Assignment 3: Community Detection

Anmol Garg

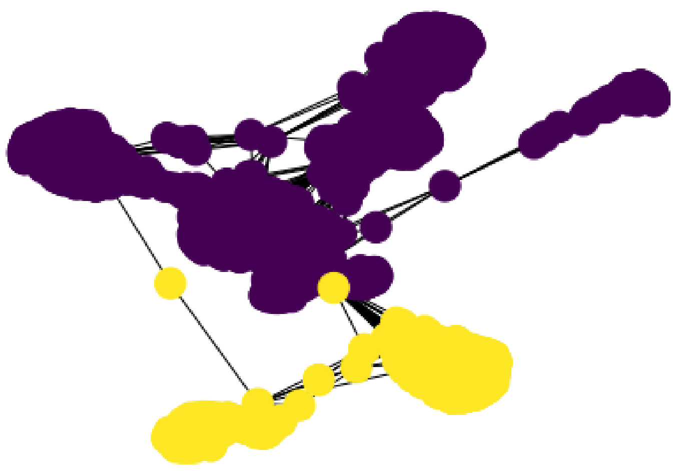
20178

Q1 Plot the sorted Fiedler vector, the associated adjacency matrix and the graph partition

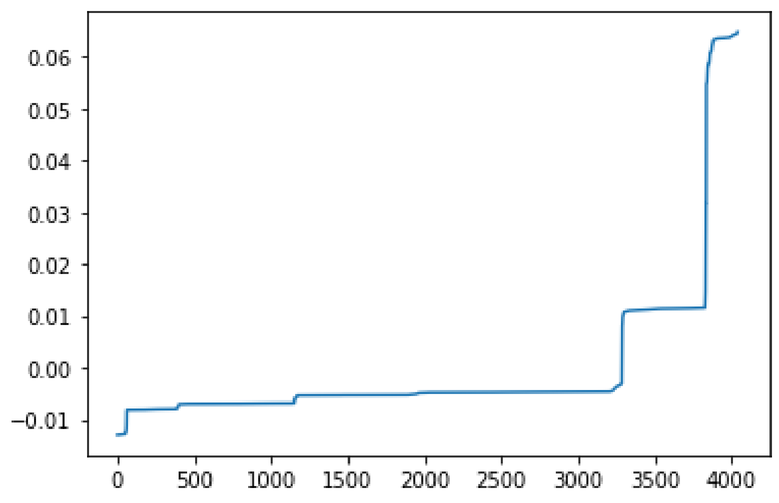
Implementation Given nodes connection list we first find the adjacency matrix. Using adjacency matrix , degree matrix is calculated . Eigenvector decomposition of L = D-M is done and the eigenvector corresponding to nonzero (or second in most cases) eigenvalue is our Fiedler vector. Using fvector , the indexes where the value of vector is positive is found to partition the nodes .

Answer:

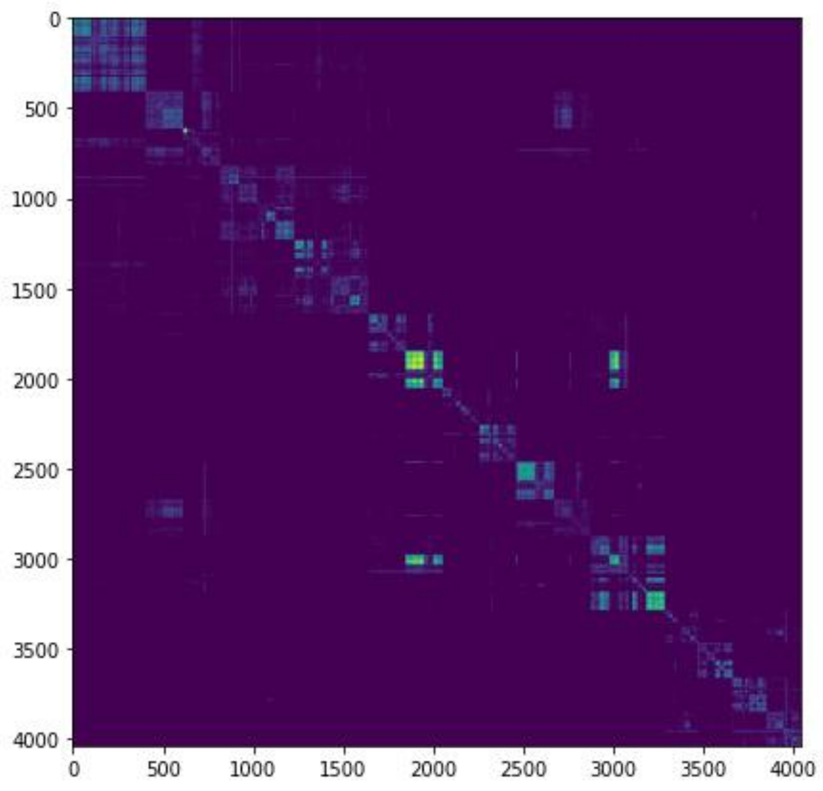
# Graph visualize after one iter. of spectral decomposition on FB dataset



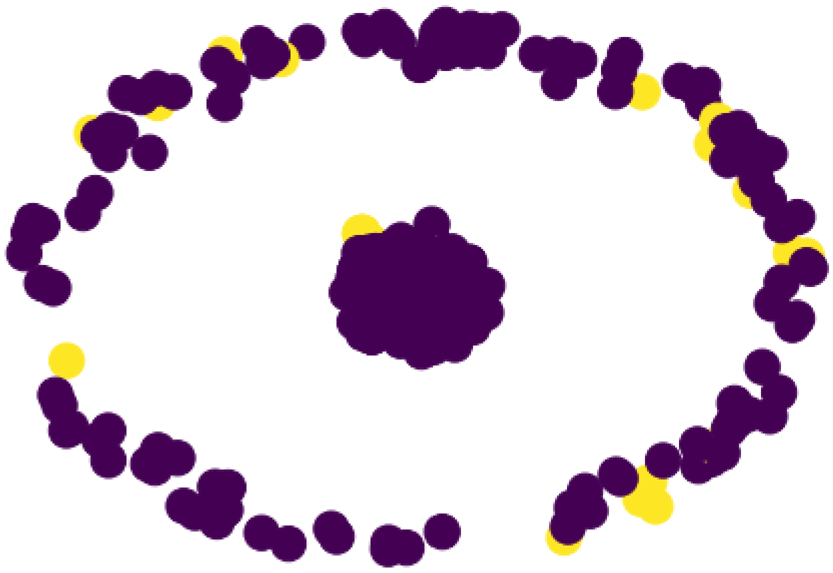
# Fiedler Vector for FB dataset one iter. of **spectral decomp**



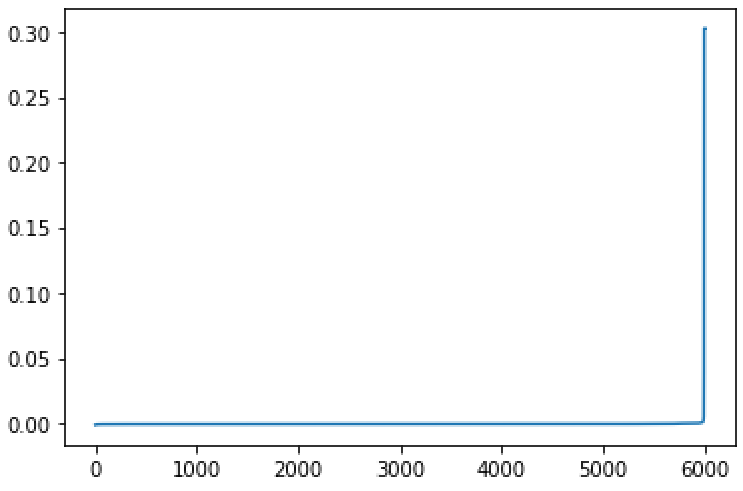
# Associated sorted graph matrix for one iter. Of spectral decomp on FB dataset



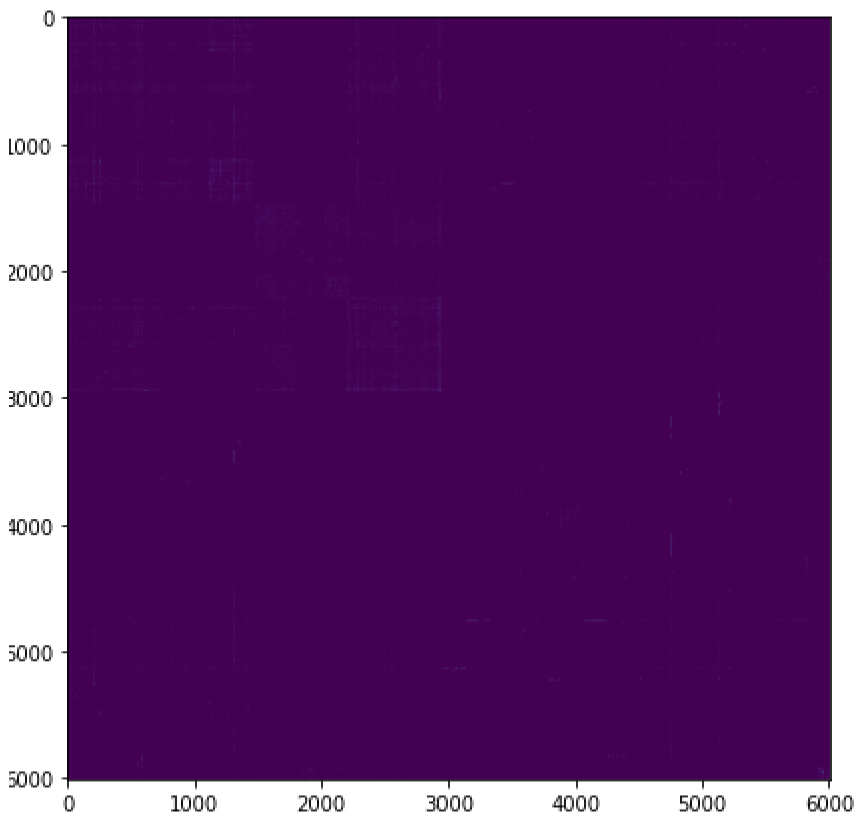
# Graph visualize after one iter. of spectral decomposition on BTC dataset



# Fiedler Vector for BTC dataset one iter of **spectral decomp**



Associated sorted graph matrix for one iter. Of spectral decomp on BTC dataset(very sparse matrix)



Q2 Automated algorithm to determine the right set of communities using the spectral decomposition method. What would be your stopping criterion

Implementation: For Spectral decomposition , spectraldecomp_oneiter function is recursively called on the graph partitions . If after partition the conductance value of two partitions is less than the given threshold(0.1) then the partition is nullified and we stop the current recursive process. Whenever the recursive function stops meaning the given list can't be broken further so must belong to same community and therefore this list is appended to "out" which is list of list(node values belonging to same communities) . With help of out graph partition is returned.

Answer: Running spectral decomposition gives us a fieldler vector which when thresholded at 0 level gives us two partions , one having fvector value above 0 and other less than 0. After portioning we check the conductance of the cut (formula shown below)

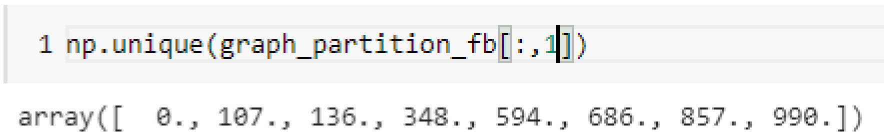
$$Conductance: \qquad Q = \frac{cut(V_1, V_2)}{min(Vol(V_1), Vol(V_2))}$$

I have set the threshold for this conductance to be at 0.1 which can be seen as hyperparameter and be changed according to experiments

So my graph recursively partitions into two separate sub-graph until any partition conductance value is less than the threshold and the graph portioning stops.

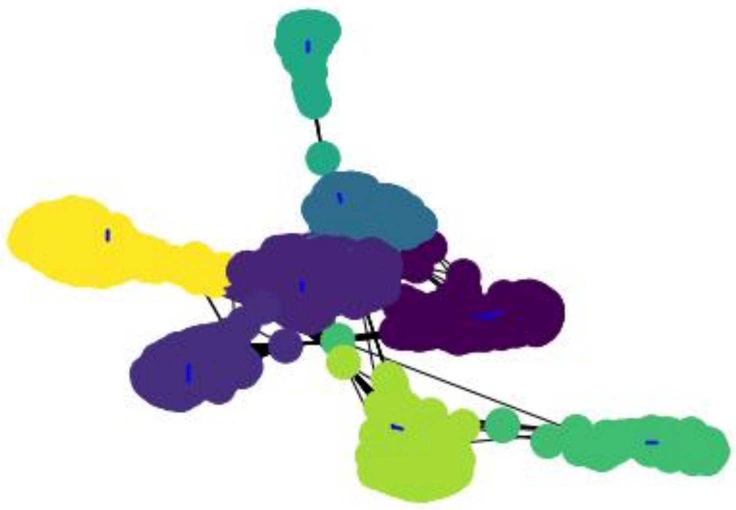
Q3 Plot the associated adjacency matrix sorted by associated sorted sub graph Fiedler vectors. Visualise and show the graph that you obtained

# Obtained 8 (marked by pen)communities after doing spectral decomposition until stopping criterion on FB dataset

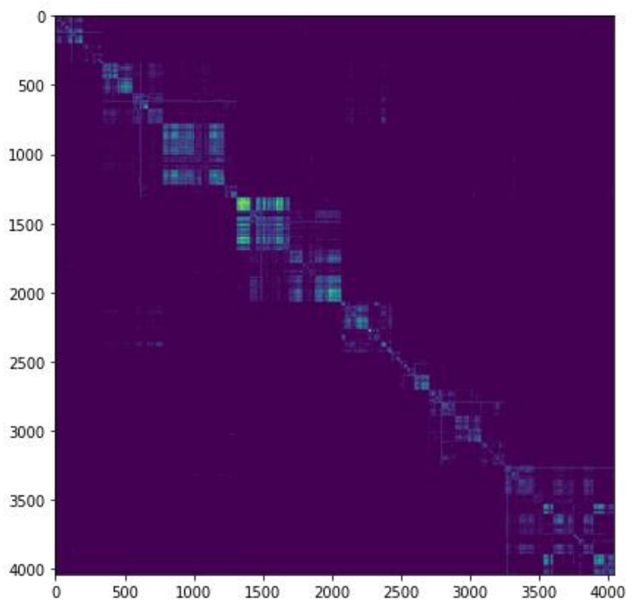


```
np.unique(graph_partition_fb[:,1])
```

```
array([  0., 107., 136., 348., 594., 686., 857., 990.])
```



# Plot of associated graph matrix sorted based on fiedler vector of subgraph partitions

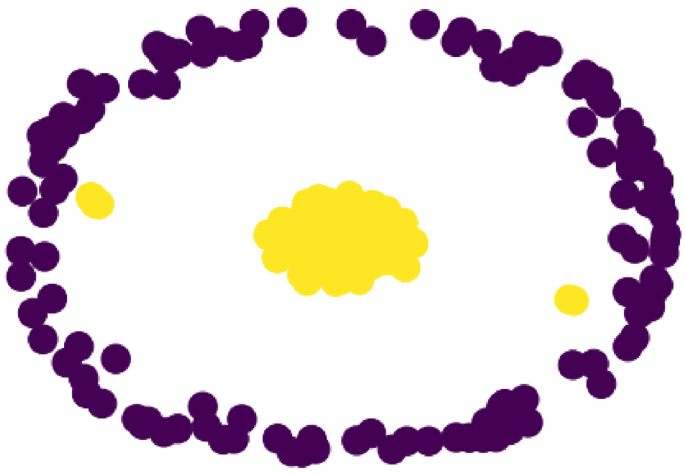


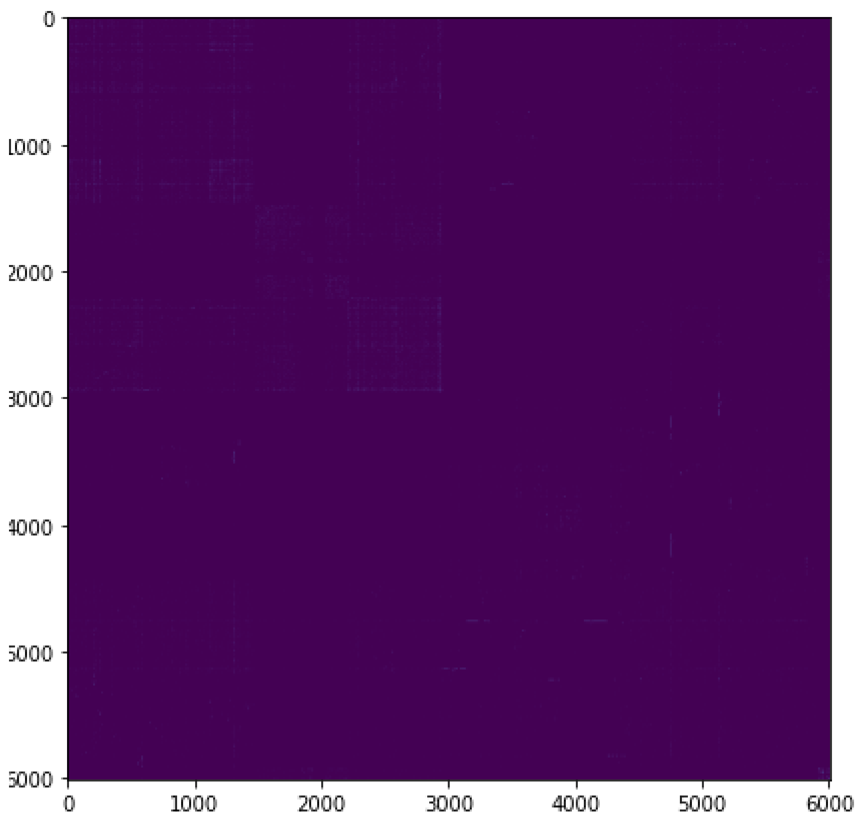
Obtained 2 communities after doing spectral decomposition until stopping criterion on BTC dataset

Our Algo has taken many disconnected nodes as one community itself(outer circle nodes doesn’t relate to any other nodes in graph)

```
np.unique(graph_partition_btc[:,1])
```

```
array([0., 1.])
```





Q4 Louvain algorithm after one iteration

For FB dataset got 119 communities and for BTC dataset got 724 different communities

Implementation: Stored two hashmaps node_neighbours and node_to_comm which saves community value of ach node . Degree of node and degree of each community is calculated every time we update our communities that is when node i move from comm A to B and the change in modularity for such transition is maximum. Change in modularity is calculated using

```
delq_i = (ki*(d1-d2))/(2*m) + (kiin2-kiin1) - ki**2/(2*m)
```

--- ki is degree of node

--- d1 is total degree of all nodes present in node i initial comm

--- d2 is total degree of all nodes present in node i target comm

--- m is total number of edges in graph

--- kiin1 is edges from node i to nodes present in initial comm

--- kiin2 is edges from node i to nodes present in target comm

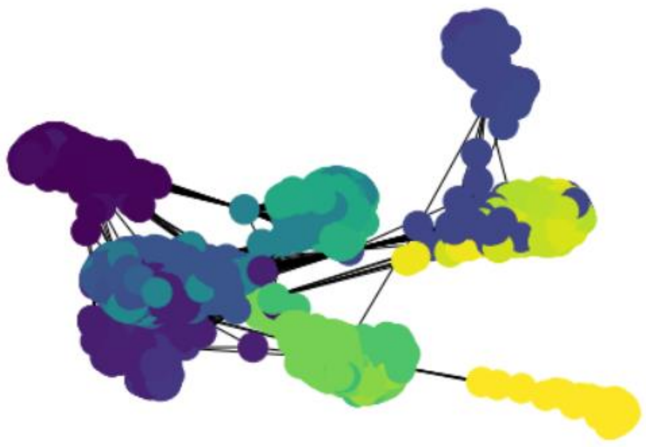
After each node transition we update our node_to_comm dictionary as well as the degree values for the new node communities.

Output for **FB** dataset for Louvain Algo

Got 119 communities after one iteration of Louvain algorithm

```
 3699. 3742. 3787. 3792. 3840. 3885. 3925. 4007. 4012. 4018. 4029.]
119
```

```
nx.draw(G_fb,node_color=graph_partition_louvain_fb[:,1])
```

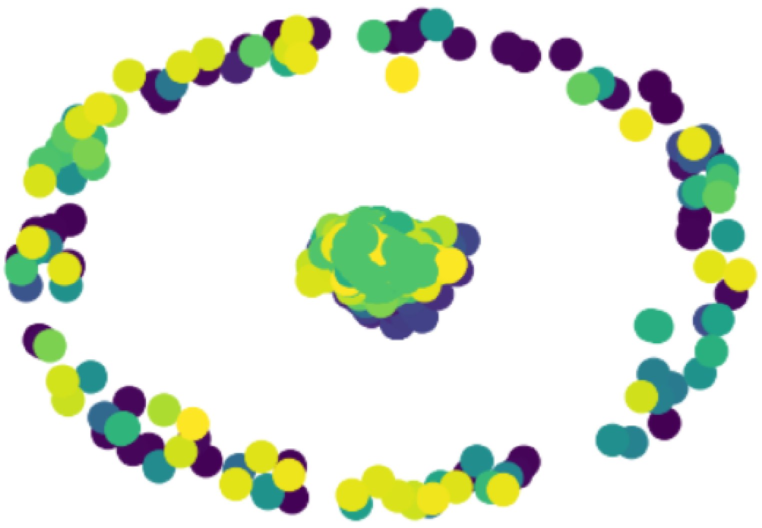


Output for **BTC** dataset for Louvain Algo

```
len(np.unique(graph_partition_louvain_btc[:,1])
# nx.draw(G_btc,node_color=graph_partition_btc[:,1])
```

```
724
```

```
nx.draw(G_btc,node_color=graph_partition_louvain_btc[:,1])
```



Q5 How would you pick the best decomposition of nodes into communities?

For each decomposition we can calculate cut value or conductance for that decomposition and the one with the greater value is better community distribution(as discussed in class)

Q6 What was the running time of the Spectral decomposition algorithm versus the Louvain algorithm on the data sets you were given?

Spectral decomposition took around 2 minutes for each dataset
Louvain algorithm took around 25 to 30 seconds for each dataset for one iteration as for spectral decomposition we have to calculate eigenvalues and eigenvectors which is computationally expensive and slow

Q7 In your opinion which algorithm gave rise to better communities, why?

Spectral decomposition gave rise to better communities compared to Louvain algorithm one iteration in terms of FB dataset. But for BTC , spectral grouped all the unconnected nodes to single community but Louvain algo better handled the above case and giving us a total of around 700 communities which seems correct as there must be very less relation between nodes if graph/matrix is sparse.

Louvain is a greedy algorithm (in terms of modularity score) which can get stuck to a local minimum but spectral on the other hand is slow .