Prediction Algorithm：

The algorithm which is implemented in phase 2 predicts ratings of a given user u by taking into account ratings of users whose tastes are similar to u's tastes. The similarity between two users u and v is defined as:

$$\mathrm{USim}(u,v) = \frac{\sum_{i \in I(u) \cap I(v)} (R_{u,i} - \overline{R_u})(R_{v,i} - \overline{R_v})}{\sqrt{\sum_{i \in I(u) \cap I(v)} (R_{u,i} - \overline{R_u})^2} \sqrt{\sum_{i \in I(u) \cap I(v)} (R_{v,i} - \overline{R_v})^2}}$$

Here $I(u) \cap I(v)$ is the set of movies that both user u and user v have rated, $R_{u,i}$ is user u's rating of movie i, $R_{v,i}$ is user v's rating of movie i, and R_u is user u's mean rating and R_v is user v's mean rating.

Now, I revised the calculation method of the similarity between two users. We can consider that two users are more likely to have the similar tastes if more movies have been commonly rated by them. So, we can compute the similarity between two users by considering the number of movies that they have commonly rated. The adjusted formula is defined as:

$$\mathrm{USim}'(u,v) = \exp\left(\frac{|I(u) \cap I(v)|}{\max_{w \in U} |I(u) \cap I(w)|} - 1\right) \times \mathrm{USim}(u,v)$$

Here | * | is the number of elements in a set, and max(w ∈ U) | I(u) ∩ I(w) | is the maximum value of the numbers of movies which have been commonly rated by user u and another user who is in user set. I used the natural exponential function to adjust the similarity between two users.

Once similarity between users is defined as above, we can predict the rating that a user u gives to a movie i by taking the "weighted" average of ratings that movie i has received from users who are similar to u. The predicted rating has been defined in phase 2:

$$P_{\mathrm{user}}(R_{u,i}) = \overline{R_u} + \frac{\sum_{v \in \mathrm{NearU}(u)} \mathrm{USim}'(u,v) \times (R_{v,i} - \overline{R_v})}{\sum_{v \in \mathrm{NearU}(u)} \mathrm{USim}'(u,v)}$$

Here NearU(u) is the set of users that have rated movie i and are very similar to user u. The term in the formula corresponding to a user v is weighted by USim`(u, v) implying that the more similar v is to u, the more "weight" v's rating gets in the prediction.

Moreover, we can also predict ratings of a given movie i by taking into account ratings of movies which the users are interested in at the same time. The similarity between two movies i and j is defined as:

$$\mathrm{ISim}(i,j) = \frac{\sum_{u \in U(i) \cap U(j)} (R_{u,i} - \overline{R_i})(R_{u,j} - \overline{R_j})}{\sqrt{\sum_{u \in U(i) \cap U(j)} (R_{u,i} - \overline{R_i})^2} \sqrt{\sum_{u \in U(i) \cap U(j)} (R_{u,j} - \overline{R_j})^2}}$$

Here $U(i) \cap U(j)$ is the set of users who have both rated movie i and movie j, $R_{u,i}$ is user u's rating

of movie i, $R_{u,j}$ is user u's rating of movie j, and R_i is movie i's mean rating and R_j is movie j's mean rating.

I also revised the calculation method of the similarity between two movies. We can consider that two movies are more likely to be similar if more users have commonly rated them at the same time. So, we can compute the similarity between two movies by considering the number of users who have commonly rated them. The adjusted formula is defined as:

$$\mathrm{ISim}'(i,j) = \exp\left(\frac{|U(i) \cap U(j)|}{\max_{k \in I} |U(i) \cap U(k)|} - 1\right) \times \mathrm{ISim}(i,j)$$

Here max(k∈I) | U(i)∩U(k) | is the maximum value of the numbers of users who have both rated movie i and another movie which is in movie set. I used the natural exponential function to adjust the similarity between two movies.

Once similarity between movies is defined as above, we can predict the rating that a user u gives to a movie i by taking the "weighted" average of ratings that user u has given to movies which are similar to movie i. The predicted rating is defined as:

$$P_{\mathrm{item}}(R_{u,i}) = \overline{R'_i} + \frac{\sum_{j \in \mathrm{NearI}(i)} \mathrm{ISim}'(i,j) \times (R_{u,j} - \overline{R_j})}{\sum_{j \in \mathrm{NearI}(i)} \mathrm{ISim}'(i,j)}$$

Here NearI(i) is the set of movies that have been rated by user u and are very similar to movie i. The term in the formula corresponding to a movie j is weighted by ISim`(i, j) implying that the more similar j is to i, the more "weight" j's rating gets in the prediction.

Finally, I will compute the average of two predicted ratings -- $P_{user}(R_{u,i})$ and $P_{item}(R_{u,i})$. This average will be the final value of predicted rating.

Implement:

I kept all the functions defined in Phase 1 and Phase 2, and defined some new functions:

- A function called maxCommonMovie with the following function header:

  def maxCommonMovie(u, userRatings):

This function computes the maximum value of the numbers of movies which have been commonly rated by user u and another user who is in user set.

- A function called userSimilarity with the following function header:

  def userSimilarity(u, v, userRatings):

This function computes the similarity between two users by the adjusted formula as above.

- A function called userKNearestNeighbors with the following function header:

  def userKNearestNeighbors(u, userRatings, k):

This function returns the list of (user ID, similarity)-pairs for the k users who are most similar to

user u. The user u herself should be excluded from candidates being considered by this function.

• A function called userCFRatingPrediction with the following function header:

def userCFRatingPrediction(u, m, userRatings, friends):

This function computed predicted rating based on user by a call to the CFRatingPrediction function.

• A function called maxCommonUser with the following function header:

def maxCommonUser(m, movieRatings):

This function computes the maximum value of the numbers of users who have both rated movie i and another movie which is in movie set.

• A function called similarity_2 with the following function header:

def similarity_2(m, n, movieRatings):

This function computes the similarity in ratings between two movies, using the users who have commonly rated the two movies.

• A function called movieSimilarity with the following function header:

def movieSimilarity(m, n, movieRatings):

This function computes the similarity between two movies by the adjusted formula as above.

• A function called movieKNearestNeighbors with the following function header:

def movieKNearestNeighbors(m, movieRatings, k):

This function returns the list of (movie ID, similarity)-pairs for the k movies which are most similar to movie m. The movie m itself should be excluded from candidates being considered by this function.

• A function called movieCFRatingPrediction with the following function header:

def movieCFRatingPrediction(u, m, movieRatings, friends):

This function is similar to the CFRatingPrediction function. It computed predicted rating based on movie. The argument corresponding to friends would have been computed by a call to the movieKNearestNeighbors function.

• A function called averageCFRatingPrediction with the following function header:

def averageCFRatingPrediction(u, m, userRatings, movieRatings, friends_user, friends_movie):

This function returns the average of predicted ratings which are separately computed by userCFRatingPrediction function and movieCFRatingPrediction function.

• A function called oneRun with the following function header:

def oneRun():

This function run the algorithm averageCFRatingPrediction with (1): 0 friends_user and 0 friends_movie, (2): 25 friends_user and 25 friends_movie, (3): 300 friends_user and 300 friends_movie, (4): 500 friends_user and 500 friends_movie, (5): all friends_user and all friends_movie. The results will be output to the screen.

- A function called averageTenRuns with the following function header:

def averageTenRuns():

This function perform 10 repetitions of the above process, by generating 10 different 80-20 splits of the data into training and testing sets, computing the rmse values of all of the prediction algorithms and then reporting the average rmse value of each prediction algorithm (averaged over the 10 repetitions). The results will be output to a file named "output_ec.txt".

Output:

Due to the limit of time, I did not completely operate the two functions above. Instead, I modified some parameters.

First, I set the percent of testing set to 0.002 (0.2%). It may lead to the result of experiment not accurate enough, so I also evaluated the other 14 algorithms mentioned in phase 1 and phase 2 to make a comparison.

Second, I performed 3 repetitions to compute average rmse values instead of performing 10 times. To make a comparison, I also evaluated the other 14 algorithms.

In the output, the results of optimized algorithm will be displayed after some words such as "New Collaborative Filtering Rating prediction RMSE (friends = *): " or "New Collaborative Filtering Rating prediction Average RMSE (friends = *): ".

The output of running one time is as:

Random prediction RMSE: 1.87749833555
User Mean Rating prediction RMSE: 1.0315138003
Movie Mean Rating prediction RMSE: 1.01984113464
User-Movie Mean Rating prediction RMSE: 0.976785963857
Collaborative Filtering Rating prediction RMSE (friends = 0): 1.0315138003
CollaborativeFiltering-Movie Mean Rating prediction RMSE (friends = 0): 0.976785963857
New Collaborative Filtering Rating prediction RMSE (friends = 0): 0.976785963857
Collaborative Filtering Rating prediction RMSE (friends = 25): 1.00157968669
CollaborativeFiltering-Movie Mean Rating prediction RMSE (friends = 25): 0.929821523716
New Collaborative Filtering Rating prediction RMSE (friends = 25): 0.979150407412
Collaborative Filtering Rating prediction RMSE (friends = 300): 0.911762582255
CollaborativeFiltering-Movie Mean Rating prediction RMSE (friends = 300): 0.90966086273
New Collaborative Filtering Rating prediction RMSE (friends = 300): 0.905925940517
Collaborative Filtering Rating prediction RMSE (friends = 500): 0.899850775095
CollaborativeFiltering-Movie Mean Rating prediction RMSE (friends = 500): 0.909971228654
New Collaborative Filtering Rating prediction RMSE (friends = 500): 0.89454054533
Collaborative Filtering Rating prediction RMSE (friends = numUsers): 0.893094174335
CollaborativeFiltering-Movie Mean Rating prediction RMSE(friends=numUsers): 0.905021006404
New Collaborative Filtering Rating prediction RMSE (friends = all): 0.893034106793

The output of running three times to compute average values is as:

Random prediction Average RMSE: 1.88888967228
User Mean Rating prediction Average RMSE: 1.03717709603
Movie Mean Rating prediction Average RMSE: 1.03049705652
User-Movie Mean Rating prediction Average RMSE: 0.986526136592
Collaborative Filtering Rating prediction Average RMSE (friends = 0): 1.03717709603
CollaborativeFiltering-Movie Mean Rating prediction Average RMSE (friends=0):0.986526136592
New Collaborative Filtering Rating prediction Average RMSE (friends = 0): 0.986526136592
Collaborative Filtering Rating prediction RMSE (friends = 25): 0.927829494892
CollaborativeFiltering-Movie Mean Rating prediction RMSE (friends = 25): 0.879309463333
New Collaborative Filtering Rating prediction RMSE (friends = 25): 0.908329394481
Collaborative Filtering Rating prediction RMSE (friends = 300): 0.853183806369
CollaborativeFiltering-Movie Mean Rating prediction RMSE (friends = 300): 0.873410957038
New Collaborative Filtering Rating prediction RMSE (friends = 300): 0.859090681201
Collaborative Filtering Rating prediction RMSE (friends = 500): 0.855962599493
CollaborativeFiltering-Movie Mean Rating prediction RMSE (friends = 500): 0.877064333666
New Collaborative Filtering Rating prediction RMSE (friends = 500): 0.851466940324
Collaborative Filtering Rating prediction RMSE (friends = numUsers): 0.859712747333
CollaborativeFiltering-Movie Mean Rating prediction RMSE (friends=numUsers):0.875781883298
New Collaborative Filtering Rating prediction RMSE (friends = all): 0.859026254055

We can see that, in some degree, this algorithm consistently improves the best rmse score from the algorithms described in the handout. Yet the superiority of this algorithm is not obvious enough, in part because of the paucity of testing data (just tested 200 rating data). However, taking both the similarity between items (here is movies) and the similarity between users into consideration makes sense in predicting ratings.

Finally, we can find that this algorithm still has many shortcomings, such as taking too long time, consuming a lot of memory. I will still keep trying to improve this algorithm in my free time.