Project Report

Of

DATA MINING AND DATA WAREHOUSING

(CSPC-308)

Submitted To:
Dr. Nonita Sharma
CSE Department

DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY JALANDHAR

TEAM INFORMATION

SNO	NAME	ROLL NO	CONTACT
1.	RAMAVATH SAIKUMAR	18103077	ramavaths.cs.18@nitj.ac.in
2.	ROHIT MITTAL	18103081	rohitm.cs.18@nitj.ac.in
3.	SHOBHIT TEWARI	18103086	shobhitt.cs.18@nitj.ac.in
4.	VIKAS SANDHU	18103094	vikas.cs.18@nitj.ac.in

DETECTION AND CLASSIFACTION
OF
DDoS ATTACKS

Abstract

In recent years, with increase in internet technology and users, the advanced threats against Cyber–Physical Systems (CPSs), attacks, are also increasing. And it can be observed that DDoS attacks contribute to the majority of overall network attacks. Networks face challenges in distinguishing between legitimate and malicious flows. The testing and implementation of DDoS strategies are not easy to deploy due to many factors like complexities, rigidity, cost, and vendor specific architecture of current networking equipment and protocols. Work is being done to detect DDoS attacks by application of Machine Learning (ML) models but to find out the best ML model among the given choices, is still an open question. This paper is motivated by the question: "Which super vised learning algorithm will give the best outcomes to detect DDoS attacks"? We used various ML techniques (Decision Tree, Random Forest, Naïve Bayes, K Nearest Neighbor and Gradient Boost) on the dataset and are able to get the maximum accuracy of 98.84%.

Keywords: Classification, Supervised Learning, DDoS detection, DDoS attack, security, Network Threats, KNN, Gradient Boost, Random Forest Classifier, Naïve Bayes, TCP SYN, UDP, MSSQL

TABLE OF CONTENTS

1. INTRODUCTION

In today's world all the important organizations such as defense, research, government organizations and nuclear plants depends highly on computer networks. Distributed Denial of Service (DDoS) attack is a menace to network security that aims at exhausting the target networks with malicious traffic. DDoS attack the attacker uses the Botnet-based Computers which are infected by Malware and fully controlled by the attacker to not only choke the computer resource but also available bandwidth and hence bringing the network to halt for some time.

This can cause loss of billions to service providing companies (It has been reported that a heavy DDoS attack can cause a loss reaching $100, 000 per hour for some organizations along with damaging the trust of its customers) and even can cause country's defense huge threats and problems. The most dangerous thing about DDoS attacks is that these can't be detected with the traditional strategies such as Signature-Based Intrusion as here the attacker uses the botnet computers to attack.

These reasons have forced us to adopt such strategies which can deal with such situations and losses. But the testing and implementation of DDoS strategies are not easy to deploy due to complexities, rigidity, cost, and vendor specific architecture of current networking equipment and protocols. Hence application of Machine Learning comes into play which is the aim of our paper.

In this paper we have analyzed different machine learning approaches namely Decision Tree, Random Forest, Naïve Bayes, K-Nearest Neighbor, Gradient Boost to find which approach works the best in detecting and classifying the different DDoS attacks.

The dataset used for this paper is **CICDDoS2019** which contains benign and the most up-to-date common DDoS attacks. The dataset mainly contains 7 types of DDoS attacks (SYN Flood, UDP Flood, UDP Lag, MSSQL, NetBIOS, LDAP, and PortMap) and benign i.e. normal flow. The dataset is collected by **Canadian Institute for Cyber Security** for studying the different attacks in 2019.Here in features extraction process from the raw data, CICFlowMeter-V3 has been used with the help of which more than 80 traffic features are extracted.

This paper consists of six sections. The second section presents a short review on DDoS attacks and its types. In the third section, under Methods, different machine learning algorithms are explained. The fourth section outlines the details of the design and implementation of the various algorithms which is followed by the comparison and discussions on the results of various algorithms. After that conclusions are made and also future scopes are discussed.

2. DDoS Attacks:

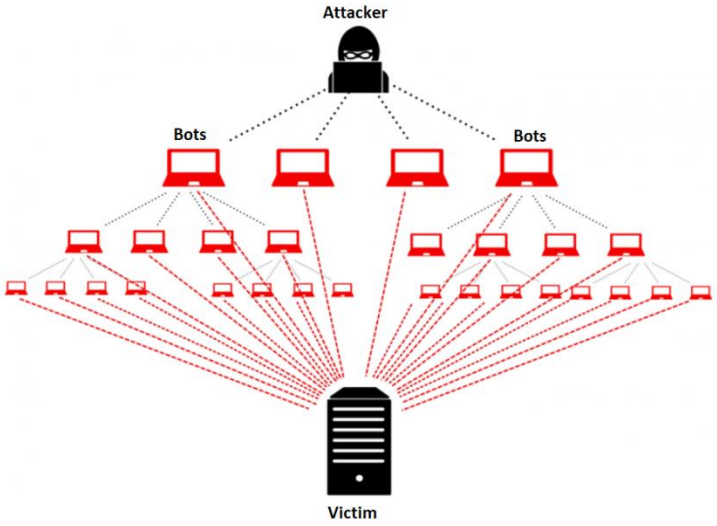
2.1 Overview:

For further discussions we must know what DDoS attacks are and what their main types are. DDoS attack refers to Distributed Denial of Service attack. It is one of the most used Denial of Service attack as it is very difficult to classify.

It is basically a malicious attempt to make an online service unavailable to users, usually by temporarily interrupting or suspending the services of its hosting server. The attacker sends the data in such a high volume that either the bandwidth completely consumes or the victims' resources get exhausted.

The attacker accomplishes this through a network of remotely controlled, hacked computers or bots. They form what is known as a "botnet". These are used to flood targeted websites, servers, and networks with more data than they can accommodate.

The botnets may send more connection requests than a server can handle or send overwhelming amounts of data that exceed the bandwidth capabilities of the targeted victim. Botnets can range from thousands to millions of computers controlled by cybercriminals. There are high chances that the owner of system acting as bot for the attacker doesn't have any idea about this.



2.2 Types of DDoS Attacks:

The DDoS attacks can mainly be classified as:

2.2.1 Volume Based Attacks:

In this type of attack the attacker tries to saturate the bandwidth of victim site and hence making the services unavailable for the victims customers. It includes UDP floods, ICMP floods etc.

2.2.2 Protocol Attacks

In this type of attack the attacker tries to use some weakness of the protocol used by the victim machine for authentication e.g. TCP uses 3way Handshake which can lead to this type of attack. e.g. SYN Floods, Ping of Death etc.

2.2.3 Application Layer Attacks

This type of attack mainly which focuses on web applications and are considered the most sophisticated and serious type of attacks. Here the aim of attacker is to crash the web server. e.g. GET/POST floods etc.

2.3 Common DDoS attacks:

The most common attacks are as follows:

2.3.1 SYN Flood

It exploits weaknesses in the TCP connection sequence, known as a three-way handshake. The host machine receives a synchronized (SYN) message to begin the "handshake." And also allocates the resources for the request and acknowledges the message by sending an acknowledgement (ACK) flag to the initial request side, but the third packet is never sent and hence the connection is never closed which leads to complete exhaust of resources and which then closes the connection. In a SYN flood, however, spoofed messages are sent and the connection doesn't close, shutting down service.

2.3.2 UDP Flood

A UDP flood targets random ports on a computer or network with UDP packets. The host checks for the application listening at those ports, but no application is found. The attacker keeps sending the packets and hence exhausting the bandwidth and hence making the services unavailable for the victim's customers.

2.3.3 UDP Lag Attack:

A UDP Lag attack is also volumetric distributed denial-of-service (DDoS) attack using the User Datagram Protocol (UDP) whose main aim is to slow down the services of the server. It is mainly used for lagging the live streaming apps and games so that users experience goes down and hence victim server suffers huge losses.

2.3.4 PortMap Attack:

The PortMap service redirects the client to the proper port number so it can communicate with the requested Remote Procedure Call (RPC) service. As several UDP-based services (DNS, NTP) before it, it's being used by attackers to hide the origin of the attack and to amplify its volume. The attacker sends a huge amount of requests to PortMap and hence the real client has to wait an infinite time to get the service of PortMap and hence leading to crash of servers.

2.3.5 LDAP Attack:

LDAP Injection is an attack used to exploit web based applications that construct LDAP statements based on user input. When an application fails to properly sanitize user input, it's

possible to modify LDAP statements using a local proxy. This could result in the execution of arbitrary commands such as granting permissions to unauthorized queries, and content modification inside the LDAP tree.

2.3.6 MSSQL Attack:

MC-SQLR lets clients identify the database instance with which they are attempting to communicate when connecting to a database server or cluster with multiple database instances. Each time a client needs to obtain information on configured MS SQL servers on the network, the SQL Resolution Protocol can be used. The server responds to the client with a list of instances. The attacker takes advantage of this and sends the requests to SQL server with spoofed IP address leading to blocking the real users from using this service.

2.3.7 NetBIOS Attack:

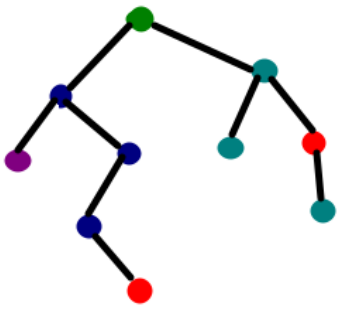
The primary purpose of NetBIOS is to allow applications on separate computers to communicate and establish sessions to access shared resources and to find each other over a local area network. The attacker sends huge amount of requests to the NetBIOS and hence leading the real users to wait more leading to halt in the services of server.

3. Methods:

There are various supervised machine learning algorithms that can be used for classification of various DDoS attacks. **Some of these are explained below:**

3.1 Decision Tree

Decision Tree is a supervised learning algorithm which is used for classification as well as regression. Its main goal is to create a training model (i.e. a set of rules) from training data and based on this model predict the target variable for given conditions. It builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets based on the attribute values while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. Leaf node represents a classification or decision.



So the important point is on what basis these subsets are made from the dataset. There are mainly two types of criteria which are information gain and gini index which are explained as below:

3.1.1 Information Gain:

Information Gain is defined as the amount of information gained from a feature about the class. In other words it calculates the reduction in entropy from transforming a dataset based on the feature. It can be calculated from entropy as the difference between the entropy of parent node and weighted average entropy of child nodes.

$$IG(S, A) = H(S) - H(S, A)$$

Where H(S) is calculated as:

$$H(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

It is most commonly used criteria for construction of decision trees from a training dataset by evaluating the information gain for each variable, and selecting the variable that maximizes the information gain, which in turn minimizes the entropy and best splits the dataset into groups for effective classification

3.1.2 Gini Index:

Gini Index calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly. So the feature with minimum gini index should be selected for construction of decision trees. It varies between values 0 and 1, where 0 expresses the purity of classification, i.e. all the elements belong to a specified class or only one class exists there i.e. Data is pure. And 1 indicates the random distribution of elements across various classes. The value of 0.5 of the Gini Index shows an equal distribution of elements over some classes.
It can be calculated by deducting the sum of squared of probabilities of each class from one. i.e.

$$\text{Gini Index} = 1 - \sum_{i=1}^{n} (P_i)^2$$

Pseudo Code:

Algorithm Decision Tree

start

∀ attributes a_1, a_2, ..., a_n

Find the attribute that best divides the training data using Information Gain and Gini Index

a_best ← the attribute with highest information gain

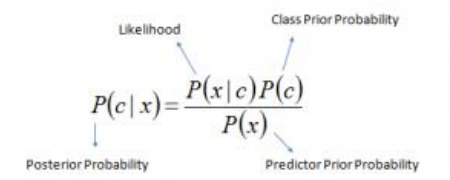
Create a decision node that splits on a_best

Recurse on the sub-lists obtained by splitting on a_best and add those nodes as children of node

end

3.2 Naïve Bayes:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is based on the assumption that all features are independent or unrelated. It is one of the simple and most effective Classification algorithms which help in building the fast machine learning models that can make quick predictions. It is probabilistic classifier, i.e. it calculates the probability of all the labels depending upon the given test case. The label with the highest probability is given as the result.
Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

Above,

- $P(c|x)$ is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

Pseudo code:

Algorithm Naïve Bayes

start

Let S = $\{a_1, a_2, ..., a_n\}$, where S = training set and a = articles:

Calculate the probability of the classes $P(C)$

Calculate likelihood of attribute A for each class $P(A|C)$

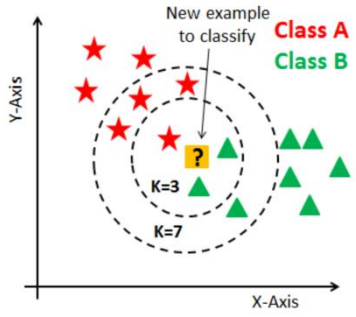
Calculate the conditional probability P(C|A)

Assign the class with the highest probability

end

3.3 K-Nearest Neighbor:

The K-Nearest Neighbor is one of the simplest algorithms used for classification. It is an instance-based classifier. When the k-NN is used, instances within a dataset are contained in a dimensional space, where a new instance is labeled based on its similarity with other instances. These instances are referred to as neighbors. A new instance is labeled x, if x is the most similar class for the neighboring observations. A distance function is applied to determine the similarity between instances. There are many distance functions e.g. Manhattan Distance, Euclidean Distance, Minkowsky Distance etc.



For the purpose of this study, the distance function employed is Euclidean. The Euclidean function is a relatively common method as it reflects the human perception of distance.

Pseudo code:

Algorithm k-Nearest Neighbour

start

Let S = $\{a_1, a_2, ..., a_n\}$, where S represents the training set and a represents article documents

$k \leftarrow$ the desired number of nearest neighbours

Compute $d(x,y)$ between new instance i and all $a \in S$

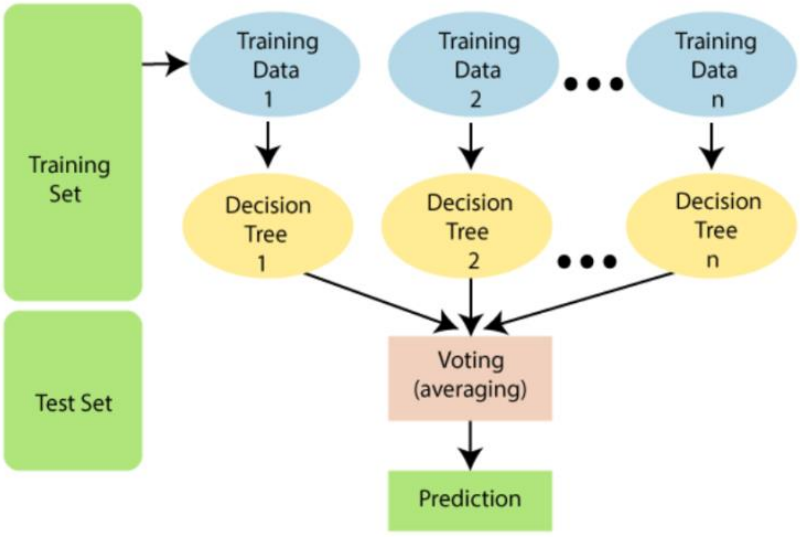
Select the k closest training samples to i

$Class_i \leftarrow$ best voted class

end

3.4 Random Forest:

Random forest is a supervised learning algorithm. The "forest" it builds, is a collection of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result. So, it is based on the principle that instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The different trees are made using random data from the datasets and using all these tress the predictions are made.



Pseudo code:

Algorithm Random Forest

Require *IDT* (a decision tree inducer), *T* (the number of iterations), *S* (the training set), μ (the subsample size), *N* (the number of attributes used in each node)

start

$t \leftarrow 1$

repeat

$S_t \leftarrow$ Sample μ instances from *S* with replacement.

Build classifier M_t using *IDT(N)* on S_t

t++

until t > T

end

3.5 Gradient Boost:

Gradient Boosting is a word made by combining both Gradient Descent and Boosting. Gradient boosting is a type of machine learning algorithm which is based on the principle that the best possible next model can be made if we combine previous models and try to minimize the overall prediction error. This happens by optimizing the loss function. So the difference between random forest and gradient boost is that in random forest all the trees are independent from each other while here the next tree is made based upon the predecessor. It mainly consists of 3parts:

3.5.1 Weak Learner/Naïve Model

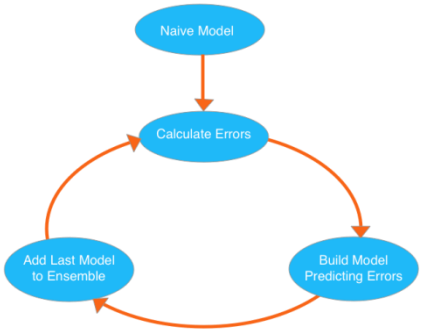
In gradient boosting, decision trees are used as the weak learner also called as Naïve Model. Trees are constructed in a greedy manner, choosing the best split points based on scores like Gini or to minimize the loss. It is collective to make the weak learners in specific ways, like a maximum number of layers, nodes, splits or leaf nodes. This is to make sure that the learners remain weak, but can still be constructed greedily.

3.5.2 Loss Function

The loss function used depends on the type of problem which is to be solved. It must be differentiable, but many standard loss functions are supported and we can define our own..

3.5.3 Additive Model

Trees are added one at a time, and existing trees within the model are not changed. A gradient descent procedure is employed to minimize the loss when adding trees. Traditionally, gradient descent is employed to minimize set of parameters, such as the coefficients during a regression of y on x equation or weights in a neural network. After calculating error or loss, weights are updated to minimise that error. After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the lost (i.e. follow the gradient).



Pseudo Code:

Algorithm *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$

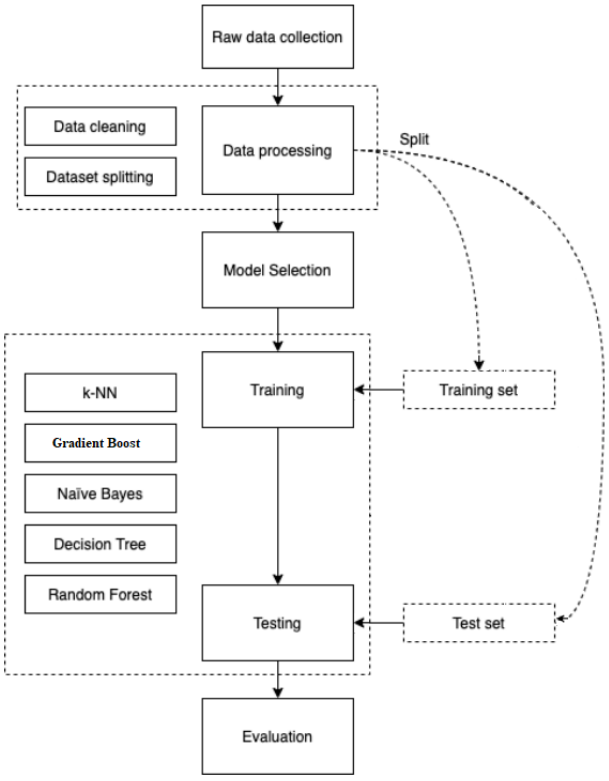
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

4. Experimentation

This section outlines the details of the design and implementation of the proposed solution. The solution is implemented in Python 3. Firstly, an overview of the solution is presented, briefly describing the phases of this implementation. Section 6.2 describes the data preparation process, including details on data cleaning and transformation, and dataset splitting. Section 6.3 presents the modeling process, with a detailed account of the training and testing processes. Section 6.4 concludes with an overview of the evaluation procedure, including a summary of the performance metrics used to analyze the intrusion detection performance of the DDoS datasets.

4.1 Overview

Figure presents a flow chart of the supervised learning process adopted in this study,



as part of the proposed solution.

Firstly, DDoS raw data is gathered from the link. After collection, data is processed to construct the final datasets for modeling. Data processing includes data cleaning, transformation of data types, and dataset splitting which is explained in section. This is followed by the model selection process.The models are trained using five different algorithms; k-nearest neighbor, Gradient Boost, Naïve Bayes, Decision tree, Random forest. Finally, the model is tested with unseen data. The results are evaluated using several performance metrics, as described in Section.

4.2 Data Preparation

4.2.1 Data Cleaning and Transformation

4.2.1.1 Missing data: Handling missing data is vital in machine learning, as it could lead to incorrect predictions for any model. Accordingly, null values are eliminated by propagating the last valid observation forward along the column axis.

```
In [18]: df.shape

Out[18]: (3894072, 87)

In [19]: df1 = df[df.isna().any(axis=1)]
         df1.shape

Out[19]: (27, 87)
```

As the number of records having missing data are very less. So, removing those records will not make any considerable difference.

```
In [20]: df=df.dropna()

In [21]: df.shape

Out[21]: (3894045, 87)
```

4.2.1.2 Infinite and large data: Handling infinite values is as important as null values as it hampers the working of algorithms. Also very large values also cause problems in the data.

```
In [27]: count = np.isinf(df).values.sum()
         count

Out[27]: 396926
```

```
In [28]: col_name = df.columns.to_series()[np.isinf(df).any()]
         print(col_name)

Flow Bytes/s        Flow Bytes/s
Flow Packets/s    Flow Packets/s
dtype: object
```

```
In [29]: print(df['Flow Bytes/s'].value_counts())

1.200000e+07    659587
4.580000e+08    261201
inf             198463
2.944000e+09    170937
8.020000e+08     69737
                 ...
1.148087e+00         1
1.219178e+00         1
1.136842e+07         1
1.526512e+00         1
1.257952e+04         1
Name: Flow Bytes/s, Length: 199034, dtype: int64
```

```
In [30]: print(df['Flow Packets/s'].value_counts())

2.000000e+06    1898111
1.000000e+06     225619
inf              198463
4.166667e+04     117175
4.081633e+04      95690
                  ...
2.979368e+01          1
3.507681e-01          1
1.525107e+01          1
1.201880e+00          1
2.034270e-01          1
Name: Flow Packets/s, Length: 175505, dtype: int64
```

As infinite values are very large so removing these can cause problems to the real model. Also it can be seen that values are very large. So can normalize it and also the infinite values be given finite value which will be larger than all the values so that its credibility doesn't go down will help.

```
In [31]: df['Flow Bytes/s']=df['Flow Bytes/s']/1000000 #for basic normalization we divided the data by 10^6
         mm = df.loc[df['Flow Bytes/s'] != np.inf, 'Flow Bytes/s'].max()
         mm=mm+200
         df['Flow Bytes/s'].replace(np.inf,mm,inplace=True)
         print(df['Flow Bytes/s'].value_counts())
```

```
1.200000e+01    659587
4.580000e+02    261201
3.144000e+03    198463
2.944000e+03    170937
8.020000e+02     69737
                 ...
1.734579e-06         1
2.757282e+00         1
1.292287e-06         1
1.845178e-06         1
9.321774e-07         1
Name: Flow Bytes/s, Length: 199032, dtype: int64
```

```
In [32]: df['Flow Packets/s']=df['Flow Packets/s']/1000000
         m = df.loc[df['Flow Packets/s'] != np.inf, 'Flow Packets/s'].max()
         m=m+200
         df['Flow Packets/s'].replace(np.inf,m,inplace=True)
         print(df['Flow Packets/s'].value_counts())
```

```
2.000000e+00    1898111
1.000000e+00     225619
2.040000e+02     198463
4.166667e-02     117175
4.081633e-02      95690
                   ...
5.673759e-03          1
3.340000e-07          1
1.917285e-07          1
2.165298e-07          1
2.526832e-07          1
Name: Flow Packets/s, Length: 175502, dtype: int64
```

4.2.1.3 Removing Unnecessary Features: The dataset has 87 columns out of which columns Timestamp, Source IP, Source Port, Destination IP, Destination Port, Flow ID are removed as these columns will affect the credibility of data because Timestamp and Flow ID are different for all records and Source IP, Destination IP, Source Port, Destination Port have just specific values as the dataset creators have used.

```
In [33]: df.drop(columns=['Flow ID'], inplace=True)
         df.drop(columns=['Timestamp'], inplace=True)
         df.drop(columns=['Source IP'], inplace=True)
         df.drop(columns=['Source Port'], inplace=True)
         df.drop(columns=['Destination IP'], inplace=True)
         df.drop(columns=['Destination Port'], inplace=True)
         df
```

4.2.1.4 Transformation: The format of the collected data might not be suitable for modeling. In such cases, data and data types need to be transformed so that the data can then be fed into the models. Accordingly, some data features were transformed into numeric or float, since models do not perform well with strings, or do not perform at all.

In [25]:

```
#SimillarHTTP is a object type so has to be converted to string
df['SimillarHTTP'] = df['SimillarHTTP'].astype('|S')

#Label Encoders for converting the string data  to integer values
encoder1=preprocessing.LabelEncoder()
df['SimillarHTTP']=encoder1.fit_transform(df['SimillarHTTP'])
df['Label']=encoder1.fit_transform(df['Label'])
```

4.2.1.5 Class Labels: Each dataset instance represents a snapshot of the network traffic at a given point in time. These instances are labeled according to the nature of the traffic i.e. whether the benign or some DDoS attack and in DDoS attack which type is it of (SYN Flood, UDP Flood, UDP Lag, MSSQL, NetBIOS, LDAP, and PortMap).So Classification is Multinomial.

```
print(df['Label'].value_counts())
```

```
Syn         1801189
MSSQL        694593
UDP          688602
NetBIOS      438261
LDAP         230013
Portmap       22480
BENIGN        14958
UDPLag         3976
Name: Label, dtype: int64
```

4.2.1.6 Splitting Datasets: A key characteristic of a good learning model is its ability to generalize to new or unseen, data. A model which is too close to a particular set of data is described as over fit, and therefore, will not perform well with unseen data. A generalized model requires exposure to multiple variations of input samples. Primarily, models require two sets of data, one to train and another to test. The training data is the set of instances that the model trains on, while the testing data is used to evaluate the generalizability of the model, that is, the performance of the model with unseen data.

```
#Dividing Data for training and testing
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=1) # 70% training and 30% test
print(X_train['Flow Packets/s'].value_counts())
#returning the training and testing data
return (X_train, X_test, Y_train, Y_test)
```

4.3 Training

During the training process, the selected algorithms are provided with training data to learn from to eventually create machine learning models. Accordingly, the training set is used. At this point in the process, the input data source needs to be provided and should contain the target attribute (class label). The training process involves finding patterns in the training set that map the input features with the target attribute. Based on the observed patterns, a model is produced which are as follows:

4.3.1 Decision Tree Classifier:

We built two decision tree classifiers with different criteria: One with Information Gain (Entropy) as criteria of splitting and other with gini index

4.3.1.1 With Entropy:

Function for creating and saving Decision Tree Classifier With Entropy Model

```
def decisiontree_entropy(X_train,Y_train):
    from sklearn import tree
    clf=tree.DecisionTreeClassifier(criterion='entropy')
    clf.fit(X_train,Y_train)
    save(clf, "C:/Users/Hp/PycharmProjects/pythonProject/decisiontree_entropy.mdl")
    return True
```

```
decisiontree_entropy(X_train,Y_train)
```

```
saved

True
```

Feature Importance:

Also we can see the importance of every feature i.e. how much the classifier depends upon given feature.

```
importance_decision_entropy = decision_entropy .feature_importances_
dictt=dict(zip(columns, importance_decision_entropy))
importance_decision_entropy={k: v for k, v in sorted(dictt.items(), key=lambda item: item[1],reverse=True)}
print ("{:<40} {:<10} ".format('Feature', 'Importance'))
# print each data item.
for key, value in importance_decision_entropy.items():
    print ("{:<40} {:<10} ".format(key, value))
```

```
Feature                                  Importance
Min Packet Length                        0.49640934178656027
Average Packet Size                      0.24683216474284056
Max Packet Length                        0.22378687853379026
ACK Flag Count                           0.011530837511773704
Init_Win_bytes_forward                   0.004629425168503786
Flow Bytes/s                             0.004545919323931296
min_seg_size_forward                     0.0017105906547327317
Total Length of Fwd Packets              0.0014545624668625171
Inbound                                  0.0014195048235718856
Fwd IAT Min                              0.0009258695594770828
Avg Fwd Segment Size                     0.0008873818205409081
Fwd Header Length.1                      0.0008793732760365883
Fwd Header Length                        0.0008312454151216615
Fwd Packet Length Max                    0.0006344423431374986
Packet Length Mean                       0.000629605856426146
Subflow Fwd Bytes                        0.0006280177142926165
Fwd Packet Length Min                    0.0005839164218679005
Flow IAT Mean                            0.00024861068435902894
Fwd Packet Length Mean                   0.00018802422036468425
Flow Duration                            0.0001632982433701273
SYN Flag Count                           0.0001256673339181236
Packet Length Variance                   0.000116328450648585
Flow Packets/s                           0.0001131596316423964
Flow IAT Min                             0.00010350086971082603
```

4.3.1.2 With Gini Index:

Function for creating and saving Decision Tree Classifier With GINI INDEX Model

```
def decisiontree_gini(X_train,Y_train):
    from sklearn import tree
    clf=tree.DecisionTreeClassifier()
    clf.fit(X_train,Y_train)
    save(clf, "C:/Users/Hp/PycharmProjects/pythonProject/decisiontree_gini.mdl")
    return True
```

```
decisiontree_gini(X_train,Y_train)
```

```
saved

True
```

Feature Importance:

```
importance_decision_gini = decision_gini .feature_importances_
dictt=dict(zip(columns, importance_decision_gini))
importance_decision_gini={k: v for k, v in sorted(dictt.items(), key=lambda item: item[1],reverse=True)}
print ("{:<40} {:<10} ".format('Feature', 'Importance'))
# print each data item.
for key, value in importance_decision_gini.items():
    print ("{:<40} {:<10} ".format(key, value))
```

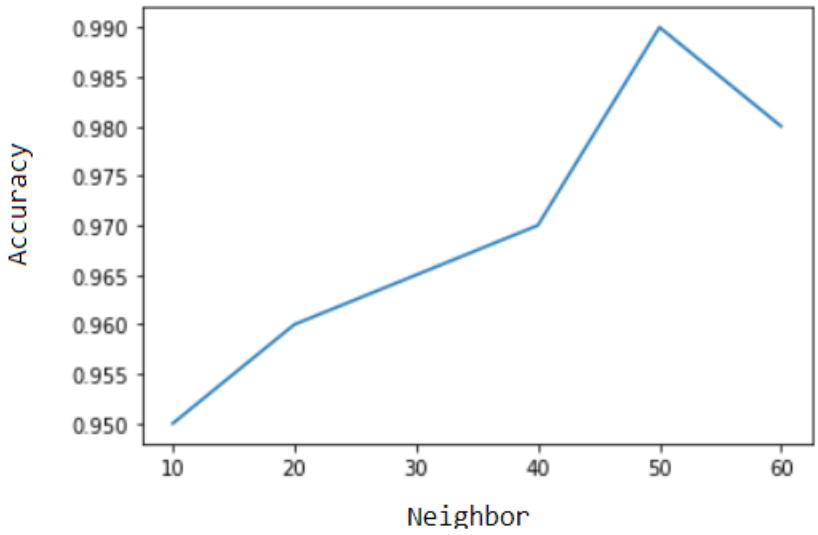
```
Feature                                  Importance
ACK Flag Count                           0.4601559073777657
Average Packet Size                      0.20715925332628973
Fwd Packet Length Max                    0.19211514839534263
Max Packet Length                        0.11650024082875801
Min Packet Length                        0.008901870212226257
Inbound                                  0.0031265227968344887
Fwd Packet Length Mean                   0.0022784878376733014
Flow Bytes/s                             0.0015899952629221258
min_seg_size_forward                     0.0014492606389429593
Total Length of Fwd Packets              0.0012729817128562264
Fwd Packet Length Min                    0.0009164165072383531
Fwd Header Length                        0.0007836354706048853
Packet Length Mean                       0.0005776801781685499
Fwd Header Length.1                      0.0004978169048885016
Subflow Fwd Bytes                        0.0004023908303609716
Avg Fwd Segment Size                     0.00034146230243139477
Fwd IAT Min                              0.0002862950529003008
Flow Packets/s                           0.00019118375038008531
Packet Length Std                        0.00018711527333152895
Init_Win_bytes_forward                   0.00018267462501565728
SYN Flag Count                           0.0001237970701384819
Flow IAT Max                             0.00011608703131644967
```

4.3.2 K Nearest Neighbor:

K nearest neighbor is tested for different values of Neighbor (with a very small part of dataset) whose accuracy is as follows:

Neighbor	Accuracy
10	0.95
20	0.96
30	0.965
40	0.97
50	0.99
60	0.98

This can be represented in form of graph as :



So we used Neighbor=50 in our model.

Function for creating and saving K Nearest Neighbor Classifier Model

```
def KNearestNeighbor(X_train,Y_train,neighbor,name):
    model = KNeighborsClassifier(n_neighbors=neighbor)
    model.fit(X_train,Y_train)
    save(model, name)
    return True
```

```
KNearestNeighbor(X_train,Y_train,50,"C:/Users/Hp/PycharmProjects/pythonProject/KNN50.mdl")
```

```
saved

True
```

4.3.3 Naïve Bayes Classifier:

There are three types of Naïve Bayes classifier present in Sklearn.

- Multinomial Naïve Bayes
- Bernoulli Naïve Bayes
- Gaussian Naïve Bayes

For this dataset Multinomial Naïve Bayes can't be used as it is used for discrete counts and also Bernoulli Naïve Bayes can't be applied as for these features should be binary data but in our dataset it is not true.

So we used Gaussian Naïve Bayes classifier.

Function for creating and saving Naive Bayes Model

```
def Naive_Bayes(X_train,Y_train):
    model = GaussianNB()
    model.fit(X_train,Y_train)
    save(model, "C:/Users/Hp/PycharmProjects/pythonProject/Naive_Bayes.mdl")
    return True
```

```
Naive_Bayes(X_train,Y_train)
```

```
saved

True
```

4.3.4 Gradient Boost Classifier:

For this paper we used XGBoost Classifier because it is fast and also gives more accuracy i.e. it is basically updated and most popular Gradient Boost Classifier.

Function for creating and saving Gradient Boost Classifier Model

```
def GradientBoost(X_train,Y_train,lr,name):
    model = xgb.XGBClassifier(learning_rate=lr)
    model.fit(X_train,Y_train)
    save(model, name)
    return True
```

```
GradientBoost(X_train,Y_train,0.5,"C:/Users/Hp/PycharmProjects/pythonProject/GradientBoost05.mdl")
```

```
saved

True
```

Feature Importance:

```
importance_gd = clf_gd.feature_importances_
dictt=dict(zip(columns, importance_gd))
importance_gd={k: v for k, v in sorted(dictt.items(), key=lambda item: item[1],reverse=True)}
print ("{:<40} {:<10} ".format('Feature', 'Importance'))
# print each data item.
for key, value in importance_gd.items():
    print ("{:<40} {:<10} ".format(key, value))
```

```
Feature                                  Importance
ACK Flag Count                           0.8490367531776428
Fwd Packet Length Std                    0.07568186521530151
Min Packet Length                        0.025686411187052727
Fwd Packet Length Max                    0.012024401687085629
Average Packet Size                      0.009461821056902409
Fwd Packet Length Min                    0.008905714377760887
Fwd Packet Length Mean                   0.007664439734071493
Max Packet Length                        0.007374416571110487
URG Flag Count                           0.0011450940510258079
Inbound                                  0.0009124670177698135
Init_Win_bytes_forward                   0.0005865833954885602
Total Fwd Packets                        0.00030265495297499
Total Length of Fwd Packets              0.00023466389393433928
Packet Length Mean                       0.00019598338985815644
min_seg_size_forward                     0.00010507513070479035
SYN Flag Count                           8.282488124677911e-05
```

4.3.5 Random Forest Classifier:

In this paper we as in Decision Tree Classifier have implemented two Random Forest Classifier with different splitting criteria one using entropy and other gini index.
We have used estimators=100 i.e. decision trees made are 100.

4.3.5.1 Using Entropy:

Function for creating and saving Random Forest Classifier With Entropy Model

```
def random_forest_entropy(X_train,Y_train):
    model = RandomForestClassifier(n_estimators=100, criterion="entropy");
    model.fit(X_train, Y_train)
    save(model, "C:/Users/Hp/PycharmProjects/pythonProject/random-forest-entropy.mdl")
    return True
```

```
random_forest_entropy(X_train,Y_train)
```

```
saved

True
```

Feature Importance:

```
importance_entropy = clf_entropy.feature_importances_
dictt=dict(zip(columns, importance_entropy))
importance_entropy={k: v for k, v in sorted(dictt.items(), key=lambda item: item[1],reverse=True)}
print ("{:<40} {:<10} ".format('Feature', 'Importance'))
# print each data item.
for key, value in importance_entropy.items():
    print ("{:<40} {:<10} ".format(key, value))
```

```
Feature                                  Importance
Min Packet Length                        0.1516582390898866
Avg Fwd Segment Size                     0.10319791081195219
Average Packet Size                      0.10243897039431166
Fwd Packet Length Min                    0.09237528551001967
Fwd Packet Length Max                    0.08174898895884546
Packet Length Mean                       0.07729604928884745
Fwd Packet Length Mean                   0.07418742326418588
Max Packet Length                        0.05945193694944073
Subflow Fwd Bytes                        0.051239177161365404
Total Length of Fwd Packets              0.04550593225797074
Flow Bytes/s                             0.03635540247212011
Protocol                                 0.020263168962096047
ACK Flag Count                           0.018727358730789064
Init_Win_bytes_forward                   0.015746603998996606
Packet Length Std                        0.0072441169618582355
Packet Length Variance                   0.006783049656778316
min_seg_size_forward                     0.005141247071090835
Fwd Header Length                        0.005064907387462579
Flow Duration                            0.004364465270563021
Fwd Packet Length Std                    0.004316034545065283
Flow IAT Std                             0.00304284067249798
Fwd IAT Max                              0.002607040638415359
act_data_pkt_fwd                         0.0025693904794002174
Fwd IAT Total                            0.002568781121420273
```

4.3.5.2 Using Gini Index:

Function for creating and saving Random Forest Classifier With GINI INDEX Model

```
def random_forest_gini(X_train,Y_train):
    model = RandomForestClassifier(n_estimators=100, criterion="gini")#max_depth=15
    model.fit(X_train, Y_train)
    save(model, "C:/Users/Hp/PycharmProjects/pythonProject/random-forest-gini.mdl")
    return True
```

```
random_forest_gini(X_train,Y_train)
```

```
saved

True
```

Feature Importance:

```
importance_gini = clf_gini.feature_importances_
dictt=dict(zip(columns, importance_gini))
importance_gini={k: v for k, v in sorted(dictt.items(), key=lambda item: item[1],reverse=True)}
print ("{:<40} {:<10} ".format('Feature', 'Importance'))
# print each data item.
for key, value in importance_gini.items():
    print ("{:<40} {:<10} ".format(key, value))
```

```
Feature                                  Importance
Max Packet Length                        0.11129622920770219
Packet Length Mean                       0.09949788110653787
Fwd Packet Length Mean                   0.08978768361272463
Fwd Packet Length Min                    0.082308915592799
Fwd Packet Length Max                    0.07948099233888806
Average Packet Size                      0.0714209534437103
Min Packet Length                        0.06650453244657727
ACK Flag Count                           0.06295935865501383
Avg Fwd Segment Size                     0.058923114953712864
Total Length of Fwd Packets              0.053642089216641695
Subflow Fwd Bytes                        0.050678226447501584
Init_Win_bytes_forward                   0.04385040786410594
Protocol                                 0.024315066262049524
Flow Bytes/s                             0.01783189049094836
Fwd Header Length.1                      0.009547008362278472
Fwd Packet Length Std                    0.007639142147601071
min_seg_size_forward                     0.007070725536660922
Packet Length Variance                   0.006548380174441645
```

4.4 Testing

In the last stage of experimentation the models are tested with unseen data. The unseen data used at this stage is the resulting test set from the data split (20%). Testing is conducted to assess how a model represents data and how well it will perform in the future. This study ensured that any tweaks to the models were done prior to testing, so that the testing data is used only once. Various performance metrics were generated to be able to analyse the performance of the DDoS datasets, such as accuracy, precision, recall, and F-measure and confusion metrics. These are described in the next section.

5 Results and Discussions:

A crucial part of understanding the performance of a model is generating performance metrics. In this study, various metrics are generated. These are described below.

5.1 Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known**.** It is basically a tabulated visualization of the performance of supervised learning algorithms. The rows represent the count of instances in an actual class, while the columns represent the count of instances in a predictive class.
The confusion Matrix for all the classifiers used are as follows:

Decision Tree With Entropy

	pred:0	pred:1	pred:2	pred:3	pred:4	pred:5	pred:6	pred:7
true:0	239	0	0	0	0	0	0	0
true:1	0	3750	13	1	0	0	0	0
true:2	0	140	10974	0	0	1	57	0
true:3	0	0	0	7040	2	0	1	0
true:4	0	0	2	354	1	1	0	0
true:5	0	0	0	0	0	29034	3	8
true:6	0	0	145	2	0	1	10975	7
true:7	0	0	0	0	0	1	8	43

Decision Tree With Gini

	pred:0	pred:1	pred:2	pred:3	pred:4	pred:5	pred:6	pred:7
true:0	238	0	0	0	0	1	0	0
true:1	0	3749	15	0	0	0	0	0
true:2	0	139	10974	1	0	1	57	0
true:3	0	0	0	7040	2	0	1	0
true:4	0	0	2	354	1	1	0	0
true:5	0	0	0	0	0	29034	3	8
true:6	0	0	143	2	0	0	10978	7
true:7	0	0	0	0	0	1	8	43

KNN

	pred:0	pred:1	pred:2	pred:3	pred:4	pred:5	pred:6	pred:7
true:0	220	0	0	0	0	1	0	18
true:1	0	3453	9	0	0	0	302	0
true:2	0	28	10219	1	0	909	15	0
true:3	0	0	0	6482	557	3	1	0
true:4	0	0	1	104	253	0	0	0
true:5	0	0	2274	0	0	26771	0	0
true:6	0	902	26	1	0	4	10195	2
true:7	6	0	0	0	0	0	0	46

Gradient Boost

	pred:0	pred:1	pred:2	pred:3	pred:4	pred:5	pred:6	pred:7
true:0	239	0	0	0	0	0	0	0
true:1	0	3750	13	1	0	0	0	0
true:2	0	138	10991	0	0	0	43	0
true:3	0	0	0	7041	1	0	1	0
true:4	0	0	1	354	0	2	1	0
true:5	0	0	0	0	0	29034	3	8
true:6	0	0	142	1	0	0	10978	9
true:7	0	0	0	0	0	1	8	43

Random Forest with Entropy

	pred:0	pred:1	pred:2	pred:3	pred:4	pred:5	pred:6	pred:7
true:0	239	0	0	0	0	0	0	0
true:1	0	3753	11	0	0	0	0	0
true:2	0	139	10979	0	0	0	54	0
true:3	0	0	0	7041	1	0	1	0
true:4	0	0	2	354	1	1	0	0
true:5	0	0	0	0	0	29034	3	8
true:6	0	0	141	2	0	0	10980	7
true:7	0	0	0	0	0	1	8	43

Random Forest with Gini

	pred:0	pred:1	pred:2	pred:3	pred:4	pred:5	pred:6	pred:7
true:0	239	0	0	0	0	0	0	0
true:1	0	3751	13	0	0	0	0	0
true:2	0	139	10977	0	0	0	56	0
true:3	0	0	0	7041	1	0	1	0
true:4	0	0	1	354	1	2	0	0
true:5	0	0	0	0	0	29033	3	9
true:6	0	0	143	1	0	0	10979	7
true:7	0	0	0	0	0	1	8	43

Naive Bayes

	pred:0	pred:1	pred:2	pred:3	pred:4	pred:5	pred:6	pred:7
true:0	39	29	7	1	147	7	3	6
true:1	0	3086	0	0	678	0	0	0
true:2	1	8478	3	1	2682	7	0	0
true:3	8	4839	0	0	2196	0	0	0
true:4	0	272	0	0	86	0	0	0
true:5	347	10767	17	120	14834	2926	13	21
true:6	0	3645	190	0	2008	0	5285	2
true:7	1	0	0	0	50	1	0	0

5.2 Accuracy: Accuracy is one metric for evaluating classification models. Accuracy is the fraction of predictions our model got right. i.e. accuracy has the following definition:

$$\text{Accuracy} = \frac{\textit{Correctly classified instances}}{\textit{Total instances}} \times 100\%$$

5.3 Precision: Although accuracy provides an indication on whether the model is being trained correctly, it does not give information on detailed information on the specific application. Consequently, other performance metrics are employed, such as precision. Precision is defined as the rate of correctly classified positives, or true positives.

$$\text{Precision} = \frac{\textit{True positives}}{\textit{True positives+False positives}}$$

5.4 Recall: Another performance metric is recall. Recall is a measure of how many of the actual positives were found or recalled. It is also a significantly important metric, as having undetected positives, or false negatives, might have serious consequences in some areas.

$$\text{Recall} = \frac{\textit{True positives}}{\textit{True positives+False negatives}}$$

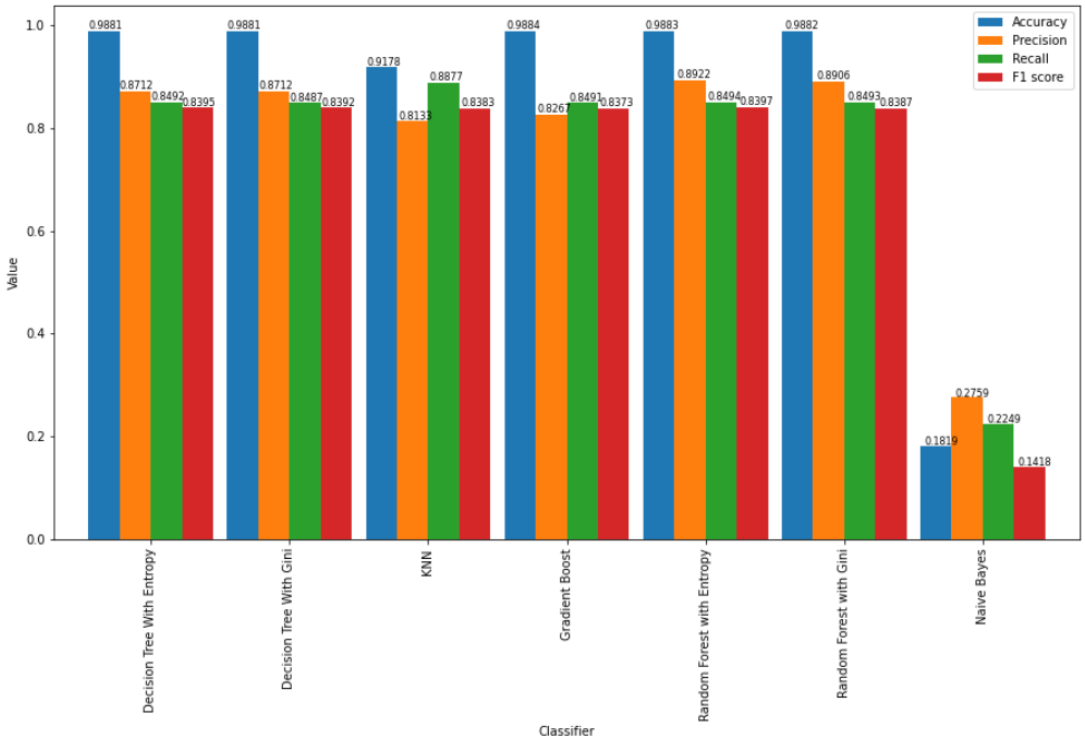
5.5 F-measure: The F-measure is a metric that provides an overall accuracy score for a model by combining precision and recall. A good F-measure score means that a model has both low false positives and low false negatives, and therefore, a model correctly identifying threats while having minimal false alarms.

$$\text{F-measure} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

The following Table presents the evaluation metrics of the machine learning models including accuracy, precision, recall, F-measure:

	Accuracy	Precision	Recall	F1 score
Decision Tree With Entropy	0.988106	0.871213	0.849193	0.839487
Decision Tree With Gini	0.988122	0.871243	0.848670	0.839242
KNN	0.917775	0.813344	0.887743	0.838256
Gradient Boost	0.988424	0.826734	0.849085	0.837309
Random Forest with Entropy	0.988329	0.892208	0.849422	0.839683
Random Forest with Gini	0.988233	0.890590	0.849318	0.838729
Naive Bayes	0.181918	0.275895	0.224891	0.141753

This can be represented in graphical way as:



The goal of this evaluation is to analyze the performance of different classifiers in terms of their capacity to detect and classify different DDoS attacks. For analyzing it should be kept in mind that the dataset is highly unbalanced. So instead of accuracy precision, recall and f1 score are more important.

From the results shown above, it shows that based upon accuracy Gradient Boost Performs best as its accuracy is 98.84% but in terms of precision, recall and F1 score Random Forest Classifier performs the best. On the other hand, the Naïve Bayes Classifier performs worst in all the parameters.

6 Conclusions and Future Work

6.1 Conclusions

Due to increased sophistication in the DDoS attack invoking methodologies and easy availability of related tools over the internet for, the detection and mitigation of the same has become very difficult. Machine Learning models are anomaly detection techniques, which are accurate and practical methods to identify DDoS traffic from legitimate traffic. These ML-based classifiers can be trained and tested using a real-life Intrusion Detection datasets, for better performance in real scenarios.

Keeping the above in mind, we contrast five such ML algorithms. The results obtained in the trials have revealed that our study successfully addressed the research question raised in this paper. From the above studies it can be concluded that Random Forest Classifier performs the best with accuracy of 99.83% and F1 score of 83.97%.

6.2 Future Work

In this paper we have used various machine learning approaches but interesting area that could be explored is how we can do the same classification using various hybrid algorithms and artificial neural networks and further be analyzed with deep learning, once the datasets could be represented in non-structured forms of data. The above can also be seen as two separate problems, which the future study could expand on.

Also the study can be done on other datasets and can see how these datasets perform in these scenarios.

7 APPENDIX

The complete code and dataset can be seen from the link:
https://github.com/rohit04445/Detection-and-classification-of-DDoS-attacks