# A Movie Recommendation System Based on Multiple Methods

Fengliang Tan
New York University
New York, USA
Ft705@nyu.edu

Xinsong Yang
New York University
New York, USA
Xy983@nyu.net

***Abstract***

With the rapidly growing popularity of the Internet and mobile Internet, the number of movie entertainment information has been extremely substantial. A movie recommendation is important in our social life due to its strength in providing personalized movies. Such a system can recommend a set of movies to users based on their interest. This paper aims to build a movie recommendation system based on multiple methods using MovieLens dataset. It uses two method Euclidean distance and Pearson correlation to calculate similarity scores of users. To make recommendations for a user, it calculates a weighted score by adding all the other users' ratings multiplied by similarity scores for each movies that the user haven't seen. It also provide item-based filtering as well as user-based filtering in the system. Different from user-based filtering technique described above, item-based filtering means calculating weighted scores using similarity between movies instead of users. It recommend the movies with the highest weighted score. Users can choose different ways to get recommendations by setting parameters. In addition, the results can also be filtered by year and genres and the key tags will be showed.

***Keywords—machine learning, item-based filtering, user-based filtering***

## I. INTRODUCTION

Dataset: We choose MovieLens as our dataset. This dataset (ml-latest-small) describes 5-star rating and free-text tagging activity from [MovieLens](http://movielens.org), a movie recommendation service. It contains 100004 ratings and 1296 tag applications across 9125 movies. These data were created by 671 users between January 09, 1995 and October 16, 2016. This dataset was generated on October 17, 2016.

Software: Anaconda 4.2.0 with Python 3.5.

Backgrounds: The amount of information in the world is increasing more rapidly than our ability to process it. People are buried by the over prosperous information since we can't tell whether a book is good or not unless we read it, which is time consuming. One of the most promising technologies to solve this problem is collaborative filtering. Collaborative filtering works by building a database of preferences for items by their users. Then, a new user is matched against the database to discover neighbors. The neighbors are other users who share the same or similar interests with our new user. We recommend items that neighbors like to the new user, as he may also like them. Collaborative filtering has been proved very successful in both research and practice as well as facing several challenges.

The first one is to improve the scalability of collaborative filtering algorithms so that they will be able to handle tens of millions data and give feedback in real time. Further, it is complicate to make sure that the recommendations are valid and qualitive. Users only need recommenders that they can trust to help them find exactly what they want instead of wasting their time. In some ways, these two challenges are conflict since the more time we spent on analyzing, the better recommendation will be made.

Project idea: In Microsoft's 1998 paper "programming collective intelligence" on collaborative filtering, this method was divided into two genre, memory-based and model-based. Model-based algorithm, using modeling algorithms of machine learning such as Bayesian belief nets, clustering, SVM, makes pre calculating for models offline and so quickly gets resulting online. Memory-based algorithm is divided into user-based, which take advantages of user's similarities, and item-based, which based on items' relationship.

Our project is to build a simple movie recommendation system based on memory-based methods using MovieLens dataset. First, we use two method Euclidean distance and Pearson correlation to calculate similarity scores of users. To make recommendations for a user, we calculate a weighted score by adding all the other users' ratings multiplied by similarity scores for each movies that the user haven't seen. We also provide item-based filtering as well as user-based filtering in the system. Different from user-based filtering technique described above, item-based filtering means calculating weighted scores using similarity between movies instead of users. Finally we recommend the movies with the highest weighted score. Users can choose different ways to get recommendations by setting parameters. In addition, the results can also be filtered by year and genres and the key tags will be showed. To make test for item-based filtering, we separate the data set into train(75% people) and test(25% people). Then we extract 70% movies people in test like as the input, we get the output and compare it with the remaining 30%. To test user-based filtering, we have to do a questionnaire survey. However, questionnaire survey is so complicated that beyond our ability because few people would be willing to do it without paying. So, we will not do the test for user-based filtering.

## II. Motivation

With the explosion of Information coming due to the prosperity of Internet and Mobile Internet, it becomes more complicate for people to find the information they really desire, which reduces the use efficiency of information. Recommender system is a useful way to solve this problem because it is built based on user's interests and demand. Recently, recommender System has become extremely common, and are utilized in a variety areas. Some applications are including music, books, movies and etc. Consumers can find what they are interested in more efficiently while merchants can make more selling due to this useful system.

Recommender System is also a very important branch of machine learning. It typically produces a list of recommendations in one of two ways – through collaborative and content-based filtering or the personality-based approach. Collaborative filtering can also be approached by two ways – through memory-based or model-based. It approaches building a model based on user's past behavior as well as similar decisions made by other users, which contributes to a split of item-based filtering and user-based filtering. Content-based filtering approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties.

## III. Related Work

Recommendation algorithms are best known for their use on e-commerce Websites, where they are use input about a customer's interests to generate a list of recommended items.[1] At Amazon.com, they use recommendation algorithms to personalize the online store for each customer. The E-commerce recommendation algorithms faces many challenges such as handling the huge amounts of data, giving results in real-time. Amazon.com has created an efficient one called item-to-item collaborative filtering, which produces recommendations in real-time, scales to massive datasets, and generates high quality recommendations. Besides, unlike traditional collaborative filtering, their algorithm is online computation scales independently of the number of customers and the number of items in the product catalog.

The goal of collaborative filtering algorithms is to suggest new items or predict the utility of a certain item for a particular person based on his previous behaviors or the opinions of those have similar interest with him. A traditional collaborative filtering can be divided into two genres – memory-based collaborative filtering and model based collaborative filtering. Memory-based algorithms utilize the entire user-item database to generate a prediction.[2] These systems use statistical techniques to find a set of users, known as neighbors, which have similarities to the target user. Once a neighbor is formed, these systems will use nearest-neighbor (or user-based collaborative filtering) techniques to produce a top-N recommendation. However, there are several challenges of user-based collaborative filtering algorithms such as: Sparsity and Scalability.

Sparsity[2]: In practice, users may purchase under 1% of the items due to the large item set. In this case, a user-based collaborative filtering recommendation system may be unable to make any recommendations for a particular user.

Scalability[2]: User-based collaborative filtering algorithm requires computation that grows with both the number of users and numbers of item. When the number is extremely huge, the running will suffer serious scalability problems.

Rather than matching the user to similar customers, item-to-item collaborative filtering matches each of the user's purchased and rated items to similar items, then combines those similar items into a recommendation list.[1] The algorithm builds a similar-items table by finding items that customers tend to buy together, so that it can determine the most similar match for a given item. Amazon.com builds a product-to-product matrix by calculating the similarity between a single product and all related product. This offline computation of similar-items table is extremely time intensive, with O(NM) as in practice. The key to item-to-item collaborative filtering's scalability and performance is that it creates the expensive similar-items table offline. The algorithm's online component — looking up similar items for the user's purchases and ratings — scales independently of the catalog size or the total number of customers; it is dependent only on how many titles the user has purchased or rated. Thus, the algorithm is fast even for extremely large data sets. Because the algorithm recommends highly correlated similar items, recommendation quality is excellent.[1] Code for item-to-item collaborative filtering is as follows:

```
For each item in product catalog, $I_1$
  For each customer $C$ who purchased $I_1$
    For each item $I_2$ purchased by
       customer $C$
      Record that a customer purchased $I_1$
       and $I_2$
  For each item $I_2$
    Compute the similarity between $I_1$ and $I_2$
```

Item-based filtering is significantly faster than user-based when getting a list of recommendations for a large dataset, but it does have the additional overhead of maintaining the item similarity table.[3] Besides, Item-based collaborative filtering generally performs better than user-based collaborative filtering in sparse datasets, and the two performance are almost the same in dense datasets. However, user-based collaborative filtering is simpler to implement and does not take extra steps, making it more appropriate with smaller in-memory datasets which changes frequently.

## IV. DATA

We choose movielens as our dataset. This dataset (ml-latest-small) describes 5-star rating and free-text tagging activity from [MovieLens](http://movielens.org), a movie recommendation service. It contains 100004 ratings and 1296 tag applications across 9125 movies. These data were created by 671 users between January 09, 1995 and October 16, 2016. This dataset was generated on October 17, 2016.

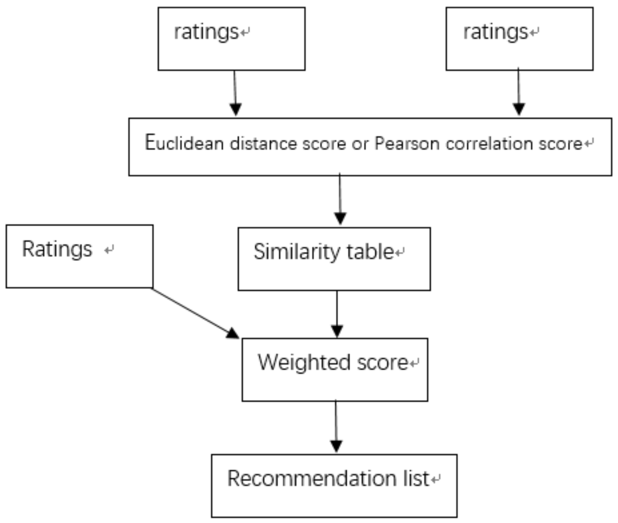
All ratings[4] are contained in the file `ratings.csv`. Each line of this file after the header row represents one rating of one movie by one user, and has the following format: userId, movieId, rating, timestamp.

All tags[4] are contained in the file `tags.csv`. Each line of this file after the header row represents one tag applied to one movie by one user, and has the following format: userId, movieId, tag, timestamp.

Movie information[4] is contained in the file `movies.csv`. Each line of this file after the header row represents one movie, and has the following format: movieId, title, genres. Movie titles are entered manually or imported from <https://www.themoviedb.org/>, and include the year of release in parentheses. Genres are a pipe-separated list, and are selected from the following: action, adventure, animation, children's, comedy, crime, documentary, drama, fantasy, film-noir, horror, musical, mystery, romance, sci-fi, thriller, war, western.

## V. ALGORITHM(S) USED

The system processing is as follow:

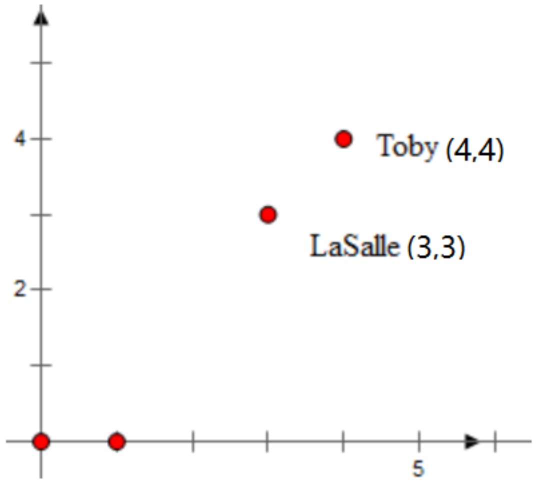


To recommend movies, the first step is to calculate similarity scores. In user-based recommendation we need to find similar users and in item-based recommendation we need to find similar movies to the movies a user have seen. There are a few algorithms for calculating similarity scores. In our system, we used two basic algorithms: Euclidean Distance Score and Pearson Correlation Score. Users can choose which algorithm to use in the program by setting a parameter.

Euclidean Distance Score: To calculate the distance between person1 and person2 in a n dimension chart, take the difference in each axis, square them and add them together, then take the square root of the sum. Then we use add 1 and inverting method to make the function gives higher value when people are similar. This new function always returns a value between 0 and 1, where a value of 1 means two people have identical preferences.

$$\mathrm{score}(a,b) = 1/(1+\sqrt{(a_1-b_1)^2+(a_2-b_2)^2+\cdots+(a_n-n)^2})$$

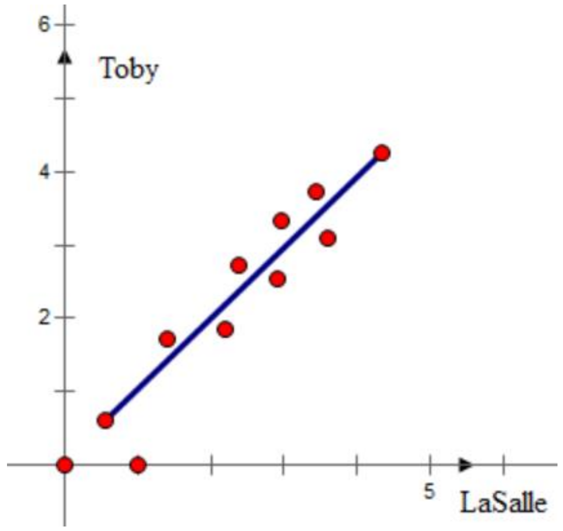
For example, we calculate the similarity of Toby and LaSalle, we find movie the both rated, and put it into a chart as follow:



Then, we can clearly see how this function works. The distance score is

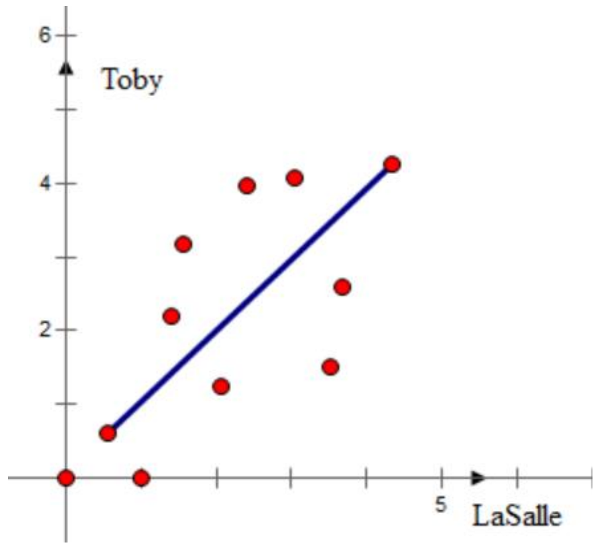
$$1/(1+\sqrt{(4-3)^2+(4-3)^2}$$

Pearson Correlation Score: Plotting the ratings of critics on a chart, we can see a straight line on the chart, which is called the best-fit line because it comes as close to all items on the chart as possible. The code for Pearson Correlation Score fist finds the items rated by both critics. Then, it calculates the



sums and the sum of the squares of the rating for the two critics, and calculates the sum of the products of their ratings.

Finally, it uses these results to calculate the Pearson Correlation Coefficient. This function will return a value between -1 and 1. A value of 1 means the two people have exactly the same rating for all items.



The spots represents movies Toby and LaSalle have rated. The line is called best-fit line. The more the spots are closer the line, the two persons are more similar.

To produce a recommendation list, we need to calculate weighted scores for latent movies and the movies that have a highest score are what we will recommend. Also, there are two basic ways to do this.

Item-based Recommendation: Firstly, for all movies the user have seen, we find their similar movies that the user have not seen. Then we multiply the ratings of the movies the user have seen by the similarities of similar movies. The same as user-based way, we add up the results of the same movie and then divide the sums of similarity scores to get the weighted scores.

| Movie | Rating | Night | R.xNight | Lady | R.xLady | Luck | R.xLuck |
|---|---|---|---|---|---|---|---|
| Snakes | 4.5 | 0.182 | 0.818 | 0.222 | 0.999 | 0.105 | 0.474 |
| Superman | 4.0 | 0.103 | 0.412 | 0.091 | 0.363 | 0.065 | 0.258 |
| Dupree | 1.0 | 0.148 | 0.148 | 0.4 | 0.4 | 0.182 | 0.182 |
| Total | | 0.433 | 1.378 | 0.713 | 1.764 | 0.352 | 0.914 |
| Normalized | | | 3.183 | | 2.598 | | 2.473 |

```
for movie in a user's ratings:
    get the movie's similar movies
    for sim_movie in similar movies:
        total_score[sim_movie] += similarity * rating
        total_sim[sim_movie] += similarity
    weighted_score[sim_movie] = total_score[sim_movie] / total_sim[sim_movie]
return max(weighted_score) and its movie id
```

User-based Recommendation: When we make recommendations for a user, first we find similar users using the similarity algorithms above. For all the movies that similar users have seen but the user have not, we multiply their rating by similarity and add up the results of the same movies. Then we use the total scores to divide the sums of similarity scores and the results are weighted scores. Finally we return the movies with highest scores as recommendations.

| Critic | Similarity | Night | S.xNight | Lady | S.xLady | Luck | S.xLuck |
|---|---|---|---|---|---|---|---|
| Rose | 0.99 | 3.0 | 2.97 | 2.5 | 2.48 | 3.0 | 2.97 |
| Seymour | 0.38 | 3.0 | 1.14 | 3.0 | 1.14 | 1.5 | 0.57 |
| Puig | 0.89 | 4.5 | 4.02 | | | 3.0 | 2.68 |
| LaSalle | 0.92 | 3.0 | 2.77 | 3.0 | 2.77 | 2.0 | 1.85 |
| Matthews | 0.66 | 3.0 | 1.99 | 3.0 | 1.99 | | |
| Total | | | 12.89 | | 8.38 | | 8.07 |
| Sim. Sum | | | 3.84 | | 2.95 | | 3.18 |
| Total/Sim. Sum | | | 3.35 | | 2.83 | | 2.53 |

```
get the similar users
for movie in similar users' ratings:
    total_score[movie] += similarity * rating
    total_sim[movie] += similarity
weighted_score[movie] = total_score[movie] / total_sim[movie]
return max(weighted_score) and its movie id
```

Besides, we improve our similarity algorithms and recommendation algorithms by adding some adjustment to avoid some extreme cases. Having observed some test results we find that most recommended movies have full score because they are recommended just from one or two movies. For example, a movie has 0.2 similarity with only one movie the user have rated 5.0 then the movie's weighted score will be 5.0. In another case, a movie has 0.8 similarity with only one movie the user have rated 4.8 then the movie's weighted score will be 4.8. We think this situation is not reasonable because it is unreliable to recommend just from one or two movies with low similarity. Considering this we add adjustment variables in similarity and recommendation algorithms.

In similarity algorithms, we set the adjustment = $(c/u1 * c/u2)^p$, which c is the number of common items between user1 and user2, and p (we set p 0.05 in our program) is a parameter that controls the strength of this adjustment. We multiply the adjustment with the original scores. By doing this if the two users have seen more common movies, their similarity will improves. And for users who have seen few common movies, their similarity will drop. However, the similarity still mostly depends on if they have similar ratings.

When calculating weighted scores, the adjustment = $t^p$, which t means the total similarity and p (we set p 0.1 in our program) is a parameter that controls the strength of this adjustment. In this way the scores of movies that have more similar movies are improved and that have less similar movies are reduced.

In real applications, people normally use real user data to test. For example, if users rate highly of the recommended movies we can think the recommendation system is useful. However, we do not have the ability to do that. So we tried two alternative ways to evaluate our recommendation systems.

The first way is to design a test experiment using the original data. We divide 80% data as training data and 20% as test data. In the 20% test data, we extract a part of the user's favorite movies and put the extracted data into the training set. Then we make recommendations for the test data and check if the recommendation list contains the extracted movies. The percent of successful recommendation can show if the system is efficient.

Another way is to do user test. We invite people to rate some movies they have seen and make recommendations for them. Then we can see how many movies from recommendation list they like.

## VI. RESULT

To test our recommendation system, we individually rated 10 movies we have seen.

Fengliang: [The Legend of Tarzan (2016): 3.5, The Last Witch Hunter (2015): 3.0, Jaws 2 (1978): 3.5, Iron Man (2008): 4.5, Godfather, The (1972): 5.0, (500) Days of Summer (2009): 4.0, X-Men: The Last Stand (2006): 4.0, Titanic (1997): 4.5, Conspirator, The (2010): 2.5, The Expendables 3 (2014) : 4.0]

Xinsong: [Pulp Fiction (1994): 4.0, House of Flying Daggers (Shi mian mai fu) (2004): 2.0, Dark Knight, The (2008): 5.0, Lord of the Rings, The (1978): 5.0, Hero (Ying xiong) (2002): 2.0, Titanic (1997): 4.0, WALL.E (2008): 5.0, Amazing Spider-Man, The (2012): 3.0, Godfather, The (1972): 4.0, Iron Man (2008) : 3.0]

First we tested the different similarity algorithm with and without adjustment, the result is shown in the chart below.

| Similarity Score | Euclidean Distance | Pearson Correlation |
|---|---|---|
| No Adjustment | 0.2222222222222222 | 0.4999999999999867 |
| Adjustment | 0.1970151445700474 | 0.44328407528259484 |

Then we used different methods to get 10 recommendation movies and for each method we counted how many movies we like to see.

```
user_based_recommendations(user_ratings,'fengliang', sim_fun=sim_distance, n=10, isAdjust=True)
>> [(6.910769613186802, 'Shawshank Redemption, The (1994)'), (6.817727802235572, 'Godfather: Part II, The (1974)'), (6.724040909766639, 'Star Wars: Episode IV - A New Hope (1977)'), (6.595186825741678, 'Pulp Fiction (1994)'), (6.569954478223291, "Schindler's List (1993)"), (6.522321142996834, 'Usual Suspects, The (1995)'), (6.519934250076949, 'Forrest Gump (1994)'), (6.486351098379551, 'American Beauty (1999)'), (6.452610332510402, 'Star Wars: Episode V - The Empire Strikes Back (1980)'), (6.4409459291188, 'Matrix, The (1999)')]
```

```
item_based_recommendations(user_ratings, movie_ratings, 'fengliang', sim_fun=sim_pearson, n=10, isAdjust=True)
>> [(4.908379092169942, 'Foreign Correspondent (1940)'), (4.9039066144653205, 'Zelig (1983)'), (4.895858459569259, 'After the Thin Man (1936)'), (4.895858459569256, 'Pit and the Pendulum (1961)'), (4.895858459569256, 'Longtime Companion (1990)'), (4.895858459569254, 'Front, The (1976)'), (4.8958584595692525, 'Country (1984)'), (4.895858459569252, 'Tex (1982)'), (4.895858459569251, 'Minus Man, The (1999)'), (4.895858459569251, 'Emerald Forest, The (1985)')]
```

| Accuracy | User-based | | Item-based | |
|---|---|---|---|---|
| | Euclidean Distance | Pearson Correlation | Euclidean Distance | Pearson Correlation |
| Fengliang | 10/10 | 8/10 | 4/10 | 3/10 |
| Xinsong | 6/10 | 8/10 | 1/10 | 1/10 |

In the test experiment we described in the algorithm chapter, we did not get ideal results. Almost no extracted movies are contained in the test users' recommendation list. For a user there are thousands of potential recommendation movies but dozens of movies the user have seen. The chance of extracted movies contained in the recommendation list is very small. So this is not an efficient experiment design to test our recommendation system.

## VII. CODE

*https://github.com/XinsongYang/movie_recommendation*

## VIII. VIDEO LINK

*https://youtu.be/GKW9JjBypR0*

## IX. EVALUATION

In our results, we can see that the similarity scores produced by the two algorithms have a large difference. But we cannot see how they differently influence the recommendation from our results.

For recommendation, it is obvious that user-based recommendation has a higher accuracy than item-based. When observing the specific recommendation lists, we also found some features of the two recommendation algorithms. The movies from user-based recommendation are all famous movies rated high by a large number of users. And the movies from item-based recommendation always have the similar genres and stories with the movies that I have seen but might be rated by a few users.

However, the analysis above is based on the results of our individual cases. It is not very convincing because sample size is too small. So if we have more time we will do more user tests for a more valid analysis. When we have more test data we can find more features of our algorithms and improve them.

## X. CONCLUSION

By doing this project, we have learned the basic ideas on how to make recommendation and we built a simple movie recommendation system with our own improvements. We realized two basic similarity algorithms and did some adjustment to make them more efficient. Also we realized user-based recommendation and item-based recommendation with adjustment. Then we did some test to evaluate and found that the user-based recommendation has a high accuracy than the item-based recommendation in our system. However, to validate our conclusion and make improvements we need to do more user tests in the future.

## XI. REFERENCE

[1] Greg Linden, Brent Smith, and Jeremy York • Amazon.com, “Recommendations, Item-to-Item Collaborative Filtering”, IEEE INTERNET COMPUTING, JANUARY/FEBRUARY, 2003.

[2] B.M. Sarwar et al., “Item-Based Collaborative Filtering Recommendation Algorithms,” 10th Int’l World Wide Web Conference, ACM Press, 2001, pp. 285-295.

[3] Toby Segaran, “Programming Collective Intelligence, Building Smart Web 2.0 Applications”, 2007.