CS 272 Web Search and Information Retrieval

Project II (100 points + 20 bonus points)

Due: 5:10pm, Tuesday, March 7, 2017

Tasks:

1. **User Based Collaborative Filtering Algorithms**
   - **1.1 .Basic User Based Collaborative Filtering Algorithms**

Cosine:

Hello, Ketha, Mani

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.824559209703639
MAE of GIVEN 10 : 0.789333333333333
MAE of GIVEN 20 : 0.769460789042153
OVERALL MAE : 0.792439665079626

You have already submitted 11 times.

GOOD LUCK!

Pearson:

Hello, Ketha, Mani

The following is the summary of your submissions:

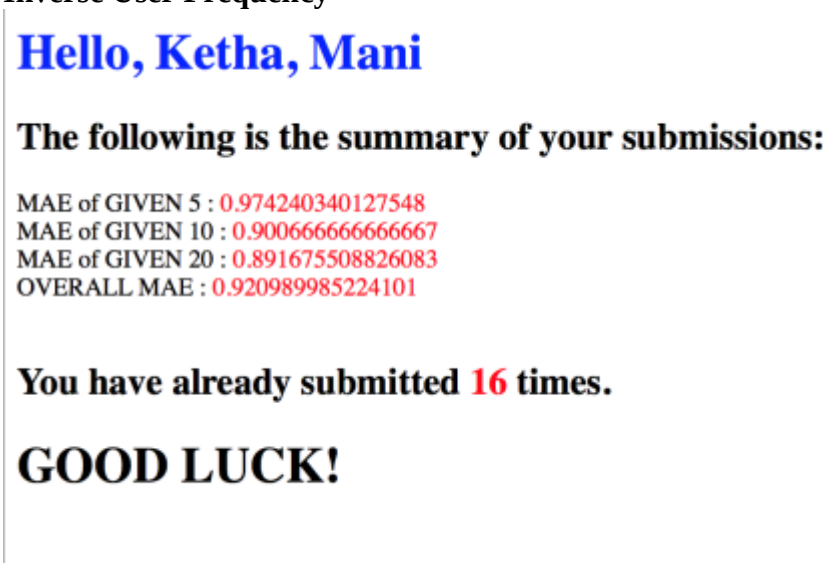
MAE of GIVEN 5 : 0.91096661247968
MAE of GIVEN 10 : 0.828833333333333
MAE of GIVEN 20 : 0.781711198996817
OVERALL MAE : 0.83574125759317

You have already submitted 12 times.

GOOD LUCK!

1.2Extensions to the Basic User-Based Collaborative Filtering Algorithm (Pearson Correlation)

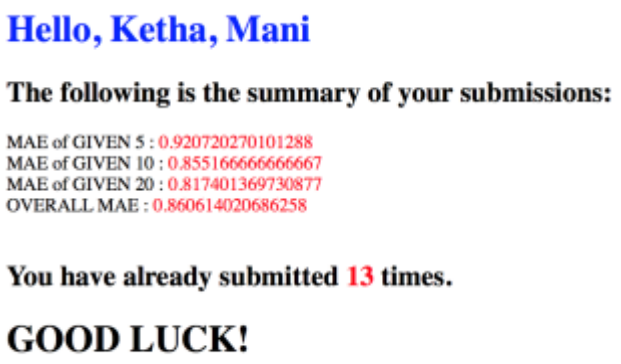
1. **Inverse User Frequency**



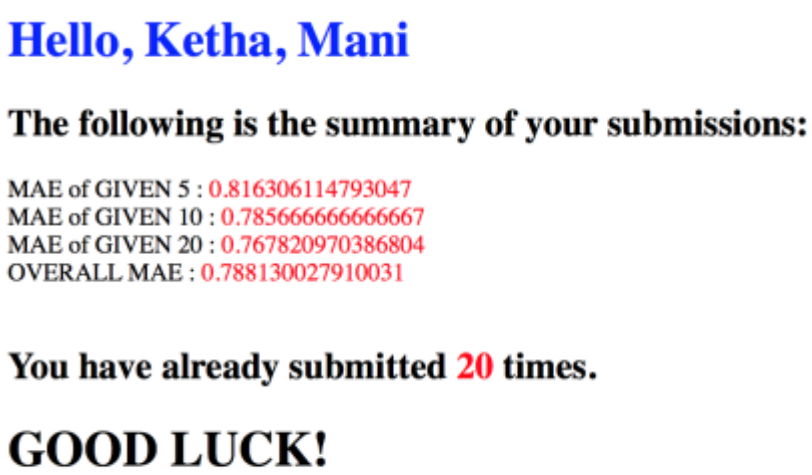
2. **Case Amplification**

   Here, I took ρ=1.5

   similarity=similarity* Math.pow(Math.abs(similarity), 1.5);



2. Item-Based Collaborative Filtering with Adjusted Cosine Similarity



3. My Own Algorithm: Ensemble Method

For my own algorithm, I was trying to improve the accuracy of my prediction. Since I was getting the best results through cosine similarity. I decided to try a combined prediction method, that is an ensemble method. In this I used cosine similarity to calculate the similarity value. But for predicting the actual value, I used pearson prediction.

An ensemble modeling is basically the process of running two or more related but different analytical models. In my algorithm I have used cosine rating and pearson rating. By synthesizing the results into a single spread, we are able to improve the accuracy of predictive analytics.

I added the following code in my pearson_rating method so that it would calculate cosine similarity instead of pearson correlation.

```
//*for combined prediction rating*//
//sorted_neighbor_list=calculate_cosine_similarity(user,k,training,movie);
```

Hello, Ketha, Mani

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.802300862823559
MAE of GIVEN 10 : 0.757833333333333
MAE of GIVEN 20 : 0.73029806115559
OVERALL MAE : 0.760712526678706

You have already submitted 17 times.

GOOD LUCK!

This method actually helped in reducing my overall MAE and basically increased the accuracy of my predictions. I was really surprised that this combined prediction helped improve my accuracy quite a bit. I think by trying more combinations I can find one which gives an even lower MAE value.

4. Results Discussion

1. Compare the accuracy of the various algorithms. Do you think your results are reasonable? How can you justify the results by analyzing the advantages and disadvantages of the algorithms?

   Ans:

   In generality, user based approaches typically have limited effectiveness. This is due to the nature of our data and since users have limited numbers or previous ratings which are available for your use. By focusing on user data it might increase the prediction accuracy. The results for my Cosine and pearson seem pretty reasonable.

   Item based filtering should be more effective at predicting ratings but my results seem to contradict that. This may possibly be due to the fact that there is a bug in my program

   My results were definitely a surprise. The best MAE value I had gotten was for the cosine similarity algorithm. Though this algorithm was simple, it was the most effective in predicting ratings. I had expected Pearson correlation to give more accurate ratings, but this was not the case. Another surprising thing was that, by applying Case Amplification to my Pearson correlation algorithm, the accuracy actually reduced and my MAE value increased compared to the basic Cosine similarity or Pearson correlation algorithms. But, when I applied IUF to my Pearson correlation algorithm, the accuracy of my predictions decreased even more and I had a higher MAE. I do think that my results are reasonable considering the algorithms involved.

   I think the reason why cosine worked better than pearson rating, even though pearson rating is supposed to generally give a more accurate prediction, is because of the size of the training data and test files. Pearson rating is a centered cosine which works very well against changes made to training data and it is invariant to shifts. In our situation, we are not facing any such shifts and also the data is very small and due to all of these reasons, this is why pearson correlation may have not been as accurate as compared to cosine similarity.

Similarly, in pearson correlation, by applying IUF and case amplification, the accuracy decreased. This may also be because of the small dataset and by calculating IUF the accuracy was reduced. This is because we have very few movies find the sum of ratings and then dividing this by the total number of ratings and finding log, this value may skewer the predictions.

2. How long does each algorithm take to complete the prediction? Discuss the efficiency of the algorithms.

   Ans: The algorithm efficiency for the simple cosine similarity algorithm was the best compared to the other algorithms. It was better by a little, in terms of time of completion for the algorithm. The difference was not too much. It took a couple of minutes to run.

   The pearson correlation algorithm also was similar to the cosine similarity algorithm. I noticed that it did run a little longer than cosine. This difference was noticed more when I tried to run it with IUF and case amplification. It took at least 10 minutes for the algorithm to run when I was trying to run the test20.txt file. This was probably because I was trying to calculate the average of the users for the entire training matrix. Since I kept doing this over and over in the loops, this probably reduced the efficiency of my algorithm. For each movie to be predicted from the test file, I would recalculate the IUF for that movie by looping through the entire train matrix. I also had to loop through the matrix when I was creating the user average list for the training matrix. All of this contributed to lowering my efficiency and increased the running time of my program. In order to generate all three result files, it took around 15-20 minutes.

   Since the item based algorithm was similar to pearson correlation, the findings for the basic algorithm were similar, in terms of efficiency.

   The efficiency of the algorithms increases with the increasing complexity of the algorithms. Though the actual formula for computing similarity or predicting the rating is very minor, this simple change adds up when we create multiple loops to gain the extra information. This adds to the overall time complexity and decreases the efficiency.

Another reason why the efficiency of my algorithm was not very good was due to the fact that I implemented my own sorting algorithm. I should have used a collections sorting method, but I wanted more control over how I was sorting the similarities. This algorithm was based on the simple bubble sort (simple and effective but not very efficient) and this added to the overall time complexity.

```
package trial;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.PriorityQueue;
import java.util.Scanner;


public class Item_Based {
	private int[][] training_matrix=new int[200][1000];
	public static ArrayList itemlist=new ArrayList();
	static ArrayList<ItemSimilar> similaritemlist=new ArrayList<ItemSimilar>();
	//CLASS K_Neighbor: user_id, rating, similarity
	//Class UMR_List:user_list, movie_list, rating_list

	/*Working correctly*/public static int[][] loading() throws
FileNotFoundException{
```

```
		int[][] trainUser = new int[200][1000];
		for(int i=0;i<trainUser.length;i++){
			for(int j=0;j<trainUser[0].length;j++){
				trainUser[i][j]=-1;
			}
		}

		Scanner sc=new Scanner(new File("train.txt"));
		for(int row=0;row<trainUser.length;row++){
			for(int column=0;column<trainUser[0].length;column++){
				trainUser[row][column]=sc.nextInt();
			}

		}
		sc.close();
		return trainUser;
	}
	/*Working correctly*/public static UMR_List predictionlist() throws
NumberFormatException, IOException{
		//System.out.println("In the test_prediction_list");
		ArrayList user_test=new ArrayList();
		ArrayList movie_test=new ArrayList();
		ArrayList rating_test=new ArrayList();
		UMR_List prediction_list;
		//BufferedReader br1 = new BufferedReader(new FileReader("test5.txt"));
		//BufferedReader br1 = new BufferedReader(new
FileReader("test10.txt"));
		BufferedReader br1 = new BufferedReader(new FileReader("test20.txt"));

		String regex="[,\\s]+";
		String line=null;
		while ((line = br1.readLine()) != null)
		{
			String[] st =line.split(regex);
			int user_read=Integer.parseInt(st[0]);
```

```
                int movie_read=Integer.parseInt(st[1]);
                int rating_read=Integer.parseInt(st[2]);
                if(rating_read==0){
                        user_test.add(user_read);
                        movie_test.add(movie_read);
                        rating_test.add(rating_read);
                }
        }
        prediction_list=new UMR_List(user_test,movie_test,rating_test);
        br1.close();
        return prediction_list;
    }
    /*Working correctly*/public static UMR_List testratedlist(int user) throws
NumberFormatException, IOException{
        //System.out.println("In the test_rated_users_list");
        ArrayList user_test=new ArrayList();
        ArrayList movie_test=new ArrayList();
        ArrayList rating_test=new ArrayList();
        UMR_List test_rated_list;
        //BufferedReader br1 = new BufferedReader(new FileReader("test5.txt"));
        //BufferedReader br1 = new BufferedReader(new
FileReader("test10.txt"));
        BufferedReader br1 = new BufferedReader(new FileReader("test20.txt"));
        String regex="[,\\s]+";
        String line=null;
        while ((line = br1.readLine()) != null)
        {
                String[] st =line.split(regex);
                //System.out.println(line);
                int user_read=Integer.parseInt(st[0]);
                int movie_read=Integer.parseInt(st[1]);
                int rating_read=Integer.parseInt(st[2]);
                if(user_read==user&&rating_read!=0){
                        user_test.add(user_read);
                        movie_test.add(movie_read);
                        rating_test.add(rating_read);
```

```
                }

            }
            test_rated_list=new UMR_List(user_test,movie_test,rating_test);
            br1.close();


            return test_rated_list;
        }


        /*Working correctly-changed to double*/public static ArrayList
trainaverage(int[][] train_matrix) throws FileNotFoundException
        {
//              PrintStream out2 = new PrintStream(new FileOutputStream("trial.txt"));
//              System.setOut(out2);
            ArrayList train_user=new ArrayList();
            for(int i=0;i<train_matrix.length;i++){
                double total=0;//changed from int
                double average=0;
                double count=0;//changed from int
                for(int j=0;j<train_matrix[i].length;j++){
                    if(train_matrix[i][j]!=0){
                        total+=train_matrix[i][j];
                        count++;
                    }
                }
                if(count==0){
                    average=0;
                }
                else{
                    average=total/count;

                }
                train_user.add(average);
                //System.out.println("User Average: "+average);
```

```
                }
//              for(int i=0;i<1;i++)
//              {
//                      double count=0;
//                      double tot=0;
//                      double avg=0;
//                      for(int j=0;j<1000;j++)
//                      {
//                              if(train_matrix[i][j]!=0){
//                              //System.out.println(train_matrix[i][j]);
//                              tot=tot+train_matrix[i][j];
//                              count++;
//                              }
//                      }
//                      avg=tot/count;
//                      System.out.println("total:"+tot+" "+"count"+count+" "+avg);
//              }

                return train_user;

        }
        /*Working correctly-changed to double*/public static ArrayList
train_movie_average(int[][] train_matrix) throws NumberFormatException,
IOException{

                ArrayList train_user=new ArrayList();
                for(int j=0;j<train_matrix[0].length;j++){
                        double total=0;
                        double average=0;
                        double count=0;
                        for(int i=0;i<train_matrix.length;i++){
                                if(train_matrix[i][j]!=0){
                                        total+=train_matrix[i][j];
                                        count++;
                                }
                        }
```

```
                        if(count==0){
                                average=0;
                        }
                        else{
                                average=total/count;

                        }
                        train_user.add(average);
                        //System.out.println("Movie Average: "+average);
                }
                return train_user;
        }
        /*Working correctly*/public static double  test_user_average_rating(int user)
throws IOException{
                //System.out.println("In the test_rated_users_list");
                double rating=0;
                double rateavg=0;
                double count=0;
                //BufferedReader br1 = new BufferedReader(new FileReader("test5.txt"));
                //BufferedReader br1 = new BufferedReader(new
FileReader("test10.txt"));
                BufferedReader br1 = new BufferedReader(new FileReader("test20.txt"));
                String regex="[,\\s]+";
                String line=null;
                while ((line = br1.readLine()) != null)
                {
                        String[] st =line.split(regex);
                        int user_read=Integer.parseInt(st[0]);
                        int movie_read=Integer.parseInt(st[1]);
                        int rating_read=Integer.parseInt(st[2]);
                        if(user_read==user&&rating_read!=0){
                                rating+=rating_read;
                                count++;
                        }
```

```
                }
                //System.out.println(rating+" "+count);
                br1.close();
                if(count==0){
                        rateavg=0;
                }
                else{
                        rateavg=rating/count;
                }
                return rateavg;
                //return test_avgrate_list;
        }

        public static ArrayList<K_Neighbor> itembased_adjustedcosinesimilarity(int user,int movie, int[][] training, int k) throws NumberFormatException, IOException{
                ArrayList<K_Neighbor> neighborlist=new ArrayList<K_Neighbor>();
                ArrayList<K_Neighbor> sortedneighborlist=new ArrayList<K_Neighbor>();
                //Step1: Get test vector

                UMR_List testvectorlist=testratedlist(user);//rated movies for user in testfile
                //Step2: Get movie list from test vector list
                ArrayList movielist=testvectorlist.get_movie_list();/*Working correctly*///got list of movies for that user in test file

                //Step2: Get train vector average list
                ArrayList trainuseraveragelist=trainaverage(training);/*Working correctly*///getting list of user averages in training



                //Step3: Go through movie list and calculate similarity
                for(int i=0;i<movielist.size();i++)//go through list of rated movies in movie list from test for user 201
                {
```

```
                    //WE ARE COMPARING TWO MOVIES AND NOT TWO
USERS
                    //MOVIES BEING COMPARED are movie(what needs to be
predicted) and movieid(in the rated test vector)

                    int ratedmovieid=(int) movielist.get(i);//got movie id

                    double num=0;
                    double den1=0;
                    double den2=0;
                    double dentotal=0;
                    int commonuser=0;
                    double similarity=0;

                    for(int j=0;j<200;j++)//go through all users to compare the two
movies(movieid(movie in predictionlist) and movie(from test))
                              //we are going through users and see what they are rating
for both the movies
                              {
                              double trainaverage=(double)
trainuseraveragelist.get(j);//user average in training/*Working correctly*/
                              //System.out.println(trainaverage);
                              if((training[j][ratedmovieid-1] != 0)&&(training[j][movie-
1] != 0))/*Working correctly*/
                                        {
                                                  num =num+ (training[j][ratedmovieid-1]-
trainaverage) * (training[j][movie-1] - trainaverage);//same user(j) but different
movies(movie,ratedmovieid)
                                                  den1 =den1+
Math.pow((training[j][ratedmovieid-1]-trainaverage), 2);//NEW
                                                  den2 =den2+ Math.pow((training[j][movie-
1]-trainaverage), 2);//NEW
                                                  commonuser =commonuser+ 1;
                                        }

                              }
```

```
                    /*Should be working correctly*/
                    if (commonuser > 1){
                        dentotal = Math.sqrt(den1) * Math.sqrt(den2);
                        similarity = num/dentotal;


                        //Dirichlet smoothing
                        similarity *= (commonuser/(commonuser + 2));

                        // improving prediction II: case amplification
                        //similarity *= Math.pow(Math.abs((float)similarity),
1.5);//changed position of float from out of abs to inside
                        K_Neighbor k1=new K_Neighbor(i,similarity);
                        neighborlist.add(k1);
                        //itemlist.add(i,movie);//NEW
                        ItemSimilar i1=new ItemSimilar(i,movie);
                        similaritemlist.add(i1);

                    }



        }
//              int c=0;
//              for(int p=0;p<neighborlist.size();p++)
//              {
//                  System.out.println(neighborlist.get(p).get_similarity());
//                  c++;
//              }
//              System.out.println("Done");
//              System.out.println("count "+c);

//              sortedneighborlist=k_list_sort(neighborlist, k);
//              int c=0;
//              for(int p=0;p<sortedneighborlist.size();p++)
```

```
//		{
//			System.out.println(sortedneighborlist.get(p).get_similarity());
//			c++;
//		}
//		System.out.println("Done");
//		System.out.println("count "+c);
		return(neighborlist);
	}
	public static ArrayList<K_Neighbor> calculate_cosine_similarity(int user, int k,int[][] training,int movie) throws NumberFormatException, IOException{


		ArrayList<K_Neighbor> neighbor_list=new ArrayList<K_Neighbor>();
		ArrayList<K_Neighbor> sorted_neighbor_list=new ArrayList<K_Neighbor>();
		ArrayList<K_Neighbor> sorted_list=new ArrayList<K_Neighbor>();

		ArrayList movie_list=new ArrayList();
		ArrayList rating_list=new ArrayList();
		ArrayList k_sort=new ArrayList();

		UMR_List test_vector_list;
		test_vector_list=testratedlist(user);
		//going to use the following in the loops
		movie_list=test_vector_list.get_movie_list();//getting list of ratedmovies in test
		rating_list=test_vector_list.get_rating_list();


		for(int userno=0;userno<200;userno++)//going through all users
		{
			//to find rating for movie

	//similarity(userno(train),user(test))=[rating(testvectormovie1)*rating(trainvectormovie1)]+[movie2]+...+[movien]
			//divide by sqrt[rating(trainvectormovie1)^2 * movie2^2....] *
```

sqrt[rating(testvectormovie1)^2 * movie2^2....]

```
double numerator=0;
double denominator_train=0;
double denominator_test=0;
double denominator_total=0;
double similarity=0;
int iuf_mv_id=0;//for iuf

for(int movieno=0;movieno<movie_list.size();movieno++)//comparing with movies that have already been rated in the test vector
{
int movie_id=(int) movie_list.get(movieno);//get first movie on testvector list
iuf_mv_id=movie_id;//for IUF
//if(((int) rating_list.get(movieno)!=0)&&(training[userno][movie_id-1]!=0))
if((training[userno][movie_id-1]!=0)&&(training[userno][movie-1]!=0))
{
numerator=numerator+(int) rating_list.get(movieno) * training[userno][movie_id-1];//added -1

denominator_test=denominator_test+Math.pow((int) rating_list.get(movieno),2);

denominator_train=denominator_train+Math.pow(training[userno][movie_id-1], 2);//added -1
}
}
//after going through entire movielist
denominator_test=Math.sqrt(denominator_test);
denominator_train=Math.sqrt(denominator_train);
denominator_total=denominator_test*denominator_train;
if(denominator_total!=0)
{
similarity=numerator/denominator_total;
```

```
                                //case Amplification
                                // similarity=similarity* Math.pow(Math.abs(similarity),
1.5);//new similarity amplifies difference between users and helps in finiding similar
users
                                //IUF
                                 //double iuf = inverse_user_frequency(iuf_mv_id,training);
                                 //similarity *= iuf;
                        }
                        K_Neighbor n1=new K_Neighbor(userno,similarity);
                        neighbor_list.add(n1);

                }//end of user(userno) for loop
                //System.out.println(neighbor_list.size());
//              sorted_list=k_list_sort(neighbor_list,k);//got sorted list
//              return sorted_list;
                //System.out.println("Finished calculate_cosine_similarity");
                return neighbor_list;

        }
        public static ArrayList<K_Neighbor> calculate_pearson_correlation(int user, int
k,int[][] training,int movie) throws NumberFormatException, IOException{
                ArrayList<K_Neighbor> neighbor_list=new ArrayList<K_Neighbor>();
                ArrayList<K_Neighbor> sorted_neighbor_list=new
ArrayList<K_Neighbor>();
                ArrayList<K_Neighbor> sorted_list=new ArrayList<K_Neighbor>();

                ArrayList movie_list=new ArrayList();
                ArrayList rating_list=new ArrayList();
                ArrayList k_sort=new ArrayList();

                UMR_List test_vector_list;
                test_vector_list=testratedlist(user);
                //going to use the following in the loops
                movie_list=test_vector_list.get_movie_list();//getting list of ratedmovies
in test
```

```
rating_list=test_vector_list.get_rating_list();

double testuseraverage=test_user_average_rating(user);

for(int userno=0;userno<200;userno++)//going through all users
{
	//to find rating for movie

//similarity(userno(train),user(test))=[rating(testvectormovie1)*rating(trainvector movie1)]+[movie2]+...+[movien]
	//divide by sqrt[rating(trainvectormovie1)^2 * movie2^2....] * sqrt[rating(testvectormovie1)^2 * movie2^2....]
	double numerator=0;
	double denominator_train=0;
	double denominator_test=0;
	double denominator_total=0;
	double similarity=0;
	int iuf_mv_id=0;//for iuf
	ArrayList trainuseraveragelist=trainaverage(training);
	double trainuseraverage=(double) trainuseraveragelist.get(userno);

	for(int movieno=0;movieno<movie_list.size();movieno++)//comparing with movies that have already been rated in the test vector
		{
			int movie_id=(int) movie_list.get(movieno);//get first movie on testvector list
			iuf_mv_id=movie_id;//for IUF
			//IUF: reducing the importance of widely rated movies and increasin importance of those movies which have not been rated as much
			//popularity factor control
			//only affects similarity
			double iuf = inverse_user_frequency(iuf_mv_id,training);

				if((training[userno][movie_id-
```

```
1]!=0)&&(training[userno][movie-1]!=0))
                                                {
                                                        //FOR NORMAL
//                                                      numerator=numerator+(((int)
rating_list.get(movieno)-testuseraverage) * (training[userno][movie_id-1]-
trainuseraverage));//added -1
//
        denominator_test=denominator_test+Math.pow((((int) rating_list.get(movieno))-
testuseraverage),2);
//
        denominator_train=denominator_train+Math.pow((training[userno][movie_id-1]-
trainuseraverage), 2);//added -1
                                                        //FOR IUF ONLY
                                                        numerator=numerator+(((int)
rating_list.get(movieno)-testuseraverage) * ((training[userno][movie_id-1]*iuf)-
trainuseraverage));//added -1

        denominator_test=denominator_test+Math.pow((((int) rating_list.get(movieno))-
testuseraverage),2);

        denominator_train=denominator_train+Math.pow(((training[userno][movie_id-
1]*iuf)-trainuseraverage), 2);//added -1
                                                }
                                        }
                                //after going through entire movielist
                                denominator_test=Math.sqrt(denominator_test);
                                denominator_train=Math.sqrt(denominator_train);
                                denominator_total=denominator_test*denominator_train;
                                if(denominator_total!=0)
                                {
                                        similarity=numerator/denominator_total;

                                        //case Amplification
                                        // similarity=similarity* Math.pow(Math.abs(similarity),
1.5);//new similarity amplifies difference between users and helps in finiding similar
users
```

```
                        // similarity *= iuf;
                        K_Neighbor n1=new K_Neighbor(userno,similarity);
                        neighbor_list.add(n1);
                    }


                }//end of user(userno) for loop
                //System.out.println(neighbor_list.size());
//                  sorted_list=k_list_sort(neighbor_list,k);//got sorted list
//                  return sorted_list;
                //System.out.println("Finished calculate_cosine_similarity");
                return neighbor_list;

        }
//
        public static ArrayList<K_Neighbor> k_list_sort(ArrayList<K_Neighbor>
unsorted_list, int k){
                //implementing bubble sort
                ArrayList<K_Neighbor> sorted_top_k_neighbors=new
ArrayList<K_Neighbor>();
                int n=unsorted_list.size();

                for(int i=0;i<n;i++){
                        for(int j=1;j<(n-i);j++){
                                double similarity1=unsorted_list.get(j).get_similarity();
                                double similarity2=unsorted_list.get(j-1).get_similarity();
                                int user_id1=unsorted_list.get(j).get_user_id();
                                int user_id2=unsorted_list.get(j-1).get_user_id();
                                //System.out.println("Similairty1: "+ similarity1);
                                //System.out.println("Similairty2: "+ similarity2);
                                if(similarity2>similarity1){
                                        double temp;
                                        int temp1;
                                        temp=similarity2;
                                        temp1=user_id2;
```

```
                                        unsorted_list.get(j-1).set_similarity(similarity1);
                                        unsorted_list.get(j-1).set_user_id(user_id1);
                                        unsorted_list.get(j).set_similarity(temp);
                                        unsorted_list.get(j).set_user_id(temp1);


                                }
                        }
                }//end 1st for loop
                //got sorted list

                //                  for(int l=0;l<k;l++){
                //                          sorted_top_k_neighbors.add(unsorted_list.get(l));
                //
        //System.out.println(unsorted_list.get(l).get_user_id()+"
"+unsorted_list.get(l).get_similarity());
                //                  }
                //problem with k:not enough similar users to get k users back
                for(int l=0;l<unsorted_list.size();l++){
                        sorted_top_k_neighbors.add(unsorted_list.get(l));
                        //System.out.println(unsorted_list.get(l).get_user_id()+"
"+unsorted_list.get(l).get_similarity());
                }
//                                  for(int l=0;l<unsorted_list.size();l++){
//
//                                          System.out.println("User_id:
"+sorted_top_k_neighbors.get(l).get_user_id()+" Similarity:
"+sorted_top_k_neighbors.get(l).get_similarity());
//                                  }
//                                  System.out.println("ONE ROUND DONE");
                return sorted_top_k_neighbors;
                //return unsorted_list;
                //ALGORITHM CORRECT
        }
```

```
	public static int itemrate(int user, int movie,int[][] training, int k) throws NumberFormatException, IOException{
		//step1: Calculate similarity
		ArrayList<K_Neighbor> similaritylist=itembased_adjustedcosinesimilarity(user,movie,training, k);//get neighbor list

		double testrateaverage=test_user_average_rating(user);//useraverage in test

		ArrayList trainmovieaverage=train_movie_average(training);//movieaveragerating in train
		double averagemovierating=(double) trainmovieaverage.get(movie-1);//changed to movie-1

		double result=0;
		double numerator=0;
		double denominator=0;
		double rate=0;
		int final_rating=0;

		UMR_List givenobject=testratedlist(user);//vector rated in test
		ArrayList movielist=givenobject.get_movie_list();
		ArrayList userlist=givenobject.get_user_list();
		ArrayList ratelist=givenobject.get_rating_list();

		for(int i=0;i<movielist.size();i++)
		{

			int movieid=(int) movielist.get(i);//list in test file which are rated
			int rating=(int) ratelist.get(i);//list in test file which are rated

			for(int j=0;j<similaritylist.size();j++)//go through k neighbors(items)
			{
				double neighborsimilarity =
```

```
similaritylist.get(j).get_similarity();//similarity
                        int userid = similaritylist.get(j).get_user_id();//userid
                        //int similarmovieid=(int) itemlist.get(userid);
                        int similarmovieid=0;
                        for(int l=0;l<similaritemlist.size();l++){
                        int a=similaritemlist.get(l).getUserId();
                        if(a==userid){
                                similarmovieid=a;
                        }
                        }

                        if( movieid == similarmovieid)//CHANGED
                        {
                                //System.out.println("Working");
                                numerator =numerator+ neighborsimilarity *
rating;//rating is the rating made by the active user on movie j
                                //that is rating made by the user(who we have to
predict the rating for a movie) in the test list vector
                                denominator =denominator+
Math.abs(neighborsimilarity);
                        }

                }
                if (denominator != 0.0){
                        result = numerator/denominator;
                }
                else if( averagemovierating != 0)
                {
                        result = averagemovierating;
                }
                else
                {
                        result = testrateaverage;
                }

//                      rate = (int)(Math.round(result));
```

```
                    rate=  (double)Math.round(result);//changed to round

                    if(Double.isNaN(rate)){
                            rate=1;
                    }

                    final_rating=(int) rate;

                    if(final_rating<=0){
                            final_rating=1;
                    }
                    else if(final_rating>5){
                            final_rating=5;
                    }



            }
            return final_rating;
    }
    public static int calculate_cosine_rating(int user, int movie, int k,int[][] training)
throws NumberFormatException, IOException{

            ArrayList<K_Neighbor> sorted_neighbor_list=new
ArrayList<K_Neighbor>();

    sorted_neighbor_list=calculate_cosine_similarity(user,k,training,movie);//list
contains trainuserno and similarity

            double test_rateavg=0;
            double train_movie_rateavg=0;

            int train_user_id=0;
            double train_similarity=0;
            double numerator=0;
```

```
double denominator=0;
int neighbor_id=0;
double rating=0;
double result=0;
int final_rating=0;

ArrayList train_movie_average_rating=new ArrayList();//new
train_movie_average_rating=train_movie_average(training);
train_movie_rateavg=(double) train_movie_average_rating.get(movie-
1);//new
//train_movie_rateavg=3;

test_rateavg=test_user_average_rating(user);//new


for(int i=0;i<sorted_neighbor_list.size();i++)
{
	train_user_id=sorted_neighbor_list.get(i).get_user_id();
	train_similarity=sorted_neighbor_list.get(i).get_similarity();
	rating=training[train_user_id][movie-1];//added -1
	//System.out.println("train_user_id "+train_user_id+"
train_similarity "+train_similarity+" movie "+movie);
	if(training[train_user_id][movie-1]!=0)//added -1
	{
		numerator=numerator+(rating*train_similarity);//rating is
from the training matrix
		denominator=denominator+train_similarity;
	}
}
if(denominator!=0){
	result=numerator/denominator;
}
else
{
	if(train_movie_rateavg!=0)
	{
```

```
                result=train_movie_rateavg;
                }
                else
                {
                result=test_rateavg;
                }
        }

        double result1=  Math.round(result);//changed to round

        if(Double.isNaN(result1)){
                result1=1;
        }

        final_rating=(int) result1;

        if(final_rating<=0){
                final_rating=1;
        }
        else if(final_rating>5){
                final_rating=5;
        }

        //System.out.println("Finished calculate_cosine_rating");
        return final_rating;
    }
    public static int calculate_pearson_rating(int user, int movie, int k,int[][] training)
throws NumberFormatException, IOException{
        //System.out.println("In pearson rating");
//      training[userno][movie_id-1]-trainuseraverage
        ArrayList<K_Neighbor> sorted_neighbor_list=new
ArrayList<K_Neighbor>();

    sorted_neighbor_list=calculate_pearson_correlation(user,k,training,movie);//list
contains trainuserno and similarity
        //*for combined prediction rating*//
```

```
		//sorted_neighbor_list=calculate_cosine_similarity(user,k,training,movie);
		//System.out.println("In pearson rating after getting similarity");
		double test_rateavg=0;
		double train_movie_rateavg=0;

		int train_user_id=0;
		double train_similarity=0;
		double numerator=0;
		double denominator=0;
		int neighbor_id=0;
		double rating=0;
		double result=0;
		int final_rating=0;

		ArrayList train_movie_average_rating=new ArrayList();//new
		train_movie_average_rating=train_movie_average(training);
		train_movie_rateavg=(double) train_movie_average_rating.get(movie-
1);//new
		//train_movie_rateavg=3;

		test_rateavg=test_user_average_rating(user);//new
		//System.out.println(test_rateavg);

		ArrayList trainuseraveragelist=trainaverage(training);//list of user
averages in train//NEW


		for(int i=0;i<sorted_neighbor_list.size();i++)
		{
			train_user_id=sorted_neighbor_list.get(i).get_user_id();
			train_similarity=sorted_neighbor_list.get(i).get_similarity();
			rating=training[train_user_id][movie-1];//added -1
			//System.out.println("train_user_id "+train_user_id+"
train_similarity "+train_similarity+" movie "+movie);
			double trainuseraverage=(double) trainuseraveragelist.get(i);
			if(training[train_user_id][movie-1]!=0)//added -1
```

```
                {
                        numerator=numerator+((rating-
trainuseraverage)*train_similarity);//NEW CHANGE HERE
                        denominator=denominator+Math.abs(train_similarity);
                }
        }
        if(denominator!=0){
                result=numerator/denominator;
                //System.out.println(result);
        }
        else
        {
                if(train_movie_rateavg!=0)
                {
                result=train_movie_rateavg;
                }
                else
                {
                result=test_rateavg;
                }
        }

        result=test_rateavg+result;//NEW
        //System.out.println("result: "+result);
        double result1=  Math.round(result);//changed to round
        //System.out.println("result1: "+result1);
        if(Double.isNaN(result1)){
                result1=1;
        }

        final_rating=(int) result1;

        if(final_rating<=0){
                final_rating=1;
        }
        else if(final_rating>5){
```

```
                final_rating=5;
        }
        //System.out.println(final_rating);
        //System.out.println("Finished calculate_cosine_rating");
        return final_rating;
    }


    public static double inverse_user_frequency(int movie,int[][]train_matrix) throws
NumberFormatException, IOException{
        int m=0;//number of users who have rated that movie
        double iuf=0;

        for(int u=0;u<200;u++)//going through training matrix
        {
            if(train_matrix[u][movie-1]!=0)
            {
                m+=1;
            }


        }
        if(m!=0){
            iuf=Math.log10(200.0/m);//total number of movies/number of
users who have rated that movie
        }
        else
            iuf=1;//means m=0 and this movie is not important
        return iuf;
    }

    public static void main(String args[]) throws NumberFormatException,
IOException{
//          PrintStream out = new PrintStream(new
FileOutputStream("trial.txt"));
//          System.setOut(out);
```

```
//                      PrintStream out = new PrintStream(new
FileOutputStream("trial1.txt"));
//                      System.setOut(out);
                        PrintStream out = new PrintStream(new
FileOutputStream("trial2.txt"));
                        System.setOut(out);
                //step 1:load the matrix
                int[][] training=loading();
                //System.out.println("Hello");
                //step 2: get the prediction list from the testfile about movies we have to
rate
                UMR_List predictlist=predictionlist();
                //step 3 iterate over the prediction list and calculate the rating
                int k=100;
                //System.out.println("Hi");
                for(int i=0;i<predictlist.get_user_list().size();i++)
                {       //System.out.println("Hello");
                        int user=(int) predictlist.get_user_list().get(i);
                        int movie=(int) predictlist.get_movie_list().get(i);
                        int rating=(int) predictlist.get_rating_list().get(i);
                        //ITEMBASED
                        //int item_rating=itemrate(user,movie,training,k);
                        //System.out.println("Hello");
                        //System.out.println(user+" "+movie+" "+item_rating);
                        //System.out.println("Hello123");
                        //COSINE
                        //int cosine_rating=calculate_cosine_rating(user,movie,k,training);
                        //int
pearson_rating=calculate_pearson_rating(user,movie,k,training);
                        int item_rating=itemrate(user,movie,training,k);
                        //System.out.println("Hello");
                        //System.out.println(user+" "+movie+" "+cosine_rating);
                        //System.out.println(user+" "+movie+" "+pearson_rating);
                        System.out.println(user+" "+movie+" "+item_rating);
                }
        }
```

```
	}
```

```
package trial;


public class K_Neighbor {
	private int user_id;
	private double similarity;
	private int rating;
public K_Neighbor(){}
public K_Neighbor(int user_id, double similarity,int rating){
	this.user_id=user_id;
	this.similarity=similarity;
	this.rating=rating;
}
public K_Neighbor(int user_id, double similarity){
	this.user_id=user_id;
	this.similarity=similarity;

}
public double get_similarity(){
	return similarity;
}
public int get_user_id(){
	return user_id;
}
public int get_rating(){
	return rating;
}

public void set_similarity(double sim1){
	this.similarity=sim1;
}
public void set_user_id(int u1){
	this.user_id=u1;
}
public void set_rating(int rating){
	this.rating=rating;
}
}
```

```
package trial;

import java.util.ArrayList;

public class UMR_List {
	private ArrayList user_list=null;
	private ArrayList movie_list=null;
	private ArrayList rating_list=null;
	public UMR_List(){

	}
	public UMR_List(ArrayList user_list,ArrayList movie_list,ArrayList rating_list){
		this.user_list=user_list;
		this.movie_list=movie_list;
		this.rating_list=rating_list;
	}
```

```
	public ArrayList get_user_list(){
		return user_list;
	}
	public ArrayList get_movie_list(){
		return movie_list;
	}
	public ArrayList get_rating_list(){
		return rating_list;
	}
	public void set_user_list(ArrayList user_list){
		this.user_list=user_list;

	}
	public void set_movie_list(ArrayList movie_list){

		this.movie_list=movie_list;

	}
	public void set_rating_list(ArrayList rating_list){
		this.rating_list=rating_list;
	}
}



package trial;

public class ItemSimilar {
	private int userid;
	private int movieid;
public ItemSimilar(){}
public ItemSimilar(int userid,int movieid){
	this.userid=userid;
	this.movieid=movieid;
}
public void setUserId(){
	this.userid=userid;
}
public void setMovieId(){
	this.movieid=movieid;
}
public int getUserId(){
	return userid;
}
public int getMovieId(){
	return movieid;
}
}
```