Homework 1

Your Name: Dong Ding

September 8, 2022

Problem 1: We have to make sure the matrix $I - \alpha A$ is invertible, otherwise the linear system has no solution.

$$det(I - \alpha A) = 0$$

is equivalent to

$$det(A - I/\alpha) = 0$$

We see that the largest value of $1/\alpha$ for which the determinant is zero is the the largest eigenvalue of matrix A. In practice α is often set close to the threshold 1/k1.

Problem 2: The number of walks of length 1 between Vi and Vj is denoted by Aij

Number of walks of size 2 from Vi to Vj that go through Vk is equivalent to

$$N_{ij}^{(2)} = \sum_{k=1}^{n} A_{ik}A_{kj} = [A^2]_{ij}$$

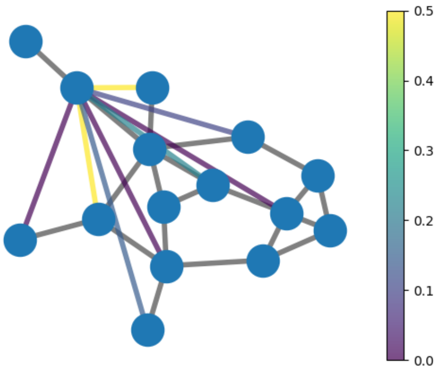
Problem 3: The idea is to define a function which calculate the Jaccard matrix by the definition, which is the common neighbors devided by their union neighbors.

Thus, we need to convert the graph into its adjacency matrix. From question 2, we can get the common neighbors by multiplying the adjacency matrix.

The union is their total neighbors minus the intersection. The code is shown below.

```python
"""
ACMS 80770-03: Deep Learning with Graphs
Instructor: Navid Shervani-Tabar
Fall 2022
University of Notre Dame

Homework 1: Programming assignment
"""

import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from networkx.algorithms import bipartite
from networkx.generators.random_graphs import erdos_renyi_graph
import copy


# -- Initialize graphs
seed = 30
G = nx.florentine_families_graph()
nodes = G.nodes()

layout = nx.spring_layout(G, seed=seed)

# -- compute jaccard's similarity
"""
    This example is using NetwrokX's native implementation to compute similarities.
    Write a code to compute Jaccard's similarity and replace with this function.
"""
def get_jaccard(G):
    node_list  = list(nodes)
    A = nx.adjacency_matrix(G)
    A= A.to_numpy_array()
    A_numer = np.matmul(A,A)
    A_deno = np.ones_like(A_numer)
    for i in range(len(A)):
        for j in range(len(A)):
            A_deno[i,j] = np.sum(A[i]) + np.sum(A[j]) - A_numer[i,j]

    Jaccard_matrix = A_numer/A_deno
    item = (node_list[i],node_list[j],Jaccard_matrix[i][j] for i in range(len(A)) for j in range(len(A)))

    return item

pred = get_jaccard(G)
```

```python
# -- keep a copy of edges in the graph
old_edges = copy.deepcopy(G.edges())

# -- add new edges representing similarities.
new_edges, metric = [], []
for u, v, p in pred:
    G.add_edge(u, v)
    print(f"({u}, {v}) -> {p:.8f}")
    new_edges.append((u, v))
    metric.append(p)

# -- plot Florentine Families graph
nx.draw_networkx_nodes(G, nodelist=nodes, label=nodes, pos=layout, node_size=600)
nx.draw_networkx_edges(G, edgelist=old_edges, pos=layout, edge_color='gray', width=4)

# -- plot edges representing similarity
"""
    This example is randomly plotting similarities between 8 pairs of nodes in the graph.
    Identify the "Ginori"
"""
ne = nx.draw_networkx_edges(G, edgelist=new_edges[:8], pos=layout, edge_color=np.asarray(metric[:8]), width=4, alpha=0.7)
plt.colorbar(ne)
plt.axis('off')
plt.show()
```



The code to access the Ginori family

```python
# -- plot edges representing similarity
"""
    This example is randomly plotting similarities between 8 pairs of nodes in the graph.
    Identify the "Ginori"
"""
for i in range(len(A)):
    if new_edges[i][0] == "Ginori":
        break


ne = nx.draw_networkx_edges(G, edgelist=new_edges[i:i+len(A)], pos=layout, edge_color=np.asarray(metric[:8]), width=4, alpha=0.7)
plt.colorbar(ne)
plt.axis('off')
plt.show()
```