Fantastic Food and Where to Find Them

0. Group Members

- Linghui Wu (linghuiwu)
- Mingtao Gao (mingtao)
- Shengwenxin Ni (nswxin)
- Yanwei Pan (panyw)

1. Overview

The project aims to meet the needs of consumers in Chicago who have special food restrictions, allergies, and religious beliefs to consider. Compared to traditional food search engines, we enable customers to look for food products not only based on the dietary labels ("dairy-free," "vegan," "kosher," etc.) but also according to ingredients contained and nutritional information, such as trans-fat per serving. Firstly, we collect nearly 17200 pieces of product information from three major grocery stores in Chicago - Whole Foods, Trader Joe's and Jewel Osco - through web-scraping, during which regular expression has been employed to grab keywords we need. After the cleaning process, data from different sources has been merged and utilized to establish SQL database including tables named "product" and "store". Then, we built the backend for the grocery search tool by connecting the SQLite database to the frontend, a user-friendly Django web interface. Through the interface, users can enter and select dietary requirements, and the product information, along with other useful information, including store locations, will be displayed.

2. Structure

2.1 Back End

The `BackEnd` folder is composed of three modules, the web-crawler, the data cleaner, and the database constructor.

- Web Crawler

In each of the `JOCrawler.py`, `TradeJoesCrawler.py` and `WholeFoodsCrawler.py`, we created a class and defined helper functions to collect and store product and store information from the three grocery stores mentioned above.

To facilitate the replication of web scraping, we provide `test.py` to run in the command line. You may specify the number of observations and the name of the store and it would generate sample csv files. Please see `Test_Intructions.md` in the `BackEnd\Crawler` folder for more details.

- Data Cleaner

Raw data gathered by crawler are stored in the relative `storename_prod.csv` and `storename_store.csv` files.

Using pandas, `DataCleaning.py` reads the csv files and processes the data by droping missing values and unifying the column names. Then, it merges the cleaned data from three sources into two dataframes representing all the products and store information, and saves them as `product.csv` and `store.csv`. Those files will be later used to construct tables in the database.

- Database Constructor

`database.sql` includes SQL codes to construct the `product` and `store` tables, to import data from csv files into the according tables, and to set the NULL values in the table.

By entering the command `.read database.sql` in the shell for SQLite in terminal, the user can construct the database for the project `foodsearch.sqlite3`.

2.2 Front End

The front end is mainly composed of two parts, the grocery search tool `search_items.py` and the Django web interface, which we referenced the codes from the second assignment.

`search_items.py` takes in `args_to_ui`, a dictionary generated by the information entered by users on the interface, and then it produces query statements to retrive results from the `foodsearch.sqlite3` database. To display information neatly, we only selected the product name, the store name, the store address, and additional information related to the dietary restrition that are selected or entered by the user. For example, if the user chooses less than 10g sugars contained in one serving, the interface will display product and store information as well as the amount of sugars contained per serving.

To run the Django web interface for the grocery search tool, you can enter `python3 manage.py runserver` in the command line inside the `FrontEnd` folder. Once the interface is started, you can access the search engine by pointing a browser to `http://127.0.0.1:8000/`.

3. Accomplishment

Our project aims to develop a grocery search engine catering to individuals' dietary restrictions. We have achieved the primary goal of this project. We successfully built a database containing product and store information and a user-interactive interface that implement keywords queries based on food ingredients, nutrition contents, dietary labels, and store information.

However, here are some aspects we can further improve:

- Provide more detailed dietary labels

Due to the broad definition of dietary labels and different documentations on each website, we only took "organic", "vegan", "dairy-free", and “kosher" into considerations. However, we may identify more dietary labels through content analysis on the ingredients.

- Offer more precise store locations

For now, our project can provide nearby stores by querying users’ zipcode and find stores that share first four-digit zipcode. We hope to convert store addresses to longitude and latitude and utilize Google Map API to calculate precise distances between the target store and the user.

- Design a more attractive and user-friendly interface

Due to limited practice on web development, we implemented the UI primarily based on previous assignment. With more in-depth learning, we hope to display thorough details of each product neatly.