**LSTM model for anomaly detection in time series data**

1. Data is loaded to simulate reference and actual data related to some system parameters like fuel, velocity, acceleration, engine status, and thruster activity.
2. Data is normalized using Min-Max scaling.
3. The data is divided into sequences with a specified sequence length. Each sequence contains reference data points.
4. LSTM model is constructed with an input shape corresponding to the sequence length and the number of features. It comprises one LSTM layer followed by dense layer.
5. The LSTM model is trained using the reference data sequences.
6. The trained model is used to predict data points for the actual data sequences. Mean Squared Error (MSE) is calculated between the predicted and actual data points. Anomalies are detected based on a threshold derived from the MSE values.
7. Detected anomalies are visualized along with the actual and reference data for each parameter.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from sklearn.metrics import confusion\_matrix, f1\_score, roc\_auc\_score, log\_loss, mean\_squared\_error, mean\_squared\_log\_error

np.random.seed(0)

num\_samples = 1000

fuel = np.random.rand(num\_samples) \* 100

velocity = np.random.rand(num\_samples) \* 5000

acceleration = np.random.rand(num\_samples) \* 100

engine\_status = np.random.randint(0, 2, size=num\_samples)

thruster\_activity = np.random.rand(num\_samples) \* 100

# Reference data

historical\_data = pd.DataFrame({

'fuel': fuel,

'velocity': velocity,

'acceleration': acceleration,

'engine\_status': engine\_status,

'thruster\_activity': thruster\_activity

})

# Normalize the data

scaler = MinMaxScaler()

scaled\_data = scaler.fit\_transform(historical\_data)

# sequence length

sequence\_length = 10

# prepare the data in sequences

sequences = []

for i in range(len(scaled\_data) - sequence\_length):

sequences.append(scaled\_data[i:i+sequence\_length])

# Convert the list of sequences to numpy array

sequences = np.array(sequences)

# Split into input (X) and output (y) variables

X = sequences[:, :-1]

y = sequences[:, -1]

# Reshape input data to be 3D [samples, timesteps, features]

n\_features = X.shape[2]

X = X.reshape((X.shape[0], X.shape[1], n\_features))

# Build the LSTM model

model = Sequential()

model.add(LSTM(50, activation='relu', input\_shape=(sequence\_length-1, n\_features)))

model.add(Dense(n\_features))

model.compile(optimizer='adam', loss='mse')

# Fit the model

model.fit(X, y, epochs=30, batch\_size=16, validation\_split=0.2, verbose=1)

# actual data

num\_actual\_samples = 1000

actual\_fuel = np.random.rand(num\_actual\_samples) \* 100

actual\_velocity = np.random.rand(num\_actual\_samples) \* 5000

actual\_acceleration = np.random.rand(num\_actual\_samples) \* 100

actual\_engine\_status = np.random.randint(0, 2, size=num\_actual\_samples)

actual\_thruster\_activity = np.random.rand(num\_actual\_samples) \* 100

actual\_data = pd.DataFrame({

'fuel': actual\_fuel,

'velocity': actual\_velocity,

'acceleration': actual\_acceleration,

'engine\_status': actual\_engine\_status,

'thruster\_activity': actual\_thruster\_activity

})

# Normalize actual data

scaled\_actual\_data = scaler.transform(actual\_data)

# Prepare the actual data in sequences

actual\_sequences = []

for i in range(len(scaled\_actual\_data) - sequence\_length):

actual\_sequences.append(scaled\_actual\_data[i:i+sequence\_length])

# Convert the list of sequences to numpy array

actual\_sequences = np.array(actual\_sequences)

# Reshape input data to be 3D [samples, timesteps, features]

actual\_X = actual\_sequences[:, :-1]

actual\_y = actual\_sequences[:, -1]

actual\_X = actual\_X.reshape((actual\_X.shape[0], actual\_X.shape[1], n\_features))

# Predict anomalies using the trained model

predicted\_data = model.predict(actual\_X)

# Calculate Mean Squared Error (MSE) as anomaly score

mse = np.mean(np.square(predicted\_data - actual\_y), axis=1)

# Threshold for anomaly detection

threshold = np.mean(mse) + 2 \* np.std(mse)

# Detect anomalies

anomalies = actual\_data.iloc[sequence\_length:][mse > threshold]

# Predicted labels

predicted\_labels = mse > threshold

# True labels

true\_labels = mse > threshold

# Compute F1 Score

f1 = f1\_score(true\_labels, predicted\_labels)

# Compute ROC AUC Score

roc\_auc = roc\_auc\_score(true\_labels, predicted\_labels)

# Compute Log Loss

loss = log\_loss(true\_labels, predicted\_labels)

# Compute Mean Squared Error

mse = mean\_squared\_error(actual\_y, predicted\_data)

# Compute Root Mean Squared Logarithmic Error

rmsle = np.sqrt(mean\_squared\_log\_error(actual\_y, predicted\_data))

print("F1 Score:", f1)

print("ROC AUC Score:", roc\_auc)

print("Log Loss:", loss)

print("Mean Squared Error:", mse)

print("Root Mean Squared Logarithmic Error:", rmsle)

# Visualize detected anomalies

for param in ['fuel', 'velocity', 'acceleration', 'engine\_status', 'thruster\_activity']:

plt.figure(figsize=(12, 6))

# Plot actual data

plt.plot(actual\_data.index, actual\_data[param], label='Actual', color='blue')

# Plot reference data

plt.plot(historical\_data.index, historical\_data[param], label='Reference', color='green',linestyle='--')

# Plot detected anomalies

plt.scatter(anomalies.index, anomalies[param], color='red', label='Anomaly')

plt.xlabel('Time')

plt.ylabel(f'{param.capitalize()} Value')

plt.title(f'Anomalies Detected for {param.capitalize()} (LSTM)')

plt.legend()

plt.grid(True)

plt.show()