A PROJECT REPORT

on

DDoS Attack Detection & Classification using Machine Learning - an Ensemble approach

Subject: Machine learning (ITE 2011)

in

B.Tech (IT)

by

Amartya Sharma (19BIT0021)

Piyush Sahu (19BIT0038)

Varun Kumar (19BIT0058)

6th semester, 2021

Under the Guidance of

Dr. DURAI RAJ VINCENT P.M

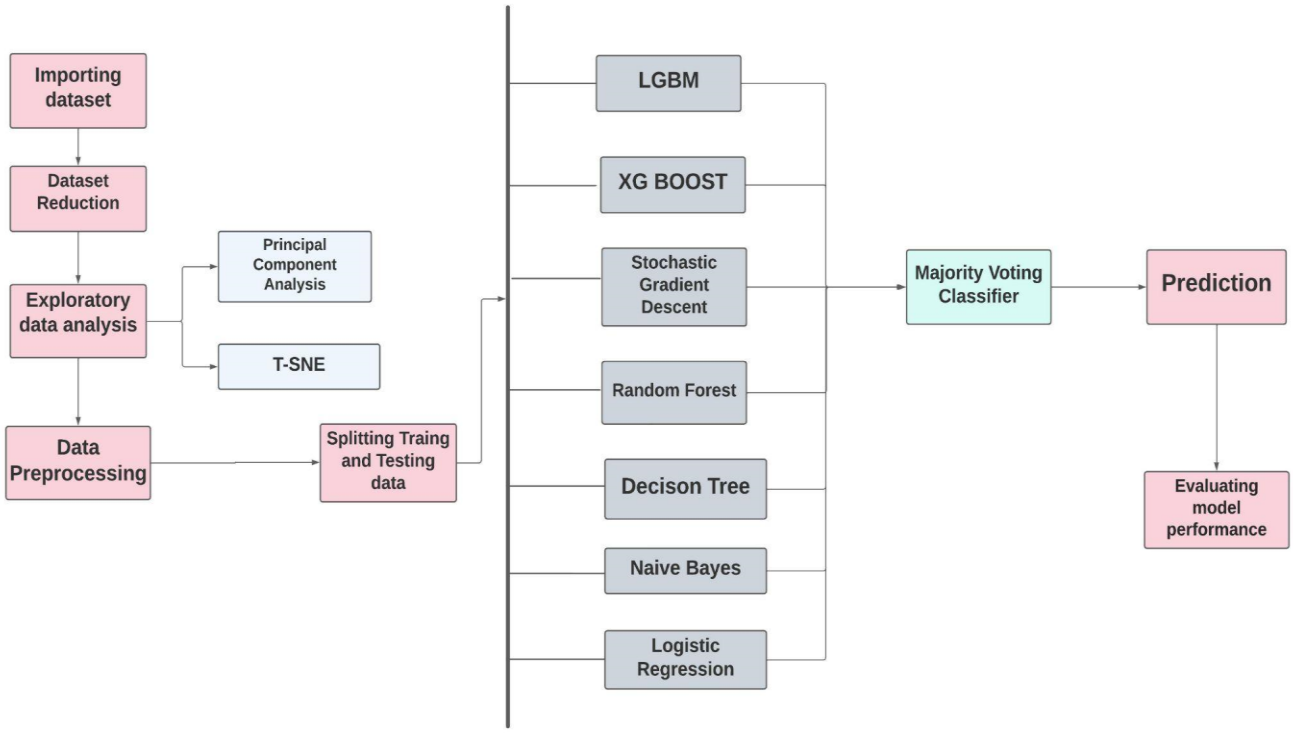
SITE

2.Abstract

DDoS (Distributed Denial of Service) is a form of cyber-threat that is one of several versions of DoS (Denial of Service) that employs IP addresses to attack a specific server or victim. DDoS attacks are expensive in terms of bandwidth and can also result in the loss of sensitive data. According to recent market data, it is one of the most common dangers facing the cybersecurity industry.As a result, it's become critical to develop better algorithms for identifying various types of DDoS Cyber Threats with greater accuracy while also taking into account the computational cost of detecting these threats. By converting the problem to a multilabel classification problem, we present an Ensemble Classifier that combines the performance of the top four performing algorithms in our implementation and compares it to different Artificial Intelligence and Machine Learning (AI and ML) algorithms to effectively detect different types of DDoS threats.

3.Description of the Project

Proposed Architecture



Importing dataset:

The dataset is the CIC-DDoS2019 dataset created by UNB which consists of 124K+ entries and over 37 columns. These columns have different features of packets from destination IP to flag count. Also the dataset has categorised packets belonging to different types of DDoS attacks, ranging from 0 to 7 each having a different meaning which is explained in table below

Label	Class of DDoS attack
0	Benign
1	NETBIOS
2	LDAP
3	MSSQL
4	Portmap
5	Syn
6	UDP
7	UDPLag

Dataset reduction

The dataset is too large thus we reduce dataset size by converting int data type to unsigned int of 8,16,32 or 64 bit or to signed int of. Also the float64 is reduced to float32 and infinity is replaced by -1. After doing all this we were able to reduce the dataset size from 361MB to 140MB i.e. 61% reduction of dataset size.

Exploratory data analysis

For exploratory data analysis we created a pie chart that shows what percentage of packet belongs to which attack category, we also have a bar graph that tells what protocols the packet used for DDoS attack, whether the packets were inbound or outbound packet. Also we used PCA, T-SNE and agglomerative clustering for visualisation, since data has 37 dimensions thus by using this we clustered the data and visualised it into 2 dimension space. Also we created a heat map to find correlation between the columns.

Data pre-processing

In data pre-processing we normalised the dataset and remove some unwanted columns like 'Unnamed:0.1' etc. and then the dataset is split into training and testing.

Model creation

We created 7 models consisting of LGBM, XGBoost, Stochastic gradient descent, random forest, decision tree, naive bayes and logistic regression. These models were trained and tested to determine their accuracy.

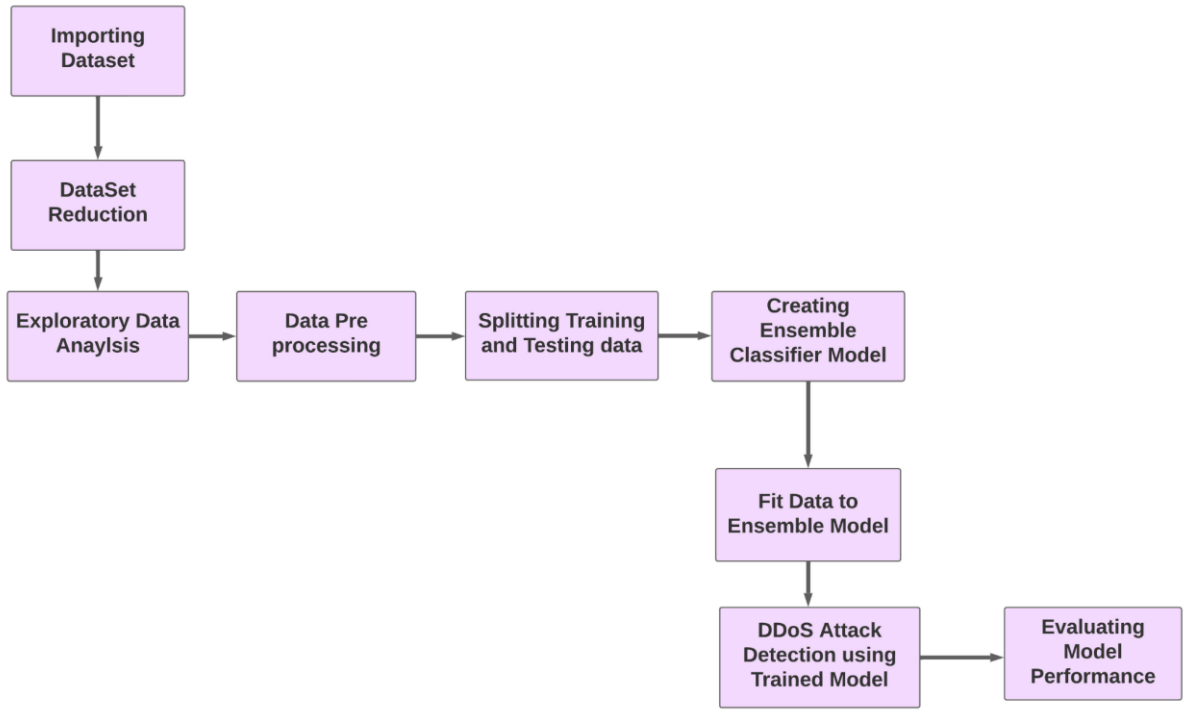
Creatine ensemble classifier model

After training all the models they are fitted into ensemble classifiers of two types, hard voting classifier and soft voting classifier where voting is done among all the models to classify packets. This model is trained and then tested and the result is observed.

Evaluation of model

We create confusion matrix and classification report for each model which had presion, recall and F1-score for each DDoS type along with accuracy, weighted average and macro average. We found that LGBM had the highest accuracy among all along with lowest training time and low probability of overfitting data.

Workflow of project



4.Literature Review

Piyush Sahu - 19BIT0038

[1]Aamir, M., & Zaidi, S. M. A. (2021). Clustering based semi-supervised machine learning for DDoS attack classification. *Journal of King Saud University-Computer and Information Sciences*, *33*(4), 436-446.

In this paper the author aimed to distinguish data in network flow from normal data to DDoS data using various semi supervised clustering based machine learning techniques.In many papers the IP address was not used to predict DDoS attack in random forest supervised training which was overcome in this literature or if some have used then it is limited to ET algorithms which this paper aimed to solve. First unlabeled data which consist of three features namely processing delay, CPU rate and traffic rate is given. This data is clustered using agglomerative clustering and PCA+k-means clustering to provide a label to these data using a voting method. If both clustering methods give opposite labels then it is clustered as 'suspicious' .Then these labelled data are fed into 3 models namely SVM, K-nearest neighbours and random forest with bagging to train them . The experiment was run twice, once without hyper parameter tuning and another with hyperparameter tuning which increased the accuracy and precision. The accuracy of SVM, KNN and RF without parameter tuning were 88.66%, 91.66%, 96.00%. The k value for kNN was changed to 11, for SVM C = 100.0, gamma = 1.0 were changed and for RF n_est=100 was made which increased accuracy by 92%, 96% and 97% respectively..

[2] Sahoo, K. S., Tripathy, B. K., Naik, K., Ramasubbareddy, S., Balusamy, B., Khari, M., & Burgos, D. (2020). An evolutionary SVM model for DDOS attack detection in software defined networks. *IEEE Access*, *8*, 132502-132513.

In this paper the author aimed to detect DDoS attack and classify it on different types of DDoS attack like UDPflood, HTTP-flood etc. in software defined network using machine learning approaches

The dataset was obtained by monitoring open flow switches for time intervals and feature extraction was done using kernel principal component analysis and since the dataset consisted of a lot of parameters thus to select optimum and correct parameters a genetic algorithm was used. These parameters were used for modelling by SVM+PCA algorithm and fitness values were checked. If optimum is found then optimum model found else using roulette wheel crossover and mutation is done.

It was found that using this method of KPCA+SVM+GA, the accuracy was 98.907% as compared to other classifiers like single SVM, KNN or PCA+GA+SVM and had lower testing time as well since it used reduced components. Also it had high recall values and precision for different types of ddos attack except smurf.

[3]Almiani, M., AbuGhazaleh, A., Jararweh, Y., & Razaque, A. (2021). DDoS detection in 5G-enabled IoT networks using deep Kalman backpropagation neural network. *International Journal of Machine Learning and Cybernetics*, *12*(11), 3337-3349.

In this paper the author aimed to present a way to detect DDoS attacks in a 5G enabled IoT network using deep learning methods like deel Kalman backpropagation neural network.
First the dataset CICDDoS 2019 dataset was taken and dataset encoding was done and removal of string values was done with other data preprocessing steps.Now kalman backpropagation network is trained , where first the Kalman filter is gain for is calculated for each layer and then during backpropagation weight correction is done.Also the error signals used to update the weights of last layer are composed of actual summation output and desired summation output. The hyperparameters for maximum accuracy were symmetrical sigmoid slope=0.2, backpropagation step size=0.2, forgetting factor=0.99, and the Layer Structure was [80 40 10 1].
The model finally had accuracy of 94%, 0.0952 false alarm rate and 97.49% detection rate , 91.22% precision.

[4]Banitalebi Dehkordi, A., Soltanaghaei, M., & Boroujeni, F. Z. (2021). The DDoS attacks detection through machine learning and statistical methods in SDN. *The Journal of Supercomputing*, *77*(3), 2383-2415.

In this paper the author aimed to detect hgh volume and low volume DDoS attack in SDN using three collectors, entropy based and classification sections.
First the data is collected in the collector section which has all the statistics like flow time, bytes sent etc. and if time is equal to period time then that data is passed to the entropy section. Here the entropy, threshold static and threshold dynamic is calculated, if the entropy is greater than threshold then the data is sent to collector again else for this input feature extraction is done and test is done on model. These models are BayesNet, J48, RandomTree, logistic regression, REPTree model thich help in classification. If DDoS is detected then an alarm is sounded.
This model was tested on 2 data sets , for data set UNB-ISCX, the accuracy was 99.85 % and FPR was 0.1%, this was better than Warusia Yassin et al, Zhiyuan Tan et al which used K-means and computer vision respectively. Also for dataset CTU-13 the accuracy was 99.12%, TPR was 98.60%. 99.64% precision and 99.11% F-measure.This was better than Ankit Bansal et al., Ruidong Chen et al and many more.

[5]Wei, Y., Jang-Jaccard, J., Sabrina, F., Singh, A., Xu, W., & Camtepe, S. (2021). Ae-mlp: A hybrid deep learning approach for ddos detection and classification. *IEEE Access*, *9*, 146810-146821.

In this paper the author proposed a unique approach of DDoS detection using autoencoder and multi layer perceptron (AE-MLP) and also classified it on different types.
FIrst auto encoder is used for features extraction which consist on usnerprived feed forward network in a symmetrical pattern with hidden layers as bottlenecks. It has 72 encoded features

and one hidden layer of 32 neurons and 24 features in the last hidden layers; these 24 features are extracted from latent space . This helped change the original dataset of high dimensionality to nonlinear low dimensional space. These reduced features are given as input to MLP which consist of supervised learning mode. This consists of 5 layers with 3 hidden layers and "relu" was used as activation function in the hidden layer and "softmax" in the final layer.
The proposed method had accuracy of 98.34%, precision of 97.91%, recall 98.48% , F1 score 98.18%. This was better than Can et.al.Sadaf.et.al and many more. The author found that shallow machine learning models are usually less accurate in comparison to deep models like that proposed in the paper.

Varun Kumar 19BIT0058

[1] Vinícius de Miranda Rios, Pedro R.M. Inácio, Damien Magoni, Mário M. Freire, Detection of reduction-of-quality DDoS attacks using Fuzzy Logic and machine learning algorithms, Computer Networks, Volume 186, 2021, 107792, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2020.107792.

In this paper, the authors have explored various machine learning techniques for the detection of a special kind of DDoS attack called **low-rate DoS attack** in which the aim is to mimic the normal traffic by keeping a low level of network traffic during the attack using the RoQ(Reduction of Quality) technique. The authors have also proposed a novel approach based on **Fuzzy Logic, MLP and Euclidean Distance** and compared this approach with baseline machine learning algorithms like **K-NN, SVM, Multinomial Naive Bayes (MNB)**. All these algorithms have been used separately to classify the incoming traffic as Legitimate or Attack based on three features from the dataset namely number of packets, entropy and average inter-arrival time. Four traffic datasets are used for building the models of which 2 have been obtained from real environments & 2 from emulated environments.
The models have been trained on both real and emulated environments and the performance has been evaluated. The metric used for evaluating the performance of the classifiers are **precision, recall, F1-score** and confusion matrix tables. The results show that the best classification results are given by MLP which lead to an F1-score of **98.04%** for attack traffic and **99.30%** for legitimate traffic. The novel proposed approach using FL, MLP, ED had an F1-score of **99.87%** for attack traffic and **99.95%** for legitimate traffic. The execution time for MLP was **0.74 ms** and **0.87ms** for emulated and real datasets respectively while the execution time for the proposed approach was 11 minutes 46 seconds and 46 minutes and 48 seconds for real and emulated environments respectively.
So, among the 4 ML algorithms, MLP leads to the best classification results. Also, the proposed approach based on FL, MLP & ED outperformed MLP but at the cost of high execution time.

[2] Nisha Ahuja, Gaurav Singal, Debajyoti Mukhopadhyay, Neeraj Kumar, Automated DDOS attack detection in software defined networking, Journal of Network and Computer Applications, Volume 187, 2021, 103108, ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2021.103108.

In this paper, the authors have focussed on the detection of DDoS attacks in Software-Defined Networks (SDN) which is a relatively new networking paradigm in which the network devices are programmable.SDNs are vulnerable to DDoS attack and to overcome this vulnerability, the authors have proposed a novel hybrid machine learning algorithm(SVC-RF) using Support vector classifier and Random Forest which classifies benign traffic from DDoS attack traffic. Since there was no SDN dataset available in the public domain, the authors have also created a dataset for the traffic flow in the SDN network. To create the dataset, the flow statistics are collected from the switches present in the topology and the statistics are then written onto the CSV file.
The proposed hybrid model consists of Support Vector Classifier and Random Forest.
First SVC is used for classification of the DDoS traffic as benign or attack. These results from the SVC are filtered again by Random Forest to make the classification. The models have been trained on the SDN dataset and the performance is evaluated. The metric used for evaluating the performance of the classifiers are Accuracy, Sensitivity, Specificity, Precision, and F1-score.
Accuracy comparison of the proposed SVM-RF hybrid model is done against other baseline ML algorithms. SVC-RF achieved the highest accuracy of 98.8%. ANN achieved an accuracy of 98.2%, Logistic Regression model provides an accuracy of 83.69% and was unable to capture the correct relationships b/w the features. KNN provided an accuracy of 95.22%. The SVC-RF model produced high precision and recall values. Thus it was concluded that the propped SVC-RF classifier showed better results as compared to other baseline ML algorithms on the SDN dataset.

[3] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del-Rincón and D. Siracusa, "Lucid: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection," in IEEE Transactions on Network and Service Management, vol. 17, no. 2, pp. 876-889, June 2020, doi: 10.1109/TNSM.2020.2971776.

In this paper, the authors have focused their attention towards reducing the computational intensity of the algorithms used for the classification of DDoS attacks so as to use such algorithms in a resource constrained environment. For this purpose, the authors have proposed a practical, lightweight, deep learning based algorithm called LUCID which exploits the properties of Convolutional Neural Networks(CNN) for the detection of DDoS attacks and classifying the network traffic as either benign or malicious.
In the proposed model, the CNN encapsulates the learning of malicious activity from the network traffic to identify DDoS patterns irrespective of where the patterns appear in the input. This encapsulation and feature learning during the training of the model eradicates the need for excessive feature engineering, ranking and selection.Thus this model can be easily deployed in online resource constrained environments. Also, a novel method has been used for the pre-

processing of the network traffic which involved generating spatial data representation used as input to the CNN. Due to these improvements, irrespective of whether the DDoS event appears at the start or the end of the input, LUCID will produce the same representation in the output. The performance of the proposed LUCID model is compared with others. The metric used for evaluating the performance of the classifiers are accuracy, **precision, recall, F1-score**. The accuracy of LUCID turned out to be 99.67%. MLP and LSTM achieved an accuracy of 86.34% and 96.24% respectively. Thus it was concluded that the proposed LUCID model showed improvements over the existing state-of-the-art algorithms. Also an improvement of 40x was observed in the processing time, hence this model is suitable for deployment in resource constrained environments

[4] Najafimehr, M., Zarifzadeh, S. & Mostafavi, S. A hybrid machine learning approach for detecting unprecedented DDoS attacks. J Supercomput (2022). https://doi.org/10.1007/s11227-021-04253-x

In this paper, the authors have proposed a novel method for the detection of DDoS attacks which overcomes the drawbacks of the existing methods by combining both supervised and unsupervised algorithms.
In the proposed model, initially, a clustering algorithm separates anomalous traffic from normal data using several features based on the flow of network traffic. After that, using several statistical measures a classification algorithm is used to label the clusters. The overall process is divided into 3 phases. Phase 1 is unsupervised learning which involves multiple preprocessing procedures to prepare the dataset. Phase 2 involves analysing the clusters and Phase 3 involves supervised learning where the classification model is built.
The performance of the proposed hybrid model is compared with others. The metric used for evaluating the performance of the classifiers are accuracy, **precision, recall, F1-score**. The evaluation results show that the proposed method is 198% more effective in detecting unprecedented types of DDoS attack when compared with the best conventional Machine Learning method i.e. Random Forest, which has been used in literature. The overall average improvement for the recall of the proposed method is 36.3% when compared to the existing methods.

[5] Matheus P. Novaes, Luiz F. Carvalho, Jaime Lloret, Mario Lemes Proença, Adversarial Deep Learning approach detection and defence against DDoS attacks in SDN environments, Future Generation Computer Systems, Volume 125, 2021, Pages 156-167, ISSN 0167-739X, https://doi.org/10.1016/j.future.2021.06.047.

In this paper, the authors have proposed a defence and detection system based on Adversarial training in Software Defined Networks (SDN) which makes use of Generative Adversarial Network(GAN) framework for the detection of DDoS attacks and also uses adversarial training to make the system less vulnerable to adversarial attacks. The proposed system consists of comprehensively-defined modules that allows monitoring the network traffic continuously using IP flow analysis enabling real-time anomaly detection.

To implement the proposed system, initially data collection is done wherein network flow data from the switches in the network. Next data preprocessing is done on the collected data. The next module is the adversarial deep learning anomaly detection module which analyzes the network's behaviour and the occurrence of DDoS attacks. Finally we have the mitigation module which is responsible to take the countermeasures to minimise the damages in case of an attack.
The performance of the proposed hybrid model is compared with others. The metric used for evaluating the performance of the classifiers are accuracy, **precision, recall, F1-score.**
The results show that GAN and CNN reached similar results with the accuracy of 94% whereas the methods like LSTM and MLP achieved accuracy of 90.29% and 92.12% respectively.
So, the results show that GAN achieved superior results due to its improved training process which involved the use of an adversary training approach making it less sensitive to various attacks.

Amartya Sharma 19BIT0021

[1]Ismail, & Mohmand, Muhammad & Hussain, Hameed & Ayaz, Ali & Ullah, Ubaid & Zakarya, Muhammad & Ahmed, Aftab & Raza, Mushtaq & Rahman, Izaz & Haleem, Muhammad. (2022). A Machine Learning based Classification and Prediction Technique for DDoS Attacks. IEEE Access. 10.1109/ACCESS.2022.3152577.

In this paper the authors focused on classifying various DDoS attacks with the help of machine learning algorithms . One of the widely used KDD dataset was used for performing this experimental study and it was carried out using classification algorithms as Random Forest and XGBoost methods . Also another dataset, the UNSW-nb15 dataset was used which has been provided by the Australian Centre of Cyber Security . The authors performed all the necessary requirements such as feature selection and normalisation of data before putting the data into the model and confusion matrix was used to showcase the result between the true positive and true negative values achieved. The result achieved was as such for the first classification which used random forest as the base model was 89% Precision , 89 % Recall with an average accuracy of 89 % , whereas for the second classification where XGBoost was taken as the base model the Precision and Recall was close 90% with an average accuracy of 90%. These results were promising when compared to the already existing models as only 79% to 85% accuracy was achieved and thus the proposed model outperformed the existing models.

[2]Cil, A. E., Yildiz, K., & Buldu, A. (2021). Detection of DDoS attacks with feed forward based deep neural network model. Expert Systems with Applications, 169, 114520. doi:10.1016/j.eswa.2020.114520

With the increase amount of cyber attacks taking place in particular DDoS attacks have been most common among them and thus it becomes necessary to early detect such attacks and even prevent the system before any severe damage is done.TIn this paper the authors proposed a Deep Neural Network approach to detect DDoS attacks which rely on real time network traffic

for packet analysis . The CICDDoS2019 dataset was used by the authors for carrying out the experiments. This dataset was divided into two parts . The Deep Neural Network consisted of 3 layers and was applied to first dataset to get an accuracy of 99.99% along with good precision value.On a comparative analysis between the accuracy attained by the two datasets , it was observed that higher accuracy was achieved for first dataset as it focuses on binary classification with less network traffic , whereas for dataset 2 an accuracy of 94.57% was achieved as it focuses on multiclass classification between different types of attacks and with higher network traffic.

[3] SaiSindhuTheja, R., & Shyam, G. K. (2021). An efficient metaheuristic algorithm based feature selection and recurrent neural network for DoS attack detection in cloud computing environment. Applied Soft Computing, 100, 106997. doi:10.1016/j.asoc.2020.106997

In this paper the author focuses on DDoS attack detection and prevention on cloud computing environments.KDD CUP 99 a benchmarking dataset was chosen for this study and algorithms such as Oppositional Crow Search Algorithm (OSCA) , a heuristic algorithm helpful in feature selection and Recurrent Neural Network (RNN) which used in classification process.This novel OSCA algorithm is proven more efficient in terms of feature selection as compared to other existing algorithms such as GA , FA and CSA as it is adopted to select the optimal feature subset, which improves the accuracy and minimises the feature subset length.
The results achieved were 98.18% Precision, 95.13% Recall , 93.56% as F-measure and 94.12% as Accuracy . The proposed model is also compared with other existing models which are based on conventional machine learning algorithms such as Artificial Neural Network , K-Nearest Classifier and Naive Bayes Classifier.The proposed model outperformed these existing models with a higher accuracy of over 3%.

[4]Tang, D., Zhang, S., Chen, J., & Wang, X. (2021). The detection of low-rate DoS attacks using the SADBSCAN algorithm. Information Sciences, 565, 229–247. doi:10.1016/j.ins.2021.02.038

In this paper the authors have focussed on distinguishing low rate **DoS attacks** from normal traffic and also reduce the number of false negatives. LDoS is a variant of DoS attacks which is capable of exploiting vulnerabilities in internet protocols to degrade the service quality. The authors have proposed a self adaptive density based spatial clustering algorithm namely SADBSCAN algorithm . It automatically identifies clusters in multi density datasets.For this work NS-2 and TestBud are used to generate data to perform the experiments .The algorithm was proposed as it solves one of the common problems in clustering multi density datasets as by showing high false negatives. The problem with LDoS is that the attacks are steady in nature and have a low rate . Cosine similarity is used as a metric to distinguish whether the data units of the cluster contain low rate DoS attacks or not. The proposed algorithm takes the idea of nearest neighbours to optimise the clustering thresholds . Thus it was effectively shown through this experiment that the proposed algorithm is highly efficient to classify between such

malicious attacks with high accuracy , Also it can be applied to large datasets for which it is difficult to form cluster groups.

[5]Swami, R., Dave, M., & Ranga, V. (2021). Detection and Analysis of TCP-SYN DDoS Attack in Software-Defined Networking. Wireless Personal Communications, 118(4), 2295–2317. doi:10.1007/s11277-021-08127-6

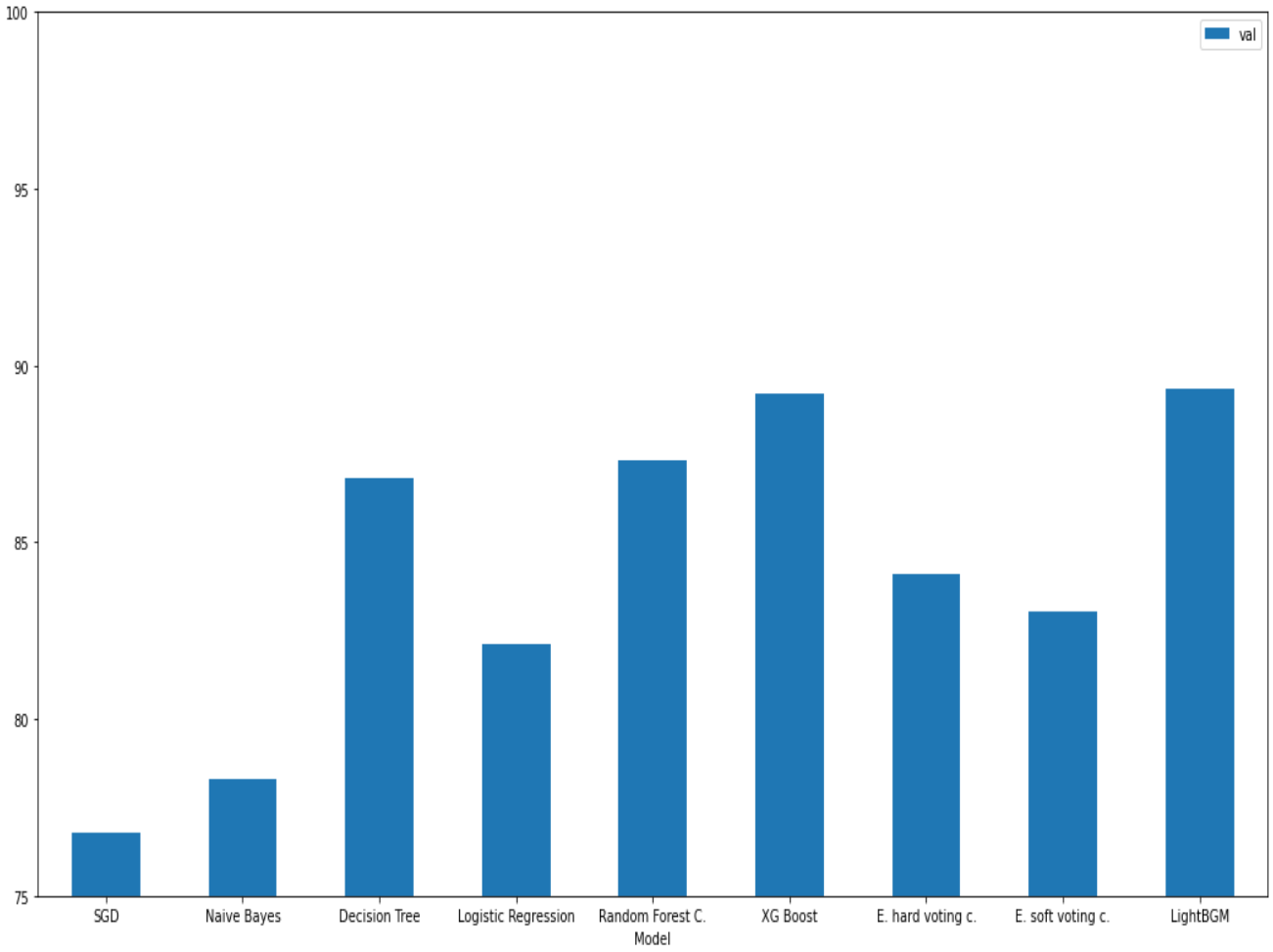
DDoS attack is one of the most powerful cyber attacks which causes services to be unavailable for normal users. In this paper the authors focused on a spoofed and non spoofed TCP - SYN flooding attack on SDN system.Also an IDS system is presented with their ml model for detection of such attacks. Cross-validation technique is used to validate the classification models.In this work the authors created a dataset by collecting traffic at the target host to perform the experiments on the proposed IDS. Feature extraction is performed on the dataset and then the data is fed into the ml model .5 machine learning model have been used namely Random Forest, Decision Tree, AdaBoost , Logistic Regression and MultiLayer Perceptron.The results from this study was 99.99% accuracy , 99.98% recall , 99.99% F measure. In comparative analysis between the ml classifiers we got to know that random forest , decision tree and adaboost had better result as compared to multilayer perceptron and logistic regression algorithms.As the accuracy , precision and f measure for MLP and LR are 99.98%,99.95% and 99.97% respectively.

5.Outcome of the Project

The accuracy of different models were as follows:

Classifier used	Accuracy
Stochastic gradient descent	77.08
Naive Bayes	78.32
Decision tree	86.81
Logistic regression	82.12
XG Boost	89.24
Ensemble hard voting classifier	84.08
Ensemble soft voting classifier	83.03
LightBGM	89.28
Random forest	87.33

Accuracy of different model bar graph



LGBM had highest accuracy with lowest training time, low memory usage, less chance of overfitting and compatible to large datasets with support for parallel learning.
This all is possible because LGBM internally used decision tree where it splits the tree leaf wise with best fit whereas other boosting algorithm usually split it along depth wise or level wise.So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms.

6.Complete Code and Complete Analysis

Google drive mounted

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Importing libraries

```
 #%tensorflow_version 2.x
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Lambda, Flatten
from keras.layers import Convolution1D, Dense, Dropout, MaxPooling1D, LSTM
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import Normalizer
from keras import callbacks
from keras.callbacks import CSVLogger
from tensorflow.keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
import numpy as np
np.random.seed(0)
```

Data Pre-processing

Reading the csv file, separating label column and storing it in another dataframe

```
import pandas as pd
df = pd.read_csv("drive/MyDrive/ML_proj/columns38_final.csv")
#df.info()
df=df.drop('Unnamed: 0.1',axis=1)
x=df.iloc[:,df.columns != 'Label']
y=df.iloc[:,-1]
# print("x\n",x.info())
y = pd.DataFrame(y)
# print('y\n',y.info())
```

Dataframe info

```
print("x\n",x.info())
y = pd.DataFrame(y)
print('y\n',y.info())
```

```
RangeIndex: 1245798 entries, 0 to 1245797
Data columns (total 37 columns):
 #   Column                      Non-Null Count    Dtype
---  ------                      --------------    -----
 0   Unnamed: 0                  1245798 non-null  int64
 1    Source Port                1245798 non-null  int64
 2    Destination IP             1245798 non-null  int64
 3    Destination Port           1245798 non-null  int64
 4    Protocol                   1245798 non-null  int64
 5   Total Length of Fwd Packets 1245798 non-null  float64
 6    Fwd Packet Length Max      1245798 non-null  float64
 7    Fwd Packet Length Min      1245798 non-null  float64
 8    Fwd Packet Length Mean     1245798 non-null  float64
 9   Bwd Packet Length Max       1245798 non-null  float64
 10   Bwd Packet Length Min      1245798 non-null  float64
 11   Bwd Packet Length Mean     1245798 non-null  float64
 12   Bwd Packet Length Std      1245798 non-null  float64
 13  Flow Bytes/s                1245753 non-null  float64
 14   Flow Packets/s             1245798 non-null  float64
 15   Flow IAT Mean              1245798 non-null  float64
 16   Flow IAT Std               1245798 non-null  float64
 17   Fwd IAT Mean               1245798 non-null  float64
 18   Fwd IAT Std                1245798 non-null  float64
 19  Fwd PSH Flags               1245798 non-null  int64
 20  Fwd Packets/s               1245798 non-null  float64
 21   Min Packet Length          1245798 non-null  float64
 22   Max Packet Length          1245798 non-null  float64
 23   Packet Length Mean         1245798 non-null  float64
 24   Packet Length Variance     1245798 non-null  float64
 25   RST Flag Count             1245798 non-null  int64
 26   ACK Flag Count             1245798 non-null  int64
 27   URG Flag Count             1245798 non-null  int64
 28   CWE Flag Count             1245798 non-null  int64
 29   Average Packet Size        1245798 non-null  float64
 30   Avg Fwd Segment Size       1245798 non-null  float64
 31   Avg Bwd Segment Size       1245798 non-null  float64
 32   Subflow Fwd Bytes          1245798 non-null  int64
 33  Init_Win_bytes_forward      1245798 non-null  int64
 34   Init_Win_bytes_backward    1245798 non-null  int64
 35   Idle Std                   1245798 non-null  float64
 36   Inbound                    1245798 non-null  int64
dtypes: float64(23), int64(14)
```

```
memory usage: 351.7 MB
x
 None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1245798 entries, 0 to 1245797
Data columns (total 1 columns):
 #   Column  Non-Null Count    Dtype
---  ------  --------------    -----
 0   Label   1245798 non-null  int64
dtypes: int64(1)
memory usage: 9.5 MB
y
 None
```

Function that replaces infinite value with -1

```
def Pre_process_data(df,col):
    '''
    Input: Data-frame and Column name.
    Operation: Fills the nan values with the minimum value in their respective column.
    Output: Returns the pre-processed data-frame.
    '''

    # removing infinite value with minus one
    #df['primary_use'] = df['primary_use'].astype("category").cat.codes
    print("Name of column with NaN: "+str(col))
    print(df[col].value_counts(dropna=False, normalize=True).head())
    df[col].replace(np.inf, -1, inplace=True)

    return df
```

Function that reduces size of data by converting data int64 data to unsigned int or signed int of lower bit size and reducing float64 to float 32bit.

```
def reduce_mem_usage(df):
    '''
    Input - data-frame.
    Operation - Reduce memory usage of the data-frame.
    '''
    start_mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is :",start_mem_usg," MB")
    #NAlist = [] # Keeps track of columns that have missing values filled in.
    for col in df.columns:
        if df[col].dtype != object:  # Exclude strings
            # Print current column type
            print("******************************")
            print("Column: ",col)
            print("dtype before: ",df[col].dtype)
            # make variables for Int, max and min
            IsInt = False
            mx = df[col].max()
            mn = df[col].min()
            #print("min for this col: ",mn)
            #print("max for this col: ",mx)
            # Integer does not support NA, therefore, NA needs to be filled
            if not np.isfinite(df[col]).all():
                #NAlist.append(col  )
                df = Pre_process_data(df,col)

            # test if column can be converted to an integer
            asint = df[col].fillna(0).astype(np.int64)
            result = (df[col] - asint)
            result = result.sum()
            if result > -0.01 and result < 0.01:
                IsInt = True
```

```
        # Make Integer/unsigned Integer datatypes
        if IsInt:
            if mn >= 0:
                if mx < 255:
                    df[col] = df[col].astype(np.uint8)
                elif mx < 65535:
                    df[col] = df[col].astype(np.uint16)
                elif mx < 4294967295:
                    df[col] = df[col].astype(np.uint32)
                else:
                    df[col] = df[col].astype(np.uint64)
            else:
                if mn > np.iinfo(np.int8).min and mx < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif mn > np.iinfo(np.int16).min and mx < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif mn > np.iinfo(np.int32).min and mx < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif mn > np.iinfo(np.int64).min and mx < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
        # Make float datatypes 32 bit
        else:
            df[col] = df[col].astype(np.float32)

        # Print new column type
        print("dtype after: ",df[col].dtype)
        print("*******************************")
# Print final result
print("___MEMORY USAGE AFTER COMPLETION:___")
mem_usg = df.memory_usage().sum() / 1024**2
print("Memory usage is: ",mem_usg," MB")
print("This is ",100*mem_usg/start_mem_usg,"% of the initial size")
return df
```

Reduction of dataset from 361 MB to 140 MB i.e. 38.8% of initial size

```
Memory usage of properties dataframe is : 361.1781311035156  MB
*******************************
Column:  Unnamed: 0
dtype before:  int64
dtype after:  uint32
*******************************
*******************************
Column:  Source Port
dtype before:  int64
dtype after:  uint16
*******************************
*******************************
Column:  Destination IP
dtype before:  int64
dtype after:  uint64
*******************************
*******************************
Column:  Destination Port
dtype before:  int64
dtype after:  uint32
*******************************
```

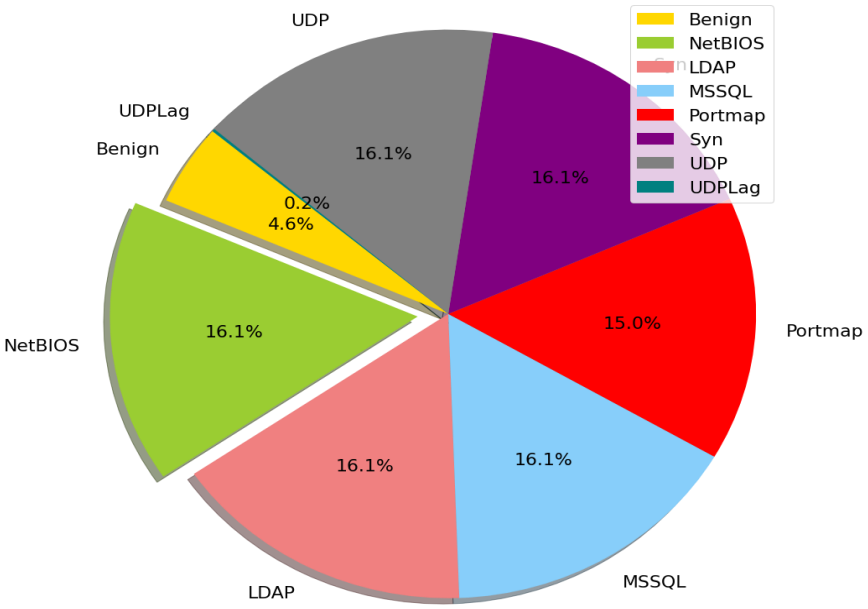
```
******************************
Column:    Inbound
dtype before:  int64
dtype after:  uint8
******************************
******************************
Column:  Label
dtype before:  int64
dtype after:  uint8
******************************
___MEMORY USAGE AFTER COMPLETION:___
Memory usage is:  140.19421768188477  MB
This is  38.81581015260981 % of the initial size
```

Exploratory data analysis

Pie chart of different types of DDoS attack packets

```
labels = ['Benign', 'NetBIOS', 'LDAP', 'MSSQL','Portmap','Syn','UDP','UDPLag']
sizes = np.array([df[df['Label']==0][' Protocol'].count(),df[df['Label']==1][' Protocol'].count(),
                  df[df['Label']==2][' Protocol'].count(),df[df['Label']==3][' Protocol'].count(),
                  df[df['Label']==4][' Protocol'].count(),df[df['Label']==5][' Protocol'].count(),
                  df[df['Label']==6][' Protocol'].count(),df[df['Label']==7][' Protocol'].count()])
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue','red','purple','grey','teal']
explode = (0, 0.1, 0, 0,0,0,0,0)  # explode 1st slice

# Plot
plt.rcParams.update({'font.size': 22})
plt.figure(figsize=(15,15))
plt.pie(sizes, labels=labels,explode=explode, colors=colors,autopct='%1.1f%%', shadow=True, startangle=140)
plt.legend()
plt.axis('equal')
plt.show()
```

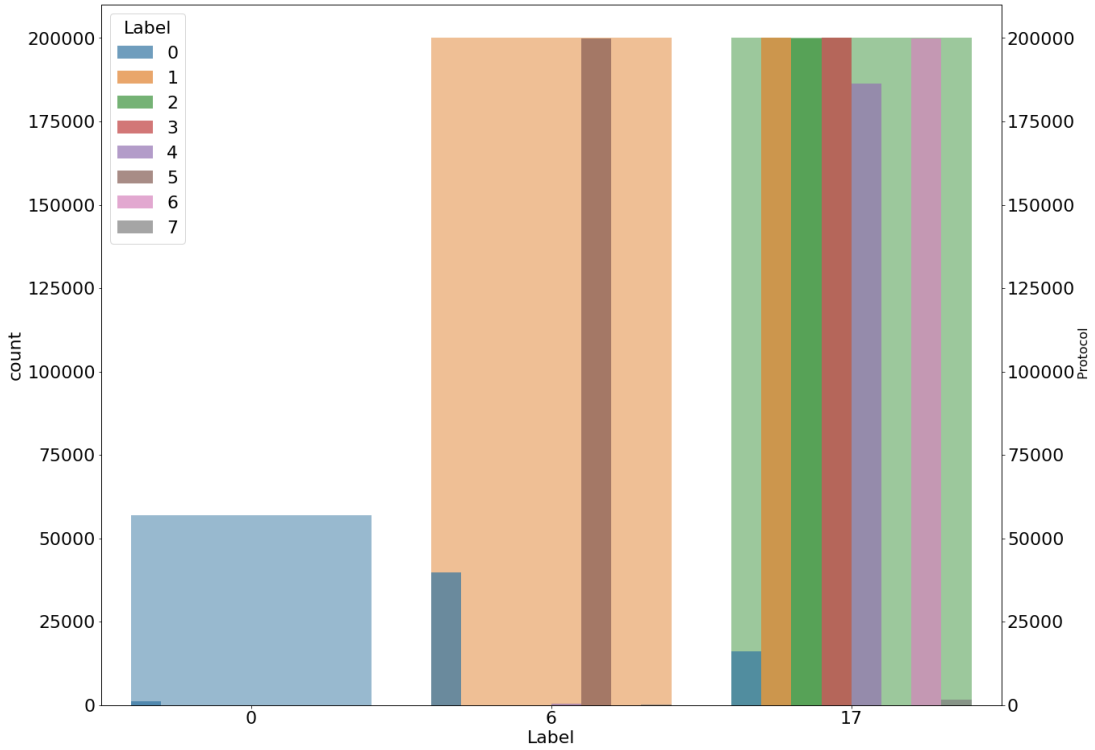


Analysis

We observe that DDoS attack types are almost equally distributed with each having around 16% except UDPLag and benign packets.

Bar graph to group packets according to protocols

```
plt.figure(figsize=(20,16))
g1 = sns.countplot(x='Label', data=df,alpha=0.5)
gt = g1.twinx()
gt = sns.countplot(x=' Protocol', hue='Label',alpha=0.7, data=df)
gt.set_ylabel(" Protocol", fontsize=16)
```

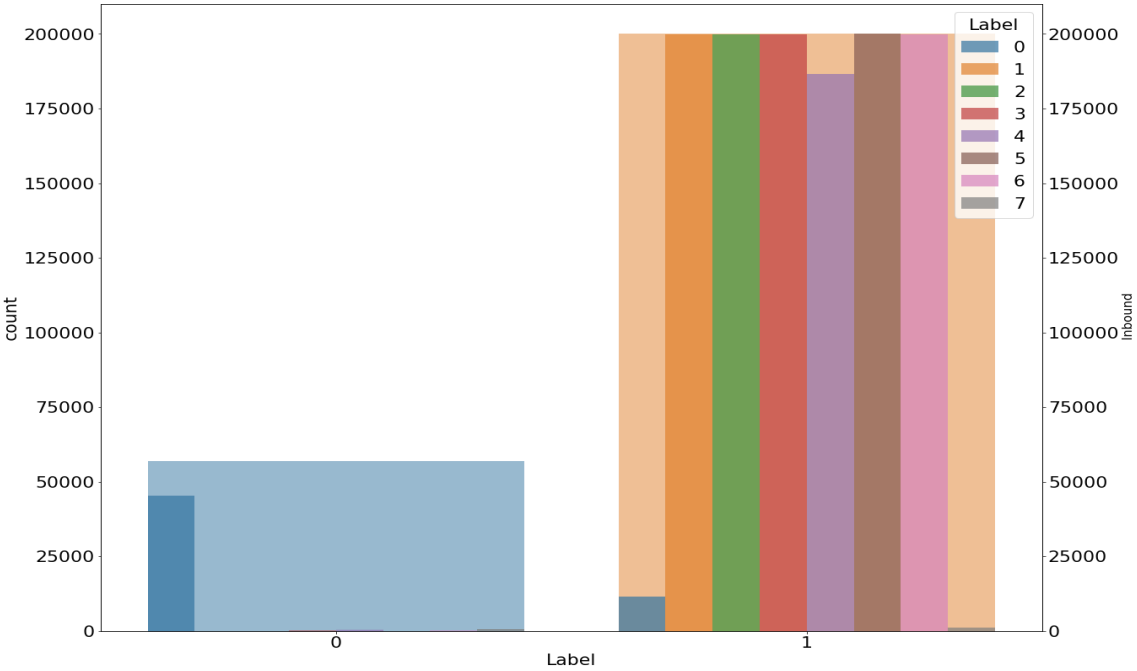


Analysis

We observe that most of the packets belong to protocol 17, only packet type 5 (Syn) belong to protocol 6 and packet type 0(benign) belongs to all the three protocol types.

Bar graph to categories packets as inbound or outbound

```
plt.figure(figsize=(20,16))
g1 = sns.countplot(x='Label', data=df,alpha=0.5)
gt = g1.twinx()
gt = sns.countplot(x=' Inbound', hue='Label',alpha=0.7, data=df)
gt.set_ylabel(' Inbound', fontsize=16)
```



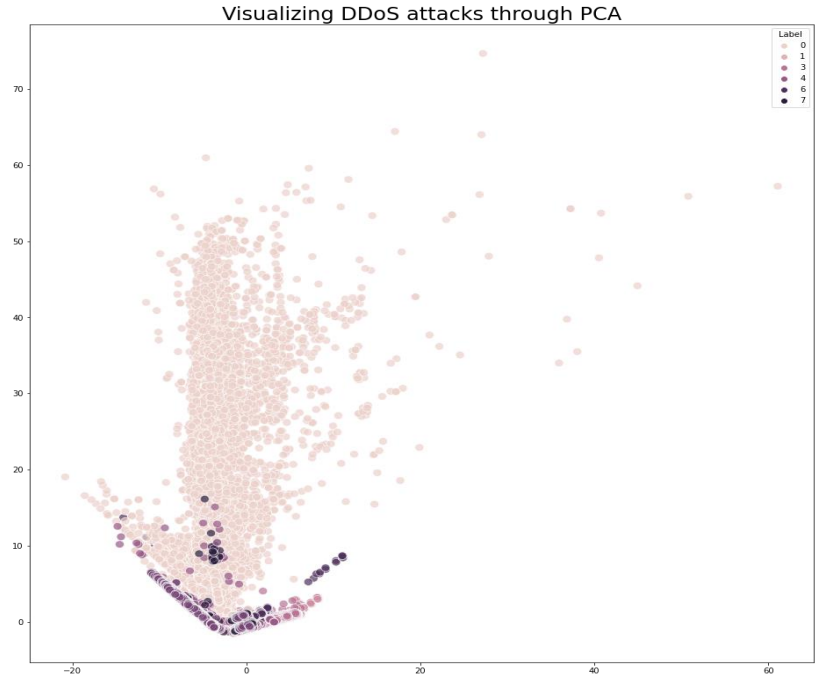
Analysis

We observe that most of packets are inbound since in a DDoS attack incoming packets are the culprit of this attack . Only benign packets were both inbound and outbound.

Dimensionality reduction

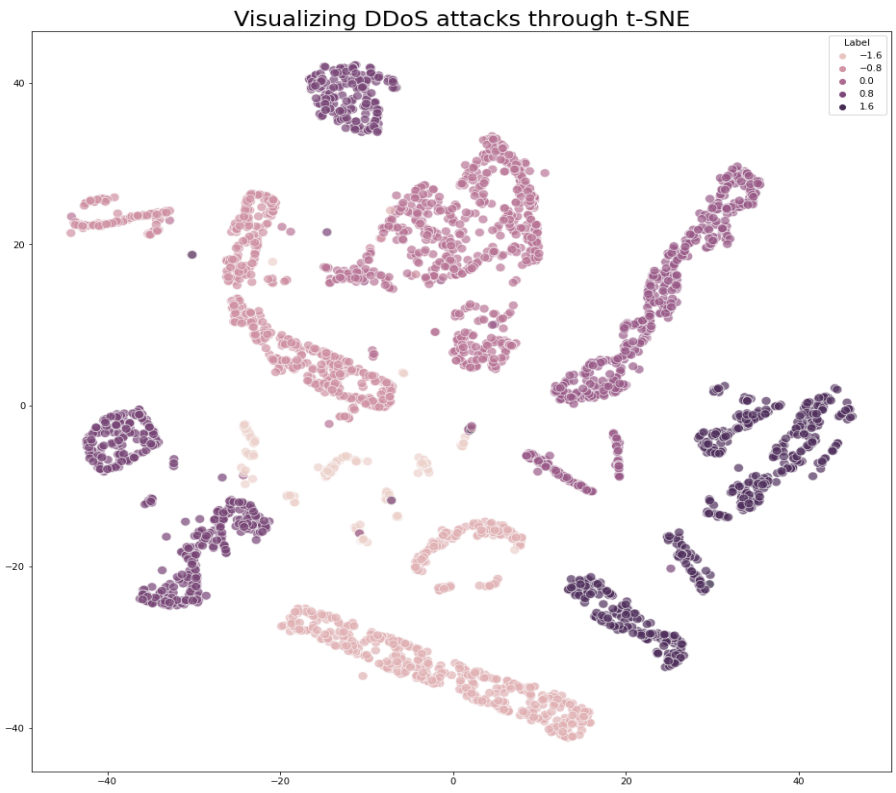
Using Principal Component Analysis(PCA) to reduce dimension from 36 attributes to 2 attributes to achieve better visualization in a two dimensional space.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(normalized_df)
plt.figure(figsize=(16,16))
g1 = sns.scatterplot(principalComponents[:, 0], principalComponents[:, 1], s= 100, hue=df['Label'], cmap='Spectral',alpha=0.7)
plt.title('Visualizing DDoS attacks through PCA', fontsize=24);
```



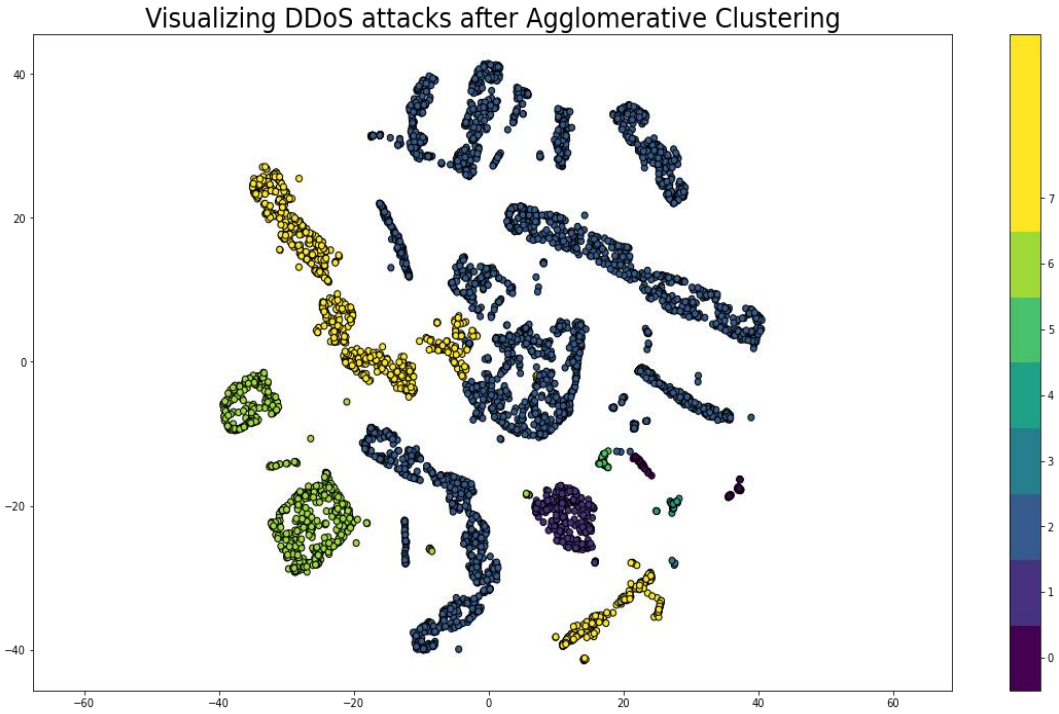
Performing dimensionality reduction with advanced dimensionality reducing algorithm namely , **T-distributed Stochastic Neighbor Embedding** (t-SNE) . Below is the code implementation with perplexity as 40, learning rate of 7000, 300 iterations in 2 dimensions

```
from sklearn.manifold import TSNE
# pca_ = PCA(n_components=2)
temp_df=normalized_df.sample(n=4000)
principalComponents = pca.fit_transform(temp_df)
tsne_ = TSNE(random_state = 42, n_components=2,verbose=0, perplexity=40, n_iter=300,learning_rate=7000).fit_transform(principalComponents)
plt.figure(figsize=(16,16))
g1 = sns.scatterplot(tsne_[:, 0], tsne_[:, 1], s= 100, hue=temp_df['Label'], cmap='Spectral',alpha=0.7)
plt.title('Visualizing DDoS attacks through t-SNE', fontsize=24);
```



Agglomerative clustering internally uses T-SNE with some modification

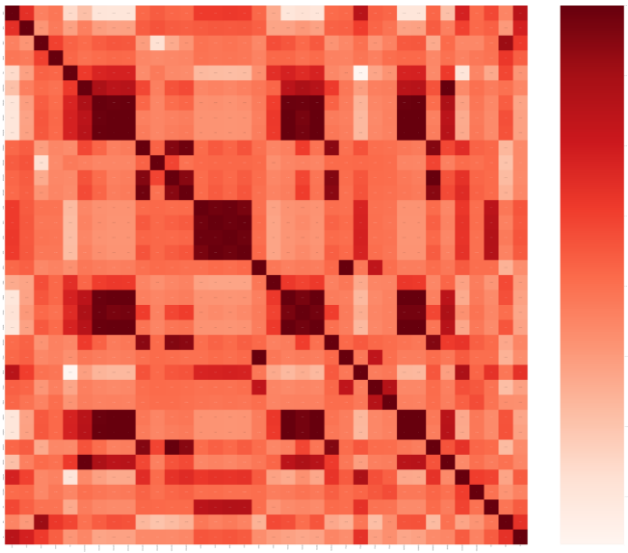
```
from sklearn.cluster import AgglomerativeClustering
Agglo = AgglomerativeClustering(n_clusters=8)
Agglo.fit(principalComponents)
plt.figure(figsize=(20,11))
plt.scatter(tsne_[:, 0],tsne_[:, 1], c=Agglo.labels_,edgecolors='black')
plt.gca().set_aspect('equal', 'datalim')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(8))
plt.title('Visualizing DDoS attacks after Agglomerative Clustering', fontsize=24);
plt.show()
```



Correlation heatmap

```
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(100,200))
cor = normalized_df.corr()
sn.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```

Output



ML Models

Stochastic gradient descent

```
#stochastic gradient descent
from  sklearn.linear_model import SGDClassifier, LinearRegression
from sklearn import metrics
sgd_model = SGDClassifier(loss="log", penalty="l2", max_iter=200)
sgd_model.fit(X_train, y_train)
```

Confusion matrix and classification report of the model

```
from sklearn import metrics
from sklearn.metrics import f1_score
print('Stochastic Gradient Descent')

print('Accuracy = ', metrics.accuracy_score(y_test, y_pred)*100)
print("Confusion Matrix =\n", metrics.confusion_matrix(y_test, y_pred, labels=None,
                                                       sample_weight=None))
print("Recall =", metrics.recall_score(y_test, y_pred, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y_test, y_pred,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                 digits=2,
                                                                 output_dict=False))

print("F1 Score = ",f1_score(y_test, y_pred, average='macro'))
```

```
Stochastic Gradient Descent
Accuracy =  77.0797880879756
Confusion Matrix =
 [[14125    29     0     0    39    65    27    36]
 [    0 46995     2     1  2751     2    11     0]
 [    3     9 49402   383    18     6     1     0]
 [    0     1   287 47912   626     0  1448     1]
 [   24 43773     0   116  2845   116    33    11]
 [    9     0     0     3     0 49651     0     1]
 [    3     0     0 21077    11     4 29060    12]
 [   25     0     0   206    52    20   143    75]]
Recall = 0.770797880879756
Classification Report =
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     14321
           1       0.52      0.94      0.67     49762
           2       0.99      0.99      0.99     49822
           3       0.69      0.95      0.80     50275
           4       0.45      0.06      0.11     46918
           5       1.00      1.00      1.00     49664
           6       0.95      0.58      0.72     50167
           7       0.55      0.14      0.23       521

    accuracy                           0.77    311450
   macro avg       0.77      0.71      0.69    311450
weighted avg       0.78      0.77      0.73    311450

F1 Score =  0.6878111419411918
```

Naive bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
model_naiveB = GaussianNB()
model_naiveB.fit(X_train, y_train)
```

Confusion matrix and classification report of the model

```
from sklearn import metrics
from sklearn.metrics import f1_score

print('Naive Bayes')

print('Accuracy = ', metrics.accuracy_score(y_test, pred)*100)
print("Confusion Matrix =\n", metrics.confusion_matrix(y_test, pred, labels=None,
                                                       sample_weight=None))
print("Recall =", metrics.recall_score(y_test, pred, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y_test, pred,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                  digits=2,
                                                                 output_dict=False))
print("F1 Score = ",f1_score(y_test, pred, average='macro'))
```

```
Naive Bayes
Accuracy =  78.31658372130357
Confusion Matrix =
 [[14215     0     0    31     0    58     0    17]
 [   52 49296     2   318     0     0    85     9]
 [    7     0 49342   457     0     3     0    13]
 [   17     0   199 32636     0     0 17393    30]
 [  156 46110     0   491     0     1    62    98]
 [    9     0     0     1     0 49651     2     1]
 [   42     0     0  1495     0     0 48593    37]
 [   47     0     0    30     0     2   258   184]]
Recall = 0.7831658372130358
```

```
Classification Report =
              precision    recall  f1-score   support

           0       0.98      0.99      0.98     14321
           1       0.52      0.99      0.68     49762
           2       1.00      0.99      0.99     49822
           3       0.92      0.65      0.76     50275
           4       0.00      0.00      0.00     46918
           5       1.00      1.00      1.00     49664
           6       0.73      0.97      0.83     50167
           7       0.47      0.35      0.40       521

    accuracy                           0.78    311450
   macro avg       0.70      0.74      0.71    311450
weighted avg       0.71      0.78      0.73    311450

F1 Score =  0.7069922093260133
```

Decision tree Classifier

```
dtc = tree.DecisionTreeClassifier()
dtc = dtc.fit(X_train, y_train)
```

Confusion matrix and classification report of the model

```
from sklearn import metrics
from sklearn.metrics import f1_score

print('Decsion tree')

print('Accuracy = ', metrics.accuracy_score(y_test, pred)*100)
print("Confusion Matrix =\n", metrics.confusion_matrix(y_test, pred, labels=None,
                                                       sample_weight=None))
print("Recall =", metrics.recall_score(y_test, pred, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y_test, pred,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                  digits=2,
                                                                 output_dict=False))
print("F1 Score = ",f1_score(y_test, pred, average='macro'))
```

```
Decsion tree
Accuracy =  86.80847648097608
Confusion Matrix =
 [[14315     0     0     1     4     1     0     0]
 [    0 29890     2     0 19869     0     0     1]
 [    1     0 49780    35     6     0     0     0]
 [    0     0    29 49546    13     1   677     9]
 [    3 19406     9    11 27456     2     2    29]
 [    1     0     1     2     0 49658     1     1]
 [    0     0     0   742     7     0 49332    86]
 [    0     0     0    13    25     3    92   388]]
Recall = 0.8680847648097608
Classification Report =
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     14321
           1       0.61      0.60      0.60     49762
           2       1.00      1.00      1.00     49822
           3       0.98      0.99      0.98     50275
           4       0.58      0.59      0.58     46918
           5       1.00      1.00      1.00     49664
           6       0.98      0.98      0.98     50167
           7       0.75      0.74      0.75       521

    accuracy                           0.87    311450
   macro avg       0.86      0.86      0.86    311450
weighted avg       0.87      0.87      0.87    311450

F1 Score =  0.862869766907399
```

Logistic Regression Classifier

```
logreg_m = LogisticRegression(random_state=0).fit(X_train, y_train)
```

Confusion matrix and classification report of the model

```
from sklearn import metrics
from sklearn.metrics import f1_score

print('Logistic regression')

print('Accuracy = ', metrics.accuracy_score(y_test, pred)*100)
print("Confusion Matrix =\n", metrics.confusion_matrix(y_test, pred, labels=None,
                                                       sample_weight=None))
print("Recall =", metrics.recall_score(y_test, pred, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y_test, pred,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                  digits=2,
                                                                 output_dict=False))
print("F1 Score = ",f1_score(y_test, pred, average='macro'))
```

```
Logistic regression
Accuracy =  82.11526729812168
Confusion Matrix =
 [[14191    16     0     0    47    66     1     0]
 [    2 42408     2    11  7337     2     0     0]
 [    5     9 49736    54    12     5     1     0]
 [    1     1   157 45621    69     0  4425     1]
 [   86 40655     2   121  5969    46     5    34]
 [    9     0     0     1     0 49651     2     1]
 [    4     0     0  2115     9     4 48007    28]
 [   23     0     0    55    31    14   233   165]]
Recall = 0.8211526729812169
Classification Report =
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     14321
           1       0.51      0.85      0.64     49762
           2       1.00      1.00      1.00     49822
           3       0.95      0.91      0.93     50275
           4       0.44      0.13      0.20     46918
           5       1.00      1.00      1.00     49664
           6       0.91      0.96      0.93     50167
           7       0.72      0.32      0.44       521

    accuracy                           0.82    311450
   macro avg       0.82      0.77      0.77    311450
weighted avg       0.81      0.82      0.80    311450

F1 Score =  0.7656626592628922
```

Random forest Classifier

```
rfc_model=RandomForestClassifier(n_estimators=100)

rfc_model=rfc_model.fit(X_train,y_train)
```

Confusion matrix and classification report of the model

```
from sklearn import metrics
from sklearn.metrics import f1_score

print('Random Forest')

print('Accuracy = ', metrics.accuracy_score(y_test, pred)*100)
print("Confusion Matrix =\n", metrics.confusion_matrix(y_test, pred, labels=None,
                                                       sample_weight=None))
print("Recall =", metrics.recall_score(y_test, pred, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y_test, pred,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                  digits=2,
                                                                 output_dict=False))
print("F1 Score = ",f1_score(y_test, pred, average='macro'))
```

```
Accuracy =  87.33376143843313
Confusion Matrix =
 [[14320     0     0     0     1     0     0     0]
 [    0 32464     3     0 17295     0     0     0]
 [    1     0 49777    40     4     0     0     0]
 [    0     0    19 49829     7     0   418     2]
 [    3 20722     5     8 26167     0     0    13]
 [    1     0     0     1     1 49659     1     1]
 [    0     0     0   707     9     0 49397    54]
 [    1     0     0    14    29     2    87   388]]
Recall = 0.8733376143843313
Classification Report =
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     14321
           1       0.61      0.65      0.63     49762
           2       1.00      1.00      1.00     49822
           3       0.98      0.99      0.99     50275
           4       0.60      0.56      0.58     46918
           5       1.00      1.00      1.00     49664
           6       0.99      0.98      0.99     50167
           7       0.85      0.74      0.79       521

    accuracy                           0.87    311450
   macro avg       0.88      0.87      0.87    311450
weighted avg       0.87      0.87      0.87    311450

F1 Score =  0.872025890026428
```

XGBoost Classifier

```
xgbmodel = XGBClassifier()
xgbmodel.fit(X_train, y_train,verbose=1)
```

Confusion matrix and classification report of the model

```
from sklearn import metrics
from sklearn.metrics import f1_score

print('XGBoost')

print('Accuracy = ', metrics.accuracy_score(y_test, pred)*100)
print("Confusion Matrix =\n", metrics.confusion_matrix(y_test, pred, labels=None,
                                                       sample_weight=None))
print("Recall =", metrics.recall_score(y_test, pred, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y_test, pred,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                  digits=2,
                                                                 output_dict=False))
print("F1 Score = ",f1_score(y_test, pred, average='macro'))
```

```
Accuracy =  89.23583239685343
Confusion Matrix =
 [[13827     4    17   145    36    13   270     9]
 [    2 46815     2     1  2932     2     0     8]
 [    2     0 49777    36     3     1     2     1]
 [    1     0    35 49850     8    22   291    68]
 [    5 28494     3    25 18349     1    11    30]
 [    2     0     1     1     0 49658     2     0]
 [    8     0     1   763    12     8 49288    87]
 [    2     2     2    28    35     1    90   361]]
Recall = 0.8923583239685343
Classification Report =
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     14321
           1       0.62      0.94      0.75     49762
           2       1.00      1.00      1.00     49822
           3       0.98      0.99      0.99     50275
           4       0.86      0.39      0.54     46918
           5       1.00      1.00      1.00     49664
           6       0.99      0.98      0.98     50167
           7       0.64      0.69      0.67       521

    accuracy                           0.89    311450
   macro avg       0.89      0.87      0.86    311450
weighted avg       0.91      0.89      0.88    311450

F1 Score =  0.8627425364586018
```

Light Gradient Boosting Machine Classifier

```
from lightgbm import LGBMClassifier
lgb_model = LGBMClassifier(random_state=123)
lgb_model.fit(X_train, y_train)
lgb_model.score(X_test, y_test)
```

Confusion matrix and classification report of the model

```
from sklearn import metrics
from sklearn.metrics import f1_score

print('Light bgm')

print('Accuracy = ', metrics.accuracy_score(y_test, y_pred)*100)
print("Confusion Matrix =\n", metrics.confusion_matrix(y_test, y_pred, labels=None,
                                                       sample_weight=None))
print("Recall =", metrics.recall_score(y_test, y_pred, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y_test, y_pred,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                  digits=2,
                                                                 output_dict=False))
print("F1 Score = ",f1_score(y_test, y_pred, average='macro'))
```

```
Accuracy =  89.28174666880719
Confusion Matrix =
 [[14321     0     0     0     0     0     0     0]
 [    1 46887     3     0  2870     0     0     1]
 [    0     0 49765    39    13     5     0     0]
 [    0     1    19 50077    18     0   160     0]
 [    8 28575     7    65 18247     1     0    15]
 [    1     0     0     1     1 49659     2     0]
 [    1     0     0  1378     8     0 48763    17]
 [    1     0     0    17    41     0   113   349]]
Recall = 0.8928174666880719
Classification Report =
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     14321
           1       0.62      0.94      0.75     49762
           2       1.00      1.00      1.00     49822
           3       0.97      1.00      0.98     50275
           4       0.86      0.39      0.54     46918
           5       1.00      1.00      1.00     49664
           6       0.99      0.97      0.98     50167
           7       0.91      0.67      0.77       521

    accuracy                           0.89    311450
   macro avg       0.92      0.87      0.88    311450
weighted avg       0.91      0.89      0.88    311450

F1 Score =  0.8778246144836931
```

Saving model

```
import joblib
filename = '/content/drive/My Drive/ML_proj/sgb_model.sav'
joblib.dump(sgd_model, filename)
```

```
['/content/drive/My Drive/security_proj/sgb_model.sav']
```

```
filename = '/content/drive/My Drive/ML_proj/model_naiveB.sav'
joblib.dump(model_naiveB, filename)
```

```
['/content/drive/My Drive/security_proj/model_naiveB.sav']
```

```
filename = '/content/drive/My Drive/ML_proj/dtc.sav'
joblib.dump(dtc, filename)
```

```
['/content/drive/My Drive/security_proj/dtc.sav']
```

```
filename = '/content/drive/My Drive/ML_proj/logreg_m.sav'
joblib.dump(logreg_m, filename)
```

```
['/content/drive/My Drive/security_proj/logreg_m.sav']
```

```
filename = '/content/drive/My Drive/ML_proj/rfc_model.sav'
joblib.dump(rfc_model, filename)
```

```
['/content/drive/My Drive/security_proj/rfc_model.sav']
```

```
filename = '/content/drive/My Drive/ML_proj/xgbmodel.sav'
joblib.dump(xgbmodel, filename)
```

```
['/content/drive/My Drive/security_proj/xgbmodel.sav']
```

Hard voter ensemble classifier

```
from sklearn.ensemble import VotingClassifier
```

```
estimators = []
estimators.append(('sgd_n',sgd_model))
estimators.append(('model_naiveB_n',model_naiveB))
estimators.append(('dtc_n',dtc))
estimators.append(('logreg_m_n',logreg_m))
estimators.append(('lgm_model',lgb_model))
estimators.append(('rfc_model_n',rfc_model))
estimators.append(('xgbmodel_n',xgbmodel))
```

```
ensemble = VotingClassifier(estimators,voting='hard')
```

```
ensemble.fit(X_train, y_train)
y_pred = ensemble.predict(X_test)
```

Classification report and confusion matrix

```
from sklearn import metrics
from sklearn.metrics import f1_score

print('Ensemble learning hrad voting')

print('Accuracy = ', metrics.accuracy_score(y_test, y_pred)*100)
print("Confusion Matrix =\n", metrics.confusion_matrix(y_test, y_pred, labels=None,
                                                       sample_weight=None))
print("Recall =", metrics.recall_score(y_test, y_pred, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y_test, y_pred,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                  digits=2,
                                                                 output_dict=False))
print("F1 Score = ",f1_score(y_test, y_pred, average='macro'))
```

```
Ensemble learning hrad voting
Accuracy =  84.0767378391395
Confusion Matrix =
 [[14261     2     0     0     0    58     0     0]
 [    2 49170     2     6   582     0     0     0]
 [    5     9 49752    50     2     4     0     0]
 [    1     1   116 49023     3     0  1130     1]
 [   89 45206     3   108  1476    17     0    19]
 [    9     0     0     1     0 49651     2     1]
 [    4     0     0  1784    10     2 48341    26]
 [   28     0     1    21    11     9   268   183]]
Recall = 0.840767378391395
Classification Report =
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14321
           1       0.52      0.99      0.68     49762
           2       1.00      1.00      1.00     49822
           3       0.96      0.98      0.97     50275
           4       0.71      0.03      0.06     46918
           5       1.00      1.00      1.00     49664
           6       0.97      0.96      0.97     50167
           7       0.80      0.35      0.49       521

    accuracy                           0.84    311450
   macro avg       0.87      0.79      0.77    311450
weighted avg       0.87      0.84      0.80    311450

F1 Score =  0.7694795289406422
```

Soft voting classifier

```
estimators_soft = []
estimators_soft.append(('sgd_n',sgd_model))
estimators_soft.append(('model_naiveB_n',model_naiveB))
estimators_soft.append(('dtc_n',dtc))
estimators_soft.append(('logreg_m_n',logreg_m))
estimators.append(('lgm_model',lgb_model))
estimators.append(('xgboost',xgb_model))


# estimators.append(('rfc_model_n',rfc_model))
# estimators.append(('xgbmodel_n',xgbmodel))


from sklearn.ensemble import VotingClassifier
ensemble_soft = VotingClassifier(estimators_soft,voting='soft')
```

```
from sklearn import metrics
from sklearn.metrics import f1_score

print('Ensemble approach soft learning')

print('Accuracy = ', metrics.accuracy_score(y_test, y_pred)*100)
print("Confusion Matrix =\n", metrics.confusion_matrix(y_test, y_pred, labels=None,
                                                       sample_weight=None))
print("Recall =", metrics.recall_score(y_test, y_pred, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y_test, y_pred,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                  digits=2,
                                                                 output_dict=False))
print("F1 Score = ",f1_score(y_test, y_pred, average='macro'))
```

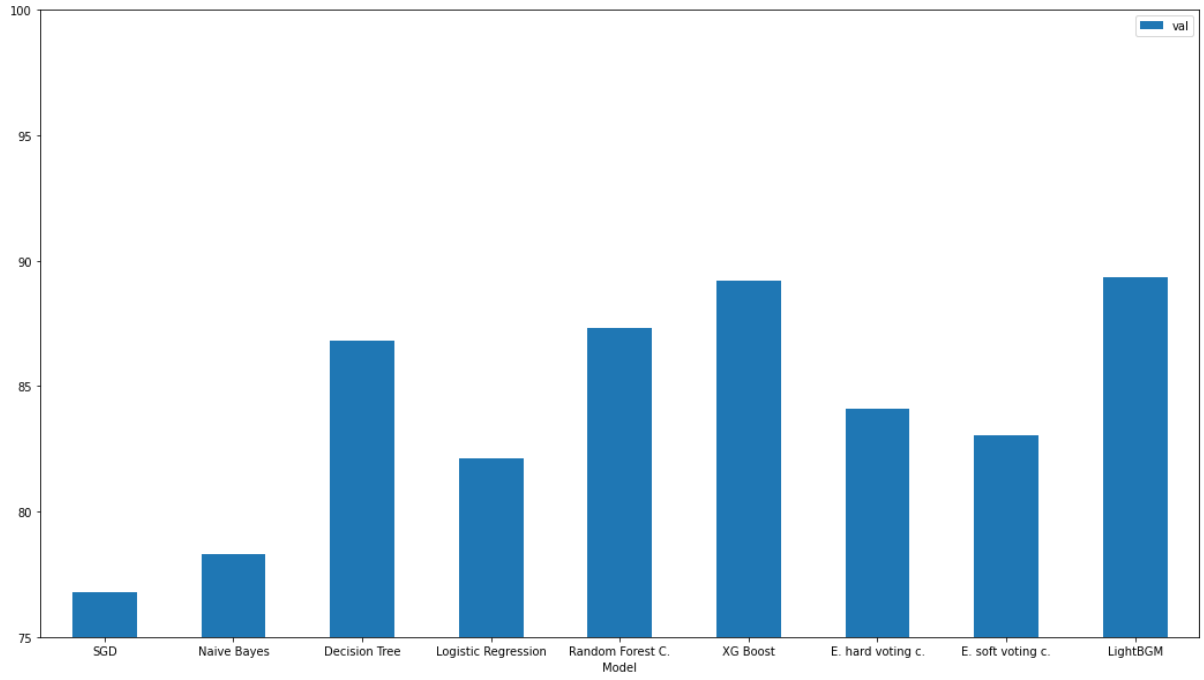
```
Ensemble approach soft learning
Accuracy =  83.0274522395248
Confusion Matrix =
 [[14254     0     0     0     9    58     0     0]
 [    2 48973     2    11   772     2     0     0]
 [    5     8 49539   252    13     5     0     0]
 [    2     0   160 46647    17     0  3447     2]
 [   85 45740     0   118   896    46     5    28]
 [    9     0     0     1     0 49651     2     1]
 [    4     0     0  1676     6     4 48445    32]
 [   27     0     0    14     8    14   274   184]]
Recall = 0.830274522395248
Classification Report =
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14321
           1       0.52      0.98      0.68     49762
           2       1.00      0.99      1.00     49822
           3       0.96      0.93      0.94     50275
           4       0.52      0.02      0.04     46918
           5       1.00      1.00      1.00     49664
           6       0.93      0.97      0.95     50167
           7       0.74      0.35      0.48       521

    accuracy                           0.83    311450
   macro avg       0.83      0.78      0.76    311450
weighted avg       0.83      0.83      0.78    311450

F1 Score =  0.758772826182561
```

Final comparative graph of accuracy of different models



We found that LGBM had highest accuracy among all the models

7.Conclusion

In this project we performed a multiclass classification to detect if DDoS attack has taken place or not. We have also found out the type of DDoS attack that has taken place. We used several machine learning classifiers. The performance of the LGBM (Light Gradient Boosting Machine) turned out to be the highest at 89%. We also made an Ensemble Vote Classifier model with LGBM, Logistic Regression, Naïve Bayes and Decision Tree. Finally, we conclude that LGBM has the highest accuracy amongst all the used models. Also, LGBM is very lightweight as compared to XG Boost with very less training time and almost similar accuracy. LGBM is also less prone to overfitting as compared to XG Boost. Hence, we can explore the possibilities of using LGBM in place of XG Boost to improve model training time and performance.

----------**THE END**----------