Lab activity: analysis of music networks.

1 Part 4: Data visualization

1.1 Introduction

In this fourth part of the practice, we will work on data visualization.

On one hand, we will use Python libraries to visualize graph properties and also to visualize comparisons between artists. On the other hand, we will use the software Gephi[1] to visualize the obtained graphs.

It is important to adjust the code in order to generate good data visualizations. You should try to generate visualizations that convey information. Make sure that all the graphs you generate are well labeled (add labels to the axes, titles to the graphs, legends when necessary, etc.) and that all the texts can be read correctly (adjust the font size, the figure size, etc. if necessary).

1.2 Data Visualization from Python

In this fourth session of the practice, we will use both artist graphs and the song dataset that we have been working with in the previous sessions. With the statement of this fourth session, a file is attached where you have the functions to be implemented in each activity (the file `Lab_AGX_202324_P4_skeleton.py`).

1. (**1 point**) Program the function `plot_degree_distribution` with the following header:

```python
def plot_degree_distribution(degree_dict: dict, normalized: bool = False, loglog: bool = False) -> None:
    """
    Plot degree distribution from dictionary of degree counts.

    :param degree_dict: dictionary of degree counts (keys are degrees, values are occurrences).
    :param normalized: boolean indicating whether to plot absolute counts or probabilities.
    :param loglog: boolean indicating whether to plot in log-log scale.
    """
```

The function will receive a dictionary (as obtained from the function from the previous session `get_degree_distribution`), where the keys will be the degrees and the values will be the number of nodes with that specific degree, and it will display

[1] Remember that you have this software installed since the beginning of the course when we covered graph visualization topics. In any case, you can find it at https://gephi.org/

a graph with this distribution. The parameter `normalized` will indicate whether to normalize the number of nodes, and the parameter `loglog` will indicate whether to display the graph in logarithmic or linear scale.

2. (**1.25 points**) Program the function `plot_audio_features` with the following header:

```python
def plot_audio_features(artists_audio_feat: pd.DataFrame, artist1_id: str, artist2_id: str) -> None:
    """
    Plot a (single) figure with a plot of mean audio features of two different artists.

    :param artists_audio_feat: dataframe with mean audio features of artists.
    :param artist1_id: string with id of artist 1.
    :param artist2_id: string with id of artist 2.
    :return: None
    """
```

The function will receive a dataframe of artists with their average audio features (as obtained from the function `compute_mean_audio_features`) and two artist identifiers, and it will generate a single figure that allows comparing the audio features of the two artists.

You can choose the type of graph that you consider most suitable for this comparison (but you should be able to compare the two artists in a single figure). Below are two possible examples of graphs that would allow comparing artists:

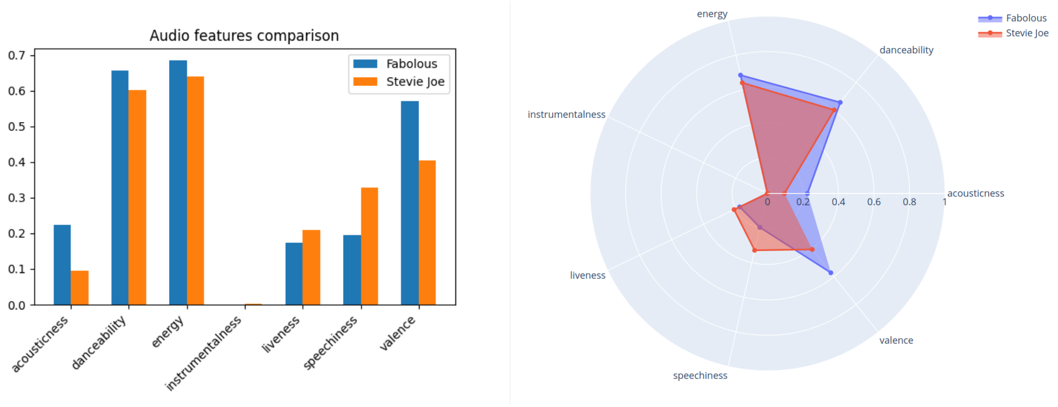

Figure 1: Comparison of the audio features of two artists using different graphs.

3. (**1 point**) Program the function `plot_similarity_heatmap` with the following header:

```python
def plot_similarity_heatmap(artist_audio_features_df: pd.DataFrame, similarity: str, out_filename: str = None) -> None:
    """
    Plot a heatmap of the similarity between artists.

    :param artist_audio_features_df: dataframe with mean audio features of artists.
    :param similarity: string with similarity measure to use.
    :param out_filename: name of the file to save the plot. If None, the plot is not saved.
    """
```

The function will receive a dataframe of artists with their average audio features (as obtained from the function `compute_mean_audio_features`) and a measure of similarity between artists. The function will generate and display a heatmap with the similarity values for each pair of artists in the dataframe, based on the average audio features. Remember to include a legend that allows interpreting the colors of the heatmap.

4. (**1.25 point**) Using the previous functions:

   (a) Generate the plots of the degree distribution of the graphs g'_B, g'_D, and g^w. Configure the `normalized` and `loglog` parameters to generate the best possible visualization according to your criteria.

   (b) Select the artist that is most similar to *Taylor Switft* from g^w and generate a comparison using the `plot_audio_features` function.

   (c) Select the artist that is less similar to *Taylor Switft* from g^w and generate a comparison using the `plot_audio_features` function.

   (d) Generate a heatmap showing the similarity between all artists in your dataset using the `plot_similarity_heatmap` function.

   (e) Generate a plot showing the number of nodes in the biggest connected component in the similarity graph g^w as the low weight edges are pruned (use the `prune_low_weight_edges` function).

1.3 Questions to answer in the report

1. (**1.5 points**) Comment on the results obtained in Exercise 4 (**it is compulsory to include the results obtained in Exercise 4 in the report**):

   (a) What are the degree distributions of the three obtained undirected graphs like?

   (b) What can you infer from the similarity heatmap regarding the algorithm that selects related artists on Spotify?

   (c) Is there any relationship between the similarity of artists obtained from their audio features and the distances of the artists in the directed graphs? For instance, consider *Taylor Swift* and her most and least similar artists as determined in exercises 4.b and 4.c.

   (d) At which percentile would you prune the edges of the weighted similarity graph g^w to ensure the size of the largest connected component is preserved while minimizing the amount of edges in the graph?

For each of the directed graphs, g_D and g_B, obtained in the first session of the practice:

2. (**1.5 points**) Generate a visualization of the graph using Gephi that assigns a color to the nodes based on the community they belong to and sizes the nodes proportionally to their betweenness centrality. Use a layout algorithm that allows for easy identification of the communities. Show the names of the most important artists (highest betweenness centrality) in each community.

3. (**1 point**) Generate a visualization of the graph using Gephi while maintaining the same node positioning as the previous exercise's graph, but now assigning node size based on their number of followers and node color based on the distance of each

node from the initial node of the crawler (the node representing the artist *Taylor Swift*). Highlight the two artists selected for the plot audio features comparison (the less and most similar artists to *Taylor Swift*).

4. (**1.5 points**) Comment on the visualizations generated with Gephi.

    - Compare graphs g_B and g_D. What can you say about their properties?
    - Can you identify common characteristics among artists belonging to the same community? Could you label the different communities?

1.4 Evaluation and expected results for this part of the practice

This fourth 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 functioning and to **respect the headers** of the provided functions.

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

- `Lab_AGX_202324_P4_skeleton.py`: A plain python file where each of the functions defined in this document has been implemented.
- `Lab_AGX_202324_report.pdf`: the final report (a pdf file) must have a section called **Part 4: data visualization** where all the questions asked in this document will be answered in detail. The answers to the questions should be numbered and in order.