Recommendation Systems

12/7/2015

David Zhuzhunashvili – Fry, lecture 3310-002

Derek Holland – Zaharatos, lecture 3310-001

Abstract:

Our topic is recommender systems, similar to the ones used by websites such as Netflix, Spotify, or Amazon. Recommender systems have a wide range of applications as shown by the number of distinctly different websites that use it. For example, Netflix uses a recommender system that recommends certain movies to specific users based on what movies similar users watched. More specifically, if user one watched Harry Potter 1 and Harry Potter 2 and rated both of them as a 4 or a 5, and if user two watched Harry Potter 1 and also rated it a 4 or a 5, the recommender system would recommend the second movie to the second user seeing as they both liked the first movie. In reality, however, Netflix looks at over 60 million users and thousands of movies to recommend specific movies to specific users based on the ratings of similar users. Spotify and Amazon use similar algorithms but for completely different products, music and many different products like books or clothes respectively.

All three of these companies face a big challenge, which is to have very well optimized code because they need to look through very many users and products. However, instead of focusing on optimization, we focused on making an accurate recommendation system. Our recommender system takes in a randomly generated matrix of users and their ratings for a

number of different movies. After computing similarities between users, the recommender system calculates the most likely rating a similar user would have given to a movie he does not currently have a rating for. This basically means that our recommender system calculates how much a user would enjoy a movie they have not yet watched based on how much they enjoyed movies that they watched in the past. The results outputted by our recommender system make logical sense and seem accurate even though it is very hard to test accuracy of a recommender system unless there is a very large amount of real user data for specific movies (without real-life data, it is very hard to tell if a recommender system recommended the movies that the users actually ended up liking).

Introduction:

As stated above, our recommender system is used to recommend movies or any other products for which the user would like to get recommendations. In reality, data is taken from users and afterwards recommendations are computed, however, in our code we allow the user to either generate random data or input their own data so they can compute recommendations for whatever they wish. In the following sections we will delve deeper into how our algorithm works, why we did not use any other algorithms, why our system can be used for a variety of products, direct results from our algorithm, and the variety of mathematical tools we used to create our finished recommender system.

Mathematical Formulation:

There are many different methods for calculating recommendations. One of the simplest methods is to cluster movies or users into smaller groups repeatedly until there are only a very few number of clusters left. Afterwards, an algorithm determines the similarity

between the smaller number of users (the clusters of users) and based on their similarity it gives the whole clusters their respective ratings. The Stanford paper on Recommendation Systems gives the example of clustering three Harry Potter movies into one cluster called Harry Potter, and the three Star Wars episodes into one cluster called Star Wars, and then using the above algorithm it calculates the most probable ratings for the users that have not yet watched the movies. However, there is a very obvious error with this algorithm, it assumes that if a user has a rating for only one Harry Potter movie, then he has an average equal to that rating for the general cluster of the three Harry Potter movies. This is very inaccurate since there are many cases where people really enjoyed the first movie of a series and hated one or both of the other ones. Another, much more accurate, method is to find two long but thin matrices (n x 2 and 2 x n or n x 3 and 3 x n or something similar) such that their product will have the dimensions of our recommendation matrix (the matrix with users as the rows and ratings as the columns with some empty inputs). After doing this, we want to go through these long but thin matrices and plug in values into one index (i, j) at a time and try to choose values that minimizes the root-mean-square error between the original matrix and the product of the two long but thin matrices. The root-mean-square error, also known as RMSE, is the sum of the squares of the differences of inputs at the same indices in two matrices (for one row of two 3 x 3 matrices, M, A, it would be $(m(1,1) - a(1,1))^2 + (m(1,2) - a(1,2))^2 + (m(1,3) - a(1,3))^2$). Although this method, known as the UV-Decomposition method, gives one of the most accurate recommendations, it also gives recommendations for some movies even though there is no direct data from another similar user pointing to this recommendation.

Based on the points discussed above, we designed an algorithm that calculates the similarities between users and based on their similarities it calculates the recommendations for unrated movies. Our algorithm uses the Euclidian dot product and Euclidian norm (dot product of two user vectors divided by the product of the two vectors' norms, the closer this value is to 1, the closer the two vectors/users are, because $\cos^{-1}1 = 0$ radians, so the two vectors are collinear. The closer the dot product divided by the product of the norms is to 0, the further apart they are, since $\cos^{-1}0 = \pi/2$ which means the two vectors/users are perpendicular to each other. There will not be any negative values after calculating the dot product divided by the product of the two vectors because the two vectors only have values from 0 to 5, which means that the dot product will always be greater than or equal to 0 and the Euclidian norm is always greater than or equal to 0) to calculate how similar any two users are (users are represented by rows while movies/items are represented by columns in an adjacency matrix that has ratings from 1 to 5 for specific movies/items). There is another method of calculating similarity using the Jaccard index which is calculated by finding the number of ratings two users have in common divided by the total number of ratings the two users have, however this is not very useful for our problem because we have ratings with specific value form 1 to 5 while the Jaccard index is only very accurate when we just have a value of 1 if there is a rating and an empty value when there is no rating (for one user/row this will look like this: [1, , 1, , 1]. Once it calculates the most similar users, it goes to those two rows in the adjacency matrix and fills in any 0 entries (0 corresponds to no rating) by finding the average difference in ratings between the two users and then adds this difference to the rating of the user with the non-zero input and then places this sum of the difference and the rating (difference between average ratings +

rating, if this ends up being less than or equal to 0, then this sum will change to a 1, since 1 is the lowest possible rating and if it ends up being greater than 5, then the sum will change to a 5, since 5 is the largest possible rating) into the indices of the 0 input. A very important part of our algorithm is that if one user has no similar users, then any zero input indices in that user's ratings do not get filled in by a recommendation value since we have no idea how much he will like a movie because we do not have any data from similar users (unlike the UV-Decomposition method). This is a key point in our algorithm because we do not want to calculate any recommendations blindly because this would result in a very large number of recommended movies to a user that he or she might not enjoy. Another key point in our algorithm is that we can use this method of calculating recommendations based on similar movies/items or similar users (the user method is described above). If two items are similar, then if we know the rating for one item from a user, then we can calculate the rating for the other item/movie for that user (if he has not yet rated the other item). For example, if all the Harry Potter movies have very high ratings, (4's or 5's) then if one user has a rating of a 4 for Harry Potter 1 then he would most likely have a 4 or a 5 rating for Harry Potter 2 or 3.

We had some simplifications in our algorithm, which are the following: our method of filling in 0 terms based on similar users' ratings (using the sum of the average difference and the rating) is not always accurate, a rating system from 1 to 5 only even though some ratings above 5 or below 1 may show that the user would enjoy the movie extremely or hate it extremely respectively, and most importantly, we assume that using the cosine of the angle

between two vectors/users is the best method of calculating the similarity between two vectors/users.

Examples and Numerical Results:

Now let's discuss how the algorithm work in more detail. At the start, the user can either create a random matrix, a random matrix with a higher probability of having 0's in its indices (40% chance 0's and (100-40)/5 = 12% chance for the numbers 1 through 5), or the user can enter a matrix of their choice. The code in Figure 1 (in appendix) randomly generates numbers so that 0 has a 40% chance of coming up (this code is important because in most real-life situations, people have a lot more unrated movies than they have rated ones, which is why we want to have a matrix with a large amount of zeroes because zeroes correspond to no rating). Once there is an adjacency matrix that we can work with, the first step of the process is to create a strictly upper triangular matrix of cosine distances (here, cosine distance refers to the similarity between two vectors/users calculated by dividing the dot product of the two vectors by the product of their norms) where the input at index (i, j) corresponds to the similarity between row/vector i and j in the adjacency matrix. Figure 2 and 3 show the adjacency matrix and the distance matrix respectively:

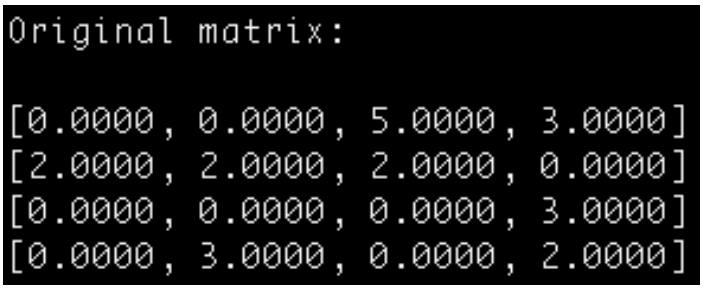

Figure 2

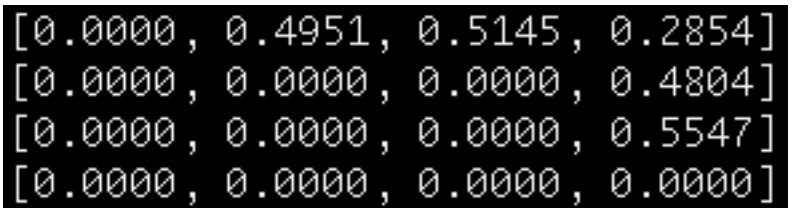

Figure 3

In the distance matrix we can see that there is a value of 0 at index (2, 3) which means that row 2 and row 3 (users 2 and 3) are not similar at all which can be confirmed by doing (2*0+2*0+2*0+0*3)/(norm(row 2)*norm(row 3)) = 0 since the numerator is 0 and the denominator is greater than 0. Furthermore, in the cosine distance matrix, index (3, 4) has the highest value (value closest to 1) which means that user 3 and user 4 (row 3 and row 4) are the most similar users. The next most similar users are users 1 and 3 (index (1, 3)) and so on. Our algorithm's next step puts in values starting from 1 going up to the number of non-zero terms in the cosine distance matrix in the indices of the corresponding cosine distance value in the cosine distance matrix. For example, there will be a 1 in the index (3, 4) and a 2 in the index (1, 3) because 0.5547 is the first largest value and 0.5145 is the second largest value and so on. Here is an example of the order matrix for the matrix in Figure 3:

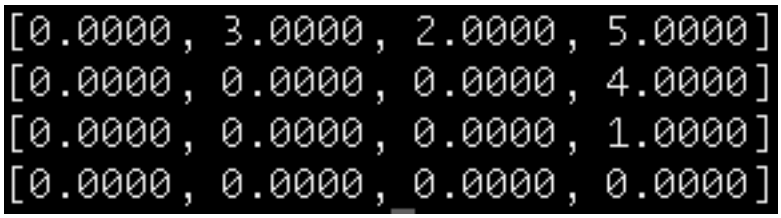

Figure 4

Once the order matrix is calculated, the next step of the algorithm is to go through the order matrix and use a rating prediction algorithm starting at the number 1 (most similar pair of users) and finishing at number 5 (the least similar pair of users). The rating prediction algorithm

was briefly described above. It takes two similar vectors (rows) and calculates the average rating for each one. It does this by summing up all the ratings and diving by the total number of ratings ([0, 0, 4]'s average would be 4). Then it finds the difference in the average ratings and finally for every 0 input in both of the vector (but it does this 1 vector at a time) it replaces it by the sum of the difference in the average ratings and the rating in of the matrices. For example, if two similar vectors are [0, 4, 2] and [4, 4, 0] then the average difference is (4 + 2)/2 – (4 + 4)/2 = -1. Then it sees that there is a 0 entry at index 1 of vector 1 so it goes to index 1 of vector 2 and adds -1 (4 + -1 = 3) to it and replaces the 0 in vector 1 by that value so the resulting vector 1 look like this: [3, 4, 2]. Now it does the same thing but for vector 2, the average difference in ratings between vector 2 and vector 1 is (4 + 4)/2 – (4 + 2)/2 = 1. Now it sees that there is a 0 at index 3 of vector 2 so it goes to index 3 of vector 1 and adds 1 to it (2 + 1 = 3) and replaces the 0 in vector 2 by that value so the new vector 2 looks like this: [4, 4, 3]. This operation also follows 3 simple rules, if after adding the average difference to the value in the vector, the value becomes less than or equal to 0, then the 0 in the vector is replaced by a 1 because 1 is the lowest possible score a user can rate a movie. On the other hand, if the sum of the value and the average difference in ratings comes out to be greater than 5, then the 0 in the vector is replaced by 5 because 5 is the maximum rating that can be given to a movie. Finally, if the value at the same indices of two vectors are 0, then nothing happens, it just leaves the 0 as they are (this is because we can not guess how much a user will like a movie if we do not have a rating of the same movie from a similar user). The algorithm keeps doing this until it goes to the final pair of similar matrices in the order matrix. To reemphasize, this operation is performed on the most similar pair of vectors first all the way to the least similar pair of

vectors. After running this operation on all the similar pairs of vectors, the algorithm goes back through the order matrix and performs the same operations, and this keeps happening until the resulting recommendation matrix stops changing (when we can not predict any new ratings). For example, if we do not have a value for an index in a certain vector called vector 1, that index might be filled up by a non-zero value from another similar vector so on the next iteration of the operation, the non-zero value in vector 1 can be used to predict a 0 value at the same index of another vector (we could not do this previously because they both would have had a 0 at that index). Figure 5 shows the result of running these operations on the adjacency matrix (with weighted connections) in Figure 2, this is the recommendation matrix resulting from finding recommendations based on the original matrix (Figure 2 matrix):

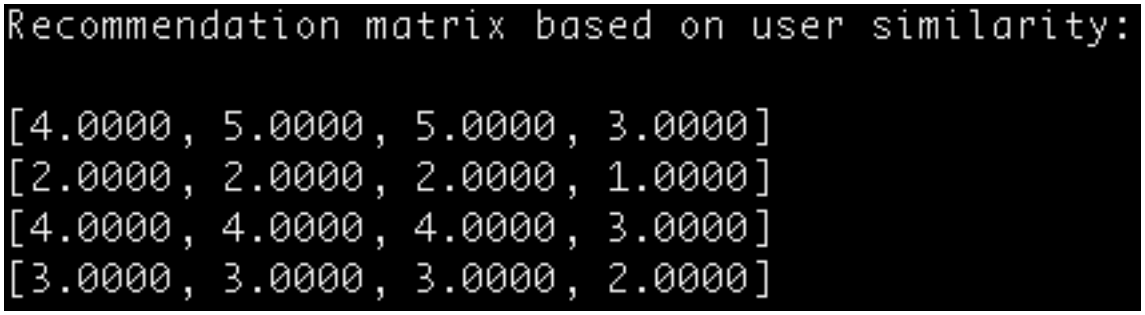

Figure 5

Our program then looks at the original matrix (Figure 2) and compares it to the recommendation matrix, wherever it sees that an unrated movie in Figure 2 (value of 0) became a 4 or a 5 (the value at index (i, j) became 4 or a 5 in the recommendation matrix and was a 0 in the original matrix), it recommends that movie to that user. Here (Figure 6) is an example for the matrix in Figure 2:

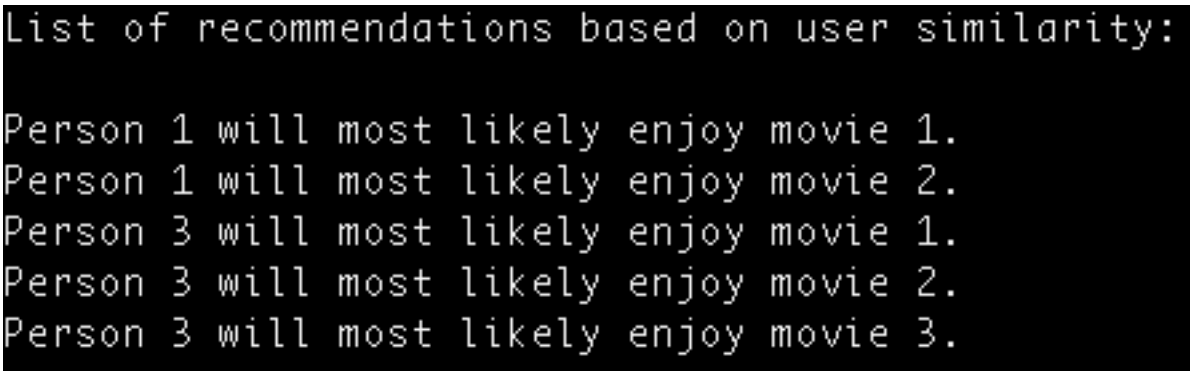

Figure 6

As mentioned above, our algorithm also calculates recommendations based on movie/item similarity so here (Figure 7) is an example of this (our algorithm simply transposes the original matrix, performs the same operations as before on the vectors and then returns the matrix transposed again, in the original format, but in this case recommendations were calculated based on movie/item similarity):

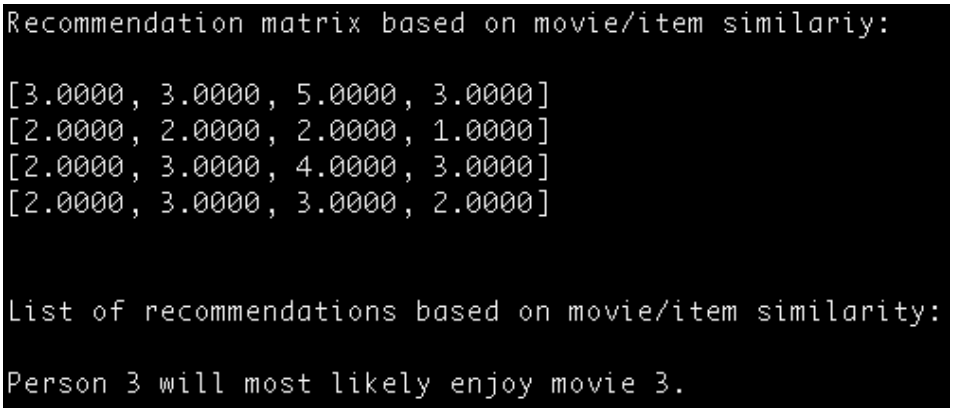

Figure 7

Figures 8 and 9 in the appendix show these computations for larger numbers of users and movies/items.

Discussion and Conclusion:

Our main goal was to create a accurate recommender system that did not make any blind assumptions when recommending movies/items. From this assignment, above everything else, we learned that recommender systems are extremely hard to implement correctly because there is no set rule which every user on a certain website follows, since every human has different behaviors. This is why there are so many different recommendation systems out there, because none of them are necessarily correct. We did not expect this fully because certain recommender systems can be pretty accurate but in the end even they end up recommending a large number of movies or items or songs that the user will never end up watching, buying, or listening to. Based on this idea that most recommender systems are inaccurate, one good method to create an amazing recommender system would be to combine the different aspects of the different recommender system into one recommender system (this is what the winners of the Netflix challenge did). These results show that our original attempt to write an accurate recommender system was mostly a failure, because simply using one recommender system is far less accurate than using a combination of many different recommender systems. To understand this subject of recommender systems, we would need to write different recommender systems and then take out the best aspects of each and try to incorporate them into one in order to make a much more accurate recommender system. This way we will truly understand what makes some recommender systems more accurate than others.

References:

Leskovec, Jure, and Anand Rajaraman. "Recommendation Systems." *Mining of Massive Datasets*. By Jeffrey D. Ullman. N.p.: n.p., n.d. 307-41. *Mining of Massive Datasets*. Cambridge University Press. Web. 6 Dec. 2015. <http://infolab.stanford.edu/~ullman/mmds/ch9.pdf>.

Appendix:

```
def random_generator():
    #If a random number x in the range 0 < x < 1 is less than or equal to 0.4 then the random number
    #that this function generates comes out ot be 0
    if(random.random() <= 0.4):
        return 0
    #Otherwise, if 0.4 < x < 1 then this function generates a number from 1 to 5 with equal probabilities
    else:
        return int((random.random()*5) + 1)
```

Figure 1

```
Original matrix:

[0.0000, 2.0000, 5.0000, 0.0000, 4.0000, 2.0000, 0.0000, 4.0000, 1.0000, 1.0000, 0.0000, 5.0000, 2.0000, 0.0000, 4.0000, 5.0000, 1.0000]
[4.0000, 0.0000, 5.0000, 1.0000, 0.0000, 0.0000, 5.0000, 0.0000, 0.0000, 0.0000, 4.0000, 5.0000, 0.0000, 4.0000, 1.0000, 2.0000, 3.0000]
[5.0000, 0.0000, 3.0000, 2.0000, 1.0000, 0.0000, 4.0000, 3.0000, 5.0000, 4.0000, 4.0000, 5.0000, 0.0000, 4.0000, 0.0000, 3.0000, 0.0000]
[0.0000, 0.0000, 0.0000, 3.0000, 0.0000, 0.0000, 4.0000, 0.0000, 0.0000, 0.0000, 5.0000, 3.0000, 0.0000, 0.0000, 0.0000, 2.0000, 4.0000]
[1.0000, 3.0000, 0.0000, 0.0000, 0.0000, 3.0000, 4.0000, 0.0000, 0.0000, 3.0000, 0.0000, 0.0000, 3.0000, 3.0000, 0.0000, 3.0000, 0.0000]
[0.0000, 4.0000, 2.0000, 1.0000, 1.0000, 4.0000, 3.0000, 1.0000, 5.0000, 2.0000, 0.0000, 0.0000, 3.0000, 0.0000, 0.0000, 5.0000, 1.0000]
[0.0000, 5.0000, 2.0000, 0.0000, 4.0000, 4.0000, 2.0000, 0.0000, 4.0000, 3.0000, 1.0000, 2.0000, 1.0000, 3.0000, 0.0000, 0.0000, 4.0000]
[0.0000, 5.0000, 0.0000, 4.0000, 0.0000, 0.0000, 2.0000, 5.0000, 2.0000, 1.0000, 0.0000, 0.0000, 2.0000, 4.0000, 0.0000, 0.0000, 4.0000]
[0.0000, 0.0000, 3.0000, 5.0000, 0.0000, 5.0000, 2.0000, 0.0000, 3.0000, 1.0000, 0.0000, 0.0000, 4.0000, 2.0000, 1.0000, 1.0000, 0.0000]
[3.0000, 0.0000, 4.0000, 2.0000, 0.0000, 5.0000, 2.0000, 1.0000, 3.0000, 1.0000, 0.0000, 2.0000, 0.0000, 1.0000, 0.0000, 0.0000, 0.0000]
[2.0000, 3.0000, 1.0000, 0.0000, 0.0000, 1.0000, 0.0000, 0.0000, 0.0000, 0.0000, 5.0000, 0.0000, 0.0000, 4.0000, 3.0000, 1.0000, 4.0000]
[2.0000, 0.0000, 1.0000, 0.0000, 0.0000, 3.0000, 0.0000, 0.0000, 0.0000, 3.0000, 0.0000, 3.0000, 2.0000, 1.0000, 0.0000, 2.0000, 5.0000]
[2.0000, 3.0000, 0.0000, 0.0000, 5.0000, 2.0000, 3.0000, 0.0000, 0.0000, 1.0000, 0.0000, 4.0000, 0.0000, 5.0000, 0.0000, 0.0000, 0.0000]
[2.0000, 3.0000, 0.0000, 0.0000, 5.0000, 0.0000, 5.0000, 1.0000, 0.0000, 0.0000, 3.0000, 5.0000, 0.0000, 4.0000, 4.0000, 3.0000, 4.0000]
[1.0000, 1.0000, 1.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 2.0000, 2.0000, 3.0000, 2.0000, 5.0000, 0.0000, 4.0000, 5.0000, 5.0000]


Recommendation matrix based on user similarity:

[2.0000, 2.0000, 5.0000, 1.0000, 4.0000, 2.0000, 5.0000, 4.0000, 1.0000, 1.0000, 3.0000, 5.0000, 2.0000, 4.0000, 4.0000, 5.0000, 1.0000]
[4.0000, 3.0000, 5.0000, 1.0000, 1.0000, 2.0000, 5.0000, 3.0000, 5.0000, 4.0000, 4.0000, 5.0000, 5.0000, 4.0000, 1.0000, 2.0000, 3.0000]
[5.0000, 3.0000, 3.0000, 2.0000, 1.0000, 5.0000, 4.0000, 3.0000, 5.0000, 4.0000, 4.0000, 5.0000, 5.0000, 4.0000, 1.0000, 3.0000, 3.0000]
[4.0000, 3.0000, 5.0000, 3.0000, 1.0000, 2.0000, 4.0000, 3.0000, 5.0000, 4.0000, 5.0000, 3.0000, 5.0000, 4.0000, 1.0000, 2.0000, 4.0000]
[1.0000, 3.0000, 2.0000, 1.0000, 1.0000, 3.0000, 4.0000, 1.0000, 5.0000, 3.0000, 1.0000, 2.0000, 3.0000, 3.0000, 4.0000, 3.0000, 1.0000]
[1.0000, 4.0000, 2.0000, 1.0000, 1.0000, 4.0000, 3.0000, 1.0000, 5.0000, 2.0000, 1.0000, 2.0000, 3.0000, 3.0000, 1.0000, 5.0000, 1.0000]
[1.0000, 5.0000, 2.0000, 1.0000, 4.0000, 4.0000, 2.0000, 1.0000, 4.0000, 3.0000, 1.0000, 2.0000, 1.0000, 3.0000, 4.0000, 5.0000, 4.0000]
[1.0000, 5.0000, 2.0000, 4.0000, 4.0000, 4.0000, 2.0000, 5.0000, 2.0000, 1.0000, 1.0000, 2.0000, 2.0000, 4.0000, 4.0000, 5.0000, 4.0000]
[3.0000, 4.0000, 3.0000, 5.0000, 1.0000, 5.0000, 2.0000, 1.0000, 3.0000, 1.0000, 1.0000, 2.0000, 4.0000, 2.0000, 1.0000, 1.0000, 1.0000]
[3.0000, 4.0000, 4.0000, 2.0000, 1.0000, 5.0000, 2.0000, 1.0000, 3.0000, 1.0000, 3.0000, 2.0000, 4.0000, 1.0000, 1.0000, 1.0000, 2.0000]
[2.0000, 3.0000, 1.0000, 1.0000, 4.0000, 1.0000, 4.0000, 1.0000, 4.0000, 1.0000, 5.0000, 4.0000, 5.0000, 4.0000, 3.0000, 1.0000, 4.0000]
[2.0000, 1.0000, 1.0000, 1.0000, 4.0000, 3.0000, 2.0000, 1.0000, 2.0000, 3.0000, 3.0000, 3.0000, 2.0000, 1.0000, 4.0000, 2.0000, 5.0000]
[2.0000, 3.0000, 2.0000, 1.0000, 5.0000, 2.0000, 3.0000, 1.0000, 4.0000, 1.0000, 3.0000, 4.0000, 1.0000, 5.0000, 4.0000, 3.0000, 4.0000]
[2.0000, 3.0000, 5.0000, 1.0000, 5.0000, 2.0000, 5.0000, 1.0000, 5.0000, 1.0000, 3.0000, 5.0000, 2.0000, 4.0000, 4.0000, 3.0000, 4.0000]
[1.0000, 1.0000, 1.0000, 1.0000, 4.0000, 3.0000, 5.0000, 4.0000, 2.0000, 2.0000, 3.0000, 2.0000, 5.0000, 1.0000, 4.0000, 5.0000, 5.0000]


List of recommendations based on user similarity:

Person 1 will most likely enjoy movie 7.
Person 1 will most likely enjoy movie 14.
Person 2 will most likely enjoy movie 9.
Person 2 will most likely enjoy movie 10.
Person 2 will most likely enjoy movie 13.
Person 3 will most likely enjoy movie 6.
Person 3 will most likely enjoy movie 13.
Person 4 will most likely enjoy movie 1.
Person 4 will most likely enjoy movie 3.
Person 4 will most likely enjoy movie 9.
Person 4 will most likely enjoy movie 10.
Person 4 will most likely enjoy movie 13.
Person 4 will most likely enjoy movie 14.
Person 5 will most likely enjoy movie 9.
Person 5 will most likely enjoy movie 15.
Person 7 will most likely enjoy movie 15.
Person 7 will most likely enjoy movie 16.
Person 8 will most likely enjoy movie 5.
Person 8 will most likely enjoy movie 6.
Person 8 will most likely enjoy movie 15.
Person 8 will most likely enjoy movie 16.
Person 9 will most likely enjoy movie 2.
Person 10 will most likely enjoy movie 2.
Person 10 will most likely enjoy movie 13.
Person 11 will most likely enjoy movie 5.
Person 11 will most likely enjoy movie 7.
Person 11 will most likely enjoy movie 9.
Person 11 will most likely enjoy movie 12.
Person 11 will most likely enjoy movie 13.
Person 12 will most likely enjoy movie 5.
Person 12 will most likely enjoy movie 15.
Person 13 will most likely enjoy movie 9.
Person 13 will most likely enjoy movie 15.
Person 13 will most likely enjoy movie 17.
Person 14 will most likely enjoy movie 3.
Person 14 will most likely enjoy movie 9.
Person 15 will most likely enjoy movie 5.
Person 15 will most likely enjoy movie 7.
Person 15 will most likely enjoy movie 8.
```

Figure 8

```
Recommendation matrix based on movie/item similariy:

[4.0000, 2.0000, 5.0000, 1.0000, 4.0000, 2.0000, 5.0000, 4.0000, 1.0000, 1.0000, 1.0000, 5.0000, 2.0000, 5.0000, 4.0000, 5.0000, 1.0000]
[4.0000, 4.0000, 5.0000, 1.0000, 5.0000, 5.0000, 5.0000, 3.0000, 5.0000, 1.0000, 4.0000, 5.0000, 2.0000, 4.0000, 1.0000, 2.0000, 3.0000]
[5.0000, 4.0000, 3.0000, 2.0000, 1.0000, 5.0000, 4.0000, 3.0000, 5.0000, 4.0000, 4.0000, 5.0000, 3.0000, 4.0000, 3.0000, 3.0000, 4.0000]
[2.0000, 4.0000, 2.0000, 3.0000, 3.0000, 3.0000, 4.0000, 3.0000, 2.0000, 1.0000, 5.0000, 3.0000, 2.0000, 4.0000, 2.0000, 2.0000, 4.0000]
[1.0000, 3.0000, 1.0000, 3.0000, 2.0000, 3.0000, 4.0000, 2.0000, 4.0000, 3.0000, 2.0000, 2.0000, 3.0000, 3.0000, 3.0000, 3.0000, 4.0000]
[2.0000, 4.0000, 2.0000, 1.0000, 1.0000, 4.0000, 3.0000, 1.0000, 5.0000, 2.0000, 1.0000, 3.0000, 3.0000, 3.0000, 5.0000, 5.0000, 1.0000]
[1.0000, 5.0000, 2.0000, 3.0000, 4.0000, 4.0000, 2.0000, 4.0000, 4.0000, 3.0000, 1.0000, 2.0000, 1.0000, 3.0000, 1.0000, 1.0000, 4.0000]
[3.0000, 5.0000, 1.0000, 4.0000, 2.0000, 2.0000, 2.0000, 5.0000, 2.0000, 1.0000, 4.0000, 2.0000, 2.0000, 4.0000, 2.0000, 2.0000, 4.0000]
[1.0000, 2.0000, 3.0000, 5.0000, 2.0000, 5.0000, 2.0000, 1.0000, 3.0000, 1.0000, 2.0000, 2.0000, 4.0000, 2.0000, 1.0000, 1.0000, 3.0000]
[3.0000, 1.0000, 4.0000, 2.0000, 2.0000, 5.0000, 2.0000, 1.0000, 3.0000, 1.0000, 4.0000, 2.0000, 2.0000, 1.0000, 1.0000, 1.0000, 2.0000]
[2.0000, 3.0000, 1.0000, 1.0000, 3.0000, 1.0000, 4.0000, 2.0000, 1.0000, 1.0000, 5.0000, 3.0000, 1.0000, 4.0000, 3.0000, 1.0000, 4.0000]
[2.0000, 1.0000, 1.0000, 3.0000, 3.0000, 3.0000, 1.0000, 1.0000, 4.0000, 3.0000, 5.0000, 3.0000, 2.0000, 1.0000, 2.0000, 2.0000, 5.0000]
[2.0000, 3.0000, 3.0000, 1.0000, 5.0000, 2.0000, 3.0000, 2.0000, 2.0000, 1.0000, 3.0000, 4.0000, 2.0000, 5.0000, 3.0000, 3.0000, 4.0000]
[2.0000, 3.0000, 4.0000, 3.0000, 5.0000, 5.0000, 5.0000, 1.0000, 4.0000, 2.0000, 3.0000, 5.0000, 3.0000, 4.0000, 4.0000, 3.0000, 4.0000]
[1.0000, 1.0000, 1.0000, 1.0000, 2.0000, 2.0000, 2.0000, 1.0000, 2.0000, 2.0000, 3.0000, 2.0000, 5.0000, 2.0000, 4.0000, 5.0000, 5.0000]


List of recommendations based on movie/item similarity:

Person 1 will most likely enjoy movie 1.
Person 1 will most likely enjoy movie 7.
Person 1 will most likely enjoy movie 14.
Person 2 will most likely enjoy movie 2.
Person 2 will most likely enjoy movie 5.
Person 2 will most likely enjoy movie 6.
Person 2 will most likely enjoy movie 9.
Person 3 will most likely enjoy movie 2.
Person 3 will most likely enjoy movie 6.
Person 3 will most likely enjoy movie 17.
Person 4 will most likely enjoy movie 2.
Person 4 will most likely enjoy movie 14.
Person 5 will most likely enjoy movie 9.
Person 5 will most likely enjoy movie 17.
Person 6 will most likely enjoy movie 15.
Person 7 will most likely enjoy movie 8.
Person 8 will most likely enjoy movie 11.
Person 10 will most likely enjoy movie 11.
Person 11 will most likely enjoy movie 7.
Person 12 will most likely enjoy movie 9.
Person 12 will most likely enjoy movie 11.
Person 13 will most likely enjoy movie 17.
Person 14 will most likely enjoy movie 3.
Person 14 will most likely enjoy movie 6.
Person 14 will most likely enjoy movie 9.
```

Figure 9