**CS 272 Web Search and Information Retrieval**

**Project II (100 points + 20 bonus points)**

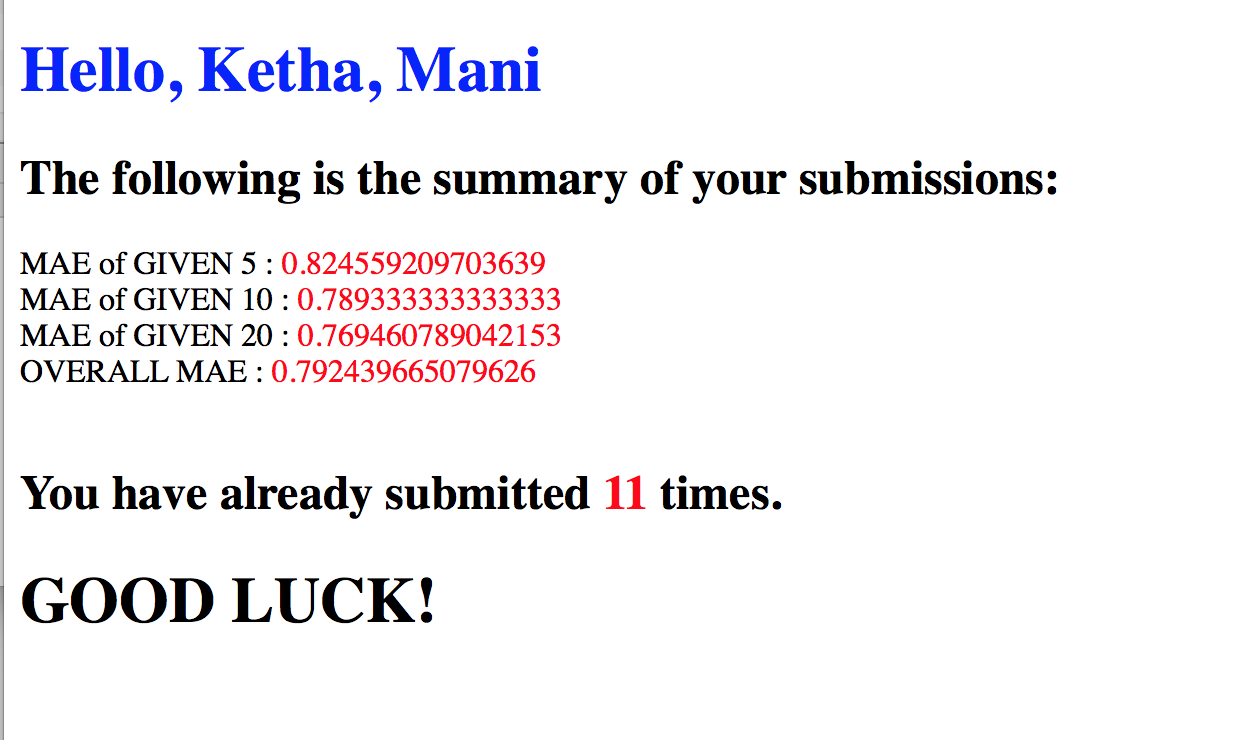
**Due: 5:10pm, Tuesday, March 7, 2017**

**Tasks:**

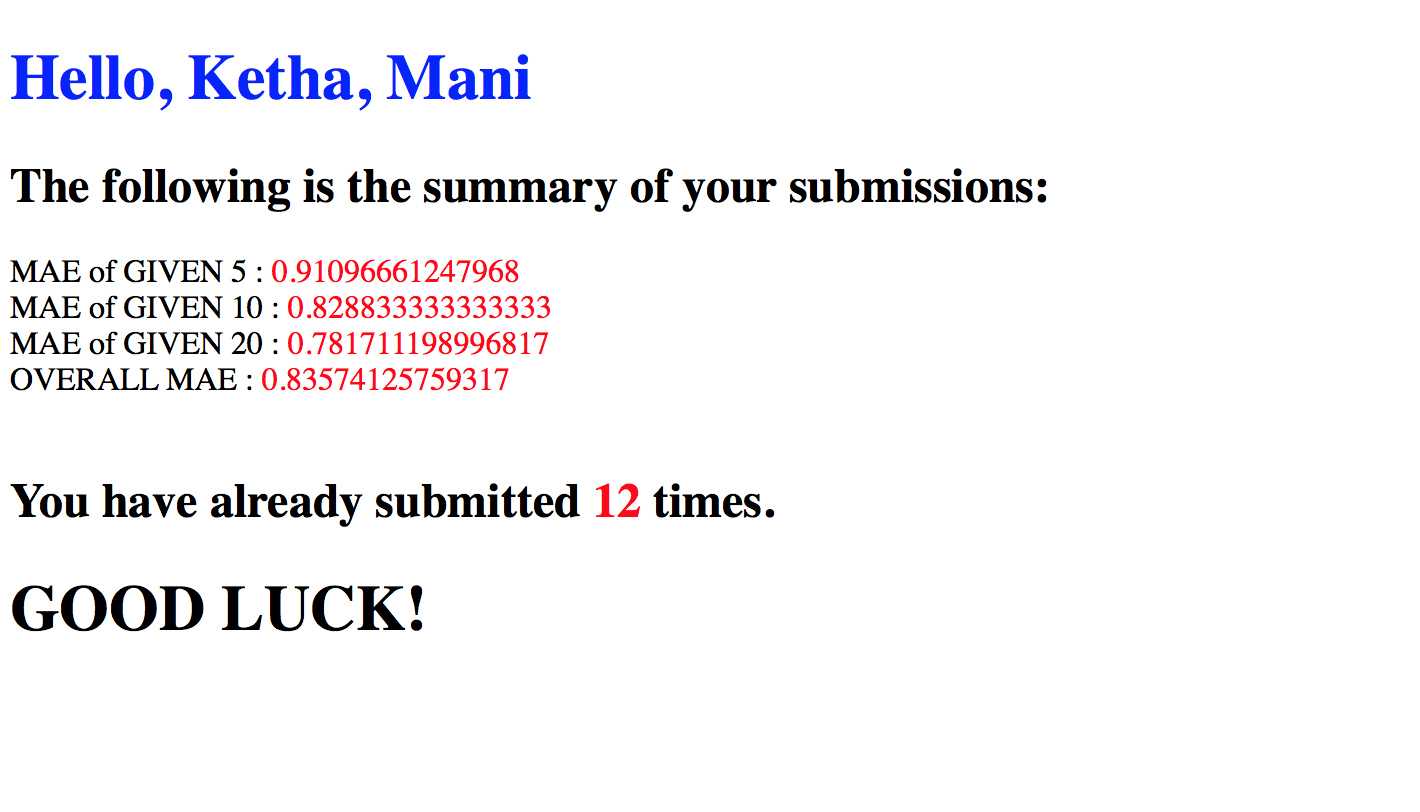
1. **User Based Collaborative Filtering Algorithms**

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

Cosine:

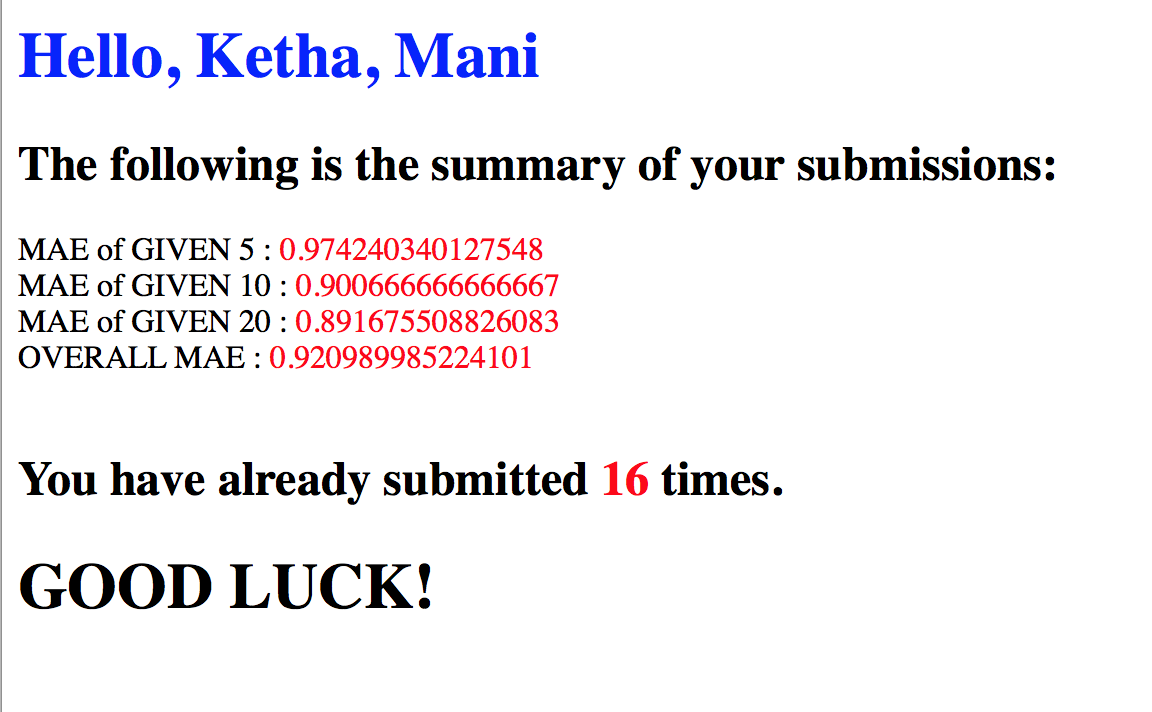


Pearson:



* 1. **Extensions to the Basic User-Based Collaborative Filtering Algorithm (Pearson Correlation)**

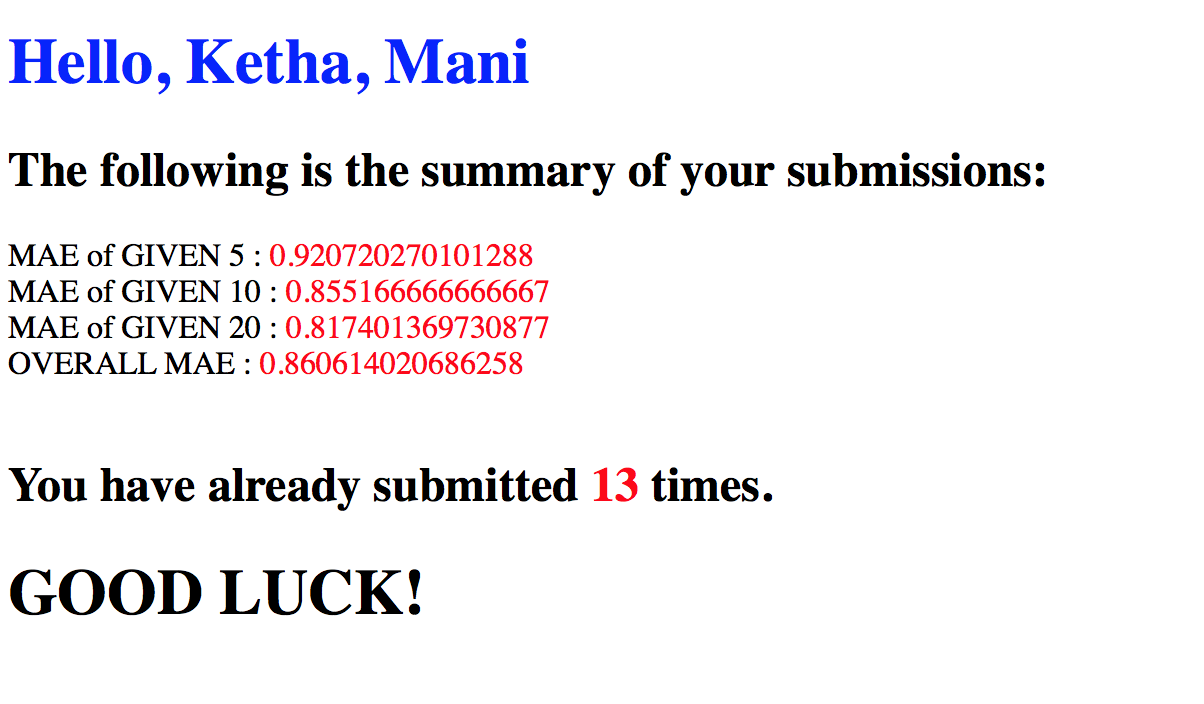
1. **Inverse User Frequency**

****

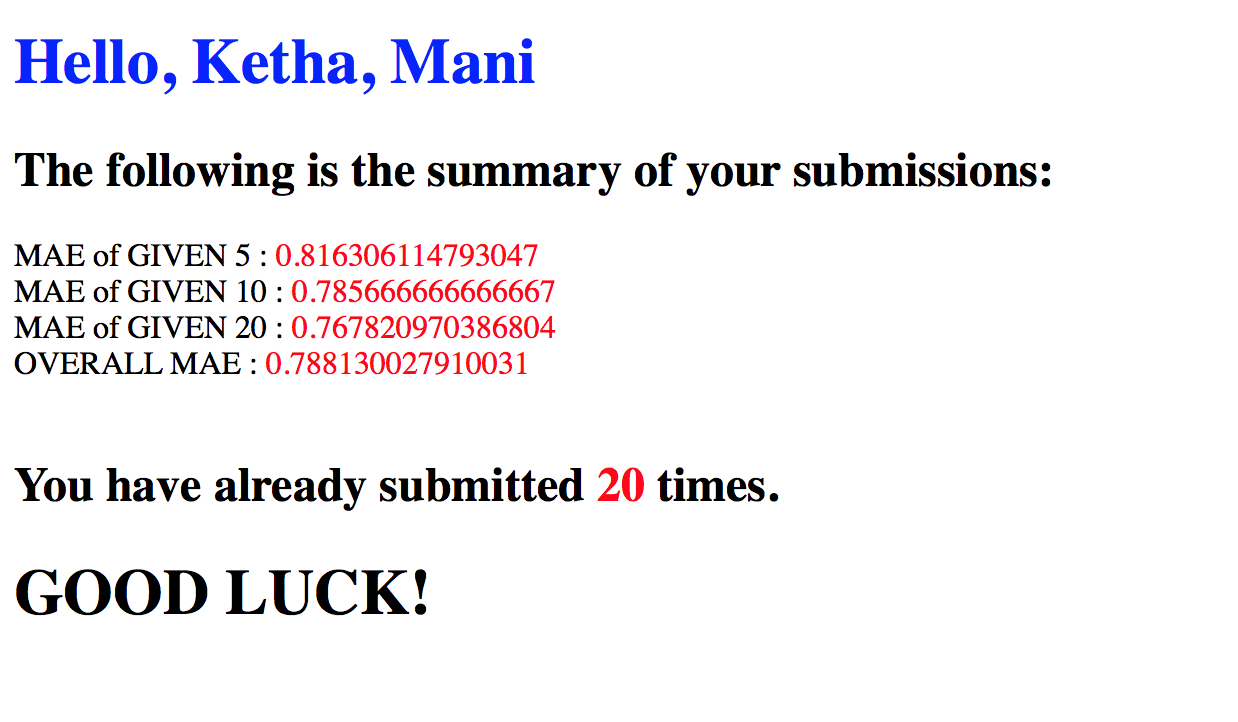
1. **Case Amplification**

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**

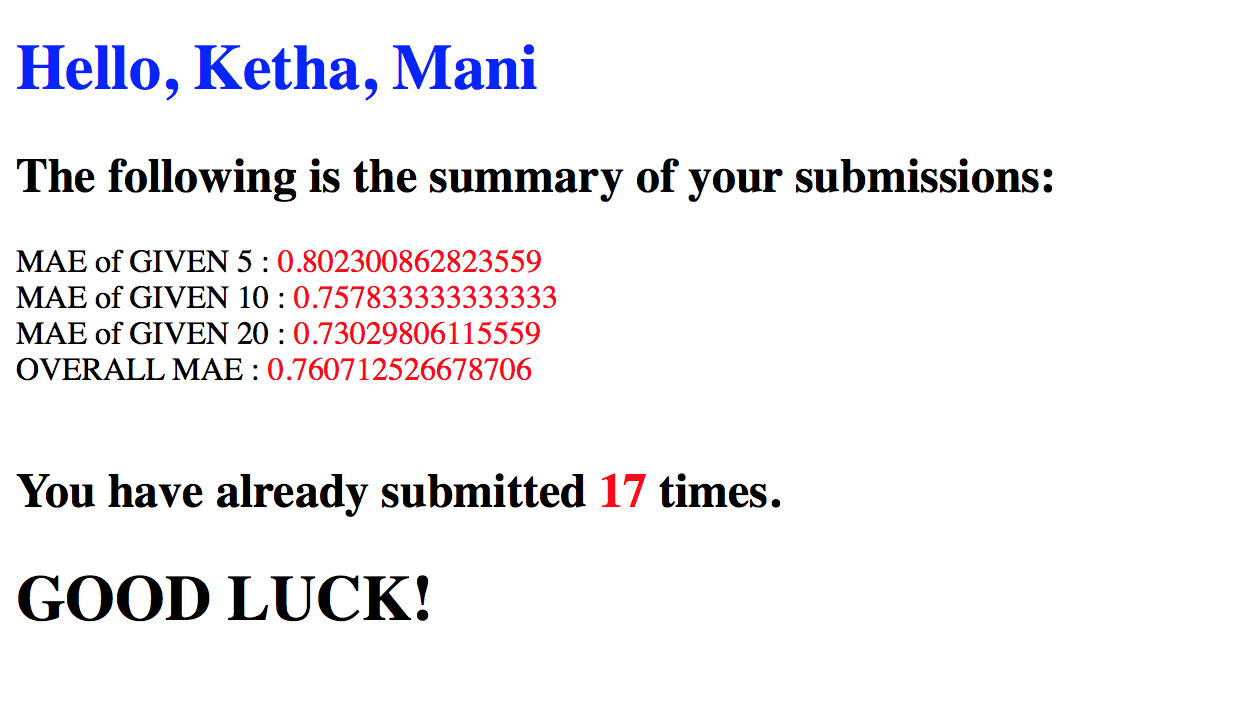
For my own algorithm, I was trying to improve the accuracy of my prediction. Since I was getting the best results through cosine similarity. I decided to try a combined prediction method, that is an ensemble method. In this I used cosine similarity to calculate the similarity value. But for predicting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



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.

1. 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(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

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;

}

}