**Relator-Bot**

**Introduction**

This document covers the prerequisites, working, methodology and architecture of Relator-Bot which leverages Graph Convolution Network (GCN) to model the relationships between users based on their Facebook interactions. The Goal of this project is to calculate the compatibility score using the common traits between the users and provide the user with possible matches.

**Project Objectives**

The main objectives of this project are as follows:

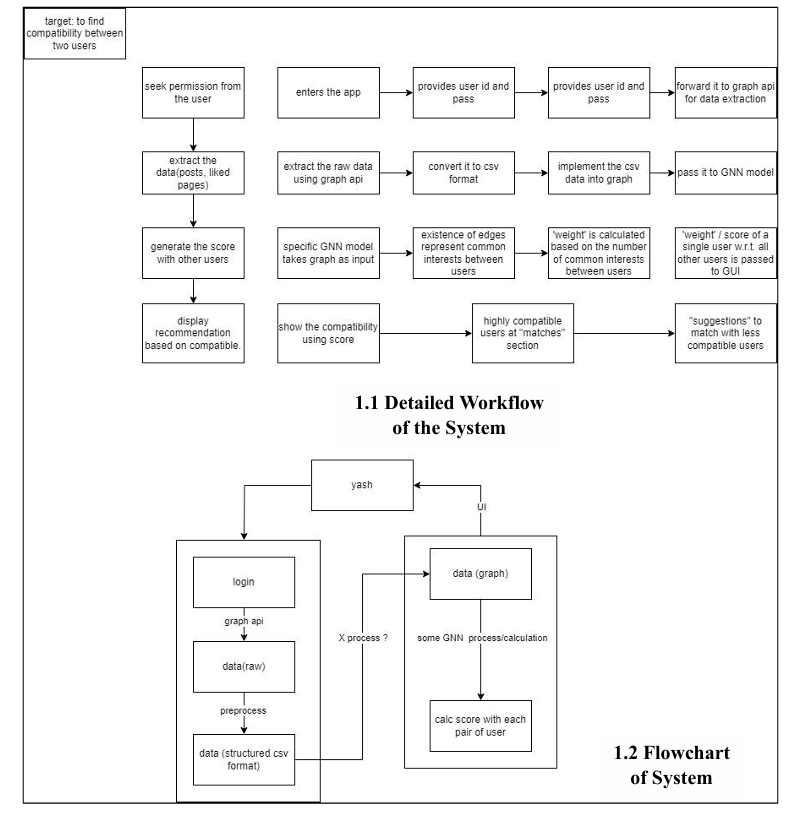
* Develop a program that extract user’s data from Facebook.
* Develop a GCN model that uses user embeddings and calculate the compatibility score/strength between users.
* Use the score to provide the user with possible matches.

**Libraries Used**

The project utilizes several Python libraries for various tasks:

* **random**: It provides functions for generating random numbers and making random choices, useful for creating synthetic data.
* **faker**: Faker is a library for generating fake data such as names, addresses, and email addresses. It's often used for testing and populating databases with dummy information.
* **pandas**: Pandas is a powerful library for data manipulation and analysis. It's particularly useful for working with structured data, such as CSV files, and provides data structures like DataFrame for easy handling of tabular data.
* **torch**: This is the main library for PyTorch, a popular machine learning framework. It's used for building and training neural networks.
* **torch\_geometric**: This library extends PyTorch with additional functionalities for dealing with geometric data, including graph neural networks.
* **networkx**: NetworkX is a library for creating, manipulating, and studying complex networks or graphs.
* **matplotlib.pyplot**: Matplotlib is a plotting library for creating static, interactive, and animated visualizations in Python. The **pyplot** module provides a MATLAB-like interface for creating plots.
* **pyvis**: Pyvis is a Python wrapper for the vis.js library, which is used for creating interactive network visualizations.
* **facebook**: This library provides access to the Facebook Graph API, allowing interaction with Facebook's data and services programmatically.

**System Architecture**



**Methodology**

The methodology of the provided programs can be broken down into several steps:

**1. Data Extraction:**

* 1. **Obtain Access Token**:
  + Log in to Facebook and navigate to the Graph API Explorer tool.
  + Generate an access token with the necessary permissions to access user data. This typically involves selecting the required permissions and clicking on "Generate Access Token".

**1.2 Identify Desired Fields**:

* + The model currently works on the following parameters: “first\_name, gender, birthday, email, profile\_link, categories”
  + The model necessarily requires this fields but additional fields and modifications can be done with few changes in the model’s parameters

**1.3 Test API Request**:

* + Use the Graph API Explorer to execute a test request to retrieve user data.
  + Verify that the requested fields are returned correctly and contain the desired information.

**1.4 Integrate with Python**:

* + Install the Facebook Python library (**facebook**) if not already installed.
  + In your Python script, import the library and use it to interact with the Facebook Graph API.
  + Set up authentication by providing the access token obtained earlier.

**1.5 API Request**:

* + Specify the endpoint for fetching user data (e.g., **'me'** for the current user).
  + Define the fields parameter to include the desired fields of user data.
  + Use the Facebook library to make an API request to retrieve user data.
  + Capture the response, which typically comes in JSON format.

**1.6 Parse and Process Data**:

* + Extract relevant information from the JSON response based on the specified fields.
  + Process the data as needed, such as formatting it for storage or further analysis.

**1.7 Store Data**:

* + Store the extracted user data in a suitable format, such as a CSV file or database for using it in the GCN model.

**2. Test Dataset Generation:**

After identifying the required fields for the model, a program that generates test dataset was developed for testing the model. The program generates a dataset using random values as follows:

**2.1 Import Required Libraries**:

* + Import the necessary libraries: **csv**, **random**, and **faker**.

**2.2 Generate Random Data**:

* + Use the **Faker** library to generate synthetic user data such as names, genders, birthdays, email addresses, etc.
  + Create a function (**generate\_random\_data**) to generate a specified number of random user entries.

**2.3 Define Data Generation Logic**:

* + Define the logic to generate random values for each user attribute, such as name, gender, birthday, age range, location, email, profile link, and categories of interest.
  + Utilize the **random** module to select random values for certain attributes.

**2.4 Write Data to CSV**:

* + Create a function (**write\_to\_csv**) to write the generated user data to a CSV file.
  + Use the **csv.writer** to write data to the CSV file row by row.

**2.5 Specify Output Filename and Location**:

* + Define the filename and location where the CSV file will be saved.
  + Ensure proper handling of file paths to avoid errors.

**2.6 Set Number of Entries**:

* + Determine the desired number of random user entries (**num\_entries**) to be generated.

**3. Graph Construction:**

* A graph representation of the filtered user data is constructed, where users are nodes, and their similarities based on interests are represented as edges.
* The networkx library is employed to build and manipulate the graph structure.

**4. Data set Filtering**

**4.1 Input Parameters:**

* + Define the input parameters for the filtering function: target\_user\_name (the user to filter for) and df (the DataFrame containing user data).

**4.2 User Row Extraction:**

* + Use pandas to filter the DataFrame (df) to extract the row corresponding to the target user name (target\_user\_name).
  + Check if the user row exists in the DataFrame.

**4.3 Conditional Filtering:**

* + If the user row exists:
    - Extract the target location and age range from the user row.
    - Filter the DataFrame (df) based on the target location and age range criteria.
  + If the user row does not exist:
    - Print a message indicating that the target user was not found in the dataset.
    - Return None or an empty DataFrame.

**4.4 Optional Filtering:**

* + Optionally, perform additional filtering based on the size of the filtered DataFrame.
  + If the filtered DataFrame has more than 500 rows:
    - Calculate the count of categories for each user and add a new column (categories\_count) to the DataFrame.
    - Sort the DataFrame based on the category count in descending order.
    - Keep only the top 500 rows with the highest category counts.

**4.5 Output:**

* + Return the filtered DataFrame containing users matching the specified criteria.
  + If the target user is not found, return None or an empty Data Frame**.**

**5. Graph Convolutional Network (GCN) Analysis:**

**5.1 Input users name and filter the data:**

* + Prompt the user to input a specific user name for analysis.
  + Read the user data from a CSV file containing information about users and their interests.
  + Use a filtering function to extract relevant data for the specified user from the dataset.

**5.2 Graph Construction:**

* + Create an empty graph using the NetworkX library to represent user connections and similarities.
  + Populate the graph with nodes representing users and edges representing similarities between users based on shared interests.
  + Calculate similarity scores between users and add weighted edges to the graph.

**5.3 GCN Model Initialization:**

* + Define the architecture of the Graph Convolutional Network (GCN) model using the PyTorch library.
  + Initialize the GCN model with input, hidden, and output dimensions.
  + Generate node features for each user based on the GCN model's initial random embeddings.
  + Execute the GCN model on the constructed graph to learn user embeddings and capture user similarities.
  + Update node features through the GCN layers to refine user representations.

**5.4 Graph Visualization:**

* + Visualize the constructed graph using NetworkX to display user connections and similarities.
  + Show the graph plot using Matplotlib for interactive exploration.

**5.5 Node-Level Analysis:**

* + Extract information about the specified user's neighbours and their compatibility scores.
  + Display compatibility scores between the specified user and its neighbours.

**5.6 Interactive Visualization:**

* + Generate an interactive network visualization using the Pyvis library, allowing users to explore user connections and interests dynamically.

**5.7 Output Generation:**

* + Save the generated network visualization as an HTML file for sharing or further analysis.

**GitHub:** [**https://github.com/Ritesh1069/Relator\_Bot.git**](https://github.com/Ritesh1069/Relator_Bot.git)

**Conclusion:** We have successfully completed the implementation of the model, incorporating all the points mentioned in the Objectives.