```
Label distribution of Testing 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
xterm                13
rootkit              13
teardrop             12
xlock                 9
land                  7
xsnoop                4
ftp_write             3
loadmodule            2
perl                  2
sqlattack             2
worm                  2
udpstorm              2
phf                   2
imap                  1
Name: label, dtype: int64
```

7.1.3 Features Description

Features Description

```
For Training Dataset
```

```
In [10]: print('Training Dataset:')
         for col_name in df_train.columns:
             if df_train[col_name].dtypes == 'object' :
                 unique_cat = len(df_train[col_name].unique())
                 print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))
```

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

```
In [11]: print('Distribution of categories in service:')
         print(df_train['service'].value_counts().sort_values(ascending=False).head())
```

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

```
For Testing Dataset
```

```
In [12]: print('Testing Dataset:')
         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))
```

```
Testing Dataset:
Feature 'protocol_type' has 3 categories
Feature 'service' has 64 categories
Feature 'flag' has 11 categories
Feature 'label' has 38 categories
```

7.1.4 Dataset column Manipulation

Dataset Column Manipulation

In [13]:
```
categorical_columns=['protocol_type', 'service', 'flag']

df_categorical_values = df_train[categorical_columns]
testdf_categorical_values = df_test[categorical_columns]
df_categorical_values.head()
```

Out[13]:

	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

Assign column names to dummy

In [14]:
```
# protocol_type
unique_protocol=sorted(df_train.protocol_type.unique())
string1 = 'Protocol_type_'
unique_protocol2=[string1 + x for x in unique_protocol]

# service
unique_service=sorted(df_train.service.unique())
string2 = 'service_'
unique_service2=[string2 + x for x in unique_service]

# flag
unique_flag=sorted(df_train.flag.unique())
string3 = 'flag_'
unique_flag2=[string3 + x for x in unique_flag]
```

Merge Dummy Categories

In [15]:
```
dumcols=unique_protocol2 + unique_service2 + unique_flag2
print(len(dumcols), end="\n\n")
print(dumcols)
```

7.1.5 Add missing categories to training dataset

Encode categorical features

In [18]:
```
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)

testdf_categorical_values_encenc = enc.fit_transform(testdf_categorical_values_enc)
testdf_cat_data = pd.DataFrame(testdf_categorical_values_encenc.toarray(),columns=testdumcols)
```

In [19]:
```
df_cat_data.head()
```

Out[19]:

	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

Add Missing Categories to Testing Dataset

In [20]:
```
trainservice=df_train['service'].tolist()
testservice= df_test['service'].tolist()
difference=list(set(trainservice) - set(testservice))
string = 'service_'
difference=[string + x for x in difference]
difference
```

Out[20]:
```
['service_urh_i',
 'service_http_2784',
 'service_red_i',
 'service_http_8001',
 'service_aol',
 'service_harvest']
```

```
 'service_aol',
 'service_harvest']
```

In [21]:
```
for col in difference:
    testdf_cat_data[col] = 0

testdf_cat_data.shape
```

Out[21]:
```
(22544, 84)
```

Join Encoded Categorical Dataframe

In [22]:
```
newdf=df_train.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)
```

In [23]:
```
newdf.head(5)
```

Out[23]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	flag_REJ	flag_RSTO	flag_RSTOS0
0	0	491	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0
1	0	146	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0
2	0	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0
3	0	232	8153	0	0	0	0	0	1	0	...	0.0	0.0	0.0
4	0	199	420	0	0	0	0	0	1	0	...	0.0	0.0	0.0

5 rows × 123 columns

7.1.6 Split dataset for different attacks

Split Dataset for different Attacks

Label:

- Normal : 0
- DoS : 1
- Probe : 2
- R2L : 3
- U2R : 4

In [25]:
```
# 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, 'apache2': 1, 'processtable': 1, 'udpstorm': 1, 'worm': 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': 3,'sendmail': 3,'named': 3,'snmpgetattack': 3,'snmpguess': 3,'xlock': 3,
                            'xsnoop': 3,'httptunnel': 3,
                            '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,
                            'mailbomb': 1, 'apache2': 1, 'processtable': 1, 'udpstorm': 1, 'worm': 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': 3,'sendmail': 3,'named': 3,'snmpgetattack': 3,'snmpguess': 3,'xlock': 3,
                            'xsnoop': 3,'httptunnel': 3,
                            '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
```

8. Feature Selection

8.1 Split dataset into X and Y

STAGE 2: FEATURE SELECTION

Eliminate redundant and irrelevant data by selecting a subset of relevant features that fully represents the given problem.

Split Dataset into X & Y

X: Dataframe of Feautures
Y: Series of Outcome Variables

In [29]:
```
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
```

```
For Test Dataset
```

In [30]:
```
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
```

In [31]:
```
colNames=list(X_DoS)
colNames_test=list(X_DoS_test)
```

Scaling the Dataframes

In [32]:
```
scaler1 = preprocessing.StandardScaler().fit(X_DoS)
X_DoS=scaler1.transform(X_DoS)
```

8.2 Scaling Dataframes

Scaling the Dataframes

In [32]:
```
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)

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)
```

Checking Standard Deviation

In [33]:
```
print(X_DoS.std(axis=0))
```

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

In [34]:
```
X_Probe.std(axis=0);
X_R2L.std(axis=0);
X_U2R.std(axis=0);
```

Univariate Feature Selection using ANOVA F-test

In [35]:
```
np.seterr(divide='ignore', invalid='ignore');
selector=SelectPercentile(f_classif, percentile=10)
```

8.3 Univariate Feature selection using ANOVA F-test

Univariate Feature Selection using ANOVA F-test

In [35]:
```
np.seterr(divide='ignore', invalid='ignore');
selector=SelectPercentile(f_classif, percentile=10)
X_newDoS = selector.fit_transform(X_DoS,Y_DoS)
X_newDoS.shape
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_selection.py:114: UserWarning: Features [ 16
44  63  66  68  86 114] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
```

Out[35]:
```
(113270, 13)
```

In [36]:
```
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[36]:
```
['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 [37]:
```
X_newProbe = selector.fit_transform(X_Probe,Y_Probe)
X_newProbe.shape
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_selection.py:114: UserWarning: Features [ 4 1
6] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
```

Out[37]:
```
(78999, 13)
```

REFERENCES:

NSL-KDD Dataset: NSL-KDD

Weblinks:

https://www.academia.edu/download/30701138/10.1.1.60.4079.pdf

https://ieeexplore.ieee.org/abstract/document/6511281/

http://www.iiard.com/index.php/IJCSMT/article/view/737

https://www.researchgate.net/publication/265187130_Effective_Network_Intrusion_Detection_using_Classifiers_Decision_Trees_and_Decision_rules

https://www.hindawi.com/journals/wcmc/2018/4680867/

https://ieeexplore.ieee.org/document/7029925

Journals:

1. Efficient classification mechanism for network intrusion detection system based on data mining techniques by A. S Subaira; P. Anitha Published in: 2014 IEEE 8th International Conference on Intelligent Systems and Control

2. Multi-Model Network Intrusion Detection System Using Distributed Feature Extraction and Supervised Learning by Sumeet Dua published in Spring 5-2020

3. An Efficient Intrusion Detection System with Convolutional Neural Network by V. Maheshwz

Reddy, I. Ravi Prakash Reddy, K. Adi Narayana Reddy in springer conference papers 30 april 2020

4. System and method for real-time insertion of data into a multi-dimensional database for network intrusion detection and vulnerability assessment by Robert Gleichauf, Steven Shaklin Published in 2001-08-28

5. Feature Selection and Deep Learning based Approach for Network Intrusion Detection by Jie Ling, Chengzhi Wu Published on April 2019.

6. Machine Learning Based research for network intrusion detection: A state-of-the-art by Kanubhai K. Patel, Bharat V. Buddhadev Published in June 2014