CS 410 Text Information System - Final Project Documentation

Team Name: Pacific Search
Project Name: Restaurant Finder
Team Members:
Daniel Felker (Captain) dfelker2@illinois.edu
Sanjay Babbar babbar2@illinois.edu
Meng-Yun Tsay mtsay2@illinois.edu

1) An overview of the function of the code (i.e., what it does and what it can be used for).

Restaurant Finder is a search engine that can return relevant restaurant results near the user's location based on the user's search term.

For example, if a user enters "italian" in the chrome extension, our app will use the query term, "italian" and longitude and latitude of user's location to search restaurant review texts in our database* and return the top 50 relevant restaurant results near the user's location on the chrome extension.

*Notice that we use Yelp's open dataset (https://www.yelp.com/dataset) as our database, which is a small subset of all Yelp's business listings. Thus, the restaurant's location coverage is only partial cities across North America. Due to database limitation, the nearest restaurants shown in the search results may be far away from the user's location.

2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.

Restaurant Finder Application Implementation:

We use the MeTA toolkit (https://meta-toolkit.org/search-tutorial.html) and review texts in our database to find the ranking list of relevant restaurants. In order to use the MeTA toolkit, it is necessary to build review texts corpus in line.toml configuration file, a dat file, where each restaurant's reviews appear in one line, and also build a config.toml file to specify our prefix, dataset, corpus, index, analyzers and analyzers.filters.

[1] MeTA Toolkit Documentation https://meta-toolkit.org/overview-tutorial.html

The dataset.py file in the restaurant_finder folder helps us build the review text corpus in line.toml configuration file and a dat file. First, we take Yelp's academic dataset's directory path as an input and creates a list of businesses in both restaurants and pop-up restaurants categories, and also a dictionary of restaurant_idx, where each key is each restaurant's business id and value is assigned index, from given yelp_academic_dataset_business.json, and writes both the results to JSON files respectively.

Second, we use the dictionary of restaurant_idx from the previous step and review texts from given yelp_academic_dataset_review.json to create a dictionary of reviews, where each key is each restaurant's business id and value is a list of that restaurant's first 100 reviews if available with 5,000 characters limitation for each review. Then, we create a list of review_txt_biz_ids by combining all the business ids in all keys from the dictionary of reviews, and write the list of review_txt_biz_ids into a text file. Finally, we create a list of review_txts by combining all the review texts in all values from the dictionary of reviews, and then use the list of review_txts to create a review.dat file and a line.toml file, where each restaurant's first 100 reviews appear in one line, under a folder named "review".

Note that there is a function called, expand_review_with_categories(restaurant, review_txt), which will add review's corresponding restaurant business category to review text. This feature can be turned on and off in switches.py. The default setting is on and this feature will be carried out when combining all the review texts as listed above.

Third, we initialize the Finder class from finder.py, which uses some functions in the MeTa toolkit in finder.py to find the ranking list of relevant restaurant results. In order to initialize the Finder class, it is required to pass in config.toml and the number of returned results as parameters. We build config.toml file by specifying "review" folder as our dataset, "line.toml" as our corpus, "idx" as our index and "ngram-word" (unigram), "icu-tokenizer" filter, "lowercase" filter, "list" (stopword.txt) filter and "porter2-filter" filter as our filter chains.

Before review texts are indexed, MeTA processes the review texts with a system of tokenizers, filters and analyzers listed in config.toml file. In our case, the "icu-tokenizer" will first convert documents into streams of tokens by following the unicode standards for sentence and word segmentation. Then, "list" filter will reject tokens that are stop words listed in stopwords.txt and "porter2-fiter" filter will transform each token according to Porter 2 English Stemmer rules and "lowercase" filter will change each token to lowercase. Lastly, "ngram-word" analyzer will use the output from the above filter chain

[1] MeTA Toolkit Documentation https://meta-toolkit.org/overview-tutorial.html

and count the sequence of n-tokens (In our case, n=1)[1]. If the inverted index file already exists, we clean the existing inverted indexes by deleting the file.

Further, we call metapy.index.make_inverted_index(config.toml) to generate inverted indexes and call find_restaurant(query_str) from Finder.py to search the query term given by the user as query_str. In the find_restaurant(query_str), we create a query object by calling metapy.index.Document() and also create a ranker object using okapi_bm25 by calling metapy.index.OkapiBM25(). Then, pass self.idx, the query object we just created and number of returned results as parameters in ranker.score(self.idx, query, num_results) to get the ranked results. Finally, we use the review inverted index in the ranked results to find the corresponding review and its business id. Then, use the business id to find the restaurants as final relevant restaurant results.

When the user searches through Chrome extension and clicks go, Chrome extension will return the longitude and latitude of the user's location along with the user's search query to our application. Then, find_restaurant(query_str) from Finder.py will sort the relevant restaurant results generated from the previous step by the nearest location near the user, and return the top 50 restaurant results.

Restaurant Finder Evaluation Implementation:

We use Yelp search as the correct relevant results to evaluate our results generated by our application.

Since we use Yelp’s academic dataset as our database and it is only a small subset of all Yelp’s business listings, our application’s results are not comparable with yelp.com’s search results. Even if we utilize Yelp's api endpoint with necessary parameters, such as city, business category and search term, the returned business results only contain at most 3 reviews per business, which are not sufficient to provide enough review texts for our ranking system.

Eventually, we get a list of restaurant business ids from Yelp's api endpoint with City parameter, take Philadelphia for example, and Business Category parameter, which is “restaurants”. Then, we get a list of overlapping restaurants and a dictionary of overlapping restaurant_idx, where each key is each restaurant's business id and value is assigned index, between yelp academic dataset and the list of restaurant business ids from Yelp's api endpoint in Philadelphia. And use the list of overlapping restaurants’ business id and review texts from given yelp_academic_dataset_review.json to create a dictionary of reviews, where each key is each restaurant's business id and value is a list

[1] MeTA Toolkit Documentation https://meta-toolkit.org/overview-tutorial.html

of that restaurant's first 100 reviews if available with 5,000 characters limitation for each review. Then, we write the list of restaurants and the dictionary of restaurant_idx into new JSON files respectively. Finally, we create a list of review_txts by combining all the review texts in all values from the dictionary of reviews, and then use the list of review_txts to create a review.dat file and a line.toml file under a folder named “review” in order to use MeTa toolkit adopted in Finder.py to find the relevant restaurant results.

To evaluate, we get a list of 50 restaurant business ids from Yelp’s api endpoint again with the same city parameter, Philadelphia, and the same Business Category parameter, which is “restaurants”, and search term, which is the same search term that we will be using to search in our application, such as “italian”. This list of restaurants business ids is our correct relevant results that will be used to compare with results generated by our application. Then, we also get a list of top 50 relevant restaurants with the same search term, “italian”, to search in our application. We compare both results from Yelp api endpoint and our application to find the number of overlapping restaurants to calculate the precision.

We pick 3 cities, including Philadelphia, Tampa, Indianapolis, and 5 search terms, including “tacos”, “italian”, “pasta”, “steak”, “thai”. For each city, we iteratively run through the above steps with each search term and calculate the mean precision for each city. The precision for each city is, 0.35 for Philadelphia, 0.48 for Tampa, 0.37 for Indianapolis.

3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable.

It is required to install Chrome extension, download yelp’s open dataset and other setup steps to run the app. The more detailed documentation of how to install and run the app are listed in the README.md files in the Restaurant Finder repository. One is for the main application and the other is for the evaluation.

The README.md file links can be found here: main application, evaluation

4) Brief description of contribution of each team member in case of a multi-person team.

Daniel Felker: Chrome extension implementation, presentation

[1] MeTA Toolkit Documentation https://meta-toolkit.org/overview-tutorial.html

Sanjay Babbar: Main restaurant searcher/ranker system algorithm for the app
Meng-Yun Tsay: Evaluation, documentation

[1] MeTA Toolkit Documentation https://meta-toolkit.org/overview-tutorial.html