CMSC 12300 Final Project Report
Group Name: 0errors0warnings
Team members: Boyang Qu, Ruixi Li, Tianxin Zheng, Haowen Shang
06/09/2019

Item-based and User-based Movie Recommendation using MapReduce and Hadoop

1. Dataset

The datasets we employed is the large latest movie-rating datasets from MovieLens, which can be downloaded from http://grouplens.org/datasets/movielens/. It is 0.76GB large, containing 27,753,444 ratings from 283,228 users on 58,098 movies. The dataset was collected by GroupLens Research between January 09, 1995 and September 26, 2018, and users were selected randomly. For privacy reason, the users and movies are represented by unique Id numbers.

The primary dataset used for our mapreduce algorithm is the movie-ratings dataset contains all rating data, with each line representing one rating by one user on one movie. There are 27,753,444 lines in this dataset. Information on each line is in the format of userId, movieId, rating, timestamp (unused). Ratings are provided on a 5-star scale with half-start increment. The dataset is ordered by userId and then movieId. In order to associate movieId entries in movie-rating dataset, we also used the movie dataset with each line in the format of movieId, title, genres (unused). There are 58,098 lines in this dataset.

The dataset consists diverse movies and users joining IMDB at different times. There is a huge difference between number of ratings on popular movies and those on those receiving little attention, while there is also huge difference between number of movies rated by active users and those by inactive users. The diversity assures that our model has a broad application range. We believe the dataset has a scale large enough for MapReduce exploration while not too large for efficient manipulation.

2. Hypotheses

Our project goal is to construct a recommendation system which could predict the rating that the user would give to a movie. We used two algorithms in our recommendation, the item-based algorithm and the user-based algorithm.

For the item-based approach, our algorithm is based on measuring the similarity score between pairs of movies. The higher the score, the more similar the two movies would be, and the higher the likelihood that the user would appreciate the recommendation. There are in total four steps in our project. Firstly, for each pair of movies X and Y in our dataset, we found out all the people who rated both X and Y. Secondly, we used these ratings to form a Movie X vector and a Movie Y vector. Thirdly, we calculated the correlation between those two vectors. Lastly, for a specific user, we recommended the movies most correlated with all watched movies by him or her based on weighted correlation scores.

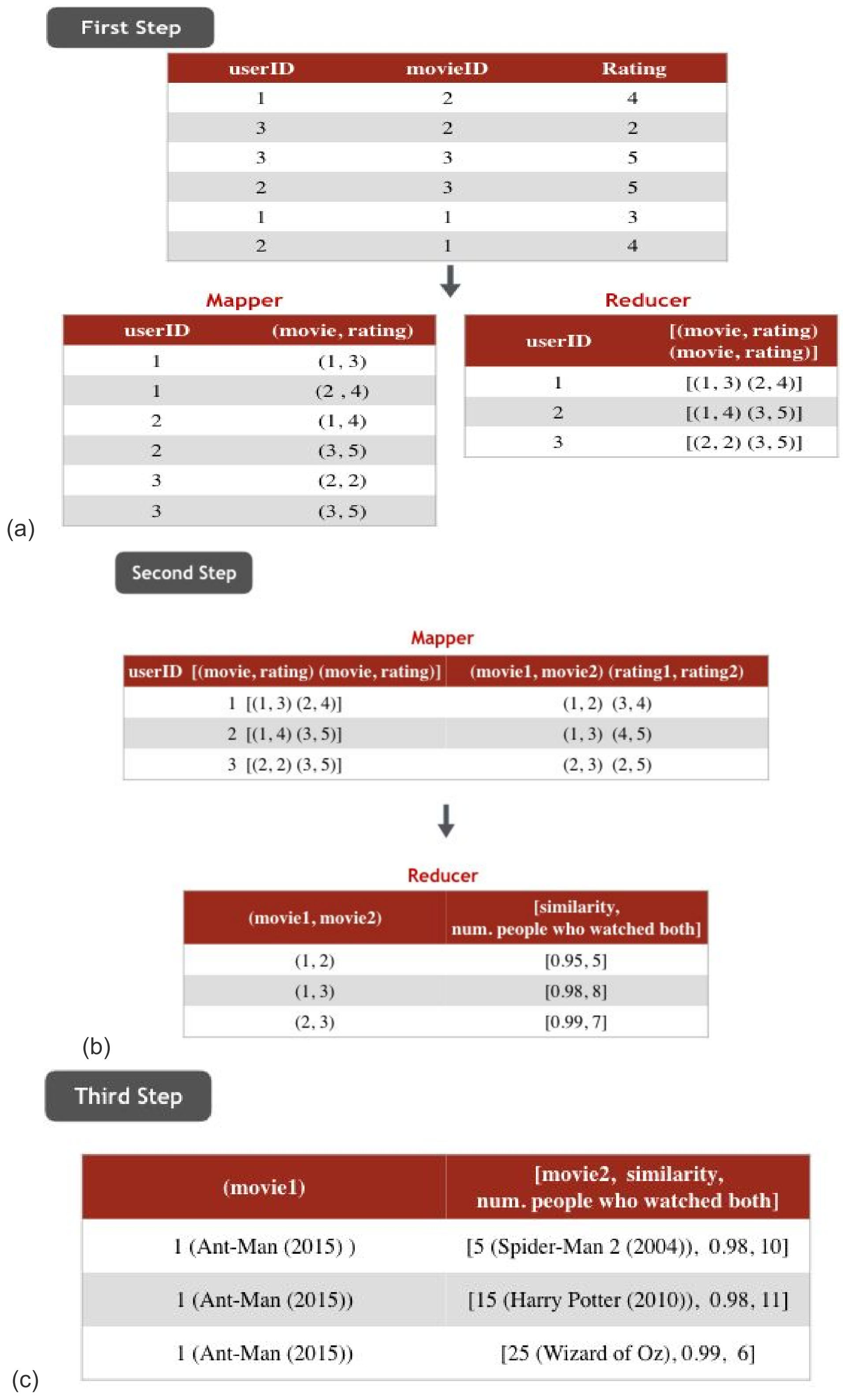
For the user-based algorithm, the way is to calculate the similarity score of pairs of users. The higher the score, the more similar the two users would be, and the higher the likelihood that the user would appreciate the movies liked by another user who were similar to him/her. There were in total four steps in our project. Firstly, for each pair of users A and B in our dataset, we found out all the common movies they rated. Secondly, we used these movies to form a Rating X vector and a Rating Y vector. Thirdly, we calculated the correlation between those two vectors and get the similarity matrix. Lastly, when user A watched a movie, we recommended the movies highly-rated by the user who had the highest similarity score.

After the score calculation, we tested our result by cross-validation. Firstly, we split our data into testing and training sets. Secondly, we put our testing set aside, run our algorithms on the training set and got the recommendation file. Thirdly, we compared the recommended movies with the movies rated by this user in the testing set and used RMSE (root mean square error) for accuracy evaluation.

3. Algorithms

3.1. Item-based Algorithm

Our first algorithm is item-based collaborative filtering. We used MapReduce approach to find similar movies based on movie ratings by different users.



First Step

userID	movieID	Rating
1	2	4
3	2	2
3	3	5
2	3	5
1	1	3
2	1	4

Mapper

userID	(movie, rating)
1	(1, 3)
1	(2 , 4)
2	(1, 4)
2	(3, 5)
3	(2, 2)
3	(3, 5)

Reducer

userID	[(movie, rating) (movie, rating)]
1	[(1, 3) (2, 4)]
2	[(1, 4) (3, 5)]
3	[(2, 2) (3, 5)]

(a)

Second Step

Mapper

userID [(movie, rating) (movie, rating)]	(movie1, movie2) (rating1, rating2)
1 [(1, 3) (2, 4)]	(1, 2) (3, 4)
2 [(1, 4) (3, 5)]	(1, 3) (4, 5)
3 [(2, 2) (3, 5)]	(2, 3) (2, 5)

Reducer

(movie1, movie2)	[similarity, num. people who watched both]
(1, 2)	[0.95, 5]
(1, 3)	[0.98, 8]
(2, 3)	[0.99, 7]

(b)

Third Step

(movie1)	[movie2, similarity, num. people who watched both]
1 (Ant-Man (2015))	[5 (Spider-Man 2 (2004)), 0.98, 10]
1 (Ant-Man (2015))	[15 (Harry Potter (2010)), 0.98, 11]
1 (Ant-Man (2015))	[25 (Wizard of Oz), 0.99, 6]

(c)

Figure 1. Example for item-based algorithm steps. (a) Step 1. (b) Step 2. (c) Step 3.

Firstly, to find all movies and their ratings watched by each person, we employed a mapper to extract user and (movie, rating) pairs and use a reducer to group all (movie, rating) pairs by user (Figure 1a.).

Secondly, to get every pair of movies watched by the same person and the corresponding rating pairs, we used a mapper to get key-value pairs, which looked like: (movie1, movie2) - (rating1, rating2). After that, we used a reducer to compute rating-based similarity between each movie pair and got the similarity scores. The yielded output is in the format: (movie1, movie2) – (similarity scores, number of people who votes both). We used Cosine similarity as our measure for similarity score, which is defined as cosine of angle between two movies. If the score is 0, the two movies are not similar at all. If the score is 1, there's high similarity between the two movies.
Thirdly, we obtained the output with movies sorted by names together with a list of similar movies. The output was written into a csv file with each line in the format of: "Movie_name" ["Similar_movie", "Similar_score (From 0 to 1)", "Number of people who votes both"].

Finally, with the similarity scores between movies pairs obtained from MapReduce, we calculated a weighted similarity score as the predicted rating for each unwatched movie by a specific user. For a user i who have given ratings $r_1, r_2, r_3, \ldots, r_n$ on n movies, the expected rating for an unrated movie j with similarity $s_1, s_2, s_3, \ldots, s_n$ with these movies is:

$$\frac{1}{n} * \sum_{x=1}^{n} (r_x * s_x) .$$

As for the runtime of our item-based algorithm, it took Google Cloud 17 hours to run the MapReduce task. We tried to run it under a local environment, it turned out we couldn't get it done within 30 hours. The extremely long runtime is because of the large size of our dataset.

3.2. User-based Algorithm

We then moved to an user-based collaborative filtering algorithm. This algorithm calculated the similarity between two users using Cosine similarity. Again, we used MapReduce to find similar users based on movie ratings.

First, to find all movies and users who watched the movie and rated it, we used a mapper to extract movie and (user, rating) pair and a reducer to group all (user, rating) pairs by the movie (Figure 2a).

Second, to get every pair of users who watched the same movie and the corresponding rating pairs, we employed a mapper to get key-value pair, which looked like: (user1, user2) - (rating1, rating2). Then, we used a reducer to compute the rating-based similarity between each user pair and got its similarity scores (Figure 2b).

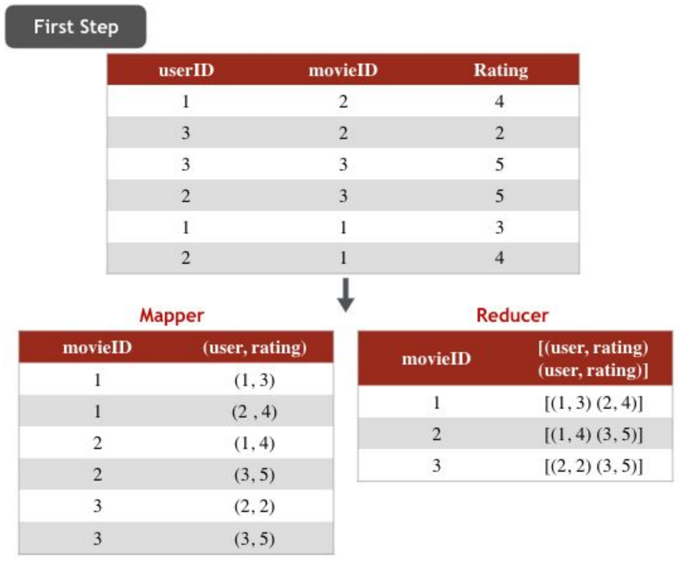
Third, to measure the similarities between each user pair, we used a reducer to compute the rating-based similarity between each user pair and got its similarity scores (Figure 2c). The yielded output was in the format of: user1, user2, similarity scores, (number of movies they both watched).

After completing the MapReduce task, we made a prediction for each movie for each user by calculating a simple weighted average of the ratings provided by the k most similar users. Specifically, we only included similar users who rated this movie. We used the following formula:

$$\text{Prediction} = \frac{\sum (W_{i,1} * \text{rating}_{i,\text{item}})}{\sum (W_{i,1})},$$

where $W_{i,1}$ is the similarity of user i with the k (we will choose 10 here) most similar users. We applied this prediction algorithm to all unrated movies by each user.

As for the runtime of user-based algorithm, we shorten it from more than 40 hours to 15 hours on Google Cloud using mapreduce approach. Again, we tried to run it in a local environment, but it did not finish after 30 hours.


First Step

userID	movieID	Rating
1	2	4
3	2	2
3	3	5
2	3	5
1	1	3
2	1	4

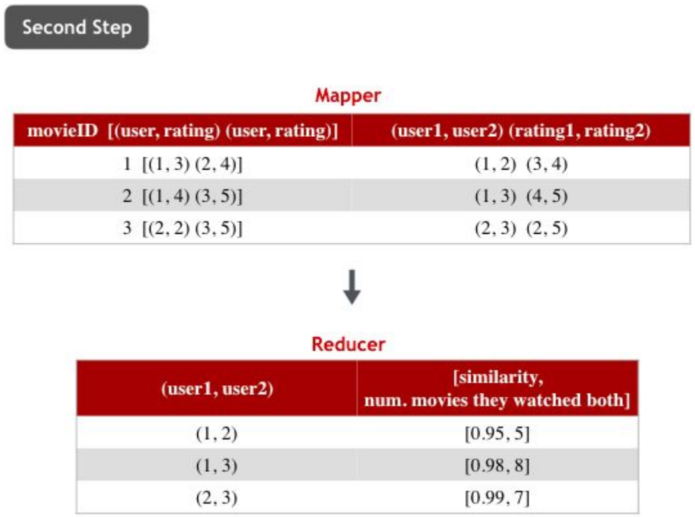
Mapper

movieID	(user, rating)
1	(1, 3)
1	(2 , 4)
2	(1, 4)
2	(3, 5)
3	(2, 2)
3	(3, 5)

Reducer

movieID	[(user, rating) (user, rating)]
1	[(1, 3) (2, 4)]
2	[(1, 4) (3, 5)]
3	[(2, 2) (3, 5)]

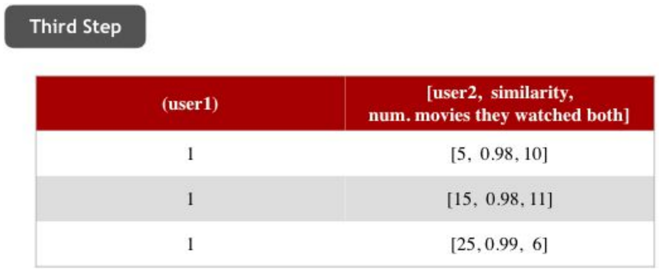
(a)


Second Step
Mapper
Reducer

movieID [(user, rating) (user, rating)]	(user1, user2) (rating1, rating2)
1 [(1, 3) (2, 4)]	(1, 2) (3, 4)
2 [(1, 4) (3, 5)]	(1, 3) (4, 5)
3 [(2, 2) (3, 5)]	(2, 3) (2, 5)

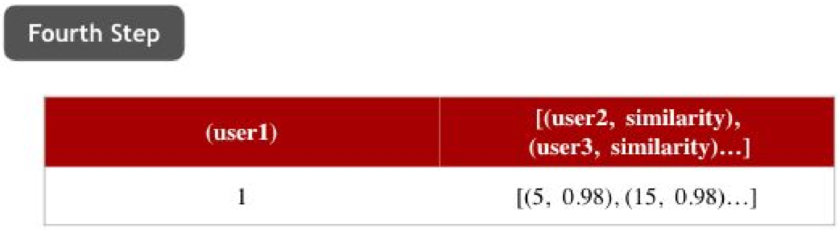
(user1, user2)	[similarity, num. movies they watched both]
(1, 2)	[0.95, 5]
(1, 3)	[0.98, 8]
(2, 3)	[0.99, 7]

(b)



(user1)	[user2, similarity, num. movies they watched both]
1	[5, 0.98, 10]
1	[15, 0.98, 11]
1	[25, 0.99, 6]

(c)


Fourth Step

(user1)	[(user2, similarity), (user3, similarity)…]
1	[(5, 0.98), (15, 0.98)…]

(d)

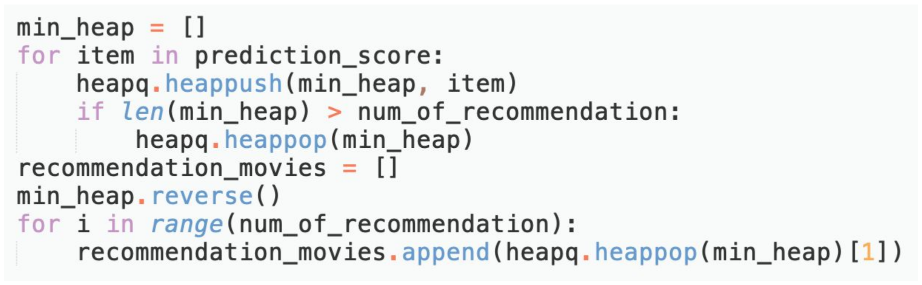
Figure 2. Example for user-based algorithm steps. (a) Step 1. (b) Step 2. (c) Step 3. (d) Step 4.

<u>4. Challenges</u>

We encountered and solved many challenges in our project, which could be divided into two categories: those related to using Google Cloud and those encountered in implementing the two algorithms.

First of all, we were not familiar with Google Cloud. Thus, it took us very long time to implement the task on Cloud. Since we wrote our code separately, we have many .py files in the beginning, believing these files would import the function in other .py files on Cloud just like the way the local computer did. However, with the help of our Professor, we found that we should write our code in a single .py file and sent it to the Cloud. In addition, running MapReduce on the Cloud required a configuration file, about which we had totally no idea. After checking the mrjob documentation and asking the professor, we added the configuration file to our project.

In the process of implementing our algorithm, we also met some challenges. The first challenge in implementing algorithms is that we have to decompose the conceptual algorithm into actual steps in MapReduce. In addition, we calculated five similarity scores between different movies or users, but the result was a little bit strange. To make better recommendations, we chose the score calculated by Cosine method, which we believed more credible than other similarity scores. For user-based approach, we implemented two versions. In the older version, we didn't use MapReduce so the computation work was quite heavy. We tried to improve efficiency by using priority queue and filtering out meaningless values (Figure 3). However, We still spent a whole night to run the file without completing the procedure. Finally, we decided to introduce MapReduce to our user-based approach. By doing so, we shortened our runtime to 40 minutes.



```
min_heap = []
for item in prediction_score:
    heapq.heappush(min_heap, item)
    if len(min_heap) > num_of_recommendation:
        heapq.heappop(min_heap)
recommendation_movies = []
min_heap.reverse()
for i in range(num_of_recommendation):
    recommendation_movies.append(heapq.heappop(min_heap)[1])
```

Figure 3. Code snippet of implementing a priority queue in the first version of user-based algorithm.

5. Results

5.1. Cross-Validation

In order to assess the accuracy of our model, we performed cross-validation of both item-based (movie-based) and user-based models. More specifically, a train/test split was done, and then the root mean square error (RMSE) was calculated for the predicted ratings and actual ratings.

If we want to recommend movies to a user, then we must learn something about past movie ratings by the user. Therefore, a small portion of users was selected as the testing users and a small portion of the movie rated by these users was finally selected as the testing movies in the testing dataset. 1/10 of the users were selected randomly as the testing users, and then ¼ of the movies watched by these users were again selected randomly into the testing dataset. Movie ratings by the remaining users and movie ratings by the testing users on the remaining movies were used as the training dataset.

After running the two algorithms on their respective training dataset, both item-based approach and user-based approach yielded predicted testing movie ratings by the testing users. Then for each approach, we calculated the root mean square error to measure the model accuracy. The root mean square error is shown below in Table 1. The RMSE values suggest that with a large dataset, the error is relatively stable at less than 0.5 stars on a 5 star rating-scale. We decided that both the item-based model and the user-based model were effective in making accurate movie recommendation for users, and the item-based model was slightly better with lower RMSE values in predicted ratings.

	Item-based approach	User-based approach
Trial 1	0.435	0.532
Trial 2	0.256	0.253
Trial 3	0.512	0.643
Trial 4	0.336	0.452
Trial 5	0.214	0.353

Table 1. Root mean square error of both approaches over five trials.

5.2. Make Recommendation

Both item-based and user-based approach yielded predicted ratings for unwatched movies by the user. Furthermore, we validated that the predicted ratings from both approaches were relatively accurate. Thus, we could conveniently recommend to a user the movies with the highest expected ratings from him/her. In conclusion, our models were able to generate accurate recommendations to users based on either movie similarities or user behaviors.