NLP Applications on Yelp Datasets

Chundi Guo Zhu Liang Xin Lu

April 30, 2023

Contents

1 Rating Prediction and Review Summary

1.1 Project Objectives

This sub-project mainly focus on using customer reviews and star ratings to **predict the review rating** and **identify key features contributing to the restaurant rating** based on Yelp review data using Machine Learning and Natural Language Processing (NLP) techniques.

1. Rating prediction:
   - Collect and clean the Yelp data related to restaurants, which should include the business name, reviews, and star ratings, etc.
   - Randomly choose 10% of the cleaned data, extract features of selected data, and perform exploratory data analysis.
   - Train the machine learning model using 9 kinds of algorithms to predict the star rating accordingly and compare the accuracy and time performance.
2. Review summary:
   - Perform sentimental analysis to label features into positive or negative.
   - Build an toy level user friendly interface to generate restaurant summary by allowing for customers or restaurant owners to select category and location.

The sub-project contributors are Chundi Guo, Xin lu.

1.2 Techniques and Tools

This project involves text processing, data exploratory analysis, model training and prediction, output visualization and interface development, all of which involve different techniques and tools. We will explain each stage's technology and tools in detail. The project comprises two types pf files, which are suffixed by `.ipynb` or `.py`. The `.ipynb` files are the primary files which document the whole working process including data cleaning, data exploration, model training and prediction, and summary generator interface. The `.py` files contain all of the functions we write for the above processes.

1.2.1 Text preprocessing

The Yelp Open Dataset was obtained from the official website. Specifically, in this sub-project we use `yelp_academic_dataset_business.json` and `yelp_academic_dataset_review.json`.

Functions used in this section are defined in `preprocess_text.py`. `json_to_csv` is used to read the two json files mentioned above and link them together by business id after choosing variables we need (such as stars rating, reviews). In the `preprocess_text` function, text preprocessing techniques are incorporated to clean and transform the raw text data into a format that can be analyzed. These techniques help to reduce noise in the text data and make it easier to extract meaningful insights. To speed it up, we use `applyparallel` method from `multiprocesspandas` to take advantage of the power of multithreads.

- Techniques:

  Tokenization: Breaking down text into words or tokens.

  Stopword removal: Removing commonly used words that do not add much meaning to the text.

  Lemmatization: Converting words to their base form or lemma.

- Tools used:

  `nltk`: A popular natural language processing library in Python.

  `multiprocesspandas`: A library provides capability to parallelize `apply()` methods on DataFrames.

1.2.2 Feature extraction

The technique used during feature extraction is called the bag of words. It is used to represent the text data in a numerical format that can be used for machine learning. Each document is represented as a vector of word frequencies, where the frequency of each word is counted and stored in a feature vector. Function `get_vocab_mtx` defined in `prediction_summary.py` allow us choose the right feature extraction method and finish the vectorization process and generate the feature matrix.

- Techniques:

  Bag of Words: A representation of text that counts the frequency of each word in a document.

- Tools used:

  `CountVectorizer` and `TfidfVectorizer`: Tools from the `Scikit-learn` library that transforms text into a bag of words.

1.2.3 Machine learning

Once we get feature extraction result, we use 9 kinds of algorithm (check Table 1) to train the model and to the prediction accordingly. All the methods (2)-(10) are listed below, while technique (1) is used to get top 10 features that lead low star ratings. Function `report_prediction` help record the time used and get the accuracy in each method.

- Tools used:

  `Scikit-learn`: A popular machine learning library in Python. It provides the tool for technique (2)-(9).

  `XGBoost`: An optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable.

1.2.4 Interface Development

Once we explore the power of the above machine learning methods, we decide to apply them into sentimental analysis and build a simple interface to generate summary report for any restaurant they are looking for. We develop functions to allow for searching nearby zipcodes by range, selecting restaurant by category and generate visualizations in `prediction_summmary.py`. The visualization option includes wordcloud of positive or negative phrases and bar plot pf the top positive or negative phrases. We consider exceptions like input error and no available restaurants in that range and the program will generate appropriate prompt if corresponding case happens.

- Tools used:

  `linearSVC`: used to perform sentimental analysis by classifying features into good or bad mode.

  `matplotlib`: used to generate the visual as a bar plot.

  `wordcloud`: used to generate the visual based on the frequency of words.

  `ipywidgets`: used to create interactive inputs and arrange the interface layout.

  `IPython.display`: used to display the output.

1.3 Results and Conclusion

The results of this project are divided into 4 parts. The first part involves wordclouds based on star rating, it gives an intuitive graph that shows relationship between words (like never come, great food, etc) and star ratings. The second part involves machine learning and prediction based on each individual review (Reviews Base). The third part involves machine learning and prediction based on all reviews of each restaurant (Restaurants Base). The forth part involves the illustration of interface input and output.

According to the data exploratory analysis from the first part, the reviews were further analyzed by grouping the star ratings into 3 levels (1 and 2 being negative, 3 being neutral, and 4 and 5 being positive) and 2 levels (1, 2, and 3 being negative, and 4 and 5 being positive) for each part. Given the large size of the database, some methods require a significant amount of time to run. To obtain timely results and compare different machine learning methods, we randomly selected 10% of restaurant data as a sample for analysis in the part that is based on individual reviews.

1.3.1 Data Exploration

Since the star rating has 5 levels, we want to categorize the ratings. Therefore, after obtaining the cleared data, we use the word cloud method to check what words are in the comments of different star ratings, Figure 1a to Figure 1e show result for the word cloud. It seems that 1 star and 2 star are quite similar, 4 star and 5 star are quite similar. 3 star contains words from 2 and 4, but with less frequency. Therefore, we will categorize star rating in 2 ways mentioned above and do prediction based on these 2 ways of category.

1.3.2 Reviews Level Prediction

After we do prediction based on review level. Table 2 shows the accuracy of the prediction when star rating is divided into 3 levels. Table 3 displays the accuracy score of the prediction when star rating is divided into 2 levels. Figure 2 reflects the running time of various methods.

From Figure 2 and Figure 3, we can observe that:

1. Running time for 3-level classification is much longer than that for 2-level classification.

2. SVM and MP methods are the most time-consuming.

3. Except for KNN method, other methods spend more time on the classification task than on the prediction task.

From Table 2 and Table 3, it can be seen that the 3-level methods do not have high accuracy in predicting neutral reviews, with the highest score being only 0.67 for the RF method. This may be due to the insufficient amount of neutral data under this classification method, and the fact that reviews with a star rating of 3 are often very close in data to those with a rating of 4 or 2.

1.3.3 Restaurant Level Prediction

We further process the data, summarize the reviews under the same restaurant, and based on the summarized reviews, use the same 8 methods to analyze and predict after feature extraction in the unit of restaurant. Figure 4 and Figure 5 show running time under this restaurant base. Time is much less than that in review base. Because sample size reduce a lot once we summarize reviews in unit of restaurant.

From Table 4 and Table 5, we could get similar conclusion, in general, 2 levels has better accuracy scores than 3 levels.

1.3.4 Feature Summary

After obtaining the prediction results, we are curious about which features and words in the reviews would affect the star rating. We first used the method of Recursive Feature Elimination (REF) and trained the model using the MNB method. Then we obtained the top 10 features that lead to low rating, which are: undiscerning, undoable, undoubtably, unemotional, unexplainably, unfinishable, unflattering, ungraciously, ungratefulness, unhealthiness. All of these refer to bad attitude. To learn more, after vectorizing the text, we performed sentiment analysis using linear SVC to classify the word groups. Then, we generated a word frequency chart using wordcloud or a bar plot using matplotlib. Finally, we used `ipywidgets` to create an interface. The result shows in Figure 6.

Until present, we have features that affect star rate. We are also curious about how these features linked to words in reviews. After feature extraction and running regression, we get some coefficients. The positive coefficients are divided into one group, and the negative ones are divided into one group. According to this

result, wordcloud is used to count the word frequency, and we get Figure 7a and Figure 7b, which shows positive words and negative words accordingly, the output of which can be viewed as a naive version of restaurant level review summary.

1.3.5 Summary Geneartor

Restaurants using Yelp hope to understand what types of reviews will affect their rating, while consumers primarily use Yelp to find nearby restaurants they want to eat at. Based on this insight, inspired by Zhu's project idea, we built an interface where consumers can search for restaurants within a targeted range by entering their address (zipcode), type of cuisine, preferred distance, and whether they want to see good or bad reviews. Figure 8 shows the input items and output (if there is no match) at the bottom (we allow for several other possible exceptions, each of them will generate unique prompt). Figure9 shows the output if there is matching result.

1.4 Figures and Tables

Table 1: Machine learning algorithms used

Algorithms	Description	Advantage part
Support Vector Machine (SVM)	Separate data points using a hyperplane that maximizes the margin between two classes	High-dimensional data
Multinomial Naive Bayes (MNB)	Based on Bayes' theorem, which assumes that the probability of a feature given a class is independent of the presence of other features	Text data
Decision Tree (DT)	Creates a tree-like model of decisions and their possible consequences	Small datasets
K Nearest Neighbor (KNN)	Classifies data points based on the majority class among its k-nearest neighbors	Small datasets
Random Forest (RMFR)	Creates multiple decision trees and combines their predictions	High-dimensional data
Gradient Boosting (GB)	Creates multiple weak models in a sequential manner, where each new model tries to improve the performance of the previous model	High-dimensional data but computationally expensive
Logistic Regression (LR)	Assumes that the log-odds of the outcome are linearly related to the input features, predicts the probability of a binary or multiclass outcome based on one or more input features	
Multilayer Perception Classifier (MLP)	A type of neural network that consists of multiple layers of nodes, each of which performs a non-linear transformation of its inputs.	Learn complex non-linear relationships between inputs and outputs
XGboost	A gradient boosting algorithm that uses a more efficient tree-building process and better regularization techniques	High-dimensional data

(a) most frequent words for 1 star

(b) most frequent words for 2 star

(c) most frequent words for 3 star

(d) most frequent words for 4 star

(e) most frequent words for 5 star

Figure 1: Most frequency words for each star rating

Table 2: Accuracy of 3-level star rating prediction (10% reviews base)

	Positive	Neutral	Negative	Weight Avg
MNB	0.92	0.41	0.73	0.83
DT	0.86	0.25	0.57	0.74
KNN	0.76	0.37	0.81	0.72
RF	0.81	0.67	0.83	0.8
GB	0.84	0.51	0.8	0.79
XGboost	0.87	0.51	0.79	0.81
LR	0.9	0.48	0.78	0.83
SVM	0.88	0.56	0.79	0.83
MP	0.91	0.39	0.74	0.82

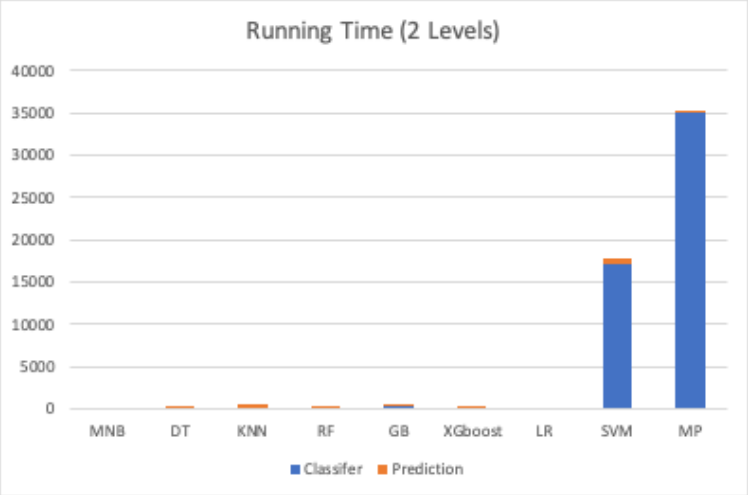


Figure 2: Runing time for 2 level star rating (10% reviews base)

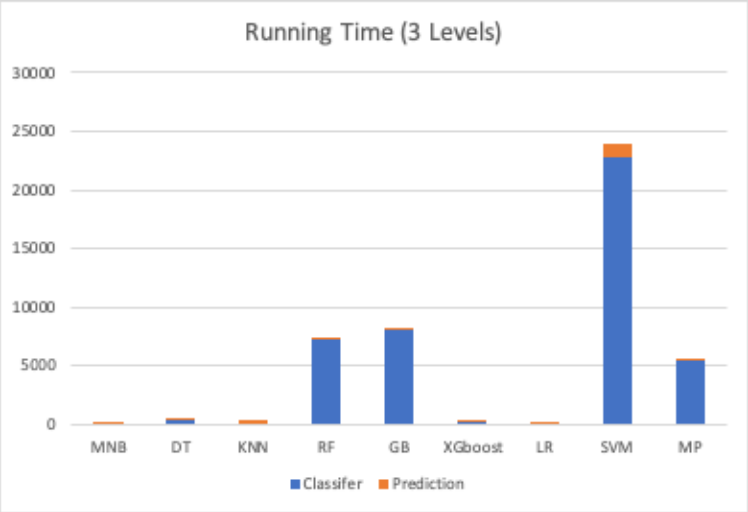


Figure 3: Runing time for 3 level star rating (10% reviews base)

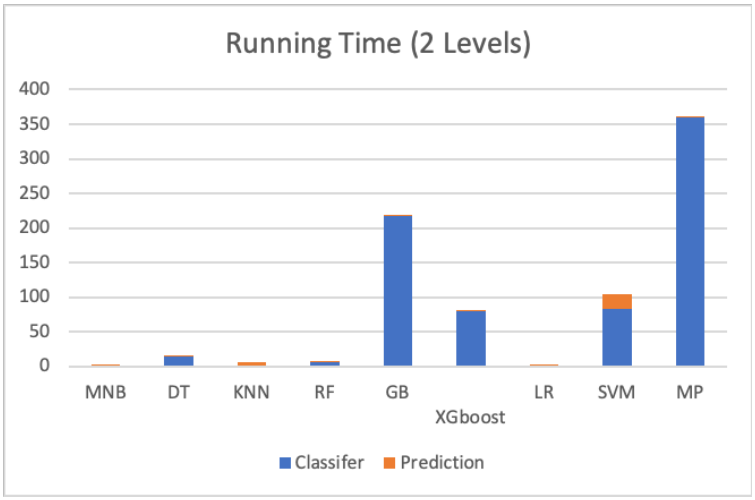


Figure 4: Runing time for 2 level star rating (Restaurant base)

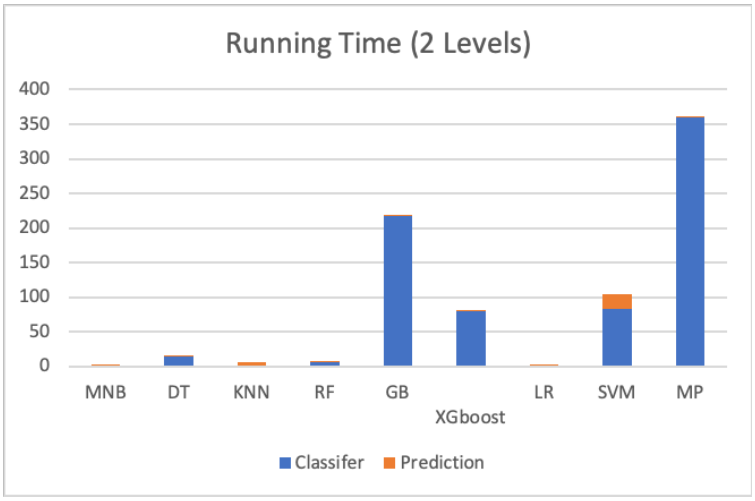


Figure 5: Runing time for 3 level star rating (Restaurant base)

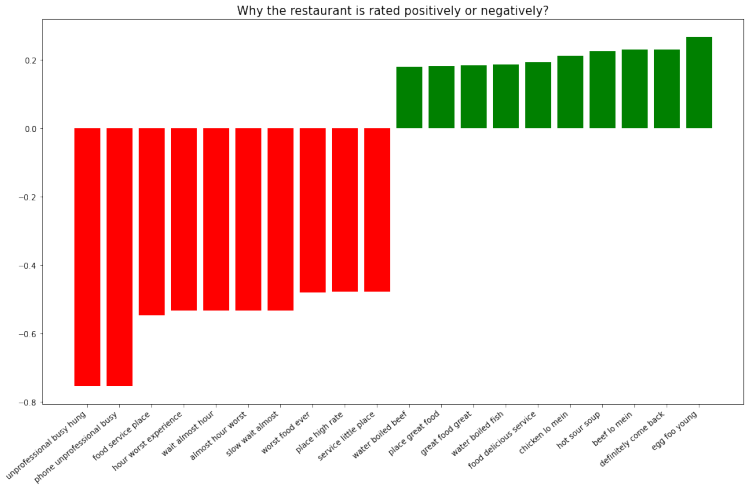


Figure 6: Main features

(a) positive words frequency

(b) negative words frequency

Figure 7: Words frequency

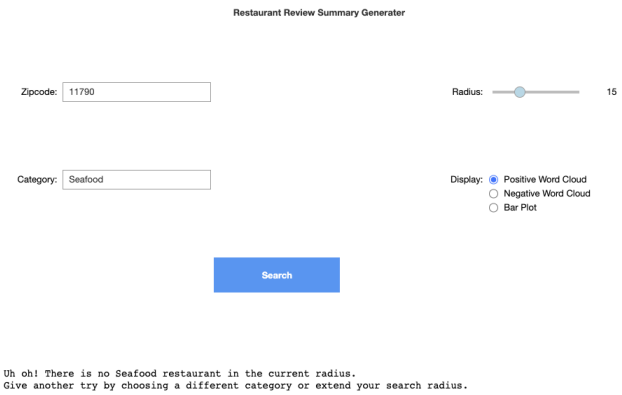


Figure 8: Interface input

Figure 9: Interface output

Table 3: Accuracy of 2-level star rating prediction (10% reviews base)

	Positive	Negative	Weight Avg
MNB	0.96	0.71	0.92
DT	0.92	0.59	0.87
KNN	0.86	0.83	0.86
RF	0.9	0.91	0.9
GB	0.91	0.85	0.9
XGboost	0.93	0.83	0.91
LR	0.94	0.81	0.92
SVM	0.94	0.84	0.92
MP	0.95	0.77	0.92

Table 4: Accuracy of 3-level star rating prediction (restaurant base)

	Positive	Neutral	Negative	Weight Avg
MNB	0	0.67	0	0.45
DT	0.7	0.84	0.5	0.79
KNN	0.65	0.71	0.82	0.7
RF	0.91	0.75	1	0.81
GB	0.92	0.86	0.82	0.87
XGboost	0.9	0.87	0.85	0.88
LR	0.89	0.77	1	0.82
SVM	0.86	0.8	1	0.83
MP	0.79	0.83	0.77	0.82

Table 5: Accuracy of 2-level star rating prediction (restaurant base)

	Positive	Negative	Weight Avg
MNB	0.72	0.88	0.85
DT	0.88	0.72	0.83
KNN	0.84	0.66	0.79
RF	0.83	0.98	0.87
GB	0.91	0.93	0.92
XGboost	0.93	0.92	0.93
LR	0.84	0.99	0.88
SVM	0.88	0.97	0.91
MP	0.96	0.92	0.92

Files related to this sub-project:

```
sub-project_1/
|-- rating_prediction.ipynb
|-- feature_summary.ipynb
|-- preprocess_text.py
|-- ngrams.py
|-- prediction_summary.py
|-- dataset/
|   |-- all_review_1.csv
|   |-- sample_review.csv
|-- img/
|   |-- single_star.png
|   |-- thumbs_up.png
|   |-- thumbs_down.png
|-- yelp_dataset/
|   |-- yelp_acadamic_dataset_business.json
|   |-- yelp academic dataset review.json
```

Instructions for Running the Software

The source data used in this project is contained in two files within the `yelp_dataset` folder: `yelp_academic_dataset_business.json` and `yelp_academic_dataset_review.json`.
To process the source data, run the `rating_prediction.ipynb` notebook, which will generate two CSV files: `all_review_1.csv` and `sample_review.csv`.

2 Restaurant Recommendater: An Interface by Python

2.1 Project Objectives

This sub-project aims to **develop an interactive interface that recommends restaurants to users** based on user-generated tip text.

The specific objectives are as follows:

1. Data processing, sentiment analysis, and database construction. Utilize Yelp's database, which includes information about the restaurants (business dataset) and user-generated tip text (tip dataset), to create a comprehensive dataset for the recommendation system. Implement a sentiment analysis model to identify positive tips that highlight specific dishes or qualities of the restaurant. This allows the recommendation system to suggest restaurants that have received praise for specific dishes, further personalizing the recommendations.

Related File: `tip_data.ipynb`

2. Develop a reusable Python class. Design the recommendation system as a reusable and modular Python class that can be easily incorporated into other projects or applications. This will allow other developers to leverage the recommendation system in their own projects and potentially contribute to its further development and improvement.

Related File: `RestaurantRecommender.py`

3. Build an interactive user interface. Implement the recommendation system in a Jupyter Notebook to create an interactive user interface that makes it easy for users to input their preferences and receive recommendations. Additionally, develop a web-based interface using `Flask` and HTML to make the recommendation system more accessible to a wider audience.

Related Files: Jupyter Notebook: `tip_recommender.ipynb`,
Web page: `app.py`, `templates/index.html`, `templates/draw_page.html`,

This sub-project's contributor: Zhu Liang

2.2 Techniques and Tools

In this section, I will elucidate the techniques and tools employed to accomplish this project, detailing the process in chronological order.

2.2.1 Database construction

The Yelp Open Dataset was obtained from the official website:`https://www.yelp.com/dataset`. Specifically, the `yelp_academic_dataset_business.json` and `yelp_academic_dataset_tip.json` datasets were used in this sub-project. Tips are brief pieces of information left by users, similar to reviews but more concise, intended to provide the most crucial information about a restaurant to other customers.

Using `json` and `pandas`, the data was read, variables were selected, and restaurant businesses were filtered out. Next, I use library `string` to filter non-English tip texts, followed by the utilization of libraries such as `NLTK` and `re` for text preprocessing. This preprocessing involved converting to lowercase, removing punctuation and digits, tokenizing words, removing stopwords, and lemmatizing words back to their standard form.

After obtaining the processed tip text, `TextBlob` was employed for sentiment analysis, and only tips with a positive sentiment were retained. At the end of this stage, two data files were generated: `tip_business.csv`, containing basic restaurant information, including restaurant name, address, zip code, star-rating, and review count; and `tip_text.csv`, which includes both the original text and the processed text. These two datasets are linked through the *business_id* variable.

2.2.2 Recommendation algorithm

To create the restaurant recommendation algorithm, I write a custom class named `RestaurantRecommender`. This class handles the data, processes it, and offers two main methods: `recommender()` and `draw()`.

The class constructor (`__init__`) initializes the class with the restaurant business data and tip data, a set of predefined keywords related to different types of cuisines, and an empty custom pool (used in the `draw()` method).

The `get_nearby_zipcodes()` method finds nearby zip codes within a specified radius (mile range) using the `uszipcode` library. It takes a zip code and radius as arguments and returns a list of nearby zip codes.

The `filter_by_zipcode()` method filters the businesses based on the given zip code and radius. It calls the `get_nearby_zipcodes()` method to get a list of nearby zip codes, and then it filters the restaurant data based on those zip codes.

The `filter_by_keyword()` method filters the tips and businesses based on a given keyword. It first finds tips containing the keyword, and then it filters the businesses based on the presence of these tips.

The `compute_custom_score()` method computes a custom score for a given restaurant. This score takes into account the restaurant's review count, star rating, and the number of tips containing the specified keyword. This score is calculated by the following:

$$score = \log(reviews\ count + 1) \times stars \times tips\ count$$

In the equation, $\log(reviews\ count + 1)$ is used to reduce the impact of the number of reviews on the overall score, and the addition of 1 ensures that the value remains positive. As a result, the number of times a particular dish is mentioned in the tips will significantly influence the recommendation order. Since tips are likely to contain the most important information that customers want to share, a dish that is mentioned frequently in tips is considered highly recommendable.

The `generate_google_maps_url()` method generates a Google Maps URL for a given restaurant's name and address using the `urllib` library.

The `recommender()` method is the core of the recommendation algorithm. It takes a keyword, zip code, radius, and an optional number of recommendations to return. The method filters the restaurants based on the given zip code and radius, filters the restaurants by the given keyword, computes the custom score for each restaurant, and returns the top N (default value is 3) restaurants based on this score.

The `draw()` method offers a random restaurant recommendation. It takes a zip code, radius, and an optional list of keywords as arguments. The method randomly chooses a keyword and calls the `recommender()` method to get a list of recommendations. It then returns a random recommendation from the list, or `None` if no recommendations were found.

2.2.3 Interface implementation

This section reports the overall implementation of the user interface. Initially, the interactive interface was developed in Jupyter Notebook. Subsequently, the `RestaurantRecommender` was saved as a .py file, and an equivalent interface with the same functionality was implemented on an HTML web page.

I have taken into consideration cases of invalid inputs and situations where no restaurants are found, and provided appropriate error messages and prompts in the interactive interface.

Jupyter Notebook implementation

The following techniques and tools were used in this implementation:

`IPython.display` was used to display the output, clear previous outputs, and render HTML content.

`ipywidgets` was used to create interactive input elements and organize the interface layout.

`textwrap` was used to wrap long text in the tips section, ensuring that the tips were displayed in a readable format.

To build the interface, I constructed two VBox widgets: `search_box` and `draw_box`, which are linked together within a single interface using buttons. By linking these two VBox widgets within the same interface, the user experience is streamlined and more intuitive.

Web page implementation

To implement the interface on a web page, I utilized `Flask`, a lightweight web framework for Python. Flask enables the creation of dynamic web applications by handling HTTP requests and rendering templates. In our case, it integrated the `RestaurantRecommender` into a `Flask` application and designed a user-friendly interface using HTML, CSS, and JavaScript.

Some of the key techniques and tools used in the web-based implementation include:

`Flask`: to build web applications with Python. It serves as the backbone of this application by handling routing, requests, and responses with my processed database.

`HTML`: to structure the content of the web page, including input fields, buttons, and other elements needed for the user interface.

`JavaScript`: to add interactivity to the web page, allowing us to capture user input, communicate with the backend Flask application, and update the page content dynamically. I rewrite some simple functions from my previous Jupyter Notebook interface to HTML files.

By leveraging these technologies and tools, we were able to create a web-based interface that provides a seamless user experience while interacting with the `RestaurantRecommender` module. These two Vboxs were rewritten as two HTML files that providing the same functionalities.

2.3 Results and Conclusion

This project generates an interface consists of two main sections: **Restaurant Recommender** and **Dine Dice** (see Figure 10). The first section allows users to search for restaurants based on a specific cuisine, zip code, and search radius. The second section provides a random recommendation based on the user's customized pool of cuisines and their selected zip code and radius. Users can switch between these two sections using the provided buttons or links.

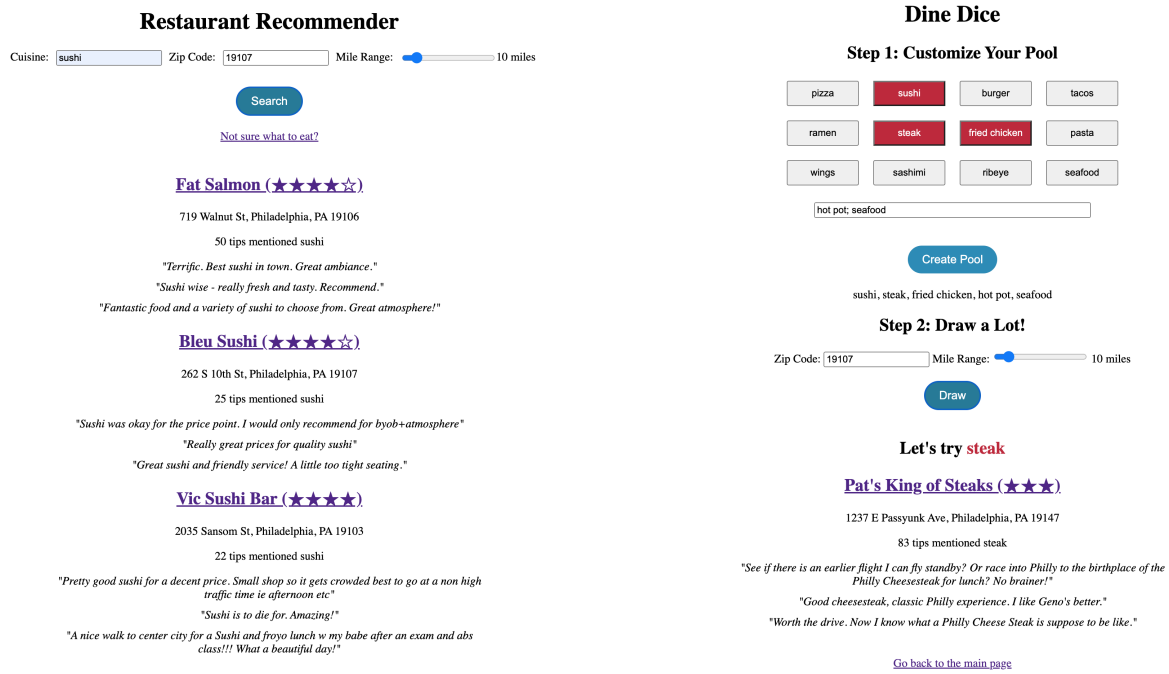
In the **Restaurant Recommender** section (Figure 10a), the user can enter their desired cuisine, zip code, and search radius, then click the *Search* button to receive recommendations. The recommendations are displayed with the restaurant's name, star rating, address, and a selection of tips related to the chosen cuisine. The restaurant's name also links to the Google Maps page of this restaurant so that the user can get even more information and direction.

In the **Dine Dice** section (Figure 10b), the user can first customize their pool of cuisines by selecting from a list of predefined options or by manually entering their preferences, separated by semicolons. After submitting the customized pool, the user can enter their zip code and search radius, then click the *Draw* button to receive a random recommendation from the pool. The recommendations list has the same format as the **Restaurant Recommender** section, but only one restaurant. This section should help users who have indecisive thoughts about their dinner.

Please refer to the Instructions for Running the Software for more information on how to view the interactive interface.

In conclusion, a restaurant recommendation system was developed that caters to users who are looking for specific cuisines or those who want a random suggestion. By utilizing NLP techniques, web technologies, and user-friendly design, this project provides a seamless experience for users, enabling them to discover new restaurants and make informed dining decisions. The flexibility and adaptability of the system make it a valuable tool for users with diverse preferences and requirements.

2.4 Graphics and References



(a) Restaurant Recommender

(b) Dine Dice

Figure 10: Example of Interface with Results

Files related to this sub-project:

```
sub-project_2/
|-- app.py
|-- RestaurantRecommender.py
|-- templates/
|   |-- draw_page.html
|   |-- index.html
|-- tip_business.csv
|-- tip_data.ipynb
|-- tip_recommender.ipynb
|-- tip_text.csv
|-- yelp_dataset/
|   |-- yelp_acadamic_dataset_business.json
|   |-- yelp_acadamic_dataset_tip.json
```

Instructions for Running the Software

The source data used in this project is contained in two files within the `yelp_dataset` folder: `yelp_academic_dataset_business.json` and `yelp_academic_dataset_tip.json`.
To process the source data, run the `tip_data.ipynb` notebook, which will generate two CSV files: `tip_business.csv` and `tip_text.csv`.

In the `tip_recommender.ipynb` notebook, a custom Python class is defined, and the interactive interface is created. The `RestaurantRecommender.py` file is a standalone version of the custom class, and `app.py` contains the Flask configuration.

The `templates` folder includes two HTML files: `index.html` and `draw_page.html`, which correspond to different parts of the interactive interface.

To check **Jupyter Notebook implementation**, please open `tip_recommender.ipynb` and run-all the notebook; this interface should show up under the last cell.

To check **Web page implementation**, please run `app.py` by typing `python app.py` on the terminal. The `Flask` server then should run on `http://127.0.0.1:5000/`