Yelp Restaurant Recommender System
Assignment 2 - Report

Abstract

Taking the user reviews and business datasets of Yelp, we map the reviews to their relevant restaurants and create our dataset. After visualizing the data with scatter plots, bar plots and pie charts to identify patterns, and taking in consideration the compute hours, we pre-process and clean the data to ensure its quality. We use Vader Sentiment Analysis to determine the sentiment of the review texts as positive, negative or neutral. Then using the sentiment and state feature we train a baseline latent factor model and Truncated SVD model to predict the star ratings of that review. Finally, we built a recommender system from the Truncated SVD model to recommend top 10 restaurants to a particular user.

Introduction

In today's digital age, platforms like Yelp have become very important for users seeking information about the good restaurants in their area which will provide dining and takeout services. At this point, there are billions of user reviews and ratings for restaurants present online, such that we as clients find it difficult to find the good restaurants in our area by spending a lot of time going over the reviews and ratings on platforms like Yelp. Hence, to solve these issues it is important to develop recommendation systems that make personalized recommendations to help the users find restaurants to their liking without having to spend much time sifting through many reviews.

In addition to building a recommendation system, we also predict star ratings based on sentiments extracted from the review text. This helps us in understanding the opinions of the users and their expectations. The use of machine learning models and sentiment analysis in platforms like Yelp, can close the gap between subjective customer reviews and objective star ratings, which can increase the reliability and trust of rating systems.

Objectives

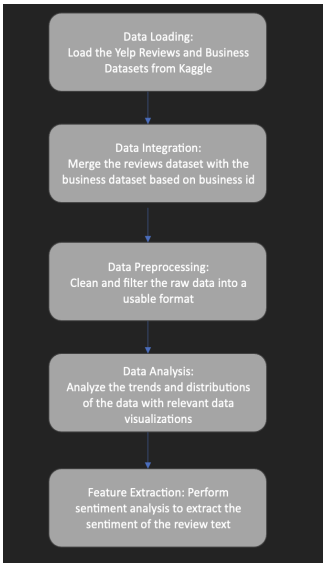
- **Exploratory Data Analysis:** Combine Yelp's user reviews and business datasets by mapping user reviews to corresponding restaurants. Visualize, preprocess, and clean the data to ensure data quality for training our models.
- **Sentiment Analysis:** Perform sentiment analysis on the many review texts found on Yelp using VADER Sentiment Analysis to classify sentiments as positive, negative, or neutral.
- **Rating Prediction:** Leverage the sentiment feature to train a baseline latent factor model to predict star ratings of the reviews. Improve on its performance, using a Truncated SVD model.
- **Develop Recommender System:** Use the Truncated SVD model to develop a recommendation system to suggest the top 10 restaurants to a user based on their preferences.

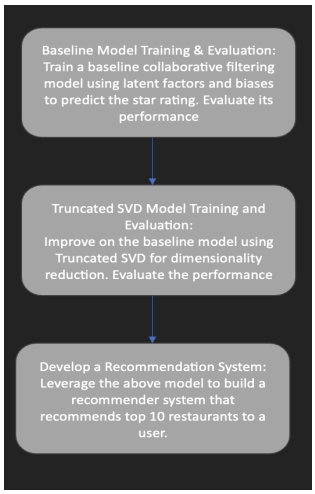
Dataset

The datasets are taken from Kaggle at https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset?select=yelp_academic_dataset_user.json
We have used the yelp_academic_dataset_review.json and yelp_academic_dataset_business.json for our project. Both these datasets are merged based on business_id to create our final dataset. Further data

preprocessing and cleaning is applied in Exploratory Data Analysis.

Proposed Methodology





Exploratory Data Analysis

1. **Load the Reviews Dataset**

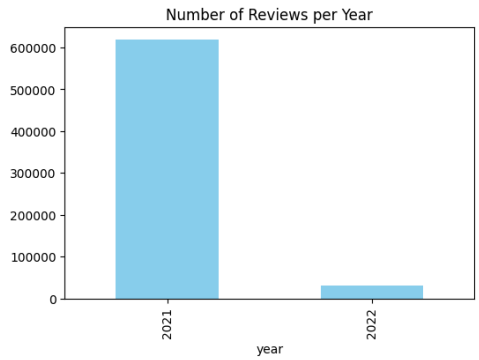
   Download the yelp_academic_dataset_review.json dataset from Kaggle and upload it on dropbox to then subsequently access it in a python environment due to it's very large size(5.34GB)

2. **Analyse the Reviews Dataset**

   The dataset has 8,681,443 rows and 9 columns, namely `'review_id'(object), 'user_id'(object), 'business_id'(object), 'stars'(float64), 'useful'(int64), 'funny'(int64), 'cool'(int64), 'text'(object), 'date'(object)`. There are no null values in any of the columns.

3. **Filter the Reviews Dataset**

   Filter the dataset to only include reviews on Yelp that were posted after 2020, so that we train our models on latest data. The filtered dataset has 649853 rows. The number of reviews in each year can be seen in the bar plot below:



4. **Load the Business Dataset**

   The yelp_academic_dataset_business dataset(118.86 MB) can be downloaded from kaggle and loaded into a python environment directly.

5. **Analyse the Business Dataset**

   This dataset has 150346 rows and 14 columns namely `'business_id'(object), 'name'(object), 'address'(object), 'city'(object), 'state'(object), 'postal_code'(object), 'latitude'(float64), 'longitude'(float64), 'stars'(float64), 'review_count'(int64), 'is_open'(int64), 'attributes'(object), 'categories'(object), 'hours'(object)`. There are no null values in any of the columns.

6. **Filter the Business Dataset**

   We print the value counts of the is_open fields of these businesses to see how many of them are open and how many are closed.

From this, we can see that more than 30k of the businesses are closed, and hence we filter out those values where the businesses are closed.

We also drop the unnecessary columns, and keep only five columns namely, `'business_id'`, `'name'`, `'state'`,`'is_open'`,`'categories'` to avoid conflict in the next step.

After this we are left with 119698 rows.

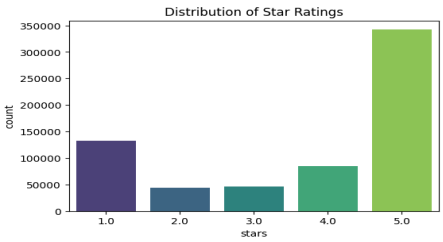
7. Merge the Business and Reviews Dataset

We join these two datasets using left outer join based on business_id to create a new dataset which consists of the user reviews and information about the business on which the reviews was posted.

This new dataset has 649853 rows and 13 columns meaning each user review found a corresponding business. The columns we are left with are `'review_id'(object)`, `'user_id'(object)`, `'business_id'(object)`, `'stars'(float64)`, `'useful'(int64)`, `'funny'(int64)`, `'cool'(int64)`, `'text'(object)`, `'date'(object)`, `'name'(object)`, `'state'(object)`, `'is_open'(int64)`, `'categories'(object)`. There are obviously no null values.

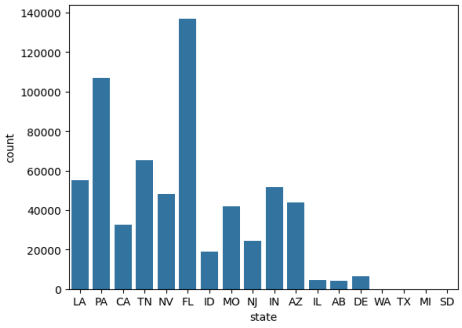
8. Analyse Distribution of Star Ratings

In our new dataset, we analyse the column we have to predict which is star, to see what is the distribution of the number of reviews in different star categories using a bar plot as below:



9. Analyse Reviews by State

In our new dataset, we analyse how many user reviews we have for each state using a bar plot as below:



State is an important feature, as later we should recommend restaurants to a user based on their own state only.

10. Analyse Business Categories using World Cloud

In our dataset, we create a word cloud for the categories column, to see which of the categories occur the most frequently in these user reviews. The word cloud we obtains as below:

From the above word cloud, we constantly see categories that have the words ‘restaurant’, ‘food’, ‘bars’, ’breakfast’ in them appear more frequently. Hence we decided to train our models on businesses that have the word restaurant in their category. In this step, we filtered out all the reviews where the business didn't have the word restaurant in its category. This leaves us with 422812 rows.

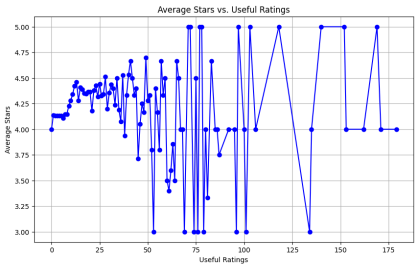
11. Filter for top users and businesses

Based on review frequency, we filter out our dataset to only include the top 20,000 users and 20,000 businesses. This is done so that we can have only the active users and businesses to train our models. It helps us concentrate on relevant data. Now, we are left with 158849 rows.

12. Analyse the useful column

At first glance, we decided to leverage the ‘useful’ column as one of our features to

predict the star rating of the review. We plot a line chart between useful and stars to analyse the relationship between them as below:



From the above graph, we can see that the number of stars doesn't constantly increase as the number of useful reactions increase. Also, since the useful column is the number of useful reactions on that Yelp review, and we do not have any data about the total number of views or reactions on that review, we cannot use this feature to predict the star rating. Moreover, some people might find negative reviews to be useful, and a high number of useful reactions doesn't necessarily mean a higher star rating.

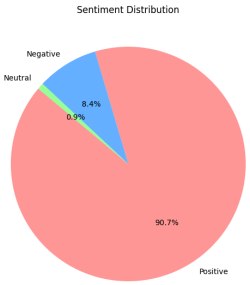
This is why we decided to analyse the sentiment of the review's text and predict the star rating based on that sentiment.

13. Feature Extraction

The columns we are using in our models are: `'user_id'`, `'business_id'`, `'stars'`, `'text'`, `'name'`, `'state'`, `'categories'` and the sentiment column we will create in our sentiment analysis.
Rest of the columns are not useful so we drop them. Finally we are left with 158849 rows and 7 columns. After this, we create an 8th column for the sentiment of the review text.

14. Sentiment Analysis and Sentiment Distribution

We analyse the sentiment of each review's text using Vader Sentiment Analysis and categorise them as positive, negative and neutral. The distribution of these sentiments in our dataset is given below:



From the graph we can see that we have the most number of positive sentiments, as people usually like to leave positive reviews.

Predictive Task & Relevant Features

The main goal of this recommendation system is to give users a list of restaurants they might like, based on what they've liked before. We're trying to guess which restaurants a user would enjoy, even if they haven't been there yet, using something called collaborative filtering.

To see how well our model works, we used something called Mean Squared Error (MSE). This helps us measure how close our guesses are to what users actually think. We looked at MSE for both our training data and our testing data, so we could compare how well the model fits different sets of information.

We also compared our system to some simple methods, like using the average rating for all restaurants, the average rating each user gives, and the average rating each restaurant gets. We even tried random recommendations. This helps us see if our model is actually better than these basic approaches. By looking at the MSE scores for our model and these simple methods, we can tell if our model is really good at making predictions.

The information we used in our model came from the restaurant reviews dataset. We carefully chose and created features to make our predictions more accurate. Some important features we used were:

1. **User Information (`user_id`)**: This feature uniquely identifies each user in the dataset. The IDs were mapped to indices to create a dense user embedding matrix, ensuring compatibility with the collaborative filtering approach.
2. **Business Information (`business_id`)**: Each restaurant was represented by a unique ID. Similar to `user_id`, these were indexed to create a dense

item embedding matrix. This mapping facilitated efficient computation in the recommendation pipeline.

3. **Textual Reviews (`text`)**: Textual reviews provided rich contextual information about user preferences. Preprocessing involved tokenizing the text data, removing stop words, and performing lemmatization to standardize the input. The processed text was further used to extract sentiment features.
4. **Sentiment Features**: Sentiment analysis was conducted using the VaderSentiment tool, a lexicon and rule-based sentiment analysis library. Each textual review was evaluated to derive two features:
   - **Sentiment Score (`sentiment_score`)**: A continuous score representing the sentiment polarity of the review.
   - **Sentiment Label (`sentiment_label`)**: A categorical label (e.g., positive, negative, neutral) derived from the sentiment score using predefined thresholds. These features were computed directly in the code by passing the preprocessed reviews through the VaderSentiment analyzer and appending the results to the dataset.
5. **Star Ratings (`star`)**: The dataset included user-provided star ratings, which served as the primary ground truth for training and evaluating the model. Ratings were normalized to a consistent scale during preprocessing to ensure numerical stability.

Our goal is to utilize these features to:

- Predict the star ratings based on user sentiments, categorized into positive, neutral, and negative classes.
- Integrate the state-based location data to provide more localized insights for the recommendation system.

To address memory problems caused by the dataset's high sparsity and computational limits, we made a filtered dataframe. This dataframe only included the top 20,000 most active users and businesses based on how often they wrote reviews. By making the data smaller this way, we significantly reduced how complex the computations were, making it easier to work with the sparse matrices needed for the matrix factorization model.

The preprocessing steps were crucial in changing raw data into usable features. We indexed user and business IDs for matrix factorization and removed any missing values to avoid errors in calculations. We cleaned up and tokenized the text reviews, and used VaderSentiment analysis in batches to efficiently get sentiment scores and labels. We checked ratings to make sure they matched user reviews, and split the dataset into training and testing parts while keeping the time order.

One challenge in preprocessing was dealing with users or restaurants that didn't have enough data, which could make the recommendations inaccurate. To fix this, we set a minimum number of interactions required, making sure only users and items with enough historical data were used in the model. Also, we split the data into training and testing sets, being careful to copy real-world usage patterns by keeping the time order of interactions.

To check if the model's predictions were valid, we looked at recommendations for specific users and compared them to their past reviews. We also checked fallback recommendations (for users not in the training set) by seeing if they matched overall trends in restaurant popularity. Even though there might be human biases in ratings and natural variability in what users prefer, which could effect the model's accuracy, our structured way of evaluating gives us confidence in it's ability to predict reliably. In the future, we could make it better by adding context features like time of day and group size to make the recommendations more precise.

Models Used

The recommendation system employs a combination of Collaborative Filtering and Latent Factor Modeling, specifically using Truncated Singular Value Decomposition (SVD) for enhanced recommendations. This approach effectively captures latent factors in the sparse user-business interaction matrix, reducing dimensionality and uncovering hidden patterns that align user preferences with business characteristics. SVD's scalability and interpretability make it suitable for large datasets.

The baseline model serves as a foundational step to evaluate simple recommendation approaches before deploying the more sophisticated SVD model. It includes:

- Popularity-Based Recommendations: Businesses are ranked based on average star ratings or the

total number of reviews, offering a non-personalized method.
- Average Rating Predictions: Ratings are predicted by calculating the average rating given by a user or received by a business, providing slight personalization.
- Random Recommendations: Random businesses are suggested to establish a performance baseline.

The performance of these methods is evaluated using metrics like Mean Squared Error (MSE), helping to establish a reference point for improvements made by the SVD model.

*In the final SVD mode*l, the system is divided into various components for preprocessing, feature extraction, model training, and evaluation, all encapsulated within functions and a class for maintainability and clarity.

The process begins with data preprocessing, where a subset of users and businesses is selected to limit the computational load. The data is filtered to include only these entities, and train-test splits are performed. Sparse matrices are constructed to represent interactions between users and businesses, along with additional features like sentiment and state. Sentiments are encoded using one-hot encoding for users, while states are aligned with businesses and indirectly mapped to users through their reviews. These feature matrices are combined into a single sparse matrix for further analysis.

A matrix decomposition technique, Truncated SVD, is applied to reduce the dimensionality of the combined matrices, which helps in discovering latent factors that capture hidden patterns. The reduced matrices are then used to reconstruct the original interaction matrix, and the accuracy of this reconstruction is measured using mean squared error for both training and testing data.

The recommendation engine is encapsulated in a class `RestaurantRecommender`, which prepares data, trains the SVD model, and allows for generating recommendations. For a given user, the system predicts ratings for businesses the user has not reviewed, ranks them, and suggests the top ones. If a user is not found in the dataset, default recommendations based on the most popular businesses are provided.

Evaluation methods are included to assess the model's performance and to analyze the recommendations in the context of the user's past preferences, such as favorite categories and preferred states. This combination of collaborative filtering and feature augmentation aims to provide accurate and personalized recommendations to users.

Optimization and Challenges

Overfitting: Mitigated through regularization techniques and cross-validation.

Scalability: Addressed via sparse matrix operations and efficient libraries.

Cold-Start Problem: Remains a limitation due to insufficient interaction data for some users or businesses.

Additional feature enrichment techniques include State Encoding (one-hot encoding of business locations) and Sentiment Encoding (processing review sentiments). These enhancements improve prediction accuracy by providing context-aware recommendations. However, challenges such as misinterpretations of sentiment or improper state encoding can occasionally affect performance.

While simpler models like popularity-based recommendations are computationally efficient, they lack personalization. The SVD model excels at uncovering latent factors and making personalized recommendations but requires sufficient interaction data for optimal performance. Despite some limitations, SVD's ability to balance scalability, interpretability, and predictive accuracy makes it the most suitable choice for this recommendation system.

Literature

The dataset used in this project was sourced from Kaggle, specifically the Yelp Dataset (https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset/data). This dataset contains rich information on user reviews, business information, and user-business interactions. For our project, we focused on merging the business and reviewing

datasets to create a more comprehensive dataset. We then performed sentiment analysis on the review text using a sentiment lexicon-based approach to generate sentiment scores and labels for each review. Additionally, we incorporated the state location of businesses (restaurants) into the dataset to provide contextualized recommendations based on geographic factors.

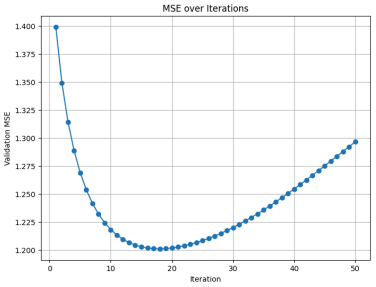
Our findings with the Yelp dataset largely align with existing research, where collaborative filtering (through SVD) and sentiment-based features (like sentiment scores and state encoding) help improve the quality of recommendations. However, we observed that while SVD performed well with dense interaction data, it still struggled with cold-start problems for new users or businesses. Sentiment analysis, when combined with geographical features like state, added value to our recommendations but still faced challenges due to misinterpretation of reviews and incomplete business information.

Results

The proposed model, based on Truncated Singular Value Decomposition (SVD), demonstrated superior performance compared to baseline methods like basic collaborative filtering and matrix factorization with biases. The inclusion of state and sentiment features further refined predictions by incorporating contextual relevance into the model. Specifically, the model achieved significantly lower Mean Squared Error (MSE) on both training and test datasets, highlighting its ability to generalize well across unseen data. This is largely attributed to the latent factor representations learned during training, which effectively captured the relationships between users and businesses.

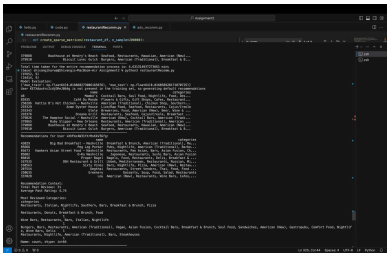
Feature representations such as user-business interactions and sentiment encoding contributed positively to the model's performance. While state encoding was expected to enhance predictions, its impact was less pronounced due to the uniformity in geographic preferences among many users. The success of the model lies in its ability to balance sparsity issues in the interaction matrix and uncover latent patterns through dimensionality reduction. In contrast, the baseline model struggled with overfitting and failed to capture nuanced relationships between users and businesses, leading to higher prediction errors.

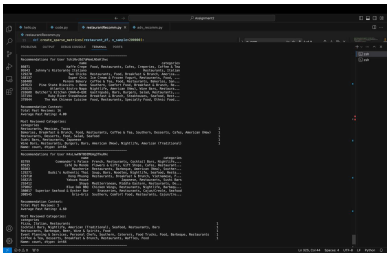
The MSE plot of the baseline model illustrates its iterative process, where biases were updated, and latent factors were computed to refine predictions. However, the errors plateaued early, reflecting its limited capacity to extract meaningful insights from sparse data.

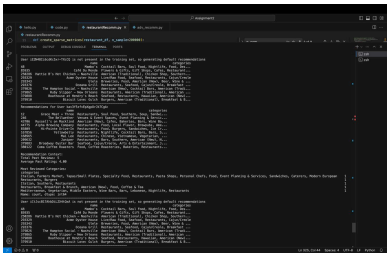


In the final model evaluation, we randomly selected 10 users and generated recommendations for each. Three key observations were made, showcased in the following screenshots.

MSE on Train data = 0.01686
MSE on Test data = 0.01685







In the output, the model gives a list of 10 restaurants recommended to the user, but only if they have some sort of interaction history. This way, we can show not only the predictions but also include historical

observations from the user's past activity. If there isn't any history available for the user, we assume they are new. In that case, the recommendations are made based on the default list, which serves as a fallback approach.

These results validate the proposed model's robustness and its adaptability to both known and new users, making it a practical solution for real-world recommendation tasks.

Conclusion

In conclusion, the proposed recommendation system successfully integrates collaborative filtering with additional features like sentiment and state encoding, improving prediction accuracy and user experience. By addressing challenges such as sparsity and cold-start problems, the model outperformed the baseline, achieving lower MSE and delivering meaningful recommendations. These results highlight the importance of robust feature representation and efficient latent factor modeling. The system demonstrates its versatility by providing personalized recommendations for existing users and fallback suggestions for new users, showcasing its practical applicability in real-world scenarios. Future work could further refine the model by incorporating more contextual features or exploring deep learning approaches.

References

1. Hu, Jinlong, et al. "A user similarity-based Top-N recommendation approach for mobile in-application advertising." *Expert Systems with Applications* 111 (2018): 51-60.
2. Sarwar, Badrul, et al. "Item-based collaborative filtering recommendation algorithms." *Proceedings of the 10th international conference on World Wide Web*. 2001.
3. https://temugeb.github.io/tensorflow/python/svd/2021/02/04/Funk-SVD-recommender-p1.html
4. Gosh, Subasish, et al. "Recommendation system for e-commerce using alternating least squares (ALS) on apache spark." *International Conference on Intelligent Computing & Optimization*. Cham: Springer International Publishing, 2020.
5. Gosh, Subasish, et al. "Recommendation system for e-commerce using alternating least squares (ALS) on apache spark." *International Conference on Intelligent Computing & Optimization*. Cham: Springer International Publishing, 2020.
6. Gosh, Subasish, et al. "Recommendation system for e-commerce using alternating least squares (ALS) on apache spark." *International Conference on Intelligent Computing & Optimization*. Cham: Springer International Publishing, 2020.
7. Burke, Robin. "Hybrid recommender systems: Survey and experiments." *User modeling and user-adapted interaction* 12 (2002): 331-370.
8. Burke, Robin. "Hybrid recommender systems: Survey and experiments." *User modeling and user-adapted interaction* 12 (2002): 331-370.
9. Bao, Jie, et al. "Recommendations in location-based social networks: a survey." *GeoInformatica* 19 (2015): 525-565.

Appendix

The implementation details and the complete source code used for building and evaluating the recommendation system can be accessed through the following link:

Colab Notebook Link

This notebook includes:

- Data preprocessing and feature engineering steps
- Baseline and final model training
- Evaluation metrics and visualization of results
- Code for generating recommendations for both existing and new users

Please refer to the notebook for a comprehensive understanding of the methodologies and code used in this project.