EFFECTIVE INTRUSION DETECTION SYSTEM USING HYBRID ENSEMBLE METHOD FOR CLOUD COMPUTING

PROJECT REPORT

PROJECT GUIDE: Dr S Bose

Submitted by

Araventh M - 2019103508

Gangaraju Tharun - 2019103520

Sourabh Sonny - 2019103064

Raj Kumar J - 2019103564

for the course

CS6811 – FINAL YEAR PROJECT

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY, CHENNAI 600 025

TABLE OF CONTENTS

CHAPTER 1
PROBLEM STATEMENT

Cloud computing is often used by organizations to store data and applications, so it is important to ensure that the platform is secure. While the cloud environment offers many benefits, there are also several security challenges that organizations face when using cloud services. Data breaches are a significant concern for cloud service providers, as sensitive data stored in the cloud can be vulnerable to theft or unauthorized access. **DoSS attacks can cause significant disruptions** to cloud services by overloading the system with traffic. This can prevent legitimate users from accessing cloud resources, causing significant financial and reputational damage to the cloud service provider. Organizations may face compliance and regulatory issues when using cloud services, as sensitive data stored in the cloud may be subject to specific regulatory requirements. Cloud service providers may not provide complete transparency about their security measures, making it difficult for organizations to assess the risks associated with using cloud services.

Intrusion Detection System (IDS) is an important security tool in cloud environments that can detect and alert users to potential security breaches or threats. IDS can monitor network traffic, system logs, and user activity to detect and identify potential security incidents or attacks. There is a **constant need for optimization of algorithms and enhancement in IDS due to the increasing threats of attacks on cloud.**

CHAPTER 2
INTRODUCTION

Cloud computing seems to be one of the most emerging technologies in recent years. Although the number of cloud projects has dramatically increased over the last few years, ensuring the availability and security of project data, services, and resources is still a crucial and challenging research issue. Attacks can exhaust the cloud's resources, consume most of its bandwidth, and damage an entire cloud project within a short period of time. Cloud computing is the preferred choice of every organization since it provides flexible and pay per use-based services to its users. However, the security and privacy is a major hurdle in its success because of its open and distributed architecture that is vulnerable to intruders. Cloud computing is made up of three abstract layers: the system layer, the platform layer, and the application layer. Whereas the **first two layers are concerned with Virtual Machines (VM) and operating systems, the last layer covers cloud-based applications** such as web-based apps. The cloud architecture is made up of two distinct components.

One of these is cloud security, which is regarded as a key barrier to cloud adoption by the majority of organizations. This is due to the open and fully dispersed nature of the cloud environment, which makes it more vulnerable to security threats and vulnerabilities. As a result, intruders are encouraged to conduct possible attacks against the cloud or against devices within the cloud. So, an intrusion detection system is needed in order to protect the data.

An Intrusion Detection System (IDS) is a type of technology used to detect malicious activity on a computer or network. It can be used to monitor the network/cloud for suspicious activities and can alert administrators when such activities are detected. The system may also take preventive measures to protect the network from further damage. There are three widely used methods**: hybrid detection, anomaly-based detection**, and signature-based detection. By comparing the acquired data with a database of patterns of

well-known assaults or a pre-set set of criteria, signature-based detection, also known as misuse detection, locates intrusions. The latter method's primary flaw is that it can only identify known attacks. Anomaly-based detection, on the other hand, finds intrusions by comparing the gathered data to a pre-determined baseline. When behavior deviates from the norm; it raises the warning that a hostile assault may be imminent. The ability to identify both known and unidentified assaults is the main benefit of anomaly-based detection. Signature-based and anomaly-based detection are combined in a hybrid detection approach. According to reports, CIDS that employ hybrid detection produce superior results than those produced by conventional approaches. There are three different types of cloud computing based on the IDS. **Host-based IDS (HIDS)** is the first intrusion detection software was designed using an original target system as the mainframe computer in which there is some outside interaction was not frequent. The HIDS will operate based on the information is collected using an individual computer system. This monitors all the inbound and the outbound packets from its computer system and will also alert users or the administrator in case there are found to be any suspicious activities like a system call, the thread or processes, the asset or configuration access by means of observance of the host along with its situation. This may be used normally for the purpose of protection of the private information is very valuable for the various server-based systems.

Networks based IDSs (NIDS) will also be capturing the traffic for the whole of the network and will further analyses it for the purpose of detecting all of the possible intrusions like its port scanning or sometimes the DoSS attacks. The NIDS normally performs such detection of intrusion by means of efficiently processing the IP and also the headers of the transport layers for all of the captured network packets. This will make use of this anomaly and then the network packets are collected and a correlation will be arrived at along with all the signatures for the many known attacks and this is used for making a comparison of user behavior along with their known profiles. There are multiple hosts operating within the network that are secured from all the attackers by using the NIDS that are deployed properly and if this has to be run in a stealth mode we also need to know the

location of its NIDS which is normally hidden from its attacker. Here the NIDS cannot perform any analysis in cases where the traffic is encrypted. Hypervisor based IDS, the hypervisor will provide levels of interaction in the VMs and the hypervisor that is based on the IDS has been placed on a hypervisor layer. This also helps in the analysis of the available information for detecting anomalous actions of the users. Such information has been based on the multiple level communication inside of the hypervisor based virtual network**. Distributed IDS (DIDS)** consist of the number of IDSs like the NIDS or the HIDS deployed in a network for analyzing traffic for the behavior of intrusive detection. The detection component will examine the behavior of the system and will transmit the collected data using a 5-standard format for the correlation manager. The Correlation manager will combine data from many multiple IDS and will generate a high-level alert keeping up the correspondence for an attack.

The **CICIDS2017 dataset** provides a realistic simulation of a real-world network environment with a diverse set of attacks and benign traffic. It is widely used by researchers and practitioners to develop and evaluate intrusion detection systems and to advance the field of network security. The dataset contains both benign traffic and various types of attacks, including **DoS (Denial of Service), DDoS (Distributed Denial of Service), Port Scan, Botnet, Brute Force, Infiltration, and Web Attack**. The attacks were generated using various tools and techniques commonly used by attackers to compromise or disrupt network systems.

CHAPTER 3
SUMMARY OF RELATED WORK

W. Wang, X. Du, D. Shan, R. Qin and N. Wang, "Cloud Intrusion Detection Method Based on Stacked Contractive Auto-Encoder and Support Vector Machine," in *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1634-1646

In this paper, they created a hybrid system that employs a stacked contractive auto-encoder (SCAE) for feature reduction and the SVM classification method for the identification of harmful attacks. They demonstrated experimentally, using the NSL-KDD and KDD Cup 99 intrusion detection datasets, that the proposed SCAE+SVM-IDS model delivers promising classification performance in six metrics when compared to three state-of-the-art approaches.

A. M. Vartouni, S. S. Kashi and M. Teshnehlab, "An anomaly detection method to detect web attacks using Stacked Auto-Encoder," 2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS), 2018, pp. 131-134

This paper employs the SAE technique in order to extract meaningful characteristics from HTTP request logs for anomaly detection in web application firewalls. As a one-class learner, Isolation Forest was used to distinguish between abnormal and normal data. The findings show that deep learning in general and deep models perform differently with various SAE structures.

A. Javadpour, S. Kazemi Abharian and G. Wang, "Feature Selection and Intrusion Detection in Cloud Environment Based on Machine Learning Algorithms," ***2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*****, Guangzhou, China, 2017, pp. 1417-1421F**

This paper introduces a novel approach based on intrusion detection systems and their varied architectures in order to improve the efficacy of intrusion detection in cloud computing. Two techniques—Pearson Linear Correlation and Mutual Information, plugin and irrelevant features are excluded—were used in this paper. In this study, the feature selection techniques for linear correlation and mutual information were integrated to assess the suggested method using the KDD99 database. Different classification techniques were used to the data, including decision tree, random forest, CART algorithm, and neural network.

G. Kene and D. P. Theng, "A review on intrusion detection techniques for cloud computing and security challenges," ***2015 2nd International Conference on Electronics and Communication Systems (ICECS)*****, Coimbatore, India, 2015, pp. 227-232**

This paper lists the various intrusions that compromise the cloud's availability, secrecy, and integrity. They have provided a thorough description of the various IDS types utilized in cloud environments. They have supplied an overview of intrusion detection methods in the form of charts and tables, which make it simple to comprehend the entire cloud computing environment. They came to the conclusion that while several IDS techniques have been suggested and have helped in the detection of intrusions in the cloud, they do not offer total protection.

M. Ficco, L. Tasquier and R. Aversa, "Intrusion Detection in Cloud Computing," 2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Compiegne, France, 2013, pp. 276-283

In this paper, a methodology for creating distributed intrusion detection systems in cloud computing is proposed. It is an open-source solution that enables the creation and deployment of security probes on the virtual assets and cloud infrastructure of the customer. The implemented architecture serves as the initial iteration of a cloud-based IDS management system.

U. Oktay and O. K. Sahingoz, "Proxy Network Intrusion Detection System for cloud computing," 2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), Konya, Turkey, 2013, pp. 98-104

This paper suggests a proxy NIDS design that performs network intrusion detection operations through the gateway. It aims to make it simple for customers and service providers to choose where to locate their protection mechanism in this way. The Proxy Network Intrusion Detection System solution allows providers to place NIDS on mesh topology gateways. As a result, providers will seek to secure their infrastructure with less resource consumption, in addition to users choosing this strategy to utilize and pay a lower value of resources.

H. A. Kholidy and F. Baiardi, "CIDS: A Framework for Intrusion Detection in Cloud Systems," *2012 Ninth International Conference on Information Technology - New Generations*, Las Vegas, NV, USA, 2012, pp. 379-385

This paper develops a framework for "CIDS," a cloud-based intrusion detection system in order to address the shortcomings of existing IDSs. To summarize the alarms and tell the cloud administrator, CIDS additionally offers a component. The CIDS architecture lacks a central coordinator and is elastic and scalable. The advantages of CIDS are discussed in this study, along with its components, architecture, and detection models.

A. Kannan, G. Q. Maguire Jr., A. Sharma and P. Schoo, "Genetic Algorithm Based Feature Selection Algorithm for Effective Intrusion Detection in Cloud Networks," 2012 IEEE 12th International Conference on Data Mining Workshops, 2012, pp. 416-423

This paper presents a new genetic-based feature selection approach for cloud networks. This approach is used to identify the best number of characteristics for intrusion detection from the KDD cup data set. Furthermore, a system for intrusion detection has been suggested that employs this feature selection technique and then applies the existing Fuzzy SVM for successful classification of intrusions using the KDD Cup dataset for safeguarding cloud networks.

I. Shiri, B. Shanmugam and N. B. Idris, "A parallel technique for improving the performance of signature-based network intrusion detection system," 2011 IEEE 3rd International Conference on Communication Software and Networks, pp. 692-696

The parallel approach presented in this paper was recommended to improve the performance of signature-based network intrusion detection systems. The use of an effective string-matching algorithm, hardware acceleration, and finally parallelism have all

been suggested in the past because a subpar passive system misses many attacks and drops many packets on a high-speed network.

C. -C. Lo, C. -C. Huang and J. Ku, "A Cooperative Intrusion Detection System Framework for Cloud Computing Networks," 2010 39th International Conference on Parallel Processing Workshops, San Diego, CA, USA, 2010, pp. 280-284

This paper provides a cooperative intrusion detection system for cloud computing networks in order to lessen the effects of DoS attacks. By doing this, cooperative IDS alerts other IDS systems if a DoS attack occurs in one of the cloud computing zones. The same assault sent by one IDS could be gathered by another IDS. The reliability of this alarm message is then assessed using the majority vote approach. The suggested system prevents a single point of failure in the IDS system. Early detection and preventive methods are implemented by these agents working together. IDSs placed in cloud computing zones, apart from the victim one, could therefore stop this kind of attack from happening.

CHAPTER 4
ARCHITECTURE DIAGRAM

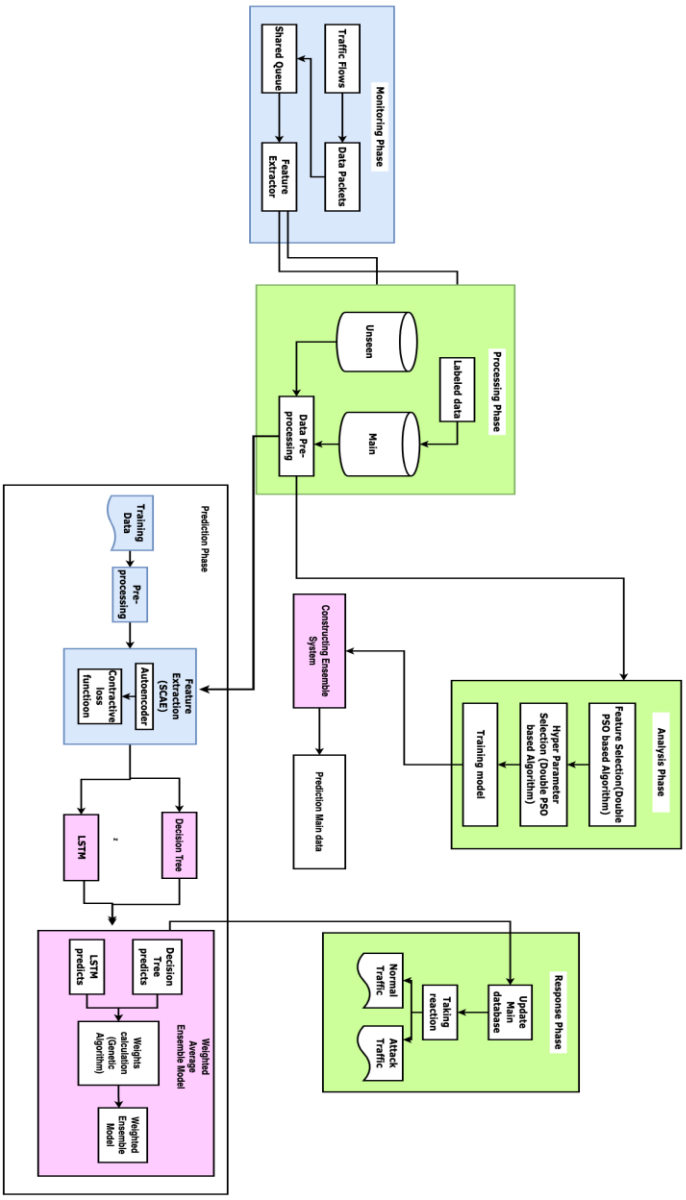


Fig 4.1 ARCHITECTURE DIAGRAM

CHAPTER 5

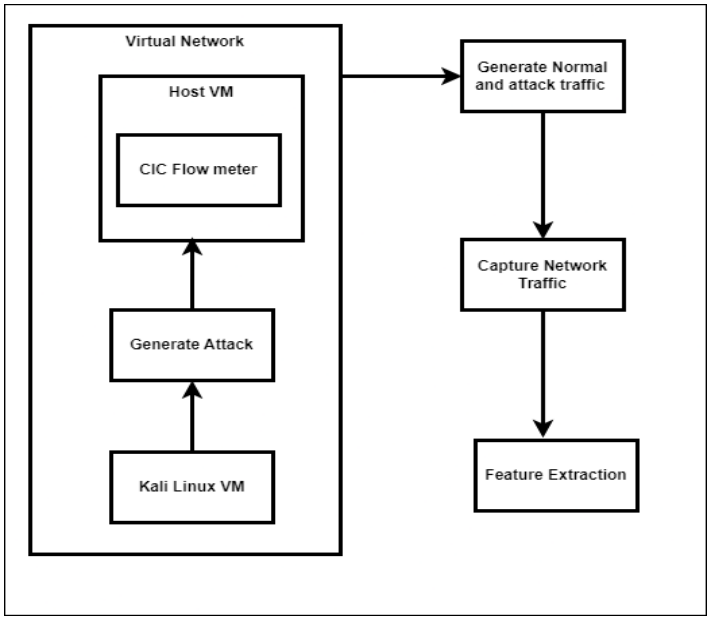
DETAILS OF MODULE DESIGN

List of Modules:

1. Monitoring Phase
2. Processing Phase
3. Analysis Phase
4. Prediction Phase
5. Response Phase

MONITORING PHASE:

The monitoring module's initial job is to capture all in-bound and out-bound data packets transiting the cloud network. The monitoring module employs sensors to sensitive network traffic to help in this operation. Furthermore, the monitoring module **may collect data packets** from various application, transport, and network protocols such as TCP, UDP, ICMP, IP, HTTP, SMTP, and so on. The traffic flows are instantly stored in the shared queue. The shared queue serves as an intermediary station between packet capture and feature extraction, storing gathered data packets until the feature extractor processes them consecutively.



Steps:

1. *Private Cloud is set with multiple Virtual Machine.*
2. *One Virtual Machine, say Kali Linux is used to generate different attack on the host machine.*
3. *CIC Flow meter is used in Host VM to capture the network traffic and export the data as CSV.*
4. *Feature Extraction is to be performed in the exported data.*

PROCESSING PHASE:

Prior to the forecast procedure, the processing module completes all necessary missions. It initially gets data records from the feature extractor and attempts to classify them (as "Normal" or "Attack"). This is possible by utilizing a **signature-based detection** process, which uses a set of pre-defined criteria (signatures) to match the examined data record against known attack patterns. If there is a match, the data record will be categorized as a "Attack". It will be called "Normal" otherwise. After that, processing module puts the labeled data record in a database we termed it as "main database".

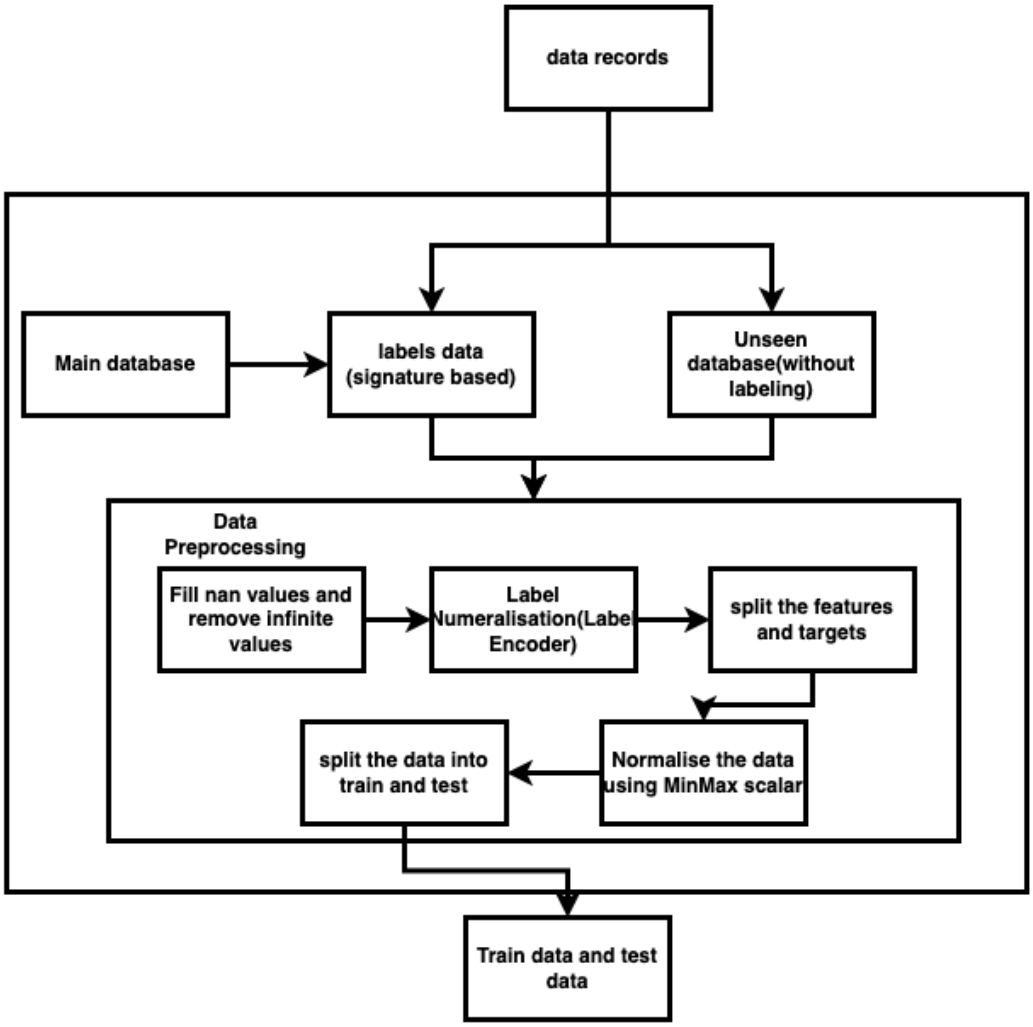


Fig 5.2 PROCESSING PHASE

Algorithm:

1. Remove Nan values and infinite values
2. Using label encoder fit the labels and transform it into numbers
3. Split the features and targets
4. Use normalizer as Min Max scaler

$$xi = (xi - min) / (max - min).$$

5. Split the data into train and test.

PREDICTION PHASE:

The prediction module, which collects the data from the processing module and also merges the data with the dataset (CICIDS2017), prepares a combined dataset. These data will go through the feature extraction algorithm and produce an output dataset, which will be used as the main dataset to train the model. The created model performs two sequential tasks, which is the core of the proposed CIDS. The first step is to build the **weighted average ensemble system** utilizing the previously trained decision tree and LSTM models, where the weights for that ensemble model will be generated by the fitness function of the genetic algorithm. The prediction module's second purpose is to successively choose a data record from the preprocessed, unseen database. Following that, the selected data record is tested using the ensemble system, and the weighted ensemble engine's final judgment is sent to the response module.

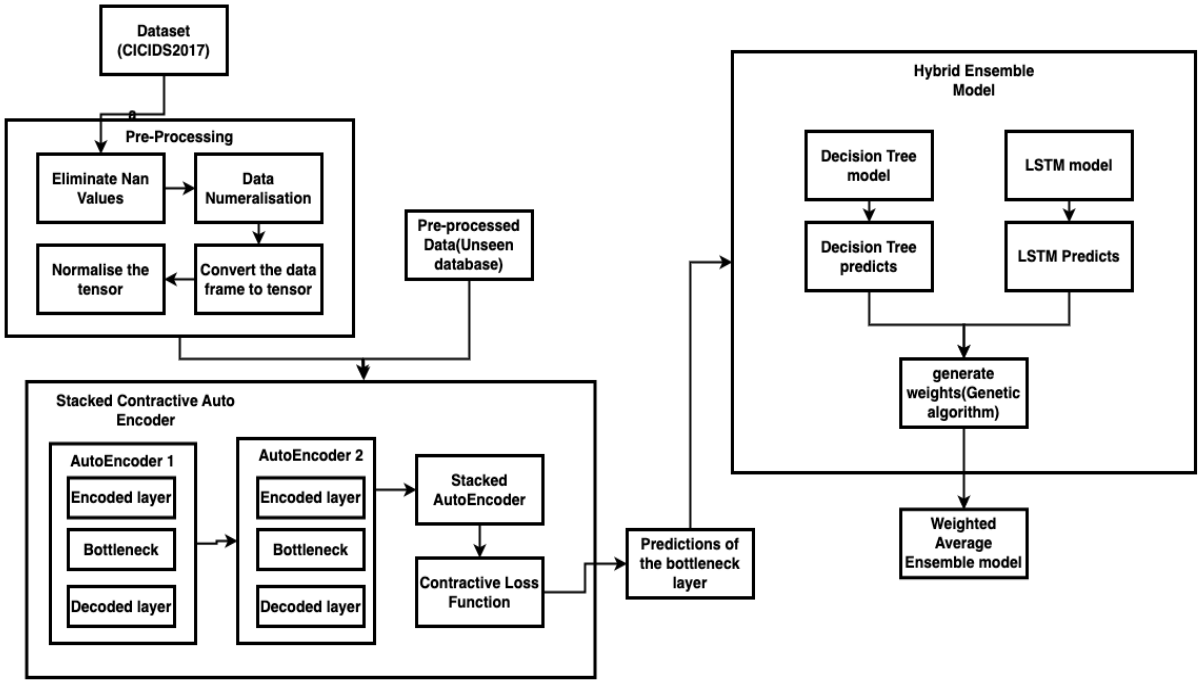


Fig 5.3 PREDICTION PHAS

First of all, convert the dataset from unbalanced to balanced label encoding, and then normalization is needed. The mini-max transformation is used for normalization,

xi = (xi - min) / (max - min).

Algorithm-SCAE:

1. *Define hyper parameters like learning rate, coefficient of contractive penalty term (beta)*
2. *Define layers of the autoencoder and stack them on each other*
3. *Define a contractive loss function where,*
    a. *Calculate jacobian matrix and compute each partial derivative and form a matrix*

$$z_{ij} = \delta f_i / \delta x_j$$

    b. *Calculate Forbenius norm where it is the root of sum of squares of each element in jacobian matrix.*

$$y = \sqrt{\sum_{i=0}^{n} \sum_{j=0}^{n} x_{ij}}$$

    c. *Calculate the contractive penalty*

reduce_mean(forbenius_norm)

    d. *Return loss output as*

$$\beta * contractivepenalty$$

4. *Compile the model with loss as mean squared error as primary loss and contractive loss as the secondary loss*

5. *Get the predictions by testing the test data and train data with the model.*

Algorithm - Weighted Average Ensemble method:

1. *Create Genetic algorithm with 100 iterations to find weights in order to get good accuracy*
    a. *Define weight bounds*

$$bounds = [(0, 1), (0, 1)]$$

    b. *Define fitness function*

$$Weight_{predicts} = weight[0] * dt_{predicts} + weight[1] * lstm_{predicts}$$

    c. *Find accuracy for weight_pred*

Accuracy = (TP + TN) / (TP + FP + TN + FN)

    d. *Repeat b and c until 100 iterations.*
2. *By considering weights from step 1, find the accuracy with the test labels and weight preds.*

ANALYSIS PHASE:

The analysis module handles the deep learning models' pre-training and training stages. The analysis module runs the **double PSO-based** method for feature and hyper parameter selection during the pre-training phase. After data preparation, the analysis module obtains a copy of the main database. The upper level of the double PSO-based method is then executed on the main database to determine the best feature subset. The main database is then reduced using only the best characteristics. Then, using the decreased master database, it conducts the bottom level of the double PSO-based method to generate the ideal hyper parameter vector.

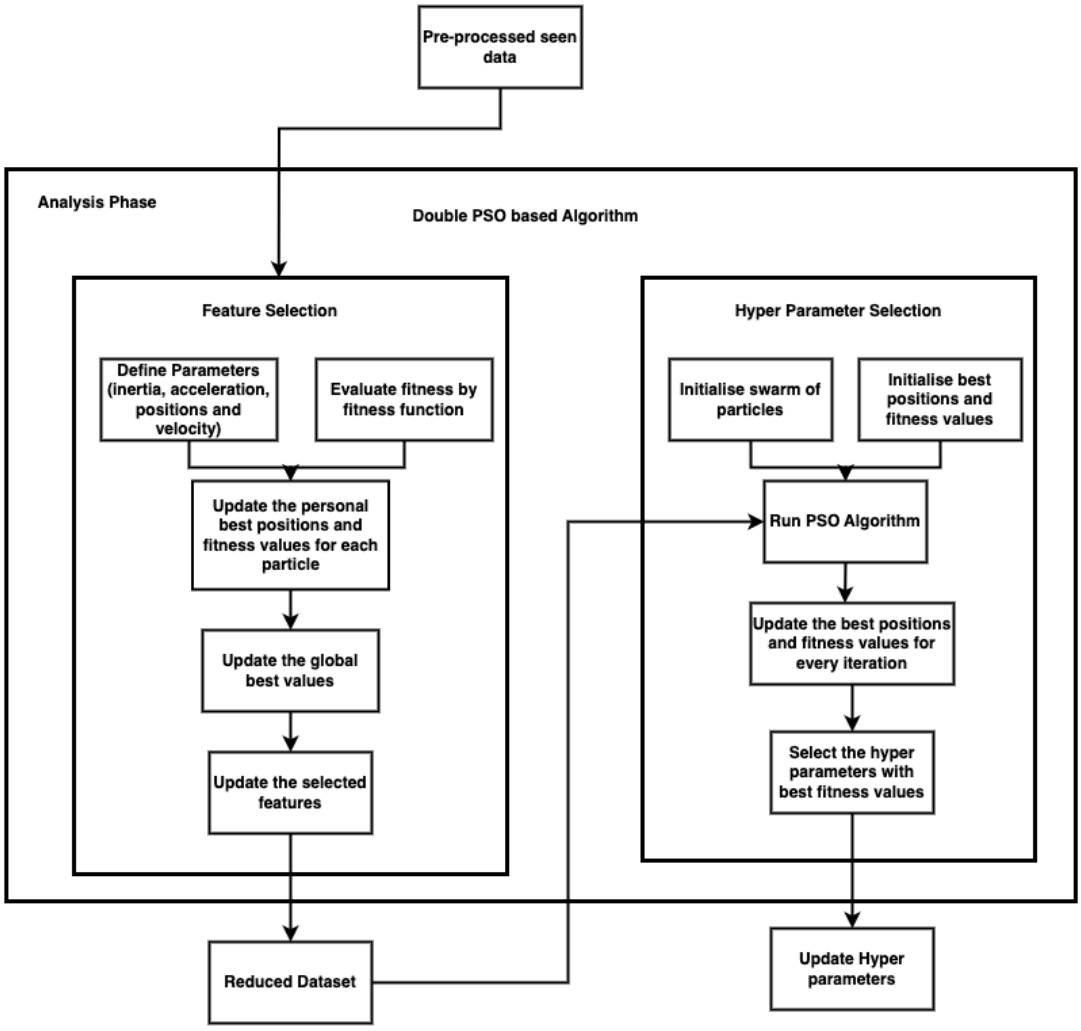


Fig 5.4 ANALYSIS PHASE

Algorithm - Particle Swarm Optimization Upper Level:

1. *Initialize PSO parameters*
2. *Evaluate the fitness of each particle based on the corresponding feature subset*
3. *Update personnel best positions and fitness values for each iteration*
4. *Update global best positions and fitness values for the swarm.*
5. *Update the velocities and positions of the particle.*

$$V_i = w * v_i + c1 * rand() * (pbest_{xi} - x_i) + c2 * rand() * (gbest_x - x_i)$$

$$x_i = x_i + v_i$$

6. *Repeat 2 to 5 until maximum iterations reached*
7. *Select the feature subset corresponding to the global best position.*

Algorithm - Particle Swarm Optimization Lower Level:

1. *Define parameters for PSO*
2. *Initialize swarm particles with randomly using search space*
3. *Evaluate Fitness of each particle*
4. *Update personnel best positions and fitness values for each iteration*
5. *Update global best positions and fitness values for the swarm*
6. *Update Velocity and position of each particle*
    a. *Calculate Inertia weight*

        $w_i = w_{start} - (w_{end} - w_{start}) * (iter / max\_iter)$

    b. *Compute Cognitive coefficient*

        $v_i = w_i * v_i + c1 * rand() * (pbest_i - x_i)$

    c. *Calculate social coefficient*

        $v_i = w_i * v_i + c2 * rand() * (gbest - x_i)$

    d. *Calculate Cognitive component and Social component*

        *Cognitive_component = cognitive_coeff * (part ['personel_best_position'] - part ['position']*

        *Social_component = social_coeff * (part ['personel_best_position'] - part ['position']*

    e. *Update the velocity and position of each particle*

        *Part['velocity'] = Part['velocity'] * inertia_weight + Cognitive_component + Social_component*

    f. *Update position of the particle by adding its velocity to its current position.*
7. *Repeat 3 to 6 until maximum iterations are done.*
8. *Select the hyper parameters with the best fitness as the optimal hyper parameter subset.*

RESPONSE PHASE:

The response module receives both the final model and the tested data record, and then labels the tested data record with the final model. The **master database is then updated** by storing the new labeled data record at the end of the master database.

The second function is in charge of reacting to the final decision of "Normal" or "Attack". In the instance of "Normal," the response module instructs the prediction module to go to the next data record in the unseen database in order to forecast its label. If, on the other hand, the final choice obtained is "Attack," the response module issues an alert that the cloud network is being subjected to potentially harmful activities.

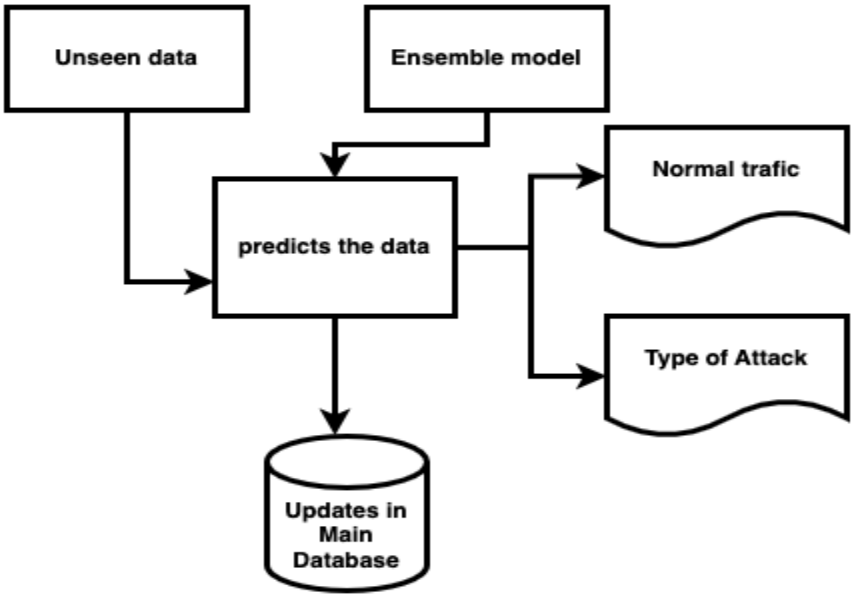


Fig 5.5 RESPONSE PHASE

CHAPTER 6

IMPLEMENTATION DETAILS (30%)

Pre-processing techniques:

LABEL ENCODING & REMOVING NAN AND INFINITE VALUES:

```
[6] dff = pd.read_csv('/content/drive/MyDrive/FYP/final_dataset.csv')

[7] dff = dff.fillna(0)
    dff = dff.replace([np.inf, -np.inf], 1e9)

[8] col_to_encode = ' Label'
    encoder = LabelEncoder()
    dff[col_to_encode] = encoder.fit_transform(dff[col_to_encode])

[9] dff[col_to_encode].unique()

    array([ 0,  7, 11,  6,  5,  4,  3,  8, 12, 14, 13,  9,  1, 10,  2])
```

```
[14] dff
```

Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...	min_seg_size_forward	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
2	0	12	0	6	6	6.000000	0.000000	...	20	0.0	0.0	0	0	0.0	0.0	0	0	0
2	0	12	0	6	6	6.000000	0.000000	...	20	0.0	0.0	0	0	0.0	0.0	0	0	0
7	4	484	414	233	0	69.142857	111.967895	...	20	0.0	0.0	0	0	0.0	0.0	0	0	0
9	4	656	3064	313	0	72.888889	136.153814	...	20	0.0	0.0	0	0	0.0	0.0	0	0	0
9	6	3134	3048	1552	0	348.222222	682.482560	...	20	0.0	0.0	0	0	0.0	0.0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2	0	12	0	6	6	6.000000	0.000000	...	20	0.0	0.0	0	0	0.0	0.0	0	0	0
2	0	0	0	0	0	0.000000	0.000000	...	32	0.0	0.0	0	0	0.0	0.0	0	0	0
1	1	6	6	6	6	6.000000	0.000000	...	20	0.0	0.0	0	0	0.0	0.0	0	0	0
2	0	248	0	242	6	124.000000	166.877200	...	20	0.0	0.0	0	0	0.0	0.0	0	0	0
1	1	6	6	6	6	6.000000	0.000000	...	20	0.0	0.0	0	0	0.0	0.0	0	0	0

Fig 6.1 LABEL ENCODING

NORMALISATION (MIN MAX SCALAR)

```
[11] # Normalize the data in each column
     features = pd.DataFrame(scaler.fit_transform(features), columns=features.columns)

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

```
features
```

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...	act_data_pkt_fwd	min_seg_size_forward	Acti Me
0	0.755119	1.333333e-07	0.000005	0.000000	9.302326e-07	0.000000e+00	0.000257	0.002581	0.001010	0.000000	...	0.000005	1.0	
1	0.755119	1.166667e-07	0.000005	0.000000	9.302326e-07	0.000000e+00	0.000257	0.002581	0.001010	0.000000	...	0.000005	1.0	
2	0.001343	5.183333e-06	0.000027	0.000014	3.751938e-05	6.316242e-07	0.009974	0.000000	0.011639	0.015713	...	0.000023	1.0	
3	0.001343	7.433333e-06	0.000036	0.000014	5.085271e-05	4.674629e-06	0.013399	0.000000	0.012269	0.019108	...	0.000033	1.0	
4	0.001343	9.774999e-06	0.000036	0.000021	2.429457e-04	4.650219e-06	0.066438	0.000000	0.058615	0.095779	...	0.000033	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1530738	0.936033	2.900000e-06	0.000005	0.000000	9.302326e-07	0.000000e+00	0.000257	0.002581	0.001010	0.000000	...	0.000005	1.0	
1530739	0.609577	1.925000e-06	0.000005	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	...	0.000000	1.0	
1530740	0.935408	3.416666e-07	0.000000	0.000003	4.651163e-07	9.153974e-09	0.000257	0.002581	0.001010	0.000000	...	0.000000	1.0	

0s completed at 3:03 PM

Fig 6.2 MIN MAX SCALAR

FEATURE EXTRACTION (SCAE):

```
[ ] # Define the hyperparameters
    input_dim = 78
    hidden_dim_1 = 64
    hidden_dim_2 = 32
    learning_rate = 0.001
    batch_size = 32
    num_epochs = 20
    beta = 1.0  # the coefficient for the contractive penalty term

    # Define the layers of the autoencoder
    input_layer = tf.keras.layers.Input(shape=(input_dim,))
    encoder_1 = tf.keras.layers.Dense(hidden_dim_1, activation="relu")(input_layer)
    encoder_2 = tf.keras.layers.Dense(hidden_dim_2, activation="relu")(encoder_1)
    decoder_1 = tf.keras.layers.Dense(hidden_dim_1, activation="relu")(encoder_2)
    decoder_2 = tf.keras.layers.Dense(input_dim, activation="sigmoid")(decoder_1)

    # Define the model and compile it
    autoencoder = tf.keras.models.Model(inputs=input_layer, outputs=decoder_2)
```

Fig 6.3 AUTO ENCODER

```python
def contractive_loss(y_true, y_pred):
    """Calculates the contractive loss for a given batch of input data."""
    mse = K.mean(K.square(y_true - y_pred), axis=1)
    W = K.variable(value=autoencoder.get_layer('dense_56').get_weights()[0]) # Get the weight matrix of the first hidden layer
    # Compute the jacobian matrix of the hidden layer outputs with respect to the input layer inputs
    h = autoencoder.get_layer('dense_56').output
    dh = h * (1 - h) # Derivative of the sigmoid activation function
    jacobian = dh[:, None] * W.T[None, :, :] # Compute the jacobian matrix
    jacobian = K.sum(jacobian ** 2, axis=(1, 2))
    return mse + 1e-4 * jacobian


autoencoder.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
                    loss=contractive_loss,
                    metrics=["accuracy"])
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=15)
mc = ModelCheckpoint('/Users/tharun/Desktop/FYP/DATASET/best_model_CNN.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
# Train the model
history = autoencoder.fit(X_train, X_train,
                          epochs=num_epochs,
                          batch_size=batch_size,
                          validation_data=(X_test, X_test),callbacks=[es,mc])
```

Fig 6.4 CONTRACTIVE LOSS FUNCTION

```
Epoch 1/20
38268/38269 [============================>.] - ETA: 0s - loss: 5.2701e-04 - accuracy: 0.3957
Epoch 1: val_accuracy improved from -inf to 0.54809, saving model to /Users/tharun/Desktop/FYP/DATASET/best_model_C
NN.h5
38269/38269 [==============================] - 93s 2ms/step - loss: 5.2701e-04 - accuracy: 0.3957 - val_loss: 1.637
5e-04 - val_accuracy: 0.5481
Epoch 2/20
38262/38269 [============================>.] - ETA: 0s - loss: 1.0424e-04 - accuracy: 0.6519
Epoch 2: val_accuracy improved from 0.54809 to 0.59265, saving model to /Users/tharun/Desktop/FYP/DATASET/best_mode
l_CNN.h5
38269/38269 [==============================] - 112s 3ms/step - loss: 1.0423e-04 - accuracy: 0.6519 - val_loss: 2.32
85e-05 - val_accuracy: 0.5926
Epoch 3/20
38265/38269 [============================>.] - ETA: 0s - loss: 2.3105e-05 - accuracy: 0.6408
Epoch 3: val_accuracy improved from 0.59265 to 0.71753, saving model to /Users/tharun/Desktop/FYP/DATASET/best_mode
l_CNN.h5
38269/38269 [==============================] - 112s 3ms/step - loss: 2.3104e-05 - accuracy: 0.6408 - val_loss: 1.70
70e-05 - val_accuracy: 0.7175
Epoch 4/20
```

```
Epoch 17: val_accuracy did not improve from 0.97603
38269/38269 [==============================] - 98s 3ms/step - loss: 1.2297e-05 - accuracy: 0.8807 - val_loss: 9.928
4e-06 - val_accuracy: 0.8245
Epoch 18/20
38258/38269 [============================>.] - ETA: 0s - loss: 1.2391e-05 - accuracy: 0.8539
Epoch 18: val_accuracy did not improve from 0.97603
38269/38269 [==============================] - 102s 3ms/step - loss: 1.2389e-05 - accuracy: 0.8539 - val_loss: 8.92
45e-06 - val_accuracy: 0.9486
Epoch 19/20
38267/38269 [============================>.] - ETA: 0s - loss: 1.2514e-05 - accuracy: 0.8910
Epoch 19: val_accuracy improved from 0.97603 to 0.99153, saving model to /Users/tharun/Desktop/FYP/DATASET/best_mod
el_CNN.h5
38269/38269 [==============================] - 96s 3ms/step - loss: 1.2514e-05 - accuracy: 0.8910 - val_loss: 1.293
0e-05 - val_accuracy: 0.9915
Epoch 20/20
38268/38269 [============================>.] - ETA: 0s - loss: 1.1610e-05 - accuracy: 0.8798
Epoch 20: val_accuracy did not improve from 0.99153
38269/38269 [==============================] - 98s 3ms/step - loss: 1.1610e-05 - accuracy: 0.8798 - val_loss: 9.542
0e-06 - val_accuracy: 0.9339
```

Fig 6.5 TRAINED SCAE

```
[ ] # Test the model
    test_loss, test_acc = best_model.evaluate(X_test, X_test)
    print(f"Test loss: {test_loss:.4f}")
    print(f"Test accuracy: {test_acc:.4f}")

    9568/9568 [==============================] - 11s 1ms/step - loss: 1.2930e-05 - accuracy: 0.9915
    Test loss: 0.0000
    Test accuracy: 0.9915

[ ]

[ ] # Extract the encoder layers
    encoder_1 = tf.keras.models.Model(inputs=input_layer, outputs=encoder_1)
    encoder_2 = tf.keras.models.Model(inputs=encoder_1.input, outputs=encoder_2)

[ ] encoded_train = encoder_2.predict(X_train)
    encoded_test = encoder_2.predict(X_test)

    38269/38269 [==============================] - 74s 2ms/step
    9568/9568 [==============================] - 10s 1ms/step
```

Fig 6.6 ACCURACY AND PREDICTIONS

LSTM MODEL:

```
[ ] model3 = models.Sequential()
    model3.add(layers.LSTM(50, input_shape = (32,1), activation = 'tanh', return_sequences = True, recurrent_activation='sigmoid', recurrent_dropout = 0.0, unro
    model3.add(layers.Dropout(0.15))
    model3.add(layers.LSTM(50, activation = 'tanh', return_sequences = True,recurrent_activation='sigmoid',recurrent_dropout = 0.0,unroll=False ,use_bias=True))
    model3.add(layers.Dropout(0.25))
    model3.add(layers.LSTM(50, activation = 'tanh', return_sequences = True,recurrent_activation='sigmoid',recurrent_dropout = 0.0,unroll=False ,use_bias=True))
    model3.add(layers.Dropout(0.35))
    model3.add(layers.LSTM(50, activation = 'tanh',recurrent_activation='sigmoid',recurrent_dropout = 0.0,unroll=False ,use_bias=True))
    model3.add(layers.Dropout(0.45))
    model3.add(layers.Flatten())
    model3.add(layers.Dense(50,activation = 'relu'))
    model3.add(layers.Dropout(0.04))
    model3.add(layers.Dense(15,activation = 'softmax'))
    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
    model3.compile(loss = tf.keras.losses.SparseCategoricalCrossentropy(),
                   optimizer = 'adam',
                   metrics = ['accuracy'])
    model3.summary()
```

Fig 6.7 LSTM MODEL

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 32, 50)            10400

 dropout (Dropout)           (None, 32, 50)            0

 lstm_1 (LSTM)               (None, 32, 50)            20200

 dropout_1 (Dropout)         (None, 32, 50)            0

 lstm_2 (LSTM)               (None, 32, 50)            20200

 dropout_2 (Dropout)         (None, 32, 50)            0

 lstm_3 (LSTM)               (None, 50)                20200

 dropout_3 (Dropout)         (None, 50)                0

 flatten (Flatten)           (None, 50)                0

 dense (Dense)               (None, 50)                2550

 dropout_4 (Dropout)         (None, 50)                0

 dense_1 (Dense)             (None, 15)                765

=================================================================
Total params: 74,315
Trainable params: 74,315
Non-trainable params: 0
_________________________________________________________________
```

Fig 6.8 MODEL SUMMARY

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=15)
mc = ModelCheckpoint('/Users/tharun/Desktop/FYP/DATASET/best_model_LSTM1.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
# Train the model
history = model3.fit(fea_train.reshape(fea_train.shape[0], fea_train.shape[1], 1), tar_train, validation_data=(fea_test, tar_test), epochs=50, batch_size=12
```

```
Epoch 1/50
9566/9568 [============================>.] - ETA: 0s - loss: 0.2482 - accuracy: 0.9146
Epoch 1: val_accuracy improved from -inf to 0.95349, saving model to /Users/tharun/Desktop/FYP/DATASET/best_model_LSTM1.h5
9568/9568 [==============================] - 136s 14ms/step - loss: 0.2482 - accuracy: 0.9146 - val_loss: 0.1157 - val_accuracy: 0.9535
Epoch 2/50
9567/9568 [============================>.] - ETA: 0s - loss: 0.1163 - accuracy: 0.9560
Epoch 2: val_accuracy improved from 0.95349 to 0.96629, saving model to /Users/tharun/Desktop/FYP/DATASET/best_model_LSTM1.h5
9568/9568 [==============================] - 135s 14ms/step - loss: 0.1163 - accuracy: 0.9560 - val_loss: 0.0882 - val_accuracy: 0.9663
Epoch 3/50
9567/9568 [============================>.] - ETA: 0s - loss: 0.0991 - accuracy: 0.9617
Epoch 3: val_accuracy improved from 0.96629 to 0.96771, saving model to /Users/tharun/Desktop/FYP/DATASET/best_model_LSTM1.h5
9568/9568 [==============================] - 134s 14ms/step - loss: 0.0991 - accuracy: 0.9617 - val_loss: 0.0783 - val_accuracy: 0.9677
Epoch 4/50
9567/9568 [============================>.] - ETA: 0s - loss: 0.0903 - accuracy: 0.9657
Epoch 4: val_accuracy improved from 0.96771 to 0.97419, saving model to /Users/tharun/Desktop/FYP/DATASET/best_model_LSTM1.h5
9568/9568 [==============================] - 141s 15ms/step - loss: 0.0903 - accuracy: 0.9657 - val_loss: 0.0693 - val_accuracy: 0.9742
Epoch 5/50
9567/9568 [============================>.] - ETA: 0s - loss: 0.0842 - accuracy: 0.9686
Epoch 5: val_accuracy did not improve from 0.97419
9568/9568 [==============================] - 140s 15ms/step - loss: 0.0842 - accuracy: 0.9686 - val_loss: 0.0720 - val_accuracy: 0.9723
Epoch 6/50
9568/9568 [==============================] - ETA: 0s - loss: 0.0794 - accuracy: 0.9708
Epoch 6: val_accuracy did not improve from 0.97419
9568/9568 [==============================] - 142s 15ms/step - loss: 0.0794 - accuracy: 0.9708 - val_loss: 0.0679 - val_accuracy: 0.9723
Epoch 7/50
9568/9568 [==============================] - ETA: 0s - loss: 0.0757 - accuracy: 0.9726
Epoch 7: val_accuracy did not improve from 0.97419
9568/9568 [==============================] - 143s 15ms/step - loss: 0.0757 - accuracy: 0.9726 - val_loss: 0.1153 - val_accuracy: 0.9488
Epoch 8/50
9567/9568 [============================>.] - ETA: 0s - loss: 0.0725 - accuracy: 0.9740
Epoch 8: val_accuracy improved from 0.97419 to 0.97801, saving model to /Users/tharun/Desktop/FYP/DATASET/best_model_LSTM1.h5
9568/9568 [==============================] - 132s 14ms/step - loss: 0.0725 - accuracy: 0.9740 - val_loss: 0.0618 - val_accuracy: 0.9780
Epoch 9/50
```

```
[ ] 9568/9568 [==============================] - 138s 14ms/step - loss: 0.0458 - accuracy: 0.9833 - val_loss: 0.0423 - val_accuracy: 0.9843
Epoch 42/50
9566/9568 [============================>.] - ETA: 0s - loss: 0.0461 - accuracy: 0.9831
Epoch 42: val_accuracy improved from 0.98436 to 0.98475, saving model to /Users/tharun/Desktop/FYP/DATASET/best_model_LSTM1.h5
9568/9568 [==============================] - 132s 14ms/step - loss: 0.0461 - accuracy: 0.9831 - val_loss: 0.0411 - val_accuracy: 0.9847
Epoch 43/50
9568/9568 [==============================] - ETA: 0s - loss: 0.0455 - accuracy: 0.9834
Epoch 43: val_accuracy did not improve from 0.98475
9568/9568 [==============================] - 141s 15ms/step - loss: 0.0455 - accuracy: 0.9834 - val_loss: 0.0438 - val_accuracy: 0.9833
Epoch 44/50
9565/9568 [============================>.] - ETA: 0s - loss: 0.0459 - accuracy: 0.9832
Epoch 44: val_accuracy did not improve from 0.98475
9568/9568 [==============================] - 131s 14ms/step - loss: 0.0459 - accuracy: 0.9832 - val_loss: 0.0473 - val_accuracy: 0.9829
Epoch 45/50
9567/9568 [============================>.] - ETA: 0s - loss: 0.0453 - accuracy: 0.9834
Epoch 45: val_accuracy did not improve from 0.98475
9568/9568 [==============================] - 140s 15ms/step - loss: 0.0453 - accuracy: 0.9834 - val_loss: 0.0473 - val_accuracy: 0.9827
Epoch 46/50
9566/9568 [============================>.] - ETA: 0s - loss: 0.0455 - accuracy: 0.9833
Epoch 46: val_accuracy did not improve from 0.98475
9568/9568 [==============================] - 132s 14ms/step - loss: 0.0455 - accuracy: 0.9833 - val_loss: 0.0413 - val_accuracy: 0.9846
Epoch 47/50
9565/9568 [============================>.] - ETA: 0s - loss: 0.0453 - accuracy: 0.9833
Epoch 47: val_accuracy did not improve from 0.98475
9568/9568 [==============================] - 139s 14ms/step - loss: 0.0453 - accuracy: 0.9833 - val_loss: 0.0414 - val_accuracy: 0.9845
Epoch 48/50
9567/9568 [============================>.] - ETA: 0s - loss: 0.0448 - accuracy: 0.9836
Epoch 48: val_accuracy did not improve from 0.98475
9568/9568 [==============================] - 130s 14ms/step - loss: 0.0448 - accuracy: 0.9836 - val_loss: 0.0419 - val_accuracy: 0.9841
Epoch 49/50
9565/9568 [============================>.] - ETA: 0s - loss: 0.0446 - accuracy: 0.9836
Epoch 49: val_accuracy did not improve from 0.98475
9568/9568 [==============================] - 132s 14ms/step - loss: 0.0446 - accuracy: 0.9836 - val_loss: 0.0436 - val_accuracy: 0.9834
Epoch 50/50
9566/9568 [============================>.] - ETA: 0s - loss: 0.0447 - accuracy: 0.9834
Epoch 50: val_accuracy did not improve from 0.98475
9568/9568 [==============================] - 136s 14ms/step - loss: 0.0447 - accuracy: 0.9834 - val_loss: 0.0424 - val_accuracy: 0.9845
```

Screenshot

Fig 6.9 TRAINED LSTM MODEL

```
[ ]  sns.set_context('notebook', font_scale= 1.3)
     plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'test'], loc='upper left')
     plt.show()
```


```
[ ]  model3.save('/content/drive/MyDrive/FYP/DATASET/LSTM.h5')
```

Fig 6.10 LSTM MODEL ACCURACY

```
[ ]  # Apply the softmax activation function to the output tensor
     from scipy.special import softmax
     pred_probs = softmax(lstm_pred1, axis=0)

     # Get the class with the highest probability
     #pred_class = np.argmax(pred_probs, axis=0)
     sns.set_context('notebook', font_scale= 1.3)
     fig, ax = plt.subplots(1, 2, figsize = (25,  8))
     ax1 = plot_confusion_matrix(tar_test, lstm_pred1, ax= ax[0], cmap= 'YlGnBu')
     #ax2 = plot_roc(tar_test, pred_class, ax= ax[1], plot_macro= False, plot_micr
```

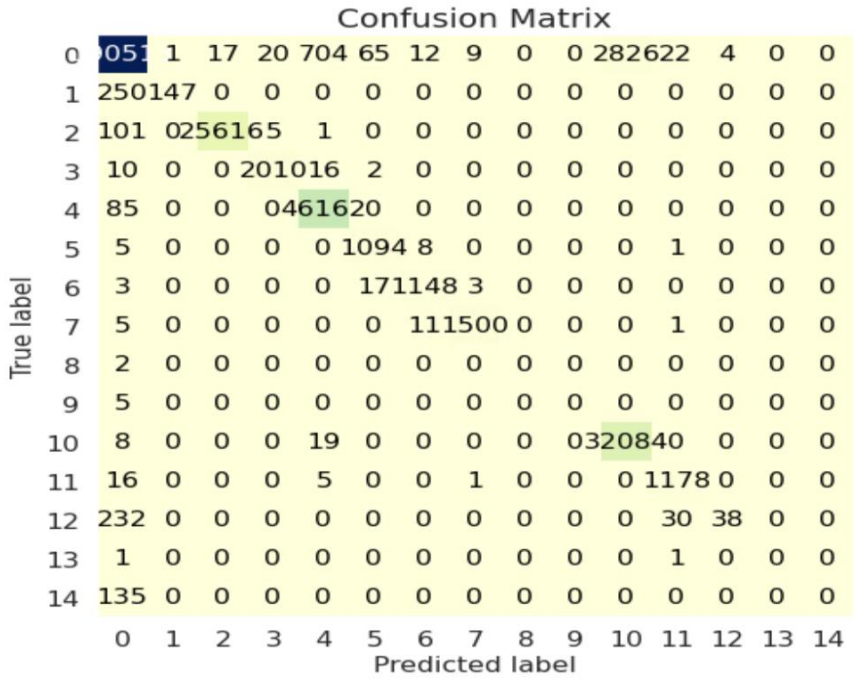


Fig 6.11 LSTM MODEL CONFUSION MATRIX

```
[ ] # Compute initial ensemble accuracy
    dt_pred1 = dt.predict(fea_test)
    lstm_pred_prob = model3.predict(fea_test)
    lstm_pred1 = np.argmax(lstm_pred_prob, axis=1)

    9568/9568 [==============================] - 54s 5ms/step

[ ] # Calculate precision and recall
    precision, recall, _, _ = precision_recall_fscore_support(tar_test, lstm_pred1, average='weighted')

    # Display or save the precision and recall
    print(f'Precision: {precision:.3f}, Recall: {recall:.3f}')

    Precision: 0.985, Recall: 0.985
```

Add text cell

Fig 6.12 PRECISION AND RECALL OF LSTM MODEL

DECISION TREE:

```
[ ] from keras.models import load_model
    model3=load_model('/content/drive/MyDrive/FYP/DATASET/LSTM.h5')

[ ] dt = DecisionTreeClassifier(random_state=42)
    dt.fit(fea_train,tar_train)

    DecisionTreeClassifier(random_state=42)

[ ] dt_pred_test=dt.predict(fea_test)

[ ] accuracy1 = accuracy_score(tar_test, dt_pred_test)
    accuracy1

    0.994685594269457

[ ] # Calculate precision and recall
    precision, recall, _, _ = precision_recall_fscore_support(tar_test, dt_pred_test, average='weighted')

    # Display or save the precision and recall
    print(f'Precision: {precision:.3f}, Recall: {recall:.3f}')

    Precision: 0.995, Recall: 0.995
```

Fig 6.13 PRECISION AND RECALL OF DECISION TREE MODEL

```
y_prob = dt.predict_proba(fea_test)
sns.set_context('notebook', font_scale= 1.3)
fig, ax = plt.subplots(1, 2, figsize = (25,  8))
ax1 = plot_confusion_matrix(tar_test, dt_pred_test, ax= ax[0], cmap= 'YlGnBu')
ax2 = plot_roc(tar_test, y_prob, ax= ax[1], plot_macro= False, plot_micro= False, cmap= 'winter')
```

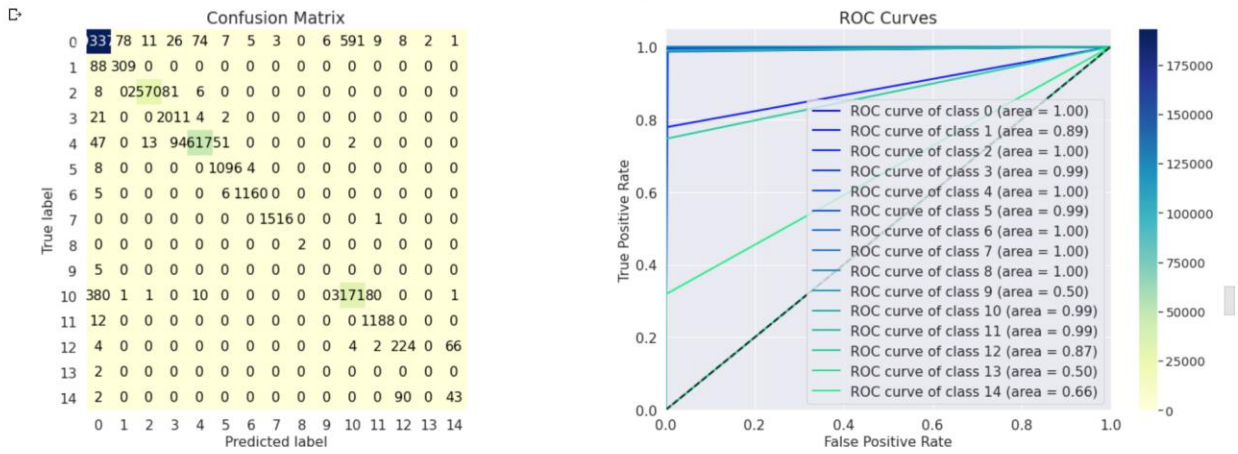


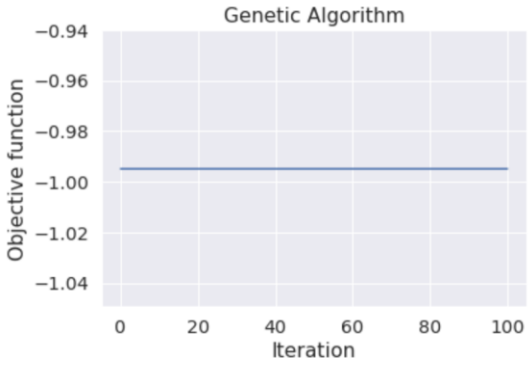
Fig 6.14 CONFUSION MATRIX AND ROC CURVE OF DECISION TREE MODEL

ENSEMBLE MODEL:

```
import numpy as np
from geneticalgorithm import geneticalgorithm as ga

# Define the fitness function that calculates the accuracy of the ensemble model
def fitness_function(weights):
    # Combine the predictions using the weights
    weighted_pred = weights[0] * dt_pred1 + weights[1] * lstm_pred1

    # Round the prediction to the nearest integer
    final_pred = np.round(weighted_pred)

    # Calculate the accuracy of the final prediction
    accuracy = accuracy_score(tar_test, final_pred)

    # Return the negative accuracy (since the genetic algorithm minimizes the objective function)
    return -accuracy

# Define the bounds for the weights (between 0 and 1)
bounds = [(0, 1), (0, 1)]
bounds = np.array(bounds)
algorithm_param = {'max_num_iteration': 100,\
                   'population_size':100,\
                   'mutation_probability':0.1,\
                   'elit_ratio': 0.01,\
                   'crossover_probability': 0.5,\
                   'parents_portion': 0.3,\
                   'crossover_type':'uniform',\
                   'max_iteration_without_improv':None}
# Create the genetic algorithm object
ga_model = ga(function=fitness_function, dimension=2, variable_type='real', variable_boundaries=bounds, algorithm_parameters=algorithm_param)
# Run the genetic algorithm for 100 generations with a population size of 20
ga_model.run()
# Get the best weights found by the genetic algorithm
weights = ga_model.best_variable
```

Fig 6.15 GENETIC ALGORITHM

```
The best solution found:
[0.98098415 0.0231155 ]

Objective function:
-0.994685594269457
```




```
[ ] ensemble_pred1 = np.average([dt_pred1, lstm_pred1], axis=0, weights=weights)
    initial_score = np.mean(ensemble_pred1 == tar_test)
    print("Initial ensemble accuracy: {:.2f}%".format(initial_score*100))

    Initial ensemble accuracy: 97.77%

[ ] y_pred_ensemble = np.array([np.argmax(np.bincount([dt_pred1[i], lstm_pred1[i]])) for i in range(len(dt_pred1))])
```

Fig 6.16 ACCURACY AND PREDICTION OF ENSEMBLE MODEL

```
[ ]  sns.set_context('notebook', font_scale= 1.3)
     fig, ax = plt.subplots(1, 2, figsize = (25,  8))
     ax1 = plot_confusion_matrix(tar_test, y_pred_ensemble, ax= ax[0], cmap= 'YlGnBu'
```

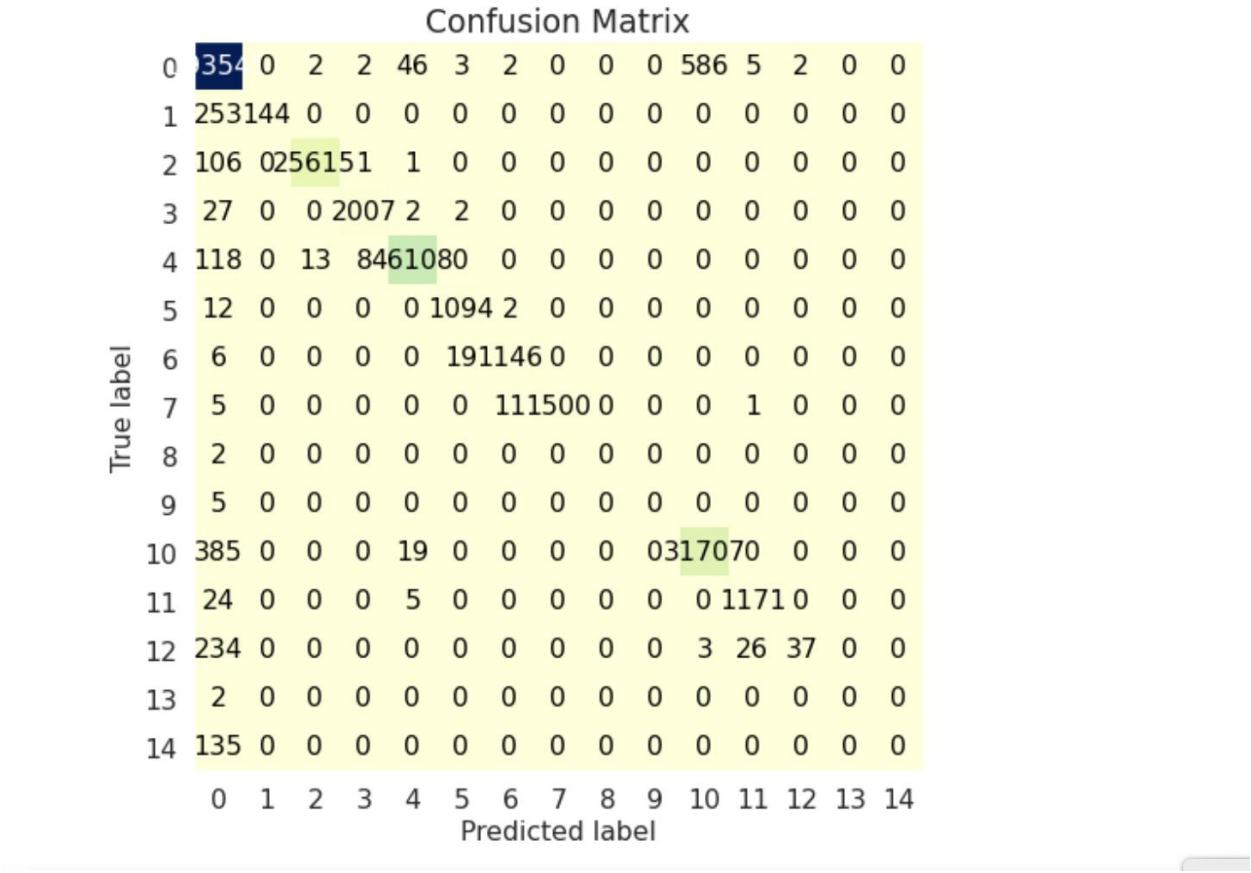


Fig 6.17 CONFUSION MATRIX OF ENSEMBLE MODEL

CHAPTER 7
METRICS FOR EVALUATION

THRESHOLD METRICS:

- **Classification Rate (CR):** Classification rate is a metric that measures the accuracy of a classifier, or how well it can predict the correct class label of a data point.

Classification Rate = Number of Correct Predictions / Total Number of Predictions

- **F Measure (FM):** F-measure is a metric used to measure the accuracy of a classification model. It is a harmonic mean between precision and recall.

F-measure = 2 * (Precision * Recall) / (Precision + Recall)

RANKING METRICS:

- **Accuracy:** It is the percentage of correctly predicted labels out of all instances in the dataset.

Accuracy = (Number of Correct Predictions) / Total Number of Predictions

- **Recall:** Recall measures the proportion of actual positive instances that are correctly identified by the model as positive.

Recall = True Positives / (True Positives + False Negatives)

- **Precision (PR):** Precision is a measure of the accuracy of a measurement or a system that produces measurements.

Precision = (Number of correctly estimated measurements / Total number of measurements) x 100

- **Area Under ROC Curve (AUC):** AUC, or Area Under the Receiver Operating Characteristic Curve, is a metric used to measure the performance of a binary classifier. It is a way to measure the accuracy and power of a classifier. AUC measures the area under the ROC curve, which is a graph of the true positive rate against the false positive rate.

$$\mathbf{AUC = \int_{0,1} ROC(t)dt}$$

PROBABILITY METRICS:

- **Root Mean Square Error (RMSE):** Root Mean Square Error (RMSE) is a measure of how well a model fits a dataset. It is the average of the squared differences between the observed values and the predicted values.

$$\mathbf{RMSE = \sqrt{(\sum (predicted - observed)^2 / n)}}$$

CHAPTER 8

TEST CASES

The models which are used as test cases are:

1. Decision Tree
2. LSTM
3. Weighted Average Ensemble Model

Performances of Each Model are listed below**:**

Metrics	**Decision Tree**	**LSTM**	**Ensemble Model**
Accuracy	99.4	98.4	97.7
Precision	99.5	98.2	98.5
Recall	99.5	97.9	98.5

CHAPTER 9

REFERENCES

1. W. Wang, X. Du, D. Shan, R. Qin and N. Wang, "Cloud Intrusion Detection Method Based on Stacked Contractive Auto-Encoder and Support Vector Machine," in IEEE Transactions on Cloud Computing, vol. 10, no. 3, pp. 1634-1646

2. A. M. Vartouni, S. S. Kashi and M. Teshnehlab, "An anomaly detection method to detect web attacks using Stacked Auto-Encoder," 2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS), 2018, pp. 131-134

3. A. Javadpour, S. Kazemi Abharian and G. Wang, "Feature Selection and Intrusion Detection in Cloud Environment Based on Machine Learning Algorithms," *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, Guangzhou, China, 2017, pp. 1417-1421F

4. G. Kene and D. P. Theng, "A review on intrusion detection techniques for cloud computing and security challenges," *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*, Coimbatore, India, 2015, pp. 227-232

5. M. Ficco, L. Tasquier and R. Aversa, "Intrusion Detection in Cloud Computing," 2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Compiegne, France, 2013, pp. 276-283

6. U. Oktay and O. K. Sahingoz, "Proxy Network Intrusion Detection System for cloud computing," 2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), Konya, Turkey, 2013, pp. 98-104

7. H. A. Kholidy and F. Baiardi, "CIDS: A Framework for Intrusion Detection in Cloud Systems," *2012 Ninth International Conference on Information Technology - New Generations*, Las Vegas, NV, USA, 2012, pp. 379-385

8. A. Kannan, G. Q. Maguire Jr., A. Sharma and P. Schoo, "Genetic Algorithm Based Feature Selection Algorithm for Effective Intrusion Detection in Cloud Networks," 2012 IEEE 12th International Conference on Data Mining Workshops, 2012, pp.

9. I. Shiri, B. Shanmugam and N. B. Idris, "A parallel technique for improving the performance of signature-based network intrusion detection system," 2011 IEEE 3rd International Conference on Communication Software and Networks, 2011, pp. 692-696

10. C. -C. Lo, C. -C. Huang and J. Ku, "A Cooperative Intrusion Detection System Framework for Cloud Computing Networks," 2010 39th International Conference on Parallel Processing Workshops, San Diego, CA, USA, 2010, pp. 280-284