Lab activity: analysis of music networks.

1 Part 3: Data analysis

1.1 Introduction

In this third session of the practice, we will start analyzing the information that we have been collecting and preprocessing in the previous sessions. This part of the practice should also serve to validate, to some extent, the work done in the other two sessions. In the analysis you carry out in this part, it is important to be critical enough to evaluate whether the strategies and implementations made in the first part of the practice are good enough, and if necessary, consider possible modifications.

1.2 Data Analysis

In this third session of the practice, we will use the artist graphs obtained in the first session and preprocessed in the second session. The statement for this third session includes a file where the functions to be implemented for each activity are specified (the file `Lab_AGiCI_202324_P3_skeleton.py`).

1. (**1 point**) Implement the function `num_common_nodes` with the following header:

```
def num_common_nodes(*arg):
    """
    Return the number of common nodes between a set of graphs.

    :param arg: (an undetermined number of) networkx graphs.
    :return: an integer, number of common nodes.
    """
```

   The function will receive an undertermined number of network graphs as input. The result of the function will be an integer counting the number of nodes that appear in all of these graphs.

2. (**1 point**) Implement the function `get_degree_distribution` with the following header:

```
def get_degree_distribution(g: nx.Graph) -> dict:
    """
    Get the degree distribution of the graph.

    :param g: networkx graph.
    :return: dictionary with degree distribution (keys are degrees, values
    are number of occurrences).
    """
```

The function will receive a graph and will return its degree distribution as a dictionary (keys will be the degrees and values will be the number of nodes with that specific degree).

3. (**1 point**) Implement the function `get_k_most_central` with the following header:

```
def get_k_most_central(g: nx.Graph, metric: str, num_nodes: int) -> list:
    """
    Get the k most central nodes in the graph.

    :param g: networkx graph.
    :param metric: centrality metric. Can be (at least) 'degree', '
    betweenness', 'closeness' or 'eigenvector'.
    :param num_nodes: number of nodes to return.
    :return: list with the top num_nodes nodes with the specified
    centrality.
    """
```

The function will receive a graph, a centrality metric name, and the number of nodes to return. The function will return the k most central nodes according to the specified metric. The function should allow calculating, at least, degree centrality, betweenness centrality, closeness centrality, and eigenvector centrality.

4. (**1 point**) Implement the function `find_cliques` with the following header:

```
def find_cliques(g: nx.Graph, min_size_clique: int) -> tuple:
    """
    Find cliques in the graph g with size at least min_size_clique.

    :param g: networkx graph.
    :param min_size_clique: minimum size of the cliques to find.
    :return: two-element tuple, list of cliques (each clique is a list of
    nodes) and list of nodes in any of the cliques.
    """
```

The function will search for cliques in a graph. The function will receive a graph as input, as well as the parameter `min_size_clique`, and will return two lists. The first list will contain each of the cliques with a size greater than or equal to `min_size_clique`. The second list will include all the different nodes that are part of any of these cliques.

5. (**1.25 points**) Implement the function `detect_communities` with the following header:

```
def detect_communities(g: nx.Graph, method: str) -> tuple:
    """
    Detect communities in the graph g using the specified method.

    :param g: a networkx graph.
    :param method: string with the name of the method to use. Can be (at
    least) 'givarn-newman' or 'louvain'.
    :return: two-element tuple, list of communities (each community is a
    list of nodes) and modularity of the partition.
    """
```

The function will receive a graph as input and will return a list of lists with the nodes grouped into communities, along with the modularity of the partitioning. The function can receive additional parameters that allow configuring the community detection (e.g., the specific algorithm to implement or the randomness seed if it is not deterministic).

1.3 Questions to be addressed in the report

1. (**0.75 points**) Indicate the number of nodes shared by each pair of graphs (g_B, g_D, g'_B, g'_D, g^w_B and g^w_D) and all the graphs together. Use the function `num_common_nodes`.

   (a) How many nodes are shared between g_B and g_D? What information does this tell us about the importance of the algorithm used by the crawler (i.e. the scheduler) to decide next nodes to crawl?

   (b) How many nodes are shared between g_B and g'_B? What information does this tell us about the reciprocity of g_B? And about the Spotify's artist related algorithm?

   (c) How many nodes are shared between g'_B and g^w_B? What information does this tell us about the similarity metric used to construct g^w_B?

2. (**0.5 points**) Calculate the 25 most central nodes in the graph g'_B using both degree centrality and betweenness centrality. How many nodes are there in common between the two sets? Explain what information this gives us about the analyzed graph.

3. (**0.5 points**) Find cliques of size greater than or equal to `min_size_clique` in the graphs g'_B and g'_D. The value of the variable `min_size_clique` will depend on the graph. Choose the maximum value that generates at least 2 cliques. Indicate the value you chose for `min_size_clique` and the total number of cliques you found for each size. Calculate and indicate the total number of different nodes that are part of all these cliques and compare the results from the two graphs.

4. (**0.5 points**) Choose one of the cliques with the maximum size and analyze the artists that are part of it. Try to find some characteristic that defines these artists and explain it.

5. (**0.5 points**) Detect communities in the graph g_D. Explain which algorithm and parameters you used, and what is the modularity of the obtained partitioning. Do you consider the partitioning to be good?

6. (**1 point**) Suppose that Spotify recommends artists based on the graphs obtained by the crawler (g_B or g_D). While a user is listening to a song by an artist, the player will randomly select a recommended artist (from the successors of the currently listened artist in the graph) and add a song by that artist to the playback queue.

   (a) Suppose you want to launch an advertising campaign through Spotify. Spotify allows playing advertisements when listening to music by a specific artist. To do this, you have to pay 100 euros for each artist to which you want to add ads. What is the minimum cost you have to pay to ensure that a user who listens to music infinitely will hear your ad at some point? The user can start listening to music by any artist (belonging to the obtained graphs). Provide the costs for the graphs g_B and g_D, and justify your answer.

   (b) Suppose you only have 400 euros for advertising. Which selection of artists ensures a better spread of your ad? Indicate the selected artists and explain the reason for the selection for the graphs g_B and g_D.

7. (**1 point**) Consider a recommendation model similar to the previous one, in which the player shows the user a set of other artists (defined by the successors of the

currently listened artist in the graph), and the user can choose which artist to listen to from that set. Assume that users are familiar with the recommendation graph, and in this case, the g_B graph is always used.

(a) If you start by listening to the artist *Taylor Swift* and your favorite artist is *Hippo Campus*, how many hops will you need at minimum to reach it? Give an example of the artists you would have to listen to in order to reach it.

1.4 Evaluation and expected results for this part of the practice

This third session of the practice will account for 25% of the total grade for the practice.

Remember that it is important for **the Python code to include sufficient comments** to understand its functionality and to **respect the function headers** provided.

The files to be obtained in this part of the practice for the final submission are as follows:

- `Lab_AGiCI_202324_P3_skeleton.py`: a plain python file, where each of the functions defined in this document must be implemented.
- `Lab_AGiCI_202324_P3_report.pdf`: A PDF file where all the questions raised in this statement have been answered in a detailed manner. The answers to the questions should be numbered and ordered.