The Overlapping Community Detection Project

October 22, 2014

1 Notation

We use:

1. n to denote the number of nodes in the network;
2. m to denote the number of edges;
3. s to denote number of seed nodes;
4. k to denote the number of communities;
5. A_1 to denote the adjacency matrix of the induced by non-seed nodes;
6. D_1 to denote the diagonal matrix of the total degrees of the non-seed nodes;
7. $x_0, \ldots, x_{s-1}$ to denote seed nodes and $u_0, \ldots, u_{n-s-1}$ to denote non-seed nodes.

2 Components of the program

The program consists of the following classes.

The `Graph` class. The `Graph` class is responsible for reading an undirected graph from file in LEDA format which it stores in an adjacency list. The adjacency list itself is a `std::vector` of `std::list<int>` and this list also has the degree information stored explicitly. One of the constructors of this class looks like:

```
Graph(const char* path_to_graph_file)
```

The interface for this class consists of the following members:

1. `num_vertices();`
2. `num_edges();`
3. `int get_degree(int);`
4. `std::list<int> get_neighbors(int);`
5. `display_graph();`

The `SeedNode` class. This is responsible for generating the set of seed nodes and their affinities to the respective communities. At the very least, it would read a file containing the seed node information which might look like as shown in Table 1.

num_seed_nodes	*num_communities*		
x_0	$\alpha(0,1)$	$\ldots$	$\alpha(0,k)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_{s-1}	$\alpha(n-1,1)$	$\ldots$	$\alpha(n-1,k)$

Table 1: File format for seed node information.

For now, we endow it with the following member functions:

1. `SeedNode(const char* path_name); //constructor`
2. `bool is_seed( int id );`
3. `int num_seed();`
4. `int get_matrix_id(int id);`
5. `int num_communities();`
6. `int get_affinity(int seedNode,int community);`
7. `int get_vertex_id(int id);` to get from the matrix id to the vertex id (only needed for seed nodes)

Later on, we might want to enrich this class by writing down how to generate seed nodes randomly and create their affinity vectors using only the adjacency list and user-specified input (perhaps).

The `MatrixBuilder`. The Matrix Builder builds the matrices A_1, D_1, and R using the adjacency list and seed node information supplied by the `Graph` class. One of the jobs that is has to do is create two vectors: an $(n-s)$-vector that maps the index of the rows of R to the (non-seed) node that the index corresponds to; an s-vector that maps the column index of R to the seed node that it corresponds to.

The `matrix_solver`. This sets up the system of linear equations and obtains, as solutions, the $n \times k$ matrix of affinities. The input parameters are the graph, the set of seed nodes, pointers to matrices A_1, D_1, R, the seed node matrix. Using these, it creates a system of linear equations for *each* community. For community l, this system looks like:

$$(D_1 - A_1)\begin{pmatrix}\alpha(u_0,l)\\ \vdots \\ \alpha(u_{n-s-1},l)\end{pmatrix} = D_1 \sum_{i=0}^{s-1} \alpha(x_0,l)R_i.$$

Here $\alpha(v,l)$ denotes the affinity of node v for community l; R_i denotes the i^{th} column of matrix R. The `matrix_solver` returns an $n \times k$ matrix of affinities in a file (see Table 2).

The `CommunityClassifier`. This reads the affinities from a file whose format is shown in Table 2. It then places each node into an appropriate number of communities. The strategy that we adopt at the moment is as follows. The user specifies a threshold $0 \leq \tau \leq 1$ and a vertex v is assigned to community c iff $\alpha_c(v) \geq \tau$. If no threshold is specified, we proceed as follows: let the largest component of the affinity vector for vertex v (which we want to assign to communities) be $\alpha_{\max}(v)$ and the smallest $\alpha_{\min}(v)$. Assign v to community c iff $\alpha_c(v) \geq \frac{1}{2}(\alpha_{\max}(v) + \alpha_{\min}(v))$. We do not take the mean affinity value since we expect a vertex to be in only a small number of communities (say, at most five), whereas the actual number of communitites may be large with the result that the affinities for most communities will be small. This large number of small affinity values will tend to lower the mean with the effect that we would place the vertex in more communities than it would actually be. The output of the classifier is written to a file.

num_vertices	*num_communities*		
v_0 (int)	$\alpha(0,1)$ (double)	$\ldots$	$\alpha(0,k)$ (double)
$\vdots$	$\vdots$	$\vdots$	$\vdots$
v_{n-1}	$\alpha(n-1,1)$	$\ldots$	$\alpha(n-1,k)$

Table 2: File format for node affinities

One of the quality measures that are used is the so-called *normalized mutual information*. This is defined as follows. Let X be a random variable taking on values $x_1, \ldots, x_r$. Then the entropy of X is $H(X)$ which is defined as:

$$H(X) = \sum_{i=1}^{r} p(x_i) \log_2 \frac{1}{p(x_i)},$$

where $p(x_i) = \mathbf{Pr}\left[X = x_i\right]$.

3 Plotting the Data

Our primary objects of experimentation are the graphs generated by the LFR benchmark. Now there are several parameters that the LFR benchmark software takes to generate the graphs, namely, the number of vertices, degree distribution parameter, size distribution of communities, average degree, mixing parameter. The other important parameters (not related to the LFR benchmark) are the fraction of seed nodes per community, the threshold used to assign communities.

We plan to plot the NMI value against the mixing parameter for several values of average degree. We do this computation for several seed node fractions. As a guideline, we may just want the number of vertices to be small, say, around 2000. Also we will put the two-community graph of hip-hop musicians generated by Felix. Basically, we should push for proof-of-concept rather than an industrial-strength software.

4 Derivation of Basic Equations

Since I keep on forgetting how the matrix equations are derived, I am putting them in this section for quick reference. Let the absorbing Markov chain that we are working with be called P. This may be written as:

$$P = \begin{bmatrix} Q & R \\ 0 & I \end{bmatrix}$$

Now the stationary state looks like:

$$P^{\infty} = \begin{bmatrix} 0 & (I-Q)^{-1}R \\ 0 & I \end{bmatrix}$$

Let $(I-Q)^{-1}R := X$, where X is an $(n-s) \times s$ matrix. The affinity of the non-seed nodes to a particular community, say c, is the sum of the columns of X weighted by the affinities of the seed nodes to the community c. Let $\alpha_c(j)$ denote the affinity of seed node j to community c. Then the affinities of the non-seed nodes to c is given by the column vector:

$$Y_c := \sum_{j=1}^{s} \alpha_c(j) \cdot X_{*j}.$$

Our goal is to compute Y_c for every community.

To do this, we first write Y_c differently.

$$\begin{aligned} Y_c := \sum_{j=1}^{s} \alpha_c(j) \cdot X_{*j} &= \sum_{j=1}^{s} \alpha_c(j) \cdot (I-Q)^{-1} R_{*j} \\ &= (I-Q)^{-1} \sum_{j=1}^{s} \alpha_c(j) \cdot R_{*j}. \end{aligned}$$

We can write the above equation as: $(I-Q)Y_c = \sum_{j=1}^{s} \alpha_c(j) \cdot R_{*j}$. Now $Q = D_1^{-1}A_1$, where D_1 is a diagonal matrix of order $n-s$ whose diagonal entries are the degrees of the non-seed vertices (in the whole graph). Matrix A_1 is the adjacency matrix of the graph *induced* by the non-seed nodes. We can now write:

$$(D_1 - A_1)Y_c = D_1 \cdot \sum_{j=1}^{s} \alpha_c(j) \cdot R_{*j}. \qquad (1)$$

Equation 1 defines a symmetrically diagonally dominant system of linear equations, one that can be solved in almost linear time. Note that the right hand side can be computed in $O(n)$ time, because R has only a linear in n number of entries.