Homework 1

Pan Du

September 13, 2022

Problem 1: In order to let Katz centrality converge, the term $(\mathbf{I} - \alpha\mathbf{A})^{-1}$ must have value, which means $\mathbf{I} - \alpha\mathbf{A}$ is invertible. So we need $det(\mathbf{I} - \alpha\mathbf{A}) \neq 0$. Note that if $\mathbf{A}$ is a singular matrix, we have

$$det(\mathbf{I} - \alpha\mathbf{A}) = 0 \Rightarrow det(\mathbf{A} - \frac{1}{\alpha}\mathbf{I}) = 0 \Rightarrow \alpha = \frac{1}{\lambda}$$

where λ is the eigen value of matrix A. α takes the minimum value when λ is the largest eigen value. So to ensure $\mathbf{A}$ is not sigular, we should let α smaller than this minimum value, in other words:

$$\alpha < \frac{1}{\lambda_1}$$

Where λ_1 is the leading eigen value of matrix A.

Problem 2: Using the "walk" concept, you will start from node v_i and go two hops, then count the distinct paths that end in node v_j, which should be equal to the number of common neighbours. Mathematically: We have adjacency matrix $\mathbf{A}$, we want to check the common neighbourhood of node v_i and v_j. The number of common neighbour equalls to the element $\mathbf{A}^2_{ij}$ which can be calculated as:

$$|N(v_i) \cap N(v_j)| = [\mathbf{A}^2]_{ij} = \sum_k \mathbf{A}_{ik}\mathbf{A}_{kj}$$

Problem 3:

A. Code written in python and pushed to github

First, we calculate the Jaccard similarity of each pair of nodes.

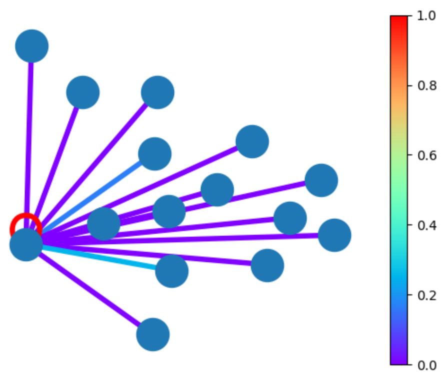
```
# calculate the Jaccard similarity between every pair of nodes
S = np.zeros((len(nodes),len(nodes)))
for i in range(len(nodes)):
    for j in range(len(nodes)):
        NinNj = np.array(A.todense()[:,i])*np.array(A.todense()[:,j]) # calculate the
                                                    intersection of N(vi) and N(vj)
        NiuNj = np.array(A.todense()[:,i])+np.array(A.todense()[:,j]) # calculate the
                                                    union of N(vi) and N(vj)
        S[i,j] = len(np.nonzero(NinNj)[0])/len(np.nonzero(NiuNj)[0]) # put into S
                                                   matrix
```

Then we create another graph where the edges will be node "Ginori" connecting to all the nodes in the graph. The value on the edges will be the corresponding similarity.

```
# creating a copy of original graph
G2 = nx.create_empty_copy(G, with_data=True)
old_edges = copy.deepcopy(G.edges())
new_edges, metric = [], []

# set new edges according to Similarity to node "Ginori"
for i in range(len(G2.nodes)):
    G2.add_edge('Ginori', list(nodes)[i])
    # print(f"({u}, {v}) -> {p:.8f}")
    new_edges.append(('Ginori', list(nodes)[i]))
    metric.append(S_dict[list(nodes)[i]])
```

B. This is the plot:


1.0
0.8
0.6
0.4
0.2
0.0