Report on User-based CF Movie Recommender System

Shiyi Wang

College of Computing, Georgia Institute of Technology

CS 4675: Advanced Internet Computing Systems and Application Development

Dr. Ling Liu

February 15, 2022

Report on User-based CF Movie Recommender System

This is a movie recommender system utilizing user-based collaborative filtering in Python. The training dataset is the MovieLens 20M Dataset from Kaggle.

1. Dataset Preparation

1.1 Loading Dataset

The dataset provided on Canvas does not include Movie names. Therefore, it is hard to evaluate the accuracy. Therefore, the dataset I choose is the MovieLens 20M Dataset from Kaggle. The dataset delineates ratings from MovieLens which is an online movie recommendation service. The dataset includes more than 20,000,000 ratings across more than 27,000 movies. These data were collected between 1995 and 2015.

	userId	movieId	rating	title
0	1	2	3.5	Jumanji (1995)
1	1	29	3.5	City of Lost Children, The (Cité des enfants p...
2	1	32	3.5	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
3	1	47	3.5	Seven (a.k.a. Se7en) (1995)
4	1	50	3.5	Usual Suspects, The (1995)
...	...	...	...	...
20000258	138493	68954	4.5	Up (2009)
20000259	138493	69526	4.5	Transformers: Revenge of the Fallen (2009)
20000260	138493	69644	3.0	Ice Age: Dawn of the Dinosaurs (2009)
20000261	138493	70286	5.0	District 9 (2009)
20000262	138493	71619	2.5	Coco Before Chanel (Coco avant Chanel) (2009)

20000263 rows × 4 columns

Figure 1 MovieLens 20M Dataset Description

There are two files in the dataset: `movie.csv` and `rating.csv`. `movie.csv` contains movie information including attributes such as `movieId`, `title`, and `genres`.

`rating.csv` contains ratings of movies by users, including attributes such as `userId`, `movieId`, `rating`, and `timestamp`.

Given that both files contain the attribute `movieId`, we can merge them together. I dropped the `timestamp` and `genres` columns in `movie.csv` since we don't need them until analysis. Therefore, the merged dataframe includes only three columns: `userId`, `movieId`, and `rating`.

1.2 Data Visualization

In order to develop a better understanding of the dataset we are using. I implemented some data visualizations (Etezadi, 2022). First. we can find the count, mean, standard deviation, and percentile information of each movie's rating. It is noticeable that the title contains a variety of languages. Luckily, Jupyter Notebook is capable of handling such situation.

```
movie_ratings.groupby('title')['rating'].describe()
```

title	count	mean	std	min	25%	50%	75%	max
#chicagoGirl: The Social Network Takes on a Dictator (2013)	3.0	3.666667	1.154701	3.0	3.000	3.00	4.00	5.0
$ (Dollars) (1971)	24.0	2.833333	1.138904	0.5	2.500	3.00	3.50	4.5
$5 a Day (2008)	39.0	2.871795	1.122251	0.5	2.250	3.00	3.50	4.5
$9.99 (2008)	55.0	3.009091	1.211408	0.5	2.500	3.00	4.00	5.0
$ellebrity (Sellebrity) (2012)	2.0	2.000000	2.121320	0.5	1.250	2.00	2.75	3.5
...	...	...	...	...	...	...	...	...
À propos de Nice (1930)	4.0	3.125000	0.478714	2.5	2.875	3.25	3.50	3.5
Árido Movie (2005)	1.0	2.000000	NaN	2.0	2.000	2.00	2.00	2.0
Åsa-Nisse - Wälkom to Knohult (2011)	2.0	1.500000	0.000000	1.5	1.500	1.50	1.50	1.5
Üvegtigris (2001)	1.0	3.000000	NaN	3.0	3.000	3.00	3.00	3.0
貞子3D (2012)	1.0	1.500000	NaN	1.5	1.500	1.50	1.50	1.5

Figure 2 Statistical Description of the Dataset

We can also find the mean rating frequency distribution below. Similarly, we can also find the distribution of count frequency. Based on the graph on the left, we see that there is a relatively normal distribution on the ratings of movies by the users. Most users give 3.5 points

for some movies. There are also cases such that the rating can be extreme. On the right, it demonstrates the diversity of the dataset where the count frequency varies.

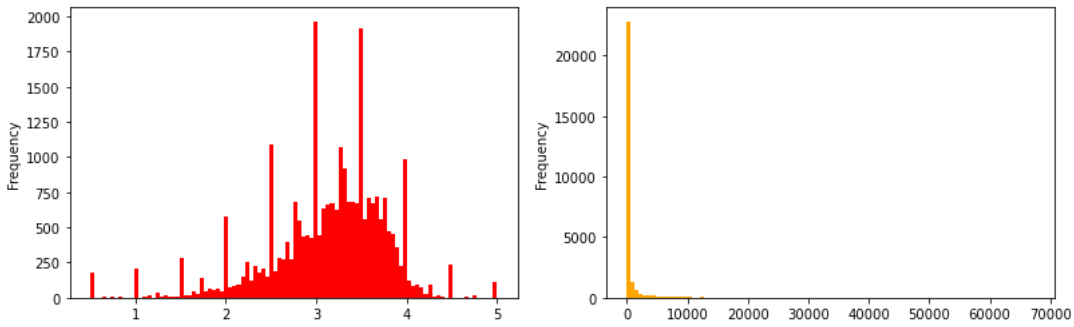

Figure 3 Mean and count of Rating Frequency Distribution

Finally, we can take a look at the most rated movies. Essentially, there are movies will a considerable amount of count with means with high accuracy given 6 decimal places.

title	count	mean
Pulp Fiction (1994)	67310.0	4.174231
Forrest Gump (1994)	66172.0	4.029000
Shawshank Redemption, The (1994)	63366.0	4.446990
Silence of the Lambs, The (1991)	63299.0	4.177057
Jurassic Park (1993)	59715.0	3.664741
...	...	...
Net, The (1995)	24618.0	3.110427
Rain Man (1988)	24591.0	3.899170
There's Something About Mary (1998)	24582.0	3.546294
Shakespeare in Love (1998)	24521.0	3.835794
Taxi Driver (1976)	24481.0	4.110576

100 rows × 2 columns

Figure 4 Mostly Rated Movies

1.3 Data Preprocessing

I truncated the dataframe to 700,000 rows so that it won't overflow the pivot table while maintaining the recommendation performance. If there are too many rows, it cannot be fully

parsed into the pivot table. On the other hand, if there are too few rows, we will not be able to predict a relatively accurate list of results.

	userId	movieId	rating	title
0	1	2	3.5	Jumanji (1995)
1	1	29	3.5	City of Lost Children, The (Cité des enfants p...
2	1	32	3.5	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
3	1	47	3.5	Seven (a.k.a. Se7en) (1995)
4	1	50	3.5	Usual Suspects, The (1995)
...	...	...	...	...
699995	4640	1517	4.0	Austin Powers: International Man of Mystery (1...
699996	4640	1580	4.0	Men in Black (a.k.a. MIB) (1997)
699997	4640	1645	5.0	Devil's Advocate, The (1997)
699998	4640	1717	4.0	Scream 2 (1997)
699999	4640	1721	3.0	Titanic (1997)

700000 rows × 4 columns

Figure 5 Truncated Dataframe (700,000)

2. Similarity Matrix

2.1 Movie Similarity

First, let's create a userId-title table to demonstrate the ratings for a specific user given a specific movie title. The entries in `mat` are mostly 0s since most users have not seen most movies. However, if a user has rated a movie, the entry that corresponds to the userId and title is the user's rating of the movie.

																					rating
	userId	1	2	3	4	5	6	7	8	9	10	...	4631	4632	4633	4634	4635	4636	4637	4638	4639	4640
title	movieId																					
Toy Story (1995)	**1**	0.0	0.0	4.0	0.0	0.0	5.0	0.0	4.0	0.0	4.0	...	0.0	0.0	0.0	3.0	3.0	5.0	0.0	0.0	0.0	0.0
Jumanji (1995)	**2**	3.5	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	4.0
Grumpier Old Men (1995)	**3**	0.0	4.0	0.0	0.0	0.0	3.0	3.0	5.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Waiting to Exhale (1995)	**4**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Father of the Bride Part II (1995)	**5**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	**...**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Clown (2014)	**130052**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Cinderella (2015)	**130073**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
The Dark Knight (2011)	**130219**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Insurgent (2015)	**130490**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Backcountry (2014)	**130642**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

13383 rows × 4640 columns

Figure 6 userId-title table

To create a movie similarity matrix based on cosine similarity, we need to import `NearestNeighbors` from scikit-learn. We set the metric to cosine, algorithm to brute, and n_neighbors to 6. According to Scikit-Learn, we use brute here because fitting on sparse input matrix will override the parameter setting to brute force. Although we are predicting 5 neighbors per requirement, n_neighbors are set to 6 because the most similar movie is the movie itself.

```
from sklearn.neighbors import NearestNeighbors
knn = NearestNeighbors(metric='cosine', algorithm='brute')
knn.fit(mat.values)
neigh_dist, neigh_ind = knn.kneighbors(mat.values, n_neighbors=6)
```

Figure 7 Parameters for KNN

`neigh_ind` describes the nearest neighbor movies of each movie. Each row in `neigh_ind` corresponds to each row in `mat`. In each row in `neigh_ind`, the movies are ranked from the most similar movie to the least similar movie.

```
array([[    0,   253,   743,  1145,  2919,  1203],
       [    1,   359,   356,   488,   468,   570],
       [    2,     4,   702,   749,   751,   482],
       ...,
       [13324, 13380, 13211,  9336, 13081, 13180],
       [13381,  9613, 11630, 12499,  6177, 12528],
       [ 7664, 12701,  8898, 12622, 11654, 12610]])
```

Figure 8 neigh_ind

`neigh_dist describes` the distance from each neighbor movie to the target movie. It is intuitive that the value in each row is in ascending order since they are getting less similar.

```
array([[0.        , 0.42265154, 0.42947531, 0.44775331, 0.44923923,
        0.45268639],
       [0.        , 0.47450917, 0.4961168 , 0.49966338, 0.50476266,
        0.5140506 ],
       [0.        , 0.51050749, 0.57075452, 0.58131416, 0.58538633,
        0.58723462],
       ...,
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.2       ],
       [0.        , 0.03847605, 0.03847605, 0.03847605, 0.03847605,
        0.03847605],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        ]])
```

Figure 9 neigh_dist

Based on the similarity matrix, now we can output the most similarly rated movies by unpacking the NumPy arrays `neigh_ind` and `neigh_dist` above. Detail implementation can be found in Jupyter Notebook. The result is shown below.

```
Finding 3 similarly rated movies:

The top 5 most similarly rated movies of Toy Story (1995) are:
No.1: Star Wars: Episode IV - A New Hope (1977), with distance 0.4226515398719132
No.2: Independence Day (a.k.a. ID4) (1996), with distance 0.4294753071589563
No.3: Star Wars: Episode VI - Return of the Jedi (1983), with distance 0.4477533119844487
No.4: Toy Story 2 (1999), with distance 0.4492392254654526
No.5: Back to the Future (1985), with distance 0.4526863884459026


The top 5 most similarly rated movies of Jumanji (1995) are:
No.1: Mask, The (1994), with distance 0.4745091705417712
No.2: Lion King, The (1994), with distance 0.4961167994981186
No.3: Mrs. Doubtfire (1993), with distance 0.4996633824666885
No.4: Jurassic Park (1993), with distance 0.5047626585955509
No.5: Home Alone (1990), with distance 0.5140506036160686


The top 5 most similarly rated movies of Grumpier Old Men (1995) are:
No.1: Father of the Bride Part II (1995), with distance 0.5105074861100034
No.2: Twister (1996), with distance 0.5707545225766049
No.3: Eraser (1996), with distance 0.5813141610420868
No.4: Nutty Professor, The (1996), with distance 0.5853863316289671
No.5: Executive Decision (1996), with distance 0.5872346222625873
```

Figure 10 similarlyRatedMovies output

I mostly agree with the recommendations. The recommendations seem consistent. The movie release years are also similar. This indicates that these movies are probably rated together.

In the first set of recommendations for Toy Story (1995), Toy Story (1995) and Toy Story 2 (1999) are filmed in series. Coincidentally, Toy Story 2 (1999) was released in theaters the same year as Star Wars Episode I: The Phantom Menace. In the second and third set of recommendations the movies recommended are mostly fallen within the same or similar genre. Jumanji (1995) is classified as Adventure, Children, Fantasy. The recommended ones are the Lion King (1994) – Adventure, Mrs. Doubtfire (1993) – Drama, Jurassic Park (1993) – Adventure, and Home Alone (1990) – Children. The same situation applies for the third set of recommendations.

2.2 User Similarity

The similar logic applies for user similarity matrix. We need to transpose `mat`. The columns and rows are swapped from `mat`. Now, the similarity matrix is able to tell the similarity between users. By using the same code, we have `user_neigh_ind` and `user_neigh_dist`.

title	$5 a Day (2008)	'71 (2014)	'Hellboy': The Seeds of Creation (2004)	'Neath the Arizona Skies (1934)	'Round Midnight (1986)	'Salem's Lot (2004)	'Til There Was You (1997)	'burbs, The (1989)	'night Mother (1986)	(500) Days of Summer (2009)	...	[REC] (2007)	[REC]² (2009)	\\"Great Performances\"\" Cats (1998)"	eXistenZ (1999)	loudQUIETloud: A Film About the Pixies (2006)	xXx (2002)	xXx: State of the Union (2005)	¡Three Amigos! (1986)	À nos amours (1983
userId																				
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4636	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4637	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4638	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4639	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4640	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

4640 rows × 13378 columns

Figure 11 transpose of mat

```
array([[   0, 2594, 1747, 1838,  146, 4420],
       [   1, 3509, 2181, 3297, 2001, 1091],
       [   2,  474, 1175, 1846, 1106, 2923],
       ...,
       [4637, 1284,  194, 3501, 3006, 3102],
       [4638, 4265, 3501,  194, 2140, 2644],
       [4639,   77, 1943,  452, 3684, 1006]])
```

Figure 12 user_neigh_ind

```
array([[1.11022302e-16, 5.84641214e-01, 6.18955188e-01, 6.50451289e-01,
        6.51675425e-01, 6.51918429e-01],
       [0.00000000e+00, 6.15645239e-01, 6.47619059e-01, 6.75994885e-01,
        6.81983213e-01, 6.83178986e-01],
       [0.00000000e+00, 5.01885587e-01, 5.05011276e-01, 5.33630046e-01,
        5.48234070e-01, 5.50334659e-01],
       ...,
       [0.00000000e+00, 5.02309813e-01, 5.19370286e-01, 5.47746511e-01,
        5.48231025e-01, 5.63428249e-01],
       [3.33066907e-16, 7.25271570e-01, 7.44864046e-01, 7.54825711e-01,
        7.64966449e-01, 7.76313939e-01],
       [1.11022302e-16, 7.45365832e-01, 7.63225677e-01, 7.63451284e-01,
        7.71552010e-01, 7.76351595e-01]])
```

Figure 13 user_neigh_dist

Finally, we can output the most similarly rated movies by unpacking the NumPy arrays `user_neigh_ind` and `user_neigh_dist` above. Detailed implementation can be found on Jupyter Notebook.

```
Finding similar taste users for user 7

The top five most similar users of user 7 are:
No.1: User ID: 4295, with distance 0.6665695596522601
No.2: User ID: 1918, with distance 0.6741264963030452
No.3: User ID: 2002, with distance 0.6784827888727569
No.4: User ID: 54, with distance 0.6904163181371773
No.5: User ID: 424, with distance 0.6910934139698723
```

Figure 14 similarUsers() output

3. User-based Movie Recommendation System

The key concept behind is that when user A asks for a movie recommendation, our system recommends movies that user B rated highly, and the user A has not yet seen. First, I insert a new user with userId = 4641 with 10 movies ratings into the dataset.

	userId	movieId	rating	title
700000	4641	318	5.0	Shawshank Redemption, The (1994)
700001	4641	1721	5.0	Titanic (1997)
700002	4641	94466	2.0	Black Mirror (2011)
700003	4641	110553	4.0	The Amazing Spider-Man 2 (2014)
700004	4641	59315	4.5	Iron Man (2008)
700005	4641	59784	3.5	Kung Fu Panda (2008)
700006	4641	73829	1.0	Mega Shark vs. Giant Octopus (2009)
700007	4641	78631	1.0	Shark in Venice (2008)
700008	4641	80586	3.0	Flipped (2010)
700009	4641	81834	4.0	Harry Potter and the Deathly Hallows: Part 1 (...

Figure 15 Newly Inserted Data

With the newly imported data, I can calculate the new similarity matrix.

```
(array([[   0, 2594, 1747, 1838,  146, 4420],
        [   1, 3509, 2181, 3297, 2001, 1091],
        [   2,  474, 1175, 1846, 1106, 2923],
        ...,
        [4638, 4265, 3501,  194, 2140, 2644],
        [4639,   77, 1943,  452, 3684, 1006],
        [4640, 3419, 2318, 1702, 2953, 3978]]),
 array([[0.00000000e+00, 5.84641214e-01, 6.18955188e-01, 6.50451289e-01,
         6.51675425e-01, 6.51918429e-01],
        [0.00000000e+00, 6.15645239e-01, 6.47619059e-01, 6.75994885e-01,
         6.81983213e-01, 6.83178986e-01],
        [0.00000000e+00, 5.01885587e-01, 5.05011276e-01, 5.33630046e-01,
         5.48234070e-01, 5.50334659e-01],
        ...,
        [3.33066907e-16, 7.25271570e-01, 7.44864046e-01, 7.54825711e-01,
         7.64966449e-01, 7.76313939e-01],
        [1.11022302e-16, 7.45365832e-01, 7.63225677e-01, 7.63451284e-01,
         7.71552010e-01, 7.76351595e-01],
        [0.00000000e+00, 7.88849958e-01, 7.98466492e-01, 8.16596762e-01,
         8.17510386e-01, 8.26165677e-01]]))
```

Figure 16 User Similarity Matrix with New Data

Now, we can build a robust version of the user-based movie recommendation system. First, we incorporate the `findSimilarUsers` from the section above. Note here instead of just printing the results, I will return a tuple to pass the flatten `neigh_users_dist` and `neigh_users_ind`. One thing important is that we want to recommend movies that the user haven't seen yet. Therefore, I use the seen variable below to record the movies a user has rated so that the recommended ones will be new to the user.

```
def getRecommendations(num_movies_recommended, avg_rating, userId):
        # clean up the zero ratings
        zero_rating = np.where(avg_rating == 0)[0][-1]
        ranked_ind = np.argsort(avg_rating)[::-1]
        ranked_ind = ranked_ind[:list(ranked_ind).index(zero_rating)]
        # check input validity by comparing with movies we have
        num_movies_recommended = min(len(ranked_ind), num_movies_recommended)
        # store seen movies in a list
        seen = list(movie_ratings[movie_ratings['userId'] == userId]['title'])
        movies = list(tmat.columns[ranked_ind])
        # recommended count
        count = 0
        # store recommended movies in the list
        recommended_movies = []
        for movie in movies:
            # only append if not seen
            if movie not in seen:
                recommended_movies.append(movie)
                count += 1
            if count == num_movies_recommended:
                break

        pprint(recommended_movies)
```

Figure 17 Remove Duplicate Movies

After defining a driver function to pack these functions together for better functionality, I can use my new ratings where I have `userId = 4641`. Here, I want to consider `num_similar_users = 5` and `num_movies_recommended = 5`.

```
recommend(userId = 4641, num_similar_users = 5, num_movies_recommended = 5)
```

```
User 4641 has rated the following movies:
['Shawshank Redemption, The (1994)',
 'Titanic (1997)',
 'Black Mirror (2011)',
 'The Amazing Spider-Man 2 (2014)',
 'Iron Man (2008)',
 'Kung Fu Panda (2008)',
 'Mega Shark vs. Giant Octopus (2009)',
 'Shark in Venice (2008)',
 'Flipped (2010)',
 'Harry Potter and the Deathly Hallows: Part 1 (2010)']


The top 5 most similar users of user 4641 are:
No.1: User ID: 3420, with distance 0.7888499580953169
No.2: User ID: 2319, with distance 0.7984664918643324
No.3: User ID: 1703, with distance 0.8165967621652661
No.4: User ID: 2954, with distance 0.8175103861079487
No.5: User ID: 3979, with distance 0.8261656774920836


Based on other users rating, we recommend:
['Matrix, The (1999)',
 'Princess Bride, The (1987)',
 'Meet the Parents (2000)',
 'Pretty Woman (1990)',
 'Braveheart (1995)']
```

Figure 18 Recommendation Output Based on New Input

Here, the output is reasonable from a user perspective. The Matrix (1999) is scientific fiction and fantasy based, which, to some extent, is similar to Spider Man and Iron Man in the rating list. Other movies such as The Princess Bride (1987) is romantic which corresponds to Titanic (1997) I rated high. I conclude that the user-based collaborative filtering recommender system is successful.

Future Applications and Extension

The extension of this project can be a quantitative instead of qualitative evaluation metric on the result of recommendations. I read the Microsoft paper regarding *Empirical Analysis of Predictive Algorithms for Collaborative Filtering*. The paper describes two metrics. The first

measures accuracy in terms of average absolute deviation over a group of individual forecasts. The second calculates how valuable a ranked list of suggested items is. (Breese et al., 1998). The effectiveness of the current recommendation system can be measurable if I can incorporate their methodologies into the design in the future.

References

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. *Www.microsoft.com*. https://www.microsoft.com/en-us/research/publication/empirical-analysis-of-predictive-algorithms-for-collaborative-filtering/?from=http%3A%2F%2Fresearch.microsoft.com%2Fpubs%2F69656%2Ftr-98-12.pdf

Etezadi, H. (2022, February 13). *Movie Recommender system*. Kaggle.com. https://www.kaggle.com/hamedetezadi/movie-recommender-system

GroupLens. (2018). *MovieLens 20M Dataset*. Kaggle.com. https://www.kaggle.com/grouplens/movielens-20m-dataset?select=movie.csv

Jeong, Y. (2021, April 22). *Item-Based Collaborative Filtering in Python*. Medium. https://towardsdatascience.com/item-based-collaborative-filtering-in-python-91f747200fab