network lab

May 4, 2022

[ ]:
```
# Initialize Otter
import otter
grader = otter.Notebook("network lab.ipynb")
```

[ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import networkx as nx
SEED = 3383
```

1 0. Data preparation

The following is a network of connections within part of the brain of a fly.

[ ]:
```
fly = nx.read_edgelist("bn-fly-drosophila_medulla_1.edges",nodetype=int)
n = fly.number_of_nodes()
print(n,"nodes")
```

This network has a feature that we have not allowed so far: some nodes link to themselves. These appear, for example, as nonzeros on the diagonal of the adjacency matrix.

[ ]:
```
A = nx.adjacency_matrix(fly).toarray()
print(sum(A[i,i] for i in range(n)),"self-linked nodes")
```

We will get rid of these self-links by zeroing out the diagonal and reconstructing the graph from the modified adjacency matrix. This will also relabel the nodes to go in order from 0 to $n-1$.

[ ]:
```
np.fill_diagonal(A,0)
fly = nx.from_numpy_array(A)
nodes = pd.Index(fly.nodes)
```

2 1. Basic info

2.1 1.1

Compute the average node degree of the network.

[ ]:
```python
kbar = ...

# No need to change this line.
print("average degree is",kbar)
```

[ ]:
```python
grader.check("info-kbar")
```

2.2 1.2

Draw the ego graph of node 80, with labels for the nodes.

[ ]:
```python

```

2.3 1.3

Create a list of all the nodes that are adjacent to node 9.

[ ]:
```python
nbrs9 = ...

# No need to change the following line.
print("neighbors of node 9:",nbrs9)
```

[ ]:
```python
grader.check("info-neighbors")
```

3 2. Clustering and distance

3.1 2.1

Create a Series for the local clustering coefficients in the network.

[ ]:
```python
cluster = ...

# No need to change the following lines.
print("clustering coeffs:")
print(cluster.describe())
```

[ ]:
```python
grader.check("cluster-values")
```

3.2 2.2

Compute $T(110)$, the number of triangles that have node 110 as a vertex. Then, apply the formula

$$\frac{2T(110)}{d_{110}(d_{110}-1)},$$

where d_{110} is the degree of node 110, to get the clustering coefficient of 110.

[ ]:
```python
# Set this to the number of triangles.
T110 = ...
```

```
# Set this to the local clustering coefficient.
C110 = ...

# No need to change the following line.
print(T110,"adjacent triangles, giving clustering coefficient",C110)
```

[ ]:
```
grader.check("cluster-110")
```

3.3 2.3

Let k be the integer closest to `kbar`. For each $q = 0, 0.02, 0.04, 0.06, \ldots$, compute and print out the average clustering coefficient of a Watts–Strogatz graph for n, k, and q, with random seed equal to `SEED`. When the average coefficient is smaller than the average clustering coefficient of the `fly` graph, stop.

[ ]:
```
# Exit this loop when the average WS clustering exceeds the average clustering␣
↪in the fly graph.
for q in np.arange(0,0.5,0.02):

# No need to change the following line.
print("final value of q is",q)
```

[ ]:
```
grader.check("cluster-WS")
```

3.4 2.4

Compute the mean shortest path length between all pairs of nodes in the largest connected component of the `fly` graph. (This computation might take 20-60 seconds, depending on the speed of the computer you use.)

[ ]:
```
avg_dist = ...

# No need to change the following line.
print("average shortest path length:",avg_dist)
```

[ ]:
```
grader.check("distance-average")
```

4 3. Degree distribution

4.1 3.1

Create a Series of the node degrees in the `fly` graph.

[ ]:
```
degrees = ...

# No need to change the following line.
print("node degrees for fly graph:\n",degrees.describe())
```

[ ]:
```
grader.check("degree-series")
```

4.2 3.2

Using log scales on both axes, plot a histogram of the degree distribution. You should see a rough linear trend in the resulting plot.

[ ]:
```

```

4.3 3.4

Create a Barabási–Albert graph with random seed `SEED`, the same number of nodes as the `fly` graph, and parameter m equal to the integer nearest to half the average degree of the `fly` graph. Find the standard deviation of its degree distribution.

[ ]:
```
# This should be assigned to a Graph object.
BA = ...

# This should get the value of the standard deviation of the degree␣
↪distribution.
BA_deg_stdev = ...

# No need to change the following lines.
print("BA graph has",len(BA),"nodes and",BA.number_of_edges(),"edges")
print("and node degrees with standard deviation",BA_deg_stdev)
```

[ ]:
```
grader.check("degree-BA")
```

5 4. Centrality

5.1 4.1

Create a data frame indexed by node and with columns named *degree*, *betweenness*, and *eigenvector* for centrality measures.

[ ]:
```
# This should be a data frame indexed by nodes and with 3 columns for␣
↪centrality measures. Fidn the mean value in each column.
centrality = ...
# These should be assigned to the means of the three columns.
dc_mean = ...
bc_mean = ...
ec_mean = ...

# No need to change the following lines.
print("centrality scores\n:",centrality)
print("with means",dc_mean,",",bc_mean,",",ec_mean)
```

[ ]:
```
grader.check("central-columns")
```

5.2 4.2

Of the three pairings degree-betweenness, degree-eigenvector, and betweenness-eigenvector, find the Pearson correlation coefficient of centrality scores for the pairing that is the *least* strongly correlated.

[ ]:
```
min_correlation = ...

# No need to change the following line.
print("weakest correlation is",min_correlation)
```

[ ]:
```
grader.check("central-corr")
```

5.3 4.3

For each type of centrality score, find the 10 highest-scoring nodes.

[ ]:
```
# Each should be an array, list, or Series.
top_degree = ...
top_betweenness = ...
top_eigenvector = ...

# No need to change these lines.
results = pd.DataFrame({"degree":top_degree,"betweenness":
↪top_betweenness,"eigenvector":top_eigenvector})
print(results)
```

[ ]:
```
grader.check("central-rank")
```

To double-check your work, the cell below will rerun all of the autograder tests.

[ ]:
```
grader.check_all()
```

5.4 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit.

Select *Kernel/Restart & Run All*, then save, then run this export cell again. Submit by pushing the resulting zip file to your GitHub assignment repo.

[ ]:
```
grader.export(pdf=False, force_save=True)
```