Travel Recommendation System

Technical Report

Group Members:

- Shahmir Yousaf (21L - 5259)
- Muhammad Asim (21L - 5240)
- Muhammad Taha (21L - 5289)
- Murtaza Ahmad (21L - 5173)

Introduction

The traditional methods of travel advice, like friends' recommendations and generic online sources, fall short in meeting diverse traveller preferences. Recognizing this, our initiative advocates for sophisticated, personalized approaches to travel recommendations. We emphasize the limitations of conventional methods, which often overwhelm users with vast, generic information. Our project aims to leverage user interactions and data to create a smarter recommendation system that evolves with individual preferences over time. By prompting users to express their preferences, we seek to tailor recommendations to their unique tastes and needs, enhancing their travel experiences. We aspire to move beyond the one-size-fits-all model, catering to travellers' increasing demand for unique and authentic experiences. With a forward-thinking approach, we aim to revolutionize the travel guidance arena by providing dynamic and personalized solutions that align with modern travellers' expectations.

Problem Statement

It is a common practice to seek recommendations when traveling to new places. We often turn to friends who have visited or lived there for advice on hotels, restaurants, and tourist attractions. Another approach is to gather information from online platforms like Yelp where reviews and recommendations about local business are posted by anonymous users. The second way is more effective, which enables us to find businesses that align with our tastes. This is the driving force behind our project: Building an intelligent travel recommendation system. The primary objective of this personalized recommendation system is to deliver precise and pertinent recommendations customized for each user. We will gather user preferences by allowing them to indicate their likes and dislikes during their initial interaction with the system. As users continue to use the system, we will do the recommendations based on their past behavior and interests. In this way, hopefully this recommendation application can improve users' traveling experience.

Data Collection

We have used the Yelp dataset obtained from kaggle. This dataset is a subset of Yelp's businesses, reviews, and user data. It was originally put together for the Yelp Dataset Challenge which is a chance for students to conduct research or analysis on Yelp's data and share their discoveries.

Key elements

- **Businesses** – This dataset contains a wide variety of establishments in many categories and regions. This dataset has 150k records.
- **Reviews** – It has information about reviews, ratings, check-ins and tips which were posted by individuals. This dataset has 6M records.

- **Users** – It contains all the information about the individual users. This dataset contains 1.9M records.

Exploratory Data Analysis

Data Understanding and Cleaning

Our original dataset consists of three files, yelp_users, consisting of user-related information, such as unique user ID, user name, the time the user signed up for Yelp, the number of reviews they have posted, the tags used frequently in the reviews, and details about their network. Figure 1 displays the screenshot of the first 5 lines of the user dataset. The file 'yelp_reviews', including columns such as 'review_id', 'business_id', 'user_id', 'text', and 'stars', basically shares the information about who posted the specific review for which business at what time yelp_business and what's the content and rate of the review. Figure 2 shows the samples from the review dataset. The last file we used is yelp_business with 150346 entries and 14 columns, illustrating exhaustive attributes of each business. The project aims to build a travel recommendation system, most of the data preprocessing focuses on the business file as we require the detailed attributes of the business to understand the users' preferences and make recommendations. Figure 3 shows the screenshot of the first 5 lines of the business dataset.

Figure 1

Samples from User Dataset

```
df_users.head(5)
```

	Unnamed: 0	user_id	name	review_count	yelping_since	useful	funny	cool	
0	0	qVc8ODYU5SZjKXVBgXdI7w	Walker	585	2007-01-25 16:47:26	7217	1259	5994	
1	1	j14WgRoU_-2ZE1aw1dXrJg	Daniel	4333	2009-01-25 04:35:42	43091	13066	27281	2009,2010,2011,2012,2013,2014,2015,20
2	2	2WnXYQFK0hXEoTxPtV2zvg	Steph	665	2008-07-25 10:41:00	2086	1010	1003	2009,2010,201
3	3	SZDeASXq7o05mMNLshsdIA	Gwen	224	2005-11-29 04:38:33	512	330	299	200
4	4	hA5lMy-EnncsH4JoR-hFGQ	Karen	79	2007-01-05 19:40:59	29	15	7	

Figure 2

Samples from Review Dataset

review_id	user_id	business_id	stars	useful	funny	cool	text	date
KU_O5udG6zpxOg-VcAEodg	mh_-eMZ6K5RLWhZyISBhwA	XQfwVwDr-v0ZS3_CbbE5Xw	3.0	0.0	0.0	0.0	If you decide to eat here, just be aware it is...	2018-07-07 22:09:11
BiTunyQ73aT9WBnpR9DZGw	OyoGAe7OKpv6SyGZT5g77Q	7ATYjTIgM3jUlt4UM3IypQ	5.0	1.0	0.0	1.0	I've taken a lot of spin classes over the year...	2012-01-03 15:28:18
saUsX_uimxRlCVr67Z4Jig	8g_iMtfSiwikVnbP2etR0A	YjUWPpI6HXG530lwP-fb2A	3.0	0.0	0.0	0.0	Family diner. Had the buffet. Eclectic assortm...	2014-02-05 20:30:30
AqPFMleE6RsU23_auESxiA	_7bHUi9Uuf5__HHc_Q8guQ	kxX2SOes4o-D3ZQBkiMRfA	5.0	1.0	0.0	1.0	Wow! Yummy, different, delicious. Our favo...	2015-01-04 00:01:03
Sx8TMOWLNuJBWer-0pcmoA	bcjbaE6dDog4jkNY91ncLQ	e4Vwtrqf-wpJfwesgvdgxQ	4.0	1.0	0.0	1.0	Cute interior and owner (?) gave us tour of up...	2017-01-14 20:54:15

Figure 3

Samples from Business Dataset

business_id	name	address	city	state	postal_code	latitude	longitude	stars	review_count	is_open	attributes
eNsfO8kk83dixA6A	Abby Rappoport, LAC, CMQ	1616 Chapala St, Ste 2	Santa Barbara	CA	93101	34.426679	-119.711197	5.0	7	0	{'ByAppointmentOnly': 'True'}
jTdTEA3yCZrAYPw	The UPS Store	87 Grasso Plaza Shopping Center	Affton	MO	63123	38.551126	-90.335695	3.0	15	1	{'BusinessAcceptsCreditCards': 'True'}
KiKi_TAnsVWINQQ	Target	5255 E Broadway Blvd	Tucson	AZ	85711	32.223236	-110.880452	3.5	22	0	{'BikeParking': 'True', 'BusinessAcceptsCredit...
Qd7CbVtyjqoe9mw	St Honore Pastries	935 Race St	Philadelphia	PA	19107	39.955505	-75.155564	4.0	80	1	{'RestaurantsDelivery': 'False', 'OutdoorSeati...
dE0EUBKlGXDVfA	Perkiomen Valley Brewery	101 Walnut St	Green Lane	PA	18054	40.338183	-75.471659	4.5	13	1	{'BusinessAcceptsCreditCards': 'True', 'Wheelc...

After a brief look at the business dataset, we got a basic idea of how many features the dataset has and what they stand for. Then the column ‘is_open’ catches our attention, like what the column name suggests, this column indicates whether the specific business is still open or already closed. Closed businesses should be excluded from the recommendation candidate list. So we only keep the businesses where the value of the column 'is_open' is true. Then we quickly examined the type of each column and if there were any missing values. We observed that the dataset is wellrecorded with few missing values, details are shown in Figure 4. However, the types of columns are various, including float, int, and object, which are not suitable for subsequent business similarity computations or model implementations.

Figure 4.

Summary About the Business Dataset

```
df_business.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150346 entries, 0 to 150345
Data columns (total 14 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   business_id   150346 non-null  object
 1   name          150346 non-null  object
 2   address       150346 non-null  object
 3   city          150346 non-null  object
 4   state         150346 non-null  object
 5   postal_code   150346 non-null  object
 6   latitude      150346 non-null  float64
 7   longitude     150346 non-null  float64
 8   stars         150346 non-null  float64
 9   review_count  150346 non-null  int64
 10  is_open       150346 non-null  int64
 11  attributes    136602 non-null  object
 12  categories    150243 non-null  object
 13  hours         127123 non-null  object
dtypes: float64(3), int64(2), object(9)
memory usage: 16.1+ MB
```

To standardize the column type for subsequent modeling, the following steps are implemented to engineer the features. It appears that the 'attributes' column contains dictionarylike values, each representing various features related to businesses, so we extract each distinct feature and put them in the pre-created one-hot encoded columns. Additionally, the 'categories' column denotes the various categories to which a business belongs. A single business may be associated with multiple categories, such as 'Restaurants,' 'Food,' and 'NightLife.' In order to facilitate analysis, we separate these categories into distinct columns. Figure 5 and Figure 6 show the value of the multi-valued columns before and after the transformation.

Figure 5

Original Value of the Multi-Valued Columns

categories	attributes	is_open
Doctors, Traditional Chinese Medicine, Naturop...	{'ByAppointmentOnly': 'True'}	0
Shipping Centers, Local Services, Notaries, Ma...	{'BusinessAcceptsCreditCards': 'True'}	1
Department Stores, Shopping, Fashion, Home & G...	{'BikeParking': 'True', 'BusinessAcceptsCredit...	0
Restaurants, Food, Bubble Tea, Coffee & Tea, B...	{'RestaurantsDelivery': 'False', 'OutdoorSeati...	1
Brewpubs, Breweries, Food	{'BusinessAcceptsCreditCards': 'True', 'Wheelc...	1

Figure 6

Sample of One-Hot Encoded Columns

	is_open	ByAppointmentOnly	BusinessAcceptsCreditCards	BikeParking	CoatCheck	RestaurantsTakeOut	RestaurantsDelivery	Caters	WiFi
0	1	0	1	0	0	0	0	0	0
1	1	0	0	1	0	1	0	1	1
2	1	0	1	1	0	1	0	0	0
3	1	0	1	0	0	1	1	0	0
4	1	0	1	1	0	0	0	0	0

Data Exploring and Visualization

As a beginning, we visualize the distribution of business categories through a bar chart. The graph shown as Figure 7, providing us the scope of the travel recommendation system. It reveals that the prominent categories include restaurants, shopping places, beauty & spa facilities, and health and medical care. This observation underscores the dataset's comprehensiveness, confirming its appropriateness for the recommendation system in a foreign country.

Figure 7

Yelp Business Category Distribution

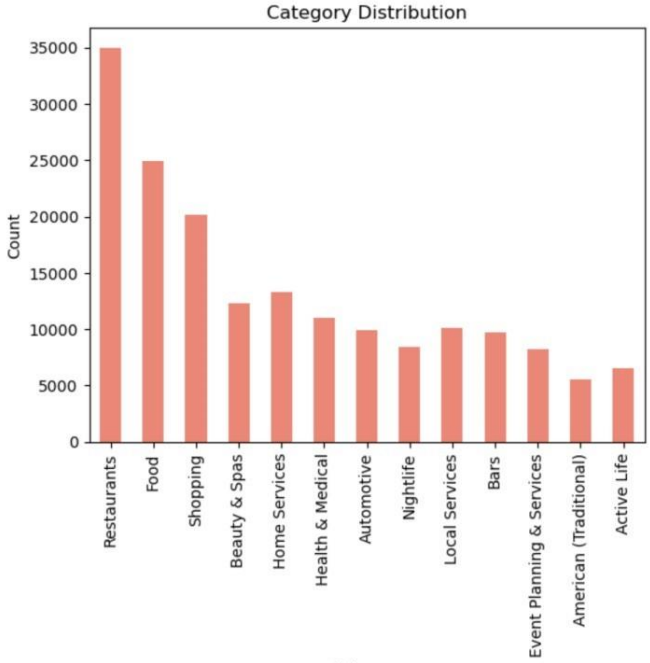
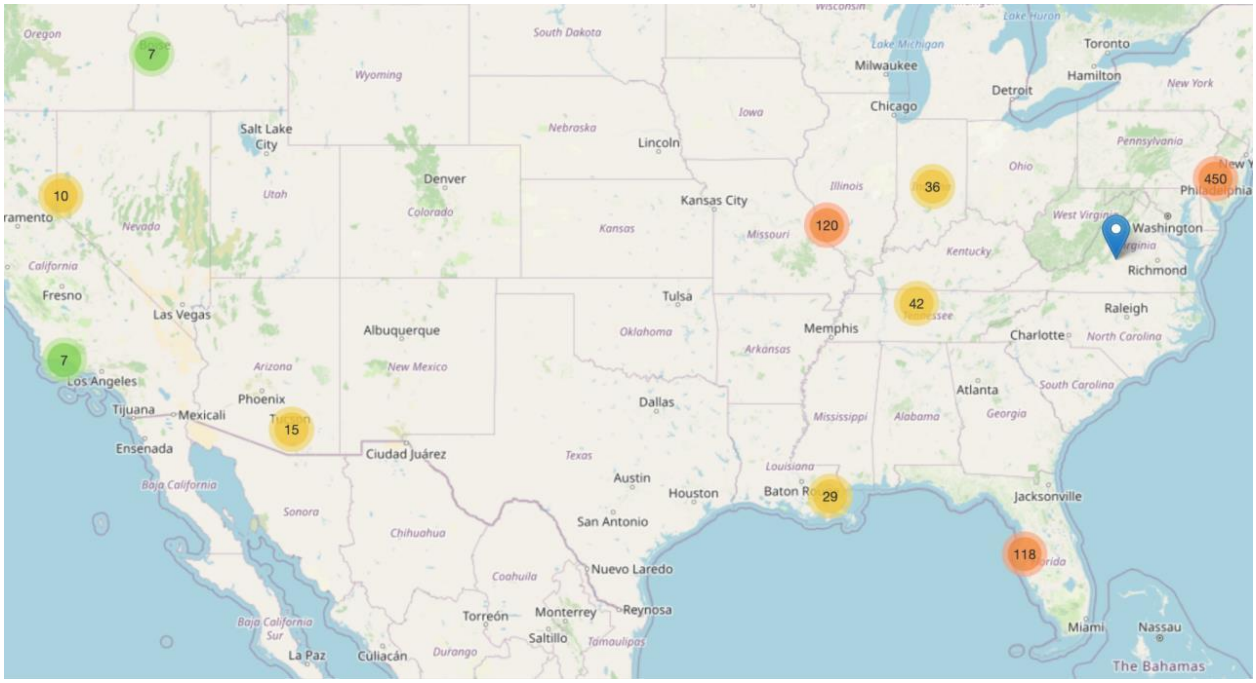


Figure 8 illustrates a map depicting the worldwide distribution of businesses on Yelp. Yelp's businesses are found all around the world, but there might be some countries without records. These visualizations help us see where to focus for travel recommendations and show us areas where we need further dataset to complete the global puzzle.

Figure 8

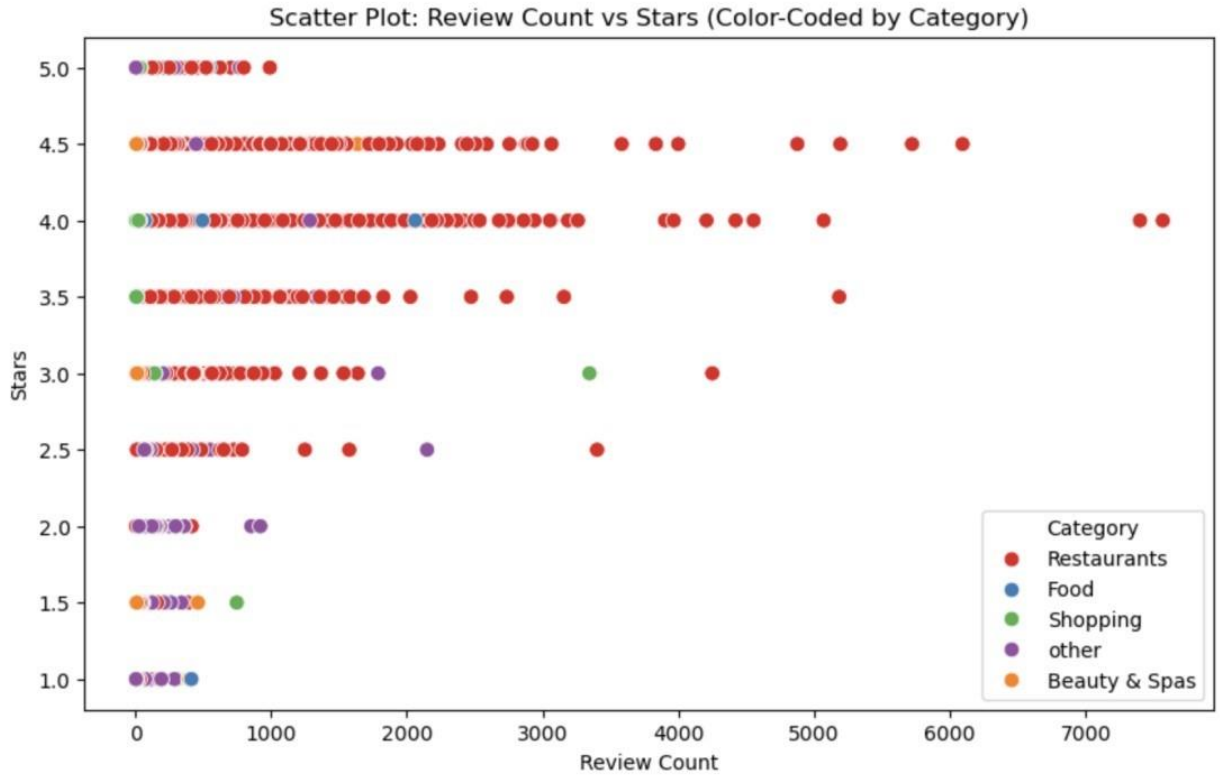
Worldwide Distribution of Yelp Businesses



By intuition, recommendations are highly related to the business's ratings, and reviews also play an important role in influencing customer preferences. Figure 9 illustrates this by showing a scatter plot of business ratings against the number of reviews, color-coded by business categories. The business categories are color-coded. Generally, we found that as the review counts rise, the star ratings tend to increase until reaching a balance point. After this point, the ratings level off, typically falling within the 4 to 4.5-star range.

Figure 9

Review Counts Against Stars Scatter Plot



Once we completed the data preprocessing process, we now have a comprehensive understanding of the dataset. Also, the data is well-prepared for later algorithm application and model implementation.

Methodology

In developing our recommendation system, we have merged the Business, User and Review datasets for this process. Two methodologies are employed: content-based filtering and collaborative filtering. For content-based filtering, we utilized TF-IDF (Term Frequency-Inverse Document Frequency) and KNN (K-Nearest Neighbors). In the case of collaborative filtering, we employed SVD (Singular Value Decomposition) and ALS (Alternating Least Squares). During the evaluation phase, we assessed the performance using MSE (Mean Squared Error), RMSE (Root Mean Squared Error), and cosine similarity.

Content based filtering

Content Filtering takes a unique approach of predicting user interests by doing analysis on the characteristics of items and the interactions users have had with those items previously. Basically, this approach does not actually focus on collecting preferences from multiple users but focuses on the attributes, features, or content of the items themselves. There are two approaches in this:

- **Item-Based Content Filtering:**

  Here the system uses items feature that a user has engaged with in the past. It makes a matrix called as item-item similarity matrix which maps the relationships between different items based on their attributes. Then, recommendations are done by identifying items with features similar to those the user that they have shown interest in past.

- **User-Based Content Filtering:**

  Here, the main point is the user profiles creation based on the content items features users have liked, rated, or interacted with. The system then analyzes the content characteristics associated with a user's past interactions and recommends items that a with the user's profile, emphasizing content similarities. This approach is advantageous when users' preferences are shaped by the specific content features of items rather than the collaborative patterns with other users.

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF when done using cosine similarity is a great combination for text analysis and information retrieval. TF-IDF main job is to get important terms within documents and creating numerical vectors which is text data.

- **Cosine similarity**

  Cosine similarity is nothing but a similarity matrix that calculates the cosine angle between two vectors. This gives measure of similarity and doesn't actually consider vector length.

  Basically, in TF-IDF vectors higher cosine similarity the better as it shows greater textual resemblance between documents. This approach is considered to be the best choice for tasks such as document clustering, content recommendation, and also information retrieval. One more point to note is that, TF-IDF with cosine similarity increases the precision of text-based applications by efficiently first by identifying and secondly by ranking documents based on their semantic content and thematic relevance.

Figure 10

```
content_data = df[['business_id', 'name', 'categories', 'attributes', 'stars', 'review_count']].copy()

# Set type to string
content_data['categories'] = content_data['categories'].astype(str)
content_data['attributes'] = content_data['attributes'].astype(str)

# handle missing values
content_data['categories'].fillna('', inplace=True)
content_data['attributes'].fillna('', inplace=True)

# make a new column called 'content'
content_data['content'] = content_data['name'] + ' ' + content_data['categories'] + ' ' + content_data['attributes']
```

```
# TF-IDF to convert text into numerical vectors
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(content_data['content'])

# Calculate cosine similarity
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In our Python based implementation of content based recommendation system for restaurants we select relevant columns such as business ID, name, categories, attributes, stars, and review count from our dataset. The textual based features in our dataset like categories, attributes, and business name are fixed for missing values. A new column called 'content' is created by combining these different text features which we apply the TF-IDF algorithm on.

In our content based recommendation system, we have used TF-IDF Vectorizer to convert the text into numerical vectors. This will allow us to use cosine similarity between different restaurants. TF-IDF vectorization helps to transform the textual content into numerical vectors, capturing the importance of word. Subsequently, cosine similarity is calculated using a linear kernel, generating a matrix reflecting the pairwise similarities between restaurents. Next, we calculate the cosine similarity using a linear kernel which is derived from the sklearn package in Python which generates a matrix reflecting the pairwise similarities between businesses.

Figure 11

```
In [36]: def get_content_recommendations(name, cosine_sim=cosine_sim, content_data=content_data, threshold=0.1):

             # Get the index of the business with the given name
             idx = content_data.index[content_data['name'] == name].tolist()

             # Check if the list is not empty
             if not idx:
                 print(f"No business found with name '{name}'")
                 return []

             idx = idx[0]

             sim_scores = list(enumerate(cosine_sim[idx]))

             # Filter out items with similarity below the threshold
             sim_scores = [(i, score) for i, score in sim_scores if score > threshold]

             sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

             # Extract the top 10 similar businesses (excluding itself)
             top_similar_businesses = sim_scores[1:11]

             # Get the indices and names of the top similar businesses
             similar_indices = [i[0] for i in top_similar_businesses]
             similar_business_info = content_data.iloc[similar_indices][['name', 'review_count', 'stars']]
             return similar_business_info
```

Next, we planned to implement a function called as get_content_recommendation where we take a restaurant name as input with a cosine similarity matrix as well as content data and a threshold for filtering out items with low similarity score. This function then finds the index and only filters out those businesses with similarity scores above a specific threshold. The top 10 similar businesses are then returned to us in output with their review count and average star rating.

Figure 12

```
In [37]: # Example: Get content-based recommendations for a specific business and threshold
         business_name = "Pho Voorhees"
         content_recommendations = get_content_recommendations(business_name, threshold=0.5)

         # Print the content-based recommendations
         print(f"Content-Based Recommendations for {business_name}:")
         print(content_recommendations)
```

```
Content-Based Recommendations for Pho Voorhees:
                         name  review_count  stars
13071              Pho Street           150    4.0
555                Pho Bistro           184    3.0
4339                Asian Pho           194    3.5
5587                 Pho King            58    4.0
2933              Phamous Cafe           220    4.0
10565                   Pho 79            42    3.0
14017             Bubble Shack            97    4.5
4927   Pho An Hoa Restaurant           160    4.0
308               Pho Friendly            93    4.5
258               Express Cafe           119    4.5
```

The above example shows us getting recommendations similar to a business named as “Pho Voorhees”. We can see in the output in the above example that we are being recommended similar restaurants to the input that we have provided. We also get a count of reviews for each restaurant as well as their average rating.

Advantages

- Extremely efficient to calculate cosine similarity is computationally efficient as it involves only the dot product of the vectors.
- Cosine similarity is particularly effective for text data, where the presence or absence of words is crucial. It measures the similarity of document vectors based on the angle between them.
- The recommendations made using similarity metrics like cosine similarity, jaccard similarity and are easy to interpret and understand. This helps to explain why a certain item is recommended based on the features present in the content.
- It is easy to implement and build a recommendation engine using content based filtering and use it along with other metrics to give the user a holistic recommendation experience. ***Disadvantages***
- Content-based filtering may always recommend similar items/products to the user which might not allow the user to explore new and different items, limiting their exposure and sometimes might be detrimental to any organization’s expansion plans.
- It only gives values between 0 to 1 as it is a cosine of the angle between vectors and as we know this cannot be negative.
- It has a problem when it has to work with multilingual texts. It treats each language differently which is actually a problem as differences in language structure can impact similarity scores.
- Last but not the least, for binary term occurrence vectors cosine similarity might not capture everything as it doesn't account for the term occurrences frequency.

• **Jaccard similarity**

Just like cosine similarity, Jaccard similarity also measures between two instances, introduced by Paul Jaccard in 1901. The similarity score is calculated by the overlap of features shared by two instances, divided by the total features present in both instances. In our case, let's consider 'Food' recommendations, where, for a specific user, we have a list of foods the user likes and a long list of candidate foods that the user may or may not show interest in. We then select one item from each of these two lists respectively and calculate the scores.

Comparing these scores, a higher score indicates a greater similarity between the two instances. Now that we have introduced two similarity metrics, the table below compares these two metrics based on their pros and cons.

Table 1

	Cosine Similarity	**Jaccard Similarity**
Affected by Magnitude	Considers both direction and magnitude.	Doesn't take frequency into consideration, only focuses on specific sets intersection.
Suitable Data Types	Suitable for real-valued data and vectors.	Works well with binary data and sets.
Performance in HighDimensional Space	Generally, performs well in high-dimensional dataset.	Easy to perform poorly in high dimensions due to sparsity.
Interpretability	The angle between two vectors, more abstract.	Intuitive Interpretation, measuring the overlap between sets.

KNN

With the features of various products extracted and quantified, we have chosen to implement the K-Nearest Neighbors (KNN) algorithm to identify products like those specified by user input. K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm that is exceptionally effective for both classification and regression problems. Its simplicity lies in its use of the surrounding neighbors of a query point to predict the output, rather than a learnable function across the entire space. KNN functions under the premise that similar things exist in proximity; in other words, it assumes that the data points that are nearby to one another are similar. This is particularly useful in recommendation systems, where the goal is to find items that are like a user's interests. This closeness is typically determined using distance metrics like Euclidean, Manhattan, or Hamming distance.

For a recommendation system, the workflow leveraging KNN can be as follows: (1) The user provides details about items they are interested in, such as restaurants or stores. These

preferences can be explicit, like a favorite cuisine, or implicit, derived from their browsing or purchase history. (2) Using the chosen distance metric, the algorithm searches for the 'K' items in the dataset that are closest to the user's preferences. These items are then sorted by their distance, with the nearest being the most similar and, presumably, the most relevant. (3) Based on the initial set of recommended items, the system can iteratively refine the suggestions. For instance, if the user interacts positively with a recommended item (like visiting a recommended restaurant), the system can then find new 'neighbors' around this item to suggest in the next iteration. This process can continue, creating a dynamic and personalized recommendation experience.

Figure 13

attributes	categories	hours	BusinessParking	Ambience	GoodForMeal	Dietary	Music	ByAppointmentOnly
{'ByAppointmentOnly': 'True'}	Doctors, Traditional Chinese Medicine, Naturop...	NaN	{}	{}	{}	{}	{}	True
{'BusinessAcceptsCreditCards': 'True'}	Shipping Centers, Local Services, Notaries, Ma...	{'Monday': '0:0-0:0', 'Tuesday': '8:0-18:30', ...	{}	{}	{}	{}	{}	{}
{'BikeParking': 'True', 'BusinessAcceptsCredit...	Department Stores, Shopping, Fashion, Home & G...	{'Monday': '8:0-22:0', 'Tuesday': '8:0-22:0', ...	{'garage': False, 'street': False, 'validated'...	{}	{}	{}	{}	False

This approach can be further enhanced by incorporating user feedback to adjust the 'K' value or the distance metric, ensuring the recommendations become increasingly precise over time. Moreover, feature weighting could be applied to give more importance to certain attributes that are deemed more significant in predicting user preferences. Additionally, to prevent the model from being overwhelmed by the curse of dimensionality, feature selection techniques may be employed to retain only the most relevant features. This ensures that the distance metrics used in KNN yield meaningful similarities between items. Ultimately, the success of the KNN-based recommendation system hinges on the quality of the features extracted and the appropriateness of the distance metric chosen, ensuring that the items recommended not only match the user's stated preferences but also resonate with their unstated needs and desires.

Advantages

- KNN is very straightforward and easy to understand, making the algorithm a good starting point for classification and regression tasks.
- Since KNN makes no assumptions about the underlying data distribution, it is suitable for non-linear data.
- KNN can use various types of distance metrics to find the nearest neighbors, making it flexible to work with different types of data.

Disadvantages

- KNN requires computing the distance from each query point to all training samples, which can be computationally intensive, especially with large datasets.
- The performance of KNN can be severely degraded if the features are not scaled uniformly because the distance metrics can be skewed towards higher magnitude features.

Collaborative filtering

Collaborative Filtering (CF) is a method used in recommender systems to predict user interests by collecting preferences from multiple users. It assumes that if one user agrees with another on one issue, they are likely to agree on others. It operates under the assumption that users who have shown similar preferences in the past will exhibit similar preferences in the future. Collaborative filtering doesn't require explicit information about items; instead, it relies on the user-item interactions or ratings.

Collaborative filtering systems can be broadly divided into two steps: finding users with similar rating patterns and using their ratings to calculate predictions for the active user. This is known as user-based collaborative filtering. Item-based collaborative filtering, on the other hand, builds an item-item matrix to infer user tastes and is more item-centric. We have implemented two approaches in collaborative filtering. Approach one is based on SVD (Singular Value Decomposition) model and approach two is by using ALS (Alternating Least Squares).

SVD - (Singular Value Decomposition)

Singular Value Decomposition is a matrix factorization technique commonly used in recommendation systems for reducing the dimensionality of the user-item rating matrix. It identifies latent factors representing users' preferences and items' characteristics by decomposing the matrix into three matrices: U for user-feature associations, Σ for singular values, and V for item-feature associations. This enables SVD to predict missing ratings and generate recommendations based on these latent factors.

We incorporated Singular Value Decomposition as a fundamental matrix factorization technique in our recommender system implementation. Utilizing the Surprise library in Python, we employed SVD to analyze a dataset containing user-item interactions, specifically focusing on user

ratings of businesses within the Yelp dataset. The merging of business and review information resulted in the creation of a consolidated dataset ('business_reviews.csv'), which was then loaded into the Surprise library for collaborative filtering tasks.

Incorporating SVD within the Surprise platform, we divided the dataset into training and testing sets. The SVD model was trained on known user-item interactions and then assessed for its predictive performance on unseen data. After merging business and review information in the preprocessing stage, we established a unified dataset ('business_reviews.csv'). Using the Surprise library, we loaded this dataset with 20% allocated for testing and 80% for training purposes. Using SVD in the Surprise platform, we divided the dataset into training and testing sets to train the SVD model on known user-item interactions and assess its predictive performance on new data. After merging business and review information to preprocess the dataset, a unified dataset ('business_reviews.csv') was created shown in Figure 14.

Figure 14

Loading Business, Review csv files and Merging them into new business_reviews.csv file

```
#Loading the Review CSV file into a DataFrame
df_review = pd.read_csv('/content/drive/Shareddrives/228 Travel Recommendation/Dataset/yelp_academic_dataset_review.

#Loading the Business CSV file into another DataFrame
df_business = pd.read_csv('/content/drive/Shareddrives/228 Travel Recommendation/Dataset/yelp_academic_dataset_busin

#Merging the two DataFrames based on the 'business_id' column
merged_df = pd.merge(df_review, df_business, on='business_id', how='inner')

#Saving the merged DataFrame to a new business_reviews CSV file
merged_df.to_csv('/content/drive/Shareddrives/228 Travel Recommendation/Dataset/business_reviews.csv', index=False)
```

```
#Loading the new CSV file into a Pandas DataFrame
file_path = '/content/drive/Shareddrives/228 Travel Recommendation/Dataset/business_reviews.csv'
data = pd.read_csv(file_path)
```

The SVD model that had been trained was then used to forecast missing ratings in the test set. This allowed us to assess its ability to predict user-item interactions that were not previously seen. The predictions relied on latent factors from the training phase, enabling an evaluation of the model's accuracy and effectiveness in predicting user ratings for businesses not included in the training data. Recommendations for a specific user involved identifying unrated businesses and predicting how they might be rated by that user.

Figure 15

Making Recommendations for a specific user

```
user_id = 'vI4vyi1dfG93oAiSRFDymA'
user_recommendations = []
```

```
data_loading = Dataset.load_from_df(data[['user_id', 'business_id', 'sta

#Fetching businesses that the user has not rated yet
rated_items = data_loading.df[data_loading.df['user_id'] == user_id]['bu
user_recommendations = []

for business_id in data_loading.df['business_id'].unique():
    if business_id not in rated_items.values:
        predicted_rating = model.predict(user_id, business_id).est
        user_recommendations.append((business_id, predicted_rating))

#Sorting the recommendations by predicted rating in descending order
user_recommendations.sort(key=lambda x: x[1], reverse=True)

#Displaying top 20 recommendations for the selected user
top_n = 20
print(f"Top {top_n} Recommendations for User {user_id}:")
for idx, (business_id, predicted_rating) in enumerate(user_recommendatio
    #Looking up 'name' from the 'data' DataFrame
    business_name = data[data['business_id'] == business_id]['name'].val
    print(f"{idx}. Business ID: {business_id}, Business Name: {business_
```

```
Top 20 Recommendations for User vI4vyi1dfG93oAiSRFDymA:
1. Business ID: NDwoKO79_T49UEKVDlHd3A, Business Name: Sustainable Wine
Tours, Predicted Rating: 4.9310282546320146
2. Business ID: B2Tuf5M1wQhdwAKnD-w7Yw, Business Name: New Orleans Airb
oat Tours, Predicted Rating: 4.928910449664647
3. Business ID: STEG37SqBC3PkwY4wgSoPg, Business Name: Taylor Home Solu
tions, Predicted Rating: 4.924875280294676
4. Business ID: QNilrbTi8912ye2ztnBMpA, Business Name: DeeTours of Sant
a Barbara, Predicted Rating: 4.92315855547764
5. Business ID: TDEV16C4GhK5wyhL-5V7ww, Business Name: Flambeaux Bicycl
e Tours, Predicted Rating: 4.922757893571927
6. Business ID: 0IjDqJexP6jTH4F_Kg4mrQ, Business Name: A New Twist Ball
oons and Face Painting, Predicted Rating: 4.922038445634181
7. Business ID: ez4kMLP6OJEIaMbMrrGRdA, Business Name: New Orleans Secr
ets Tours, Predicted Rating: 4.921074405323677
8. Business ID: im3hUe2nigm2Xm-Z1SNXIg, Business Name: B & B Heating an
d Air, Predicted Rating: 4.916877307732881
9. Business ID: NfKcglAqZ3eZMIepH1YwYw, Business Name: Matt Glynn - Sch
umacher Mortgage, Predicted Rating: 4.915183954572072
10. Business ID: 4-P4Bzqd01YvKX9tp7IGfQ, Business Name: Drink & Learn,
Predicted Rating: 4.913210407877951
11. Business ID: 1RqfozJoosHAsKZhc5PY7w, Business Name: Walls Jewelry R
epairing, Predicted Rating: 4.911830600191876
```

Evaluation of SVD

We have conducted an assessment of the model using Cosine similarity, which is a mathematical method employed to measure the similarity between two non-zero vectors. In our project recommendation system, we apply cosine similarity to evaluate the likeness or resemblance between vectors that represent predicted ratings and those representing actual ratings. By transforming both predicted and actual ratings into arrays (vectors) and considering each user's ratings as a vector in a multi-dimensional space (users × items), cosine similarity computes the cosine value of the angle between these vectors. This calculation measures how comparable the direction of the predicted ratings vector is to that of the actual ratings vector**.**

Figure 16

```
#doing predictive rating and actual ratings

predicted_ratings = [pred.est for pred in predictions]
actual_ratings = [pred.r_ui for pred in predictions]
```

```
#Calculating Cosine Similarity between precitive rating and actual ratin

from sklearn.metrics.pairwise import cosine_similarity

#Converting the lists to NumPy arrays
predicted_ratings_array = np.array([predicted_ratings])
actual_ratings_array = np.array([actual_ratings])

#Reshaping arrays
predicted_ratings_array = predicted_ratings_array.reshape(1, -1)
actual_ratings_array = actual_ratings_array.reshape(1, -1)

#Calculating cosine similarity
similarity_score = cosine_similarity(predicted_ratings_array, actual_rat
```

```
print(f"Cosine Similarity Score: {similarity_score[0, 0]}")
```

```
Cosine Similarity Score: 0.9476924933511831
```

The higher the cosine similarity score (closer to 1), the better the alignment between predicted ratings and actual ratings, indicating strong similarity in item ratings by the model compared to users' preferences. With an obtained cosine similarity score of about 0.95, there is an extremely close match or likeness in the vectors representing predicted and actual ratings. This suggests that recommendations generated by the model based on predicted ratings are highly consistent with user's actual ratings. Additionally, using cosine similarity as an evaluation metric has provided us with insights into how well the system captures user preferences, thereby confirming the accuracy and reliability of its predictions against actual user ratings.

Advantages

- The SVD method effectively reduces the dimensions of the user-item interaction matrix, allowing the system to efficiently manage large datasets while retaining important information about user-item connections.
- The model provides fairly precise forecasts for unavailable ratings by acquiring hidden features from observed interactions, enabling the production of trustworthy suggestions for unreviewed items.
- The Surprise library makes it easier to implement recommendation systems based on SVD, offering convenient features for data loading, model training, prediction, and assessment.

Disadvantages

- Scalability can be a concern due to the computational demands of SVD, especially when dealing with very large datasets. Our dataset, which contains almost seven million records, posed significant computational challenges and required considerable time for model training.

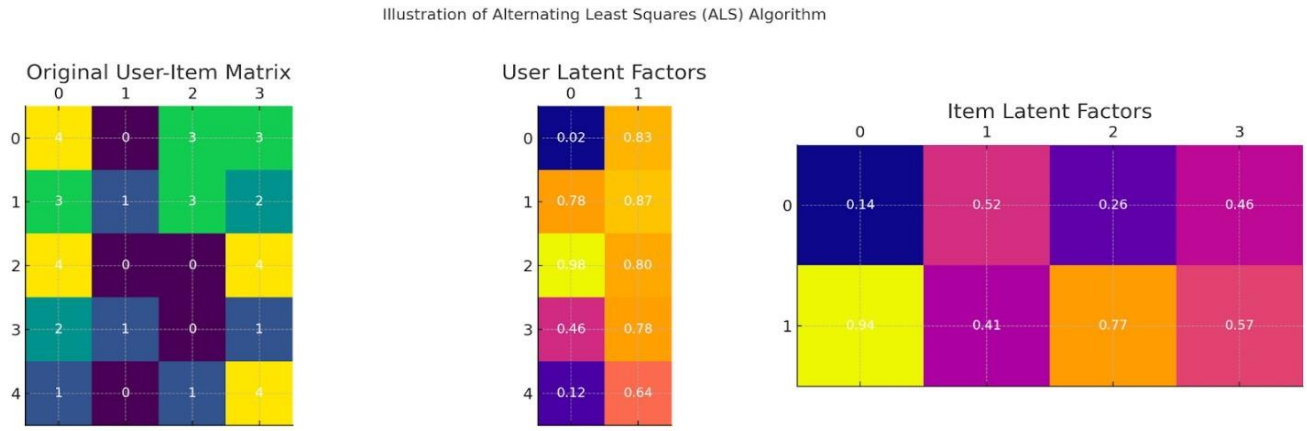
- SVD models may be susceptible to overfitting, particularly when working with smaller datasets, which could potentially affect their ability to generalize effectively to new data. Due to our extensive dataset, we have not encountered any issues of overfitting.

- SVD might face challenges when working with extremely sparse datasets, where users have rated only a small portion of the items available. This limitation can restrict the model's capacity to generalize and make precise forecasts.

ALS (Alternating Least square)

We apply the Alternating Least Squares (ALS) algorithm to leverage the user item ratings provided in the review table to infer the latent factors in the recommendation models. ALS is a cornerstone in the field of collaborative filtering, widely used for building recommendation systems. Part of Apache Spark's MLlib, ALS specializes in uncovering latent factors in user-item matrices, effectively predicting user preferences and item popularity. This algorithm operates by decomposing a large matrix of user-item interactions into two smaller matrices, one for users and one for items, whose product approximates the original matrix. The alternating aspect comes from the way the algorithm works: it keeps one of these matrices fixed and solves for the other, then alternates between them iteratively. This process minimizes the least squares problem, thereby optimizing the fit to the data. ALS's scalability and efficiency in handling large datasets make it particularly valuable for big data scenarios, where traditional matrix factorization methods might struggle. Its implementation in PySpark allows for easy integration into distributed systems, leveraging Spark's powerful data processing capabilities to handle complex, large-scale recommendation tasks.

Figure 17



To construct the user-item matrix that can be later leveraged by the ALS algorithm, we use the review table which contains the user id and product item and stars, to get the numerical user id and numeral product id, we need to join the review table with the user table and the business table. Then we use PySpark to extract the tuples and construct the input RDD as shown in the graph below.

Figure 18

```
[9]: # Merge review_user with business on business_id column
review_user = review.merge(user_reset, on='user_id', how='outer')
review_final = review_user.merge(business_reset, on='business_id', how='outer')
review_final.head()
```

[9]:

	user_id	business_id	stars	uid	name_x	pid	name_y
0	mh_-eMZ6K5RLWhZylSBhwA	XQfwVwDr-v0ZS3_CbbE5Xw	3.0	124234.0	Melanie	1867.0	Turning Point of North Wales
1	5EGs7LX3Z8ZogvOOgNLsnA	XQfwVwDr-v0ZS3_CbbE5Xw	2.0	121817.0	Phyllis	1867.0	Turning Point of North Wales
2	q3Kv3wFOwu1Rd2l6T_VWOQ	XQfwVwDr-v0ZS3_CbbE5Xw	5.0	89372.0	Gabby	1867.0	Turning Point of North Wales
3	wmqsehbFirZPlAluJUakeQ	XQfwVwDr-v0ZS3_CbbE5Xw	3.0	34438.0	Charles	1867.0	Turning Point of North Wales
4	NMLvjdY7lOdtfU0TepvUuA	XQfwVwDr-v0ZS3_CbbE5Xw	2.0	9488.0	Zach	1867.0	Turning Point of North Wales

The training code is extremely simple as the ALS algorithm is implemented inside the PySpark MLLib. There are a few parameters to tune, mostly the rank, lambda (regularization factor) and number of iterations. We tested with different configs and finally arrived at the optimal values.

Figure 19

```
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating

# Build the recommendation model using Alternating Least Squares
rank = 50
numIterations = 5
model = ALS.train(rdd, rank, numIterations, lambda_=0.1, nonnegative=True)
```

Evaluation of ALS

We conducted the evaluations of ALS based on three important metrics: (1) Mean Squared Error (MSE), (2) Root Mean Squared Error (RMSE), and (3) Cosine Similarity.

Figure 20

```
from sklearn.metrics import mean_squared_error, root_mean_squared_error
from sklearn.metrics.pairwise import cosine_similarity

print(mean_squared_error(gt, pred))
print(root_mean_squared_error(gt, pred))
print(cosine_similarity([gt, pred]))
```

Table 2

MSE	**RMSE**	**Cosine**
3.8602	1.9647	0.8839

Advantages

- ALS can efficiently handle large datasets, which is crucial for recommendation systems that need to process a vast number of user-item interactions.
- The algorithm lends itself well to parallel processing, making it a good fit for distributed systems like Apache Spark.

Disadvantages

- The performance of ALS is sensitive to the choice of hyperparameters, which can be challenging to optimize.

- ALS doesn't inherently consider contextual information (like time of day, location, etc.), which can be crucial for certain types of recommendations.

Comparison Of Evaluation Metrics

In this section, we present the evaluation results of the machine learning models. To measure the model performance for recommendation models, typical classification and regression metrics cannot be easily applied. Since we will need to understand the effectiveness as it compares with models that have different scores distributions.

First, we choose to use the MSE metric as well as the RMSE metric to measure the difference between the actual star's users gave with the predicted stars that we assign. Mean Squared Error (MSE) measures the average of the squares of the errors. MSE is a risk metric corresponding to the expected value of the squared (quadratic) error or loss. A lower MSE indicates a more accurate model in predicting quantitative data. It's particularly useful when you want to emphasize larger errors more than smaller ones since the errors are squared. RMSE is the root square of the MSE metric. It measures the standard deviation of the residual errors. Lower values of RMSE indicate better fit. RMSE is generally more interpretable in the same units as the response variable.

Figure 21

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$$\text{RMSE} = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

$$\text{Cosine Similarity} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Table 3

Algorithm	**MSE**	**RMSE**	**Cosine**
Collaborative (ALS)	3.8602	1.9647	0.8839

Content-Based (KNN)	2.2486	1.4995	0.9302
Content-Based (TF-IDF)	2.0885	1.4451	0.9344
Collaborative (SVD)	1.6546	1.2864	0.9476

From the results, we can see that ALS did not perform as well as the other three models as there are multiple reasons: (1) data sparsity: the training data is extremely sparse with millions of items but each user only have around ten reviews on average, (2) cold start: since each user only has a very small number of reviews, it would be easy to run into cold start problem after splitting the training and test dataset. KNN and TF-IDF are the two content-based approaches which the TF-IDF algorithm showed slight advantage since it was able to leverage more information and the features are more normalized to the dataset. Overall, SVD performed the best compared to the other three algorithms which is expected as it leverages not only the item side features but also the user-item relationships.

Summary

Our project integrates content-based and collaborative filtering methods, utilizing Yelp's extensive dataset to offer a detailed and personalized travel recommendation system. By combining the customization options of content-based recommendations with the accuracy of collaborative filtering, our system aims to revolutionize travel guidance by providing users with a sophisticated and tailored approach for discovering businesses that align perfectly with their individual preferences.