# AI Dining Suggestion App

## CS 297 Report

**Bao Pham (009621001)**
**Advisor: Dr. Chris Pollett**

## Abstract

Trying to decide what to eat can be challenging and time-consuming. Google or Yelp are two popular search engines for food queries, and they offer a broad, filtered variety of restaurants around certain areas, but not necessarily customized for a specific user. With that being said, the goal of this project is making a mobile application which can make restaurant suggestions based on a user's personal preference. In CS 297, a skeleton of the application including front-end and back-end was implemented along with dataset preparation to train the AI model. Furthermore, CS 298 will focus on training and implementing reinforcement learning for the model.

## Introduction

Currently, Google and Yelp are two applications that provide lists of restaurants. Going through a list of possible options often takes time and still does not provide a suggestive list of what most likely suits the user's taste. In addition, when a user starts browsing for options on Google or Yelp, the reviews also play an important role on which restaurant that user would pick. However, user's taste at a certain time should not always depend on the reviews of a restaurant. In other words, just because a restaurant has bad reviews does not mean the user should not go there while it might have some dishes that the user likes.

Moreover, these applications do not really learn about a user's taste or observe a user's restaurant pattern to filter out options from the list. They simply gather the restaurants around a user's location at a certain time and apply some pre-set filters on top of it, such as price range. There have been a few papers that try to build a recommendation system for Yelp [4][5]. However, these AI models are trained based on the available restaurant data provided by Yelp, but not dynamic or temporal data that is also relevant to the user's taste on different situations. Thus, for this

project, the goal is to build an AI model that can learn from one's dining pattern over time to help make restaurant suggestions (with the occasional adventurous option) at any time. As this is a year-long project, the first half during CS 297 would focus on building up a skeleton for the app. This skeleton would include the front-end application for user interaction and the back-end to collect user's activities and prepare a dataset. This time will also be spent creating the jumpstart for the AI component. The second part of the project during CS 298 will mainly focus on the AI component as well as reinforcement learning for it.

The core and experimental component of this project is the AI model, which has two phases. Initially, the model would be trained based on the average user data which is collect from Yelp Dataset [10]. After that, the model would be enhanced with reinforcement learning to make better suggestions for a specific user.

The rest of this report discusses four deliverables that have been implemented during CS 297 and how they support the project as the whole.

## Deliverable 1: API Implementation with Google and Yelp data

Data collection is the first step needed for this project. The point of this deliverable was to create an API that collects data from Google and Yelp given a specific location, then responds to the front-end application's request with a list of restaurants. Google[6] and Yelp[7] are the main sources for restaurant data. At this stage, the data that is provided from the source should include name, location/address, price level, images and category.

Making sample requests for data from Yelp came with a couple issues: rate limit and image list limit. Yelp only allows ten queries per second and three images per restaurant. With that being said, Yelp data is now mainly used to cross-check with Google data and catch the restaurant that

Google may not have data for, as Yelp is more well known for its review data. On the other hand, Google does not have query rate limit and image list limit. Thus, the application runs all results from Yelp through Google for extra information (Figure 1).


```
{
    "geometry": {
        "location": {
            "lat": 37.344459,
            "lng": -121.838002
        },
        "viewport": {⌨}
    },
    "icon": "https://maps.gstatic.com/mapfiles/place_api/icons/restaurant-71.png",
    "id": "022beae10dff8f3de57a303f0e0605479c758cef",
    "name": "La Mejor Taqueria",
    "opening_hours": {
        "open_now": true
    },
    "photos": [⌨],
    "place_id": "ChIJG798FjfNj4AR7Th0RBwZWU8",
    "plus_code": {
        "compound_code": "85V6+QQ San Jose, California",
        "global_code": "849W85V6+QQ"
    },
    "price_level": 2,
    "rating": 4.3,
    "reference": "ChIJG798FjfNj4AR7Th0RBwZWU8",
    "scope": "GOOGLE",
    "types": [
        "restaurant",
        "point_of_interest",
        "food",
        "establishment"
    ],
    "vicinity": "2003 Story Rd # 975, San Jose"
},
```


Figure 1. A sample result from Google request

With all of the results from Google, the API should make sure there are no duplicates from both Google and Yelp and run them through the filters for price level, rating, radius and hours. Another challenge for this deliverable was how to limit the queries for Google by caching to reduce processing time and to prevent the application from surpassing the free tier. In order to do so, MongoDB was chosen to store the data for every request from Google. This certainly helped in reducing the number of queries because restaurant data is stored in the database. As a result, when the application requests for restaurant data and if the restaurants are in the database, the API can just return them instead of making repetitive queries to Google.

There is a huge amount of queries that are sent to Google because of the initial query to request restaurants at certain locations, as they only show one image for reference. Therefore, extra queries are needed to get more images (at least ten) where each image when being shown is

another query. However, the total number decreased a significant amount after database integration.

Once all of the requests were justified, a NodeJS server was created to allow the user to interact with the API via the application. It has been also hosted on Amazon AWS to make it more accessible for testing purposes in later stages (Figure 2).


Figure 2. Simple flow of the API

## Deliverable 2: External Services Integration for API

For this deliverable, the goal was to collect data from other external sources besides Google and Yelp to enhance the suggestions made by the AI model later on. Oftentimes, food decisions rely on some other factors, not the restaurants themselves. For example, during cold weather, people tend to prefer warm food like soup or pasta rather than sushi. In addition, other factors such as traffic or waiting time also influence one's decision on what to eat. Because of these reasons, extra temporal information also needs to be collected to help the AI model make better suggestions given a certain time of day.

As mentioned above, three extra types of information were added to the API: temperature at a given location, time to commute to the restaurant and the busy hours of the restaurant. With temperature, Open Weather Map[9] service is used (Figure 3). The reason it was chosen was

because its free tier is sufficient enough for testing purposes. Google Distance Matrix[4] and Google Place APIs serve as the main source for traffic information and busy hours.


Figure 3. Sample response from Open Weather Map

The challenge for this deliverable was at which point the extra information should be integrated to the flow of the application. It could be implemented after the user makes some action so that the AI model can learn about the user better, or it could be implemented as one of the filters to avoid outliers for the suggestions. However, that would make the AI model face a small over-fitting issue and would not be able to provide some interesting options.

After much consideration, having the external information applied before the restaurant data is sent back to the app would be the best option.

## Deliverable 3: Basic UI for the App

The point of this deliverable was to create a simple front-end application that can make queries to the API and parse the results into a list of restaurants for user interactions. When a user interacts with the application, data will be recorded and sent to API which in turn works with MongoDB database for training AI model purposes. In addition, a side goal of the app is to

create an “assistant” that will be able to learn about the user’s food preference and help narrow the lengthy list of restaurants down to a handful of possible choices user might actually like. With that being said, it also should come up with the suggestions quickly to overcome the traditional browsing-for-restaurant practice.

Inspired by Tinder, an application for online dating, which has changed the dating style to quick decision making by first appearance. Similarly, this application wants to do the same thing, but with restaurants instead of people (Figure 4).

There are two main buttons on the first scene of the app. One is for the user to save the restaurant to their favorite list if they decide not to go to that restaurant at the moment. The other button brings up Map applications so that it can navigate the user directly there. Besides that, the user would interact with the restaurant list via swipe actions. Swiping right is for going to the next restaurant on the list and swiping left is for going back to the previous one. On the next scene, this is for personal preferences or can be simply understood as a way to set up filters for what restaurants should be returned back from the API. It includes some basic settings such as price level, commute time or food category.

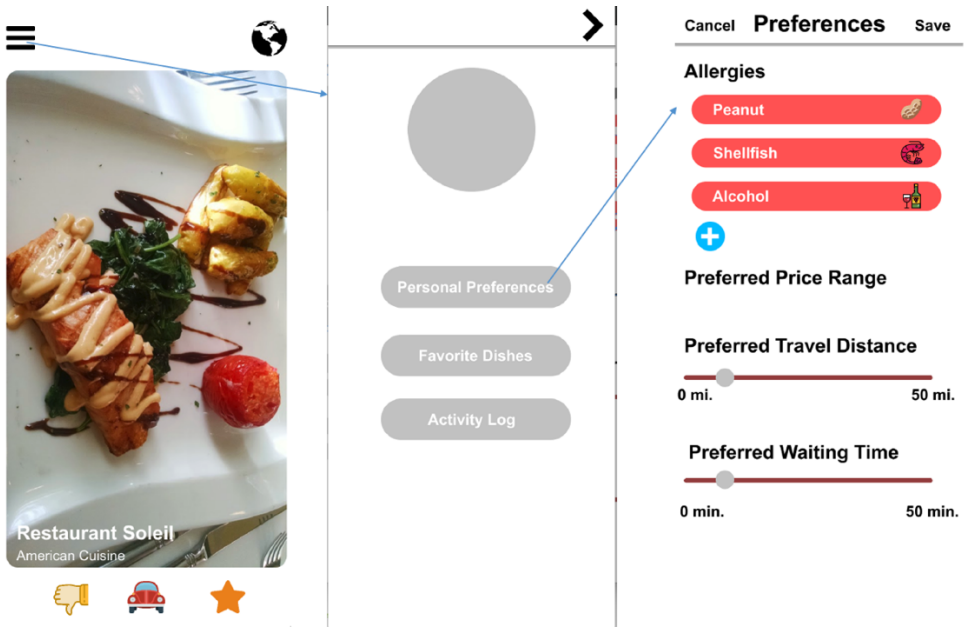


Figure 4. Simple mockup for the app

Some of these features might not be implemented as the main goal focuses on AI portion. However, the mockup offers basic ideas of what the app should look like. Due to time constraints and target platform for the app (mobile devices in this case), React Native technology was chosen to speed up the development time for mobile application and multi-platform deployment advantage. After a few development iterations, the features have been reconsidered in terms of user interactions. Dislike button was no longer needed as Dislike did not really define user's taste in the long run; using it might filter out the restaurants in same category unintentionally when the AI model goes through the list. Two navigation buttons were added to provide another way for the user to go through restaurant list. A Favorite button was created to replace the Like button so that user can save the restaurant for later visit. Within a restaurant card, a deck of images was also implemented so that the user can browse through and see what type of food the restaurant might have. Tapping back and forth to navigate between images was also enabled. When the user taps on the center of the image, zooming mode will be activated for viewing images in high resolution (Figure 5).


Figure 5. Fully functional basic app

## Deliverable 4: Prepare AI Model to be Trained and Observed

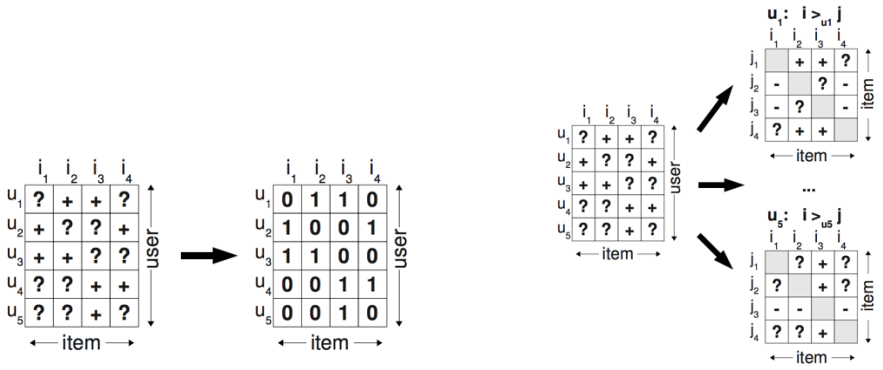
As mentioned above, one of the core components for this project is the AI model. Thus, the goal of this deliverable was to prepare the training dataset and test the model with a small sample of it. Starting with looking at what people have done for recommendation systems, there are a few well known projects, including YouTube Recommendation System[2] and the Netflix Prize[3].

In YouTube Recommendation paper, there are two layers of machine learning implemented. These are called Candidate Generation and Ranking. When compared with the restaurant data, they are somewhat similar. With restaurant data, there should be also two layers of machine learning. One is to filter out the restaurants from a long list for a certain location. After that, reinforce learning should take place to suggests a handful of restaurants that suit a specific user's taste.

Within Candidate Generation layer, a major part that sets out the outliers from the possible list is collaborative filtering. Looking at the Netflix Prize challenge, collaborative filtering is built based on a matrix between movies and users. A neighborhood model is rooted from the matrix to determine the similarities between users and movies. As a result, predictions are calculated and made from these neighborhoods.

Inspired by these papers, a neural nets approach is used for this component of the project with collaborative filtering. The dataset is obtained from Yelp Dataset[8] for training purpose. Originally, Yelp Dataset provides a few hundred thousand businesses with around eight million user reviews. After filtering out the data (which is only restaurant related), the dataset was left with around 70,000 businesses and 4 million user reviews. At this stage, small chunk of data was taken out for testing purpose with different neural nets algorithms due to limit training time.

The first algorithm that was experimented is Bayesian Personalized Recommender[1] (BPR). Unlike the usual approach that can be found in the Netflix Prize challenge in which the model is optimized to predict an element S in the list as 1 and 0 for the rest of the elements, BPR helps avoid the issue where the rest of the elements which will be ranked in the future are presented to the model as negative feedback (0 values). BPR proposes a different approach by using item pairs as training data and optimizes for correctly ranking item pairs instead of scoring single items because Netflix Prize challenge model would just replace the missing item with negative values (Figure 6). In order to do so, BPR uses a training dataset which consists of both positive and negative pairs as well as missing values, but in this case the missing values between two non-observed items are exactly the item pairs that have to be ranked in the future. This makes the training data and the test data disjointed from each other.




(1) (2)

Figure 6. Traditional model (1) and BPR model (2)

With a small subset of the dataset, the loss function clearly shows a rapid increase in accuracy after ten iterations. The small subset of data has 100,000 interactions between users and restaurants. These are the rating data of 41,654 users and 31,999 restaurants. Each batch size is

1000 ratings and the dimensional embedding is 20. For testing purpose, the result shows that the model can get down to 0.6 loss value and this is considered acceptable at this stage (Figure 7).

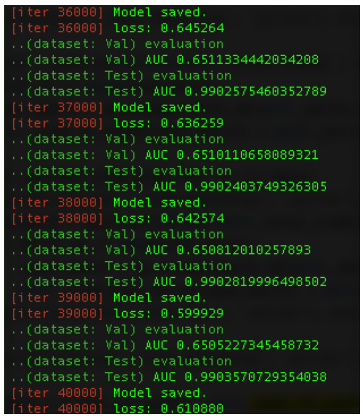



Figure 7. Result from training model with test dataset

## Conclusion

The main focus for CS 297 was to build a toolbox for the project so that during CS 298, these tools can be used to accomplish the project's goal. As this project commits to deliver an application, the skeleton of the application plays a critical role throughout the development process. Similar to building a house, the foundation should be well engineered before other details can be applied on top. Thus, for most of the time in CS 297, developing a smooth flow for data from front-end to back-end and vice versa was heavily prioritized.

In addition, connecting and polishing front-end and back-end of the application was also challenging. Due to the fact that not all restaurant data source is structured the same way, standardizing all data and making sure that all data is used efficiently was one of the difficulties during the development process. Moreover, the application would be very hard to use if the UI does not get to an acceptable performance level and, because of that, polishing as development continues was never undermined during CS 297.

With the above four deliverables the skeleton of the application is almost completed. This includes front-end, back-end and data collecting methods from the application. The rest of the work for CS 298 would mainly focus on training the AI model with possible modifications and implementing reinforcement learning to improve the suggestions for a specific user. Current API will be used to collect permanent data as well as temporal data to support learning process of the model.

**References**


1. Rendle, Steffen, et al. *BPR: Bayesian Personalized Ranking from Implicit Feedback*. arxiv.org/pdf/1205.2618.

2. Covington, Paul, et al. *Deep Neural Networks for YouTube Recommendations*. Google AI. Google AI, 1 Jan. 1970, ai.google/research/pubs/pub45530.

3. Chiang, Mung. *Networked Life: 20 Questions and Answers*. Cambridge University Press, 2012. *Chapter 4: How does Netflix recommend movies?*

4. Sawant Sumedh and Gian Pai. *Yelp Food Recommendation System - Machine Learning*. cs229.stanford.edu/proj2013/SawantPai-YelpFoodRecommendationSystem.pdf.

5. Naomi, Idan, et al. *Irene Ng Zack Kloock Recommender Systems Rheanna Gallego* math.uci.edu/icamp/summer/research/student_research/recommender_systems_slides.pdf.

6. Google Places API. https://developers.google.com/places/web-service/intro

7. Yelp Business API. https://www.yelp.com/developers/documentation/v3/business

8. Yelp Challenge Dataset. https://www.yelp.com/dataset/challenge

9. OpenWeatherMap API. https://openweathermap.org/api