COMP533 Introduction to Database Systems

Graduate Project

User Identity Exploration and Preference Prediction Using Yelp Dataset

Zheng You (zy24)
Luoqi Wu (lw31)
Yan Xu (yx28)

Contents

1 Problem Description

Yelp is a local-search service publishing crowd-sourced reviews on businesses which heavily influence customers' decision-making and in turn, affecting business performance. The goal of this project is to extract data from the Yelp dataset and to perform data explorations based on practical needs. This dataset is aggregated and provided by Yelp [8].

This dataset is interesting and meaningful from a research perspective. It can reveal insights such as identifying abnormal users whose reviews and ratings might be misleading. Another interesting exploration is to compare the ability of different recommendation strategies. Through analysis over the dataset, we hope to find some interesting conclusions.

1.1 Data

The Yelp dataset consists of 6 JSON formatted files, each of which is composed of one JSON object per line [10]. The 6 JSON files contain information about business profiles, reviews, user profiles, check-in time, tips, and business photos.

Based on the name-value pair nature of JSON objects, fields of interest are easily located. For instance, business data include locations and categories. Hence, it is feasible to find the number of businesses in certain categories within an area. Queries can also be performed over different data such as comparing the average rating based on the review data and the rating in the business profile to see the differences. The final fields of interest are extracted based on exploration needs.

The Yelp dataset is originated in the Yelp Dataset Challenge [9]. There have been 11 rounds of competitions in the past and a great many explorations have been performed over this dataset. Over the past winners, some team predicts potential customers using machine learning and natural language processing techniques [5]. Another team presents new techniques of recurrent neural networks over this dataset [6].

1.2 Database System

The data is loaded into a PostgreSQL database, version 10.5.

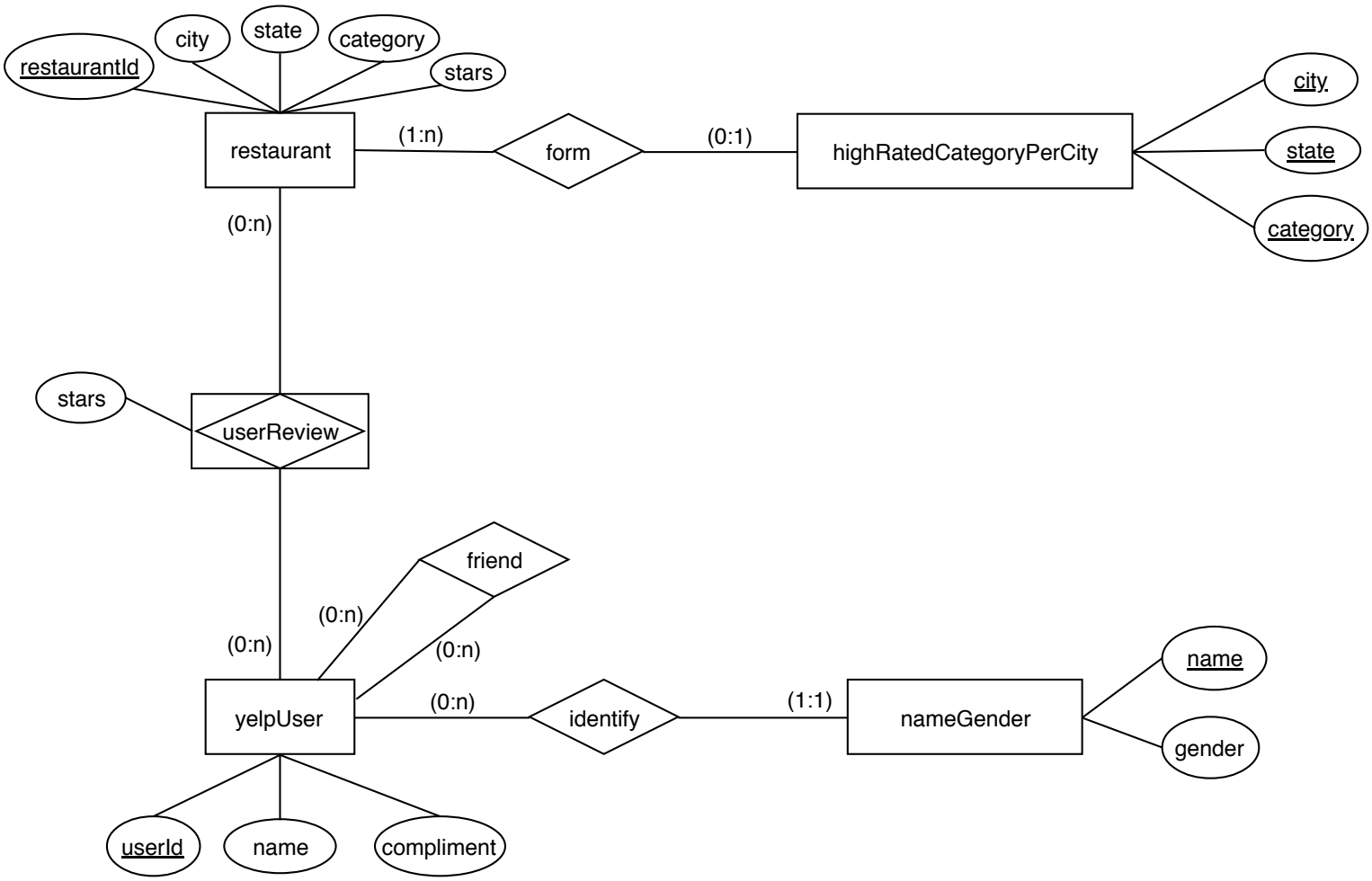
2 Team

This work is completed by the following three people. Everyone is assigned an even amount of work.

Name	NetID	Department	Degree Program
Zheng You	zy24	Computer Science	Master
Luoqi Wu	lw31	Computer Science	Bachelor
Yan Xu	yx28	Chemical and Biomolecular Engineering	PhD

3 Design

3.1 Entity Relationship Diagram



3.2 Entity Descriptions

The following entities and attributes are created for this project.

restaurant - A restaurant. Attributes include:

- restaurantId - A unique sequence value assigned to each restaurant
- city - The city where the restaurant is located
- state - The state where the restaurant is located
- category - The category which the restaurant belongs to
- stars - The rating of the restaurant

highRatedCategoryPerCity - This table contains the highest rated categories for each city. It is derived from the restaurant entity. Attributes include:

- city - A city name
- state - The state where the city is located
- category - One of the top 5 highest rated categories in a city

yelpUser - A Yelp user. Attributes include:

- userId - A unique sequence value assigned to each user
- name - The first name of the user
- compliment - The number of compliments the user received

nameGender - This table contains the gender information for common first names. Attributes include:

- name - A first name
- gender - The gender associated with the first name

Note this table will be imported from outer sources because gender information is not revealed by Yelp. There is a variety of available service on the Internet. Some are free as [7] and some are charged with fees like [3]. We choose to use the baby names from Social Security Card Applications in the United States [1] because of its comprehensive and openness in the public domain. In addition, there are some names which are gender-neutral such as Taylor. We use the frequency information in the baby names dataset and vote for the majority. Say, there are 18,000 times when Taylor is given to baby girls but only 300 times when it is given to baby boys. In this case, we would assign Taylor to be a female name.

3.3 Relationship Descriptions

There are four relationships in the Entity-Relationship Diagram: **form**, **identify**, **userReview**, and **friend**. Relationships **form** and **identify** are one-to-many relationships, which do not need to be independent tables. The **userReview** and **friend** are many-to-many relationships, which will become stand-alone relations.

3.4 Integrity Constraints

Foreign key constraints are used to ensure referential integrity. Specifically,

- The city, state, and category in **highRatedCategoryPerCity** are foreign keys referencing the city, state, and category in **restaurant**.
- The restaurantId in userReview is a foreign key referencing the restaurantId in **restaurant**. The userId in **userReview** is a foreign key referencing the userId in **yelpUser**.
- The userId in **friend** is a foreign key referencing the userId in **yelpUser**.

4 Methods

4.1 Data Import

1. Convert the JSON formatted data into the CSV format.

   We choose to use Python scripts to convert JSON formatted raw dataset into the CSV format. Specifically, there are three raw datasets converted: business, user, and review.

2. Import the JSON formatted data into PostgreSQL to create relations based on a chosen method.

   The PostegreSQL provides the `COPY` command to easily import the CSV data.

4.2 Data Cleaning and Preprocessing

The data cleaning process is conducted in three aspects.

Business: There are 188,593 businesses and a variety of business types in the raw dataset. We are only interested in the restaurant business. In the Python script, we filter out the business which is not a restaurant. Specifically, each restaurant business has the word "Restaurants" in the "categories" attribute in the JSON file. We filter out records which do not contain this word in the attribute. In addition, we are going to do some explorations in the restaurant categories. The "categories" attribute contains multiple classifications for a restaurant. To simplify the problem, we only consider the first category which is listed in this attribute and belongs to the category list published by Yelp [2]. This process eventually reduces the number of restaurants to 55,770.

User: We are going to find groups of people who know each other (we define as friend cycles) through the "friends" attribute in the JSON file. Therefore, it is natural to filter out records whose "friends" attribute is empty. In addition, in order to limit the execution time to find friend cycles, we need to limit the number of users further. In this case, we only consider users with 5,000 or more received compliments. This gives us 771 users and 1,546,047 records of friend relationships, which reduce the execution time of finding friend cycles within 7 minutes.

Review: Since we have limited the number of users considered, it is natural to only consider reviews made by these users regarding the restaurants we consider. This gives us 29,797 reviews.

4.3 Data Explorations

4.3.1 User Identity Exploration

Abnormal User Identification: we try to find abnormal users whose ratings deviate relatively high compared with restaurant actual ratings. This is assuming the attribute stars in the **restaurant** relation is accurate.

1. Calculate users' ratings towards each restaurant a user has visited based on his reviews.
2. Calculate the difference for each user between his rating and the actual rating of each restaurant he has reviewed. Compute the mean and variance of these differences for each user.
3. Identify fraudulent users.

Variance \ Mean	Low	High
Low	Normal	Abnormal
High	Normal	Abnormal

   The table above is inspired by the bias-variance trade-off and the confusion matrix in the machine learning, where the mean is the bias and the variance is the variance. We classify users with high mean values (greater than 1) as abnormal regardless of variance values.

4. Calculate the ratio of abnormal users to the entire user group.

   We have calculated the ratio to be 12.59%.

Gender-rating Relationship: we would like to find the relationship between genders and ratings. Which gender may give higher ratings on average.

1. Add an additional column gender to the **yelpUser** relation.

   The raw dataset does not include the gender information possibly because of privacy issues. We use the relation **nameGender** generated from another source [1] to join with the **yelpUser** table to deduce this information and assign it to users. Among the 771 users we consider, we find 621 males, 69 females, and 81 unknowns. The appearance of unknowns is because their first names are common and are not included in the **nameGender** relation.

2. Calculate the number of reviews and average ratings for male and female. We exclude users whose genders are unknown in this case.

gender	reviewNum	avgStar
M	23622	3.69
F	4362	3.65

   The table above shows that males have roughly five times more reviews than females, even though there are about ten times more males than females. This indicates that females are more likely to give reviews. Additionally, both male and female have the relatively the same ratings, which is a little surprising to us. Originally we thought males are more critical and females are more generous on ratings.

4.3.2 Preference Predication

When a traveler goes to a new city, he is likely to try local popular restaurant categories. In addition, friends tend to share similarities on tastes. During this exploration, we try to compare two different restaurant category recommendation strategies: recommendation by city and recommendation by friends. Generally speaking, these two strategies will generate some categories to a user. And each user will have his own preferred categories. In order to compare the goodness of these two strategies, we use the Jaccard Index [4] to calculate the similarities between a user's personal preference and recommended categories. The higher the index is, the better the recommendation is. Because there are 154 restaurant categories in Yelp's category list [2], we choose 20 as the size of the user preference set and the recommendation set.

1. Identify personal preferred restaurant categories, called set **U**.

   This can be identified by calculating the review count and the average rating a user gives for each category. We use this metric to combine this two factors: $0.5 \times \text{reviewNum} + 0.5 \times \text{avgStar}$. The set **U** consists of the top 20 categories with the highest values under this metric.

2. Identify restaurant categories **recommended by city**.

   (a) Identify the user city.

   We define the user city as the city a user frequents most. The city information is only included in the **restaurant** relation. We use the **review** and the **restaurant** relations to find the city which a user frequents most. Naturally, a user may only review a restaurant after he has visited it.

   (b) Find the recommended restaurant categories based on the user city, called set **C**.

   We have created the relation **highRatedCategoryPerCity** which contains the top 20 highest rated categories per city. A simple join operation between this relation and user cities will gives us set **C**.

(c) Calculate the Jaccard index between the set **U** and the set **C**, where

$$\text{Jaccard Index} = \frac{|U \cap C|}{|U \cup C|}$$

(d) Aggregate the Jaccard Index for all users to compute the average. The higher the average value is, the better the recommendation by city strategy is.

The calculation result is 0.0014, which is a pretty low value. This indicates a bad overlap between set **U** and set **C**. But this is understandable because there is no clear correlation between a user's preference and a city's popular restaurant categories.

3. Identify restaurant categories **recommended by friends**.

(a) Identify user friend cycles.
This is a connected component problem and we employ an imperative SQL program to implement the breadth-first search (bfs) algorithm to solve this problem. The resulting relation contains two attributes, friendCycleId, and userId. Users within a friend cycle will share the same friendCycleId.

(b) Find recommended categories from a user's friend cycle, called set **F**.
When a user's friend cycle and each user's personal preference are determined, the recommended categories are the top 20 categories which are most common in the group.

(c) Calculate the Jaccard index between the set **U** with the set **F**.

$$\text{Jaccard Index} = \frac{|U \cap F|}{|U \cup F|}$$

(d) Aggregate the Jaccard Index for all users to compute the average. The higher the average value is, the better the recommendation by friend strategy is.

The calculation result is 0.1328, which is much better than the result from the city recommendation strategy. This indicates this strategy is a relatively good recommendation strategy.

5 Deliverables

5.1 Tables

5.1.1 Initial Table Definition

restaurant - This table contains the businesses we consider, which are restaurants and whose categories are valid restaurant categories. It is extracted from the raw "yelp_academic_dataset_business.csv" dataset.

yelpUser - This table contains the user who has friends and has 5,000 or more received compliments. It is extracted from the raw "yelp_academic_dataset_user.csv" dataset.

userReview - This table contains the reviews made by users in the **yelpUser** table regarding the restaurants in the **restaurant** table. It is extracted from the raw "yelp_academic_dataset_review.csv" dataset.

nameGenderRaw - This table contains the raw information extracted from the name data source [1]. It contains first names and associated genders along with frequencies for each year from 1880 to 2017. The distributed frequency information will be aggregated in later steps.

friend - This table contains the friend relationship between users in the **yelpUser** table.

5.1.2 User Identity Exploration

userRestaurantRating - This table contains each user's rating towards every restaurant he has reviewed.

userRatingDifference - This table contains each user's difference between his rating and the restaurant's actual rating for every restaurant he has reviewed.

nameGender - This table contains the common first names with associated genders. It is extracted from the table **nameGenderRaw**.

5.1.3 Preference Predication

userCategory - This table contains each user's top 20 favorite categories, which is set **U**.

userCity - This table contains each user's most frequent city. The information is taken from each user's reviews.

highRatedCategoryPerCity - This table contains the top 20 highest rated categories per city.

cityRecommendation - This table contains categories recommended to each user based on his most frequent city, which is set **C**.

cityRecommendationJaccardIndex - This table contains the Jaccard Index of set **U** and set **C** for each user.

friendCycle - This table contains the friend cycle information. Users within the same friend cycle share the same friend cycle Id.

friendRecommendation - This table contains categories recommended to each user based on his friends in the same friend cycle, which is set **F**.

friendRecommendationJaccardIndex - This table contains the Jaccard Index of set **U** and set **F** for each user.

5.2 Key Queries

5.2.1 Initial Table Definition

1. The following query is used to extract each user's string-based comma-separated friend userIds to tuples in table **friend**.

```
INSERT INTO friend(userId, friendId)
  SELECT u.userId, temp.friendId
  FROM yelpUser u,
       unnest(string_to_array(u.friends, ',')) AS temp(friendId);
```

5.2.2 User Identity Exploration

1. The following query is used get the **userRestaurantRating** table.

```
INSERT INTO userRestaurantRating
  SELECT ur.userId, r.restaurantId,
         ROUND(AVG(ur.stars), 2) AS metric
  FROM userReview ur JOIN restaurant r ON ur.restaurantId = r.restaurantId
  GROUP BY ur.userId, r.restaurantId;
```

2. The following query is used get the **userRatingDifference** table.

```
-- Variance is set to 0 for users with only 1 review
WITH tempDifference AS(
  SELECT ur.userId, ROUND(AVG(ABS(ur.userRating - r.stars)), 2) AS avgDiff,
         COALESCE(ROUND(VARIANCE(ABS(ur.userRating - r.stars)), 2), 0.00) AS varDiff
  FROM userRestaurantRating ur JOIN restaurant r ON ur.restaurantId = r.restaurantId
  GROUP BY ur.userId
)
INSERT INTO userRatingDifference
  SELECT userId, avgDiff, varDiff
  FROM tempDifference;
```

3. The following query is used to get the numbers of abnormal users and total users.

```
SELECT (SELECT COUNT(urd.userId)
        FROM userRatingDifference urd
        WHERE urd.avgDiff > 1.00) AS abnormalUser, COUNT (*) AS totalUser
FROM userRatingDifference;
```

4. The following query is used to add the "gender" column to the **yelpUser** table.

```
ALTER TABLE yelpUser ADD COLUMN gender CHAR(1);

WITH genderAssignement AS(
  SELECT y.name, n.gender
  FROM yelpUser y JOIN nameGender n ON y.name = n.name
)
UPDATE yelpUser
SET gender = g.gender
FROM genderAssignement g
WHERE yelpUser.name = g.name;
```

5. The following query is used to get the review counts and the average ratings for male and female genders.

```
SELECT y.gender, COUNT(u.stars) AS reviewNum, ROUND(AVG(u.stars), 2) as avgStar
FROM yelpUser y, userReview u
WHERE y.userId = u.userId AND
      gender IS NOT NULL
GROUP BY y.gender;
```

5.2.3 Preference Predication

1. The following query is used to get the set **U** (**userCategory**). The `PARTITION` command is frequently used to get top-k information, such as determining a user's city (**userCity**) and top rated categories for each city (**highRatedCategoryPerCity**).

```
WITH userCategoryMetric AS(
  SELECT ur.userId, r.category,
         ROUND(0.5 * COUNT(ur.stars) + 0.5 * AVG(ur.stars), 2) AS metric
```

```
  FROM userReview ur JOIN restaurant r ON ur.restaurantId = r.restaurantId
  GROUP BY ur.userId, r.category
)
INSERT INTO userCategory
SELECT temp.userId, temp.category
FROM (SELECT userId, category,
             ROW_NUMBER() OVER (PARTITION BY userId
                                ORDER BY metric DESC) AS rowNum
      FROM userCategoryMetric) AS temp
WHERE temp.rowNum <= 20
ORDER BY temp.userId;
```

2. The following query is used to get the **userCity** table.

```
WITH userCityTotal AS(
  SELECT ur.userId, r.state, r.city, COUNT(ur.reviewId) as numReview
  FROM userReview ur JOIN restaurant r ON ur.restaurantId = r.restaurantId
  GROUP BY ur.userId, r.state, r.city
)
INSERT INTO userCity
SELECT temp.userId, temp.city, temp.state
FROM (SELECT *, ROW_NUMBER() OVER (PARTITION BY userId
                                   ORDER BY numReview DESC) AS rowNum
      FROM userCityTotal) AS temp
WHERE temp.rowNum <= 1
ORDER BY temp.userId;
```

3. The following query is used to get the **highRatedCategoryPerCity** table.

```
WITH restaurantCategory AS(
  SELECT state, city, category,
         ROUND(AVG(stars), 2) AS avgStar, COUNT(*) AS count
  FROM restaurant r
  WHERE city IS NOT NULL AND state IS NOT NULL
  GROUP BY (state, city, category)
),
topCategories AS(
  SELECT state, city, category, avgStar
  FROM (SELECT *, ROW_NUMBER() over (PARTITION BY (state, city)
                                     ORDER BY avgStar DESC) AS rowId
        FROM restaurantCategory) AS s
  WHERE rowId <= 20
)
INSERT INTO highRatedCategoryPerCity
  SELECT state, city, category
  FROM topCategories;
```

4. The following query is used to get the **cityRecommendation** table.

```
INSERT INTO cityRecommendation
SELECT uc.userId, hr.category
FROM userCity uc, highRatedCategoryPerCity hr
WHERE uc.state = hr.state AND uc.city = hr.city;
```

5.3 Code

5.3.1 User Identity Exploration

1. The following function is used to aggregate the distributed frequency information for each first name/gender pair across all years and adopt the vote-for-majority policy to determine each first name's gender. The result is the **nameGender** table.

```
DROP TABLE IF EXISTS nameGenderFreq;
CREATE TEMPORARY TABLE nameGenderFreq(
  name VARCHAR(32),
  gender CHAR(1),
  frequency NUMERIC(5, 4),
  PRIMARY KEY(name, gender, frequency)
);

WITH nameGenderCount AS(
  SELECT name, gender, SUM(frequency) AS individualCount
  FROM nameGenderRaw
  GROUP BY name, gender
),
total AS(
  SELECT name, SUM(individualCount) AS totalCount
  FROM nameGenderCount
  GROUP BY name
)
INSERT INTO nameGenderFreq(name, gender, frequency)
  SELECT g.name, g.gender, ROUND(g.individualCount * 1.0 / t.totalCount, 4) AS ratio
  FROM nameGenderCount g, total t
  WHERE g.name = t.name
  ORDER BY g.name, g.gender;

-- use vote-for-majority policy to determine gender for each first name
CREATE OR REPLACE FUNCTION assignGenderToName() RETURNS VOID AS
$$
DECLARE
  cursor CURSOR FOR SELECT * FROM nameGenderFreq;
  currName VARCHAR(32);
  prevName VARCHAR(32) DEFAULT '-1';
  currGender CHAR(1);
  prevGender CHAR(1) DEFAULT 'X';
  currFreq NUMERIC(5, 4);
  prevFreq NUMERIC(5, 4) DEFAULT -1;
BEGIN
OPEN cursor;
LOOP
  FETCH cursor INTO currName, currGender, currFreq;
  EXIT WHEN NOT FOUND;

  IF currName = prevName THEN
    IF currFreq > prevFreq THEN
      UPDATE nameGender SET gender = currGender WHERE name = currName;
    END IF;
  ELSE
    INSERT INTO nameGender(name, gender) VALUES (prevName, prevGender);
  END IF;

  prevName = currName;
  prevGender = currGender;
  prevFreq = currFreq;
```

```
END LOOP;
-- don't forget the last name
INSERT INTO nameGender(name, gender) VALUES (prevName, prevGender);
CLOSE cursor;
-- remove the redundant first row
DELETE FROM nameGender WHERE name = '-1';
END;
$$
LANGUAGE plpgsql;

SELECT assignGenderToName();
```

5.3.2 Preference Predication

1. The following function is used to calculate the Jaccard Index between set **U** and set **C**. A similar query will be used for the Jaccard Index between set **U** and set **F**.

```
CREATE OR REPLACE FUNCTION getCityRecommendationJaccardIndex(id VARCHAR(32))
RETURNS VOID AS
$$
DECLARE
  intersectionSize INTEGER;
  unionSize INTEGER;
BEGIN
  intersectionSize = (SELECT COUNT(*)
                      FROM (SELECT u.category
                            FROM userCategory u
                            WHERE u.userId = id
                            INTERSECT
                            SELECT c.cityCategory
                            FROM cityRecommendation c
                            WHERE c.userId = id) AS temp);
  unionSize = (SELECT COUNT(*)
               FROM (SELECT u.category
                     FROM userCategory u
                     WHERE u.userId = id
                     UNION
                     SELECT c.cityCategory
                     FROM cityRecommendation c
                     WHERE c.userId = id) AS temp);

  INSERT INTO cityRecommendationJaccardIndex
  VALUES(id, ROUND(intersectionSize * 1.0 / unionSize));
END;
$$
LANGUAGE plpgsql;

SELECT getCityRecommendationJaccardIndex(userId)
FROM (SELECT DISTINCT u.userId
      FROM userReview u) AS temp;
```

2. The following function is used to construct the table **friendCycle**. It uses the Breadth-First Search (BFS) algorithm to find connected components. Specifically, indices are used to accelerate the query.

```
-- use index to make it run faster
CREATE INDEX yelpUserId ON yelpUser(userId);
CREATE INDEX friendUserId ON friend(userId);
CREATE INDEX friendFriendId ON friend(friendId);
```

```
-- table to hold unvisited users
DROP TABLE IF EXISTS unVisited;
CREATE TEMPORARY TABLE unVisited(
  userId VARCHAR(32),
  PRIMARY KEY(userId)
);

-- table to hold users of one block
DROP TABLE IF EXISTS block;
CREATE TEMPORARY TABLE block(
  userId VARCHAR(32),
  PRIMARY KEY(userId)
);

-- getFriendCycles() function
CREATE OR REPLACE FUNCTION getFriendCycles()
RETURNS TABLE(friendCycleId INTEGER, userId VARCHAR(32)) AS
$$
DECLARE
  unVisitedNum INTEGER;
  blockSizeBefore INTEGER;
  blockSizeAfter INTEGER;
  blockId INTEGER DEFAULT 1;
BEGIN
  -- mark all users as unvisited
  INSERT INTO unVisited SELECT u.userId FROM yelpUser u;
  unVisitedNum = (SELECT COUNT(*) FROM unVisited);

  WHILE unVisitedNum > 0
  LOOP
    -- insert a user into the block table
    INSERT INTO block SELECT u.userId FROM unVisited u LIMIT 1;

    -- loop until all friended users are inserted into the block table
    blockSizeBefore = -1;
    blockSizeAfter = 0;
    WHILE blockSizeBefore != blockSizeAfter
    LOOP
      -- update blockSizeBefore
      blockSizeBefore = (SELECT COUNT(*) FROM block);

      INSERT INTO block
        SELECT DISTINCT u.userId
        FROM unVisited u
        WHERE (u.userId IN (SELECT f.friendId
                            FROM friend f, block b
                            WHERE f.userId = b.userId) OR
               u.userId IN (SELECT f.userId
                            FROM friend f, block b
                            WHERE f.friendId = b.userId))
              AND u.userId NOT IN (SELECT b.userId FROM block b);

      -- remove inserted users
      DELETE FROM unVisited u
      WHERE u.userId IN (SELECT b.userId FROM block b);

      -- update blockSizeAfter
      blockSizeAfter = (SELECT COUNT(*) FROM block);
    END LOOP;
```

```
    -- we are only interested in friend cycle with more than 1 person
    IF blockSizeAfter > 1 THEN
      RAISE NOTICE 'Friend Cycle %', blockId;
      RETURN QUERY
        SELECT blockId, u.userId
        FROM yelpUser u, block b
        WHERE u.userId = b.userId
        ORDER BY blockId;
      blockId = blockId + 1;
    END IF;

    -- clear block table
    TRUNCATE TABLE block;

    -- update unVisitedNum
    unVisitedNum = (SELECT COUNT(*) FROM unVisited);
  END LOOP;
END;
$$
LANGUAGE plpgsql;

DROP TABLE IF EXISTS friendCycle;
CREATE TABLE friendCycle(
  friendCycleId INTEGER,
  userId VARCHAR(32),
  PRIMARY KEY(friendCycleId, userId)
);

INSERT INTO friendCycle
  SELECT * FROM getFriendCycles();
```

3. The following function is used to construct the table **friendRecommendation**.

```
-- id is an input userId
CREATE OR REPLACE FUNCTION getFriendRecommendation(id VARCHAR(32))
RETURNS VOID AS
$$
DECLARE
BEGIN
  WITH friendCategory AS(
    SELECT uc.category
    FROM friendCycle f, userCategory uc
    WHERE f.userId = uc.userId AND
          uc.userId != id AND
          f.friendCycleId = (SELECT f.friendCycleId
                             FROM friendCycle f
                             WHERE id = f.userId)
  ),
  categoryCount AS(
    SELECT category, COUNT(category) AS COUNT
    FROM friendCategory
    GROUP BY category
    ORDER BY COUNT(category) DESC
  )
  INSERT INTO friendRecommendation(userId, friendCategory)
    SELECT id, c.category
    FROM categoryCount c
    LIMIT 20;
END;
$$
```

```
LANGUAGE plpgsql;

SELECT getFriendRecommendation(userId)
FROM friendCycle;
```

4. The following function is used to calculate the Jaccard Index between set **U** and set **F**.

```
CREATE OR REPLACE FUNCTION getFriendRecommendationJaccardIndex(id VARCHAR(32))
RETURNS VOID AS
$$
DECLARE
  intersectionSize INTEGER;
  unionSize INTEGER;
BEGIN
  intersectionSize = (SELECT COUNT(*)
                      FROM (SELECT u.category
                            FROM userCategory u
                            WHERE u.userId = id
                            INTERSECT
                            SELECT f.friendCategory
                            FROM friendRecommendation f
                            WHERE f.userId = id) AS temp);
  unionSize = (SELECT COUNT(*)
               FROM (SELECT u.category
                     FROM userCategory u
                     WHERE u.userId = id
                     UNION
                     SELECT f.friendCategory
                     FROM friendRecommendation f
                     WHERE f.userId = id) AS temp);

  INSERT INTO friendRecommendationJaccardIndex
  VALUES(id, ROUND(intersectionSize * 1.0 / unionSize, 2));
END;
$$
LANGUAGE plpgsql;

SELECT getFriendRecommendationJaccardIndex(userId)
FROM friendCycle;
```

5.4 Files

business_json_to_csv.py - This file contains the code to convert the "yelp_academic_dataset_business.csv" JSON file to CSV format. Specifically, only businesses which are restaurants and whose categories can be determined are preserved in the final result.

json_to_csv.py - This file contains the code to convert the "yelp_academic_dataset_user.csv" and "yelp_academic_dataset_review.csv" JSON files to CSV format. There is no additional processing.

name_to_csv.py - This file contains the code to process the text files from the baby name dataset [1] to a CSV file.

6 Discussion

This dataset contains a large number of records and is given in JSON format, which presents some challenges.

6.1 Design Decisions

In this project, the design decisions are centered about how to represent the JSON data in a relational manner and how to perform proper filtering to reduce the amount of data.

The JSON formatted data is made up of name-value pairs. The names can be naturally used for columns in relations. Therefore, it is necessary to extract all the names in the JSON data. In some cases, it is cumbersome to become every name from the JSON data to be an individual column. It would make sense to combine come names into one column in tables.

In other cases, it makes sense to make separate tables for columns with multiple values. For instance, a user has a column "friends", which is a string-based comma-separated friend userIds. It is easier to process if we extract those friend userIds to multiple tuples in another table **friend**.

6.2 Challenges

The challenges can be categorized into two aspects: data preprocessing and data explorations.

First, we need to become familiar with the JSON format data provided by Yelp. Then, we employ Python scripts to extract the data and perform necessary filtering to reduce the data volume. The filtering work is not determined in the first place. We have performed some experiments regarding the result data size to finalize the eventual threshold. Finally, we use SQL commands to extract fields of interest and create separate tables.

In data explorations, there are some answers which are not easy to get through declarative SQL queries. Therefore, we implement some imperative SQL functions to accomplish some complex tasks such as finding friend cycles. We have used as many as possible declarative SQL queries in these functions to accelerate the execution speed. Specifically, we have created indices to make find friend cycles faster.

6.3 Assumptions

Each restaurant business in the dataset will have the "Restaurants" keyword in its "categories" value in the JSON data. If a certain business does not contain this keyword in its "categories" value, it is not a restaurant and will not be considered in later explorations.

6.4 Next Time / Next Steps

There are three aspects we believe could be improved.

1. The Yelp Dataset is somewhat skewed in the sense that it does not provide comprehensive information. For instance, it rarely provides information about California and Texas, which have a large number of populations and restaurants. We would request more comprehensive data in the future.

2. The metric to determine abnormal users is somewhat arbitrary. Currently, we define users whose difference means are great than 1 star to be abnormal. The threshold could be studied further to be more accurate.

3. In this project, we only extract one category for each restaurant. In fact, each restaurant typically has more categories. We could include all categories for each restaurant to have a more accurate prediction on user preferences.

7 Conclusion

Overall, we believe this project is successful. We are able to extract data from JSON formatted file and import them into PostgreSQL. In addition, we have performed meaningful data explorations and gained more understandings both in declarative SQL and imperative SQL. The challenges we met are not unexpected and we overcome these obstacles smoothly. Finally, we have learned meaningful lessons in this project and will adopt them for future work.

Reference

[1] *Baby Names from Social Security Card Applications - National Level Data*, URL: https://catalog.data.gov/dataset/baby-names-from-social-security-card-applications-national-level-data.

[2] *Category List*, URL: https://www.yelp.com/developers/documentation/v3/category_list.

[3] *Gender API*, URL: https://gender-api.com/.

[4] *Jaccard index*, URL: https://en.wikipedia.org/wiki/Jaccard_index.

[5] Li, R. et al., "CORALS: Who are My Potential New Customers? Tapping into the Wisdom of Customers' Decisions", in: (2017).

[6] Ming, Y. et al., "Understanding hidden memories of recurrent neural networks", in: *arXiv preprint arXiv:1710.10777* (2017).

[7] *SexMachine PyPi*, URL: https://pypi.org/project/SexMachine/.

[8] *Yelp Dataset*, URL: https://www.yelp.com/dataset.

[9] *Yelp Dataset Challenge*, URL: https://www.yelp.com/dataset/challenge.

[10] *Yelp Dataset Documentation*, URL: https://www.yelp.com/dataset/documentation/main.