INTRUSION DETECTION SYSTEM USING XGBOOST

Problem Statement - The task is to build a network intrusion detector, a predictive model capable of distinguishing between bad connections, called intrusions or attacks, and good normal connections.

What is Intrusion detection system?

Intrusion Detection System is a software application to detect network intrusion using various machine learning algorithms.IDS monitors a network or system for malicious activity and protects a computer network from unauthorized access from users, including perhaps insider.

Abstract -

The main aim of the project is to build a Intrusion Detection System using Xgboost . Xgboost is an Machine Learning algorithm. In this process we will be using different Machine Learning Algorithm like Support Vector Machine, Random Forest and Xgboost and then we will see that Xgboost is the best method to use for Intrusion detection system.

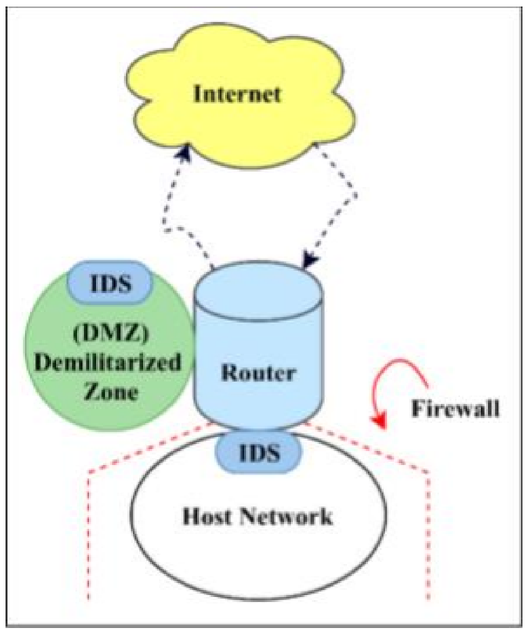
For this we also have to perform resampling of the data. For this i will be using 3 processes and then compare them and find which among of them is the best to use for the above ML algorithms.

OBJECTIVES

The main objective is to determine or detect an attack on the network . this model will use the previous data and predict the attack on the network by using ML algorithms . the main objective is to make the network more secure and detect the attacks and prevent the users data for being taken or manipulated by either blocking the intrusions or by any other means

Why IDS ?

The main idea that sits behind deploying IDS in a network is to stop attacks happening from outside and within the network. The findings in this paper can help in building a strong IDS, which can keep an eye on the data entering a network and simultaneously filter out suspicious entries. It is recommended that the IDS be deployed at two points. As there is a firewall protecting the host network or the private network, it is better to place the IDS behind the firewall,



as seen in Figure 1.

Thereby the work of the IDS is reduced and there will be a saving of resources, as the IDS can only tackle suspicious entries that were unable to be detected by the firewall. Figure 1. Recommended placement of Intrusion Detection System (IDS) in a network. The IDS deployed can work efficiently and look for suspicious activities within the network. The main attacks come from outside the host network, from the internet that is trying to send data to the host network. The main area where this issue can be tackled is the DMZ, which is a demilitarized zone (the servers that are responsible to connect the host network with the outside world). So, there can be an IDS that can be deployed endemic to the DMZ and this can help in eliminating the majority of the mischievous data trying to penetrate the firewall. For more security, a no-access policy should be assigned to the DMZ servers because, if the DMZ gets compromised, the host network remains safe

Description of System

An Intrusion Detection System is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered.

Dataset: The BoT-IoT dataset was created by designing a realistic network environment in the Cyber Range Lab of the center of UNSW Canberra Cyber. The environment incorporates a combination of normal and botnet traffic. The dataset's source files are provided in different formats, including the original pcap files, the generated argus files and csv files. The files were separated, based on attack category and subcategory, to better assist in labeling process.

Preprocessing: Data processing is the process of preparing raw data and making it suitable for the machine learning model. An important step in the development of a machine learning model. And while you do any data processing, it is imperative that you clean it up and set it in formatted form.

Normalization: Normalization is generally a method used in the preparation of machine learning model. The goal is to convert numerical column values in the inorder to use the same scale, without distorting the differences in the range of values and loss of information.

Sampling: Data sampling refers to statistical methods for selecting observations from the domain with the objective of estimating a population parameter. Oversampling and undersampling in data analysis are techniques used to adjust the class distribution of a data set (i.e. the ratio between the different classes/categories represented).

Cross-validation: It is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. Genarlly it is known as k-fold cross-validation.

Hyper Parameter Tuning: Choosing a set of optimal hyper parameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process.

Testing and Prediction: The model which will give best accuracy will be choosen for prediction. Precision, Recall and f-score are also calculated for knowing more about the particular algorithm.

The reasons XGBoost is picked to be the preferred classification model-

- XGBoost is approximately 10 times faster than existing methods on a single platform, therefore eliminating the issue of time consumption especially when pre-processing of the network data is done.
- XGBoost has the advantage of parallel processing, that is uses all the cores of the machine it is running on. It is highly scalable, generates billions of examples using distributed or parallel computation and algorithmic optimization operations, all using minimal resources. Therefore, it is highly effective in dealing with issues such as classification of data and high-level preprocessing of data.
- The portability of XGBoost makes it available and easier to blend on many platforms. Recently, the distributed versions are being integrated to cloud platforms such as Tianchi of Alibaba, AWS, GCE, Azure, and others. Therefore, flexibility offered by XGBoost is immense and is not tied to a specific platform, hence the IDS using XGBoost can be platform-independent, which is a major advantage. XGBoost is also interfaced with cloud data flow systems such as Spark and Flink.
- XGBoost can be handled by multiple programming languages such as Java, Python, R, C++.
- XGBoost allows the use of wide variety of computing environments such as parallelization (tree construction across multiple CPU Cores), Out of core computing, distributed computing for handling large models, and Cache Optimization for the efficient use of hardware.
- The ability of XGBoost to make a weak learner into a strong learner (boosting) through its optimization step for every new tree that attaches, allow the classification model to generate less False Alarms, easy labelling of data, and accurate classification of data.
- Regularization is an important aspect of XGBoost algorithm, as it helps in avoiding data overfitting problems whether it be tree based or linear models. XGBoost deals effectively with dataoverfitting problems, which can help to deal when a system is under DDoS attack, that is flooding of data entries, so the classifier is needed to be fast (which XGBoost is) and the classifier should be able to accommodate data entries.

- There is enabled cross-validation as an internal function. Therefore, there is no need of external packages to get cross validation results.

- XGBoost is well equipped to detect and deal with missing values.

- XGBoost is a flexible classifier as it gives the user the option to set the objective function as desired by setting the parameters of the model. It also supports user defined evaluation metrics in addition to dealing with regression, classification, and ranking problems.

- Availability of XGBoost at different platforms makes it easy to access and use.

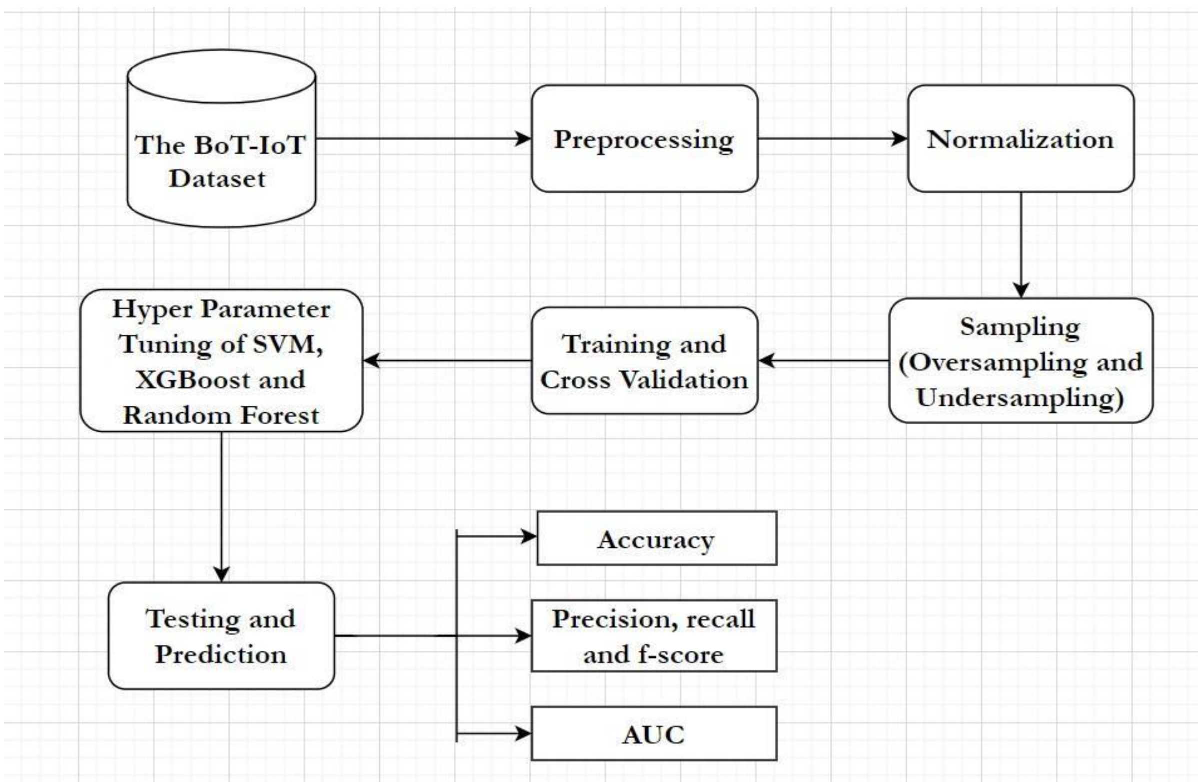
- Save and Reload functions are available, as XGBoost gives the option of saving the data matrix and relaunching it when required. This eliminates the need of extra memory space.

- Extended Tree Pruning, that is, in normal models the tree pruning stops as soon as a negative loss is encountered, but in XGBoost the Tree Pruning is done up to a maximum depth of tree as defined by the user and then backward pruning is performed on the same tree until the improvement in the loss function is below a set threshold value. All these important functionalities add up and enable the XGBoost to outperform many existing models.

Type of attacks detected -:

Attack class	DoS	Probe	R2L	U2R
Attack type	Back Land Neptune Pod Smurf Teardrop Apache2 Udp storm Process table Worm	Satan Ipsweep Nmap Portsweep Mscan Saint	Guess_Password Ftp_write Imap Phf Multihop Warezmaster Warezclient Spy Xlock Xsnoop Snmpguess Snmpgetattack Httptunnel Sendmail Named	Buffer_overflow Load module Rootkit Perl SQL attack X term Ps

Design of System



MODEL -:

Preprocessing -:

- ### Loading of dataset :

```
[ ] import pandas as pd
    import numpy as np
    import seaborn as sns
    import sklearn
    import warnings
    warnings.filterwarnings("ignore")
```

```
[ ] from google.colab import drive

    drive.mount('/content/gdrive')

    Mounted at /content/gdrive
```

```
[ ] data1=pd.read_csv('/content/gdrive/My Drive/Dataset/data1.csv')
```

```
data1.head()
```

	pkSeqID	stime	flgs	flgs_number	proto	proto_number	saddr	sport	daddr	dport	pkts	bytes	state	state_number	ltime	seq	dur	mean	stddev	sum
0	1	1.528089e+09	e	1	tcp	1	192.168.100.147	49960	192.168.100.7	80	8	1980	RST	1	1.528089e+09	9	7.056393	0.068909	0.068909	0.137818
1	2	1.528089e+09	e	1	arp	2	192.168.100.7	-1	192.168.100.147	-1	2	120	CON	2	1.528089e+09	10	0.000131	0.000131	0.000000	0.000131
2	3	1.528089e+09	e	1	tcp	1	192.168.100.147	49962	192.168.100.7	80	8	2126	RST	1	1.528089e+09	11	7.047852	0.064494	0.064494	0.128988
3	4	1.528089e+09	e	1	tcp	1	192.168.100.147	49964	192.168.100.7	80	8	2024	RST	1	1.528089e+09	12	7.047592	0.064189	0.064189	0.128378
4	5	1.528089e+09	e	1	tcp	1	192.168.100.147	49966	192.168.100.7	80	8	2319	RST	1	1.528089e+09	13	7.046841	0.063887	0.063887	0.127774

```
[ ] data2 = pd.read_csv("/content/gdrive/My Drive/Dataset/UNSW_2018_IoT_Botnet_Full5pc_3.csv")
```

```
data2.head()
```

	pkSeqID	stime	flgs	flgs_number	proto	proto_number	saddr	sport	daddr	dport	pkts	bytes	state	state_number	ltime	seq	dur	mean	stddev	su
0	2000001	1.528096e+09	e s	2	tcp	1	192.168.100.148	64480	192.168.100.3	80	5	770	REQ	3	1.528096e+09	86607	14.713629	2.669374	1.924042	8.00812
1	2000002	1.528096e+09	e s	2	tcp	1	192.168.100.148	64481	192.168.100.3	80	5	770	REQ	3	1.528096e+09	86608	14.713629	2.669375	1.924043	8.00812
2	2000003	1.528096e+09	e s	2	tcp	1	192.168.100.148	64484	192.168.100.3	80	5	770	REQ	3	1.528096e+09	86609	14.713628	2.669375	1.924043	8.00812
3	2000004	1.528096e+09	e s	2	tcp	1	192.168.100.148	64485	192.168.100.3	80	5	770	REQ	3	1.528096e+09	86610	14.713627	2.669375	1.924043	8.00812
4	2000005	1.528096e+09	e s	2	tcp	1	192.168.100.148	64490	192.168.100.3	80	5	770	REQ	3	1.528096e+09	86611	14.713627	2.674472	1.926380	8.02341

```
[ ] data1.shape

    (1000000, 46)
```

```
[ ] data2.shape

    (1000000, 46)
```

- ### Adding two dataset

```
[ ] data = data1.append(data2)
```

```
[ ] data.shape

    (2000000, 46)
```

```
data.head()
```

	pkSeqID	stime	flgs	flgs_number	proto	proto_number	saddr	sport	daddr	dport	pkts	bytes	state	state_number	ltime	seq	dur	mean	stddev	sum
0	1	1.528089e+09	e	1	tcp	1	192.168.100.147	49960	192.168.100.7	80	8	1980	RST	1	1.528089e+09	9	7.056393	0.068909	0.068909	0.137818
1	2	1.528089e+09	e	1	arp	2	192.168.100.7	-1	192.168.100.147	-1	2	120	CON	2	1.528089e+09	10	0.000131	0.000131	0.000000	0.000131
2	3	1.528089e+09	e	1	tcp	1	192.168.100.147	49962	192.168.100.7	80	8	2126	RST	1	1.528089e+09	11	7.047852	0.064494	0.064494	0.128988
3	4	1.528089e+09	e	1	tcp	1	192.168.100.147	49964	192.168.100.7	80	8	2024	RST	1	1.528089e+09	12	7.047592	0.064189	0.064189	0.128378
4	5	1.528089e+09	e	1	tcp	1	192.168.100.147	49966	192.168.100.7	80	8	2319	RST	1	1.528089e+09	13	7.046841	0.063887	0.063887	0.127774

- **Checking the database type and checking for NULL values**

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2000000 entries, 0 to 999999
Data columns (total 46 columns):
 #   Column                            Dtype
---  ------                            -----
 0   pkSeqID                           int64
 1   stime                             float64
 2   flgs                              object
 3   flgs_number                       int64
 4   proto                             object
 5   proto_number                      int64
 6   saddr                             object
 7   sport                             object
 8   daddr                             object
 9   dport                             object
 10  pkts                              int64
 11  bytes                             int64
 12  state                             object
 13  state_number                      int64
 14  ltime                             float64
 15  seq                               int64
 16  dur                               float64
 17  mean                              float64
 18  stddev                            float64
 19  sum                               float64
 20  min                               float64
 21  max                               float64
 22  spkts                             int64
 23  dpkts                             int64
 24  sbytes                            int64
 25  dbytes                            int64
 26  rate                              float64
 27  srate                             float64
 28  drate                             float64
 29  TnBPSrcIP                         int64
```

```
data.isnull().sum()
```

```
pkSeqID                  0
stime                    0
flgs                     0
flgs_number              0
proto                    0
proto_number             0
saddr                    0
sport                    0
daddr                    0
dport                    0
pkts                     0
bytes                    0
state                    0
state_number             0
ltime                    0
seq                      0
dur                      0
mean                     0
stddev                   0
sum                      0
min                      0
max                      0
spkts                    0
dpkts                    0
sbytes                   0
dbytes                   0
rate                     0
srate                    0
drate                    0
TnBPSrcIP                0
TnBPDstIP                0
TnP_PSrcIP               0
TnP_PDstIP               0
TnP_PerProto             0
TnP_Per_Dport            0
```

- **Dropping useless columns and checking for columns having object datatype-:**

```
[ ] data.drop(["pkSeqID","flgs","proto","state","attack"],axis=1,inplace=True)
```

```
[ ] data.shape

(2000000, 41)
```

```
data.dtypes[data.dtypes=='object']

saddr          object
sport          object
daddr          object
dport          object
category       object
subcategory    object
dtype: object
```

- **Replacing HEX values with INT values and converting the datatypes to support format using Label Encoder-:**

```
[ ] data['dport']=data['dport'].replace(['0x5000'],'20480')
    data['dport']=data['dport'].replace(['0x0303'],'771')
    data['sport']=data['sport'].replace(['0x5000'],'20480')
    data['sport']=data['sport'].replace(['0x0303'],'771')
    data["dport"] = data["dport"].astype(str).astype(int)
    data["sport"] = data["sport"].astype(str).astype(int)
```

```
data.dtypes[data.dtypes=='object']

saddr          object
daddr          object
category       object
subcategory    object
dtype: object
```

```
[ ] from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()
    data["saddr_enc"] = le.fit_transform(data.saddr)
    data["daddr_enc"] = le.fit_transform(data.daddr)
    data["category_enc"]= le.fit_transform(data.category)
    data["subcategory_enc"]= le.fit_transform(data.subcategory)
    data.drop(['saddr','daddr','category','subcategory'],axis=1,inplace=True)
```

- **Final data**

```
data.head()
```

	stime	flgs_number	proto_number	sport	dport	pkts	bytes	state_number	ltime	seq	dur	mean	stddev	sum	min	max	spkts	dpkts	sbytes	dbytes	rat
0	1.528089e+09	1	1	49960	80	8	1980	1	1.528089e+09	9	7.056393	0.068909	0.068909	0.137818	0.000000	0.137818	5	3	650	1330	0.99200
1	1.528089e+09	1	2	-1	-1	2	120	2	1.528089e+09	10	0.000131	0.000131	0.000000	0.000131	0.000131	0.000131	1	1	60	60	7633.58837
2	1.528089e+09	1	1	49962	80	8	2126	1	1.528089e+09	11	7.047852	0.064494	0.064494	0.128988	0.000000	0.128988	5	3	796	1330	0.99321
3	1.528089e+09	1	1	49964	80	8	2024	1	1.528089e+09	12	7.047592	0.064189	0.064189	0.128378	0.000000	0.128378	5	3	694	1330	0.99324
4	1.528089e+09	1	1	49966	80	8	2319	1	1.528089e+09	13	7.046841	0.063887	0.063887	0.127774	0.000000	0.127774	5	3	989	1330	0.99335

- **Shuffling the Dataset and Picking Randomly 50% of Data from the Dataset -:**

```
data = data.sample(frac=0.5)
data
```

	stime	flgs_number	proto_number	sport	dport	pkts	bytes	state_number	ltime	seq	dur	mean	stddev	sum	min	max	spkts	dpkts	sbyt
659843	1.528099e+09	1	3	9188	80	10	600	4	1.528099e+09	31227	15.017910	3.925332	0.311253	11.775997	3.485205	4.151208	10	0	6
302631	1.528096e+09	2	1	62901	80	7	890	1	1.528096e+09	127092	12.471382	2.064312	1.326024	6.192936	0.190451	3.064419	6	1	8
14951	1.528081e+09	2	1	3170	80	5	770	3	1.528081e+09	13475	31.253151	0.000000	0.000000	0.000000	0.000000	0.000000	5	0	7
751223	1.528099e+09	1	3	41236	80	10	600	4	1.528099e+09	122607	14.460194	3.899510	0.850551	11.698529	2.756767	4.796090	10	0	6
209033	1.528081e+09	2	1	61088	80	4	616	3	1.528081e+09	207557	21.910122	0.000000	0.000000	0.000000	0.000000	0.000000	4	0	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
333934	1.528081e+09	2	1	61406	80	5	770	3	1.528081e+09	70305	24.307741	0.000000	0.000000	0.000000	0.000000	0.000000	5	0	7
264476	1.528081e+09	5	1	30669	80	6	548	1	1.528081e+09	847	28.889830	0.153009	0.001339	0.306018	0.151671	0.154347	4	2	4
654148	1.528085e+09	1	3	13409	80	7	420	4	1.528085e+09	36872	26.003195	2.725399	1.574823	10.901598	0.000000	3.738362	7	0	4
113653	1.528096e+09	2	1	9826	80	4	616	3	1.528096e+09	200260	12.091130	3.984468	0.013611	7.968935	3.970857	3.998078	4	0	6
395454	1.528081e+09	2	1	21867	80	5	770	3	1.528081e+09	131826	22.427517	1.029850	1.783753	4.119402	0.000000	4.119402	5	0	7

1000000 rows × 41 columns

```
[ ] data['category_enc'].value_counts()
```

```
0    500534
1    499466
Name: category_enc, dtype: int64
```

- **Assigning the target variable to Y**

```
[ ] y = data['category_enc']
```

```
[ ] data.drop(["category_enc"],axis=1,inplace=True)
```

- **Normalization -:**

Normalization

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
features = data
cols = features.columns
scaled_features = scaler.fit_transform(features)
data = pd.DataFrame(scaled_features,columns = cols)
```

```
data.head()
```

	stime	flgs_number	proto_number	sport	dport	pkts	bytes	state_number	ltime	seq	dur	mean
0	1.273510	-0.821022	1.286076	-1.239089	-0.030077	0.374127	-0.076817	0.965157	1.273366	-1.180390	-0.624478	1.264979
1	0.791819	0.326981	-0.777549	1.586752	-0.030077	0.028383	0.401790	-1.510539	0.791002	0.108473	-0.837051	0.071409
2	-1.163996	0.326981	-0.777549	-1.555696	-0.030077	-0.202113	0.203746	0.139925	-1.163605	-1.419058	0.730771	-1.252542
3	1.273651	-0.821022	1.286076	0.446956	-0.030077	0.374127	-0.076817	0.965157	1.273433	0.048175	-0.671034	1.248418
4	-1.162841	0.326981	-0.777549	1.491370	-0.030077	-0.317361	-0.050411	0.139925	-1.163715	1.190291	-0.049146	-1.252542

- **Splitting of Data into test and train**

```
[ ]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(data, y, test_size = 0.2, random_
```

- **Loading Data into Dmatrix and Finding into MAE**

```
import xgboost as xgboost
dtrain = xgboost.DMatrix(X_train, label=y_train)
dtest = xgboost.DMatrix(X_test, label=y_test)

from sklearn.metrics import mean_absolute_error
```

```
[ ]  mean_train = np.mean(y_train)
     baseline_predictions = np.ones(y_test.shape) * mean_train
     mae_baseline = mean_absolute_error(y_test, baseline_predictions)
     print("Baseline MAE is {:.2f}".format(mae_baseline))

     Baseline MAE is 0.50
```

```
params = {
    # Parameters that we are going to tune.
    'max_depth': 6,
    'min_child_weight': 1,
    'eta':.3,
    'subsample': 1,
    'colsample_bytree': 1,
    # Other parameters
    'objective':'reg:squarederror',
}
```

- **HypeParameter Tuning**

Taking different parameters and finding the best solution -:

HyperParameter Tuning

Parameters num_boost_round and early_stopping_rounds

```
from xgboost import XGBClassifier
params['eval_metric'] = "mae"
num_boost_round = 999
xgb_model = XGBClassifier(**params)
xgb_model = xgboost.train(
    params,
    dtrain,
    num_boost_round=num_boost_round,
    evals=[(dtest, "Test")],
    early_stopping_rounds=10
)
```

```
[0]     Test-mae:0.34984
Will train until Test-mae hasn't improved in 10 rounds.
[1]     Test-mae:0.244783
[2]     Test-mae:0.171656
[3]     Test-mae:0.120101
[4]     Test-mae:0.083991
[5]     Test-mae:0.058763
[6]     Test-mae:0.04121
[7]     Test-mae:0.028813
[8]     Test-mae:0.020165
[9]     Test-mae:0.014126
[10]    Test-mae:0.009888
[11]    Test-mae:0.006918
[12]    Test-mae:0.004846
[13]    Test-mae:0.003388
```

Parameters max_depth and min_child_weight

```
#Let's make a list containing all the combinations max_depth/min_child_weight that we want to try.
gridsearch_params = [
    (max_depth, min_child_weight)
    for max_depth in range(9,12)
    for min_child_weight in range(5,8)
]
```

```
[ ] # Define initial best params and MAE
min_mae = float("Inf")
best_params = None
for max_depth, min_child_weight in gridsearch_params:
    print("CV with max_depth={}, min_child_weight={}".format(
                             max_depth,
                             min_child_weight))
    # Update our parameters
    params['max_depth'] = max_depth
    params['min_child_weight'] = min_child_weight
    # Run CV
    cv_results = xgboost.cv(
        params,
        dtrain,
        num_boost_round=num_boost_round,
        seed=42,
        nfold=5,
        metrics={'mae'},
        early_stopping_rounds=10
    )
```

```
        early_stopping_rounds=10
    )
    # Update best MAE
    mean_mae = cv_results['test-mae-mean'].min()
    boost_rounds = cv_results['test-mae-mean'].argmin()
    print("\tMAE {} for {} rounds".format(mean_mae, boost_rounds))
    if mean_mae < min_mae:
        min_mae = mean_mae
        best_params = (max_depth,min_child_weight)
print("Best params: {}, {}, MAE: {}".format(best_params[0], best_params[1], min_mae))
```

```
CV with max_depth=9, min_child_weight=5
        MAE 1e-06 for 35 rounds
CV with max_depth=9, min_child_weight=6
        MAE 1e-06 for 35 rounds
CV with max_depth=9, min_child_weight=7
        MAE 1e-06 for 35 rounds
CV with max_depth=10, min_child_weight=5
        MAE 1e-06 for 35 rounds
CV with max_depth=10, min_child_weight=6
        MAE 1e-06 for 35 rounds
CV with max_depth=10, min_child_weight=7
        MAE 1e-06 for 35 rounds
CV with max_depth=11, min_child_weight=5
        MAE 1e-06 for 35 rounds
CV with max_depth=11, min_child_weight=6
        MAE 1e-06 for 35 rounds
CV with max_depth=11, min_child_weight=7
        MAE 1e-06 for 35 rounds
Best params: 9, 5, MAE: 1e-06
```

```
[ ] #We get the best score with a max_depth of 10 and min_child_weight of 6, so
params['max_depth'] = 9
params['min_child_weight'] = 5
```

Parameter ETA

```
min_mae = float("Inf")
best_params = None
for eta in [.3, .2, .1, .05, .01, .005]:
    print("CV with eta={}".format(eta))
    # We update our parameters
    params['eta'] = eta
    # Run and time CV
    %time cv_results = xgboost.cv(params, dtrain,num_boost_round=num_boost_round, seed=42, nfold=5, metrics=['mae'], early_stopping_rounds=
    # Update best score
    mean_mae = cv_results['test-mae-mean'].min()
    boost_rounds = cv_results['test-mae-mean'].argmin()
    print("\tMAE {} for {} rounds\n".format(mean_mae, boost_rounds))
    if mean_mae < min_mae:
        min_mae = mean_mae
        best_params = eta
print("Best params: {}, MAE: {}".format(best_params, min_mae))
```

- ## Using the values from hyperParameter tuning -:

  ## And performing the XGboost again

Final dictionary of parameters after tuning

```
params_final = {
    'colsample_bytree': 1.0,
     'eta': 0.3,
     'eval_metric': 'mae',
     'max_depth': 9,
     'min_child_weight': 5,
     'objective': 'reg:squarederror',
     'subsample': 1.0
}
xgb_model = xgboost.train(
    params_final,
    dtrain,
    num_boost_round=num_boost_round,
    evals=[(dtest, "Test")],
    early_stopping_rounds=10
)
```

```
[0]     Test-mae:0.34984
Will train until Test-mae hasn't improved in 10 rounds.
[1]     Test-mae:0.244783
[2]     Test-mae:0.171656
[3]     Test-mae:0.120101
[4]     Test-mae:0.083991
[5]     Test-mae:0.058763
[6]     Test-mae:0.04121
[7]     Test-mae:0.028813
[8]     Test-mae:0.020165
[9]     Test-mae:0.014126
[10]    Test-mae:0.009888
```

- ## Result of doing HyperParameter tuning

Improved Mean Absolute Error

```
[ ] mean_absolute_error(best_model.predict(dtest), y_test)
```

```
1.3186104780015739e-06
```

MAE reduced to roughly around 1.251e-06 from 2.2e-05

```
[ ] from sklearn.metrics import accuracy_score
    from sklearn.metrics import mean_squared_error
    from xgboost import XGBClassifier
    best_model = XGBClassifier()
    best_model.set_params(**params_final)
    best_model.fit(X_train, y_train)
    y_pred = best_model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    print("RMSE: %f" % (rmse))
```

```
RMSE: 0.000000
```

- ## Finding ROC Score and Accuracy -:

```
[ ] from sklearn.metrics import roc_auc_score
    from sklearn.metrics import roc_curve
    from sklearn.metrics import auc
    from matplotlib import pyplot
    y_pred = best_model.predict_proba(X_test)[:,1]
    print ('roc auc score:', roc_auc_score(y_test,y_pred))
```

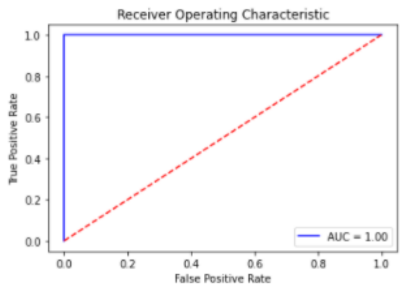
```
roc auc score: 1.0
```

```
[ ] predictions = [round(value) for value in y_pred]
    # evaluate predictions
    accuracy = accuracy_score(y_test, predictions)
    print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 100.00%
```

- **Drawing the final ROC curve-:**

```python
from sklearn import metrics
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
roc_auc = metrics.auc(fpr, tpr)
pyplot.title('Receiver Operating Characteristic')
pyplot.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
pyplot.legend(loc = 'lower right')
pyplot.plot([0, 1], [0, 1],'r--')
pyplot.ylabel('True Positive Rate')
pyplot.xlabel('False Positive Rate')
pyplot.gcf().savefig('roc.png')
pyplot.show()
```



- **Conclusion**

```python
from sklearn.metrics import classification_report
predictions = best_model.predict(X_test)
print(classification_report(y_test.values, predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    100024
           1       1.00      1.00      1.00     99976

    accuracy                           1.00    200000
   macro avg       1.00      1.00      1.00    200000
weighted avg       1.00      1.00      1.00    200000
```

FOR MORE CLEAR IMAGE AND CLARIFICATION I HAVE PERFORMED THE SAME FOR 2 MORE DATA SET AND THE RESULTS ARE AS SHOWN BELOW-:

- ## Loading of Datasets

```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import warnings
warnings.filterwarnings("ignore")
```

```
[ ] from google.colab import drive

    drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
[ ] data1 = pd.read_csv("/content/gdrive/My Drive/Dataset/UNSW_2018_IoT_Botnet_Full5pc_2.csv")
```

```
data1.head()
```

	pkSeqID	stime	flgs	flgs_number	proto	proto_number	saddr	sport	daddr	dport	pkts	bytes	state	state_number	ltime	seq	dur	
0	1000001	1.528085e+09	e	1	udp	3	192.168.100.148	37153	192.168.100.6	80	8	480	INT	4	1.528085e+09	120567	25.001644	3.5
1	1000002	1.528085e+09	e	1	udp	3	192.168.100.148	37154	192.168.100.6	80	8	480	INT	4	1.528085e+09	120568	25.001644	3.5
2	1000003	1.528085e+09	e	1	udp	3	192.168.100.148	37155	192.168.100.6	80	8	480	INT	4	1.528085e+09	120569	25.001644	3.5
3	1000004	1.528085e+09	e	1	udp	3	192.168.100.148	37156	192.168.100.6	80	8	480	INT	4	1.528085e+09	120570	25.001644	3.5
4	1000005	1.528085e+09	e	1	udp	3	192.168.100.148	37157	192.168.100.6	80	8	480	INT	4	1.528085e+09	120571	25.001644	3.5

```
[ ] data2 = pd.read_csv("/content/gdrive/My Drive/Dataset/UNSW_2018_IoT_Botnet_Full5pc_4.csv")
    data2.head()
```

	pkSeqID	stime	flgs	flgs_number	proto	proto_number	saddr	sport	daddr	dport	pkts	bytes	state	state_number	ltime	seq	dur	mean	stddev	sum
0	3000001	1.528099e+09	e	1	udp	3	192.168.100.147	6226	192.168.100.3	80	15	900	INT	4	1.528099e+09	109223	13.657889	3.91046	1.367803	11.73138
1	3000002	1.528099e+09	e	1	udp	3	192.168.100.147	6227	192.168.100.3	80	15	900	INT	4	1.528099e+09	109224	13.657889	3.91046	1.367802	11.73138
2	3000003	1.528099e+09	e	1	udp	3	192.168.100.147	6228	192.168.100.3	80	15	900	INT	4	1.528099e+09	109225	13.657889	3.91046	1.367802	11.73138
3	3000004	1.528099e+09	e	1	udp	3	192.168.100.147	6229	192.168.100.3	80	15	900	INT	4	1.528099e+09	109226	13.657889	3.91046	1.367802	11.73138
4	3000005	1.528099e+09	e	1	udp	3	192.168.100.147	6230	192.168.100.3	80	15	900	INT	4	1.528099e+09	109227	13.657889	3.91046	1.367803	11.73138

```
data1.shape
```

```
(1000000, 46)
```

+ Code + Text

```
[ ] data2.shape
```

```
(668522, 46)
```

- ## Adding of Dataset

```
[ ] data = data1.append(data2)
```

```
[ ] data.shape
```

```
(1668522, 46)
```

```
data.head()
```

	pkSeqID	stime	flgs	flgs_number	proto	proto_number	saddr	sport	daddr	dport	pkts	bytes	state	state_number	ltime	seq
0	1000001	1.528085e+09	e	1	udp	3	192.168.100.148	37153	192.168.100.6	80	8	480	INT	4	1.528085e+09	120567
1	1000002	1.528085e+09	e	1	udp	3	192.168.100.148	37154	192.168.100.6	80	8	480	INT	4	1.528085e+09	120568
2	1000003	1.528085e+09	e	1	udp	3	192.168.100.148	37155	192.168.100.6	80	8	480	INT	4	1.528085e+09	120569
3	1000004	1.528085e+09	e	1	udp	3	192.168.100.148	37156	192.168.100.6	80	8	480	INT	4	1.528085e+09	120570
4	1000005	1.528085e+09	e	1	udp	3	192.168.100.148	37157	192.168.100.6	80	8	480	INT	4	1.528085e+09	120571

- ## Preprocessing :

Checking of types of data and if null or not -:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1668522 entries, 0 to 668521
Data columns (total 46 columns):
 #   Column                           Non-Null Count    Dtype
---  ------                           --------------    -----
 0   pkSeqID                          1668522 non-null  int64
 1   stime                            1668522 non-null  float64
 2   flgs                             1668522 non-null  object
 3   flgs_number                      1668522 non-null  int64
 4   proto                            1668522 non-null  object
 5   proto_number                     1668522 non-null  int64
 6   saddr                            1668522 non-null  object
 7   sport                            1668522 non-null  object
 8   daddr                            1668522 non-null  object
 9   dport                            1668522 non-null  object
 10  pkts                             1668522 non-null  int64
 11  bytes                            1668522 non-null  int64
 12  state                            1668522 non-null  object
 13  state_number                     1668522 non-null  int64
 14  ltime                            1668522 non-null  float64
 15  seq                              1668522 non-null  int64
 16  dur                              1668522 non-null  float64
 17  mean                             1668522 non-null  float64
 18  stddev                           1668522 non-null  float64
 19  sum                              1668522 non-null  float64
 20  min                              1668522 non-null  float64
 21  max                              1668522 non-null  float64
 22  spkts                            1668522 non-null  int64
 23  dpkts                            1668522 non-null  int64
 24  sbytes                           1668522 non-null  int64
 25  dbytes                           1668522 non-null  int64
 26  rate                             1668522 non-null  float64
 27  srate                            1668522 non-null  float64
 28  drate                            1668522 non-null  float64
```

```
data.isnull().sum()
```

```
pkSeqID                            0
stime                              0
flgs                               0
flgs_number                        0
proto                              0
proto_number                       0
saddr                              0
sport                              0
daddr                              0
dport                              0
pkts                               0
bytes                              0
state                              0
state_number                       0
ltime                              0
seq                                0
dur                                0
mean                               0
stddev                             0
sum                                0
min                                0
max                                0
spkts                              0
dpkts                              0
sbytes                             0
dbytes                             0
rate                               0
srate                              0
drate                              0
TnBPSrcIP                          0
TnBPDstIP                          0
TnP_PSrcIP                         0
TnP_PDstIP                         0
TnP_PerProto                       0
```

- ## Dropping of useless columns -:

```
[ ] data.drop(["pkSeqID","flgs","proto","state","attack"],axis=1,inplace=True)
```

```
[ ] data.shape

(1668522, 41)
```

- **Replacing HEX values with INT values and converting the datatypes to support format using Label Encoder-:**

```
[ ] data['sport'] = data['sport'].astype(str)
    search_string='0x'
    result = set([i for i in data['sport'] if i.startswith(search_string)])
    result
```

```
{'0x0008', '0x000d', '0x0011', '0x0303'}
```

```
data['sport']=data['sport'].replace(['0x5000'],'20480')
data['sport']=data['sport'].replace(['0x0303'],'771')
data['sport']=data['sport'].replace(['0x0008'],'8')
data['sport']=data['sport'].replace(['0x000d'],'13')
data['sport']=data['sport'].replace(['0x0011'],'17')
data["sport"] = data["sport"].astype(str).astype(int)
```

```
[ ] data['dport'] = data['dport'].astype(str)
    data['dport'] = data.dport.apply(lambda x: int(x,16) if len(x)>1 and x[1]=="x" else int(x))
```

```
[ ] data.dtypes[data.dtypes=='object']
```

```
saddr          object
daddr          object
category       object
subcategory    object
dtype: object
```

- **Converting the datatype to support format using Label Encoder-:**

```
[ ] from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()
    data["saddr_enc"] = le.fit_transform(data.saddr)
    data["daddr_enc"] = le.fit_transform(data.daddr)
    data["category_enc"]= le.fit_transform(data.category)
    data["subcategory_enc"]= le.fit_transform(data.subcategory)
    data.drop(['saddr','daddr','category','subcategory'],axis=1,inplace=True)
```

```
[ ] data.head()
```

	stime	flgs_number	proto_number	sport	dport	pkts	bytes	state_number	ltime	seq	dur	mean
0	1.528085e+09	1	3	37153	80	8	480	4	1.528085e+09	120567	25.001644	3.565624
1	1.528085e+09	1	3	37154	80	8	480	4	1.528085e+09	120568	25.001644	3.565624
2	1.528085e+09	1	3	37155	80	8	480	4	1.528085e+09	120569	25.001644	3.565624
3	1.528085e+09	1	3	37156	80	8	480	4	1.528085e+09	120570	25.001644	3.565624
4	1.528085e+09	1	3	37157	80	8	480	4	1.528085e+09	120571	25.001644	3.565624

- **Selecting any 50% data randomly -:**

```
data = data.sample(frac=0.5)
data
```

	stime	flgs_number	proto_number	sport	dport	pkts	bytes	state_number	ltime	seq	dur	mean	stddev	sum	min	max	spkts	dpk
520037	1.528099e+09	1	3	29342	80	14	840	4	1.528099e+09	104957	12.937773	3.638428	1.870712	10.915283	0.992966	4.983363	14	
602683	1.526345e+09	6	3	39705	18250	1	60	4	1.526345e+09	7641	0.000024	0.000024	0.000000	0.000024	0.000024	0.000024	1	
128908	1.528085e+09	1	3	54071	80	6	360	4	1.528085e+09	249476	20.539581	4.099117	0.022350	12.297350	4.067679	4.117305	6	
705703	1.528096e+09	2	1	18330	80	2	308	3	1.528096e+09	54463	10.078871	0.000000	0.000000	0.000000	0.000000	0.000000	2	
586340	1.528085e+09	1	3	859	80	8	480	4	1.528085e+09	182598	24.671419	3.679908	0.558909	14.719631	3.273262	4.639475	8	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
252957	1.528099e+09	1	3	19156	80	15	900	4	1.528099e+09	100030	13.956930	3.982636	0.822615	11.947907	2.979575	4.994506	15	
562560	1.528099e+09	1	3	22905	80	8	480	4	1.528099e+09	147480	14.897301	3.613022	0.631985	10.839066	2.719263	4.062235	8	
330682	1.528099e+09	1	3	18814	80	10	600	4	1.528099e+09	177755	13.782870	3.680391	0.646283	11.041172	2.767383	4.173445	10	
552980	1.528085e+09	6	3	5784	80	7	420	4	1.528085e+09	149238	24.775734	3.029919	1.717800	12.119675	0.054800	4.043037	7	
947843	1.528096e+09	2	1	35593	80	5	770	3	1.528096e+09	34447	16.017357	2.700666	1.909805	8.101997	0.000000	4.079854	5	

834261 rows × 41 columns

- **Loading target (y) and dropping it from data -:**

```
data['category_enc'].value_counts()
```

```
0    463263
1    325337
3     45400
2       218
4        43
Name: category_enc, dtype: int64
```

```
[ ] y = data['category_enc']
```

```
[ ] data.drop(["category_enc"],axis=1,inplace=True)
```

- **Normalization**

NORMALIZATION

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
features = data.iloc[:,:-1]
cols = features.columns
scaled_features = scaler.fit_transform(features)
data = pd.DataFrame(scaled_features,columns = cols)
```

```
[ ] data.head()
```

	stime	flgs_number	proto_number	sport	dport	pkts	bytes	state_number	ltime	seq	dur	mean	stddev	sum	min	max	spkts	dp
0	0.256064	-0.284255	0.566509	-0.200234	-0.135721	0.023583	-0.002569	0.479336	0.256050	-0.247548	-0.167881	0.769188	1.077141	0.158223	-0.091915	0.907990	0.040393	-0.005
1	-4.402405	6.160166	0.566509	0.337902	3.703371	-0.038923	-0.006366	0.479336	-4.402442	-1.506113	-0.620770	-1.841555	-1.291937	-1.012039	-0.781686	-2.010063	-0.057403	-0.005
2	0.217798	-0.284255	0.566509	1.083907	-0.135721	-0.014882	-0.004906	0.479336	0.217804	1.621483	0.098222	1.099756	-1.263633	0.306399	2.044009	0.400860	-0.019789	-0.005
3	0.246386	1.004629	-1.727500	-0.772071	-0.135721	-0.034115	-0.005159	-0.476685	0.246365	-0.900575	-0.267957	-1.841572	-1.291937	-1.012041	-0.781703	-2.010077	-0.049880	-0.005
4	0.217971	-0.284255	0.566509	-1.679315	-0.135721	-0.005266	-0.004322	0.479336	0.217989	0.756565	0.242858	0.798952	-0.584132	0.566100	1.492148	0.706623	-0.004743	-0.005

- **Splitting the data and performing XgBoost**

Splitting of Dataset into test and train set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size = 0.2, random_state = 1)


import xgboost as xgboost
dtrain = xgboost.DMatrix(X_train, label=y_train)
dtest = xgboost.DMatrix(X_test, label=y_test)


from sklearn.metrics import mean_absolute_error
mean_train = np.mean(y_train)
baseline_predictions = np.ones(y_test.shape) * mean_train
mae_baseline = mean_absolute_error(y_test, baseline_predictions)
print("Baseline MAE is {:.2f}".format(mae_baseline))
```

```
Baseline MAE is 0.62
```

- **Calculating ROC and Accuracy before resampling (hyperparameter tuning)**

Calculation before Resampling

```
[ ] from xgboost import XGBClassifier
    best_model = XGBClassifier()
    best_model.fit(X_train, y_train)
    y_pred = best_model.predict(X_test)
    predictions = [round(value) for value in y_pred]
```

```
[ ] from sklearn.metrics import roc_auc_score
    from sklearn.metrics import roc_curve
    from sklearn.metrics import auc
    from matplotlib import pyplot
    y_pred = best_model.predict_proba(X_test)[:,1]
    predictions = [round(value) for value in y_pred]
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE: %f" % (rmse))
```

```
RMSE: 0.702341
```

```
[ ] accuracy = accuracy_score(y_test, predictions)

    print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 94.51%
```

NOTE -: We can see that Accuracy is nearly 94.9%

- **Performing hyperparameter tuning**

```
[ ] from sklearn.utils import compute_sample_weight
    sample_weight = compute_sample_weight('balanced', y_train)
    sample_weight
```

```
array([0.35998468, 0.35998468, 0.51318155, ..., 0.35998468, 0.51318155,
       0.51318155])
```

```
[ ] params_final = {
        'colsample_bytree': 1.0,
        'eta': 0.3,
        'eval_metric': 'mae',
        'max_depth': 9,
        'min_child_weight': 5,
        'objective': 'reg:squarederror',
        'subsample': 1.0
    }
```

```
best_model.set_params(**params_final)
best_model.fit(X_train, y_train,sample_weight=sample_weight)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1.0, eta=0.3,
              eval_metric='mae', gamma=0, learning_rate=0.1, max_delta_step=0,
              max_depth=9, min_child_weight=5, missing=None, n_estimators=100,
              n_jobs=1, nthread=None, objective='multi:softprob',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              seed=None, silent=None, subsample=1.0, verbosity=1)
```

```
[ ] y_pred = best_model.predict(X_test)
    predictions = [round(value) for value in y_pred]
```

- **Calculating the Accuracy and Roc after tuning -:**

```
[ ]              max_depth=9, min_child_weight=5, missing=None, n_estimators=100,
                 n_jobs=1, nthread=None, objective='multi:softprob',
                 random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
                 seed=None, silent=None, subsample=1.0, verbosity=1)
```

```
[ ] y_pred = best_model.predict(X_test)
    predictions = [round(value) for value in y_pred]
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE: %f" % (rmse))

RMSE: 0.003462
```

```
[ ] accuracy = accuracy_score(y_test, y_pred)

    print("Accuracy: %.2f%%" % (accuracy * 100.0))

    Accuracy: 100.00%
```

We can see that the detection of intrusion accuracy is 100% now .

Result and conclusion -:

The conclusion of the entire project is that by using XGBOOST we can conduct Intrusion detection by training our model by the use of datas in previous attacks done by the attackers and prevent the attack in the future and we also see that the model provides the features of HYPER PARAMETRIC tuning which helps us to improve aur time constraint and the accuracy of the project we can also see how the accuracy of the project was enhanced from 94.9% to 100% for both the dataset.This shows the effectivity of the project in the field of intrusion detection by the help of MACHINE LEARNING .

REFERENCE -:

1. Effective Intrusion Detection System Using XGBoost Sukhpreet Singh Dhaliwal * ID,Abdullah-Al Nahid ID and Robert Abbas ID School of Engineering, Macquarie University, Sydney NSW 2109, Australia; https://www.mdpi.com/2078-2489/9/7/149

2. A COMPARATIVE ANALYSIS OF PHISHING WEBSITE DETECTION USING XGBOOST ALGORITHM 1HAJARA MUSA, 2DR. A.Y GITAL, 3 F. U. ZAMBUK, 4ABUBAKAR UMAR, 5AISHATU YAHYA UMAR,6JAMILU USMAN WAZIRI 1 & 5 Department of Mathematical Sciences, Gombe State University, Gombe, Nigeria 2,3,& 4 Department of Mathematical Sciences, Abubakar Tafawa Balewa University Bauchi, Nigeria 6 Department of Mathematical Sciences, Federal University Gusau, Nigeria https://www.researchgate.net/publication/309168778_A_Comparative_Analysis_of_Phishing_Detection_and_Prevention_Techniques

3. An efficient XGBoost–DNN-based classification model for network intrusion detection system Preethi Devan1 • Neelu Khare1 Received: 29 October 2018/Accepted: 7 January 2020/Published online: 19 January 2020 Springer-Verlag London Ltd., part of Springer Nature 2020 http://link.springer.com.egateway.vit.ac.in/article/10.1007%2Fs00521-020-04708-x

4. Network Intrusion Detection Based on PSO-Xgboost Model HUI JIANG1 , ZHENG HE2,3, GANG YE2,3, AND HUYIN ZHANG1 1 School of Computer Science, Wuhan University, Wuhan 430072, China 2National Engineering Research Center forMultimedia Software, School of Computer Science, Wuhan University, Wuhan 430072, China3Hubei Key Laboratory of Multimedia and Network Communication Engineering, WuhanUniversity, Wuhan 430072, China https://ieeexplore.ieee.org/document/9044370

5. A Novel PCA-Firefly Based XGBoost Classification Model for Intrusion Detection in Networks Using GPU Sweta Bhattacharya 1, Siva Rama Krishnan S 1, Praveen Kumar Reddy Maddikunta 1, RajeshKaluri 1 Saurabh Singh 2, Thippa Reddy Gadekallu 1,* , Mamoun Alazab 3,* and Usman Tariq4,* 1 School of Information Technology and Engineering, VIT - Vellore, Tamil Nadu 632014,India; 2School of Computer, Information and Communication Engineering, Kunsan NationalUniversity, Gunsan 54150, Korea; 3Senior IEEE Member, IT and Environment, Charles DarwinUniversity, 0815 Darwin, Australia 4 College of Computer Engineering and Sciences, PrinceSattam bin Abdulaziz University, Al-Khraj 11942, Saudi https://www.mdpi.com/2079-9292/9/2/219