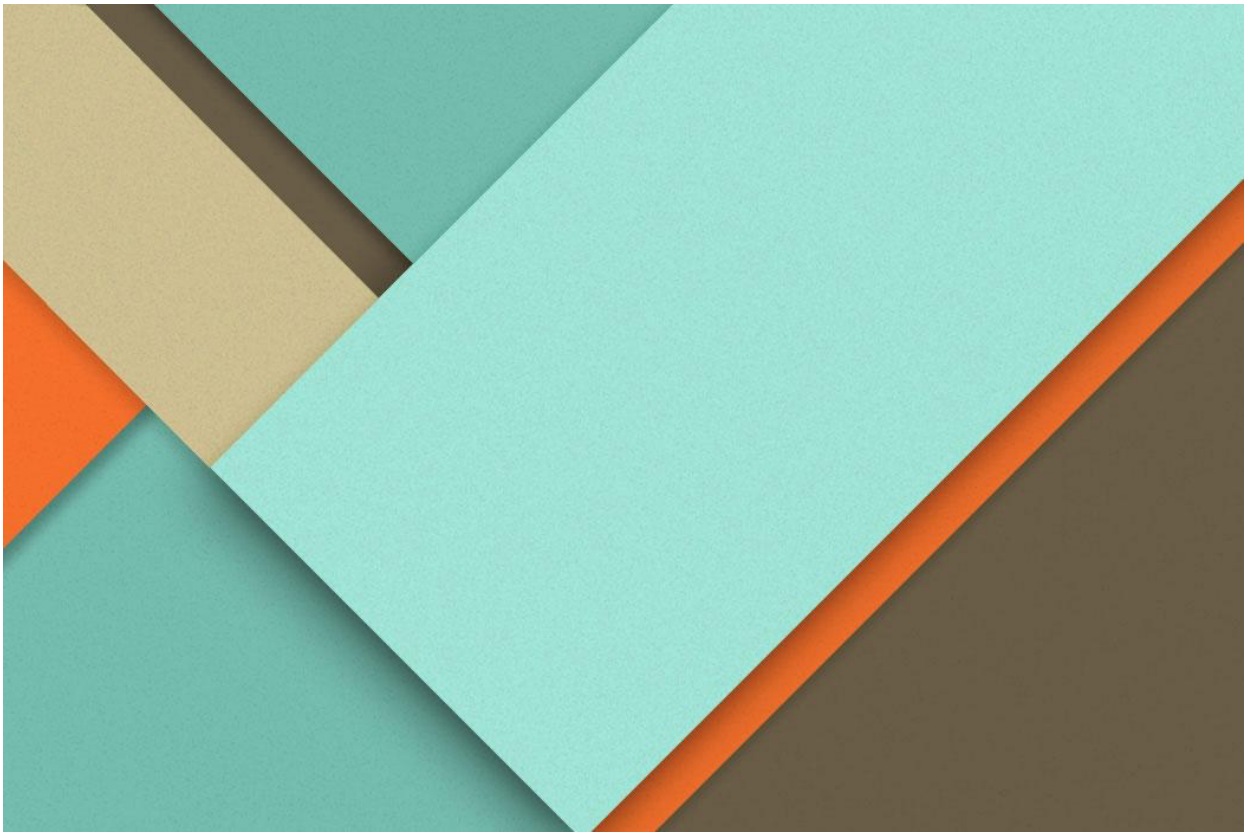

Friends Recommendation System

Jessica Z, Zhang
Nagendra V., Achanta
Jingwen Lu

Problem Statement

The COVID-19 pandemic has brought about many changes in the last couple of years. Personal interactions like eating and socializing with friends have been limited and travel plans are always up in the air. Now, as mask mandates are slowly being removed and vaccine requirements lifted, we are finally getting back into the groove of our new day-to-day lives, but things are not the same. The pandemic was accompanied by historic drops in output in almost all major economies and U.S. GDP fell by 8.9 percent in the second quarter of 2020, the largest single-quarter contraction in more than 70 years. Increased money supply and supply chain issues impacted the economy the most, forcing higher inflation which in turn had major impacts on the food industry and peoples' eating habits. Inflation is still high today and has driven up supermarket bills so much that, in some cases, it's more economical to dine in a restaurant than it is to shop and cook. There are numerous other studies stating there's been a surge in people eating out during pandemic based on Yelp reviews data.

For this project, we explored the possibility of using Yelp data to identify shared interests amongst people and encourage meetups. Since the founding of the application in 2004, Yelp has gathered immense amounts of crowd-sourced reviews of businesses and has amassed a large user base. However, even though Yelp offers customers a wide variety of information about businesses, it does not have strong user interactive functions that allow users to connect with people who share similar preferences. Therefore, users may find it challenging to identify businesses that suit their interest. To address this problem, we created a friend recommendation system for Yelp. This system employs machine learning algorithms to explore user data including their prior reviews, ratings, and check-ins. The system provides the user friend recommendations and allows them to connect with people who share similar interests. In addition, each pair is given a list of common interests they can explore together. Our goal was to enhance user experience on Yelp through this friend recommendation system and allow businesses to get more exposure through this process.

Data Source

Data Description

Our original data source contains 5 JSON files that is around 6GB total. Below is the description for each of them:

- Business.json: includes the data for each business on file (business name and business ID), location data (address, city, state, postal code, latitude, longitude), attributes (stars, review counts,hours) , and categories.
- Review.json: includes full text reviews for all businesses and the user who wrote them.
- User.json: includes user's friend mapping and the metadata associated with the user (total number of reviews, membership length and other's rating of the user's comment)
- Checkin.json: check-ins on a business. It records the time when each user makes a check-in for the business.
- Tip.json: Tips written by a user on a business which are shorter than reviews and aims to convey quick suggestions.

Data Cleaning

For our project, we mainly focused on the business.json, review.json, and user.json files. The datasets are quite large, ranging from 120 MB - 6GB, so we had to perform sufficient data cleaning to get a concise subset of information best suited for our problem's needs. Some of our initial thoughts were to focus on a particular location as well as a particular category of business, but we still wanted to allow enough variation so that the friend recommendations were more robust. Since our main goal is to match users based on shared interests, we made sure that when we performed the data cleaning, we focused on key metrics that brought value to these interests.

Business.json

Starting with the business.json file, since most of the data comes from businesses in the United States, we focused on states in the United States and filtered out all the non-US states. Our next decision was which categories to focus on. Although most of the data on yelp fall within the category of restaurants, people's interests encompass more than just food, so we wanted to consider other types of businesses too. We decided on an iterative approach to narrow in on the most highly rated businesses. Based on our observation, there are high frequency words for category description. In order to improve the result of recommendation and determine more specific common interests, we removed the high frequency words. First, we found the top 5 most common categories and kept only the businesses that fall under these categories. In our case, the top 5 categories ended up being: Restaurants, Food, Shopping, Home Services and Beauty & Spas. We then removed these very broad, high frequency categories from our category list so that we can focus on the smaller subcategories. We then repeated the process and found the top 5 categories in the new filtered business. In the 2nd round, the top 5 categories were: Nightlife, Bars, American (Traditional), Sandwiches, and Pizza. This time, we only removed these categories from our category list without filtering down the businesses. We repeated this one more time and in the 3rd round, the top 5 categories were: Local Services, Fashion, Home & Garden, Event Planning & Services and Breakfast & Brunch. Again we removed the categories without filtering out the businesses. The results of this cleaning are stored in our data file business.csv.

Reviews.json

Moving on to the reviews.json file, we first filtered the file based on the businesses left in business.csv since the reviews on those businesses are the only ones relevant to us. Next, since we want to match users based on what they like the best, we eliminated the reviews where the ratings are 3 stars and below since lower reviews tend to indicate less interest. Therefore, for the scope of this project, only the reviews with rating of 4 stars and 5 stars were used. The results of this cleaning are stored in our data file reviews.csv.

User.json

For our last file, we performed an inner join on user_id with the cleaned reviews.json file so that we only keep the users that wrote reviews in the reviews.csv. We also wanted to take into account the possibility of non-active users so we filtered to only keep users who have written more than or equal to 50 reviews. The results of this cleaning are stored in our data file users.csv.

For each method, we organized and filtered the data a bit differently, but below is an example of a merged data table of business.csv, reviews.csv and users.csv with the key columns used in our analysis.

Column name	Explanation	Example
review_id	Unique review ID	UaxdF7QaeKpHFTwGYB16WA
business_id	The unique ID of the business the review is about	hyFzDuyOWNG2rg5GYJ2wiQ
user_id	The unique ID of the user who wrote the review	5DLeXZ3ghivbgacb7OVW2g
stars	The numerical star rating the user gave the business	5.0
business_name	The name of the business	Alpen Rose
categories	An array of the categories the business falls under	Steakhouses, Seafood, Cocktail Bars
city	The city the business is located	Philadelphia
state	The state the business is located	PA
friends	An array of the user's friends as user IDs	8V0Med2klBrs3_oLHRet7Q, Opnlzyx_jZWqpicZ0ugshA
user_name	The user's first name	April
review_count	The number of reviews the user has written	420

Methodology

A simple overview of our recommendation system is that it will take in text input of an existing user ID and output 5 people to meetup with. To create this, we will try out a couple different approaches which are outlined below.

Matrix-Based approach

As a baseline, we first explored similarities between users using principal component analysis and clustering. Since we also have text data for the reviews, we also tried expanding on our approach by using natural language processing.

Similarity by category breakdown

For this method, we classified and clustered the user database based on the categories of businesses the users highly rated in reviews. Each step of the process is expanded upon below.

Data pre-processing

In addition to the data cleaning performed to the raw data, there were some extra filtering conditions we implemented to combat the size of the dataset. Since the analysis is at the level of the user, even after the initial cleaning, we still had 158,745 total users. Thus, we limited analysis to only users with more than 150 reviews which brought our total users down to 49,607.

To prepare for the creation of the matrix, we grouped the reviews by user_id and then merged the categories of all the businesses that user reviewed into one large list of categories. In the context of our analysis, this represented the interests of the users.

Build vectorized matrix

With the lists of interests of each user, we then proceeded to create a sparse matrix. Each row of the matrix represents a user and each column of this matrix represents a category. Initially we tried using the built in MultiLabelBinarizer function from the *sklearn* package to create this matrix, however this function doesn't take into account duplicates. Since each user's interest list is not super long, we wanted to weigh the categories the user showed interest in multiple times heavier than ones they only showed interest in one or two times. Therefore, we had to reference a similar function online to account for duplicates called *MultiLabelCounter*. In this function, for each category in the user's list of interests, the number of times that category appeared in the list is tallied up before being put inside the matrix. The dimensions of the final matrix was (49604, 3676).

Create user groupings

With the resulting sparse matrix, we first totaled up each category column and only kept the categories with more than 10 good reviews to narrow in on the more popular categories. This brought our total categories from 3676 to 2380. Then we performed PCA to get a reduced dimension version of the data. To determine the optimal number of dimensions to reduce to, we looked at the percent of variance explained and picked the number that explained more than

90%. For us, that number was 50 dimensions. Once we got the reduced data, we were able to proceed with k-means clustering.

To determine the ideal number of clustering, we used the elbow method and 2 clusters were recommended. However, in the scope of our project, since we want to use the clusters to identify users with similar interests, a larger number of clusters is ideal. Therefore we increased the number of clusters to 15.

Recommendation system:
To implement the recommendation system, we created two separate functions: common_interests and friend_recommender.

Friend_recommender takes in a user_id and determines which cluster that user falls under. It then iterates through all the users in that cluster and uses L2-norm calculation to determine how close each user is to the original user. With these distances, the top 5 closest users are taken as recommended users for the original user to meet up with. Common_interest is called by the friend_recommender function and is used to give common categories both users in a given pair are interested in. It takes in a pair of users and looks at those users' respective sparse matrix rows. Based on the category columns both users have values in, the function will tally up the total counts and then return the 3 categories with the highest count. Below are 3 separate case study examples of the system being used.

```
1 friend_recommender('--FQ61qe5wDSc6lDENlrUQ')
```

	Recommended User	Common Interests
0	oRafkFmvH7qp-z_rsFXdTg	[American (Traditional), Caribbean, Laotian]
1	tKE1-v8F1r0fgm5-vxi5ag	[American (Traditional), Caribbean, Laotian]
2	Q7JZ4mptLqMe0ZDSWtgjuw	[American (Traditional), Arts & Entertainment, Music Venues]
3	Q83R2UssCoErEJX36wrlvg	[American (Traditional), Arts & Entertainment, Music Venues]
4	nqUm_8yPU6a-KxojtREcww	[American (Traditional), Arts & Entertainment, Music Venues]

```
1 friend_recommender('zzhO1jW4skCDWNrWtSLbTw')
```

	Recommended User	Common Interests
0	dDTC6H2_yMHzEljVJOoDNA	[Chinese, Shanghainese]
1	-hAbdeB1C42iO93iyg-57Q	[Chinese]
2	LFSlJ3auEGvO97mBGXA9Xw	[Chinese]
3	rCgzn387hf4a2Wx-iAr5pg	[Chinese, Dim Sum, Shanghainese]
4	TqQtK-rfIHDhQ2dVMMVuhw	[Chinese, Coffee & Tea]

```
1 friend_recommender('zyV4wWbSQOJAVRZnMUslfQ')
```

	Recommended User	Common Interests
0	XC9nZtaQutqVuxabJppftQ	[American (New), Active Life, Golf]
1	NuMGh7LKV27I20MDAuVy2Q	[American (New), Active Life, Golf]
2	bY1svhjnaxLqoAuLSuij6g	[American (New), Hotels, Hotels & Travel]
3	GTzd901i1MFhuR8wbhgZdQ	[American (New), Active Life, Golf]
4	eF1Ku3VZtPqezbHjaGg1qg	[American (New), Italian, Steakhouses]

Similarity by keywords used in written reviews

Text preprocessing

For this part of our project, we need the dataset with all user information and their corresponding reviews across different businesses. We have the dataset filtered out with all the positive ratings; file named 'reviews' with features 'user_id', 'business_id', 'text(review)', 'useful' etc. We limited the data size by filtering out useful factor for reviews > 0.0 and include only users with review count >100. These steps draw down our entire data for this task to just 1000 users and around 150,000 data points (multiple rows/ reviews for a given user).

We further processed the user reviews using the following NLP functions in *nltk* module:

- Stop words removed using the predefined words dataset in English
- Text processed using *Lemmatizer* to remove text extensions
- Removed repetitive words in a given user review
- Parts of speech tagging (POS) to include only 'Nouns' and 'Adjectives'

Build vectorized matrix

We separated this task in two ways by characterizing the keywords containing food items and keywords from overall review.

Vectorizing Food items mentioned(bag of words):

Bag-of-words approach, as what it indicates in the name, assumes all the words are hold in bags where the order of words is ignored. We concatenated all the reviews given by each user in the dataset and each user's review set is parsed to filter only food names using the food dataset available in *nltk.corpus* module. We used count vectorizer in *sklearn.feature_extraction* to generate a matrix with users and frequency of food items used in their reviews. The final matrix of dimensions (985,576) was generated for this task.

Vectorizing keywords from entire review:

We count the frequency of each word wi occurring in one document dj. Then adopt ci,j to denote this value. A more advanced representation is called 2-gram, which partly overcomes the shortage of disregarding orders. A 2-gram term is a contiguous sequence of n items. i.e., for a sentence 'This is nice.', the collection of 2-gram terms is 'this is' and 'is nice'. This partition utilizes the context information and captures more textual structure from raw data. However, the cost is the increment of the number of features which is not good for prediction. With comparable advantages for both methods, we have implemented both frequency count and 2-gram and tested an analysis which works better in terms of prediction accuracy.

After simple counting, we convert the frequency into term frequency-inverse document frequency (TF-IDF) values. They are defined as follows:

$$\text{Term Frequency (TF)} = \frac{c_{i,j}}{\sum_i c_{i,j}}$$

$$\text{Inverse Document Frequency (IDF)} = \log(N/n)$$

Here, N is the total number of documents, and n is the number of documents(words in a review) a term t has appeared. TF-IDF value is the product of TF and IDF. TF highlights the words frequently appearing in one document, and IDF can mark the terms that are distinct. Thus, this value TF- IDF cooperates the inter and intra information and is widely used in the natural language process.

Term frequency-Inverse document frequency (Tf-Idf) vectorizer from *sklearn.feature_extraction* module was used to analyze the less frequent text in each user review set. While this technique is mainly used to remove the high frequency english words in the document, it actually killed the purpose of our task by removing almost all important words and retained a lot of garbage words. We decided not to use this method for our project.

Create user groupings

With the resulting sparse matrix, we normalized the data and performed PCA to get a reduced dimension version of the data (n_components=3). To determine the optimal number of dimensions to reduce to, we looked at the percent of variance explained and picked the number that explained more than 90%. Once we got the reduced data, we were able to proceed with k-means clustering. Finally, to determine the ideal number of clusters, we used the elbow method and 15 clusters were recommended.

Recommendation system:

To implement the recommendation system, we created two separate functions: common_interests and friend_recommender just like the above case but the similarity scores are based on common food interests.

Friend recommendation takes in a user_id and determines which cluster that user falls in and iterates through all the users in that cluster. We used L2-norm calculation to determine how close each user is to the given user and the top 5 closest users are suggested based on their food interests. Common interest function call takes in a pair of users and looks at those users' food notes and returns the top 3 items. Below are 2 separate case study examples of the system being used.

```
1 friend_recommender('Va1VeFo3so8d444ohkSGjg')
```

	Recommended User	Food Interests
0	hKBQ-PFIcB-t5FK3HUxoyQ	[side, chicken, pork]
1	e5hwGgorUG6KX-eSO_UCcA	[side, chicken, cheese]
2	L8Euta_K7ZDA5iwBHDyhbA	[side, chicken, pork]
3	ET3TgSSQ3shm2mj5vZ8IVQ	[chicken, side, cheese]
4	2RFyJnbJ5AHT9hQp_tMeYw	[side, cheese, chicken]

```
1 friend_recommender('ojyWQJtQ_ESVDvqL-69tLA')
```

	Recommended User	Food Interests
0	eg0cwrodeKGLeLIDTSCXoA	[cheese, side, veggie]
1	rfFCDTjJMN7ChYcCKkcHag	[veggie, cheese, bread]
2	Pk-dihlvXRaVfE65ghVOLA	[veggie, chocolate, potato]
3	2LfozZiejXfQHcS52-Fv5w	[veggie, veg, cheese]
4	yU4MW48QQ1eAdOkztAVf3g	[side, cheese, veggie]

Graph Based Recommendation System

This method models the relationships between users and business using graph theory. For this method to be applicable to our project, the following steps were taken:

Build the graph

The ".Graph()" function from the *networkx* package was used to build our graph. Each node represents a user or a business. An edge was created if the user has reviewed the business. In our case, based on what we have previously cleaned for the data, only the businesses that the user has rated 4 stars or above would qualify for an edge. This way, we preserve the interest of each user. It also aligned with our strategy to recommend users with similar interests.

Identify the clusters

A common approach for clustering in recommendation systems is the community detection algorithm. For this project, we used the Louvain algorithm. The Louvain algorithm has been proven to perform better than other techniques in terms of speed and accuracy for community discovery in networks (Blondel et al., 2008). The scalability of the Louvain method, which enables it to handle very large networks with millions of nodes and edges, is one of its primary benefits (Traag et al., 2019). As a result, it can be especially beneficial in systems that have a modular structure. When the algorithm begins, each node belongs to its own community. After the initiation, each node then move in the community that maximize:

$$\Delta Q = \left[\frac{\sum_{\text{in}} + 2k_{i,\text{in}}}{2m} - \left(\frac{\sum_{\text{tot}} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{\text{in}}}{2m} - \left(\frac{\sum_{\text{tot}}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

After this, each newly constructed community will become a new node. The above step will be repeated until no further increase in modularity is achieved. For the scope of our project, we used the "community.best_partition()" function from the *networkx* and *community* package. It detects the community of each node and assigns the community number of the nodes. Therefore, it assigns our users and business into different communities. Since our recommendation would identify users within the same community, the community information will be used in the following stage for similarity computation.

Compute Node similarity

After determining the communities, we moved to the calculation of the similarity between our target user and the users within its community. First of all, we identified our target users. This user has to be within our user_id list from the user file. After that, we locate the community this user belongs to. After the user community has been determined, we extracted all the nodes that are users and removed the nodes that are business. This is due to the fact that for the scope of our project, we are only identifying the similarity between users but not the businesses. The businesses were utilized to connect the users to the communities, but not used for calculating similarity. Next, we calculated the similarity between each node. The similarity method that was

selected in this project is the Jaccard Similarity. The Jaccard Index can be defined as the following:

$$J(A,B) = |A \cap B| / |A \cup B|$$

Essentially, it calculates the size of the interactions of the two sets and divides it by the size of their union. In our project, for each of the users within the target user community, we calculated its neighbors by using the "*.neighbor()*" function. After that we calculated the length of the target user's neighbor interacting with the other user and divided by the length of the target user's neighbor unioning the other user. The result becomes the similarity score between our target user and the other users within the community. For each of the users within the target user's community, we repeat the previous steps and calculate their similarity scores with the target user. All the similarity scores were calculated and saved as a dictionary with keys being the user ID and the values being the similarity scores.

Recommend User

After getting the similarity scores dictionary, we then sorted the keys (i.e. user_id) based on the similarity score descendingly and returned the first five users as our recommendation for friends.

Evaluation and Final Results

Both methods have successfully determined 5 users as recommendations to any given target user. Friend recommendation systems are often subjective and hard to quantify accuracy. Evaluation can be done through qualitative approaches such as user survey and questionnaire to investigate areas of improvement.

In the meanwhile, we noticed a number of areas that can be improved.

Lack of food specific datasets

There are hardly any predefined corpus datasets of common food categories, ingredients, taste, diverse cuisine specific words, etc. This limitation had an impact on how we parsed our user reviews for key words and in turn on our user friend recommendations.

Run Time

For the Graph based method, the run time is relatively high when using the entire dataset. We did not want to scarify accuracy by reducing the size of the original data set. Therefore, run time might take up to 10 min to return the recommended user. User experiences can be further improved if there is a way to improve the run time of this method.

Recommendation Diversity

The purpose of this system is to connect people with similar interests in business. We successfully assisted users in finding other users who shared their interests. If we can make

suggestions for them to meet by recommending the business, the user experience will be further enhanced. By doing this, we can help people connect while simultaneously boosting traffic to businesses and enhancing their profitability. The filtering on key words and community detection for businesses in our present code have given us a basis. From this perspective, additional research can be conducted to enhance the functionality of our system.