Scientific Computation
Spring 2022
Project 4

Due: Friday April 1st 17:00 BST

There are five files for this assignment: 1) the project description (this file), 2) *project4.py*, a Python module which you will complete and submit on Blackboard (see below for details), 3) *report4.tex*, a template file for your short report which will also be submitted on Blackboard, and 4) *data4.npy* and 5) *label4.npy*, datafiles needed for the assignment.

In this assignment you will implement and apply the Walktrap (WT) method for partitioning a graph into communities. WT considers random walks on graphs and is based on the idea that random walkers tend to "cluster" near densely-connected nodes that can be thought of as a community.

Consider a partition of a graph into m communities with $m \leq N$. The communities will be labeled as integers, $0, 1, ..., m-1$, and each node in the graph will be assigned to one of these communities.

One step of a random walk on an N-node undirected graph is defined as follows. Say a walker is at node i. Then the probability that the walker moves to node j is A_{ij}/k_i. Here, A is the adjacency matrix for the graph so $A_{ij} = 1$ if nodes i and j are linked and is 0 otherwise; k_i is the degree of node i, the number of links connected to node i. The *transition matrix*, P is defined as $\mathrm{P} = \mathrm{D}^{-1}\mathrm{A}$ where D is a diagonal matrix with $D_{ii} = k_i$. You have been provided code which loads in the adjacency matrix for a graph and constructs P and other useful matrices.

Let $\mathrm{M} = \mathrm{P}^t$ where t is a positive integer. Then, M_{ij} is the probability that a walker moves from node i to node j over t steps. M plays a key role in the WT method and is used to define distances between nodes and between communities (and between nodes and communities).

1. (5 pts) The WT distance between nodes i and j, d_{ij}, is defined as,

$$d_{ij}^2 = \sum_{l=1}^{N} \frac{(M_{il} - M_{jl})^2}{k_l}.$$

   We see that if walkers i and j being similarly likely to move to a third node indicates thet i and j are "close". This equation can conveniently be stated in matrix-vector form as,

$$d_{ij}^2 = \|\mathrm{D}^{-1/2}\mathrm{m}_i - \mathrm{D}^{-1/2}\mathrm{m}_j\|^2, \tag{1}$$

where m_i^T is the i^{th} **row** in M, and $\|\cdot\|$ is the length of an N-element vector. Complete the function, *WTdist*, so that it efficiently constructs a numpy array corresponding to an N x N matrix, X where $X_{ij} = d_{ij}$. Make a plot comparing the WT distance and "conventional distance" (number of links on a shortest path) between node 0 and all other nodes in the graph. Include the figure and a 1-2 sentence description of the figure in your report.

2. (4 pts) We now consider how to define the distance between two communities. Let C_a be the set of nodes assigned to community a, and let $l_a = |C_a|$, the number of nodes in community a. The probability for a walker to go from community a to node j in t steps is, $R_{C_a,j} = \frac{1}{l_a}\sum_{i\in C_a} M_{ij}$. The squared distance between two communities a and b is defined as,

$$r_{ab}^2 = \sum_{l=1}^{N} \frac{(R_{al} - R_{bl})^2}{k_l}.$$

Complete the function, *WTdist2*, to compute the squared distance between each distinct pair of communities. The communities are defined via a list, $Clist$, and $Clist[a]$ is a list containing the nodes assigned to the a^{th} community. The distances should be stored in an m x m numpy array, Y, where $Y[a,b] = r_{ab}^2$. Note that initially, each node is assigned to its own community, so $Clist[a] = [a]$. Also note that WT only requires distances for 'linked-pairs' of communities: communities that have at least one link between them.

3. (8 pts) You will now implement the full WT algorithm. Each iteration, two communities are merged. The communities to be merged are selected based on a cost function. The cost of merging communities a and b is:

$$s_{ab} = \frac{1}{N}\frac{l_a l_b}{l_a + l_b} r_{ab}^2.$$

Large values of this cost correspond to large distances between nodes in the merged community. The algorithm then proceeds as follows: First, compute the cost of merging for all linked pairs of communities. Then, each iteration:

(a) merge the two linked communities with the lowest cost

(b) update the squared distances r_{ab}^2 and costs as needed

(c) repeat the two steps above until N_c communities remain where N_c is a specified parameter.

Say that community b is merged into community a so that the nodes in b are moved to a. Then, for any other community, j, the cost, s_{aj} can be updated with,

$$s_{aj}^{(new)} = \frac{(l_a + l_j)s_{aj} + (l_b + l_j)s_{bj} - l_j s_{ab}}{l_a + l_b + l_j},$$

provided that s_{ab}, s_{aj}, and s_{bj} had been computed prior to merging. Add code to the *main* function that efficient implements this algorithm. Some code for the initial cost calculation has been provided which you may use, modify, or discard. In practice, we retain the partitions generated each iteration, but here, the function

should just return a N_c-element list where each element is a list of nodes which belong to one of the N_c communities. Considerably more detail about this method can be found in this paper: `https://arxiv.org/abs/physics/0512106v1` though some of the notation is different. Provide a description of your implementation in your report, and include a careful explanation of the steps you have taken to make your code efficient.

4. (3 pts) The provided graph corresponds to the functional network of the human brain. The brain's hemispheres correspond to a natural partition into two communities and the corresponding labels for the graph nodes can be found in *label4.npy*. Apply the WT method to this network using $t = 6$ and 8 and with sensible values of N_c to assess the quality of the results. Add relevant code to analyzeWT, and provide a description of your findings in your report. For $t = 6$, make a figure illustrating the partition that has been computed, and add a 1-2 sentence description of the figure to your report.

Further guidance

1. You may add additional functions as needed.
2. You may use any Python modules we have used during the term. Please do not use other modules without permission.
3. This assignment is for M4/M5 (MSci/MSc) students only.

Submitting the assignment

To submit the assignment for assessment, go to the course blackboard page, and find the link for "Project 4" Click on the "Project 4", and then click on "Write Submission" and add the statement:"This is all my own unaided work unless stated otherwise" confirming the work as your own

Click on "Browse My Computer" and upload your final files (project4.py, report4.pdf). Finally, click "Submit".