

## **Restaurant recommendation system based on cuisine similarity**

Lechen Zhang

### **I. Motivation**

It's convenient today to search for your desired meal and restaurants on Yelp or other restaurant apps, but in many cases, we don't really know what we want. For those who felt exhausted after everyday work, it's impossible to click into every restaurant and select from all cuisines.

Therefore, I'm planning to design a restaurant recommendation system, so that users can get a cuisine recommendation list based on their preference of other restaurants before. Users can also type in their additional requirements to filter the recommendation, like price, pungency or ingredient.

### **II. Project code**

<https://github.com/orange0629/si507project>

### **III. Data sources**

#### **LEVEL of challenge: 12**

- For my first source: 4 pts: Web API you haven't used before that requires API key or HTTP Basic authorization
- For my second source: 8 pts: Crawling [and scraping] multiple pages in a site you haven't used before

#### **1. TripAdvisor search result**

The first data source is used for retrieving the list of restaurants. To get an abundant dataset, I planned to collect the restaurant data from New York City. Hence, the "geo" parameter here should be 60763. This function is not a public API of the website. I found this method by capturing packages; hence the return format is HTML.

**Base url:** <https://www.tripadvisor.com/RestaurantSearch>

**Parameter:**

```
parameter_dictionary = {'Action': 'PAGE', 'ajax': '1', 'availSearchEnabled': 'false', 'sortOrder': 'popularity',  
                        'geo': '60763', 'itags': '10591', 'geobroaden': 'false', 'o': 'a'+str(idx)}
```

**Return format:** HTML

**Available records:** 8897 restaurants in 297 pages

**Retrieved records:** 1020 restaurants in 34 pages

**Fields:** The only useful field here is the URLs of all these restaurants.

**Access method:** First use requests.get() to cache the html search result files. The URL

needs authorization, hence I added a header value to the requests.get() function to simulate a user browser. Then, crawl the URLs by BeautifulSoup and store in a list.

**Code:** In the Part 1 of *Step1\_Data\_Caching.ipynb*

**Cache stored in:** cache/rest\_list


























 NY_0.html	2022/12/4 19:50	Chrome HTML D...	302 KB	Sort by:
 NY_30.html	2022/12/4 19:51	Chrome HTML D...	296 KB	• Ranking
 NY_60.html	2022/12/4 19:51	Chrome HTML D...	291 KB	• Name
 NY_90.html	2022/12/4 19:51	Chrome HTML D...	291 KB	Looking to expand your search outside of New York City? We have suggestions.
 NY_120.html	2022/12/4 19:51	Chrome HTML D...	291 KB	Expand your search.
 NY_150.html	2022/12/4 19:51	Chrome HTML D...	289 KB	 <b>61. Momosan Ramen &amp; Sake</b> <a href="#">URL</a>
 NY_180.html	2022/12/4 19:51	Chrome HTML D...	294 KB	 <b>62. Beyond Sushi</b> <a href="#">URL</a>
 NY_210.html	2022/12/4 19:51	Chrome HTML D...	285 KB	 <b>63. Beyond Sushi</b> <a href="#">URL</a>
 NY_240.html	2022/12/4 19:51	Chrome HTML D...	294 KB	 <b>64. Beyond Sushi</b> <a href="#">URL</a>
 NY_270.html	2022/12/4 19:51	Chrome HTML D...	290 KB	 <b>65. Beyond Sushi</b> <a href="#">URL</a>
 NY_300.html	2022/12/4 19:51	Chrome HTML D...	291 KB	 <b>66. Beyond Sushi</b> <a href="#">URL</a>
 NY_330.html	2022/12/4 19:51	Chrome HTML D...	292 KB	 <b>67. Beyond Sushi</b> <a href="#">URL</a>
 NY_360.html	2022/12/4 19:51	Chrome HTML D...	291 KB	 <b>68. Beyond Sushi</b> <a href="#">URL</a>
 NY_390.html	2022/12/4 19:51	Chrome HTML D...	295 KB	 <b>69. Beyond Sushi</b> <a href="#">URL</a>
 NY_420.html	2022/12/4 19:51	Chrome HTML D...	297 KB	 <b>70. Beyond Sushi</b> <a href="#">URL</a>

Fig. 1 Caching of TOP 1000 Restaurants

## 2. TripAdvisor restaurant pages

After getting the URLs of restaurants, I can now crawl all these restaurant web pages into caches, and extract useful information from the HTML files.

**URLs:** Got from Data source 1 as follows (972 in total):

```
Out[3]: ['https://www.tripadvisor.com/Restaurant_Review-g60763-d1236281-Reviews-Club_A_Steakhouse-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d1878682-Reviews-Olio_e_Piu-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d13504265-Reviews-Boucherie_Union_Square-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d11918545-Reviews-Boucherie_West_Village-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d19245342-Reviews-The_Consulate-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d21410713-Reviews-La_Grande_Boucherie-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d479337-Reviews-Bleecker_Street_Pizza-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d10145683-Reviews-Petite_Boucherie-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d10165896-Reviews-SottoCasa_Pizzeria-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d7284388-Reviews-Mei_Jin_Ramen-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d12426739-Reviews-Piccola_Cucina_Estiatorio-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d12874338-Reviews-Los_Tacos_No_1-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d7617787-Reviews-K_Rico_Steakhouse-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d7814915-Reviews-Loi_Estiatorio-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d4363835-Reviews-Piccola_Cucina_Osteria-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d3603515-Reviews-Spice_Symphony-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d3191154-Reviews-Bagels_Schmear-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d2345886-Reviews-Jungsik-New_York_City_New_York.html',
'https://www.tripadvisor.com/Restaurant_Review-g60763-d14981486-Reviews-Bua_Thai_Ramen_Robata_Grill-New_York_City_New_York.html']
```

**Return format:** HTML

**Available records:** 1020 restaurants in 972 pages

**Retrieved records:** 972 restaurant pages

**Crawled Fields:** [name, URL, rating, food rating, service rating, value rating, cuisine, review number, neighborhood, price level, review tags, reviews, image's url]

**Access method:** First use requests.get() to cache the html search result files. Then, crawl the URLs by BeautifulSoup and store in a DataFrame. Finally output to csv.

**Code:** In Part 3 and Part 4 of *Step1\_Data\_Caching.ipynb*

**Cache stored in:** cache/restaurants (971 files, 1.3 GB in total)

**Output stored in:** cache/restaurant\_dataset.csv (971 rows \* 13 columns)

**Output cache (json version of csv file):** restaurants.json (971 items)

**Note:** The raw crawled data is stored in csv file, because it's well-structured and easier for preprocessing and building the graph structure. After the preprocessing was done, the processed data was stored into restaurant.json. Therefore, the csv file serve as a intermedium in data processing.

```
MINGW64/c/Users/zhang/Desktop/si507project
$ ls cache/restaurants/ -hs
total 1.3G
1.4M 12_Chairs.html
1.3M 15_EAST_Tocqueville.html
1.4M 230_Fifth.html
1.3M 2_Bros_Pizza.html
1.4M 2nd_Avenue_Deli.html
1.3M 44_X.html
1.3M 53rd_6th_Halal.html
1.4M 54_Below.html
1.5M 5_Napkin_Burger.html
1.3M 99_Cent_Fresh_Pizza.html
1.3M ABA_Turkish_Restaurant.html
1.3M ABC_Cocina.html
1.3M ABC_Kitchen.html
1.3M Adrienne_s_Pizzabar.html
1.3M Ai_Fiori.html
1.3M Ajiisen_Ramen.html
1.3M Aldea.html
1.3M Aldo_Sohm_wine_Bar.html
1.3M Almond.html
1.3M Alta.html
1.3M Amarone_Scarlatto.html
1.3M Amelie.html
```

Fig. 2. Restaurant caching evidence

```
{
  "id": 0,
  "Name": "12 Chairs",
  "url": "https://www.tripadvisor.com/Restaurant_Review-g60763-d533199-Reviews-12_Chairs-New_York_City_New_York.html",
  "Rating": 4.5,
  "Rating_food": 4.5,
  "Rating_service": 4.5,
  "Rating_value": 4.0,
  "cuisine": "[Mediterranean', 'Middle Eastern', 'Israeli']",
  "Review_num": 255,
  "Neighborhood": "Downtown Manhattan (Downtown)",
  "Price_low": 2,
  "Price_high": 3,
  "Strong_tag": "[brunch', 'hummus', 'shakshuka', 'sabich', 'labne', 'stuffed cabbage', 'french toast', 'lamb burger', 'potatoes', 'israeli food', 'excellent israeli', 'small dishes', 'seated right away', 'fantastic food', 'new york', 'siniya', 'nyc']",
  "Comments": "Delicious Israeli Food \ud83d\ude0bFantastic FindGood food, bad serviceFantastic Food That's Reasonably Priced in Great AtmosphereDelicious Israeli brunch!Great \u201cfeelathome\u201d experience if you are middle eastern!Nice Israeli restaurantExcellent Israeli foodMiddle Eastern madnessHummus was the highlightSammy the Manager is GREAT.Awful service and music tooooooooudIsraeli/MediterraneanThis is a great for breakfastFantastic food, great service, slightly cramped seatingVery cozy and authentic place to try Israeli food in the West Village!\nFriends recommended this place for a while, so decided to try it out and it did not disappoint \ud83d\ude4c\n\nThe restaurant is often packed, so better to make a reservation, especially for...brunch.\nKeep in mind that tables are really close to each other :) but I guess that\u2019s quite common for Manhattan"
```

Fig. 3. Json cache of restaurant crawl result

### 3. Data structure

The data structure I created is a graph by using the Python dictionary. Each node in the graph represents a restaurant, and the link between restaurants are based on the number of "tags" they share. For famous restaurants, there're many tags created by users (which is crawled in "Strong\_tag" field of my data). However, 1/3 of restaurants don't have these user-created tags. To solve this problem, I also crawled all the restaurant review text in my dataset, and use the "Strong tags" list to match the tags in the reviews. By using this method, I created plenty of tags for every restaurant in my dataset.

Then, for restaurants that share the same tags, they can be considered as "linked". The more tags they share, the stronger they link together. We can build a graph based on this relationship. If restaurant 0 and 1 share the same tags [pizza, burger, chips], then our graph dictionary structure can be {0: {1: [pizza, burger, chips]}} and {1: {0: [pizza, burger, chips]}}. By this method, we built a large graph, which is stored in graph\_dict.json.

The python file I used to construct this graph is *Step2\_Data\_Structuring.ipynb*. I

put many notes in the notebook for better understanding the steps.

The JSON file of my graph is stored in **graph\_dict.json**

The stand alone python file to read the json of my graph is **Step3\_Demo.py**.

The screenshots of data and data structures are shown below:

```
{
  "0": {
    "id": 0,
    "Name": "12 Chairs",
    "url": "https://www.tripadvisor.com/Restaurant_Review-g60763-d533199-Reviews-12_Chairs-New_York_City_New_York.html",
    "Rating": 4.5,
    "Rating_food": 4.5,
    "Rating_service": 4.5,
    "Rating_value": 4.0,
    "cuisine": ["Mediterranean", "Middle Eastern", "Israeli"],
    "Review_num": 255,
    "Neighborhood": "Downtown Manhattan (Downtown)",
    "Price_low": 2,
    "Price_high": 3,
    "img_url": "https://media-cdn.tripadvisor.com/media/photo-p/14/f5/ee/cb/photo1jpg.jpg",
    "tags": ["brunch", "hummus", "shakshuka", "sabich", "labne", "stuffed cabbage", "french toast", "lamb burger", "potatoes", "israeli food", "excellent israeli", "small dishes", "seated right away", "fantastic food", "siniya"]
  },
  "1": {
    "id": 1,
    "Name": "15 EAST @ Tocqueville",
    "url": "https://www.tripadvisor.com/Restaurant_Review-g60763-d457830-Reviews-15_EAST_Tocqueville-New_York_City_New_York.html",
    "Rating": 4.5,
    "Rating_food": 4.5,
    "Rating_service": 4.5,
    "Rating_value": 4.0,
    "cuisine": ["French", "Japanese", "Sushi"],
    "Review_num": 451,
    "Neighborhood": "Union Square",
    "Price_low": 4,
    "Price_high": 4,
    "img_url": "https://media-cdn.tripadvisor.com/media/photo-s/16/ee/53/29/15-east-restaurant.jpg",
    "tags": ["dessert", "rice", "prix fixe", "wine list", "lobster", "ues", "tartare", "sushi", "steak", "pie"]
  },
  "2": {
    "id": 2,
    "Name": "230 Fifth",
    "url": "https://www.tripadvisor.com/Restaurant_Review-g60763-d1460354-Reviews-230_Fifth-New_York_City_New_York.html",
    "Rating": 4.0,
    "Rating_food": 3.5,
    "Rating_service": 3.5,
    "Rating_value": 3.5,
    "cuisine": ["American", "Bar", "Fusion"],
    "Review_num": 4019,
    "Neighborhood": "Tenderloin",
    "Price_low": 2,
    "Price_high": 3,
    "img_url": "https://media-cdn.tripadvisor.com/media/photo-s/1c/86/c1/85/deck-seating.jpg",
    "tags": ["brunch", "the empire"]
  }
}
```

Fig. 4. Restaurant data after preprocessing

```
{
  "175": ["brunch", "french toast", "potatoes"],
  "208": ["brunch", "tuna", "hostess", "ues", "ham"],
  "321": ["brunch", "rice", "block", "hostess", "ues"],
  "490": ["brunch", "rice", "tender", "ues"],
  "492": ["brunch", "rice", "waiters", "ues"],
  "650": ["brunch", "rice", "tuna", "tender", "ues"],
  "715": ["brunch", "rice", "waiters", "ues"],
  "824": ["brunch", "rice", "tuna", "tender", "ues"],
  "917": ["brunch", "rice", "tuna", "waiters"],
  "57": ["brunch", "rice", "tuna", "hostess", "ues", "ham"],
  "140": ["brunch", "rice", "tender", "waiters", "ues"],
  "169": ["brunch", "the empire state building", "rice", "tuna", "ues"],
  "190": ["brunch", "rice", "tuna", "tender"],
  "425": ["brunch", "tender", "waiters", "ues"],
  "509": ["brunch", "rice", "tuna", "ues"],
  "536": ["brunch", "rice", "waiters", "ues"],
  "654": ["brunch", "rice", "tuna", "tender"],
  "698": ["brunch", "the empire state building", "tender", "waiters", "ues"],
  "748": ["brunch", "rice", "waiters", "ues"],
  "840": ["brunch", "tender", "ues", "ham"],
  "842": ["brunch", "rice", "tender", "waiters", "ues"],
  "860": ["brunch", "rice", "tuna", "ues"],
  "883": ["brunch", "rice", "tuna", "ues"],
  "890": ["brunch", "rice", "tender", "ues"],
  "783": ["the empire state building", "rooftop bar", "rice", "tender", "waiters", "ues"],
  "78": ["the empire state building", "rice", "hostess", "ues", "ham"],
  "876": ["the empire state building", "rooftop bar", "rice", "hostess", "ues", "ham"],
  "739": ["rooftop bar", "rice", "tender", "waiters"],
  "90": ["rice", "tender", "waiters", "ues", "ham"],
  "145": ["rice", "tender", "ues", "ham"],
  "255": ["rice", "tuna", "tender", "ues"],
  "295": ["rice", "tuna", "waiters", "ues", "ham"],
  "298": ["rice", "tender", "waiters", "ues"],
  "341": ["rice", "tuna", "ues", "mouth"],
  "398": ["rice", "hostess", "waiters", "ues"],
  "437": ["rice", "tender", "hostess", "ues"],
  "514": ["rice", "hostess", "waiters", "ues"],
  "517": ["rice", "block", "hostess", "waiters"],
  "565": ["rice", "hostess", "ues"]
}
```

Fig. 5. Screenshot of Graph Structure json file

#### 4. Data presentation

The project outcome will be very interactive to users. I will use Flask and Plotly to make an interactive website and demonstrate the recommendations. The user will get the following interact options:

1. In the home page (<http://localhost:5000/>), there're 100 restaurants that are randomly chosen, in order to make the user has the feeling of the dataset. Each restaurant has detailed information and a preview image. It also includes a link that directly point to the webpage it was crawled from.
2. In the "Neighbors of Restaurants" page (<http://localhost:5000/neighbor>), users can

input a restaurant id, and the system can automatically find top 100 similar restaurants to this restaurant. For each restaurant found, users can click the link to see the detailed relationship to that restaurant (by using the path function below).

3. In the Path Between Restaurants page (<http://localhost:5000/path/>), users can input 2 restaurant id, and the system will calculate the shortest path in between. Also, the webpage will show the detailed path to the user. For example, if there's a path from 1 to 2 to 3, then the system will tell why 1 and 2 are connected, and why 2 and 3 are connected. Users can use this function to discover the similarity between restaurants.
4. In the Personalized recommendation page (<http://localhost:5000/similar>), users can input a series of restaurants' id. Then the system will calculate a list of restaurants that have the shortest average distance to the given restaurant list. This function can be very useful in practice, because users usually have a list of preferred restaurants, and they want to find a new restaurant based on this list.
5. In the Statistics of Cuisine page (<http://localhost:5000/stat>), users can input a cuisine name like "Chinese", and the system will automatically calculate the statistics of all "Chinese" restaurants. The distributions are shown by Plotly. For example, users can see the neighborhood distribution, Rating distribution and Price distribution. They can also see the tag distribution showing that what tags are frequently occurring in Chinese restaurants. For example, the "dumplings" or "spring rolls".

## 5. Demo Video

<https://drive.google.com/file/d/1ZiLiZiuU8p55kWTaD3lbSFzMAPKqg71X/view>