PROJECT REPORT ON

DENIAL OF SERVICE (DDOS) DETECTION USING GRADIENT BOOSTING ALGORITHM

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

Submitted by

122003291 - SAI ADITYA VISWANADHAM - CSE

Under the Guidance of

Prof. Sasikala Devi. N

School of Computing
SASTRA DEEMED TO BE UNIVERSITY
(A University established under section 3 of the UGC Act, 1956)
Tirumalaisamudram
Thanjavur - 613401
December (2020)

SHANMUGHA

ARTS, SCIENCE, TECHNOLOGY & RESEARCH ACADEMY (SASTRA DEEMED TO BE UNIVERSITY)

(A University Established under section 3 of the UGC Act, 1956)

TIRUMALAISAMUDRAM, THANJAVUR – 613401

BONAFIDE CERTIFICATE

Certified that this project work entitled **"DENIAL OF SERVICE (DDOS) DETECTION USING GRADIENT BOOSTING ALGORITHM"** submitted to the Shanmugha Arts, Science, Technology & Research Academy (SASTRA Deemed to be University), Tirumalaisamudram - 613401 by Sai Aditya Viswanadham (122003291), CSE in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in their respective programme. This work is an original and independent work carried out under my guidance, during the period August 2020 - December 2020.

Prof. Sasikala Devi. N

ASSOCIATE DEAN
SCHOOL OF COMPUTING

Submitted for Project Viva Voce held on____________________

Examiner – I

Examiner – II

TABLE OF CONTENTS

ACKNOWLEDGEMENTS………………………………………………………………….. (iv)

ABSTRACT……………………………………………………………………………….. (v)

CHAPTER 1 INTRODUCTION………………………………………………………... (1)

CHAPTER 2 RELATED WORKS……………………………………………………… (4)

CHAPTER 3 PROPOSED WORK……………………………………………………… (6)

CHAPTER 4 SOURCE CODE………………………………………………………….. (9)

CHAPTER 5 RESULTS…………………………………………………………………. (42)

CHAPTER 6 PERFORMANCE EVALUATION………………………………………. (47)

CHAPTER 7 CONCLUSION AND FUTURE WORKS……………………………….. (49)

CHAPTER 8 REFERENCES……………………………………………………………. (50)

ACKNOWLEDGEMENTS

Firstly, I would be grateful for the management and faculty for their huge efforts of smooth running of Academics without any hurdles in this pandemic situation.

I would like to express my sincere gratitude to **Dr S. Vaidyasubramaniam, Vice-Chancellor** for his encouragement during the span of my academic life at SASTRA Deemed University.

I would specially thank and express my gratitude to **Prof. Sasikala Devi .N, Senior Assistant Professor, School of Computing** for providing me an opportunity to do this project and for her overwhelming support and guidance to successfully complete the project.

I also thank all the Teaching and Non-teaching faculty, and all other people who have directly or indirectly help me through their support, encouragement and all other assistance extended for completion of my project and for successful completion of all courses during my academic life at SASTRA Deemed University.

Finally, I thank God Almighty for his endless blessings and my parents who help me acquire this interest in project and aided me in completing it within the deadline without much struggle.

ABSTRACT

A Security of information is of utmost importance to organizations striving to survive in a competitive marketplace. Network security has been an issue since the internet is changing the face of computing. Distributed Denial of Service (DDOS) attack is an attempt to make services unavailable to the legitimate users. DDOS attack is one of the least sophisticated categories of security. DDOS is a persistent attack which affects the availability of the network. It also has the ability to be one of the most disruptive and most powerful by taking websites and digital services offline for a significant period. DDOS attacks are emerging as the most devastating attacks for organizations. As the attacks and their impact are growing rapidly, methods like Signature based detection and Scrubbing are challenged. These attacks continue to grow in magnitude, frequency and sophistication. Manual analysis is eliminated in anomaly-based DDOS detection with zero misclassifications achieving perfect accuracy. This paper demonstrates DDOS anomaly detection on the open CIC datasets using Stochastic Gradient Boosting (SGB) Machine Learning model. Maximum accuracy is achieved by tuning the hyper-parameters.

KEYWORDS: Denial of Service (DOS) ; Stochastic Gradient Boosting (SGB) ; Scikit-Learn ; ROC ; XGBOOST

List of Tables

Table No.	Table Name	Page No.
Table 1	Algorithm Performance (Unbalanced)	47
Table 2	Algorithm Performance (Balanced)	48

List of Figures

Figure No.	Figure Name	Page No.
Figure 1.1	DDOS Attack	2
Figure 1.2	Types of DOS attacks	3
Figure 5.1	ROC for SGB Balanced	42
Figure 5.2	ROC for Naive Bayes Balanced	42
Figure 5.3	ROC for Decision Tree Balanced	43
Figure 5.4	ROC for K-NN Balanced	43
Figure 5.5	ROC for Random Forest Balanced	44
Figure 5.6	ROC for SGB Unbalanced	44
Figure 5.7	ROC for Naive Bayes Unbalanced	45
Figure 5.8	ROC for Decision Tree Unbalanced	45
Figure 5.9	ROC for K-NN Unbalanced	46
Figure 5.10	ROC for Random Forest Unbalanced	46

NOTATIONS

Notation	Description
Loss(.)	Loss function
rps	Requests per second
bps	Bits per second
pps	Packets per second
F_0	Initial Model
F_1	Boosted Version of F_0
h_n	Fit residual
γ	Initial Prediction
m	Iterations count

ABBREVIATIONS

DOS	Denial of Service
DDOS	Distributed Denial of Service
SGB	Stochastic Gradient Boosting
KNN	K-Nearest Neighbour
ACK	Acknowledgement
SKLEARN	Scikit-Learn
ROC	Receiver Operating Characteristic
TPR	True Positive Rate
FPR	False Positive Rate

CHAPTER 1

INTRODUCTION

The minimal processing and best-effort forwarding of any packet, malicious or not, was the prime concern when the Internet was designed. This architecture creates an unregulated network path, which can be exploited. The key design feature of the internet and the protocols used makes it vulnerable to various security issues and Denial of Service stands first on the list because it disrupts the availability of services. A Denial of Service attack (DOS) is one of the most powerful attacks on the internet. A distributed denial of service (DDOS) attack is a coordinated attack on the availability of services of a victim system or network resources, launched indirectly through many compromised computers on the Internet. A Distributed Denial of Service (DDOS) attack is an attempt to perform a DOS attack from multiple sources to increase the effectiveness of the DOS attack. The goal of the attack is to render the website or service inoperable and make it unavailable to its legitimate users. Distributed Denial of Service (DDOS) attack is an attempt to make services unavailable to the legitimate users. A DDOS attack is one of the least sophisticated categories of security. DDOS is a persistent attack that affects the availability of the network. It also has the potential to be one of the most disruptive and most powerful by taking websites and digital services offline for a significant period. DDOS attacks are emerging as the most devastating attacks for organizations. As the attacks and their impact are growing swiftly, methods like Scrubbing and Signature-based detection are challenged. These attacks continue to grow in frequency, sophistication, and magnitude.

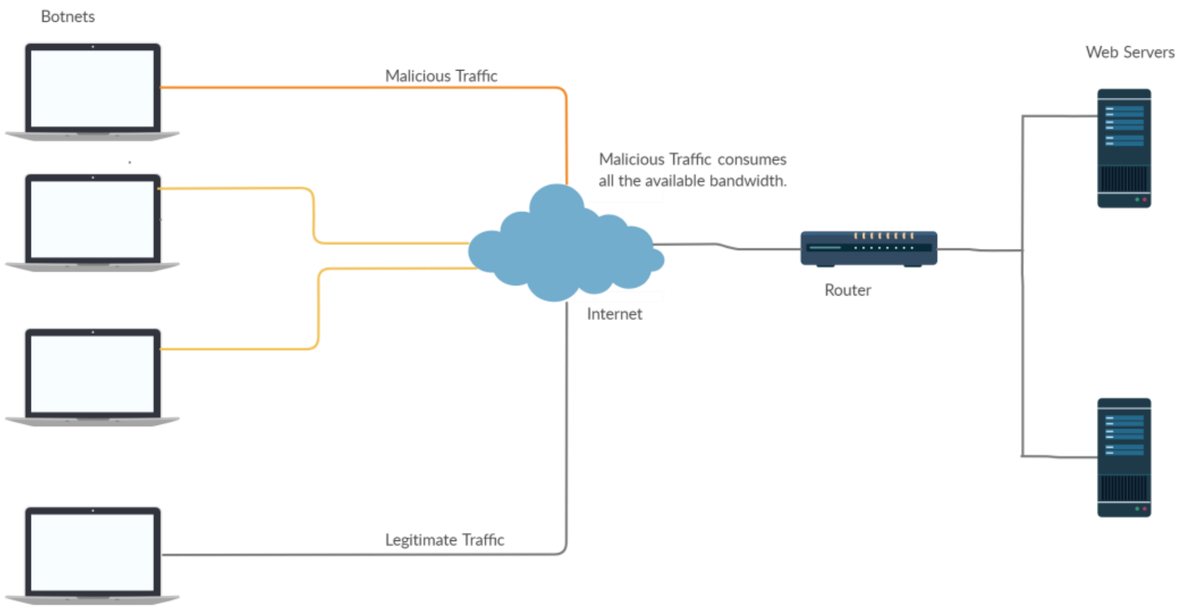

Fig 1.1 DDOS Attack

DDOS attacks can be broadly distributed into three categories namely Application-level attacks, Protocol attacks and volumetric attacks.

Application Layer Attacks:

An application layer attack targets computers by using a fault in an operating system or applications. This results in the attacker to bypass the normal access controls. These attacks can be performed either on a server or a client computer. These are most sophisticated attacks and can be more effective even with a single machine generating traffic. Some of the most common application layer attacks are low-and-slow attacks like Apache range header and Slowloris attack. The strength of these attacks is measured in requests per second (rps).

Protocol attacks:

Protocol attacks aim to exhaust server resources. They target intermediaries between the server and website such as load balancers and firewalls. These attacks target the vulnerabilities in

Layer 3 and Layer 4. One example of this type of attack is Smurf DDOS. The strength of these attacks is measured in packets per second (pps).

Volumetric Attack:

Volumetric attacks are the most common attacks where the attackers try to flood a website traffic clogging up the available bandwidth. One example of volume based attack is UDP flood where the attacker overwhelms random ports so that more number of UDP packets are answered and the system is unable to handle the volume of requests and thus becomes unresponsive. The strength of these attacks is measured in bits per second (bps).

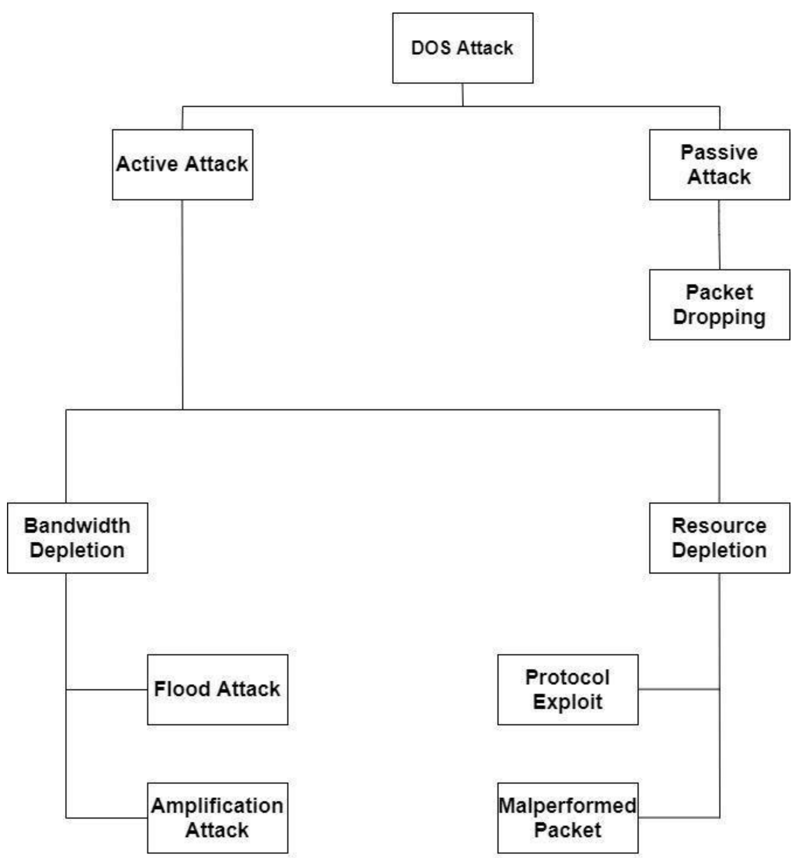

Fig 1.2 Types of DOS attacks

CHAPTER 2

RELATED WORKS

There are many methodologies and schemes proposed for the Detection of DOS Attacks. Carl proposed detection techniques like Activity profiling, wavelet based signal analysis and the testing results provided an insight into our ability to successfully identify DOS flooding attacks. Doshi came up with new techniques which can automatically detect consumer IoT attack traffic. His work demonstrated that using IoT-specific network behaviours limited the number of end points, feature selection can result in high accuracy DDOS Detection in IoT network traffic with various machine learning algorithms. The results indicated that home gateway routers or network intermediates could automatically detect DDOS attacks using low-cost machine learning algorithms. Yuan proposed a Deep Learning approach which can automatically extract high level features from low level ones and gain powerful representation. He designed a recurrent deep neural network to learn patterns from sequences of network traffic and trace network attack activities. The results demonstrated a better performance of our model compared with the previous models.

Ujjan proposed Sampled flow and adaptive polling based sampling with Snort Intrusion Detection (IDS) and deep learning based model. The flexible decoupling property of SDN enables us to program network devices and the evaluation of the proposed system demonstrates higher detection accuracy with 95% of True Positive rate with less than 4% False positive rate. Lee proposed a method for proactive detection of DDOS attack by exploiting its architecture which consists of the selection of handlers and agents, the communication and compromise, and attack. The results showed that each phase of attack scenario is partitioned well and precursors of the DDOS and the attack itself can be detected. Chen proposed a new spectral template-matching approach to countering shrew DDOS Attacks. The method proposed calls for collaborative detection and filtering of shrew DDOS attacks and was implemented with NS-2 simulator. The proposed method retains 99% of legitimate TCP flows and achieves 95% successful detection. Zhong proposed a DDOS attack detection model based on a data mining algorithm. He used FCM Cluster and Apriori association algorithm to extract network traffic model and network packet protocol status model. The results showed that the attacks can be detected efficiently and swiftly.

CHAPTER 3

PROPOSED WORK

The proposed architecture for DDOS Anomaly Detection consists of three phases:

i) Feature Extraction and Building Dataset

ii) Classifier for Detection.

iii) Algorithm Gradient Boosting

Feature Extraction and Building Dataset:

The dataset used to train our model is extracted from three open data sets published by CIC Canada. The actual datasets contain traffic captured using various attack tools and only DDOS and DOS traffic were extracted. Multiple datasets are combined to simulate the real time traffic. An unbalanced dataset is created to simulate real time attack volume proportion

Classifier for Detection:

Stochastic Gradient Boosting (SGB) is one of the most powerful algorithms for building predictive models. It is an ensemble learning method, which combines the predictive power of individual models to boost the accuracy of the final model. SGB is used to detect DDOS attacks. The main idea of boosting is to add new models sequentially. Boosting starts with a decision tree with a lesser number of splits and sequentially boosts its performance to build new trees.

Boosting trees are grown sequentially and each tree is grown using information from the previous trees to improve the performance. Bagging is another type of ensemble model that uses Decision Trees. Decision trees are used as base models of the ensemble. Boosting operation is performed as follows:

1) For the given dataset first, the initial model (F_0) naively predicts the label γ which results in error or residual $F_0 - y$

2) New model h_1, instead of predicting the label of actual data points, it tries to fit residual in the step1. At this step overall model can be formulated as

$$F_1 = F_0 + h_1$$

Where F_1 is a boosted version of F_0.

3) Now error in the previous step is F_1-y, and h_2 is the function which tries to fit residual. F_1 & h_1 can be combined to give function F_2 which results in a better version of F_1.

$$F_2 = F_1 + h_2$$

This process is repeated until the desired accuracy is achieved. After n iterations,

$$F_n = F_{n-1} + h_n$$

In boosting operation "h_n" tries to fit residual or Loss function, but in gradient boosting it fits gradient loss of function.

Algorithm Gradient Boosting:

procedure boosting

fit estimator F^1

for i in [1,M] //M

$Loss^i = L(y_i, F(X_i))$

calculate neg gradient: $-(\partial L^i/\partial X^j) = -(2/n) * (Y_j - F^i(X_j)) \ \forall \ i$

Fit a weak estimator H^i on $(X, \partial L/\partial X)$

// P changes the step size

Prediction: $F^M(X) = F^i(X) + P*h^i(X) = F^1 + P * \sum^m h^i(X)$

In the above algorithm (Gradient Boosting) for each additive base learner is trained with the residual dataset over full data points but in the case of stochastic gradient boosting algorithm, at each iteration a subsample of the training data is drawn at random (without replacement) from the full training dataset. The randomly selected subsample is then used, instead of the full sample, to fit the base learner.

CHAPTER 4

SOURCE CODE

Stochastic Gradient Boosting Unbalanced:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from xgboost import plot_importance

from numpy import loadtxt

from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import f1_score

from sklearn.metrics import precision_score

import pickle

from tqdm import tqdm

from sklearn.metrics import confusion_matrix

from matplotlib import pyplot

from sklearn.metrics import recall_score

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.manifold import TSNE
```

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

train=pd.read_csv("/home/hybrid/unbalaced_20_80_dataset.csv",index_col=0)

Y=train

print(Y.shape)

total = len(Y)*1.

ax=sns.countplot(x="Label", data=Y)

for p in ax.patches:

        ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5))

ax.yaxis.set_ticks(np.linspace(0, total, 2))

ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))

plt.show()

pd.set_option('display.max_columns',85)

train.sample(n=5)

from sklearn import preprocessing

for f in train.columns:

    if train[f].dtype=='object':

        label = preprocessing.LabelEncoder()

        label.fit(list(train[f].values))

        train[f] = lbl.transform(list(train[f].values))

train.fillna((-999), inplace=True)

train=np.array(train)
```

```
train = train.astype(float)
Y = train['Label']
X = train.drop("Label",axis=1)
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model =XGBClassifier(max_depth=5,learning_rate=0.2,subsample=0.4,colsample_bytree=1.0,c
olsample_bylevel=0.1,n_estimators=200,n_jobs=-1)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy:",accuracy)
f1score=f1_score(y_test, y_pred)
print("f1-acore:",f1score)
cm=confusion_matrix(y_test, y_pred)
print("confusion matrix:\n",cm)
pr=precision_score(y_test,y_pred)
print("Precision:",pr)
rs=recall_score(y_test,y_pred)
print("Recall_score:",rs)
misclassified_samples = X_test[y_test != y_pred]
mc=misclassified_samples.shape[0]
print("Misclassified :",mc)
```

```
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_pred, y_test)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC SGB (XGBOOST)(Imbalanced dataset)')

plt.legend(loc="lower right")

plt.show()

from xgboost import plot_importance

plot_importance(model, ax=None, height=0.1, xlim=None, ylim=None, title='Feature importanc
e by SGB(XGBOOST) Imbalanced dataset', xlabel='F score', ylabel='Features', importance_type
='weight', max_num_features=10, grid=True,)

pyplot.show()

model = XGBClassifier(max_depth=5,learning_rate=0.2,subsample=0.4,colsample_bytree=1.0,c
olsample_bylevel=0.1,n_estimators=200,n_jobs=-1)

model.fit(X_train, y_train)
```

Stochastic Grading Boosting Balanced:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import dask.dataframe as dd

from xgboost import plot_importance

from numpy import loadtxt

from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import f1_score

from sklearn.metrics import precision_score

import pickle

from tqdm import tqdm

from sklearn.metrics import confusion_matrix

from matplotlib import pyplot

from sklearn.metrics import recall_score

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import GridSearchCV

import warnings; warnings.simplefilter('ignore')

import matplotlib.pyplot as plt

import seaborn as sns
```

```
from sklearn.manifold import TSNE

import sklearn

print(sklearn.__version__)

train = pd.read_csv("/home/hybrid/final_dataset.csv",index_col=0,low_memory=False)

Y=train

#print(Y)

print(Y.shape)

total = len(Y)*1.

ax=sns.countplot(x="Label", data=Y)

for p in ax.patches:

        ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5))

ax.yaxis.set_ticks(np.linspace(0, total, 2))

ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))

plt.show()

pd.set_option('display.max_columns',85)

train.sample(n=5)

from sklearn import preprocessing

for f in train.columns:

    if train[f].dtype=='object':

        label = preprocessing.LabelEncoder()

        label.fit(list(train[f].values))

        train[f] = lbl.transform(list(train[f].values))

train.fillna((-999), inplace=True)
```

```
train=np.array(train)

train = train.astype(float)

Y = train['Label']

X = train.drop("Label",axis=1)

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)

model = XGBClassifier(n_jobs=-1)

model.fit(X_train, y_train)

para1={'subsample': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0]}

para2={'colsample_bytree': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0]}

para3={'colsample_bylevel':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0]}

model = XGBClassifier(n_jobs=-1)

model_t = GridSearchCV(model, param_grid=para1,cv=3,verbose=True)

model_t.fit(X_train, y_train)

print("Best Hyper Parameters:",model_t.best_params_)

model = XGBClassifier(n_jobs=-1)

model_2 = GridSearchCV(model, param_grid=para2,cv=3,verbose=3)

model_2.fit(X_train, y_train)

print("Best Hyper Parameters:",model_2.best_params_)


model = XGBClassifier(n_jobs=-1)

model_3 = GridSearchCV(model, param_grid=para3,cv=3,verbose=3)
```

```
model_3.fit(X_train, y_train)

print("Best Hyper Parameters:",model_3.best_params_)

model = XGBClassifier(n_jobs=-1)

model_4 = GridSearchCV(model, param_grid=para5,cv=3,verbose=3,scoring='accuracy')

model_4.fit(X_train, y_train)

print("Best Hyper Parameters:",model_4.best_params_)

model = XGBClassifier(n_jobs=-1)

model_5 = GridSearchCV(model, param_grid=para4,cv=3,verbose=3,scoring='accuracy')

model_5.fit(X_train, y_train)

print("Best Hyper Parameters:",model_5.best_params_)

model =XGBClassifier(max_depth=5,learning_rate=0.2,subsample=0.4,colsample_bytree=1.0,c

olsample_bylevel=0.1,n_estimators=200,n_jobs=-1)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("accuracy:",accuracy)

f1score=f1_score(y_test, y_pred)

print("f1-acore:",f1score)

cm=confusion_matrix(y_test, y_pred)

print("confusion matrix:\n",cm)

pr=precision_score(y_test,y_pred)

print("Precision:",pr)

rs=recall_score(y_test,y_pred)
```

```
print("Recall_score:",rs)

misclassified_samples = X_test[y_test != y_pred]

mc=misclassified_samples.shape[0]

print("Misclassified :",mc)

from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_pred, y_test)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc, marker=
'.')

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic SGB(XGBOOST)')

plt.legend(loc="lower right")

plt.show()
```

Decision Tree Unbalanced:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
```

```
from xgboost import plot_importance
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
import pickle
from tqdm import tqdm
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import recall_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
train = pd.read_csv("/home/hybrid/unbalaced_20_80_dataset.csv",index_col=0,low_memory=Fa
lse)
from sklearn import preprocessing
for f in train.columns:
    if train[f].dtype=='object':
        label = preprocessing.LabelEncoder()
        label.fit(list(train[f].values))
        train[f] = lbl.transform(list(train[f].values))
```

```
train.fillna((-999), inplace=True)
train=np.array(train)
train = train.astype(float)
Y = train['Label']
X = train.drop("Label",axis=1)
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model=DecisionTreeClassifier(max_depth=5,random_state=0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy:",accuracy)
f1score=f1_score(y_test, y_pred)
print("f1-acore:",f1score)
cm=confusion_matrix(y_test, y_pred)
print("confusion matrix:\n",cm)
pr=precision_score(y_test,y_pred)
print("Precision:",pr)
rs=recall_score(y_test,y_pred)
print("Recall_score:",rs)
misclassified_samples = X_test[y_test != y_pred]
mc=misclassified_samples.shape[0]
```

```
print("Misclassified :",mc)

from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_pred, y_test)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC(Decision tree, depth=5) Imbalanced dataset')

plt.legend(loc="lower right")

plt.show()
```

Decision Tree Balanced:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from xgboost import plot_importance

from numpy import loadtxt

from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
import pickle
from tqdm import tqdm
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import recall_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
train = pd.read_csv("/home/hybrid/final_dataset.csv",index_col=0,low_memory=False)
from sklearn import preprocessing
for f in train.columns:
    if train[f].dtype=='object':
        label = preprocessing.LabelEncoder()
        label.fit(list(train[f].values))
        train[f] = lbl.transform(list(train[f].values))
train.fillna((-999), inplace=True)
train=np.array(train)
train = train.astype(float)
Y = train['Label']
X = train.drop("Label",axis=1)
```

```
seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)

model=DecisionTreeClassifier(max_depth=5,random_state=0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

from sklearn import tree

from sklearn.externals.six import StringIO

import pydot

dot_data = StringIO()

tree.export_graphviz(model, out_file=dot_data)

graph = pydot.graph_from_dot_data(dot_data.getvalue())

graph[0].write_pdf("tree.pdf")

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("accuracy:",accuracy)

f1score=f1_score(y_test, y_pred)

print("f1-acore:",f1score)

cm=confusion_matrix(y_test, y_pred)

print("confusion matrix:\n",cm)

pr=precision_score(y_test,y_pred)

print("Precision:",pr)

rs=recall_score(y_test,y_pred)
```

```
print("Recall_score:",rs)

misclassified_samples = X_test[y_test != y_pred]

mc=misclassified_samples.shape[0]

print("Misclassified :",mc)

from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_pred, y_test)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic(Decision Tree, depth=5)')

plt.legend(loc="lower right")

plt.show()
```

KNN Unbalanced:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from xgboost import plot_importance
```

```
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
import pickle
from tqdm import tqdm
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import recall_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
train = pd.read_csv("/home/hybrid/unbalaced_20_80_dataset.csv",index_col=0,low_memory=Fa
lse)
from sklearn import preprocessing
for f in train.columns:
    if train[f].dtype=='object':
        label = preprocessing.LabelEncoder()
        label.fit(list(train[f].values))
        train[f] = lbl.transform(list(train[f].values))
```

```
train.fillna((-999), inplace=True)

train=np.array(train)

train = train.astype(float)

Y = train['Label']

X = train.drop("Label",axis=1)

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)

model=KNeighborsClassifier(n_neighbors=6,algorithm='kd_tree',n_jobs=25)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("accuracy:",accuracy)

f1score=f1_score(y_test, y_pred)

print("f1-acore:",f1score)

cm=confusion_matrix(y_test, y_pred)

print("confusion matrix:\n",cm)

pr=precision_score(y_test,y_pred)

print("Precision:",pr)

rs=recall_score(y_test,y_pred)

print("Recall_score:",rs)

misclassified_samples = X_test[y_test != y_pred]

mc=misclassified_samples.shape[0]
```

```
print("Misclassified :",mc)

from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_pred, y_test)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC (KNN, n=6) Imbalanced dataset')

plt.legend(loc="lower right")

plt.show()
```

KNN Balanced:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from xgboost import plot_importance

from numpy import loadtxt

from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import f1_score

from sklearn.metrics import precision_score

import pickle

from tqdm import tqdm

from sklearn.metrics import confusion_matrix

from sklearn.metrics import recall_score

from sklearn.neighbors import NearestNeighbors

from sklearn.neighbors import KNeighborsClassifier

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

train = pd.read_csv("/home/hybrid/final_dataset.csv",index_col=0,low_memory=False)

from sklearn import preprocessing

for f in train.columns:

    if train[f].dtype=='object':

        label = preprocessing.LabelEncoder()

        label.fit(list(train[f].values))

        train[f] = lbl.transform(list(train[f].values))

train.fillna((-999), inplace=True)

train=np.array(train)

train = train.astype(float)


Y = train['Label']
```

```
X = train.drop("Label",axis=1)
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model=KNeighborsClassifier(n_neighbors=6,algorithm='kd_tree',n_jobs=25)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy:",accuracy)
f1score=f1_score(y_test, y_pred)
print("f1-acore:",f1score)
cm=confusion_matrix(y_test, y_pred)
print("confusion matrix:\n",cm)
pr=precision_score(y_test,y_pred)
print("Precision:",pr)
rs=recall_score(y_test,y_pred)
print("Recall_score:",rs)
misclassified_samples = X_test[y_test != y_pred]
mc=misclassified_samples.shape[0]
print("Misclassified :",mc)
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_pred, y_test)
roc_auc = auc(fpr, tpr)
```

```
plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic(KNN, n=6)')

plt.legend(loc="lower right")

plt.show()
```

Naive Bayes Unbalanced:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from xgboost import plot_importance

from numpy import loadtxt

from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import f1_score

from sklearn.metrics import precision_score

import pickle
```

```
from tqdm import tqdm
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import recall_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
train = pd.read_csv("/home/hybrid/unbalaced_20_80_dataset.csv",index_col=0,low_memory=Fa
lse)
train.size
from sklearn import preprocessing
for f in train.columns:
    if train[f].dtype=='object':
        label = preprocessing.LabelEncoder()
        label.fit(list(train[f].values))
        train[f] = lbl.transform(list(train[f].values))
train.fillna((-999), inplace=True)
train=np.array(train)
train = train.astype(float)
Y = train['Label']
X = train.drop("Label",axis=1)
seed = 7
```

```
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)

model=BernoulliNB(binarize=0.0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("accuracy:",accuracy)

f1score=f1_score(y_test, y_pred)

print("f1-acore:",f1score)

cm=confusion_matrix(y_test, y_pred)

print("confusion matrix:\n",cm)

pr=precision_score(y_test,y_pred)

print("Precision:",pr)

rs=recall_score(y_test,y_pred)

print("Recall_score:",rs)

misclassified_samples = X_test[y_test != y_pred]

mc=misclassified_samples.shape[0]

print("Misclassified :",mc)

from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_pred, y_test)

roc_auc = auc(fpr, tpr)

plt.figure()
```

```
plt.plot(fpr,tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc, marker='
.')
plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
plt.xlim([0.0, 1])
plt.ylim([0.0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Naive Bayes Bernoulli (Imbalanced dataset)  ')
plt.legend(loc="lower right")
plt.show()
```

Naive Bayes Balanced:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from xgboost import plot_importance
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
import pickle
from tqdm import tqdm
```

```
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import recall_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
train = pd.read_csv("/home/hybrid/final_dataset.csv",index_col=0,low_memory=False)
from sklearn import preprocessing
for f in train.columns:
    if train[f].dtype=='object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(train[f].values))
        train[f] = lbl.transform(list(train[f].values))
train.fillna((-999), inplace=True)
train=np.array(train)
train = train.astype(float)
Y = d0['Label']
X = d0.drop("Label",axis=1)
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model=BernoulliNB(binarize=0.0)
```

```
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("accuracy:",accuracy)

f1score=f1_score(y_test, y_pred)

print("f1-acore:",f1score)

cm=confusion_matrix(y_test, y_pred)

print("confusion matrix:\n",cm)

pr=precision_score(y_test,y_pred)

print("Precision:",pr)

rs=recall_score(y_test,y_pred)

print("Recall_score:",rs)

misclassified_samples = X_test[y_test != y_pred]

mc=misclassified_samples.shape[0]

print("Misclassified :",mc)

from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_pred, y_test)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic(Naive Bayes-Bernoulli)')
plt.legend(loc="lower right")
plt.show()
```

Random Forest Unbalanced:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from xgboost import plot_importance
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
import pickle
from tqdm import tqdm
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import recall_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
train = pd.read_csv("/home/hybrid/unbalaced_20_80_dataset.csv",index_col=0,low_memory=Fa
lse)
from sklearn import preprocessing
for f in train.columns:
    if train[f].dtype=='object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(train[f].values))
        train[f] = lbl.transform(list(train[f].values))
train.fillna((-999), inplace=True)
train=np.array(train)
train = train.astype(float)
Y = train['Label']
X = train.drop("Label",axis=1)
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model = RandomForestClassifier(n_estimators=100,max_depth=5,random_state=0,n_jobs=-1)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print("accuracy:",accuracy)
f1score=f1_score(y_test, y_pred)
print("f1-acore:",f1score)
cm=confusion_matrix(y_test, y_pred)
print("confusion matrix:\n",cm)
pr=precision_score(y_test,y_pred)
print("Precision:",pr)
rs=recall_score(y_test,y_pred)
print("Recall_score:",rs)
misclassified_samples = X_test[y_test != y_pred]
mc=misclassified_samples.shape[0]
print("Misclassified :",mc)
import pandas as pd
feature_importances = pd.DataFrame(model.feature_importances_,
                                   index = X_train.columns,
                                    columns=['importance']).sort_values('importance', ascending=False)
feature_importances[0:10]
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_pred, y_test)
roc_auc = auc(fpr, tpr)
plt.figure()
```

```
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC(Random Forest,depth=5) Imbalanced dataset')

plt.legend(loc="lower right")

plt.show()
```

Random Forest Balanced:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import make_classification

from xgboost import plot_importance

from numpy import loadtxt

from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import f1_score

from sklearn.metrics import precision_score
```

```
import pickle
from tqdm import tqdm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
train = pd.read_csv("/home/hybrid/final_dataset.csv",index_col=0,low_memory=False)
from sklearn import preprocessing
for f in train.columns:
    if train[f].dtype=='object':
        label = preprocessing.LabelEncoder()
        label.fit(list(train[f].values))
        train[f] = lbl.transform(list(train[f].values))
train.fillna((-999), inplace=True)
train=np.array(train)
train = train.astype(float)
Y = train['Label']
X = train.drop("Label",axis=1)
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model = RandomForestClassifier(n_estimators=100,max_depth=5,random_state=0,n_jobs=-1)
```

```
model.fit(X_train, y_train)


y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy:",accuracy)
f1score=f1_score(y_test, y_pred)
print("f1-acore:",f1score)
cm=confusion_matrix(y_test, y_pred)
print("confusion matrix:\n",cm)
pr=precision_score(y_test,y_pred)
print("Precision:",pr)
rs=recall_score(y_test,y_pred)
print("Recall_score:",rs)
misclassified_samples = X_test[y_test != y_pred]
mc=misclassified_samples.shape[0]
print("Misclassified :",mc)
import pandas as pd
feature_importances = pd.DataFrame(model.feature_importances_,
                                   index = X_train.columns,
                                    columns=['importance']).sort_values('importance', ascending=False)
feature_importances[0:10]
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_pred, y_test)
```

```
roc_auc = auc(fpr, tpr)


plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic(Random Forest,depth=5)')

plt.legend(loc="lower right")

plt.show()
```

CHAPTER 5

RESULTS

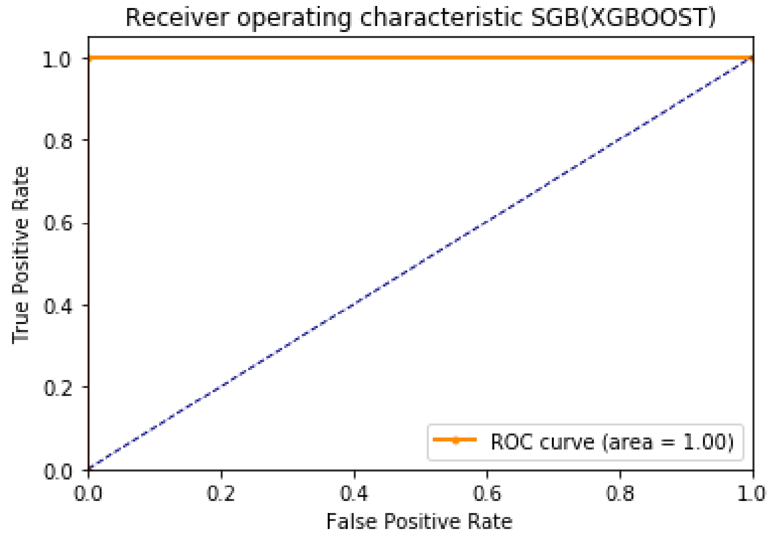

Fig 5.1 ROC for SGB Balanced

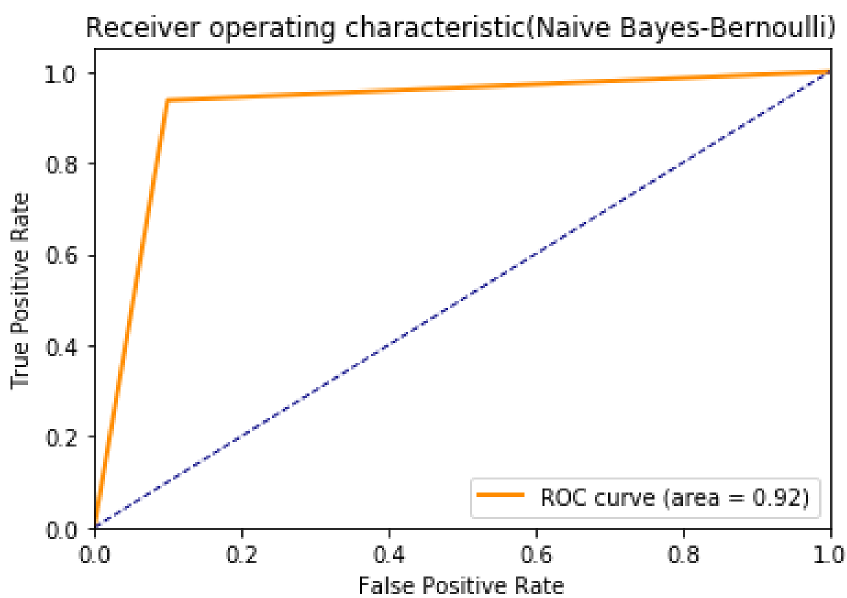

Fig 5.2 ROC for Naive Bayes Balanced

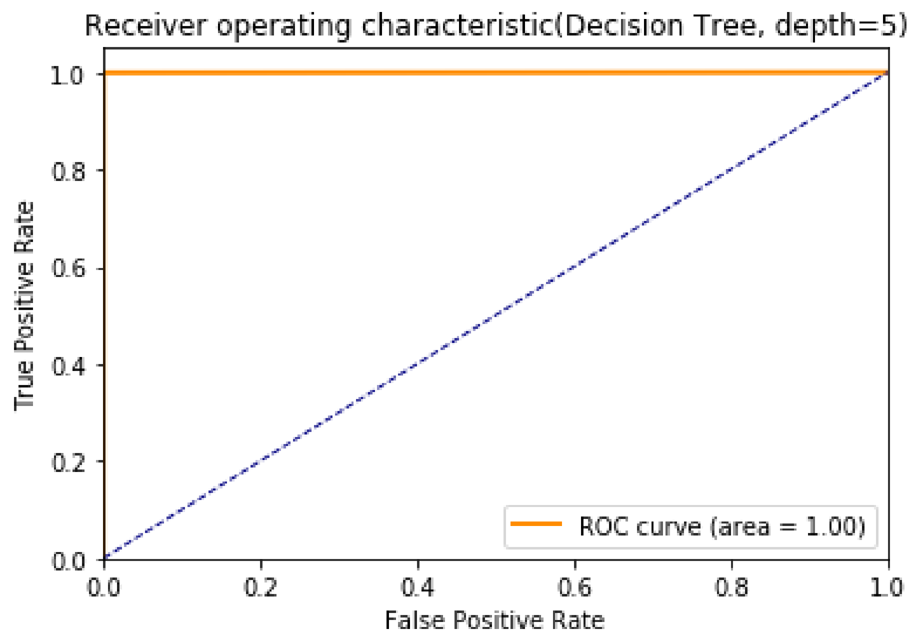

Fig 5.3 ROC for Decision Tree Balanced.

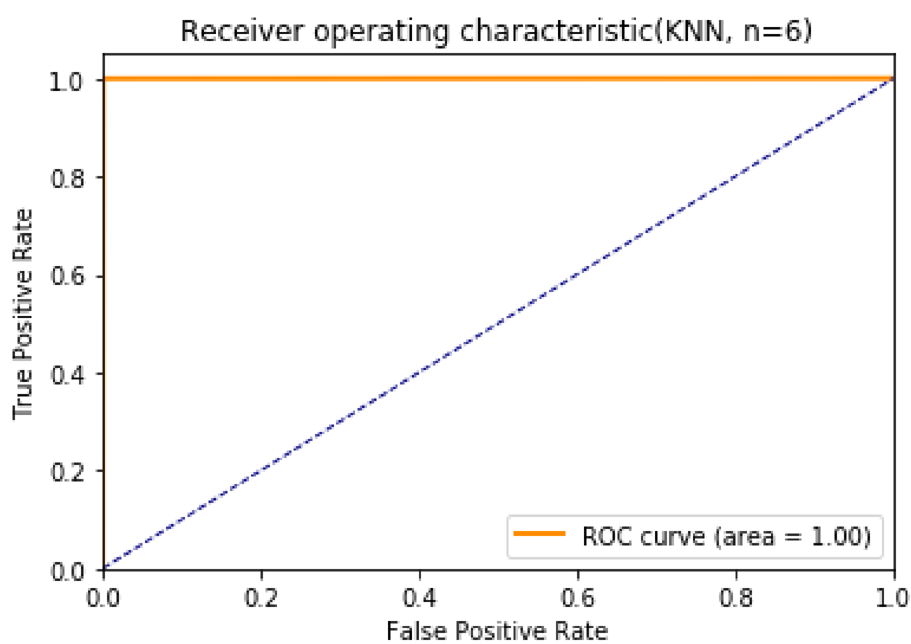

Fig 5.4 ROC for K-NN Balanced.

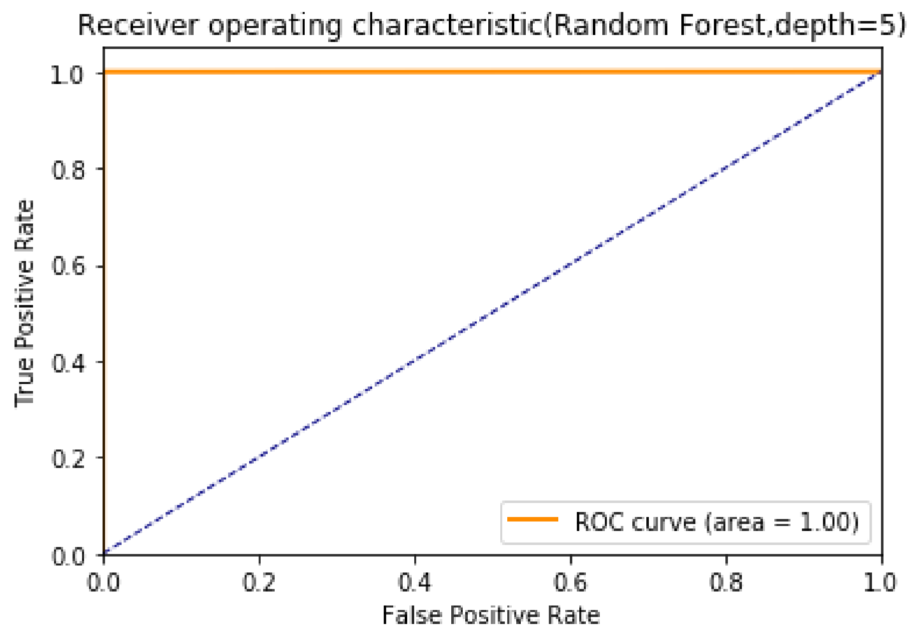

Fig 5.5 ROC for Random Forest Balanced.

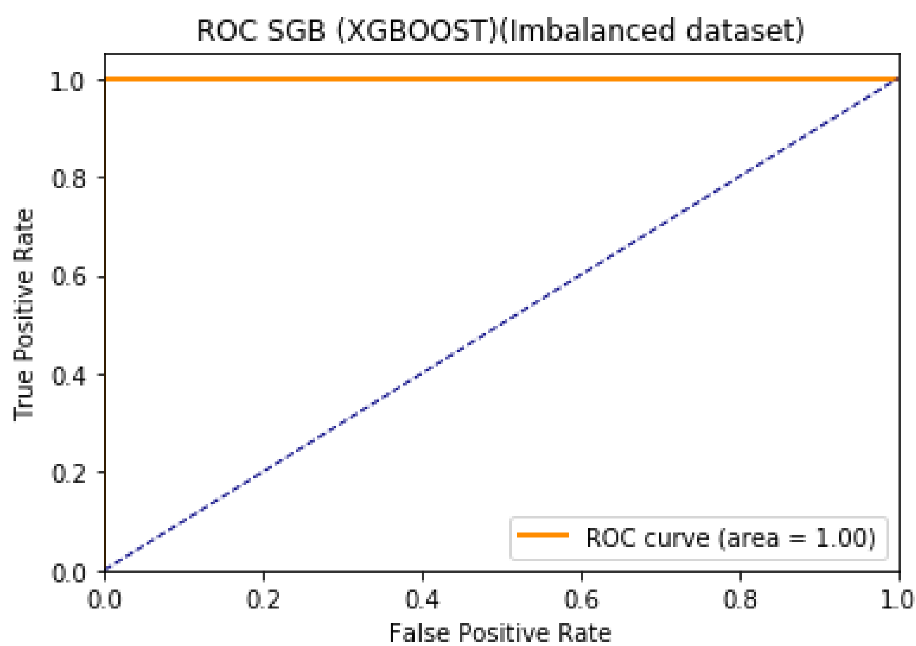

Fig 5.6 ROC for SGB Unbalanced

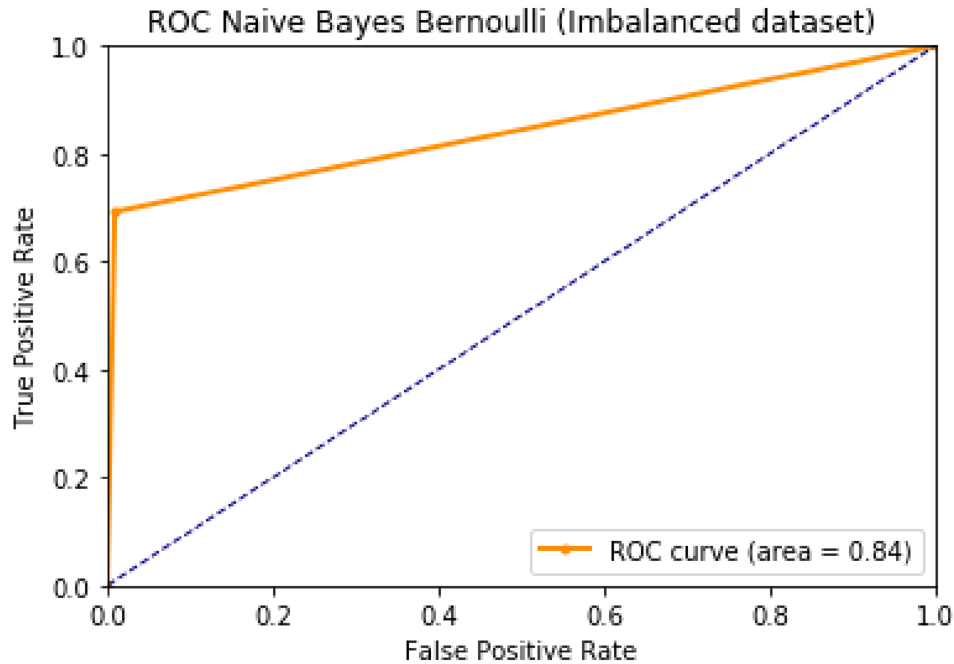

Fig 5.7 ROC for Naive Bayes Unbalanced

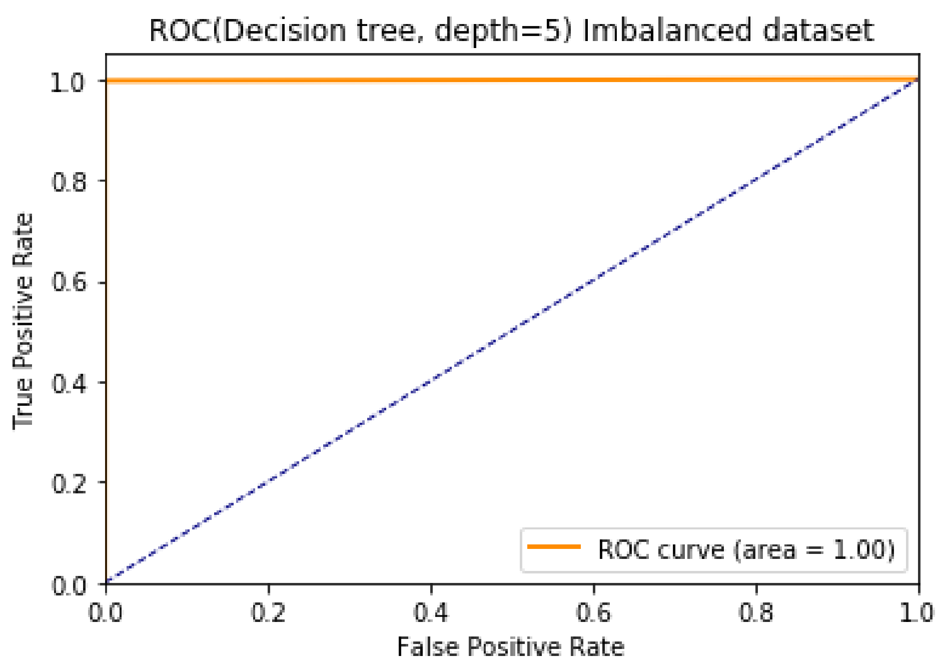

Fig 5.8 ROC for Decision Tree Unbalanced

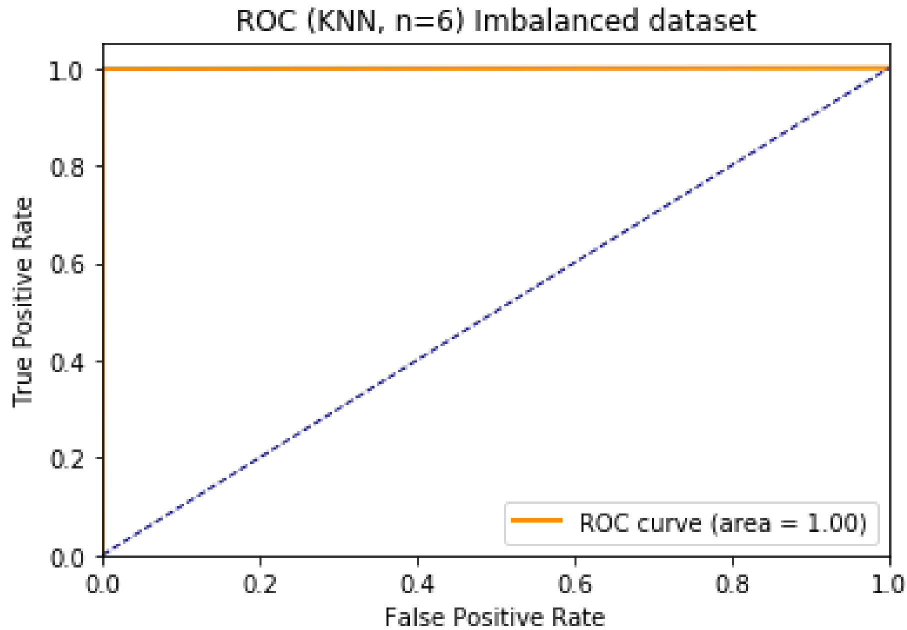

Fig 5.9 ROC for K-NN Unbalanced

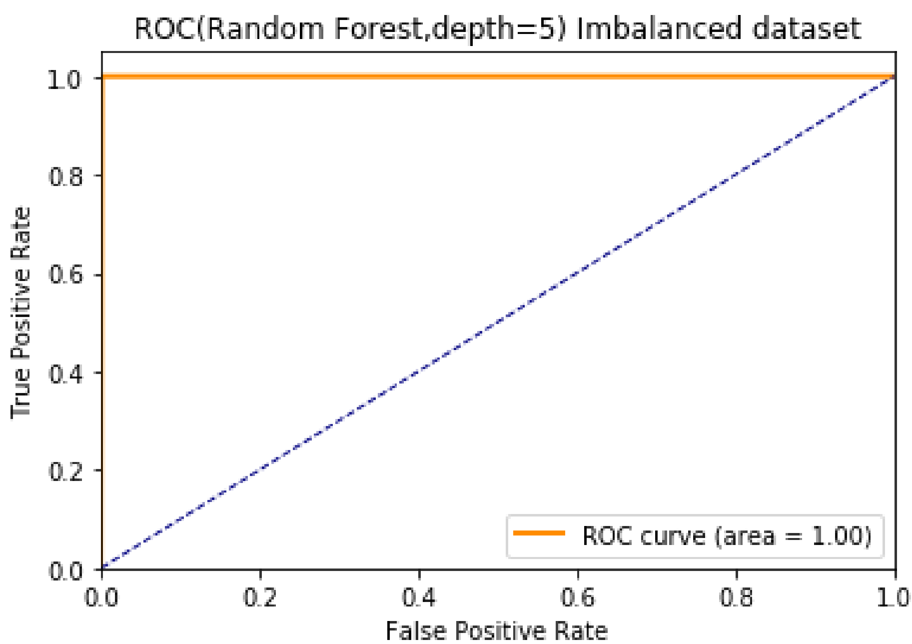

Fig 5.10 ROC for Random Forest Unbalaned

CHAPTER 6

PERFORMANCE EVALUATION

In this section, we give a detailed performance analysis on our Stochastic Grading Boosting Algorithm and show a comparison with other Machine Learning Models like Naive Bayes, K-NN, Decision Tree and Random Forest. The above algorithms are implemented using the Scikit-Learn python library. The machine learning models are trained over balanced and unbalanced dataset and accuracy, Precision, Confusion Matrix F1 score and results are evaluated to compare the proposed model with different machine learning models. Receiver Operating Characteristic (ROC) is plotted which illustrates the diagnostic ability of a binary classifier system as the threshold varies.

Table 6.1 Results over Unbalanced Dataset

Metric	SGB	RF	DT	K-NN	NB
Accuracy	100	99.93	99.67	99.94	92.07
F1-Score	100	99.72	99.64	99.83	80.91
Precission	100	99.97	99.67	99.85	70.43
Recall	100	99.43	99.65	99.79	95.98
Confusion Matrix	[[2086364 0] [0 4270 84]]	[[2085725 736] [978 213 4788]]	[[2084946 1418] [334 426 750]]	[[2085745 619] [859 426 225]]	[[1904428 1 81936] [17213 40 9871]]
Total Misclassifications	0	2931	2053	1721	199149
Execution time (min)	12 min	3 min	2 min	1.2 hrs	2 min

Table 6.2 Results over Balanced dataset

Metric	SGB	RF	DT	K-NN	NB
Accuracy	100	99.97	99.95	99.93	91.56
F1-Score	100	99.96	99.98	99.95	91.34
Precission	100	99.94	99.96	99.96	91.45
Recall	100	99.97	99.94	99.94	89.01
Confusion Matrix	[[2086461 0] [0 2135 766]]	[[2085725 736] [978 213 4788]]	[[2084882 1579] [756 213 5010]]	[[2085311 1150] [1145 213 4621]]	[[1958145 1 28316] [217195 19 18571]]
Total Misclassifications	0	1987	2415	2287	345511
Execution time (min)	10 min	4 min	3 min	5 hrs	3 min

Stochastic Gradient Boosting algorithm is implemented using an open source software library XGBOOST, which simulates a gradient boosting algorithm with default parameters. Besides these parameters, the maximum number of trees in the ensemble, depth of each tree and learning rate of gradient also need to be set with optimal values. These parameters can be set by specifying values for “n_estimators”, “max_depth” and “learning_rate” respectively. For tuning these hyper-parameters, we used Grid search technique with 3-fold cross-validation. The final Model is implemented with the tuned hyper-parameters. The area under the curve and ROC is the same for RF, DT, K-NN for balanced and unbalanced datasets. The total misclassifications are higher compared to other models. Based on the above analysis, for the same hyper-parameters, Decision Tree was found to have better metrics compared to RF. SGB performance is unchanged in balanced and unbalanced datasets with zero misclassifications.

CHAPTER 7

CONCLUSION AND FUTURE WORKS

CONCLUSION

In this paper, a methodology to detect DDOS attacks by using a Stochastic Gradient boosting algorithm was proposed. The proposed model was trained over the hybrid dataset extracted and the results are compared with other machine algorithms. It can be inferred that SGB outpassed K-NN, Decision trees, Random forest, and Naive Bayes by achieving 100% performance metrics. The proposed model is also trained on unbalanced dataset to simulate real-time traffic where SGB achieved zero misclassifications whereas metrics of other models are recorded lower compared to that of balanced dataset. Finally SGB can be used to automate DDOS detection without any human interruption.

FUTURE WORKS

The proposed scheme can be extended by using IDS where the rules will filter out the traffic. Feature selection can be improved by removing the least contributing features. XGBoost can be deployed with a cluster computing framework Spark in a multi-node cluster setup to achieve high latency.

CHAPTER 8

REFERENCES .

(An, 2017) Antonakakis.M, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in Proc. of USENIX Security Symposium, 2017.

(Bo, 2013) Bogdanoski, M., Suminoski, T., & Risteski, A. (2013). Analysis of the SYN flood DOS attack. International Journal of Computer Network and Information Security (IJCNIS), 5(8), 1-11.

(Ca, 2006) Carl, G., Kesidis, G., Brooks, R. R., & Rai, S. (2006). Denial-of-service attack-detection techniques. *IEEE Internet computing*, *10*(1), 82-89.

(Ch, 2006) Chen, Y., & Hwang, K. (2006). Collaborative detection and filtering of shrew DDOS attacks using spectral analysis. Journal of Parallel and Distributed Computing, 66(9), 1137-1151.

(Do, 2004) Douligeris, C., & Mitrokotsa, A. (2004). DDOS attacks and defense mechanisms: classification and state-of-the-art. Computer Networks, 44(5), 643-666.

(Do, 2018) DOShi, R., Apthorpe, N., & Feamster, N. (2018, May). Machine learning DDOS detection for consumer internet of things devices. In 2018 IEEE Security and Privacy Workshops (SPW) (pp. 29-35). IEEE.

(Gi, 2015) Girma, A., Garuba, M., Li, J., & Liu, C. (2015, April). Analysis of DDOS attacks and an introduction of a hybrid statistical model to detect DDOS attacks on cloud computing environment. In 2015 12th International Conference on Information Technology-New Generations (pp. 212-217). IEEE.

(Ji, 2004) Jin, S., & Yeung, D. S. (2004, June). A covariance analysis model for DDOS attack detection. In 2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577) (Vol. 4, pp. 1882-1886). IEEE.

(Le, 2008) Lee, K., Kim, J., Kwon, K. H., Han, Y., & Kim, S. (2008). DDOS attack detection method using cluster analysis. Expert systems with applications, 34(3), 1659-1665.

(Li, 2018) Li, C., Wu, Y., Yuan, X., Sun, Z., Wang, W., Li, X., & Gong, L. (2018). Detection and defense of DDOS attack–based on deep learning in OpenFlow-based SDN. International Journal of Communication Systems, 31(5), e3497.

(Mi, 2004) Mirkovic, J., & Reiher, P. (2004). A taxonomy of DDOS attack and DDOS defense mechanisms. ACM SIGCOMM Computer Communication Review, 34(2), 39-53

(Na, 2015) Nagpal, B., Sharma, P., Chauhan, N., & Panesar, A. (2015, March). DDOS tools: Classification, analysis and comparison. In 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 342-346). IEEE.

(Ni, 2016) Niyaz, Q., Sun, W., & Javaid, A. Y. (2016). A deep learning based DDOS detection system in software-defined networking (SDN). arXiv preprint arXiv:1611.07400.

(Uj, 2020) Ujjan, R. M. A., Pervez, Z., Dahal, K., Bashir, A. K., Mumtaz, R., & González, J. (2020). Towards sFlow and adaptive polling sampling for deep learning based DDOS detection in SDN. Future Generation Computer Systems, 111, 763-779.

(Ya, 2016) Yadav, S., & Subramanian, S. (2016, March). Detection of Application Layer DDOS attack by feature learning using Stacked AutoEncoder. In 2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT) (pp. 361-366). IEEE.

(Yu, 2017) Yuan, X., Li, C., & Li, X. (2017, May). DeepDefense: identifying DDOS attack via deep learning. In 2017 IEEE International Conference on Smart Computing (SMARTCOMP) (pp. 1-8). IEEE.

(Za, 2013) Zargar, S. T., Joshi, J., & Tipper, D. (2013). A survey of defense mechanisms against distributed denial of service (DDOS) flooding attacks. IEEE communications surveys & tutorials, 15(4), 2046-2069

(Zh, 2010) Zhong, R., & Yue, G. (2010, April). DDOS detection system based on data mining. In Proceedings of the 2nd International Symposium on Networking and Network Security, Jinggangshan, China (pp. 2-4).