EO 259 - Assignment 1

Shankaradityaa Venkateswaran
Sr no: 22190

August 28, 2024

1 Implementation Summary

To implement the Louvain algorithm, I followed the algorithm shown in the slide with some reference to the orginal paper. I mainly used pandas and python dictionaries and lists to keep track of the list of edges, nodes, and their corresponding communities. Initally I did not want to change the nodeIDs to 0 to n-1, so I kept it the same and went ahead without changing antyhing, hence my use of pandas to keep track of the communities. Just as in the algorithm implementation, I calculate the modularity change for each node and its adjacent communities, and I move the node to the community that gives the maximum modularity change. I keep doing this for all the nodes and check the modularity of the entire graph at the end. To traverse all the nodes I create a modified version of the adjacency list which has the nodeID followed by its neighbours as the elements. This allows me to preserve the nodeIDs without mapping them to something else. The stopping condition is when the modularity of the entire graph is no longer positive. At the output I give a partition array which has the community number for each node. This array has the same ordering as in which the nodes were parsed by going through the edges, hence it will not be in order.
To implement the Girvan-Newman algorithm, I discussed with my collegues on how to implement it. In the end, after seeing that my louvain algorithm was very inefficient, I decided to use numpy arrays, and forgo using pandas in this algorithm. I also decided to create a mapping of the nodeIDs to 0 to n-1, as I realized that my implementation of the Louvain algorithm was very inefficient. I used a proper adjacency list to keep track of the neighbours of each node. I then calculated the betweenness of each edge, and removed the edge with the highest betweenness. I then recalculated the betweenness of all the edges and repeated the process until the number of levels that I specified is reached. While implementing this algorithm I decided the stopping condition to be number of levels to be formed. I then output the community matrix as specified.
In both cases I have a slew of helper functions meant to calculate the modularity, the betweenness, the adjacency list, the credits, maintain the mapping of the nodeIDs, to calculate the modularity change, and many other small tasks. I made it my goal to have everything modular so that I can reuse the code over and over again if needed.
I did not find the time to implement the dendogram visualization funciton, hence I have left it blank.

The running times of both my algorithms are extremely long. I have run them on smaller graphs and gotten appropriate results, but they did not terminate in time for the larger graphs. I could not find the time to optimize my code, but as a learning exercise I will look over and try to do so in the future.

2 Questions

1

I have run the Girvan Newman algorithm on a reducded version of the graphs, but I did not save my output at that point, and now I am out of time to run the algorithm again. The algorithm shows appropriate results for toy examples as well.

2

The stopping criterion which I chose was the number of levels. I chose this because I did not want to run the algorithm for too long, and I wanted to see the results of the algorithm after a certain number of levels. I could have chosen the modularity to be the stopping criterion, but I thought that my implementation of modularity calcution was not very effective, thus I discarded it.

3

I did not get enough time to implement the dendogram visualization function. I will try to implement it in the future as an exercise to learn more.

4

I tried to run the louvain algorithm on the two datasets, but the algorithm was taking too long to run. In hindsight, my implementation was clunky, took a lot of memory, and performed a lot of operations which took up a lot of time. I did not get anywhere close to getting my louvain implementation to run on the larger datasets. As an exercise in learning, I will try to implement the algorithm in a more efficient manner in the future. While it is taking too long to run, on smaller graphs(toy examples), it gives appropriate results.

5

If I had more time, the best decompositon of nodes into communities would be done throught modularity change, as it is a good indicator of how well the communities are formed.

6

The Girvan-Newman algorithm was taking around an hour to remove one edge while running on the wiki-how data, while the louvain algorithm was taking even longer to get one run throught all the nodes. I cannot form an accurate estimate of the running times, but I can guess that my Girvan-Newman implementation is faster than my louvain implementation.

7

In my opinion, the louvain algorithm gives better rise to communities as it builds communities bottom up unlike Girvan-Newman which breaks the graph into communitites. I also found that Givan-Newman is prone to breaking nodes into single node communities, which is not recommended while forming communities. Hence I think the louvain algorihtm gives better rise to commmunities.

3 LLM Usuage

While writing the code, I did use Github Copilot to assist me.