Contents

Introduction

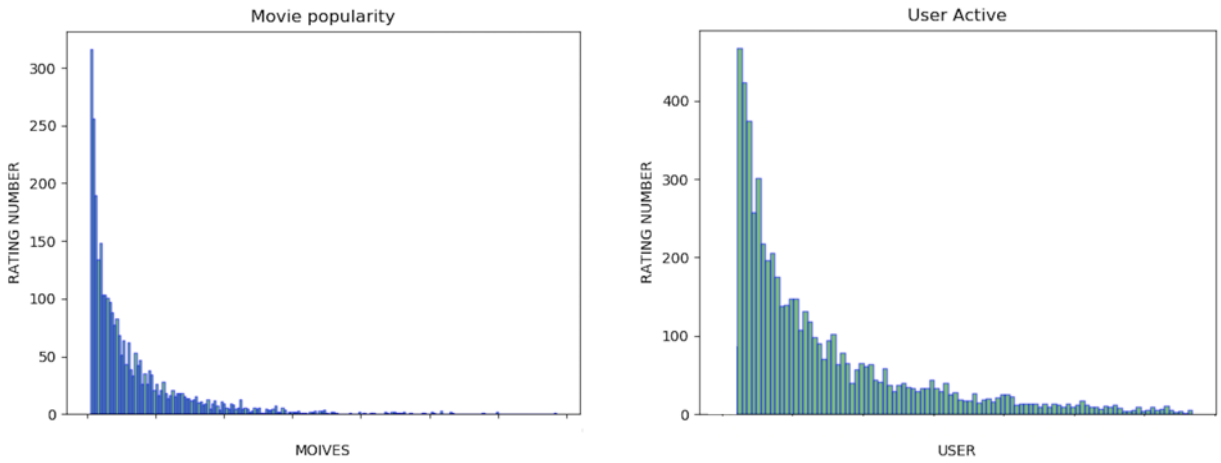
In recent years, collaborative filtering (CF) has been successfully used to provide users with personalized products and services. Among many various collaborative filtering technologies, nearest neighborhood (NN) is best known as the standard method. There are two kinds of algorithm based on NN which are user-based CF and item-based CF. In this project, our group has implemented both algorithms on a movie recommendation system. Then, different improvement methods are proposed and evaluated on real data sets. As well as testing and review of both methods on our data set.

Data set

Our project is training and testing on the MovieLens data set provided by GroupLens[1]. We choose MovieLens 1M Dataset which contains one million rating of 3900 movies made by 6040 users. Showing item popularity and user active in dataset are both long tail distribution as below

Figure 2-3 [2](Left) showing popularity of movies long-tailed distribution

Figure 2-4 [3] (Right) showing activity of users long-tailed distribution



[1] *Reference 1. Grouplensorg. 2015. GroupLens.[Online]. [23 July 2019]. Available from: https://grouplens.org/datasets/movielens/1m/*

[2] *Appendix Figure 2-3*

[3] *Appendix Figure 2-4*

Implementation of standard UserCF

There are two steps to approach a basic userCF:

(1) Finding a collection of users who are similar to the target user.

(2) Recommend movies which target user haven't be watched, but liked by his similar users.

To calculate the similarity of users in step (1) , we choose to adapt Cosine similarity rather than Jaccard similarity.

$$w_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)||N(v)|}}$$

W_{uv} similarity between user u and user v
$N(u)$ set of movies rated by user u
$N(v)$ set of movies rated by user v

However, the complexity of coding W_{uv} can be as high as O(|U|*|U|) which is pretty time consuming. Additionally, a lot of users have never liked any same item so as to cause wasting time on calculating similarity between users such as $|N(u) \cap N(v)| = 0$. To tackle this problem, we build a inverted index of items to user and calculate the numerator, which for users such $|N(u) \cap N(v)| != 0$ then divided by denominator as in above formula.

After getting the similarity between pair users, UserCF will predict the rating of items by using the formula below:

$$p(u,i) = \sum_{v \in S(u,K) \cap N(i)} w_{uv} r_{vi}$$

W_{uv} *similarity between user u and user v*
r_{vi} *rating of movie i by user v*
$N(i)$ *set of users who rated item i*
$S(u,K)$ *set of the most K similar users with u*
$p(u,i)$ *predicted rating of i by user u*

UserCF improvement: User_IIF

Above all, we adopted Cosine similarity but it is not good enough to deal with some items which are too popular. Assuming both users rated films such as "Harry Potter", it is not a good idea to recognize they have strong similar interesting because almost every one rated "Harry Potter".

To tackle this "Harry Potter" issue, we implement ***UserCF_IIF*** algorithm[4] for calculating similarity which can reduce the contribution of popular movies to the user similarity.
The formula to calculate user similarity as below:

$$w_{uv} = \frac{\sum_{i \in N(u) \cap N(v)} \frac{1}{\log 1 + |N(i)|}}{\sqrt{|N(u)||N(v)|}}$$

$N(u)$ *set of items rated by user u*
$N(v)$ *set of items rated by user v*
$N(i)$ *set of items rated by both u and v*
Therefore, there is more contribution to similarity if two users rated an unpopular movie.
The comparison of performance of Basic UserCF and UserCF_IIF will be showing in the experimentation.

Experimentation of UserCF and User_IIF

The offline experimentation design.

Step 1, Dividing data set

Randomly divide data set (ratings.dat) into 5 parts according to the uniform distribution, and select one as the testing set.

Step 2, Establish model and predict

Establish the model on the training set. Then predict user behavior (recommendation list) on the testing set as well as count the evaluation index. There are 10 movies in each users recommendation list.

Step 3, Cross-validation

Repeating step 2 three times with different training and testing set to avoid overfitting. Then calculate the average value of each evaluation index.

Evaluation index:

System will predict the movie rating by each user and recommend top 10 movies. However, when we test and evaluate a model, we are not comparing the predicted rating with actual rating. In other words, system is providing ten movies which are most similar with users rating list. Additionally, recommendation result is without regard to user's biographical information and the content of movies. Our algorithms in this project would only consider user's behavior (rating list).

---

[4] Reference 2 John S. Breese、 David Heckerman and Carl Kadie paper" Empirical Analysis of Predictive Algorithms for Collaborative Filtering"(Morgan Kaufmann Publishers，1998)。

Precision

Precision is describing how much of the final recommendation list has occurred in the testing record. *R(u) is recommend movie list to user u, |R(u)| = 10*
T(u) is the set of rating movies by user u in testing data set.

$$\text{Precision} = \frac{\sum_u |R(u) \cap T(u)|}{\sum_u |R(u)|}$$

Recall

Recall is telling how many percent of the recommendation in the testing rating record.
For example, if the value of recall is equal to 20%, this means that there are 2 recommended movies which actually rated by user. By knowing that each user have at least 20 rating records and we only provide 10 movies when test on 20% of the rating data, 20% recall is relatively acceptable. Actually, recall will be higher when we try to provide 20 movies due to more movies are in the user's favorite 20 list. Both Recall and Precision are representing the accuracy of the algorithm.
R(u) is recommend movie list to user u, |R(u)| = 10
T(u) is the set of rating movies by user u in testing data set.

$$\text{Recall} = \frac{\sum_u |R(u) \cap T(u)|}{\sum_u |T(u)|}$$

Coverage

Coverage is the number of movies has ever been recommended divided by the total number of movies. It's telling us how many percent of the movies has been recommended. Thus, coverage is the index which represents the ability to discover unpopular movies in the long-tail. If every movie has been recommended at least once to any user, coverage will be equal to 1.
U is all users set,
$R(u)$ *is the set of recommended movies for user u.*
| I | is the Cardinality of all movies.

$$\text{Coverage} = \frac{\left|\bigcup_{u \in U} R(u)\right|}{|I|}$$

Popularity

Popularity index is representing the average popularity of each movie has been recommended. This indicator will be lower if the system recommended more unpopular movies.

$$Popularity = \frac{\sum_{\boldsymbol{m \in R(u) \cap u \in U}} \mathbf{log}(\mathbf{1} + \boldsymbol{F(m)})}{|\boldsymbol{U}||\boldsymbol{R(u)}|}$$

U is all users set,
$F(m)$ *is the frequency of movie m rated by users.*
$R(u)$ *is the set of recommended movies for user u.*

UserCF performance

Due to the fact that K is only one important parameter algorithm, which represents selecting number of K most similar users, off-line experiment measures the performance of UserCF at different K values as shown in Table 2-4 [5]below

K	precesion	recall	coverage	popularity
5	0.159702	0.096671	0.444775	6.978366
10	0.181109	0.109629	0.354088	7.105241
20	0.197748	0.119701	0.283433	7.205566
40	0.203113	0.122948	0.220087	7.290052
60	0.203444	0.123148	0.196806	7.336128
80	0.202566	0.122617	0.177856	7.367399
160	0.195993	0.118639	0.133189	7.442401

To show the performance of UserCF, we compared with *Random* and *MostPopular* which are representing two kinds of extreme cases.
Random algorithm is randomly recommending ten movies which users haven't watch. It's coming with 100% coverage but low precision and low recall due to the uniform distribution.
MostPopular algorithm is always recommending top ten most popular movies haven't been watched by target user. This will cause a very low coverage but high popularity.
Table 2-5 [6] is showing the performance of Random and MostPopular.

Table	precesion	recall	coverage	popularity
Random	0.005132	0.003107	1	4.419563
MostPopular	0.104868	0.063478	0.028966	7.747065

As shown in Table 2-4 and Table 2-5,
the precision and recall index of UserCF are nearly double of that in MostPopular algorithm. At the same time, the recommendation results of UserCF are less popular whereas the coverage is much higher than MostPopular.
Meanwhile, it can be proven that K certainly is an important parameter of UserCF which can influence on each index more or less.

[5] Appendix Table 2-4
[6] Appendix Table 2-5

Comparison between UserCF and UserCF_IIF

Additionally, we evaluate the performance of UserCF_IIF through experiments and compare it with standard UserCF. We choose **K = 60** because it is giving the best performance of UserCF in previous experiment.

As shown in Table 2-6[7], UserCF_IIF is slightly better than UserCF in performance. This indicates that the algorithm has been improved due to considering the popularity of the movie when calculating the similarity between users.

Table 2-6

K = 60	precesion	recall	coverage	popularity
UserCF	0.203444	0.123148	0.196806	7.336128
UserCF-IIF	0.228592	0.19938	0.1969	7.228899

Conclusion of User-base CF

Performance of user-based CF is better than only providing the most popular movies. There are indeed somehow personalized recommendation. However, userCF has some drawbacks. As the number of users increases, it becomes more and more difficult to calculate the user similarity matrix. Second, it is difficult to explain the recommendation results. User will not make sense when see the recommendation. Therefore, the famous e-commerce company Amazon has proposed another algorithm which is item-based collaborative filtering algorithm.

[7] Appendix Table 2-6

Item-based Collaborative Filtering

This part will introduce how we implement the basic item base CF as well as propose an improved method of the algorithm. Then evaluate algorithms through the experiments on data set. The item-based collaborative filtering algorithm (ItemCF) gives recommendations of items that are similar to the items user liked before. For example, the algorithm will recommend “Iron Man” for user because he watched “Avengers” before.

However, the ItemCF algorithm does not use the content attributes of the item to calculate the similarity between items. It mainly calculates the similarity between items by analyzing the user's behavior record(rating.dat). The algorithm considers that item A and item B have a large similarity due to users who like item A also like item B.

Implementation of standard Item Base CF

There are two steps to implement a basic itemCF:

(1) Calculating the similarity between items

(2) Generate recommendation list based on item similarity and target user’s behavior history. (rating list)

Step(1): Many shopping websites are showing a title as “Customers who bought this item i also bought item j”. From the definition of this sentence, we use the following formula to define the similarity of items.

$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|}$$

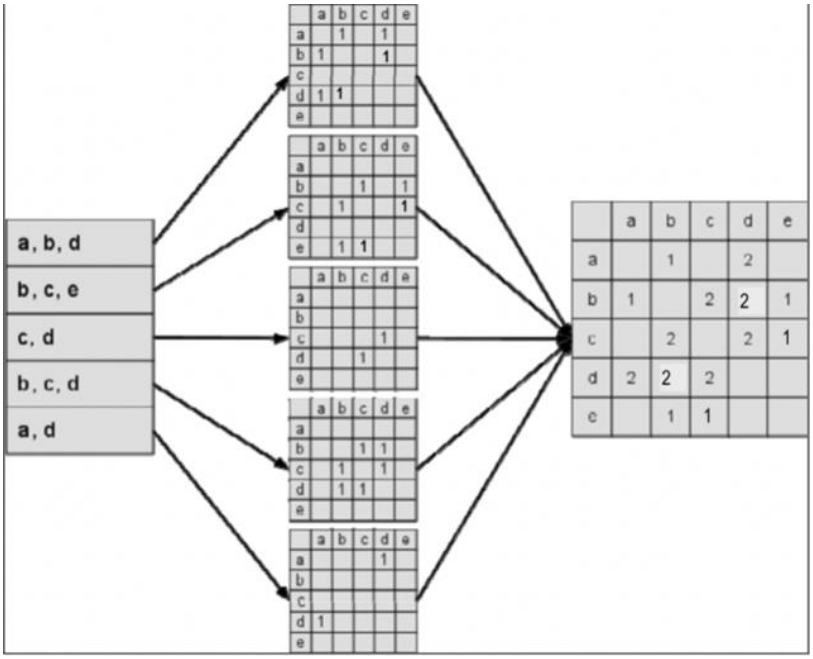
$N(i) set\ of\ users\ rate\ item\ i$
$N(j)\ \ set\ of\ users\ rate\ \ item\ j$
$w_{ij}\ similarity\ between\ item\ i\ and\ j$

Above formula can be interpreted as the ratio of how much item j has been liked by users of item i. Therefore, If more users of item i also like item j, then the similarity of items will be higher. Although this formula seems to make sense, there is still a “Harry Potter” problem. Similarity is influenced by the popularity of items. W_{ij} will be close to 1 If item j is too popular such as “Harry Potter”,which has been liked by almost everyone. Thus, “Harry Potter” will be similar with any other items. No matter what movies are in the user’s history, “Harry Potter” will be always in the recommendation list. This is obviously not a good feature for algorithm to discover information in the long tail.

To avoid “Harry Potter” issue, we redefined the similarity formula as below:

$$w_{ij} = \frac{\left|N(i) \cap N(j)\right|}{\sqrt{\left|N(i)\right|\left|N(j)\right|}}$$

This formula punished the weight of item j so that reducing the similarity of popular items
Two items produce similarity because they are commonly enjoyed by many users, that is, each user can “contribute” similarity to the item through their rating list.
It is worth noting that a hypothesis here. Each user has a limited interest to a few types of movies. so if two movies belong to user’s rating list, then the movies should belong to a limited number of types.
Moreover, if two movies belong to the rating list of many users, they may belong to the same movie type. Thus, the two movies have a great similarity.
Similar to the UserCF algorithm, the first step of itemCF is to calculate the item similarity using inverted index. So for each user, build a list containing the items he rated, then add one to his co-occurrence matrix C. figure 2-11[8] showing five users rating list as an example.



In general, movies will have a greater similarity if they belong to the same series, types or protagonists. Then, Table 2-7 [9]is the results of calculating item similarity using above method. As shown in the table, although not using any content attributes in the calculation process, results of the program indeed show some similarity in content. Therefore, we verified our previous hypothesis here.

[8] Appendix Figure 2-11
[9] Appendix Table 2-7

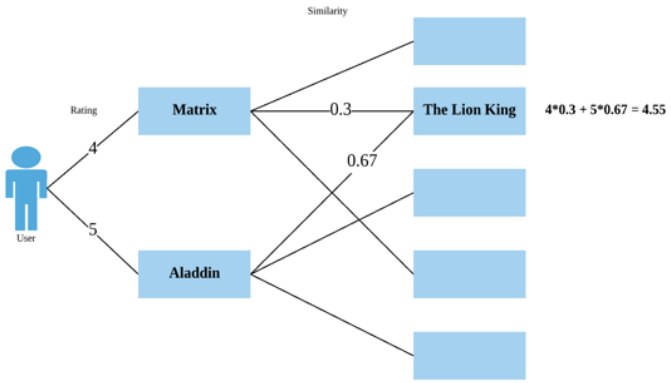
B	C	D
movie_id_1	movie_id_2	similarity
Matrix, The (1999)	Who Framed Roger Rabbit? (1988)	0.6994
Matrix, The (1999)	Arachnophobia (1990)	0.68535
Matrix, The (1999)	Total Recall (1990)	0.68332
Matrix, The (1999)	Jurassic Park (1993)	0.67611
Matrix, The (1999)	Arachnophobia (1990)	0.62744
Aladdin (1992)	The Lion King(1994)	0.676119
Aladdin (1992)	Snow White and the Seven Dwarfs (1937)	0.66967
Aladdin (1992)	James and the Giant Peach (1996)	0.66803
Aladdin (1992)	Cinderella (1950)	0.65209
Aladdin (1992)	Sleeping Beauty (1959)	0.63838

After getting the similarity between items. ItemCF algorithm will predict the rating of item j by user u based on the formula below:

$$p_{uj} = \sum_{i \in N(u) \cap S(j,K)} w_{ji} r_{ui}$$

P_{uj} *is the possible interest of item j by user u.*
W_{ji} *is the similarity between item j and i*
r_{ui} *is the rating of item i by user u.*
$N(u)$ *is the set of items which liked/rated by all users,*
$S(j,K)$ *is the set of most similar K items with item j.*

The meaning of this formula is that the more similar an item is to the user's rating list, the more likely it is in the user's recommendations. A simple example is given as below:
Assume one user rated "Matrix" and "Aladdin". ItemCF will find the most K=3 similar movies to each of "Matrix" and "Aladdin". Then, calculate how much user will be interested in every movie based on the formula above. Given example as below. Let the rating of "Matrix" and "Aladdin" be 4 and 5 respectively.
The similarity between "Matrix" and "The Lion King" is equal to 0.3.
The similarity between "Aladdin" and "The Lion King" is equal to 0.67.
Then the predicted interest of "The Lion King" by user is 4*0.3 + 5*0.67 = 4.55



Impact of user activity on item similarity

In itemCF, two items are similar because they are in many users rating list. In other words, every each user's rating list has some contribution to the similarity of items. However, not every user has the same weight of contribution.

Suppose there is such a user Bob, he is a wholesaler and bought 80% of movies to sell. Then his history list contains 80% of all movies. Assuming total number of movies is 4000 and Bob has rated 3200 movies. From the previous discussion of ItemCF, this means that there are 3200 movies have pair similarities only because of Bob's behavior. Thus, a dense matrix of 3200 to 3200 is about to born in memory.

Although Bob is very active, but it is not all in his own interest to buy these movies. Moreover, Bob's movies list covered many genres. So, the contribution of Bob to similarity of the movies should be much less than someone who only bought and rate his favorite movies. We can simply ignore this Bob because he is too active. Otherwise, IUF is another method as below.

Inverse User Frequency (IUF)

John S. Breese [10]proposed a parameter called Inverse User Frequency, which is the reciprocal of the logarithm of user activity. He also believes that active user's contribution to the similarity of items should be less than inactive users. This project implemented IUF as an argument in ItemCF to improve the formula for calculating the item similarity as below:

$$w_{ij} = \frac{\sum_{u \in N(i) \cap N(j)} \frac{1}{\log 1 + |N(u)|}}{\sqrt{|N(i)||N(j)|}}$$

w_{ij} *similarity between item i and j*
N(j) set of users rated movie j
N(i) set of users rated movie i
N(u) set of items rated by u

---

[10] Reference John S. Breese、 David Heckerman and Carl Kadie "Empirical Analysis of Predictive Algorithms for Collaborative Filtering"(Morgan Kaufmann Publishers， 1998)。

Experimentation of ItemCF and ItemCF_IUF

The offline experimentation design.

Step 1, Dividing data set
Randomly divide data set (ratings.dat) into 5 parts according to the uniform distribution and select one as the testing set.

Step 2, Establish model and predict
Establish the model on the training set. Then predict user behavior (recommendation list) on the testing set as well as count the evaluation index. There are 10 movies in each user's recommendation list.

Step 3, Cross-validation
Repeating step 2 three times with different training and testing set to avoid overfitting. Then calculate the average value of each evaluation index.

Performance of ItemCF

Table 2-8[11] lists ItemCF evaluation results of the algorithm at different K values.

Table2-8

K	precesion	recall	coverage	popularity
5	0.158344	0.095598	0.4429	6.979792
10	0.18128	0.109498	0.35597	7.107214
20	0.195801	0.118215	0.280596	7.207151
40	0.203632	0.122719	0.218392	7.289697
60	0.203731	0.123451	0.194331	7.33365
80	0.203057	0.122705	0.174278	7.364603

According to the data in Table2-8, we can draw the following conclusions
Precision and Recall: It's showing that precision and recall of the ItemCF is neither positively nor negatively correlated with K. Thus, it is very important to select an appropriate value of K to obtain the highest accuracy.

Popularity: The popularity of ItemCF algorithm is not completely positively correlated with K. As K increases, the popularity of the results will gradually increase, but popularity will not change significantly once K reach a certain value.

Coverage: Increasing K will reduce the coverage of the system.

[11] Appendix Table 2-8

Performance of ItemCF_IUF

To compare with basic ItemCF, we set **K= 60** due to it is giving the best performance in previous experiment. Result of ItemCF_IUF show as in Table 2-9[12],
ItemCF-IUF is doing a significantly improvement on precision and recall. Also, IUF is better than itemCF in terms of coverage and popularity. In this sense, IUF does improve the overall performance of ItemCF.

K = 60	precesion	recall	coverage	popularity
ItemCF	0.203731	0.123451	0.194331	7.33365
ItemCF-IUF	0.228592	0.19938	0.1969	7.228899

Comprehensive comparison between UserCF and ItemCF

From the principle of two algorithms, we can conclude that UserCF is focus on the hotspots of small groups with similar user interests. Whereas ItemCF is focus on the personal historical interest records. In other words, the recommendation of UserCF are more social and reflecting the popular items in the same user group. Recommendation of ItemCF is more personalized and reflecting the user's own interest inheritance.

Performance:
Technically, UserCF has to maintain a similarity matrix of users and ItemCF need to maintain a similarity matrix of items. So, UserCF requires a lot of space to maintain if there are a lot of users. Similarly, if there are many items, then maintaining the item similarity matrix could be expensive.

Domain:
UserCF is more suitable for areas required strong timeliness but weak personalized interest (News). ItemCF is better at digging information in the long tail. Therefore, ItemsCF has huge advantage on area where users have strong personalized demand (Amazon).

Real-time:
Recommendation of ItemCF will be updated in real-time after users has a new behavior (YouTube). In contrast, it's not necessary for UserCF to change the recommendation immediately once user has a few new behavior (Twitter).

Cold start:
Due to the similarity matrix of users is regularly calculated offline, userCF cannot generate personalized recommendation immediately after a new user has a few behaviors on items. In contrast, as long as the new user acts on any item, itemCF can recommend other related items immediately.

[12] Appendix Table 2-9

Explanation of the recommendation result
UserCF is difficult to provide a reasonable explanation. Whereas itemCF can make result more convincing because recommendation is based on his own behavior on items.

Plotting result

Also, we plot result of experiment as in Figure2-13[13], Figure2-14[14], Figure2-15[15]

Figure 2-13 Recall index at different K value

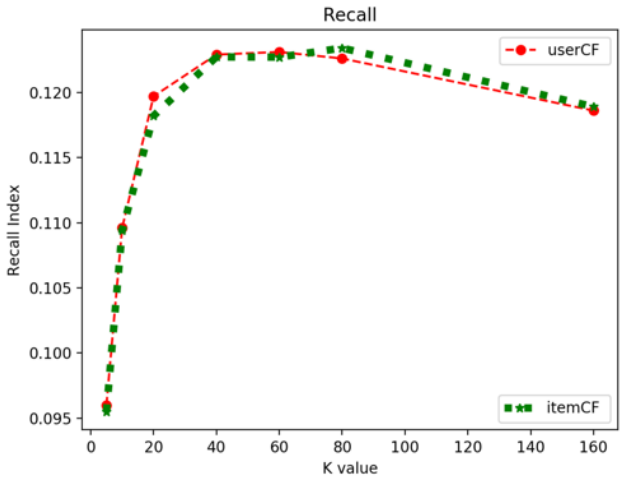
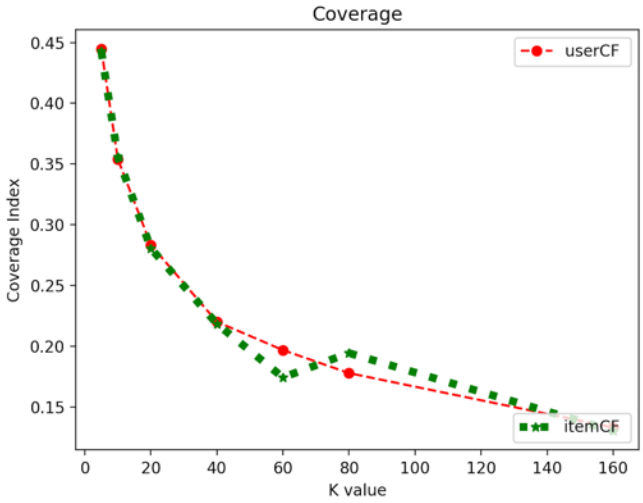
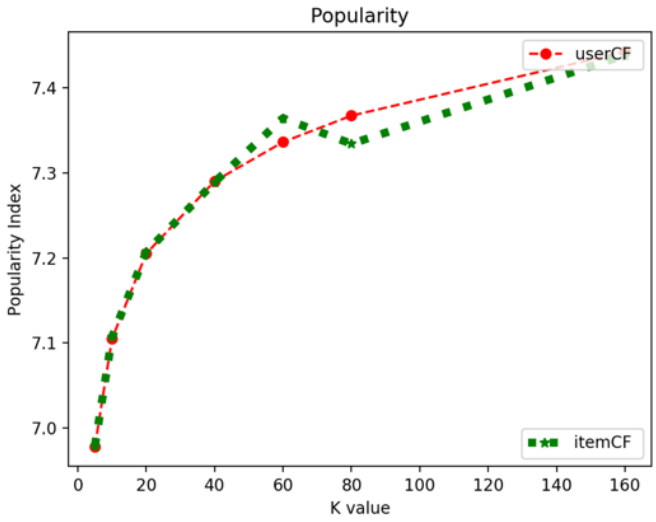


Figure 2-14 Coverage index at different K value



[13] Appendix Figure 2-13
[14] Appendix Figure 2-14
[15] Appendix Figure 2-15

Figure 2-15 Popularity index at different K value



In practice, when we have to choose which algorithm to implement in recommendation system, we are not only considering the off-line experimental performance but also considering the cost.

Future Work

Normalization of similarity matrix

Generally, items in the popular class have larger similarity weight. So the recommendation coverage is relatively lower due to the system is more likely to suggest popular items.
To improve the performance of ItemCF, it is possible to normalize the value in similarity matrix. By the study of Karypis[16], normalizing to the maximum value of matrix can improve the accuracy as well as the diversity of the coverage. Normalized similarity matrix W can be obtained by the following formula:

$$w_{ij}^{'} = \frac{w_{ij}}{\max\limits_{j} w_{ij}}$$

"Harry Potter" problem

In the basic ItemCF, we use the most classical formula to calculate item similarity. In practice, a popular item such as "Harry Potter" can still make a large impact on w_{ij}. To deal with this "Harry Potter" problem, we can simply ignore this popular item. Otherwise, using an alternative formula can increase the "punishment" to popular items. As reference in the blog of Greg Linden [17],this

---

[16] Reference George Karypis paper "Evaluation of Item-based Top-N Recommendation Algorithms"

[17] Reference Greg Linden blog http://glinden.blogspot.com/2006/03/early-amazon-similarities.html。

method can significantly improve the coverage and popularity of the result at some sacrifice of precision and recall. Given the formula as below.

$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|^{1-\alpha} |N(j)|^{\alpha}}$$

However, It's not possible to solve the "Harry Potter" problem completely with NN method. Thus, it is better to use latent variable analysis technology to do further improvement.

Latent Factor Model

Further work can be done by implementing LFM which is a machine learning based method with a good theoretical basis. This method has some advantages compared to neighborhood-based methods. LFM is a model to find the implicit semantics of the text. Related terms are LSI, PLSA, LDA and Topic Model.

Cold Start

There are mainly three categories of cold start.

1. User cold start
2. Item cold start
3. System cold start

This project is only able to use MostPopular algorithm to do user cold start. After further work in the future, user cold start should be personalized.

Picky user case

The algorithm in this study can only recommend movies based on the behavior of the users (rating list). Assuming we are recommending movies to a picky user Bob, who is always mark 1 star to all movies he watched. The neighborhood-based algorithms are likely to suggest more movies he doesn't like. Future work on the recommendation system should be able to handle those picky users.

Reference

- *Reference. Grouplensorg. 2015. GroupLens. [Online]. [23 July 2019]. Available from: https://grouplens.org/datasets/movielens/1m/*
- *Reference John S. Breese、 David Heckerman and Carl Kadie paper"Empirical Analysis of Predictive Algorithms for Collaborative Filtering"(Morgan Kaufmann Publishers， 1998)。*
- *Reference John S. Breese、 David Heckerman and Carl Kadie "Empirical Analysis of Predictive Algorithms for Collaborative*
- *Filtering"(Morgan Kaufmann Publishers ， 1998)。*
- *Reference George Karypis paper "Evaluation of Item-based Top-N Recommendation Algorithms"*
- *Reference Greg Linden blog http://glinden.blogspot.com/2006/03/early-amazon-similarities.html。*
- *项亮 (2012.6). 推荐系统实践. 北京: 人民邮电出版社. p35-p72.*
- *负雪明烛的博客. 2019. 动手实现基于协同过滤的电影推荐系统 | 负雪明烛的博客. [ONLINE] Available at: http://fuxuemingzhu.cn/2018/06/12/MovieLens-Recommender/. [Accessed 11 August 2019].*

Appendix

- *Figure 2-3*
- *Figure 2-4*
- *Figure 2-11*
- *Figure 2-13*
- *Figure 2-14*
- *Figure 2-15*
- *Table 2-4*
- *Table 2-5*
- *Table 2-6*
- *Table 2-7*
- *Table 2-8*
- *Table 2-9*
- *All Appendix Table and figure in file "Appendix"*