Hybrid Yelp Restaurant Recommendation Engine

Lei Yao
University of Waterloo
Waterloo, Ontario
l27yao@uwaterloo.ca

Ruifan Yu
University of Waterloo
Waterloo, Ontario
r33yu@uwaterloo.ca

Yuefei Xiang
University of Waterloo
Waterloo, Ontario
y33xiang@uwaterloo.ca

ABSTRACT

Recommender systems are techniques that provide suggestions to users on items or services that might be useful. Popular online shopping websites, such as Amazon, eBay, benefit from the recommendation system as it helps improve shopping experience of users. In this project, we built a hybrid Yelp restaurant recommendation engine by using both collaborative filtering and the content-based recommendation. Collaborative filtering includes K-Nearest Neighbors model and Alternating Least Squares (ALS) algorithm. K-Nearest Neighbors model can predict a user's rating to an unknown restaurant by using the similarity between two users. Considering the sparsity of review data, a matrix factorization approach called ALS is used to predict missing ratings through mining a small set of latent factors. Content-based recommendation analyzes the relationship among different restaurants' reviews. Latent Semantic Analysis (LSA) model is leveraged to distill the review content into a set of relevant concepts, then the similarity among different restaurants is discovered based on these hidden concepts. The result shows that the performance of our hybrid recommendation engine is much better than a naive baseline model.

KEYWORDS

Recommendation engine, K-Nearest Neighbors, Alternating Least Squares, Latent Semantic Analysis

1 INTRODUCTION

As a result of rapid data growth, people are always get lost in the plethora products and inundated by multiple choices. Spending too much time in making decisions destroys the experience of shopping. To solve this problem, entity recommendation is a good service for customers when they are shopping online. Thus, recommender systems play an important role in mining the customer interests and improving users' shopping experience.

Commonly used techniques for the recommendation system include collaborative filtering, content-based recommendation, and hybrid recommendation systems. Collaborative filtering [1] methods predict what users might like based on user similarity, which is computed by analyzing a large amount of data about users' behaviors, activities or preferences. One important assumption of the collaborative filtering is that people whose preferences agreed in the past still have the same taste in the future, then similar kinds of items will be chosen as those they bought in the past. Several approaches are commonly used for collaborative filtering. The first one is a user-based model, which focuses on the relationship between two customers. The predicted rating of an item a user might give is calculated by computing the weighted sum of different users ratings on this item [2]. Similar to the user-based model, the similarity between two items is calculated in order to make a prediction, this is called a item-based model [3]. Matrix Factorization approach is widely used in the collaborative filtering recommendation, Alternating Least Squares (ALS) [4] is one such algorithm that has been used to iteratively solve a series of least squares regression problems.

Content-based recommendation makes recommendation based on the content or attributes of items. Features of items are analyzed in order to find similar items, then these similar items can be recommended to customers. One popular example of a content-based recommendation system is Pandora Radio, which plays music with similar characteristics to that of a song provided by the user as an initial seed. In content-based recommendation, Word2vec [5] algorithm is a commonly used approach that computes distributed vector representation of words. The concept of the Word2vec algorithm is that similar words are close in the vector space, so the output of Word2vec model are lists of words along with their similar words. Another typical algorithm used in content-based recommendation is Latent Dirichlet allocation (LDA) [6], which represents documents as mixtures of topics that represent original corpus. Besides these two methods, TD-IDF algorithm is widely used to extract document features.

Since the volume of data is growing so large, Apache Spark [7] is leveraged in our project to make sure that our recommendation system can achieve both scalability and efficiency. This framework is good at leveraging computer memory in computing and gives the advantage of handling iterative and interactive algorithms with efficiency.

In this project, we aimed to build a hybrid recommendation system that is able to make sophisticated restaurant recommendations based on the raw dataset provided by Yelp. Yelp is one of the most popular review sites that contains a massive amount of user and business data. We optimized these raw data to organized, digestible data, implemented collaborative filtering recommendation and content-based recommendation separately, and finally built a hybrid recommendation engine through model ensemble.

The remainder of this paper is structured as follow. Section 2 introduces the structure and contents of the dataset we used. Section 3 discusses a naive baseline implemented for the recommender system. Section 4 and Section 5 discuss the implementation of both collaborative filtering recommendation system and content-based recommendation system. Metrics used in evaluating these two kinds of recommendation systems are discussed in Section 6. The evaluation result is detailed in Section 7. We then conclude our project and propose some future work in Section 8.

2 DATASET

The Yelp dataset (about 5GB) is used in our project. It is an integrated dataset that contains actual information of restaurants, users, users' review, and tips from users. The format of raw data is Json. We extracted parts of useful data from this dataset, which is listed in Table 1 to 3:

Fields	Description
business_id	The unique id of a restaurant
state	The state a restaurant belongs to
stars	Average customers' rating of a restaurant

Table 1: Business dataset

Fields	Description
review_id	The unique id of a review
user_id	The unique id of a customer
business_id	The unique id of a restaurant
stars	Customers' rating in a review
text	Review contents

Table 2: Review dataset

Fields	Description
user_id	The unique id of a customer
average_stars	The average rating given by a user

Table 3: User dataset

Fields	value
Number of reviews	4153150
Number of users	1029432
Number of restaurants	144072
Average number of reviews per user	4.13
Median number of reviews per user	5
Average star for each restaurant	3.64
Average star for each customer	3.72

Table 4: Data Statistics

Apart from the given information, we collected some interesting statistics for the Yelp dataset, these information is shown in Table 4. The median and the average number of reviews per user are around 5. Over half of the users in the dataset have written less than five reviews, so we can infer that about five restaurants are visited per user. Compared with the number of restaurants, this dataset is very sparse, which means that we have considerably few preference data for per user.

3 NAIVE BASELINE

Before researching on the collaborative filtering and content-based recommendation, we implemented a naive baseline, which is similar to the approach mentioned in the [8]. This method simply outputs a predicted rating given by a user to a restaurant based on a series of average ratings:

$$\hat{R}_{ur} = \bar{R} + (\bar{R}_u - \bar{R}) + (\bar{R}_r - \bar{R})$$

where $\hat{R}_{ur}$ is a predicted rating given by user u to restaurant r. $\bar{R}$ is the average rating over all reviews. $\bar{R}_u$ and $\bar{R}_r$ represent the average rating of user u and the average rating of restaurant r. The $\hat{R}_{ur}$ is deviated by the summation of the reviews' average, user bias, and restaurant bias.

4 COLLABORATIVE FILTERING RECOMMENDATION

Collaborative Filtering (CF) is a commonly used approach for recommender systems. Assume in a typical case, we have a list of m users $U = \{u_1, u_2, ..., u_m\}$, as well as a list of n items $I = \{i_1, i_2, ..., i_n\}$. Noting M as a $m \times n$ matrix where each $M_{i,j}$ entry stands for the rating score that user u_i gives to item i_j.

The intuition behind this method is that we assume a group of users, who are similar with each other, may share the same taste or preference towards a specific item. Therefore, collaborative filtering collects the past behaviors of a target user as the description of the characteristic. This technique aims at filling in the missing entries of a user-item association matrix based on a collaboration of multiple users that are selected by a similarity measurement on the characteristic. When calculating the similarity measurement, we regard each user as a vector, then we use several method to evaluate the similarity score. The recommendation task becomes generating the prediction of preference of a user and selecting top K items with highest predicted rating as recommendations. Since we focus on the similarity between users, it is called user-based collaborative filtering. Meanwhile, as we can regard a user as a vector, we can also regard item as a vector where each entry stands for the rating given by a corresponding user. Similarly, the method is called item-based collaborative filtering.

In this part, we mainly focus on the user-based collaborative filtering approach. We use K-Nearest Neighbors and ALS as the implementation of user-based collaborative filtering.

4.1 K-Nearest Neighbors Approach

K-Nearest Neighbors collaborative filtering was first introduced by GroupLens[9]. This is a pretty straightforward implementation of the core idea of collaborative filtering: finding other users whose rating history behaviors are similar to target user. Similar to KNN in machine learning applications, in this scenario, there are some similarity function: $f : u \times u \rightarrow R$ computing the similarity between two users.

4.1.1 Pearson Correlation. This function calculates the statistical correlation based on the common ratings of two users to produce their similarity. The similarity function is computed as:

$$sim(u,v) = \frac{\sum_{i \in U_u \cap U_v}(R_{u,i} - \overline{R}_u)(R_{v,i} - \overline{R}_v)}{\sqrt{\sum_{i \in U_u \cap U_v}(R_{u,i} - \overline{R}_u)^2}\sqrt{\sum_{i \in U_u \cap U_v}(R_{v,i} - \overline{R}_v)^2}}$$

This function maps two user-vector to $[-1, 1]$, where 1 means totally positive correlation and -1 means totally negative correlation. It applies the standard deviations of two vectors as the normalization term.

A weakness of Pearson Correlation is that when ratings are sparse, which means there are few common ratings between two users, this measurement is very high and not reliable. Introducing a threshold on the number of common ratings for full similarity(correlation of 1) can alleviate this problem.

4.1.2 Cosine Similarity. As a common distance measurement for vectors, cosine similarity is widely used to show the angle between two vectors. This measurement can be efficiently computed with the help of some linear algebra package. The calculation is taking the dot product and dividing it by their L_2 norm.

$$Sim(u,v) = \frac{R_u \cdot R_v}{||R_u||_2||R_v||_2} = \frac{\sum_i R_{u,i}R_{v,i}}{\sqrt{\sum_i R_{u,i}^2}\sqrt{\sum_i R_{v,i}^2}}$$

A good property of cosine similarity is that we can assume all unknown ratings as 0. Therefore, they will not influence the similarity score.

However, there is one special case, suppose there are two users and two items, the rating vectors are $(0, 1)$ and $(0, 5)$, where 1 means dislike and 5 meas like, the cosine similarity between them is 1, meaning these two users are very similar, which is contradicting to reality. This problem can be alleviated by mapping the rating to a range symmetric to 0. For instance, we can map $[0, 5]$ to $[-2.5, 2.5]$ and use the positive and negative signal to represent the preference.

4.1.3 Generating Predictions. Once we extract top K nearest neighbors by the similarity function mentioned above, we can generate the prediction based on these vectors.

A straightforward method is directly calculating the average rating as the prediction. Another method is using a weighted average. The contribution of a vector is positive proportion to its similarity to target user.

$$Prediction_{u,i} = \frac{\sum_{j \in S} Sim(i,j)R_{u,j}}{\sum_{j \in S} |Sim(i,j)|}$$

4.1.4 Implementation. As for the implementation of collaborative filtering recommendation in this project, the original dataset is JSON file with userID, RestaurantID, and rating. Considering the user vector can be very sparse, we use SparseVector to store the user-item rating for each user. Therefore, we map this RDD to similarity score and take top K entries based on the similarity score. The prediction of user i to item j is approximated by the weighted average of the ratings given by these neighbors. The process is described in Figure 1.

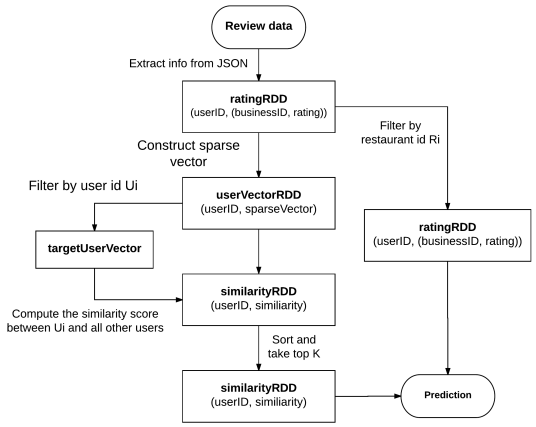

Figure 1: The implementation of KNN in Apache Spark

4.1.5 The weakness of KNN. Although KNN method is quite easy for implementation, there are two weakness points of it:

- Sparsity: In real life, the user-item rating matrix may be extremely sparse, which has been mentioned in dataset section. A stable KNN collaborative filtering method needs confidentially similar score to find correct neighbors.
- Scalability: K-Nearest Neighbors method suffers from the growing scale of the dataset. The complexity is $O(mn)$, where m stands for m users, and n stands for n items. As the volume of dataset is increasing all the time, it becomes more and more difficult to correctly detect the similar user group in a given time.

4.2 Matrix Factorization Approach

In order to overcome the weaknesses that has been described in 4.1.5, latent models are used to represent the similarity between users in a lower dimensional space. The matrix factorization has these assumptions

- Each user can be represented by k features.
- Each item(Restaurant) can be described as k features.
- the dot product of the corresponding vectors will give us the approximation of the rating that a user give to an item.

Therefore, users and items are described by a small set of latent factors that can be used to predict missing entries. Matrix factorization can be described as a learning process in which we are given a sparse rating matrix and we try to decompose it into a product of two smaller matrices with lower dimensions. This idea is similar to PCA and other methods used in machine learning for dimension reduction.

Once we find these two lower dimension matrices, one is $m \times k$ and the other is $k \times n$, we already transform the original vectors to k hidden features. To compute a prediction, we just need to computer

the dot product of relevant row in users matrix and relevant column in items(restaurants) matrix.

In addition, to find the similarity between two items or two users, we can apply the same similarity measurements as discussed above in K-Nearest Neighbors method.

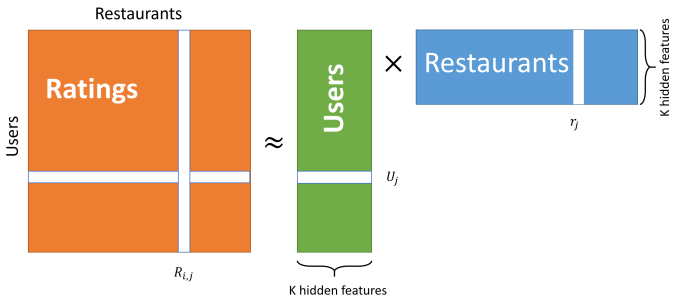

Figure 2: The illustration of Matrix Factorization

The remaining part is using an optimization algorithm to achieve matrix factorization. To formulate the problem, we need to solve this problem:

$$\min_{X,Y} \sum_{R_{u,i} \in R} (R_{u,i} - x_u^T y_i)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$$

This objective function is non-convex and yet there is no sufficient algorithm to optimize it. However, in this scenario, we do not need a exact solution, as a little accuracy loss is acceptable. We can apply gradient descent method.

4.2.1 Alternating Least Squares (ALS). A special algorithm called Alternating Least Squares (ALS) is very suitable for this case. The idea is very simple: instead of co-updating in gradient descent, in each iteration, we fix one matrix, then solve the other and vice versa. To be exact, we fix X, optimize Y, fix Y, optimize X, and repeat over and over again until we achieve error tolerance or convergence.

The algorithm is described as:

Algorithm 1 ALS for Matrix Factorization

initialize X, Y
for $t \in 1, 2, ..., T$ **do**
  for $u = 1...n$ **do**
    $x_u = (\sum y_i y_i^T + \lambda I_k)^{-1} \sum R_{u,i} y_i$
  end for
  for $i = 1...m$ **do**
    $y_i = (\sum x_u x_u^T + \lambda I_k)^{-1} \sum R_{u,i} x_u$
  end for
end for

There is an API in Spark.MLlib using alternating least squares (ALS) algorithm to learn latent factors. In this project, we call this API to get wanted matrices.

4.3 Prediction Correction

From empirical knowledge, we know that the average rating level between different users may vary from one user to other users. For instance, some users who are strict with higher standard may give relatively lower rating for all restaurants. In order to obtain better prediction, we need to perform a correction on prediction.

Let $\hat{U}$ be the average rating of all users, $\hat{u_i}$ be the average rating of user i, and p be the prediction by our model.

The value $\Delta = \hat{u_i} - \hat{U}$ gives an estimation of rating preference of user i compared with all users. Therefore, we add this difference into our result such that the final prediction takes user's preference into consideration. The final output p' is defined as:

$$p' = p + \Delta = p + \hat{u_i} - \hat{U}$$

5 CONTENT-BASED RECOMMENDATION

For content-based recommendation,user reviews are used as the profile of each restaurant, and a technique called *Latent Semantic Analysis* (LSA), which is widely used in natural language processing (NLP) and information retrieval, is used to process the review. LSA can distill the review content of each restaurant into a set of relevant concepts by using a linear algebra technique called *Singular value decomposition* (SVD). LSA can help discover the similarity among different restaurants by selecting the most important concepts. Based on a user's history tastes, potential restaurants can be recommended to users. The flowchart of content-based recommendation is shown in Figure 3.

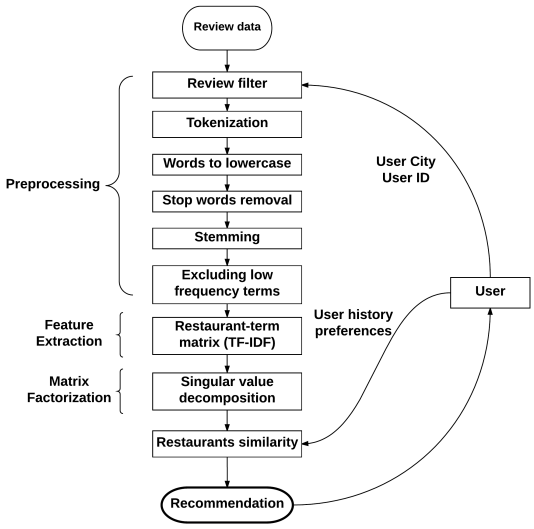

Figure 3: Content-based recommendation

5.1 Assumptions

Since we use user reviews as the profile of each restaurant and recommend similar restaurants to each user based on his or her history tastes, several assumptions are made as following

- The text in a user's review can reflect his personal tastes and preferences in restaurants.
- By looking at all the reviews for a specific restaurant, we can infer the strengths, weaknesses, and attributes of the restaurant
- The user has the same "tastes" regardless of which restaurants he goes to.
- Ordering of words, sentence structure, and negations in user reviews are not important in our model.

5.2 Text Preprocessing

Text data is complex to work with for several reasons. First, raw data of user reviews contains a lot of punctuations, wrong words, stop words, spurious white spaces, and non-alphanumeric characters, which are meaningless for analysis. Second, one word may have different forms. Hence, we preprocess the raw review. There is a common text preprocessing pipeline by which we followed in review preprocessing.

(1) **Tokenization**: the first step is to remove all punctuations and white spaces in reviews and split up the raw review of each restaurant into a collection of terms (tokens).
(2) **Words to lowercase**: we preserved terms that contain only characters and converted each term to lowercase for each restaurant.
(3) **Stop words removal**: stops words refer to common words that occur many times across almost all reviews. Examples of typical English stop words include the, of, is, and, and so on. We need to remove these words from tokenized reviews.
(4) **Stemming**: it is the conversion of whole words to a base form. For example, plurals might be converted to singular (*cats* becomes *cat*), and forms such as *listening* and *listener* might be converted to *listen*. Most of the time, different forms of one word has the same meaning.
(5) **Excluding terms based on frequency**: if the overall occurrence of one word in all reviews is very low, this word is not useful in our model, so it is necessary to exclude rare words.

We collected a stop words library and removed words that occur in this library. There is also an English vocabulary and its stemmed equivalent, which are used by us in building a HashMap. We check each term in reviews and replace it with its stemmed base form.

5.3 Feature Extraction

Before performing analysis, transforming the raw text of each restaurant's user reviews into a term-restaurant matrix is required by LSA . In this matrix, each row represents a restaurant, and each column represents a term that occurs in the user reviews. The value at each entry corresponds to the importance of the column's term to the row's restaurant. The most commonly used method is *term frequency* times *inverse document frequency* (TF-IDF), which is a feature vectorization method in calculating the importance of a term to a document.

TF-IDF is useful in capturing the importance of a term to a restaurant in two ways. First, the more often a term occurs in a restaurant's reviews, the more important it is to that restaurant. Second, it is more meaningful to have a term that occurs relatively rare in all reviews than a term that occurs frequently in reviews of almost all restaurants. Hence, it is necessary to include the inverse of the number of restaurants a term occurs in. The less the restaurant frequency of a term, the more discriminating power it has.

For each restaurant, we first construct a HashMap, which contains information about how many times each word occurs in this restaurant's reviews. Then, keys of each restaurant's HashMap are emitted, so we can count the restaurant frequency of each term, this number is used to compute inverse document frequency of each term. Words that occur in a few restaurants' reviews are excluded, since too many terms will make it impossible for our restaurant-term matrix to fit in memory, we sorted terms based on their frequency and took top 10,000 frequent words. Given all these data, TF-IDF value of each term in each restaurant can be calculated. Finally, we build a sparse matrix M with rows representing all restaurants, columns representing 10,000 terms and each position representing the corresponding TF-IDF value.

To find the similarity between two restaurants, we can achieve a relevance score between two restaurants by computing the *cosine similarity* between two rows vectors in matrix M. However, this score only takes TF-IDF value into account, relying heavily on simple frequency counts. LSA uses a technique called *singular value decomposition*, which can provide a deeper understanding of the text and computational efficiency.

5.4 Singular Value Decomposition

Singular value decomposition (SVD) is a matrix factorization and dimensionality reduction method, which is applied to TF-IDF restaurant-term matrix M. The singular value decomposition takes an $m \times n$ matrix and factorizes it into three matrices, which approximately equal to the original matrix M when they are multiplied together.

$$M \approx USV^T$$

U is an $m \times k$ matrix, S is an $k \times k$ matrix and V is an $n \times k$ matrix. Here, m is the number of restaurants and n is the number of terms. There is also a decomposition parameter k, which is less than n. k indicates the number of concepts in the original matrix and is usually chosen to be much smaller than n. SVD ensures the multiplication of three factorized matrices closely approximates the original matrix. Matrix U defines a mapping between all restaurants and all hidden concepts. Matrix V defines a mapping between all terms and all hidden concepts. S is a diagonal matrix that holds the singular values, the magnitude of each singular values corresponds to the importance of each concept.

Spark machine library has a built-in API for SVD computation, the input is a sparse matrix M, the output contains three factorized matrices U, S, and V. However, the computation requires $O(nk)$ storage on the driver, which might be a bottleneck for a large matrix computation.

SVD packs the important information into a lower-dimensional representation, computation can be done on this representation instead of the original restaurant-term matrix M. This practice can provide efficiency. Meanwhile, reconstructing matrix M by using three factorized matrices has several advantages. First, reconstructed matrix accounts for synonymy by condensing similar

terms. Second, it also accounts for polysemy by placing less weight on terms that have multiple meanings. Third, noise in the original matrix M can be thrown out.

5.5 Recommendation

We recommend restaurants based on an assumption that a user are more likely to choose a restaurant that has a high relevance score with restaurants to which he has given high stars.

The relevance score can be computed without using the reconstructed matrix. Some linear algebra manipulations show that the cosine similarity between two rows (two restaurants) in the reconstructed matrix is equal to the cosine similarity between the corresponding rows in US, where U and S are two of factorized matrices. Finding the cosine similarity between a restaurant and all other restaurants is equivalent to first normalizing each row in US to length 1 and then multiplying the row corresponding to that restaurant by it. Finally, each value in the resulting vector is the relevance score between a restaurant and the target restaurant.

For each restaurant, we can get its similar restaurants based on relevance scores. The computation of a predicted star that one user gives to a restaurant uses the same method as that has been mentioned in Section 4.3.

6 EVALUATION METRIC

For naive baseline model and collaborative filtering model, inputs are users, restaurants and stars that users gave to different restaurants. For content-based recommendation, the input is a matrix, which contains the relationship (TF-IDF) between restaurants and their corresponding reviews. Although the input is different for above methods, all of them can predict a rating that one user gives to a specific restaurant, this is shown in Figure 4.

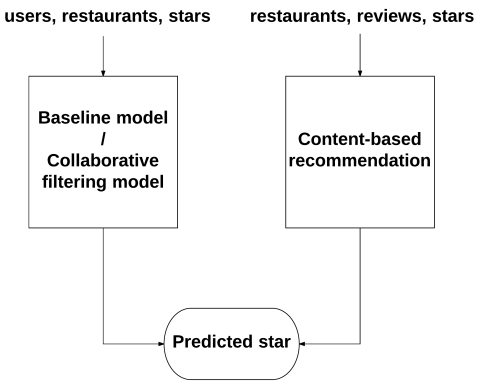

Figure 4: Different inputs, same output

Since outputs of all models are the same, we can evaluate different models by using the same metric. We use 5-fold cross validation and each time we randomly choose 20% of all reviews, each contains information about a business id, a user id, and a star. Given a user id and a restaurant id, we can predict a star that user might give to that restaurant and compare it with the actual star in this review, which is the ground truth of our evaluation part.

Evaluation metrics used in evaluating recommendation systems include root mean square error (RMSE) and mean absolute error (MAE). As the Microsoft Research Paper [10] on evaluating recommendation systems states, both mean square error and mean absolute are accurate and frequently used in measuring the performance of a recommendation system. RSME and MAE are given below:

$$RMSE = \sqrt{\frac{1}{|S|}\sum_{(u,r)\in S}(\hat{R}_{ur} - R_{ur})^2}$$

$$MAE = \sqrt{\frac{1}{|S|}\sum_{(u,r)\in S}|\hat{R}_{ur} - R_{ur}|}$$

where $\hat{R}_{ur}$ is a predicted rating given by user u to restaurant r, R_{ur} is the actual rating given by user u to restaurant r and S is the total number of evaluation reviews.

7 RESULTS

For collaborative filtering recommendation, input only contains information about user id, restaurant id, and the rating, all data can fit in memory. However, for content-based recommendation system, all review data cannot fit in memory for SVD matrix computation. Since every time a user uses our recommendation engine, we can get his location, which can be used to filter review dataset. For final evaluation, we take reviews of all restaurants in Ontario, Canada, the total number of restaurants is 24,507 and the total number of reviews is 500,802.

Here are the result of running each implemented algorithm on Yelp dataset. As mentioned above, we use 5-fold cross validation and get RMSE and MAE of each recommendation systems. We also build one hybrid recommendation engine, which uses the result of both ALS system and content-based recommendation system by using ensemble technique. Meanwhile, we collected the average running time of each system when a user sends a request.

Method	RMSE	MAE
Naive Baseline	2.21	1.34
K-NN	1.83	1.19
ALS	1.55	1.02
Content-based Recommendation	1.43	0.85
Hybrid engine	1.37	0.79

Table 5: Evaluation results

As is shown in Table 5, naive baseline method has the worst performance, since it just takes average ratings of all users and all restaurants into account without considering other factors. K-NN methods performs slightly better than the baseline. Both ALS and content-based recommendation methods use matrix factorization, ALS uses rating information while content-based recommendation

uses reviews of each restaurant. Content-based recommendation has a better performance than ALS. Finally, hybrid recommendation engine performs better than both ALS and content-based recommendation since it uses ensemble method, which can help reduce the variance of different models.

Method	Average running time (min)
Naive Baseline	1.5
K-NN	2
ALS	10
Content-based Recommendation	8
Hybrid Engine	15

Table 6: Average running time

Since both ALS and content-based recommendation use matrix factorization method, which includes many iterations, both models spend much more time than other simpler methods. Content-based method also involves preprocessing and feature extraction steps, which can be done offline. Hence, there are still many opportunities for us to improve the speed of the hybrid recommendation engine.

8 CONCLUSION AND FUTURE WORK

In this project, we build a Hybrid Yelp Restaurant Recommendation Engine by two ways: the collaborative filtering, the content-based filtering. The former one generates recommendations according to the rating that each user gives to restaurants. K-NN model and ALS model are implemented. The comparison shows that while the K-NN model is easier to be implemented, ALS model is preferable when the rating matrix is sparse and the dataset has a considerable scale. In the content-based filtering, our engine output recommendations according to reviews. SVD technique is applied to the TF-IDF matrix to discover the similarity among different restaurants.

The evaluation of our engine shows that while both the collaborative filtering and the content-based filtering gives good predictions, the hybrid of these two ways leads to a better performance. From the running time perspective, although matrix factorization approaches are slower than algebra based models, it still can be accepted considering the dataset scale.

A task for our future work is to implement the LDA model that we mentioned in the introduction. It is a popular model for content-based filtering. This model allows us to learn the latent subtopics in review texts, which then can be used in various computations to produce a review analysis or weighted, personalized star rating. Assuming the LDA model is able to output the probabilities of topics for a review text, for example for a review: 60% Topic A, 40% Topic B. Then restaurants with similar review topics can be selected to be recommended to users.

Another task is researching on the Natural Language Processing. The relationship between review text and rating deserves to be analyzed. Moreover, it is reasonable to assume that the accuracy of recommendation can be improved according to the research on the most common positive and negative words used in our reviews.

REFERENCES

[1] David Heckerman John S. Breese and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *n Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence (UAI'98)*, 1998.

[2] Cai L Yao G. User-based and item-based collaborative filtering recommendation algorithms design. 2003,.

[3] Konstan J et al Sarwar B, Karypis G. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th international conference on World Wide Web. ACM*, pages 285–295, 2001.

[4] Pentreath N. *Machine Learning with Spark*. Packt Publishing Ltd, 2015.

[5] Chen K et al. Mikolov T, Sutskever I. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems.*, pages 3111–3119, 2013.

[6] Jordan M I. Blei D M, Ng A Y. Latent dirichlet allocation. *Journal of machine Learning research*, pages 993–1022, 3(Jan) 2003.

[7] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, San Jose, CA, 2012. USENIX. ISBN 978-931971-92-8. URL https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia.

[8] Koren Y. Bell R M. Scalable collaborative filtering with jointly derived neighborhood interpolation weights[c]. *Seventh IEEE International Conference on. IEEE*, pages 43–52, 2007.

[9] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[10] Guy Shani and Asela Gunawardana. Evaluating recommender systems. Technical report, November 2009. URL https://www.microsoft.com/en-us/research/publication/evaluating-recommender-systems/.