

AN OPTIMIZED ENSEMBLED FRAMEWORK TO SECURE IOT NETWORK

Project report submitted in partial fulfilment of the requirement for the degree
of Bachelor of Technology

in

Computer Science and Engineering

By

HARSHIL CHAUDHARY(191371)

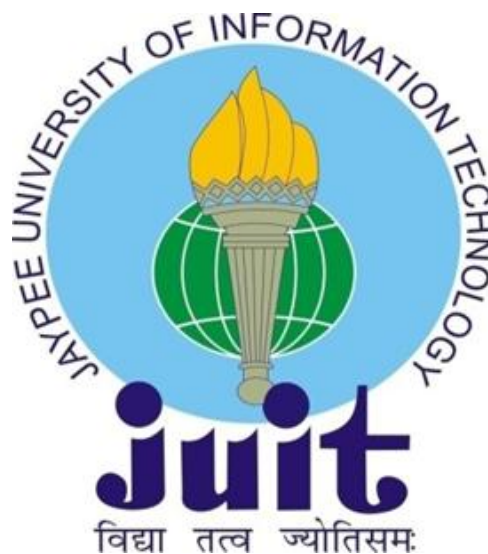
VISHALIKA KATTIYAR (191432)

Under the supervision of

Dr. Himanshu Jindal

to

Department of Computer Science & Engineering and Information Technology



**Jaypee University of Information Technology,
Waknaghat , Solan-173234, Himachal Pradesh**

DECLARATION

We hereby declare that the work presented in this report entitled **An optimised ensembled framework to Secure IoT network** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from August 2022 to November2022 under the supervision of **Dr. Himanshu Jindal** (Professor, Department of CSE).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Harshil Chaudhary
191371

Vishalika Katiyar
191432

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Himanshu Jindal
Assistant Professor (SG)
Department of Computer Science and Engineering
Dated:

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT
PLAGIARISM VERIFICATION REPORT

Date:

Type of Document (Tick): PhD Thesis M.Tech Dissertation/ Report B.Tech Project Report Paper

Name: _____ Department: _____ Enrolment No _____

Contact No. _____ E-mail. _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor)

Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none"> • All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String 		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature

Librarian

ACKNOWLEDGEMENT

Firstly, we express my heartiest thanks and gratefulness to almighty God for his divine blessing making it possible to complete the project work successfully.

We are really grateful and wish my profound indebtedness to our supervisor **Dr. Himanshu Jindal, Assistant Professor (SG)**, Department of CSE, Jaypee University of Information Technology, Waknaghat. His deep knowledge & keen interest in the field of “**Machine Learning and IoT**” helped us to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

We would like to express our heartiest gratitude to **Dr. Himanshu Jindal**, Department of CSE, for his kind help to finish our project.

We would also generously welcome each one of those individuals who have helped us straightforwardly or in a roundabout way in making this project a win. In this unique situation, we might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, we must acknowledge with due respect the constant support and patience of my parents.

Harshil Chaudhary
191371

Vishalika Katiyar
191432

LIST OF ABBREVIATIONS

IOT	Internet of Things
Dos	Denial of Service
ML	Machine Learning
CNN	Convolutional Neural Network
OPTICS	Ordering points to Identify the clustering structures
PCA	Principal Component Analysis
ReLU	Rectified Linear Units
R2L	MediaPipe
U2R	Computer Vision
FAR	False Alarm Rate
PR	Precision - Recall
IDS	Intrusion Detection System
LSTM	Long Short Term Memory
Impl	Implementation
PCA	Principal Component Analysis

LIST OF FIGURES

CHAPTER 1

- Figure 1.1 : Model Architecture
- Figure 1.2 : The Testbed Visualization for UNSW-NB15
- Figure 1.3 : Dataset Preprocessing for replacing missing values
- Figure 1.4 : Normalising Features
- Figure 1.5 : Applying PCA for feature selection
- Figure 1.6 : Calculating the evaluating matrices

CHAPTER 3

- Figure 3.1 : Optimised ensemble framework
- Figure 3.2 : Dataset Details
- Figure 3.3 : Necessary Libraries
- Figure 3.4 : Loading Dataset
- Figure 3.5 : Dataset Shuffling
- Figure 3.6 : Dataset
- Figure 3.7 : Libraries For Model 2
- Figure 3.8 : Libraries For Model 2
- Figure 3.9 : Replacing missing values and shuffling dataset
- Figure 3.10 : Applying PCA for feature selection
- Figure 3.11 : Splitting dataset into a training and testing dataset
- Figure 3.12 : Building the LSTM Model
- Figure 3.13 : Training the model using 10 epochs
- Figure 3.14 : Splitting the dataset and creating Algorithm
- Figure 3.15 : Creating Bagging Classifier

- Figure 3.16 : Fitting the Bagging Classifier
- Figure 3.17 : Splitting the dataset and defining the model
- Figure 3.18 : Fitting the model

CHAPTER 4

- Figure 4.1 : Result of Mini Batch Algorithm
- Figure 4.2 : Result of Fuzzy CMeans Algorithm
- Figure 4.3 : Result of OPTICS Algorithm
- Figure 4.4 : Loss Graph for training dataset
- Figure 4.5 : Loss graph for testing dataset
- Figure 4.6 : Confusion Matrix 1
- Figure 4.7 Confusion Matrix 2
- Figure 4.8 : RSME Score
- Figure 4.9 : Evaluation Metrics of Bagging Decision tree
- Figure 4.10 : Evaluation Metrics of bagging Decision Tree
- Figure 4.11 : Accuracy graph of Bagging Decision Tree
- Figure 4.12 : Loss graph of Bagging Decision Tree
- Figure 4.13 : Evaluation Metrics of random Forest Classifier
- Figure 4.14 : confusion matrix of Random Forest Algorithm
- Figure 4.15 : Accuracy graph of Random Forest Classifier
- Figure 4.16 : Loss graph of Random Forest Classifier

CHAPTER 7

- Figure 7.1 : Import and Install Dependencies
- Figure 7.2 : Normalisation and Feature selection
- Figure 7.3 : Clustering into Normal/Malicious using Mini Batch Algorithm

- Figure 7.4 : Implementation of Fuzzy C MEANS
- Figure 7.5 : Implementation of OPTICS
- Figure 7.6 : Weighted Voted
- Figure 7.7 : Implementation of CNN
- Figure 7.8 : Accuracy graph
- Figure 7.9 : Loss graph
- Figure 7.10 & Figure 7.11 : Confusion matrix
- Figure 7.12 : LSTM Impl.
- Figure 7.13 : Bagging Decision Tree Impl.
- Figure 7.14 : Random Forest Classifier Impl.
- Figure 7.15 : LSTM Result
- Figure 7.16 : Evaluation of Bagging Decision Tree
- Figure 7.17 : Accuracy graph for Bagging Decision Tree
- Figure 7.18 : Loss graph for Bagging Decision Tree
- Figure 7.19 : Results for Random Forest Classifier
- Figure 7.20 : Accuracy graph for Random Forest Classifier
- Figure 7.21 : Calculate Loss graph for Random Forest Classifier
- Figure 7.22 : Loss graph for Random Forest Classifier

LIST OF TABLES

- Table 1.1 : Dataset Statistics
- Table 1.2 : Labelled Features

TABLE OF CONTENTS

CERTIFICATE	I
PLAGIARISM CERTIFICATE	II
ACKNOWLEDGEMENT	III
LIST OF ABBREVIATIONS	IV
LIST OF FIGURES	V
LIST OF TABLES	VIII
ABSTRACT	XI
1. Chapter-1 INTRODUCTION	1
1 Introduction	1
2 Problem Statement	2
3 Objectives	3
4 Methodology	4
2. Chapter-2 LITERATURE SURVEY	12
1 Literature Survey	12
3. Chapter-3 SYSTEM DEVELOPMENT	16
1 Model 1	16
➤ Methodology Used	
➤ Tools Used	
➤ Proposed Method	
➤ Unsupervised Learning	
➤ Model Details	
2 Model 2	21
➤ Methodology Used	
➤ Tools Used	
➤ Proposed Method	
➤ Models Used	
4. Chapter-4 PERFORMANCE ANALYSIS	29
1 Performance of Model 1	29
2 Performance of Model 2	32
5. Chapter-5 CONCLUSIONS	38

1	Conclusions	40
2	Future Scope	41
3	Applications Contributions	42
	6. REFERENCES	39
	7. APPENDICES	42

ABSTRACT

Through the Internet, the Internet of Things (IoT) has changed how people live. Cyber-physical systems (CPS) and other traditional disciplines have been transformed into smart regions by IoT, which has specified numerous smart solutions for everyday issues. The majority of the Internet of Things' edge devices have incredibly low processing speeds. To overturn the IoT networks, attackers can use these devices to initiate a variety of network attacks. In addition to that, as IoT devices are added to a greater degree, the potential for new and unknown security threats grows exponentially. Due to this reason, an intelligent security framework for IoT networks must be developed that can identify such threats.

We have created an unsupervised ensemble learning model in this paper that can identify new or unidentified attacks in an IoT network from an unlabeled dataset. A deep learning model is trained to recognise IoT network assaults using the system-generated tagged dataset. The study also offers a feature selection technique for selecting the dataset's most critical elements for threat detection. The research indicates that the proposed model can recognise unlabeled IoT network datasets, and DBN (Deep Belief Network) outperforms the other models with an accuracy of \rightarrow (97.5%) along with a False Alarm rate \rightarrow (2.3%) when trained using labelled datasets provided by the proposed approach.

Keywords Used : Unsupervised Machine Learning, Clustering, Feature Selection, Attacks, Internet of things, Deep Learning, Fuzzy C-Means, Optics, Mini-Batch Algorithm.

CHAPTER - 1

1.1 Introduction

Globally, the Internet of Things (IoT) has experienced exponential development. Despite the fact that the IoT is adopted by millions of people, attacks like man in the middle, spoofing, and denial of service (DOS) make it difficult for these networks to function. The privacy and security of the consumer is highly compromised by these cyberattacks, which also jeopardises the entire IoT ecosystem. Therefore, it is still quite difficult for researchers to forecast and identify new unidentified network assaults inside an IoT network. Recently, detecting and classifying various assaults in an IoT network has been greatly aided by newly developed machine learning and deep learning models. However, as the number of attacks are on a rise, these algorithms become more and more computationally complex.

Huge amounts of data are produced as a result of the sensors used in IoT networks and CPS continuously monitoring their environment. IoT networks and CPS are vulnerable to numerous cyberattacks because of the enormous volumes of data that are stored in data centres, some of which may contain sensitive information about systems or individuals. Attacks like Denial of Service (DoS), Remote to Local (R2L), Brute Force, Probing (Probe), User to Root (U2R), Man-in-the-middle, Scanning, Ransomware, Password assault, etc. are becoming more prevalent and obvious, wrecking havoc and causing irreparable harm. Additionally, the attack surface is expanded when a network uses a lot of diverse IoT devices. Because data centres contain vast amounts of information sources, attackers frequently target them. The low storage and computing capability of IoT network devices makes it impossible for them to detect and defend against potential online cyber attacks. Also, the detection of new or undiscovered assaults makes attack detection increasingly difficult due to the expansion of the threat vectors for IoT networks.

This introduction chapter provides a summary of the security risks to IoT networks and IoT devices along with the main driving forces behind this project. It also includes a small briefing on the topics worked and a general summary of the chapters ahead is given along with the important results obtained.

1.2 Problem Statement

The acceptance of small embedded devices, often referred to as the Internet of Things, have grown exponentially during the past few years. These technologies are gradually entering every element of contemporary life and altering how individuals, organisations, and sectors of the economy function. The desire for smart apps that can work independently without requiring human interference has become one of the main motivations behind the progress in this field (for control and data analysis). The creation of efficient applications, enhanced communication protocols, and breakthroughs in integrated systems design have all contributed to the acceleration of IoT growth.

The absence of built in security features comparable to those found in conventional systems, such as servers, desktop computers, and tables, has been among the most obvious shortcomings of IoT. This issue results from the fact the majority of IoT devices lack the processing power needed to execute sophisticated security procedures and encryption. The “three Ps,” which we categorise as prototype, production, and packaging, have traditionally received much more interest from IoT device manufacturers due to rising market competitiveness in these fields rather than working on its security. This reality makes it extremely challenging to develop a single security architecture that works well in all circumstances.

It just becomes crucial to make sure that bona-fide security measures are put in place to maintain the confidentiality of user's data and safeguard sensitive user data from being accessed by malicious attackers. Due to the progressive nature of the IoT environments, the conventional security trilogy --> (confidentiality, integrity, and authentication) is no longer feasible and must now also contain other security features, such as access control and availability.

1.3 Objectives

1.3.1 Model 1

- We suggest using PCA to identify features in order to find attacks in an IoT network. In order to boost the system's overall performance, the most pertinent features from the available network datasets are chosen for amalgamation after a thorough examination of the performance metrics and identification of essential parameters.
- The model ought to be able to detect out of sight patterns inside the network dataset & clusterize the network traffic into categories of normal and malicious to find new or unknown threats. We have employed a number of clustering algorithms to accomplish this. For a specific IoT network traffic, the output of each clustering method is coupled using a weighted voting mechanism to more accurately forecast the class label (normal / malicious).
- After a thorough performance examination, the weights assigned to each clustering technique's output were determined. An ensemble machine learning approach that turns an unlabeled dataset into a labelled dataset as a result of the combination of clustering techniques and weighted voting is able to discover new/unknown assaults in the IoT network traffic.
- The suggested model's labelled dataset is used to train a deep learning model for IoT network threat detection. It is advised to employ a convolutional neural network architecture for IoT network attack detection.

1.3.2 Model 2

- We suggest using PCA to select the features to identify or to find the attacks in an IoT network. And before applying the PCA dataset should be pre processed by different methods like by replacing the missing values in the dataset. And by using label encoder to replace all the string values int values. Then after these steps all the features should be normalised by applying standard scalers.

- After PCA, three different algorithms will be introduced and all the required libraries will be installed. Three algorithms include -
 1. LSTM (Long Short Term Memory)
 2. Bagging Decision Tree
 3. Random Forest classifier

Second and third algorithms are classifiers and LSTM is a regression model.

- In model 1 we utilised the weighted voting from all three algorithms and predicted their cumulative results to classify an entry as malicious or non-malicious. In model 2 we will apply three different algorithms namely LSTM, Bagging Decision Tree, Random Forest classifier. These predict the malicious and non-malicious entries individually.
- In total we have tried to create two different models including both individual and cumulative prediction of cyber threats. These both models have their own qualities and working process.

1.4 Methodology

1.4.1 Model 1

The study suggests an optimised unsupervised ensemble strategy for classifying network traffic into Malicious and Normal, hence discovering new or undiscovered attacks. It uses system detected network traffic for training the deep learning model to identify attacks inside an IoT network. The suggested mechanism has an overall attack detection accuracy of 97.6 percent and a false alarm rate of 2.3 percent, on the UNSW NB-15 dataset picked from kaggle.

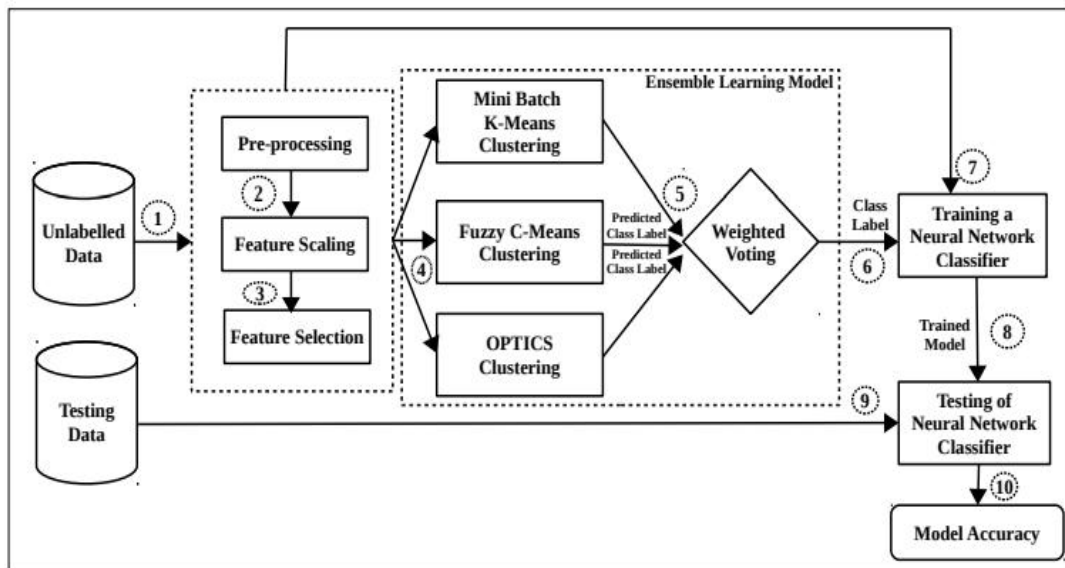


Figure 1.1 - Model Architecture

Stage 1: Data preprocessing

The suggested model is seen in figure (1), which accepts the unlabeled dataset from an IoT network as an input that contains both malicious and non-malicious IoT network traffic.

Dataset Details

Network intrusion dataset ---> UNSW-NB15 is used here. It consists of nine distinct attacks, involving DoS (Denial of Service), Fuzzers, Malware , Backdoors, Exploits, Reconnaissance, Shellcode, Generic and Worms. Raw network packets are included in the collection. 175,341 records make up the training set, while 82,332 records from the attack and normal types make the testing set.

Statistical features		16 hours	15 hours
No._of_flows		987,627	976,882
Src_bytes		4,860,168,866	5,940,523,728
Des_bytes		44,743,560,943	44,303,195,509
Src_Pkts		41,168,425	41,129,810
Dst_pkts		53,402,915	52,585,462
Protocol types	TCP	771,488	720,665
	UDP	301,528	688,616
	ICMP	150	374
	Others	150	374
Label	Normal	1,064,987	1,153,774
	Attack	22,215	299,068
Unique	Src_ip	40	41
	Dst_ip	44	45

Table 1.1 : Dataset Statistics

#	Name	T	Description
48	<i>attack_cat</i>	N	The name of each attack category. In this data set, nine categories (e.g., Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms)
49	<i>Label</i>	B	0 for normal and 1 for attack records
<i>Type (T.) N: nominal, I: integer, F: float, T: timestamp and B: binary</i>			

Table 1.2 : Labelled Features

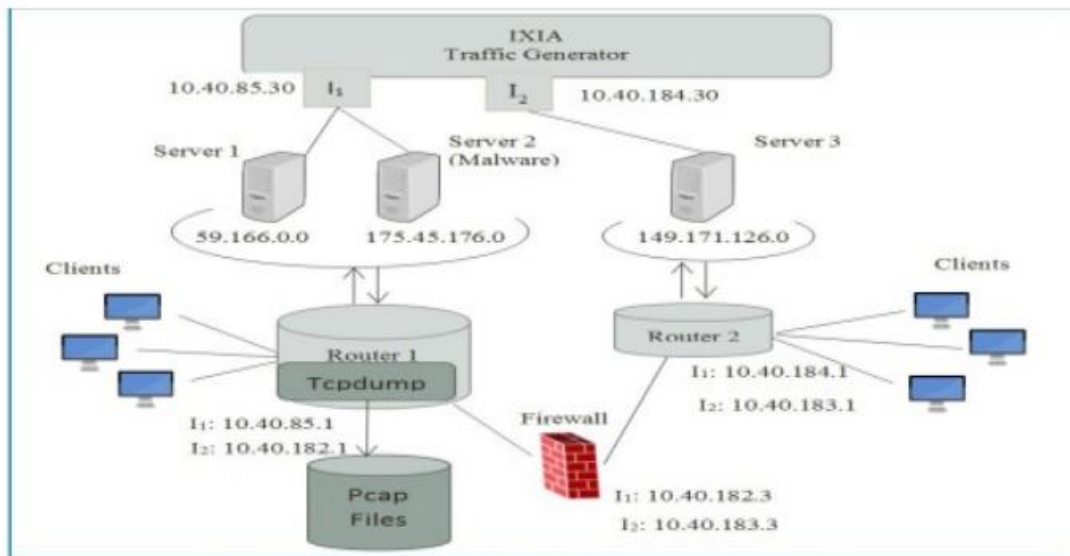


Figure 1.2 : The Testbed Visualization for UNSW-NB15

The unlabeled input dataset is initially pre-processed to remove any duplicate or missing data by replacing NA values and then replacing with 0 and then shuffling the dataset. Post the data pre-processing, a standard scaler is used to normalise the dataset, i.e., scale the value of each characteristic. The suggested "Feature Selection" module is then provided with our pre-processed and scaled dataset to select pertinent features for identifying network assaults.

Stage 2: Feature Selection

The suggested "Feature Selection" module is then provided with our pre-processed and scaled dataset to select pertinent features for identifying network assaults. Any kind of machine learning algorithm's effectiveness depends on choosing the right features from the available dataset. Machine learning models that are either over-fitted or under-fitted and this anomaly can be solved by choosing essential characteristics from a dataset. To find essential features for the proposed model, we used PCA (Principal component analysis).

We are applying PCA (Principal component analysis) to selected relevant features and in above PREPROCESSED data we can observe it contains 39 features and

after applying the PCA algorithm it selects 25 important features.

Stage 3: Ensemble Learning & Deep Learning Model

The foundation of ensemble learning is the idea that results can be improved by combining the outputs of different learning models. This ensemble approach can be used in 2 ways to create numerous anticipated outputs. One being - Independent ensemble and the other is Coordinated ensemble construction. In an independent ensemble construction approach, a learning algorithm can be executed independently multiple times on various training data subsets or different learning models can be executed independently on the same dataset, to produce multiple results which can then be combined using ensemble technique. Whereas, in coordinated ensemble building the outcome of one learning algorithm can be utilised as an input to another learning algorithm, thereby making all the base learning algorithms dependent on each other.

We have used an ensemble learning approach by applying 3 algorithms :

1. Mini Batch Algorithm.
2. Fuzzy C means for clustering.
3. OPTICS clustering.

The predicted output of each clustering technique is either 0 or 1, where 0 represents non-malicious traffic (cluster-1) and 1 represents malicious traffic (cluster-2). The predicted output from each clustering algorithm, for each data entry, is combined using weighted voting using equation 3 to create two clusters – one containing benign data and another containing malicious data.

A labelled dataset is produced by the suggested ensemble model. The generated annotated dataset is then utilised to train different deep learning models. In our research, we used a CNN network.

1.4.2 Model 2

Stage 1: Data preprocessing

Data Preprocessing usually involves cleaning, preparing and transforming raw data into that format which can be easily processed by the algorithms. And the need for this data preprocessing arises because of several reasons. Raw data is often incomplete, or contains errors which sometimes lead to inaccurate results. Data should be in a specific format such as Numerical values, therefore these all steps like converting the categorical data into numerical data comes under pre processing.

```
[ ] #dataset processing replace missing values
    dataset.fillna(0, inplace = True)
    dataset = dataset.values
    print("Dataset preprocessing completed")
```

```
Dataset preprocessing completed
```

Fi

Figure 1.3 : Dataset Preprocessing for replacing missing values

```
[ ] #applying standard scaler to normalized features
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    print("Features normalization task completed")
```

```
Features normalization task completed
```

Figure 1.4 : Normalising Features

Stage 2: Feature Selection

Principal Component Analysis (PCA) is a technique popularly used for feature selection because it identifies the most important and right features that the models can work on. PCA does this by transforming the original features into a set of new features called principal components that are made of different combinations of original features.

```
[ ] #applying PCA for features selection
    print("Total features found in dataset before applying PCA : "+str(X.shape[1]))
    pca = PCA(n_components = 25)
    X = pca.fit_transform(X)
    print("Total features found in dataset after applying PCA : "+str(X.shape[1]))
```

```
Total features found in dataset before applying PCA : 40
Total features found in dataset after applying PCA : 25
```

Figure 1.5 : Applying PCA for feature selection

After normalising the dataset there were 40 features in the dataset and after applying the PCA for feature selection there are 24 features that would be used by the algorithms.

Stage 3: Deploying Model

In this model 2 we are using 3 three different algorithms to train the model by splitting the dataset into training set and testing set and then finding the accuracy of the model of prediction.

The models are :-

1. LSTM - long Short Term Memory
2. Bagging Decision Tree
3. Random Forest Classification

After PCA is done for feature selection, required libraries for each algorithms are installed and the dataset is divided into training and testing sets then each model is build and trained on the training dataset and then testing the algorithms on testing datasets and then check models -

- Accuracy_score
- Precision_score
- Recall
- F1_score
- Confusion matrix

```
[ ] # Calculate the evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
```

Figure 1.6 : Calculating the evaluating matrices

CHAPTER - 2

LITERATURE SURVEY

- [1] The aim of this article is to provide a broad overview of the security risks in the IoT sector and to discuss some possible interactions. To this end, after a general introduction to security in the IoT domain, we discuss the specific security mechanisms adopted by the most popular IoT communication protocols. Then, we report and analyse some of the attacks against real IoT devices reported in the literature, in order to point out the current security weaknesses of commercial IoT solutions and remark the importance of considering security as an integral part in the design of IoT systems. We conclude this article with a reasoned comparison of the considered IoT technologies with respect to a set of qualifying security attributes, namely integrity, anonymity, confidentiality, privacy, access control, authentication, authorization, resilience, self organisation.
- [2] Rossie PS et al. discussed the problem of distributed detection of a non-cooperative (Unknown emitted signal) target with a wireless sensor network. Davies' framework is exploited herein to design the generalised forms of Rao and locally optimum detection (LOD) tests. For our generalised Rao and LOD approaches, a heuristic approach for threshold optimization is also proposed. The simulation results confirm the promising performance of our proposed approaches.
- [3] Javed F et al. provides a detailed comparison of the OS's designed for IoT devices on the basis of their architecture, scheduling methods, networking technologies, programming models, power and memory management methods, together with other features required for IoT applications. In addition, various applications, challenges, and case studies in the field of IoT research are discussed.

- [4] Hassan WH et.al presents an analysis of recent research in IoT security from 2016 to 2018, its trends and open issues. The main contribution of this paper is to provide an overview of the current state of IoT security research , the relevant tools, IoT modellers and simulators.
- [5] In this paper , the investigation of the prospects of using machine learning classification algorithms for securing IoT against DoS attacks has been done. A comprehensive study carried out on the classifiers which can advance the development of anomaly-based intrusion detection systems (IDS's). Performance assessment of classifiers is done in terms of prominent metrics and validation methods. Popular datasets CIDDs- 001, UNSW- NB15, and NSL-KDD are used for benchmarking classifiers. Friedman and Neymenyi tests are employed to analyse the significant differences among classifiers statistically. In addition , Raspberry Pi is used to evaluate the response time of classifiers on IoT specific hardware.
- [6] Kumar N at al. in this proposal, presented a cognitive spammer framework that removes spam pages when search engines calculate the web page rank score. The framework detects web spam with the support of Long Short-term Memory network by training the link features. This training resulted with an accuracy of 95-25 as more that 1,11,000 hosts are being correctly classified. However, the content features are trained by a neural network. The proposed scheme has been validated with the WEBSpAM-UK 2007 dataset. Prior to processing, the dataset is pre-processed using a new technique called "Split by Oversampling and Train by Under-fitting". The ensemble and cross validation approach has been used for optimization of results with an accuracy of 96.96%.

- [7] Eskandari M et al. presented Passban, an intelligent intrusion detection system (IDS) able to protect the IoT devices that are directly connected to it. The proposed solution is that it can be deployed directly on very cheap IoT gateways (e.g., single board PCs currently costing few tens of U.S. dollars), hence taking full advantage of the edge computing paradigm to detect cyber threats as close as possible to the corresponding data sources. We will demonstrate that Passban is able to detect various types of malicious traffic, including Port Scanning, HTTP and SSH Force, and SYN Flood attacks with very low false positive rates and satisfactory accuracies.
- [8] This paper proposed a non-symmetric deep autoencoder (NDAE) for unsupervised feature learning. Furthermore, a novel deep learning classification model is constructed using stacked NDAEs. The proposed classifier has been implemented in a graphics processing unit (GPU)-enabled TensorFlow and evaluated using the benchmark KDD Cup '99 and NSL-KDD datasets. Promising results have been obtained from the model thus far, demonstrating improvements over existing approaches and the strong potential for use in modern NIDSs.
- [9] This paper presents a novel security framework and an attack detection mechanism using a Deep Learning model to fill in the gap, which will efficiently detect malicious devices. The proposed mechanism uses a Convolutional Neural Network (CNN) to extract the accurate feature representation of data and further classifies those by Long Short-term Memory (LSTM) Model. The dataset used in the experimental evaluation is from twenty Raspberry Pi infected IoT devices. In addition, it is observed that the proposed model outperformed various recently proposed DL- based attack detection mechanisms.

[10] Nanda P et al. proposed a mutual information based algorithm that analytically selects the optimal feature for classification. This mutual information based feature selection algorithm can handle linearly and nonlinearity dependent data features. An Intrusion Detection System (IDS), named Least Square Support Vector Machine based IDS (LSSVM-IDS), is built using the features selected in the proposed feature selection algorithm. The performance of LSSVM-IDS is evaluated using three intrusion detection evaluation datasets, namely KDD Cup 99, NSL-KDD and Kyoto 2006+ dataset. The evaluation results show that our feature selection algorithm contributes more critical features for LS SVM-IDS to achieve better accuracy and lower computational cost compared with the state-of-the-art methods.

CHAPTER - 3

SYSTEM DEVELOPMENT

3.1 Model 1

This chapter contains information on the dataset and its patterns. The dataset is provided visually in the form of graphs and tables. The proposed model is also presented, as is the process that led to it. The architecture of the proposed Optimised Ensemble Framework to secure IOT Network is shown in figure 10.

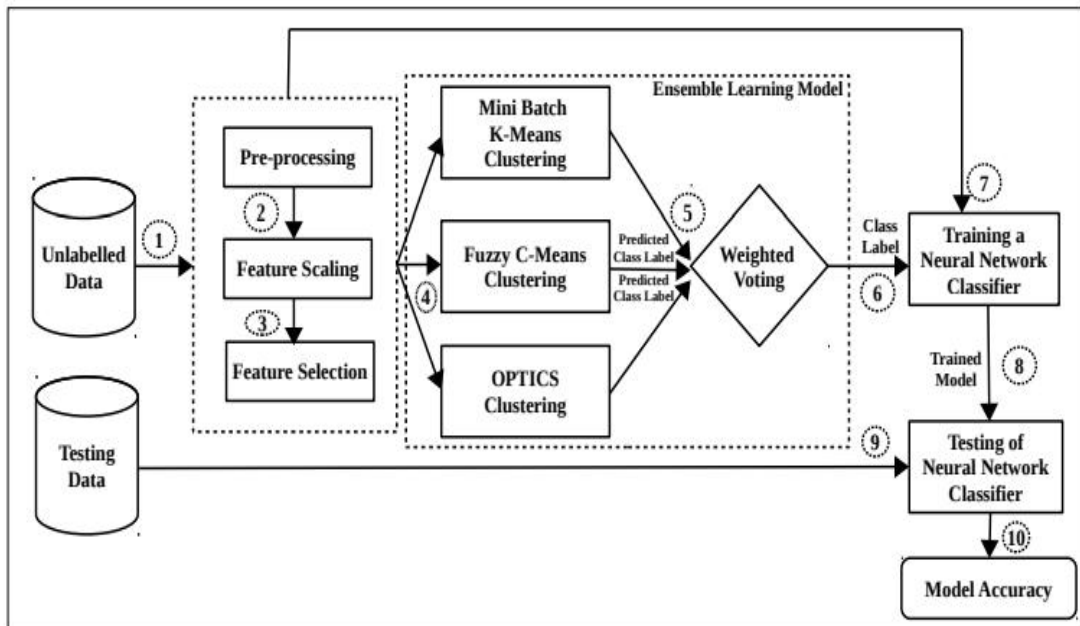


Figure 3.1 : Optimised ensemble framework

3.1.1 Methodology Used

- Install & Import necessary Dependencies.
- Data collection (UNSW_NB15 dataset from kaggle)
- Data Pre-processing
- Data Normalisation & then applying PCA

- Splitting the dataset into training & testing datasets
- Training dataset with MINI BATCH K MEANS clustering
- Training dataset with Fuzzy C MEANS clustering
- Training dataset with OPTICS clustering
- Applying Weighted Voting to get label with highest weight
- Defining Deep Learning layers & Applying CNN model
- Performing prediction on test data & calculating accuracy

3.1.2 Tools Platforms/Technology/Languages Used

- Imported UNSW_NB15 dataset from kaggle.
- Tools & Platforms used are Anaconda Navigator for setting up the environment for Jupyter Notebook.
- Language Used: Python

3.1.3 Proposed Method

In this section we introduce the proposed model for the detection of attacks in an IoT network. The proposed model which takes an unlabelled IoT network dataset (containing IoT network traffic: malicious as well as non-malicious) as input. The input unlabelled dataset is first pre-processed for any missing/redundant data. After data pre-processing, the value of each feature is being scaled using a standard scaler. The pre-processed and scaled dataset is then given to the proposed feature selection module to select relevant features for detecting network attacks. After feature selection, each data entry of the dataset (for selected features) is given to three clustering mechanisms (Mini batch K-means, Fuzzy C-means and OPTICS) simultaneously which cluster the data into malicious (represented by 1) or non-malicious (represented by 0).

The output of each clustering algorithm is combined using a weighted voting, thereby predicting the most appropriate/accurate class label for the data entry. The combination of clustering algorithms and voting mechanism formulates the proposed ensemble learning model. So, after processing the entire unlabelled dataset by the ensemble learning model, a labelled dataset is generated. This process of converting the unlabelled IoT network dataset into a labelled dataset suffices that any new/unknown IoT network attacks can be also detected by the system.

Due to the limited availability of computational resources in IoT devices, the proposed ensemble learning model can be deployed at a cloud layer and the labelled dataset generated by the system can be used to train a deep learning model for detecting new/unknown attacks in an IoT network.

Hence at step-7 in figure 1 we propose to use a deep learning model, which is to be trained using the system generated labelled network dataset. Finally, the trained deep learning model can be deployed in any IoT device at fog or edge layer of fog computing architecture. The deep learning model can be then updated timely at cloud layer using the proposed ensemble learning model for detecting new/unknown attacks that may be encountered in future.

3.1.4 Unsupervised Learning

Ensemble learning is based on the principle that the combination of outputs from various learning models can produce more accurate results. The ensemble learning model can be implemented in two ways to produce multiple predicted results: Independent ensemble construction and Coordinated ensemble construction. In an independent ensemble construction approach, a learning algorithm can be executed independently multiple times on various training data- subsets or different learning models can be executed independently on the same dataset, to produce multiple results which can then be combined using ensemble technique. Whereas, in coordinated ensemble construction the output of one learning algorithm can be used as an input to another learning algorithm, thus making all the base learning algorithms dependent on each other.

In the proposed model, we have used an independent ensemble construction approach and used weighted voting for combining the output from different base learning models. The main purpose of the proposed ensemble learning model is to predict a class label for each data vector in the given unlabelled dataset. It is because of this we have used clustering techniques for predicting the class labels for data vectors. In the proposed model, we have used Mini Batch K-Means, Fuzzy C-Means and OPTICS (Ordering Points to Identify the Clustering Structure) clustering as the base learning models. The predicted output of each clustering technique will be either 0 or 1, where 0 represents non-malicious traffic (cluster-1) and 1 represents malicious traffic (cluster-2). The predicted output from each clustering algorithm, for each data entry, is combined using weighted voting using equation 3 to create two clusters – one containing benign data and another containing malicious data.

After thorough performance analysis of the clustering technique used in our proposed model, the weights associated with the predicted value from Mini Batch K-Means and OPTICS clustering were set to 0.25 each and for Fuzzy C-Means it was set to 0.5.

3.1.5 Model 1 Details

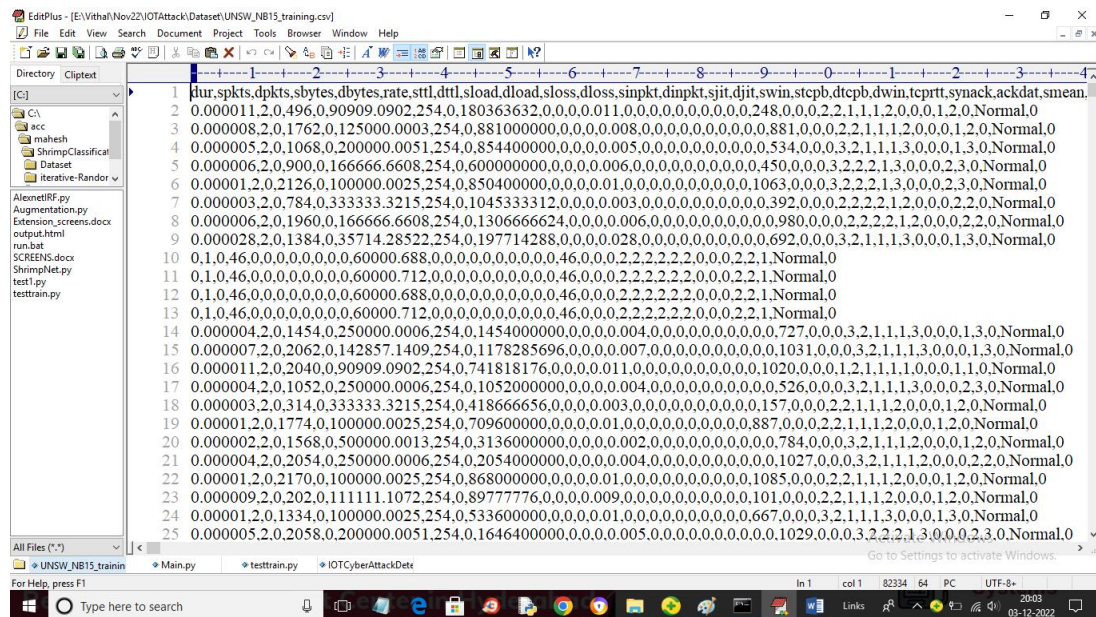


Figure 3.2 : Dataset Details

In the given screen, the first row consists of dataset column names and the remaining rows consist of dataset values. By using the above three datasets we will be able to train all the three clustering and deep learning algorithms.

```

In [1]: #importing python require packages
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score #function to calculate accuracy
from sklearn.decomposition import PCA #PCA for features selection
import matplotlib.pyplot as plt
from sklearn.cluster import MiniBatchKMeans #Loading mini batch kmeans algorithms
from fcmmeans import FCM #Loading fuzzy cmeans algorithm
from sklearn.cluster import OPTICS #Loading optics clustering algorithm
from keras.utils.np_utils import to_categorical
from keras.layers import MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D #class for deep learning convolution neural networks
from keras.models import Sequential
from keras.models import model_from_json
import pickle
from sklearn.model_selection import train_test_split
import os
import seaborn as sns
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from IPython.display import display

Using TensorFlow backend.
c:\users\admin\appdata\local\programs\python\python37\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning:
g: Passing (type, 1) or 'i' type as a synonym of type is deprecated; in a future version of numpy, it will be understood as (typ

```

Figure 3.3 : Necessary Libraries

In the above screen we are importing require python packages and you can read blue colour comments to know about coding.

```

np_resource = np.dtype [("resource", np.ubyte, 1)]

In [2]: #reading and displaying dataset
dataset = pd.read_csv("dataset/iot_data.csv")
display(dataset)

```

	dur	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	dload	...	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cr
0	0.000011	2	0	496	0	99909.090200	254	0	1.803836e+08	0.000000	...	1	2	0	
1	0.000008	2	0	1762	0	125000.000300	254	0	8.810000e+08	0.000000	...	1	2	0	
2	0.000005	2	0	1068	0	200000.005100	254	0	8.544000e+08	0.000000	...	1	3	0	
3	0.000006	2	0	900	0	166666.666800	254	0	6.000000e+08	0.000000	...	1	3	0	
4	0.000010	2	0	2126	0	100000.002500	254	0	8.504000e+08	0.000000	...	1	3	0	
...
82327	0.000005	2	0	104	0	200000.005100	254	0	8.320000e+07	0.000000	...	1	2	0	
82328	1.106101	20	8	18062	354	24.410067	254	252	1.241044e+05	2242.109863	...	1	1	0	
82329	0.000000	1	0	46	0	0.000000	0	0	0.000000e+00	0.000000	...	1	1	0	
82330	0.000000	1	0	46	0	0.000000	0	0	0.000000e+00	0.000000	...	1	1	0	
82331	0.000009	2	0	104	0	111111.107200	254	0	4.622222e+07	0.000000	...	1	1	0	

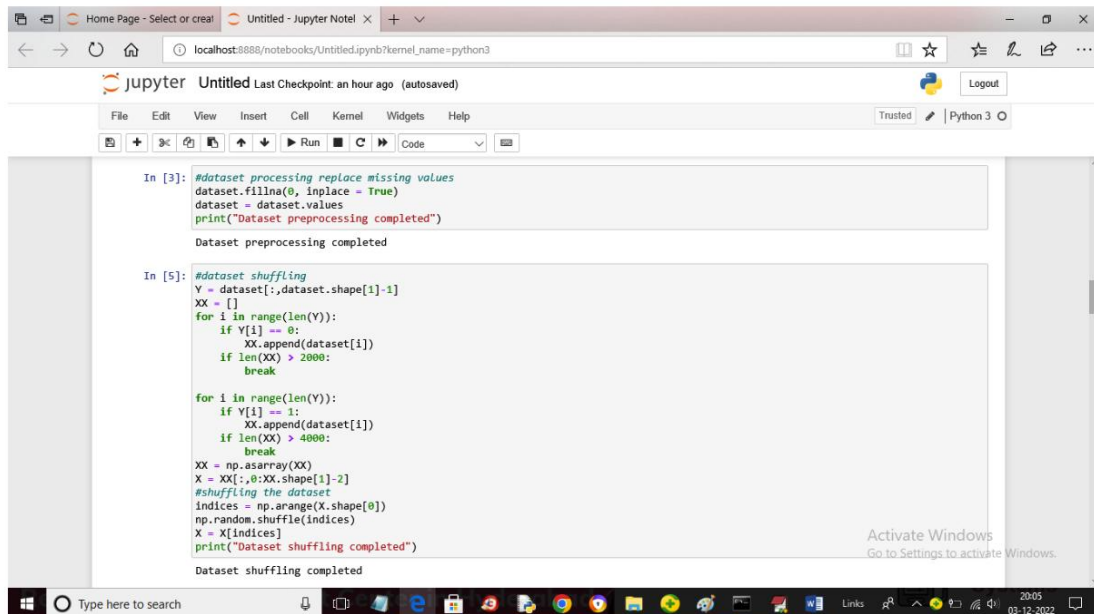
```

82332 rows x 41 columns

In [3]: #dataset processing replace missing values
dataset.fillna(0, inplace = True)

```


Figure 3.4 : Loading Dataset



```
In [3]: #dataset processing replace missing values
dataset.fillna(0, inplace = True)
dataset = dataset.values
print("Dataset preprocessing completed")
Dataset preprocessing completed

In [5]: #dataset shuffling
Y = dataset[:,dataset.shape[1]-1]
XX = []
for i in range(len(Y)):
    if Y[i] == 0:
        XX.append(dataset[i])
        if len(XX) > 2000:
            break

for i in range(len(Y)):
    if Y[i] == 1:
        XX.append(dataset[i])
        if len(XX) > 4000:
            break
XX = np.asarray(XX)
X = XX[:,0:XX.shape[1]-2]
#shuffling the dataset
indices = np.arange(X.shape[0])
np.random.shuffle(indices)
X = X[indices]
print("Dataset shuffling completed")
Dataset shuffling completed
```

Figure 3.5 - Dataset Shuffling

3.2 Model 2

3.2.1 Methodology Used

- Install & Import necessary Libraries..
- Data collection (UNSW_NB15 dataset from kaggle)
- Data Pre-processing
- Data Normalisation
- Applying PCA for feature selection
- Splitting the dataset into training & testing datasets
- Training dataset with LSTM Algorithm
- Training dataset with Bagging Decision Tree
- Training dataset Random Forest Classifier

- Calculating Accuracy_score, Precision_score, Recall, F1_Score, Confusion matrix.
- Implement Accuracy graph and Loss graph

3.2.2 Tools Platforms/Technology/Languages Used

- Imported UNSW_NB15 dataset from kaggle.
- Tools & Platforms used are Python Colaboratory..
- Language Used: Python

3.2.3 Proposed Method

In this Model 2 we will first upload the dataset that we downloaded from kaggle.

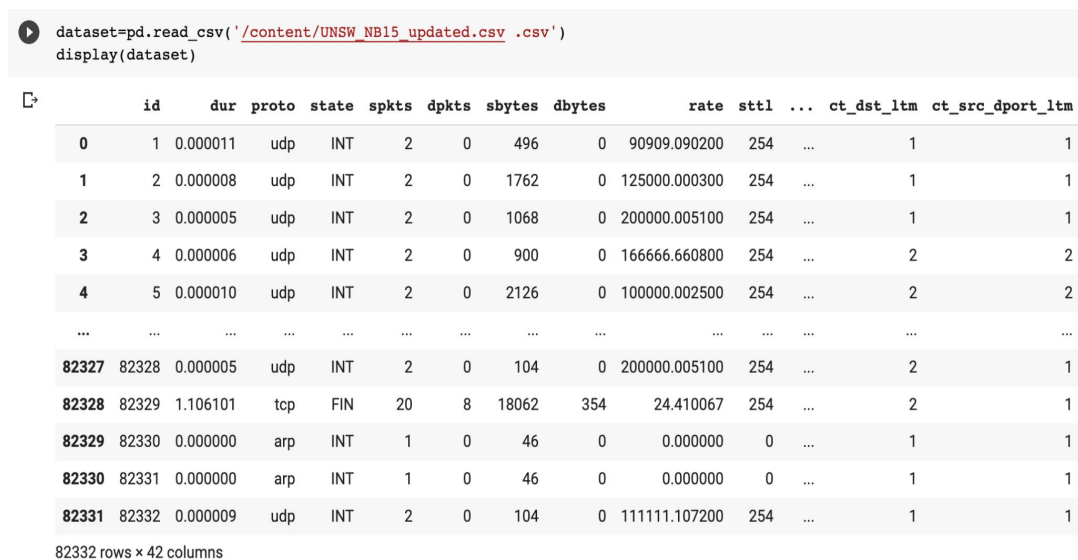


figure 3.6 : Dataset

F

Then we will import all the necessary libraries that should be used to build and run all the algorithms that we will implement in this model.

```

▶ #importing python require packages
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.svm import SVC
from keras.layers import Dense, Dropout, Bidirectional, LSTM
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.cluster import MiniBatchKMeans
from fcmeans import FCM

```

Figure 3.7 : Libraries For Model 2

```

from sklearn.cluster import OPTICS
from sklearn.ensemble import GradientBoostingClassifier
from keras.utils.np_utils import to_categorical
from keras.layers import MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D
from keras.models import Sequential
from keras.models import model_from_json
from keras.models import Sequential
from keras.layers import LSTM, Dense
import tensorflow as tf
from tensorflow.keras import layers
import pickle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import os
import seaborn as sns
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from IPython.display import display
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, Activation
from keras.optimizers import Adam

```

Figure 3.8 : Libraries For Model 2

After Importing these libraries we will preprocess the data.

Steps that we will include in pre processing are -

1. Using Label Encoder we will convert all the string values to integer values.
2. We will replace all the missing values from the dataset.
3. Shuffling of the dataset.
4. Then features will be normalised using standard scalar.
5. Then we will apply PCA to select the main and important features on which the algorithms will work.

```
[ ] #dataset processing replace missing values
dataset.fillna(0, inplace = True)
dataset = dataset.values
print("Dataset preprocessing completed")
```

Dataset preprocessing completed

```
[ ] #dataset shuffling
Y = dataset[:,dataset.shape[1]-1]
XX = []
for i in range(len(Y)):
    if Y[i] == 0:
        XX.append(dataset[i])
    if len(XX) > 2000:
        break

for i in range(len(Y)):
    if Y[i] == 1:
        XX.append(dataset[i])
    if len(XX) > 4000:
        break

XX = np.asarray(XX)
X = XX[:,0:XX.shape[1]-2]
#shuffling the dataset
indices = np.arange(X.shape[0])
np.random.shuffle(indices)
X = X[indices]
print("Dataset shuffling completed")
```

Dataset shuffling completed

Figure 3.9 : Replacing missing values and shuffling dataset

```
[ ] #applying standard scaler to normalized features
scaler = StandardScaler()
X = scaler.fit_transform(X)
print("Features normalization task completed")

Features normalization task completed

[ ] print(X.shape)

(4001, 40)

[ ] #applying PCA for features selection
print("Total features found in dataset before applying PCA : "+str(X.shape[1]))
pca = PCA(n_components = 25)
X = pca.fit_transform(X)
print("Total features found in dataset after applying PCA : "+str(X.shape[1]))

Total features found in dataset before applying PCA : 40
Total features found in dataset after applying PCA : 25
```

Figure 3.10 : Applying PCA for feature selection

After data preprocessing is done we will be applying the following steps :-

- Split the dataset into a training set and testing set.
- Build LSTM Regression Algorithm and train it on a training dataset.
- Test models accuracy, precision, recall, f1 score and confusion matrix on testing dataset.
- Build Bagging Decision Tree Algorithm and train it on a training dataset.
- Test models accuracy, precision, recall, f1 score and confusion matrix on testing dataset.
- Build Random Forest Tree Algorithm and train it on a training dataset.
- Test models accuracy, precision, recall, f1 score and confusion matrix on testing dataset.

3.2.4 Models Used

1. LSTM - Long Short Term Method Algorithm

```
[ ] # split data into training and testing sets
    train_size = int(len(X) * 0.7)
    test_size = len(X) - train_size
    train_data, test_data = X[0:train_size:], X[train_size:len(X),:]
```

```
[ ] # convert data into time series dataset
    def create_dataset(dataset, time_step=1):
        X, Y = [], []
        for i in range(len(dataset)-time_step-1):
            a = dataset[i:(i+time_step), 0]
            X.append(a)
            Y.append(dataset[i + time_step, 0])
        return np.array(X), np.array(Y)

    time_step = 100
    X_train, Y_train = create_dataset(train_data, time_step)
    X_test, Y_test = create_dataset(test_data, time_step)
```

Figure 3.11 : Splitting the dataset into a training and testing dataset

```
[ ] # reshape input to be [samples, time steps, features]
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
[ ] # build LSTM model
    model = Sequential()
    model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
    model.add(LSTM(50, return_sequences=True))
    model.add(LSTM(50))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
```

Figure 3.12 : Building the LSTM Model

```
[ ] # train the model
    model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batch_size=64, verbose=1)
```

```

Epoch 1/10
43/43 [=====] - 10s 136ms/step - loss: 7.2507 - val_loss: 7.1699
Epoch 2/10
43/43 [=====] - 6s 134ms/step - loss: 7.2421 - val_loss: 7.1573
Epoch 3/10
43/43 [=====] - 5s 118ms/step - loss: 7.2411 - val_loss: 7.1591
Epoch 4/10
43/43 [=====] - 6s 136ms/step - loss: 7.2421 - val_loss: 7.1616
Epoch 5/10
43/43 [=====] - 6s 150ms/step - loss: 7.2410 - val_loss: 7.1602
Epoch 6/10
43/43 [=====] - 6s 135ms/step - loss: 7.2359 - val_loss: 7.1613
Epoch 7/10
43/43 [=====] - 5s 118ms/step - loss: 7.2413 - val_loss: 7.1627
Epoch 8/10
43/43 [=====] - 6s 131ms/step - loss: 7.2360 - val_loss: 7.1615
Epoch 9/10
43/43 [=====] - 5s 117ms/step - loss: 7.2350 - val_loss: 7.1660
Epoch 10/10
43/43 [=====] - 5s 119ms/step - loss: 7.2366 - val_loss: 7.1593
<keras.callbacks.History at 0x7ff6de97fca0>

```

Figure 3.13 : Training the model using 10 epochs

2. Bagging Decision Tree Algorithm

```

[ ] # Split your dataset into features and target variable
    X = dataset[:, :-1]
    y = dataset[:, -1]

```

```

[ ] # Split your dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

```

```

[ ] # Create a decision tree classifier
    dt = DecisionTreeClassifier()

```

Figure 3.14 : Splitting the dataset and creating Algorithm

```

[ ] # Create a bagging classifier with 100 decision trees
    bagging = BaggingClassifier(dt, n_estimators=100)

```

Figure 3.15 : Creating Bagging Classifier

```
[ ] # Fit the bagging classifier to your training data
    bagging.fit(X_train, y_train)
```

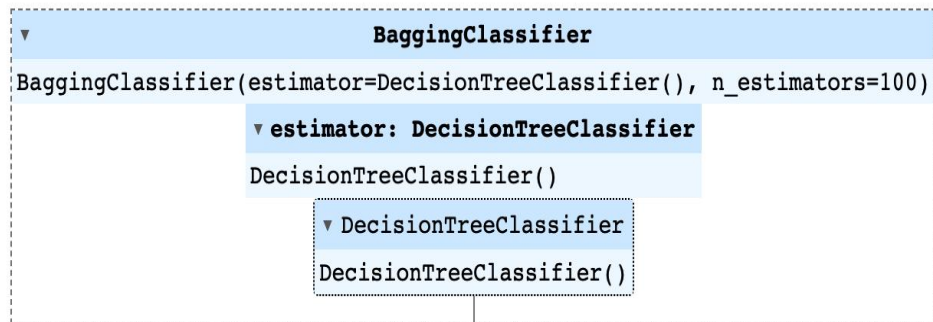


Figure 3.16 - Fitting the Bagging Classifier

3. Random Forest Algorithm

```
[ ] # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[ ] # Define the model
    model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
```

Figure 3.17 : Splitting the dataset and defining the model

```
[ ] # Fit the model to the training data
    model.fit(X_train, y_train)
```

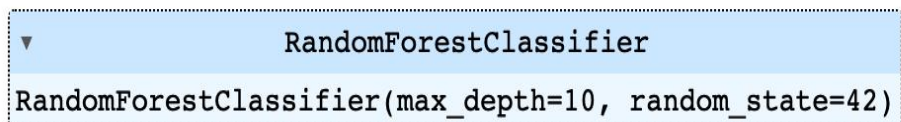


Figure 3.18 : Fitting the model

CHAPTER - 4

PERFORMANCE ANALYSIS

4.1 Performance of Model 1

The following results were produced :

- **Count of 0's and 1's after Applying Mini Batch algorithm**

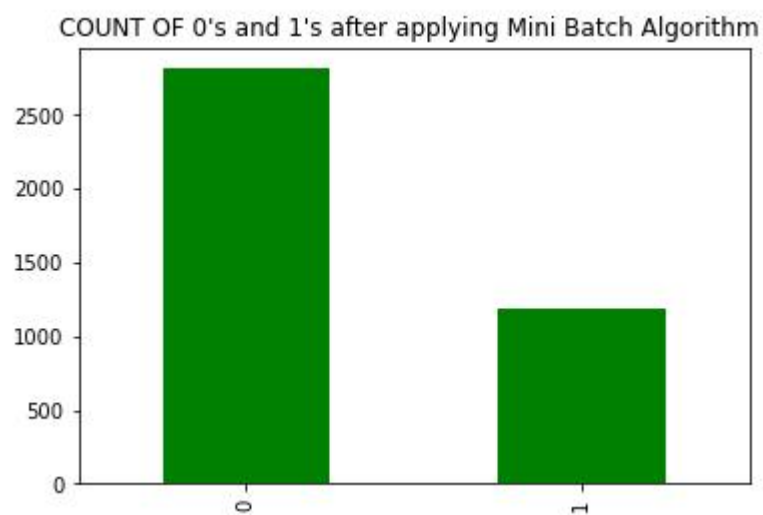


Figure 4.1 : Result of Mini Batch Algorithm

- **Count of 0's and 1's after Applying Fuzzy C means algorithm**

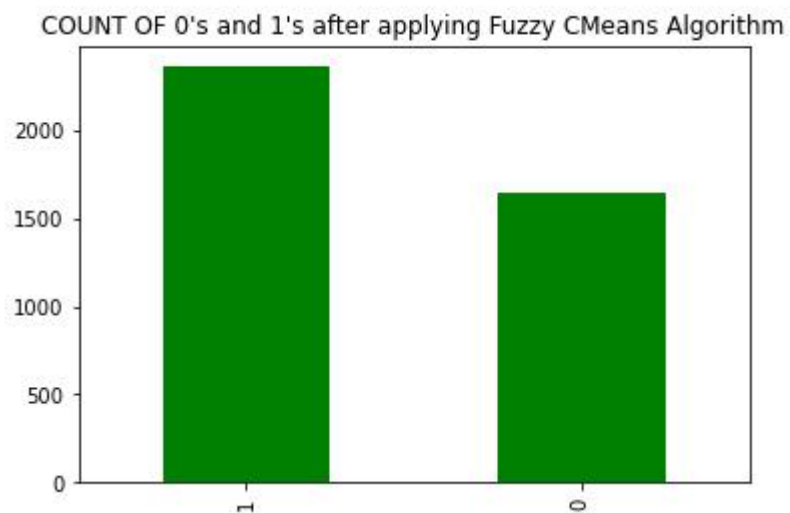


Figure 4.2 : Result of Fuzzy CMeans Algorithm

- **Count of 0's and 1's after Applying OPTICS algorithm**

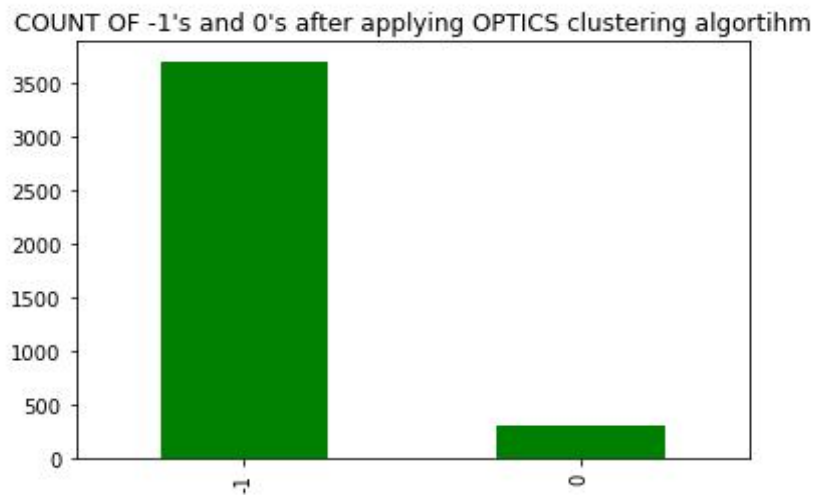


Figure 4.3 : Result of OPTICS Algorithm

- **LOSS v/s EPOCH graph for training Dataset**



Figure 4.4 : Loss Graph for training dataset

- **LOSS v/s EPOCH graph for testing Dataset**

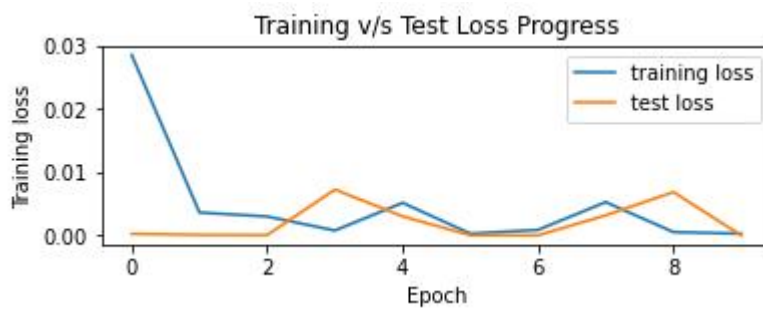


Figure 4.5 : Loss graph for testing dataset

- **Confusion matrix for training Dataset**

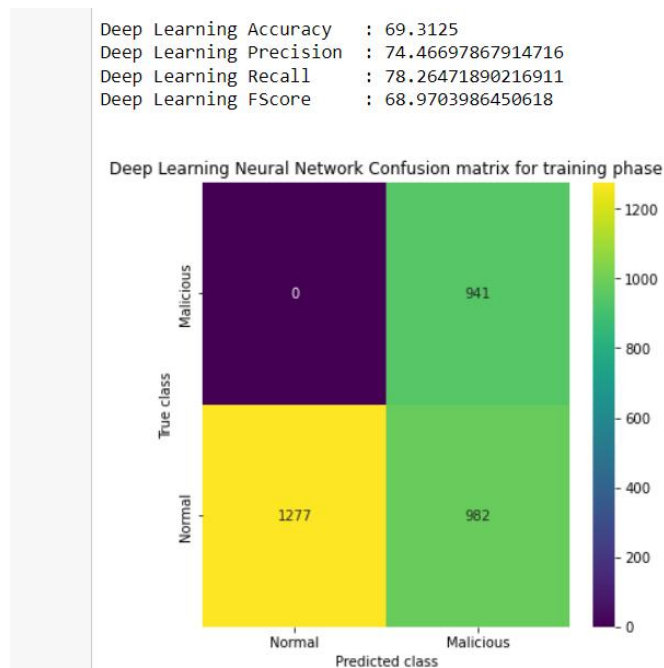


Figure 4.6 : Confusion Matrix 1

- **Confusion matrix for training Dataset**

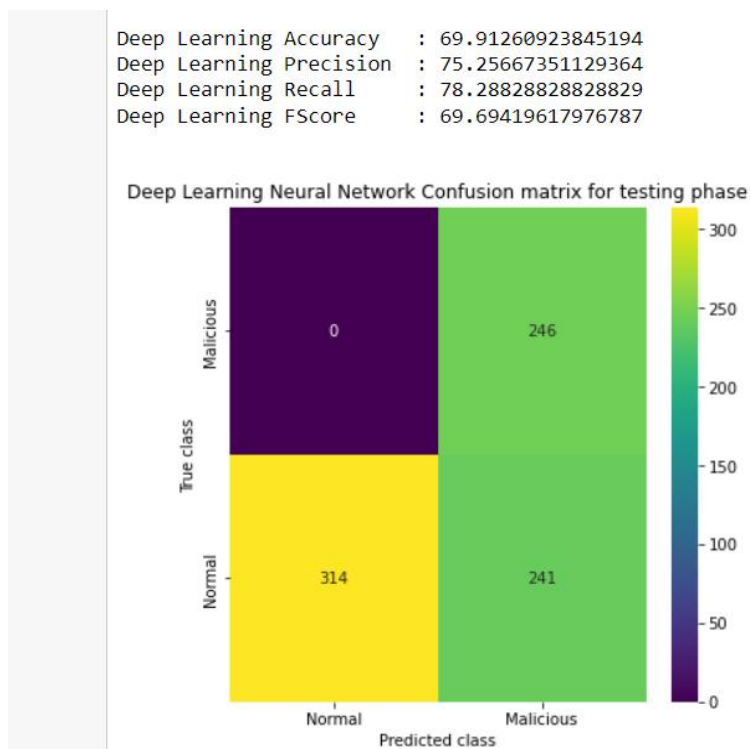


Figure 4.7 Confusion Matrix 2

4.2 Performance of Model 2

The following results were produced :-

- **RMSE_Score of LSTM Model**

```
[ ] # Predict the output values for the test set
    y_pred = model.predict(X_test)

    # Calculate the RMSE of the predictions
    from sklearn.metrics import mean_squared_error
    rmse = np.sqrt(mean_squared_error(Y_test, y_pred))

    # Print the RMSE
    print("RMSE:", rmse)

35/35 [=====] - 1s 25ms/step
RMSE: 2.6756898939925904
```

Figure 4.8 : RSME Score

- **Accuracy_Score, Precision_Score, Recall, F1_Score, Confusion Matrix of Bagging Decision Tree Algorithm**

```
[ ] # Print the evaluation metrics
    print("Accuracy score:", accuracy)
    print("Precision score:", precision)
    print("Recall score:", recall)
    print("F1 score:", f1)
    print("Confusion matrix:\n", cm)

Accuracy score: 0.9995141798749013
Precision score: 0.9994448762073942
Recall score: 0.9996668517490284
F1 score: 0.9995558516544526
Confusion matrix:
[[7457    5]
 [    3 9002]]
```

Figure 4.9 : Evaluation Metrics of Bagging Decision tree

- **Confusion Matrix of Bagging Decision Tree Algorithm**

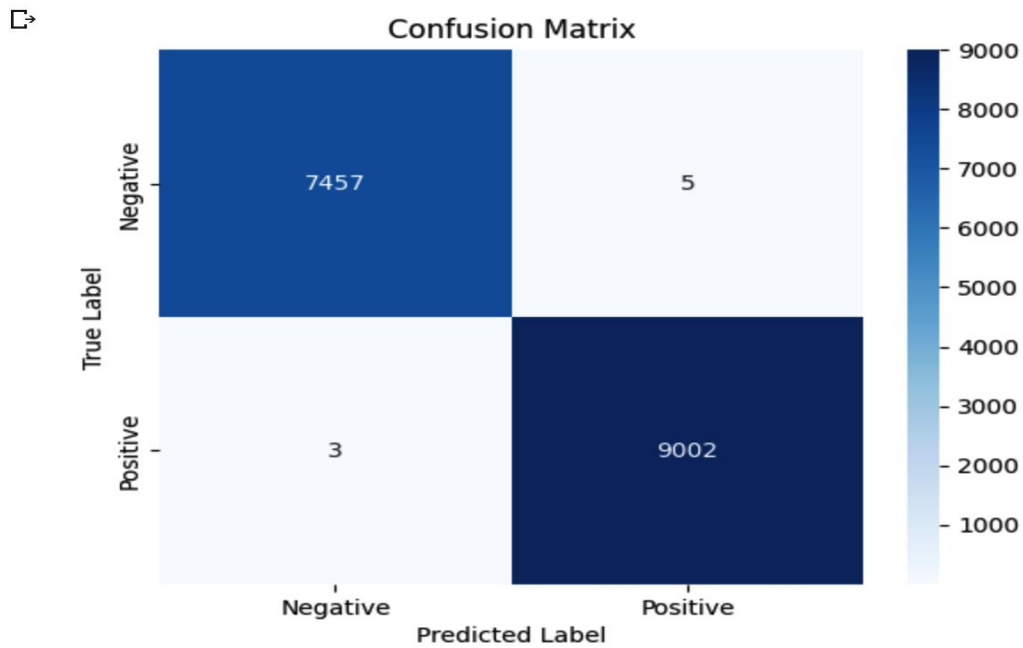


Figure 4.10 : Evaluation Metrics of bagging Decision Tree

- **Accuracy Graph of Testing and training Sets of Bagging Decision Tree Algorithm**

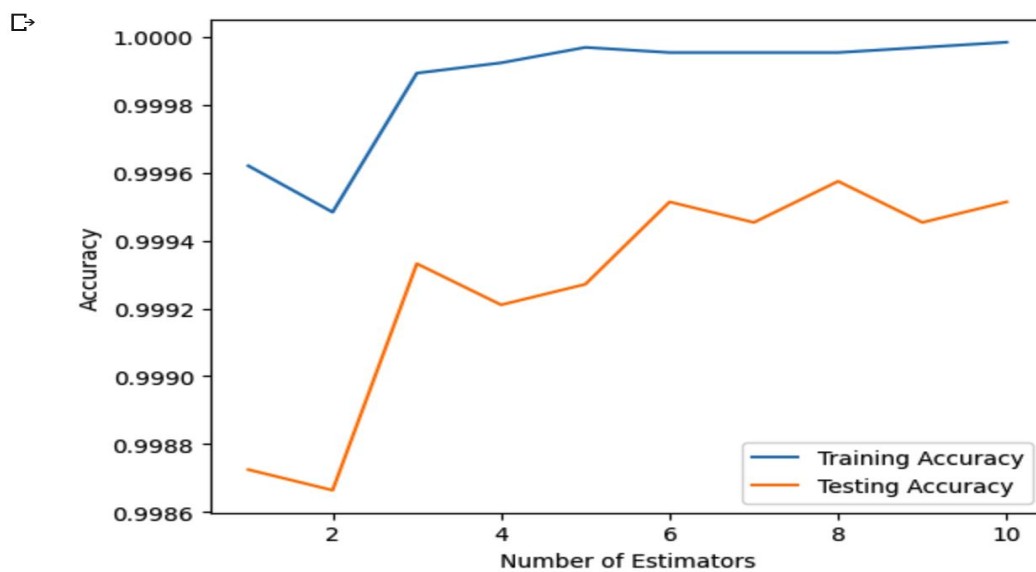


Figure 4.11 : Accuracy graph of Bagging Decision Tree

- **Loss Graph of training and testing sets of Bagging Decision Tree Algorithm**

↳

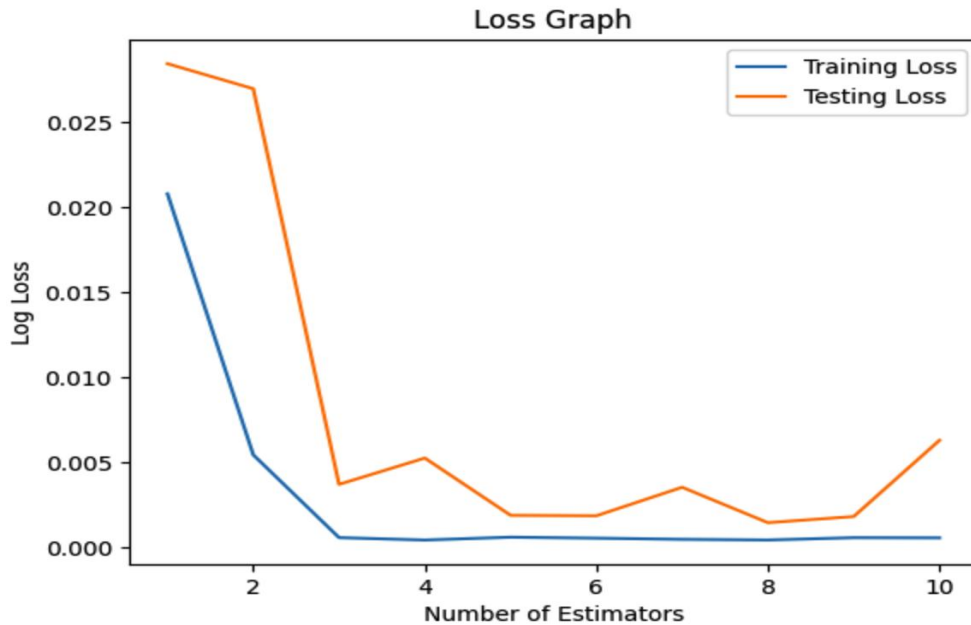


Figure 4.12 : Loss graph of Bagging Decision Tree

- **Accuracy_Score, Precision_Score, Recall, F1_Score, Confusion Matrix of Random Forest Classifier Algorithm**

```
[ ] # Print the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:\n", cm)
```

```
Accuracy: 0.885
Precision: 0.9230769230769231
Recall: 0.8648648648648649
F1 Score: 0.8930232558139535
Confusion Matrix:
[[81  8]
 [15 96]]
```

Figure 4.13 : Evaluation Metrics of random Forest Classifier

- **Confusion Matrix of Random Forest Algorithm**

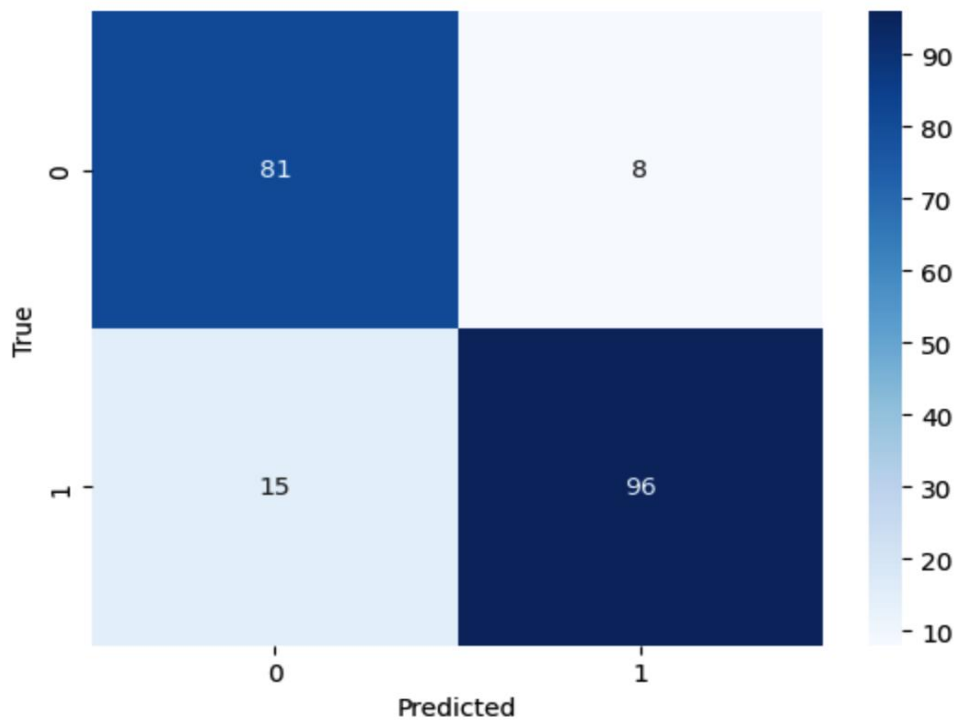


Figure 4.14 : confusion matrix of Random Forest Algorithm

- **Accuracy Graph of Testing and training Sets of Random Forest Algorithm**

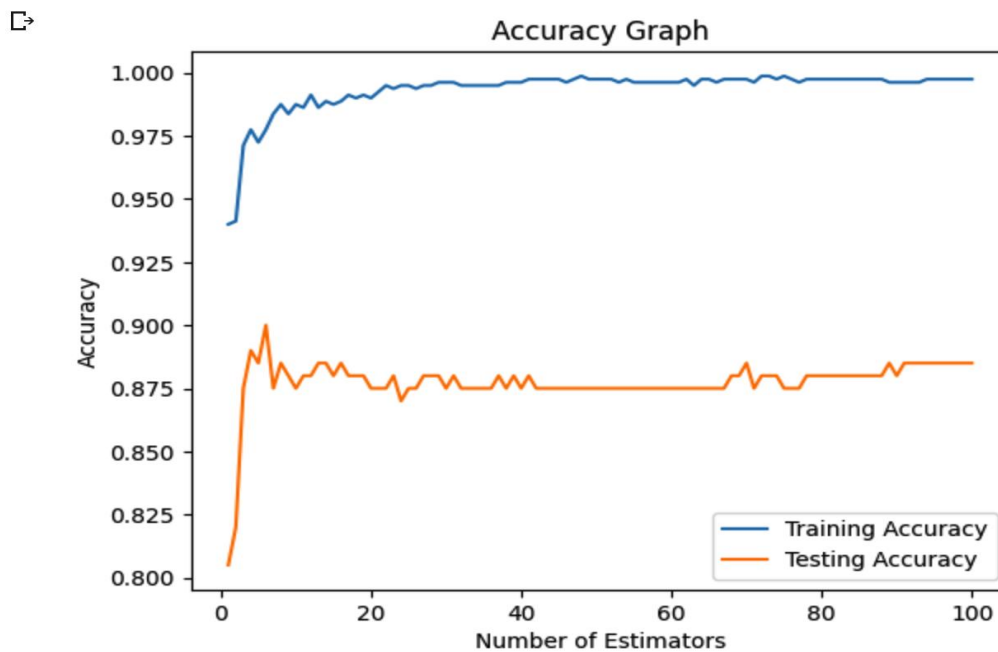


Figure 4.15 : Accuracy graph of Random Forest Classifier

- **Loss Graph of training and testing sets of Random Forest Algorithm**

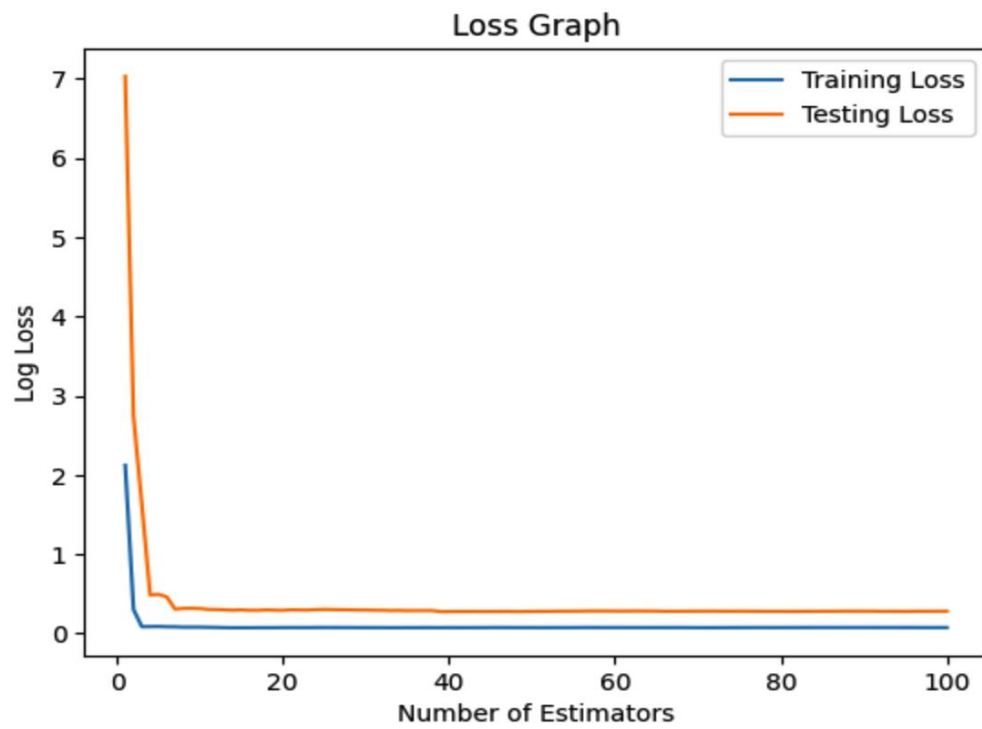


Figure 4.16 : Loss graph of Random Forest Classifier

CHAPTER - 5

CONCLUSION

5.1 Conclusion

Globally, the Internet of Things (IoT) has experienced exponential development. Despite the fact that the IoT is adopted by millions of people, attacks like man in the middle, spoofing, and denial of service make it difficult for these networks to function. The privacy and security of the consumer are compromised by these cyberattacks, which also jeopardise the entire IoT ecosystem. Therefore, it is still difficult for researchers to forecast and identify new network assaults inside an IoT network. The proposed framework is used to identify what attack has been done. The dataset was picked using “Kaggle”, and data preprocessing was done. Then, various algorithms like:

Model 1

- Mini Batch
- FUZZY C MEANS
- Optical Clustering

Model 2

- LSTM
- Bagging Decision Tree
- Random Forest Classifier

has been applied and their accuracy was compared. Later, using the first three algorithms an ensemble model was built to achieve higher accuracy. And accuracy and some other evaluation parameters were compared of the other three models.

5.2 Future Scope

- By analysing the unlabeled IoT network activity, the suggested unsupervised ensemble-based learning model can be implemented at the cloud layer to detect novel/unknown assaults.
- It can be implemented using cloud computing architecture.
- The suggested model's network traffic labelling may then be utilised to train the DBN deep learning model to recognise network attacks.
- The trained DBN may then be used at the fog node to analyse network traffic from edge devices and spot attacks, with constant updates just at cloud layer to spot fresh assaults.
- The taught DBN may then be used at the fog layer to monitor the network activity of edge devices and find attacks, with retraining at the cloud layer as needed to spot new assaults.
- To further examine the effectiveness and complexity of the suggested model, we will deploy it on a real-world Internet of Things network using a fog - based architecture in the future.

5.3 Application Contribution

We proposed an unsupervised Ensemble Learning model which will identify future unidentified attacks on an IOT Network. A deep learning model is trained to recognise attacks on Iot Network. The model is trained using the different and best algorithms to detect the future attacks. First the PCA was used to train the model to distinguish the attacks and non attacks like 0 and 1 (0's are no attack and 1's are attacks). Then Mini Batch Algorithm is used to train the model and after this algorithm two more algorithms are used to increase the accuracy of the model and those two algorithms are Fuzzy C MEANS and Optical Clustering. After applying all these algorithms on the dataset the data points are divided into 0's and 1's and then the model will make confusion matrices to show all the values and the model will make the accuracy and loss graphs.

CHAPTER - 6

REFERENCES

- [1] Javed F, Afzal MK, Sharif M, Kim BS. Internet of Things (IoT) operating systems support, networking technologies, applications, and challenges: comparative review. *IEEE Communications Surveys & Tutorials* 2018; 20(3): 2062-2100.

- [2] Ciunzo D, Rossi PS, Varshney PK. Distributed detection in wireless sensor networks under multiplicative fading via generalised score tests. *IEEE Internet of Things Journal* 2021; 8(11): 9050-9071.

- [3] Meneghello F, Calore M, Zucchetto D, Polese M, Zanella A. IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices. *IEEE Internet of Things Journal* 2019; 6(5): 8182-8201.

- [4] Hassan WH, others. Current research on Internet of Things (IoT) security: A survey. *Computer networks* 2019; 148: 283-294.

- [5] T.seals. "Wicked Botnet Uses Passel of Exploits to Target ToT", May 2018.

- [6] S. Bandyopadhyay et al. "Internet of Things : Applications and Challenges in Technology and Standardization". In *Wireless Personal Communication*, volume 58(1), pages 49-69, 2011.

- [7] NSL-KDD dataset. <https://www.unb.ca/cic/datasets/nsl.html> (accessed: 02.08.2020).

- [8] N. Moustafa, TON_IOT Datasets, 2019
(online) <http://dx.doi.org/10.21227/fesz-dm97> (accessed: 02.08.2020).

- [9] Gunupudi RK, Nimmala M, Gugulothu N, Gali SR. CLAPP: A self constructing feature clustering approach for anomaly detection. *Future Generation Computer Systems* 2017; 74: 417-429.
- [10] Su T, Sun H, Zhu J, Wang S, Li Y. BAT: Deep learning methods on network intrusion detection using NSL-KDDdataset. *IEEE Access* 2020; 8: 29575-29585.
- [11] Souza dCA, Westphall CB, Machado RB, Sobral JBM, Santos Vieira dG. Hybrid approach to intrusion detection in fog-based IoT environments. *Computer Networks* 2020; 180: 107417.
- [12] Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the KDD CUP 99 dataset. In: *IEEE.*; 2009: 1-6.
- [13] Venkatraman S, Surendiran B. Adaptive hybrid intrusion detection system for crowd sourced multimedia internet of things systems. *Multimedia Tools and Applications* 2020; 79(5): 3993-4010.
- [14] Ahmad MS, Shah SM. Mitigating Malicious Insider Attacks in the Internet of Things using Supervised Machine Learning Techniques. *Scalable Computing: Practice and Experience* 2021; 22(1): 13-28.
- [15] Bovenzi G, Aceto G, Ciuonzo D, Persico V, Pescapé A. A hierarchical hybrid intrusion detection approach in IoT scenarios. In: *IEEE.* ; 2020: 1-7.
- [16] Sahay R, Geethakumari G, Mitra B. A novel blockchain based framework to secure IoT-LLNs against routing attacks. *Computing* 2020; 102(11): 2445-2470.
- [17] Veeramakali T, Siva R, Sivakumar B, Senthil Mahesh P, Krishnaraj N. An intelligent internet of things-based secure healthcare framework using blockchain technology with an optimal deep learning model. *The Journal of Supercomputing* 2021; 77(9): 9576-9596.

- [18] Babu MJ, Reddy AR. SH-IDS: Specification Heuristics Based Intrusion Detection System for IoT Networks. *Wireless Personal Communications* 2020; 1-23.
- [19] Makkar A, Kumar N. An efficient deep learning-based scheme for web spam detection in the IoT environment. *Future Generation Computer Systems* 2020; 467-487.
- [20] Eskandari M, Janjua ZH, Vecchio M, Antonelli F. Pasban IDS: an intelligent anomaly-based intrusion detection system for IoT edge devices. *IEEE Internet of Things Journal* 2020; 7(8): 6882-6897.
- [21] Verma A, Ranga V. Machine learning based intrusion detection systems for IoT applications. *Wireless Personal Communications* 2020; 111(4): 2287-2310.
- [22] Sahu AK, Sharma S, Tanveer M, Raja R. Internet of Things attack detection using hybrid Deep Learning Model. *Computer Communications* 2021.
- [23] Fotohi R, Pakdel H. A Lightweight and Scalable Physical Layer Attack Detection Mechanism for the Internet of Things (IoT) using Hybrid Security Schema. *Wireless Personal Communications* 2021: 1-18.
- [24] Ambusaidi MA, He X, Nanda P, Tan Z. Building an intrusion detection system using a filter-based feature selection algorithm. *IEEE transactions on computers* 2016; 65(10): 2986-2998.
- [25] Shone N, Ngoc TN, Phai VD, Shi Q. A deep learning approach to network intrusion detection. *IEEE transactions on emerging topics in computational intelligence* 2018; 2(1):41-50.
- [26] Meidan Y, Bohadana M, Mathov Y, et al. N-baiot–network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing* 2018; 17(3): 12-22.

CHAPTER - 7

APPENDICES

• Code for Dataset Pre Processing

```
In [140]: #importing python require packages
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score #function to calculate accuracy
from sklearn.decomposition import PCA #PCA for features selection
import matplotlib.pyplot as plt
from sklearn.cluster import MiniBatchKMeans #Loading mini batch kmeans algorithms
from fcmeans import FCM #Loading fuzzy cmeans algortihm
from sklearn.cluster import OPTICS #Loading optics clustering algorithm
from keras.utils.np_utils import to_categorical
from keras.layers import MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D #class for deep learning convolution neural networks
from keras.models import Sequential
from keras.models import model_from_json
import pickle
from sklearn.model_selection import train_test_split
import os
import seaborn as sns
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from IPython.display import display
```

Figure 7.1 : Import and Install Dependencies

```
In [144]: #applying standard scaler to normalized features
scaler = StandardScaler()
X = scaler.fit_transform(X)
print("Features normalization task completed")

Features normalization task completed

In [145]: print(X.shape)

(4001, 39)

In [146]: #applying PCA for features selection
print("Total features found in dataset before applying PCA : "+str(X.shape[1]))
pca = PCA(n_components = 25)
X = pca.fit_transform(X)
print("Total features found in dataset after applying PCA : "+str(X.shape[1]))

Total features found in dataset before applying PCA : 39
Total features found in dataset after applying PCA : 25
```

Figure 7.2 : Normalisation and Feature selection

- **Code for Implementation of Model 1**

```
In [147]: #running Mini Batch Algorithm
kmeans = MiniBatchKMeans(n_clusters=2, random_state=0, batch_size=6)
kmeans.fit(X)
kmeans_label = kmeans.predict(X)
print("Kmeans cluster labels where 0 means Normal and 1 means Malicious attack")
print(kmeans_label)

C:\Users\new\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1043: UserWarning:
y leak on Windows with MKL, when there are less chunks than available threads. You c
or by setting the environment variable OMP_NUM_THREADS=1
  warnings.warn(

Kmeans cluster labels where 0 means Normal and 1 means Malicious attack
[0 1 0 ... 0 0 0]
```

Figure 7.3 : Clustering into Normal/Malicious using Mini Batch Algorithm

```
In [149]: #implementing Fuzzy CMeans Algorithm
fcm = FCM(n_clusters = 2)
fcm.fit(X)
centers = fcm.centers
fcm_label = fcm.predict(X)
print("FCM cluster labels where 0 means Normal and 1 means Malicious attack")
print(fcm_label)

FCM cluster labels where 0 means Normal and 1 means Malicious attack
[0 1 0 ... 0 0 1]
```

Figure 7.4 : Implementation of Fuzzy C MEANS

```
In [151]: #implementing OPTICS clustering algortihm
optics = OPTICS(max_eps=25, min_samples=100, xi=0.1)
optics.fit(X)
optics_label = optics.labels_
print("Optics cluster labels where -1 means Normal and 0 means Malicious attack")
print(optics_label)

Optics cluster labels where -1 means Normal and 0 means Malicious attack
[-1 -1 -1 ... -1 -1 -1]
```

Figure 7.5 : Implementation of OPTICS

```
In [153]: #function for weighted voting between all 3 clustering algorithm and label w
#for deep learning algorithm
def weightedVoting(kmeans, fcm, optics):
    weighted_label = []
    for i in range(len(kmeans)):
        label = [kmeans[i], fcm[i], optics[i]]
        label = max(label, key=label.count)
        weighted_label.append(label)
    return np.asarray(weighted_label)
```

```
In [154]: #get weighted label for unlabeled dataset
weighted_label = weightedVoting(kmeans_label, fcm_label, optics_label)
print("Selected Weighed Voting Labels")
print(weighted_label)
```

```
Selected Weighed Voting Labels
[0 1 0 ... 0 0 0]
```

Figure 7.6 : Weighted Voted

```
In [156]: #now 80% training will be used to train deep learning model and 20% will be applied on trained model to calculate
#prediction accuracy
if os.path.exists('model/model.json'): #Load the model if already trained
    with open('model/model.json', "r") as json_file:
        loaded_model_json = json_file.read()
        dl = model_from_json(loaded_model_json)
        dl.load_weights("model/model_weights.h5")
        dl.make_predict_function()
else: #if model not trained then start training
    #defining deep learning object
    dl = Sequential()
    #defining neural network layer with 32 filters of kernel size 1 X 1. This layer filter data 32 times
    dl.add(Convolution2D(32, 1, 1, input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[2]), activation = 'relu'))
    #max pooling will collect relevant features from filtered data
    dl.add(MaxPooling2D(pool_size = (1, 1)))
    #adding another layer with more 32 layers
    dl.add(Convolution2D(32, 1, 1, activation = 'relu'))
    #collect the filtered data
    dl.add(MaxPooling2D(pool_size = (1, 1)))
    #convert multi dimension data to single dimension
    dl.add(Flatten())
    #defining output layer of size 256
    dl.add(Dense(output_dim = 256, activation = 'relu'))
    #defining y label as the prediction output
    dl.add(Dense(output_dim = y_train.shape[1], activation = 'softmax'))
    #now compile the model
    dl.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
    #now start training model and then saved the model
    hist = dl.fit(X_train, y_train, batch_size=8, epochs=10, shuffle=True, verbose=2, validation_data=(X_test, y_test))
    dl.save_weights('model/model_weights.h5')
    model_json = dl.to_json()
    with open("model/model.json", "w") as json_file:
        json_file.write(model_json)
    f = open('model/history.pkl', 'wb')
    pickle.dump(hist.history, f)
    f.close()
    f = open('model/history.pkl', 'rb')
print()
print("Deep Learning CNN model training completed and below is the model architecture")
print(dl.summary())
```

Figure 7.7 : Implementation of CNN

- **Code for Result and Analysis of Model 1**

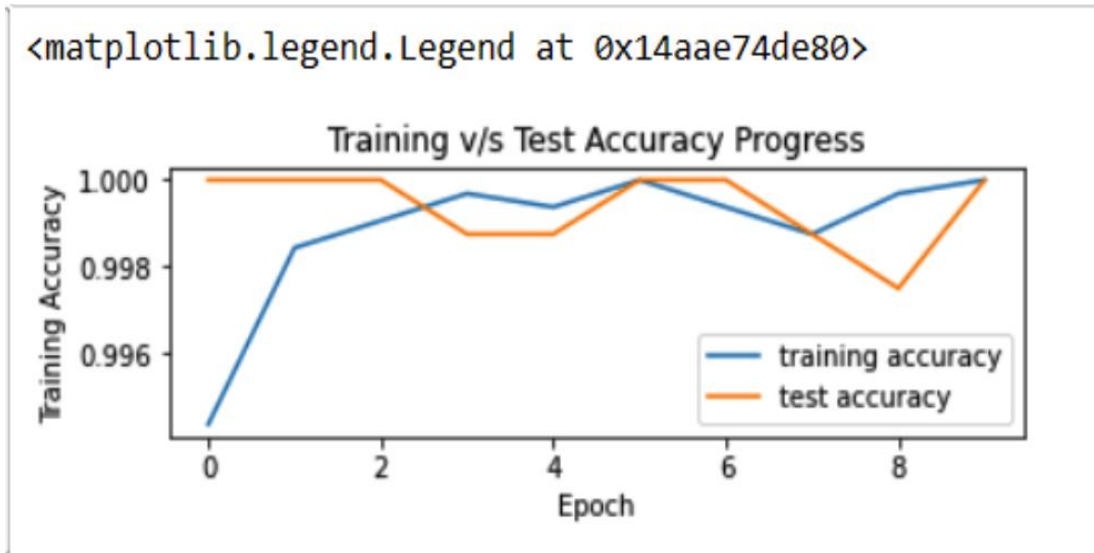


Figure 7.8 :Accuracy graph

```
plt.subplot(2,1,1)
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('Training v/s Test Loss Progress')
plt.ylabel('Training loss')
plt.xlabel('Epoch')
plt.legend(['training loss', 'test loss'], loc='upper right')
```

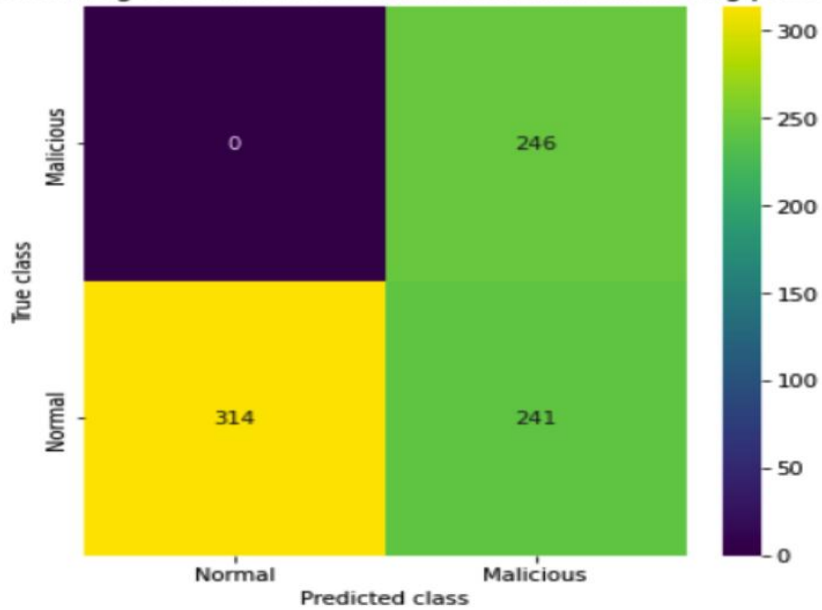
```
<matplotlib.legend.Legend at 0x14aae782280>
```



Figure 7.9 : Loss graph

Deep Learning Accuracy : 69.91260923845194
Deep Learning Precision : 75.25667351129364
Deep Learning Recall : 78.28828828828829
Deep Learning FScore : 69.69419617976787

Deep Learning Neural Network Confusion matrix for testing phase



Deep Learning Accuracy : 69.91260923845194
Deep Learning Precision : 75.25667351129364
Deep Learning Recall : 78.28828828828829
Deep Learning FScore : 69.69419617976787

Deep Learning Neural Network Confusion matrix for testing phase

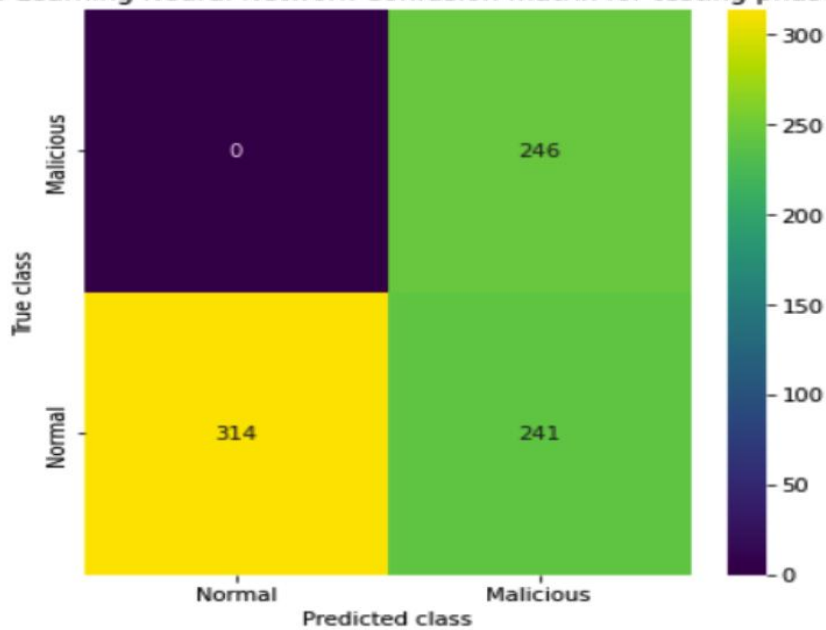


Figure 7.10, Figure 7.11 : Confusion matrix

• Implementation of Model 2

```
[ ] # reshape input to be [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

[ ] # build LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(50, return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

Figure 7.12 : LSTM Impl.

```
[ ] # Split your dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

[ ] # Create a decision tree classifier
dt = DecisionTreeClassifier()

[ ] # Create a bagging classifier with 100 decision trees
bagging = BaggingClassifier(dt, n_estimators=100)

[ ] # Fit the bagging classifier to your training data
bagging.fit(X_train, y_train)
```

Figure 7.13 : Bagging Decision Tree Impl.

```
[ ] # Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[ ] # Define the model
model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)

[ ] # Fit the model to the training data
model.fit(X_train, y_train)
```

Figure 7.14 : Random Forest Classifier Impl.

- **Code for Results and analysis of Model 2**

```
[ ] # Predict the output values for the test set
    y_pred = model.predict(X_test)

    # Calculate the RMSE of the predictions
    from sklearn.metrics import mean_squared_error
    rmse = np.sqrt(mean_squared_error(Y_test, y_pred))

    # Print the RMSE
    print("RMSE:", rmse)

35/35 [=====] - 1s 25ms/step
RMSE: 2.6756898939925904
```

Figure 7.15 : LSTM Result

```
[ ] # Calculate the evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
```

```
[ ] # Print the evaluation metrics
    print("Accuracy score:", accuracy)
    print("Precision score:", precision)
    print("Recall score:", recall)
    print("F1 score:", f1)
    print("Confusion matrix:\n", cm)
```

Figure 7.16 : Evaluation of Bagging Decision Tree

```
[ ] # Calculate the accuracy for the training and testing sets
train_accuracy = []
test_accuracy = []
for i in range(1, 11):
    bagging.set_params(n_estimators=i)
    bagging.fit(X_train, y_train)
    train_accuracy.append(bagging.score(X_train, y_train))
    test_accuracy.append(bagging.score(X_test, y_test))

[ ] # Plot the accuracy graph
plt.plot(range(1, 11), train_accuracy, label="Training Accuracy")
plt.plot(range(1, 11), test_accuracy, label="Testing Accuracy")
plt.xlabel("Number of Estimators")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

Figure 7.17 : Accuracy graph for Bagging Decision Tree

```
[ ] # Calculate the loss for the training and testing sets
train_loss = []
test_loss = []
for i in range(1, 11):
    bagging.set_params(n_estimators=i)
    bagging.fit(X_train, y_train)
    train_loss.append(log_loss(y_train, bagging.predict_proba(X_train)))
    test_loss.append(log_loss(y_test, bagging.predict_proba(X_test)))

[ ] # Plot the loss graph
plt.plot(range(1, 11), train_loss, label="Training Loss")
plt.plot(range(1, 11), test_loss, label="Testing Loss")
plt.xlabel("Number of Estimators")
plt.ylabel("Log Loss")
plt.title("Loss Graph")
plt.legend()
plt.show()
```

Figure 7.18 : Loss graph for Bagging Decision Tree

```
[ ] # Print the results
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)
    print("Confusion Matrix:\n", cm)
```

Figure 7.19 : Results for Random Forest Classifier

```
[ ] # Calculate the accuracy for the training and testing sets
    train_accuracy = []
    test_accuracy = []
    for i in range(1, 101):
        model.set_params(n_estimators=i)
        model.fit(X_train, y_train)
        train_accuracy.append(accuracy_score(y_train, model.predict(X_train)))
        test_accuracy.append(accuracy_score(y_test, model.predict(X_test)))
```

```
[ ] # Plot the accuracy graph
    plt.plot(range(1, 101), train_accuracy, label="Training Accuracy")
    plt.plot(range(1, 101), test_accuracy, label="Testing Accuracy")
    plt.xlabel("Number of Estimators")
    plt.ylabel("Accuracy")
    plt.title("Accuracy Graph")
    plt.legend()
    plt.show()
```

Figure 7.20 : Accuracy graph for Random Forest Classifier

```
[ ] from sklearn.metrics import log_loss
```

```
[ ] # Calculate the loss for the training and testing sets
    train_loss = []
    test_loss = []
    for i in range(1, 101):
        model.set_params(n_estimators=i)
        model.fit(X_train, y_train)
        train_loss.append(log_loss(y_train, model.predict_proba(X_train)))
        test_loss.append(log_loss(y_test, model.predict_proba(X_test)))
```

Figure 7.21 : Calculate Loss graph for Random Forest Classifier

```
[ ] # Plot the loss graph
plt.plot(range(1, 101), train_loss, label="Training Loss")
plt.plot(range(1, 101), test_loss, label="Testing Loss")
plt.xlabel("Number of Estimators")
plt.ylabel("Log Loss")
plt.title("Loss Graph")
plt.legend()
plt.show()
```

Figure 7.22 : Print Loss graph for Random Forest Classifier

