Documentation: Champhunt Recommendation System

Jiyanshu Dhaka

documentation to explain recommendation system. this system suggests friends based on shared champhunt dataset (such as followers, following, location, etc.).

Data Structure

The data structure includes features:

_id	usedDeals	following	followers	defaultRun	active	email	password	createdDate	modifiedDate	__v	token	firstName	lastName	profilePhoto	locationPermission

Data Structure Overview

Libraries

```
!pip install pymongo
!pip install scikit-learn
```

- **pymongo**: library enables interaction with MongoDB, allowing me to read and write data in champhunt database, which is was big like approx. 700MB in size.
- **scikit-learn**:ML library to find similarity between users based on their connections (like followers, following, locations etc.).

Code

```
from pymongo import MongoClient
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

- **MongoClient**: Connects to the MongoDB server.
- **TfidfVectorizer** and **cosine_similarity**: Used for checking text similarities .

Connecting to the Database

Next, I connect to my MongoDB database:

```
client = MongoClient("mongodb+srv://champhunt:champhunt_2424@cluster0.
    hk3qy.mongodb.net/champhunt_dev?retryWrites=true&w=majority")
db = client["champhunt_dev"]
users_col = db["users"]
```

Here, I connect to the MongoDB server and select the `champhunt_dev` database and the `users` collection.

Checking Location Similarity

I created a function to check if two users are in the same location:

```
def check_location_similarity(user1, user2):
    loc1 = user1.get('location', '').strip()
    loc2 = user2.get('location', '').strip()
    return 1.0 if loc1 and loc1 == loc2 else 0.0
```

This function checks if the locations of two users match, returning `1.0` if they do, or `0.0` if they do not.

Recommending Friends

function to recommend friends for specific user:

```
def recommend_friends(target_id, all_users):
    recs = []
    target_user = next((user for user in all_users if user["_id"] ==
       target_id), None)
    if not target_user:
        return []
```

This function initializes an empty list called `recs` to hold recommendations and finds the target user using their ID.

Following and Followers

Next, I retrieve the lists of users that the target user follows and who follow them:

```
following = {f['followingUserId'] for f in target_user.get("following",
   [])}
followers = {f['followerUserId'] for f in target_user.get("followers", [])
   }
max_followers = len(followers)
max_following = len(following)
```

I create sets for following and followers and keep track of their counts.

Checking Each Potential Friend

I check each user to see if they could be a potential friend:

```
for friend in all_users:
    if friend["_id"] in following or friend["_id"] == target_id:
        continue
```

I skip users that the target user already follows or the user themselves.

Finding Common Connections

I find common followers and following:

```
common_followers = followers.intersection({f['followerUserId'] for f in
   friend.get("followers", [])})
common_following = following.intersection({f['followingUserId'] for f in
   friend.get("following", [])})
```

This checks which followers and following are shared between the target user and the potential friend.

Location Score

Next, I check if their locations match:

```
loc_score = check_location_similarity(target_user, friend)
```

Calculating the Score

I calculate a score based on the common connections:

```
score = (len(common_followers) + len(common_following) + loc_score) / (
    max_followers + max_following + 1)
```

The score for each potential friend is:

$$\text{score} = \frac{\text{common followers} + \text{common following} + \text{location score}}{\text{max followers} + \text{max following} + 1} \tag{1}$$

the score is balanced against the maximum possible values, to prevent overestimation.

Adding Recommendation

If the score is positive, I add the friend to the recommendations list:

```
if score > 0:
    recs.append((friend, score))
```

Sorting and Returning Recommendations

Finally, I sort the recommendations by score and return the top 100:

```
recs.sort(key=lambda x: x[1], reverse=True)
return recs[:100]
```

User Interaction

I retrieve user data from the database and print the names:

```
users = list(users_col.find())
print("Available User Names:")
for user in users:
    print(f"{user.get('firstName')} {user.get('lastName')}")
```

Asking for Input

I let the user type in a name to get recommendations:

```
while True:
    user_input = input("\nEnter User Name (first and last) or 'exit' to
        quit: ").strip()
    if user_input.lower() == 'exit':
        break
```

The loop continues until the user types 'exit'.

Finding User ID

Next, I find the user ID based on the input:

```
name_parts = user_input.split()
user_id = None
for user in users:
    if (user.get('firstName', '').lower() == name_parts[0].lower() and
        user.get('lastName', '').lower() == name_parts[1].lower()):
        user_id = user["_id"]
        break
```

Generating Recommendations

If I find the user ID, I generate and show recommendations:

```
if user_id:
    friend_recs = recommend_friends(user_id, users)
    if friend_recs:
        print(f"Recommendations for User: {user_input}:")
        for friend, score in friend_recs:
            print(f"Name: {friend.get('firstName')} {friend.get('lastName
                ')}, Score: {score:.2f}")
    else:
        print(f"No recommendations for User: {user_input}.")
```

Closing the Connection

Finally, I close the connection to the database:

```
client.close()
```

Example

example of recommendation :

```
Enter User Name (first and last) or 'exit' to quit: Ruchika Patil
Recommendations for User: Ruchika Patil:
Name: Jay Mawari, Score: 0.60
Name: Pratik Nikat, Score: 0.53
Name: Anish Prajapati, Score: 0.47
Name: Amar Chaudhary, Score: 0.47
Name: Hardik Joshi, Score: 0.40
Name: Ronaldo Cristiano, Score: 0.40
Name: Yash Pawar, Score: 0.40
```

This recommendation system uses user connection data and location to suggest friends. The score is based on shared connections,following , follower, location, making recommendation relevant for user.