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.

.

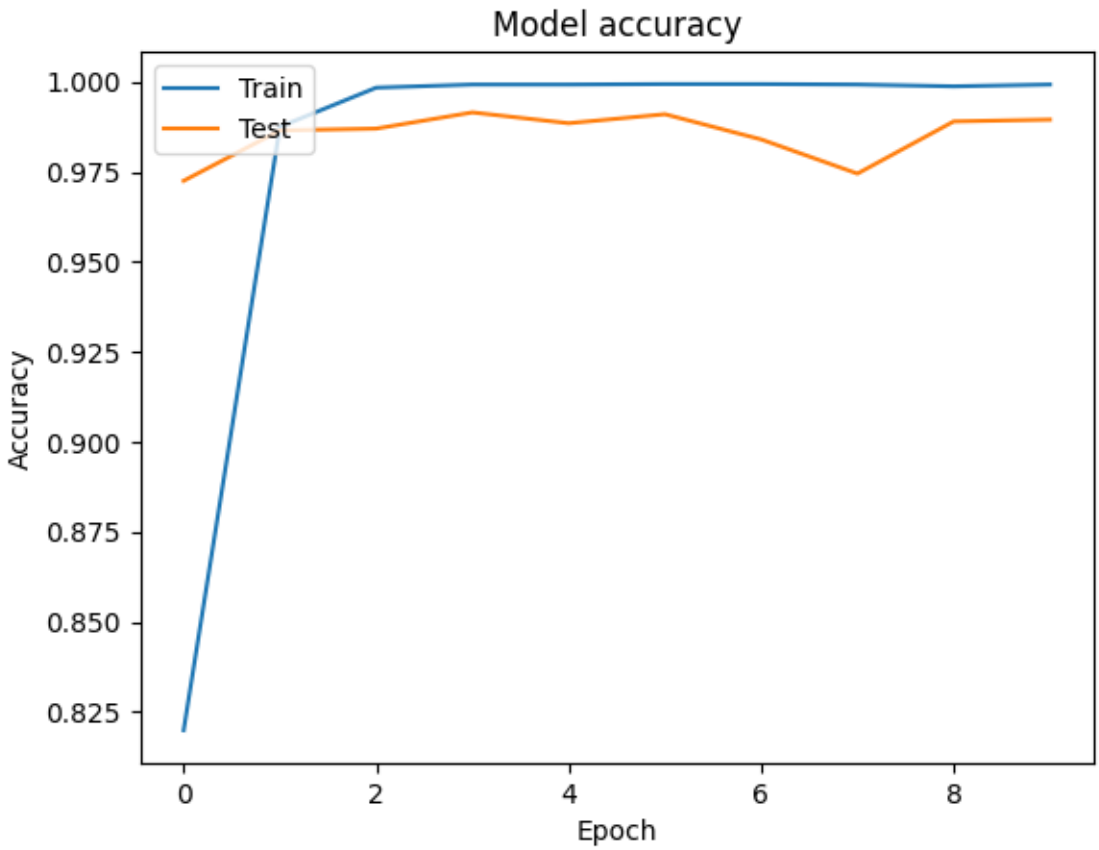
Program:

```
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: 0.436
7 - accuracy: 0.8199 - val_loss: 0.1203 - val_accuracy: 0.9725
Epoch 2/10
250/250 [==============================] - 43s 172ms/step - loss: 0.054
3 - accuracy: 0.9875 - val_loss: 0.0522 - val_accuracy: 0.9865
Epoch 3/10
250/250 [==============================] - 43s 172ms/step - loss: 0.009
5 - accuracy: 0.9984 - val_loss: 0.0533 - val_accuracy: 0.9870
Epoch 4/10
250/250 [==============================] - 43s 171ms/step - loss: 0.003
8 - accuracy: 0.9992 - val_loss: 0.0499 - val_accuracy: 0.9915
Epoch 5/10
250/250 [==============================] - 43s 171ms/step - loss: 0.003
4 - accuracy: 0.9992 - val_loss: 0.0493 - val_accuracy: 0.9885
Epoch 6/10
250/250 [==============================] - 43s 171ms/step - loss: 0.002
1 - accuracy: 0.9994 - val_loss: 0.0480 - val_accuracy: 0.9910
Epoch 7/10
250/250 [==============================] - 43s 171ms/step - loss: 0.002
1 - accuracy: 0.9994 - val_loss: 0.0796 - val_accuracy: 0.9840
Epoch 8/10
250/250 [==============================] - 43s 172ms/step - loss: 0.005
1 - accuracy: 0.9992 - val_loss: 0.0771 - val_accuracy: 0.9745
Epoch 9/10
250/250 [==============================] - 43s 172ms/step - loss: 0.004
7 - accuracy: 0.9987 - val_loss: 0.0406 - val_accuracy: 0.9890
Epoch 10/10
250/250 [==============================] - 43s 171ms/step - loss: 0.002
0 - 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()
```


Model accuracy
Train
Test
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
Accuracy
0
2
4
6
8
Epoch

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.

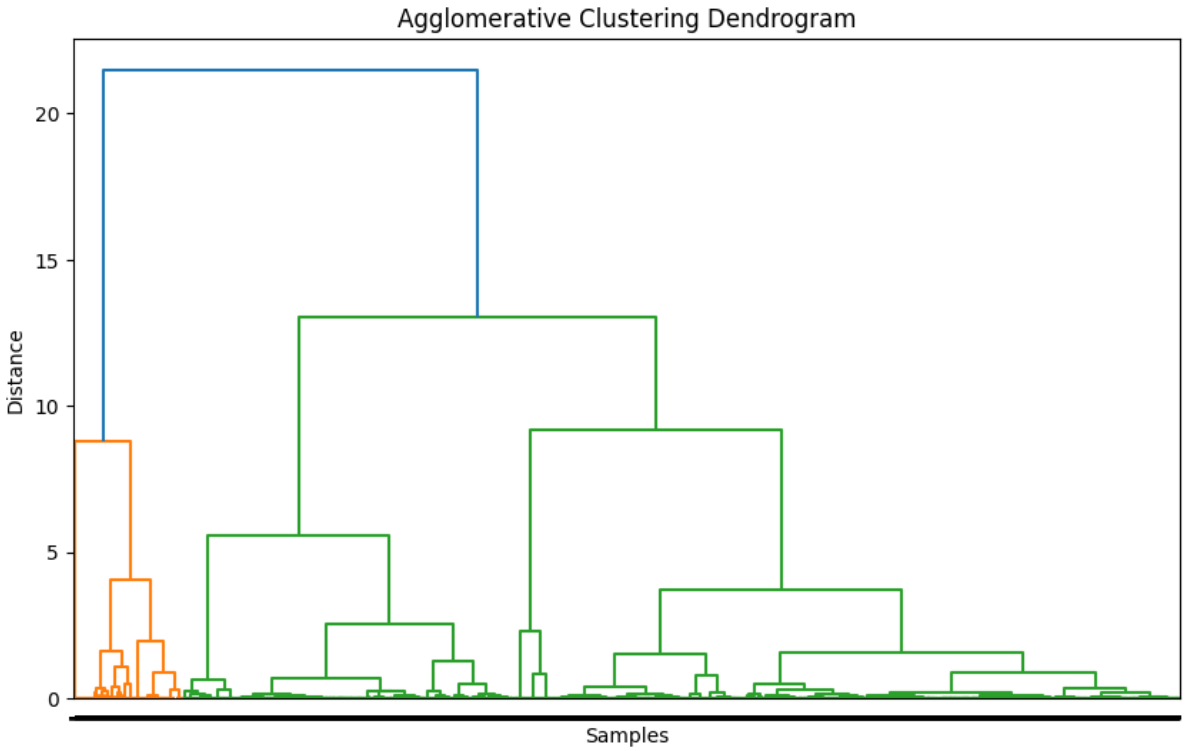
Algorithm:

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:


Agglomerative Clustering Dendrogram
Distance
20
15
10
5
0
Samples

```
# 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.