**FML**

**MINI PROJECT**

**Batch: 1**

**Team number: 4**

**Team members:**

Katherine Deborah G(2018121)

Elakkiya R(2018109)

Durka VL(2018110)

Kavitharan(2018122)

**Encoding:** TF IDF

**Neural Network:** LSTM

**Feature Selection:** Particle Swarm Optimization

**Supervised Learning:** Support Vector Machine

**Unsupervised Learning:** Agglomerative Clustering

**Mini Project: Phase - I**

**Data Pre-processing and Deep Learning Algorithm - LSTM**

## Aim:

This phase aims to pre-process the given URL dataset using TF-IDF vectorization and implement an LSTM-based deep learning algorithm for binary classification.

## Algorithm:

1. Import the necessary libraries (Pandas, sklearn, TensorFlow, Keras, and Matplotlib).
2. Load the URL dataset from a CSV file.
3. Pre-process the dataset using TF-IDF vectorization with the TfidfVectorizer from sklearn.
4. Split the dataset into training and testing sets using train\_test\_split from sklearn.
5. Define the architecture of the LSTM-based deep learning model using the Sequential class from Keras.
6. Add a Dense layer with 128 units as the input layer.
7. Add a Reshape layer to reshape the input to (128, 1) for the LSTM layer.
8. Add an LSTM layer with 128 units.
9. Add a Dense layer with 1 unit and a sigmoid activation for binary classification.
10. Compile the model with the binary cross-entropy loss and the Adam optimizer.
11. Train the model on the training data using the fit method, specifying the number of epochs and batch size.
12. Plot the training and validation accuracy during training using Matplotlib.

.

## Program:

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense,Reshape

import matplotlib.pyplot as plt

# Load URL dataset

url\_df = pd.read\_csv('url.csv')

## Output:

Epoch 1/10

250/250 [==============================] - 50s 181ms/step - loss: 0.4367 - accuracy: 0.8199 - val\_loss: 0.1203 - val\_accuracy: 0.9725

Epoch 2/10

250/250 [==============================] - 43s 172ms/step - loss: 0.0543 - accuracy: 0.9875 - val\_loss: 0.0522 - val\_accuracy: 0.9865

Epoch 3/10

250/250 [==============================] - 43s 172ms/step - loss: 0.0095 - accuracy: 0.9984 - val\_loss: 0.0533 - val\_accuracy: 0.9870

Epoch 4/10

250/250 [==============================] - 43s 171ms/step - loss: 0.0038 - accuracy: 0.9992 - val\_loss: 0.0499 - val\_accuracy: 0.9915

Epoch 5/10

250/250 [==============================] - 43s 171ms/step - loss: 0.0034 - accuracy: 0.9992 - val\_loss: 0.0493 - val\_accuracy: 0.9885

Epoch 6/10

250/250 [==============================] - 43s 171ms/step - loss: 0.0021 - accuracy: 0.9994 - val\_loss: 0.0480 - val\_accuracy: 0.9910

Epoch 7/10

250/250 [==============================] - 43s 171ms/step - loss: 0.0021 - accuracy: 0.9994 - val\_loss: 0.0796 - val\_accuracy: 0.9840

Epoch 8/10

250/250 [==============================] - 43s 172ms/step - loss: 0.0051 - accuracy: 0.9992 - val\_loss: 0.0771 - val\_accuracy: 0.9745

Epoch 9/10

250/250 [==============================] - 43s 172ms/step - loss: 0.0047 - accuracy: 0.9987 - val\_loss: 0.0406 - val\_accuracy: 0.9890

Epoch 10/10

250/250 [==============================] - 43s 171ms/step - loss: 0.0020 - accuracy: 0.9992 - val\_loss: 0.0447 - val\_accuracy: 0.9895

# Preprocess dataset with TF-IDF vectorization

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(url\_df['Domain'])

y = url\_df['Label']

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

X\_train = X\_train.toarray()

X\_test = X\_test.toarray()

# Define model architecture

model = Sequential()

model.add(Dense(128, input\_shape=(X\_train.shape[1],)))

model.add(Reshape((128, 1)))

model.add(LSTM(128))

model.add(Dense(1, activation='sigmoid'))

# Compile the model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model on training data

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, y\_test))

# Plot the accuracy during training

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

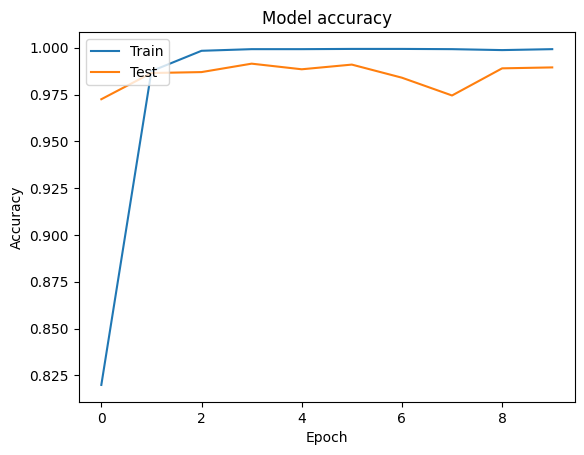
plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()



## Result:

The dataset is pre-processed using TF-IDF vectorization, and an LSTM model is implemented and verified.

**Mini Project: Phase - II**

**Feature Selection and Supervised Learning – Support Vector Machine (SVM)**

## Aim:

The aim of this code is to optimize the hyperparameters of an SVM classifier using Particle Swarm Optimization (PSO) for the given malicious URL dataset and evaluate the performance of the final model by calculating the accuracy.

## Algorithm:

1. Import the necessary libraries: numpy, sklearn.model\_selection, sklearn.svm, sklearn.metrics, sklearn.preprocessing, sklearn.utils, and sklearn.feature\_extraction.text.
2. Load the dataset from a CSV file using numpy.genfromtxt().
3. Extract the dataset's features (domain names) and labels.
4. Encode the labels as integers using LabelEncoder().
5. Split the data into training and testing sets using train\_test\_split().
6. Convert the domain names to numerical features using count vectorization with CountVectorizer().
7. Define the SVM classifier using SVC().
8. Define the fitness function for PSO, which takes a particle as input, sets the SVM hyperparameters, trains the SVM classifier, predicts the labels for the test set, and calculates the accuracy score as the fitness value.
9. Define the bounds for the hyperparameters (C and gamma) as a tuple of arrays.
10. Run PSO optimization using GlobalBestPSO() from pyswarms.single, specifying the number of particles, dimensions, bounds, and options.
11. Extract the best hyperparameters from the optimizer's results.
12. Set the SVM classifier with the best hyperparameters.
13. Train the SVM classifier with the best hyperparameters on the training data.
14. Predict the labels for the test set using the trained SVM classifier.
15. Calculate the accuracy score of the final model using accuracy\_score().
16. Print the accuracy score.

## Program:

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

## Output:

40,074 - pyswarms.single.global\_best - INFO - Optimize for 5 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}

pyswarms.single.global\_best: 100%|███████████████████████████████████████████████████|5/5, best\_cost=-.995

2023-05-18 21:25:45,501 - pyswarms.single.global\_best - INFO - Optimization finished | best cost: -0.9945, best pos: [53.30170224 8.98207399]

Accuracy: 0.9945

from sklearn.preprocessing import LabelEncoder

from sklearn.utils import shuffle

from sklearn.feature\_extraction.text import CountVectorizer

from pyswarms.single import GlobalBestPSO

# Load the dataset (assuming it is in a CSV file)

data = np.genfromtxt('url.csv', delimiter=',', dtype=None, encoding=None, names=True)

# Extract features and labels from the dataset

X = data['Domain']

y = data['Label']

# Encode labels as integers (0 for benign, 1 for malicious)

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

# 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)

# Convert domain names to numerical features using count vectorization

vectorizer = CountVectorizer()

X\_train = vectorizer.fit\_transform(X\_train).toarray()

X\_test = vectorizer.transform(X\_test).toarray()

# Define the SVM classifier

svm = SVC()

# Define the fitness function for PSO

def fitness\_function(particles):

# Extract the hyperparameters from particles

C = particles[0][0] # SVM regularization parameter

gamma = particles[0][1] # RBF kernel parameter

# Set the SVM hyperparameters

svm.set\_params(C=C, gamma=gamma)

# Train the SVM classifier

svm.fit(X\_train, y\_train)

# Predict the labels for the test set

y\_pred = svm.predict(X\_test)

# Calculate the accuracy score as fitness value

accuracy = accuracy\_score(y\_test, y\_pred)

return -accuracy # Negative value for maximization problem

# Define the bounds for the hyperparameters (C and gamma)

bounds = (np.array([1e-6, 1e-6]), np.array([100.0, 10.0]))

# Run PSO optimization

options = {'c1': 0.5, 'c2': 0.3, 'w': 0.9}

optimizer = GlobalBestPSO(n\_particles=10, dimensions=2, bounds=bounds, options=options)

best\_cost, best\_pos = optimizer.optimize(fitness\_function, iters=5)

# Extract the best hyperparameters

best\_C = best\_pos[0]

best\_gamma = best\_pos[1]

# Set the SVM classifier with the best hyperparameters

svm.set\_params(C=best\_C, gamma=best\_gamma)

# Train the SVM classifier with the best hyperparameters

svm.fit(X\_train, y\_train)

# Predict the labels for the test set using the trained SVM

y\_pred = svm.predict(X\_test)

# Calculate the accuracy score of the final model

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

## Result:

The accuracy score of the optimized SVM classifier using Particle Swarm Optimization for the given malicious URL dataset is implemented and verified.

**Mini Project: Phase - III**

**Unsupervised Learning – Agglomerative Clustering**

## Aim:

The Objective of this phase is to implement an Unsupervised learning algorithm – Agglomerative Clustering for the given malicious URL dataset.

## Algorithm:

1. Import the necessary libraries: pandas, sklearn.cluster, sklearn.feature\_extraction.text, matplotlib.pyplot, sklearn.decomposition, and scipy.cluster.hierarchy.
2. Load the dataset from a CSV file using pd.read\_csv().
3. Extract the domain names from the dataset.
4. Convert the domain names to numerical features using TF-IDF vectorization with TfidfVectorizer().
5. Perform dimensionality reduction using PCA with PCA(), specifying the desired number of components.
6. Transform the feature matrix to the reduced dimensional space using pca.fit\_transform().
7. Perform Agglomerative Clustering with AgglomerativeClustering(), specifying the number of clusters and linkage method.
8. Obtain the cluster labels using clustering.fit\_predict().
9. Perform hierarchical clustering using linkage() and obtain the linkage matrix.
10. Plot the dendrogram using dendrogram(), specifying the linkage matrix and cluster labels.
11. Customize the plot with appropriate labels and titles.
12. Display the dendrogram using plt.show().

## Program:

import pandas as pd

from sklearn.cluster import AgglomerativeClustering

from sklearn.feature\_extraction.text import TfidfVectorizer

import matplotlib.pyplot as plt

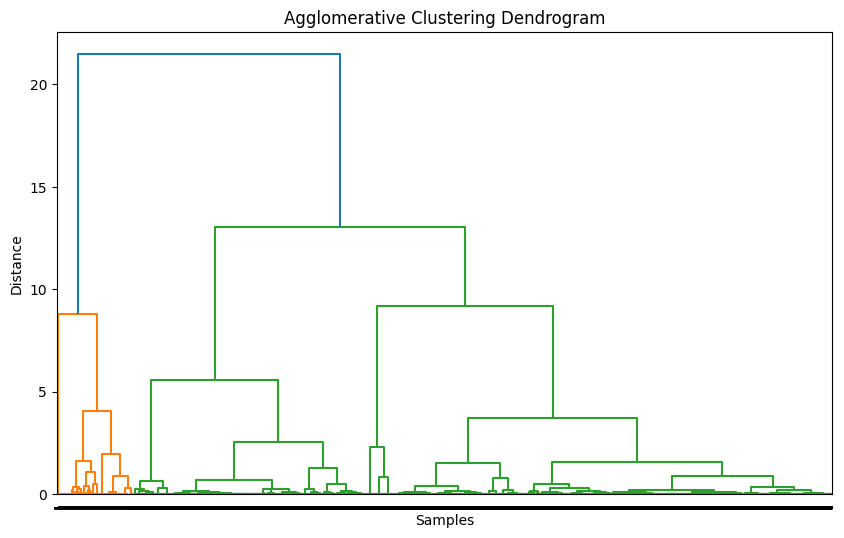
from sklearn.decomposition import PCA

from scipy.cluster.hierarchy import dendrogram, linkage

# Load the dataset

df = pd.read\_csv('url.csv')

## Output:



# Extract the domain names

X = df['Domain']

# Convert domain names to numerical features using TF-IDF vectorization

vectorizer = TfidfVectorizer()

X\_features = vectorizer.fit\_transform(X)

# Perform dimensionality reduction using PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_features.toarray())

# Perform agglomerative clustering

clustering = AgglomerativeClustering(n\_clusters=2, linkage='ward')

clusters = clustering.fit\_predict(X\_pca)

# Perform hierarchical clustering and obtain the linkage matrix

Z = linkage(X\_pca, method='ward')

# Plot dendrogram

plt.figure(figsize=(10, 6))

dendrogram(Z, labels=clusters)

plt.xlabel('Samples')

plt.ylabel('Distance')

plt.title('Agglomerative Clustering Dendrogram')

plt.show()

## Result:

The result is a dendrogram that visualizes the Agglomerative Clustering results for the given malicious URL dataset.