CSCI-4974/6971: Homework 2
<v1.0> updated February 22, 2024

Link Prediction and Centrality

Due Date: Friday 15 March 2024, 11:59pm via Submitty

For this assignment, we're going to analyze the connectivity and properties of several graphs. For background material and reference, use the notes from class and all of the associated reading materials. The datasets can be downloaded below:

- DNC emails: `http://cs.rpi.edu/~slotag/classes/SP24m/hw/out.dnc-temporalGraph.data`
- Congressional tweets: `http://cs.rpi.edu/~slotag/classes/SP24m/hw/congress.data`

We will be using Submitty for collecting homeworks. Upload a single *.py file that outputs responses for all of the below. There will be separate gradeables listed for the 4974/6971 sections. Pay careful attention to output formatting. Your code should be runable as a script on the command line (via `bash$ python3 hw02.py`). **You can use any NetworkX/NumPy/SciPy functionality you wish, but do not use any other external libraries**.

1. We're going to predict links on our good ol' friend the DNC temporal graph. Download the dataset `out.dnc-temporalGraph.data` and process it as we've done before in class. We'll use 25% of the edges to predict on the creation of future edges.

   - Read in the graph dataset.
   - Sort edges in temporal order.
   - Generate a new graph containing the first 25% of the links in the original graph.
   - Use this new graph for predicting future links.

   The below predictors will be compared. Refer to the notes and prior in-class code examples for how to implement these predictors.

   - Common neighbors.
   - Adamic/Adar.
   - Preferential Attachment.

- Strong triadic closure – To compute a weight for a triad triad (u, v, w), sum up the total number of edges between (u, v) and (v, w).
- **CSCI-6964 students:** Katz centrality. Use β value of $\beta = 0.0002$. You can compute it via explicit shortest paths or by using the linear algebraic formulation.
- **CSCI-6964 students:** Personalized PageRank – Note that although we worked with directed graphs, the same algorithm can easily be applied on undirected graphs. There's also a much faster iterative approach that you could choose to implement. If you just use the random walk model, perform at least 1000 walk iterations from each vertex. Set the probability that you jump back to the start at 20% per edge traversal.
- **Hint:** Be careful about how you create the temporal graph and add new edges. We want to calculate these predictions *only* using the link structure on our partial graph.

For each predictor, sort the values in decreasing order, and take the top 100. This will be our set of predicted links. We'll evaluate these sets for each predictor based on *precision* (true positives over total predictions) versus the full graph. Output the precision for each predictor.

2. Next we'll be looking at the roles that various centrality measures make on the connectivity of a given graph. We'll use a new graph, `congress.data`, for this purpose. This is a graph of tweet replies for members of congress. An edge exists, if a congressperson replied to the tweet of another congressperson. Download this graph and compute the following centrality measures:

   - Degree centrality.
   - Closeness centrality.
   - Betweenness centrality.
   - Eigenvector centrality.
   - PageRank centrality.

   Once all of these measures are computed for all vertices, sort the measurements in decreasing order. Then, create a new graph copy for each centrality measure. Iteratively delete the highest centrality vertex in each graph copy for each measure, while checking whether the graph remains connected. The final values you want to compare are, for each centrality measure, how many of the highest centrality nodes must be removed in order to disconnect the graph.