Progress report

CS410 UIUC
Team: Hungry Panda

Intro

We divided the tasks into three parts: architecture, frontend and backend. After initializing the architecture of the project, we start developing in parallel. Each team member selects the part that he or her is interested in to work on. We meet on a weekly basis to discuss the project progress.

Architecture

@Tingyu Zeng

- **Completed** - The frameworks for frontend and backend were selected: frontend will use HTML, CSS, JavaScript and Chrome extension APIs while backend will be built with a Flask server (Python). The skeleton of two applications was initialized with basic dependencies configured. The manifest.json for Chrome extension was configured with basic features and set up as a good starting point for future development. The Flask server was set up with minimal dependencies and NLTK was installed and tested. An endpoint was created to test and demo the interaction between the frontend and backend. A simple README was written to explain how to install the dependencies and how to start developing.
- **Pending** - Further develop the endpoint so that it will receive and process the data from the request, instead of using mocked data; Configure the environment variables for the applications instead of using hard coded value; Provide error handling.
- **Challenges** - For simplicity this project of 2 apps is maintained in one single repo with 2 different programming languages. Setting them up and making them work together is challenging. There are some CORS issues to be checked and resolved. In the future, if the server needs to be deployed, an additional step in the pipeline might be needed to generate the distributed files according to the hosting platform.

Extension: data crawling

@Sicheng Meng

Completed and pending:
After several meetings, we have been discussed and answered some most important questions such as what data/metadata we need, where and how to fetch the data, what format the data should be, etc. We have made the decision to crawl the single restaurant page when the user run our extension on a certain restaurant's Yelp page, instead of mass scraping Yelp data in advanced. Although we're not sure if we'll actually use it in our analysis, besides user reviews, we are also interested in the metadata of the review as well as the user who wrote the reviews. This includes review post time, location, review votes, and rating

distribution of a user, etc. The next step is coding - I haven't started writing the code, but with the most important questions answered, rest of the work is just a matter of time.

Challenges I've met:

- Yelp does not allow any scrapping of the site, but its public xxx API only provides the top three reviews of each merchant, but we need every single review for a restaurant. Thus, we'll have to find a way to crawl the review data for a given restaurant page without getting blocked by Yelp.
- I don't have any experience in web crawling, Json or JavaScript, so I'll have to learn everything from scratch.

At the moment, I'm still looking for the best way to crawl the data. I have three more weeks to work on this task without new class materials or programming assignments, so I'll have plenty of time. Some potential solutions include…

Extension: data presentation

@Fangsheng Yang

- **Completed** - Created Manifest.json file to import extension information, version, actions, and permissions. Loaded Built a demo Chrome extension to display test strings.
- **Pending** - Still working on the scripts to properly display the final output. Design the User Interface Elements. Working on the interaction logic of Chrome extension before and after the backend.
- **Challenge**: Since this is the final step of the project, I need to use some 'fake' data to start working. In the future I may need to spend some time with backend teammates to make some modifications.

Backend: info retrieval and sentiment analysis

@Yiwei Kuang + @Jiachun Tang

- **Completed** - Considering that our data crawling part has not been completed, which means we did not get real data, so we first used yelp's public database to process some demonstration data. First we preprocessed the data: Since we are only interested in restaurants, we filter out all other types of businesses in the business dataset. Also, we matched the corresponding IDs in the business and review datasets to remove all reviews that are not related to the restaurant. Finally, we have the cleaned data. We then do the following on the "comments" data: 1. Split tokens on spaces. 2. Remove all punctuation from words. 3. Remove all words that do not consist entirely of alphabetic characters. 4. Remove all words with known stop words. 5. Remove all words with length <= 1 character.

- **Pending** - We will be mocking some dataset as our input and take that to implement our ranking system algorithm.The preparation of mocking data could be very various, since sometime we will be getting "simple" data, but sometime it is not the case. Therefore, we

will be formatting the data as a readable and processable format type. For subsequent use of the bag-of-words model, we will define a vocabulary, which includes most of the food vocabulary.

- **Challenge**: It tends to be more challenging when it comes to how we want to design the ranking system within our food recommendation system. Furthermore, the weighted factor is another important keypoint that challenges us here. From the data side-wise, we also will be giving different scores and weighted factors based on the context, whether it is more favorable or unfavorable. We will be creating a Food Glossary based upon Yelp website and out of it, the popular food words in the world and merging it as a word system. The composition of food names is usually very diverse. Therefore, when preprocessing the text, the food name may be segmented incorrectly, resulting in the wrong recognition of this comment. We are thinking about relevant coping strategies.