

Configuration Manual

Msc Research Project

Msc CyberSecurity

Viknesh Aditya Rajendran

Student ID: 19216343

School Of Computing

National College of Ireland

Supervisor : Prof: Imran Khan

Introduction:

The proposal of this research work is “Ensemble techniques to enhance wireless intrusion detection system in IoT”. This research work was implemented by utilizing many software and hardware requirements. The primary of the goal of this research is to develop an Ensemble based Intrusion Detection System to detect the IoT botnets. However, this model categorises the types of botnets and determines if a packet is malicious or benign based on the md5 has values. In order to achieve this model, two ensemble models were combined, which is LightGbm and CatBoost via single voting classifier. The dataset utilized in this research was IoT-BDA dataset. The dataset contains the findings from IoT-BDA Framework's analysis of 4069 distinct IoT botnet samples that were collected using honeypots.

1. System Requirements

Hardware and software are needed to process the model quickly and efficiently.

1.1. Hardware Requirements

The implementation was performed on an DELL vostro laptop, the configuration of the device is as follows

1. Processor- 11th Gen Intel(R) Core(TM) i5-1135G7
2. RAM - 8 GB
3. Hard Disk – 1 TB PM981 NVMe SSD
4. System Type x64-based PC
5. OS – Windows 11 64 – bit

1.2. Software Requirements

The software requirements are listed below.

Software	Version
Python	3.8.3
Anaconda Navigator	2.1.1
Jupyter Notebook	6.4.11
Numpy	1.21.0
Scikit-learn	0.23.0
Google colab	3.6
Pandas	1.4.3
LightGbm classifier	3.3.2.99
CatBoost Classifier	1.0.6
Label Encoder	0.12

1.3 Acquiring the results for the md5 hashes:

The dataset doesn't consist of the records for the final columns to perform the testing part. Records for the final columns were generated by combining two engines, such as avast and kaspersky, in the virus total checker, and based on md5 hashes, labels for the types of botnets and whether they

are malicious or benign were obtained. Finally, the output was labelled in the.csv file using the keywords "label" for benign and malicious and "final label" for botnet types such. Then, the preprocessing part has been started.

Malicious	0d4a95b1	ELF:Mirai	HEUR:Backdoor.Linux.Mirai.au
Malicious	4d5d979e	ELF:CVE-2	HEUR:Backdoor.Linux.Mirai.b
Malicious	2d0c79c6	ELF:Mirai	HEUR:Backdoor.Linux.Mirai.b
Malicious	5ee8149f	ELF:Mirai	HEUR:Backdoor.Linux.Mirai.b
Malicious	8dde128bdc260972		HEUR:Backdoor.Linux.Mirai.b
Malicious	7c187aa6	ELF:Mirai	HEUR:Backdoor.Linux.Mirai.b
Malicious	6f42d38c2	ELF:Mirai	HEUR:Backdoor.Linux.Mirai.ba
Malicious	6ec274d7	ELF:Mirai	HEUR:Backdoor.Linux.Gafgyt.a
Malicious	a11305ff0	ELF:Mirai	HEUR:Backdoor.Linux.Mirai.a
Malicious	68438f365	ELF:Mirai	HEUR:Backdoor.Linux.Mirai.ba
Malicious	5b3881e5	ELF:Mirai	HEUR:Backdoor.Linux.Mirai.b
Malicious	73c7f1e8c	ELF:Mirai	HEUR:Backdoor.Linux.Mirai.ba
Malicious	b7d90fd2	ELF:Mirai	HEUR:Backdoor.Linux.Mirai.cn
Malicious	919b7b2e	ELF:Svirtu	HEUR:Backdoor.Linux.Mirai.b

2. Data pre-processing

This part represents all steps required for preparing data for the machine learning model.

2.1 Importing Libraries:

The libraries shown in the image below were all used in this research project.

```
import numpy as np
# importing libraries numpy for numerical caluclation
import pandas as pd
from google.colab import drive
# pandas to read the data and for applying preporcessing
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.ensemble import VotingClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
```

Step1: The very first was to import the pandas library. Pandas has a faster built-in function and preprocessing are performed faster when compared to other packages

Step2: From the google colab application, google drive has been mounted, Where the .csv files are stored.

Step 3: Dataset has been imported by using pandas library from google drive. Pandas library utilizes its function read_csv in order to pass the path or directory of the dataset which has to be loaded

2.2 Data Loading

```
df = pd.read_csv('content/drive/MyDrive/criptodata/data2/cleaned_analysis.csv')
df.head()
```

	idanalysis	url	filename	botnet	md5	architecture	task	honeypot	tracking	strings	...
0	1	http://87.246.6.100.80/bins/yakuza_arm7	f44bb259d0f630e19db246ba2ac88a2d8d71	unknown	0d4e9b1bee76c488c6c72c457a6baa	arm	ad9fa223e514-40ab-8492-bc9348c65e5f	telnet	()	"home/handle/aboriginal/aboriginal/build/tem...	...
1	3	http://185.112.249.11.80/nope/daddyscum.x86	daddyscum.x86	185.112.249.11/nope/daddyscum	4d5d979edb12d241e5108bd719575269	x86	e151813b-09724863-889c-a27657e458cf	telnet	()	"<?xml version='1.0' ?><:Envelope xmlns='ht...	..."130.255
2	4	http://142.93.15.164.80/bins/uzaveB.x86	uzaveB.x86	142.93.15.164/bins/uzaveB	2d0c79c617b1a196d83e128e5f4215af	x86	bc8633aa-9642-4922-9330-ab6e5c49a29	telnet	"{telnet: 1}"	NaN	...
3	5	http://188.209.49.44/b/arm6	arm6-20191121-1833	188.209.49.44/b/	5ee8149f1c0a570120363150cc3bec52	arm6	97843a75-1271-467f-96cb-3eaaa000a94	dropper	"{dropper: 1}"	NaN	..."18.4
4	6	http://31.214.157.113.80/bins/orphic_arm6	orphic_arm6	31.214.157.113/bins/orphic	8dde128bdc2609721029b2302f2874f5	arm6	6e12265-8c3d-4648-699f-c6ac1e03116	telnet	"{telnet: 1}"	NaN	..."18.4

After importing, we not first connect our Google Collaboratory environment with our Google Drive. Because our dataset is stored inside our drive. In the next cell, we import our dataset using the library **Pandas** and its using its function **read_csv** where we pass the path or directory of our dataset which we want to load.

Step4: `df.columns()` are utilized to view the entire column of the loaded dataset and `df.shape` are utilized to view the number of rows and columns

2.3 Data Classification

```
data = df[['url', 'botnet', 'md5', 'architecture', 'honeypot', 'tracking', 'tcp', 'udp', 'endpoints', 'antisandbox', 'cnc', 'scanning', 'ddos', 'dns', 'http', 'tor', 'spooF', 'antidebugging', 'antiexecution', 'persistence', 'info_gathering', 'stealth', 'kernel_modules', 'process_injection', 'linking', 'encoded', 'firewall', 'entropy']]
data
```

	md5	architecture	honeypot	tracking	tcp	udp	endpoints	antisandbox	...	antiexecution
0	0d4e9b1bee76c488c6c72c457a6baa	arm	telnet	{}	"{23: 31866}"	"2323: 3541"	"5034: 11"	{}	...	"{tcp: {}"
1	4d5d979edb12d241e5108bd719575269	x86	telnet	{}	"{23: 1065}"	"7098: 11"	"37215: 695"	{}	...	"data_in": 200
2	2d0c79c617b1a196d83e128e5f4215af	x86	telnet	"{telnet: 1}"	"{23: 42850}"	"26663: 27"	{}	"{ip: 142.93.15.164}"	...	"data_in": 1761
3	5ee8149f1c0a570120363150cc3bec52	arm6	dropper	"{dropper: 1}"	{}	{}	{}	set()	...	"md5": "5ee8149f1c0a570120363150cc3bec52" "232d8bc20
4	8dde128bdc2609721029b2302f2874f5	arm6	telnet	"{telnet: 1}"	{}	{}	{}	set()	...	"md5": "8dde128bdc2609721029b2302f2874f5" "1449c98f0

In this section, important features are chosen out of the 42 columns. These are the important features that are needed to be used inside for building the models both for the classifying the packets whether its malicious or benign as well as for classifying the botnet types

Step5 : In this step, total number of null values will be displayed by utilizing the `isnull()` function and `dropna()` has been utilized in the next cell in order to drop the rows which contains any single null value

```
print(data.isnull().sum())
url          0
botnet       0
md5          0
architecture 0
honeypot     1
tracking     0
tcp          0
udp          0
endpoints   0
antisandbox  0
cnc          0
scanning    0
ddos        0
dns         0
http        0
tor         0
spooft      0
antidebugging 0
antiexecution 0
persistence 0
info_gathering 0
stealth     0
kernel_modules 0
process_injection 0
linking     0
encoded     0
firewall    0
entropy     0
dtype: int64
```

data.head()

	url	botnet	md5	architecture	honeypot	tracking	tcp	udp	endpoints	antisandbox	...	antiscanning	persistence	info_gathering	stealth	kernel_modules
0	http://97.246.6.190/8080/analytica.am7	unknown	9d49976a6b7648b627d47476ba5	amd	linux	0	0	0	0	0	...	0	0	0	0	0
1	http://195.112.249.11/8080/analytica.am8	195.112.249.11/8080/analytica.am8	4b5d776eb13241618b0d19075238	x86	linux	0	0	0	0	0	...	0	0	0	0	0
2	http://142.93.15.194/8080/analytica.am5	142.93.15.194/8080/analytica.am5	28576c17b1916d5315964215af	x86	linux	0	0	0	0	0	...	0	0	0	0	0
3	http://192.206.49.44/8080/	192.206.49.44/8080/	5ed14f6d578120361310c3be32	amd	droppe	0	0	0	0	0	...	0	0	0	0	0
4	http://71.214.157.113/8080/analytica.am6	71.214.157.113/8080/analytica.am6	6861336d36977627673255145	amd	linux	0	0	0	0	0	...	0	0	0	0	0
...
494	http://194.15.36.150/8080/analytica.am7	194.15.36.150/8080/analytica.am7	a4727674223461756534808584	amd	linux	0	0	0	0	0	...	0	0	0	0	0
495	http://194.15.36.150/8080/analytica.am8	194.15.36.150/8080/analytica.am8	a744693c3e3e9193266473601	amd	linux	0	0	0	0	0	...	0	0	0	0	0
496	http://194.15.36.150/8080/analytica.am9	194.15.36.150/8080/analytica.am9	c822a4e4d5d5077630834835	amd	linux	0	0	0	0	0	...	0	0	0	0	0
497	http://45.95.198.19/8080/analytica.am10	45.95.198.19/8080/analytica.am10	955594917541ac76345238091	amd	linux	0	0	0	0	0	...	0	0	0	0	0
498	http://64.227.17.38/8080/analytica.am11	64.227.17.38/8080/analytica.am11	a73e16e47184d3348f208852c	amd	linux	0	0	0	0	0	...	0	0	0	0	0

2.4 Data Cleaning

```
for each_column in data.columns:
    print(each_column)
    for each_value in data[each_column]:
        if ":" in str(each_value):
            new_value = each_value.replace("'",'').replace('["','').replace(']','').replace('}','').replace('(', '').split(':')[1]
            data[each_column] = data[each_column].replace(each_value,new_value)
        else:
            data[each_column] = data[each_column].replace(each_value,0)
```

```
url
botnet
md5
architecture
honeypot
tracking
tcp
udp
endpoints
antisandbox
cnc
scanning
ddos
dns
http
tor
spooft
antidebugging
antiexecution
persistence
info_gathering
stealth
kernel_modules
process_injection
linking
encoded
firewall
entropy
```

Step6: One of the crucial components of data preparation is data learning. The bulk of the values in this dataset are in the Json/Dict format, which is also known as key value pairs. However, the main concern is to retrieve the value from the cells so that they could be used for model training.

To achieve this task, values will be examined whether they are in the format of dictionary or not. If it yes, then they would move forward to use the as replace unnecessary characters and symbols. So that values can be retrieved and save it in the same index.

2.3 Label Encoding

Step 7: Label encoder library has been imported and in the next cell datatypes for each columns has been displayed.

```
[ ] from sklearn.preprocessing import LabelEncoder

[ ] data.dtypes

url                object
botnet             object
md5               int64
architecture      int64
honeypot          float64
tracking           object
tcp               object
udp               object
endpoints         object
antisandbox       object
cnc               object
scanning          object
ddos              object
dns               object
http              object
tor               object
spoofer           object
antidebugging     object
antiexecution     object
persistence       object
info_gathering    object
stealth           object
kernel_modules   object
process_injection object
linking           object
encoded           object
firewall          object
entropy           object
dtype: object
```

```
[ ] for col in ['url','botnet','tracking','tcp','udp','endpoints','antisandbox','cnc','scanning','ddos','dns','http','tor','spoofer','antidebugging','antiexecution','persistence','info_gathering','stealth','kernel_modules','process_injection','linking']:
    print(col)
    data[col] = LabelEncoder().fit_transform(data[col].astype(str))

url
botnet
tracking
tcp
udp
endpoints
antisandbox
cnc
scanning
ddos
dns
http
tor
spoofer
antidebugging
antiexecution
persistence
info_gathering
stealth
kernel_modules
process_injection
linking
encoded
firewall
entropy
```

Step8: In the process of Label Encoding, conversion of all the data into strings has been performed first and then converted them into labels. Through this process, all the columns will be converted into labels. Because ML model accepts inputs only in the form of integers or floating points.

2.4 Dimension Reduction

Step9: PCA library has been imported for the purpose of reduction of columns. Then the entire data of the respected data frames has been converged in to four columns without losing much information. This PCA methodology will helps to increase training process and decreases variance from the model

```
[ ] from sklearn.decomposition import PCA

[ ] pca = PCA(n_components=4)
transformed_data = pca.fit_transform(data)

[ ] transformed_data

array([[ 6.94899797e+02, -6.62606542e+02, -1.57532089e+02,
        1.28184738e+01],
       [ 1.41150116e+03,  3.80484892e+02,  1.63643651e+02,
        2.66229399e-01],
       [-2.83931924e+02,  6.16378204e+02,  1.36756479e+03,
        -5.99755102e+01],
       ...,
       [-7.94562470e+02,  1.05364960e+02,  2.55537239e+01,
        2.16018897e+02],
       [ 1.09489024e+03, -3.23563991e+02, -4.05942641e+01,
        1.69033364e+01],
       [-2.47938030e+02, -4.73370233e+02, -1.28878670e+02,
        2.75955853e+02]])
```

Step10: Following this, the.csv file was imported that had all of the records of the malicious, benign, and botnet categories that were extracted using the md5 hashes provided by virus total checker. In order to take the first 4069 rows from the dataframes iloc function has been utilized .

```
[ ] label_df = pd.read_excel('/content/drive/MyDrive/criptodata/data2/viknesh_dataset (1).xlsx')

[ ] label_df = label_df.iloc[:4069,:]

[ ] label_df
```

	Unnamed: 0	Hash	label	MD5 Hash	final_label	llast
0	2021-05-02 18:10:37	0d4a95b1beeb76c488c6c72c457a6baa	Malicious file: 37 / 60	0d4a95b1beeb76c488c6c72c457a6baa	ELF:Mirai-AHV [Trj]	HEUR:Backdoor.Linux.Mirai.au
1	2021-05-03 09:18:56	4d5d979edb12d241e5108bd719575269	Malicious file: 40 / 61	4d5d979edb12d241e5108bd719575269	ELF:CVE-2017-17215-A [ExpI]	HEUR:Backdoor.Linux.Mirai.b
2	2020-02-17 15:07:27	2d0c79c617b1a196d83e128e5f4215af	Malicious file: 36 / 60	2d0c79c617b1a196d83e128e5f4215af	ELF:Mirai-GB [Trj]	HEUR:Backdoor.Linux.Mirai.b
3	2020-02-16 14:34:24	5ee8149f1c0a570120363150cc3bec52	Malicious file: 31 / 60	5ee8149f1c0a570120363150cc3bec52	ELF:Mirai-VL [Trj]	HEUR:Backdoor.Linux.Mirai.b
4	2020-05-28 03:19:08	8dde128bdc2609721029b230212874f5	Malicious file: 33 / 58	8dde128bdc2609721029b230212874f5	NaN	HEUR:Backdoor.Linux.Mirai.b
...
4064	2020-09-18 22:23:27	475a27be6e2ee6dedd466350fb4c865e	Malicious file: 37 / 56	475a27be6e2ee6dedd466350fb4c865e	ELF:Mirai-AAU [Trj]	HEUR:Backdoor.Linux.Mirai.au
4065	2020-03-24 21:54:32	2295dc7a4b55fabaa13a89cf7e61f15	Malicious file: 33 / 60	2295dc7a4b55fabaa13a89cf7e61f15	ELF:Mirai-ID [Trj]	HEUR:Backdoor.Linux.Mirai.ba
4066	2020-03-24 21:54:43	44a19565fa61174a5bdb3c6dcb418f6b	Malicious file: 16 / 60	44a19565fa61174a5bdb3c6dcb418f6b	ELF:Svirtu-AA [Trj]	HEUR:Backdoor.Linux.Gafgyt.a
4067	2020-03-21 18:22:40	2c571d900d92a5a32ed8e2efc15742b5	Malicious file: 13 / 59	2c571d900d92a5a32ed8e2efc15742b5	ELF:Mirai-GH [Trj]	HEUR:Backdoor.Linux.Mirai.c
4068	2020-03-24 21:23:58	a84725b73d224dc175e5e324d600f584	Malicious file: 13 / 60	a84725b73d224dc175e5e324d600f584	ELF:Gafgyt-FH [Trj]	HEUR:Backdoor.Linux.Gafgyt.a

Step 11: In this step frequency for each value with respect to the column "final_label" were then displayed. Secondly, botnet types were extracted from each value in the column "final_label"

```
label_df['final_label'].value_counts()[:20]
```

```
attacks = []
for i in label_df['label']:
    attacks.append(i.split()[0])
```

Step 12: A list was created in order to save all the values and then appended them, after which they are then converted into numpy arrays

```
[ ] import numpy as np
    attacks = np.array(attacks)
```

Step 13: In this step, concatenation was done for the label and final label, which are xdata. Secondly, the null values are dropped from the xdata by using dropna()

```
[ ] xdata = pd.concat([data,label_df[['label','final_label']],axis=1)
```

```
[ ] xdata = xdata.dropna()
```

```
[ ] xdata1 = xdata[['url', 'botnet', 'md5', 'architecture', 'honeypot', 'tracking', 'tcp',  
'udp', 'endpoints', 'ant sandbox', 'cnc', 'scanning', 'ddos', 'dns',  
'http', 'tor', 'spoof', 'antidebugging', 'antiexecution', 'persistence',  
'info_gathering', 'stealth', 'kernel_modules', 'process_injection',  
'linking', 'encoded', 'firewall', 'entropy']]
```

Step 14: The next step is to extract the label from the column termed 'label'. After which the frequency of the column 'label' will be counted

```
] ydata1 = []  
for i in xdata['label']:  
    ydata1.append(i.split()[0])
```

```
xdata['label'].value_counts()
```

```
Malicious file: 31 / 61 55  
Malicious file: 33 / 60 54  
Malicious file: 32 / 60 53  
Malicious file: 31 / 60 53  
Malicious file: 30 / 61 52  
..  
Malicious file: 22 / 57 1  
Malicious file: 30 / 56 1  
Malicious file: 36 / 56 1  
Malicious file: 26 / 57 1  
Malicious file: 22 / 59 1
```

Step 15: In this step Label Encoder is utilized again in the ydata 1 via fit_transform function. Secondly, label column and other independent columns were concatenated.

```
[ ] ydata1 = LabelEncoder().fit_transform(ydata1)
```

```
[ ] xdata2 = pd.concat([xdata1.reset_index(),pd.Series(ydata1)],axis=1)
```

Step 16: In this step, unique value for the column "final_label" were displayed

```
[ ] xdata['final_label'].unique()
```

```
array(['ELF:Mirai-GH [Trj]', 'ELF:Svirtu-AA [Trj]', 'ELF:Mirai-APD [Trj]',  
'Other:Malware-gen [Trj]', 'ELF:Mirai-ACU [Trj]'], dtype=object)
```

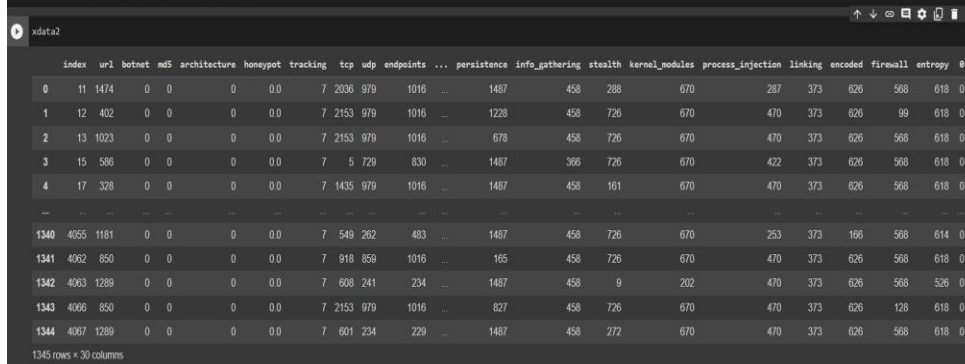
Step 17: The list were created again for the column "final_label" and then appending it ydata2

```
] ydata2 = []  
for i in xdata['final_label']:  
    ydata2.append(i)
```

```
[ ] ydata2 = np.array(ydata2)
```


Step 18: Data are transformed in this stage using PCA.

```
pca = PCA(n_components=30)
transformed_xdata1 = pca.fit_transform(xdata1)
transformed_xdata2 = pca.fit_transform(xdata2)
```



```
[ ] transformed_xdata1
```

```
array([[ -369.80769972, -550.0173564 , -568.02481226,  41.04735489],
       [-682.03890567,  523.53936112, -366.05720207,  2.94201659],
       [-910.91869025,  -80.23506528,  195.78253252, -173.22461041],
       ...,
       [ 889.84359969, -217.57835811,  713.39017223,  761.6484275 ],
       [-808.67767229,  84.78686027,  -41.58459481, -220.19111346],
       [1103.73946781, -252.88049145,  71.75930677,  64.54425851]])
```

Step 19: The.csv file containing all the malicious and benign data based on the md5 hash values is imported once again in this stage. Second, the variable label_df has been given the value "label."

```
[ ] import pandas as pd
label_df = pd.read_excel('/content/drive/MyDrive/cryptodata/data2/viknesh_dataset (1).xlsx').i
label_df['label']
```

```
[ ] label_df = label_df['label']
```

Step 20: In this stage, data and label_df are combined, and each value in the column "label" is tallied.

```
data = pd.concat([data,label_df],axis=1)
```

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

```
Malicious file: 14 / 60    172
Malicious file: 14 / 59    109
Malicious file: 18 / 60     68
Malicious file: 18 / 59     50
Malicious file: 30 / 61     48
...
Malicious file: 15 / 58      1
Malicious file: 26 / 56      1
Malicious file: 33 / 63      1
Malicious file: 32 / 54      1
Malicious file: 30 / 56      1
```

```
[ ] _xdata1 = data[['url', 'botnet', 'md5', 'architecture', 'honeypot', 'tracking', 'tcp',  
                'udp', 'endpoints', 'antisandbox', 'cnc', 'scanning', 'ddos', 'dns',  
                'http', 'tor', 'spooof', 'antidebugging', 'antiexecution', 'persistence',  
                'info_gathering', 'stealth', 'kernel_modules', 'process_injection',  
                'linking', 'encoded', 'firewall', 'entropy']]
```

Step 21: The 'label' column in this phase was found to be ydata1, and then the distinct values were presented. It is concluded that the excellent, clean, and unknown are benign. The ydata1 has been subjected to label encoding through the fit_transform function.

```
[ ] data['label'] = ydata1  
  
[ ] data['label'].unique()  
  
array(['Malicious', 'Unknown', 'Clean:', 'Good:'], dtype=object)
```

```
[ ] ydata1.unique()  
  
array(['Malicious', 'Benign'], dtype=object)
```

```
[ ] ydata1 = LabelEncoder().fit_transform(ydata1)
```

2.5 Train/Test Split:

Lightgbm, Catboost, classification_report, train_test_split were imported.

```
[ ] from lightgbm import LGBMClassifier  
  
    from catboost import CatBoostClassifier  
  
[ ] from sklearn.model_selection import train_test_split  
  
[ ] from sklearn.metrics import classification_report
```

```
▶ xtrain,xtest,ytrain,ytest = train_test_split(transformed_xdata1,ydata1,test_size=0.30,random_state=9)  
  xtrain,xtest,ytrain,ytest = train_test_split(transformed_xdata2,_ydata2,test_size=0.10,random_state=9)
```

The division of data into train and test is the last and most crucial step in the processing of data. By separating the test and train sets of data. It assesses the model's performance in both visible and hidden data. The model is being trained on the seen data, while the test set is the unobserved data that will be used to evaluate the model's performance. The train and test functions are utilized to train and test the data

3. Model

3.1 Model Training & Testing:

The next phase of the data preprocessing is the model building part. The proposed methodology states that the two different model LightGBM and Catboost will be combined together via soft voting classification which takes the probability of the output from LightGbm and Catboost . The boosting algorithms helps to reduce the variance and biasness from the model.

Soft voting classification method was opted. Through the help of Voting Classification, we can combine the above selected boosting models and make them one. The models have been trained and tested were done in the ratio of 90 :10 for classifying the botnet types which is xdata2 and ydata2 and for the xdata1 and ydata1 were trained and tested in the ratio of 70: 30 which is for classifying initially whether the packets are benign or malicious.

For xdata2 and ydata2 (classification of IoT botnets):

CatBoost Training:

```
cb_model = CatBoostClassifier(iterations=1000)
cb_model.fit(xtrain,ytrain)
```

Catboost Testing

```
predictions = cb_model.predict(xtest)
```

LightGBM Training:

```
[ ] lgb_model = LGBMClassifier()
    lgb_model.fit(xtrain,ytrain)
```

LightGbm Testing:

```
predictions = lgb_model.predict(xtest)
```

Soft voting classification Training:

```
[ ]
    cb_model = CatBoostClassifier(iterations=1000)
    lgb_model = LGBMClassifier()
    soft_voting = VotingClassifier(estimators = [('cat_boost', cb_model), ('lgb_model', lgb_model)], voting = 'soft')
    soft_voting.fit(xtrain,ytrain)
```

Soft voting classification Testing:

```
predictions = soft_voting.predict(xtest)
```

For xdata1 and ydata1 (classifying whether its benign or malicious):

The same procedures used for xdata2 and ydata2 are also used to train and evaluate xdata1 and ydata1. However, as noted above in the train and split portion, testing and training are divided 70:30. The major training and testing of Xdata1 and Ydata1 is done to determine if the output is malicious or benign if the output is 0 it is considered as benign and if the output is, then it is considered as malicious.

Soft voting classification Training:

```
[ ]
cb_model = CatBoostClassifier(iterations=1000)
lgb_model = LGBMClassifier()
soft_voting = VotingClassifier(estimators = [('cat_boost', cb_model), ('lgb_model', lgb_model)], voting = 'soft')
soft_voting.fit(xtrain,ytrain)
```

Soft voting classification Testing:

```
predictions = soft_voting.predict(xtest)
```

4. Evaluation:

For the classification Benign or Malicious:

The Performance of the soft voting classification were examined for both the models i.e; whether to classify whether the packets or malicious as well as for the botnet types

The first section focus on checking the classification performance of Benign and Malicious. For this Accuracy, F1 score, recall were calculated . The accuracy we are achieving with the help of combined Cat Boost and LightGbm Classifier via softvoting classifier is 99%.

```
Learning rate set to 0.5
0:   learn: 0.3554486      total: 1.56ms  remaining: 0us
      precision    recall  f1-score   support

     0       0.00      0.00      0.00         5
     1       0.99      1.00      1.00       595

 accuracy          0.99         600
 macro avg         0.50         600
 weighted avg      0.98         600
```

False Alarm Rate:

Flase Alarm Rate is one of the important merics for the IDS.

```
print("The False Alarm rate for Malicious and Benign is: {}".format(FAR))
The False Alarm rate for Malicious and Benign is: 0.5008403361344538
```

For the classification of the Botnet types:

```
precision    recall  f1-score   support

 ELF:Mirai-ACU [Trj]      0.97      0.88      0.92         33
 ELF:Mirai-APD [Trj]     1.00      0.90      0.95         49
 ELF:Mirai-GH [Trj]      0.90      0.93      0.92        134
 ELF:Svirtu-AA [Trj]     0.92      0.92      0.92         89
 Other:Malware-gen [Trj] 0.95      0.98      0.97         99

 accuracy          0.93        404
 macro avg         0.95         404
 weighted avg      0.93         404
```

5. Conclusion:

In this research work, Ensemble based IDS was built to detect and IoT botnets. The model was built by combining LightGBM and Catboost into a single voting classification which decides the probability of the outputs from LightGbm and catboost into a single output. However, this model classifies whether the packet is malicious or benign and it will also classify the botnet types. The models were trained and tested on the ratio of 70:30 for classifying whether it is benign or malicious which attained the accuracy of 99 percent and the False Alarm rate was 0.5, and for classifying botnet types the model was trained and tested in the ratio of 90 : 10 in which the 93 percent accuracy was attained.