Abstract

Recommender system is a technology that help users to discover products and contents. A recommender system makes personalized prediction of “rating” or “preference” for a user by extracting knowledge from existed user behavior, which is extremely useful in modern online application. While PageRank is an algorithm which output a probability distribution to measure the importance of web pages. In this report, we try to explain how PageRank can be used in both content-based and item-based recommender system, with simple implementation on the MovieLens dataset.

1.Introduction

In the age of information explosion, massive amount of information from different individuals is collected through multiple ways. Personalized prediction and analysis based on these data are subtly influence our life. Songs on music player, items on Amazon, videos on YouTube and even posts on Facebook can be the output of recommender system.

For a recommender system, the main job is to predict the preference for different items and suggest the top-ranked items to the users. Although PageRank is originally used for hyperlinked set of websites, the ranking logic from the algorithm still can be applied to recommender systems. In the report, we will explain how PageRank can be used in user-based recommendation (in section 3) and item-based recommendation (in section 4).

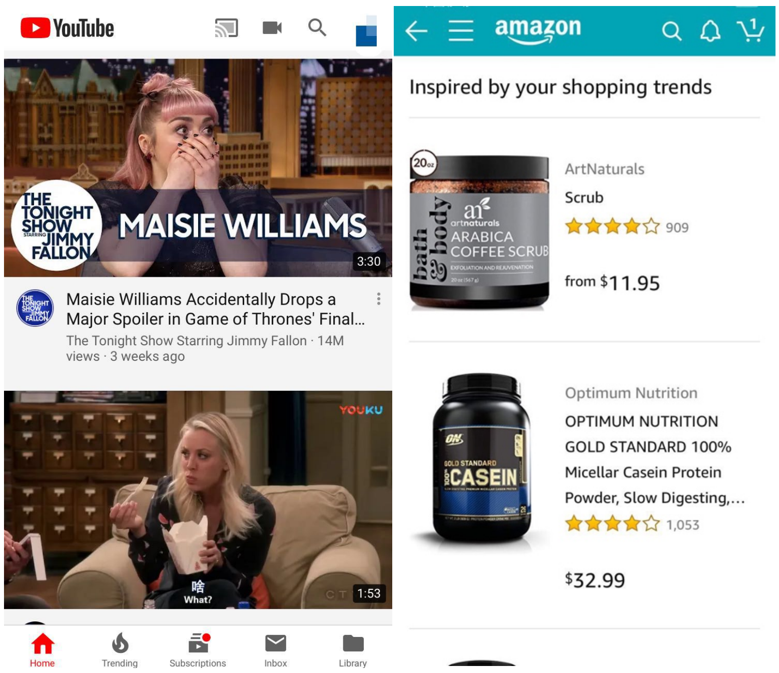

YouTube
THE TONIGHT SHOW STARRING JIMMY FALLON
MAISIE WILLIAMS
3:30
Maisie Williams Accidentally Drops a Major Spoiler in Game of Thrones' Final...
The Tonight Show Starring Jimmy Fallon · 14M views · 3 weeks ago
YOUKU
What?
1:53
Home
Trending
Subscriptions
Inbox
Library
amazon
Inspired by your shopping trends
ArtNaturals
Scrub
909
from $11.95
Optimum Nutrition
OPTIMUM NUTRITION GOLD STANDARD 100% Micellar Casein Protein Powder, Slow Digesting,...
1,053
$32.99

Fig. 1. Recommendations from YouTube and Amazon

2.The problem

To define formally, a recommender system contains a set of users μ_i, i=1,…,U_n as well as a set of products p_j, $j = 1, \ldots, P_n$. The main goal of the system is to compute the preference degree of user μ_i for product p_j based on existing data, and represent the result in numeric form, denoted by $\hat{r}_{i,j}$. Then, for each user μ_i , the system will compute its expected preference for all the products $\hat{r}_{i,j}$, $j = 1, \ldots, P_n$. After sorting, top-ranked products will be suggested to the user. While in PageRank, we are given a web graph G, and for a webpage u, its rank can be calculated based on the graph [1].

$$R(u) = c_1 \sum_{v \in B_u} \frac{R(v)}{N_v} + C_2 E(u)$$

where B_u denotes the set of u's backlinks, N_v denotes the number of forward links of page v and $E(u)$ is a distribution of rank according to which user may jump randomly. Consider ranks for all the nodes in matrix form, which leads to the solution of an eigenvalue problem [2].

$$(I - \alpha M)r = (1 - \alpha)v$$

where I is identity matrix, M is column stochastic matrix , α is constant number, v is a stochastic column vector and r is the ranking vector. For a recommender system, the output can be also considered as a ranking vector. Thus, the key step to apply PageRank is formulating a stochastic matrix based on user ratings.

2.1 MovieLens Data

To help understand how the algorithm is used, we used MovieLens dataset from MovieLens project (http://MovieLens.umn.edu) to explain the definition and give some examples, as well as simple implementation. The dataset consists over 20 million ratings from 138,000 users to 27,000 movies. Each rating can be expressed as a tuple $t_{i,j} = \{u_i, m_j, r_{i,j}\}$, where u_i is user id, m_j is movie id and $r_{i,j}$ is the rating which is an integer between 1(bad) and 5(good).

3.User-based Recommendation

For user-based recommendation, the system is trying to make a personalized recommendation based on both the movie profile and the user profile of whom you are suggesting to. Even for noob who have no idea about the system. The logic can be clear: we can extract user's preferred product from his profile, based on his reference, we recommend similar products to the user. But how can we determine the similarity? Even from a superficial point of view, different pair of movies will have different correlation degree. Consider a big fan of superhero, definitely he will have great expectation for recent movie *Avengers: Endgame*. To better determine the correlation between movies, we consider number of user who rated both two movies as a measurement. Further, we can liken movies to the websites in PageRank and define the stochastic matrix.

3.1 Movie Correlation Matrix

Formally, we define the user set U, movie set M, and the rating set L. Thus the user set who have watched both movie i and movie j can be expressed:

$$\beta_{i,j} = \begin{cases} \{u_k: (u_{k,i} \in L) \wedge (u_{k,j} \in L)\} & if\ i \neq j \\ \emptyset & if\ i = j \end{cases}$$

Then we can consider the correlation in a graph, each movie is a node, there will be an edge between movie i and movie j someone rated both the movies, that is $\beta_{i,j}$ is nonempty. The edge value will be $|\beta_{i,j}|$, where $|\cdot|$ denotes the number of elements in the set. A toy example is shown below.

User	Rated Movies		
1	Iron Man	The Matrix	Avengers
2	Titanic	God Father	Forrest Gump
3	God Father	Forrest Gump	
4	Iron Man	Avengers	
5	God Father	Titanic	The Matrix

Table1. Example data for section 3.1

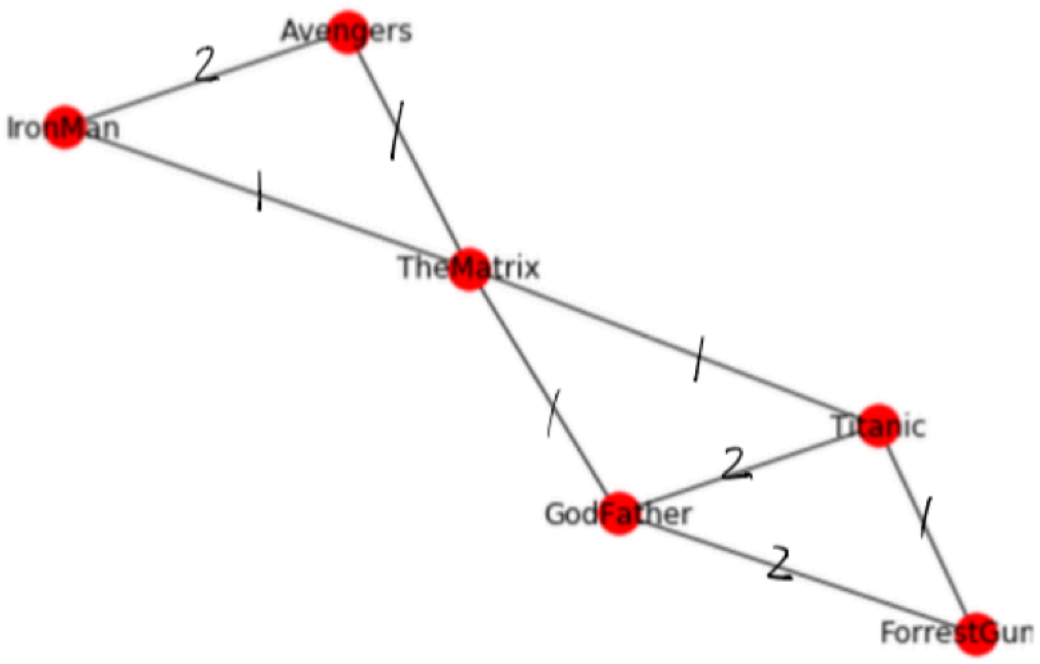
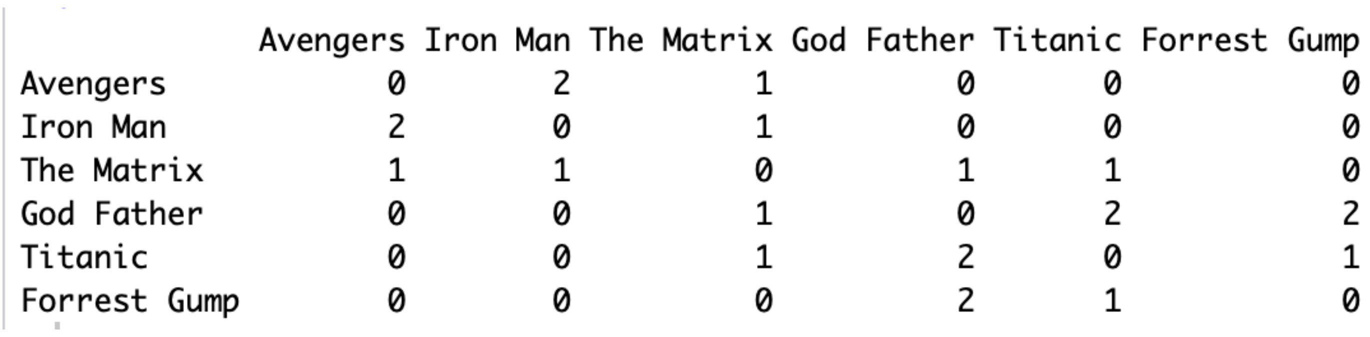


Fig.2 Corresponding graph for Table1

Now, we try to construct a column stochastic matrix based on the graph. Recall that within the PageRank concept, the rank of a webpage is given by rank of webpages which link to it. Similarly, we can predict the user's rating to a certain movie based on user's ratings of other movies that are correlated to the movie. With this idea, we compute a $|M|\text{x}|M|$ matrix $\tilde{C}$ containing the number of users who rated both movies, that is:

$$\tilde{C}_{i,j} = |\beta_{i,j}|$$



```
             Avengers Iron Man The Matrix God Father Titanic Forrest Gump
Avengers            0        2          1          0       0            0
Iron Man            2        0          1          0       0            0
The Matrix          1        1          0          1       1            0
God Father          0        0          1          0       2            2
Titanic             0        0          1          2       0            1
Forrest Gump        0        0          0          2       1            0
```

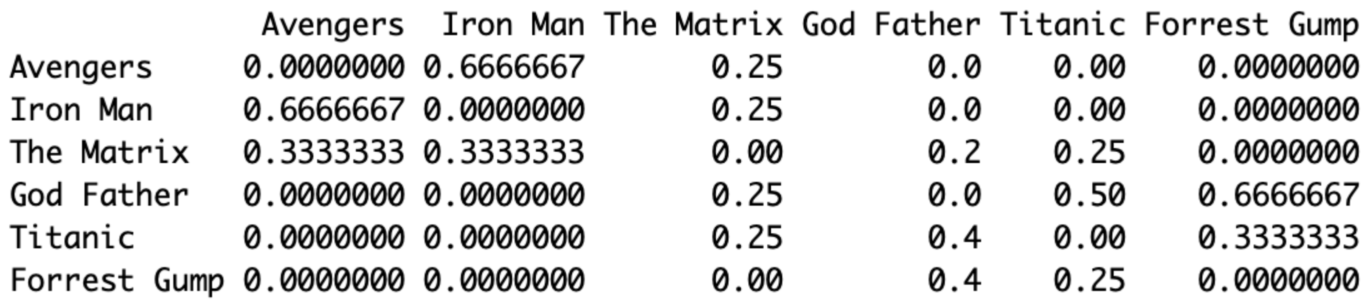
Fig3. matrix $\tilde{C}$ for Table1

With this matrix $\tilde{C}$, which is actually symmetric, we construct a new matrix by normalizing the matrix.

$$C_{i,j} = \frac{\tilde{C}_{i,j}}{\sum_i \tilde{C}_{i,j}}$$

Now the new matrix C is a column stochastic, i.e. it satisfies the following property:

$$\sum_{i=1}^{|M|} C_{i,j} = 1 \;\forall\, j \in [1, |M|]$$



```
              Avengers  Iron Man The Matrix God Father Titanic Forrest Gump
Avengers     0.0000000 0.6666667       0.25        0.0    0.00    0.0000000
Iron Man     0.6666667 0.0000000       0.25        0.0    0.00    0.0000000
The Matrix   0.3333333 0.3333333       0.00        0.2    0.25    0.0000000
God Father   0.0000000 0.0000000       0.25        0.0    0.50    0.6666667
Titanic      0.0000000 0.0000000       0.25        0.4    0.00    0.3333333
Forrest Gump 0.0000000 0.0000000       0.00        0.4    0.25    0.0000000
```

Fig4. Normalized matrix $\tilde{C}$ for Table1

3.2 ItemRank Algorithm

Recall that in section 2, the PageRank problem can be formulated as:

$$(I - \alpha M)r = (1 - \alpha)v$$

Now, we have explained how column stochastic matrix $\tilde{C}$ can be calculated in last part(section 3.1). The other parameter v which has non-negative elements summing up to 1 is still unavailable. As mentioned before, we are trying to recommend products to user based on his or her profile. So it is nature to construct the vector based on user ratings. The unnormalized component of $\tilde{R}_{u_i}$ can be represented by:

$$\tilde{R}_{u_i}^{j} = \begin{cases} r_{i,j} & if\ u_{i,j} \in L \\ 0 & if\ u_{i,j} \notin L \end{cases}$$

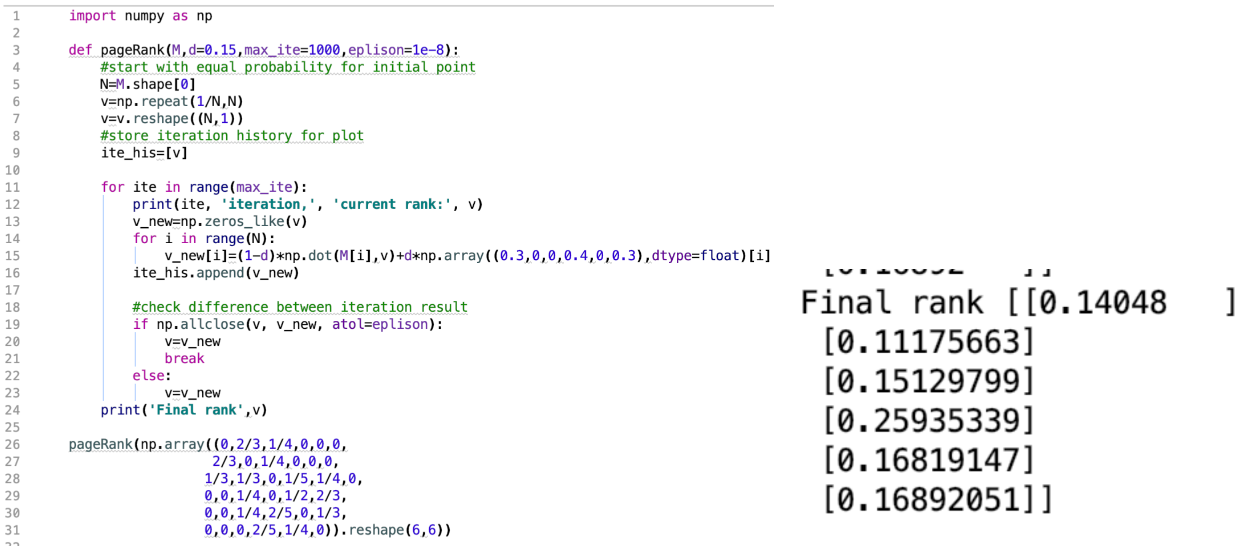
Similar to the matrix, we do normalization to the vector, $R_{u_i} = \frac{\tilde{R}_{u_i}}{|\tilde{R}_{u_i}|}$. The final recommender equation is:

$$IR_{u_i} = \alpha \cdot C \cdot IR_{u_i} + (1-\alpha) \cdot R_{u_i}$$

where IR_{u_i} is the ranking vector of products for user u_i, α is a constant(commonly 0.85). Stick to the data from table1, consider the rating example for user 6 as follows (based on the 1 to 5 scale as stated in section 2.1):

User	Rated Movies
6	God Father:4 Forest Gump:3 Avengers:3

The resulting vector is $[0.3,0,0,0.4,0,0.3]^{\mathrm{T}}$, using programming to get numerical result, thus the recommended movie for user 6 are Titanic, followed by The Matrix and Iron Man.



```python
import numpy as np

def pageRank(M,d=0.15,max_ite=1000,eplison=1e-8):
    #start with equal probability for initial point
    N=M.shape[0]
    v=np.repeat(1/N,N)
    v=v.reshape((N,1))
    #store iteration history for plot
    ite_his=[v]

    for ite in range(max_ite):
        print(ite, 'iteration,', 'current rank:', v)
        v_new=np.zeros_like(v)
        for i in range(N):
            v_new[i]=(1-d)*np.dot(M[i],v)+d*np.array((0.3,0,0,0.4,0,0.3),dtype=float)[i]
        ite_his.append(v_new)

        #check difference between iteration result
        if np.allclose(v, v_new, atol=eplison):
            v=v_new
            break
        else:
            v=v_new
    print('Final rank',v)

pageRank(np.array((0,2/3,1/4,0,0,0,
                    2/3,0,1/4,0,0,0,
                   1/3,1/3,0,1/5,1/4,0,
                   0,0,1/4,0,1/2,2/3,
                   0,0,1/4,2/5,0,1/3,
                   0,0,0,2/5,1/4,0)).reshape(6,6))
```

```
Final rank [[0.14048    ]
 [0.11175663]
 [0.15129799]
 [0.25935339]
 [0.16819147]
 [0.16892051]]
```

Fig5. Code snippets and Numerical results

3.3 Implementation on MovieLens Data

In this section, we would like to implement the algorithm on the MovieLens Data, which has already been introduced in section 2.1. However, as stated, the MovieLens Data contains too much tuples for more than 20 thousand movies, even the correlation matrix based on two users can be extremely complicated (Fig.6). Thus, we'd like to show some results in this section, details about codes are available at appendix part, as well as https://github.com/hogan98/ItemRank-UserRank.

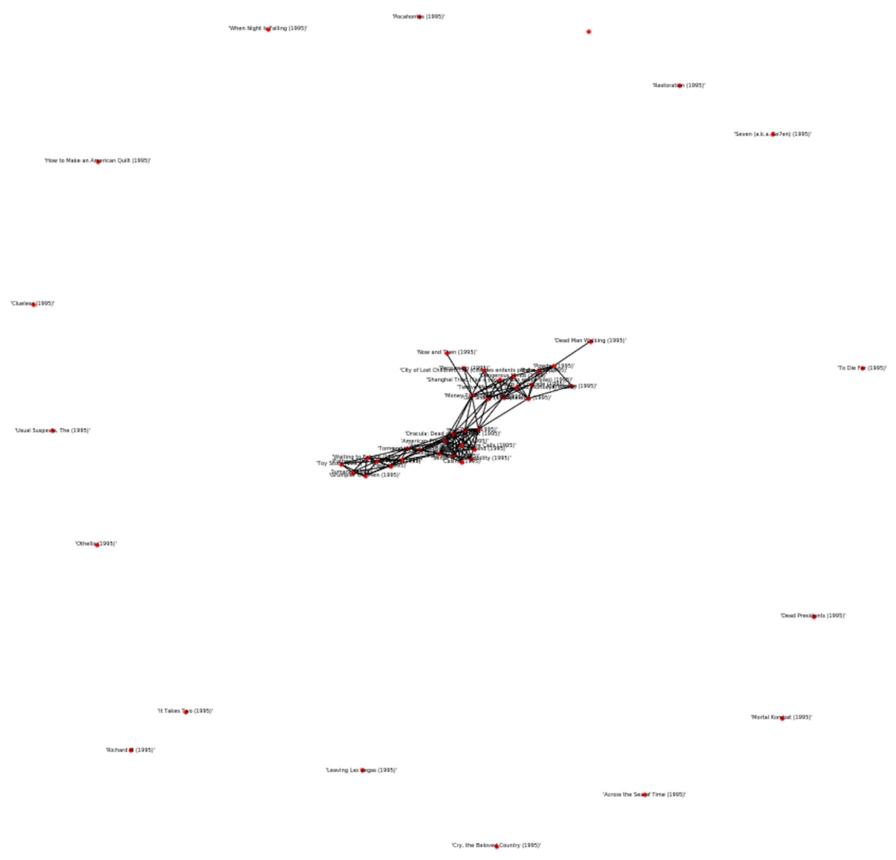
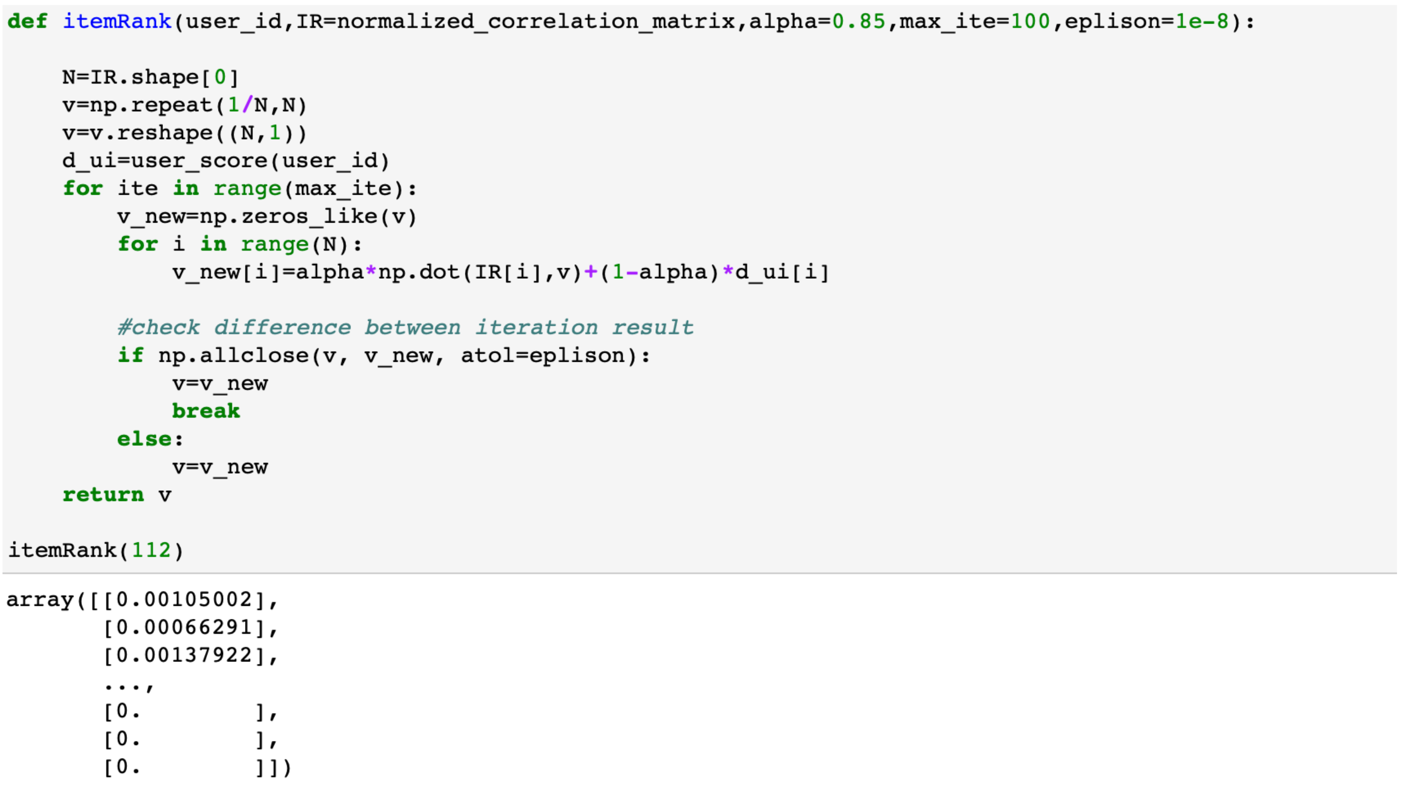

'When Night Is Falling (1995)'
'Pocahontas (1995)'
'Restoration (1995)'
'Seven (a.k.a. Se7en) (1995)'
'How to Make an American Quilt (1995)'
'Clueless (1995)'
'Dead Man Walking (1995)'
'Now and Then (1995)'
'City of Lost Children, The (Cité des enfants perdus, La) (1995)'
'Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)'
'Money Train (1995)'
'To Die For (1995)'
'Usual Suspects, The (1995)'
'Dracula: Dead and Loving It (1995)'
'American President, The (1995)'
'Waiting to Exhale (1995)'
'Toy Story (1995)'
'Sense and Sensibility (1995)'
'Casino (1995)'
'Grumpier Old Men (1995)'
'Othello (1995)'
'Dead Presidents (1995)'
'It Takes Two (1995)'
'Mortal Kombat (1995)'
'Richard III (1995)'
'Leaving Las Vegas (1995)'
'Across the Sea of Time (1995)'
'Cry, the Beloved Country (1995)'

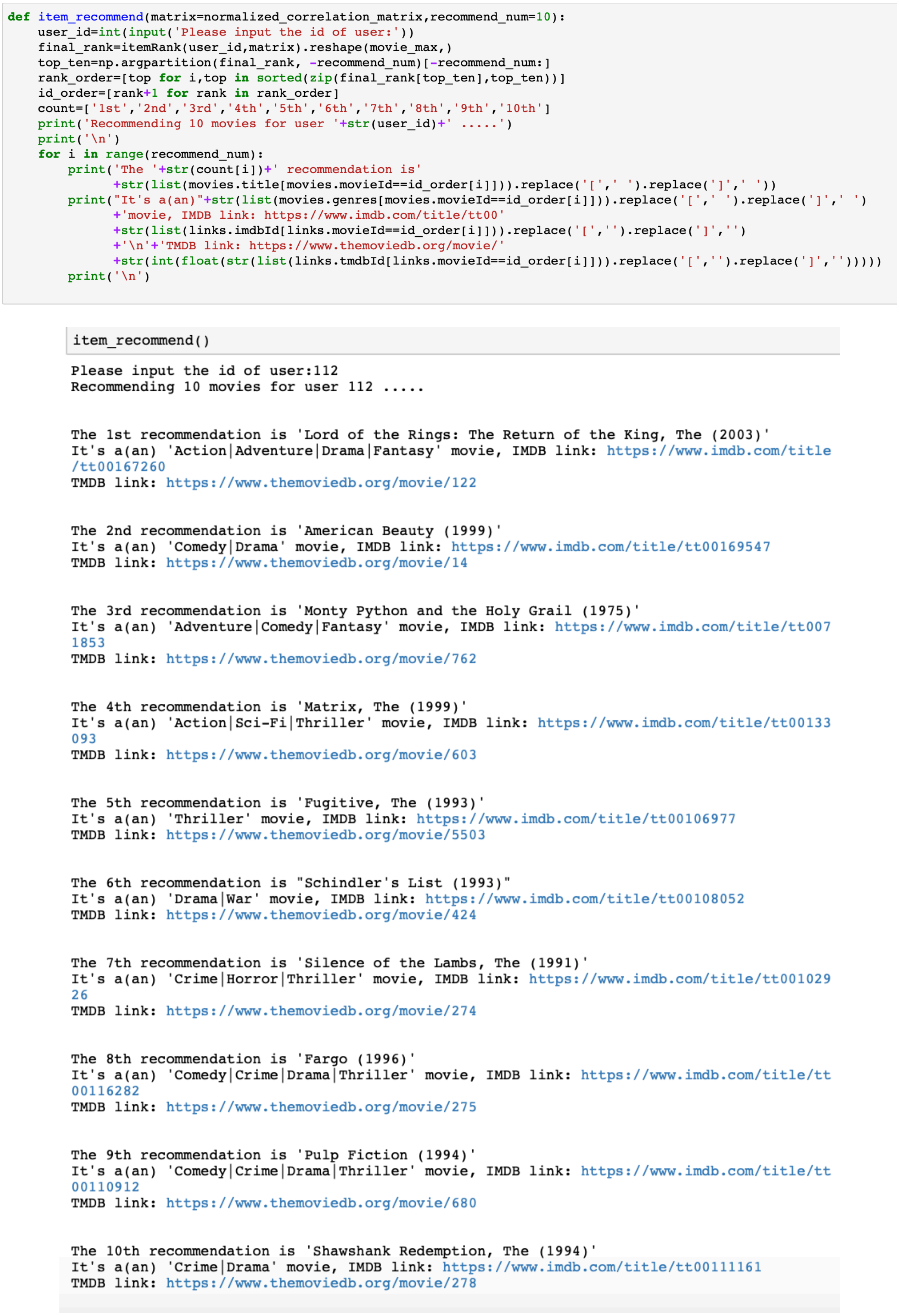
Fig6. Correlation Graph based on two users.



```
def itemRank(user_id,IR=normalized_correlation_matrix,alpha=0.85,max_ite=100,eplison=1e-8):

    N=IR.shape[0]
    v=np.repeat(1/N,N)
    v=v.reshape((N,1))
    d_ui=user_score(user_id)
    for ite in range(max_ite):
        v_new=np.zeros_like(v)
        for i in range(N):
            v_new[i]=alpha*np.dot(IR[i],v)+(1-alpha)*d_ui[i]

        #check difference between iteration result
        if np.allclose(v, v_new, atol=eplison):
            v=v_new
            break
        else:
            v=v_new
    return v

itemRank(112)
```

```
array([[0.00105002],
       [0.00066291],
       [0.00137922],
       ...,
       [0.        ],
       [0.        ],
       [0.        ]])
```

Fig7. Screenshot of computation of ranking



```python
def item_recommend(matrix=normalized_correlation_matrix,recommend_num=10):
    user_id=int(input('Please input the id of user:'))
    final_rank=itemRank(user_id,matrix).reshape(movie_max,)
    top_ten=np.argpartition(final_rank, -recommend_num)[-recommend_num:]
    rank_order=[top for i,top in sorted(zip(final_rank[top_ten],top_ten))]
    id_order=[rank+1 for rank in rank_order]
    count=['1st','2nd','3rd','4th','5th','6th','7th','8th','9th','10th']
    print('Recommending 10 movies for user '+str(user_id)+' .....')
    print('\n')
    for i in range(recommend_num):
        print('The '+str(count[i])+' recommendation is'
              +str(list(movies.title[movies.movieId==id_order[i]])).replace('[',' ').replace(']',' '))
        print("It's a(an)"+str(list(movies.genres[movies.movieId==id_order[i]])).replace('[',' ').replace(']',' ')
              +'movie, IMDB link: https://www.imdb.com/title/tt00'
              +str(list(links.imdbId[links.movieId==id_order[i]])).replace('[','').replace(']','')
              +'\n'+'TMDB link: https://www.themoviedb.org/movie/'
              +str(int(float(str(list(links.tmdbId[links.movieId==id_order[i]])).replace('[','').replace(']','')))))
        print('\n')
```

```python
item_recommend()
```

```
Please input the id of user:112
Recommending 10 movies for user 112 .....


The 1st recommendation is 'Lord of the Rings: The Return of the King, The (2003)'
It's a(an) 'Action|Adventure|Drama|Fantasy' movie, IMDB link: https://www.imdb.com/title
/tt00167260
TMDB link: https://www.themoviedb.org/movie/122


The 2nd recommendation is 'American Beauty (1999)'
It's a(an) 'Comedy|Drama' movie, IMDB link: https://www.imdb.com/title/tt00169547
TMDB link: https://www.themoviedb.org/movie/14


The 3rd recommendation is 'Monty Python and the Holy Grail (1975)'
It's a(an) 'Adventure|Comedy|Fantasy' movie, IMDB link: https://www.imdb.com/title/tt007
1853
TMDB link: https://www.themoviedb.org/movie/762


The 4th recommendation is 'Matrix, The (1999)'
It's a(an) 'Action|Sci-Fi|Thriller' movie, IMDB link: https://www.imdb.com/title/tt00133
093
TMDB link: https://www.themoviedb.org/movie/603


The 5th recommendation is 'Fugitive, The (1993)'
It's a(an) 'Thriller' movie, IMDB link: https://www.imdb.com/title/tt00106977
TMDB link: https://www.themoviedb.org/movie/5503


The 6th recommendation is "Schindler's List (1993)"
It's a(an) 'Drama|War' movie, IMDB link: https://www.imdb.com/title/tt00108052
TMDB link: https://www.themoviedb.org/movie/424


The 7th recommendation is 'Silence of the Lambs, The (1991)'
It's a(an) 'Crime|Horror|Thriller' movie, IMDB link: https://www.imdb.com/title/tt001029
26
TMDB link: https://www.themoviedb.org/movie/274


The 8th recommendation is 'Fargo (1996)'
It's a(an) 'Comedy|Crime|Drama|Thriller' movie, IMDB link: https://www.imdb.com/title/tt
00116282
TMDB link: https://www.themoviedb.org/movie/275


The 9th recommendation is 'Pulp Fiction (1994)'
It's a(an) 'Comedy|Crime|Drama|Thriller' movie, IMDB link: https://www.imdb.com/title/tt
00110912
TMDB link: https://www.themoviedb.org/movie/680


The 10th recommendation is 'Shawshank Redemption, The (1994)'
It's a(an) 'Crime|Drama' movie, IMDB link: https://www.imdb.com/title/tt00111161
TMDB link: https://www.themoviedb.org/movie/278
```

Fig8. Screenshots of recommendation results based on ranking

4.Item-based Recommendation

In section 3, we explained how to build a PageRank-based recommendation system based on rating data and user profile. However, this kind of approach do have limitations. Consider when the user profile data is inefficient or even unavailable, maybe the user is new to the system or inactive due to the annoying CAPTCHA. Let's push the question to an extreme way, how should we recommend items to user when he only rated one item, which leads us to the item similarity question. In past researches, there are several approaches to measure the item similarity, a famous one is Adjusted Cosine [3]:

$$Sim_{i,j} = \frac{\sum_{k:(u_{k,i}\in L)\wedge(u_{k,j}\in L)}(r_{k,i}-\overline{r_k})\cdot(r_{k,j}-\overline{r_k})}{\sqrt{\sum_{k:(u_{k,i}\in L)\wedge(u_{k,j}\in L)}(r_{k,i}-\overline{r_k})^2}\cdot\sqrt{\sum_{k:(u_{k,i}\in L)\wedge(u_{k,j}\in L)}(r_{k,i}-\overline{r_k})^2}}$$

where $\overline{r_k}$ is the average rating of user k. But it comes the fact, in the above formula, everyone is considered with same weight. However, the fact is when someone's option is similarly to many people, his ratings should be highlier valued, for which PageRank plays a role. To imply PageRank in this problem, we considered users as webpages and give them ranks and introduce the user ranks into similarity calculation.

4.1 User Correlation Graph

Stick to the notations in section 3.1. Then the movie set that are both rated by user i and j can be expressed:

$$\gamma_{i,j} = \begin{cases} \{m_k: (u_{i,k}\in L)\wedge(u_{j,k}\in L)\} & if\ i\neq j \\ \emptyset & if\ i=j \end{cases}$$

Then we can consider the correlation in a graph, each user is a node, there will be an edge between user i and user j if they both rated at least one movie, that is $\gamma_{i,j}$ is nonempty. The edge value will be $|\gamma_{i,j}|$, where $|\cdot|$ denotes the number of elements in the set. To explains how it works, let's stick to the data in table1 with extra ratings.

User	Rated Movies		
1	Iron Man:4	The Matrix:4	Avengers:5
2	Titanic:4	God Father:5	Forrest Gump:5
3	God Father:4	Forrest Gump:3	
4	Iron Man:4	Avengers:4	
5	God Father:4	Titanic:5	The Matrix:3

Table2. Example Data with rating

From the table, user 1 and user 2 do not rate same movie, while user1 and user 4 both rated Iron Man and Avengers …., which lead us to the graph.

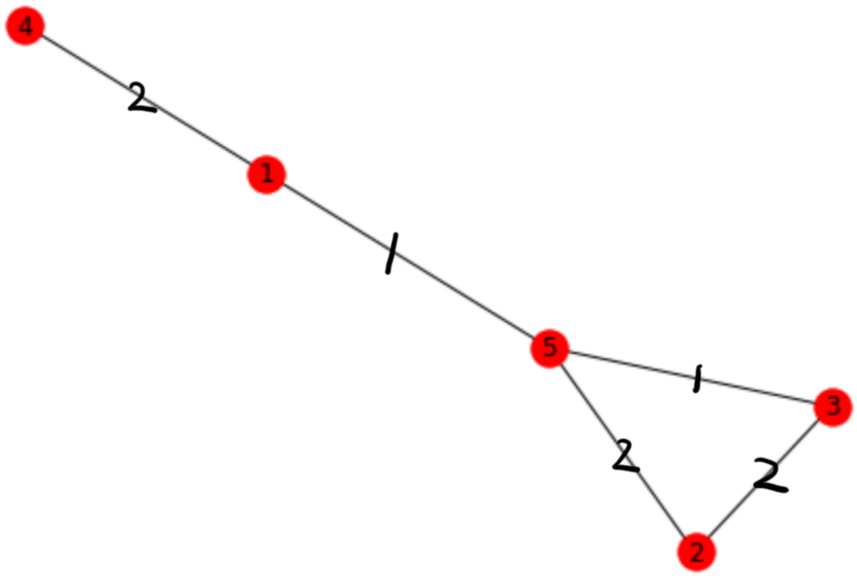


Fig.9 User correlation graph for table 2

Similar to what we have down in section 3.1, we compute a $|U|\text{x}|U|$ matrix $\tilde{E}$ containing the number of movies who rated by both users, that is:

$$\tilde{E}_{i,j} = |\bar{\gamma}_{i,j}|$$

	user1	user2	user3	user4	user5
user1	0	0	0	2	1
user2	0	0	2	0	2
user3	0	2	0	0	1
user4	2	0	0	0	0
user5	1	2	1	0	0

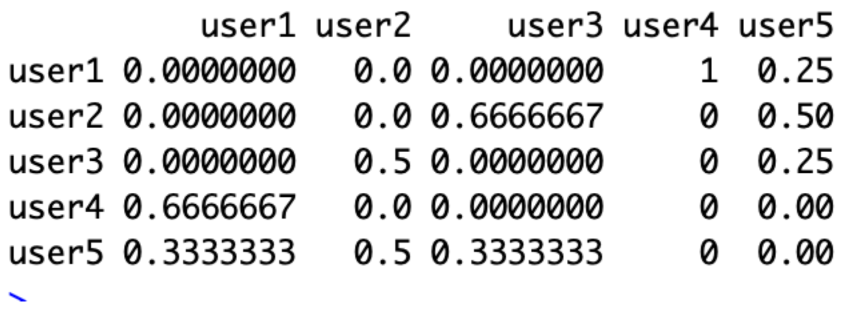
Fig10. matrix $\tilde{E}$ for Table2

With this matrix $\tilde{E}$, which is actually symmetric, we construct a new matrix by normalizing the matrix.

$$E_{i,j} = \frac{\tilde{E}_{i,j}}{\sum_i \tilde{E}_{i,j}}$$

Now the new matrix E is a column stochastic, i.e. it satisfies the following property:

$$\sum_{i=1}^{|M|} E_{i,j} = 1 \; \forall \, j \in [1, |U|]$$



```
      user1 user2     user3 user4 user5
user1 0.0000000   0.0 0.0000000     1  0.25
user2 0.0000000   0.0 0.6666667     0  0.50
user3 0.0000000   0.5 0.0000000     0  0.25
user4 0.6666667   0.0 0.0000000     0  0.00
user5 0.3333333   0.5 0.3333333     0  0.00
```

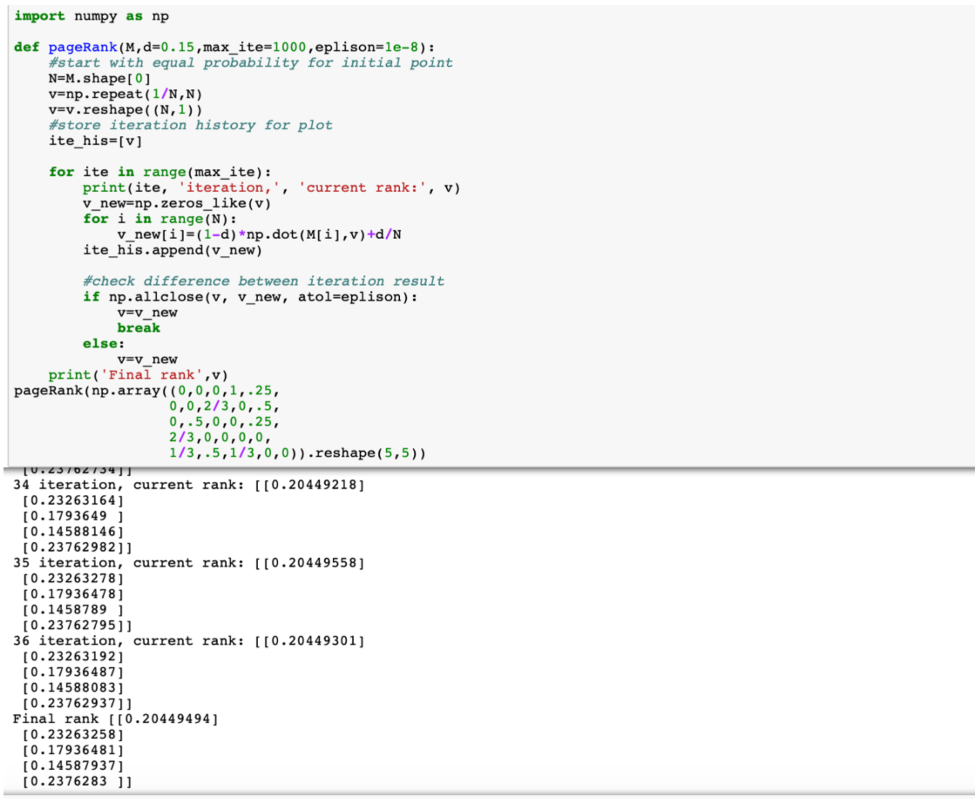
Fig11. Normalized matrix $\tilde{E}$ for Table1

4.2 Item-based recommendation algorithm.

With the result matrix E from section 4.1, simply apply PageRank as stated in the beginning of section 4, which leads us to the user rank.

$$UR = \alpha \cdot E \cdot UR + (1-\alpha) \cdot \frac{1^T}{|U|}$$

where 1^T is all one column vector. With data from table 2, the final user rank is $[0.20, 0.23, 0.17, 0.14, 0.24]^T$, which indicates that ratings from user 5 should be highly evaluated.



```python
import numpy as np

def pageRank(M,d=0.15,max_ite=1000,eplison=1e-8):
    #start with equal probability for initial point
    N=M.shape[0]
    v=np.repeat(1/N,N)
    v=v.reshape((N,1))
    #store iteration history for plot
    ite_his=[v]

    for ite in range(max_ite):
        print(ite, 'iteration,', 'current rank:', v)
        v_new=np.zeros_like(v)
        for i in range(N):
            v_new[i]=(1-d)*np.dot(M[i],v)+d/N
        ite_his.append(v_new)

        #check difference between iteration result
        if np.allclose(v, v_new, atol=eplison):
            v=v_new
            break
        else:
            v=v_new
    print('Final rank',v)
pageRank(np.array((0,0,0,1,.25,
                   0,0,2/3,0,.5,
                   0,.5,0,0,.25,
                   2/3,0,0,0,0,
                   1/3,.5,1/3,0,0)).reshape(5,5))
```

```
 [0.23762734]]
34 iteration, current rank: [[0.20449218]
 [0.23263164]
 [0.1793649 ]
 [0.14588146]
 [0.23762982]]
35 iteration, current rank: [[0.20449558]
 [0.23263278]
 [0.17936478]
 [0.1458789 ]
 [0.23762795]]
36 iteration, current rank: [[0.20449301]
 [0.23263192]
 [0.17936487]
 [0.14588083]
 [0.23762937]]
Final rank [[0.20449494]
 [0.23263258]
 [0.17936481]
 [0.14587937]
 [0.2376283 ]]
```

Fig12. Screenshot of computation of user rank.

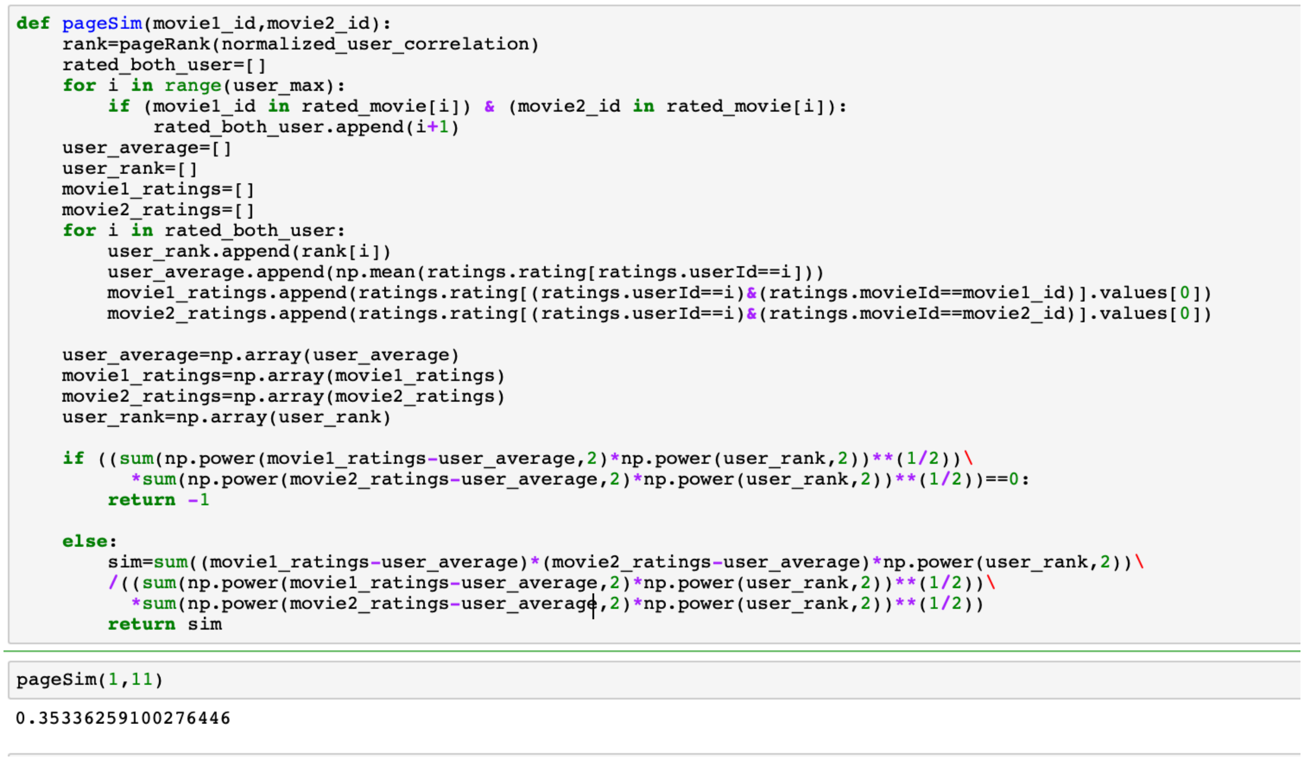
With users rank, we combine them to the computation of Adjusted Users, which leads us to the formula of the algorithm:

$$Sim_{i,j} = \frac{\sum_{k:(u_{k,i}\in L)\wedge(u_{k,j}\in L)}(r_{k,i}-\bar{r_k})\cdot(r_{k,j}-\bar{r_k})\cdot R_u^2}{\sqrt{\sum_{k:(u_{k,i}\in L)\wedge(u_{k,j}\in L)}(r_{k,i}-\bar{r_k})^2\cdot R_u^2}\cdot\sqrt{\sum_{k:(u_{k,i}\in L)\wedge(u_{k,j}\in L)}(r_{k,i}-\bar{r_k})^2\cdot R_u^2}}$$

Where R_u is the user rank computed in section4.1, other parameters are kept same as they are in the original formula.

4.3 Algorithm implementation on MovieLens Data

As the same reason stated in section 3.3, we will show some of the implementation.

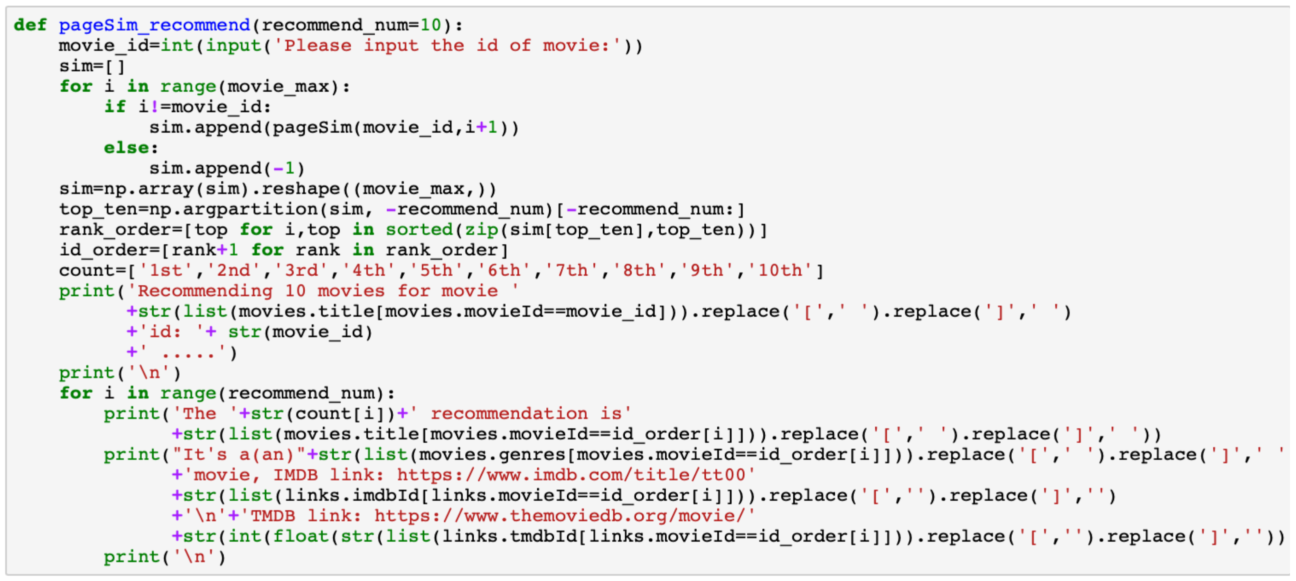


```
def pageSim(movie1_id,movie2_id):
    rank=pageRank(normalized_user_correlation)
    rated_both_user=[]
    for i in range(user_max):
        if (movie1_id in rated_movie[i]) & (movie2_id in rated_movie[i]):
            rated_both_user.append(i+1)
    user_average=[]
    user_rank=[]
    movie1_ratings=[]
    movie2_ratings=[]
    for i in rated_both_user:
        user_rank.append(rank[i])
        user_average.append(np.mean(ratings.rating[ratings.userId==i]))
        movie1_ratings.append(ratings.rating[(ratings.userId==i)&(ratings.movieId==movie1_id)].values[0])
        movie2_ratings.append(ratings.rating[(ratings.userId==i)&(ratings.movieId==movie2_id)].values[0])

    user_average=np.array(user_average)
    movie1_ratings=np.array(movie1_ratings)
    movie2_ratings=np.array(movie2_ratings)
    user_rank=np.array(user_rank)

    if ((sum(np.power(movie1_ratings-user_average,2)*np.power(user_rank,2))**(1/2))\
          *sum(np.power(movie2_ratings-user_average,2)*np.power(user_rank,2))**(1/2))==0:
        return -1

    else:
        sim=sum((movie1_ratings-user_average)*(movie2_ratings-user_average)*np.power(user_rank,2))\
        /((sum(np.power(movie1_ratings-user_average,2)*np.power(user_rank,2))**(1/2))\
          *sum(np.power(movie2_ratings-user_average,2)*np.power(user_rank,2))**(1/2))
        return sim
```

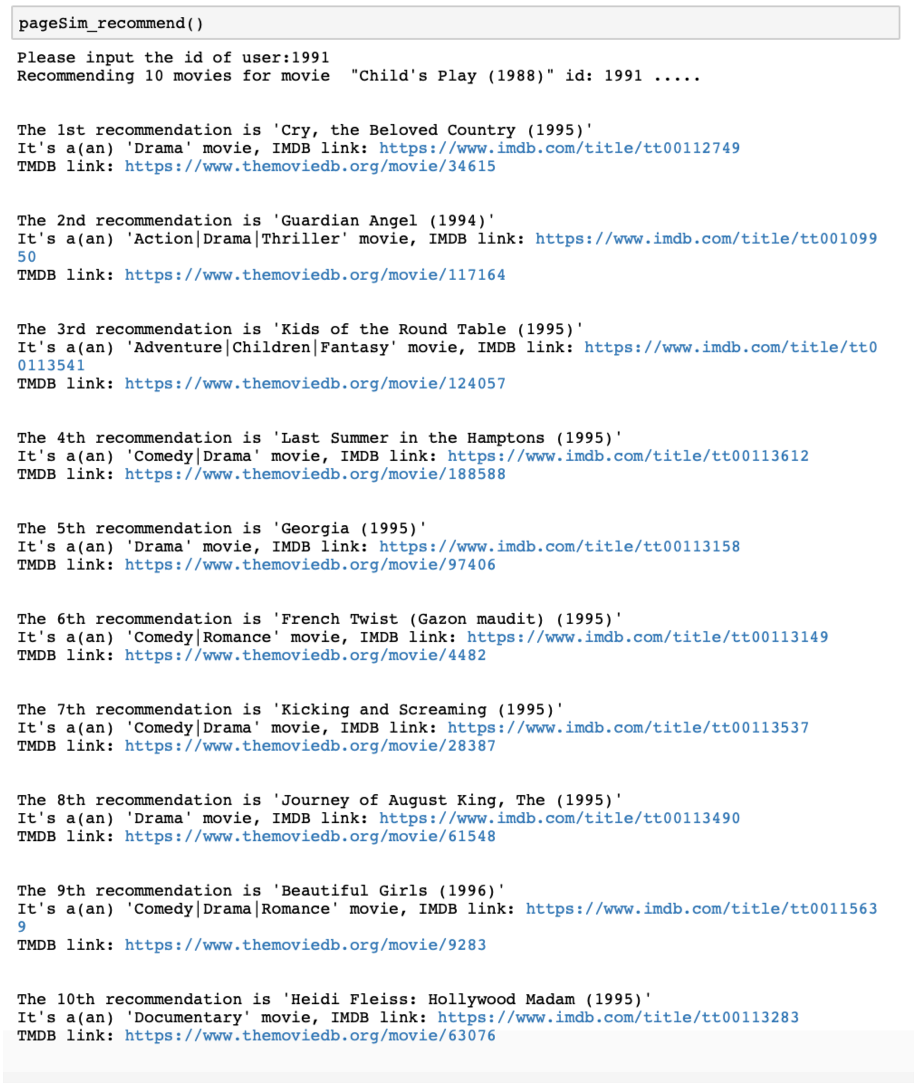
```
pageSim(1,11)
```

```
0.35336259100276446
```

Fig13. Screenshot of computation of item similarity.



```
def pageSim_recommend(recommend_num=10):
    movie_id=int(input('Please input the id of movie:'))
    sim=[]
    for i in range(movie_max):
        if i!=movie_id:
            sim.append(pageSim(movie_id,i+1))
        else:
            sim.append(-1)
    sim=np.array(sim).reshape((movie_max,))
    top_ten=np.argpartition(sim, -recommend_num)[-recommend_num:]
    rank_order=[top for i,top in sorted(zip(sim[top_ten],top_ten))]
    id_order=[rank+1 for rank in rank_order]
    count=['1st','2nd','3rd','4th','5th','6th','7th','8th','9th','10th']
    print('Recommending 10 movies for movie '
          +str(list(movies.title[movies.movieId==movie_id])).replace('[',' ').replace(']',' ')
          +'id: '+ str(movie_id)
          +' .....')
    print('\n')
    for i in range(recommend_num):
        print('The '+str(count[i])+' recommendation is'
              +str(list(movies.title[movies.movieId==id_order[i]])).replace('[',' ').replace(']',' '))
        print("It's a(an)"+str(list(movies.genres[movies.movieId==id_order[i]])).replace('[',' ').replace(']',' '
              +'movie, IMDB link: https://www.imdb.com/title/tt00'
              +str(list(links.imdbId[links.movieId==id_order[i]])).replace('[','').replace(']','')
              +'\n'+'TMDB link: https://www.themoviedb.org/movie/'
              +str(int(float(str(list(links.tmdbId[links.movieId==id_order[i]])).replace('[','').replace(']',''))
        print('\n')
```



```
pageSim_recommend()
```

```
Please input the id of user:1991
Recommending 10 movies for movie  "Child's Play (1988)" id: 1991 .....


The 1st recommendation is 'Cry, the Beloved Country (1995)'
It's a(an) 'Drama' movie, IMDB link: https://www.imdb.com/title/tt00112749
TMDB link: https://www.themoviedb.org/movie/34615


The 2nd recommendation is 'Guardian Angel (1994)'
It's a(an) 'Action|Drama|Thriller' movie, IMDB link: https://www.imdb.com/title/tt001099
50
TMDB link: https://www.themoviedb.org/movie/117164


The 3rd recommendation is 'Kids of the Round Table (1995)'
It's a(an) 'Adventure|Children|Fantasy' movie, IMDB link: https://www.imdb.com/title/tt0
0113541
TMDB link: https://www.themoviedb.org/movie/124057


The 4th recommendation is 'Last Summer in the Hamptons (1995)'
It's a(an) 'Comedy|Drama' movie, IMDB link: https://www.imdb.com/title/tt00113612
TMDB link: https://www.themoviedb.org/movie/188588


The 5th recommendation is 'Georgia (1995)'
It's a(an) 'Drama' movie, IMDB link: https://www.imdb.com/title/tt00113158
TMDB link: https://www.themoviedb.org/movie/97406


The 6th recommendation is 'French Twist (Gazon maudit) (1995)'
It's a(an) 'Comedy|Romance' movie, IMDB link: https://www.imdb.com/title/tt00113149
TMDB link: https://www.themoviedb.org/movie/4482


The 7th recommendation is 'Kicking and Screaming (1995)'
It's a(an) 'Comedy|Drama' movie, IMDB link: https://www.imdb.com/title/tt00113537
TMDB link: https://www.themoviedb.org/movie/28387


The 8th recommendation is 'Journey of August King, The (1995)'
It's a(an) 'Drama' movie, IMDB link: https://www.imdb.com/title/tt00113490
TMDB link: https://www.themoviedb.org/movie/61548


The 9th recommendation is 'Beautiful Girls (1996)'
It's a(an) 'Comedy|Drama|Romance' movie, IMDB link: https://www.imdb.com/title/tt0011563
9
TMDB link: https://www.themoviedb.org/movie/9283


The 10th recommendation is 'Heidi Fleiss: Hollywood Madam (1995)'
It's a(an) 'Documentary' movie, IMDB link: https://www.imdb.com/title/tt00113283
TMDB link: https://www.themoviedb.org/movie/63076
```

Fig14. Screenshots of recommendation results based on similarity

References

1. L. Page, S. Brin, R. Motwani, and T. Winograd, The Pagerank Citation Ranking: Bringing Order to the web, technical report, Stanford University, Stanford, CA, 1998.
2. Gleich, D, PageRank Beyond the Web, SIAM Rev. 57-3 (2015), pp. 321-363
3. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web Hongkong, pp. 285–295 (2001)
4. Gori, M., Pucci, A.: Itemrank: a random-walk based scoring algorithm for recommender en- gines. In: Proc. of the 2007 IJCAI Conf., pp. 2766–2771 (2007)
5. Jiang F, Wang Z J. Pagerank-based collaborative filtering recommendation. In Proc. the 1st Int. Conf. Information Computing and Applications, October 2010, pp. 597–604.
6. F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

Appendix