Contend-Based Movie Recommendation System

Ying Cai & Chong Meng

Abstract: Our project constructs a contend-based movie recommendation system which can uncover potential audiences for a movie for the purpose of advertisement and can recommend movies to the customer based on his preference. The model we use is the KNN model. The Cosine similarity is used as the metric of distance between movies. Movie popularity is also included in our model.

I. Introduction

With the accumulation of the online records, currently abundant datasets are established for all kinds of activities such as shopping and watching movies online. These large datasets give us the possibility to analyze these activities and uncover the important features, which then can further help us to improve the services of online merchant.

Our recommendation system is a content based system[1]. After we analyze the similarity between movies based on movie content, we try to find the preference of a customer based on the analysis of his history watching list, then generate a recommendation list of new movies which may be preferred by this customer. This recommendation will help and encourage the customer to choose a movie to watch. This accurate recommendation will win customers' loyalty since they get a wonderful experience in your services. In addition to recommend movies to user, we can also aim at a specific movie and find the potential customers who are more likely to watch this movie. This recommendation will dramatically improve the effect of advertisement for a new movie and reduce its cost, since it decreases the size of the targeted customers.

The main goal of our project is to build such recommendation system based on MovieLens dataset. The version of MovieLens dataset we are working with (265MB) contains about 27,000,000 ratings applied to 58,000 movies by 280,000 users. It also contains information such as each movie's genres which is very important for calculating the similarity between two movies.

This report consists of 5 parts. The first part explains our design and the method we used in this project. Part 2 demonstrates the procedure we preprocess the data to make the data more valuable. The third part is the discussion about our results of recommendation. In the section of conclusion, we discuss some aspects which could be improved and future work. The last part we show how to run our code to get the results we used in this report.

II. Methods and Evaluation Metrics

Our movie recommender is an item-based recommendation system. It has 2 parts. The first part, we recommend potential watchers to a movie for advertisement purpose. The second part, we recommend movies to a user.

The similarity between movies is decided by their contents, more specifically, their genres. Cosine similarity is employed here. It is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Following is the function. The i and j are 2 movies, k represents a genre. $d(i)$ and $d(j)$ are the degree of movie i and j, in other words, the number of genre tags that they have respectively. We computed the Cosine similarity for each pair of movies and keep them in a matrix.

$$\sigma_{ij} = \cos\theta = \frac{\sum_k A_{ik} A_{kj}}{\sqrt{\sum_k A_{ik}^2}\sqrt{\sum_k A_{kj}^2}} = \frac{n_{ij}}{\sqrt{d(i)d(j)}} \in [0,1]$$

[2]

1) Part 1: potential-customer recommendation.

This recommendation list is generated by the KNN (k nearest neighbors) model[3]. The Cosine similarity is used as the measurement of distance between movies. For a chosen movie, we choose k other movies which are the most similar (closest) to it. The customers, who have watched those k movies, may be more likely to watch this movie. All those customers are put in the list and recommended.

2) Part 2: recommend movies to a customer.

To recommend movies to a user, we take the rating of users and the popularity of movies into our consideration. The customer's watching history is ranked according to the ratings given by this customer. We choose the movies which have a high rating, and find the similar movies using the same method we use for Part 1. Those movies are put in a list and ranked according to their popularity. Popularity is measured by the number of watchers that each movie has. The more popular movies in the list are chosen and recommended to the customer.

Our results are evaluated by 2 evaluation metrics, precision and recall. Precision is the fraction of relevant instances among the retrieved instances[4]. Recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances[4]. In the experiment of recommending potential customers, precision evaluates that how many customers that are in our recommendation list and really watch this movie, and recall evaluates that how many customers who really watch this movie are included in our list. In the experiment of recommending movies to a user, the precision evaluates that how many movies that are in our recommendation list and really watched by the customer, and the recall counts that how many movies which on in the customer's real-watch list and also in our recommendations.

III. Data exploration and Preprocessing

The MovieLens dataset[5] contains some extreme cases. Some movies have very few watchers, and some customers have just watched quite few movies. The few records it has, the more difficult to figure out the similarity or the preference. And sometimes, these extreme cases will mislead the recommendation. When we randomly split the data set into training and test subsets, these cases may cause another problem. The customers who has watched very few movies may have all their records in either one set. Therefore, those extreme samples are not as valuable as other cases. Considering that they would introduce a lot of extra computation cost, we eliminate extreme cases for both movies and users.

We remove movies watched by less than 25 audience and users who watched less than 25 movies out from the dataset. Before the data processing, we have 27,753,444 ratings applied to 58,098 movies by 283,228 users. After refining the data, we have 25,876,299 ratings applied to 16,655 movies by 153,524 users. This shows that about 90% of the rating records are given by nearly a half of audiences and applied to about one fourth of all movies. So, our data preprocessing eliminates large amount inactive users and movies with a slightly decreasing of whole rating record.

After the refinement, we explore the data set. Fig. 1 is the word cloud of movies' title, with the size of font represent the frequency. The word cloud of movies' genres is shown in fig 2, which tells us the most popular genres of movie are 'Comedy' and 'Drama'.

Fig 1. word cloud of movie title

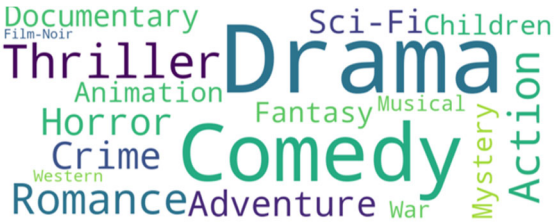

Fig 2. word cloud of movie genres

The distribution of ratings is depicted in fig 3, it was normalized by 3.5 which is mean value of all ratings. We can see that most users are very general to give a high rating.

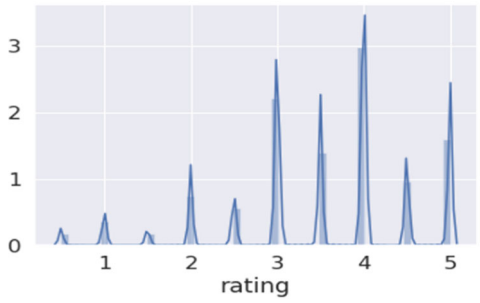

Fig 3. User rating distribution

For movies' popularity, we count the number of watchers that each movie has. Fig 4 shows the distribution. For example, there are 277 movies watched by 27 users and 200 movies watched by 30 users. We also count the number of movies that each customer has watched. Fig 5 shows the distribution. Our result indicates that about 2,660 users have watched 26 movies and 2,500 users have watched 27 movies. We don't include all values in the figures since the tail is very long.

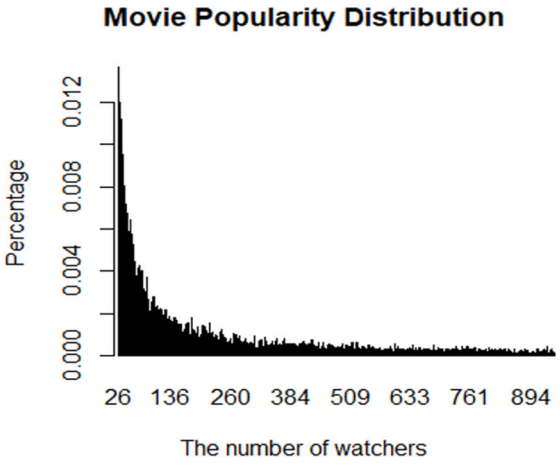

Fig 4. Movies' popularity

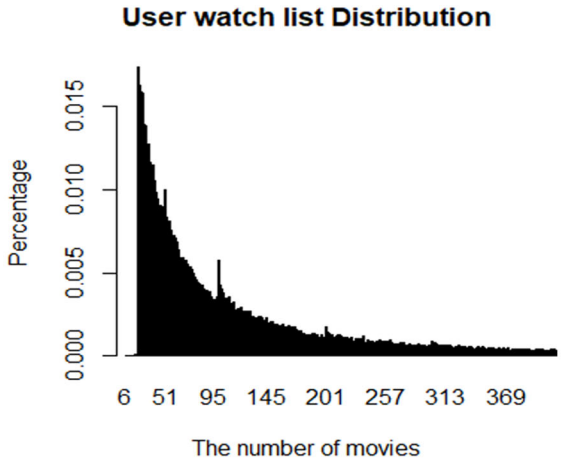

Fig 5. User watch history distribution

In order to test our recommendation precision and recall rates, we split our data to two parts, for test and training respectively. We take 20% from the total rating records randomly as the test cases. Our test set has 5,175,260 records applied to 16,653 movies by 153,488 users. The remaining 80%, which contains 20,701,039 ratings applied to 16,655 movies by 153,524 users, is our training set.

IV. Experimental Setup and Result Analysis

1) Part 1: Uncover potential watchers for a movie

2 parameters are important for this model. First is the value of K. How many neighbors that we choose to generate the list? Second is the value of the Cosine similarity. How large is it that we think they are "similar" enough to be put in the list?

Because there are only 20 genre tags for all movies, and a lot of movies have only 1 genre tag, many movies have the similarity equaling to 1 with a lot of other movies. So simply ranking the similarity and choosing the top movies does not apply to this case. We use the K to control the least length of the list and adjust the similarity threshold. For example, if we set K=1, we firstly put all movies whose similarity with this movie is greater than 0.99 into the list. If the length of the list is now 0, we decrease required similarity threshold by 0.1, put all movies whose similarity with this movie is greater than 0.89 into the list. The step is stopped until the list has at least 1 movie. All users who have watched the movies that are in the list will be put in the recommendation list as the potential audiences. In order to find a better K and similarity decreasing interval, we set K equals to 1, 3, 5 or 10, interval equals to 0.1 or 0.02. All movies are used to test models and tune parameters. Results are shown in Fig 6.

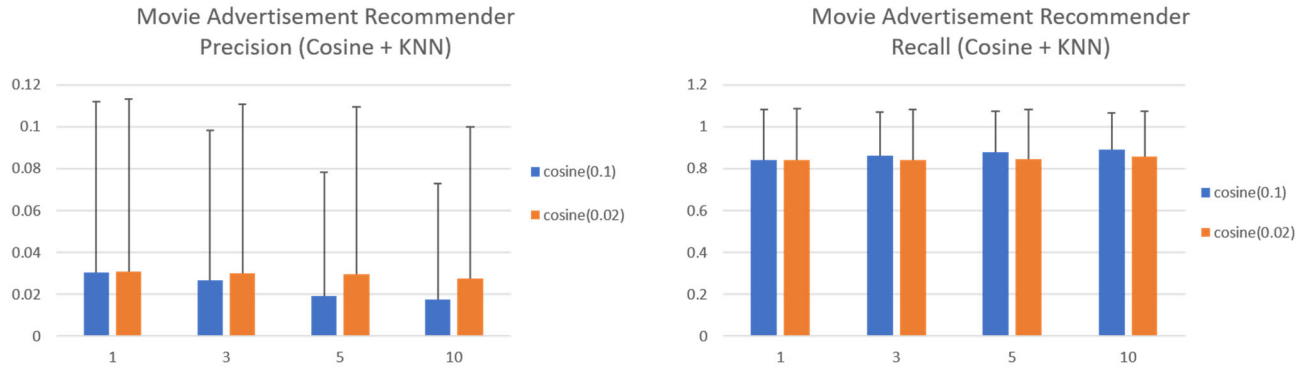

Fig 6: Uncover potential audiences

All combinations have a high recall, more than 80%, but a poor precision. For the Cosine similarity threshold decreasing interval, 0.02 is better than 0.1 obviously. K=1 or 3 is better than 5 or 10. Considering that the major purpose of this experiment is finding potential audiences, the poor precision is not so unacceptable. Advertiser may prefer a higher recall. Before we have known that a lot of movies have quite few watchers. If it is similar to some popular movies which has a long customer list, the recommendation list would be very long. This is why the precision is low and recall is large.

2) Part 2: Recommend movies to a user

Besides the K and Cosine similarity parameters, we include the user preference and movie popularity in this model, so we have 2 more parameters to tune, preference threshold and popularity threshold. We choose 2 values for the preference threshold. For the user in test, we know his/her max and mean ratings. The movies to which the use gives a rating higher than mean or (mean+(max-mean)/2) are selected. Then we use the same methods used in Part 1 to find the movies which are similar to those selected movies. All similar movies are ranked by their popularity. The top 50, 100, or 150 movies are selected from this list and recommended to the user. In Fig 7, we set the Cosine similarity decreasing threshold equals to 0.02, the preference threshold equals to or (mean+(max-mean)/2), K = 1, 3, 5, and 10. The recommendation list has 50, 100 or 150 movies, respectively. The result shows that the longer the list is, the higher recall and lower precision it would have. The K values do not affect results significantly.

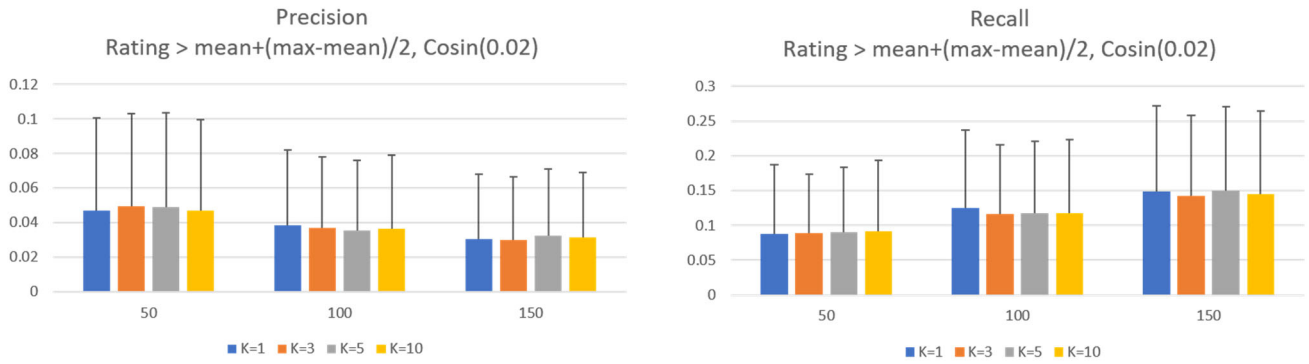

Fig 7: Recommend movies to the customer

Then we set K = 3, the Cosine similarity decreasing threshold equals to 0.1, and compare the impact of preference threshold in Fig 8. The recall increases and the precision decreases along with the increasing of recommendation list. The lower preference value has a significant advantage over the high threshold. The higher preference threshold represents more personal preference of the customer, but will generate a shorter similar movie list. The lower preference threshold would generate a longer movie list thus has more opportunity to include more popular movies. This result demonstrates that movie popularity plays an important role in this recommendation system.

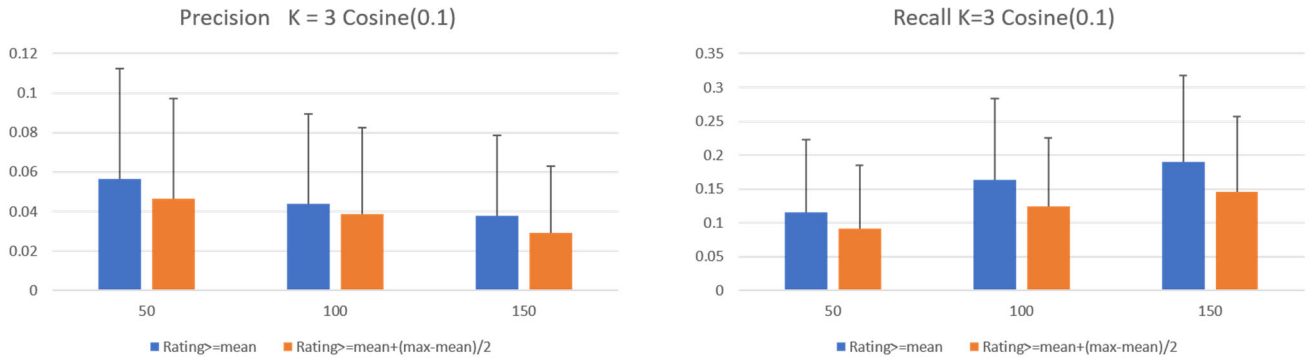

Fig 8: Recommend movies to the customer

Then we performed another experiment. This time we set preference threshold equals to mean, and K= 3, to compare the impact of similarity decreasing intervals. The result is shown in Fig 9. The results from different similarity decreasing intervals are comparable. Large interval will make the candidate movie list even longer. But the low preference threshold may have already generated a long enough movie candidates list for popularity screening, and covered the differences introduced by the different similarity decreasing intervals.

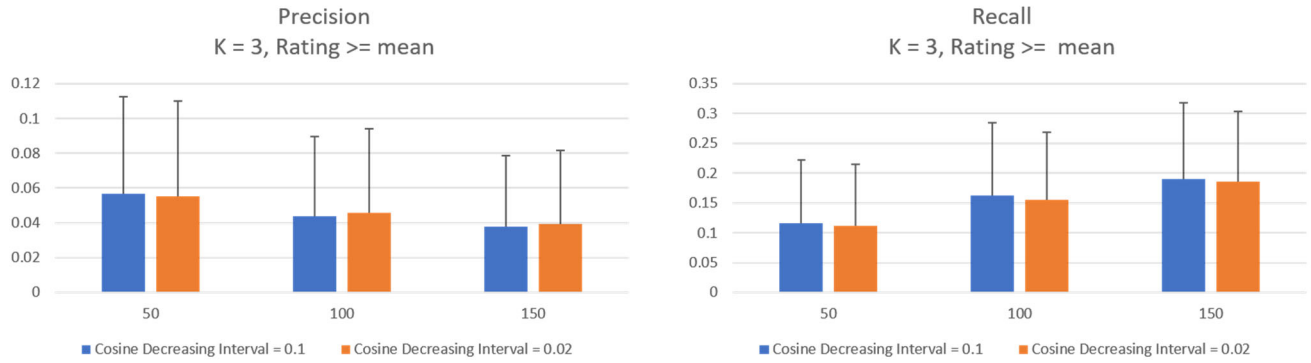

Fig 9: Recommend movies to the customer

V. Conclusion

Our recommendation system can uncover potential audiences for targeting movie advertisement, based on the similarity between movies. This recommendation has a high recall rate and poor precision. The movie similarity is computed just by their genres. If we have more features of movies, we can further distinguish the movies from each other and have a more accurate similarity score. We believe that would improve our system's performance. In our experiment, we didn't limit the number of customers in the recommendation list. In fact, if we choose different list length for different kind of movies, the result may be different. Since the movies belonging to some genre have more audiences than other movies.

In order to recommend movies to user, we employ the user preference, movie similarity, KNN model and movie popularity. They collaborate to generate a recommendation list. The precision is low and the SD bar is big, partially because of the long tail of distribution. For example, for a user who has watched about 30 movies, there is about 6 records (20% of all his records) in the test data set belong to this user. If we recommend 50 or 150 movies to this user, the highest precision is 6/50 or 6/150. Unfortunately, a lot of users in this data set watches few movies.

Through our experiments, we find that the user preference, combined with the movie content similarity, impacts the result a lot. Including movie popularity is important to our system too. The parameter K for the KNN model does not show much importance, partially because its power of choosing candidates is covered by others.

VI. Execution of my code

The dataset we are working with is pretty large, you can download it from the link http://files.grouplens.org/datasets/movielens/ml-latest.zip. After download, please unzip it to 'code' folder.

There are 3 .py files of our code. DataProcess.py reads the original data from 'ml-latest' fold and write pruned data to current path. movie.py reads the pruned data and save mapping information for late usage. recom.py calculate the recommendation and evaluate the results, results will be stored in .json file.

You can run our code by submitting the batch job directly in palmetto cluster: **qsub recom.pbs** It will load the anaconda3/5.1.0 module and activate root python environment. Then it will execute these three python files one by one. After the job, you can find our results stored in *.json file.

We produce the figures in the report by importing results and plotting in Excel.

VII. Reference

[1] Content-based Recommender Systems: State of the Art and Trends. Pasquale Lops, Marco de Gemmis, Giovanni Semeraro. Recommender Systems Handbook, Springer, pp 73-105.

[2] CPSC-8480 Lecture slides "I6b" and "I7a".

[3] k-nearest neighbors algorithm, Wikipedia, https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[4] Precision and recall, Wikipedia, https://en.wikipedia.org/wiki/Precision_and_recall.

[5] GroupLens.org, https://grouplens.org/datasets/movielens/.