Contents

1. Project Overview

Abstract

This report analyzes the developer social network of GitHub. The raw data set focused on in this research was provided by B. Rozemberczki, C. Allen and R. Sarkar. In the first half of the report, the collaboration network of Open Hub was used for *SQL server* processing. In the later part, the GitHub social network data was cleaned and processed with R and Python, and was later visualized through Gephi. Then, attempts are made to identify possible community structures among developers. Other than using Node2Vec[1] for classification problem, a brief comparison between detected communities and labeled groups is provided.

Introduction

I am interested in network based analysis. Meanwhile, I'm a common user of forums targeting at data science. It appeals to me to discover the hidden information and relationships of different users from the open-source platforms, such as GitHub and Open Hub. In this research, the Github social network is mainly focused.

GitHub is a website with over 56 million developers, aiming to shape the future of software programming. This research uses the data set of a large social network of GitHub developers, collected from the public API in June 2019. Developers in the network already had binary labels with regard to whether their job title is web or machine learning.

Aiming to figure out the regularity behind the network, research questions are raised as follow:

1. What is the topological structure of GitHub social network?
2. Who is the "celebrity" in the GitHub social network by different measures?
3. Are web and machine learning people usually found in different communities by community detection algorithms?
4. Are web people generally more popular than machine learning people? Or else?

2. Network Data Collection and Processing

The Open Hub Collaboration Network

In the first half of this report, the data set used for *SQL server* processing is the collaboration network of Open Hub developers' provided by Prof. Daning Hu. It contains two files: the "tutorial_accounts.txt" and "tutorial_contributors.txt". The account data contains personal information of each developer, while the contributor data contains information of each project, with collaborated author included. The two data sets have 1409 nodes (developers) with 6689 links (projects) indicating the project collaboration.

First, the raw data of "tutorial_accounts" is processed. There are 4 attributes in total, including account_id, country_code, kuodo_rank and name. With some of the attributes types adjusted to save space, the attributes and data types are created in the query.

```
create table accounts (account_id nvarchar(50), country_code
nvarchar(50), kudo_rank nvarchar(50), name nvarchar(100));
```

After creating two tables in *SQL server*, I loaded two files by the similar query as below.

```
bulk insert contributor from '\data\tutorial_contributor.txt'
with (fieldterminator='\t ',rowterminator='\n')
```

Then, the raw data of "tutorial_contributor" is processed. There are 6 attributes including projectid, account_id, man_months, primary_language_id and account_name.

The next step needed to be done is to acquire the relation data about different developers. Two developers' "account_id" are likely to be linked through the same "project_id".

```
create table contributor (projectid int, developercount
nvarchar(50), account_id int, man_months nvarchar(50),
primary_language_id nvarchar(50), account_name nvarchar(255));
```

Then, the most important step is to select node and tie data from the above two data sets. The node data is done by selecting all of "tutorial_accounts.txt". For tie data, it is a bit more complicated. The relationship between different account IDs are based on the same project IDs they share: there are developers who work together in the same project. This is done through selecting "fromnode" and "tonode" separately from "tutorial_contributor" by matching the same "projectid". Additionally, some of them might collaborate with each other more than once. Thus, the repetitive collaborations with different "projectid' are counted, and named as "strength". By executing the query, the data sets of two modes can be transferred into one mode.

```
select p.account_id as fromnode, q.account_id as tonode,
count(*) as strength
from contributor as p, contributor as q
where p.projectid = q.projectid And p.account_id < q.account_id
group by p.account_id, q.account_id
```

Thus, in summary, by using *SQL server*, I loaded the data successfully, selected useful attributes, cleaned the data, and eliminated the repetitive data, ending up with two new datasets: node data and tie data (with edges from one node to another). Since the Open Hub network is undirected, the order of “from_node ” and “to_node” does not really matter.

Account Attributes	Contributor Attributes	Edge List
id country_code kudo_rank	account_id project_id	from_node to_node strength

Table 1: Table of Selected Attributes after Data Processing

The GitHub Social Network

The GitHub social network is initially collected for the Multi-Scale Attributed Node Embedding (MUSAE) project. It is also accessible through Stanford Network Analysis Project (SNAP). It contains three files: “musae_git_edges.csv”, “musae_ git_features.json” and “musae_git_target.csv”. All the nodes are developers who have starred at least 10 repositories and edges are mutual follower relationships between them. In “musae_git_features.json”, there are transformed data based on the location, repositories starred, employer and e-mail address. For safety reasons, the data set is anonymous, with no headings of attributes. However, it is still adequate for algorithms, such as Node2Vec to complete a classification problem.

features.json	target.txt	edges.txt
anonymous attributes $(x_1, x_2, \cdots, x_{15})$	name ml_target	id_1 id_2

Table 2: Table of Attributes for GitHub Social Network

However, this research focuses more on interpretability of network structures, thus two data sets are mainly processed: “musae_git_target.csv” and “musae_git_edges.csv” . The “musae_git_target.csv” contains “ml_target”, a boolean value indicating whether the developer is a machine learning person or not; “name” is the developers’ GitHub ID. This network contains 37700 nodes (developers) and 289003 edges. Nodes are 27961 (74.17%) web developers and 9739 (15.83%) machine learning developers, and the edges are the mutual follower relationships between them.

3. Network Modeling: Two-mode to one-mode transformation

To visualize the network in *Netdraw*, a file with node and data data is needed. The .vna file works by combining node and tie data together.

By looking at the edges, both of the Open Hub network and GitHub developer network could be constructed as undirected networks. The collaboration relationship in Open Hub and mutual follower relationship in GitHub don't have clear direction.

The current network displayed by the "tutorial_contributor" has two nodes, project and developer. This relationship forms a bipartite network, with only links between two different attributes. In reality, researchers focus more on the relationship between developers rather than focusing on the precise projects. A one-mode model with homogeneous nodes is more simple for analysis. For this reason, the one-mode data need to be transformed.

A script was implemented in order to "fold" the two-node network, which created edges linking developers to developers. The transformation process is shown as in Figure 1. This diagram is a simplified version consisting two developers, where developer 1 has participated in project A and project B, and developer 2 has also contributed to project A. This transformation can be realized by the query in the previous section.

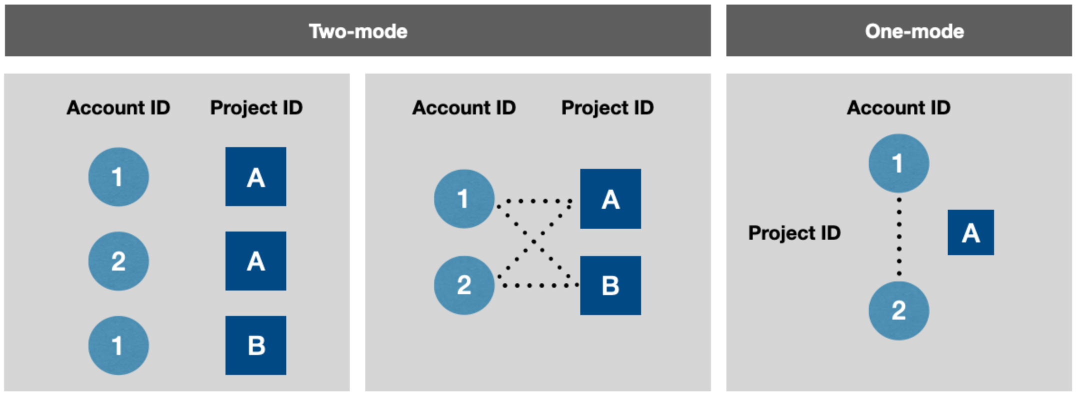

Figure 1: Two-mode and One-mode Network

4. Network Visualization

Other than using *Netdraw* for visualization, in this section, Gephi is used for its better processing speed for larger data sets. In this case, the network has got 37700 nodes and 289003 edges. It is a gigantic network, not friendly for *Netdraw*.

The first step after loading data into *Gephi* is to choose to an appropriate layout. The Yifan Hu layout is an algorithm that brings together the good parts of force-directed algorithms and a multilevel algorithm to reduce algorithm complexity. This is one of the algorithms that works really well with large networks. However, in Figure 2 it still provides very dense and indistinguishable links at the beginning.

Then, I tried to color the nodes with rankings of their degree centrality. Node size were set with respect to their degree. Some points are discovered, with two points stood out from others. Most of the points in the networks have very few degrees. This further illustrates that most of the developers in this network have minor mutual followers with others.

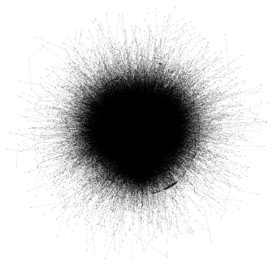

Figure 2: Network with Yifan Hu Layout

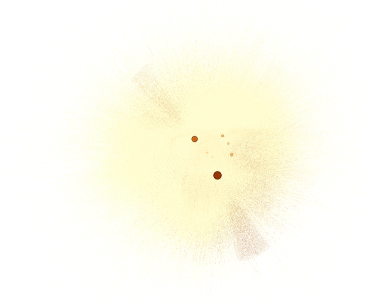

Figure 3: Colored by degree centrality. The larger and darker points stand for developers having large number of mutual followers.

The two outstanding points are *dalinhuang99* and *nfultz*, receiving 9458 and 7085 mutual follower connections. *DalinHuang99* has mutual followers with almost 25% of the developers in this network. He is likely an influential developer, or "celebrity" among developers. Both of them are labeled as web person instead of machine learning person. It raises to a question that is a web person generally more popular than a machine learning person?

I used different color to distinguish whether a developer is a machine learning person or not. Figure 4 provides a direct visualization. About 74.16% developers are web people, while the remaining 25.83% are from machine-learning. It is surprising to discover a possible pattern that developers of the same field are likely to connect to developers within the same disciplines. Most of the influential people in this network are web people. The GitHub data set is an uneven sample between web developers and machine-learning developers, and developers with top ten degree centrality are all labeled as "web".

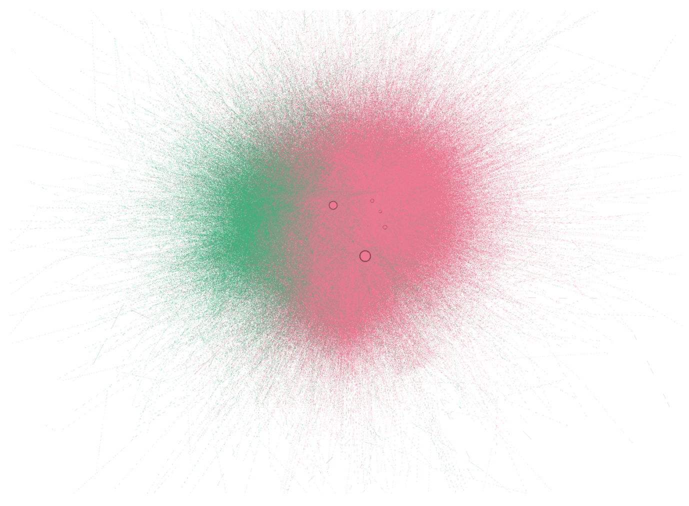

Figure 4: Colored by "ml_target". Pink is a web person, green is a machine learning person.

Other than plotting the network with only nodes and edges, labels could also be added, with size proportional to some specific node characteristics. Figure 5 is a plot by setting node size and names proportional to PageRank score. In Figure 6, it is a graph with pink as web person and green as machine learning person. Size of the nodes is set with respect to their closeness centrality. Two IDs can be easily discovered in this network: “nfultz” and “dalinhuang99”.

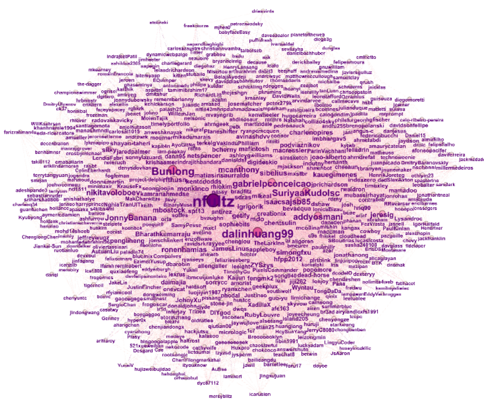

Figure 5: Colored by Betweenness

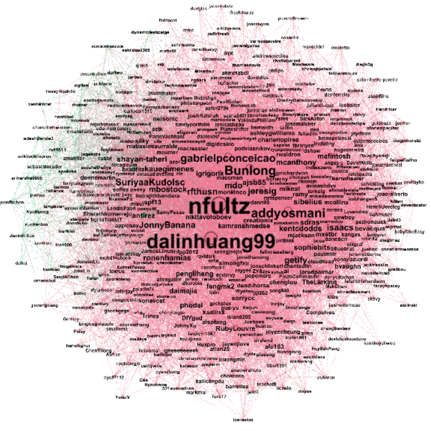

Figure 6: Simplified network colored by partitions of community detection, size is proportional to the betweenness centrality.

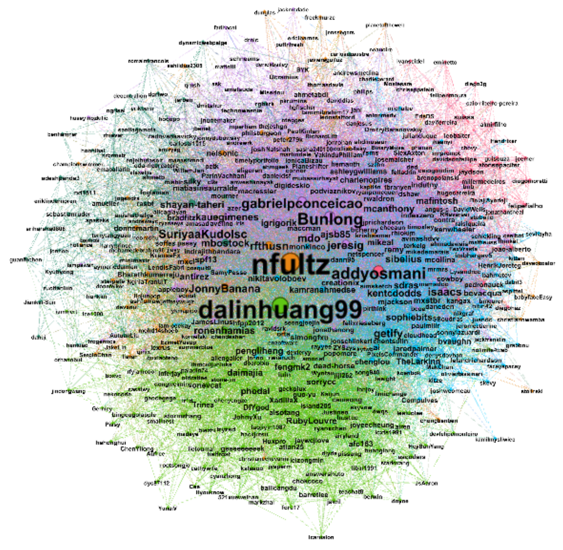

Figure 7: Community Detection Within the GitHub Social Network by Louvain Method

Additionally, some visualizations could also be conducted with respect to possible community structure inside the network. Figure 7 provides a community structure by detection through Louvain method of modularity score. Nine communities are detected, with most of the machine learning person placed in the same community.

Besides visualization of the whole network, it is also possible to look at network for web or machine learning developers separately. In Figure 8, the network of machine learning people with degree larger than 100 is shown, with node size scaled by PageRank. Two developers are found to be obviously more influential than others. Figure 9 shows the network of web person with degree larger than 500, people who are more influential in this field. It is very intuitive to find that "nfultz" is the most influential developer among them. The speculation is that "nfultz" are connected to more influential developers than other web developers do.

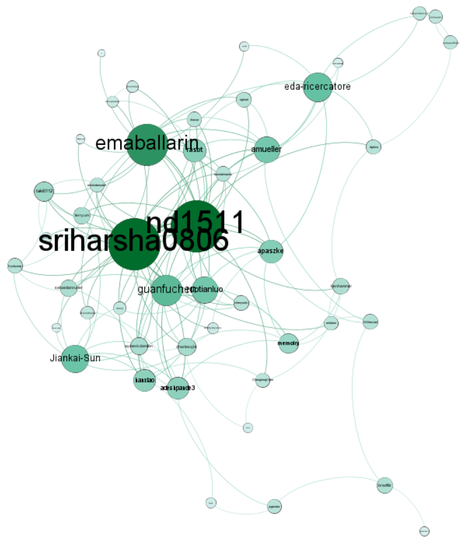

Figure 8: Machine Learning Simplified Network

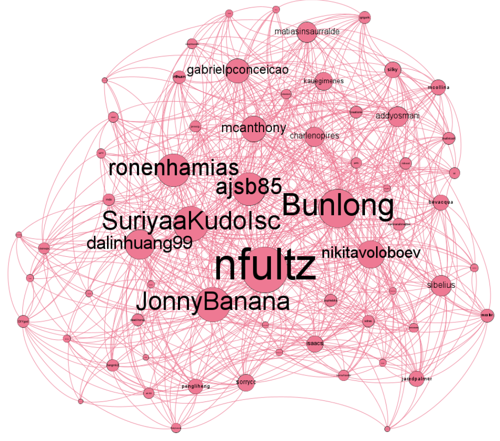

Figure 9: Web Simplified Network

5. Network Analysis

In this section, node level analysis, group and link level analysis, network level analysis are conducted through calculation and interpretation of some widely adapted statistics and visualization.

5.1 Node Level Analysis

There are four commonly used network centrality measures including degree centrality, closeness centrality, betweenness centrality and eigenvector centrality. They all have different standard of defining what is an "influential" node inside the network. Four measures are all calculated and compared for each case.

5.1.1 Network Degree Centrality Measure

The degree centrality of the graph is the conceptually the most simple measure. It is defined as the number of links connected to a node (in undirected graph). If we denote the graph $G = (V, E)$ with V vertices and E edges, then the degree centrality $C_D(v) = deg(v)$.

In the GitHub social network, the degree centrality varies greatly among developers. There are developers that only have 1 mutual follower with another developer in this network. For the developer with highest degree centrality: *Dalinhuang99*, he has $C_D(v) = 9458$. Developers with top 10 degree centrality are listed as in Table 3. They are all labeled as web person. The 10^{th} developer, *ronenhamias*, has 1568 links, only 16.58% of "Dalinghuang99". This indicates that there is a very sharp decline of degree in descending direction. The degree distribution might be considered as a fat-tail distribution. More will be provided in up-coming sections about the degree distribution.

name	degree	ml_target
dalinhuang99	9458	0
nfultz	7085	0
addyosmani	3324	0
Bunlong	2958	0
gabrielpconceicao	2468	0
rfthusn	2343	0
nelsonic	1924	0
getify	1797	0
mdo	1571	0
ronenhamias	1568	0

Table 3: Top 10 Developers of Degree Centrality

5.1.2 Network Betweenness Centrality Measure

The betweenness centrality measures the node's role in acting as a bridge between separate clusters by computing the ratio of all shortest paths in the network that contain a given node. It could be calculated as :

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is total number of shortest paths from node s to node t, and $\sigma_{st}(v)$ is the number of paths passing through v.

The larger betweenness centrality is, the more important a node is by acting as a "connector". The top 10 developers with highest betweenness centrality is shown. Since "dalinghuang99" and "nfultz" have obviously larger betweenness centrality in this network,

name	betweenness centrality	ml_target
dalinhuang99	0.269599268	0
nfultz	0.240541422	0
Bunlong	0.055322748	0
addyosmani	0.043408133	0
Prielpconceicao	0.035337494	0
rfthusn	0.030840307	0
ronenhamias	0.027611668	0
nelsonic	0.025799162	0
shayan-taheri	0.021260576	0
JonnyBanana	0.020315262	0

Table 4: Top 10 Developers of Betweenness Centrality

they are two connectors that bind the developers from different disciplinary fields. These points might be the locations of knowledge integration.

The role of "dalinhuang99" and "nfultz" as connectors in this network could be further discovered through the network diagram. For simplicity, only nodes having degree larger than 100 are filtered to be displayed in Figure 10.

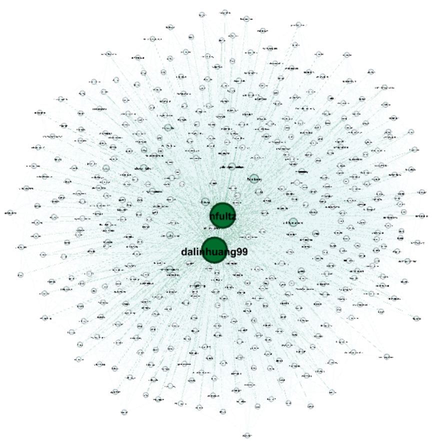

Figure 10: Colored by Betweenness

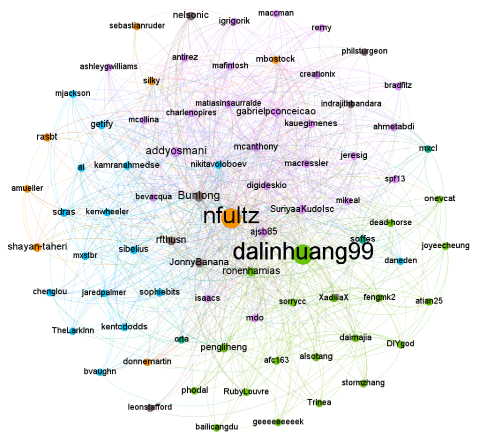

Figure 11: Simplified network colored by partitions of community detection, size is proportional to the betweenness centrality.

In Figure 11 selected only 80 nodes in this network, the partitions are colored by communities through calculating modularity class and using Louvain method set by Gephi. It further illustrates the role of *dalinhuang99* and *nfultz* as connectors to possible research subfields.

5.1.3 Network Normalized Closeness Centrality Measure

In a connected graph, the normalized closeness centrality of a node is the average length of the shortest path between the node and other nodes in the graph. Thus, the more central a node is in the whole network, the closer it will be to the other nodes. The definition is the reciprocal of farness:

$$C(x) = \frac{1}{\sum_y d(y,x)}$$

where $d(y,x)$ is the distance between x and y.[2] Its normalized form is multiplied by $N-1$. Thus, top 10 developers with highest closeness centrality are listed as in Table 5.

name	closeness centrality	ml_target
nfultz	0.5230814	0
dalinhuang99	0.5177865	0
Bunlong	0.4663236	0
addyosmani	0.4503416	0
JonnyBanana	0.4474606	0
SuriyaaKudoIsc	0.4473331	0
ronenhamias	0.4429861	0
rfthusn	0.4422533	0
ajsb85	0.4395975	0
mcanthony	0.4317339	0

Table 5: Top 10 Deleopers of Normalized Closeness

All the top 10 developers in Table 5 are web people. Table 5 further indicates that web people are closer to all of the other nodes in this network. For the closeness centrality measure, “nfultz” is closer to other nodes than “dalinhuang99”, though the later has larger degree centrality. It is possible that “nflutz” is closer to each developer in this network through some nodes that he/she connected to.

5.1.4 Network Eigenvector Centrality Measure

Eigenvector centrality measures a node’s importance while giving consideration to the importance of its neighbors. It uses the adjacency matrix to find eigenvector centrality. A high eigenvector score means that a node is connected to many nodes who themselves have high scores.

The relative centrality score of vertex v can be defined as:

$$x_v = \frac{1}{\lambda} \sum_{t \in M(v)} x_t = \frac{1}{\lambda} \sum_{t \in G} a_{v,t} x_t$$

where $M(v)$ is a set of neighbours, λ is a constant.

name	eigenvector centrality	label
dalinhuang99	1	0
nfultz	0.847549	0
addyosmani	0.454245	0
Bunlong	0.418212	0
gabrielpconceicao	0.335656	0
rfthusn	0.268875	0
jeresig	0.249382	0
SuriyaaKudoIsc	0.24635	0
getify	0.232628	0
isaacs	0.228632	0

Table 6: Top 10 Developers of Eigenvector Centrality

Table 6 provides the developers with top ten eigenvector centrality. “dalinghuang99” and “nfultz” still ranked at the top. The eigenvector centrality for “dalinhuang99” is 1, which means that it has very important neighbours. “nfultz” also has many important neighbours. These two developers, again, are very different from the other developers. It is likely that the network has the so-called “rich club” structure.

Network PageRank Score

Google’s PageRank is a variant of the eigenventor centrality that measures the influence of a node by its neighbours. It has been widely applied and adjusted to directed network, while still applicable to undirected network. The PageRank score $PR(E)$ transferred from one node to the targets for the next iteration is divided equally among all outbound links. The probability is expressed as a numeric value between 0 and 1.

In this report, the PageRank ($PR(E)$) score of each node is calculated with α value set as 0.85, and the visualization is done for the graph with degree larger than 100. It acts very similarly to previous centrality measures. The table of top 10 developers is provided in the Appendix.

In Figure 12, several influential developers are discovered. Then partitions are shown with respect to their labels. The first finding is when nodes with larger degree were selected, the proportion of being a machine-learning person tends to decrease. In Figure 13, only the web developers have distinguishable influential “celebrities” among them, while machine learning developers have relatively same and small $PR(E)$. Thus, “celeberties” are hard to be found for machine-learning developers from the diagram.

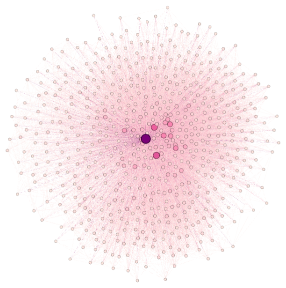
Figure 12: Colored by PageRank Score

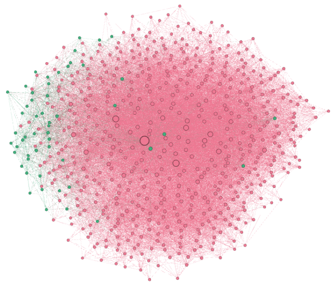
Figure 13: Colored by Labels, Size by PageRank Score

5.1.5 Comparisons and Interpretation

In the above sections, this report calculated four traditional network centrality measures: degree centrality, betweenness centrality, closeness centrality and eigenvector centrality. Additionally, PageRank scores of each developer are also provided.

Degree centrality focuses on number of nodes each developer connects to. Betweenness centrality focuses on the impact of being a "connector" in the network. Closeness centrality measures the ability of each node to reach the other nodes in short distances on average. Eigenvector based centrality measures take in consideration of neighbours' impact.

centrality measures	meaning	range	top 3 developers
degree centrality	quantify node connectivity	$[1, \infty]$	dalinhuang99, nfultz, addyosmani
betweenness centrality	quantify node importance in information flow	$[0, 1]$	dalinhuang99, nfultz, Bunlong
closeness centrality	easy access to all nodes	$[0, 1]$	nfultz, dalinhuang99, Bunlong
eigenvector centrality	take in consideration neighbours (global)	$[0, 1]$	dalinhuang99, nfultz, addyosmani
PageRank	can dump or scale hub effect	$[0, 1]$	nfultz, dalinhuang99, Bunlong

Table 7: Summary of Five Traditional Network Measures

Table 7 provides a summary of the four traditional network measures, as well as PageRank, with top three developers displayed under each measure.

No matter which centrality measure is used, "dalinhuang99" and "nfultz" are always the most influential developers. Thus, they are undoubtedly "hubs" in the network. High betweenness centrality further indicates that these two developers play an important role in bridging other developers together in the information flow. For closeness and PageRank score measures, "nfultz" is found to be more influential than "dalinhuang99". It is very likely that "dalinhuang99" has mutual followers with more developers, while "nfultz" has follower relations to more influential developers.

For the later part of this report, centrality measure used to scale the size of the node would be degree centrality without further specifications.

5.2 Group and Link Level Analysis

The GitHub social network used in this report has no weight for links, thus I only looked at the connectiveness, cutpoints and community detection for group and link level analysis.

Visualization is necessary in displaying different communities. For this reason, I choose to keep the nodes with degree larger than 500 for better display.

5.2.1 Cluster Analysis

Connectiveness

When identifying the components in the GitHub social network, it is found to be connected, which means it only has 1 giant component. Thus, the focus of this report is still on deciding the community structure within a large connected graph.

Community Detection

There are many popular community detection algorithms including fast greedy, Louvain, fluidc, Girvan-Newman, infomap etc. Community detection is not the focus of this course. For this consideration, only solutions by fast greedy algorithm and Louvain algorithm are provided in this report.

The fast greedy algorithm provides 3 detected communities in the subgraph, while the louvain algorithm provides 4 communities.

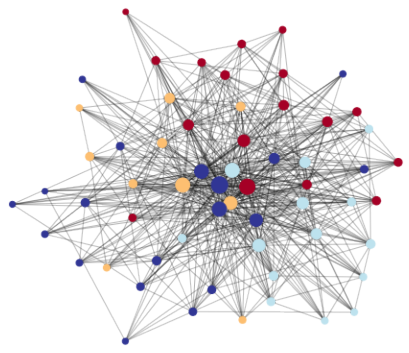
Figure 14: Louvain Algorithm

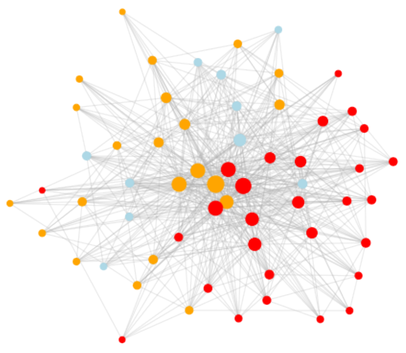
Figure 15: Fast Greedy Algorithm

By visualization, the fast greedy algorithm provides better community structure with label eliminated. A labeled version is provided in the Appendix. Some influential developers connect with each others and separate the subject into 3 parts. “nfultz”, “dalinhuang99”, “jonnyBanana” and “ronenhami” play an important role in one community (orange community). “Bunlong” and “SuriyaaKudolsc” play an important role in another (red community).

A divisive algorithm Girvan-Newman was also applied here for finding 2 communities. Focusing on the binary clustering problem, results with labels are provided in appendix.

5.2.2 Cutpoint

An articulation point or cut vertex is any node whose removal (along with all its incident edges) increases the number of connected components of a graph. An undirected connected graph without articulation points is biconnected. Figure 16 provides a diagram of articulation points.

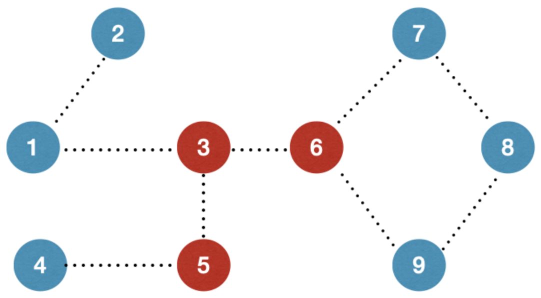

Figure 16: Articulation point illustartion. Articulation points are: 3, 5, and 6. By removing each node, the number of connected components increases.

However, even if developers have different web/machine learning title, it is not able to find any articulation point in this simplified network (degree larger than 500). The simplified network is a biconnected graph. It is possible that influential developers both have a large number of mutual followers. This illustrates a common phenomenon that followers are likely to follow a number of influential developers at the same time.

For the whole graph, by calculation there are 3188 articulation points. When a subgraph of degree larger than 5 is selected, there are 2 cutpoints in the network. The query in R is as follow:

```
> select_network1<-induced.subgraph(network,V(network)[degree(network)>5])
> articulation.points(select_network1)
+ 2/19052 vertices, named, from 34f76f6:
[1] kiyui n1try
```

The subgraph with degree larger than 5 has 19052 nodes which would be very hard to be visualized in the graph. Highlighted points are always overlapped by other points through kinds of layout. But finding these point is simple, they are "kiyui" and "n1try". For these two points, they are both web people.

Figure 17 and 18 are egonet and 2_{nd} degree network of "kiyui", it is very obvious that "kiyui" directly connects to all web people, but it's 2nd degree network displays that it seems to be a bridge between web and machine-learning people.

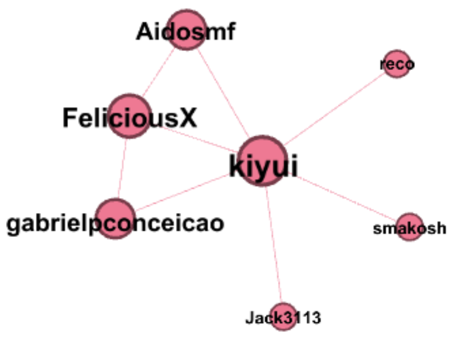

Figure 17: “kiyui” Egonet

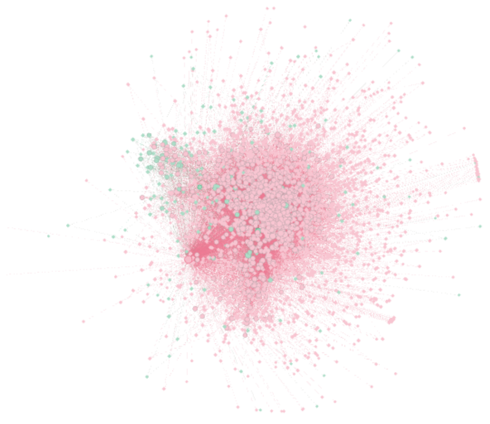

Figure 18: “kiyui” 2_{nd} Degree Network

5.3 Network Level Analysis

Since in later sessions, network visualizations is not usually needed, the whole GitHub social network is used for analysis. Statistics measuring the structure of network are then calculated. Comparisons between different network structures are made.

5.3.1 Network Density

Network density is defined to be ratio of the number of edges and number of possible edges. There are n nodes and L links in graph G. The process is illustrated as in the diagram of Figure 19.

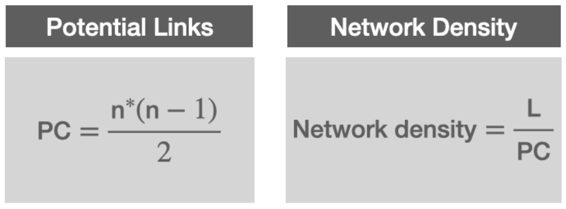
Potential Links
$PC = \frac{n^*(n-1)}{2}$
Network Density
$\text{Network density} = \frac{L}{PC}$

Figure 19: Illustration of Network Density

The network density is calculated as 0.004066878, which means it is very far from being a complete subgraph.

5.3.2 Average Degree

Average degree for the GitHub social network is 15.332. This indicates that a developer on average has mutual followers to other 15 developers. The maximum degree is 9458 achieved by “dalinhuang99”, and the minimum is 1. This is realized by the query below:

```
>mean(degree(network))
[1] 15.33172
```

5.3.3 Degree Distribution

In real networks, most networks follow the scale-free distribution. The bins of histogram are divided evenly with respect to the range of degree at first. This had poor illustration. Most of the nodes' degree is in the first bin. Then, I rescaled the axis to degree from 1 to 100 and 20 bins, an exponential decreasing trend is shown for the degree distribution.

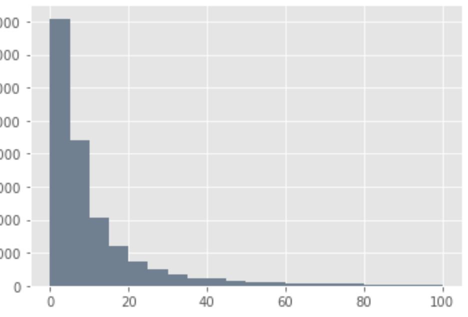

Figure 20: Degree Distribution

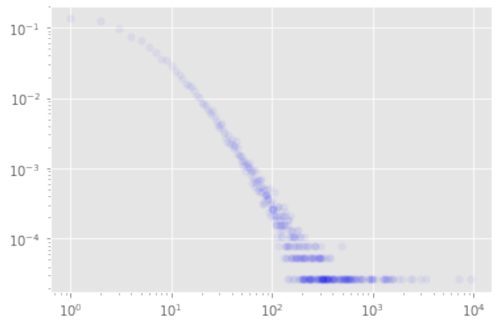

Figure 21: Degree Distribution (Logarithm Axis)

Further more, I rescaled both axis by logarithm, and calculated the regression coefficient for the distribution. Network with scale-free property means that the degree distribution of the network exhibits a power tail with an exponent γ. Calculating the coefficient of linear regression, the degree distribution satisfies:

$$P(k) \sim k^{-1.466}$$

This is a scale-free network with exponent $\gamma = 1.466 < 2$. This is a very skewed network, with number of links growing faster than number of nodes. It naturally posses the small world property.

Degree Correlation

From the previous calculation, a basic structure of scale-free network is found. The scale-free network has the "hub" pattern, which connects significantly more people than the majority of developers. Then it raises a question that whether the "hub" is likely to connect to equally influential people? Or will it be actually connected to people less famous?

This is done by calculating the degree correlation of the GitHub social network. It is defined as the Pearson coefficient of degrees between a connected node pair. It is positive if similar vertices (based on degree centrality) tend to connect to each, and negative otherwise.

```
> assortativity_degree(network, directed = FALSE)
[1] -0.07521713
```

The degree correlation is -0.075, which is not a very significant value. The rather small negative correlation illustrates that actually the influential developers are slightly more likely to connect to less influential developers. This is very reasonable in the GitHub social network of describing mutual follower relationship. It is more common for GitHub users to follow a "celebrity" at the first place. After a period of time, having exposed to more information, users start to follow someone less famous.

5.3.4 Diameter

The diameter of the network is defined to be the maximum of the shortest path between nodes. In the GitHub social network, the diameter is 9. It illustrates that all of the developers would have mutual follower relationships in less than 9 steps. This is done by the query as follow:

```
> diameter(network)
[1] 9
```

5.3.5 Average Path Length

The average path length is defined to be the average shortest path between two nodes over all pairs. Let d_{ij} be the shortest path between node i and j, then for an undirected graph G of N nodes, the average path length is calculated as:

$$\ell = \frac{1}{N(N-1)} \sum_{i \neq j} d_{ij}$$

Through the query as follow, the average path length is 3.246409. This further indicates that the network may has the property of a "small-world" model, since on average one developer have mutual followers to another in approximately 3 steps.

```
> mean_distance(network)
[1] 3.246409
```

5.4 Clustering Coefficient

Clustering coefficient measures the relationship of whether neighbours tend to cluster together. This could be illustrated as a 3-clique in the network. The larger the clustering coefficient, the more nodes the network tend to cluster together.

There are two types of clustering coefficients: local and global. The local clustering coefficient is the proportion of links between the vertices within its neighbourhood divided by the number of links that could possibly exist between them. For undirected graph, it is calculated as:

$$C_i = \frac{2\left|\left\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\right\}\right|}{k_i\left(k_i - 1\right)}$$

An edge e_{ij} connects vertex v_i with vertex v_j. And k_i is the number of neighbours of a vertex. The global clustering coefficient is based on triplets of nodes:

$$C = \frac{\text{number of closed triplets}}{\text{number of all triplets (open and closed)}}$$

By calculation, the average local clustering coefficient is 0.1934205 for the whole graph. The global clustering coefficient is 0.01235719. There are 523810 triangles in the network. The clustering coefficient of machine learning developers and web developers are calculated independently, provided in Table 8. The global clustering remains the same since R ignores choosing the vertex ids for this situation.

From Table 8, it could be inferred that, though machine learning network is significantly smaller than the web network, it has higher local clustering coefficient. The machine learning developers tend to have mutual followers with more people in the same discipline.

network	average local cc	global cc
whole network	0.1934205	0.01235719
web network	0.1933899	0.01235719
machine learning network	0.193516	0.01235719

Table 8: Summary of Clustering Coefficients for GitHub Network

5.4.1 Centralization Score

The centralization score is defined to be a graph-level score based on node-level centrality measure. The most centralized structure is some version of the star graph.

By implementing the query below, the centrality score is 0.2504753.

```
> centr_degree(network)\$centralization
[1] 0.2504753
```

5.4.2 Random Network

Plotting a network with equal number of nodes as the GitHub social network would be somewhat hard to get a visual understanding. Thus, some simulations are made with relatively smaller size for comparisons.

Random Network is the most simple network that we could derive by two algorithms. It is also called as Erdos-Renyi network. $G(N, L)$ model connects the nodes with L links randomnly, and $G(N, p)$ model connects each pair of N labeled nodes with probability p, it is a model introduced by Gilbert[3].

Python has functions for generating both functions. Below is a query using the *gnp_random_graph* function in Python. A $G(100, 0.2)$ random graph is generated:

```
>>> G3=nx.generators.random_graphs.gnp_random_graph(100,0.2)
```

From Figure 22, the graph has very similar degrees, and does not have any "hubs" in this network. The mean average path length is 1.83717, and average local clustering coefficient is 0.18678.

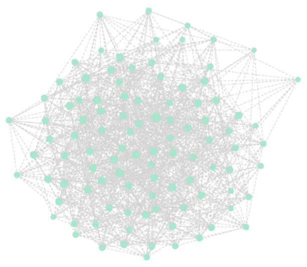
Figure 22: Random Network

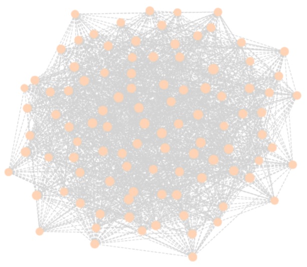
Figure 23: Small-world

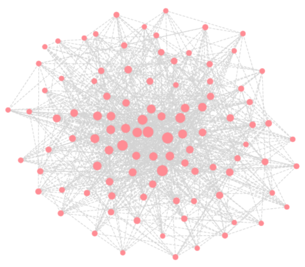
Figure 24: BA Model

5.4.3 Scale-free Network

Scale-free network is a kind of network of specific degree distribution as power-law. In the previous section, the GitHub social network is a scale-free network. The degree distribution follows:

$$P(k) \sim k^{-\gamma}$$

γ is usually a number between 2 and 3.

Since a lot of network has been fond to follow this rule, the Barabasi-Albert model explains successfully how this kind of network could be generated. It defines the preferential attachment depicting the preference of nodes connecting to nodes with larger degree. It explains the "rich get richer" phenomenon, and the formation of hubs. In the query below, the *barabasi_albert_graph* function is applied, with a new node having 10 connections to the previous nodes.

```
>>>G5=nx.barabasi_albert_graph(100, 10)
```

From Figure 23, the degree varies a lot between each nodes. It follows power-law distribution. There are "hubs" in this network. The mean average path length is 1.69717, and average local clustering coefficient is 0.30487.

Small-World Model

The small-world model used here is the original version from Watts and Strogtaz. Since the theory of six-degree of separation came to live, a lot of social networks have been found to follow the same regularity. The average path length between two total strangers are actually shorter than imagined. Small-world model could constructed as special cases of scale-free network or random graph.

Here I used the python *watts_strogatz_graph* function, it contains two parameters k and p other than the number of nodes. Each node is joined with its k nearest neighbors in a ring topology. And p here controls the probability of rewiring each edge. The smaller the value of p, the larger the clustering coefficient[4].

```
>>>G4=nx.generators.random_graphs.watts_strogatz_graph(100, 30, 0.6)
```

From Figure 24, nodes' degrees differs from each other, mildly. For the The mean average path length is 1.86424, and average local clustering coefficient is 0.25549.

5.4.4 Comparison

The three networks constructed above all have 100 nodes, but with different degree distribution and network structure. The average path length, clustering coefficient and degree distribution are compared to make further illustration.

By taking the random graph as the null model, the small-world model has a shorter average mean path, and a larger clustering coefficient. This indicates that nodes inside small-world could reach each other more easily. The scale-free model however, has the largest average path length, but it has very distinct "hubs" and a relatively large clustering coefficient. This is also summarized in Table 9.

network model	average path length	average local cc
random graph G(100, 0.2)	1.83717	0.18678
small-world WS(100, 30, 0.6)	1.69717	0.30487
scale-free BA(100, 10)	1.86424	0.25549

Table 9: Summary of Three Network Structures

Additionally, these three networks have different degree distributions. In Figure 25, the random network has degree that approximately follows a Binomial distribution. A Poisson distribution is usually applied for larger random networks. For Figure 26, this is a network generated by Watts-Strogatz small world model. It usually creates a network with symmetric degree distribution. The scale-free network generated by BA model follows a power-law distribution as in Figure 27. Most nodes in scale-free network have very few degree, while a few nodes have larger degree.

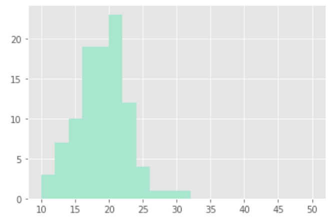
Figure 25: Random Network

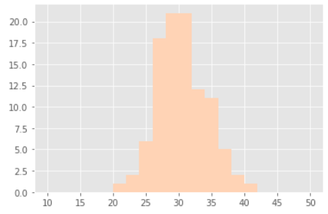
Figure 26: Small-world

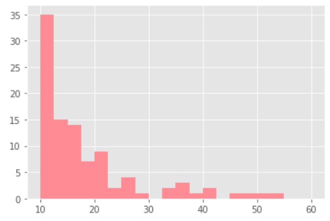
Figure 27: BA Model

6. Discussion and Conclusions

In this report, the data processing with SQL server is shown, especially for transforming from two-mode to one-mode data. Then network visualizations with Python and Gephi are displayed for the whole network, as well as some subgraphs. Then, this report focus on the GitHub social network of mutual followers, finding two influential developers and their roles in the network. based on node level analysis. Two of the community detection algorithms: fast-greedy and Louvain method are provided in order to find out possible community structures. Two obvious communities are found within the web people.

In the section of network level analysis, the GitHub social network is found to be a scale free network with power law degree distribution. Thus, in the later sessions, three network structures are compared, by visualization and statistics.

More works could be done within this data set. For the unbalanced data, it is better to figure out a way of oversampling the minority category for the network data. It could be applied to a classification problem, through Node2Vec, GCN[5] or other node embedding algorithms. The online social network is a topic that worth noted, since researchers nowadays are exposed to more online resources and forums. The spread of COVID-19 makes it also a great opportunity for focusing on online behavioral research.

References

[1] Grover, A., Leskovec, J. *Node2vec*. [*Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*]. 2016.

[2] Alex Bavelas. *Communication patterns in task-oriented groups*. [*J. Acoust. Soc. Am*]. 2(6):725–730,1950.

[3] R. Solomonoff and A. Rapoport. *Connectivity of random nets*. [*Bulletin of Mathematical Biology*]. 13:107-117, 1951.

[4] Saurabh, S., Madria, S., Mondal, A., Sairam, A. S., Mishra, S. *An analytical model for information gathering and propagation in social networks using random graphs*. [*Data Knowledge Engineering*]. 129, 101852, 2020.

[5] Thomas N. Kipf, Max W. *Hybrid Graph Convolutional Networks for Semi-Supervised Classification.* [*Proceedings of 2019 the 9th International Workshop on Computer Science and Engineering*]. 2019.

Appendix

Table of Top 10 PageRank Score for Developer

name	PageRank Score	label
nfultz	0.018495	0
dalinhuang99	0.01096	0
Bunlong	0.010167	0
gabrielpconceicao	0.007469	0
SuriyaaKudoIsc	0.007462	0
addyosmani	0.006735	0
ajsb85	0.006043	0
JonnyBanana	0.00577	0
nikitavoloboev	0.005475	0
mcanthony	0.005472	0

Table 10: Table of Top 10 PageRank Score for Developer

Fast Greedy Algorithm with labels

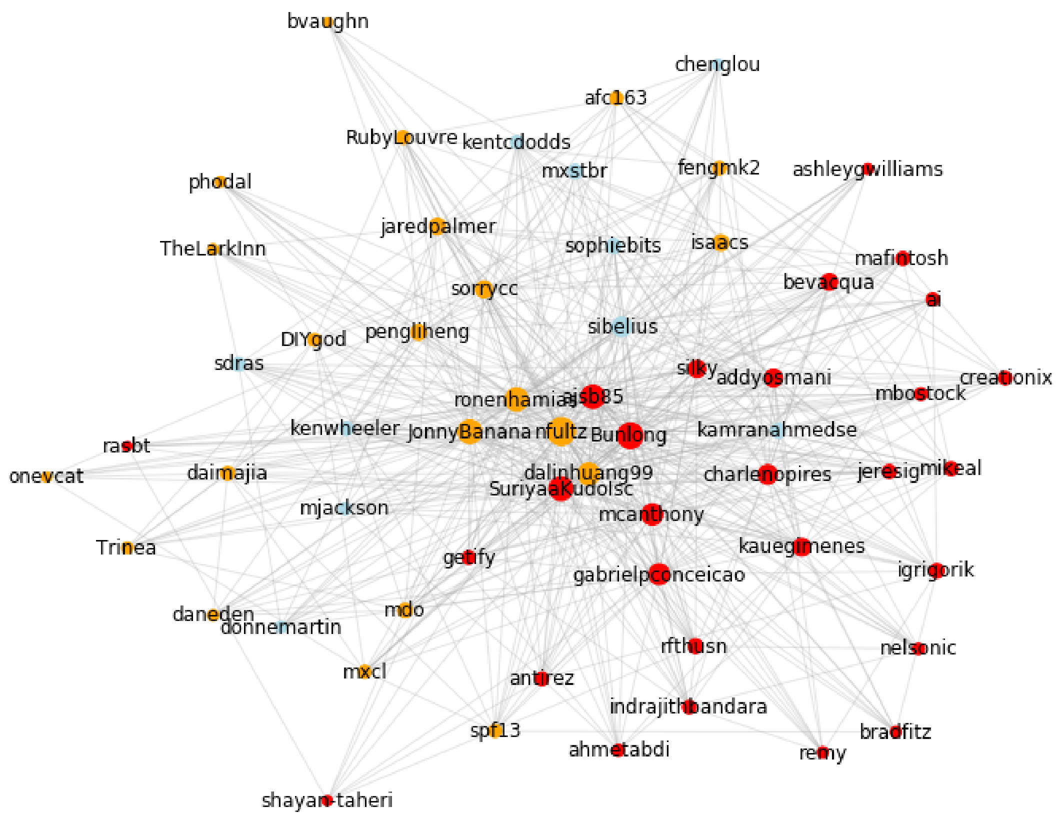

Figure 28: Fast Greedy Algorithm with labels. “nfultz”, “dalinhuang99”, “jonnyBanana” and “ronenhami” plays important role in one community (orange community). “Bunlong” and “SuriyaaKudolsc” play important role in another (red community).

Original subgraph with labels

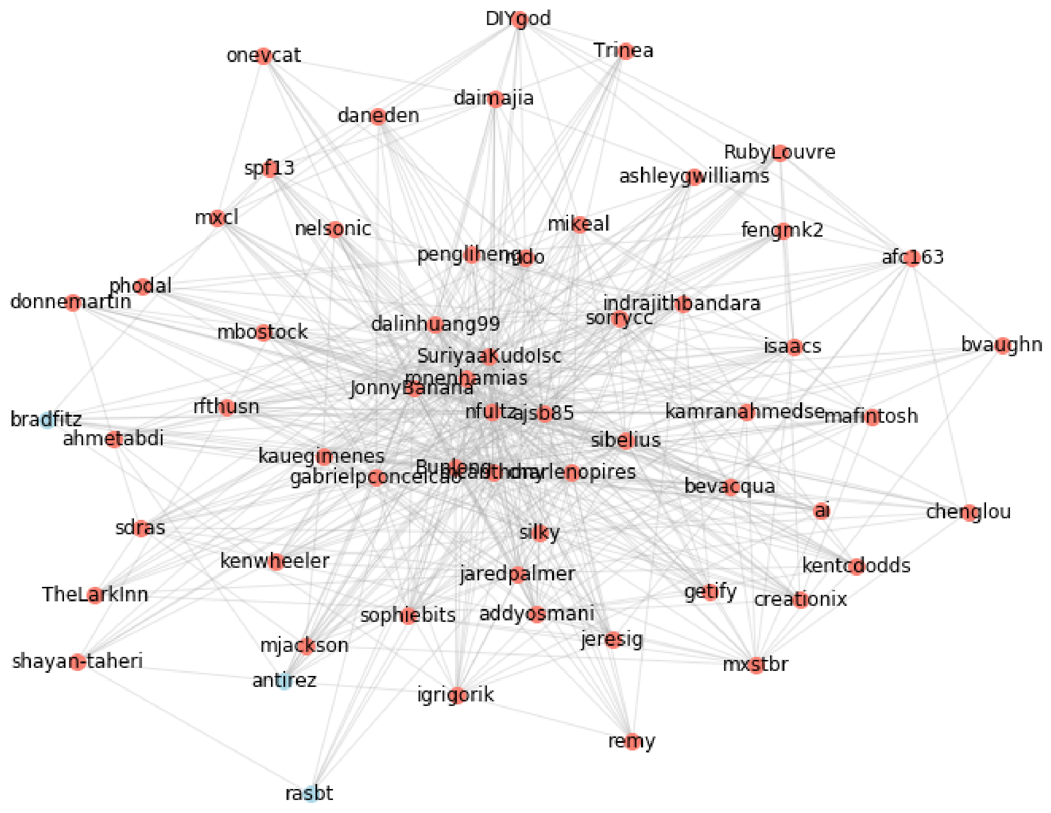

Figure 29: Subgraph with degree larger than 500, there are only three machine learning people: “bradfitz”, “antirez”, and “rasbt”.

Subgraph colored by Girvan-Newman

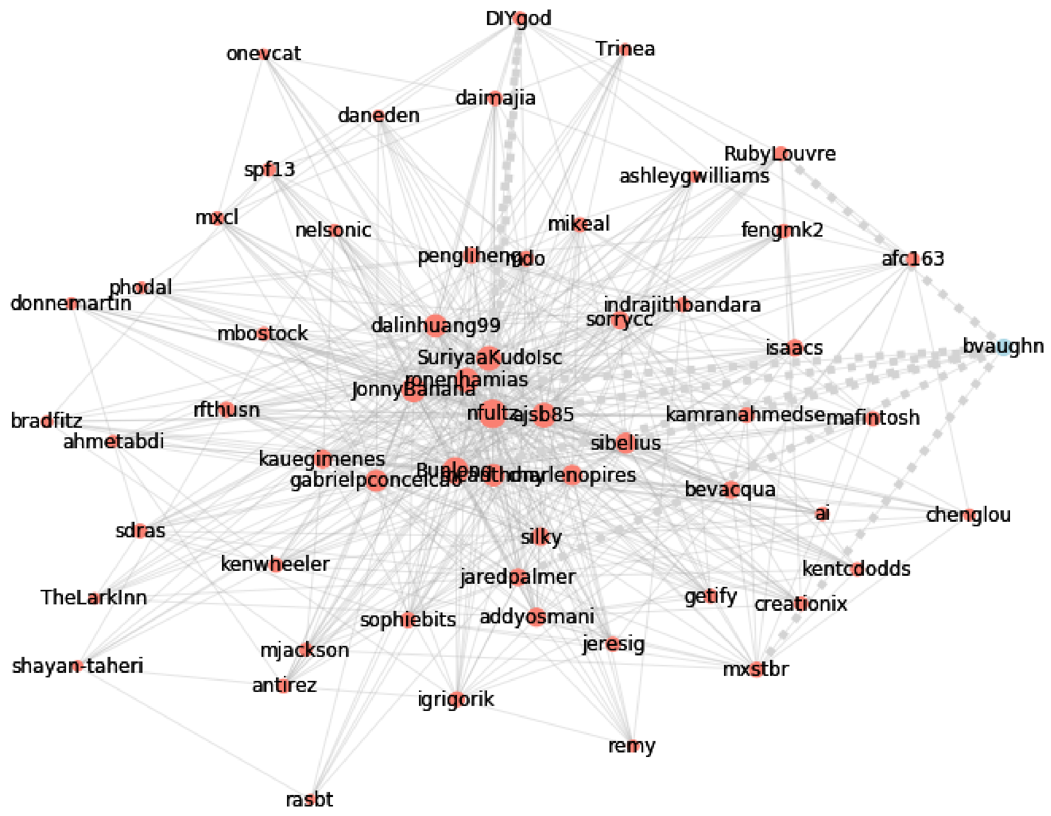

Figure 30: Subgraph with degree larger than 500, “bvaughn” is the first developer being separated from the others. This is possibly the influential developers are highly connected with each other. The most marginal developers are removed at first, instead of testing real community structures.