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

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

from sklearn.metrics import roc\_curve, precision\_recall\_curve

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(100, activation='relu', input\_shape=(sequence\_length-1, n\_features), dropout=0.2, recurrent\_dropout=0.2))

model.add(Dense(n\_features))

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

# Fit the model with potentially more epochs and different batch size

model.fit(X, y, epochs=50, batch\_size=32, 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()

# Visualize detected anomalies

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

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

# Get anomalies for the actual parameter

param\_anomalies = anomalies[param]

# Plot detected anomalies

plt.scatter(param\_anomalies.index, param\_anomalies, label=f'{param.capitalize()} Anomaly', alpha=0.7)

# Annotate each anomaly group with its value and contextual information

grouped\_anomalies = param\_anomalies.groupby(param\_anomalies.diff().fillna(0).ne(0).cumsum()).agg(['first', 'last'])

if not grouped\_anomalies.empty:

for idx, anomaly\_group in grouped\_anomalies.iterrows():

start\_index = anomaly\_group.index[0]

end\_index = anomaly\_group.index[-1]

if end\_index in param\_anomalies.index:

anomaly\_value = param\_anomalies[end\_index] # Get the contextual information for the anomaly

context\_info = actual\_data.loc[start\_index:end\_index, param].mean()

plt.text(end\_index, anomaly\_value, f'{param.capitalize()} Group Anomaly: {anomaly\_value:.2f}\nContextual Info: {context\_info:.2f}', color='red', fontsize=8, verticalalignment='bottom')

plt.xlabel('Time')

plt.ylabel('Parameter Value')

plt.title('Grouped Anomalies Detected (LSTM)')

plt.legend()

plt.grid(True)

plt.show()

# 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]

# Plot anomaly scores and dynamic threshold for each parameter

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

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

time\_indices = actual\_data.index[sequence\_length:sequence\_length+len(mse)]

plt.plot(time\_indices, mse, label='Anomaly Score', color='orange')

dynamic\_threshold = [np.mean(mse[:i]) + 2 \* np.std(mse[:i]) for i in range(1, len(mse) + 1)]

plt.plot(time\_indices, dynamic\_threshold, label='Dynamic Threshold', color='green', linestyle='--')

plt.xlabel('Time')

plt.ylabel('Anomaly Score')

plt.title(f'Anomaly Scores and Dynamic Threshold for {param.capitalize()} Over Time (LSTM)')

plt.legend()

plt.grid(True)

plt.show()

# Plot anomaly scores and dynamic threshold for all parameters

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

time\_indices = actual\_data.index[sequence\_length:sequence\_length+len(mse)]

plt.plot(time\_indices, mse, label='Anomaly Score', color='orange')

dynamic\_threshold = [np.mean(mse[:i]) + 2 \* np.std(mse[:i]) for i in range(1, len(mse) + 1)]

plt.plot(time\_indices, dynamic\_threshold, label='Dynamic Threshold', color='green', linestyle='--')

plt.xlabel('Time')

plt.ylabel('Anomaly Score')

plt.title('Anomaly Scores and Dynamic Threshold Over Time (LSTM)')

plt.legend()

plt.grid(True)

plt.show()

predicted\_values = predicted\_data.flatten()

# predicted\_values and mse have the same size

min\_size = min(len(predicted\_values), len(mse))

predicted\_values = predicted\_values[:min\_size]

mse = mse[:min\_size]

# Calculate anomaly scores

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

# Determine threshold for anomaly detection

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

# Detect anomalies

predicted\_anomalies = mse > threshold

# Calculate True Positive Rate (TPR) and False Positive Rate (FPR) for ROC curve

fpr, tpr, \_ = roc\_curve(true\_labels, mse)

# Calculate Precision and Recall for Precision-Recall curve

precision, recall, \_ = precision\_recall\_curve(true\_labels, mse)

# Plot ROC Curve

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', label='ROC Curve')

plt.plot([0, 1], [0, 1], color='red', linestyle='--')

plt.xlabel('False Positive Rate (FPR)')

plt.ylabel('True Positive Rate (TPR)')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend()

plt.grid(True)

plt.show()

# Plot Precision-Recall Curve

plt.figure(figsize=(8, 6))

plt.plot(recall, precision, color='green', label='Precision-Recall Curve')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve')

plt.legend()

plt.grid(True)

plt.show()