CSCI 347: Project 2: Exploring Graph Data

Choose a data set that you are interested in from one of the following sources:

SNAP collection: https://snap.stanford.edu/data
Network Repository: http://networkrepository.com/index.php

Run all the analysis in this project on the largest connected component of the graph.

Note that many of these datasets are quite large. If analyzing the data its taking too long, you may pre-process it by taking a sample of the graph first, and then extracting the largest connected component, to get the graph down to a manageable size.

Part 1: Think about the data

In a well-written paragraph, answer the following questions:

1. [3 points] Why are you interested in this data set?
2. [3 points] Clearly state if/how the data was pre-processed (Was the largest connected component extracted? Was a sample of vertices or edges taken? If so, describe the sampling process that was used.)
3. [6 points] **Before** doing any analysis, answer the following questions:
    1. [2 points] What characteristics do you expect the vertices with high centrality values to have and why? Specifically, think about non-graph characteristics. For example, we might expect highly central cities to have high populations or to house major industries in a graph where nodes represent cities and edges are roads between them.
    2. [2 points] Do you think that the degree distribution will exhibit a power law? Why or why not?
    3. [2 points] Do you think the graph will have the small-world property? Why or why not?

Part 2: Write functions for graph analysis in Python

Write the following functions in Python. You may assume that the input graph is simple — that it is undirected, unweighted, and has no parallel edges and no loops. Functions provided by networkx **can** be used within your code, as long as the function does not perform the same task as what you are being asked to implement (for example, you cannot use networkx's "betweenness_centrality" function within your own betweenness centrality function, but you can use networkx's functions for finding shortest paths). You may also assume that vertices are represented as integers (so the pair (1,3) indicates that there is an edge between vertex 1 and 3, for example).

4. [5 points] Number of vertices: A function that takes the following input: a list of edges representing a graph, where each edge is a pair. The output should be the number of vertices.

5. [5 points] Degree of a vertex: A function that takes the following input: a list of edges representing a graph, where each edge is a pair, and a vertex index that is an integer. The output should be the degree of the input vertex.

6. [5 points] Clustering coefficient of a vertex: A function that takes the following input: a list of edges representing a graph, where each edge is a pair, and a vertex index that is an integer. The output should be the clustering coefficient of the input vertex.

7. [5 points] Betweenness centrality of a vertex: A function that takes the following input: a list of edges representing a graph, where each edge is a pair, and a vertex index that is an integer. The output should be the betweenness centrality of the input vertex.

8. [5 points] Average shortest path length: A function that takes the following input: a list of edges representing a graph, where each edge is a pair. The output should be the average shortest path length of the graph.

9. [5 points] Adjacency matrix. A function that takes the following input: a list of edges representing a graph, where each edge is a pair. The output should be the dense adjacency matrix of the graph.

10. [5 points **EXTRA CREDIT**] Implement power iteration to find the eigenvector centrality of each node in a network: Write a function that takes as input an adjacency matrix, and outputs the eigenvector corresponding to the dominant eigenvector of that matrix (the eigenvector corresponding to the largest eigenvalue). The output can be used to view the eigenvector centrality of each vertex, when the input is a transposed adjacency matrix. This function must implement power iteration. You may **not** use linear algebra functions in numpy, spicy, or any other library to find eigenvectors/values, but you **may** use linear algebra functions for matrix-vector multiplication, computing the dot product, the norm. You may also use a function that implements *argmax.*

Part 3: Analyze the graph data

Report the following, using tables or figures as appropriate. You may treat the graph as an undirected, unweighted graph with no loops or parallel edges. You **may** use networkx functions for all of Part 3, but you are encouraged to test out your functions from Part 2 on real-world data.

- [5 points] Produce a visualization of the graph (or graph sample that you used).
- [3 points] Find the 10 nodes with the highest degree.
- [3 points] Find the 10 nodes with the highest betweenness centrality.
- [3 points] Find the 10 nodes with the highest clustering coefficient. If there are ties, choose 10 to report and explain how the 10 were chosen.
- [3 points] Find the top 10 nodes as ranked by eigenvector centrality
- [3 points] Find the top 10 nodes as ranked by Pagerank
- [3 points] Comment on the differences and similarities in questions 12-16. Are the highly ranked nodes mostly the same? Do you notice significant differences in the rankings? Why do you think this is the case?
- [3 points] Compute the average shortest path length in the graph. Based on your result, does the graph exhibit small-world behavior?
- [5 points] Plot the degree distribution of the graph on a log-log-scale. Does the graph exhibit power law behavior? Include the plot and the code used to generate it in your submission.
- [3 points **EXTRA CREDIT**] Create a log-log plot with the logarithm of node degree on the x-axis and the logarithm of the average clustering coefficient of nodes with that degree on the y-axis. Does the clustering coefficient exhibit power law behavior (is there a clustering effect)? Include the plot and the code used to generate it in your submission. μL

- Ensure that your code is well-commented to facilitate understanding of the methods used to generate your results.
- Your submission must include the complete written report and all associated code.
- The full names of all members should be included in the first page of the report as well as name of the file (it can include only last names).
- The team size must be between 2-4 members.
- Each team is required to submit their report and code only once per team, and the same team member should submit on both D2L and GradeScope platforms. Multiple submission by same person is allowed.
- The submission on GradeScope will be the primary document for grading purposes.
- To clarify individual contributions, please prefix your initials before each question (or part) to indicate who worked on each problem.
- Please be aware of the following points, as failure to comply will result in a deduction of points:
  - teams with fewer than 2 or more than 4 members.
  - Missing partners name on either the report or file name
  - Multiple submissions by the same team, even if identical.
  - Different versions of the report submitted by various team members
  - If your report does not include the necessary code, or if the code provided is not appropriately attached as a supplementary document or as inline code following each question.
  - Reports that are disorganized, unreadable, or contain excessive unrelated code
  - Missing initials for each question.