Project 2: Social Network Mining

Large Scale Social and Complex Networks: Design and Algorithms

Ananya Deepak Deoghare, Kimaya Milind Kulkarni, Shruti Mohanty

005627628, 805528337, 705494615

ECE 232E Spring 2022

Introduction

The project is intended to investigate numerous intriguing aspects and significant insights of a network's connectedness, degree distribution, as well as individualized network characteristics and neighborhood-based metrics. We investigated the social networks Facebook (undirected) and Google+ for this study (directed).

1 Facebook network

1.1 Structural properties of the Facebook network

Question 1.1:
We obtained our Facebook dataset from `http://snap.stanford.edu/data/egonets-Facebook.html` and unzipped the edgelist file facebook_combined.txt.gz and created the Facebook network. In this question, we are required to build the Facebook network from the edgelist file and give the number of nodes and edges. To build the network, we first read the contents of facebook_combined.txt file using the read.table() function, and then use the graph.data.frame() function from the igraph package in R to generate an undirected graph from the dataframe. Figure 1 depicts the resultant network: According to Facebook's specifications, the formed graph should be an undirected network with each vertex representing a single user and the edge between two nodes denoting the friendship of two users. As a result, while calling read graph, we need carefully set the directed option to false. We don't have to worry about the varied definitions of connection and the enormous linked component since our graph is undirected (GCC). Using the functions gorder(), gsize(), and is connected, we may obtain the following result: for the graph under consideration, we define it as G = (E,V,WE,WV) where $|V| = 4039$ nodes (users) and $|E| = 88234$ edges (links). Furthermore, the graph is linked.

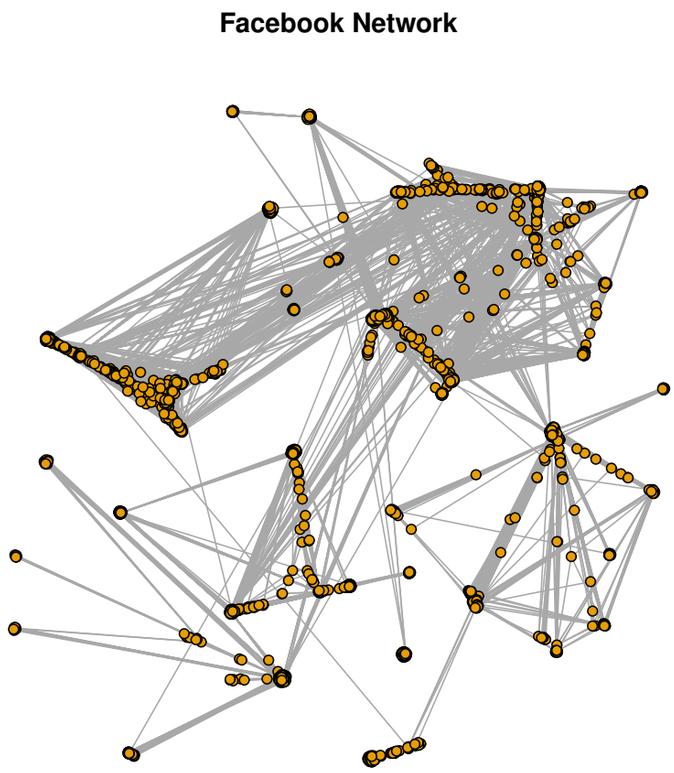


Figure 1: Generated Facebook network created from edgelist file

Question 1.2:

In this question, we are required to determine whether or not the Facebook network is connected. To determine the network's connection, we utilize the is.connected() method. We discovered that the entire network is linked. This indicates that there are no isolated users (nodes) in the network that have no friends, and that all users have some connection (edges) with other users. In theory, social networks may be modeled as Barabasi-Albert networks with weak preferred attachment, implying that users with greater degrees of connection are more likely to get edges with newer members. Because each new node is always connected to an existing node, the nal network is theoretically constantly connected by construction.

We just need to examine the diameter of the graph rather than the enormous linked component because the graph is connected. The diameter of our Facebook network, as determined by the function diameter, is 8. The answers to Questions 1 and 2 are both consistent with the numbers displayed on the SNAP website.

Question 2:

In this question, we are required to calculate the network's diameter. A graph's dimension is defined as the maximum distance between any two nodes (greatest shortest path between any two nodes). We use the diameter() method to calculate the network's diameter, which is **8**.

Question 3:

For undirected graphs, the degree of a vertex v_i is defined as:

$$deg(v_i) = \| \{v_i, v_j \in E\} \|$$

where, E is the set of edges of the graph. The degree distribution of the graph, which is the probability distribution function of the random variable , denoting the degree of a randomly selected vertex of the graph, is given by:

$$P(X = k) = \frac{\text{Number of vertices with degree k}}{\text{Number of vertices}}$$

The average or expected degree of the network is given by: where m = number of nodes

$$\left(\mathbb{E}_m(deg) = \frac{1}{m}\sum_{k=1}^{k_{\max}} k \times \mathcal{N}_k\right) = \left(\mathbb{E}(deg)\sum_{k=1}^{k_{\max}} k \times P_k\right) \forall (m \to \infty)$$

picked, N_k = number of vertices with degree k, P_k = probability that a randomly picked node has degree k.

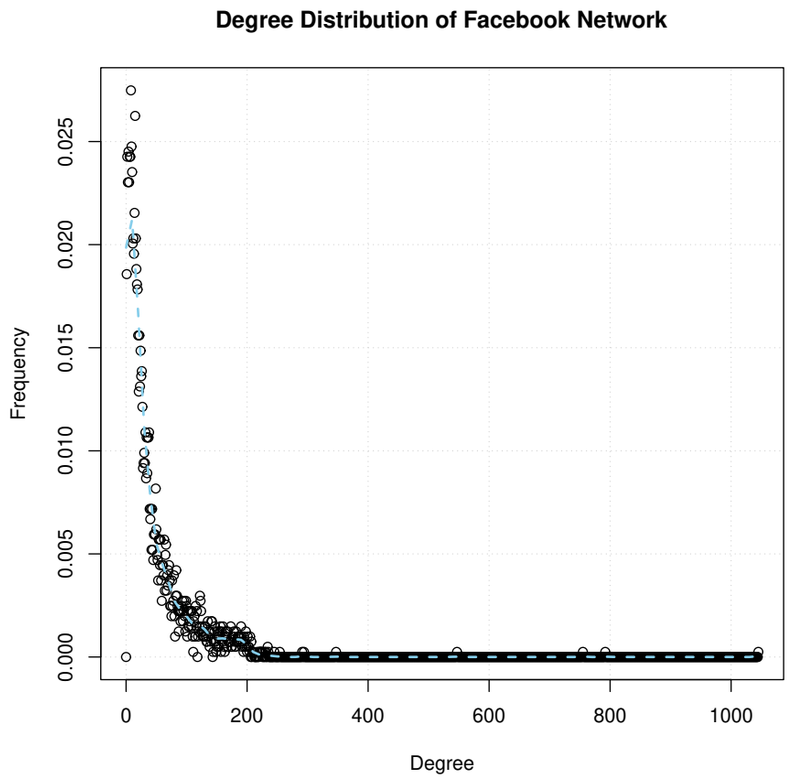


Figure 2: Degree distribution of the Facebook network

Figure 2 depicts the degree distribution of the Facebook network. Because the displayed curve has a strong similarity to the power-law distribution, we may safely plot the degree distribution in log-log scale in the following questions. After obtaining the degree distribution, it is simple to calculate the average degree. We just multiply degrees by their respective frequency and then add them all together. This approach yields **an average degree of 43.691013**.

From Figure 2, we observe that the degree distribution follows the power-law model, exhibiting the following fat-tailed degree distribution:

$$P_k \propto \frac{1}{k^\gamma} = \frac{1}{k^\gamma \sum_{k=1}^{k_{max}} k^{-\gamma}}, k \in 1, 2, ..., k_{max}, \gamma > 0$$

$-\gamma$ is the gradient of the $log(P_k)$ versus $log(k)$ curve, which is linear. This is expected for scale-free power-law networks with weak linear preferential attachment.

The graph also reveals that nodes with degrees close to one are very high, with a dramatic drop off for degrees less than 100, and then it levels off about 200, when the relative frequency vertice value is near to zero. This means that all of our users are connected in some way (since the graph is all connected), yet the majority of users are only aware of their immediate friends or, at best, some of their friends' friends. The average network degree is 522.5, although this is not an accurate measurement because the figure is slanted to the left.

Question 4:

We are required to plot the degree distribution obtained in Question 3 in log-log scale and estimate the slope of the best fit linear regression line in this question. The best-fit line of the data was added in blue. In order to create the line, we removed all degree and degree distribution pairs with non-real numbers and used the remaining data create a best-fit line. The plot is depicted in Figure 3. **The line's gradient is -1.2475**.

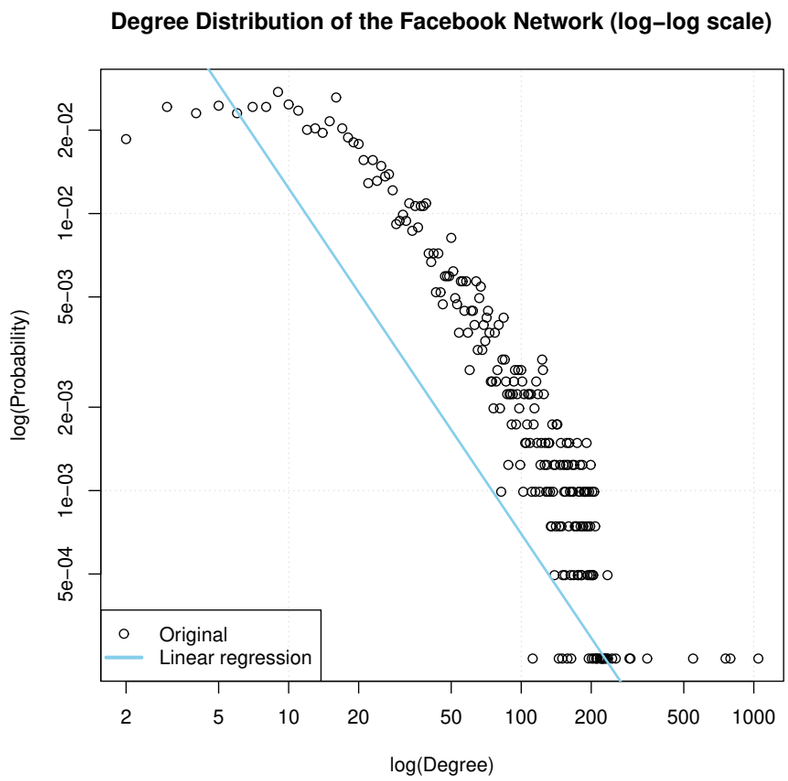


Figure 3: Degree distribution of the Facebook network in log-log scale.

Figure 3 shows that the degree-distribution in the log-log scale is about linear. The degree distribution of scale-free networks is linear on the log-log scale, as stated in Question 3:

$$P_k \propto \frac{1}{k^\gamma} = \frac{1}{k^\gamma \sum_{k=1}^{k_{max}} k^{-\gamma}}, k \in 1, 2, ..., k_{max}, \gamma > 0$$

Since $1 < \gamma <= 2$, the predicted degree and variance are both unlimited, resulting in a dense network with a significant number of high degree nodes. However, since $\gamma << 3$, nodes in the Facebook network have been weakly preferentially attached rather than highly preferentially attached. In preferred attachment models ($\gamma \approx 3$), nodes with higher degrees or connectedness are more likely to get edges with younger nodes, with no heuristics for determining whether or not an incoming node is related to a high-degree node. As a result, the network is sparse, with a significant number of high-degree nodes. In social networks, however, individuals tend to create communities with known (mutual) acquaintances, resulting in tightly packed groups of known users with varied connection.

Question 5:

In this section we created a personalized network of the user whose ID is 1. We also count the number of nodes and edges of the network. A personalized network is defined as a subgraph induced by user v_i and its neighbours. The network basically concerns the neighbours of a target (core) node. The generated network is shown in the following figure. This network has 348 nodes and 2866 edges.

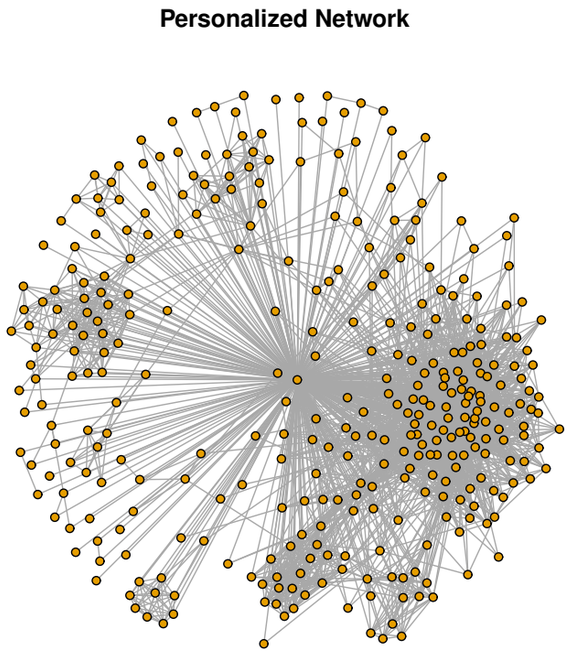


Figure 4: Personalized network for user ID 1

Question 6:

For user id 1, the diameter of the personalized network is 2. The trivial lower bound of the personalized network diameter is 1 and the trivial upper bound for the personalized network is 2.

Question 7:

The largest distance between any pair of nodes (greatest shortest path between any two nodes) is the diameter of a graph. If the diameter of a personalized network is 2, that implies there exist users who are not connected directly in the subgraph, but are connected through a mutual user. The network is not fully connected and there exists vertices v_i and v_j in the network such that that v_i and v_j are connected via a mutual tertiary node . Basically, not all users in the network know each other directly.

Although, if the diameter of a personalized network is 1, then all users in the network know each other directly and there exists direct edges between all vertices in the network. And we can say that the network in this case is fully connected.

Question 8:

As per this section, we found the total number of core nodes and their average degree in the Facebook network. Core nodes are the nodes that have more than 200 neighbors. The Facebook network has 40 core nodes, with an average degree of 279.375.

Question 9:

In the following question, we found the community structures of the personalized networks of 5 core nodes (node IDs 1, 108, 349, 484 and 1087) using Fast-Greedy, Edge-Betweenness, and Infomap community detection algorithms and also reported the resulting modularity scores.A clustering of the vertices in the network such that the number of inter cluster edges is much smaller than the number of intra cluster edges is the community structure of a graph. Basically, community structures segregate regions of high connectedness from the overall sparse network structure. The following figure is the example of the same.

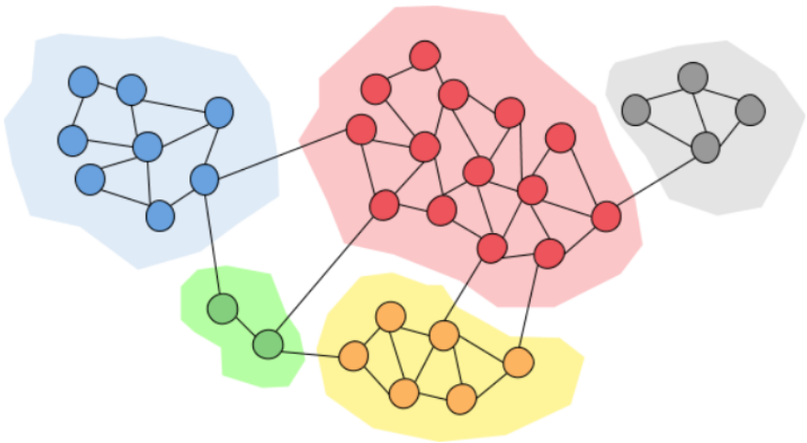


Figure 5: Community structure showing 5 clusters of high connectedness in a random graph

To find community structures, we use Mark Newman modularity index. For a particular cluster C_i, the modularity index m_{C_i} is given as:

$$m_{Ci} = f_{C_i} - r_{C_i} \tag{1}$$

Here, f_{C_i} is the number of edges in C_i and m_{C_i} is the expected number of edges in if the network was created randomly with the same degree distribution. From the above information the equation can be written as follows:

$$m_{C_i} = \sum_{i,j \in C_i} A_{ij} - \frac{k_i k_j}{2m} \tag{2}$$

In the above equation A refers to the node-node incidence matrix, k stands for the degree of nodes selected for random stub (dangling edges) matching and m is the number of edges in r_{C_i}. For a network P partitioned into disjoint clusters, the modularity index of the network is given by the following equation:

$$Q(P) = \sum \frac{1}{2m} \sum_{i,j \in C_i} A_{ij} - \frac{k_i k_j}{2m} \tag{3}$$

The node-node incidence matrix is a mathematical representation of a graph with the rows and columns corresponding to the vertices and edges of a graph respectively. For constructing A, we follow the given steps/conditions:

- If an edge is entering towards node , A_{ij} is -1 (for undirected graphs, -1 and +1 will just be 1).
- If an edge is leaving from node, A_{ij} is +1.

- If an edge is neither entering towards node nor leaving, A_{ij} is 0.

As we know the maximum value of Q(P) is 1. The goal of the clustering algorithm is to find the network partitioning with the highest modularity index. Modularity measures the strength of the network division into modules with strong interconnections (community structures). The higher the modularity, the larger is the number of edges within communities, while the communities are sparsely connected to each other.

Now let us discuss the greedy algorithm used to find modularity index:

- Start with clusters.
- Compute the change in Q(P) if a pair of clusters among all the existing clusters is merged and perform the merge if Q(P) increases after merging.
- Repeat step 2 until no significant changes are observed

The edge-betweenness of an edge e, denoted as w(e), in terms of the shortest path between each pair of nodes in the network is denoted as:
w(e) = Number of shortest paths e is a part of
Edges which act as bridges between areas of high connectedness have high values of w(e) because the shortest paths between nodes in different clusters are connected to these edges. Let us understand the steps for edge-betweenness community detection algorithm (Girvan–Newman algorithm):

- Compute w(e) for all edges and delete the edge with the highest w(e).
- Repeat step 1 for Q(P) the modified network and compute the . If Q(P) increases, repeat step 1, else terminate the algorithm. merging.

Infomap finds the community structure that minimizes the expected description length of a random walker trajectory. Informap uses probability flow on random walks (information theoretic approach) as a surrogate for information flow in the network, segregating the network into modules based on a compressed description of probability flow.
The following table shows the modularity scores of the 5 personalized networks obtained via the three community detection algorithms.

Table 1: Performance Metric for Linear SVM Classifier with GLOVE embeddings

Core Node	Node Count	Edge Count	Fast Greedy	Edge-Betweenness	Infomap
1	348	2866	0.4131	0.3533	0.3891
108	1046	27795	0.435929	0.506755	0.508223
349	230	3441	0.2517	0.1335	0.0960
484	232	4525	0.5070	0.4890	0.5152
1087	206	7409	0.1455	0.0276	0.0269

The following figure shows the community structures for each of the core nodes for the three algorithms.

We can make the following observations from the above figure and table:

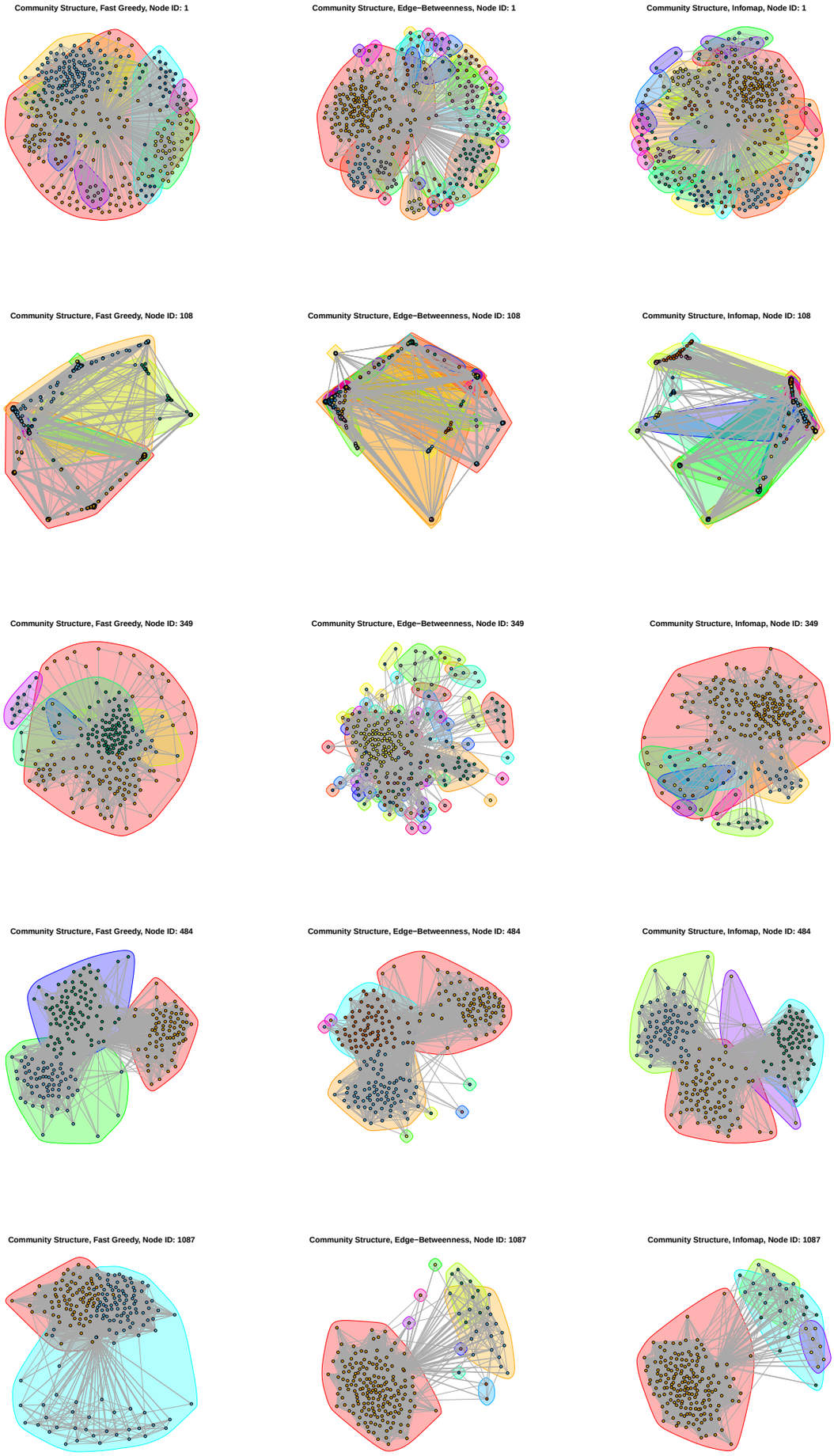


Figure 6: Community structures of personalized networks for 5 core nodes using 3 algorithms.

- Edge-betweenness algorithm performs the worst on average among all three algorithms based on modularity score. This is partially observable from the figure, where we see that the edge-betweenness community detection algorithm tends to leave out a significant number of nodes within isolated communities of single members, leading to weak intra-community connectedness. The reason for this is that the edge-betweenness generates communities via hierarchical decomposition by building a full dendrogram, assuming that there is only a single path to go from one group to another group and cutting the tree into groups via the modularity score of the partitions at each level of the tree. This process leaves out nodes with few vertices in their own communities, as the probability of them being included in a cut (community) at the initial stages of the algorithm is low. In addition, we observed that edge-betweenness is the slowest among the three methods. This is because of the computational complexity of calculations involved, with the scores being recalculated after every edge removal.

- Fast-greedy algorithm performs the best on average among all three algorithms based on modularity score, followed by Infomap. Lets us compare fast greedy with the other algorithms. Compared to the edge-betweenness algorithm, fast-greedy is bottom-up instead of top-down, optimizing the modularity in a greedy manner such that the merges are locally optimal, yielding the largest possible increase in the modularity with each merge. In other words, the greedy algorithm is optimized to increase modularity, whereas edge-betweenness aims at decomposing the network based on the number of shortest paths, which does not always yield dense communities with sparse inter-community connections. The algorithm is also faster than edge-betweenness without requiring any parameter turning.

- Infomap and edge-betweenness algorithms perform similar to each other.

- Fast-greedy algorithm does not perform well on large networks, such as for core node 108. Furthermore, compared to Infomap or edge-betweenness, the fast-greedy algorithm often tends to merge nodes from otherwise visibly separate communities. This is because of the resolution limit problem of the fast-greedy method, with the tendency of merging nodes within a community below a given number of vertices to neighbouring nodes. The problem is amplified for large networks, where the greedy algorithm tends to form fewer communities compared to Infomap or edge-betweenness.

- The personalized network for node 484 achieves the highest average modularity score. Figure shows that the density of clusters are roughly homogenous and form distinguishable regions of high connectedness with sparse connectivity among the high-density regions.

- The personalized network for node 1087 has the lowest modularity for all three algorithms among the five personalized networks. This is because the network for node 1087 has the lowest node to edge ratio. It can be seen that, a higher value of m indicates that an incoming node is connected to a larger number of older nodes. While this should result in strong intra-community connectedness, the global sparsity among different communities is lost due to the connectedness requirement brought on by high values of m, resulting in edges being formed among otherwise distinct clusters and hence weakening the community structures. Mathematically,

$$Q(P) = \sum \frac{1}{2m} \sum_{i,j \in C_i} A_{ij} - \frac{k_i k_j}{2m} \tag{4}$$

If the number of edges in the network m increase then Q(P) drops. Hence, less clusters are formed in the overall graph.

Question 10:

For this section we found the community structures of the personalized networks of 5 core nodes (node IDs 1, 108, 349, 484 and 1087) using Fast-Greedy, Edge-Betweenness, and Infomap community detection algorithms but with the core node removed, as well as compare the resulting modularity scores with those obtained in Question 9. The table below shows the comparison of modularity scores of the 5 personalized networks obtained via the three community detection algorithms with and without the core node involved. The figure below shows the community structures for each of the core nodes for the three algorithms without the core nodes.

Table 2: Performance Metric for Linear SVM Classifier with GLOVE embeddings

Core Node	Code Node Present?	Fast Greedy	Edge-Betweenness	Infomap
1	Yes	0.4131	0.3533	0.3891
1	No	0.4418	0.4161	0.4180
108	Yes	0.435929	0.506755	0.508223
108	No	0.4581	0.5213	0.5206
349	Yes	0.2517	0.1335	0.0960
349	No	0.2456	0.1505	0.2448
484	Yes	0.5070	0.4890	0.5152
484	No	0.5342	0.5154	0.5434
1087	Yes	0.1455	0.0276	0.0269
1087	No	0.1481	0.0324	0.0273

From the above table, we see that the modularity scores have increased after removing the core node for all 5 personalized networks. This is because the core node acts as the bridge between all of the other nodes, which causes the network with the core node less capable of being partitioned into communities with strong intra-community connectedness and sparse inter-community connectedness compared to the network without the core node. With the presence of the core node, it is difficult to classify the core node into a single community as it is connected to every other node while also making it difficult to assign densely packed and sparsely inter-connected communities among the other nodes, resulting in a dense network with low modularity. In addition, with the core node present, the other nodes are not always directly connected to each other or the rest of the graph, resulting in many isolated communities (communities with a single node) for edge-betweenness as observed in Figure 6. With the core node removed, the edges from the core node to all other nodes are removed, allowing the community partition algorithms to find densely packed areas of high connectedness with sparse inter-community edges. The probability of strong intra-community connections is greater for networks with core nodes removed, and likewise, the extent of sparsity among communities are amplified in such networks. These facts are also evident from the following figure, where we see that all the algorithms, especially edge-betweenness, have a decreased tendency of forming groups with isolated nodes. In addition, the community structures are still well-defined and well-partitioned, despite absence of the core nodes.

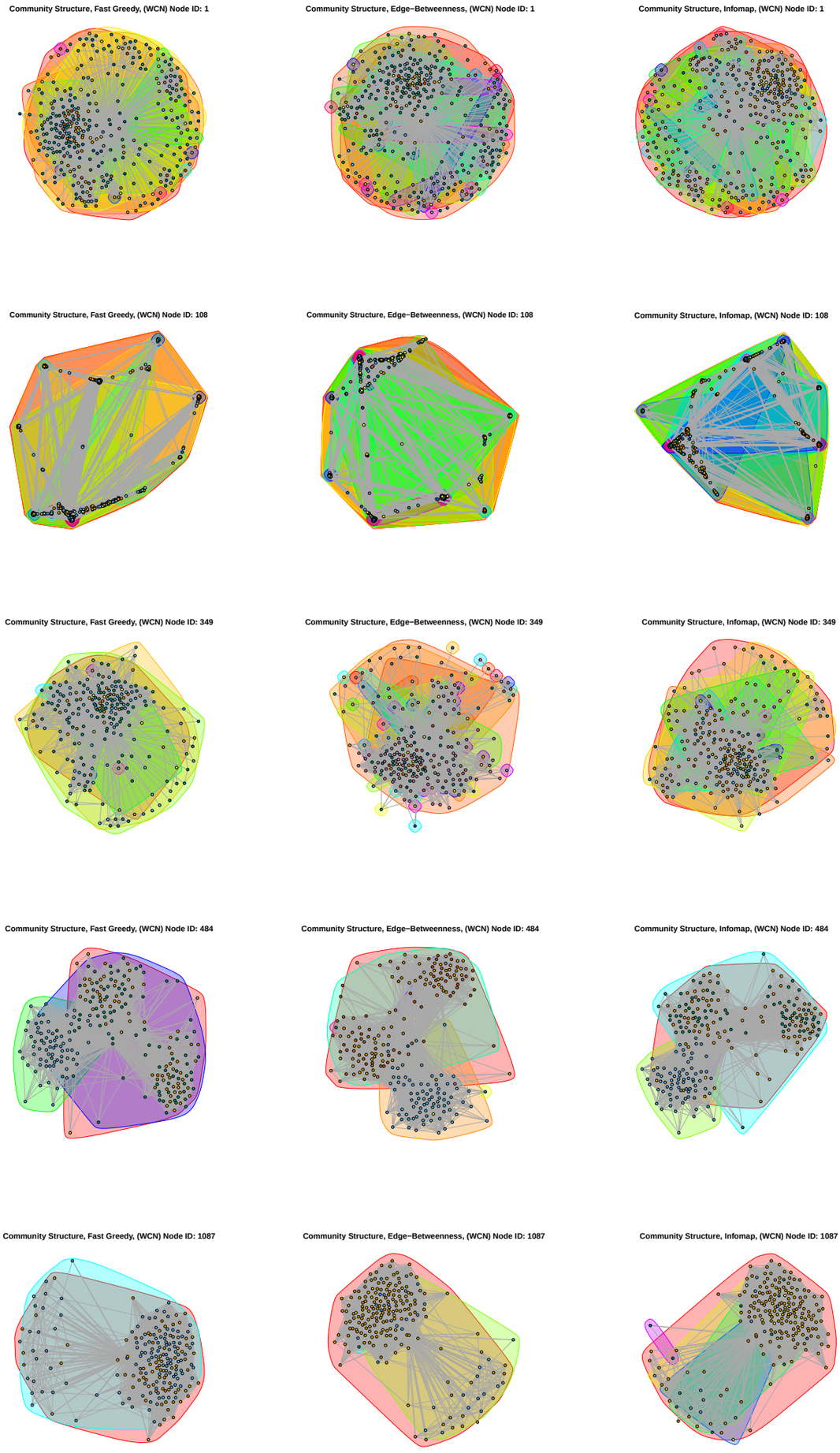


Figure 7: Community structures of personalized networks for 5 core nodes using 3 algorithms, with core node removed.

Question 11:
Write an expression relating the Embeddedness between the core node and a non-core node to the degree of the non-core node in the personalized network of the core node.

The embeddedness of a node v_i is defined as the number of mutual vertices a given node shares with the core node v_c . The embeddedness between these two variables and in the personalized network is given by the equation -

$$\text{Embeddedness}^{P}_{v_i,v_c} = deg(v_i)^{P} - 1$$

$$\text{Embeddedness}_{v_i,v_c} = deg(v_i) - N_{v_i} - 1 = |deg(v_c) \cap deg(v_i)| - 1$$

Embeddedness is defined as the overlap of two users' social circles. Here N_{vi}, is the number of neighbors the node v_i that are absent in the personalized network. This equation is constrained on the conditions that $deg(v_i)^{P}$ is the degree of v_i in the personalised network and not the original network.

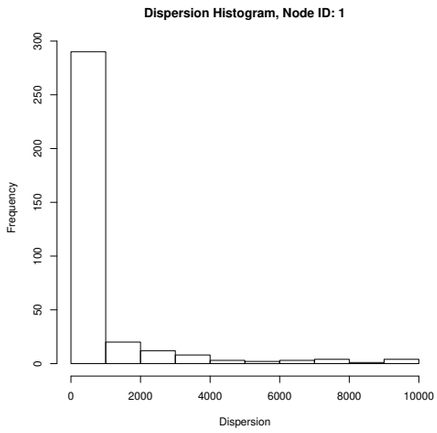
Question 12:
For each of the core node's personalized network (use the same core nodes as Question 9), plot the distribution histogram of embeddedness and dispersion. In this question, you will have 10 plots.

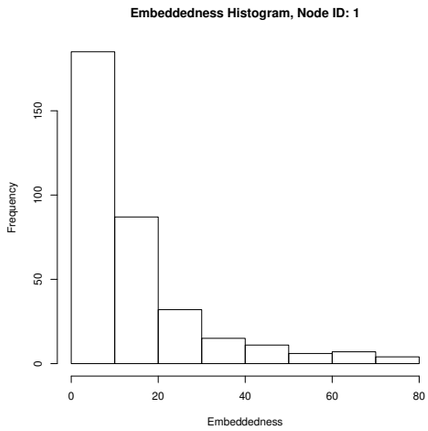
In this question, we shall plot the embeddedness and dispersion distribution histogram for each of the core node's personalized networks (node IDs 1, 108, 349, 484, and 1087). In the preceding question, we defined embeddedness. Neighbors, intersection, and distances are the functions utilized in this figure. The total of distances between every pair of mutual vertices a node shares with the core node, evaluated in a modified subgraph graph with the target node and core node removed, is the node's dispersion.

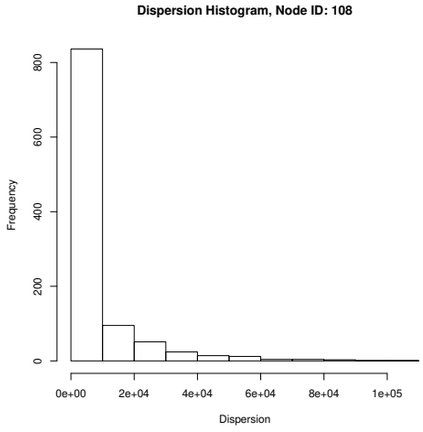
$$disp(u, v) = \sum_{s,t \in C_{uv}} d_v(s, t)$$

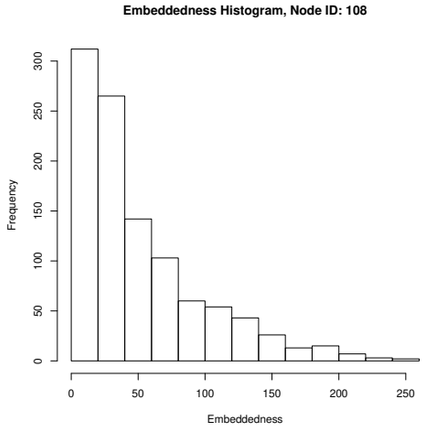
d_v in this equation is the distance function. The hisotgram plots are shown below. Our inference from these plots are -

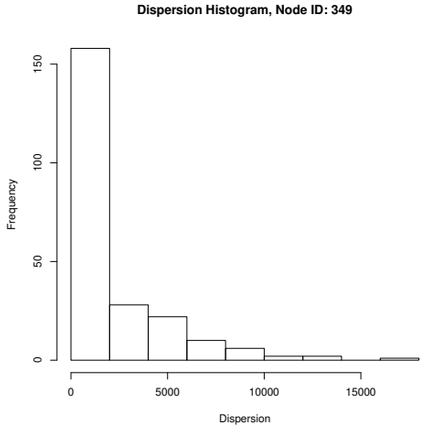
- The number of edges (degree) of the core node in the personalized network determines the range (as well as predicted value) of dispersions; the larger the number of edges, the bigger the dispersion. The expected value of embeddedness is also influenced by the amount of edges in the personalized network.
- The number of nodes in the personalized network determines embeddedness. Even if embeddedness is not a good measure of strong relationships, because tightly tied nodes might have low degrees of embeddedness, the bigger the number of nodes and connectivity among them in the social circle, the greater the embeddedness.
- Embeddedness in the personalized network has no explicit relationship with the number of mutual nodes or edges.
- Because of their inherent relationship with the degree distribution of personalized networks, which follows a power-law distribution due to weak preferential attachment, both embeddedness and dispersion generally follow the power-law distribution.

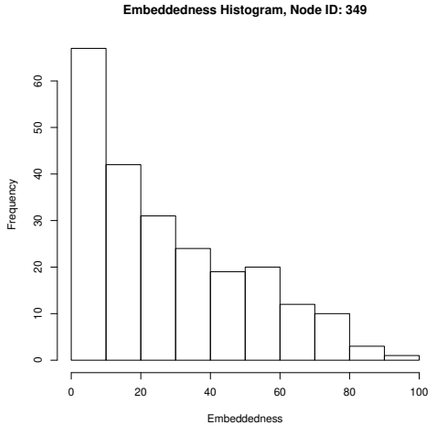

Dispersion Histogram, Node ID: 1
Frequency
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
Dispersion

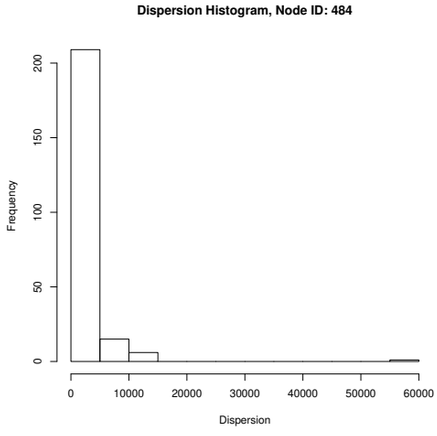

Embeddedness Histogram, Node ID: 1
Frequency
0
50
100
150
0
20
40
60
80
Embeddedness

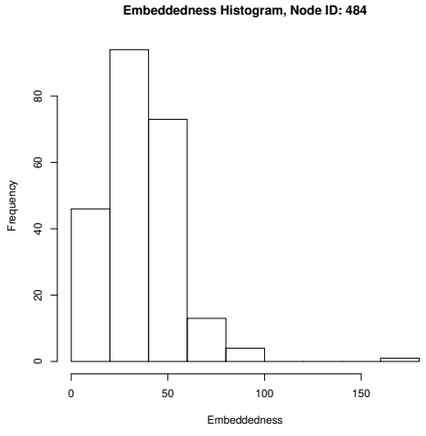

Dispersion Histogram, Node ID: 108
Frequency
0
200
400
600
800
0e+00
2e+04
4e+04
6e+04
8e+04
1e+05
Dispersion


Embeddedness Histogram, Node ID: 108
Frequency
0
50
100
150
200
250
300
0
50
100
150
200
250
Embeddedness


Dispersion Histogram, Node ID: 349
Frequency
0
50
100
150
0
5000
10000
15000
Dispersion


Embeddedness Histogram, Node ID: 349
Frequency
0
10
20
30
40
50
60
0
20
40
60
80
100
Embeddedness


Dispersion Histogram, Node ID: 484
Frequency
0
50
100
150
200
0
10000
20000
30000
40000
50000
60000
Dispersion


Embeddedness Histogram, Node ID: 484
Frequency
0
20
40
60
80
0
50
100
150
Embeddedness

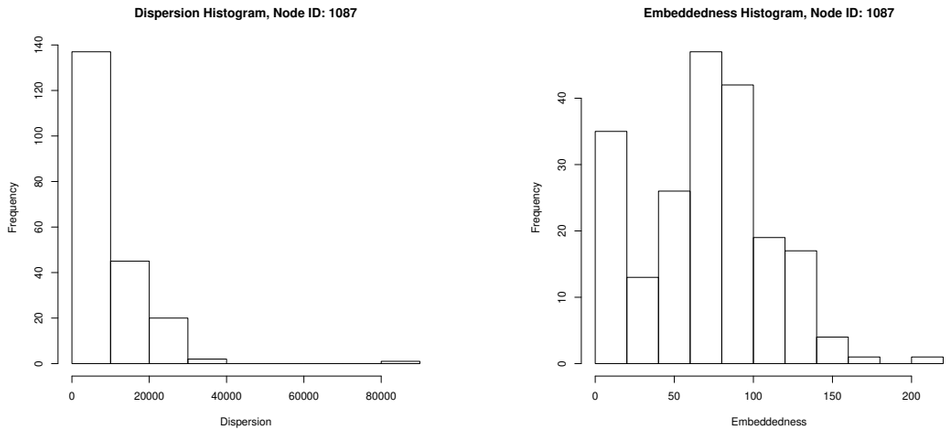


Figure 9: Embeddedness and dispersion histograms for each of the core node's personalized networks.

Question 13:

For each of the core node's personalized network, plot the community structure of the personalized network using colors and highlight the node with maximum dispersion. Also, highlight the edges incident to this node. To detect the community structure, use Fast-Greedy algorithm. In this question, you will have 5 plots.

In this question, we must plot the personalized network's community structure (using Fast-Greedy) for each of the core nodes and highlight the node with the most dispersion, as well as its incident edges. The plots are shown in the figure below. The nodes with the greatest dispersion, as well as their incident edges, are highlighted in red.

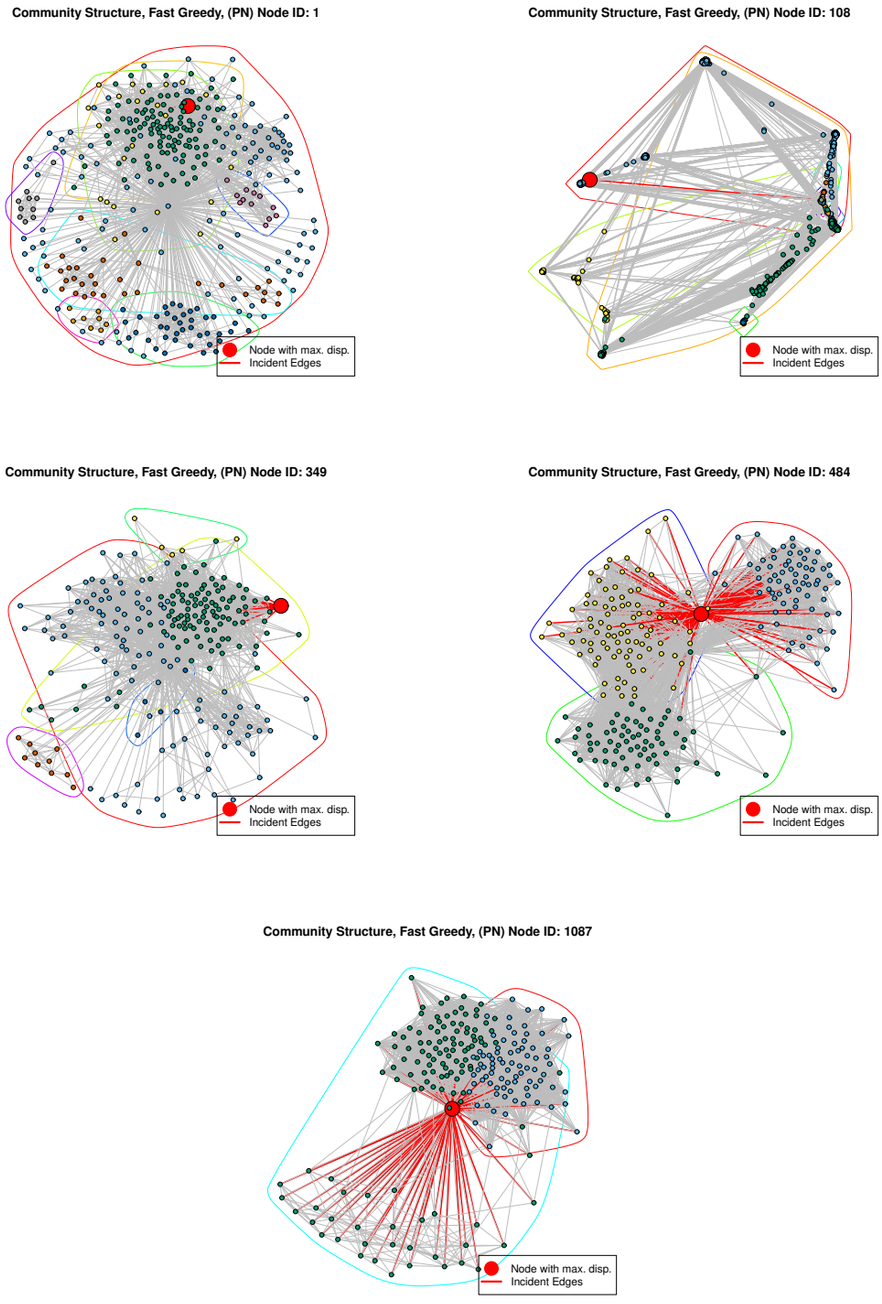


Figure 10: Community structure of personalized networks for core nodes, with maximum dispersion node and incident edges to that node highlighted.

Question 14:

Repeat Question 13, but now highlight the node with maximum embeddedness and the node with maximum ratio of dispersion, and embeddedness (excluding the nodes having zero embeddedness if there are any). Also, highlight the edges incident to these nodes. Report the id of those nodes.

In this question, we are asked to plot the community structure (using Fast-Greedy) of the personalized network for each of the core nodes and highlight the node with maximum embeddedness along with its incident edges, as well as highlight the node with maximum dispersion/embeddedness (excluding the nodes having zero embeddedness if there are any) along with its incident edges. The figure below shows the plots with nodes of maximum embeddedness highlighted in green.

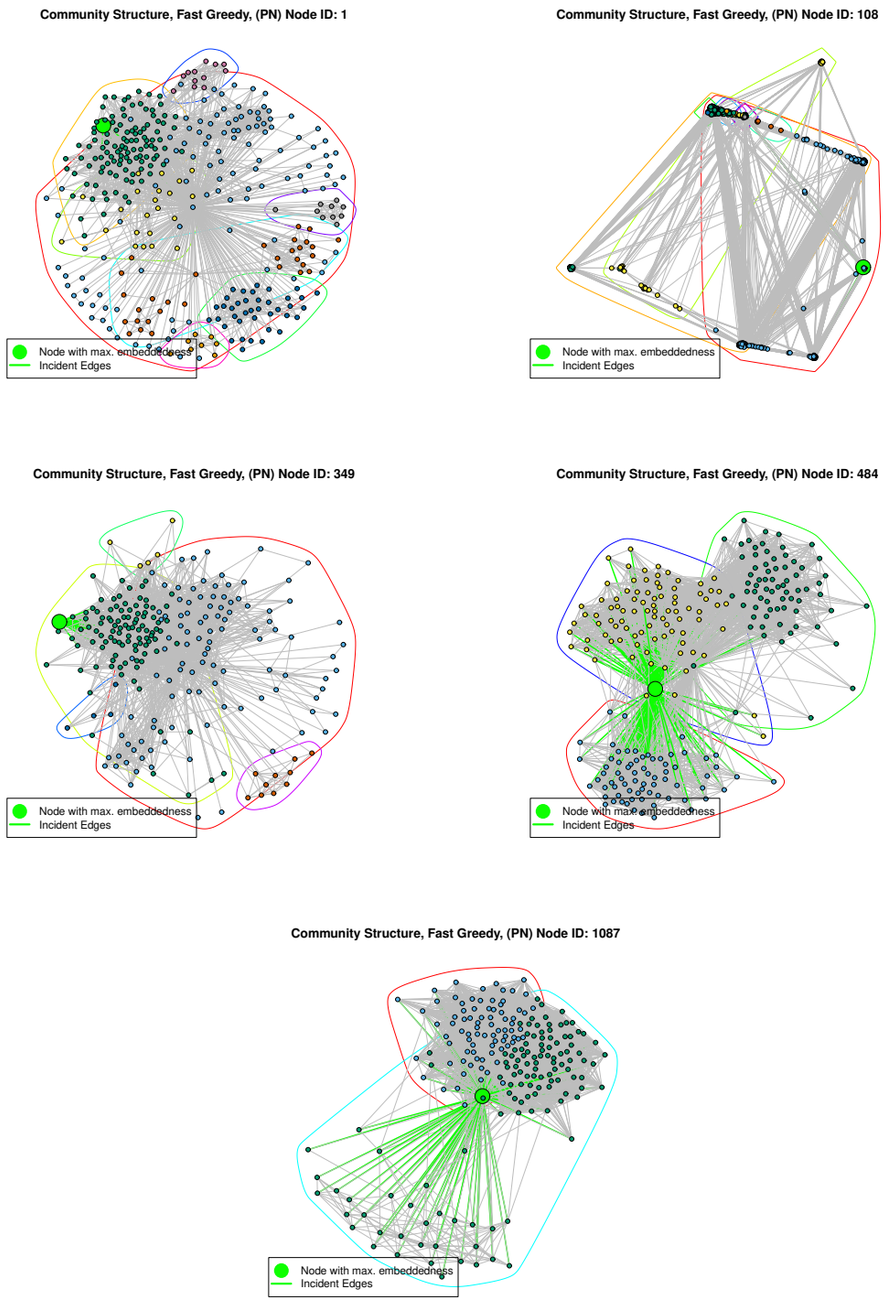


Figure 11: Community structure of personalized networks for core nodes, with maximum embeddedness node and incident edges to that node highlighted.

The figure below shows the plots with nodes of maximum dispersion/embeddedness highlighted in blue.

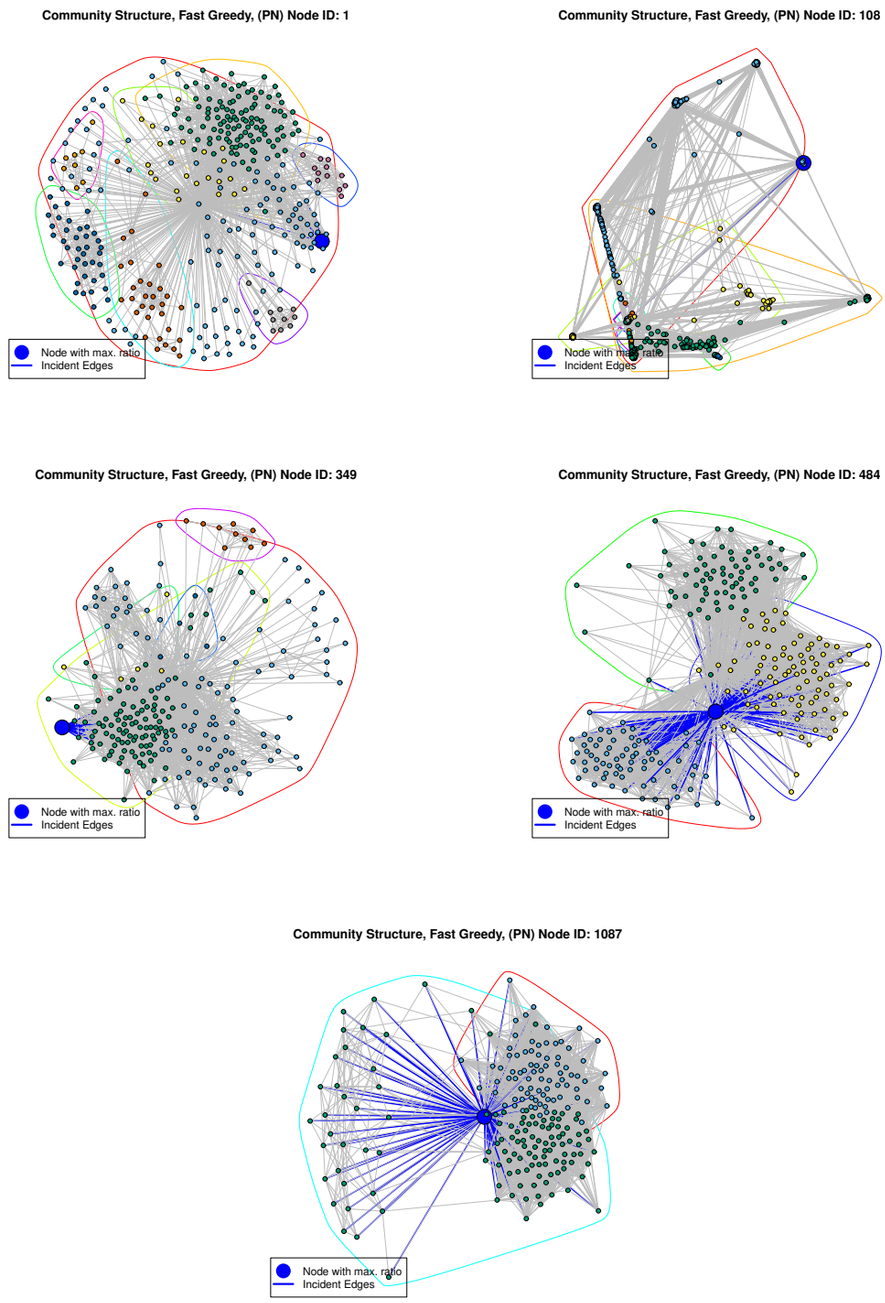


Figure 12: Community structure of personalized networks for core nodes, with maximum dispersion/embeddedness node and incident edges to that node highlighted.

Question 15:
Use the plots from Question 13 and 14 to explain the characteristics of a node revealed by each of this measure.

- Embeddedness: We defined embeddedness of a node v_i as the number of mutual vertices a given node shares with the core node v_c. The formula to obtain it directly from the original equation is given by -

$$\text{Embeddedness}_{v_i,v_c} = deg(v_i) - N_{v_i} - 1 = |deg(v_c) \cap deg(v_i)| - 1$$

  Where N_i is the number of neighbors of v_i absent in the personalized network. As a result, embeddedness is defined as the overlap of two users' social circles. The number of nodes in the personalized network determines embeddedness. The more nodes in a social circle there are and the more connected they are, the more embedded they are. The number of edges in the personalized network determines the predicted value of embeddedness. Furthermore, large communities are more likely to have a node with a high embeddedness score. In addition, the density of incident edges from the target node is substantially higher than dispersion. This is due to the fact that embeddedness is proportional to the node's degree. However, because users with strong links are likely to be affiliated with users from many communities, this is not a fair measure of the strength of ties.

- Dispersion: The sum of distances between every pair of mutual vertices the node shares with the core node, calculated in a modified subgraph graph with the target node and core node removed, is the dispersion of a node. The number of edges (degree) of the core node in the personalized network determines the range (as well as expected value) of dispersions; the larger the number of edges, the greater the dispersion of a target node. Furthermore, dispersion evaluates how well two people's common friends are connected; in other words, the mutual nodes of two strongly connected nodes are not well connected and must have a dispersed structure. As a result, because the network must have a scattered structure with connection among users from distinct groups, the density of incident edges from a node with greatest dispersion will likely be lower than a node with maximum embeddedness. Dispersion will be greater for nodes whose connections are spread out (distributed) than for nodes with tight connectivity. So, dispersion, rather than embeddedness, is a better indicator of the strength of linkages or relationships.

- Dispersion/Embeddedness Ratio: When the dispersion of a node is very high while the embeddedness is very low, the maximum value of dispersion/embeddedness (normalization of dispersion) occurs. Mutual friends from different communities with greater relationships are likely to be found in such a node. Smaller networks with dispersed structures have a greater ratio, as do users who aren't particularly connected.

Question 16:

For this question, we create a personalized network with core node 415, using the make_ego_graph() function on the original Facebook network. Next, we find the list of all node with degree 24. The length of the list was computed as 11.

Question 17:
Compute the average accuracy of the friend recommendation algorithm, and based on the average accuracy values, which friend recommendation algorithm is the best?

In this question, we are asked to compute the average accuracy of friendship recommendation algorithms using the three different neighbourhood based measures and find out which one is the best for the personalized network found in Question 16. The three measures used are -

- Common neighbours: The concept of common neighbors is founded on the idea that if two nodes have a lot of common neighbors, there's a good chance they'll link in the future. The number of mutual friends between two users is directly related to the possibility of future connectivity. The point system is founded on the assumption that people who live next door to each other will be introduced to each other by a mutual friend.
- Jaccard: In statistics, Jaccard's coefficient is used to calculate sample set similarity. All of a node's friends are denoted as a set in link prediction, and the prediction is done by ranking the similarity of the neighbor set for each pair of nodes. It is based on the idea that while two users may have a large number of mutual friends, not all of them are owing to strong links when compared to each user's total number of neighbors. The coefficient might be anywhere between 0 and 1. It adjusts for node degree and takes into consideration the relative number of common neighbors.
- Adamic Adar: It is based on the idea that if a mutual friend of two users has a large number of friends, the mutual friend is less likely to introduce the two people to one another than if the mutual friend has a smaller number of neighbors. The lower the score, the more friends a node has, therefore neighbors with fewer friends get a higher weighting. To put it another way, someone with a small number of friends is more likely than someone with a large number of friends to introduce his friends to one another. The Adamic Adar score is boosted by a mutual friend between two users who have few neighbors.

We obtain the following accuracies for the three measures:

- Common neighbours: 88.25%
- Jaccard: 84.81%
- Adamic Adar: 88.21%

The Common neighbours method, followed by Adamic Adar and Jaccard, performs the best. It's worth noting that the accuracies for all three networks are really close (within 4% of each other). Furthermore, Adamic Adar and common neighbors function nearly identically, with only a 1% difference in accuracy. Our conclusions from this question is :

- The coefficient of Jaccard is based on the idea that only nodes with strong ties contribute to future friend recommendations (intersection over union). With the implied premise that two users with too many friends may not have significant relationships

inside their communities, the Jaccard's coefficient will be low if the degree of the target nodes is large. The Jaccard's coefficient, on the other hand, performs significantly worse than just taking the intersection over mutual friends if the nodes within a network are not themselves closely linked in a network (common neighbours). This is especially noticeable in small networks, where the likelihood of strong linkages is lower than in larger networks.

- The algorithms' accuracies are all within 4% of one another. This is due to the fact that we are working with a tiny personalized network with fewer nodes and edges than large social networks. In addition, we only propose friends to users who have a limited number of friends. The network's reach does not include all implicit assumptions made by each of the measurements, opting instead for the most prevalent ones, which does not result in substantial changes in link prediction algorithm performance. To put it another way, we're calculating on local data rather than global similarity features, which doesn't accurately reflect how large-scale social network friendship suggestions function.

2 Google+ Network

The structure and features of the Google+ network were then examined. We utilized the data provided at `http://snap.stanford.edu/data/egonets-Gplus.html`, specifically gplus.tar.gz, to do this. We unzipped the data to gain access to it. For those who had built more than two circles of friends, we developed personal networks.

Question 18:
Out of 132 ego-networks, there are 57 personal networks. We discovered this by reading all of the.circles files in the dataset and counting the number of lines in each.circles file. Personal networks are networks for people who have more than two circles.

Or, in other words, in this dataset, there are a total of 132 personalized networks. There are 57 users who have more than two circles.

Question 19:

Figures below show the in-degree and out-degree for 3 personalized networks with node IDs 109327480479767108490, 115625564993990145546 and 101373961279443806744. Mathematically, the in-degree and out-degree of node i in a network is given by:

$$deg(i)^{in} = \sum_{i} A_{ij}$$

$$deg(i)^{out} = \sum_{j} A_{ji}$$

where A_{ij} is defined as the node-node incidence matrix or adjacency matrix.

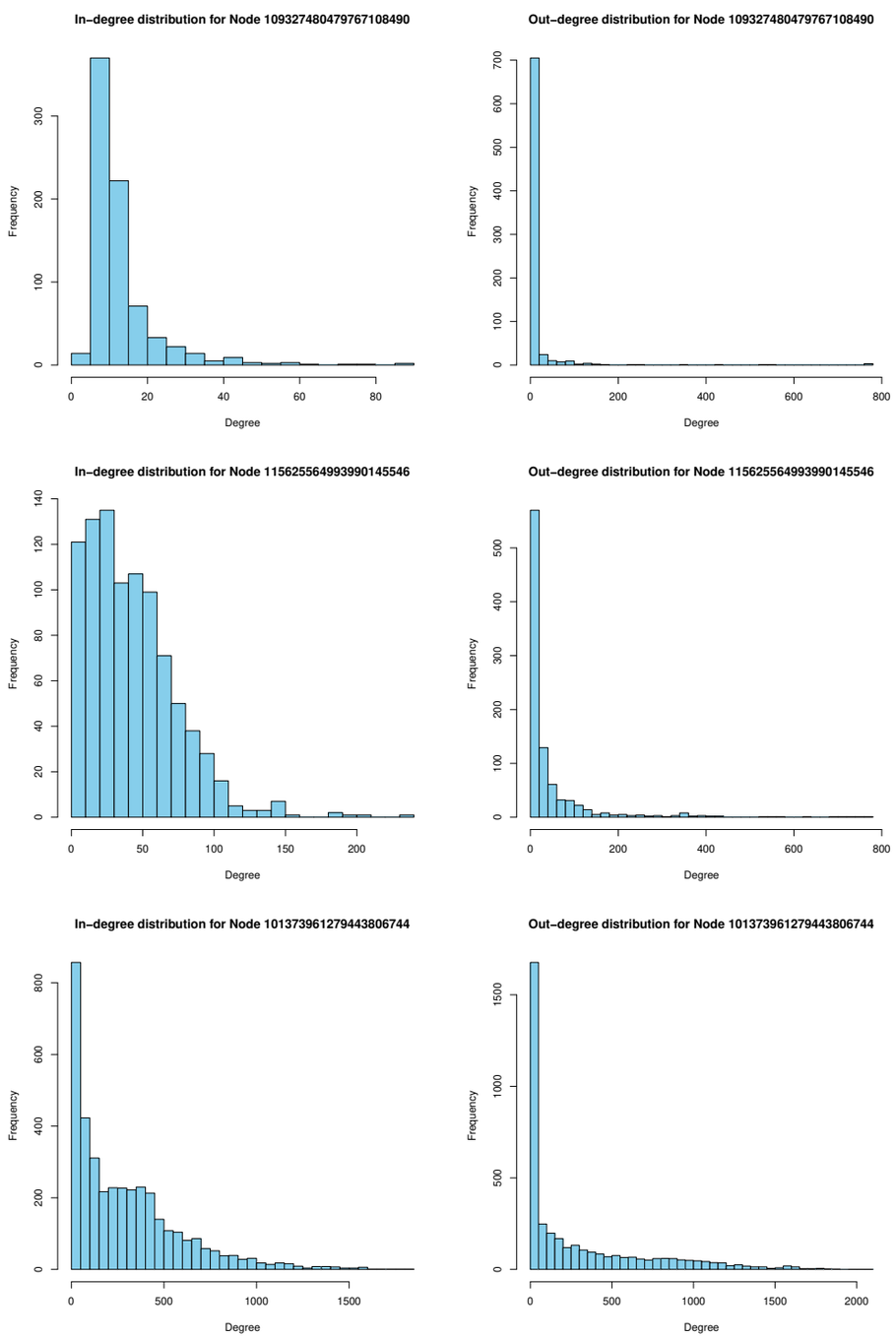


Figure 13: In-degree and out-degree of the 3 personalized Nodes in Google+ dataset

The above figure shows that the out-degree distributions of all three networks are quite similar, following a power-law distribution. The in-degree distributions vary among the three networks and broadly follow the power-law distribution (though not as closely as out-

degree distributions). The in-degree and out-degree distributions are significantly different for each network.

- The customized network of node 109327480479767108490 is the most severely skewed to the right of the three out-degree distributions.
- While the expected values from the out-degree distributions are similar for personalized networks with nodes 109327480479767108490 and 115625564993990145546, the expected in-degree values differ, with node 115625564993990145546 having a higher value than node 109327480479767108490. Furthermore, the network's in-degree distribution with node 115625564993990145546 is essentially linear. This means that there are more incoming connections to each node on the network with node 15625564993990145546 than on the network with node 109327480479767108490. This indicates that, as compared to the network with node 109327480479767108490, the network with node 115625564993990145546 generates communities of strong connectivity with sparse inter-community connections rather than relying on a few popular individuals or hubs.
- The network's out-degree distribution with node 101373961279443806744 rolls off the slowest, showing a lack of defined community structures and minimal modularity.
- Personalized networks appear to taper off exponentially as the degree grows in nodes 109327480479767108490 and 101373961279443806744. The highest in-degrees are held by a very tiny proportion of users in both networks. These might be considered "hubs," or exceptionally prominent persons inside the network to whom most people connect. If these hubs are removed, the network will most certainly suffer greatly. In contrast, removing one of the low-degree nodes may not significantly alter the network topology.
- The customized network of Node 115625564993990145546 looks to have a more linear fall-off, thus a bigger proportion of the users have a reasonably high in-degree compared to the other two networks. This might imply that the network is less reliant on "hubs" and that members are more closely related in terms of popularity. Across all three customized networks, the out-degree distributions appear to be the same.

Question 20:

In this question, we are asked to extract the community structures (using walktrap community detection) and modularity scores of the three personalized networks from Question 19. Figure below shows the community structures.

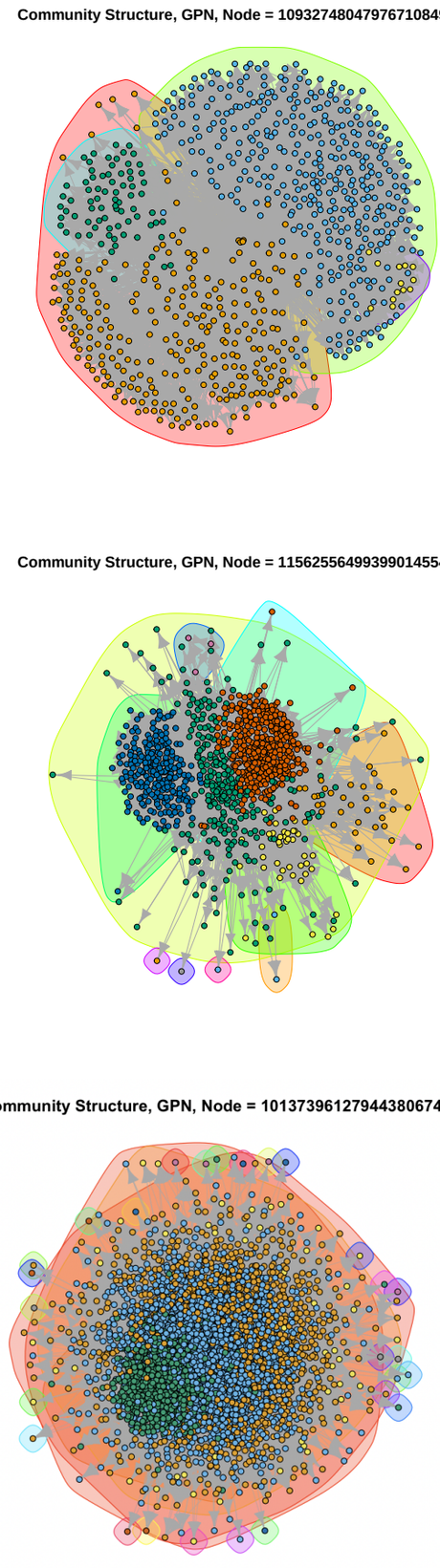


Figure 14: Community structure of three personalized networks from Google+ dataset

The Modularity scores are given below:

Ego Node ID	Modularity
109327480479767108490	0.25276535939251
115625564993990145546	0.319472554647349
101373961279443806744	0.191090282684037

Table 3: Modularity scores of personal networks using Walktrap

Figure above and the modularity scores show that the modularity scores are not the same. The lowest modularity score is of Node 101373961279443806744, suggesting that the network is least capable of being separated into densely packed modules with high connectivity and few links across communities. Figure above shows that the majority of the nodes in the personalized network for node 101373961279443806744 are grouped in a single chunk in the center of the plot. We also discovered that the customized network for node 101373961279443806744 has the most edges m and nodes of any network. A higher value of m implies that an entering node is linked to a greater number of older nodes. While this should result in significant intra-community connection, the global sparsity across different communities is lost owing to the connectedness requirement imposed by large values of m, resulting in edges forming among otherwise separate clusters and therefore weakening community structures. Mathematically,

$$Q(P) = \sum \frac{1}{2m} \sum_{i,j \in C_i} (A_{i,j} - \frac{k_i k_j}{2m})$$

If the number of edges in the network m(not to be confused with the actual m we are talking about, which is the number of old nodes the incoming node connects with) increases, then Q(P) drops. As a result, less clusters are formed in the overall graph.

Node 115625564993990145546 has the greatest modularity score. Figure above shows that dense communities or clusters are forming in the community structure plot, with sparse connectedness between these clusters.

Question 21:

Homogeneity and completeness are two criteria for assessing clustering algorithm outcomes. The two frequently have a negative association.

Homogeneity:

A clustering result is homogeneous if all of its clusters include only data points from a single class or circle. Homogeneity h is defined mathematically as the conditional entropy of labels or circles C given cluster assignments or circles K, and it is independent of the absolute value of ground truth labels. The entropy of partition X is denoted by H(X), where X signifies

$$h = 1 - \frac{H(C|K)}{H(C)}$$

$$H(C|K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{n_{c,k}}{n}\cdot\log\left(\frac{n_{c,k}}{n_k}\right)$$

$$H(X) = -\sum_{h=1}^{|H|}\frac{n_h}{n}\cdot\log\left(\frac{n_h}{n}\right)$$

$$n_{c,k} = n_k \cap n_c$$

non-overlapping sets of sample points. When the partition sizes are equal, the information entropy is maximized, and it is minimized when some group inside the partition takes up all of the data points. When each cluster K_i contains just data from C_i, the homogeneity score is maximized, i.e. $H(C|K) = 0$.

The amount of variance in each expected community is measured by homogeneity. A greater homogeneity score is obtained if every anticipated community has people who all share the same circle. The lowest homogeneity score is obtained when members of a forecast community are evenly distributed throughout all feasible circles.

Completeness:

A clustering result is complete if all of the data points in a particular circle are members of the same community. Completeness is defined mathematically in terms of conditional entropy of clusters K given ground truth labels C.

$$c = 1 - \frac{H(K|C)}{H(K)}$$

Completeness if maximized when each ground truth class C_i is part of some cluster K_i. In the ideal case (c = 1), a single cluster should encompass all members of a circle.

The completeness measure examines each circle and determines the degree of variance in each circle member's community assignment. When every member of a circle is anticipated to belong to the same community, the circle has a high completeness score. If a circle's distribution of community assignments is consistent, the circle has a low community score.

Question 22:
The h and c values for the 3 personalized nodes are summarized in the table below.

Ego Node ID	Homogeneity, h	Completeness, c
109327480479767108490	0.8518851	0.3298739
115625564993990145546	0.4518903	-3.423962
101373961279443806744	0.003866707	-1.504238

Table 4: Homogeneity and Completeness scores for three personalized networks in Google+ network

Observations we make from the scores in the above table:

- The degree-distribution revealed that the graph was biased towards low-degree users with a tiny minority of high-degree hubs, which might explain why the Walktrap algorithm failed to discover groups that corresponded well with the circles. This raises the likelihood that the walker will jump to a popular hub and subsequently to another community during random Walktrap treks, rather than staying inside its own circle.

- One explanation for the high homogeneity score is that 28 of the 31 communities detected in this network had no members who also circled ata. Because homogeneity falls with the number of conditional entropies contributed by each community, and these 28 communities with no circle-data users produce 0 conditional entropy, homogeneity stays high. We could anticipate the homogeneity score to be lower if all of the community members have circle data.

- We find a reasonably high degree of homogeneity in the network with node ID 109327480479767108490 This implies that the majority of members in a community belong to the same circle. The completeness score is low yet positive, indicating that some users from one circle have not been grouped into a single community. Figure 13 demonstrates these facts. In terms of completeness, we can see that not all orange dots have been assigned to the red community; instead, some have been assigned to the yellow and green communities. However, the completeness score is still significantly greater than the other two networks. The scores indicate that the members of the personalized network of node ID 109327480479767108490 are well-segregated into individual communities with few overlaps among different circles.

- The homogeneity score for the network with node ID 115625564993990145546 is nearly half that of the network with node ID 09327480479767108490. This suggests that more members from different circles in this network have been incorrectly classified as belonging to the same community. The community structure graphic in Figure 13 demonstrates this. Furthermore, the completeness score is both low and negative. This implies that $H(K|C) > H(K)$. This can occur when some of the circles are not assigned to any community and are grouped together (each community has a few circles). In other words, a negative completeness suggests a sparse C and a significant mismatch between the number of circles and the number of communities.

- With the network with node ID 101373961279443806744, the homogeneity score is the lowest of all three, indicating that most people from different circles in this network have been incorrectly classified into a single community. This is also obvious from

the low modularity scores, which show that this network is least capable of being partitioned into dense communities with sparse inter-community linkages. In other words, the communities are not well-separated. The completeness score is once again negative but greater than the network with node 115625564993990145546, showing that $H(K|C) > H(K)$. The causes for negative completeness were explored in the preceding point.

- There is no discernible community structure seen in the community clustering diagram for Node 101373961279443806744. For the larger communities, there is a substantial degree of overlap. It is not unexpected to see a very low completeness score because each of the projected communities shares users, therefore regardless of whether there is user overlap within each circle, the completeness score will be decreased.

Question 23:
Use Graph Convolutional Networks. What hyperparameters do you choose to get the optimal performance? How many layers did you choose?

Academic articles are the nodes in the Cora dataset, and citations between them are the links: if publication A cites publication B, the graph has an edge from A to B. Our machine will learn to predict the subject by classifying the nodes into one of seven categories. A Graph Convolution Network is used in this question (GCN). A "graph convolution" layer lies at the heart of the GCN neural network model. This layer is identical to a traditional dense layer, but with the addition of the graph adjacency matrix, which allows information about a node's relationships to be used. In this question, there are three main components: data preparation, GCN layer creation, and training and evaluation.

The libraries involved are Pandas, Tensorflow, Keras, and StellarGraph. The design process for this question has been described below :

- The first step is to import the necessary Python libraries. For convenience, we import stellargraph as sg, similar to how pandas is frequently imported as pd. Using the Cora loader from the datasets submodule, we can get a StellarGraph graph object containing this Cora dataset. It also gives us subject classes for ground-truth nodes.
- The Cora dataset contains 2708 scientific publications that are divided into seven categories. There are 5429 linkages in the citation network. A 0/1-valued word vector describes each publication in the dataset, indicating the existence or absence of the associated word from the dictionary. There are 1433 distinct terms in the dictionary. We want to train a graph-ML model that can predict the node's "topic" attribute.
- We'll use a subset of the nodes for training and the rest for validation and testing in machine learning. We'll utilize the train test split function from scikit-learn. We'll use 140 node labels for training, 500 for validation, and the remaining for testing.
- Class imbalance in the training set may need to be adjusted, for example, by applying a weighted cross-entropy loss in model training, with class weights inversely proportional to class support. For the sake of simplicity, we shall overlook the class disparity in this scenario.

Creation of GCN layers :

- The layers themselves, such as graph convolution, dropout, and even traditional dense layers, are part of the model, as is a data generator that converts the fundamental graph structure and node properties into a format that can be fed into the Keras model for training or prediction. The FullBatchNodeGenerator class is the ideal generator for our assignment because GCN is a full-batch model and we're doing node classification.
- layer_sizes: the number of hidden GCN layers and their sizes, activations: the activation to apply to each GCN layer's output, dropout: the rate of dropout for the input of each GCN layer.
- In the first model, we used two GCN layers with 16 units each. A ReLU activation function, and dropout factor of 0.5. In the second model, we used two GCN layers

with 32 and 64 units respectively. The dropout factor is 0.5. Apart from this, there are additional dense layers with 30 units, and a dropout factor of 0.4. The last layer in both the models is a prediction layer with a softmax activation function.

- Model Training and evaluation - Adam optimiser with a learning rate of 0.01 was used. The model has been trained for 200 epochs.

The prediction values on the test set and the plots are reported below.

Table 5: Metrics for a GCN

Method	**Accuracy (%)**	**F-1 Score**
GCN	77.90	0.7814
GCN+Dense Layers	79.79	0.8211

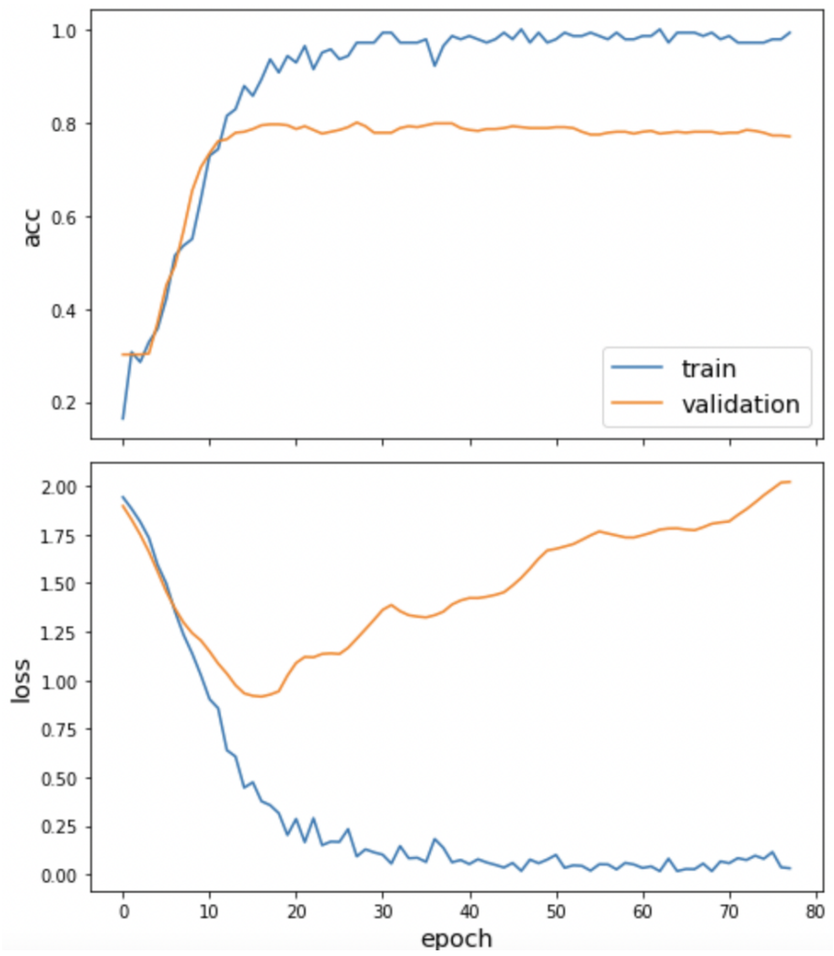


Figure 15: Training, Validation Loss and Accuracy Plot using GCN

It can be good to have a more complete picture of what information the model has learned about the nodes and their surroundings in addition to just predicting the node class. This implies embedding the node into a latent vector space that captures that information, which can take the form of a look-up table that maps the node to a vector of numbers or a neural network that generates those vectors. We'll use the second option for GCN, applying the prediction layer before the last graph convolution layer of the GCN model.

These embeddings can be seen as points on a graph, colored by their true subject labels. We should anticipate to observe attractive clusters of papers in the node embedding space if the

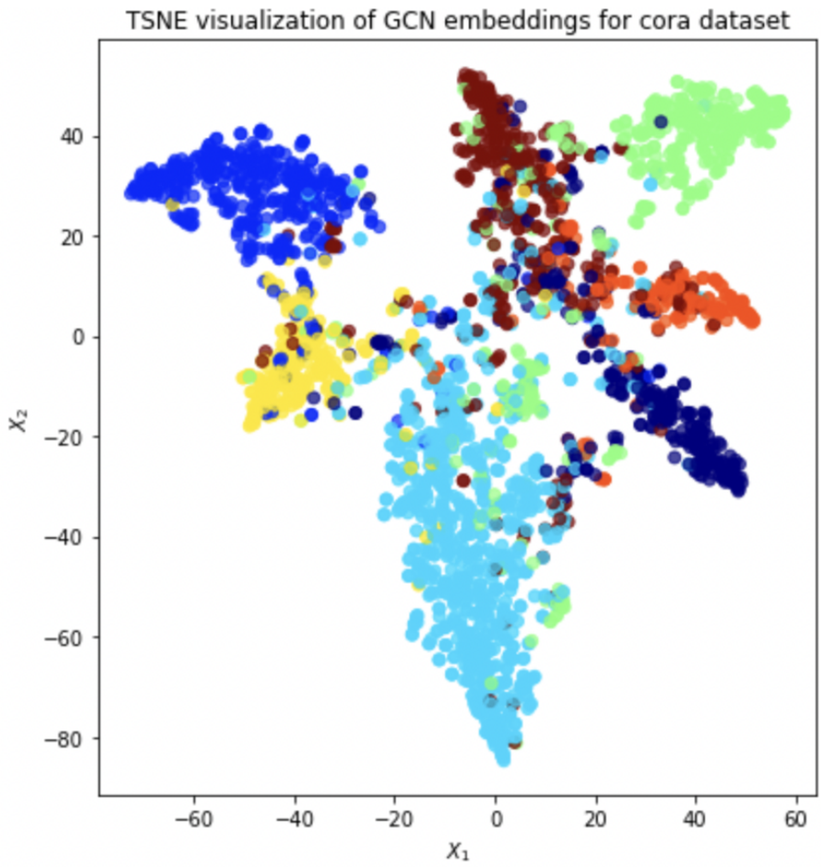


Figure 16: Visualization of GCN embeddings

model has learned relevant information about the nodes based on their class, with papers of the same subject belonging to the same cluster.

The second model of GCN with 32 and 64 units followed by Dense Layers of 30 units gave us the best results. The model was fit with an adam optimiser with learning rate of 0.01 for 200 epochs.

Question 24:
In this question we extracted structure based node features using Node2Vec used SVM classifier for classification. We also used 1433 dimensional text features obtained using tf-idf for classification and in the third part of the question we were asked to combine the features and use this as input data for classification.

Node2Vec :
A notable problem when working with networks, is transforming the network structure into numerical representation which can then be passed onto traditional machine learning algorithms. Node2vec is an algorithmic framework for representational learning on graphs.

Given any graph, it can learn continuous feature representations for the nodes, which can then be used for various downstream machine learning tasks. Node2Vec is an algorithm that allows the user to map nodes in a graph G to an embedding space. Generally, the embedding space is of lower dimensions than the number of nodes in the original graph G. The algorithm tries to preserve the initial structure within the original graph. Essentially, the nodes which are similar within the graph will yield similar embeddings in the embedding space. These embedding spaces are essentially a vector corresponding to each node in the network. The graph embeddings are commonly used as input features to solve machine learning problems oriented around link prediction, community detection, classification, etc.

Node2Vec is a two step representation learning algorithm. The steps are as follows:

- Use second-order random walks to generate sentences from a graph. A sentence is a list of node ids. The set of all sentences makes a corpus.
- The corpus is then used to learn an embedding vector for each node in the graph. Each node id is considered a unique word/token in a dictionary that has size equal to the number of nodes in the graph. The Word2Vec algorithm, is used for calculating the embedding vectors.

We use the stellargraph library to implement Node2Vec. To implement Word2Vec, to learn representations for each node in the graph, we have used a python library called gensim. Here, Word2vec is a technique for natural language processing published in 2013. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector. The vectors are chosen carefully such that a simple mathematical function (the cosine similarity between the vectors) indicates the level of semantic similarity between the words represented by those vectors. We have used SVM as our classifier.

Tf-idf Text Features :

We have occurrence count information in our dataset but the issue with occurrence count is that longer documents will have higher average count values than shorter documents, even though they might talk about the same topics. To avoid these potential discrepancies it suffices to divide the number of occurrences of each word in a document by the total number of words in the document: these new features are called tf for Term Frequencies. Another

refinement on top of tf is to downscale weights for words that occur in many documents in the corpus and are therefore less informative than those that occur only in a smaller portion of the corpus. This downscaling is called tf–idf for "Term Frequency times Inverse Document Frequency". We used tf-idf to find all the text features from the inputs given in the dataset. We passed this through SVM classifier and got 62.60 % accuracy.

Combination of Both Node2Vec and Text Features :

To improve the accuracy of these methods we tried data fusion. Data fusion is the process of integrating multiple data sources to produce more consistent, accurate, and useful information than that provided by any individual data source. Hence, we fused the features obtained from node2vec and text features by concatenating them and passed this data through an SVM classifier and got an accuracy of 59.61%. The concatenated data actually gave us the least accuracy. The reasons for the same are listed later in the section.

We used SVM classifier to classify multiclass Cora dataset into their respective classes. We used 140 samples to train data and remaning samples to test it. In machine learning, support-vector machines are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. A support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification.

Tha table below shows results for all three methods. We have considered accuracy, F1 score, Precision, recall values of all three methods. We had initially expected that fusion of the two features would give us better performance than both the kinds of data passed through the network individually but as can be seen from the values concatenation of features actually worsened the performances.

We have used following metric to measure the performance of our classifier on different methods.

- Accuracy: Accuracy is the ratio of correctly predicted observation to the total observations.
- F1 Score: F1 score is the harmonic mean of precision and recall.
- Recall: Model recall score represents the model's ability to correctly predict the positives out of actual positives.
- Precision: The model precision score measures the proportion of positively predicted labels that are actually correct.

Table 6: Metrics for Classification using SVM

Method	**Accuracy (%)**	**F-1 Score**	**Recall**	**Precision**
Node2Vec	74.28	0.7260	0.7430	0.7212
Text Features	62.60	0.5326	0.5206	0.5817
Combined Features	59.61	0.5045	0.4957	0.5488

We can make the following observations from the table:

- It can be seen from the above table that structure based node features obtained using Node2Vec method perform the best among the given method. As we know, the node2vec framework learns low-dimensional representations for nodes in a graph by optimizing a neighborhood preserving objective. The objective is flexible, and the algorithm accomodates for various definitions of network neighborhoods by simulating biased random walks. Specifically, it provides a way of balancing the exploration-exploitation tradeoff that in turn leads to representations obeying a spectrum of equivalences from homophily to structural equivalence.

- From these results which show Node2Vec performs better than the other two methods we can also say that learning useful representations from highly structured objects such as graphs is useful as besides reducing the engineering effort, these representations can lead to greater predictive power.

- Text features lack the structure of Node2vec has, hence we expected that the performance on text features would be worse than node2vec. Although, we had imagined that fusion of two kinds of features will give us better performance than at least the lesser performing featur but it actually gave the least accuracy value among the three. Hence, we can also infer that fusion of data does not always improve the performance. Choosing the right classifier and fusing the correct features is crucial.

Question 25:
In this question, we are asked to simulate PageRank without teleportation using random walk. The Google search engine uses the PageRank algorithm to compute global significance ratings or pagerank scores, which takes advantage of the node's graphical structure. While PageRank has shown to be quite effective, the web's fast development in size and diversity is driving a growing demand for greater ranking flexibility. Each user should, in theory, be able to set their own sense of significance for each question.

Personalized PageRank (PPR) is a widely used node proximity measure in graph mining and network analysis. Given a source node s and a target node t, the PPR value $\pi(s, t)$ represents the probability that a random walk from s terminates at t, and thus indicates the bidirectional importance between s and t. Personalized PageRank (PPR), as a variant of PageRank, focuses on the relative significance of a target node with respect to a source node in a graph.

We performed 3 operations:

- Pagerank without Teleportation without TF-IDF
- Pagerank without Teleportation with TF-IDF
- Pagerank with Teleportation with TF-IDF

As we know each algorithm differs slightly from the previous one. We also use Cosine Similarity between the 2 nodes. Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0 degrees is 1, and it is less than 1 for any angle in the interval $(0, \pi]$ radians. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors oriented at 90 degrees relative to each other have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.

TF-IDF is just a way to measure the importance of tokens in text; it's just a very common way to turn a document into a list of numbers (the term vector that provides one edge of the angle you're getting the cosine of).

To compute cosine similarity, you need two document vectors; the vectors represent each unique term with an index, and the value at that index is some measure of how important that term is to the document and to the general concept of document similarity in general.

You could simply count the number of times each term occurred in the document (Term Frequency), and use that integer result for the term score in the vector, but the results wouldn't be very good. Extremely common terms (such as "is", "and", and "the") would cause lots of documents to appear similar to each other. (Those particular examples can be handled by using a stopword list, but other common terms that are not general enough to be considered a stopword cause the same sort of issue.)

TF-IDF adjusts the raw term frequency by taking into account how frequent each term

occurs in general (the Document Frequency). Inverse Document Frequency is usually the log of the number of documents divided by the number of documents the term occurs in.

Below are few observations that we made on the obtained results:

- Random walks without Teleportation and without tfidf did much worse than the walks with Teleportations. This makes sense intuitively as it implies that with teleportation enabled, the random walker's chance of only visiting highly related nodes has dropped, as he now reaches some lower degree nodes as well. Pagerank with vanilla teleportation, on the other hand, has a smaller slope, indicating that the number of nodes visited by the random walker in customized Pagerank is still more than pagerank with vanilla teleportation. This is due to the fact that personalized Pagerank teleportation only occurs to nodes with a high Pagerank score, wiping out the benefit of vanilla teleportation to nodes with a low Pagerank score, as a low Pagerank score indicates a lesser degree of nodes. Pagerank with teleportation combines Pagerank and Pagerank without teleportation.
- Random walk with Teleportation with and without TF-IDF do not vary very much in their accuracies.
- We also notice the precision, recall, f1-scores are higher for nodes 1,2 and 3 than the other nodes.
- The number of Unvisited nodes is higher for the teleportation methods than without teleportation. This makes sense, as teleportation takes the random walker to one of the seed documents of that class (with a uniform probability of 1/20 per seed document)
- We also notice that as we increase the probabilities for the teleportation, the number of unvisited nodes increases. We notice that for teleportation probability 0.1, the unvisited nodes lies around 20, but for teleportation probability 0.2, the univisted nodes is more than double the values.

The below table summarizes the result for the 3 operations mentioned above. The Table consists of only the accuracy values. The code consists of the recall, precision, f1-score and the support values for all the runs. The macro average and the weighted average values are also mentioned in the code. These scores help us understand which method works better on an average and also helps us understand more about the architecture. (Also, note that the values of Accuracy, precision, etc. varies over various runs as everything is randomized. Hence the values in the python notebook may be slightly different from the values mentioned in the table.)

Teleportation	Teleportation Probability	TF-IDF	Accuracy(%)	f1-score
No	0	No	35.492	0.3242
Yes	0.1	No	71.1871	0.7028
Yes	0.2	No	74.245	0.7342
No	0	Yes	37.062	0.3414
Yes	0.1	Yes	71.790	0.7014
Yes	0.2	Yes	70.6639	0.6957

Table 7: Pagerank Random Walk for various permutations of Teleportation and TF-IDF

Results for Cora Dataset:
Over the last 3 questions, we used "Cora" dataset and tried to tackle a classification issue in situations when additional information is offered in the form of "graph structure." Cora is a collection of 2708 Machine Learning-related articles. Each paper is tagged with one of the seven classifications listed below: Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, Theory. Only 20 papers are identified in each class (a total of 140 for the seven classes). They are known as "seed" papers. Each document has a collection of textual characteristics. These are instances of 1433 words from the lexicon.

The results for all the methods are summarized in the table below.

Method	Teleportation Probability	TF-IDF	Accuracy(%)	f1-score
GCN	N/A	N/A	77.90	0.7814
GCN + Dense Layers	N/A	N/A	79.79	0.8211
Text Features	N/A	Yes	60.60	0.5326
Node2vec - tfidf fusion	N/A	Yes	59.61	0.5045
Pagerank Random Walk	No, 0	No	35.49	0.3242
Pagerank Random Walk	Yes, 0.1	No	71.18	0.7028
Pagerank Random Walk	Yes, 0.2	No	74.24	0.7342
Pagerank Random Walk	No, 0	Yes	37.06	0.3414
Pagerank Random Walk	Yes, 0.1	Yes	71.79	0.7014
Pagerank Random Walk	Yes, 0.2	Yes	70.66	0.6957

Table 8: Compiled Results for Q23 - Q25

We can notice that GCN and GCN with Dense Layers does better than the other methods. We know that Graph Convolution Networks do much better than Random walk and Node2vec. The difference between Node2vec and GCN is subtle but significant. Node2vec features a walk bias variable α, which is parameterized by p and q. The parameter p prioritizes a breadth-first-search (BFS) procedure, while the parameter q prioritizes a depth-first-search (DFS) procedure. The decision of where to walk next is therefore influenced by probabilities 1/p or 1/q.
Pagerank and GCN are the most popular and SOTA models which are used to solve many network mining tasks.