Assignment 5: Network Models and Statistical Analysis

Spring 2025

Overview

The objective of this assignment is to understand network generation and network statistical analysis. This assignment covers topics in lessons 12 and 13.

Submission

Please submit your Jupyter Notebook **A5-YOURGTUSERNAME.ipynb** with **requirements.txt** so that we may be able to replicate your Python dependencies to run your code as needed.

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

In this assignment you will use “**football.gml**” to answer parts 1 & 2. This network is the NCAA College Football Network. For the Hierarchical Random Graph in Part 2, you will be using the modified file “**football-hrg.gml**” created by the **PyHRG** library (**NOTE: You don’t need to use PyHRG for this section, the library is provided simply to show you how the graph object was generated).** For part 3 you will be using the “**slashdot.txt**” file. The graph loading code is already provided for you. You ***DO NOT*** need to alter it.

NCAA College Football Network

Part 1: Structural Properties of the Graph [25 points]

The goal for the first part of this assignment is to show the structural properties of the empirical network. Just like the previous assignment, the code to load the graph is already provided and you don’t need to make any modifications.

1. Complete the ***calculate_graph_statistics*** function to calculate the network diameter, characteristic path length (CPL), average clustering coefficient, transitivity, assortativity, and degree sequence (a list of all the degrees for each node). Return a dictionary of the results with the name of each statistic as the key.

2. Complete the ***sweep_louvain_resolutions*** function to identify community structures within the graph. Use the Louvain algorithm to find the best partition. Use 10 different resolution parameters from min_resolution to max_resolution (inclusive), and compare the result of each partition to the ground truth (you can find the ground truth in the value field associated with each node in the graph) using the normalized mutual information function. Return the list of resolutions and NMIs from the resulting community assignments.

   Additionally, complete the ***plot_nmi_vs_resolution*** function to display the NMI for each resolution as a line plot.

3. Complete the ***calculate_best_partition*** function to find the resolution associated with the partition that generates the highest NMI.

   Additionally, complete the ***plot_best_partition*** function to visualize the network with both the ground truth community assignments and the best partition assignments. Plot both networks side by side as subplots within a single figure. Make sure to label which graph is which (louvain vs ground truth).

4. Complete the ***calculate_inter_community_density*** function to calculate the intercommunity connection density matrix (see matrix P in L12: Generating Networks with Community Structure for the definition).

   Additionally, complete the ***plot_p_matrix*** function to plot the inter-community detection density matrix as a heatmap. Make sure to annotate the values within each cell and provide a legend.

5. Run the code in the 1.5 cell. How does the resolution impact the NMI? Is the partition for the best NMI a good match to the ground truth? Justify your answer based on the visual plot and the NMI value itself.

Part 2: Graph Generation [40 points]

In this section you will use different graph generators that you learned about in the lessons to generate graphs based on the statistics you calculated in part 1.

1. Complete the ***generate_configuration_graphs*** and ***generate_sbm_graphs*** functions to generate n_graphs=100 graphs using the Configuration Model and Stochastic Block

model graph generator in NetworkX (with the **configuration_model** function, use the **create_using=nx.Graph()** argument to get a Graph and not MultiGraph).

2. Here you will be working with the Hierarchical Random Graphs. We have composed a dendrogram fitted on the empirical network which you can find in "**football-hrg.gml**", using **PyHRG**. The dendrogram is formulated as a directed graph. Each leaf node (node with no outgoing edges) in the dendrogram represents a node in the empirical network. Each non-leaf node stores the information about its left/right child ("L" / "R") and the probability of leaf nodes in the left tree connecting to the leaf nodes in the right tree ("p"), as node attributes.

   Complete the ***calculate_edge_probability*** function to create a dictionary of edge probabilities between all pairs of leaf nodes.

   Then, complete the ***generate_graph_from_probs*** function to generate a networkx graph based on the edge probabilities calculated in the previous part. Specifically, if two nodes *i* and *j* have probability *p* of having an edge, generate an edge randomly with probability *p* for each pair of nodes.

   Finally, complete the ***generate_hrg_graphs*** function to generate n_graphs=100 HRG graphs using these edge probabilities and the generate function you just completed above. This should function very similarly to the functions in part 2.1, but you will call your own ***generate_graph_from_probs*** inside the loop instead of calling a networkx generator.

3. Complete the ***calculate_generated_statistic*** function to calculate the network diameter, CPL, average clustering coefficient, transitivity, and assortativity for each graph in a list of graphs. Format the return as a dictionary where the key is a string describing the property and the value is a list of the result for each of the graphs in the list. Note if any of your graphs are not connected, you may use the largest connected component.

   Next, complete the ***compare_generated_to_ground_truth*** function to compare the diameter, CPL, average clustering coefficient, transitivity, and assortativity for the empirical network with the sampled values from each of the graph models using a one-sample t-test.

   Additionally, complete the ***plot_graph_statistics*** function to visualize the distribution of each individual network property (e.g. CPL, average clustering) as a boxplot. Create the plot as a single plot with a series of five side-by-side boxplots (one for each property) as subplots. Label the overall plot with the generator name and label each individual box plot with the property being shown along the y axis.

4. How do the statistics compare against the three different types of graph generators? Are there any particular measures that are very different? Can you explain how the algorithm for each generator influences the statistics that are different? Which graph generator

would you say is the best fit for the ground truth? Justify your answer based on the t-test results and the box plots.

Slashdot Network

Part 3: Sampling [35 points]

In this part, we will be working with the Slashdot social network. The file “**slashdot.txt**” stores the list of edges in this network.

1. Complete the ***estimate_network_size*** function to implement the capture-recapture estimation method to compute the number of nodes in the network by randomly choosing samples of 500, 1000, 2000, 5000, and 10000 nodes each time. Repeat the experiment 1000 times (i.e. 1000 trials for sample size 500, 1000 trials for sample size 1000 and so on). Store your results in a dictionary where the key is the sample size and the value is a list of the estimated network sizes for each trial. *If the size of the n3 set is 0, you can skip that iteration.*

   Additionally, complete the ***plot_estimate_histogram*** function to plot a histogram of the estimated number of nodes in the network (the y-axis is the frequency of occurrence and the x-axis is the estimated number of nodes) for all the trials. Additionally, plot the true_size as a vertical line on the histogram.

   Finally, complete the ***plot_sample_size_error*** function to plot a line plot of the estimated number of nodes along with error bars representing the standard deviation for each sample size. The line plot value will be the mean over the 1000 trials for a given sample size and the error bars the standard deviation for that sample size. Along with this plot the true_size as a horizontal line. *Use 1 plot for all estimates, even across different sample sizes.*

2. Complete the ***estimate_edges function*** to estimate the number of edges using the induced subgraph sampling and Horvitz-Thompson estimator by sampling 5,000 nodes in the network. Repeat this n_iter=100 times and save the resulting edge count estimates in a list.

   Additionally, complete the ***plot_edge_estimate_distribution*** function to plot a histogram of the estimates of the number of edges along with the ground truth (the y-axis is the frequency of occurrence and the x-axis is the estimated number of edges).

3. Run the code in cell 3.3. How does the estimate of the number of nodes change as you increase the sample size? What is the smallest sample size you would consider a good choice for estimating the number of nodes?