UCLA

232E - Large-Scale Social and Complex Networks: Design and Algorithms

Report on

Project 2: Social Network Mining

Authors
Jayanth Shreekumar (805486993)
Leo Ly (805726182)
Yuheng He (505686149)

Spring 2022

Facebook Network

Question 1.1

The snap dataset consists of circles of friends from Facebook. We created the graph using the edgelist given in facebook_combined.txt using read.table() and then creating a graph using graph.data.frame(). The graph is displayed below:

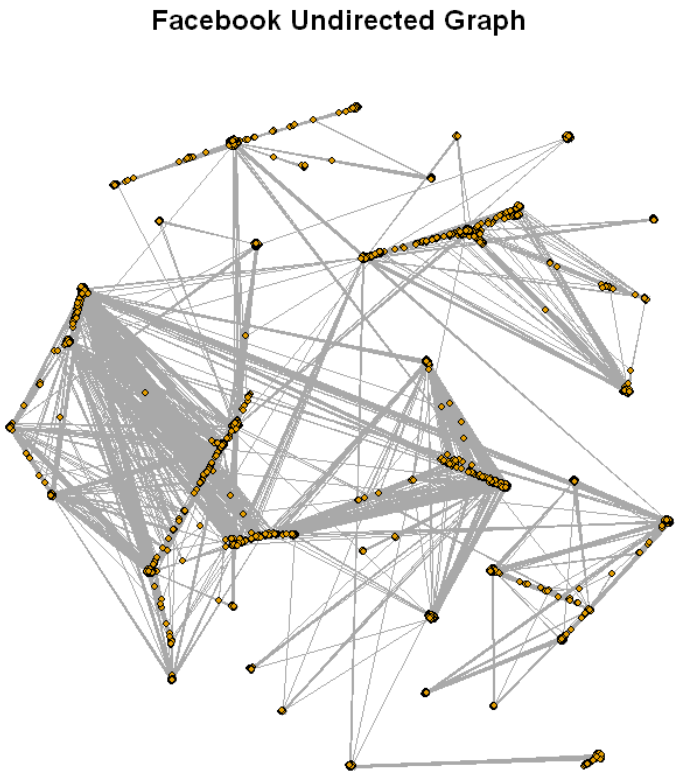


Figure 1: SNAP Facebook Graph Visualized

The number of nodes of a graph is given by gorder() and was found to be 4039 while the number of edges is given by gsize() and was found to be 88234.

Question 1.2

We checked for connectivity of the graph using is_connected() and the graph is connected, so its GCC is the whole graph and the number of nodes in the GCC is 4039. Thus, a Barabasi network models the Facebook graph, where people who are connected to a lot of other people, have a higher chance of meeting new people, thus having a preferential attachment quality to it.

Question 2

The diameter of the graph was found using the diameter() function and it was found to be 8.

Question 3

The degree of a node in an undirected graph is the number of connections (edges) that it has with other nodes in the graph. The probability that a node has a given degree k is given by:

$$P(K = k) = \frac{\text{number of vertices with degree k}}{\text{total number of nodes}}$$

where K is the random variable that takes on degree values. We studied in class that the preferential attachment model is a power-law network, and that its degree distribution follows the curve given by:

$$p_k \propto \frac{1}{k^\gamma}$$

where $\gamma > 0$ and k is a positive integer.The degree distribution is given below:

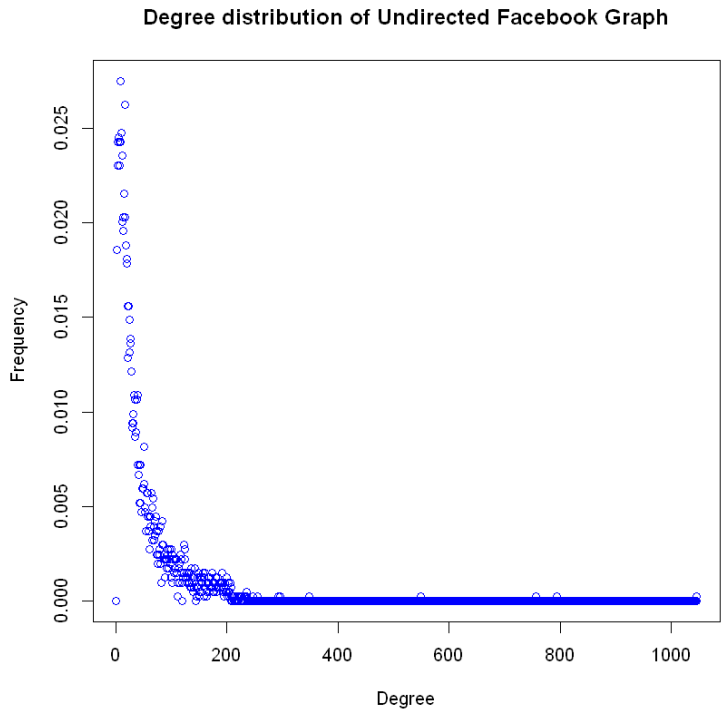


Figure 2: Degree distribution of the graph

The average degree of the graph is then defined simply as the ratio of the number of edges to the number of nodes in the graph:

$$\text{Average degree} = \frac{\text{Number of edges}}{\text{Number of nodes}}$$

We found the average degree of the Facebook graph to be 43.69101.

Question 4

We plotted the degree distribution on the log-log scale:

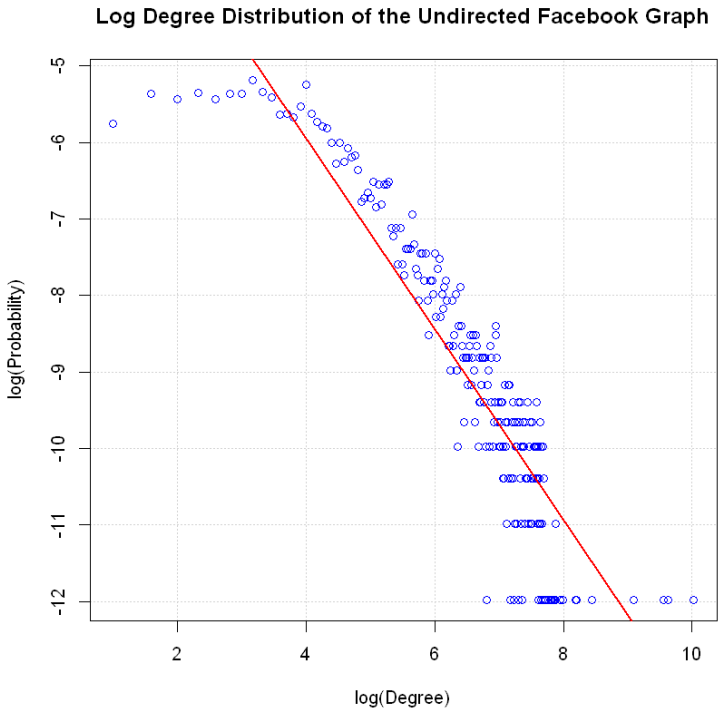


Figure 3: Degree distribution of the graph on the log scale

The slope of the fitted linear regression line was found to be -1.2475, this corresponds to γ. We see clearly that as the slope is lesser than 3, the Facebook graph follows weak preferential attachment.

Question 5

A personalized network of an user is defined as the sub-graph induced by and it's neighbors. We created a personalized network using the function make_ego_graph() and picking the 1st one to get the one with ID 1. **The generated personalized network has 348 nodes and 2866 edges**. It is displayed below:

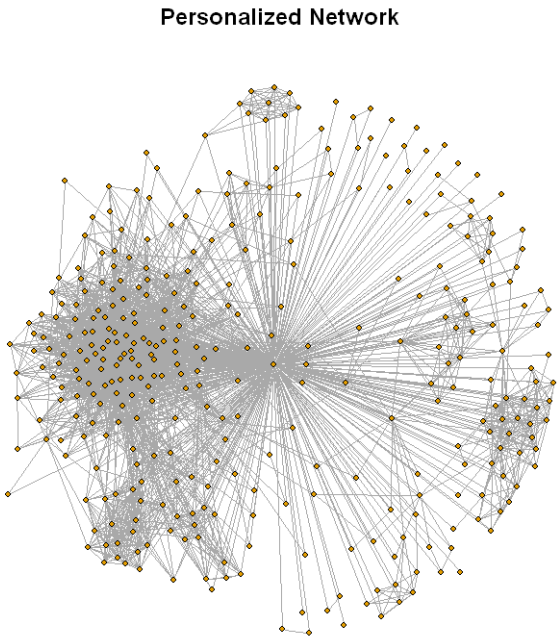


Figure 4: Personalized Network with ID 1

Question 6

The diameter of the personalized network with ID 1 was found to be 2. Clearly the lower bound is 1 and the upper bound is 2.

Question 7

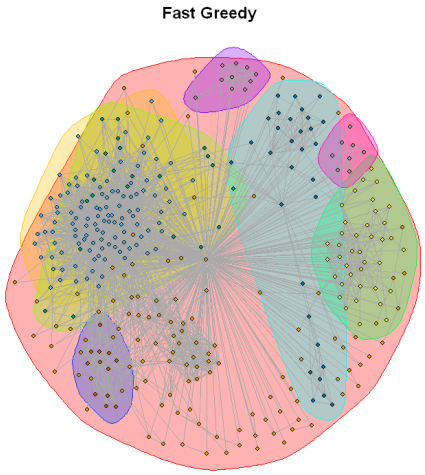
If the diameter of the graph is equal to the lower bound, then it means that every edge that can be created in the graph is present. This means that every person in the social circle knows everyone else.

If the diameter is 2, then there exists at least one pair of users that are not friends with each other.

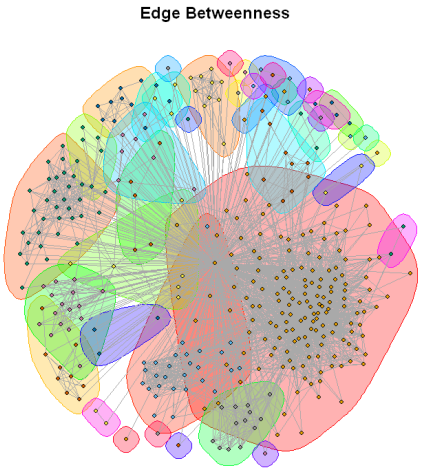
Question 8

A core node is defined as the nodes that have more than 200 neighbors. For our personalized network, **we found 40 core nodes, with the average degree of the core nodes being equal to 279.375**.

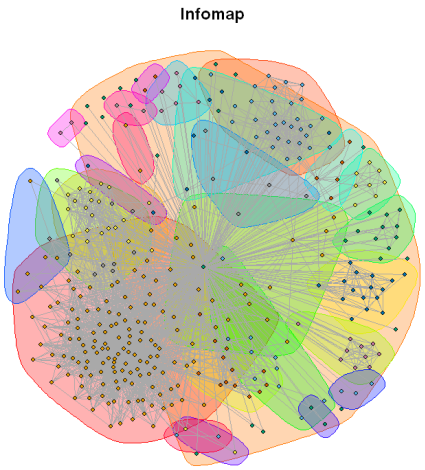
Question 9



(a) Core Node 1: Fast Greedy



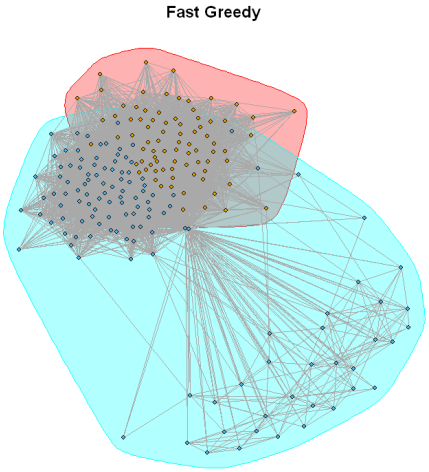
(b) Core Node 1: Edge Betweenness



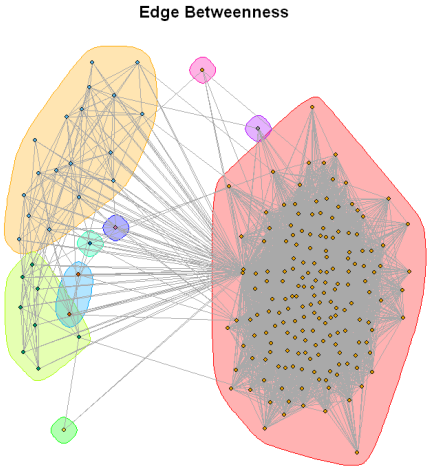
(c) Core Node 1: Infomap

Figure 5: Core Node 1 Community Structures

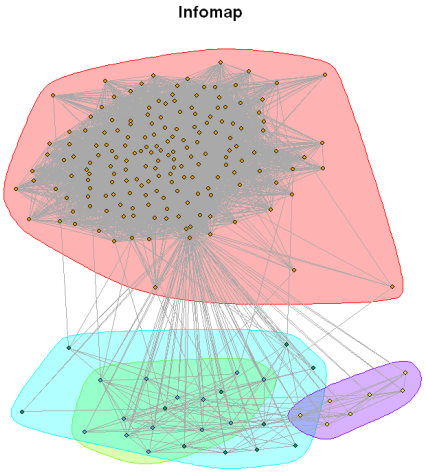
Graph nodes often form groups that are almost independent from the rest of the graph, with which they share a few edges, but the edges between nodes in that small group is denser. Such groups of nodes are called communities and the community structure describes the communities existing in a given graph. Modularity is a measure of the structure of networks or graphs which measures the strength of division of a network into communities. We looked at the community structures of 5 core nodes: 1, 108, 349, 484, 1087 using three different algorithms: fast_greedy, edge_betweenness, and infomap.



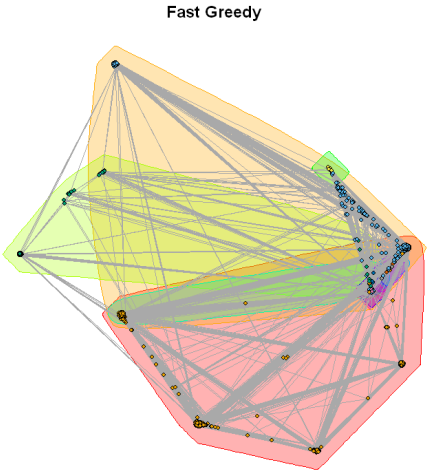
(a) Core Node 1087: Fast Greedy



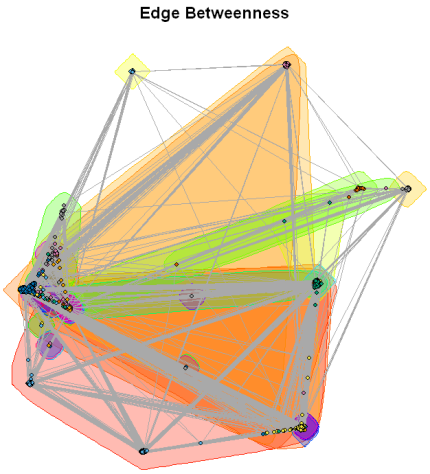
(b) Core Node 1087: Edge Betweenness



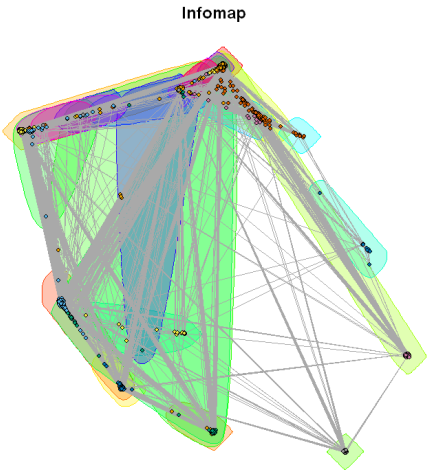
(c) Core Node 1087: Infomap



(d) Core Node 108: Fast Greedy

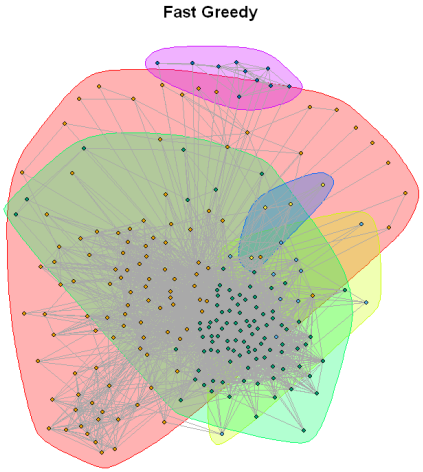


(e) Core Node 108: Edge Betweenness

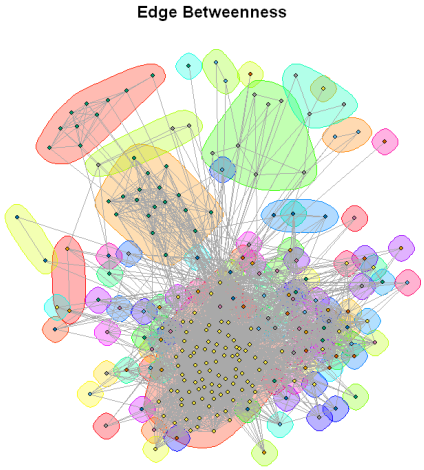


(f) Core Node 108: Infomap

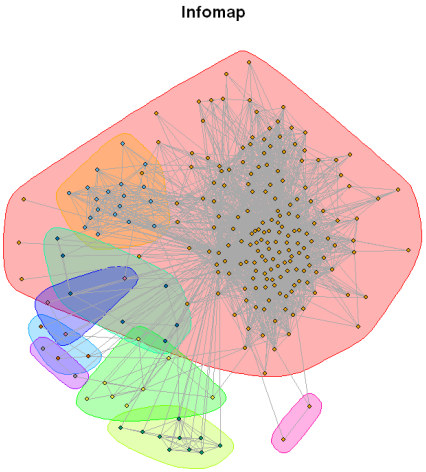
Figure 6: Community Structures of Core Nodes 1087 and 108



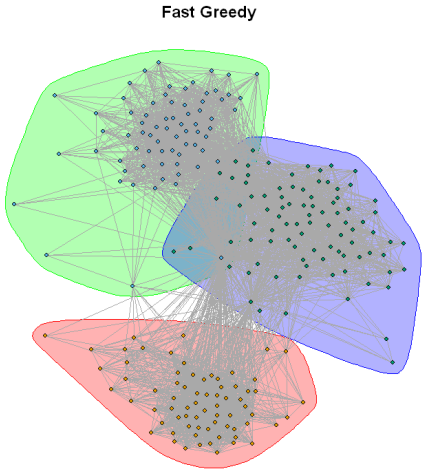
(a) Core Node 349: Fast Greedy



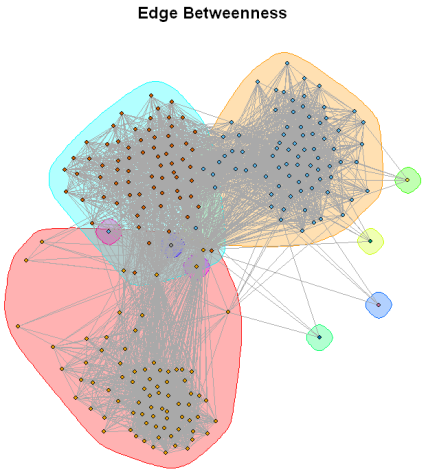
(b) Core Node 349: Edge Betweenness



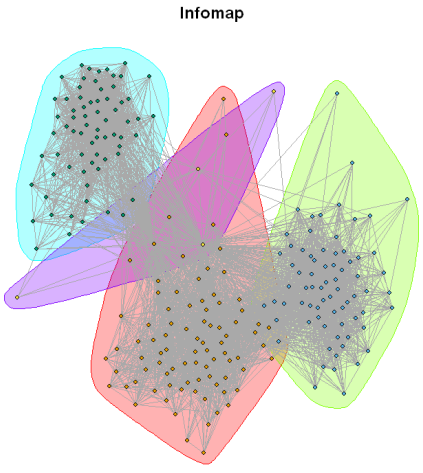
(c) Core Node 349: Infomap



(d) Core Node 484: Fast Greedy



(e) Core Node 484: Edge Betweenness



(f) Core Node 484: Infomap

Figure 7: Community Structures of Core Nodes 349 and 484

The fast greedy algorithm is an efficient approach to obtain community structure based on modularity. The graph starts with a sub-network composed only of links between highly connected nodes. The algorithm samples random edges that improve the modularity of the sub-network and adds them to the graph. This iterative process is repeated as long as the modularity keeps improving. Finally, the community structure is obtained based on the connections in the sub-network. The idea behind edge betweenness algorithm is that edges connecting communities would have large connectivity and the shortest paths between nodes in different communities are connected through these edges. The basic idea behind the InfoMap algorithm is to use community partitions of the graph as a Huffman code that compresses the information about a random walker exploring your graph.

We obtained the following modularities for our experiments:

ID	**Fast Greedy**	**Edge Betweenness**	**InfoMap**
1	0.413101	0.353302	0.389118
108	0.435929	0.506755	0.508454
349	0.251715	0.133528	0.095464
484	0.507002	0.489095	0.515279
1087	0.145531	0.027624	0.026907

Table 1: Modularities

Based on the modularities, we make the following observations:

1. The fast greedy algorithm performs the best.
2. Edge betweennes algorithm is the worse performing, slightly worse than InfoMap.
3. Core node 484 has the highest modularity across algorithms while node 1087 has the worst modularity. This means that core node 1087 does not have well defined communities like 484.

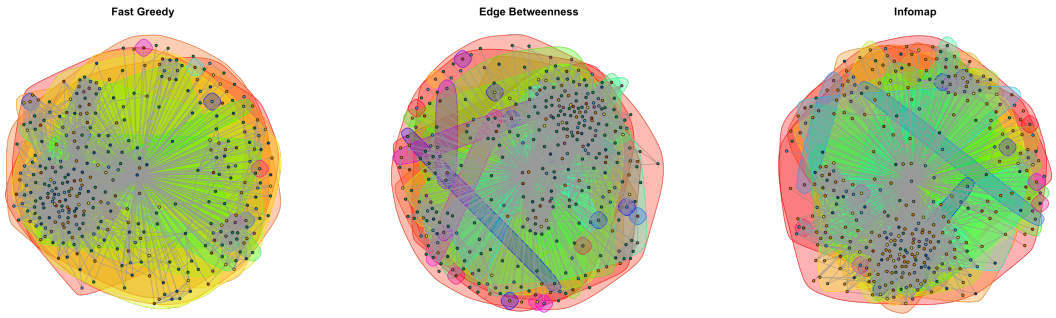
Question 10

In this part, we discovered the community structure core nodes using fast greedy, edge betweenness, and infomap with core nodes removed. That said, we preserved the neighbors but not the core codes themselves. The results are plotted in Fig. 8 & 9. We also obtained the modularity score for the modified personalized network and it is presented in Table 2. We can put Table 1 and 2 together to make comparison between corresponding values.

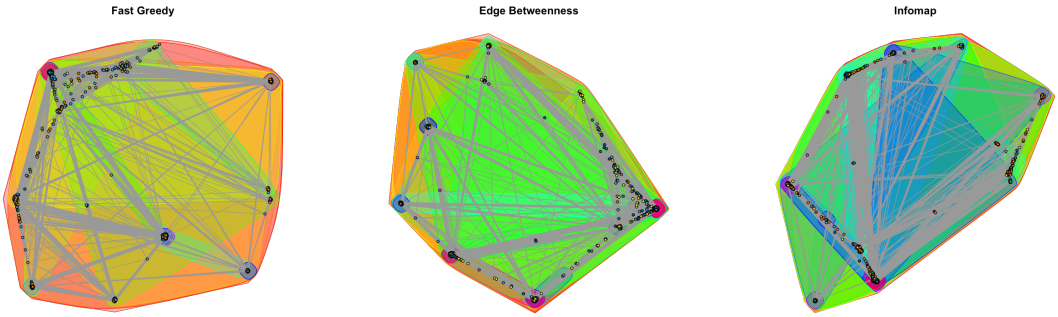
ID	**Fast Greedy**	**Edge Betweenness**	**InfoMap**
1	0.44185	0.41614	0.41800
108	0.45812	0.52132	0.52096
349	0.24569	0.15056	0.24481
484	0.53421	0.51544	0.54344
1087	0.14819	0.03249	0.02737

Table 2: Modularities for Modified Personalized Network

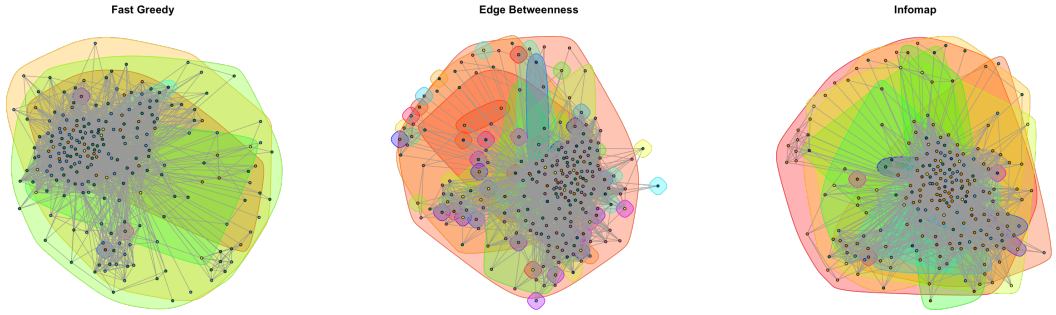
We can see that after removing the core nodes, the modularity scores increase acreoss five personalized network. This makes sense in that core nodes serve as connections between all other nodes and such a property lower the capability for the network being partitioned into communities. The presence of the core nodes prevents other nodes to directly connect to each other, resulting in many individual communities. With the core node removed, the edges from the core node to all other nodes are removed, allowing the community to be partitioned into densely packed areas of high connectedness with sparse inter-community edges.



(a) Community Structure of Node 1 with Fast Greedy (b) Community Structure of Node 1 with Edge Betweenness (c) Community Structure of Node 1 with Infomap

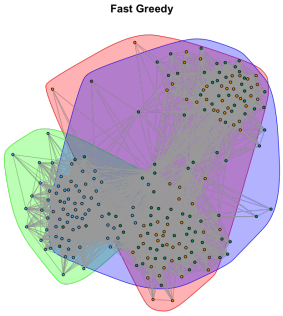


(d) Community Structure of Node 108 with Fast Greedy (e) Community Structure of Node 108 with Edge Betweenness (f) Community Structure of Node 108 with Infomap

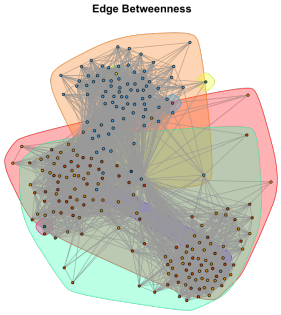


(g) Community Structure of Node 349 with Fast Greedy (h) Community Structure of Node 349 with Edge Betweenness (i) Community Structure of Node 349 with Infomap

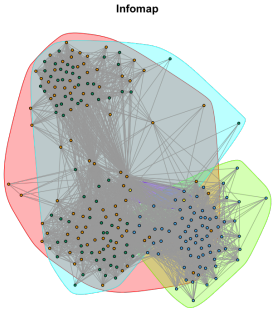
Figure 8: Community Structure of Core Node Modified Personalized Network



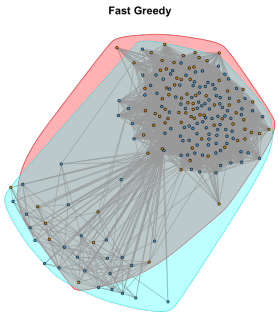
(a) Community Structure of Node 484 with Fast Greedy



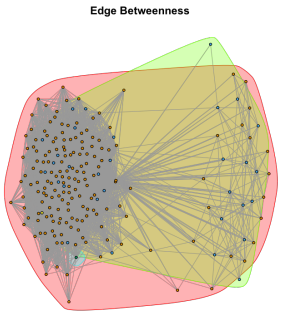
(b) Community Structure of Node 484 with Edge Betweenness



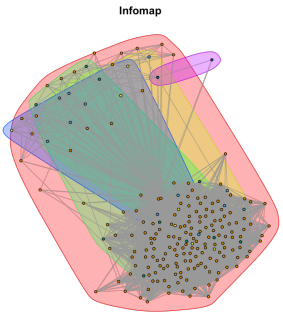
(c) Community Structure of Node 484 with Infomap



(d) Community Structure of Node 1087 with Fast Greedy



(e) Community Structure of Node 1087 with Edge Betweenness



(f) Community Structure of Node 1087 with Infomap

Figure 9: Cont'd Community Structure of Core Node Modified Personalized Network

Question 11

Embeddedness of a node is defined as the number of mutual friends a node shares with the core node. Therefore, the embeddedness between the core node and a non-core node should be the number of overlaps between the core node and non-core node in the circle.

$$emb(v_c, v_i) = deg(v_c) \cap deg(v_i)$$

where V_c is the core node and v_i is the non-core node.

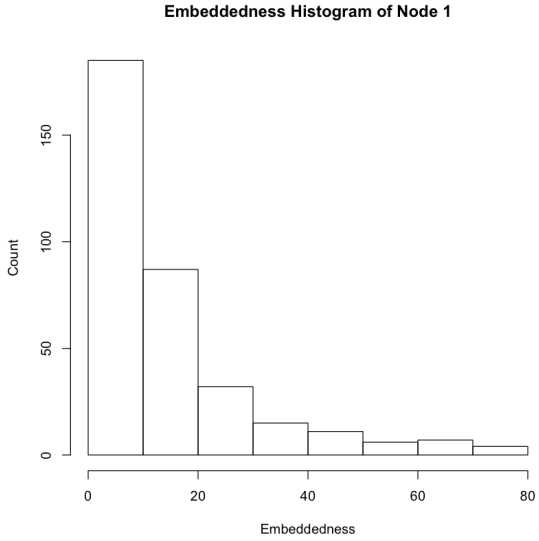
Question 12

In this part, we plotted the distribution histogram of embeddedness and dispersion for each of the core node's personalized network. The core nodes are node 1, 108, 349, 484, and 1087. The definition of embeddedness is given previously. Dispersion of a node is the sum of of all pairwise distances of the mutual friends the node shares with the core node. The expression is given as follows.

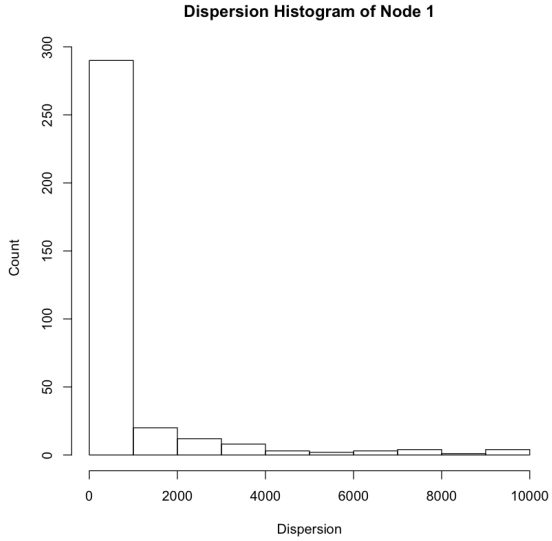
$$disp(u, v) = \sum_{s,t \in C_{uv}} d_v(s, t)$$

where d_v is a distance the nodes of C_{uv}, C_{uv} is the set of common neighbors of u and v measured by G_u, which is defined as the modified subgraph where the node and the core node are removed.

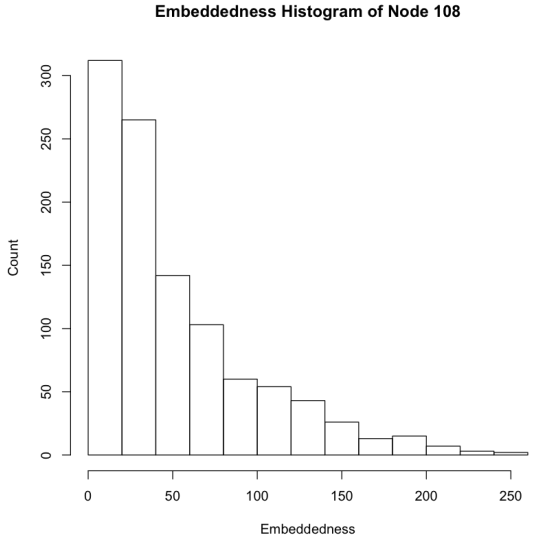
The Embeddedness and Dispersion graphs are shown in Fig. 10 & 11



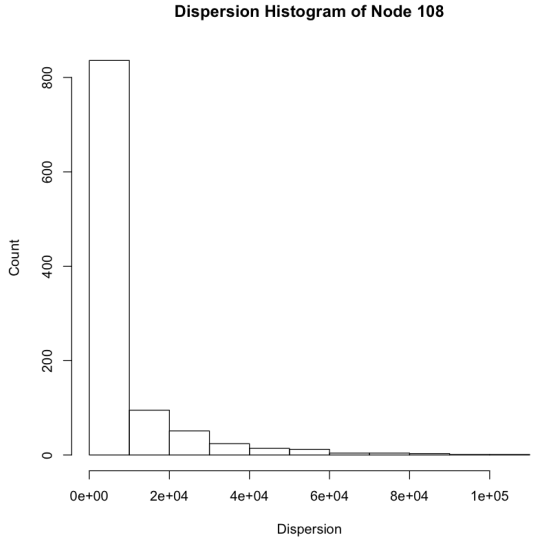
(a) Embeddedness of Node 1



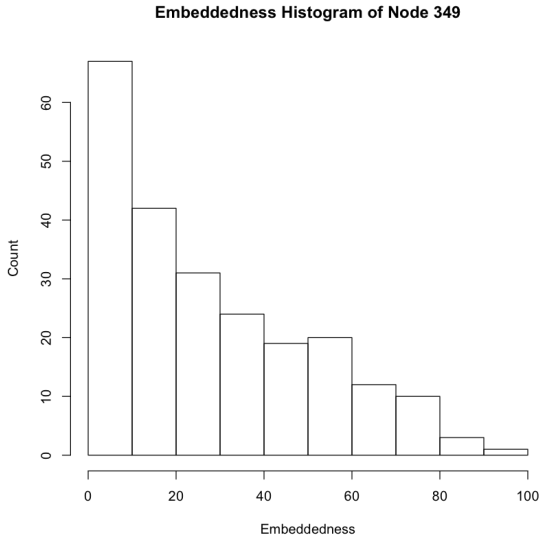
(b) Dispersion of Node 1



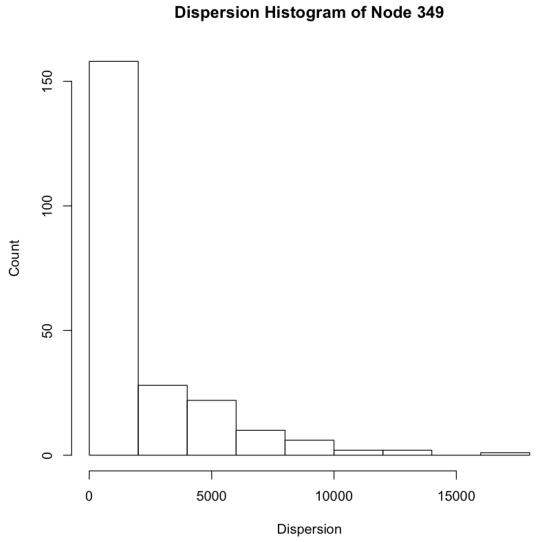
(c) Embeddedness of Node 108



(d) Dispersion of Node 108

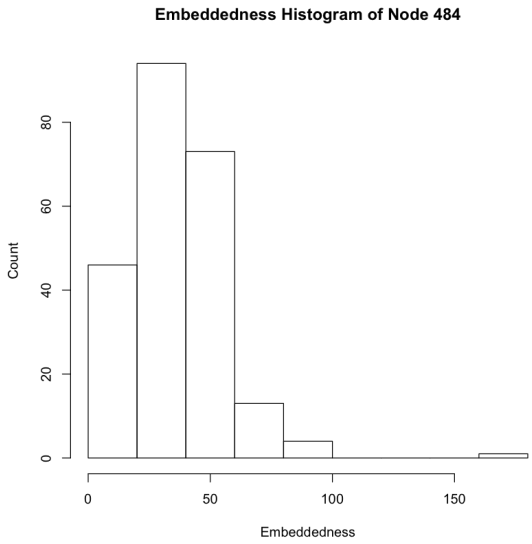


(e) Embeddedness of Node 349

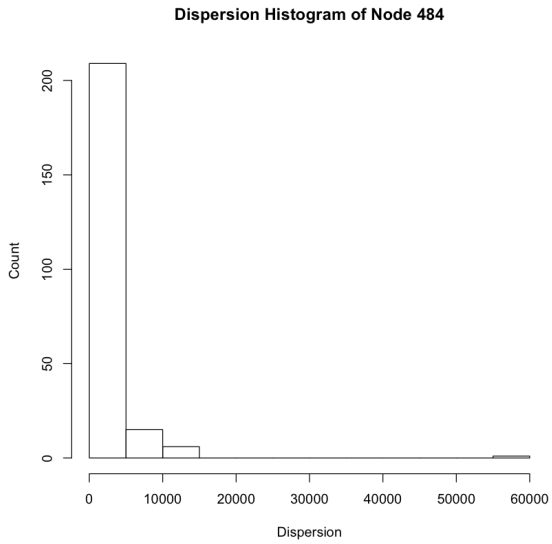


(f) Dispersion of Node 349

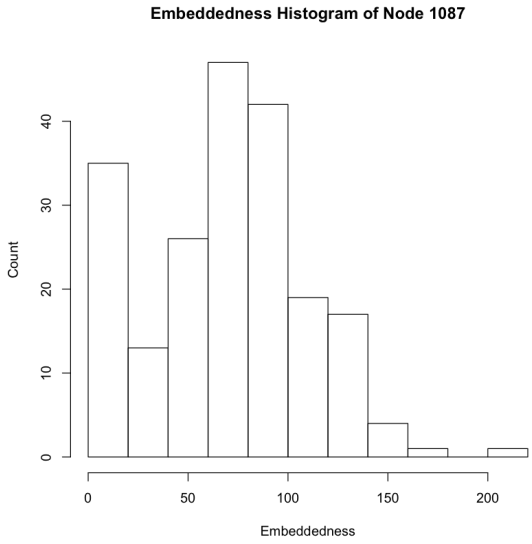
Figure 10: Embeddedness and Dispersion of the Core Nodes



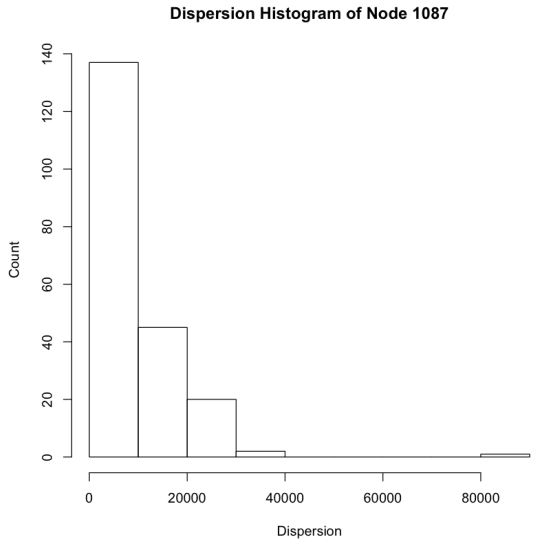
(a) Embeddedness of Node 484



(b) Dispersion of Node 484



(c) Embeddedness of Node 1087



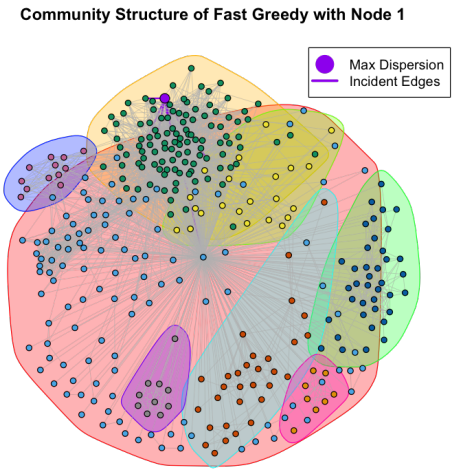
(d) Dispersion of Node 1087

Figure 11: Cont'd Embeddedness and Dispersion of the Core Nodes

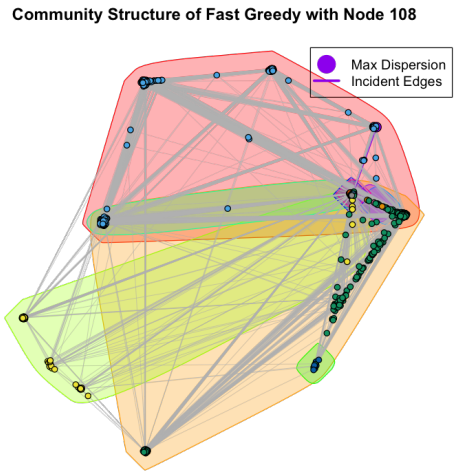
As is observed in Fig. 10 & 11, The embeddedness depends on the number of nodes in the personalized network. With higher number of nodes and connectivity, the embeddedness is greater. It is observed from Fig. 10f & 11d that the dispersion distribution displays lower skewness compared with others, this is explained by their relatively low modularity scores

Question 13

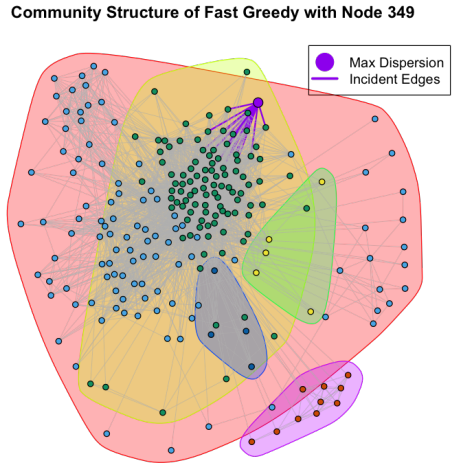
In this part, we found the community structure of the personalized network for the listed core nodes using Fast Greedy. We highlighted the node with max dispersion and also the incident edges. The results are presented in Fig. 12.



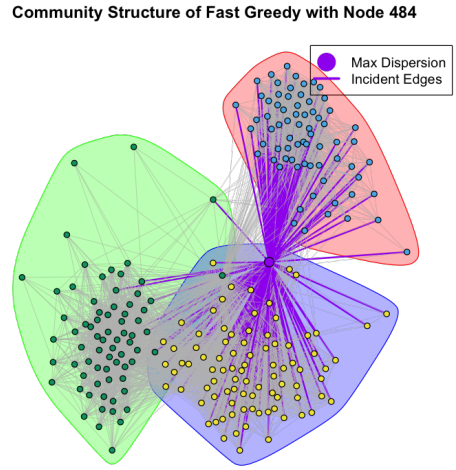
(a) Community structure with Max Dispersion of Node 1



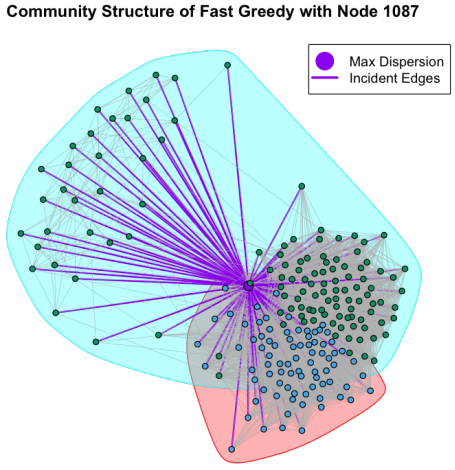
(b) Community structure with Max Dispersion of Node 108



(c) Community structure with Max Dispersion of Node 349



(d) Community structure with Max Dispersion of Node 484



(e) Community structure with Max Dispersion of Node 1087

Figure 12: community Structure of the Personalized Network with Max Dispersion

Question 14

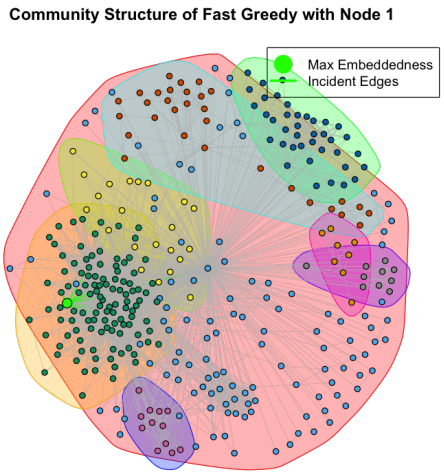
In this part, we found the community structure of the personalized network for the listed core nodes using Fast Greedy. We highlighted the node with max embeddedness and also the incident edges. The results are presented in Fig. 13. We also highlighted the node with max dispersion/embeddedness (called ratio in the graph) and the corresponding incident edges in Fig. 14.

Question 15

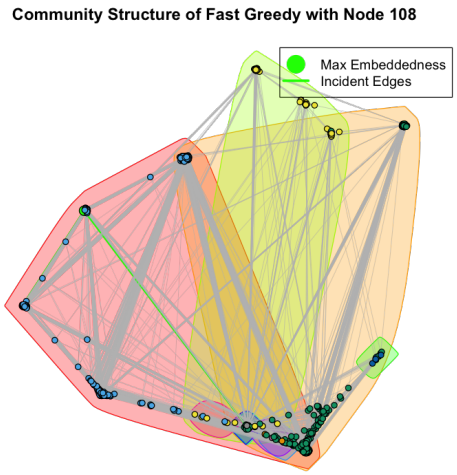
Max Dispersion: As is defined earlier dispersion of a node is the sum of of all pairwise distances of the mutual friends the node shares with the core node. The max dispersion is the node that has the greatest sum of the said distance. We observed that the node and the core node are not in the same community all the time. it is explained by the fact that the further apart the node and the core node, the greater the dispersion, resulting in the node with max dispersion.

Max Embeddedness: Embeddedness of a node is defined as the number of mutual friends a node shares with the core node. Therefore, max embeddedness node is the one that shares the maximum number of friends with the core node. It is observed from the plot that most of the max embeddedness nodes locate in the center of the communities.This makes sense in that when placed in the center of the communities, the nodes can have the most amount of neighbors besides the core nodes.

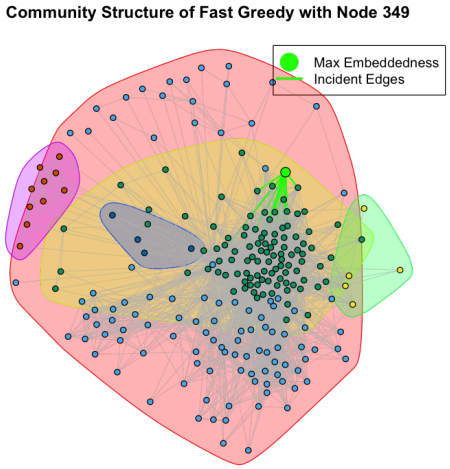
Max Dispersion/Embeddedness: The max ratio node represents the node with high dispersion and low embeddedness. These nodes are likely to be the one that have friends outside of the communities and do not share many mutual friends with the core node. The dispersion/embeddedness ratio is higher within smaller communities of dispersed networks with friends not strongly connected.



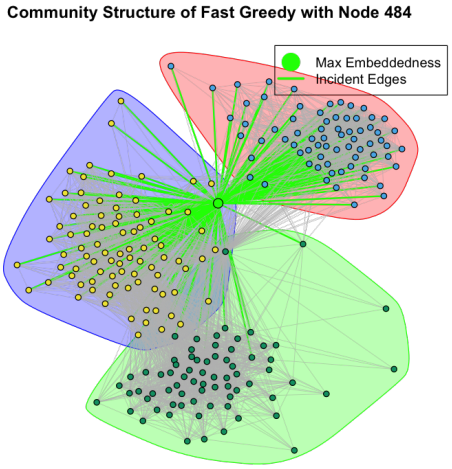
(a) Community structure with Max Embeddedness of Node 1



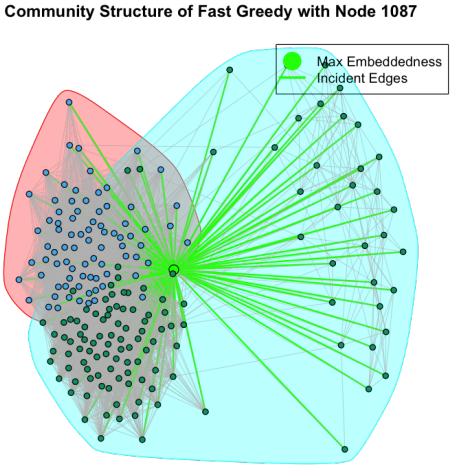
(b) Community structure with Max Embeddedness of Node 108



(c) Community structure with Max Embeddedness of Node 349

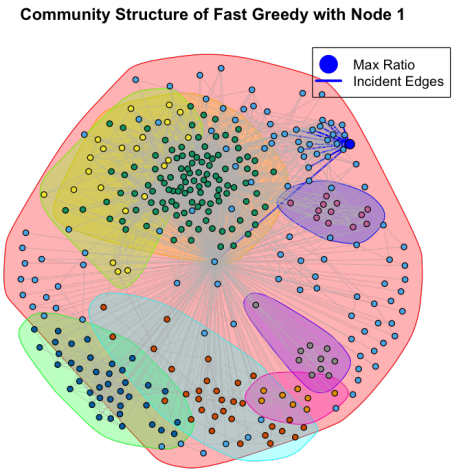


(d) Community structure with Max Embeddedness of Node 484

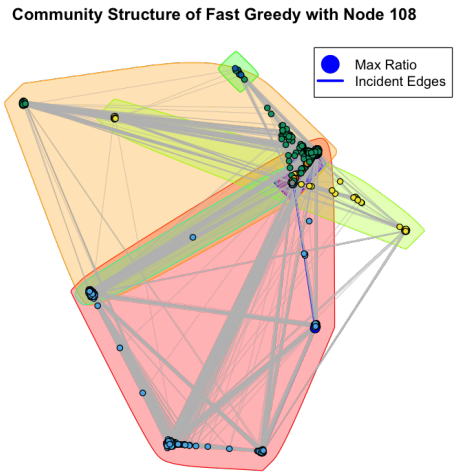


(e) Community structure with Max Embeddedness of Node 1087

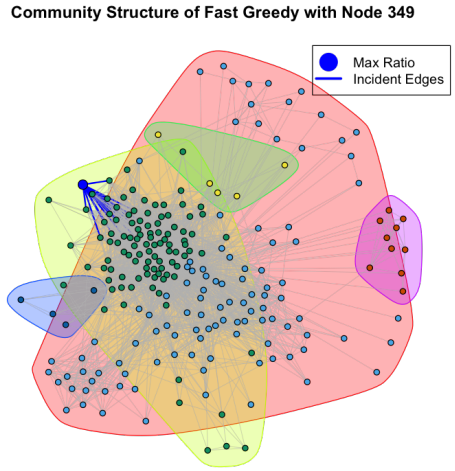
Figure 13: community Structure of the Personalized Network with Max Embeddedness



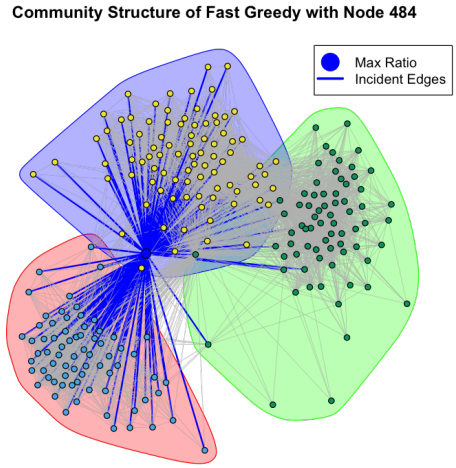
(a) Community structure with Max Dispersion/Embeddedness of Node 1



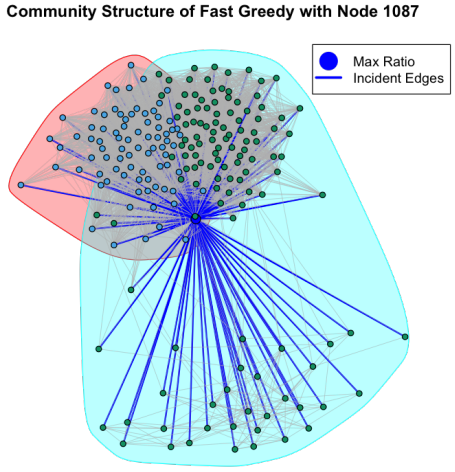
(b) Community structure with Max Dispersion/Embeddedness of Node 108



(c) Community structure with Max Dispersion/Embeddedness of Node 349



(d) Community structure with Max Dispersion/Embeddedness of Node 484



(e) Community structure with Max Dispersion/Embeddedness of Node 1087

Figure 14: community Structure of the Personalized Network with Max Dispersion/Embeddedness

Question 16

In this part, we created a personalized network with node 415, and created this list by picking all nodes with degree 24. **We found the value $|N_r| = 11$.**

Question 17

In this part, we compared the performance of different friend recommendation algorithms.

- S_i is the neighbor set of node i in the network.
- S_j is the neighbor set of node j in the network.

Common Neighbors: Common neighbor measure between node i and node j is defined as $|S_i \cap S_j|$.

Jaccard: Jaccard measure between node i and node j is defined as $\frac{|S_i \cap S_j|}{|S_i \cup S_j|}$.

Adamic Adar: Adamic-Adar measure between node i and node j is defined as $\sum_{k \in S_i \cap S_j} \frac{1}{\log(|S_k|)}$.

The accuracy results are given in Table 3. **It is observed from the table that all accuracy scores are above** 85% **which means all algorithms perform well. The Common Neighbors has the highest accuracy score.**

Algorithm	Common Neighbors	Jaccard	Adamic Adar
Accuracy	0.88295	0.85303	0.88106

Table 3: Average Accuracy of Different Friend Recommendation Algorithms

Google+ Network

In this part, we explore the structure of the Google+ network. Data can be found at: `http://snap.stanford.edu/data/egonets-Gplus.html`

Question 18

There are 57 personal networks for users who have more than 2 circles.

Question 19

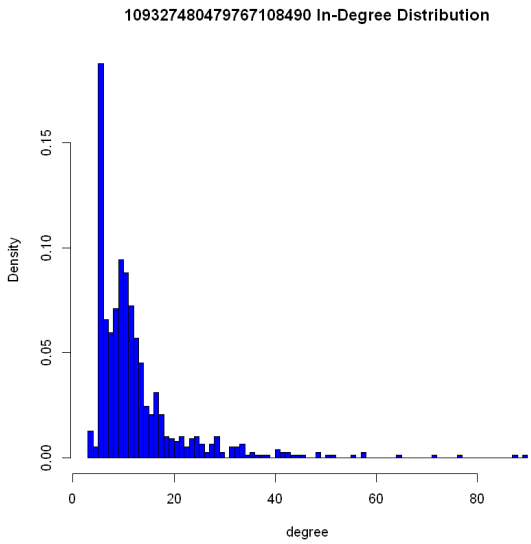
We look at the degree distribution for the following 3 networks (for brevity, we will write the last 4 digit of the node ID):

- 109327480479767108490 (node 8490)
- 115625564993990145546 (node 5546)
- 101373961279443806744 (node 6744)

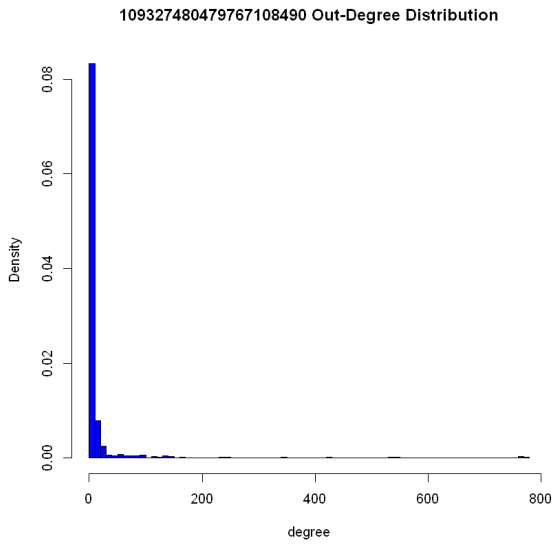
The in-degree and out-degree distribution of the given networks are shown below in Fig. 15.

Overall, in-degrees are higher than out-degrees in all three nodes. All nodes have out-degrees heavily skewed-right, which fall down exponentially and sharply, and out-degree distributions also appear to follow power law distribution. Out-degree distributions are similar among the three given nodes. However, in-degree distributions are significantly different among the 3 networks.

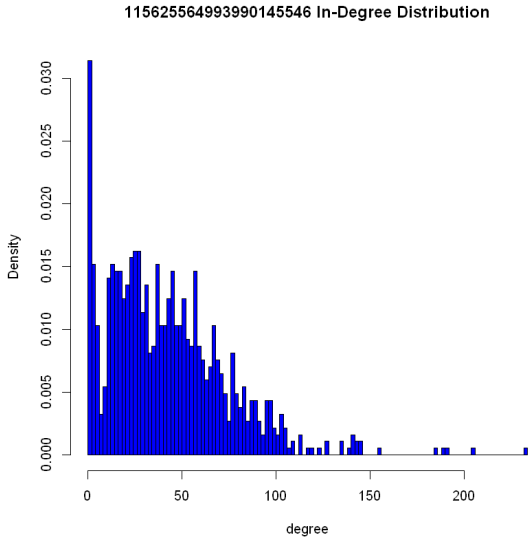
- Nodes 8490 and 6744 appear to follow power law distribution, as their degrees fall off exponentially as degree increases. Node 5546, on the other hand, does not follow power law distribution. Node 5546 has its degree decrease linearly.
- Node 6744 has the highest in-degree among the 3 with its maximum is over 1500. The node rolls off slowly, indicating low community structure and modularity.
- Node 5546 falls off almost linearly, indicating strong connectedness among members, hence a strong community structure.



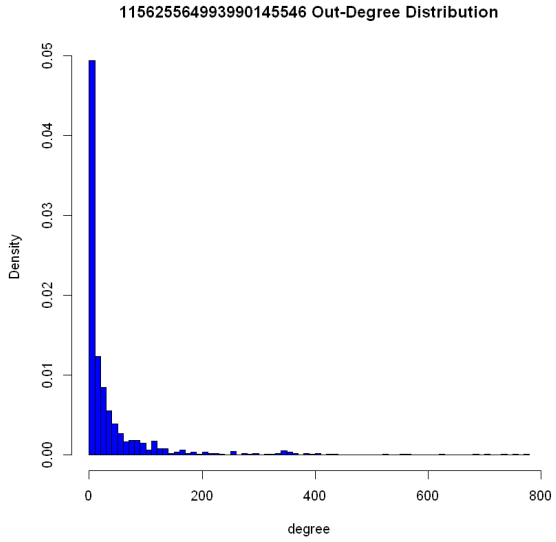
(a) Node 8490: In-degree distribution



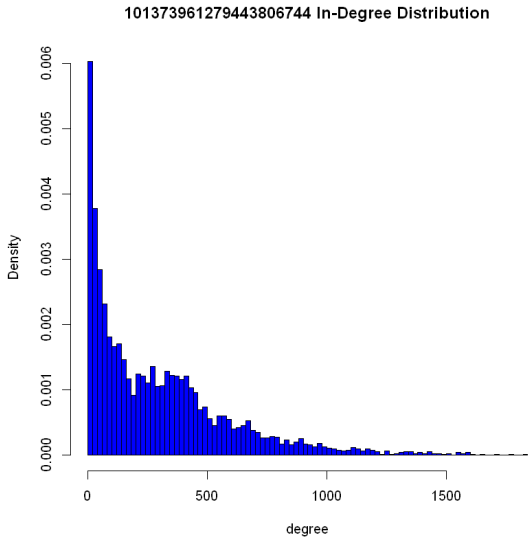
(b) Node 8490: Out-degree distribution



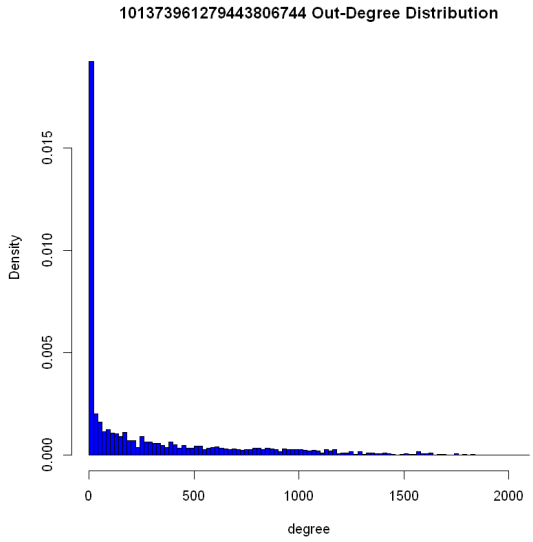
(c) Node 5546: In-degree distribution



(d) Node 5546: Out-degree distribution



(e) Node 6744: In-degree distribution



(f) Node 6744: Out-degree distribution

Figure 15: Degree distribution of Google+ networks

Community structure of personal networks

Question 20

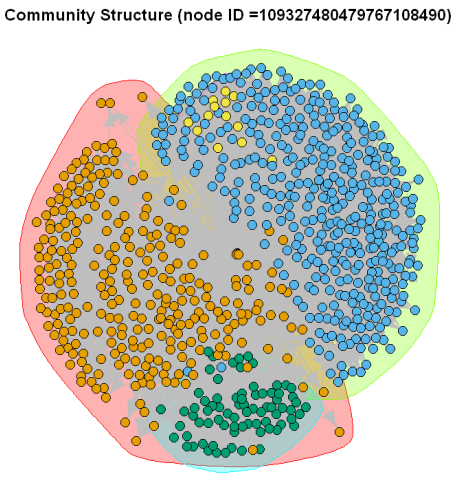
In this part, we explore the connections between communities and user circles. Table 4 shows the modularity scores for each network.

Node	**Modularity score**
8490	0.2528
5546	0.3195
6744	0.1911

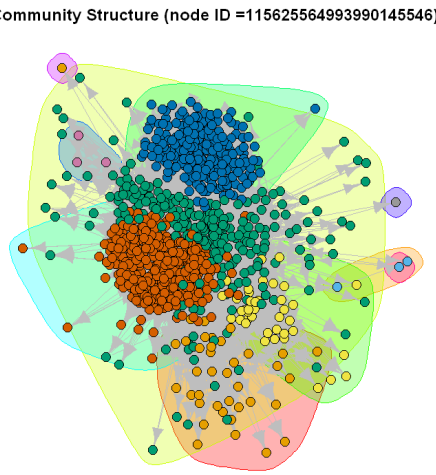
Table 4: Modularity scores

Modularity measures the ability to divide the network into modules, or communities in terms of Google+ networks. Networks with high modularity are interconnected, which means communities stay strongly connected within but have sparse connection between nodes outside modules or communities. From Table 4, we see that these nodes do not share modularity score. Node 5546 has the highest modularity. It further explains the in-degree distribution shown in Figure 15c, which does not have exceptionally high in-degrees as node 6744 but has many connections at moderate degrees. Contrary to node 5546, node 6744 has exceptionally high in-degree but has the lowest modularity. It means their are only a few individuals, or "hubs", that are well connected, and removal of these hubs can seriously damage the structure of the network. Node 6744 indicates a weak sense of community and low interconnectedness.

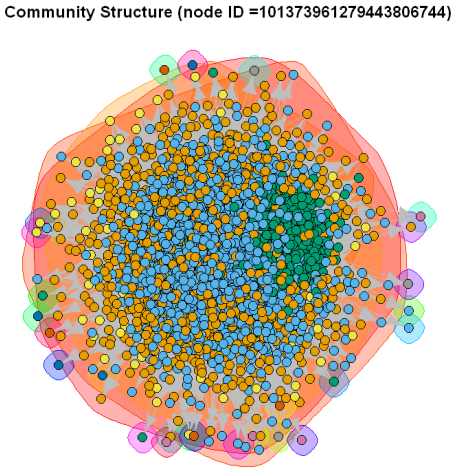
By using walktrap community detection algorithm, we are able to plot communities for the 3 given nodes in Figure 16. Based on Fig. 16, node 8490 is simple with distinct circles and communities. Node 5546 is more complex but we are still able to see 3 major distinct communities. Different circles seem to mix in this node. Node 6744 is the most complex network with communities and circles intermingle. No distinct structure can be easily seen in node 6744.



(a) Node 8490



(b) Node 5546



(c) Node 6744

Figure 16: Community structure

After exploring the communities, we look into the relationship between circles and communities. We define two measures:

- Homogenity
- Completeness

Let's introduce some notation:

- C is the set of circles, $C = C_1, C_2, C_3, ...$
- K is the set of communities, $K = K_1, K_2, K_3, ...$
- a_i is the number of people in circle C_i

- b_i is the number of people in community K_i with circle information
- N is the total number of people with circle information
- A_{ji} is the number of people belonging to community j and circle i

Then, with the above notation, we have the following expressions for the entropy

$$H(C) = -\sum_{i=1}^{|C|} \frac{a_i}{N} \log(\frac{a_i}{N}) \tag{1}$$

$$H(K) = -\sum_{i=1}^{|K|} \frac{b_i}{N} \log(\frac{b_i}{N}) \tag{2}$$

and conditional entropy

$$H(C|K) = -\sum_{j=1}^{|K|} \sum_{i=1}^{|C|} \frac{A_{ji}}{N} \log(\frac{A_{ji}}{b_j}) \tag{3}$$

$$H(K|C) = -\sum_{i=1}^{|C|} \sum_{j=1}^{|K|} \frac{A_{ji}}{N} \log(\frac{A_{ji}}{a_i}) \tag{4}$$

Now we can state the expression for homogenity, h as

$$h = 1 - \frac{H(C|K)}{H(C)} \tag{5}$$

and the expression for completeness, c as

$$c = 1 - \frac{H(K|C)}{H(K)} \tag{6}$$

Question 21

Based on the expressions for h and c, homogenity and completeness are measures to evaluate clustering techniques. Homogenity evaluates if a community has users belong to the same circle. A perfect score of 1 means each community has its members share exactly one circle. A low score in homogenity means each community has its members are equally distributed to all possible circles. On the other hand, completeness metric considers each circle and evaluates the variation in community assignment. A perfect score of 1 in completeness means each given circle has its members share exactly one community.

Question 22

Homogenity and completeness scores for the 3 nodes are shown in Table 5.

Node	**Homogenity**	**Completeness**
8490	0.8519	0.3299
5546	0.4519	-3.4240
6744	0.0039	-1.5042

Table 5: Homogenity and completeness scores

- Node 8490 has the highest homogenity and completeness scores among the 3 nodes. A score of 0.8519 in homogenity means almost each community in this network belongs to the same circle.

- Node 5546 has the medium score in terms of homogenity (0.4519). However, it has the lowest completeness score, which indicates that one circle often has multiple communities.
- Node 6744 has the lowest score in homogenity (0.0039). Looking at Figure 15e and 15f, we can see that degree distribution ranges are high (from 0 to 2000). It can be explained that with such high degrees, there are many different circles in each community, leading to low scores in not only homogenity but also completeness.

Cora Dataset

Introduction

In this section, we perform supervised learning on the Cora dataset. In supervised learning, we are provided with data called the input features along with the class that data point belongs to, its label. Our task is to assign classes to datapoints that have features, but do not have a label.

The Cora dataset[SNB+08] is consists of a set of 2708 documents that are machine learning related papers. It has 7 classes, this is a classification task where we need to assign labels to the unknown features and categorize them as belonging to one of these 7 classes:

- Case Based
- Genetic Algorithms
- Neural Networks
- Probabilistic Methods
- Reinforcement Learning
- Rule Learning
- Theory

Each class has a total of 20 labelled documents, called seed documents, and each document also has its own set of text features and vocabulary, which are occurrences of of 1433 words each in the dictionary. We follow three different experiments to utilize these features along with the graph structure of the dataset to perform supervised classification.

Graph Convolutional Networks

Graph Neural Networks (GNNs) are a class of deep learning methods designed to perform inference on data described by graphs. The most fundamental part of GNN is a graph, and GNNs are tuned to perform tasks on graphs, the same way convolutional neural networks (CNNs) are tuned to work well on images. **Graph Convolutional Networks (GCNs)**[KW16] are a subset of GNNs that make use of graph convolutions known from spectral graph theory to define parameterized filters that are used in a multi-layer neural network model similar to CNNs. We can perform a variety of operations using GCNs, and for this section, we perform node classification, where only a part of the graph is labelled, and we need to label the other nodes. This is called semi-supervised classification.

For this task, we utilized Google Colab and PyTorch with its easy-to-use library PyTorch Geometric, that was designed to work on graph neural networks. PyTorch Geometric has the Planetoid class that already contains Cora dataset, so we utilized that for this section. After loading the dataset, we recorded the details and found them to be as follows:

Number of nodes (ML documents): 2708
Number of edges: 5278
Number of classes: 7
Number of features per node: 1433

Then we built the graph neural network whose architecture is as follows:

```
class GraphConvolutionalNetwork(torch.nn.Module):
    def __init__(self, num_features, num_classes, hidden_dim, dropout):
        super().__init__()
        self.dropout = torch.nn.Dropout(dropout)
        self.conv1 = GCNConv(num_features, hidden_dim)
        self.relu = torch.nn.ReLU(inplace=True)
        self.conv2 = GCNConv(hidden_dim, 32)
        self.conv3 = GCNConv(32, num_classes)

    def forward(self, x, edge_index):
        x = self.dropout(x)
        x = self.conv1(x, edge_index)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.conv2(x, edge_index)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.conv3(x, edge_index)
        return x
```

As recommended in the original GCN paper[KW16], we used a dropout rate of 0.5 and two different hidden dimensions for the three-layer GCN: 16, and 32. The hyperparameters are recorded below:

Hyperparameter	**Value**
Number of GCN layers	3
Hidden dimensions	16, 32
Dropout rate	0.5
Activation function	ReLU
Loss	CrossEntropyLoss()
Optimizer	Adam
Learning Rate	0.01
Epochs	300
Weight Decay	5e-4
Early Stopping	Patience of 10 on Validation loss

Table 6: Hyperparameters

Here are the results that we recorded using 2, 3, and 4 layers of GCNConv():

Number of layers	**Accuracy(%)**
2	78
3	80.3
4	80.3

Table 7: Results

We noticed that while there is an improvement in accuracy when we increase the number of layers from 2 to 3, the accuracy stagnates after 3. **Therefore, we conclude that the ideal number of GCN layers is 3.** Here are the loss and accuracy curves when we used 3 GCN layers:

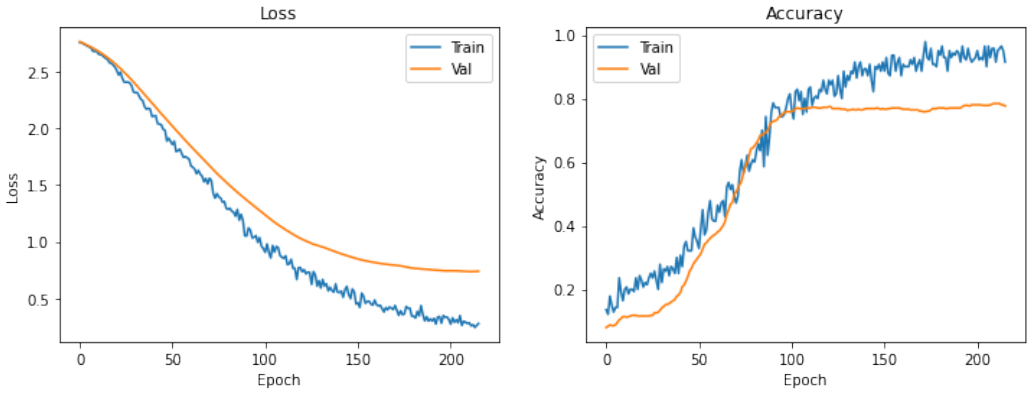


Figure 17: Loss and Accuracy curves for the architecture using 3 GCN layers

Node2Vec

In this part, we first combined SVM classification after extracting node features using Node2Vec [GL16]. We then compared the result with classifying Cora datasets using only text features. Finally, we combined both text features and node feature extraction for benchmarking purposes.

Node2Vec is an embedding method that translates graphs into low-level numerical representations. At a high level, we can think that Node2Vec is based on Word2Vec, where word meanings are extracted and mapped into vectors. Similarly, each node in a graph can be seen as a word. Contrary to Word2Vec, instead of given sequences such as sentences in a document, Node2Vec algorithm traverses nodes by biased random walks, which behave both as depth-first search (DFS) and breadth-first search (BFS).

In the first part of Node2Vec, we need to simulate the biased random walk for documentation (corpus) generation . We used the following hyperparameters settings:

- Maximum length of each random walk: 100
- Number of random walks per each node: 10
- p = 0.5 - the probability, 1/p, of returning to source node
- q = 2 - the probability, 1/q, for moving to a node away from the source node

We used the Word2Vec to learn representations for each node in the Cora dataset. We then applied the SVM classifier to classify research paper topics. We were able to achieve an accuracy score of 75.14%, which was slightly lower than the accuracy score of Graph Convolutional Networks.

For benchmarking purposes, we applied SVM classifier to text features of the Cora dataset. Unsurprisingly, the accuracy score was much lower, at 41.02%. Two possible explanations are unbalanced in training data and not enough training data. With 7 classes in the Cora dataset, SVM classifier may require much more training data if it solely bases on text features. Comparing to the previous result, utilizing Node2Vec, we can see that SVM is more effective in classifying the node representation using Word2Vec.

Lastly, we combined text features and node representation as learning features for the SVM classfier. We got the accuracy score of 67.47%, which is higher than classifying directly on text features but lower than classifying solely on node representation Node2Vec. It shows that using text

features does not synergies with node representation data and also negatively affects it. Possible ways to address this issue is using a different classifier such as XGBoost or scaling down text features as a way to reduce their importance in the training data.

Personalized Page Rank with TFIDF

In this part, we applied the personalized PageRank of each document in seven runs, one for each class. Similarly to other ideas, 20 seed documents were selected for each class run. PageRank was run only on the giant connected component. Each seed node was run with 1000 random walks.

We first run the PageRank with teleportation taking the random walker to one of the seed node of that class, with uniform probability of 1/20 per seed document. Teleportation probability (α) was varied in the following values: 0, 0.1, 0.2. The f1 scores and accuracy are shown in Table 8 and Table 9. For all cases, it should be noted that all nodes are visited in each scenario.

Class	$\alpha = 0$	$\alpha = 0.1$	$\alpha = 0.2$
Case Based	0.36	0.69	0.62
Genetic Algorithms	0.50	0.87	0.89
Neural Networks	0.33	0.72	0.71
Probabilistic Methods	0.54	0.80	0.79
Reinforcement Learning	0.11	0.71	0.70
Rule Learning	0.12	0.59	0.56
Theory	0.26	0.63	0.63

Table 8: F1 scores for random walks with uniform teleportation

	$\alpha = 0$	$\alpha = 0.1$	$\alpha = 0.2$
Accuracy	34.47%	73.20%	71.55%

Table 9: Accuracy for random walks with uniform teleportation

The results show that by introducing teleportation with $\alpha = 0.1$, the overall accuracy drastically improved from 34.47% to 73.20%. It can be explained that the ability to teleport gives each node a better chance to be visited. Without teleportation, random walks favor well connected nodes, which may skew or lower the prediction results. However, too much teleportation is not a good thing. As we can see with results for $\alpha = 0.2$, overall accuracy went down to 71.55%. One possible explanation is high teleportation probability makes the random walker resets back to seed nodes too often before being able to fully explore the visiting node branches.

In the next step, we combined PageRank with cosine similarity between the text features of the current document and the next neighboring document. Particularly, assume we are currently visiting a document x_0 which has neighbors $x_1, x_2, ..., x_k$. Then the probability of transition to each neighbor is:

$$p_j = \frac{\exp(x_0 * x_j)}{\sum_{l=1}^{k} \exp(x_0 * x_l)}; \text{for } j = 1, 2, ..., k \tag{7}$$

Similarly to the previous step, we varied the teleportation probability at 0, 0.1, and 0.2. Table 10 and Table 11 summarize the results in this step.

Comparing results from this step to the previous step, we can see that there is an overall increase in accuracy, but not by much. It suggests that utilizing similarity between text features in documents help improve the classifying task. However, we encountered an issue when running scenarios with 1000 random walks for each node - the number of nodes not visited for $\alpha = 0, \alpha = 0.1, \alpha = 0.2$ are 0, 9, 48, respectively. The number of unvisited nodes is proportional to inaccuracy. It is interesting that the unvisited nodes are not observed in scenarios without TFIDF. One possible reason is the cosine similarity, while slightly encourages the random walker to visit nodes with similar text features, significantly lowers the chance to be visited for nodes whose text features are not similar to others.

Class	$\alpha = 0$	$\alpha = 0.1$	$\alpha = 0.2$
Case Based	0.41	0.70	0.66
Genetic Algorithms	0.54	0.88	0.89
Neural Networks	0.36	0.75	0.68
Probabilistic Methods	0.60	0.80	0.74
Reinforcement Learning	0.14	0.73	0.68
Rule Learning	0.11	0.62	0.64
Theory	0.26	0.57	0.65

Table 10: F1 scores for random walks with transition probabilities proportional to cosine similarity between text features

	$\alpha = 0$	$\alpha = 0.1$	$\alpha = 0.2$
Accuracy	37.99%	73.72%	71.43%

Table 11: Accuracy for random walks with with transition probabilities proportional to cosine similarity between text features

We re-run the random walkers multiple times. However, number of unvisited nodes still exist. We hypothesize that the accuracy score can be improved by increasing the number of random walks.

Bibliography

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[KW16] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[SNB+08] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.