Data Science Task

Task: Providing Authors with Co-author suggestions

SUBMISSION BY :

ANJALI KEDIA

anjali.kedia2021@vitstudent.ac.in

+91 9560427614

Part 3: Cloud Chronicles

I attempted to upload the model on cloud, by using AWS Lambda.

I created the lamda_function.py, model.pth, requirements.txt files and uploaded them to the AWS console.

```
import json
import torch
import pandas as pd
from torch_geometric.data import Data
import networkx as nx
from torch_geometric.nn import GCNConv
import torch.nn as nn
import torch.nn.functional as F
import numpy as np

# Define your GNN model class (same as your existing code)

class GNNModel(nn.Module):

    def __init__(self, num_nodes, num_features, num_classes):

        super(GNNModel, self).__init__()

        self.conv1 = GCNConv(num_features, 64)  # GCN layer with 64 output channels
        self.conv2 = GCNConv(64, num_classes)   # GCN layer with output size equal to the number of
classes

    def forward(self, data):

        x, edge_index = data.x, data.edge_index
# Apply the first GCN layer followed by a ReLU activation

        x = self.conv1(x, edge_index)

        x = F.relu(x)

        # Apply the second GCN layer

        x = self.conv2(x, edge_index)
        return x

# Load the model

num_nodes = len(all_authors)  # The total number of authors in your dataset
num_features = 64  # Adjust as needed based on your node features
num_classes = 1  # Adjust to the number of classes in your classification task
model = GNNModel(num_nodes, num_features, num_classes)
model.load_state_dict(torch.load('model.pth'))
model.eval()

# Create a Lambda handler function

def lambda_handler(event, context):

    print('Hello world')

    print(ev)

    return {

        "message" :"Received"

    }

    try:

        # Parse the author_id from the event

        author_id = event['queryStringParameters']['id']



        # Perform inference

        likeliness = calculate_likeliness(author_id)



        # Return the result as JSON
 response = {

            'authorID': author_id,

            'likeliness': likeliness,

            'rank': 1  # You can set the rank as needed

        }



        return {

            'statusCode': 200,

            'headers': {

                'Content-Type': 'application/json',

                'Access-Control-Allow-Origin': '*'          },
            'body': json.dumps(response)
        }
    except Exception as e:
        # Handle exceptions and return an error response
        return {
            'statusCode': 500,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({'error': str(e)})
            "message": 'error'
        }


# Implement your inference logic
def calculate_likeliness(author_id):
    try:
        # Load your data and perform necessary preprocessing
        data = Data(edge_index=edge_index)
        data.x = torch.randn(num_nodes, num_features)  # Placeholder node features
        # Perform inference using your model
        with torch.no_grad():
            predictions = model(data)
            predictions = torch.sigmoid(predictions)  # Apply sigmoid activation for probability scores
            predictions = predictions.cpu().numpy()
        # Implement your logic to calculate likeliness
        likeliness_score = get_likeliness_score(predictions, author_id)
        return likeliness_score
    except Exception as e:
        # Handle exceptions and return an error message
        return str(e)

# Implement your custom logic to calculate likeliness

def get_likeliness_score(predictions, author_id):

    try:

        # Find the index corresponding to the provided author_id

        author_index = None

        for i, author_instance in enumerate(df['author_id']):

            if author_instance == author_id:

                author_index = i

                break

        if author_index is not None:

            likeliness_score = predictions[author_index][0]  # Assuming binary classification

            return likeliness_score

        else:

            return "Author not found in the dataset"  # Handle the case when author_id is not found

    except Exception as e:

        # Handle exceptions and return an error message

        return str(e)
```

I also created the API that accepts the author_id as a parameter.

API link:
https://lz962m9kjk.execute-api.eu-north-1.amazonaws.com/prod/authorID_5f9c4_ab08c_ac745_7e911_1a30e

I am facing encoding errors, which I am still working on.

```
{"errorMessage": "'utf-8' codec can't decode byte 0x80 in position 64: invalid start byte", "errorType": "UnicodeDecodeError", "stackTrace": ["  File
\"/var/lang/lib/python3.8/site.py\", line 208, in addsitedir\n    addpackage(sitedir, name, known_paths)\n", "  File \"/var/lang/lib/python3.8/site.py\", line 164, in addpackage\n
for n, line in enumerate(f):\n", "  File \"/var/lang/lib/python3.8/codecs.py\", line 322, in decode\n    (result, consumed) = self._buffer_decode(data, self.errors, final)\n"]}
```

Thank you!