Intelligent Dining Decision Assistant Application Documentation and Final Report

Team Members

Pauline Brunet (NetID: pbrunet2) (Captain)

Louis Hamilton (NetID: louisch3)

Austin Harmon (NetID: austin31)

Mitchell Kopczyk (NetID: kopczyk2)

Catherine Orlando (NetID: co24)

Project Overview

The Intelligent Dining Decision Assistant Application is a Python program that offers a graphical user interface (GUI) for finding restaurants based on user preferences and, optionally, location. It serves as a convenient tool for searching for restaurants by entering keywords or phrases related to food preferences and refining the results by specifying a city and state. The core functionality of the application utilizes the Okapi BM25 ranking algorithm to provide users with relevant search results, ensuring that the most suitable restaurants are presented first.

Implementation Documentation

This software is implemented in Python and comprises two main components. Firstly, the "RestaurantSearch" class handles the backend logic of the restaurant search. It initializes the program by creating an inverted index of restaurant documents and computes essential document statistics, such as document length and average document length. This class also loads restaurant data from text files located in a designated folder, indexes them for search, performs searches based on user queries, and formats search results for display. It also calculates relevance scores using the Okapi BM25 ranking algorithm and offers the option to filter results by location. Secondly, the "GUI" class is responsible for creating the graphical user interface using the Tkinter library. It sets up the main window, initializes GUI elements such as text entry fields and buttons, and connects the GUI to the "RestaurantSearch" class.

Setup Instructions:

To use the Intelligent Dining Decision Assistant Application, follow these steps.

1. Ensure you have Python installed on your system. The code relies on the Tkinter library for the GUI, which is typically included with Python, eliminating the need for additional installation.

2. Either download the files or clone the repository into a private repo

   a. If downloading, place the restaurant data text files in a folder named "restaurants" within the same directory as the script. Each text file should represent a restaurant, with the restaurant's name and location on the first two lines and the restaurant's description on subsequent lines.

   b. If the repository was cloned ensure that both the restaurant_search.py and the restaurants.zip files appear by using the “ls” command. The restaurant.zip file needs to be unzipped prior to running the code. Type the command “unzip restaurants.zip” on the command line.

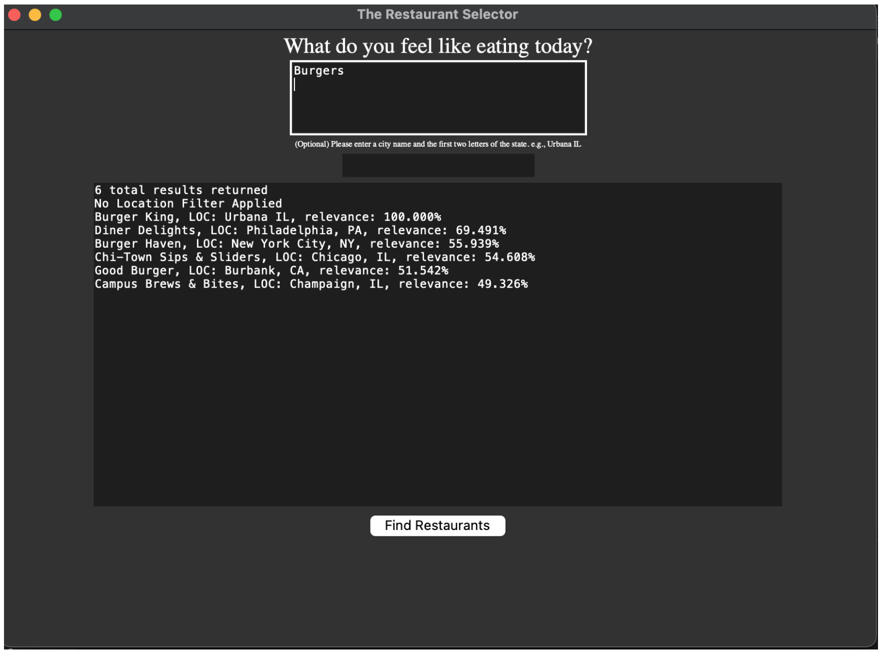
3. Run the application by executing the Python file in your integrated development environment (IDE) or in your terminal using the command “python restaurant_search.py restaurants”. This action opens the GUI window.

4. To search for restaurants, enter your desired food-related query in the first text entry field (specified by “What do you feel like eating today?”).

5. (Optional) In the second text box (specified by “Please enter a city name and the first two letters of the state. e.g., Urbana IL”) input a city name and the first two letters of the state to apply location-based filtering.

6. Click the "Find Restaurants" button to initiate the search. The results will appear in the text area below the search input fields, displaying restaurant names, locations, and relevance scores. If a location filter was applied, only relevant results matching the specified location will be displayed. You can extend the application by adding more restaurant data text files to the "restaurants" folder. Overall, this user-friendly application streamlines the process of finding restaurants based on food preferences and location, making it a valuable tool for restaurant enthusiasts.

Screenshots of Example Results:

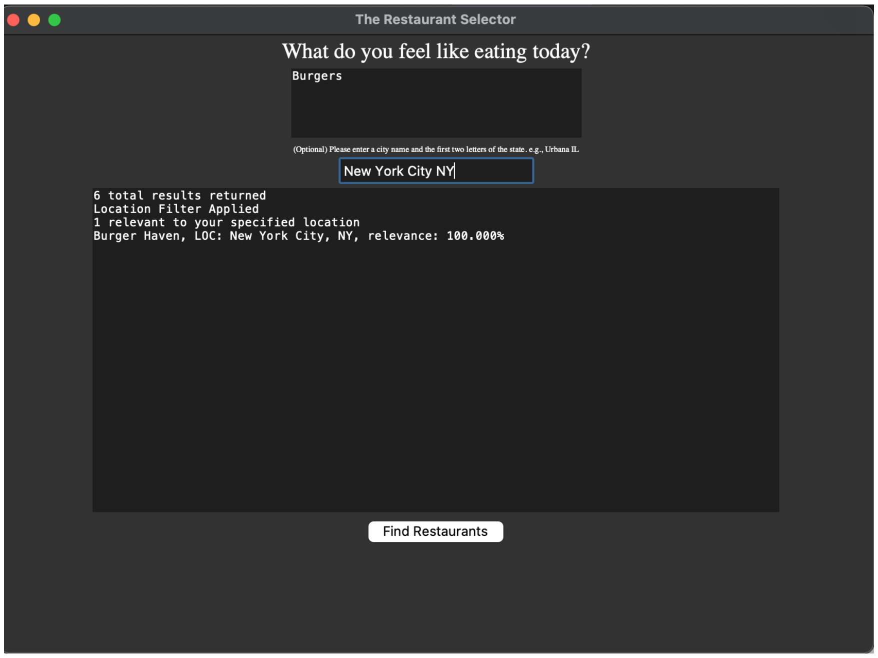


Burgers

(Optional) Please enter a city name and the first two letters of the state. e.g., Urbana IL

```
6 total results returned
No Location Filter Applied
Burger King, LOC: Urbana IL, relevance: 100.000%
Diner Delights, LOC: Philadelphia, PA, relevance: 69.491%
Burger Haven, LOC: New York City, NY, relevance: 55.939%
Chi-Town Sips & Sliders, LOC: Chicago, IL, relevance: 54.608%
Good Burger, LOC: Burbank, CA, relevance: 51.542%
Campus Brews & Bites, LOC: Champaign, IL, relevance: 49.326%
```

Above image shows the GUI result when inputting the food item “Burgers” without a location added. Result returns all relevant restaurants and their relevance score.



Burgers

(Optional) Please enter a city name and the first two letters of the state. e.g., Urbana IL

New York City NY

```
6 total results returned
Location Filter Applied
1 relevant to your specified location
Burger Haven, LOC: New York City, NY, relevance: 100.000%
```

Above GUI image shows results inputting a food item “Burgers” along with a location “New York City NY”. Output returns relevant restaurant “Burger Haven”, its location “New York City, NY” , and the relevance score “100.000%”.

Project Workload Breakdown:

Mitchell Kopczyk: Undertook the task of expanding our restaurant dataset by adding more restaurant entries. He also dedicated efforts to enhance the codebase by refactoring it into a more structured component-based architecture. This strategic refactoring has significantly improved the organization and overall structure of our application, resulting in a more robust and well-organized software system.

Catherine Orlando: Aided in software testing. Ensured that all users are able to download and run the code successfully. Helped combine the restaurant documents to include an expanded list. Helped with writing reports and final documentation.

Louis Hamilton: Refactored code to provide the necessary statistics for BM25 and implemented the BM25 formula within the code. Scaled the final relevance results by setting the highest to 100%. Improved string comparison for the location filter by handling punctuation, spaces, and capitalization. Created the presentation video.

Austin Harmon: Proposed topic, generated base code to produce a GUI that generates restaurant results based on the provided food input. Added location filtering. Helped write reports.

Pauline Brunet: Managed project execution, followed-up with team members, prepared reporting, merged code into repo, performed testing and QA, helped writing reports and documentation.