Abstract

Recommender systems help users making decisions by recommending items of interest to them. We are trying to develop a recommendation system that considers the spatial aspects of the user while recommending. Our work focuses on implementing the concept of Spatial Autocorrelation – similar values cluster together on a map, by using some statistical measures. Primary focus of our work is to tessellate the user's space with respect to location by using voronoi diagram. Recommendations for the new user will be generated by combining their location and the preferences of other users that share those locations.

Introduction

Recommender systems employ information filtering (IF) technique to present information items (movies, music, books, news, images, web pages, etc.) those are likely of interest to the user. Typically, a recommender system compares the user's profile to some reference characteristics, and aims to predict the probable 'rating' that a user would give to an item she/he had not yet been considered. These systems usually builds user profile by collecting user data explicitly (rate an item, rank a collection of items, etc.) or implicitly (item viewing times, recording the items a user purchases online, analyzing users social network, etc.). The recommender system compares the collected data to similar data collected from others and calculates a list of recommended items for the user. Collaborative Filtering (CF) is the most widely used technique for recommendation algorithms. CF tries to identify users that have relevant interests and preferences by calculating similarities among user profiles. The idea behind this method is that, it may be of benefit to one's search for information to consult the behavior of other users who share the same or relevant interests and whose opinion can be trusted.

Objective of the project

The proposed project work will try to develop a recommendation (Movie, Music, e-commerce etc.) system to help users in making decisions by suggesting items of interest to the users. While making a recommendation, the user's location will also be considered as a parameter along with his or her preferences. A relevant piece of information in many location aware applications is the user's current location. Knowledge of the position, when combined with the user preferences, permits efficient service in a location-aware recommendation systems.

For our work, we will consider a dataset which has information about the users and their preferences for items. User's location is represented by cities (Zip-codes). Our primary task is to decompose the users space in such a way that the correlated users (users having similar preferences over items) cluster together around a city. Voronoi diagram will be used for space partitioning. Voronoi diagram partitions the entire space into some smaller cells and recommendations will be provided for the users in those cells. Recommendations will be generated based on both the location and the preferences of the user. Collaborative Filtering method will be used for recommendation purpose. The system will offer suggestions to a new user according to his choices and the preferences of other users who share the same location with the current user or one in nearby locations. Similarity among the users is computed using some similarity metrics. We will use Pearson's correlation coefficient to measure correlation among users.

PRELIMINARIES

Collaborative Filtering

Collaborative Filtering (also known as social information filtering) is based on the basic principle of finding a subset of users who have similar tastes and preferences to that of the active user, and offering recommendations based on that subset of users. That is, given an active user, *u*, compute his n similar users $\{u_1, u_2, \ldots u_n\}$and predict *u*'s preference based on the preferences of $\{u_1, u_2, \ldots u_n\}$. Similar users mean users who share the same kind of tastes and preferences over items.

Collaborative Filtering works based on the following assumptions:

- Users with similar interest have common preferences and vice versa.
- Sufficiently large number of user preferences is available.

Voronoi diagram

In mathematics, a Voronoi diagram is a special kind of decomposition of a metric space determined by distances to a specified discrete set of objects in the space, e.g., by a discrete set of points. It is named after Georgy Voronoi, also called Voronoi tessellation, Voronoi decomposition, or a Dirichlet tessellation (after Lejeune Dirichlet).

In the simplest case, we are given a set of points S in the plane, which are the Voronoi sites. Each site s has a Voronoi cell, also called a Dirichlet cell, V(s) consisting of all points closer to s than to any other site. The segments of the Voronoi diagram are all the points in the plane that are equidistant to the two nearest sites. The Voronoi nodes are the points equidistant to three (or more) sites; basically they are the centers of the circles in which those sites lie.

Imagine you have a map over a city with n cell phone towers. A cell phone always connects to the closest tower, so you'd like to split up the city in zones, where each zone has exactly one cell phone tower and each location inside such a zone is closest to the cell phone tower found in the same zone.

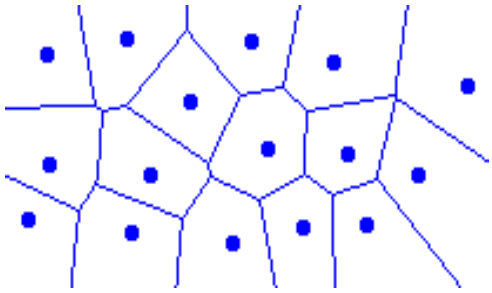


Fig.1. Voronoi Diagram

Pearson correlation coefficient

Consider two users *a* and *b* who rate a set of m items. For example, the ratings of user *a* on item *i* are denoted as $r_{a,i}$. Pearson correlation between the user *a* and user *b* is defined as the ratio of covariance of user *a* and *b* to that of the product of standard deviation of users *a* and *b*. Mathematically,

$$C_{a,b}=\frac{\sum_{i=1}^{m}(r_{a,i}-\hat{r}_a)(r_{b,i}-\hat{r}_b)}{\sqrt{\sum_{i=1}^{m}(r_{a,i}-\hat{r}_a)^2\sum_{i=1}^{m}(r_{b,i}-\hat{r}_b)^2}}$$

Where $r_{i,j}$ is the user *i*'s rating for product j; m is the total number of items or products; and

$$\hat{r}_x=\frac{\sum_{i=1}^{m}r_{x,i}}{m}$$

is the average rating for user 'x' on all the 'm' items.

The correlation coefficient value ranges between -1 and +1. The correlation value of 1 (and -1) is treated as positive (and negative) preferences between users. A correlation value of 0 means that the users have no common set of preferences

Past Works

A lot of work has been done in the area of recommendation engines in the past. Several systems (in the movie domain) have been developed which attempted to recommend movies according to user preferences. Examples of online recommendation systems based on the movie domain include Jinni [4], MovieLens [5], and Netflix [6]. They used collaborative filtering approach to recommend movies. Brunato, Battiti, Villani, and Delai [7] combined the user's current location along with the preferences for providing recommendation services. They developed a location dependent recommender system which is based on a standard web browser. The system recommends a specific URL to a user in a given location that considers where and how often it was accessed by the previous users. Brunato and Battiti [8] used user's position as a relevant piece of information while selecting and ranking links of interest to the users. Their work designed a middleware layer, the location broker that maintains a historic database of location and corresponding links used in the past. Google developed Hotpot [9], a recommendation engine for places, to make local recommendations more personal and relevant, by recommending places based on your ratings and ratings of your friends. It allows you to rate places, to make and invite friends to share those ratings and as you rate sites via Hotpot, the service will recommend other similar places that you might also like. Li. Mi. Zhang and Wu [10] integrated GPS into recommender system to create a location-aware recommender system. They explored the increasing demand of mobile commerce and developed a recommender system for tourism mobile commerce. The system can recommend attractions to the customer with the customer's rating of attractions and customer's sensitivity to location. Yang, Cheng, and Dia [11] proposed a location-aware recommender system for mobile shopping. The system identified the customer's shopping needs and suggested vendors WebPages which includes offers and promotions depending on the location of the customer.

Our Contribution

In this work, we have proposed a spatially-aware recommender system using Voronoi diagrams. And, we have tested our ideas on the MovieLens [5] dataset. MovieLens is a collaborative filtering recommender system developed by GroupLens Research Group. The

dataset contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000.The proposed work will use this dataset to recommend movies to users that are likely to be preferred by them. While recommending, our primary focus will be on the location of the user. We use zip-code (city) to identify the user's location. We try to explore the concept of Spatial Autocorrelation - similar values cluster together on a map, by using some statistical measures. Voronoi diagram is used to tessellate the user's space with respect to location. Space decomposition partitions the entire user's space into smaller regions (polygons) and we will provide recommendation to users in those polygons. Our work tries to find the presence of correlation in the polygons and then recommend movies with a view that the suggested movies will be liked by the user. Collaborative filtering algorithms will be used for recommendation. The system will offer recommendations to a new user according to his choices and the preferences of other userswho share the same location with the current user. Similarity metrics' such as Pearson's correlation coefficient [1] is used to compute the similarly among the users. In this work, we try to recommend items by computing user-user or item-item similarity among the users in the polygons.

Implementation

Let us suppose we have a dataset in movie domain. The dataset has information about the users, movies and the ratings given by the users to the movies. The user data file has information about the location of the user. The locations are represented by Zip-codes (Cities). We need to create a voronoi diagram using these Zip-codes. We will consider only those zip-codes that have a minimum number of users with a minimum number of ratings per user. We can use the following approaches:

- Directly represent the zip-codes as some coordinates and construct the Voronoi diagram.
- Develop a procedure that converts the zip-codes into equivalent coordinates by computing the centroid of the polygon represented by the zip-code. Construct the Voronoi diagram.

- Map other cities to the Voronoi diagram

Each of the Voronoi cells will be considered for measuring the correlation among the users in the cell. We will use Pearson's correlation coefficient.

Dataset Used:

User data file has the following structure:

user id | age | gender | occupation | zip code

Data sample:

```
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
6|42|M|executive|98101
7|57|M|administrator|91344
8|36|M|administrator|05201
9|29|M|student|01002
10|53|M|lawyer|90703
11|39|F|other|30329
12|28|F|other|06405
13|47|M|educator|29206
14|45|M|scientist|55106
15|49|F|educator|97301
16|21|M|entertainment|10309
```

The user ids are the ones used in the Ratings file.

From the above file we consider only those zip codes (cities) which have a minimum number of users (suppose 20). We need to represent these zip codes as some coordinates and the voronoi diagram will be constructed using those zip codes. After constructing the voronoi diagram, we will map all other cities into the voronoi diagram.

Movie data file has the following structure:

movie id `::` movie title (release date) `::` `genre1` | `genre2` | `genre3`

Data sample:

```
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama
```

Ratings data file has the following structure:

user id | movie id | rating

Each user has rated more than one movie. Ratings are in the integer scale of 1 - 5. The data is randomly ordered.

Data sample:

```
196    242    3
186    302    3
22     377    1
244    51     2
166    346    1
298    474    4
115    265    2
253    465    5
305    451    3
6      86     3
62     257    2
```

Decomposition Algorithm:

In this work, we have used Voronoi diagram based approach for space decomposition. Space partitioning is done on the basis of zip-codes (cities). User data file in the MovieLens dataset has information about the user and their locations. Our work tries to decompose this user space. Location is represented by zip-codes. We construct the Voronoi diagram using these zip-codes. Our algorithm finds some zip codes that have a minimum number of users (threshold). These zip-codes become the Voronoi sites. Each site S has a Voronoi cell V(S) consisting of all the points closer to S than any other site. To construct the Voronoi diagram, zip-codes should be represented by some coordinates. Our work considers the longitude-latitude of the centroids of the polygonal regions representing the zip-codes as the coordinates. The algorithm considers the distance of a point (zip-code) from each of the site points and the point is allocated to the region represented by the site that has the minimum distance to the point. Continuing this way we map each point onto some Voronoi cell. The output of the algorithm is the set of Voronoi polygons of user zip-codes. Partitioning algorithm decomposes the entire user space into some smaller polygons.

We next try to find out the correlation coefficient values for different pairs of users lying in each of the polygons, using Pearson's correlation coefficient formula (described above). The algorithm uses the rating file of dataset, to find out the different ratings given by the users lying in each of these polygons to the different movies. And based on the common set of movies rated by any pair of users, this algorithm finds out correlation coefficient for that pair of users.

A. Algorithm Voronoi_Decomposition

Step1: Represent zip-code centroids as coordinates (longitude-latitude). Let this set be denoted by P.

Step2: Select those zip-codes (sites) from user space that satisfies the threshold value. Let this subset of P be denoted by T.

Step3: Create the Voronoi diagram with the sites from T.

Step4: Locate each zip-code (centroid) c in P-T in the Voronoi diagram constructed in Step 3.

Step5: Allocate the zip-code centroid c to the appropriate Voronoi polygon (represented by the site in P closest to c).

B. Algorithm Find_Correlation

Step1: Choose one polygon from the Voronoi polygons.

Step 2: Find the ratings on different movies, given by all the users within this polygon.

Step 3: For a pair of user, first find common set of rated movies and apply Pearson's correlation coefficient formula, to calculate correlation coefficient value for that pair of user.

Step 4: Repeat step 3 for the rest of the users.

Step 5: Repeat step 2-3 for other polygons also.

The output of the Voronoi_Decomposition algorithm is the set of Voronoi polygons represented by their centroids (sites) and a list of centroids located in each such polygon. Each and every city (zip-code) within a polygon is closer to its centroid than the centroid of any other polygon. The output of the Find_Correlation algorithm is the set of the correlation

coefficient values for different pairs of users within those polygons, which determine the presence or absence of correlation among the users within those polygons.

Recommendation Algorithm

Once the Voronoi diagram has been constructed, polygons defined, and correlation among users measured, we can start our recommendation algorithm. Recommendations can be provided to a naïve user or an existing user. For a new user we identify the location (zip-code) and accordingly map the user to its destined Voronoi polygon. Collaborative filtering technique is used to recommend items of interest to the user. As per some standard experimental calculations (like Geary's index) and results, it is well established that spatial correlation exists in the polygons, and hence it seems very promising that if we recommend items based on the preferences of the users who share the same neighbourhood with the current active user, quality of recommendation will increase. Similarity in the user's preferences in a polygon is measured by Pearson's correlation coefficient [1]. During pre-processing, we calculate Pearson's coefficient value for every pair of users in the polygon. The algorithm is briefly described below:

A. Algorithm Recommend_Movies

Step 1: Select a user (*say active user*) for the recommendation, and identify the location (zip-code) of that user.

Step 2: Map the user in the Voronoi polygon according to his/her location.

Step 3: Find a set U of users from the same voronoi polygon (found in step-2), those are highly correlated with the active user, means having correlation coefficient values greater than a threshold value (say 0.6).

Step 4: Find the set M_a of highly rated movies of active user (say of having rating 4 or 5 out of 5 scale), and find out Top-2 categories of genres in which most of the movies of this set lie.

Step 5: Also find out the set M_u of movies highly rated by rest of the users of set U.

Step 6: Filter out the subset of movies M_{rec} from set M_u, which are falling in both the Top-2 genres (calculated in step 4) as well as not yet seen by the active user.

Step 7: Recommend this set of movies M_{rec} to the active user.

Experiments And Results

To validate our scheme, we tested our proposed algorithms on the MovieLens dataset. The dataset has a User database with demographic information of the users. It has a Movie database with detailed information about movies. The dataset also includes a Ratings database having the ratings given to different movies by the users. We use this dataset as it has enough information that we need to implement our algorithm. The user location information in the database is represented by their corresponding zip-codes (city). This location data becomes the basis of the Voronoi_Decomposition algorithm. We have tested the algorithm with three threshold values, (5, 10, 15), which indicates the minimum number of users that a zip-code must have to be considered as a site in Voronoi diagram. Remaining cities will be mapped to its corresponding site by the algorithm. Table I shows a sample result of the experiment.

TABLE I. RESULT OF SPACE DECOMPOSITION

Threshold Value	Number of Polygons
15	10
10	35
5	175

As the threshold value increases, number of polygon decreases. We next test our Find_Correlation algorithm for all the polygons found by the decomposition algorithm. We defined a set of ranges for the correlation values and counted the number of pairs of users that's fall in a particular range. Table II depict a sample result of the correlation algorithm.

TABLE II. CORRELATION RESULT FOR POLYGONS WITH THRESHOLD = 15

(Correlation Range) ⟶ Polygon no	**[-1–0)**	**[0–.25)**	**[.25–.6)**	**[.6–1]**	**% above 0.6**
1	1040	895	926	359	11.15
2	61776	69033	64189	23605	10.79
3	127542	141086	135821	50832	11.16
4	107155	117974	110970	40190	10.68
5	57872	67631	63365	20608	09.83
6	8877	9060	8385	2900	09.92
7	64962	65866	63088	24980	11.41
8	19528	19663	17262	7194	11.30
9	2283	2305	1912	746	10.29
10	26469	28818	26900	9500	10.36

At last, we tested our proposed recommendation algorithm 'Recommend_Movies', to find out how fine, our system is actually recommending. The idea behind this testing procedure is as follows:

A: Testing Algorithm

Step 1: Choose a particular user u_i (i.e. Active user), and find out the set of movies M_u (*refer Algorithm Recommend_Movies)* for this user.

Step 2: Filter out the subset $M_{(R,a)}$ of movies from this set, which the active user has already seen and rated. And Calculate average of the ratings, given to all the movies of this subset. [Say Avg_R].

Step 3: Also Calculate average of the ratings given by active user to all the movies,he has seen and rated so far.[Say Avg_{all}].

Step 4: Find out the difference between these two above average ratings. [Say diff(i), where diff(i)= Avg_R - Avg_{all}].

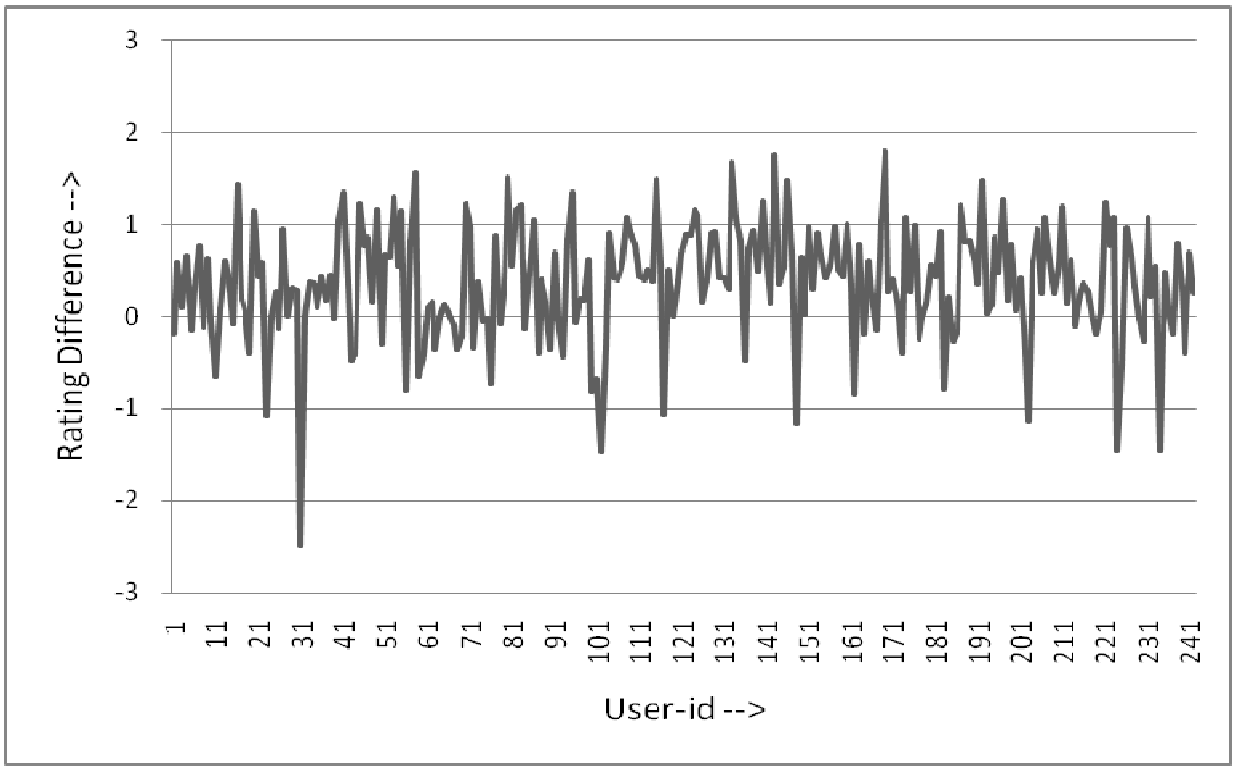
Step 5: Repeat Step 2-3 for N number of users and store all these differences in a table.

Step 6: From this table of rating differences for different users, calculate the (i) average of absolute of all the differences [Avg_u] (ii) the standard deviation [SD_u] for all these difference values (iii) Number of users having positive value of diff(i) [Pos_count_u] (iv) Number of users having negative value of diff(i) [Neg_count_u].

Test Cases:

1. For (≈ 250) Users

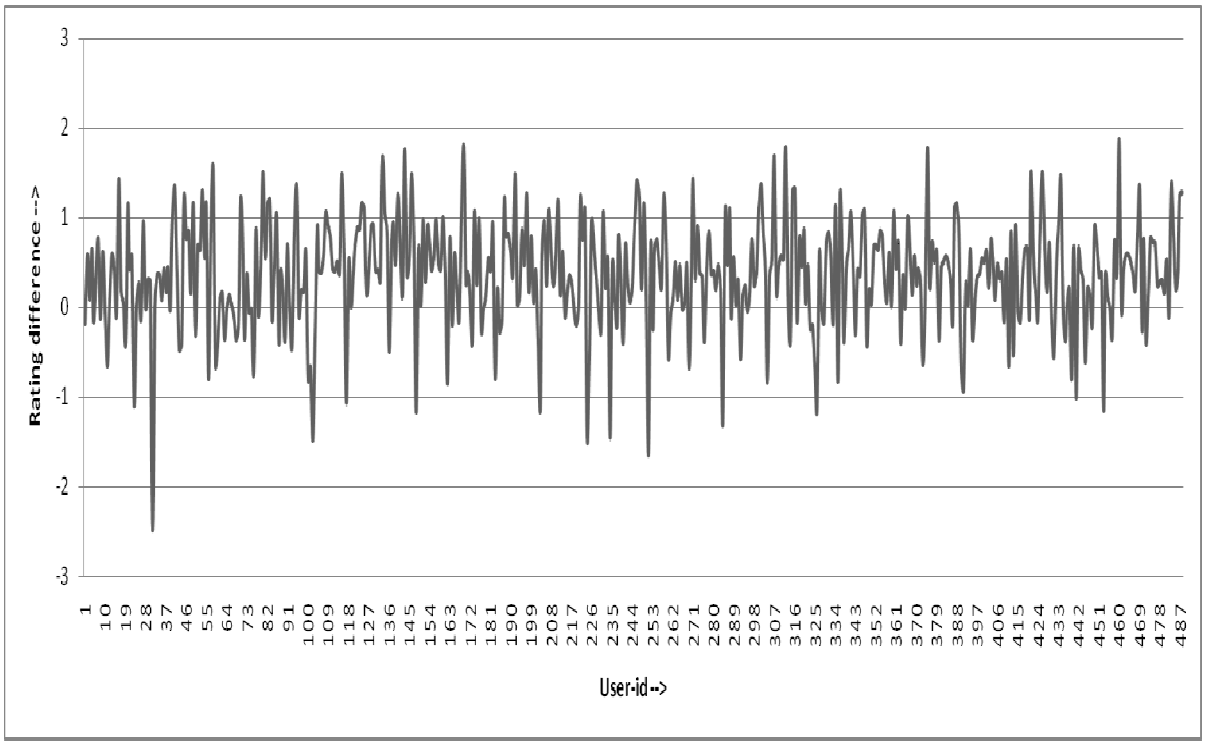
GRAPH I. RATING DIFFERENCE [diff (i)] VALUES FOR 242 (≈ 250) USERS.



RESULTS: [A] Pos_count_u = 181, Neg_count_u = 61 [B] Avg_u = 0.35702798. [C] SD_u = 0.63110024.

2. For (≈ 500) Users

GRAPH II. RATING DIFFERENCE [diff (i)] VALUES FOR 488 (≈ 500) USERS.



RESULTS: [A] Pos_count_u = 373, Neg_count_u = 115 [B] Avg_u = 0.3604241. [C] SD_u = 0.60164124.

From these results, we can infer that:

(i) $(Pos\_count_u)/(Neg\_count_u) \approx 3 : 1$, so out of every four users, three users are being recommended relatively better movies by our algorithm, than they have already seen and rated.

(ii) Since $Avg_u \approx 0.3$ and $SD_u \approx 0.6$, so although the one user out of four, which are not being recommended better movies, Still the average rating of those recommended set of movies(which are not better) differ from the average rating on all the movies he has seen so far, just by $[0.3 \pm 0.6]$.

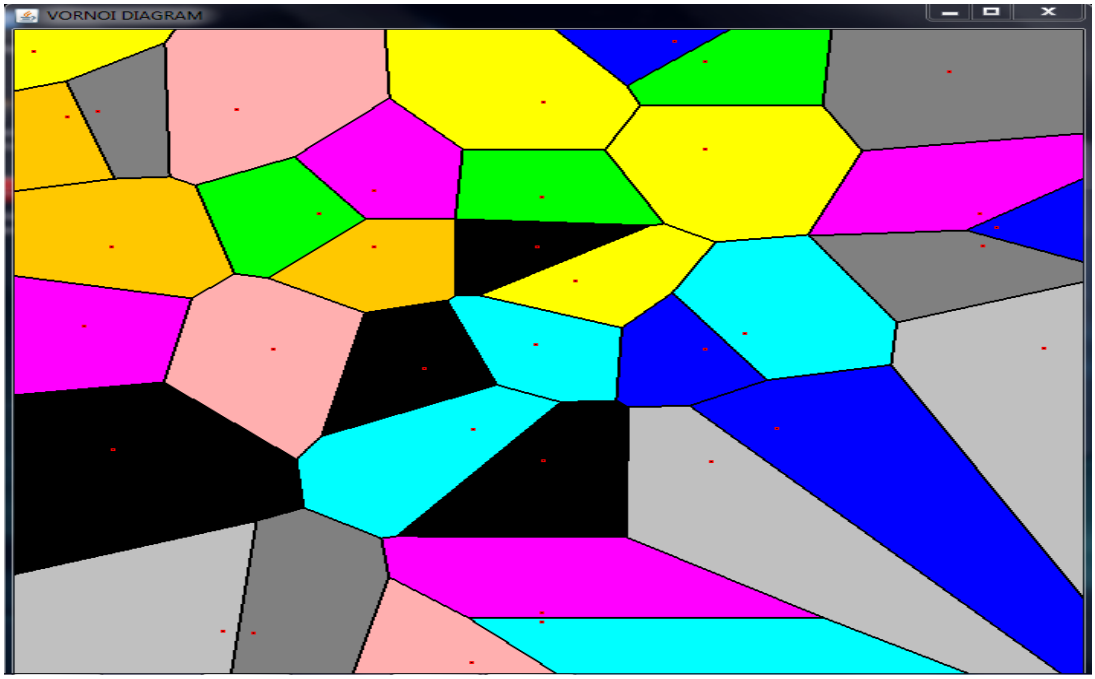
Project Asset : An GUI Application for constructing voronoi diagram

Along with our working on different datasets, we have also made an GUI application in java for constructing voronoi diagram, from a given list of voronoi sites. Here we have used an algorithm based on the concept of "**Raster Scan**" method . Each of the pixels on the drawing interface was scanned one by one, the distances of this pixel from all the given site points was calculated and based on the minimum distance out of them, the corresponding voronoi site was marked to them. Based on these marks, each of the pixels was assigned a particular color, so that all the pixels within a voronoi cell will have the same color. And the border of each voronoi cell was marked with black color, to separate two adjacent voronoi cells.

Input: A "coordinates.txt" file, containing zipcode | x-coord | y-coord

Output: A graphical user interface showing voronoi diagram made out of these coordinates

Screenshot:



PROGRAM CODE:

1.Readfile.java:-

```
import java.io.*;
import java.util.*;

// Coordinates format
class CoordRead
{
	private long x;
	private long y;
	public CoordRead()
	{
		this(0,0);
	}
	public CoordRead(long x_coord,long y_coord)
	{
		setx(x_coord);
		sety(y_coord);
	}
	public void setx(long x_coord)
	{
		x=x_coord;
	}
	public int getx()
	{
		return (int)x;
	}
	public void sety(long y_coord)
	{
		y=y_coord;
	}
	public int gety()
	{
		return (int)y;
	}

}
// Reading Text file
public class Readfile
{
	private Scanner input;
	public  int n=0;
	public  int coords[][]=new int[100][2];
	public void openFile()
	{
		try
		{
			input = new Scanner(new File("coordinates.txt"));
		}
		catch(FileNotFoundException fileNotFoundException)
		{
			System.err.println("Error Opening File.");
			System.exit(1);
		}
	}
	public  void readRecords()
	{
		CoordRead record=new CoordRead();
		int zip_id;
		try
		{
			while(input.hasNext())
			{
```

```
                        n++;
                        zip_id=input.nextInt();
                        record.setx(input.nextLong());
                        record.sety(input.nextLong());
                        coords[n-1][0]=record.getx();
                        coords[n-1][1]=record.gety();
                }
        }
        catch(NoSuchElementException elementException)
        {
                System.err.println("File Improperly Formed.");
                input.close();
                System.exit(1);
        }
        catch(IllegalStateException stateException)
        {
                System.err.println("Error reading fromfile.");
                System.exit(1);
        }

    }
    public void closeFile()
    {
            if(input!=null)
                input.close();
    }
}
```

2.voronoi.java :

```
import java.io.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.util.*;

public class voronoi extends JPanel
{
    private BufferedImage canvas;
    // Constructor
    public voronoi(int width, int height,Readfile obj)
    {
      int n=obj.n;
      int coords[][]=new int[100][2];
      coords=obj.coords;
      int col[] = new int[4];
      int fk;
      canvas=new BufferedImage(width,height,
      BufferedImage.TYPE_INT_ARGB);
      fillCanvas(Color.white);
      // For coloring each pixels...
      for(int i=0;i<width;i++)
           for(int j=0;j<height;j++)
           {
                int k=min_dist(i,j,obj); // k is the index of that                site
point, from where the dist of coord(i,j) is                          minimum.
                int kle=min_dist(i-1,j,obj);
                int kri=min_dist(i+1,j,obj);
                int kup=min_dist(i,j+1,obj);
                int klo=min_dist(i,j-1,obj);
                int c=0;
                if(k!=kle)
                {
                        col[c] = kle;
                        c++;
                }
                if(k!=kri)
```

```
        {
                col[c] = kri;
                c++;
        }
        if(k!=kup)
        {
                col[c] = kup;
                c++;
        }
        if(k!=klo)
        {
                col[c] = klo;
                c++;
        }
        if(k!=kle || k!=kri || k!=klo || k!=kup)
        {
                //k=rand(col,4);
                k=1;
        }
        if(k>9)
        {
                k=k%10;
        }
        switch(k)
        {
                case 0:
                        canvas.setRGB(i,j, Color.cyan.getRGB());
                        break;
                case 1:
                        canvas.setRGB(i,j, Color.black.getRGB());
                        break;
                case 2:
                        canvas.setRGB(i,j, Color.blue.getRGB());
                        break;
                case 3:
                     canvas.setRGB(i,j,
        Color.magenta.getRGB());
                        break;
                case 4:
                        canvas.setRGB(i,j, Color.pink.getRGB());
                        break;
                case 5:
                        canvas.setRGB(i,j, Color.gray.getRGB());
                        break;
                case 6:
                        canvas.setRGB(i,j, Color.green.getRGB());
                        break;
                case 7:
                        canvas.setRGB(i,j,Color.lightGray
        .getRGB());
                        break;
                case 8:
                        canvas.setRGB(i,j, Color.orange.getRGB());
                        break;
                case 9:
                        canvas.setRGB(i,j, Color.yellow.getRGB());
                        break;
        }

}
// For coloring SITE POINTS ...
for(int i=0;i<n;i++)
{
        int x=coords[i][0];
        int y=coords[i][1];

        for(int m=x-1;m<x+2;m++)
```

```
                                for(int p=y-1;p<y+2;p++)
                                {
                                        if(m==x&&p==y)
                                                canvas.setRGB(m,p,
                                        Color.black.getRGB());
                                        else
                                                canvas.setRGB(m,p,
                                        Color.red.getRGB());
                                }
                        }

                }

                // Methods

            public Dimension getPreferredSize()
            {
                return new Dimension(canvas.getWidth(), canvas.getHeight());
            }
            public void paintComponent(Graphics g)
            {
                super.paintComponent(g);
                Graphics2D g2 = (Graphics2D) g;
                g2.drawImage(canvas, null, null);
            }
                public static int min_dist(int x,int y,Readfile obj1)
                 // Returns the index of site point(that row of array)              from which distance
of coord(x,y) is minimum , otherwise          if  distance  from  two  sites  point  are  equal  it
will return -1
                {

                        int min_pos=0,count=0;
                        double min=Math.sqrt(((x-obj1.coords[0][0])*
                    (x-obj1.coords[0][0]))+((y-obj1.coords[0][1])*(y-
obj1.coords[0][1])));

                        for(int i=1;i<obj1.n;i++)
                        {
                                double dist=Math.sqrt(((x-obj1.coords[i][0])*
                             (x-obj1.coords[i][0]))+((y-obj1.coords[i][1])*(y-
                obj1.coords[i][1])));

                                if(dist<min)
                                {
                                        min=dist;
                                        min_pos=i;
                                }

                        }
                        return min_pos;
                }


                public void fillCanvas(Color c)
             {
                int color = c.getRGB();
                for (int x = 0; x <canvas.getWidth(); x++)
                  {
                     for (int y = 0; y <canvas.getHeight(); y++)
                         {
                         canvas.setRGB(x, y, color);
                     }
                }
                repaint();
             }
                public static void main(String args[])
                {
```

```
            Readfile Coordfile=new Readfile();
            Coordfile.openFile();
            Coordfile.readRecords();
            Coordfile.closeFile();
            int width = 1200;
        int height = 700;

            voronoi panel = new voronoi(width, height,Coordfile);

            System.out.println("Site Point Coordinates are :");
                    for(int i=0;i<Coordfile.n;i++)
                    {
                            System.out.println(Coordfile.coords[i][0]+"
            "+Coordfile.coords[i][1]);
                    }
                    System.out.println("Total No. of Site Points is :
    "+Coordfile.n);

                    JFrame frame = new JFrame("VORONOI DIAGRAM - A GUI
    APPLICATION");
            frame.add(panel);
            frame.pack();
            frame.setVisible(true);
                    frame.setResizable(true);
                    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }
}
```

PROGRAM CODES FOR IMPLEMENTING RECOMMENDER SYSTEM.

1. 'Find_sites.java'

- This program find out all the zip codes which have minimum of 15 users from file "users.dat" and keep all these zip-codes in a separate file "zip_cen.dat".
Input: 'users.dat' file

Output: 'zip_cen.dat' and 'all_zips.dat'

Code:

```
import java.io.*;
import java.util.*;
class Find_sites
{
	public static void main(String args[])
	{
		String s;
		String zips= null;
		String zip_sites[]=new String[3600];
		int zipcount[]=new int[3600];
		int id,age,deg,flag,k=0;

		// Reading input from file "users.dat" and storing only "zipcode" part into array zip_sites[]and their corresponding counts(i.e. no. of users) into array "zipcount[]".

		try
		{
			Scanner in = new Scanner(new FileReader("users.dat"));
			while(in.hasNext())
			{

				id=in.nextInt();
				s=in.next();
				age=in.nextInt();
				deg=in.nextInt();
				zips=in.next();


				flag=0;
				for(int j = 0;j <k; j++)  // k denotes no. of zipcodes currently stored in the array "zip_sites[]".
				{
					if(zips.compareTo(zip_sites[j])==0)   // Checking If new zip code read from file, exist in array "zip_sites[]" or not.
					{
						zipcount[j]++;
						flag=1;
					}
				}
				if(flag==0)
				{

					zip_sites[k]=zips;
					zipcount[k]++;
					k++;
				}
			}
			in.close();
```

```
		}
		catch(FileNotFoundException fileNotFoundException)
		{
			System.err.println("Error Opening File.");
			System.exit(1);
		}

	//	Writing	All	distinct	Zip-Codes	into	file	"all_zips.dat"	and	Vornoi	Site
Points(centers-having >15 users)	into file "zip_cen.dat"
		try
		{
			PrintWriter out1 = new PrintWriter("zip_cen.dat");
			PrintWriter out2 = new PrintWriter("all_zips.dat");
			int centre_count=0;
			for(int i=0;i<k;i++)
			{
				if(zipcount[i]>=15)
					centre_count++;
			}
			out1.println(centre_count); // for writing no. of zip centers at the top
of "zip_cen.dat".
			out2.println(k); // for writing total no. of distinct zip codes at the
top of "all_zips.dat".
			for(int i=0;i<k;i++)
			{
				out2.println(zip_sites[i]);
				if(zipcount[i]>=15)
					out1.println(zip_sites[i]+" "+zipcount[i]);
			}
			out1.close();
			out2.close();
		}
		catch(FileNotFoundException fileNotFoundException)
		{
			System.err.println("Error Opening File.");
			System.exit(1);
		}
	}
}
```

2. 'Find_zipcen_coords.java'

-This program finds out the longitude and latitudes of all the zipcodes present in "zip_cen.dat" and "allzips.dat" and store these in "zip_cen_coordinates.dat" and "zip_coordinates.dat" respectively.

Input: 'zip_cen.dat' or 'allzips.dat' file and 'zips_sm.txt'.

Output: 'zip_cen_coordinates.dat' or 'zip_coordinates.dat'

Code:

```
import java.io.*;
import java.util.*;
class Find_zipcen_coords
{
	public static void main(String args[])
	{

		try
		{
```

```
// CREATING "zip_cen_coordinates.dat" file..
int freq;
Scanner in1 = new Scanner(new FileReader("zip_cen.dat"));
int size=in1.nextInt();
String zipcen[]=new String [size];

for (int i=0;i<size;i++)
{
	zipcen[i]=in1.next();
	freq=in1.nextInt();
}
in1.close();

Scanner in2 = new Scanner(new FileReader("zips_sm.txt"));
PrintWriter out = new PrintWriter("zip_cen_coordinates.dat");
int a;
String zips=null;
String b=null;
String c=null;
double longt;
double latt;
String d=null;
String e=null;
out.println(size);
while(in2.hasNext())
{
	a=in2.nextInt();
	zips=in2.next();
	b=in2.next();
	c=in2.next();
	longt=in2.nextDouble();
	latt=in2.nextDouble();
	d=in2.next();
	e=in2.next();
	for(int i=0;i<size;i++)
	{
		if(zips.compareTo(zipcen[i])==0)
		{
			out.println(zips+" "+longt+" "+latt);
		}
	}
}

in2.close();
out.close();


// CREATING "zip_coordinates.dat" file..
Scanner in3 = new Scanner(new FileReader("all_zips.dat"));
int size1=in3.nextInt();
String zip1[]=new String [size1];

for (int i=0;i<size1;i++)
{
	zip1[i]=in3.next();
}
in3.close();

Scanner in4 = new Scanner(new FileReader("zips_sm.txt"));
PrintWriter out1 = new PrintWriter("zip_coordinates.dat");
int a1;
String zips1=null;
String b1=null;
String c1=null;
double longt1;
double latt1;
String d1=null;
```

```
                        String e1=null;
                        while(in4.hasNext())
                        {
                                a1=in4.nextInt();
                                zips1=in4.next();
                                b1=in4.next();
                                c1=in4.next();
                                longt1=in4.nextDouble();
                                latt1=in4.nextDouble();
                                d1=in4.next();
                                e1=in4.next();
                                for(int i=0;i<size1;i++)
                                {
                                        if(zips1.compareTo(zip1[i])==0)
                                        {
                                                out1.println(zips1+" "+longt1+" "+latt1);
                                        }
                                }
                        }

                        in4.close();
                        out1.close();

                }
                catch(FileNotFoundException fileNotFoundException)
                {
                        System.err.println("Error Opening File.");
                        System.exit(1);
                }
        }
}
```

3. 'Find_zip_voronoi.java'

-This program finds out the zip centres corresponding to each of the individual zipcodes, and save these records to "voronoi_zip_coordinates.dat".

Input: 'zip_cen_coordinates.dat' and 'zip_coordinates.dat'.
Output: 'voronoi_zip_coordinates.dat'

Code:

```
import java.io.*;
import java.util.*;
class Find_zip_voronoi
{
        public static void main(String args[])
        {

                try
                {

                        Scanner in1 = new Scanner(new FileReader("zip_cen_coordinates.dat"));
                        int size=in1.nextInt();
                        String zipid[]= new String [size];
                        double coord_arr[][]=new double[size][2];
                        int i=0;
                        while(in1.hasNext())
```

```
                        {

                                zipid[i]=in1.next();
                                coord_arr[i][0]=in1.nextFloat();
                                coord_arr[i][1]=in1.nextFloat();
                                i++;
                        }

                        Scanner in2 = new Scanner(new FileReader("zip_coordinates.dat"));
                        PrintWriter out = new PrintWriter("voronoi_zip_coordinates.dat");
                        String zip;
                        double longtx,latty;
                        while(in2.hasNext())
                        {
                                zip=in2.next();
                                longtx=in2.nextDouble();
                                latty=in2.nextDouble();

                                int min_pos=0;
                                double           min=Math.sqrt(((longtx-coord_arr[0][0])*(longtx-
coord_arr[0][0]))+((latty-coord_arr[0][1])*(latty-coord_arr[0][1])));
                                for(int j=1;j<size;j++)
                                {
                                        double   dist=Math.sqrt(((longtx-coord_arr[j][0])*(longtx-
coord_arr[j][0]))+((latty-coord_arr[j][1])*(latty-coord_arr[j][1])));

                                        if(dist<min)
                                        {
                                                min=dist;
                                                min_pos=j;
                                        }

                                }

                                out.println(zip+" "+zipid[min_pos]);
                        }
                        in1.close();
                        in2.close();
                        out.close();
                }
                catch(FileNotFoundException fileNotFoundException)
                {
                        System.err.println("Error Opening File.");
                        System.exit(1);
                }
        }
}
```

4. 'zipsite_users.java'

-This program finds out all the users with their zipcodes lying inside each of the polygons of the voronoi diagram.

Input: 'voronoi_zip_coordinates.dat' and 'users.dat' files.

Output: A directory 'zipsite_users_N' containing files 'zipsiteN.dat' where N is the no. of polygons .

Code:

```
import java.io.*;
import java.util.*;
class Find_zipsite_users
{
        public static void main(String args[])
        {
                try
```

```
{
	Scanner in0 = new Scanner(new FileReader("zip_cen.dat"));
	int zip_count=in0.nextInt();
	String given_zipcen[]=new String[zip_count];
	int i1=0;
	int user_count;
	while(in0.hasNext())
	{
		given_zipcen[i1]=in0.next();
		user_count=in0.nextInt();
		i1++;
	}

	//Making a new Directory "zipsite_users_N" inside current working directory.
	 String newDir = "zipsite_users_N";
	  File f = new File(newDir);
		f.mkdir();
	 // Getting path of current working directory
	String currentdir = System.getProperty("user.dir");
	System.out.println("Current Working Directory : "+ currentdir);

	for(int k=0;k<zip_count;k++)
	{
				Scanner in = new Scanner(new FileReader("voronoi_zip_coordinates.dat"));
				String zips;
				String zipcen;
				int count=0;
				while(in.hasNext())
				{
					zips=in.next();
					zipcen=in.next();
					if(given_zipcen[k].compareTo(zipcen)==0)
					{
						count++;
					}
				}
				in.close();

				String ziparr[] = new String[count];
				Scanner in1 = new Scanner(new FileReader("voronoi_zip_coordinates.dat"));
				int i=0;
				while(in1.hasNext())
				{
					zips=in1.next();
					zipcen=in1.next();
					if(given_zipcen[k].compareTo(zipcen)==0)
					{
						ziparr[i]=zips;
						i++;
					}
				}

				in1.close();
				String s=null;
				String zipread=null;
				int id,age,deg;
				Scanner in2 = new Scanner(new FileReader("users.dat"));

				// Appending the paths,resulting into the absolute path for output files..i.e. inside zipsite_users_N directory.
				String outpath=currentdir+"\\zipsite_users_N\\zipsite"+Integer.toString(k)+".dat";
				PrintWriter out = new PrintWriter(outpath);
				out.println(given_zipcen[k]);
```

```
                                        while(in2.hasNext())
                                        {

                                                id=in2.nextInt();
                                                s=in2.next();
                                                age=in2.nextInt();
                                                deg=in2.nextInt();
                                                zipread=in2.next();
                                                for(int j=0;j<ziparr.length;j++)
                                                {
                                                        String st=ziparr[j];
                                                        if(st.compareTo(zipread)==0)
                                                        {
                                                                out.println(zipread+" "+id);
                                                        }
                                                }

                                        }
                                        in2.close();
                                        out.close();
                        }
                }
                catch(FileNotFoundException fileNotFoundException)
                {
                        System.err.println("Error Opening File.");
                        System.exit(1);
                }
        }

}
```

5. 'Find_zipcen_ratings.java'

-This program finds out the ratings on different movies, given by all the users lying inside the voronoi cell of a particular zip_centre, and stores them in file 'zipsite_ratingsN.dat'.

Input: 'zipsiteN.dat' and 'ratings.dat' file.
Output: A directory named 'zipsite_users_ratings_N' containing 'zipsite_ratingsN.dat' where N is the no. of polygons.

Code:

```
import java.io.*;
import java.util.*;
class Find_zipcen_ratings
{
        public static void main(String args[])
        {
                try
                {
                        String currentdir = System.getProperty("user.dir"); // Getting path of
current working directory.
                        String inpath = currentdir+"\\zipsite_users_N";
                        File InpFolder = new File(inpath); // Getting into zip_users_N directory.
                        File Inpfiles[];
                        Inpfiles = InpFolder.listFiles(); // Listing all files inside zip_users_N
directory.

                        // Retrieving all rating.dat file datas into memory(i.e. array) for
faster computation
                        int rating_arr[][]=new int[1000300][3];
                        Scanner in2 = new Scanner(new FileReader("ratings.dat"));
```

```
                        int i=0;
                        String timestamp;
                                while(in2.hasNext())
                                {
                                        rating_arr[i][0]=in2.nextInt();
                                        rating_arr[i][1]=in2.nextInt();
                                        rating_arr[i][2]=in2.nextInt();
                                        timestamp=in2.next();
                                        i++;
                                }
                        in2.close();

                        //Making a new Directory "zipsite_users_ratings_N" inside current working
directory.
                        String newDir = "zipsite_users_ratings_N";
                        File f = new File(newDir);
                        f.mkdir();
                        System.out.println("Current Working Directory : "+ currentdir);

                        //   Naming   all   the   required   output   files   to   be   made   in
"zipsite_users_ratings_N" directory.
                //      String                  outfiles[]                  =                  {
"zipsite_ratings1.dat","zipsite_ratings2.dat","zipsite_ratings3.dat","zipsite_ratings4.dat","zi
psite_ratings5.dat","zipsite_ratings6.dat","zipsite_ratings7.dat","zipsite_ratings8.dat","zipsi
te_ratings9.dat","zipsite_ratings10.dat"};

                        for(int k = 0;k<Inpfiles.length; k++)
                        {
                                Scanner in1 = new Scanner(new FileReader(Inpfiles[k]));
                                String zipcen=in1.next();
                                String
outpath=currentdir+"\\zipsite_users_ratings_N\\zipsite_rating"+Integer.toString(k)+".dat";
                        //      String
outpath=currentdir+"\\zipsite_users_ratings_N\\"+outfiles[k];
                                PrintWriter out = new PrintWriter(outpath);
                                out.println(zipcen);
                                String zip;
                                int userid,r_uid,r_mid,rating;

                                while(in1.hasNext())
                                {
                                        zip=in1.next();
                                        userid=in1.nextInt();
                                        int j=0;
                                        while(j!=i)
                                        {
                                                if(userid==rating_arr[j][0])
                                                {
                                                        out.println(rating_arr[j][0]+"
"+rating_arr[j][1]+" "+rating_arr[j][2]);
                                                }
                                                j++;
                                        }
                                }
                                in1.close();
                                out.close();
                        }
                }catch(FileNotFoundException fileNotFoundException)
                        {
                                System.err.println("Error Opening File.");
                                System.exit(1);
                        }
        }

}
```

6. 'Find_Correlation.java'

-This program finds out the correlation coefficient for every pairs of users in each voronoi cells using pearson correlation coefficient formula. It calculates user-user correlation between users.

Input: 'zipsiteN.dat' and 'zipsite_ratings_N' files.
Output: A directory named 'zipsite_users_Correlation_N' containing 'correlationN.dat' files where N is the number of polygons and 'count_range.dat' file(which counts the number of pairs of users in different correlation range in each of the polygons).

Code:

```
import java.io.*;
import java.util.*;
class Find_correlation
{
	static int user_count_in_diff_range[][];
	public static int get_index(String arr[],String val)
	{
		int index;
		for(index=0;index<arr.length;index++)
		{
			if(val.compareTo(arr[index])==0)
				break;
		}
		return index;
	}
	public static void main(String args[])
	{
		try
		{
			String currentdir = System.getProperty("user.dir"); // Getting path of
current working directory.
			String newDir = "zipsite_users_Correlation_N";
			File f = new File(newDir);
			f.mkdir();
			// Making distinct User's Arrray userid[].

			String inpath = currentdir+"\\zipsite_users_N";
			File InpFolder = new File(inpath); // Getting into zip_users_N directory.
			File Inpfiles_users[];
			Inpfiles_users = InpFolder.listFiles(); // Listing all files inside
zip_users_N directory.

			String inpath2 = currentdir+"\\zipsite_users_ratings_N";
			File InpFolder2 = new File(inpath2); // Getting into zip_users_ratings_N
directory.
			File Inpfiles_ratings[];
			Inpfiles_ratings = InpFolder2.listFiles(); // Listing all files inside
zip_users_ratings_N directory.
			user_count_in_diff_range = new int [Inpfiles_users.length][4];
			for(int k1 = 0;k1<Inpfiles_users.length; k1++)
			{
				Scanner in_user0 = new Scanner(new
FileReader(Inpfiles_users[k1]));
				String zipcen=in_user0.next();
				int user_count=0;
				String zips,user;
				while(in_user0.hasNext())
				{
					zips=in_user0.next();
					in_user0.next();
```

```
                        user_count++;
                    }
                    in_user0.close();
                    String userid[]= new String[user_count];
                    Scanner         in_user1         =         new         Scanner(new
FileReader(Inpfiles_users[k1]));
                    zipcen=in_user1.next();
                    int i=0;
                    while(in_user1.hasNext())
                    {
                        zips=in_user1.next();
                        userid[i]=in_user1.next();
                        i++;
                    }
                    in_user1.close();

                    // Making distinct Movies Array m_id[].
                    String movie;
                    String m_id[] = new String[4000];
                    Scanner         in_mov0         =         new         Scanner(new
FileReader(Inpfiles_ratings[k1]));
                    zipcen = in_mov0.next();
                    int movie_count,flag,k=0;
                    while(in_mov0.hasNext())
                    {
                        in_mov0.next();
                        movie=in_mov0.next();
                        in_mov0.next();

                        flag=0;
                        for(int j = 0;j <k; j++)
                        {
                            if(movie.compareTo(m_id[j])==0)
                            {
                                flag=1;
                                break;
                            }
                        }
                        if(flag==0)
                        {
                            m_id[k]=movie;
                            k++;
                        }
                    }
                    movie_count=k;
                    in_mov0.close();

                    // Making Rating Array rating_arr[movie_count][user_count].
                    int rating_arr[][]= new int[movie_count][user_count];
                    Scanner         in_mov1         =         new         Scanner(new
FileReader(Inpfiles_ratings[k1]));
                    zipcen = in_mov1.next();
                    int rating,user_index,mov_index;
                    while(in_mov1.hasNext())
                    {
                        user=in_mov1.next();
                        movie=in_mov1.next();
                        rating=in_mov1.nextInt();

                        user_index=get_index(userid,user);
                        mov_index=get_index(m_id,movie);

                        rating_arr[mov_index][user_index]=rating;
                    }

                    // FINDING CORRELATIONS AMONG USERS
```

```
                                String
outpath=currentdir+"\\zipsite_users_Correlation_N\\correlation"+Integer.toString(k1)+".dat";
                                PrintWriter out = new PrintWriter(outpath);
                                out.println(zipcen+" "+user_count);
                                int a=0,b=0,c=0,d=0;
                                for(i=0;i<user_count-1;i++)
                                {
                                        for(int j=i+1;j<user_count;j++)
                                        {
                                                int common_movie_index[] = new int[movie_count];
                                                int m=0,n=0,sum_ratg_a=0,sum_ratg_b=0;
                                                for(k=0;k<movie_count;k++)
                                                {

        if(rating_arr[k][i]!=0&&rating_arr[k][j]!=0)
                                                        {
                                                                m++;
                                                                sum_ratg_a += rating_arr[k][i];
                                                                sum_ratg_b += rating_arr[k][j];
                                                                common_movie_index[n]=k;
                                                                n++;
                                                        }
                                                }
                                                Double
part1,part2,avg_rat_a,avg_rat_b,num=0.0,denom,sum1=0.0,sum2=0.0,corr_coeff,de_part1,de_part2;
                                                avg_rat_a=sum_ratg_a/(double)m;
                                                avg_rat_b=sum_ratg_b/(double)m;

                                                for(int p=0;p<n;p++)
                                                {
                                                        int index1=common_movie_index[p];
                                                        // FOR NUMERATOR PART
                                                        part1=rating_arr[index1][i]-avg_rat_a;
                                                        part2=rating_arr[index1][j]-avg_rat_b;
                                                        num +=(part1*part2);

                                                        //FOR DENOMINATOR
                                                        de_part1=Math.pow(part1,2);
                                                        de_part2=Math.pow(part2,2);
                                                        sum1 += de_part1;
                                                        sum2 += de_part2;
                                                }
                                                denom=Math.sqrt((sum1*sum2));
                                                corr_coeff=num/denom;
                                                out.println(userid[i]+"              "+userid[j]+"
"+corr_coeff);
                                                if(Double.isNaN(corr_coeff))
                                                {
                                                        continue;
                                                }
                                                if(corr_coeff<0)
                                                {
                                                        a++;
                                                }
                                                else if(corr_coeff<0.25)
                                                        {
                                                                b++;
                                                        }
                                                        else if(corr_coeff<0.6)
                                                        {
                                                                c++;
                                                        }
                                                        else if(corr_coeff<=1)
                                                        {
                                                                d++;
                                                        }
                                        }
```

```
                                }

                                out.close();
                                user_count_in_diff_range[k1][0]=a;
                                user_count_in_diff_range[k1][1]=b;
                                user_count_in_diff_range[k1][2]=c;
                                user_count_in_diff_range[k1][3]=d;
                        }
                        String
outpath1=currentdir+"\\zipsite_users_Correlation_N\\count_range.dat";
                        PrintWriter out1 = new PrintWriter(outpath1);
                        for(int c=0;c<Inpfiles_users.length;c++)
                        {

        out1.println(user_count_in_diff_range[c][0]+"|"+user_count_in_diff_range[c][1]+"|"+user_
count_in_diff_range[c][2]+"|"+user_count_in_diff_range[c][3]);
                        }
                        out1.close();
                }
                catch(FileNotFoundException fileNotFoundException)
                {
                        System.err.println("Error Opening File.");
                        System.exit(1);
                }
        }

}
}
```

7. 'Find_Recommendation.java'

-This program will recommend you movies to active user depending on active user preference of movie using collaborative filtering.

Input: 'movies.dat' and 'correlationN' files.
Output: recommended set of movies.

Code:

```
import java.io.*;
import java.util.*;
class Find_recommendation
{
        static int   userid;
        static double thresh_val;
        static                  String                      mov_catg[]                      =
{"Animation","Children's","Comedy","Adventure","Fantasy","Romance","Drama","Action","Crime","Th
riller","Horror","Sci-Fi","Documentary","War","Musical","Mystery","Film-Noir","Western"};
        static int catg_count[] = new int[18];
        public static int chk_in_array(int arr[],int val)
        {
                int index,flag=0;
                for(index=0;index<arr.length;index++)
                {
                        if(val==arr[index])
                        {
                                flag=1;
                                break;
                        }
                }
                return flag;
        }
        public static int find_index(int arr[],int val)
        {
```

```
		int index,ret_index=0;
		for(index=0;index<arr.length;index++)
		{
			if(val==arr[index])
			{
				ret_index=index;
				break;
			}
		}
		return ret_index;
	}
	public static String find_movie_details(int movieid)
	{
		int movid,i,j;
		String movie_name=null;
		String categ=null;
		String movie_detail=null;
		try
		{
			Scanner in1 = new Scanner(new FileReader("movies.dat"));
			in1.useDelimiter("[:\n]+");

			while (in1.hasNext())
			{
				movid=in1.nextInt();
				if(movid==movieid)
				{
					movie_name=in1.next();
					categ=in1.next();
				}
				else
					in1.nextLine();
			}
			movie_detail=movie_name+" :: "+categ;
			in1.close();

		}catch(FileNotFoundException fileNotFoundException)
			{
				System.err.println("Error Opening File.");
				System.exit(1);
			}
		return movie_detail;
	}
	public static void main(String args[])
	{
		// Taking Userid as input.

		try
		{
			BufferedReader in= new BufferedReader(new InputStreamReader(System.in));
			System.out.print("Enter your userid (1-6040):");
			System.out.flush();
			userid=Integer.parseInt(in.readLine());
			System.out.flush();
			System.out.print("Enter the Threshold Value:");
			thresh_val=Double.parseDouble(in.readLine());
			System.out.println("userdid=    "+userid+"         "+"Threshold    Value
:"+thresh_val);
		}catch(IOException e) {
			System.out.println("Input Output Error");
			System.exit(1);
			}
		try
		{
			// Searching to which zipcell, that user belongs.
			String currentdir = System.getProperty("user.dir");
			String inpath = currentdir+"\\zipsite_users_N";
```

```
                        File InpFolder = new File(inpath); // Getting into zip_users_N directory.
                        File Inpfiles_users[];
                        Inpfiles_users = InpFolder.listFiles(); // Listing all files inside
zip_users_N directory.
                        int k,flag=0,user;
                        String zipcen;
                        String           zips,found_file="Not           EXIST",correl_file="Not
EXIST",rating_file="Not Exist";
                        for(k = 0;k<Inpfiles_users.length; k++)
                        {
                                Scanner         in_user0        =         new         Scanner(new
FileReader(Inpfiles_users[k]));
                                zipcen=in_user0.next();
                                flag=0;
                                while(in_user0.hasNext())
                                {
                                        zips=in_user0.next();
                                        user=in_user0.nextInt();
                                        if(user==userid)
                                        {
                                                flag=1;
                                                found_file="zipsite"+Integer.toString(k)+".dat";

        correl_file="correlation"+Integer.toString(k)+".dat";

        rating_file="zipsite_rating"+Integer.toString(k)+".dat";
                                                break;
                                        }
                                }
                                in_user0.close();
                                if(flag==1)
                                {
                                        break;
                                }
                        }
                        System.out.println("The file is: "+found_file+" and "+correl_file);
                        if(flag==1)
                        {
                                // Making an array of users highly coorelated with active user.
                                String
inpath2=currentdir+"\\zipsite_users_Correlation_N\\"+correl_file;
                                Scanner in_corr = new Scanner(new FileReader(inpath2));
                                zipcen=in_corr.next();
                                int user_count=in_corr.nextInt();
                                int uid1,uid2;
                                double coeff;
                                int userid_arr[]=new int[user_count];
                                Double user_coeff[]=new Double[user_count];
                                int rest_user_count=0;
                                while(in_corr.hasNext())
                                {
                                        uid1=in_corr.nextInt();
                                        uid2=in_corr.nextInt();
                                        coeff=in_corr.nextDouble();
                                        if(Double.isNaN(coeff))
                                        {
                                                continue;
                                        }
                                        else
                                        {
                                                if((uid1==userid      ||      uid2==userid)      &&
(coeff>=thresh_val))
                                                {
                                                        if(uid2!=userid)
                                                                userid_arr[rest_user_count]=uid2;
                                                        else
                                                                userid_arr[rest_user_count]=uid1;
```

```
                                                user_coeff[rest_user_count]=coeff;
                                                rest_user_count++;
                                        }
                                }
                        }
                        in_corr.close();

                // Making two array of movie's ids which are higly rated by active user
and rest of users respectively..
                        String
inpath3=currentdir+"\\zipsite_users_ratings_N\\"+rating_file;
                        Scanner in_rating = new Scanner(new FileReader(inpath3));
                        zipcen=in_rating.next();
                        int read_uid,read_mid,read_rating;
                        ArrayList al1 = new ArrayList(20);
                        ArrayList al2 = new ArrayList(60);

                        while(in_rating.hasNext())
                        {
                                read_uid=in_rating.nextInt();
                                read_mid=in_rating.nextInt();
                                read_rating=in_rating.nextInt();
                                if(read_rating==4 || read_rating==5)
                                {
                                        if(read_uid==userid)
                                        {
                                                al1.add(read_mid);
                                        }
                                        else
                                        {
                                                if(chk_in_array(userid_arr,read_uid)==1)
                                                        {
                                                                al2.add(read_mid);
                                                        }
                                        }
                                }
                        }
                        al1.trimToSize();
                        al2.trimToSize();
                        //Converting arraylist to array of exact size
                        Object active_fav_movie_arr1[]=al1.toArray();
                        Object rest_fav_movie_arr1[]=al2.toArray();
                        // CONVERTING Object TYPE ARRAY INTO int type array.
                        int active_fav_movie_arr[]=new int[active_fav_movie_arr1.length];
                        int rest_fav_movie_arr[]=new int[rest_fav_movie_arr1.length];
                        for(int i=0; i<active_fav_movie_arr1.length; i++)
                        {
                                active_fav_movie_arr[i]=((Integer)
active_fav_movie_arr1[i]).intValue();
                        }
                        for(int i=0; i<rest_fav_movie_arr1.length; i++)
                        {
                                rest_fav_movie_arr[i]=((Integer)
rest_fav_movie_arr1[i]).intValue();
                        }
                        System.out.println();
                    //Finding the top categories of user's choice by updating the count
of each values of array catg_count[], and then taking 2-3 top count values, which gives
corresponding categories.
                    // AND Forming 2D array for finding categories of movies of rest
users.
                    int rest_mov_catg[][]=new int[rest_fav_movie_arr.length][18];
                        Scanner in0 = new Scanner(new FileReader("movies.dat"));
                        in0.useDelimiter("[:\n]+");
                        int movid,i,j;
                        String movie_name;
                        String categ;
```

```
                        String delim = "[|\r]";
                        String types[];
                        while (in0.hasNext())
                        {
                                movid=in0.nextInt();
                                        if(chk_in_array(active_fav_movie_arr,movid)==1   ||
chk_in_array(rest_fav_movie_arr,movid)==1)
                                        {
                                                movie_name=in0.next();
                                                categ=in0.next();
                                                types=categ.split(delim);

        if(chk_in_array(active_fav_movie_arr,movid)==1)
                                                {
                                                        for(i=0;i<types.length;i++)

        for(j=0;j<mov_catg.length;j++)
                                                                {

        if(mov_catg[j].compareTo(types[i])==0)
                                                                        {
                                                                                catg_count[j]
+=1;
                                                                                break;
                                                                        }
                                                                }
                                                }

        if(chk_in_array(rest_fav_movie_arr,movid)==1)
                                                {
                                                        int
mov_index=find_index(rest_fav_movie_arr,movid);
                                                        for(i=0;i<types.length;i++)

        for(j=0;j<mov_catg.length;j++)
                                                                {

        if(mov_catg[j].compareTo(types[i])==0)
                                                                        {

        rest_mov_catg[mov_index][j]=1;


                                                                        }
                                                                }
                                                }
                                        }
                                        else
                                        {
                                                in0.nextLine();
                                        }
                        }
                        in0.close();
                        for(k=0;k<catg_count.length;k++)
                        {
                                System.out.println(mov_catg[k]+"  "+catg_count[k]);
                        }

                        //Finding first two top categories(i.e. index in catg_count[] and
eventually mov_catg[])
                        int max1,max2,catg_max1_index=0,catg_max2_index=0;
                        max1=catg_count[0];
                        for(i=1;i<catg_count.length;i++)
                        {
                                if(catg_count[i]>max1)
                                {
                                        max1=catg_count[i];
                                        catg_max1_index=i;
```

```
                                }
                        }
                        if(max1==catg_count[0])
                                max2=catg_count[1];
                        else
                                max2=catg_count[0];
                        for(i=0;i<catg_count.length;i++)
                        {
                                if(i!=catg_max1_index && catg_count[i]>max2)
                                {
                                max2=catg_count[i];
                                catg_max2_index=i;
                                }

                        }
                        System.out.println("Top             two             categories:
"+mov_catg[catg_max1_index]+" "+mov_catg[catg_max2_index]);
                        //RECOMMENDING MOVIES..
                        System.out.println("***********RECOMMENDED MOVIES***********");
                        for(i=0;i<rest_fav_movie_arr.length;i++)
                        {
                                if(rest_mov_catg[i][catg_max1_index]==1                 &&
rest_mov_catg[i][catg_max2_index]==1)
                                {


        if(chk_in_array(active_fav_movie_arr,rest_fav_movie_arr[i])==0)
                                                System.out.println(rest_fav_movie_arr[i]+"
"+find_movie_details(rest_fav_movie_arr[i]));
                                }

                                }
                        }
                        else
                        {
                                System.out.println("SORRY!! INVALID USER.!!");
                                System.exit(1);
                        }

                }catch(FileNotFoundException fileNotFoundException)
                        {
                                System.err.println("Error Opening File.");
                                System.exit(1);
                        }
        }
}
```

Test program:

```
import java.io.*;
import java.util.*;
class Test_prg1
{
        static int   userid,pos_usr_count=0,neg_usr_count=0;
        static double thresh_val;
        static int test_user_count = 350; // The no. of user for which program is testing.
        static float rat_diff_sum=0,rat_diff_count=0;
        static ArrayList rat_diff_arr_list = new ArrayList(test_user_count);
        static                      String                      mov_catg[]                    =
{"Animation","Children's","Comedy","Adventure","Fantasy","Romance","Drama","Action","Crime","Th
riller","Horror","Sci-Fi","Documentary","War","Musical","Mystery","Film-Noir","Western"};
        static int catg_count[] = new int[18];
        public static int chk_in_array(int arr[],int val)
        {
                int index,flag=0;
                for(index=0;index<arr.length;index++)
```

```
		{
			if(val==arr[index])
			{
				flag=1;
				break;
			}
		}
		return flag;
	}
	public static int find_index(int arr[],int val)
	{
		int index,ret_index=0;
		for(index=0;index<arr.length;index++)
		{
			if(val==arr[index])
			{
				ret_index=index;
				break;
			}
		}
		return ret_index;
	}
	public static String find_movie_details(int movieid)
	{
		int movid,i,j;
		String movie_name=null;
		String categ=null;
		String movie_detail=null;
		try
		{
			Scanner in1 = new Scanner(new FileReader("movies.dat"));
			in1.useDelimiter("[:\n]+");

			while (in1.hasNext())
			{
				movid=in1.nextInt();
				if(movid==movieid)
				{
					movie_name=in1.next();
					categ=in1.next();
				}
				else
					in1.nextLine();
			}
			movie_detail=movie_name+" :: "+categ;
			in1.close();

		}catch(FileNotFoundException fileNotFoundException)
			{
				System.err.println("Error Opening File.");
				System.exit(1);
			}
		return movie_detail;
	}
	public static void main(String args[])
	{
		thresh_val=0.6;
		try
		{
				String currentdir = System.getProperty("user.dir");
				String newDir = "TEST_PRG1_RESULT";
				File f = new File(newDir);
				f.mkdir();
				String
test_outpath=currentdir+"\\TEST_PRG1_RESULT\\Test_prg1_result.dat";
				PrintWriter test_out = new PrintWriter(test_outpath);
```

```
                                for(int usr=1;usr<test_user_count;usr++)
                                {
                                        userid=usr;
                                        // Searching to which zipcell, that user belongs.
                                        String inpath = currentdir+"\\zipsite_users_N";
                                        File  InpFolder  =  new  File(inpath);  //  Getting  into
zip_users_N directory.
                                        File Inpfiles_users[];
                                        Inpfiles_users  =  InpFolder.listFiles();  //  Listing  all
files inside zip_users_N directory.
                                        int k,flag=0,user;
                                        String zipcen;
                                        String    zips,found_file="Not    EXIST",correl_file="Not
EXIST",rating_file="Not Exist";
                                        for(k = 0;k<Inpfiles_users.length; k++)
                                        {
                                                Scanner     in_user0     =     new     Scanner(new
FileReader(Inpfiles_users[k]));
                                                zipcen=in_user0.next();
                                                flag=0;
                                                while(in_user0.hasNext())
                                                {
                                                        zips=in_user0.next();
                                                        user=in_user0.nextInt();

                                                        if(user==userid)
                                                        {
                                                                flag=1;

        found_file="zipsite"+Integer.toString(k)+".dat";

        correl_file="correlation"+Integer.toString(k)+".dat";

        rating_file="zipsite_rating"+Integer.toString(k)+".dat";
                                                                break;
                                                        }
                                                }
                                                in_user0.close();
                                                if(flag==1)
                                                {
                                                        break;
                                                }
                                        }
                                        if(flag==1)
                                        {
                                                try
                                                {
                                                        //   Making   an   array   of   users   highly
coorelated with active user.
                                                        String
inpath2=currentdir+"\\zipsite_users_Correlation_N\\"+correl_file;
                                                        Scanner    in_corr    =    new    Scanner(new
FileReader(inpath2));
                                                        zipcen=in_corr.next();
                                                        int user_count=in_corr.nextInt();
                                                        int uid1,uid2;
                                                        double coeff;
                                                        int userid_arr[]=new int[user_count];
                                                        Double user_coeff[]=new Double[user_count];
                                                        int rest_user_count=0;
                                                        while(in_corr.hasNext())
                                                        {
                                                                uid1=in_corr.nextInt();
                                                                uid2=in_corr.nextInt();
                                                                coeff=in_corr.nextDouble();
                                                                if(Double.isNaN(coeff))
```

```
                                                        {
                                                                continue;
                                                        }
                                                        else
                                                        {
                                                                if((uid1==userid          ||
uid2==userid) && (coeff>=thresh_val))
                                                                {
                                                                        if(uid2!=userid)

        userid_arr[rest_user_count]=uid2;
                                                                        else

        userid_arr[rest_user_count]=uid1;

        user_coeff[rest_user_count]=coeff;
                                                                        rest_user_count++;
                                                                }
                                                        }
                                                }
                                                in_corr.close();

                                        // Making two array of movie's ids which are higly
rated by active user and rest of users respectively..
                                                String
inpath3=currentdir+"\\zipsite_users_ratings_N\\"+rating_file;
                                                Scanner   in_rating   =   new   Scanner(new
FileReader(inpath3));
                                                zipcen=in_rating.next();
                                                int read_uid,read_mid,read_rating;
                                                ArrayList  al1  =  new  ArrayList(60);//  For
only Highly rated movieids by active user.
                                                ArrayList  al2  =  new  ArrayList(60);//  For
only Highly rated movieids by correlated users.
                                                ArrayList  al3  =  new  ArrayList(60);//  For
all movieids rated by active user.
                                                ArrayList  al4  =  new  ArrayList(60);//  For
all ratings given to movies by active user.


                                                while(in_rating.hasNext())
                                                {
                                                        read_uid=in_rating.nextInt();
                                                        read_mid=in_rating.nextInt();
                                                        read_rating=in_rating.nextInt();
                                                        if(read_uid==userid)
                                                        {
                                                                al3.add(read_mid);
                                                                al4.add(read_rating);
                                                        }
                                                        if(read_rating==4                ||
read_rating==5)
                                                        {
                                                                if(read_uid==userid)
                                                                {
                                                                        al1.add(read_mid);
                                                                }
                                                                else
                                                                {

        if(chk_in_array(userid_arr,read_uid)==1)
                                                                                        {

        al2.add(read_mid);
                                                                                        }
                                                                }
                                                        }
```

```
}
al1.trimToSize();
al2.trimToSize();
al3.trimToSize();
al4.trimToSize();
//Converting arraylist to array of exact
size
Object
active_fav_movie_arr1[]=al1.toArray();
Object
active_all_movie_arr1[]=al3.toArray();
Object
active_all_rating_arr1[]=al4.toArray();
Object rest_fav_movie_arr1[]=al2.toArray();
// CONVERTING Object TYPE ARRAY INTO int
type array.
int              active_fav_movie_arr[]=new
int[active_fav_movie_arr1.length];
int              active_all_movie_arr[]=new
int[active_all_movie_arr1.length];
int             active_all_rating_arr[]=new
int[active_all_rating_arr1.length];
int                rest_fav_movie_arr[]=new
int[rest_fav_movie_arr1.length];
for(int                                 i=0;
i<active_fav_movie_arr1.length; i++)
{
       active_fav_movie_arr[i]=((Integer)
active_fav_movie_arr1[i]).intValue();
}
for(int                                 i=0;
i<active_all_movie_arr1.length; i++)
{
       active_all_movie_arr[i]=((Integer)
active_all_movie_arr1[i]).intValue();
}
for(int                                 i=0;
i<active_all_rating_arr1.length; i++)
{
       active_all_rating_arr[i]=((Integer)
active_all_rating_arr1[i]).intValue();
}
for(int  i=0;  i<rest_fav_movie_arr1.length;
i++)
{
       rest_fav_movie_arr[i]=((Integer)
rest_fav_movie_arr1[i]).intValue();
}

//Finding  the  top  categories  of  user's
choice by updating the count of each values of array catg_count[], and then taking 2-3 top
count values, which gives corresponding categories.
//  AND  Forming  2D  array  for  finding
categories of movies of rest users.
int                   rest_mov_catg[][]=new
int[rest_fav_movie_arr.length][18];
Scanner    in0    =    new    Scanner(new
FileReader("movies.dat"));
in0.useDelimiter("[:\n]+");
int movid,i,j;
String movie_name;
String categ;
String delim = "[|\r]";
String types[];
while (in0.hasNext())
{
       movid=in0.nextInt();
```

```
        if(chk_in_array(active_fav_movie_arr,movid)==1                                  ||
chk_in_array(rest_fav_movie_arr,movid)==1)
                                                                        {
        movie_name=in0.next();
                                                                                categ=in0.next();
        types=categ.split(delim);
        if(chk_in_array(active_fav_movie_arr,movid)==1)
                                                                                {
        for(i=0;i<types.length;i++)
        for(j=0;j<mov_catg.length;j++)
                                                                                        {
        if(mov_catg[j].compareTo(types[i])==0)
        {
                catg_count[j] +=1;
                break;
        }
                                                                                        }
                                                                                }
        if(chk_in_array(rest_fav_movie_arr,movid)==1)
                                                                                {
                                                                                        int
mov_index=find_index(rest_fav_movie_arr,movid);
        for(i=0;i<types.length;i++)
        for(j=0;j<mov_catg.length;j++)
                                                                                        {
        if(mov_catg[j].compareTo(types[i])==0)
        {
                rest_mov_catg[mov_index][j]=1;

        }
                                                                                        }
                                                                                }
                                                                        }
                                                                        else
                                                                        {
                                                                                in0.nextLine();
                                                                        }
                                                                }
                                                        in0.close();
                                                        //Finding  first  two  top  categories(i.e.
index in catg_count[] and eventually mov_catg[])
                                                        int
max1,max2,catg_max1_index=0,catg_max2_index=0;
                                                        max1=catg_count[0];
                                                        for(i=1;i<catg_count.length;i++)
                                                        {
                                                                if(catg_count[i]>max1)
                                                                {
```

```
                                                        max1=catg_count[i];
                                                        catg_max1_index=i;
                                                }
                                        }
                                        if(max1==catg_count[0])
                                                max2=catg_count[1];
                                        else
                                                max2=catg_count[0];
                                        for(i=0;i<catg_count.length;i++)
                                        {
                                                        if(i!=catg_max1_index     &&
catg_count[i]>max2)
                                                        {
                                                        max2=catg_count[i];
                                                        catg_max2_index=i;
                                                        }

                                        }

                                        int rat_sum=0,recc_count=0;
                                        for(i=0;i<rest_fav_movie_arr.length;i++)
                                        {

        if(rest_mov_catg[i][catg_max1_index]==1 && rest_mov_catg[i][catg_max2_index]==1)
                                                        {

        if(chk_in_array(active_all_movie_arr,rest_fav_movie_arr[i])==1)
                                                                {
                                                                        int
index=find_index(active_all_movie_arr,rest_fav_movie_arr[i]);
                                                                        rat_sum
+=active_all_rating_arr[index];
                                                                        recc_count+=1;
                                                                }
                                                        }

                                                        /*
        if(rest_mov_catg[i][catg_max1_index]==1 || rest_mov_catg[i][catg_max2_index]==1)
                                                        {

        //System.out.println(rest_fav_movie_arr[i]);

        if(chk_in_array(active_fav_movie_arr,rest_fav_movie_arr[i])==0)

        System.out.println(rest_fav_movie_arr[i]);
                                                        }*/

                                        }
                                        float       avg_rat_on_recc_movies       =
(float)rat_sum/(float)recc_count;

                                        //  CALCULATING  AVERAGE  RATING  OF  ACTIVE
USER ON ALL THE MOVIES RATED BY HIM.

                                        int all_rat_sum=0,all_count=0;
                                        for(int
i1=0;i1<active_all_rating_arr.length;i1++)
                                        {
                                                all_rat_sum                     +=
active_all_rating_arr[i1];

                                                all_count +=1;
                                        }
                                        float        avg_rat_on_all_movies        =
(float)all_rat_sum/(float)all_count;

                                        if(Double.isNaN(avg_rat_on_recc_movies)  ||
Double.isNaN(avg_rat_on_all_movies))
                                        {
                                                continue;
                                        }
```

```
                                                float         diff=(avg_rat_on_recc_movies-
avg_rat_on_all_movies);
                                                float abs_diff=Math.abs(diff);
                                                rat_diff_arr_list.add(diff);
                                                rat_diff_sum += diff;
                                                rat_diff_count +=1;
                                                if(diff>=0)
                                                {
                                                       pos_usr_count+=1;
                                                }
                                                else
                                                       neg_usr_count+=1;
                                                test_out.println(userid+"
"+avg_rat_on_recc_movies+" "+avg_rat_on_all_movies+" "+(diff));
                                         }catch(InputMismatchException
inputmismatchException)
                                         {
                                                continue;
                                         }
                                  }
                                  else
                                  {
                                         continue;

                                  }

                           }

                           // Converting rat_diff_arr_list into array of exact size.
                           rat_diff_arr_list.trimToSize();
                           Object rat_diff_arr1[]=rat_diff_arr_list.toArray();
                           float rat_diff_arr[]=new float[rat_diff_arr1.length];
                           for(int i=0; i<rat_diff_arr1.length; i++)
                           {
                                  rat_diff_arr[i]=((Float) rat_diff_arr1[i]).floatValue();
                           }
                           // Calculating Average difference.
                           float avg_diff = rat_diff_sum/rat_diff_count;
                           // Calculating Standard Deviation
                           double temp_val1,temp_val2,sum_tempval2=0.0;
                           for(int i1=0; i1<rat_diff_arr.length; i1++)
                           {
                                  temp_val1 = rat_diff_arr[i1]-avg_diff;
                                  temp_val2 = Math.pow(temp_val1,2);
                                  sum_tempval2 += temp_val2;
                           }
                           double std_dev = Math.sqrt(sum_tempval2/rat_diff_count);
                           test_out.println("Avg.  Diff.  ="+avg_diff+"  |  Std.  Deviation
="+(float)std_dev);
                           test_out.println("No. of User with pos. diff ="+pos_usr_count+"
No. of User with neg. diff ="+neg_usr_count);
                           test_out.close();
                    }catch(FileNotFoundException fileNotFoundException)
                           {
                                  System.err.println("Error Opening File.");
                                  System.exit(1);
                           }

      }
}
```

Conclusion

We have proposed a variant of Collaborative Filtering which is location-aware by using Voronoi polygons to partition the space effectively. Exploring other clustering techniques to spatially partition the data and comparing with the Voronoi approach will be the focus of our future research.

References

[1]. Bhasker, B. and Srikumar, K., 2010. *Recommender Systems in e-Commerce*, Tata McGraw Hill.

[2]. Herlocker, J. L., Konstan, J.A., Riedl, J., 2000. Explaining Collaborative Filtering Recommendations. *CSCW '00 Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work* (Philadelphia, USA, December 02-06, 2000).

[3]. Van Kreveld, M., de Berg, M., Cheong, O., 2008. *Computational Geometry: Algorithms and Applications*,Springer.

[4]. Jinni, URL : http://www.jinni.com/

[5]. MovieLens, URL : http://www.movielens.org/

[6]. Netflix, URL : http://www.netflix.com/

[7]. Brunato, M., Battiti, R., Villani, A. and Delai, A., 2002. A Location-Dependent Recommender System for the web. *Technical report DIT-02-0093, Universita` di Trento.*

[8]. Brunato, M. and Battiti, R., 2003. PILGRIM: A Location Broker and Mobility-Aware Recommendation System. *In proceedings of IEEE PerComp2003* (March 23-26, 2003), 265-272.

[9]. HotPot, URL : http://www.google.com/hotpot

[10]. Li, X., Mi, Z., Zhang, Z. and Wu, J., 2010. A Location-Aware Recommender System for Tourism Mobile Commerce.*In proceedings of 2nd International Conference Information Science and Engineering (ICISE)* (Hangzhou, China, December 04-06, 2010), 1709-1711.

[11]. Yang, W., Cheng, H. Dia, J., 2008. A Location-Aware Recommender System for Mobile Shopping Environments. *Expert Systems with Applications: An Intenational Journal* (Volume 34, Issue 1, January, 2008), 437-445.