Assignment-2

Y Rithvik

September 6, 2023

1 Implementation Summary

1.1 Importing Facebook Data

Implemented the following in $import\_facebook\_data$ function

- The function takes a path to a file as input.
- The function reads the data from the file and converts it into a NumPy array.
- The function sorts the data by each row(since it is an undirected graph, this makes edge (x,y) and (y,x) the same).
- The function returns the unique values of the data(this removes duplicate edges if present).

1.2 Importing Bitcoin Data

Implemented the following in the $import_bitcoin_data$ function

- The function reads the data from the CSV file using pandas and drops the last two columns (edge weights and time stamp columns).
- The above step converts the graph to an unweighted, undirected graph.
- Converted this to an undirected graph by sorting each row and then removing duplicates using np.unique.

1.3 Spectral Decomposition One Iteration

Implemented the following in $spectralDecomp\_OneIter$ function

- The function takes the node connectivity list as input, and computes the unique node id's present in this using np.unique function
- Mapped the node IDs to 0 to the number of nodes for computing the adjacency matrix.
- Computed the adjacency matrix using the mapped node IDs and node connectivity list.
- Computed Laplacian matrix(L) and Degree matrix(D) using the adjacency matrix.
- Computed Fiedler vector by solving the generalized eigenvalue problem ($Lx = \lambda Dx$) using scipy.linalg.eigh function.
- Computed the graph partition using the sign of Fiedler vector values and mapping them to node IDs.

1.4 SpectralDecomposition

Implemented the following in *SpectralDecomposition* function

- This function calls the recursive function *spectralDecomp*.
- *spectralDecomp* function is a recursive function that calls *spectralDecomp_OneIter* and passes the node connectivity list to it.
- Using the returned graph partition from the previous function call, the node connectivity list is split into two disjoint sets.
- Then two recursive calls are made to the *SpectralDecomposition* function in which the two node connectivity lists (obtained in the previous step) are passed.
- From the above step we obtain two graph partitions which are then concatenated and returned.
- Implemented stopping criteria by checking if the fiedler vector is stable(further details regarding this is mentioned later in the report).

1.5 Create Sorted Adjacency Matrix

Implemented the following in *createSortedAdjMat* function

- This function creates and plots the adjacency matrix in which nodes are sorted according to the community IDs obtained from graph partition obtained from spectral decomposition.
- First we get sorted indexes of the second column of the graph partition which contains the community id's.
- Then the first column of the graph partition which contains the node IDs is arranged according to the sorted indexes obtained in the previous step.
- Obtained the sorted adjacency matrix, by rearranging the rows and columns of the adjacency matrix according to the sorted node IDs obtained in the previous step(use np.take function for this)
- Also plotted the adjacency matrix and visualized the contained communities using networkx library.

1.6 Louvian Algorithm

Implemented the following in the $louvain_one_iter$ function

- Implemented the louvian algorithm in this function.
- In this function we traverse through every node, compute deltaQmerge and deltaQdemerge, and sum them up to obtain deltaQ for all unique neighboring communities of the node.
- Computed the max of all the deltaQ and if it is positive, then the node is assigned to that community.
- The above two steps are repeated till none of the nodes are reassigned to a different community.
- Computed the graph partition from the nodes and their assigned communities and returned that.

2 Results

2.1 Question 1

Plots for sorted fiedler vector, sorted adjacency matrix and graph partition for spectral decomposition one iterations.

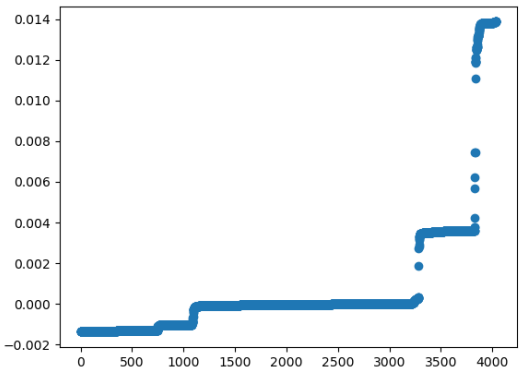

Figure 1: Sorted Fiedler Vector- Facebook Dataset

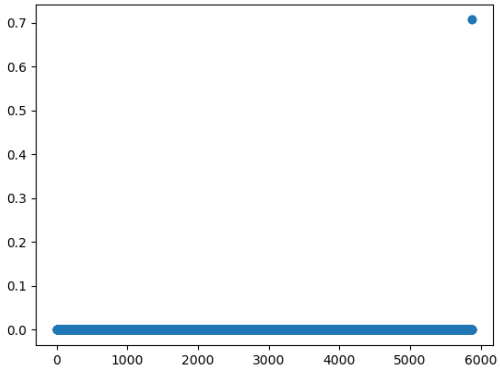

Figure 2: Sorted Fiedler Vector- Bitcoin Dataset

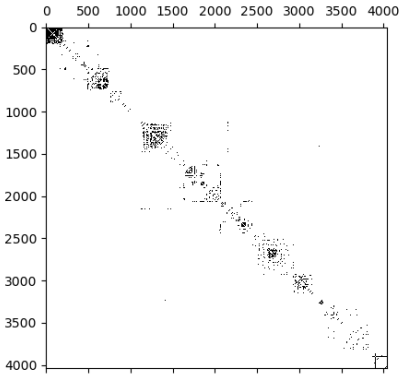

Figure 3: Sorted Adjacency Matrix - Facebook Dataset

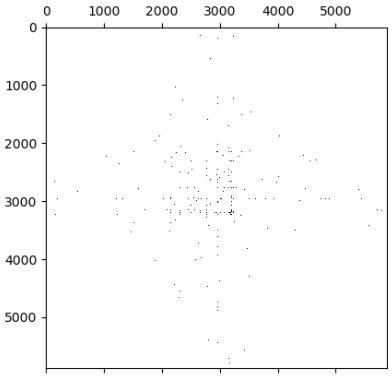

Figure 4: Sorted Adjacency Matrix- Bitcoin Dataset

Figure 5: Graph Visualization - Facebook Dataset

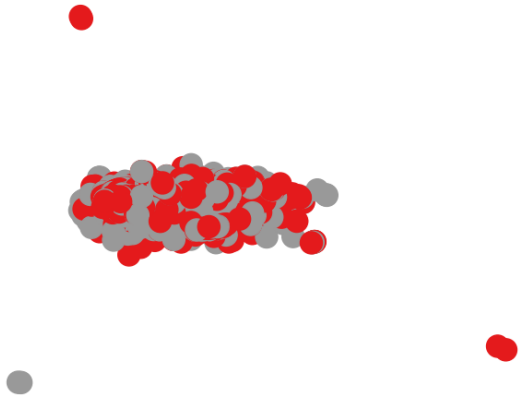

Figure 6: Graph Visualization- Bitcoin Dataset

2.2 Question 2

Stopping Criteria Explanation:

- The stopping criteria in the function `spectralDecomposition()` is based on the concept of stability. An unstable Fiedler vector means that the graph cannot be decomposed into two subgraphs with significantly different centralities.
- The stability of the fiedler vector is defined by the continuity of the values of the sorted fiedler vector, if the sorted fiedler vector is continuous then the fiedler vector is not stable and the corresponding partition would be unstable.
- The function `isStable()` checks if the Fiedler vector is stable by looking at the maximum difference between consecutive values in the sorted Fiedler vector. If the maximum difference is less than a scalar(k) multiple of the mean of the differences of the sorted Fiedler vector then the fiedler vector is unstable and we do not partition the graph any more.
- k is a hyperparameter which controls the number of partitions, as k increases number of partitions typically decreases.
- In general, a lower k is better for finding smaller communities, while a higher k is better for finding larger communities.

Description of the Algorithm:

- The algorithm starts by computing the Fiedler vector of the graph.
- The algorithm then checks if the Fiedler vector is stable. If it is not stable, then the algorithm stops.
- If the Fiedler vector is stable, then the algorithm splits the graph into two subgraphs based on the Fiedler vector.
- The algorithm recursively calls itself on the two subgraphs.
- This process continues until the Fiedler vector is stable or until there are no more subgraphs to decompose.

The stopping criteria ensures that the algorithm does not continue to split the graph into smaller and smaller subgraphs. This can help to prevent the algorithm from overfitting the data.
Note: k value used in the code is **200**

2.3 Question 3 (Spectral Decomposition)

Plot of the associated adjacency matrix sorted by associated sorted sub graph Fiedler vectors and graph visualizations for spectral decomposition.

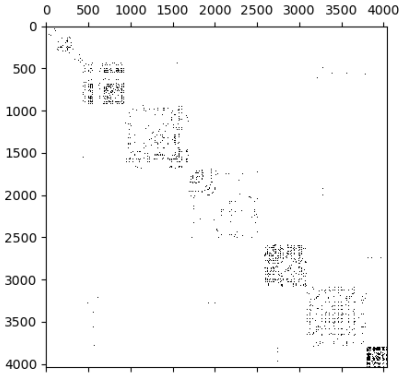

Figure 7: Sorted Adjacency Matrix - Facebook Dataset

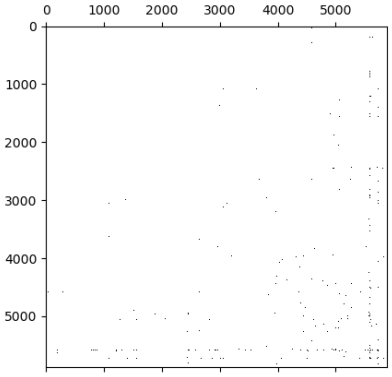

Figure 8: Sorted Adjacency Matrix- Bitcoin Dataset

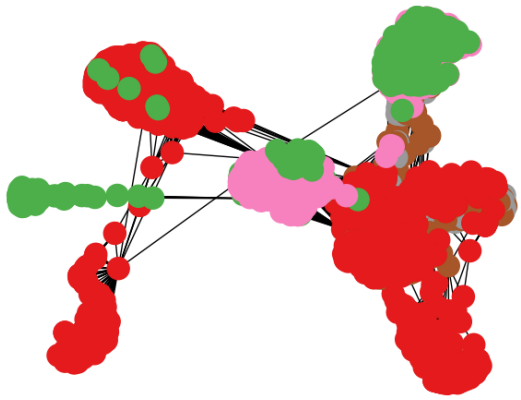

Figure 9: Graph Visualization - Facebook Dataset

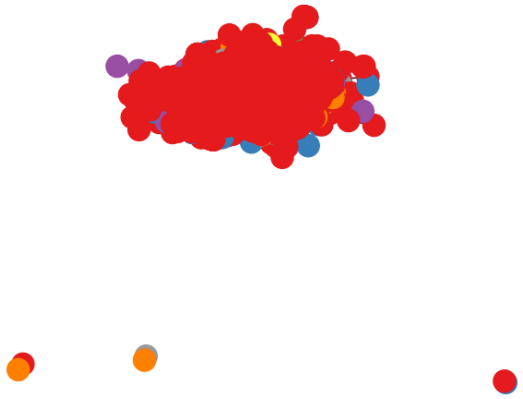

Figure 10: Graph Visualization- Bitcoin Dataset

2.4 Question 4 Louvain algorithm

Plot of the associated adjacency matrix sorted by associated sorted sub graph Fiedler vectors and graph visualizations for Louvain algorithm.

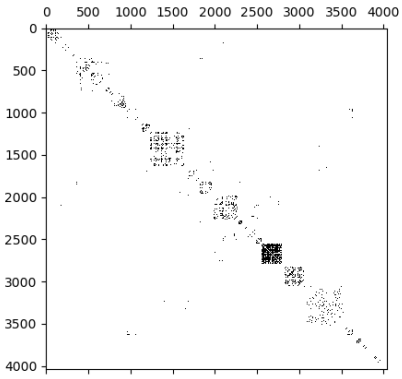

Figure 11: Sorted Adjacency Matrix - Facebook Dataset

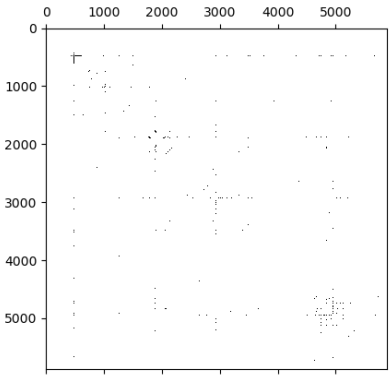

Figure 12: Sorted Adjacency Matrix- Bitcoin Dataset

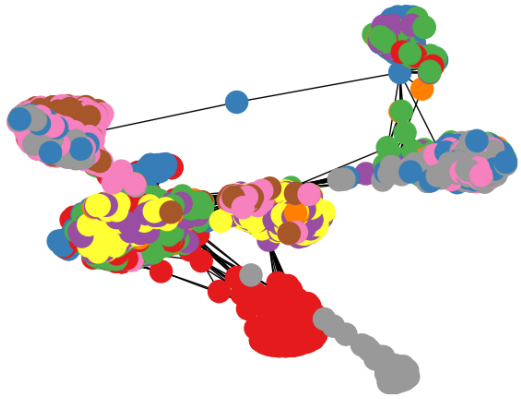

Figure 13: Graph Visualization - Facebook Dataset

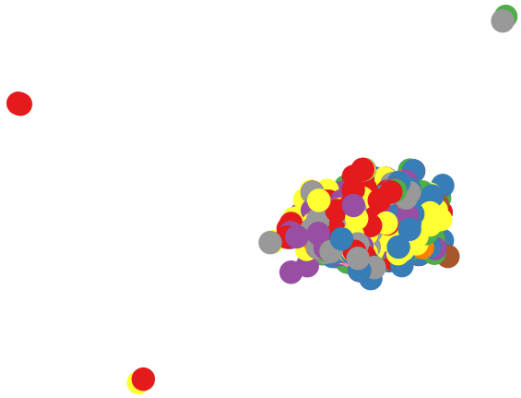

Figure 14: Graph Visualization- Bitcoin Dataset

Dataset	No. of Communities	Modularity	Final Community IDs
Facebook Data	101	0.8105424154958314	[42 69 95 135 151 152 174 179 182 189 192 201 241 256 278 282 300 307 351 382 401 453 505 572 612 642 650 702 757 831 852 865 877 882 895 1016 1044 1069 1111 1178 1208 1224 1240 1245 1253 1486 1586 1627 1657 1698 1748 1760 1775 1829 1892 1905 2113 2167 2168 2401 2424 2431 2435 2536 2565 2595 2721 2792 2801 2817 2902 2917 3003 3012 3122 3194 3290 3310 3311 3407 3543 3581 3585 3598 3641 3657 3668 3675 3676 3699 3700 3742 3787 3792 3846 3885 3925 4007 4012 4018 4029]
Bitcoin Data	397	0.4433171535500787	[12 33 46 61 143 208 213 345 404 410 445 470 493 512 526 531 558 563 579 595 596 605 624 638 644 657 658 665 676 678 687 698 706 711 .... 4310 4340 4347 4367 4393 4396 4416 4421 4423 4461 4504 4512 4528 4529 4530 4561 4568 4573 4624 4652 4659 4733 4749 4757 4766 4771 4777 4796 4813 4837 4876 4885 4887 4889 4901 4923 4947 4976 4986 4989 5005 5025 5036 5050 5070 5123 5126 5145 5177 5193 5262 5270 5272 5296 5322 5327 5335 5348 5354 5359 5362 5396 5426 5470 5485 5518 5537 5553 5558 5561 5575 5624 5645 5648 5688 5706 5718 5726 5729 5764 5775 5815 5818 5823 5830 5834 5835 5847 5877]

Table 1: Results of Louvain Algorithm

Note: For the bitcoin dataset the node IDs are mapped to numbers between o and 5881(number of nodes), and entries in the above table are according to mapped nodes, for a detailed table please refer to the results.txt file submitted.

2.5 Question 5

- **Metrics for Evaluating Node Decomposition into Communities:**
  - Modularity
  - Normalized Mutual Index
- **Modularity Metric:**
  - Modularity is a metric used to evaluate the quality of a node partition into communities. It quantifies the density of edges within communities in comparison to what would be expected by random chance. Higher Modularity scores indicate better partitions.
- **Normalized Mutual Index:**
  - The Normalized Mutual Index is another metric that assesses the similarity between the detected communities and the actual community structure (if known). It provides a measure of how well the detected communities align with the true structure.

- **Visual Inspection:**
  - In addition to metrics, visual inspection of the communities and their interconnections can also be a valuable way to determine the quality of a partition. This allows for a qualitative assessment of the detected communities and their relationships.

2.6 Question 6

Dataset	Running Time(Spectral)	Running Time(Lovain)
Facebook Data	46.74334383010864 seconds	8.026901721954346 seconds
Bitcoin Data	96.23750400543213 seconds	25.585325002670288 seconds

Table 2: Running time

Observation: It is observed that the running time for the Louvain algorithm is comparatively less compared to spectral decomposition.

2.7 Question 7

- In my opinion Louvain algorithm provides better communities compared to spectral decomposition algorithm.
- My opinion can be backed by the modularity values of the obtained graph partitions after running the two algorithms.
- Also Louvain is faster than spectral decomposition.

Dataset	Modularity(Spectral)	Modularity(Louvain)
Facebook Data	0.7950467918186478	0.8105424154958314
Bitcoin Data	0.0459682869632125	0.4433171535500787

Table 3: Modularity