Menu Engineering Supercharged

MDS Capstone Final Report

Hankun Xiao, Yasmin Hassan, Jessie Zhang, Zhiwei Zhang

Table of contents

1 Introduction

This technical report serves as a behind-the-scenes narrative of our capstone project for Heymate!. It documents how we tackled technical challenges and highlights key decisions made during development and deployment. It also provides a reference for future development.

2 User Requirement Collection and Solution Consulting

This capstone project with Heymate! differed from typical capstone projects. Usually, the project partner provides specific objectives for the deliverable as well as the dataset. In the project proposal from Heymate!, they expected our data output as a dashboard to improve customer retention. However, at the beginning of the project, we did not receive a useful dataset.

Given this situation, we had a two-hour meeting with the partner to understand their business needs and consult on potential solutions. We received clarification that the partner actually wanted a **menu recommendation model** ("the recommendation model") for their clients

(restaurant owners). For example, one of Heymate!'s clients, a Chinese Restaurant in Vancouver, wanted recommendations on what items to add and what items to remove, following marketing trends.

Correspondingly, we agreed with the project partner and the project supervisor that we would shift the project from "Customer Retention Supercharged" to "**Menu Engineering Supercharged**," and we agreed upon the techinical deliverables:

- A data cleaning module that can clean menu data and extract key features.
- A knowledge base (database table) with proper schema design and initial data.
- A recommendation algorithm.
- A visualization demo for reference.

3 Project Preparation - Dataset

To build the recommendation model, we needed training data. We compared several datasets shared by the partner, our advisor, and our own research:

- **Dothub Dataset Shared by the Partner:** This dataset is 66.4 GB and consists of 6,479,347 rows, including restaurant names and addresses in the US. We didn't choose this dataset because it did not include restaurant menu data.
- **Yelp Dataset Shared by the Partner:** This dataset includes 6,990,280 reviews from 150,346 restaurants. While restaurant reviews could potentially be a good feature for building the model, it lacked menu data. This limitation made it unsuitable for direct use.
- **What's On The Menu? Dataset by New York Public Library:** This dataset includes about 45,000 menus from the 1840s to 2016, containing restaurant information and menu data. However, the dataset is old and doesn't necessarily reflect the most up-to-date trends, so we didn't choose it.
- **Uber Eats Dataset ("the Uber Eats Data"):** This dataset includes over 63,000 restaurants and more than 5 million menus from Uber Eats in the US, collected in 2023. It contains restaurant names, review information, and individual menu items. Due to its recency and completeness, we agreed on this dataset with the partner.

4 Dataset EDA and Feature Engineering

Given the nature of the data, there were missing values, spelling mistakes, variations, and language differences. We needed to apply methods to clean the data and perform necessary feature engineering before proceeding with the analysis.

For feature engineering, we aimed to extract **dish base** and **dish flavors** from the item name, category, and descriptions in the menu. Here is an example:

> **Original Menu Item:** "Classic Cheeseburger"
> **Category:** "Classic Burgers"
> **Description:** "Smashed beef patty with cheddar cheese and your choice of toppings and sauce."
>
> **Extracted Features:**
> - **Dish Base:** Burger
> - **Dish Flavors:** Cheese, Beef

We considered the following approaches:

- **Regular Expressions:** Regular expressions are effective for processing data with consistent patterns. However, in our data, each restaurant had its own preference for writing the menu, making this method inconsistent for universal application.
- **An Existing Word Embedding Model from HuggingFace, Specialized with Menu Corpus:** This method was suggested by Professor Verada, who specializes in machine learning within the UBC MDS teaching team. Unfortunately, the Uber Eats data contained information in different languages, and we couldn't find a suitable multi-language model from HuggingFace.
- **A Generalized Large Language Model (LLM):** This method involves using an external LLM to clean the data. We added specific instructions through prompt engineering on how the data should be cleaned and engineered. The processed data was then retrieved from the model. Additionally, the partner shared their organization's ChatGPT subscription with us and provided API tokens. After balancing cost and efficiency, we chose the **GPT-4o mini model**.

Here is a table comparing the different approaches:

Approach	Pros	Cons
Regular Expressions	Fast to compute; Free	Limited flexibility
Existing Word Embedding Model (HuggingFace)	Semantic understanding; Efficiency; Free	Multi-lingual challenge
Generalized Large Language Model (LLM)	Highly flexible; Supporting Multi-Multi-lingual; Semantic understanding	Costly; Slow; API Reliance

5 Research on the Recommendation Algorithm

We researched three common recommendation methods used in the industry: **Collaborative Filtering**, **Content-Based Filtering**, and **Popularity-Based Recommendation**.

- **Collaborative Filtering / Matrix Factorization:** This method is widely used in industry, including by companies like Netflix, to generate recommendations based on user-item interaction data (e.g., purchase history or user ratings). However, in the Uber Eats dataset, we do not have access to such user interaction data, making this method infeasible for our use case.
- **Content-Based Filtering with Vector Embeddings:** This involves using a pre-trained large language model to convert menu items from human-readable text into numeric vectors. For example, "Fried Chicken" might be transformed into $[0.1, 0.5, 0.9]$, while "Dim Sum" could become $[0.2, 0.6, 0.1]$. These numeric representations allow us to quantify the similarities and differences between different cuisines, enabling further recommendations. Unfortunately, due to time and resource constraints, we were unable to deploy a large language model locally or fine-tune one.
- **Popularity-Based Recommendation:** Popularity is a widely accepted baseline metric in the industry, used by platforms like Yelp, Spotify, and GitHub. For instance, GitHub calculates a popularity score based on the number of forks a repository receives. For Heymate!, we designed a custom formula that incorporates the **number of reviews**,

average rating of a restaurant, and the **frequency of a cuisine** to calculate a popularity score. These data points are available in the Uber Eats dataset. Additionally, this method is easy to understand and communicate to restaurant owners. Given its advantages in both interpretability and feasibility, we chose to adopt a **Popularity-Based Recommendation** approach in our project.

6 Techinical Implemetation

During development, the team utilized **GitHub** and **Slack** for communication and collaboration. We modularized each component for future improvement.

- `visualization_demo.ipynb`: Demonstrates data visualizations, likely with charts or graphs.
- `knowledge_base_update.ipynb`: A notebook manages the task of updating the knowledge base of menu data.
- `archived_function_app.py`: Stores deprecated or old Azure Function logic.
- `batch_cleaning.py`: Cleans raw menu data in batches.
- `credential_validation_test_unit.py`: Unit tests for verifying database and API credentials.
- `flask_deploy.py`: Deploy the data cleaning module locally under the Flask framework.
- `menu_recommender.py`: Recommends menu items and return the output as a JSON object.
- `popularity_score_calculator.py`: Calculates popularity scores for items.
- `util_database_reader.py`: Utility to read data from a database.
- `util_database_uploader.py`: Utility to upload or write data to a database.
- `util_llm_data_cleaner.py`: Utility to send and retrieve the data to ChatGPT's API.
- `util_task_logger.py`: Logs tasks or events during execution for tracking/debugging.

7 Deployment of the Data Cleaning Module

7.1 Research and Decision Making

Due to the numerous amount of data (3 million rows), we decided to deploy the data cleaning module using a **distributed deployment approach**. This means multiple working instances would clean batches concurrently and run on cloud machines. We considered three approaches:

- **Google Cloud Function (part of Google Cloud Platform):** One team member had experience working with it in the past; however, the partner did not have a Google Cloud Subscription.
- **Flask Function on an EC2 instance on Amazon Web Services (AWS):** This framework was introduced in the Cloud Computing Course during the MDS program. However, the partner did not have an AWS Subscription.
- **Azure Function:** Azure Function is similar to Google Cloud Function, and working instances can be triggered via an HTTP web request. The partner had an Azure subscription and shared the credentials with us.

7.2 Initial Deployment and Blockers

Given the cloud computing resources from the partner, we chose **Azure Function** as our primary deployment plan. To automate the deployment process, the team utilized the continuous deployment workflow from GitHub by adding the following flow:

```
- name: 'Deploy to Azure Functions'
  uses: Azure/functions-action@v1
  id: fa
  with:
    app-name: ${{ env.AZURE_FUNCTIONAPP_NAME }}
    package: script
    publish-profile: ${{ secrets.AZURE_FUNCTIONAPP_PUBLISH_PROFILE }}
    scm-do-build-during-deployment: true
    enable-oryx-build: true
```

Unfortunately, the deployment failed due to security setup issues. The partner had never used Azure Function before, and their engineers did not have the time to configure the proper security settings for us. We also did not want to risk compromising security, as the database stores sensitive business and customer information.

7.3 Alternative Deployment

We decided to deploy the data cleaning module locally on our computers under the **Flask infrastructure**. We could still utilize the distributed deployment framework to accelerate the process. With the team's effort, we successfully cleaned:

- All of Heymate!'s internal menu database, including over 6,000 entries.
- More than 30,000 entries in the Uber Eats dataset.

8 Suggestions for Future Development

We provided detailed user instructions on how to set up the environment and execute the code in the **README file of the repository**. There are a few areas where this project could be extended:

- **Deployment of the Data Cleaning Module:** We currently deploy the code locally under the Flask framework. The code can be deployed as an Azure Function with proper setup. We provided `archived_function_app.py` as a reference code.
- **Diverse Dataset:** We currently ingest only one dataset, the Uber Eats dataset. Potentially, more datasets could be added to enrich the recommendation model.
- **Visualization Improvement:** We provided a demo visualization of the top recommended items Figure 1. When integrating the system into Heymate!'s restaurant management system, the developer should adjust the font and coloring to ensure that the visualization is coherent to the scheme.

- **Save cost on the LLM usage**: Save cost on LLM usage: To save costs in the long run, Heymate! can consider using an in-house local LLM such as Ollama. This can significantly reduce the usage cost of ChatGPT.

Figure 1: Visualization Demo”