Movie Recommendation Systems using Collaborative Filtering

Haoyu Li[*]
University of Rochester
Rochester, NY
hli73@ur.rochester.edu

Zenghe Huang[†]
University of Rochester
Rochester, NY
zhuang29@ur.rochester.edu

ABSTRACT

In this paper, we built up a recommendation system from scratch using the collaborative filtering. Two types of CF approach were investigated and implemented in this project, the user-based collaborative filtering and the item-based collaborative filtering. The movieLens 20M dataset was used and preprocessed to satisfiy the scope of this project. The theory behind the two algorithms and the implementation details were addressed in this paper. Several experiments were performed to proofread the correctness of the code. As the problem is essentially a regression problem, the mean square error was selected as the criteria to evaluate the two approaches.We were able to achieve an astonishingly **0.602** for the user-user CF and **0.578** for the item-item CF.

KEYWORDS

Movie Recommendation System, User-user collaborative Filtering, Item-item Collaborative Filtering, MovieLens, Pearson Correlations

ACM Reference Format:
Haoyu Li and Zenghe Huang. 2018. Movie Recommendation Systems using Collaborative Filtering. In *Proceedings of CSC 440 (CSC 440 Final Report).*

1 INTRODUCTION

There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized in most online platforms that we use. Given a large set of items and a description of the user's needs, they present to the user a small set of the items that are well suited to the description. Similarly, a movie recommendation system provides a level of comfort and personalization that helps the user interact better with the system and watch movies that cater to his needs. The main objective of this project is to build the Movie Recommendation Systems using the MovieLens data set. The recommendation system shall predict the ratings of a movie that the user haven't seen yet. Based on the predicted ratings, the top k movies with highest scores can be recommended to the users and satisfies the user's preference. We are trying to tackle this problem with User-User Collaborative Filtering approach and Item-Item Collaborative Filtering approach. In this paper, we will describe the preprocessing procedures, the technical approaches, and evaluation methods of both algorithms. We will compare one versus athe other in our results section.The dataset we used is MovieLens 20M Dataset, which includes tag genome data with 12 million relevance scores across 1,100 tags.

[*]Electrical and Computer Engineering Department
[†]Master of Alternative Energy

2 RELATED WORKS

We categorized existing movie recommendation approaches based on the techniques they employ as follows. The first approach is the Deep learning Neural Networks for Collaborative Filtering.[1] This approach expects the model to learn the values of embedding matrix itself. The latent features of both user and movie can be find out from embedding matrices. The approach can pass input to multiple relu, linear or sigmoid layers and learn the corresponding weights by any optimization algorithm.

The other existing approach is the content based recommendation.[2] This algorithm take into account that likes and dislikes of the user and generates a user profile. The user profile weighted sum up the item profiles according to the users' ratings. By calculating the cosine similarity of the user profile and item profile, it can generate the similarity of the user profile with all the items in the dataset. Advantages of Content Based approach is that data of other users is not required and the recommender engine can recommend new items which are not rated currently.

Another existing approach uses hybrid model to utilizes genomic tages of movie coupled with the content-based filtering to recommend similar movies.[3] This approach uses principal component analysis and Pearson correlation techniques to reduce the tags, then it result in finding similar type of movies and give personalized recommendation to users.

3 METHODOLOGY

Collaborative Filtering techniques make recommendations for a user based on ratings and preferences data of many users. The main underlying idea is that if two users have both liked certain common items, then the items that one user has liked that the other user. Collaborative filtering-based systems use the actions of users to recommend other items. In general, they can either be user based or item based. Item based collaborative filtering uses the patterns of users who browsed the same item as me to recommend me a product. User based collaborative filtering uses the patterns of users like other to recommend a product. There are several advantages for us to choose collaborative filtering to build up our Movie Recommendation System. The first thing is that it is easy and straightforward to implement. Meanwhile, it is content-independent, and it can provide us some serendipitous recommendations.

At the high level, we want to rank each item by assigning a score to them. There are several ways of calculating the score, such as PageRank, Reddit,HN,Bandit methods. For these algorithms, what we are doing is finding a score for these items called the score s(j),

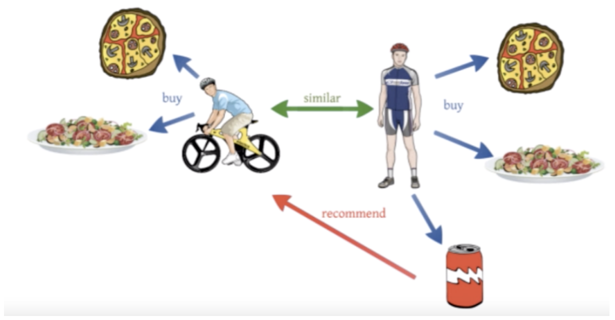


Figure 1: Collaborative Filtering

s means score and j refers to a particular item. So, for example, if we have M items, then j goes from 1 to M. However, the key is that the score is not personalized, it is non-specific to any particular user. No matter who you are, you will be seeing the same score for a particular item. The basic algorithm is to make s(j) the average rating for j, which can be expressed using eq.1

$$s(j) = \frac{\sum_{i \in \Omega_j} r_{ij}}{|\Omega_j|} \tag{1}$$

Ω_j is the set of all users who rated item j, and r_{ij} is the rating that user i gave to item j. However, our goal is to personalize the score, which means instead of using s(j) where the score only depends on the item j, we are going to use s(i,j), indicating the scores rely on both user i and item j. Therefore, in the following section, we will be modifying the eq.1 a little bit such that it can do what we are expecting. One of the most important concepts in recommendation system is the rating matrix. We use notation R to express the rating matrix. r_{ij} is the rating that user i gave for item j. As mentioned earlier, i goes from 1 to N and j goes from 1 to M. N is the number of users and M is the number of movies. So R_{N*M} is the user-item ratings matrix of size N by M. The user-item matrix is often really sparse, which means most entries are missing or empty. In other words, these entries are undefined or don't exist. However, this is not the same as saying these entries are zero. If the rating is zero, this is just saying the user gave a score of zero to the movie, however, this is not the case since the rating system provided by movieLens is from 0.5 to 5. Intuitively, the matrix is suppose to be sparse because the number of users is always much greater than the number of movies. The movies that a particular user have watched is only a small fraction in the movie database. If, for example, the matrix is dense instead of sparse, that means every user has watched every movie. If that's the case, we will have nothing to recommend.

3.1 User-User Collaborative Filtering

First, let's look at user-user collaborative filtering in a high level. User- User Collaborative Filtering uses the logic and recommends items by finding similar users to the active user to whom we are trying to recommend a movie. For example, User i has high ratings on movies 1, 2, 3, 4 while User i' like movies 2, 3, 4, 5. They have same taste on movies 2, 3 and 4. So the recommendation system will make prediction and recommend movie 5 to User i and recommend movie 1 to User i'. Recall in eq.1, the main limitation for this equation is that it is non-personalized. We don't even take into account of the user i at all. One way of fixing this is to put the weights on the ratings. Intuitively, we want the user who behaves similar to the recommended user matters more than the ones who behave different than the recommended user. The equation an be expressed as follows:

$$s(i,j) = \frac{\sum_{i \in \Omega_j} w_{ii'} r_{i'j}}{\sum_{i' \in \Omega_j} w_{ii'}} \tag{2}$$

In eq.2, $r_{i'j}$ means the rating that user i' gave on movie j. $w_{ii'}$ is the weight between user i and user i', we want this weight to be large if user i and user i' are in agreement, and small otherwise. One issue of using this average rating is that someone's interpretation regarding the ratings might be different to one another, in other words, user can be biased, they can be optimistic or they can be pessimistic. If they are optimistic, they might rate all good movies they see a five, and they find a terrible movie, they might gave a 3. But if they are very pessimistic, maybe they gave those movies they dislike a 1, and maybe they really like something, they gave those movies a 4. To overcome this problem, instead of calculating the absolute rating, we can calculate how much a user rates a movie deviate from their own average. This can be achieved because a user's average represents the bias. If a user has a very high average, that means this user is optimistic. Otherwise, if a user has a very low average, that means this user is pessimistic. For example, if a user likes Star Wars, then we can expect that the rating that this user gave to Star Wars must be much higher than the average rating of other movies that the user has ever given. So, the deviation can be expressed using eq.3

$$dev(i,j) = r(i,j) - \bar{r_i} \tag{3}$$

Therefore, the average we are interested is not an average of ratings but rather an average of deviation. The predicted deviation is the average of other deviations, which can be expressed using eq.4

$$\hat{dev}(i,j) = \frac{1}{|\Omega_j|} \sum_{i' \in \Omega_j} r(i',j) - \bar{r_{i'}} \tag{4}$$

This is just the sum of all deviations for all users who rated item j divided by the number of users who rated item j. Once we have the predicted deviation, we can just add that to the user's own average rating to get the predicted rating. Our final step is just to add the weighting back to the deviation formula as shown in eq.4. Now that the score for user i and item j can be expressed using the equation shown as follows:

$$s(i,j) = \bar{r_i} + \frac{\sum_{i' \in \Omega_j} w_{ii'} (r_{i'j} - \bar{r_{i'}})}{\sum_{i' \in \Omega_j} |w_{ii'}|} \tag{5}$$

This is equal to the weighted average deviation for item j added to user i's own average rating. Noted that we put an absolute value sign on the weights in the denominator, because these weights can actually be negative. Now that we can use the Pearson correlation coefficient to calculate the weights. The standard form of Pearson correlation coefficient is really straightforward, which can be expressed using eq.6

$$Q_{xy} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{6}$$

We can't use this formula directly, because our matrix is sparse. We have to modify this formula such that it can meet our purposes. Since we don't have all the data we need, we just gonna make use of the data we do have. The modified equation is given in eq.7

$$w_{ii'} = \frac{\sum_{j\in\psi_{ii'}}(r_{ij} - \bar{r_i})(r_{i'j} - \bar{r_{i'}})}{\sqrt{\sum_{j\in\psi_i}(r_{ij} - \bar{r_i})^2}\sqrt{\sum_{j\in\psi_{i'}}(r_{i'j} - \bar{r_{i'}})^2}} \tag{7}$$

In this equation, ψ_i is the set of movies j that user i has rated. $\psi_{i'}$ is the set of movies j that both user i and user i' have rated. Set of $\psi_{ii'}$ is just the intersection between the set ψ_i and $\psi_{i'}$. In our detailed implementation, if the two users have zero or just a few movies in common, we just don't consider these users at all because there's no way for us to find a correlation between these users. In fact, we set a lower threshold for how many movies two users must have seen in common before we even bother to calculate a weight. This is just standard probabilistic reasoning, if we don't have that many samples, then our estimation won't be accurate. In our code, we set the threshold to be five, which means if the two users don't have at least 5 movies in common, these two users will never be in each others prediction. The final thing we need to deal with is that, normally, we don't consider all other users in our calculation, or rather just the most similar or the most correlated users. The main advantage of doing this is that it will make our code runs much faster. If the number of users is large, then doing the summation in eq.5 will take a long time. Since want to be able to sort our items in our database, that means we have to do the same summation for all items in our database. One thing we can do to speed up the calculation is to precompute the weights beforehand, then reuse them. Another thing is that instead of summing over all users, we only take the ones with the highest weight. We just keep track of the k most similar users to each other. In order to achieve that, we used an ordered list in our calculation, the list will be maintaining a size of k. Since the list is ordered, any weights that less than the kth element will be removed from the list. For a user-user based recommendation system, we use k somewhere between 25 to 50. In our experiment, we tried 25,35, and 50, it turned our that having a neighbor size of 25 giving us the most promising results.

3.2 Item-Item Collaborative Filtering

Unlike the user-user collaborative filtering that looks at the user-item matrix row-wisely, the item-item collaborative filtering looks the matrix column-wisely. Namely, instead of finding two users are similar, we find 2 movies are similar in this approach. As shown in fig.2, we say two items are similar if their column vector's distance is small. In order to find two items are similar, we can use the Pearson correlation coefficient again. For user-user, we want to sum over all the items, however, for item-item, we want to sum over all users. The item-correlation is shown in eq.8

$$w_{jj'} = \frac{\sum_{i\in\Omega_{jj'}}(r_{ij} - \bar{r_j})(r_{ij'} - \bar{r_{j'}})}{\sqrt{\sum_{i\in\Omega_j}(r_{ij} - \bar{r_j})^2}\sqrt{\sum_{i\in\Omega_{j'}}(r_{ij'} - \bar{r_{j'}})^2}} \tag{8}$$

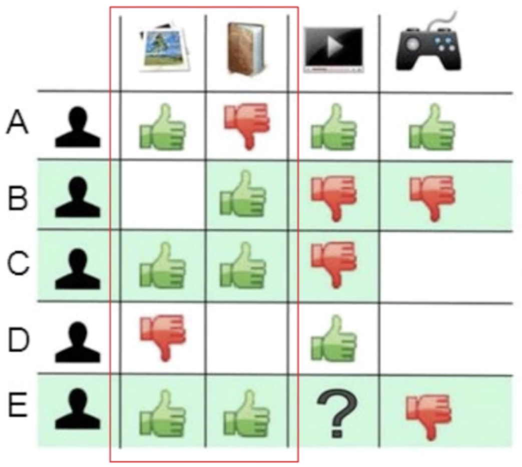


Figure 2: Item-Item Collaborative Filtering

In this equation, Ω_j is the users who rated item j. $\Omega_{jj'}$ is the users who rated both item j and item j'. $\bar{r_j}$ is the average rating for item j. Now that we have the item-item similarity, we can naturally come up with the item-score equation for item-item collaborative filtering. The equation can be found in eq.9

$$s(i,j) = \bar{r_j}\frac{\sum_{j'\in\psi_i} w_{jj'}(r_{ij'} - \bar{r_{j'}})}{\sum_{j'\in\psi_i}|w_{jj'}|} \tag{9}$$

$w_{jj'}$ is the weight between item j and item j', which tells us how similar these two items are. Recall the ψ_i is the set of items that the user i has rated. The deviation term which is multiplied by the weight term telling us how much better user i rates item j' compare to the average rating for j' over all users. In other words, this is how much user i likes j' comparing to everyone else. Conceptually, the logic behind the item-item collaborative filtering is pretty much the same as the user-user collaborative filtering. The key difference is that, for user-user collaborative filtering, it choose items for a user, because those items have been liked by similar users. Nevertheless for item-item collaborative filtering, it chooses items for a user, because this user has liked similar items in the past. Another difference is that, when comparing 2 items, we have much more data to use than when comparing 2 users. In other words, two items have much more users in common than two users have items in common. For example, in our 20M movieLens dataset, we have around 20k movies and up to 100k users, we just have more users to choose from. Another critical difference that is really influential in practice is that item-item CF runs much faster than user-user CF. If we want to calculate scores for each item for some users, we need to calculate the weights between each item and every other items, that's $O(M^2N)$, such that there are M^2 item-item weights, and for each correlation, the vector length is N. Compare that to user-user collaborative filtering, where we have a time complexity of $O(M^2N)$. Again, in recommendation system, the number of users N is much greater than the number of movies M, which makes the user-based CF run much slower. Similar to the user-user CF, we need to limit the number of neighbor k used in the calculation. By tring out different k values, we found that k equals to 15 works best for us. In the experiment section, we will discuss the item-item CF can actually obtain a slightly more accurate results comparing to the user-user CF.

4 EXPERIMENT

4.1 Preprocessing

4.1.1 Relabeling. In this project, the database we used is the movieLens 20M dataset. The reason why we choose this dataset over the 1M dataset that is we want to be able to extract the most effective information from the databse, and 20M dataset has everything we need. There are two files that we used in this project. The first one is the rating.csv file, and the second one is movie.csv file. The rating.csv is structured like a table, the columns are userId, movieId, rating, and timestamp. Immediately, we know that we can ignore the timestamp because our CF algorithms do not make use of it. We looked into the database using *panda.read_csv* function, and it turned out that the user ids go from 1 to 100k. However, we need to convert it to a 2D numpy array, therefore, we relabeled the userId such that it starts from 0. Similarly, we also need to relabel the movieIds. The movieIds also go from 1 to 100k. Unlike the userIds, they are not sequential. In fact, the dataset contains only about 20k movies, which is an order of magnitude less than the value of maximum movieId. We created a new mapping that starts from zero and only counts as much as we have to.

4.1.2 Dataset Shrinking. The second step we performed for preprocessing is to shrink the dataset. The full dataset is too large to perform an $O(N^2)$ algorithm. So the idea we used to shrink the dataset is that we only want to select a subset of users and a subset of movies. We would like to have as much as information about each user and each movie as possible. So, we should choose the users who have rated the most movies, and the movies who've been rated by most users. In our code, we select N(numebr of users) to be 1000, and M(number of movies) to be 50. Once we've have diminished the size of the dataset, we then save it to a file named partialRating.csv, and this file will be used later to create our adjacency lists.

4.1.3 Inner Join the two tables. Once dataset have been shrink, we can merge the partialRating.csv file with movies.csv using inner join. Because we want to know what's the semantic meaning of these movieId represent for. Therefore, we need to create the mapping between the movie titles and their ids. In our case, the number of movies selected is 50, so we will have 50 corresponding names. In order to have a general impression on what's our selected movies are, we used wordCloud to visualize the titles of the movie, and the results is shown in fig.3.

The larger the font size is, the more frequent the words appear.

4.1.4 Convert to adjacency lists(Using HashMaps). The final step for preprocessing is to convert the table to adjacency lists. The table is not an ideal data structure to access the data, especially when the table is extremely sparse. It will cause a lot of redundant overheads in terms of space and time complexity. Therefore, we need to convert the table to HashMaps, which is one of the most effective data structures accessing data using key-value pairs. Also in the code, we want to know given a user i, which movies j did they

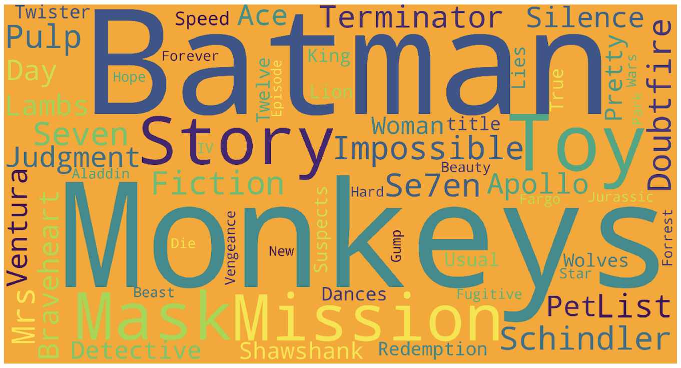


Figure 3: Shrinked Dataset Exploration using WordCloud

rate. So we create a HashMap entitled userToMovie(j) to achieve that. Similarly, in order to know given a movie j, which user i have rated it before, we need a HashMap called movieToUser(j). In order for us to know given user i and movie j, what's the rating, we need a HashMap called userMovieToRating(i,j).It takes two keys to allocate the value, the user id i and the movie id j. One last thing we need to do is to create the HashMap that uses movieIds as the keys, and the movie titles as the values. Having all the HashMap created, we can make the key-value lookup in a O(1) time, and looping through the HashMap is only the size of Ω, where Ω is the set of ratings.

4.2 User-user collaborative filtering versus item-item collaborative filtering

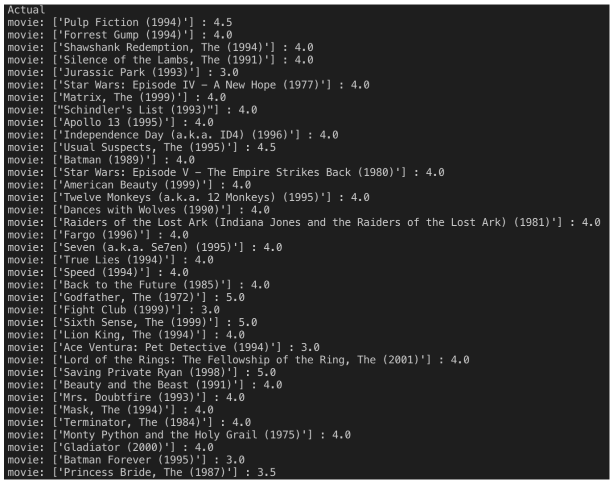
Next, we will show how the user-user and item-item collaborative filtering perform using a case study. First of all, the parameters we used for running the two algorithms are as follows:

	User-User	**Item-Item**
Number of Movies(N)	1000	1000
Number of Users(M)	50	50
Number of Neighbors(K)	25	15
Threshold	5	5

Table 1: Parameters used to run the two CFs

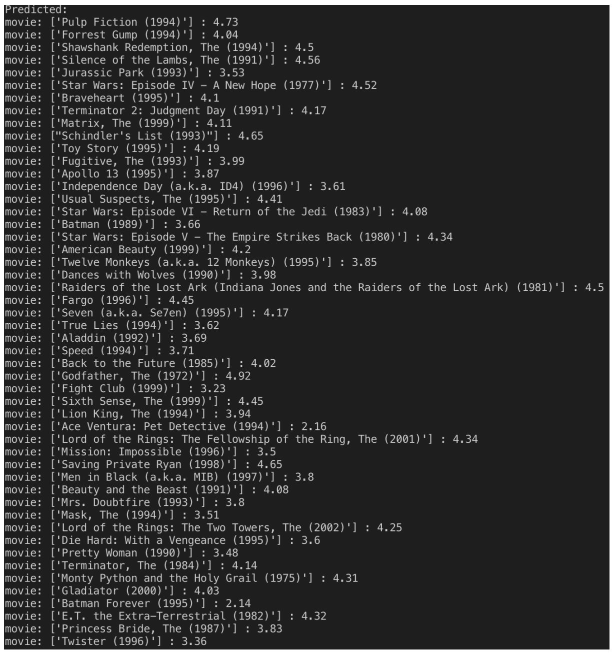
We selected the user with id 0 in our diminished dataset as an example. As can be seen in fig.4, the user has watched 43 of the 50 selected movies initially. Which means we can still predict the ratings of the other 7 movies that the user hasn't previously seen, and make a recommendation based on the prediction. fig.5 is the predicted ratings for user with id 0 using user-user based approach. We can see that, for example, the movie *Twister*(1996) which the user hasn't seen before, has a predicted rating of 3.36, which means the user might not in favor of this movie. For the movies that the user has seen before, it's not hard to see that the predicted ratings are pretty similar to the user's own rating.

fig.6 is the predicted ratings for user with id 0 using item-item approach, we will be quantitatively discussing the accuracy in the following section.



```
Actual
movie: ['Pulp Fiction (1994)'] : 4.5
movie: ['Forrest Gump (1994)'] : 4.0
movie: ['Shawshank Redemption, The (1994)'] : 4.0
movie: ['Silence of the Lambs, The (1991)'] : 4.0
movie: ['Jurassic Park (1993)'] : 3.0
movie: ['Star Wars: Episode IV - A New Hope (1977)'] : 4.0
movie: ['Matrix, The (1999)'] : 4.0
movie: ["Schindler's List (1993)"] : 4.0
movie: ['Apollo 13 (1995)'] : 4.0
movie: ['Independence Day (a.k.a. ID4) (1996)'] : 4.0
movie: ['Usual Suspects, The (1995)'] : 4.5
movie: ['Batman (1989)'] : 4.0
movie: ['Star Wars: Episode V - The Empire Strikes Back (1980)'] : 4.0
movie: ['American Beauty (1999)'] : 4.0
movie: ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] : 4.0
movie: ['Dances with Wolves (1990)'] : 4.0
movie: ['Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)'] : 4.0
movie: ['Fargo (1996)'] : 4.0
movie: ['Seven (a.k.a. Se7en) (1995)'] : 4.0
movie: ['True Lies (1994)'] : 4.0
movie: ['Speed (1994)'] : 4.0
movie: ['Back to the Future (1985)'] : 4.0
movie: ['Godfather, The (1972)'] : 5.0
movie: ['Fight Club (1999)'] : 3.0
movie: ['Sixth Sense, The (1999)'] : 5.0
movie: ['Lion King, The (1994)'] : 4.0
movie: ['Ace Ventura: Pet Detective (1994)'] : 3.0
movie: ['Lord of the Rings: The Fellowship of the Ring, The (2001)'] : 4.0
movie: ['Saving Private Ryan (1998)'] : 5.0
movie: ['Beauty and the Beast (1991)'] : 4.0
movie: ['Mrs. Doubtfire (1993)'] : 4.0
movie: ['Mask, The (1994)'] : 4.0
movie: ['Terminator, The (1984)'] : 4.0
movie: ['Monty Python and the Holy Grail (1975)'] : 4.0
movie: ['Gladiator (2000)'] : 4.0
movie: ['Batman Forever (1995)'] : 3.0
movie: ['Princess Bride, The (1987)'] : 3.5
```

Figure 4: The ratings that a specific user gave among the selected m(50) movies



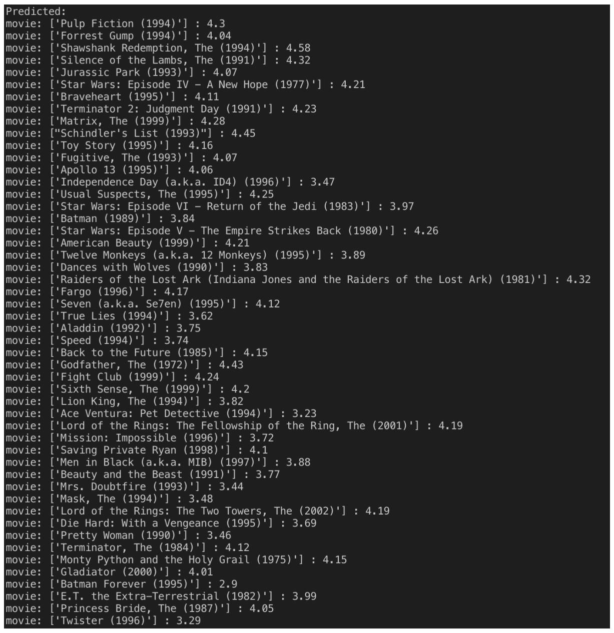
```
Predicted:
movie: ['Pulp Fiction (1994)'] : 4.73
movie: ['Forrest Gump (1994)'] : 4.04
movie: ['Shawshank Redemption, The (1994)'] : 4.5
movie: ['Silence of the Lambs, The (1991)'] : 4.56
movie: ['Jurassic Park (1993)'] : 3.53
movie: ['Star Wars: Episode IV - A New Hope (1977)'] : 4.52
movie: ['Braveheart (1995)'] : 4.1
movie: ['Terminator 2: Judgment Day (1991)'] : 4.17
movie: ['Matrix, The (1999)'] : 4.11
movie: ["Schindler's List (1993)"] : 4.65
movie: ['Toy Story (1995)'] : 4.19
movie: ['Fugitive, The (1993)'] : 3.99
movie: ['Apollo 13 (1995)'] : 3.87
movie: ['Independence Day (a.k.a. ID4) (1996)'] : 3.61
movie: ['Usual Suspects, The (1995)'] : 4.41
movie: ['Star Wars: Episode VI - Return of the Jedi (1983)'] : 4.08
movie: ['Batman (1989)'] : 3.66
movie: ['Star Wars: Episode V - The Empire Strikes Back (1980)'] : 4.34
movie: ['American Beauty (1999)'] : 4.2
movie: ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] : 3.85
movie: ['Dances with Wolves (1990)'] : 3.98
movie: ['Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)'] : 4.5
movie: ['Fargo (1996)'] : 4.45
movie: ['Seven (a.k.a. Se7en) (1995)'] : 4.17
movie: ['True Lies (1994)'] : 3.62
movie: ['Aladdin (1992)'] : 3.69
movie: ['Speed (1994)'] : 3.71
movie: ['Back to the Future (1985)'] : 4.02
movie: ['Godfather, The (1972)'] : 4.92
movie: ['Fight Club (1999)'] : 3.23
movie: ['Sixth Sense, The (1999)'] : 4.45
movie: ['Lion King, The (1994)'] : 3.94
movie: ['Ace Ventura: Pet Detective (1994)'] : 2.16
movie: ['Lord of the Rings: The Fellowship of the Ring, The (2001)'] : 4.34
movie: ['Mission: Impossible (1996)'] : 3.5
movie: ['Saving Private Ryan (1998)'] : 4.65
movie: ['Men in Black (a.k.a. MIB) (1997)'] : 3.8
movie: ['Beauty and the Beast (1991)'] : 4.08
movie: ['Mrs. Doubtfire (1993)'] : 3.8
movie: ['Mask, The (1994)'] : 3.51
movie: ['Lord of the Rings: The Two Towers, The (2002)'] : 4.25
movie: ['Die Hard: With a Vengeance (1995)'] : 3.6
movie: ['Pretty Woman (1990)'] : 3.48
movie: ['Terminator, The (1984)'] : 4.14
movie: ['Monty Python and the Holy Grail (1975)'] : 4.31
movie: ['Gladiator (2000)'] : 4.03
movie: ['Batman Forever (1995)'] : 2.14
movie: ['E.T. the Extra-Terrestrial (1982)'] : 4.32
movie: ['Princess Bride, The (1987)'] : 3.83
movie: ['Twister (1996)'] : 3.36
```

Figure 5: Predicted ratings for a specific user using user-based CF

4.3 Actual Ratings versus Predicted Ratings

Since this is actually a regression problem, we evaluated the mean square error. We took our model predicted ratings and compared them to actual ratings, square the difference, and then got the average of these square differences. The mean square error can be expressed using eq.10.

$$MSE = \frac{1}{|\Omega|} \sum_{i,j \in \Omega} (r_{ij} - \hat{r_{ij}})^2 \tag{10}$$



```
Predicted:
movie: ['Pulp Fiction (1994)'] : 4.3
movie: ['Forrest Gump (1994)'] : 4.04
movie: ['Shawshank Redemption, The (1994)'] : 4.58
movie: ['Silence of the Lambs, The (1991)'] : 4.32
movie: ['Jurassic Park (1993)'] : 4.07
movie: ['Star Wars: Episode IV - A New Hope (1977)'] : 4.21
movie: ['Braveheart (1995)'] : 4.11
movie: ['Terminator 2: Judgment Day (1991)'] : 4.23
movie: ['Matrix, The (1999)'] : 4.28
movie: ["Schindler's List (1993)"] : 4.45
movie: ['Toy Story (1995)'] : 4.16
movie: ['Fugitive, The (1993)'] : 4.07
movie: ['Apollo 13 (1995)'] : 4.06
movie: ['Independence Day (a.k.a. ID4) (1996)'] : 3.47
movie: ['Usual Suspects, The (1995)'] : 4.25
movie: ['Star Wars: Episode VI - Return of the Jedi (1983)'] : 3.97
movie: ['Batman (1989)'] : 3.84
movie: ['Star Wars: Episode V - The Empire Strikes Back (1980)'] : 4.26
movie: ['American Beauty (1999)'] : 4.21
movie: ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] : 3.89
movie: ['Dances with Wolves (1990)'] : 3.83
movie: ['Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)'] : 4.32
movie: ['Fargo (1996)'] : 4.17
movie: ['Seven (a.k.a. Se7en) (1995)'] : 4.12
movie: ['True Lies (1994)'] : 3.62
movie: ['Aladdin (1992)'] : 3.75
movie: ['Speed (1994)'] : 3.74
movie: ['Back to the Future (1985)'] : 4.15
movie: ['Godfather, The (1972)'] : 4.43
movie: ['Fight Club (1999)'] : 4.24
movie: ['Sixth Sense, The (1999)'] : 4.2
movie: ['Lion King, The (1994)'] : 3.82
movie: ['Ace Ventura: Pet Detective (1994)'] : 3.23
movie: ['Lord of the Rings: The Fellowship of the Ring, The (2001)'] : 4.19
movie: ['Mission: Impossible (1996)'] : 3.72
movie: ['Saving Private Ryan (1998)'] : 4.1
movie: ['Men in Black (a.k.a. MIB) (1997)'] : 3.88
movie: ['Beauty and the Beast (1991)'] : 3.77
movie: ['Mrs. Doubtfire (1993)'] : 3.44
movie: ['Mask, The (1994)'] : 3.48
movie: ['Lord of the Rings: The Two Towers, The (2002)'] : 4.19
movie: ['Die Hard: With a Vengeance (1995)'] : 3.69
movie: ['Pretty Woman (1990)'] : 3.46
movie: ['Terminator, The (1984)'] : 4.12
movie: ['Monty Python and the Holy Grail (1975)'] : 4.15
movie: ['Gladiator (2000)'] : 4.01
movie: ['Batman Forever (1995)'] : 2.9
movie: ['E.T. the Extra-Terrestrial (1982)'] : 3.99
movie: ['Princess Bride, The (1987)'] : 4.05
movie: ['Twister (1996)'] : 3.29
```

Figure 6: Predicted ratings for a specific user using item-based CF

The Ω is just the set of pairs (i,j) where user i has rated item j. By running the two algorithms using the parameters shown in tab.1, we were able to get $MSE_{user-user}$ to be 0.602, and $MSE_{item-item}$ to be 0.578, which are very promising. fig.7 is just a bar chart comparing the actual ratings and the predicted ratings using item-item collaborative filtering. The blue bar indicates the predicted ratings whereas the green bar indicates the actual ratings.

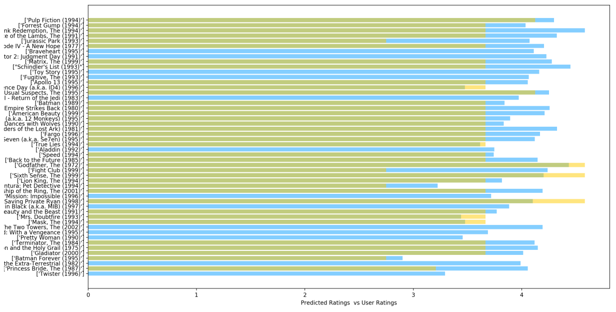


Figure 7: Prediction using item-based CF versus user-rated movies

5 CONCLUSIONS

In this project, we proposed a movie recommendation system based on Collaborative Filtering techniques. Both User-User Collaborative Filtering and Item-Item Collaborative Filtering are applied. Compared with previous publications, we enhanced the efficiency of collaborative filtering approach by preprocessing and select the effective data. Our experiment on preprocessing steps reduce the redundancy of useless data, and increase the efficiency when applying the algorithms. And the WordCloud module provided a visualized understanding of popularity of movies' keyword. To compare two approaches, we evaluated the mean square error for both collaborative filtering method, 0.602 for the user-user CF and 0.578 for the item-item CF. The overall results that both approaches predict are reasonable while item-item CF can provide a more accurate prediction.

REFERENCES

[1] James Le, http://nbviewer.jupyter.org/github/khanhnamle1994/movielens/blob/master/Deep_Learning_Model.ipynb
[2] Prateek Sappadla, Yash Sadhwani, Pranit Arora, Movie Recommender System, Search Engine Architecture, Spring 2017,
[3] Syed M. Ali, Gopal K. Nayak, Rakesh K. Lenka, Rabindra K. Barik, Movie Recommendation System Using Genome Tags and Content-Based Filtering, 08 April 2018