Final Project Report

Submitted by: Joyce Guo, Kasey La, and Sohini Roy

Introduction

Pantry Pal is an application that functions as a recipe recommendation system. It is meant to be useful for users that have many ingredients but don't know what to make with them. Pantry Pal suggests what recipe to make, along with the full ingredient list and directions of the recipe, depending on the user input of ingredients.

Methodology

In order to search for recipes based on user input, we created a vector database of recipes. All recipes were from a dataset on Kaggle, containing more than 2 million recipes in total. For the purposes of this project, we scaled it down to 50,000 to make working on the dataset more manageable. In this dataset, each entry consisted of a recipe, which included the recipe title, ingredients with measurements, directions, a recipe link, the source, and the ingredients without measurements, which was compiled by the creators of the dataset using NER. We then used a vector similarity search over the vector database to index relevant recipes. We did not use any chunking due to the recipes being relatively short and thus an appropriate context window without any chunking.

We included an option that allowed for inclusive and exclusive results, with inclusive meaning that the recipe contains some of the queried ingredients but may have others, and exclusive meaning that the recipe contains at most all of the queried ingredients, but importantly not anything extra. We anticipated that these two modes would be helpful for the user depending on if they were willing to buy new groceries or if they simply wanted to use everything they already had.

To ensure consistent ingredient matching, the user selects from a predefined list of ingredients defined from the dataset in order to make their query. This helps with better matching and allows easier implementation of set logic for the inclusive/exclusive filtering. In addition, to accommodate for preferences, the user may also optionally add on certain fields to help limit their search, such as dietary restrictions and specific ingredients that they might not like or want to use.

Dietary restrictions are implemented using a rule-based filtering system. Each restriction contains a list of ingredients that would violate said restriction, and recipes are checked and filtered from them. Specific ingredients to avoid are filtered similarly, but rather than a predetermined list, the filtering is based on the user's inputted ingredients they wish to avoid.

The additional keyword search is to help the user further narrow down their search based on things that aren't prompted in the other fields. For example, if a user has tomatoes and wants to make dip with it, the user can input "sauce" or "dip" here to specify that. This search is also done with a vector similarity search over our vector embeddings.

The frontend has multiple fields for the user to enter relevant information, namely the user's list of ingredients to search for, the search method (inclusive/exclusive), an optional list of dietary restrictions, an optional list of ingredients to avoid, and an optional field of text for any additional keywords to limit the search. The user also has the option to login in order to save recipes so that they are easy to find. To use the interface, the user simply inputs the necessary fields and the search will return a list of recipes.

Implementation

There were a few steps we took for processing the dataset. We cleaned the ingredients list column and also joined the directions column entries from a Python list of single directions to a

string of numbered directions. In addition, we removed several recipes that were not edible or intangible (such as “a happy marriage” and “finger paint”).

Our frontend is implemented using Streamlit and SQLAlchemy. The figure below is a screenshot of the user interface. Several search results will show on the webpage for the user to view. The backend database allows users to optionally login which opens up the option to save recipes that they would like to save for future reference using SQLite.

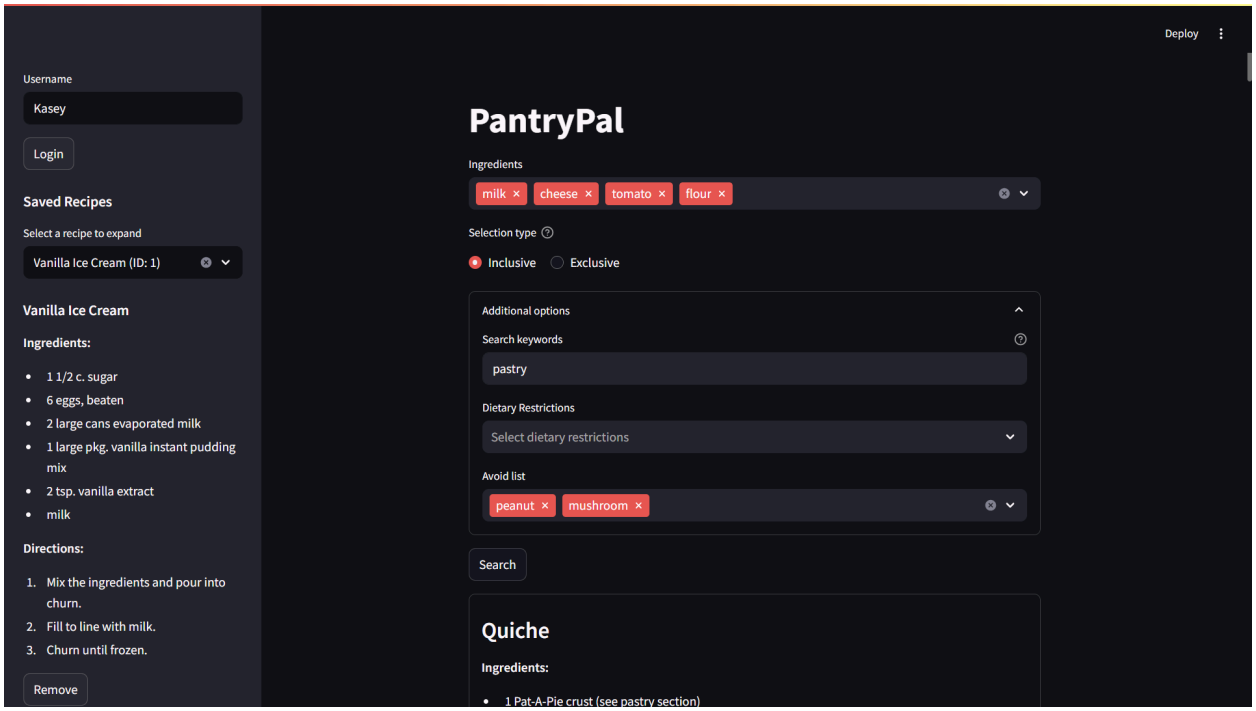

Figure 1. Screenshot of PantryPal user interface

For the vector database, we used a sentence embedding model (all-MiniLM-L6-v2) from Hugging Face to build the dense embedding space. The search component for both the ingredients list and the optional keyword text is implemented using Facebook AI Similarity Search (FAISS). If the user does not give any keyword(s), the app retrieves the top 1000 recipes from the database (to ensure that there are an adequate number of recipes to filter) using the ingredients list as the embedding. If the user does input keyword(s), the app does the same

ingredient-based retrieval. Then, it re-ranks those retrieved recipes using a second semantic search, using the given keyword as the embedding. After this, the retrieved recipes are filtered using set logic depending on whether the user selects their mode to be inclusive or exclusive.

The retrieved recipes can also be filtered based on dietary restrictions and ingredients to avoid. The dietary restrictions are implemented using a json file consisting of a list of restrictions and ingredients that would violate said restrictions. For both of these filtering restrictions, if a retrieved recipe contains a given ingredient, it is excluded from the final output.

Challenges

The ingredients in the dataset were not very clean due to poor extraction efforts from the creators of the dataset. For instance, many recipes contained “artichoke hearts” in the list of measurements, however, in our smaller subset of 50,000 recipes alone, there were 89 instances of only the word “hearts” found in the ingredients. There were also intangible results such as “points” and “long”. Due to these issues, we decided to perform ingredient extraction from the measurements ourselves, though that came with many complications as well. Minimally, we were able to collect a large set of words to remove, including many units, their plural forms, and their various abbreviations (e.g., “qt”, “qts”, “quart”, “quarts”). We also considered removing verbs from the measurements using spaCy, but that caused issues with those that are necessary for differentiating different ingredients (e.g., “condensed milk” and “evaporated milk” are not the same as one another nor are they the same as “milk”). The finalized list of ingredients is still not perfect, as singular and plural forms of ingredients are treated as separate ingredients, but there was not enough time to handle this problem. Along the same lines of the ingredients column not being clean, some of the directions for recipes were not fully scraped (i.e., multiple lines in the

original recipe not included in the dataset) or were scraped incorrectly (e.g., an individual line of direction being split into multiple lines).

Our original plan was to allow users to filter recipes by cuisine type (e.g., Asian, European) as well as heat type (e.g., stove, oven, no-heat). We wanted to implement this by processing the data more but due to the lack of time and constraints with data mining, it was too difficult to extract that information for each recipe. However, we ended up implementing this to some degree with the keyword search feature, which we found works quite well, especially for common things like “pasta”, “chinese”, and “pastries”. Dietary restrictions were also too difficult to extract, however, we were able to get a feasible list from GPT-4o though there are other issues on that front due to ingredients not being fully normalized.

We initially also wanted this application to be RAG-based. Our first pipeline involved taking in a user’s input as text and then simply giving the LLM an augmented prompt, possibly with some other fields included from the user input. However, as we worked more on the project, we found that the addition of an LLM’s output would not add much to the major features we wanted to implement. The crux of this application comes from the similarity search and set filtering functions. Adding in an augmented prompt with retrieved recipes would perhaps be helpful for cuisine preference (which the keyword search already does a decent job at), but would not help much for the inclusive/exclusive filtering. Namely, we felt that LLMs would not be good at set logic, and therefore it would be better to simply use Python sets and logic for the inclusive/exclusive filtering. In addition, because the intended output for our system would always be a recipe or multiple recipes, it was fairly straightforward to simply output what the similarity search had returned, making a RAG-based application seem less helpful.

Contribution

Joyce

- Wrote most of this report
- Implemented optional keyword search function (*search.py*)
- Helped with formatting directions in preprocessing (*preprocess.py*)
- Helped modify FAISS index (*faiss_index.py*)

Kasey

- Extracted data from the dataset for data mining purposes (*extract_dataset.py*)
- Removed bad recipes that have inedible ingredients (*cleanup.py*)
- Minimally extracted ingredients from measurements via *extract_ingredients.py* (*cleaned_ingredients.json*)
- Developed the frontend on Streamlit (*app.py*)
- Wrote the README file containing instructions for running the application

Sohini

- Preprocessed data i.e., scaled down dataset size, cleaned list columns, added to JSON (*preprocess.py*)
- Helped create FAISS index (*faiss_index.py*)
- Helped with *search.py* to retrieve search results given ingredients from user
- Created dietary restrictions list (*dietary_restriction_exclusion_list.json*)