Assignment 3: Centrality, Modularity and Community Detection

Summer 2023

Overview

In module 3 we expanded upon the concept of centrality that we first introduced in lesson 1. We discussed several different metrics for centrality that help us determine what the “most important” nodes are under different circumstances.

- Understand centrality metrics
- Learn how to apply centrality metrics and core identification algorithms through case studies
- Understand the community detection problem and how to apply common detection algorithms
- Understand the LFR benchmark
- Understand Normalized Mutual Information

Submission

Please submit your Jupyter Notebook **A3-YOURLASTNAME.ipynb** with **requirements.txt** so that we may be able to replicate your Python dependencies to run your code as needed. **UNLIKE** previous assignments, there is no export_assignment function at the end and you do not need to submit anything outside of your completed notebook (with the cell outputs and images saved) and the uploaded images along with part 5.

With Anaconda, you can do this by running:
conda list -e > requirements.txt

Ensure all graphs and plots are properly labeled with unit labels and titles for x & y axes. Producing readable, interpretable graphics is part of the grade as it indicates understanding of the content – **there may be point deductions if plots are not properly labeled.**

Getting Started

The Networkx implementation of the Louvain community finding algorithm has been slightly modified as of Networkx v3.0. To ensure consistency, please make sure that you are using Networkx v3.0 or greater.

Part 1: Centrality [30 points]

In this assignment you will be calculating six different centrality metrics for two different data sets. The first data set is "**US_airports.txt**". It is an **undirected, weighted** network of flights among the 500 busiest commercial airports in the United States. The weights represent the number of seats available on the flights between a pair of airports.

The second dataset is "**Yeast.txt**" which represents a **directed, unweighted (weights all equal to 1)** transcription network of operons and their pairwise interactions via transcription factor-based regulation, within the yeast Saccharomyces cerevisiae. Import this network **as an undirected network.**

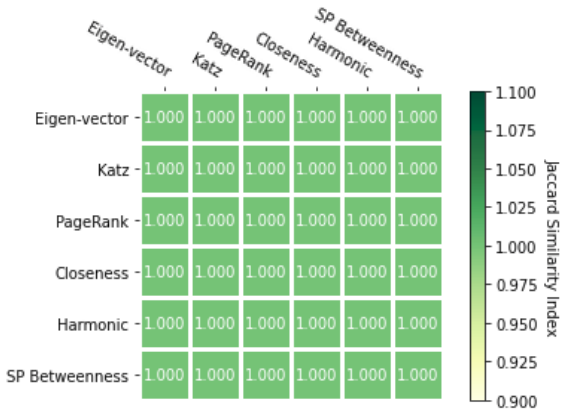
The six centrality metrics we will using for part one are the following:

- Eigenvector Centrality
- Katz Centrality *
- PageRank Centrality *
- Closeness Centrality **
- Harmonic Centrality **
- Shortest-paths Betweenness Centrality **

* *make sure the parameters lead to converged result (see tips at end the end of this section)*
** *convert the weighted airport graph to an unweighted graph for these metrics. The algorithms compute the weight as a distance which is not its value in this network.*

1. [2 points] Complete the load_graphs function to load both the airport and yeast networks from their data files as **undirected graphs**.

2. [12 points] Complete the top_10_nodes function to compute the top 10 nodes according to each of the centrality metrics listed above, using the ordering shown in the graphic to the right (Eigen-vector, Katz, PageRank etc.). E.g. compute each centrality metric for all nodes in each network, and then use that result to rank the top 10 nodes by each centrality metric. You will be plotting computations on top of these top 10 node sets in the next sections to create the heatmap shown in the graphic.



3. [14 points] Complete the calculate_similarity_matrix function to compare the top 10 nodes you obtained from each metric in part 1.1 using the Jaccard Similarity Index. You will need to implement your own Jaccard Similarity Index based on set-wise comparisons to compare the sets of top 10 nodes coming from different centrality

metrics - do not rely on existing packages because the similarity we are expecting is defined on sets, while some package implementations perform an element-wise operation.

Complete the plot_similarity_heatmap function to plot the similarity index using a heatmap of the data that has the following format, but populating it with your own index values. *Note: You must display the actual index values to receive full credit.*

4. [2 points] Based on the lecture [L6: The Notion of "Node Importance"], which centrality metric would you consider to be the most relevant for this network? Justify your answer. *Note: There are multiple metrics that can be used in different scenarios, it is sufficient to provide one scenario and explain why your chosen metric is relevant.* Write your answer in the markdown cell under section 1.4.

Tips for Part 1:

- Note that NetworkX requires alpha parameters for Katz centrality, which is reciprocal of lambda we discussed in the lecture. **PLEASE BE CAREFUL IN SETTING IT**. You may want to compare your results with Eigenvector centrality as a sanity check. Since these metrics are closely related, getting quite different results (under 80%) may indicate a problem with convergence.
- By default, the closeness centrality function of NetworkX performs a modified version of the computation presented in the lesson. Therefore, you need to pass **wf_improved=False** parameter to use the desired version.

Part 2: Community Detection with Zachary's Karate Club [25 points]

Now we will move from the topic of centrality to the topic of communities. You will find in your student file a graph of Zachary's Karate Club network imported from NetworkX along with a list of labels that represent the ground truth. You will then run each of the following community detection algorithms on the Zachary's Karate Club data set.

- C-finder (also called K-Clique)
- Greedy Modularity Maximization
- Louvain Algorithm for community detection

All of these algorithms are available via the NetworkX package in versions >=3.0.

1. [10 points] Complete the compute_cfinder_communities, compute_greedy_communities, and compute_louvain_communities functions such that each of these functions return a list of integers with length equal to the number of nodes, where each value in the list

should represent the community that node is assigned to. For any nodes that are assigned into multiple communities by the same algorithm, classify them into one additional community that represents nodes that belong to multiple communities. So for example, if one algorithm outputs 3 communities 1, 2 and 3, and a node belongs to two of them, then label that node with 4, which represents multiple community membership.

For C-finder, you will need to chose a value for the K parameter. A good value can be found by looking at the density of the graph itself (see L8: Critical Density Threshold in CFinder). Once you have chosen a value of K set it to be the default value for the k argument in compute_cfinder_communities function. You may not match the ground truth exactly, but tune the parameters to get as close a match as possible. Use the cell in section 2.3 to analyze the results for different values of K.

2. [12 points] Complete the plot_network_communities function to produce a plot for karate club network and a given set of community assignments. For example, one plot will be the karate club network as categorized into communities by the Louvain algorithm. Another will be the CFinder and another for Greedy. Please label all relevant parameters, such as k for CFinder. Use different colors for different communities. If one node belongs to more than one community, give it a different color and label it separately. Make sure you provide either a legend for your color choice.

3. [3 points] Look at the network visualizations for each of the algorithms above. Which algorithm performs the best? Which if any perform badly? Write your response in the markdown cell under section 2.3.

Part 3: Community Detection with LFR Networks [25 points]

Next you are going to generate an LFR benchmark graph using the code provided for you in your notebook.

1. [5 points] Complete the generate_network function to generate a graph using the LFR_benchmark_graph function and the parameters defined already. Extract the community assignments from this graph using the “community” field associated with each node and create a list of community assignments similar to those you created in part 2.

2. [10 points] Complete the normalized_mutual_information function to calculate the normalized mutual information of two different lists of community assignments. You will need to implement the NMI function yourself. You may use numpy and the standard library, but not any library function that performs the implementation for you. While you may not use any other libraries to implement your nmi function, you may compare against sci-kit learn normalized_mutual_info_score() function to test your results.

3. [5 points] Complete the sweep_mu_values function to calculate the NMI between the ground truth and the resulting community assignments provided by Modularity Maximization and Louvain, for 10 values of mu from 0.1 to 1 in 10 steps. You may use your functions from part 2.1 to generate the node level lists of community assignments.

4. [3 points] Complete the plot_nmi_values function to plot the NMI values as a function of mu for the two algorithms in the same plot. (1 plot) This means there will be two lines on the one plot, one line for NMI values as a function of mu for Modularity Maximization and one for Louvain.

5. [2 points] Look at the resulting plot from part 3.4. Describe the differences between the trend for the Greedy Modularity and Louvain algorithms. What does the NMI tell you about the communities? How does mu impact this? Write your answer in the markdown cell under section 3.5.

Part 4: Community Detection on Real World Data [15 points]

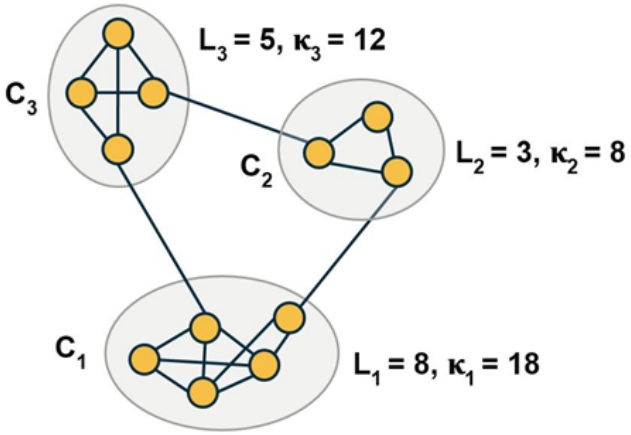
In this section you will run the community detection algorithms for "**US_airport.txt**" and "**yeast.txt**".

1. [5 points] Complete the calculate_community_sizes function to calculate the sizes of each community detected for both greedy modularity maximization and louvain for an input graph G which is passed in as a parameter to this function.

2. [5 points] Complete the plot_community_size_distributions functions to plot the community size distributions for size distributions given by the two algorithms for a single graph. E.g. your function should call the function calculate_community_size_distributions once for each algorithm (Modularity Maximization and Louvain) with the input graph G, and then plot the resulting community size distributions (one per algorithm) on a single plot. You will then call plot_community_size once per input network (once for airports.txt, once for yeast.txt) to obtain plots for each network.

3. [5 points] Look at the resulting community size distribution plots for both networks. Compare and contrast the distributions for both algorithms on each graph. What can you say about the distributions? Write your answer in the markdown cell under section 4.3.

Part 5: Modularity Calculations [5 points]



Answer the following food for thought question from Lesson 7 in a markdown box you created at the end of .ipynb file for Part 5*:*

Use the modularity formula derived in “L7: Modularity Metric- Derivation” to calculate the modularity of each of the following partitions on the above network:

1. All nodes are in the same community,
2. Each node is in a community by itself,
3. Each community includes nodes that are not connected with each other,
4. A partition in which there are no inter-community edges.

For each partition sketch a the graph above without any community assignments and draw your own communities that meet the requirements listed. **FOR EACH** partition

- Upload an image of the graph with your community assignments and embed the image into the notebook in the markdown section for part 5 (like this)
- Write down your calculation of the modularity for that given community assignment and add it along with the image into the markdown.

Note: You are not modifying the node or edge structure of the graph, just changing the way you are selecting the communities. There may be more than 1 correct solution that fulfills each criteria above - you only need to find 1 that meets the criteria for each part.