VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CSE3502
Information Security Management

Review 3

Detecting Network Intrusion using Multi-model Decision Trees

Group ID: 11

Group Members

1. Nikhitha Perapola - 19BDS0125
2. Gonella SL Sravya - 19BDS0140
3. Shakti Balaji - 19BDS0023

Faculty: Prof. Vidhya R

TABLE OF CONTENTS

1. PROBLEM STATEMENT

1. Idea

Intrusions are a series of actions taken to jeopardize the integrity, confidentiality, or availability of computer systems' valued assets. Unauthorized access to the system's resources is gained by intruders. They employ a variety of methods to get access to sensitive information and change data stored on the system. This can sometimes wreak havoc on the system, rendering it useless. The easiest way to deal with this problem is to use an IDS. It consists of a combination of software and hardware modules that can be used to detect and pinpoint unwanted network access experiments. An IDS can audit all network-related actions, which may then be utilized to suspect the presence of invasion traces within the network.

An IDS's ultimate purpose is to send out notifications to the System Administrator when suspicious activity is detected. There are two types of intrusion detection technologies:

a. Anomaly Detection: In this type of detection, the system detects malicious tasks by identifying deviations, or how the network activity diverges from normal patterns.

b. Misuse Detection: In this type of system intrusion, intrusions are recognised based on previously observed patterns, i.e. malevolent behavior. This strategy can be used to more precisely identify and localize known attack patterns.

2. Scope

An ideal IDS keeps track of everything that happens on the network and determines whether it's harmful or not. The selection is based on the information resources' availability, confidentiality, and integrity. Collecting data, selecting features, analysing the data, and performing actions are all duties that our Intrusion Detection System is built to do.

a. Data collection: information on the network's overall traffic flow must be gathered. Other details, such as the number of active hosts, the protocols utilised, and the various types of traffic flowing, must also be determined.

b. Feature Selection: Because our model uses four separate and distinct categories of assaults, Feature Selection will aid in the categorization of data while evaluating and utilising data in the IDS.
c. Data Analysis: We assess the data we've collected regarding our chosen features to see if it's unnatural or not.
d. Tasks to be Completed: The IDS alerts or notifies the system administrator when a malicious attack has occurred. The type of attack is also described in depth. To further reduce the risk of an attack, the IDS blocks unneeded network ports and processes.

3. Novelty

Most IDS systems use a single model for all four mentioned attacks. This is inconvenient and takes a lot more time, as different attacks utilize different features while following through with necessary steps. The existing methods show that most people try to find one perfect model for Network Intrusion Detection. They have used ID3, C4.5, and C5.0 decision tree algorithms on their datasets to find out which algorithm yields the best results. The use of decision trees in Network Intrusion Detection is encouraged due to its ease of pruning and feature selection. Hence, we too have chosen to work with decision trees. However, we believe that intrusion detection is not a “one size fits all” solution. That is why we have chosen to go with a multi-model approach to build separate decision tree classifiers for each type of attack - DOS, Probe, R2L, U2R. This will increase the accuracy of our model across all kinds of possible attacks

4. Comparative Statement:

1. **Implementation of network intrusion detection system using variant of decision tree algorithm ~ Relan, N.G.Patil, D.R.**

   The use of data mining techniques in intrusion detection is based on the concept of computerization. Machine learning refers to a machine's ability to enhance its performance over time by learning from prior experiences.

By uncovering previously unknown attack patterns, data mining aids in intrusion detection and management, as well as easing the route for IDS to become aware of new vulnerabilities and invasions. It is a logical procedure to use a decision tree. When it comes to extracting features and rules, it has a lot of advantages. As a result, decision tree techniques are applied in the field of intrusion detection. This research tries to use the C4.5 decision tree with a pruning approach to intrusion detection, as well as demonstrating that it works well.

In this paper, they looked at the performance of IDS using multiple decision tree algorithms, and they found that C5.0 performed better. Additional data mining research could aid in the refinement, automation, and simplification of this technology for industrial use.

2. **Intrusion Detection System Using Decision Tree Algorithm ~Manish Kumar Asst. Professor, Dr. M. Hanumanthappa Dr. T. V. Suresh Kumar - 2012**

   The article proposes a way for optimizing the matching process by employing a clustering algorithm to generate a decision tree that is directly generated from and tailored to the installed intrusion detection signatures. Using decision trees for the detection process, it is possible to quickly locate all firing rules (i.e., rules that match an input element) with a small number of comparisons. The rest of the chapter delves into the fundamentals of decision trees and how they apply to IDS.

   The research examines the results of the completed classification using the KDD99 dataset in order to assess the performance of a network intrusion detection method. The ID3, C4.5, and C5 algorithms' results were compared. In this paper, they looked at the performance of IDS using multiple decision tree algorithms, and they found that C5.0 performed better. Additional data mining research could aid in the refinement, automation, and simplification of this technology for industrial use.

3. **Deep Learning Approach for Intelligent Intrusion Detection System ~ R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat and S. Venkatraman**

   Previously proposed solutions took advantage of the system call's semantic meaning and exchanges. All of the information made public was limited to intrusions and cyberattacks via a Host-based intrusion mechanism. As a result, giving access to big data technology in the realm of cyber security, including IDS, is critical. Big data technological advancements make it easier to extract diverse patterns of legitimate and malicious actions from massive volumes of network and system activity data in a timely way, which helps IDS perform better.The majority of the findings in this study were based on the KDDCup99 dataset. The suggested architecture outperformed prior proposed architecture in terms of overall performance, and also collected network- and host-level activity in a distributed way utilizing DNNs to detect assaults more precisely. The suggested framework's performance can be improved further by including a module for monitoring DNS and BGP events in networks, adding more nodes, and using a distributed approach to train complicated DNN architectures on sophisticated hardware.

4. **An incremental decision tree algorithm based on rough sets and its application in intrusion detection ~ Feng Jiang, Yuefei Sui, Cungen Cao**

   Because most existing decision tree algorithms generate decision trees using all attributes and ignore the relevancy and dependency among attributes, this may result in additional memory and computational overhead. This study proposes and applies an incremental decision tree algorithm based on rough sets called IDTRS to IDS. The importance of attributes and the dependency of attributes in rough sets are used as heuristic information in IDTRS to choose splitting attributes. Due to the

fact that the classical rough set model can only cope with discretized attributes, all continuous attributes must be replaced with discretized attributes as a result of the discretization process. In the future, we intend to extend our approach to a neighborhood rough set that can handle both continuous and discrete features without the need for discretization.

5. **1. An intelligent algorithm with feature selection and decision rules applied to anomaly intrusion detection ~ Shih-Wei Lin, Kuo-Ching Ying, Chou-Yuan Lee, Zne-Jung Lee**

The data from the KDD Cup intrusion detection contest from 1999 is utilized in this paper to demonstrate the superiority of the suggested algorithm. The k-fold approach is employed in the proposed algorithm to evaluate classification accuracy for KDD '99 and output the optimal test accuracy and decision rules. The number k was set to ten in this investigation, implying that the data was divided into ten parts. The training data is retrieved in nine parts, while the testing data is retrieved in one. Various approaches are used to compare the simulation results, including the proposed algorithm, the hybrid process of DT, SA, and feature selection, the hybrid process of particle swarm optimization (PSO), SVM, and feature selection, the hybrid process of SA, SVM, and feature selection, the hybrid process of SA, SVM, and feature selection, the hybrid process of SA, SVM, and feature selection, the hybrid process of SA, SVM, and feature selection, the hybrid process of SA, SVM, and feature selection This research supports the advancement of anomalous intrusion detection. However, because it only used a single decision tree for all four proposed attack types, this system was ineffectual against the diverse features.

5. Dataset

NSL-KDD is a new version of the KDD ‘99 dataset. This is an effective benchmark dataset to help researchers compare different intrusion detection methods. This dataset fits perfectly with the goals we have chosen for our IDS, and has a plethora of features that can be scaled down according to the category of attack we are looking to experiment with.

6. Test Bed

- Jupyter Notebook (Python)
- Google Colab Notebook
- Numpy
- Pandas
- sklearn

7. Expected Result

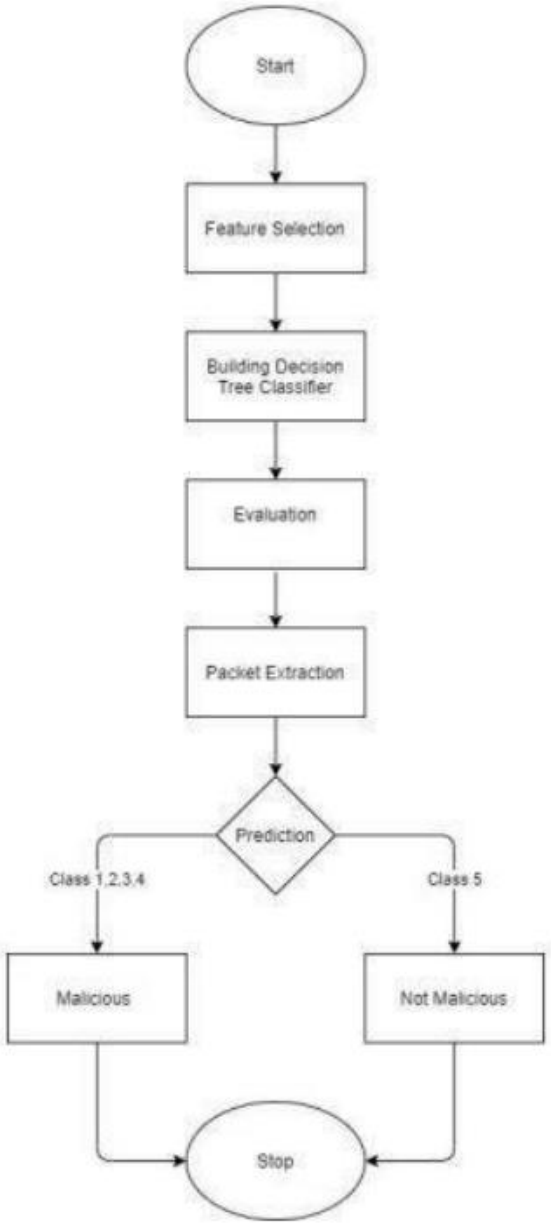
The primary functions of intrusion detection systems are discovering anomalies and producing detailed reports for the intrusions discovered. An IDS is a programmable software that is developed to detect intrusions within the network. It is employed to hunt and pinpoint the intruders causing chaos within the network.

The expected results from this project is proving the functionality of IDS, that are are integrity, availability, confidentiality and accountability. An IDS is built using both software and hardware. It can detect highly dangerous intrusions within the network. This IDS is expected to detect unauthorized packets and malicious communications that happen in computer systems and networks with high precision, accuracy, recall and f- score. These are the factors on which this project can be evaluated. The most vital ingredient for the success of intrusion detection systems is feature selection.

2. ARCHITECTURE

1. High Level Design

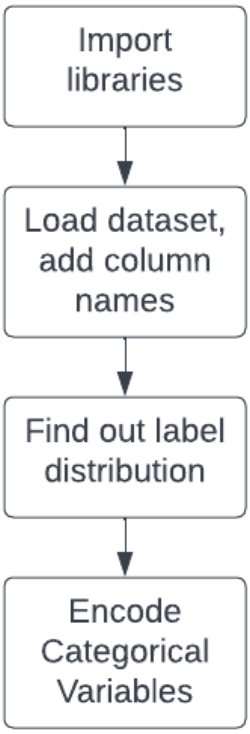
This diagram depicts the basic high-level steps involved in the project without going into detail about any of the processes involved.



2. Low Level Design

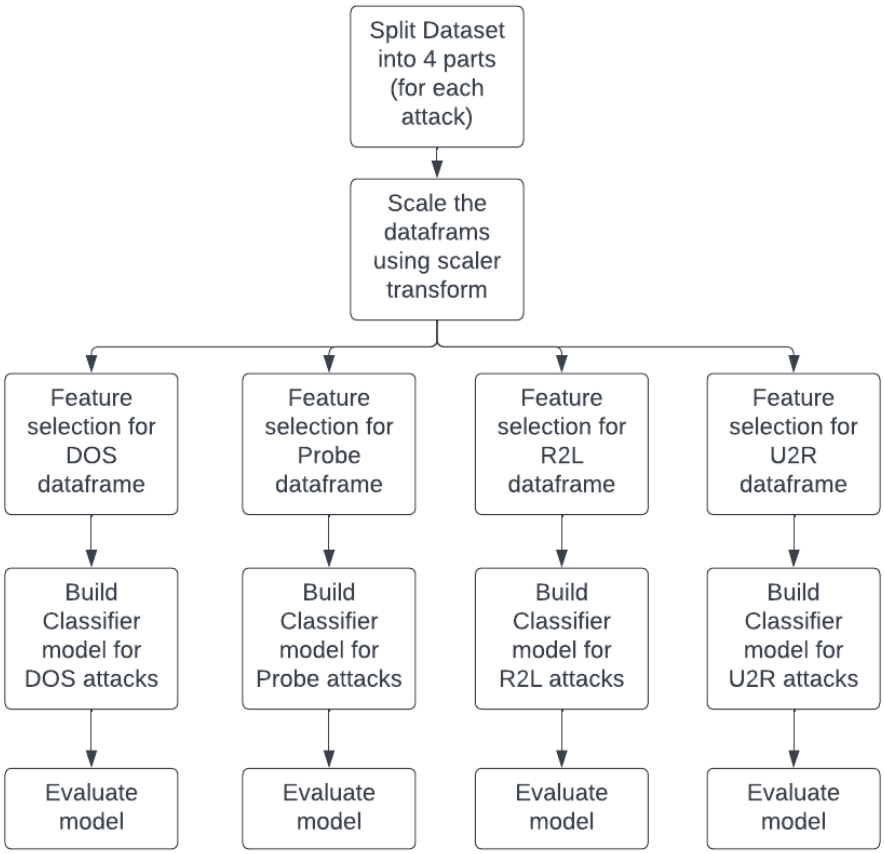
2.1 Low-level diagram for Pre-processing

This diagram involves the detailed steps involved in the preprocessing of the dataset, before the modelling can be done. It consists of importing the necessary libraries, loading the dataset and adding column names for it. Then, we move on to finding out the label distribution. This data can be used to figure out a possible class imbalance and give us greater insight into the frequency of each attack. Lastly, we encode the categorical values, to make them easier to work with.



2.2 Low-level diagram for Modelling

This diagram talks in detail about the process of multi-model decision trees. We first split the dataset into 4 parts, with each dataframe for one attack. We then scale the dataframes using scalar transform to ensure they are all in the same range. Now, we move on to feature selection using the RFE function. We then use these features to build 4 different classifier models, and then evaluate them.



3. IMPLEMENTATION

1. Algorithms followed

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 algorithm 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

2. Mathematical Model followed

For some of the attack classes, the highest ranks i.e. the top 4 features are chosen for classification. But for some types of attack classes we can only take one 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. 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 Neighbors in most of the cases. The common measures used to select attributes at each node in Decision trees are Info gain and Gain ratio. Thus, it makes for the best model to help support the design and architecture of our project.

4. RESULTS AND DISCUSSION

1. Implementation with coding

Import libraries

Version Check

```
[ ]  import pandas as pd
     import numpy as np
     import sys
     import sklearn
     from IPython.display import Image
     from sklearn import tree
     import pydotplus
     print(pd.__version__)
     print(np.__version__)
     print(sys.version)
     print(sklearn.__version__)
```

```
1.3.5
1.21.5
3.7.12 (default, Jan 15 2022, 18:48:18)
[GCC 7.5.0]
1.0.2
```

Double-click (or enter) to edit

Attach 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"]
```

```
[ ]  url = 'https://raw.githubusercontent.com//NIDS/master/KDDTrain%2B_2.csv'
     df = pd.read_csv(url, header=None, names = col_names)
     url = 'https://raw.githubusercontent.com//NIDS/master/KDDTest%2B_2.csv'
     df_test = pd.read_csv(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)
```

Load the Dataset

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

```
<bound method NDFrame.head of        duration protocol_type   service flag  src_bytes  dst_bytes  land  \
0             0           tcp  ftp_data   SF        491          0     0
1             0           udp     other   SF        146          0     0
2             0           tcp   private   S0          0          0     0
3             0           tcp      http   SF        232       8153     0
4             0           tcp      http   SF        199        420     0
...         ...           ...       ...  ...        ...        ...   ...
125968        0           tcp   private   S0          0          0     0
125969        8           udp   private   SF        105        145     0
125970        0           tcp      smtp   SF       2231        384     0
125971        0           tcp    klogin   S0          0          0     0
125972        0           tcp  ftp_data   SF        151          0     0

        wrong_fragment  urgent  hot  ...  dst_host_srv_count  \
0                    0       0    0  ...                  25
1                    0       0    0  ...                   1
2                    0       0    0  ...                  26
3                    0       0    0  ...                 255
4                    0       0    0  ...                 255
...                ...     ...  ...  ...                 ...
125968               0       0    0  ...                  25
125969               0       0    0  ...                 244
125970               0       0    0  ...                  30
125971               0       0    0  ...                   8
125972               0       0    0  ...                  77

        dst_host_same_srv_rate  dst_host_diff_srv_rate  \
0                         0.17                    0.03
1                         0.00                    0.60
2                         0.10                    0.05
3                         1.00                    0.00
4                         1.00                    0.00
```

Load the Dataset

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

```
125969             0      0   0  ...               244
125970             0      0   0  ...                30
125971             0      0   0  ...                 8
125972             0      0   0  ...                77

        dst_host_same_srv_rate  dst_host_diff_srv_rate  \
0                         0.17                    0.03
1                         0.00                    0.60
2                         0.10                    0.05
3                         1.00                    0.00
4                         1.00                    0.00
...                        ...                     ...
125968                    0.10                    0.06
125969                    0.96                    0.01
125970                    0.12                    0.06
125971                    0.03                    0.05
125972                    0.30                    0.03

        dst_host_same_src_port_rate  dst_host_srv_diff_host_rate  \
0                              0.17                         0.00
1                              0.88                         0.00
2                              0.00                         0.00
3                              0.03                         0.04
4                              0.00                         0.00
...                             ...                          ...
125968                         0.00                         0.00
125969                         0.01                         0.00
125970                         0.00                         0.00
125971                         0.00                         0.00
125972                         0.30                         0.00

        dst_host_serror_rate  dst_host_srv_serror_rate  dst_host_rerror_rate  \
```

Sample view of the training dataset

```
[ ]  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
```

```
[ ]  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
```

Label Distribution of the Training set and Test set

```
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))

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
```

Identifying categorical features

```
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
```

Insert the categorical features into a 2D numpy array

```
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()
```

	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

Make column names for dummies

```
# 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', 'service_auth', 'service_bgp', 'service_courier', 'service_csnet_ns', 'service
['Protocol_type_icmp', 'Protocol_type_tcp', 'Protocol_type_udp', 'service_IRC', 'service_X11', 'service_Z39_50', 'service_auth', 'service_bgp', 'service_courier', 'service_csnet_ns', 'service_ctf', 'service
```

Transform categorical features into numbers

```
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

```
[ ]  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()
```

	Protocol_type_icmp	Protocol_type_tcp	Protocol_type_udp	service_IRC	service_X11	service_Z39_50	service_aol	service_auth	service_bgp	service_courier	...	flag_REJ	flag_RSTO	flag_RSTOS0	flag_RSTR
0	0.0	1.0	0.0	0.0	0.0	0.0	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	...	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	...	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	...	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	...	0.0	0.0	0.0	0.0

5 rows × 84 columns

Add missing 6 categories from the training set into the test set

```
[ ]  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
```

```
['service_aol',
 'service_red_i',
 'service_http_2784',
 'service_harvest',
 'service_urh_i',
 'service_http_8001']
```

join the encoded categorical dataframe with the non-categorical dataframe

```
[ ]  for col in difference:
       testdf_cat_data[col] = 0
     testdf_cat_data.shape
```

```
(22544, 84)
```

```
[ ]  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)
```

```
[ ]  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)
```

Split Dataset into 4 datasets for every attack category

```
[ ]  # take label column
     label_df = newdf['label']
     label_df_test = newdf_test['label']
     # change the label column
     new_label_df = label_df.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})
     new_label_df_test = label_df_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'] = new_label_df
     newdf_test['label'] = new_label_df_test
```

```
                                                'perl': 4, 'rootkit': 4, 'ps': 4,
                                                'sqlattack': 4,'xterm': 4})
# put the new label column back
newdf['label'] = new_label_df
newdf_test['label'] = new_label_df_test
print(newdf['label'].head())
```

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

```
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)
```

```
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)
```

Feature Scaling

```
# 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
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  """
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  app.launch_new_instance()
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
```

Scale the dataframes

```
[ ]  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)
```

```
[ ]  print(X_DoS.shape)
     print(X_Probe.shape)
     print(X_R2L.shape)
     print(X_U2R.shape)
```

```
(113270, 122)
(78999, 122)
(68338, 122)
(67395, 122)
```

Feature Selection

```
[ ]  from sklearn.feature_selection import RFE
     from sklearn.tree import DecisionTreeClassifier
     classif = DecisionTreeClassifier(random_state=0)
     rfe = RFE(estimator=classif, 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)
```

```
[ ]  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)
```

```
[ ]  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)
```

```
[ ]  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 feature selection

```
[ ]  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_host_count', 'dst_host_same_srv_rate', 'dst_host_serror_rate', 'dst_host_sr

Features selected for Probe:  ['src_bytes', 'dst_bytes', 'rerror_rate', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_rerror_rate', 'service_finger', 'service_

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_r

Features selected for U2R:  ['duration', 'src_bytes', 'dst_bytes', 'hot', 'root_shell', 'num_file_creations', 'num_shells', 'srv_count', 'dst_host_count', 'dst_host_same_srv_rate', 'dst_host_srv_diff_host_r
```

Build in Model

```
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)
```

```
DecisionTreeClassifier(random_state=0)
```

```
[ ]  X_DoS_test2 = X_DoS_test[:, rfecolindex_DoS]
     X_Probe_test2 = X_Probe_test[:, rfecolindex_Probe]
     X_R2L_test2 = X_R2L_test[:, rfecolindex_R2L]
     X_U2R_test2 = X_U2R_test[:, rfecolindex_U2R]
```

2. Results

Result

DoS Attack

```
[ ]  Y_DoS_pred2 = clf_rfeDoS.predict(X_DoS_test2)
     # Create confusion matrix
     pd.crosstab(Y_DoS_test, Y_DoS_pred2, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	1
Actual attacks		
0	9602	109
1	2625	4835

Probe Attack

```
[ ]  Y_Probe_pred2 = clf_rfeProbe.predict(X_Probe_test2)
     # Create confusion matrix
     pd.crosstab(Y_Probe_test, Y_Probe_pred2, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	2
Actual attacks		
0	8709	1002
2	944	1477

R2L Attack

```
     Y_R2L_pred2 = clf_rfeR2L.predict(X_R2L_test2)
     # Create confusion matrix
     pd.crosstab(Y_R2L_test, Y_R2L_pred2, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Predicted attacks	0	3
Actual attacks		
0	9649	62
3	2560	325

U2R Attack

```
[ ]  Y_U2R_pred2 = clf_rfeU2R.predict(X_U2R_test2)
     # Create confusion matrix
     pd.crosstab(Y_U2R_test, Y_U2R_pred2, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Evaluation

DoS attack

```
[ ]  from sklearn.model_selection import cross_val_score
     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: 0.99738 (+/- 0.00267)
Precision: 0.99692 (+/- 0.00492)
Recall: 0.99705 (+/- 0.00356)
F-measure: 0.99698 (+/- 0.00307)
```

Probe Attack

```
[ ]  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: 0.99085 (+/- 0.00559)
Precision: 0.98674 (+/- 0.01179)
Recall: 0.98467 (+/- 0.01026)
F-measure: 0.98566 (+/- 0.00871)
```

R2L Attack

```
[ ]  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: 0.97459 (+/- 0.00910)
Precision: 0.96689 (+/- 0.01311)
Recall: 0.96086 (+/- 0.01571)
F-measure: 0.96379 (+/- 0.01305)
```

U2R Attack

```
[ ]  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: 0.99652 (+/- 0.00278)
Precision: 0.87538 (+/- 0.15433)
Recall: 0.89540 (+/- 0.14777)
F-measure: 0.87731 (+/- 0.09647)
```

3. Mapping the results with problem statement and existing systems

We tested the model by employing a test dataset. We obtained results for various evaluation metrics - Accuracy, Precision, Recall, F-measure:

Accuracy:

The accuracy (AC) is defined as the distribution of the total number of correct predictions. The equation is estimated as:

Accuracy = TP+TN/TP+TN+FP+FN

Precision:

Precision is defined as the ratio of the total number of true positives, the sum of the number of true positives and the number of false positives. It is calculated by the equation:

Precision = TP/TP+FP

Recall:

Recall can be defined as the proportion of the total number of positive examples rightly listed, divided by the total number of positive ones. High Recall suggests correct identification of class. It is also defined in scientific terms as Detection Frequency, True Positive Rate, or Sensitivity. The equation computes as:

Recall = TP/TP+FN

F-score:

The F score is also known as F1 score or F measure. It is a measure of the accuracy of a test. The F score is interpreted as the weighted harmonic mean of the test's precision and recall.

DOS ATTACK

Accuracy: 0.99738 (+/- 0.00267)
Precision: 0.99692 (+/- 0.00492)
Recall: 0.99705 (+/- 0.00356)
F-measure: 0.99698 (+/- 0.00307)

PROBE ATTACK

Accuracy: 0.99085 (+/- 0.00559)
Precision: 0.98674 (+/- 0.01179)
Recall: 0.98467 (+/- 0.01026)
F-measure: 0.98566 (+/- 0.00871)

R2L ATTACK

Accuracy: 0.97459 (+/- 0.00910)
Precision: 0.96689 (+/- 0.01311)
Recall: 0.96086 (+/- 0.01571)
F-measure: 0.96379 (+/- 0.01305)

U2R ATTACK

Accuracy: 0.99652 (+/- 0.00278)
Precision: 0.87538 (+/- 0.15433)
Recall: 0.89540 (+/- 0.14777)
F-measure: 0.87731 (+/- 0.09647)

4. Conclusion:

Almost every system nowadays is vulnerable to cyber-attacks. There is a pressing need to improve the security of our day-to-day systems. This is where an IDS comes in handy. It assists us in efficiently detecting harmful actions as well as anomalies. However, current software that detects anomalies generates a large number of false positives, resulting in a rise in the number of false alarms. Due to the same issue, the results of several additional experiments were severely erroneous. Using the NSL-KDD dataset, we ran an experiment to evaluate the accuracies and detection rates of several techniques. We were able to deduce from our findings that our strategy, Single Level Multimodel with Decision Trees, performed extraordinarily well and had very low false alarm rates. To choose our approach, we took into account criteria such as accuracy, precision, recall, and F-Measure.

5. REFERENCES

i. R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat and S. Venkatraman, "Deep Learning Approach for Intelligent Intrusion Detection System," in IEEE Access, vol. 7, pp. 41525-41550, 2019, doi: 10.1109/ACCESS.2019.2895334.

ii. Implementation of network intrusion detection system using variant of decision tree algorithm Relan, N.G.Patil, D.R.
https://ieeexplore.ieee.org.egateway.vit.ac.in/stamp/stamp.jsp?tp=&arnumber=7029925

iii. Intrusion Detection System Using Decision Tree Algorithm -
Manish Kumar Asst. Professor, Dr. M. Hanumanthappa Dr. T. V. Suresh Kumar - 2012
https://ieeexplore.ieee.org.egateway.vit.ac.in/stamp/stamp.jsp?tp=&arnumber=6511281

iv. An incremental decision tree algorithm based on rough sets and its application in intrusion detection

https://link.springer.com.egateway.vit.ac.in/content/pdf/10.1007%2Fs10462-011-9293-z.pdf

v. An intelligent algorithm with feature selection and decision rules applied to anomaly intrusion detection
https://www.sciencedirect.com.egateway.vit.ac.in/science/article/pii/S1568494612002402?via%3Dihub