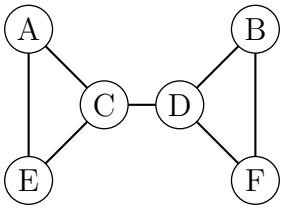
Math 495R Homework 22

A graph is a collection of points (called *vertices* or *nodes*) and a collection of paths (edges) connecting those points. Two nodes are said to be adjacent if there is an edge connecting them. Graphs often contain hidden structure. For instance, it is often interesting to identify communities of nodes which are closely connected to each other, but less closely connected to other nodes.



For instance, in the graph above the nodes A, C, and E might form one community and B, D, and F might form another. In larger graphs communities might be more difficult to identify.

Zachary's Karate club is a standard example of studying community detection. Zachary studied a university karate club of 34 members. He surveyed the students about their interactions and connections outside of class in order to build a graph of their relationships. Noticing tension between the club president and the instructor, Zachary analyzed the graph to predict whether each member was more loyal to the president or the instructor. When the club eventually split, Zachary's model accurately predicted which side of the divide all but one member ultimately ended up on.

In this lab you will write a program to detect communities using the *adjacency matrix* of a graph. If a graph has n nodes, then the adjacency matrix A of the graph is the $n \times n$ matrix with

$$A_{i,j} = \begin{cases} 1 & \text{if nodes } i \text{ and } j \text{ are adjacent} \\ 0 & \text{otherwise.} \end{cases}$$

For instance, the adjacency matrix for the graph above is

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

where we have ordered the nodes alphabetically. Note that the adjacency matrix (as defined here) is always symmetric.

The data file posted online contains adjacency matrices that encode the classes responses to the study network survey. Your program should identify study communities that exist within the class.

(1) Write a function `PrincipalEigenvector(A)` that takes for input an adjacency matrix A. This function should *balance* the adjacency matrix A (as described below) to obtain a matrix B and then return an eigenvector $\mathbf{v}$ corresponding to the greatest eigenvalue of B. This eigenvector will encode the first division into communities. This should be done as follows:

(a) Construct the matrix K of the same size of A whose (i,j)-th entry, $K_{i,j} = k_i \cdot k_j$ where k_i is the sum of entries in the ith row of A. The number k_i is called the *order* of the i-th vertex and counts how many edges come out from it

(b) Let m be the the sum of all the entries in the matrix (twice the number of vertices in the graph). Then compute

$$B = A - \frac{1}{m}K.$$

(c) Compute the eigenvalues and eigenvectors of B using the `numpy.linalg.eig` command. Recall that `eig(B)` returns a list of eigenvalues and a matrix whose **columns** (not rows) are the eigenvectors for B.

(d) Find the largest eigenvalue λ and return the eigenvector corresponding to λ.

If A is as above, `PrincipalEigenvector(A)` should return the vector

$$\mathbf{v} = [4.44..., -4.44..., 3.25..., -3.25..., 4.44..., -4.44...].$$

(2) Write a function `Split(A,L,v)` which takes for input an $n \times n$ adjacency matrix A, a list of labels L of length n which give the names of the vertices of the graph in the same ordering used in the matrix, and a vector $v \in \mathbb{R}^n$. The function should construct two matrices A_1, A_2 and two lists of labels L_1 and L_2 defined as follows:

- If the i-th entry of $\mathbf{v}$ is positive, then put the i-th row of A into A_1, and the i-th entry of L into L_1.
- If the entry is negative, then put the i-th row of A into A_2, and the i-th entry of L into L_2.
- Finally, remove the columns of A_1 corresponding to negative entries of $\mathbf{v}$ and the columns of A_2 correspond to positive entries.

Return A_1, A_2, L_1, and L_2.

If $L = [A, B, C, D, E, F]$ and A and $\mathbf{v}$ are as above, then $L_1 = [A, C, E]$, $L_2 = [B, D, F]$, and

$$A_1 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \quad \text{and} \quad A_2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

(3) Write a function `Communities(A,L)` which takes for input an $n \times n$ adjacency matrix A and a list of labels L of length n which give the names of the vertices of the graph in the same ordering used in the matrix. The function should recursively compute the community structure of the graph and return a list of nested lists which describes this structure. This should be accomplished as follows:

(a) Use the function from part 1 to compute the principal eigenvector $\mathbf{v}$.

(b) The base case for the recursion is when every entry of $\mathbf{v}$ is the same sign. In this case, return the list of labels as it was passed in.

(c) If the entries of $\mathbf{v}$ have mixed signs, then use the function from part 2 to split A and L into A_1, A_2, L_1 and L_2.

(d) Compute `S1=Communities(A1,L1)` and `S2=Communities(A2,L2)`.

(e) Return the list `[S1,S2]`.

The nested output shows how smaller communities are nested within larger communities. For instance an output of

$$[\ [\ [A, B], [C, D]\ ],\ [E, F, G, H]\ ]$$

would indicate that E, F, G, and H form a unified community, where as A, B, C, and D form a community that fractures into pairs.

Using A and L as above, your program should return

$$[\ [A, C, E], [B, D, F]\ ]$$