:Ranking Dishes using Sentiment Analysis on Yelp Reviews

Brandon Harrison Alex Chen Evan Weiss Akshay Gupta

GitHub: https://github.com/BrandonHarrisonCode/yelp-sentiment-analyis

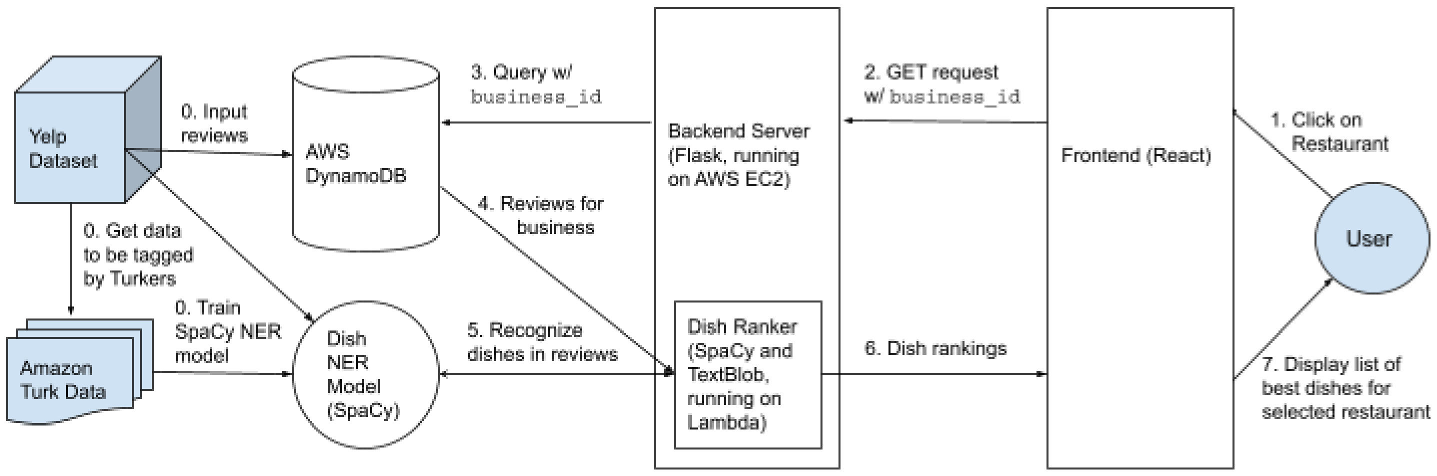
Introduction

Our goal in this project is to use NER and Sentiment Analysis tools on restaurant review data from Yelp (processed with help from Amazon Mechanical Turk) in order to develop dish rankings for restaurants, and provide these rankings to users through a friendly React frontend with Google Maps integration. Due to the time constraints for this project, we focused on design and implementation, but ran out of time and were not able to fully integrate our frontend and backend. We prioritized the NLP aspects of the project such as spaCy named entity recognition and TextBlob sentiment analysis, instead of the complete pipeline to make a robust site.

Dataset

We used the text dataset (specifically `business.json` and `review.json`) from Yelp's Dataset Challenge. This gave us a nice clean source of data to parse through for dish names and sentiment data.

Approach


Yelp Dataset
0. Input reviews
AWS DynamoDB
3. Query w/ business_id
Backend Server (Flask, running on AWS EC2)
2. GET request w/ business_id
Frontend (React)
1. Click on Restaurant
User
0. Get data to be tagged by Turkers
4. Reviews for business
0. Train SpaCy NER model
Amazon Turk Data
Dish NER Model (SpaCy)
5. Recognize dishes in reviews
Dish Ranker (SpaCy and TextBlob, running on Lambda)
6. Dish rankings
7. Display list of best dishes for selected restaurant

Above is a diagram depicting interactions between components of our architecture. Step 0 connections are preprocessed and run once in order to populate the DynamoDB database and train the NER model. Once those components are ready, we can support a basic user interaction, the steps of which are labeled in the diagram above. We will go into detail for each component of our architecture in this report.

Our project falls roughly into these stages:

1. **Entity Recognition Training:** We needed to be able to accurately tell what a dish was in any of the reviews we looked at. In order to do this we needed to create a large collection of training data of what is and what isn't a dish, and then train an NLP tool (in our case *spaCy*) to identify new dishes.
2. **Dataset Parsing:** The Yelp dataset contains thousands upon thousands of restaurants, and hundreds of thousands if not millions of reviews. This data needs to be organized so that each restaurant is paired with their corresponding reviews.
3. **Dish Extraction and Sentiment Analysis:** For each restaurant, each review must be combed for unique dishes. Once each dish is recognized, our project performs sentiment analysis on the surrounding sentence around the dish in order to see how people feel about each dish.
4. **Ranking and Databases:** The sentiment surrounding each dishes is accumulated and then scored so that the dishes can be ranked from most to least positively considered. This information is stored into an AWS DynamoDB database to cache the results for our frontend.

5. **User Interaction:** Using a React-based frontend, a user can see each of the restaurants displayed in a spatial orientation through Google Maps. Each restaurant is placed in accordance to its real-world geolocation, and a Flask server returns the top ten ranked dishes for each restaurant.

Recognizing Dishes

Collecting data with Amazon Turk

We decided to use Amazon Turk to collect data that we could later use to train a dish recognition model. To get started we took a sample of 1000 reviews from Yelp restaurant reviews and passed them into Turk. For each review we asked users to look through the review and find dishes mentioned by name that the restaurant serves. The users were then tasked to report all of the dishes mentioned in a comma separated or to put 'none' if the text contained no dishes. While the idea originally seemed like a great way to get training data, we ended up running into quite a few issues. The first was that many Turk users would try to cheat the system and just put "none" in the hopes that we would not spend time validating the responses. Because of this we had to block multiple users and relist their assignments. Additionally, an alarming number of the responses were of poor quality and contained multiple errors such as incorrect structuring, extraneous solutions, missing solutions, and of course misspellings. In the end it took more time to manually review the Turk responses than it would have to make our own dataset, but in the end we were able to salvage more than 600 training reviews that we could use.

Training a SpaCy NER Model

We leveraged spaCy's ability to train new entity types in order to recognize dishes in our data. Thankfully, given enough data, training a named entity recognition model through spaCy was straightforward. Training was a simple as giving an example sentence and an index location of the new entity: once spaCy was able to identify the dish it added the example sentence configuration to its own internal model. We also included training examples where no dishes were present to lower the chances of false positives. In order to do this we made a script that generated a list in this format from the Turk data we collected earlier. Originally, we used the entire review as the context to train spaCy on, but we quickly noticed that the excess of data was confusing spaCy and making our model wary of unrecognized dishes. Instead, we restricted the amount of data only to a sentence-by-sentence basis. Although the accuracy of our entity recognition improved significantly, our training time similarly increased as well. We considered the tradeoff to be worthwhile since the training was a one-time cost.

Ranking Dishes

Data Matching

In order to apply our model, we needed to generate new data. The Yelp dataset provides an abundance of both businesses and reviews, but this large amount of data was as crippling as it was helpful: reading through any of the data took minutes, and processing could take hours. Yelp recommended that we request the help of Big Data tools like Hadoop, but given the time constraints and our lack of experience in Big Data we had to restrict ourselves to preprocessing the data into more usable formats. We preprocess the businesses and reviews by stripping any businesses that aren't classified as restaurants. We then cull reviews that aren't associated with businesses that are left. This cuts the data that we need in nearly half, greatly speeding up the process of working with the data.

Creating ratings

Now that the data is preprocessed and our model is trained, we are able to identify dishes and perform our sentiment analysis. To do so, we iteratively progress through our data; first iterating through each restaurant, then each review associated with the restaurant, then each sentence in each review. SpaCy analyzes the sentences with our custom model to determine what dishes are in the sentence, if any. If a dish is found in the sentence, we run the sentence through TextBlob's pretrained sentiment analysis model to get the "polarity" of the sentiment, a measure of whether the sentiment was positive or negative and the intensity of the sentiment, in a range of [-1, 1]. This scoring process works on a review based level.

Aggregating ratings
In order to rank the dishes of the restaurant, we needed to aggregate each list of ratings from each review into a single score for each dish in the restaurant. The naive approach of averaging the ratings fails for dishes that are ranked infrequently: a dish ranked 1.0 but only ranked once would have a higher score than a dish with a thousand reviews that averages .9. Instead, our software implements Bayes Average Rating. Bayes Average Rating starts each dish at a "neutral prior" Each time the dish is mentioned, the ranking moves the score by a limited amount to create a moving average. This prevents a small number of reviews from swaying the whole average. Furthermore, a dish with more positive reviews will be scored higher than one with fewer as it is moved further away from the neutral prior. More information can be found in the above link and in the article "How Not To Sort By Average Rating".

Combining Similar Dishes
One heuristic we used in order to simplify and improve our rankings was to reduce dishes to the "base dish" to avoid having multiple rankings for similar dishes like `double cheeseburger` and `cheese burger with two patties and fries`. For this we decided to go with a simple similarity approach. After cutting out parts of dishes that start with the words `with` or `and` we did a jaro distance calculation with a threshold. The only problem was setting the threshold. We found that if it was too high it wouldn't merge dishes such as `soft scrambled eggs` with `scrambled eggs`. If it was too low, then it would combine distinct dishes such as `sweet roast` and `pork roast`. After a bit of testing, we found that 0.8 was a nice middleground that gave us the best results. Another benefit we realized after implementing this was that if the model makes a mistake such as including extra words in a particular sentence, it often gets reduced to its similar form. For example look at `octopus as well` in the below table:

Before combining similar dishes	After combining similar dishes (threshold = 0.8)
1 - dim sum 0.573 2 - braised beef 0.217 3 - spring rolls 0.208 4 - dim-sum 0.2 5 - octopus as well 0.174 6 - wonton 0.174 7 - cookies 0.174 8 - bunch our staples 0.167 9 - leaf wrapped sausage and sticky rice 0.167 10 - preserved egg congee 0.167 11 - egg yolk bun 0.167 12 - sticky rice 0.154 13 - shui mai 0.13 14 - fried noodles 0.13 15 - sautéed jumbo shrimp and grilled vegetables 0.13 16 - lobster 0.13 17 - fish fried rice 0.13 18 - chicken curry 0.13 19 - snow pea leaf with garlic 0.13 20 - beef sauté 0.13	1 - dim sum 0.598 2 - braised beef 0.292 3 - spring rolls 0.208 4 - wonton 0.208 5 - lobster 0.208 6 - cookies 0.174 7 - bunch our staples 0.167 8 - leaf wrapped sausage 0.167 9 - preserved egg congee 0.167 10 - egg yolk bun 0.167 11 - octopus 0.167 12 - sticky rice 0.154 13 - dumplings 0.148 14 - sautéed jumbo shrimp 0.13 15 - beef sauté 0.13 16 - pork 0.125 17 - beef fried rice 0.125 18 - steamed fish 0.12 19 - fried rice 0.111 20 - bubble tea 0.087
**Highlights are to show correspondences between the before and after lists*	

Creating the Web Application

Database

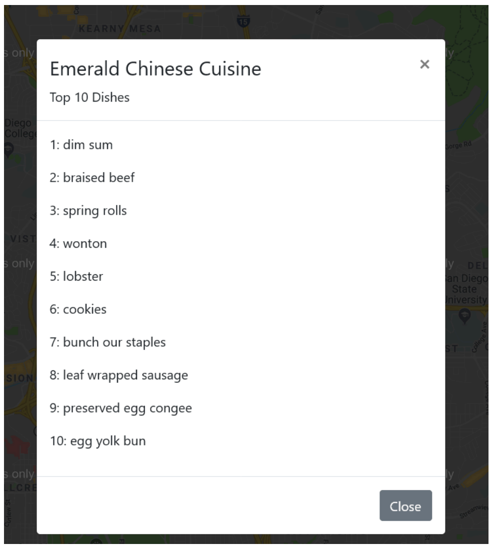
We stored all of our information in a cache implemented through DynamoDB. We stored information about each restaurant such as the name, the latitude, the longitude, the address, and a list of dishes along with a score for each dish. We implemented dynamic scaling onto the database to increase the read and write capacity of our database in response to load. This allowed our database to effectively deal with large writes or a large amount of concurrent users. Due to the amount of data, we attempted to parallelize as much of our data processing into DynamoDB as possible. We created AWS Lambda functions in order to split apart our data and process one restaurant at a time. This would have allowed us to have hundreds of computers simultaneously performing the data aggregation, sentiment analysis, and ranking steps concurrently so that multiple restaurants could be analyzed at the same time. We leveraged AWS's beta tool, the Serverless Application Model (SAM) for deploying our functions and interfacing with the database. However, we ran into several roadblocks that ultimately halted oru progress. Primarily, AWS Lambda functions can not be larger than 50 megabytes, and our trained model and NLP libraries easily put us over this limit. We attempted to work around these problems but ultimately had to abandon the notion due to time constraints and simply performed our preprocessing and analysis on our local machines and on temporarily created EC2 instances. Although this worked, the analysis only progresses at something around 500 restaurants per hour. With over 50,000 restaurants in our dataset, only a small subset was able to be processed by the conclusion of our project.

Backend

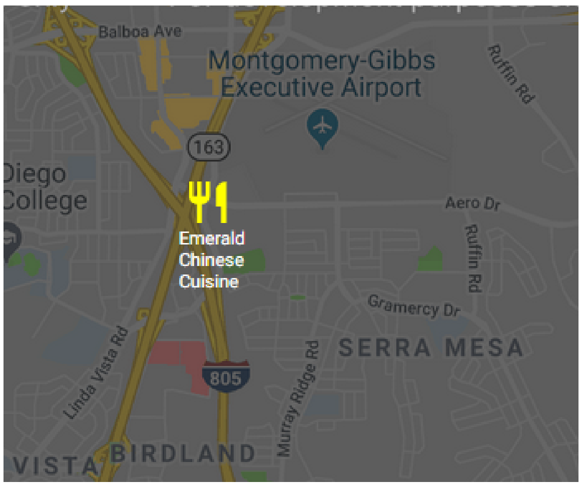
For the backend of our web application, we used Flask. We created two endpoints, both of which interacted with DynamoDB using boto3 to retrieve data. The first endpoint takes in a latitude, longitude and radius, and returns all restaurants within that radius from the specified coordinates - this is piped to the frontend and determines which restaurants to display on the map at a given point in time. The second endpoint takes in a restaurant ID and returns a ranked list of dishes at the restaurant - this is then piped to the frontend to be displayed when the user clicks on a particular restaurant. We deployed the Flask backend on an EC2 instance in order to allow our backend to handle greater load. AWS ECS was considered since we had developed many of our testing applications using Docker, but ultimately we determined that the transparency of managing our own server would lead to less errors in the long run given our short time frame.

Frontend

The frontend of our application was built using React, a popular Javascript library for creating usable and interactive web interfaces. The interface consists of a Google Maps embedding, wherein restaurants are marked with pins which the user can click on in order to display the top 10 dishes at a particular restaurant, as ranked by our model.


Emerald Chinese Cuisine
Top 10 Dishes

1: dim sum
2: braised beef
3: spring rolls
4: wonton
5: lobster
6: cookies
7: bunch our staples
8: leaf wrapped sausage
9: preserved egg congee
10: egg yolk bun



Lessons Learned And Limitations

We learned that Turk might not be the best tool to use for data is going towards a text-based extraction model. Turk can be great for other tasks such as drawing polygons around objects for image recognitions but is still error-prone. As mentioned in our section “Collecting data with Amazon Turk”, we ran into a number of issues from Turker’s being lazy, and verifying their work to ensure good data took a decent amount of manual labor on our part. In the future, it might be worth looking into other options for obtaining training data such generating it through a generative adversarial network or variational autoencoder.

Working with the huge files in the dataset was a bit of a pain. Since we are dealing with multiple gigabytes of data, even downloading it to our local machines takes dozens of minutes. Once we have the data files locally, loading the files into python takes minutes each time, making the development process quite slow. From this project, we learned the importance of developing with smaller test files, i.e. small subsets of the data that can be used to speed up runtime during development while emulating the data.

In addition, we found that AWS has great utility for dealing with scale while being not too hard to use. As one of our members (Brandon) had extensive experience with AWS, we were able to quickly spin up various AWS tools to deal with the large scale of data and help our website run efficiently in the cloud. We stored the large data files into a S3 bucket so that each group member didn’t need all of the data on their own computer. We created a DynamoDB instance containing the reviews and ranking of dishes for each restaurant that our backend server could easily interact with, preventing the problems that come with having multiple local databases. Finally, we used EC2 to host our backend server, allowing our website to potentially run indefinitely and at scale. We used Docker to ensure that code that ran locally was transportable and would run without issues in the cloud. These are all tools that we aren’t generally exposed to in class, but are essential for industry software engineering.

Given more time, here are some of the things we would like to explore:

- Improving our model for NER with more data from Mechanical Turk
- Training a targeted sentiment analysis model for the food domain, rather than a general pretrained sentiment analysis model.
- Incorporating additional review metadata into our ranking algorithm, such as the star rating of the review, the number of “useful” markers for the review, etc. which is provided to us in the Yelp dataset
- Improving our frontend with a search bar and other general usability and aesthetic improvements