4DV510

Weekly Exercise 3:

Community Detection with the Fast Newman Algorithm

Input:

The input graph should be read from an *edgelist* file. Each line describes one edge of the graph, with the *source* and the *target* node separated by a space.

Parameters:

- The *number of communities* you want to extract. By default, use 2. Note that the algorithm is originally hierarchical and can actually generate a full dendrogram of clusters, but in your case you can stop it when the desired number of communities is reached.

Output:

Two things should be printed in the output: (a) In the first line, print the final Q quality of the partitioned graph. This should be a single number with at least 10 decimal places. (b) In the second line, follow the same format you used for the 1st exercise (DBSCAN clustering) to output the contents of each of the obtained communities. Refer to nodes by their numerical label, as obtained from the input file (edge list).

- By default, write the results to the console. Each community's nodes should be in parenthesis, separated by commas, in ascending order. Communities should be separated by spaces, in ascending order of the first element. For example, for 12 nodes and 3 communities, you might output something like this:

```
(0,5,9,11) (1,2,8) (3,6)
```

Notes:

- As with previous assignments, follow the algorithm described in the original paper as closely as possible. No optimizations are necessary.
- I expect a self-contained, ready-to-run, OS-independent software package, with clear instructions on how to run it, preferably via console. The input file and the parameters should be provided, by default, as command-line parameters. If you need to do it in some other way, such as by using an input file for configuration, or by writing the parameters directly into the source code, please send explicit and clear instructions on how to do everything. Assume that the person running your program is totally unfamiliar with your project and must be directed step by step.

Summary of algorithm as described in the paper (Newman, 2004):

1. Start with every node in its own community.
2. Let e_{ij} be the **fraction** of edges in the network that connect vertices in community *i* to those in community *j*. If $i = j$, then e_{ii} is the fraction of edges that are *inside* the community. In the beginning (when every community has only a single node) this value is *0*, but as communities are joined later, it will increase.
3. Let $a_i = \sum_j e_{ij}$.
4. The quality of a specific partitioning of the graph is given by $Q = \sum_i (e_{ii} - a_i^2)$. This is a measurement of how far away from random is the current partitioning (the farther away, the better). We want to maximize Q so that we have the best possible partitioning. For that, we join two communities at a time (as is usual in hierarchical agglomerative algorithms).
5. The change in Q upon joining any two communities is given by $\Delta Q = 2(e_{ij} - a_i a_j)$.
6. Join the two communities that have the largest ΔQ.
7. Update the e_{ij}, a_i, and a_j for all *i* and *j*. Notice here that, since you joined two communities, there will be edges *inside* the community (e_{ii}). These are the edges that existed between the two communities that were joined in the previous step.
8. If the number of communities is still more than the desired, go back to step 5.

Reference:

Newman, M. E. (2004). *Fast algorithm for detecting community structure in networks.* Physical review E, 69(6), 066133.