Books Recommendation System Based on Collaborative Filtering and Contented

Zhiyuan Wang
University of Southern California
Email: wang723@usc.edu

Zhiqing Chen
University of Southern California
Email: Zhiqingc@usc.edu

Zhiheng Luo
University of Southern California
Email: zhihengl@usc.edu

***Abstract*—The object of this project is to make a book recommendation system for users. We did some analysis of the goodreads-10k dataset. Then we use a hybrid recommendation approach with collaborative filtering followed by content-base approach to make a personalized recommendation for each user. The RMSE for the collaborative filtering is 0.9868.**

1. Introduction

We are going to implement a recommendation system based on goodreads-10k dataset. We will introduce our dataset, problems, algorithms and challenge firstly.

1.1. Data Set

The dataset is from Kaggle goodreads-10k. It has five files.

ratings.csv contains ratings sorted by time. Ratings go from one to five. Both book IDs and user IDs are contiguous. For books, they are 1-10000, for users, 1-53424.

to_read.csv provides IDs of the books marked "to read" by each user, as user_id, book_id pairs, sorted by time. There are close to a million pairs.

books.csv has metadata for each book (goodreads IDs, authors, title, average rating, etc.)

book_tags.csv contains tags/shelves/genres assigned by users to books. Tags in this file are represented by their IDs. They are sorted by goodreads_book_id ascending and count descending.

tags.csv translates tag IDs to names. [1]

We will only use ratings.csv, books.csv, book_tags.csv to make our recommendation system.

1.2. Problems

Online social network has become a very important part in our life. However, there are huge volumns of information on the Internet and it is important to find the useful information and deliver it to users. Therefore, many online media company like Amazon, Youtube, Douban provides personalized recommendations to their users using their recommendation system. The idea is to discover the relevant information between users or items.

In our project, we will build a book recommendation system to recommend books to users among the large volumes of books we have. We plan to use collaborative filtering to find similarities between users and then use content-based approach to find similarities between books. We will use this hybrid recommendation system to make recommendation.

1.3. Algorithms

1.3.1. Collaborative filtering. Collaborative filtering is one of the techniques used for making recommendation system.

The motivation for collaborative filtering comes from the idea that people often get the best recommendations from someone with tastes similar to themselves. Collaborative filtering encompasses techniques for matching people with similar interests and making recommendations on this basis.

Collaborative filtering algorithms often require (1) users' active participation, (2) an easy way to represent users' interests, and (3) algorithms that are able to match people with similar interests.

Typically, the workflow of a collaborative filtering system is:

1. A user expresses his or her preferences by rating items (e.g. books, movies or CDs) of the system. These ratings can be viewed as an approximate representation of

the user's interest in the corresponding domain.

2. The system matches this user's ratings against other users' and finds the people with most "similar" tastes.

3. With similar users, the system recommends items that the similar users have rated highly but not yet being rated by this user (presumably the absence of rating is often considered as the unfamiliarity of an item) [2]

1.3.2. Content-based. Contented based is one of method for recommendation system. This method will use the information of book tags to get features vector for book and user. Then recommendation will be made based on the similarity between user and book.

Generally, the work flow will be:

1, Build item profile based on features. The features can be binary or non-binary. We will use TF-IDF to get the features matrix.

2, Build user profile. User profile will be built based on the books user have rated.

3, Make recommendation. There are two ways to make recommendation. The first one is based on similarity calculating. The second one is regarding this problem as a classification problem. We will make recommendation based on similarity.

Content based result can not be evaluated numerically because we do not have the standard result of recommendation yet. We will analysis the result between collaborative filtering and contented based and why they are different.

1.4. Challenge

1.4.1. Memory. We have a very big dataset with 53424 users and 10000 books. The size of the rating array would be around 2GB, which is big for both memory and application like python or spark to deal with.

1.4.2. Time efficiency. We will have some very time costly matrix operations and the running time of our program may be huge.

2. Experiments

The Amazon EC2 t2.micro only has 1GB memory and this is not enough for us to build our recommendation system. So the environment we are using to build our recommendation system is Google VM instance n1-standard-4 which has 4 vCPUS and 15 GB memory

2.1. Data Set Analysis

Before we build the recommender system, we explored the dataset to find some interesting insight.

First, let us have a look at our dataset goodbooks-10k in kaggle, it contains two main files Ratings.csv and Books.csv.The Ratings.csv contains all users' ratings of the books(a total of about 6 million ratings, from 10000 books and 53424 users). The books.csv contains more information on the books such as author, year, title, etc.

user_id	book_id	rating
1	258	5
2	4081	4
2	260	5
2	9296	5
2	2318	3

Figure 1. Ratings.csv

original_publication_year	original_title	...	ratings_count
2008.0	The Hunger Games	...	4780653
1997.0	Harry Potter and the Philosopher's Stone	...	4602479
2005.0	Twilight	...	3866839
1960.0	To Kill a Mockingbird	...	3198671
1925.0	The Great Gatsby	...	2683664

Figure 2. Books.csv

2.1.1. Clean the Data. This is a real-life dataset, we need to do some cleaning job first. When exploring the data, we find that there are some multiple ratings with same users, same books and same ratings. Hence we need to remove these duplicate data. Finally, we remove 4487 duplicate ratings from around 6 millions ratings.

2.1.2. Distribution of Ratings. We can see that people tend to give positive ratings to books. Most of ratings are in range 3-5 and few ratings are in range 1-2.

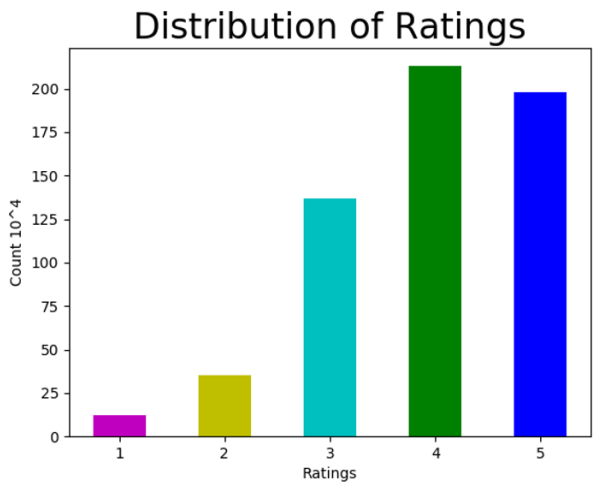

Figure 3. Distribution of Ratings

2.1.3. Number of Ratings per user. The result looks like the standard normal distribution. Most of people give about 100 ratings and people at least give 20 ratings, which will help us to build a precise recommender system.

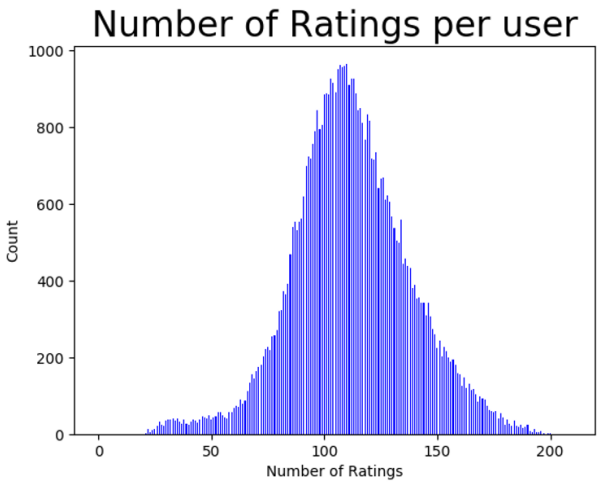

Figure 4. Number of Ratings per user

2.1.4. Distribution of mean user ratings. People have different tendencies to rate books. Some people love to give higher ratings to books. Some people only give 5 stars unless it is the perfect book for them, and give the low ratings like 1 or 2.

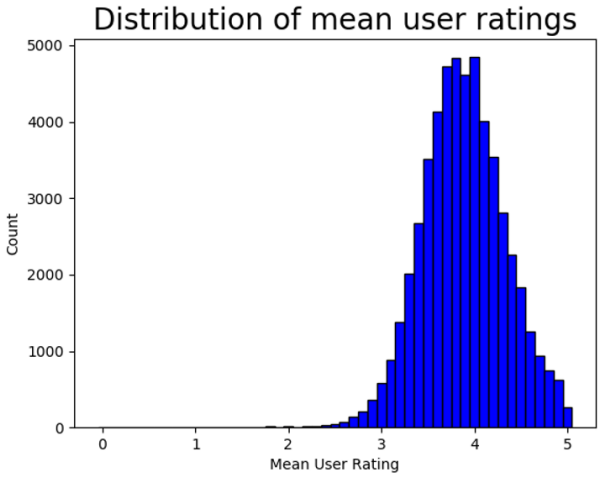

Figure 5. Distribution of mean user ratings

2.1.5. Number of Ratings per book. As we can see in the graph 1, the result is discrete. Some book have more than 3000 ratings, while most of books have hundreds of books. And we narrow the x-axis the number of ratings from 0 to 1000, we get the graph 2. We can see that most of books have about 200 ratings.

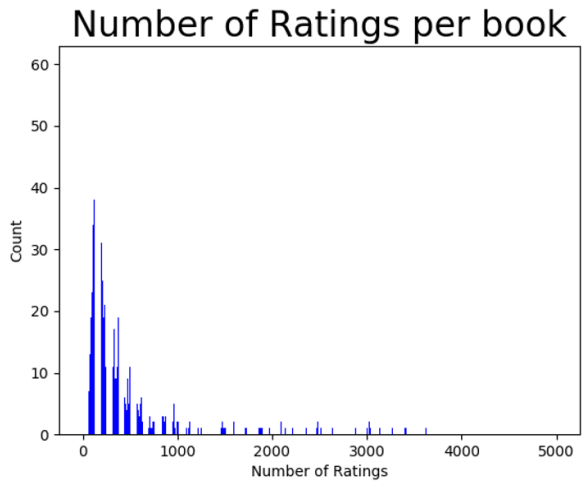

Figure 6. Number of Ratings per book 1

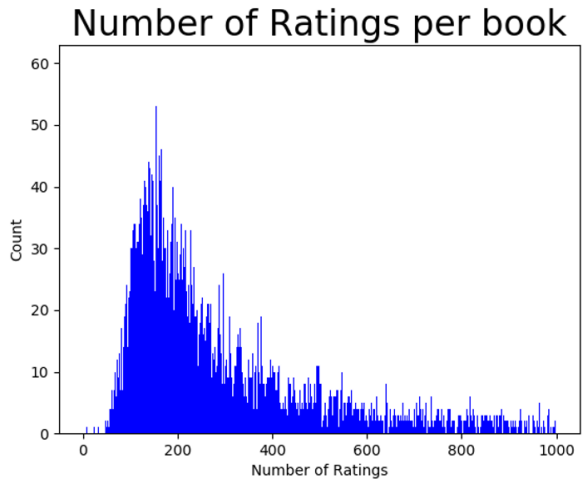

Figure 7. Number of Ratings per book 2

2.1.6. Distribution of mean book ratings. The most of book get the average rating 4 from the users.

2.1.7. Top 10 Rated books. In this part, we sort the result of the average ratings for books to get the top 10 rated books. These books are public popular to all the readers.

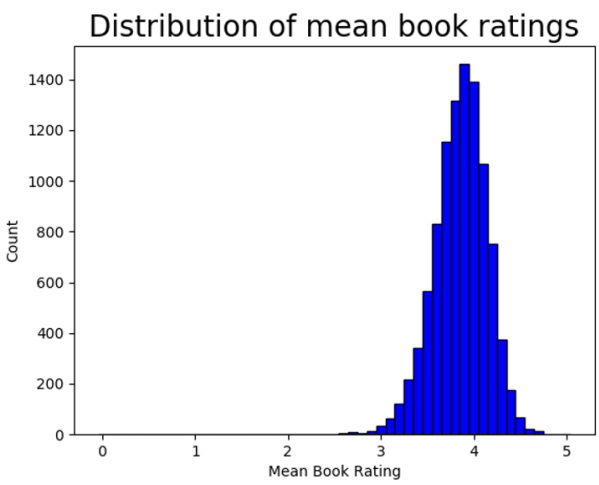

Figure 8. Distribution of mean book ratings

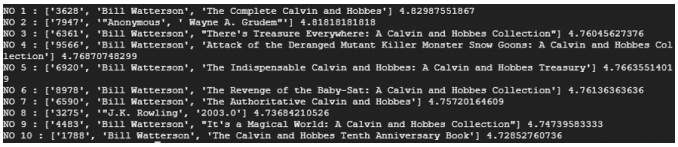

Figure 9. Top 10 Rated books

2.2. Collaborative Filtering

The algorithm we use in collaborative filtering approach is to use UV decomposition and alternate least square approach.

Figure 10 shows how we use UV decomposition. We decompose the big rating array into two small user and book array. The latent factor is usually much smaller than the number of the users and books to achieve memory efficiency. We keep updating user array and book array alternatively and finally we want the dot product of the user and book array to converge after iterations.

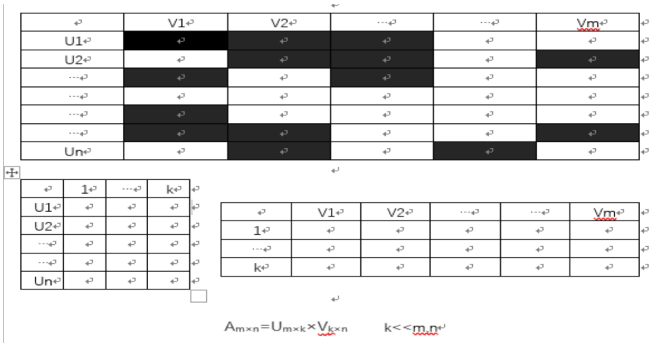

Figure 10. UV decomposition

2.2.1. Algorithm. The algorithm we use is UV decomposition with alternating least square approach. The updating rule is:

$$(A^T A + \lambda I)x = A^T b$$

where A is the user array or book array, λ is the regularization factor, x is the updated row or column of the book or user array, b is the corresponding row or column of the rating array.

We split the data into 6:3:1 for training data, validation data and testing data.

2.2.2. Evaluation. We use RMSE to show to performance of our collaborative filtering algorithm. RMSE is calculated by:

$$RMSE = \sum_{i,j}(R_{i,j} - B_i U_j)$$

Where i, j are entries that are only contained in training, validation and test set.

2.2.3. Memory Challenge. The size of the big rating array in our case is around 2GB, which is beyond the limit usage of 32-bit python and spark broadcast. So instead of using the big rating array directly, we store the ratings in the map when doing the ALS approach. That is, we only store the non-blank entry in the map which only cost around 100 MB memory and thus the memory efficiency would be much better. Later, we would only store the small user array and book array to calculate the ratings and thus the memory usage would be less.

2.2.4. Time efficiency. The running time of updating user and book array and creating row or column of rating array can be huge. We get around this by using the numpy library in python which is much more efficient than using the for-loop in python. Also we use spark to update array in parallel, which is more time efficient. The running time of all UV decomposition including training, validation and testing is around 30 minutes.

2.3. Contented Based Recommendation

After getting 100 candidates for each user from ALS algorithm, we apply contented based recommendation to find final 10 results for each user.

2.3.1. Item Profile. We firstly build the item profile by the file of book tags.

In this file, there are three columns. The first one is book_id, the second one is tag_id and the third one is count. Count means how much people give tag to this book.

goodreads_book_id	tag_id	count
1	30574	167697
1	11305	37174
1	11557	34173
1	8717	12986
1	33114	12716
1	11743	9954
1	14017	7169
1	5207	6221
1	22743	4974
1	32989	4364
1	27199	3857

Figure 11. Book_tags

We build the no-binary features by Tf*Idf. Tf-Idf means Term FrequencyInverse Data Frequency.
TF is the frequency of the tags in each books in this dataset. It is a ratio of number of times. It will be calculated by this formula.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

IDF could calculate the weight of rare tags in all books. The rare tags will have a high IDF score. It will be calculated by this formula.

$$idf(w) = \log\left(\frac{N}{df_t}\right)$$

Combine TF and IDF, we could get the TF-IDF score for a tag. TF-IDF will be calculated by this formula.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

	Feature 3054	Feature 162	Feature 53
Book1	0.016	0.24	0.12
Book2	0	0.015	0.04

Figure 12. TFIDF of books

2.3.2. User Profile. After getting the item profile, we need build user profile based on it. Because user gives rate to every movie for our dataset, we will use weighted features based on ratings. It means that we will use the sum of feature vector of movie times rating divide sum of rating. The formula will be:

$$u = \frac{\sum r_i * F_i}{\sum r_i}$$

	Feature 3054	Feature 162	Feature 53
usr1	0.65	0.33	0.26
usr2	0	0.41	0.06

Figure 13. Features of Users

2.3.3. Similarity. We can calculate the similarity of user book by many ways. We choose to use Cosine Similarity as our method. Cosine Similarity is:

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

We will get a similarity matrix after applying this method.

	Book 1	Book 2	Book 3
usr1	1.51	0.78	0.62
usr2	0.89	1.23	0.11

Figure 14. Similarity Matrix

2.3.4. Result. After calculating the similarity of each user and candidate, we will rank the score in decreasing. For every user, we will only store 10 highest similarity books. We output the result as a file and write a program to look up it, which could avoid costing calculating every time we need a result. For those books without name, we will output id of book instead.

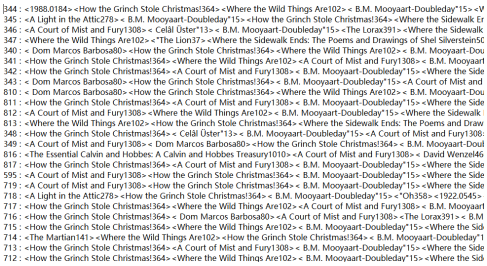

Figure 15. Result File

Figure 16. Look up Program

3. Evaluation Results

3.1. Collaborative Filtering

3.1.1. Result. We use validation set to get our best latent factor and regularization factor.

According to Figure 17, with the increase of latent factor, the RMSE goes down first and then goes up. The best latent factor is 12.

In Figure 18, with the increase of lambda, the RMSE goes up. This is because if lambda is too big, we tend to underfit our data. The best regularization factor is 0.1.

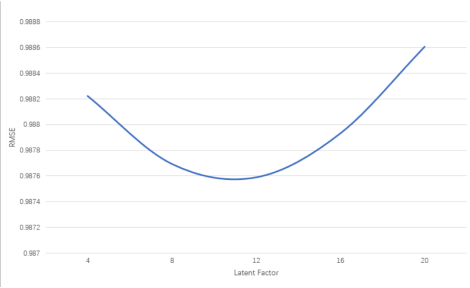

Figure 17. RMSE and Latent Factor

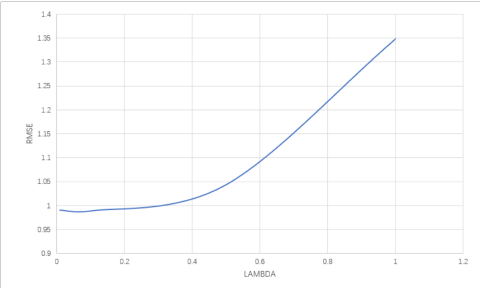

Figure 18. RMSE and Lambda

The final test error is 0.9868

3.1.2. MLlib. We use spark MLlib to get an idea of the performance of our recommendation system. The RMSE of it is 0.8322. Compared with MLlib, out recommendation system is OK to make book recommendation systems. MLlib use some hybrid algorithms to make a better performance when making their model compared with our naive UV decomposition model. We will use content-based approach based on the result of our collaborative approach to make a better book recommendation for users.

3.1.3. Output. We get our final user array and book array. For each user we store their top 100 rating books in the file as model. The final content-based approach will be carried out based on this file.

In the final output, we observe that the ratings of the top 100 books of some users are very close. We think this is because those books are very popular and users tend to give them high ratings. For example, both Harry Porter and Twilight are very popular and are rated by a large volume of users with positive ratings. So when they are recommended to new users based on collaborative filtering, they tend to have very close and high ratings.

3.2. Contented Based Recommendation

344 : 1,94,77,14,62,75,9,40,16,89,
345 : 86,15,95,63,10,30,41,76,90,28,
346 : 48,47,16,11,64,29,31,91,18,42,
347 : 80,91,65,17,32,12,30,78,43,19,
340 : 98,90,73,10,58,5,71,36,23,12,
341 : 91,74,43,11,59,6,37,72,13,86,
342 : 92,44,12,60,7,38,25,73,87,27,
343 : 2,93,13,45,8,74,28,39,26,88,
810 : 73,65,48,84,33,79,46,11,97,86,
811 : 66,18,85,34,80,12,1,98,87,61,
812 : 19,50,86,35,81,2,13,48,88,99,
813 : 51,68,36,87,82,3,14,49,89,63,
348 : 98,49,18,50,79,13,33,44,93,31,
349 : 51,8,99,19,67,14,80,45,94,21,
816 : 1,23,95,39,90,52,17,85,92,66,
817 : 72,24,91,40,86,7,18,53,5,93,
595 : 99,48,67,16,82,62,93,43,29,80,
719 : 25,73,92,41,8,87,19,68,6,54,
718 : 63,72,91,15,18,67,53,86,40,5,

Figure 19. The corresponding rank for final result in ALS result

3.2.1. Difference between results. From figure 19 we can find that the result of contented based is different from the result of ALS. For example, user 340 is recommended by 98,90,73,10,5,71,36,23,12 as ten highest similarity books, which contains many books whose score is low at ALS.

3.2.2. Reason. We conclude there are three reasons for this situation.

First, The score of 100 highest books by ALS are close to each other for, which means the users may have similar ratings for these books.

Second, Contented based algorithm use additional data source from dataset, which provide additional information. Contented based algorithm make use of the tags of books

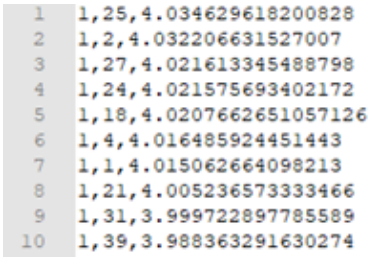

Figure 20. Top 10 scores by ALS for user 1

and how many people label these tags to books. ALS does not make use of this information, so their results are different.

Third, For contented based model, we assume that people tend to like books whose tags are similar. However, this assumption may not hold for all tags. For example, a book may have tag of publishing year, which may not influence people's choice remarkably. Unfortunately, the dataset do not provide information about what they are for every tags. We can not analysis whether this opinion holds.

4. Conclusion

We successfully make a book recommendation system for our users based on goodreads-10k dataset.

4.1. Application

Recommender systems are utilized in a variety of areas, and are most commonly recognized as playlist generators for video and music services like Netflix, YouTube and Spotify, product recommenders for services such as Amazon, or content recommenders for social media platforms such as Facebook and Twitter. These systems can operate using a single input, like music, or multiple inputs within and across platforms like news, books, and search queries. There are also popular recommender systems for specific topics like restaurants and online dating. Recommender systems have been developed to explore research articles and experts, collaborators, financial services, and life insurance. [3]

In our project, the book recommendation system can be used to make personalized recommendation for users where there are a large volume of books for users to read. The recommendation system is very easy to use. Users only need to type in their user ID to get a personalized recommendation. Some very useful functions can be very easily implemented based on our recommendation system. For example, if we want the model to have some interactions with users, we can keep track of the user actions when they see the books recommended by our system. Then we can make a better recommendation system based on user actions.

References

[1] goodbooks-10k https://github.com/zygmuntz/goodbooks-10k

[2] Collaborative filtering Wikipedia https://en.wikipedia.org/wiki/Collaborative_filtering

[3] Recommendation system https://en.wikipedia.org/wiki/Recommender_system

[4] Recommendation system, University of Southern California, Inf553 Data Mining