[7]B. Tech. Project Part-II (COC4980)
Final Report

DARK WEB TRAFFIC DETECTION AND CHARACTERIZATION SYSTEM

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF

Bachelor of Technology

IN

COMPUTER ENGINEERING

BY

MOHAMMAD YUSUF
19COB149
MOHD NAVED KHAN
19COB023

Under the Guidance of

Mr. Mohammed Abdul Qadeer

Department of Computer Engineering
Zakir Husain College of Engineering & Technology
Aligarh Muslim University
Aligarh (India)-202002, India

2022-2023

Declaration

The work presented in project entitle “Dark Web Traffic Detection and Characterization System” submitted to the department of Computer Engineering, Zakir Husain College of Engineering and Technology, Aligarh Muslim University, for the award of the degree of Bachelors of Technology in Computer Engineering, during the session 2022-23, is my original work. I have neither plagiarized nor submitted the same work for the award of any degree.

Date: 8th May 2023

Place: Aligarh

Mohammad Yusuf

Mohd Naved Khan

Certificate

This is to certify that the project entitled “Dark Web Traffic Detection and Characterization System”, being submitted by “Mohammad Yusuf” and “Mohd Naved Khan”, in partial fulfillment of requirements for the award of the degree of Bachelors of Technology in Computer Engineering, during the session 2022-23, in the department of Computer Engineering, Zakir Husain College of Engineering and Technology, Aligarh Muslim University, is a record of candidates own work carried out by him under my supervision and guidance.

Mr. Mohammed Abdul Qadeer
Department of Computer Engineering
ZHCET, AMU, Aligarh

Acknowledgement

Firstly, I am thankful to the ALMIGHTY for providing me with this opportunity to exhibit through this project. I would like to express my gratitude to my supervisor, Mr. Mohammed Abdul Qadeer, for the fantastic ideas and recommendations he provided, as well as, most significantly, his trust in me to carry out this task. The motivation to get things done has come to me as a result of his expertise, comments, and direction.

Many thanks to my classmates at the Department of Computer Engineering for creating such a friendly and simulating working environment.

Last, but not the least, I am very much grateful to my parents and all the members of my family for their moral support and care that they show towards me during the period of work.

Table of contents

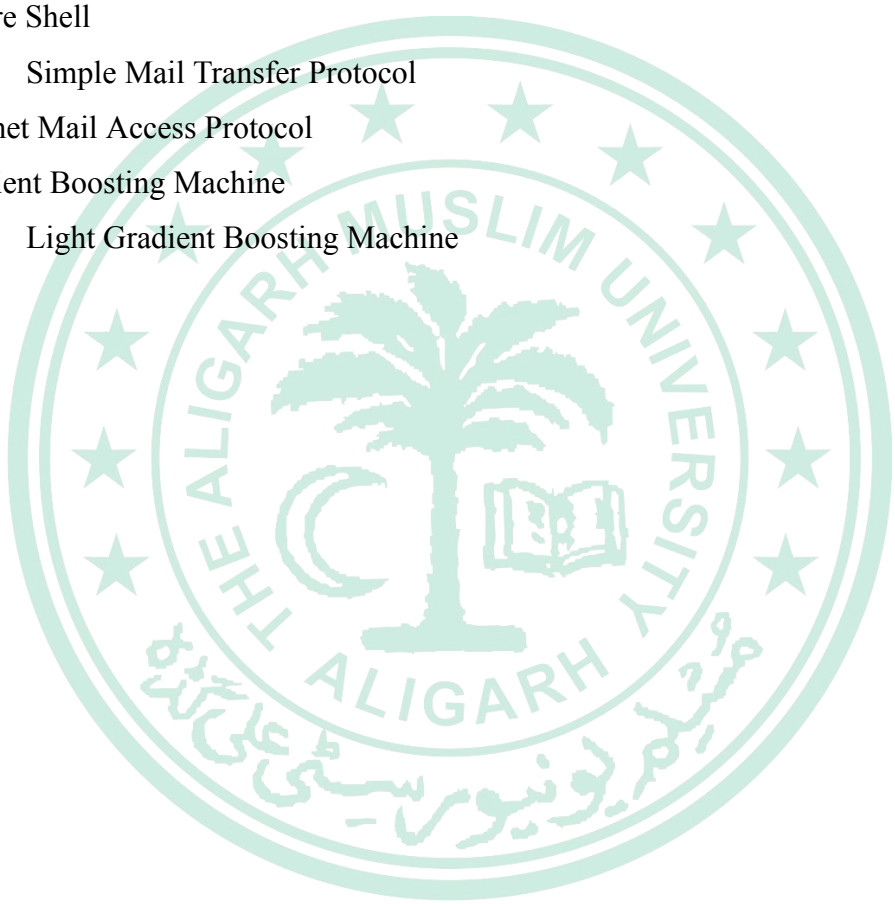
Abstract

Network traffic detection is closely associated with network security which makes it a hot research topic. With the development of encryption technology, many crimes have occurred on the dark web due to the fact that traffic from such networks is becoming more difficult to filter out. Nowadays adversaries are working on multi-layer encryption strategies in dark-web. For example, instead of just using Tor, they are using VPN and Tor together which make it more complicated and difficult to monitor and detect. So, classifying traffic flows according to the applications that generate them like VoIP, Audio Streaming, P2P, File-Transfer and Video Streaming is an important task for monitoring the trends of the applications in multilayer network communications. However, an accurate method that can reliably identify the generating application of flow is still to be developed. Most of the existing techniques are based on machine learning classifiers but there are less efforts made to detect and characterize darknet traffic using deep learning.

The objective of this project is to classify data into 8 different constituents of Darknet using various machine learning (ML) algorithms, including Decision Tree, K-Nearest Neighbors (KNN), Random Forest Classifier, Gradient Boosting Machine (GBM), and LightGBM. The classification is based on parameters such as source and destination IP, flow ID, and packet information. To improve classification accuracy, a novel ensemble model is proposed that combines different algorithms using voting techniques. Feature extraction is performed to identify the most useful and correlated parameters out of the 85 available. The performance of each classifier and the proposed model is evaluated using baseline classification, 10-fold cross-validation, and hyperparameter tuning. The evaluation measures used are accuracy, precision, recall, and F1 score. The proposed ensemble model is compared to existing state-of-the-art models and other classifiers.

Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
TOR	The Onion Routing
QoS	Quality of Service
P2P	Peer to Peer
NAT	Network Address Translation
VPN	Virtual Private Network
TCP	Transmission Control Protocol
IP	Internet Protocol
VoIP	Voice over Internet Protocol
FTP	File Transfer Protocol
SSH	Secure Shell
SMTP	Simple Mail Transfer Protocol
IMAP	Internet Mail Access Protocol
GBM	Gradient Boosting Machine
LGBM	Light Gradient Boosting Machine



List of Figures:

List Of Tables/Charts:

Chapter 1 Introduction

1.1 Overview

Tor is a low-latency anonymity network based on the concept of onion routing [Reed et al. 1998]. The network today consists of approximately 6,000 volunteer-operated routers [Tor Project 2015a], known as Onion Routers (ORs). Each OR creates a router descriptor that contains its contact information, such as its IP address, ports, public keys, and its bandwidth capabilities, and sends the descriptor to directory authorities. These authorities construct a network consensus document, which they send, along with the descriptors, to directory servers. Tor clients, nicknamed Onion Proxies (OPs), download the descriptors and the consensus from the directory servers in order to build paths, referred to as circuits, through the network before they can communicate with their Internet destinations. Each circuit usually consists of three ORs, or hops, which are referred to as the entry guard, middle, and exit OR, according to their position in the circuit. ORs in a circuit are connected by TCP connections, and TLS [Dierks and Rescorla 2008] is used to provide hop-by-hop authenticity, data integrity and confidentiality. Traffic in Tor travels in fixed size units (512 bytes) called cells. Figure 1 visualizes the Tor network.

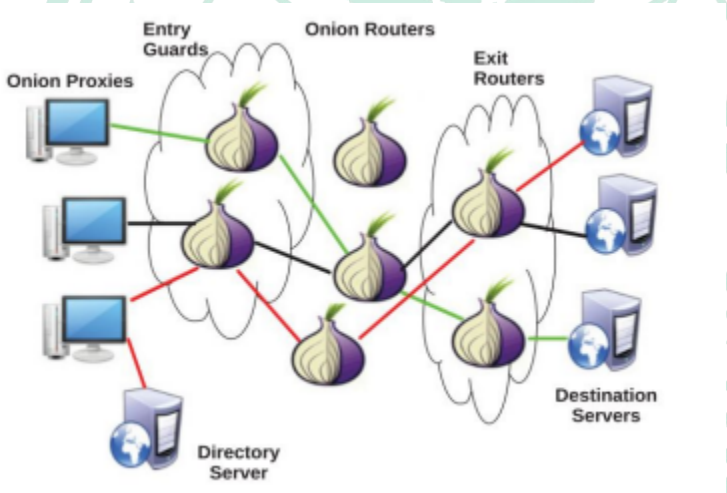


Fig 1.1 Tor Network

The OP builds circuits by first selecting three routers, Xi, according to Tor's bandwidthweighted router selection algorithm. Next, to start establishing the circuit, the OP sends a create_fast command to X1, which responds with a created_fast reply. To extend the encrypted channel, the OP sends an extend command to X1, containing in its payload a create command and the first half of a Diffie-Hellman (DH) handshake for router X2, encrypted to X2's public key. Router X1 forwards this create command to router X2, and when it receives a created cell back from router X2, it forwards its payload in an extended cell to the OP to finish the client's DH handshake with router X2. The same procedure is carried out for each subsequent OR added to the circuit, establishing a shared session key between the OP and each of the routers in the circuit. Cells sent along a circuit by an OP are multiply encrypted, with one layer of encryption (using the above session key) for each hop in the circuit. Each hop decrypts one layer before passing the cell to the next hop. For performance reasons, an OP preemptively creates a number of spare circuits for its user applications. In order to communicate with the client application (for example, a web browser), the OP exposes a SOCKS proxy, which the browser is configured to use. When the browser connects a new TCP stream to the OP's SOCKS proxy, the OP attaches the new stream to an appropriate pre-established circuit. (OPs can multiplex several TCP streams over one circuit.) Once a circuit begins being used, it generally has a lifetime of 10 minutes, after which no new streams will be attached to it, though the circuit will continue to exist until all streams it contains are closed.

Note that only the exit node can observe the user's traffic, and only the entry guard knows the identity of the user. If both the entry guard and exit node cooperate, however, they can use traffic analysis to link the initiator to her destination.

1.2 Background & Motivation

Tor allows servers to provide content and services over the network while maintaining their anonymity. First, a hidden service generates a public/private key pair and chooses some routers randomly as introduction points. Then, to advertise its service, the hidden service creates a signed descriptor containing the introduction point information and its own public key. Based on the contents of the descriptor and a validity time period t, a descriptor ID will be generated, and then the descriptor will be published in a Distributed Hash Table (DHT) hash ring that is formed by hidden service directories. The hidden service computes the directory responsible for holding its descriptor based on a closeness metric between the descriptor ID and the directory's fingerprint, which is the SHA-1 hash of the directory's public key. (At the time of writing, descriptors are uploaded to six hidden service directories.) Note that the responsible directories for a specific descriptor change after time t. The hidden service publishes its onion address,which is an address of the form abc.onion where abc is a truncated hash of the hidden service public key. The hidden service also maintains circuits to its introduction points and informs them of its public key. When a client is interested in connecting to hidden service X, it first searches for the latter's onion address abc.onion, which can be queried through one of the public routers that is part of the DHT. Next, the client starts preparing for the connection to X by constructing a circuit to a randomly chosen Rendezvous Point (RP). The client sends a command cell to the RP containing a 20-byte arbitrary ID to serve as a rendezvous cookie.

Tor allows servers to provide content and services over the network while maintaining their anonymity. First, a hidden service generates a public/private key pair and chooses some routers randomly as introduction points. Then, to advertise its service, the hidden service creates a signed descriptor containing the introduction point information and its own public key. Based on the contents of the descriptor and a validity time period t, a descriptor ID will be generated, and then the descriptor will be published in a Distributed Hash Table (DHT) hash ring that is formed by hidden service directories. The hidden service computes the directory responsible for holding its descriptor based on a closeness metric between the descriptor ID and the directory's fingerprint, which is the SHA-1 hash of the directory's public key. (At the time of writing, descriptors are uploaded to six hidden service directories.) Note that the responsible directories for a specific descriptor change after time t.

The hidden service publishes its onion address,which is an address of the form abc.onion where abc is a truncated hash of the hidden service public key. The hidden service also maintains circuits to its introduction points and informs them of its public key. When a client is interested in connecting to hidden service X, it first searches for the latter's onion address abc.onion, which can be queried through one of the public routers that is part of the DHT. Next, the client starts preparing for the connection to X by constructing a circuit to a randomly chosen Rendezvous Point (RP). The client sends a command cell to the RP containing a 20-byte arbitrary ID to serve as a rendezvous cookie.

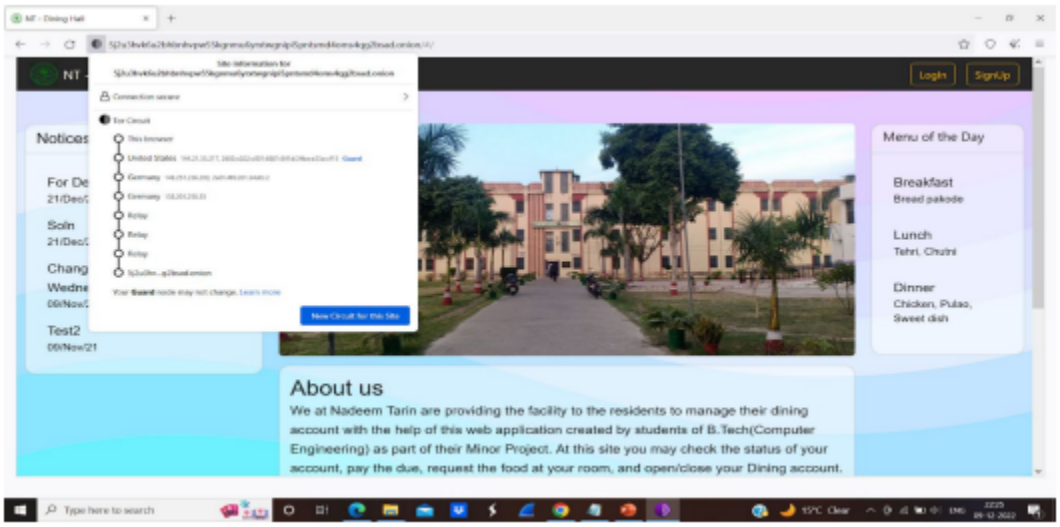


Fig 1.2 Minor Project hosted on Tor

Traffic classification has been the topic of many research efforts, but the quick evolution of Internet services and the pervasive use of encryption makes it an open challenge. Encryption is essential in protecting the privacy of Internet users, a key technology used in the different privacy enhancing tools that have appeared in the recent years. Tor is one of the most popular of them, it decouples the sender from the receiver by encrypting the traffic between them, and routing it through a distributed network of servers. With Tor over VPN it leaves the ISP of the user to see if the user is connected to VPN only, thus creating multi-layer encrypted traffic. We present a time analysis on Tor traffic flows, captured between the client and the entry node. We define two scenarios, one to detect Tor traffic over VPN flows and the other to detect the application type: Browsing, Chat, Streaming, Mail, VoIP, P2P or File Transfer.

1.3 Objective

Network traffic detection is closely associated with network security which makes it a hot research topic. With the development of encryption technology, many crimes have occurred on the dark web due to the fact that traffic from such networks is becoming more difficult to filter out. Nowadays adversaries are working on multi-layer encryption strategies in dark-web. For example, instead of just using Tor, they are using VPN and Tor together which make it more complicated and difficult to monitor and detect. So, classifying traffic flows according to the applications that generate them like VoIP, Audio Streaming, P2P, File-Transfer and Video Streaming is an important task for monitoring the trends of the applications in multilayer network communications. However, an accurate method that can reliably identify the generating application of flow is still to be developed. Most of the existing techniques are based on machine learning classifiers but there are less efforts made to detect and characterize dark net traffic using deep learning. In this project, we aim to address multi-layer traffic in dark web environments and the ultimate objective is to offer a tool using AI to network operators that will provide a meaningful detection and classification per application, and with useful insight into the multi-layer traffic behavior.

1.4 Original Contribution

We build an ensemble model using voting technique.A voting ensemble is an ensemble machine learning model that unites the forecasts from multiple other models.It is a technique that may be used to improve model performance, perfectly achieving better performance than any single model used in the ensemble.We have compared this model with existing ensemble techniques like GBM, LGBM.

Chapter 2 : Literature Review

2.1 Overview

In this section brief discussion about related work in the domain of Traffic Detection.We have discussed various datasets available for carrying out model training and their features and then we have discussed various methodologies employed for classification.

Tor Research papers	Type	Features	Applications	Protocols
Alsabah et.al 2012	Packet Information	Packet features such as circuit lifetime,cell inter-arrival times,number of cells sent recently from the network		
Luo et.al 2014	Network Traffic	Burst volumes such as total size of all packets and directions		P2P,FTP.I M, Web
Lashkari et.al 2016	Network Traffic	Time-related features	Browsing,Email ,Chat,AudioStreaming, Video Streaming,File Transfer,P2P,VoIP	HTTP,HTTPS, Web, SMTP, PoP3/ IMAP/ SSL,SFTP

Authors	ML Techniques used	Comparison with others
Tama, B. A., & Rhee, K. H. (2019)	They proposed an improved detection performance of anomaly-based intrusion detection system (IDS) using gradient boosted machine (GBM).	The performance of GBM is compared with RF, DNN, SVM, CART. GBM is the best performer in terms of specificity and AUC value.
Khafajeh, H. (2020)	They presented a comparative study of AdaBoost, GBM, Random Forest, Extra Trees, and Logistic Regression, KNN, Perceptron with LightGBM based on classification performance and computational cost.	This model is selected due to, high-performance level, being accessible, various selections and fast implementation of hyper parameters.

Table 1: Literature Review

Chapter 3 Design Methodology

3.1 Introduction

The methodology being followed comprises 4 stages, namely Data Preprocessing, Feature Engineering,Model Building and Model Evaluation.All of them have been discussed in detail below.

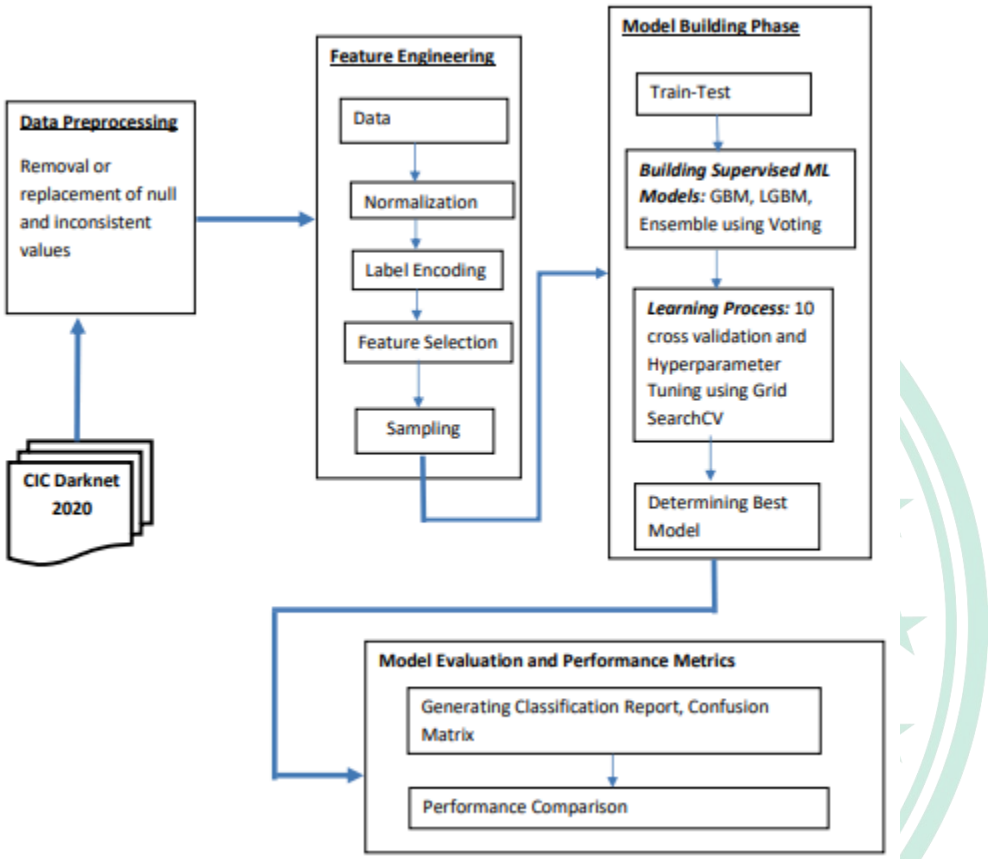


Fig 3.1 System Architecture

About the Dataset:

In **CICDarknet2020** dataset, a two-layered approach is used to generate benign and darknet traffic at the first layer. The darknet traffic constitutes Audio-Stream, Browsing, Chat, Email, P2P, Transfer, Video-Stream and VOIP which is generated at the second layer. To generate the representative dataset, previously generated datasets, namely, ISCXTor2016and ISCXVPN2016, have been amalgamated and respective VPN and Tor traffic are combined in corresponding Darknet categories.

No. of rows: 1.4 Lacs approx.

No. of columns: 85

Traffic Category	Applications used
Audio-Stream	Vimeo and Youtube
Browsing	Firefox and Chrome
Chat	ICQ, AIM, Skype, Facebook and Hangouts
Email	SMTPS, POP3S and IMAPS
P2P	uTorrent and Transmission (BitTorrent)
Transfer	Skype, FTP over SSH (SFTP) and FTP over SSL (FTPS) using Filezilla and an external service
Video-Stream	Vimeo and Youtube
VOIP	Facebook, Skype and Hangouts voice calls

Fig 3.2 Darknet Traffic Details

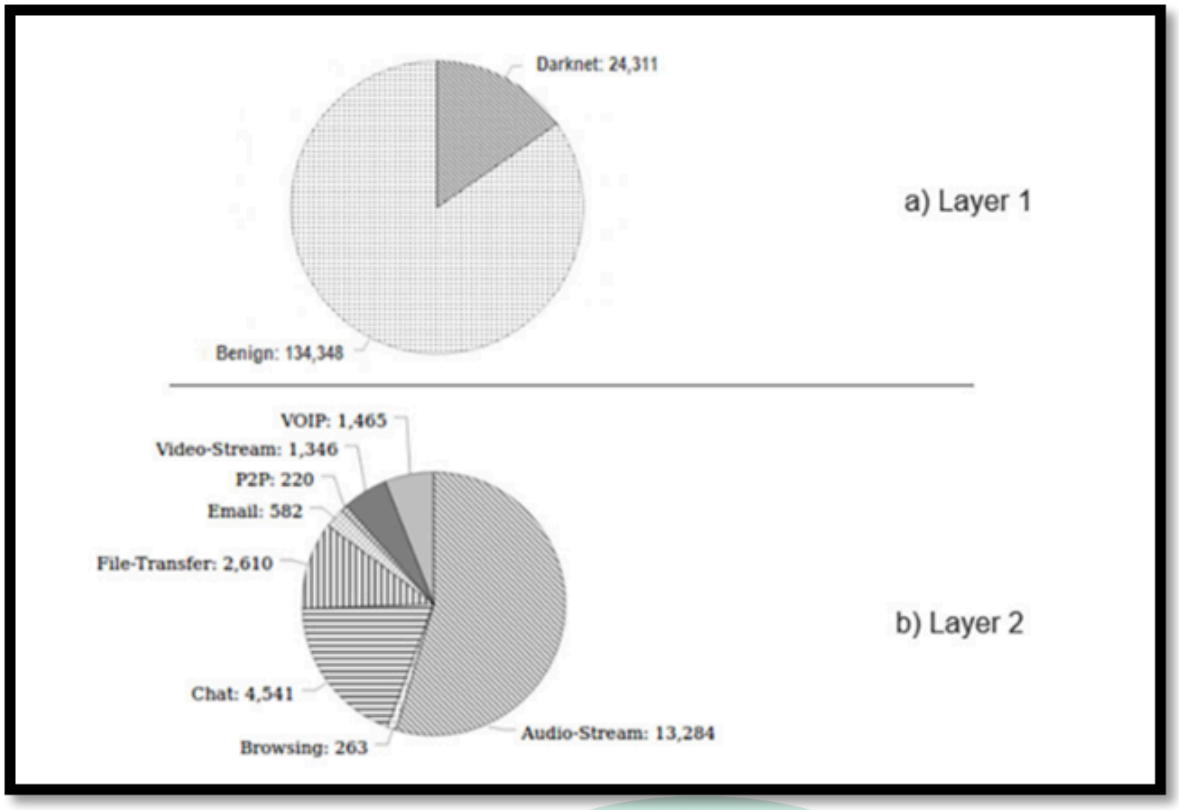


Fig 3.3 (a) presents the details of number of samples of benign and darknet traffic at first layer and (b)highlights the number of encrypted flows in our darknet traffic

First we will be importing all the python libraries and sklearn modules.

Importing thedataset:

The dataset DARKNET 2020 can be downloaded from

https://www.unb.ca/cic/datasets/darknet2020.html

Data Cleaning Phase:

We will be handling the missing values and replacing them with the mean value of the respective column.Also the null and NaN values will also be replaced by the mean of the column.The columns also contain infinite values. To deal with such values, first we will find

the row index of such values, then convert them into NaN and delete the entirerow.

Data Preprocessing Phase:

- Label encoding is done for all the classes to convert string to number which is needed for further computation.
- This dataset is then balanced, that is, the number of rows of each class is made approximately equal by either over sampling or undersampling.
- Target variable is then separated from the dataframe and a new dataframe is created for the target variable. This target variable is then encoded using Label Encoder to convert into integer from object.
- The dataset is then normalized using min-max scalar.
- The number of columns is large and we will not be needing all of them for classification. So we will be performing feature extraction using a ranking technique to extract only the relevant columns and proceed further with only those columns.
- There are few columns like timestamp and flow-id that can be removed directly. Also columns like Src IP and Dst IP can be modified to be used further instead of deletion.

Model Training and TestingPhase:

- First the dataset is split into training and testing dataset.
- Various classifiers are defined: **GBM, LightGBM, Random Forest, Decision Tree Classifier and Knn.**
- Initially all the models are defined with the respective hyper parameters. Classification Report is generated and accuracy is obtained. Confusion matrix is also obtained.

- Voting Technique is then used to combine and form an ensemble classifier using Random Forest, Naïve Bayes and Knn classifiers. Both hard voting and soft voting are used and the performance of this ensembled model proved to be better than the individual performances of all the three models as well as GBM and LightGBM.
- 10-Fold Cross Validation is done on both hard voting and soft voting classifiers to determine the performance of the model on the unseen data.
- After this, hyper parameter tuning is done on these models on the training dataset, using GridSearchCV.

Data Flow Diagram

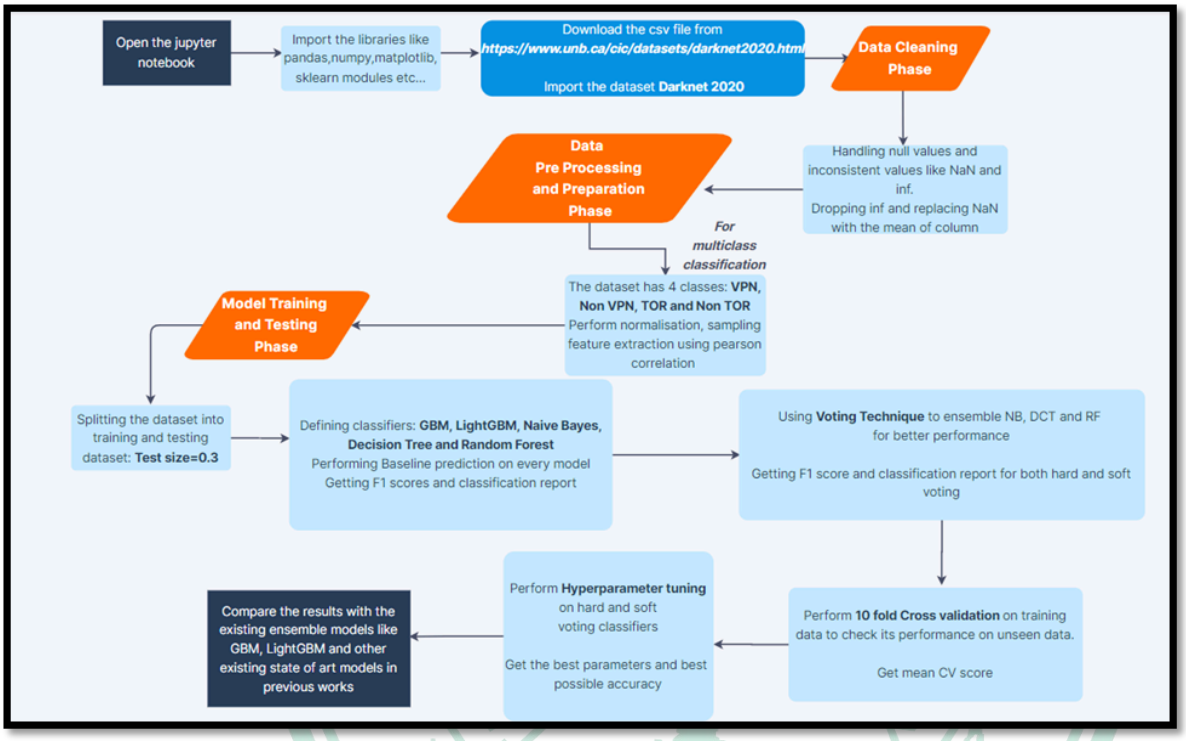


Fig 3.4 Data Flow Diagram

Chapter 4 : Implementation

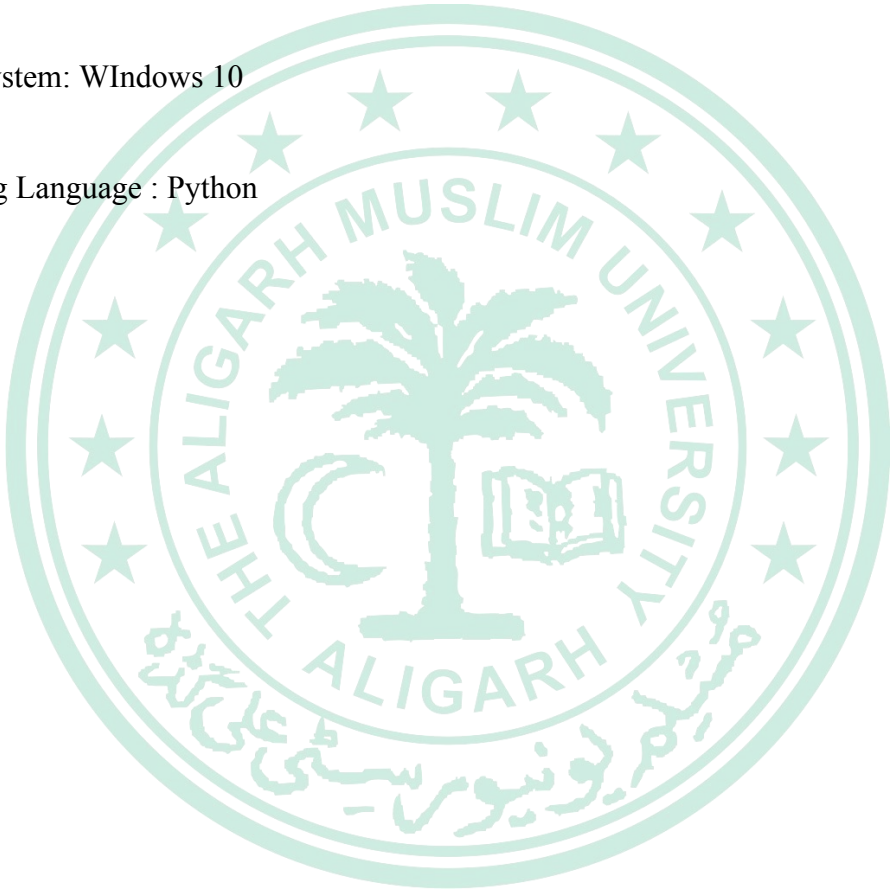
4.1 Hardware & Software Requirements

Hardware:

- Processor:
- RAM: 16 GB RAM
- Hard Disk:

Software:

- Operating System: WIndows 10
- IDE:
- Programming Language : Python



4.2 Implementation details and results

4.2.1 Importing Dataset and Modules:

```
#Load the dataset: CIC-Darknet-2020
df = pd.read_csv('Darknet2020.csv', error_bad_lines=False)
df.head()
```

[3] Python

```
C:\Users\Yusuf\AppData\Local\Temp\ipykernel_9956\3259108598.py:2: FutureWarning: The error_bad_lines argument has been deprecated and will be removed in a future

  df = pd.read_csv('Darknet2020.csv', error_bad_lines=False)
```

IP	Dst Port	Protocol	Timestamp	Flow Duration	Total Fwd Packet	Total Bwd packets	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label	Label.1
99	443	6	24/07/2015 04:09:48 PM	229	1	1	...	0	0	0	0	0.000000e+00	0.000	0.000000e+00	0.000000e+00	Non-Tor	AUDIO-STREAMING
99	443	6	24/07/2015 04:09:48 PM	407	1	1	...	0	0	0	0	0.000000e+00	0.000	0.000000e+00	0.000000e+00	Non-Tor	AUDIO-STREAMING
99	443	6	24/07/2015 04:09:48 PM	431	1	1	...	0	0	0	0	0.000000e+00	0.000	0.000000e+00	0.000000e+00	Non-Tor	AUDIO-STREAMING
20	443	6	24/07/2015 04:09:48 PM	359	1	1	...	0	0	0	0	0.000000e+00	0.000	0.000000e+00	0.000000e+00	Non-Tor	AUDIO-STREAMING
27	19305	6	24/07/2015 04:09:45 PM	10778451	591	400	...	0	0	0	0	1.437760e+15	3117718.131	1.437760e+15	1.437760e+15	Non-Tor	AUDIO-STREAMING

```
#85 columns of the dataset
df.columns
```

[4]

```
Index(['Flow ID', 'Src IP', 'Src Port', 'Dst IP', 'Dst Port', 'Protocol',
       'Timestamp', 'Flow Duration', 'Total Fwd Packet', 'Total Bwd packets',
       'Total Length of Fwd Packet', 'Total Length of Bwd Packet',
       'Fwd Packet Length Max', 'Fwd Packet Length Min',
       'Fwd Packet Length Mean', 'Fwd Packet Length Std',
       'Bwd Packet Length Max', 'Bwd Packet Length Min',
       'Bwd Packet Length Mean', 'Bwd Packet Length Std', 'Flow Bytes/s',
       'Flow Packets/s', 'Flow IAT Mean', 'Flow IAT Std', 'Flow IAT Max',
       'Flow IAT Min', 'Fwd IAT Total', 'Fwd IAT Mean', 'Fwd IAT Std',
       'Fwd IAT Max', 'Fwd IAT Min', 'Bwd IAT Total', 'Bwd IAT Mean',
       'Bwd IAT Std', 'Bwd IAT Max', 'Bwd IAT Min', 'Fwd PSH Flags',
       'Bwd PSH Flags', 'Fwd URG Flags', 'Bwd URG Flags', 'Fwd Header Length',
       'Bwd Header Length', 'Fwd Packets/s', 'Bwd Packets/s',
       'Packet Length Min', 'Packet Length Max', 'Packet Length Mean',
       'Packet Length Std', 'Packet Length Variance', 'FIN Flag Count',
       'SYN Flag Count', 'RST Flag Count', 'PSH Flag Count', 'ACK Flag Count',
       'URG Flag Count', 'CWE Flag Count', 'ECE Flag Count', 'Down/Up Ratio',
       'Average Packet Size', 'Fwd Segment Size Avg', 'Bwd Segment Size Avg',
       'Fwd Bytes/Bulk Avg', 'Fwd Packet/Bulk Avg', 'Fwd Bulk Rate Avg',
       'Bwd Bytes/Bulk Avg', 'Bwd Packet/Bulk Avg', 'Bwd Bulk Rate Avg',
       'Subflow Fwd Packets', 'Subflow Fwd Bytes', 'Subflow Bwd Packets',
       'Subflow Bwd Bytes', 'FWD Init Win Bytes', 'Bwd Init Win Bytes',
       'Fwd Act Data Pkts', 'Fwd Seg Size Min', 'Active Mean', 'Active Std',
       'Active Max', 'Active Min', 'Idle Mean', 'Idle Std', 'Idle Max',
       'Idle Min', 'Label', 'Label.1'],
      dtype='object')
```

```
#Column info
df.info()
```

[5]

```
Output exceeds the size limit. Open the full output data in a text editor
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 141530 entries, 0 to 141529
Data columns (total 85 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   Flow ID                     141530 non-null  object
 1   Src IP                      141530 non-null  object
 2   Src Port                    141530 non-null  int64
 3   Dst IP                      141530 non-null  object
 4   Dst Port                    141530 non-null  int64
 5   Protocol                    141530 non-null  int64
 6   Timestamp                   141530 non-null  object
 7   Flow Duration               141530 non-null  int64
 8   Total Fwd Packet            141530 non-null  int64
 9   Total Bwd packets           141530 non-null  int64
 10  Total Length of Fwd Packet  141530 non-null  int64
 11  Total Length of Bwd Packet  141530 non-null  int64
 12  Fwd Packet Length Max       141530 non-null  int64
 13  Fwd Packet Length Min       141530 non-null  int64
 14  Fwd Packet Length Mean      141530 non-null  float64
 15  Fwd Packet Length Std       141530 non-null  float64
 16  Bwd Packet Length Max       141530 non-null  int64
 17  Bwd Packet Length Min       141530 non-null  int64
 18  Bwd Packet Length Mean      141530 non-null  float64
 19  Bwd Packet Length Std       141530 non-null  float64
...
 83  Label                       141530 non-null  object
 84  Label.1                     141530 non-null  object
dtypes: float64(24), int64(55), object(6)
memory usage: 91.8+ MB
```

Correlation Matrix:

```
#Correlation matrix
corr = df.corr()
corr.head()
```

Python

```
C:\Users\Yusuf\AppData\Local\Temp\ipykernel_9956\2559860652.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future v
  corr = df.corr()
```

	Src Port	Dst Port	Protocol	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	Fwd Packet Length Max	Fwd Packet Length Min	...	Fwd Act Data Pkts	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Activ M
Src Port	1.000000	-0.246451	-0.097384	0.065328	-0.036259	-0.014248	-0.019712	-0.008271	0.076486	-0.090022	...	-0.028619	0.138354	NaN	NaN	NaN	Na
Dst Port	-0.246451	1.000000	-0.321199	0.039227	0.022094	0.014775	0.004451	0.010865	0.004448	-0.178715	...	0.014722	0.246275	NaN	NaN	NaN	Na
Protocol	-0.097384	-0.321199	1.000000	-0.266954	-0.034735	-0.026164	-0.023039	-0.020874	-0.195123	0.564044	...	-0.023370	-0.872467	NaN	NaN	NaN	Na
Flow Duration	0.065328	0.039227	-0.266954	1.000000	0.142110	0.100288	0.072529	0.057008	0.340744	-0.068930	...	0.145455	0.240411	NaN	NaN	NaN	Na
Total Fwd Packet	-0.036259	0.022094	-0.034735	0.142110	1.000000	0.744834	0.457391	0.635688	0.125575	-0.020982	...	0.698507	0.029652	NaN	NaN	NaN	Na

5 rows × 79 columns

4.2.2 Data Cleaning Phase:

In this stage, I will be performing data cleaning like dealing with missing values, inf values, changing the columns like splitting columns if required. So when all the columns that may cause error further due to their data type or format or problematic values will be rectified and then we can proceed towards the data preprocessing stage.

Dropping Null values:

Data cleaning like dealing with missing values,inf values,changing the columns like splitting columns if required for model training.

```
df.dropna() #dropping null values
```

Python

	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Total Fwd Packet	Total Bwd packets	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean
0	10.152.152.11-216.58.220.99-57158-443-6	10.152.152.11	57158	216.58.220.99	443	6	24/07/2015 04:09:48 PM	229	1	1	...	0	0	0	0	0.000000e+00
1	10.152.152.11-216.58.220.99-57159-443-6	10.152.152.11	57159	216.58.220.99	443	6	24/07/2015 04:09:48 PM	407	1	1	...	0	0	0	0	0.000000e+00
2	10.152.152.11-216.58.220.99-57160-443-6	10.152.152.11	57160	216.58.220.99	443	6	24/07/2015 04:09:48 PM	431	1	1	...	0	0	0	0	0.000000e+00
3	10.152.152.11-74.125.136.120-49134-443-6	10.152.152.11	49134	74.125.136.120	443	6	24/07/2015 04:09:48 PM	359	1	1	...	0	0	0	0	0.000000e+00
4	10.152.152.11-173.194.65.127-34697-19305-6	10.152.152.11	34697	173.194.65.127	19305	6	24/07/2015 04:09:45 PM	10778451	591	400	...	0	0	0	0	1.437760e+15
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
141525	10.8.8.246-224.0.0.252-55219-5355-17	10.8.8.246	55219	224.0.0.252	5355	17	22/05/2015 01:55:03 PM	411806	2	0	...	0	0	0	0	0.000000e+00
141526	10.8.8.246-224.0.0.252-64207-5355-17	10.8.8.246	64207	224.0.0.252	5355	17	22/05/2015 02:09:05 PM	411574	2	0	...	0	0	0	0	0.000000e+00

Modifying Src and Dst IP columns:

```
#splitting the Src IP into octets,getting first two ocets
newIP = []
for value in df['Src IP']:
    IP = value.split(".")
    octet1= IP[0]
    octet2= IP[1]
#     print(octet2)
    newIP.append(float(octet1 + '.' + octet2))
```

```
df1 = pd.DataFrame(newIP)  #a new dataframe with the above obtained series
df1.head()
```

	0
0	10.152
1	10.152
2	10.152
3	10.152
4	10.152

```
newIP1 = [] #splitting the Dst IP into octets,getting first two ocets
for value in df['Dst IP']:
    IP = value.split(".")
    octet1= IP[0]
    octet2= IP[1]

#     print(octet2)
    newIP1.append(float(octet1 + '.' + octet2))
```

```
df2 = pd.DataFrame(newIP1)
df2.head()
```

	0
0	216.580
1	216.580
2	216.580
3	74.125
4	173.194

4.2.3 Data Pre Processing Stage:

Label Encoding:

```
# label encoding the data : Label and Label.1
from sklearn.preprocessing import LabelEncoder

Le = LabelEncoder()

df['Label']= Le.fit_transform(df['Label'])
df['Label.1']= Le.fit_transform(df['Label.1'])
```

Python

```
df.head()
```

Python

	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Total Fwd Packet	Total Bwd packets	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Id
0	10.152.152.11-216.58.220.99-57158-443-6	10.152	57158	216.580	443	6	24/07/2015 04:09:48 PM	229	1	1	...	0	0	0	0	0.000000e+00	0.000	0.0000(
1	10.152.152.11-216.58.220.99-57159-443-6	10.152	57159	216.580	443	6	24/07/2015 04:09:48 PM	407	1	1	...	0	0	0	0	0.000000e+00	0.000	0.0000(
2	10.152.152.11-216.58.220.99-57160-443-6	10.152	57160	216.580	443	6	24/07/2015 04:09:48 PM	431	1	1	...	0	0	0	0	0.000000e+00	0.000	0.0000(
3	10.152.152.11-74.125.136.120-49134-443-6	10.152	49134	74.125	443	6	24/07/2015 04:09:48 PM	359	1	1	...	0	0	0	0	0.000000e+00	0.000	0.0000(

```
df5=df.drop(['Flow ID','Timestamp'], axis = 1) #dropping the unnecessary columns
df5.head()
```

Python

	Src IP	Src Port	Dst IP	Dst Port	Protocol	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	
0	10.152	57158	216.580	443	6	229	1	1	0	0	...	0	0	0	0	0.000000e+00	0.000	0.000000e+00	0.0(
1	10.152	57159	216.580	443	6	407	1	1	0	0	...	0	0	0	0	0.000000e+00	0.000	0.000000e+00	0.0(
2	10.152	57160	216.580	443	6	431	1	1	0	0	...	0	0	0	0	0.000000e+00	0.000	0.000000e+00	0.0(
3	10.152	49134	74.125	443	6	359	1	1	0	0	...	0	0	0	0	0.000000e+00	0.000	0.000000e+00	0.0(
4	10.152	34697	173.194	19305	6	10778451	591	400	64530	6659	...	0	0	0	0	1.437760e+15	3117718.131	1.437760e+15	1.4:

5 rows × 83 columns

The next four operations convert exponential values to float values for the ease of further computation.

```
df5['Idle Mean']=df5['Idle Mean']/1e15
```
Python

```
df5['Idle Max']=df5['Idle Max']/1e15
```
Python

```
df5['Idle Min']=df5['Idle Min']/1e15
```
Python

```
df5['Idle Std']=df5['Idle Std']/1e7
```
Python

```
df5.head(5)
```
Python

	Src IP	Src Port	Dst IP	Dst Port	Protocol	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label	L
0	10.152	57158	216.580	443	6	229	1	1	0	0	...	0	0	0	0	0.00000	0.000000	0.00000	0.00000	0	
1	10.152	57159	216.580	443	6	407	1	1	0	0	...	0	0	0	0	0.00000	0.000000	0.00000	0.00000	0	
2	10.152	57160	216.580	443	6	431	1	1	0	0	...	0	0	0	0	0.00000	0.000000	0.00000	0.00000	0	
3	10.152	49134	74.125	443	6	359	1	1	0	0	...	0	0	0	0	0.00000	0.000000	0.00000	0.00000	0	
4	10.152	34697	173.194	19305	6	10778451	591	400	64530	6659	...	0	0	0	0	1.43776	0.311772	1.43776	1.43776	0	

5 rows × 83 columns

Replacing null values with mean of column

```
df5.fillna(df5.mean()).head(5) #filling null values with the mean of the column
```
Python

	Src IP	Src Port	Dst IP	Dst Port	Protocol	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label	L
0	10.152	57158	216.580	443	6	229	1	1	0	0	...	0	0	0	0	0.00000	0.000000	0.00000	0.00000	0	
1	10.152	57159	216.580	443	6	407	1	1	0	0	...	0	0	0	0	0.00000	0.000000	0.00000	0.00000	0	
2	10.152	57160	216.580	443	6	431	1	1	0	0	...	0	0	0	0	0.00000	0.000000	0.00000	0.00000	0	
3	10.152	49134	74.125	443	6	359	1	1	0	0	...	0	0	0	0	0.00000	0.000000	0.00000	0.00000	0	
4	10.152	34697	173.194	19305	6	10778451	591	400	64530	6659	...	0	0	0	0	1.43776	0.311772	1.43776	1.43776	0	

5 rows × 83 columns

```
df5.notnull().values.all() #this shows there are no more null values
```
Python

```
False
```

```
# Replacing infinite with nan
df5.replace([np.inf, -np.inf], np.nan, inplace=True)

# Dropping all the rows with nan values
df5.dropna(inplace=True)
```
Python

```
df5.head(5)
```
Python

	Src IP	Src Port	Dst IP	Dst Port	Protocol	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Labe
0	10.152	57158.0	216.580	443.0	6.0	229.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.00000	0.000000	0.00000	0.00000	0.
1	10.152	57159.0	216.580	443.0	6.0	407.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.00000	0.000000	0.00000	0.00000	0.
2	10.152	57160.0	216.580	443.0	6.0	431.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.00000	0.000000	0.00000	0.00000	0.
3	10.152	49134.0	74.125	443.0	6.0	359.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.00000	0.000000	0.00000	0.00000	0.
4	10.152	34697.0	173.194	19305.0	6.0	10778451.0	591.0	400.0	64530.0	6659.0	...	0.0	0.0	0.0	0.0	1.43776	0.311772	1.43776	1.43776	0.

5 rows × 83 columns

Sampling:

Sampling is used to balance the data that is to balance the number of each class in the Label.

Oversampling: to increase the number of minority classes

Undersampling: to decrease the number of majority classes

```
df5['Label'].value_counts()
```

```
0.0    93309
1.0    23861
3.0    22919
2.0     1392
Name: Label, dtype: int64
```

So the data is highly imbalanced.....

Now I will undersample class 0.0 wrt class 1.0 and oversample class 3.0 wrt class 2.0

```
#1. Find the number of the minority class
non_tor = len(df5[df5['Label']==0])
non_vpn = len(df5[df5['Label']==1])
vpn = len(df5[df5['Label']==2])
tor = len(df5[df5['Label']==3])

print(non_tor)
print(non_vpn)
print(vpn)
print(tor)
```

```
93309
23861
1392
22919
```

```
[34]
index_non_tor = df5[df5['Label']==0].index
index_non_vpn = df5[df5['Label']==1].index
index_tor = df5[df5['Label']==2].index
index_vpn = df5[df5['Label']==3].index
```
Python

```
[35]
#4. Randomly sample the majority indices with respect to the number of minority classes
random_indices = np.random.choice(index_non_tor,non_vpn,replace='False')
```
Python

```
[36]
#5. Concat the minority indices with the indices from step 4
under_sample_indices = np.concatenate([index_non_vpn,random_indices])
```
Python

```python
#Get the balanced dataframe - This is the final undersampled data
under_sample_df = df5.iloc[under_sample_indices]
under_sample_df.head()
```

Python

	Src IP	Src Port	Dst IP	Dst Port	Protocol	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max
93403	131.202	64717.0	131.202	13000.0	6.0	81.0	2.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.00000	0.000000e+00	0.00000	0.(
93404	131.202	42530.0	178.237	443.0	6.0	119829241.0	5.0	3.0	24.0	0.0	...	0.0	0.0	0.0	0.0	1.22599	5.406120e+07	1.43033	2.
93405	131.202	42534.0	178.237	443.0	6.0	119828205.0	5.0	3.0	24.0	0.0	...	0.0	0.0	0.0	0.0	1.22599	5.406120e+07	1.43033	2.
93406	131.202	17208.0	77.720	11113.0	17.0	138272.0	2.0	2.0	126.0	85.0	...	0.0	0.0	0.0	0.0	1.43000	9.762882e-03	1.43033	1.4
93407	8.600	0.0	8.000	0.0	0.0	5103.0	2.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.00000	0.000000e+00	0.00000	0.(

5 rows × 83 columns

```python
under_sample_df['Label'].value_counts()
```

Python

```
0.0    23849
1.0    23824
2.0       49
Name: Label, dtype: int64
```

```python
#New dataframe containing only tor data
df_tor = df5[df5['Label']==3]
```

Python

Over sampled dataset:over_sample_df

```python
over_sample_df = pd.concat([vpn_sample,df_tor], axis=0)
```

Python

```python
over_sample_class_counts=pd.value_counts(over_sample_df['Label'])
```

Python

```python
over_sample_class_counts
```

Python

```
2.0    22919
3.0    22919
Name: Label, dtype: int64
```

Balanced dataset:

```
balance_df = pd.concat([under_sample_df,over_sample_df], axis=0)
```
Python

```
balance_df['Label'].value_counts()
```
Python

```
0.0    23849
1.0    23824
2.0    22968
3.0    22919
Name: Label, dtype: int64
```

```
balance_df.head()
```
Python

	Src IP	Src Port	Dst IP	Dst Port	Protocol	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	
93403	131.202	64717.0	131.202	13000.0	6.0	81.0	2.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.00000	0.000000e+00	0.00000	0.(
93404	131.202	42530.0	178.237	443.0	6.0	119829241.0	5.0	3.0	24.0	0.0	...	0.0	0.0	0.0	0.0	1.22599	5.406120e+07	1.43033	2.
93405	131.202	42534.0	178.237	443.0	6.0	119828205.0	5.0	3.0	24.0	0.0	...	0.0	0.0	0.0	0.0	1.22599	5.406120e+07	1.43033	2.
93406	131.202	17208.0	77.720	11113.0	17.0	138272.0	2.0	2.0	126.0	85.0	...	0.0	0.0	0.0	0.0	1.43000	9.762882e-03	1.43033	1.(
93407	8.600	0.0	8.000	0.0	0.0	5103.0	2.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.00000	0.000000e+00	0.00000	0.(

Normalization:

Normalization is a scaling technique, applied during the data preparation phase to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is needed only when features of machine learning models have variable ranges.

All the values of the dataset are then normalized to a given range which makes the further computation easier. This is necessary for the consistency of the values of the dataset. I will be using MinMaxScaler for normalization. So all the values will be converted into values between 0 to 1.

$$x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Fig 4.1 Min-Max Scaler Formula

```python
#MinMaxScaling

import pandas as pd
from sklearn import preprocessing

x = balance_df.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df_final = pd.DataFrame(x_scaled,columns = balance_df.columns)
df_final.head()
```

Python

	Src IP	Src Port	Dst IP	Dst Port	Protocol	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	...	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Active Min	Idle Mean	
0	0.587746	0.987533	0.512169	0.198367	0.352941	6.666667e-07	0.000004	0.000000	0.000000e+00	0.000000e+00	...	0.454545	0.0	0.0	0.0	0.0	0.000000	0
1	0.587746	0.648976	0.697131	0.006760	0.352941	9.985770e-01	0.000017	0.000006	3.119689e-08	0.000000e+00	...	0.454545	0.0	0.0	0.0	0.0	0.839719	5
2	0.587746	0.649037	0.697131	0.006760	0.352941	9.985684e-01	0.000017	0.000006	3.119689e-08	0.000000e+00	...	0.454545	0.0	0.0	0.0	0.0	0.839719	5
3	0.587746	0.262581	0.301854	0.169574	1.000000	1.152258e-03	0.000004	0.000004	1.637837e-07	1.267845e-07	...	0.181818	0.0	0.0	0.0	0.0	0.979452	9
4	0.038525	0.000000	0.027684	0.000000	0.000000	4.251667e-05	0.000004	0.000000	0.000000e+00	0.000000e+00	...	0.000000	0.0	0.0	0.0	0.0	0.000000	0

5 rows × 82 columns

Feature Selection

Getting the correlation matrix:

```python
corr1 = df_final.corr()
corr1.head()
```

Python

	Src IP	Src Port	Dst IP	Dst Port	Protocol	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	...	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Active Min	Idl Mea
Src IP	1.000000	-0.305289	-0.101929	0.255228	0.057233	0.069783	0.206548	0.105654	0.178355	0.028758	...	0.010916	NaN	NaN	NaN	NaN	-0.07581
Src Port	-0.305289	1.000000	0.427949	-0.374122	0.072011	-0.116201	-0.212785	-0.122040	-0.183028	-0.029426	...	0.057141	NaN	NaN	NaN	NaN	-0.06280
Dst IP	-0.101929	0.427949	1.000000	-0.430562	0.268788	-0.051667	-0.123019	-0.036536	-0.108736	0.011276	...	-0.136733	NaN	NaN	NaN	NaN	-0.15993
Dst Port	0.255228	-0.374122	-0.430562	1.000000	-0.255386	0.108729	0.169103	0.088196	0.138648	0.024325	...	0.187956	NaN	NaN	NaN	NaN	0.16093
Protocol	0.057233	0.072011	0.268788	-0.255386	1.000000	-0.460125	-0.185954	-0.167998	-0.150598	-0.085103	...	-0.795992	NaN	NaN	NaN	NaN	-0.52083

5 rows × 82 columns

As we can see for many pairs of columns, correlation does not even exist as an integer. Also for various columns, the correlation is as high as 1.000. This means that the columns have almost similar effect on the performance and can be treated as one. So we can keep one of them and remove the other to decrease the no. of unnecessary columns and improve the computation speed and performance.

Here the Pearson correlation coefficient is used and 0.9 has been considered as the threshold correlation.

```python
columns = np.full((corr1.shape[0],), True, dtype=bool)
for i in range(corr1.shape[0]):
    for j in range(i+1, corr1.shape[0]):
        if corr1.iloc[i,j] >= 0.9:
            if columns[j]:
                columns[j] = False
selected_columns = df_final.columns[columns]
dataset_final = df_final[selected_columns]
```

Python

```python
dataset_final.shape
```

Python

```
(93560, 61)
```

Finally only 61 out of 85 columns are left.

We can proceed further with this dataset.

4.3 Model Building Phase:

This phase includes:

- Splitting into training and testing data
- Defining the model: GBM, LightGBM, Decision Tree, Random Forest, Knn
- Performing baseline tuning on test dataset for each model: Getting evaluation metrics
- Building an ensemble model combining Decision Tree , Random Forest and Knn using hard and soft voting techniques and performing baseline tuning: Getting evaluation metrics
- Performing 10 fold cross validation on the model with best performance in the baseline tuning
- Performing Hyperparameter Tuning using GridSearchCV: Getting the best parameters
- Comparing the models before and after tuning and determining the best possible score.

1. Train Test Split

```python
#train_test_split
from sklearn.model_selection import train_test_split

y=target
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 0)
```

Python

2. Model Building and Testing:

- Building GBM Model and performing baseline tuning using different learning rates on test dataset

```python
from sklearn.ensemble import GradientBoostingClassifier  #GBM algorithm
from sklearn.metrics import f1_score

lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gbc = GradientBoostingClassifier(n_estimators=20, learning_rate=learning_rate, max_features=2, max_depth=2, random_state=0)
    gbc=gbc.fit(x_train, y_train.values.ravel())
    y_pred_gbc=gbc.predict(x_test)
    f1_gbc = f1_score(y_test, y_pred_gbc,average='micro')

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gbc.score(x_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gbc.score(x_test, y_test)))
    print('F1-score: {}'.format(np.round(f1_gbc,4)))
```

[54] Python

- Building LightGBM Model and performing baseline tuning using different learning rates on test dataset

```python
from lightgbm import LGBMClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report

# making predictions with the Light Gradient Boosting classifier
lr_list = [0.01, 0.75, 1]

for learning_rate in lr_list:
    lgbm = LGBMClassifier(boosting_type='dart',max_depth=3, learning_rate=learning_rate, n_estimators=200,random_state=45)
    lgbm = lgbm.fit(x_train, y_train.values.ravel())
    y_pred_lgbm = lgbm.predict(x_test)

    f1_lgbm = f1_score(y_test, y_pred_lgbm,average='micro')

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(lgbm.score(x_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(lgbm.score(x_test, y_test)))
    print('F1-score: {}'.format(np.round(f1_lgbm,4)))
```

[55] Python

```
#Get the confusion matrix
from sklearn.metrics import confusion_matrix

cf_matrix = confusion_matrix(y_test,y_pred_knn)
print(cf_matrix)
```

[60]

```
[[6608  224  138  234]
 [ 365 6057  185  489]
 [   0    0 7040    0]
 [ 466  562  137 5563]]
```

```
import seaborn as sns
sns.heatmap(cf_matrix, annot=True, fmt="d", linewidths=.5)
```

[61]

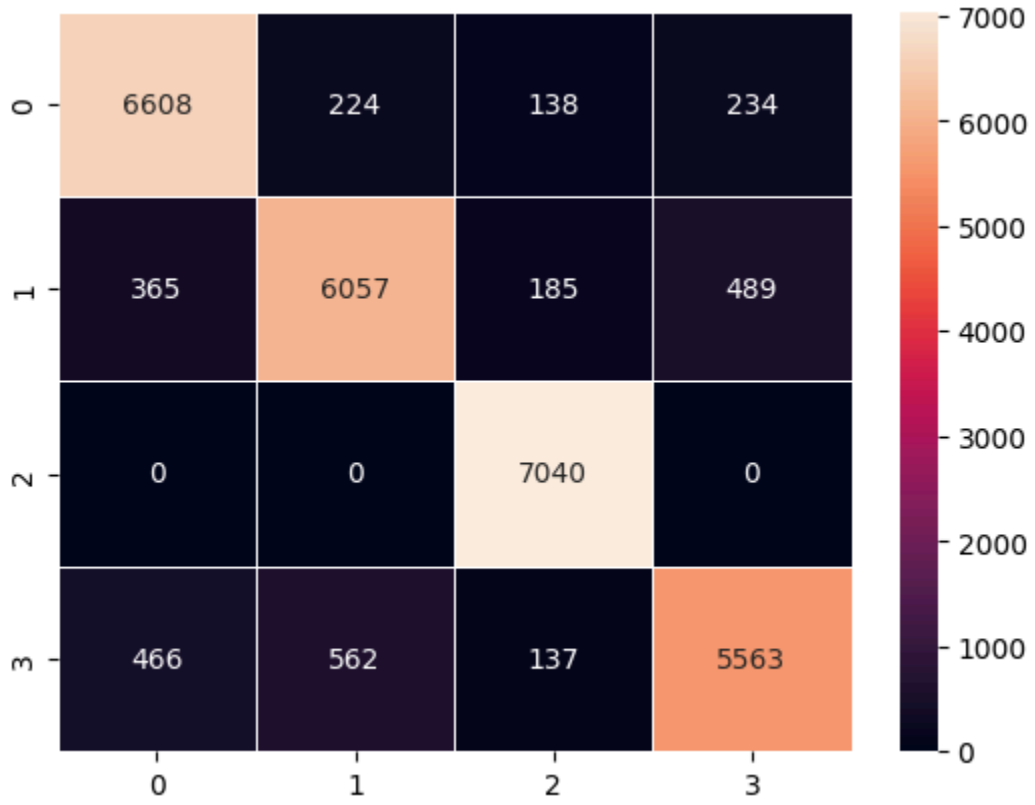


Fig 4.2 Confusion Matrix of K-NN

- Building Decision Tree Classifier, Random Forest Classsifer and KNN Classsifier and performing baseline tuning on test dataset

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report

# making predictions with the decision tree classifier
dtc = DecisionTreeClassifier(random_state=42,max_depth=5)
dtc = dtc.fit(x_train, y_train.values.ravel())
y_pred_dtc = dtc.predict(x_test)
print('Accuracy of the DTC on test set: {:.3f}'.format(dtc.score(x_test, y_test)))
print(classification_report(y_test, y_pred_dtc))


# making predictions with the k nearest neighbour model
knn = KNeighborsClassifier(n_neighbors = 5, metric = 'euclidean', p = 2)
knn= knn.fit(x_train, y_train.values.ravel())
y_pred_knn = knn.predict(x_test)
print('Accuracy of the KNN on test set: {:.3f}'.format(knn.score(x_test, y_test)))
print(classification_report(y_test, y_pred_knn))


# making predictions with the random forest model
rfc = RandomForestClassifier(n_estimators=200,max_depth=5)
rfc = rfc.fit(x_train, y_train.values.ravel())
y_pred_rfc = rfc.predict(x_test)
print('Accuracy of the Random Forest on test set: {:.3f}'.format(rfc.score(x_test, y_test)))
print(classification_report(y_test, y_pred_rfc))


# evaluating the models based on their f-1 scores
f1_dtc = f1_score(y_test, y_pred_dtc,average='micro')
f1_knn = f1_score(y_test, y_pred_knn,average='micro')
f1_rfc = f1_score(y_test, y_pred_rfc,average='micro')
```

3. Building Ensemble Model using Voting Technique:

A voting ensemble is an ensemble machine learning model that unites the forecasts from multiple other models.

It is a technique that may be used to improve model performance, perfectly achieving better performance than any single model used in the ensemble.

A voting ensemble works by joining the forecasts from multiple models. It can be used for classification or regression. In the case of classification, the likelihoods for each label are tallied and the label with the majority vote is projected.

There are two tactics to the majority vote prediction for classification; they are hard voting and soft voting.

Hard voting involves tallying the predictions for each class label and projecting the class label with the most votes. Soft voting involves tallying the predicted likelihoods (or probability-like scores) for each class label and predicting the class label with the largest chance.

- **Hard Voting**. Predict the class with the largest sum of votes from models

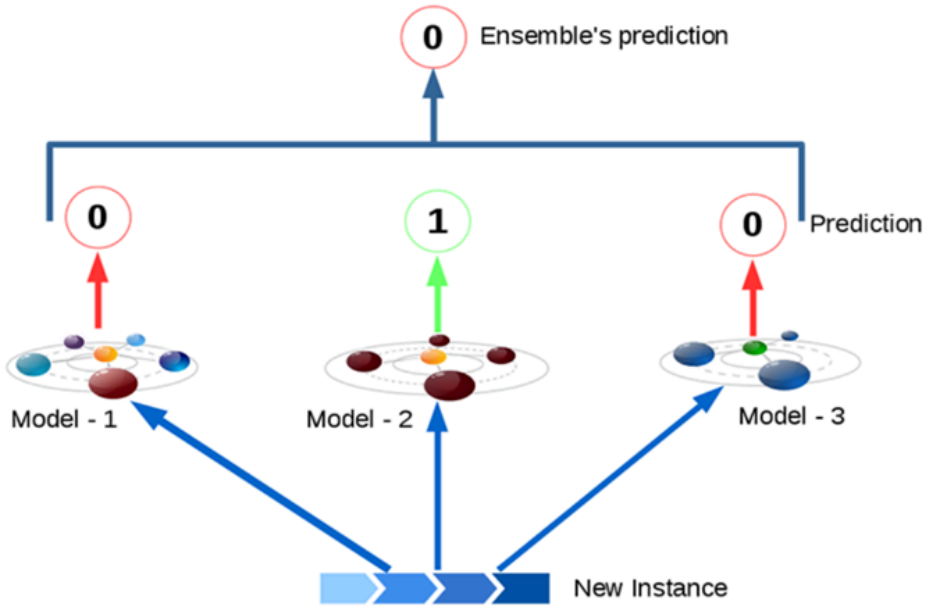


Fig 4.3. Hard Voting

Soft Voting. Predict the class with the largest summed probability from models.

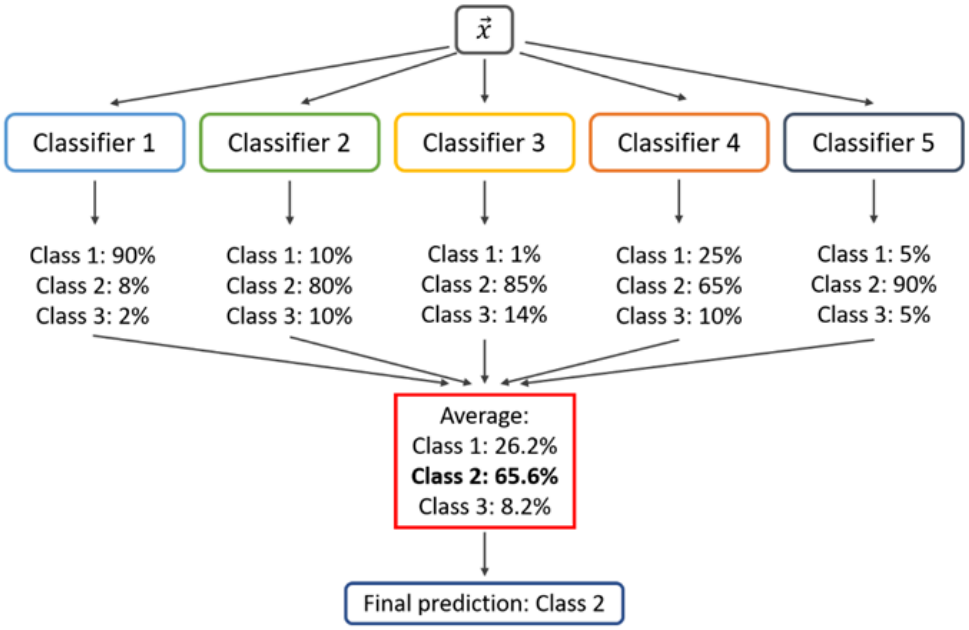


Fig 4.4. Soft Voting

```python
# import voting classifier
from sklearn.ensemble import VotingClassifier

# create a voting classifier with hard voting
voting_classifier_hard = VotingClassifier(
    estimators = [('dtc',DecisionTreeClassifier(random_state=42,max_depth=8)),
                  ('knn', KNeighborsClassifier(n_neighbors = 5, metric = 'euclidean', p = 2)),
                  ('rfc', RandomForestClassifier(n_estimators=200,max_depth=8))],

    voting='hard')

# create a voting classifier with soft voting
voting_classifier_soft = VotingClassifier(
    estimators = [('dtc',DecisionTreeClassifier(random_state=42,max_depth=8)),
                  ('knn', KNeighborsClassifier(n_neighbors = 5, metric = 'euclidean', p = 2)),
                  ('rfc',RandomForestClassifier(n_estimators=200,max_depth=8))],
    voting='soft')

# make predictions with the hard voting model
voting_classifier_hard.fit(x_train, y_train.values.ravel())
y_pred_vch = voting_classifier_hard.predict(x_test)

# make predictions with the soft voting model
voting_classifier_soft.fit(x_train, y_train.values.ravel())
y_pred_vcs = voting_classifier_soft.predict(x_test)

# evaluate both models with the f-1 score
f1_vch = f1_score(y_test, y_pred_vch,average='micro')
f1_vcs = f1_score(y_test, y_pred_vcs,average='micro')

# print the f-1 scores
print('F1-score of the hard voting classifier: {}'.format(np.round(f1_vch,4)))
print('F1-score of the soft voting classifier: {}'.format(np.round(f1_vcs,4)))
```

[62] Python

4. Performing 10 Fold Cross-Validation on the hard and soft voting classifiers:

```python
#Cross validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

cv = KFold(n_splits=10, random_state=1, shuffle=True)

VCH_accuracies = cross_val_score(estimator = voting_classifier_hard, X = x_train, y = y_train.values.ravel(), cv = cv)
print("Mean_Acc_VotingHard : ", VCH_accuracies.mean())

VCS_accuracies = cross_val_score(estimator = voting_classifier_soft, X = x_train, y = y_train.values.ravel(), cv = cv)
print("Mean_Acc_VotingSoft : ", VCS_accuracies.mean())
```

[65] Python

5. Hyperparameter Tuning using GridSearchCV on the hard and soft voting classifiers.

```python
#HyperParameter Tuning: Learning rate,n_estimators_max_depth (using 5 fold Grid search CV)
from sklearn.model_selection import GridSearchCV


params= {'knn__n_neighbors':[5,10,8], 'rfc__n_estimators':[100,200,300],'rfc__max_depth':[5,8,10]}

tuning = GridSearchCV(estimator=voting_classifier_hard, param_grid=params, cv=5,verbose=3)
tuning.fit(x_train,y_train.values.ravel())

print('The best Parameters are:',tuning.best_params_)
print('The best score is:',tuning.best_score_)
```

[66] Python

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
[CV 1/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=100;, score=0.969 total time=  10.5s
[CV 2/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=100;, score=0.964 total time=  10.0s
[CV 3/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=100;, score=0.968 total time=  10.2s
[CV 4/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=100;, score=0.973 total time=  10.1s
[CV 5/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=100;, score=0.973 total time=   9.9s
[CV 1/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=200;, score=0.968 total time=  17.4s
[CV 2/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=200;, score=0.969 total time=  17.2s
[CV 3/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=200;, score=0.970 total time=  17.3s
[CV 4/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=200;, score=0.972 total time=  17.8s
[CV 5/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=200;, score=0.975 total time=  17.8s
[CV 1/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=300;, score=0.971 total time=  25.2s
[CV 2/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=300;, score=0.966 total time=  25.1s
[CV 3/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=300;, score=0.968 total time=  24.8s
[CV 4/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=300;, score=0.974 total time=  24.4s
[CV 5/5] END knn__n_neighbors=5, rfc__max_depth=5, rfc__n_estimators=300;, score=0.975 total time=  25.6s
[CV 1/5] END knn__n_neighbors=5, rfc__max_depth=8, rfc__n_estimators=100;, score=0.992 total time=  13.1s
[CV 2/5] END knn__n_neighbors=5, rfc__max_depth=8, rfc__n_estimators=100;, score=0.988 total time=  12.2s
[CV 3/5] END knn__n_neighbors=5, rfc__max_depth=8, rfc__n_estimators=100;, score=0.989 total time=  12.1s
[CV 4/5] END knn__n_neighbors=5, rfc__max_depth=8, rfc__n_estimators=100;, score=0.991 total time=  12.2s
[CV 5/5] END knn__n_neighbors=5, rfc__max_depth=8, rfc__n_estimators=100;, score=0.991 total time=  12.3s
[CV 1/5] END knn__n_neighbors=5, rfc__max_depth=8, rfc__n_estimators=200;, score=0.993 total time=  22.1s
[CV 2/5] END knn__n_neighbors=5, rfc__max_depth=8, rfc__n_estimators=200;, score=0.989 total time=  22.1s
[CV 3/5] END knn__n_neighbors=5, rfc__max_depth=8, rfc__n_estimators=200;, score=0.989 total time=  22.4s
[CV 4/5] END knn__n_neighbors=5, rfc__max_depth=8, rfc__n_estimators=200;, score=0.991 total time=  23.0s
...
```

[67]

```python
#HyperParameter Tuning:n_neighbors,n_estimators,_max_depth (using 5 fold Grid search CV)
from sklearn.model_selection import GridSearchCV


params= {'knn__n_neighbors':[5,8,10], 'rfc__n_estimators':[100,200,300],'rfc__max_depth':[5,8,10]}

tuning = GridSearchCV(estimator=voting_classifier_soft, param_grid=params, cv=5,verbose=0)
tuning.fit(x_train,y_train.values.ravel())

print('The best Parameters are:',tuning.best_params_)
print('The best score is:',tuning.best_score_)
```

Python

Chapter 5 Result Analysis

5.1. PERFORMANCE METRICS/OUTPUTS

- **Evaluation Metrics of Gradient Boosting Classifier:**

```
Learning rate:  0.05
Accuracy score (training): 0.774
Accuracy score (validation): 0.773
F1-score: 0.7732
Learning rate:  0.075
Accuracy score (training): 0.789
Accuracy score (validation): 0.788
F1-score: 0.7876
Learning rate:  0.1
Accuracy score (training): 0.809
Accuracy score (validation): 0.808
F1-score: 0.8082
Learning rate:  0.25
Accuracy score (training): 0.878
Accuracy score (validation): 0.877
F1-score: 0.877
Learning rate:  0.5
Accuracy score (training): 0.917
Accuracy score (validation): 0.917
F1-score: 0.9175
Learning rate:  0.75
Accuracy score (training): 0.944
Accuracy score (validation): 0.944
F1-score: 0.9436
Learning rate:  1
Accuracy score (training): 0.956
Accuracy score (validation): 0.956
F1-score: 0.9558
```

As we can see, GBM gives the best results when the learning rate is 1. Training accuracy is 95.6% and validation accuracy is 95.6%

- **Evaluation Metrics of LightGBM:**

```
Learning rate:  0.01
Accuracy score (training): 0.947
Accuracy score (validation): 0.948
F1-score: 0.9481
Learning rate:  0.75
Accuracy score (training): 0.392
Accuracy score (validation): 0.393
F1-score: 0.3932
Learning rate:  1
Accuracy score (training): 0.074
Accuracy score (validation): 0.077
F1-score: 0.0767
```

As we can see, LightGBM gives the best results when the learning rate is 0.01. Training accuracy is 94.7% and validation accuracy is 94.8%

- **Evaluation Metrics of Decision Tree Classifier**

```
Accuracy of the DTC on test set: 0.959
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      7204
         1.0       0.88      0.98      0.93      7096
         2.0       0.98      0.87      0.92      7040
         3.0       1.00      0.99      1.00      6728

    accuracy                           0.96     28068
   macro avg       0.96      0.96      0.96     28068
weighted avg       0.96      0.96      0.96     28068
```

- **Evaluation Metrics of K-Nearest Neighbour Classifier**

```
Accuracy of the KNN on test set: 0.900
              precision    recall  f1-score   support

         0.0       0.89      0.92      0.90      7204
         1.0       0.89      0.85      0.87      7096
         2.0       0.94      1.00      0.97      7040
         3.0       0.88      0.83      0.85      6728

    accuracy                           0.90     28068
   macro avg       0.90      0.90      0.90     28068
weighted avg       0.90      0.90      0.90     28068
```

- **Evaluation Metrics of Random Forest Classifier:**

```
Accuracy of the Random Forest on test set: 0.923
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.96      7204
         1.0       0.85      0.93      0.89      7096
         2.0       1.00      0.84      0.91      7040
         3.0       0.93      0.93      0.93      6728

    accuracy                           0.92     28068
   macro avg       0.93      0.92      0.92     28068
weighted avg       0.93      0.92      0.92     28068
```

- **F1 Scores:**

```
F1-score of decision tree classifier: 0.9594
F1-score of k nearest neighbour: 0.9002
F1-score of random forest classifier: 0.9228
```

- **Evaluation metrics of Voting Classifier built by ensembling DTC, Random Forest and Knn:**

```
F1-score of the hard voting classifier: 0.99
F1-score of the soft voting classifier: 0.9938
```

As we can see the performance of both hard and soft voting classifiers is better than the individual performance of the three models ensembled above as well as the pre existing ensemble models like GBM and LightGBM.

Hence the new ensemble model built is a successful one and has a great performance.

- **10 Fold Cross Validation**

This shows the performance of the classifier on the unseen data.

```
Mean_Acc_VotingHard :  0.9910370652707308
Mean_Acc_VotingSoft :  0.9941824834279227
```

- **Hyperparameter Tuning:**

For Hard Voting Classifier, the parameters tuned are 'knn__n_neighbors', 'rfc__max_depth', 'rfc__n_estimators'

```
The best Parameters are: {'knn__n_neighbors': 5, 'rfc__max_depth': 10, 'rfc__n_estimators': 200}
The best score is: 0.9948390629071391
```

For Soft Voting Classifier, the parameters tuned are 'knn__n_neighbors', 'rfc__max_depth', 'rfc__n_estimators'

```
The best Parameters are: {'knn__n_neighbors': 5, 'rfc__max_depth': 10, 'rfc__n_estimators': 100}
The best score is: 0.9962438213955667
```

5.2 Final Comparison:

Model	Gradient Boosting	Light GBM	Decision tree	Random Forest	k-NN	Voting (Hard)	Voting (Soft)
Accuracy	95.6	94.8	95.9	92	90	99.48	99.62
F1 Score	95.58	94.81	95.91	90.03	92	99.48	99.62

Table 2: Performance Comparison (Table)

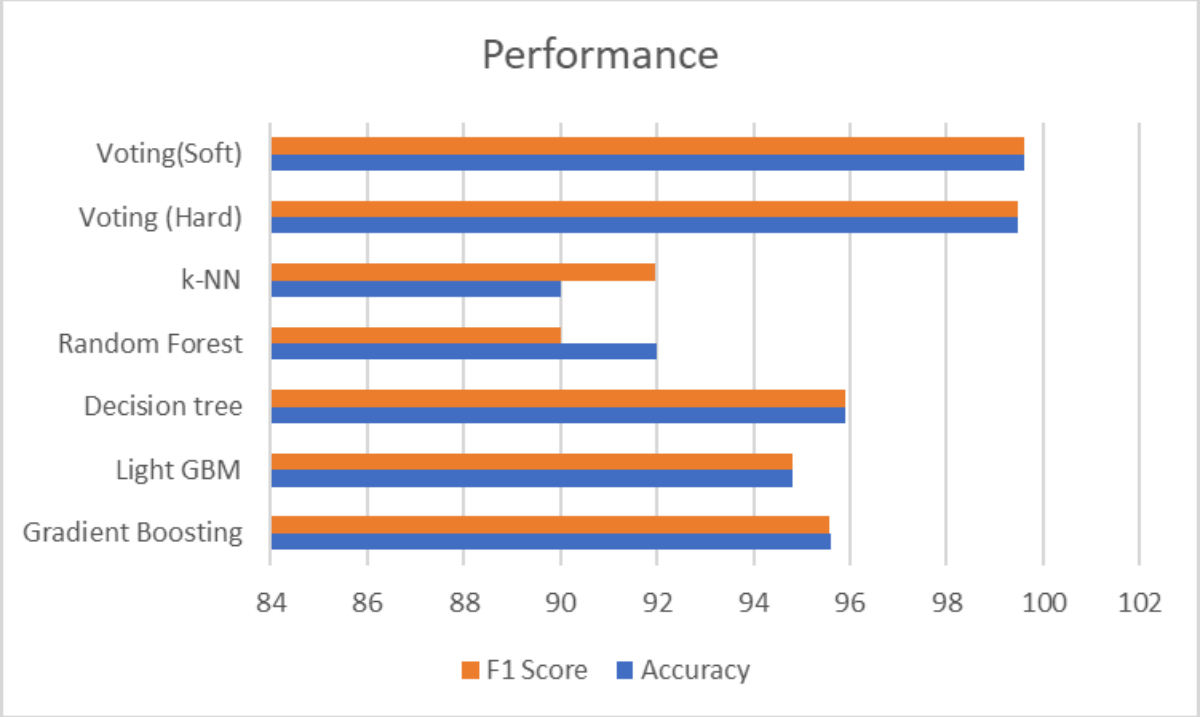


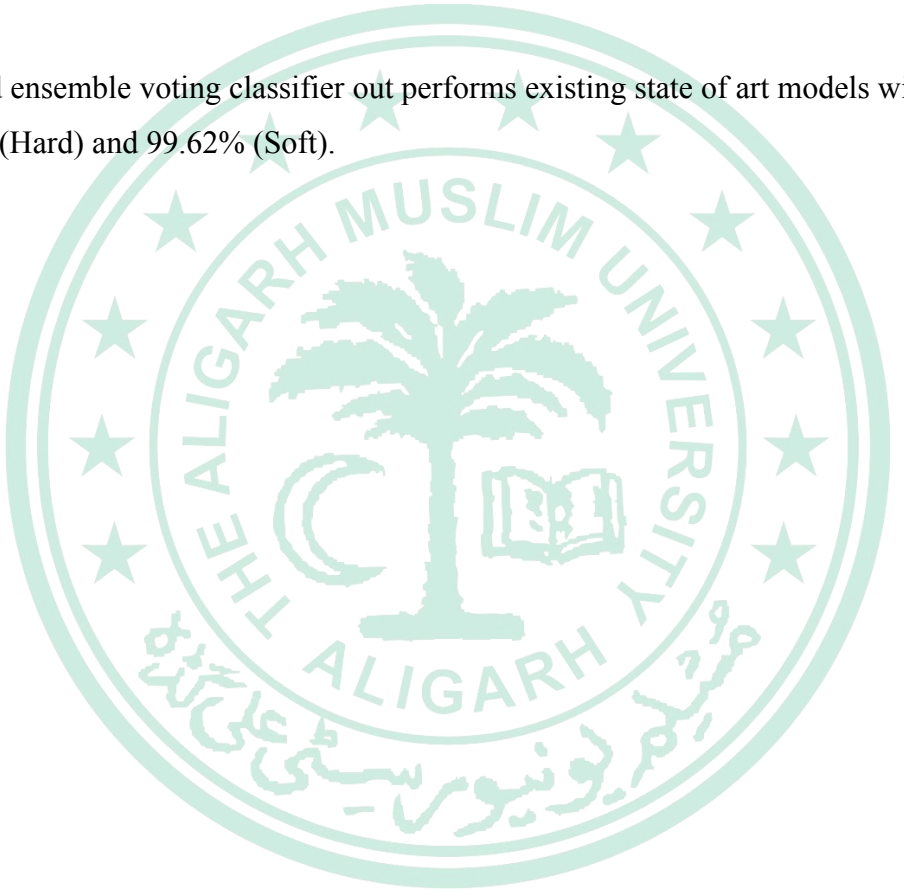
Table 3: Performance Comparison (Chart)

Existing Techniques:

Year	Evaluation Model	Accuracy	I.F. %
2021	Recurring Neural Network (RNN)	94.51%	5.28%↑
2017	Longitudinal Analysis of Network Traffic (LANT)	94.00%	5.85%↑
2020	Hierarchical Classification Method (HCM)	96.60%	3.00%↑
2021	AdaBoost Decision Trees (AB-DT)	97.30%	2.26%↑
2020	Convolutional Neural Network (CNN)	86.00%	15.70%↑
2020	Artificial Neural Network and Apache Spark (ANN-AS)	94.66%	5.11%↑
2021	Convolutional Neural Network (CNN) and K-Means (KM)	97.40%	2.16%↑
2020	Sparse Structure Learning with LASSO selection (SSL)	97.10%	2.47%↑
2021	Convolutional Neural Network (CNN)	97.65%	1.89%↑
2019	Random Forest Classifier (RFC)	78.30%	27.08%↑
2017	Logistic Regression Classifier (LRC)	96.60%	3.00%↑

Fig 5.1: Comparison with Existing State of Art Models

Clearly the proposed ensemble voting classifier out performs existing state of art models with accuracy of 99.48% (Hard) and 99.62% (Soft).



Chapter 6 Conclusion and Future Work

An efficient Darknet traffic detection system (DTDS) has been proposed, modeled, implemented, assessed in this project.. The developed DTDS ML models were evaluated on CIC-Darknet-2020 dataset involving a large number of captured cyber-attacks and hidden services provided by the Darknet grouped into four classes (VPN, TOR, Non-VPN, Non-TOR). This work shows that the DTDS-based Ensemble model built using voting technique by ensembling classifiers like DTC, Random Forest and Knn is superior among these three models the other evaluated models, scoring 99.48% (Hard voting) and 99.62% (Soft Voting) in classification accuracy.

The proposed model can be efficiently deployed in firewalls and other intrusion detection devices to detect Tor and VPN activities in communication networks.

Chapter 7 References

1. Arash Habibi Lashkari, Gurdip Kaur, and Abir Rahali, “DIDarknet: A Contemporary Approach to Detect and Characterize the Darknet Traffic using Deep Image Learning”, 10th International Conference on Communication and Network Security, Tokyo, Japan, November 2020.
2. Coutinho Marim, Mateus & Ramos, Paulo & Oliveira, Roberto & Borges, Alex & Silva, Edelberto. (2021). Caracterização e Classificação do Tráfego da Darknet com Modelos Baseados em Árvores de Decisão. 10.5753/sbrc.2021.16716.
3. Kaur, Gurdip & Habibi Lashkari, Arash & Rahali, Abir. (2020). DIDarknet: A Contemporary Approach to Detect and Characterize the Darknet Traffic using Deep Image Learning. 10.1145/3442520.3442521
4. Bai, X., Zhang, Y., and Niu, X. (2008). Traffic identification of tor and web-mix. In 2008 Eighth International Conference on Intelligent Systems Design and Applications, volume 1, pages 548–551
5. AlSabah, M., Bauer, K., and Goldberg, I. (2012). Enhancing tor’s performance using real-time traffic classification. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS ’12,pages 73–84, New York, NY, USA. ACM.
6. KHAFAJEH, H. (2020). AN EFFICIENT INTRUSION DETECTION APPROACH USING LIGHT GRADIENT BOOSTING. Journal of Theoretical and Applied Information Technology, 98(05).
7. Tama, B. A., & Rhee, K. H. (2019). An in-depth experimental study of anomaly detection using gradient boosted machines. Neural Computing and Applications, 31(4), 955-965.
8. Muhammad Sarwar, Faisal Mehmood, Muhammad Abid, Abdul Qayyum Khan, Sufi Tabassum Gul, Adil Sarwar Khan “High impedance fault detection and isolation in power distribution networks using support vector machines” (2019)

9. Abu Al-Haija, Q., Krichen, M., & Abu Elhaija, W. (2022). Machine-learning-based traffic detection system for IoT applications. *Electronics*, *11*(4), 556.

10. Jie Gu , Lihong Wang, Huiwen Wang , Shanshan Wang “A novel approach to intrusion detection using SVM ensemble with feature augmentation” (2019) DOI: 10.1016/j.cose.2019.05.022

11. Gautam M. Borkar, Leena H. Patil, Dilip Dalgade, Ankush Hutke “A novel clustering approach and adaptive SVM classifier for intrusion detection in WSN: A data mining concept” (2019) DOI: 10.1016/j.suscom.2019.06.002

12. Weijian Fanga , Xiaoling Tan , Dominic Wilbur “Application of intrusion detection technology in network safety based on machine. learning” (2020) DOI: 10.1016/j.ssci.2020.104604