ONLINE SOCIAL NETWORK SYNTHETIC DATA GENERATOR

Copyright (C) 2018 David F. Nettleton (david.nettleton@upf.edu)
License: GNU GENERAL PUBLIC LICENSE v3.0 See LICENSE for the full license

Congratulations, you have in your hands a powerful, flexible, realistic synthetic data generator for online social networks, implemented in Eclipse Java. It is standalone and doesn't require any back-end database or auxiliary applications.

Depending on if you are a basic, intermediate or advanced user, we would suggest the following:

(i) Basic user: use the program *as is*.
(ii) Intermediate user: try tweaking the user and social network profiles in "rAssignDataGeneralizedSensAttr4.java" and see the results (below are details). Note that this version defined 10 profiles which can be assigned in different proportions, and 10 communities (one per profile).
(iii) Use the software as the basis to develop your own version, and/or incorporate it in some larger system.

Note that two versions will exist (shortly) in github: the first one (this one) is relatively simpler because each user can only belong to one community and the number of communities is fixed; the second one is somewhat more complex because it allows overlapping communities in which a user can belong to multiple communities, and the number of communities is variable (depends on graph dataset).

In both cases the system expects two input files:

File 1 contain a graph, defined as edges, one per row, in the format id1 id2, i1 id3, .. etc. where id is a node id and 'id1 id2' indicates there is an edge (link) between these two nodes.

File 2 contains the community labels for each of the users, that is, each user has an associated community label in the format id mod, where id is the user and mod is the community id (usually, 1,2,3, etc.).

So, first you will need obtain a predefined (empty) graph topology (nodes and links between them). Different options exist: (i) you can generate one using RMAT; (ii) you can obtain one from online repositories (versions for LiveJournal, Youtube and other major online apps exist); (iii) you already have your own graph.

Next, you will have to process the graph using a community detection algorithm. The one I use is the Louvain method available in the Gephi graph software freeware package. This will assign a label to each node indicating its community. Or you may have your own approach for obtaining/assigning the community labels. By default, one user will be assigned one label. For this version of the program, you will have to fix the number of communities to 10 (you can do this in Gephi by playing with the modularity until you obtain exactly 10).

Note that some example input and output files are included in the Java Project.

Once you have these two files you are now ready to execute the data generator. This will assign, to each node/user, a set of data attributes. These data attributes include “static” demographic information such as age, gender, place of residence, political and sexual tendencies, etc. Also, it will create dynamic data, such as likes (sports, product brands, hobbies, ..).

The basic rules when creating the data is that the neighbors of a user tend to be more similar than users who are more distant (2, 3 or more hops) in the graph. Also, similar users tend to agglomerate together, hence the communities. The algorithm starts by assigning “seed users”, one per user seed profile (by default the number of user seed profiles is equal to the number of communities). Next step is to generate neighbors for each seed, which will be similar to the seeds, but covering certain distributions for each key characteristics, and with some random variations (so that users are not closed in bubbles). All of this will take into account maintaining a “similar” profile identify for a community, and also maintaining overall proportions of users and their profiles, and individual characteristics (e.g. gender, age, ..) in the complete graph.

The input and output files are set in RV.java. Currently assigned is a “toy” graph from the literature, which has been widely referenced.

```
String nomFixterIn1    = "karate.csv";
String nomFixterIn2    = "karatemodauth.csv";
String nomFixterOut    = "karate_out.csv";
String nomFixterOutg   = "karate_outg.csv";
String nomFixterOut1   = "karate_out1.csv";
String nomFixterOut2   = "karate_out2.csv";
```

RV then calls “leer_datos_de_casos” which a principal routine which reads the input files, and calls the major processing routines.

You will see in that after reading the second (community) file, there is a line before the processing starts, which says:

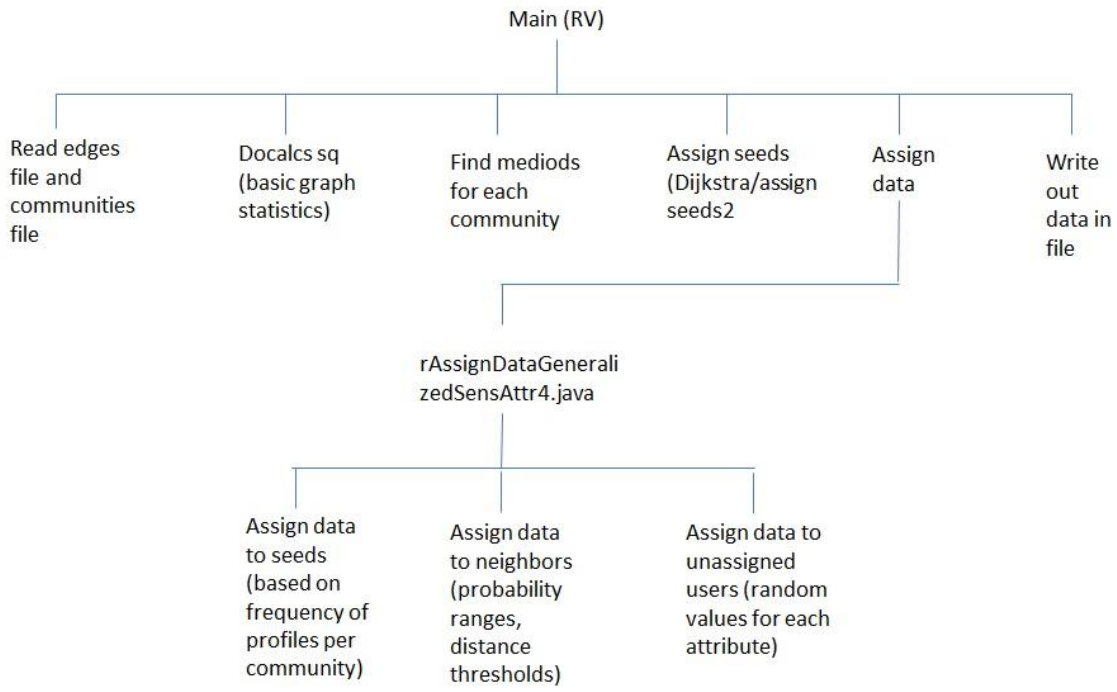
int seedsize = 110; // 110 seeds for 1K synth file, 5K seeds for amazon, 12k seeds for youtube and livejournal.

The seedsize is graph dependent and take into account that the more seeds, the more time it will take to locate them. The values are orientative. Actually, as the Karate graph has about 30 nodes the number of seeds should be proportionately less, for example, one per community (about 4 if you run this through Gephi Louvain).

And that’s all you should need to know to get started !

The following is only for those with programming skills who would like to adapt the Java code or know internal working details. It is not necessary in order to run the program and obtain datasets.

The Java programs are structured as follows:



The Java Eclipse project contents, folder by folder, are:

Main: "GenDataNO.java"
RV: "RV.java", "rAssignDataGeneralizedSensAttr4.java", "calcDistanceBetweenUsers.java", "StatsCommunities.java"
User: "User.java"
doCalcs: "doCalcs.java"
Dijkstra: "assignSeeds2.java", "findMediod.java"
dataFile: "dataFile.java"
Community: "Community.java"

The basic definition of a user is in User/User.java :

```
private String residence;
private String age;
private int age_lowerb;
private int age_upperb;
private String gender;
private String like1;
private String like2;
private String like3;
```

```
private String religion;
private String maritalstatus;
private String profession;
private String politicalorientation;
private String sexualorientation;
```

If you change this you will have to change the corresponding data assignment and processing in "rAssignDataGeneralizedSensAttr4.java".

rAssignDataGeneralizedSensAttr4.java

This program is where the business gets done of assigning the data to the users. Note that previous to this, we have identified the mediods and placed the seeds in each community, which are the starting point for data assignment and propagation in the graph.

It consists of three main routines which are executed in sequence:

AssignSeeds : this assigns the prototypical data profiles to the seeds.
AssignNeighbors: this assigns the data profiles to the neighbors of the seeds in a probabilistic manner.
AssignUnassigned: any seeds which are still unassigned after the first two routines get assign here, in a semi-probabilistic (similarity to their neighbors) and semi-random manner.

First the 10 profiles are assigned, the first being profile 0:

```
profile_age[0]  = age[6];  profile_gen[0]  = gender[0];   profile_res[0]  = residence[3];  profile_rel[0]  = greligion[8];
profile_mar[0]  = gmaritalstatus[1];  profile_prof[0] = gprofession[3];
profile_lk1[0]  = glike1[3];  profile_lk2[0]  = glike2[3];  profile_lk3[0]  = glike3[3];
profile_pol[0]  = gpoliticalorientation[3]; profile_seo[0]  = gsexualorientation[2];
```

As you will see, everything is hard-coded in the Java program. So an experienced programmer can customize this as they want. Of course, the profiles and control parameters could be defined in an external file and loaded by the program, so I'll leave that up to you.

The list of categories for each characteristic is defined previously in the code. For example:

```
age[0] = "18-25";
age[1] = "18-25";
age[2] = "18-25";
age[3] = "26-35";
age[4] = "26-35";
age[5] = "26-35";
age[6] = "36-45";
age[7] = "36-45";
age[8] = "46-55";
age[9] = "56-65";
age[10] = "66-75";
age[11] = "76-85";
```

Note that the number of times an age reange appears in the vector represents its frequency in the selection. So age range "18-25" appears three times out of 12, so it has a relative probability of 3/12 of being assigned.

However, in the case of religion, a separate vector contains the desired assignment proportions. For example, "Hindu" has a probability of 13.8% of being assigned (138 over 1000).

```
greligion[0] = "Buddhist";  religionf[0] = 68; //6.8%
greligion[1] = "Christian"; religionf[1] = 304;//30.4%
greligion[2] = "Hindu";     religionf[2] = 138;//13.8%
greligion[3] = "Jewish";    religionf[3] = 2;//0.2%
greligion[4] = "Muslim";    religionf[4] = 257;//25.7%
greligion[5] = "Sikh";      religionf[5] = 3;//0.3%
greligion[6] = "Traditional Spirituality";  religionf[6] = 1;//0.1%
greligion[7] = "Other Religions";         religionf[7] = 110;//10.9%
greligion[8] = "No religious affiliation";  religionf[8] = 117;//11.7%
```

The overall profile frequencies are controlled in AssignSeeds, thus:

```
profilefreq[0]=216; // sum to 1000
profilefreq[2]=211;
profilefreq[1]=172;
profilefreq[5]=157;
profilefreq[3]=97;
profilefreq[4]=81;
profilefreq[8]=28;
profilefreq[6]=24;
profilefreq[9]=9;
profilefreq[7]=5;
```

The profiles are assigned (in AssignSeeds) to communities thus:

```
profilevalues[0]=0; // assign profile 0 to community 0
profilevalues[1]=2; // assign profile 2 to community 1
profilevalues[2]=1; // assign profile 1 to community 2
profilevalues[3]=4; // assign profile 4 to community 3
profilevalues[4]=5; // assign profile 5 to community 4
profilevalues[5]=3; // assign profile 3 to community 5
profilevalues[6]=7; // assign profile 7 to community 6
profilevalues[7]=9; // assign profile 9 to community 7
profilevalues[8]=6; // assign profile 6 to community 8
profilevalues[9]=8; // assign profile 8 to community 9
```

Note that 10 profiles are assigned to 10 communities. In this version it is assumed the number of communities is 10. In your community detection algorithm, you can execute it so that it finds 10 communities (like in clustering where you predefine the number of desired clusters.

Then in AssignNeighbors there are some assignment thresholds:

```
//Specific random thresholds for each attribute
//double   //distfage=1.1,         limfage=0.5,   // AGE, that is, distage must be limage or less (0.2)
//distfres=1.1,          limfres=0.5,   // RESIDENCE (0.3)
//distfpol=1.1,          limfpol=0.5,   // POLITICAL ORIENTATION (0.2)
//distfseo=1.1,          limfseo=0.5,   // SEXUAL ORIENTATION (0.0)
//distfrel=1.1,          limfrel=0.5,   // RELIGION (0.0)
//distfmar=1.1,          limfmar=0.5,   // MARITAL STATUS (0.0)
//distfpro=1.1,          limfpro=0.5,   // PROFESSION (0.0)
//distflikes=1.1, limflikes=0.5; // LIKES (0.3)
//distfgen=1.1,          limfgen=0.0;              // GENDER (0.0)
```

Also, in AssignNeighbors, there are probability ranges for each attribute, for example, in the case of profession:

```
//int LimPro1=60, LimPro2=61, LimPro3=71, LimPro4=81, LimPro5=91, LimPro6=86; // PROFESSION (low)
int LimPro1=50, LimPro2=51, LimPro3=61, LimPro4=71, LimPro5=81, LimPro6=76; // PROFESSION (medium)
//int LimPro1=40, LimPro2=41, LimPro3=51, LimPro4=61, LimPro5=71, LimPro6=66; // PROFESSION (high)
```

And finally, distance threholds for each attribute:

```
//*************************************************************************************************
//DISTANCE THRESHOLDS
// The distance threshold depends on how the distance is calculated and may differ from attribute to attribute
// It indicates the distance of the two closest categories. In the case of residence it indicates the distance
// for categories in the same county, state and max. distance, respectively
//*************************************************************************************************
        double dpolthresh=(double)((double)1.0/(double)6.0);  // POLITICS

        double dprothresh=(double)((double)1.0/(double)2.0);  // PROFESSION

        double dgenthresh=(double)1.0; // GENDER

        double dagthresh=(double)((double)1.0/(double)6.0);  // AGE

        double dresthresh1=(double)((double)1.0/(double)4.0); //county =.25 RESIDENCE
        double dresthresh2=(double)((double)1.0/(double)2.0); //state =.5
        double dresthresh3=(double)1.0; //1.0

        double drelthresh=(double)((double)1.0/(double)2.0);   // RELIGION

        double dseothresh=(double)((double)1.0/(double)2.0); // =.5   SEO

        double dmarthresh=(double)((double)1.0/(double)2.0); // =.5    MARITAL

        double dlikthresh1 = 0.15; // v close  > 0.00 and < .15  LIKES
        double dlikthresh2 = 0.24; // m close  > 0.15 and < .24
```

Finally, the routine AssignUnassigned mops up the user nodes which were not assigned as seeds or neighbors of seeds. Note that the seed assignment aims to optimize graph coverage, thus minimizing the number of unassigned seeds, however the final percentage of unassigned is dependent on graph topology and community mapping.

```
//We assign the modal values 90% of the time, and random assignment 10% of the time
```

(The modal value of the characteristics of an unassigned nodes neighbors is calculated and this is the profile which is assigned 90% of the time).

The remaining 10% of the time, a random characteristic set is assigned:

```
agix =        ((int)(generator.nextInt(12))); // rnd will be between 0 and limit-1
genix =  ((int)(generator.nextInt(2))); // rnd will be between 0 and limit-1
resix =  ((int)(generator.nextInt(6))); // rnd will be between 0 and limit-1
relix =  ((int)(generator.nextInt(1000)));   // rnd will be between 0 and limit-1
marix =  ((int)(generator.nextInt(96)));   // rnd will be between 0 and limit-1
profix =  ((int)(generator.nextInt(100))); // rnd will be between 0 and limit-1
polix =  ((int)(generator.nextInt(99)));   // rnd will be between 0 and limit-1
seoix =  ((int)(generator.nextInt(100)));   // rnd will be between 0 and limit-1
 lkix1 =  ((int)(generator.nextInt(265))); // rnd will be between 0 and limit-1
```

So, that's a quick tour. If you have any doubts just let me know at david.nettleton@upf.edu !