Course Project Report

Huangxun Chen
1155050519
chenhx1992@gmail.com

Zhihao Zhou
1155014412
cheerzzh@gmail.com

April 15, 2014

1 INTRODUCTION

Various kinds of social network platforms nowadays play an indispensable role in many peoples daily life. People utilize these platforms to build social networks or relations to share interests, activities, and backgrounds and even real-life connection across political, economic, and geographic borders. In this report, we would choose openrice.com, an online social network platform that focuses on food-related experience sharing, as our analyzing target.

Openrice is a Hong Kong-based social network platform which aims to provide a comprehensive, unbiased and user-friendly platform for people to access and share dinning information and experience in Asia area, focusing in Hong Kong, Macao, China, Singapore, Indonesia,the Philippines and Thailand. The core spirit of Openrice can be summarized as Review- Restaurant comments through collective wisdom, Information- Comprehensive Dinning Guide, Community- Dinning experience sharing and Eatery- Link-up eateries with gourmets, which could be abbreviated as RICE[1]. According to 2013 Nelsen Telebus survey in Hong Kong, around 70 percent of respondents will use Openrice to look for dinning and restaurant information, 75 percent of them agree that Openrice provides the most credible restaurant reviews and 84 percent claim that Openrice has truly influential role in their choice of restaurants, which all indicates Openrice is quite successful in serving its purpose. Up to now, more than 811,00 users have registered on Openrice and left over 679,000 comments along with 43,000 restaurants. All these data construct an enormous social network graph from which we will extract and analyze several subsets of the whole graph to get insights and features of Openrice.

2 DATA EXTRACTION

In this project, we write a python crawler to crawl user data from Openrice with the help of some helpful and powerful python libraries, such as urllib, httplib2, BeautifulSoup, re, time and xlwt3. In this section, we will introduce our extraction process, in other words, python crawler algorithm detailedly.

First, we would like to illustrate some data structures and controling variable, which play an important role in our crawler program.

urls(Figure 1): users' url list queue, urls in urls stand for the user page will be crawled later, and they will be crawled one by one from head to tail, if one page has been crawled, we pop it away from this list queue.

visited(Figure 1) : users' url list queue, urls in visited stand for the user page have been found, and its index i means that it is the (i+1)th crawled page. We use visited list to give every user page(node) an unique ID and meanwhile ensure that our program don't crawler the same page repeatedly.

[1] eCMO Conference 2013 - Openrice A local brand created for HK people

network(Figure 1): user information list, every element in this list is a dictionary storing one user page(node) information. Every dictionary has two keys, 'info' and 'follow'. Value for key 'info' is a string, containing all useful information of a particular user, such as user nickname, location, food preference and some numerical parameters. Value for key 'follow' is a ID set storing all users focused by this specific user.

count: the number of nodes the program has crawled. This variable can control the size of the final network, in our program, we set its upper limit is 1200, in other word, our network contain 1200 nodes eventually.

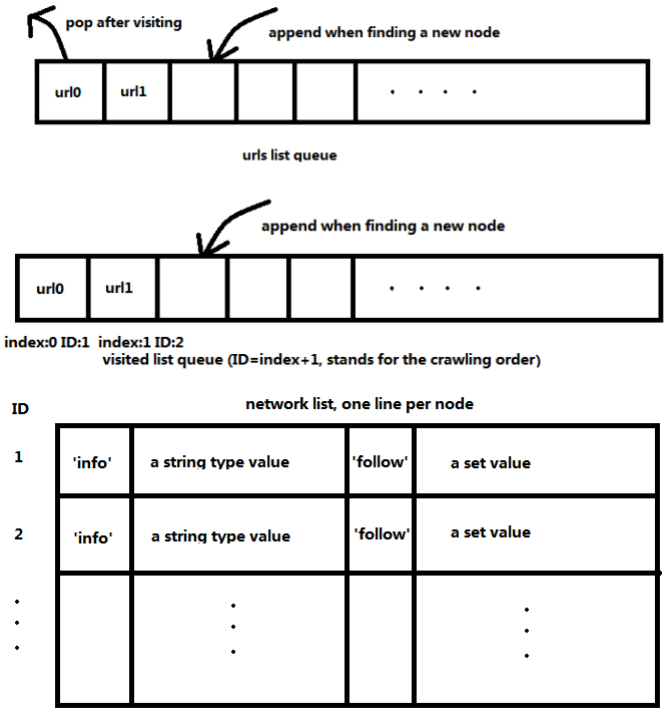

Figure 1

Next, we would like to give snapshot of an user page crawled by our python crawler and show details about our crawl algorithm in Algorithm 1.

Figure 2: *snapshot of an user page crawled by our python crawler*

Algorithm 1 Python Crawler for OpenRice

Input:
start node url;
Output:
A .xls file, containing two sheets, one stores node information, the other stores edge information;

Initial:
urls = [start node url];
visited = [start node url];
network = [];
count = 0;

while urls is not empty **do**
Fetch the head of urls;
Request the corresponding user page(like Figure 2) with the help of libraries urllib and httplib2;
Crawl current page information by using library BeautifulSoup(including Figure 2 ① ②);
Update network list: append a new element, and storing above information for key 'info', and set the value of key 'follow' a empty set;
Enter into "focus" page(Figure 2 ③) and crawl usls of users focused by current user page by page(Figure 2 ⑤);
for every urls found (Figure 2 ④) **do**
if url is not in visited **then**
Append this url to the tail of urls;
Append this url to the tail of visited;
Update current node 'follow' information in network list;
else
Find the index of this url in visited, its ID is index+1;
Update current node follow information in network list;
end if
end for
Pop the head of urls list;
count ← count + 1;
if count > upper limit **then**
break;
end if
end while
return

3 DATA PREPROCESSING

Now all information is stored in network list, every element stands for a node, node information can be retrieved from value of key 'info', edge information can be retrieved from value of key 'follow'. Using regular expression supported by library re, our program parse 'info' string into a information vector and write them into an excel sheet called "Node Graph", one line in sheet records one node's information.

Edge messages provided by value of key 'follow' are retrieved and stored in an excel sheet called "Edge Graph", this sheet contains two columns, one line per edge, meaning the left one follow the right one, thus our network graph is a directed graph.

To sum, our program parse and format the important information and write them into .xls file including two sheets "Node Graph" and "Edge Graph" by making use of powerful features provided by library re and xlwt3.

Figure 3 shows snapshot of crawler output "Node Graph" and "Edge Graph".

ID	name	L1	L2	L3	F1	F2
1	supersupergirl	上環	中環		荷蘭菜	英國菜
2	kanlee	上環	中環	灣仔	韓國菜	日本菜
3	lawrence	大角咀	尖沙咀	旺角	日本菜	意大利菜
4	hedonist	中環	灣仔	尖沙咀	星馬菜	意大利菜
5	yan	尖沙咀	元朗	荃灣	意大利菜	法國菜
6	snoopy	灣仔	元朗	荃灣	小食/熟食店	咖啡茶室
7	480773	中環	銅鑼灣	太古	川菜 (四川)	日本菜
8	spicygal	北角	銅鑼灣	天后	印度菜	日本菜
9	herbert				車品品的西米布甸	
10	garycwm	尖沙咀	美孚	荃灣	京川滬	星馬菜
11	gourmetkc	中環	鰂魚涌	旺角	的菜色有很多	真的數也數不完
12	foodiary	中環	銅鑼灣	旺角	港式	日本菜
13	mjqueen7e				粵菜 (廣東)	港式
14	jamel	銅鑼灣	灣仔	尖沙咀	滬菜 (上海)	日本菜

Source	Target
1	2
1	3
1	4
1	5
1	6
1	7
1	8
1	9
1	10
1	11
1	12
1	13
1	14
1	15

Figure 3: *crawler output "Node Graph" and "Edge Graph"*

4 DATA FURTHER PROCESSING

As we can see above, on one hand, the data store in Node Graph contain many traditional Chinese characters which is not convenient for data analysis later; on the other hand because the food preference messages on web page are written by users themselves, their format are quite different and our python program actually finishes part of formating jobs. Therefore, the data need further processing.(Figure 4)

Figure 4: *Original data set needs further processing*

First, we find the set of location and food preference and then construct two mapping table.(Figure 5) Later we replace the original value with the corresponding number in mapping table.

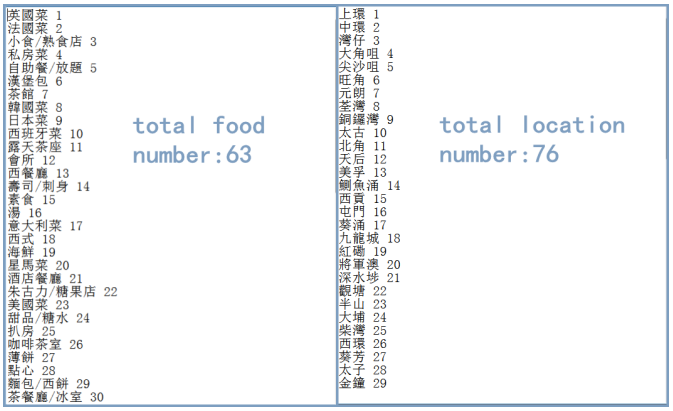

Figure 5: *mapping table for location and food preference*

Then, for some food preference which are not standard like ones in Figure 4, we delete them directly. We think this information loss does little harm to our network and analysis because they account for a very small portion of the

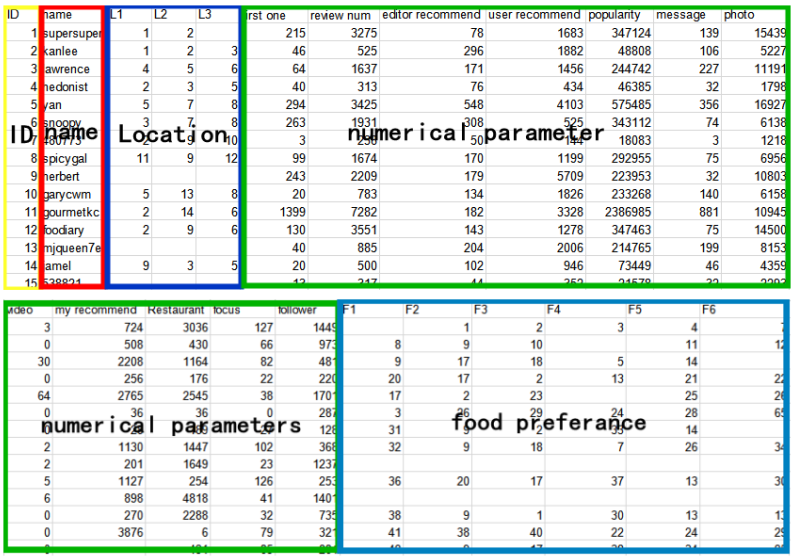

Figure 6: *Data set after further data processing*

whole information.

Above processing is done by manually using some functions and features provided by Excel. And data set after further data processing is shown in Figure 6.

5 NETWORK CONSTRUCTION

We construct our network graph based on our processed information in the open-source software Gephi. Since we want to focus more on data analysis, so we put all the detailed processure of constructing network graph in Gephi in Appendix A.

6 DATA ANALYSIS

6.1 General Network Structure

In this project, we select different start nodes randomly as the input of Algorithm 1.

ID: supersupergirl url:http://supersupergirl.openrice.com/

ID: 300627060 url:http://www.openrice.com/restaurant/userinfo.htm?userid=300627060

ID: 92146 url: http://www.openrice.com/restaurant/userinfo.htm?userid=92146

ID: 813584 url: http://www.openrice.com/restaurant/userinfo.htm?userid=813584

Thus we have several datasets with the same size but a slightly different network structure. Section Small World Penomenon is based on the start node with id 813584. Section Homophily Analysis and Power Law of In-degree is based on the start node with id 300627060.

As shown in Figure 7, nodes are painted based on their modularity class caculated by the built-in algorithm which is carried on by Gephi(Figure 8).

6.2 Small World Penomenon

There exists a basic structure issue in many social networks that nodes can be usually connected by very short path through the whole network. This fact that social network are abundant in short path linking even remote nodes is known as Small World Phenomenon, or the Six degrees of separation. In order to test whether this effect exist in social network of Openrice, we constructed one larger user graph containing 3000 users(Figure 7), starts with one new randomly chose user with ID: 813584(http://www.openrice.com/restaurant/userinfo.htm?userid=813584). By the computation results of Gephi, the average path length between any two arbitrage users in this graph is around 4.851, which indicates on average, one user can reach any other user within around 4.851 steps on average. Hence, these are strong evidences imply the existence of small world phenomenon in Openrice user network. This may due to the extreme popularity of some so-called core user- those who regularly post large number of comments, photos and articles and thus attract lots of followers. These core users somehow serve as hubs in the network to connect remote users.

This small world phenomenon in Openrice is particularly important for restaurants, which want to diffuse new information through the network, since they can make use of the core users to target much more users than simply use broad spreading method.

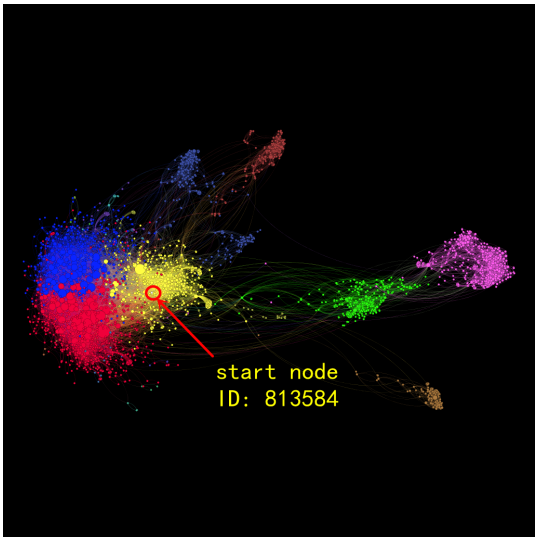

Figure 7: *Network Graph for crawled dataset with 3000 nodes*

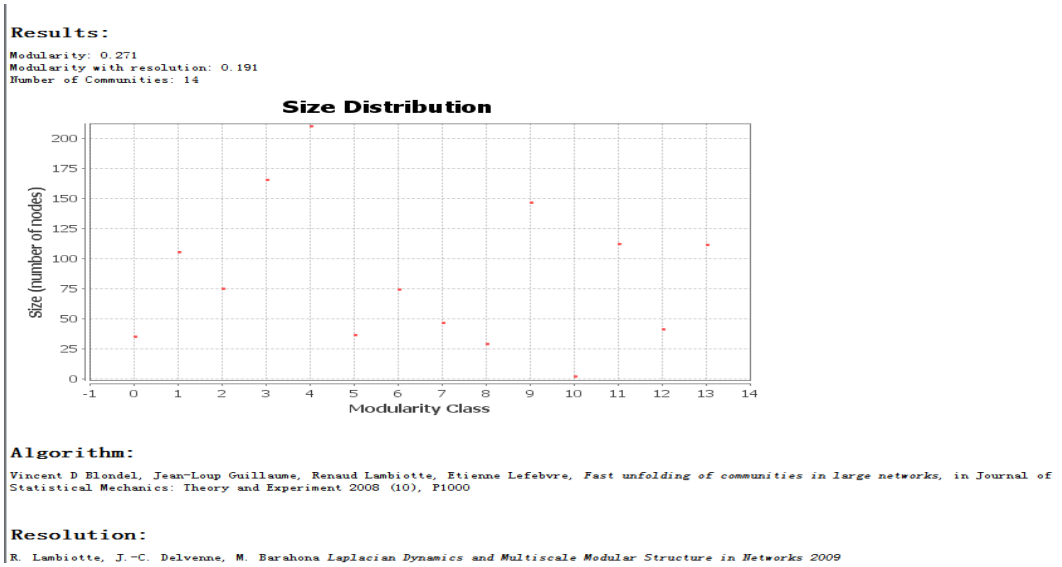

Figure 8: *Modularity class distribution result of network whose start node is ID 92146*

6.3 Homophily Analysis

6.3.1 Location Preference Analysis

In this subsection, we expect to test our network's homophily feature upon location preference information. We divided location labels into six groups according to their real place on Hongkong subway map(Figure 9). By the way, we use Hongkong subway map rather than Hongkong map because subway trasportation is highly developed in Hongkong, which has a heavy impact on people's concept of distance and trip mode.Therefore, our division actually depends on the subway route length between each other. Then, we carry on some filter operations on the whole graph(Figure 10) so that we obtain six network graphs.(Figure 11 12 13 14 15 16)

According to filter result, not surprisingly, most people prefer resterants in Zone 1 and Zone 2. As we all known, bunch of resturants with different flavors gather in these two areas and meanwhile most of white collar workers work there. Above two factors lead to that about 1/3 people in our network choose Zone 1 as their location label and around 1/4 choose Zone 2. What is more, we could observe that compared to other zones, Zone 1 group and Zone 2 group have higher value of edges/nodes, which means that they are more compacted. The possible reason is that people in these two group are more active on Openrice website, in other word, they are more willing to share their own experience and also often focus others' sharing and comments. We guess maybe large portion of them are the young.

It seems that location preference don't hold quite obvious homophily feature in our network, for all group in Figure 11 to Figure 16 almost have nodes with all possible colors, especially Zone 1 group and Zone 2 group. Node with the same modularity class may not have same location preference. The possible reason is that people with their own location preference also would like to focus some people with different location preference. Subway transportation is so convenient that they could seek food in their own location preference at weekdays and in other locations at weekends.

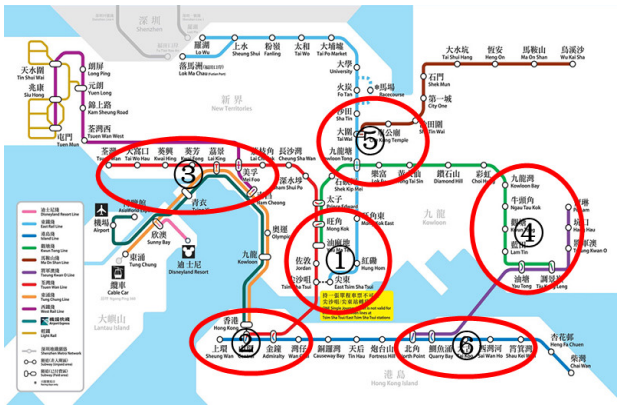

Figure 9: *HongKong Map*

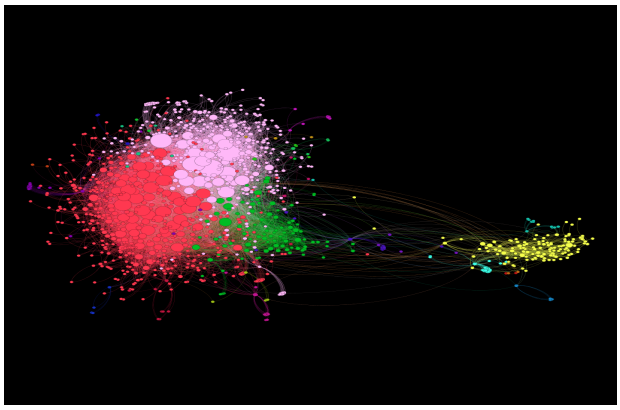

Figure 10: *The whole Network Graph (node: 1201; edges: 12582)*

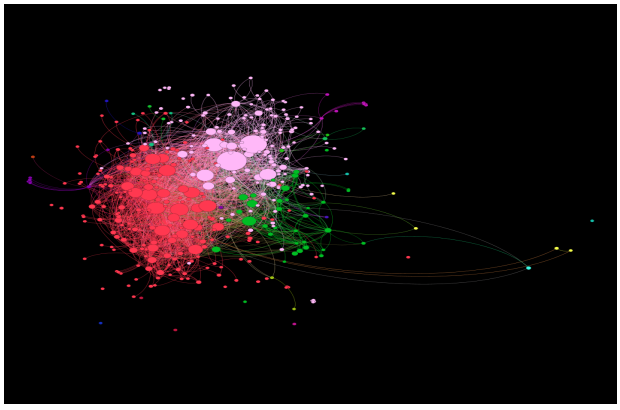

Figure 11: *Network Graph of people whose location preference is Zone ① (nodes: 400; edges: 2490)*

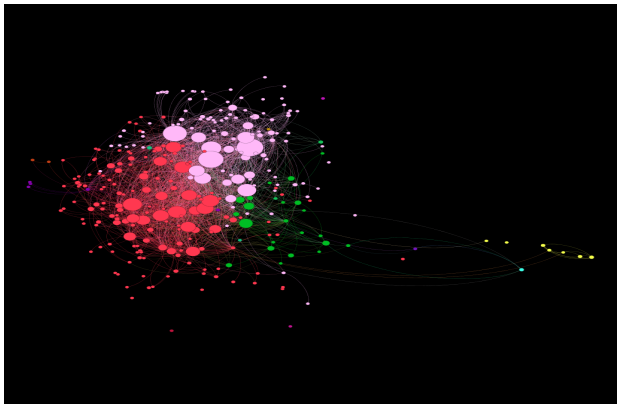

Figure 12: *Network Graph of people whose location preference is Zone ② (nodes: 311; edges: 2495)*

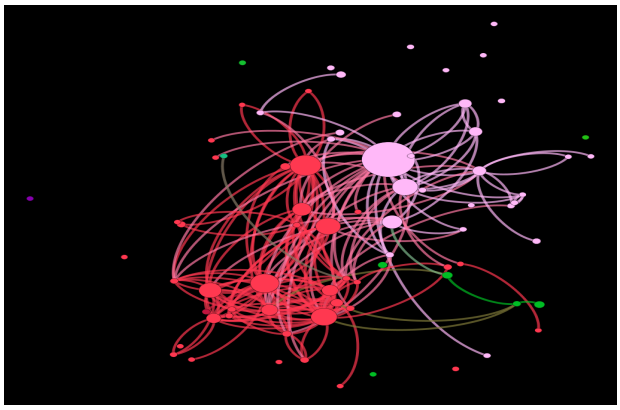

Figure 13: *Network Graph of people whose location preference is Zone ③ (nodes: 76; edges: 205)*

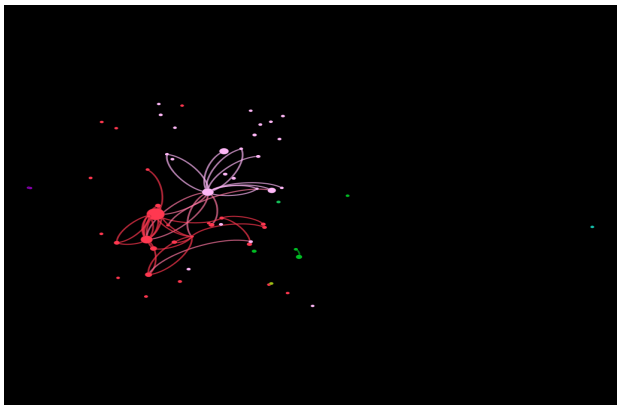

Figure 14: *Network Graph of people whose location preference is Zone ④ (nodes: 60; edges: 42)*

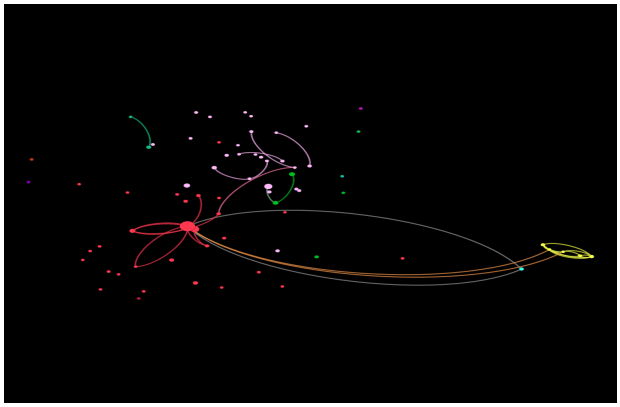

Figure 15: *Network Graph of people whose location preference is Zone ⑤ (nodes: 72; edges: 33)*

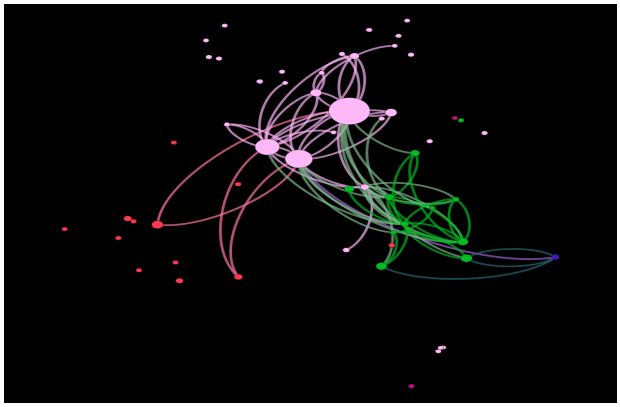

Figure 16: *Network Graph of people whose location preference is Zone ⑥ (nodes: 56; edges: 77)*

6.3.2 Food Preference Analysis

In this subsection, we expect to test our network's homophily feature upon food preference information. We choose seven different cuisine all around the world. including Hongkong, Japan, France, Shanghai, Taiwan, Singapore and India. Then, we carry on some filter operations on the whole graph(Figure 17) so that we obtain seven network graphs.(Figure 18 19 20 21 22 23 24)

According to filter result, Japan cuisine are the most popular one which has about 5/12 nodes of the whole network. Similar size of nodes choose Hongkong cuisine and France cuisine, however, "Hongkong cuisine" group with less edges are more scattered than "France cuisine" group. That is possibly because Hongkong cuisine are very common in Hongkong. Almost all people in Hongkong are so familiar with Hongkong cuisine that they feel there is no need to focus others who also like Hongkong cuisine. But situation is different for France cuisine, not all people know France cuisine well. Thus they need to focus someone who has similar taste to get some comments or suggestions.

Similar with location preference analysis, it seems that food preference don't hold quite obvious homophily feature in our network, for all group in Figure 18 to Figure 24 almost have nodes with all possible colors, especially "Hongkong" group, "France" group and "Japan" group. Node with the same modularity class may not have same food preference. The possible reason is that nowadays people are more curious to different flavors and cuisines and willing to have a try. Thus, people with their own food preference also would like to focus some people with different food preference.

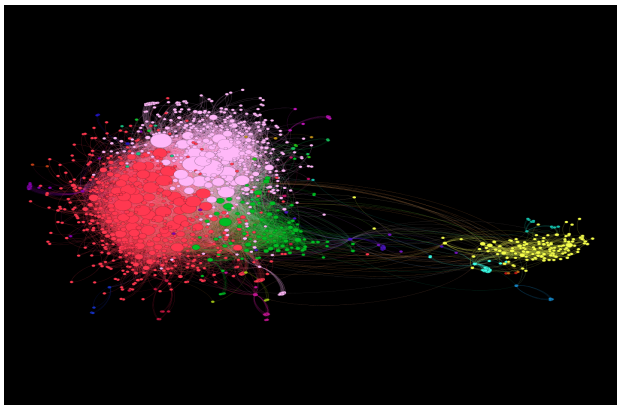

Figure 17: *The whole Network Graph (node: 1201; edges: 12582)*

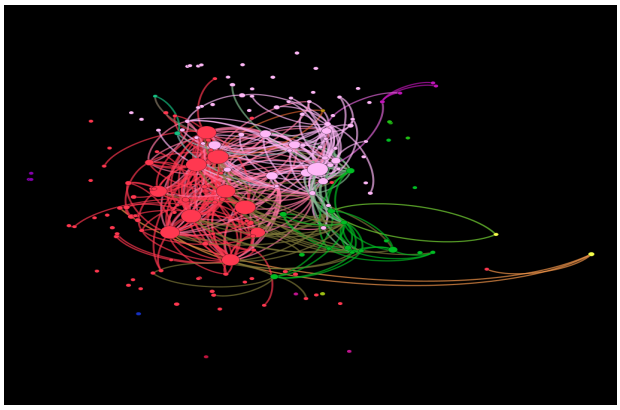

Figure 18: *Network Graph of people whose food preference is "Hong Kong cuisine" (nodes: 177; edges: 389)*

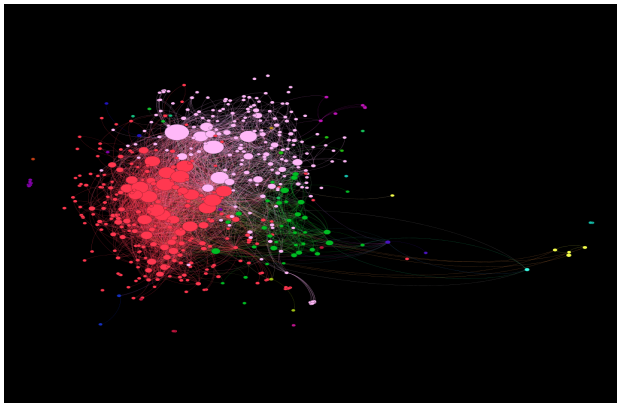

Figure 19: *Network Graph of people whose food preference is "Japan cuisine" (nodes: 494; edges: 3120)*

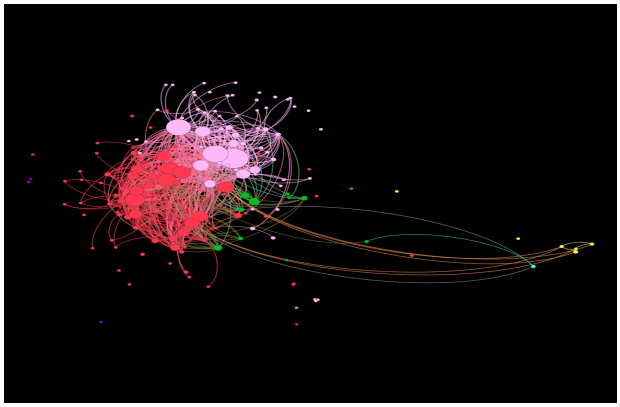

Figure 20: *Network Graph of people whose food preference is "France cuisine" (nodes: 161; edges: 776)*

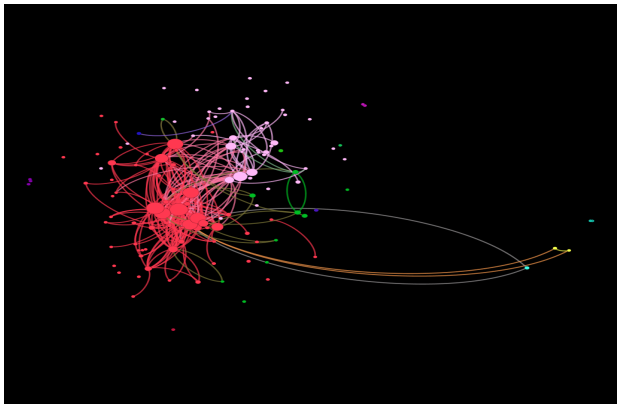

Figure 21: *Network Graph of people whose food preference is "Shanghai cuisine" (nodes: 129; edges: 263)*

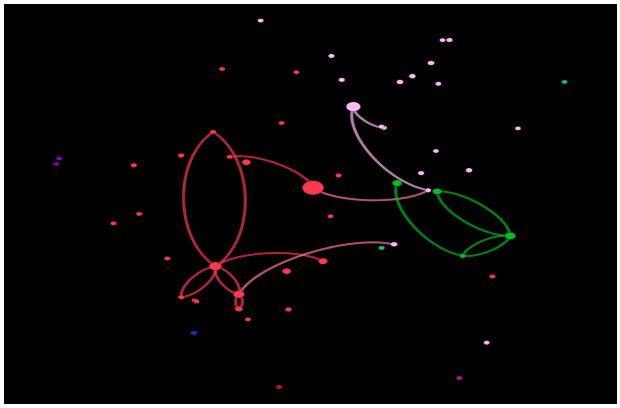

Figure 22: *Network Graph of people whose food preference is "Taiwan cuisine" (nodes: 55; edges: 19)*

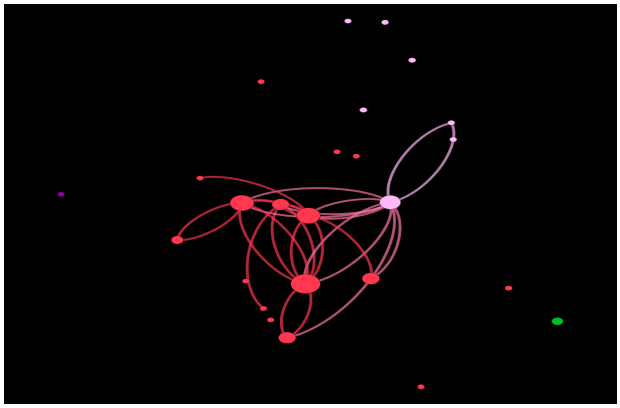

Figure 23: *Network Graph of people whose food preference is "Singapore cuisine" (nodes: 25; edges: 28)*

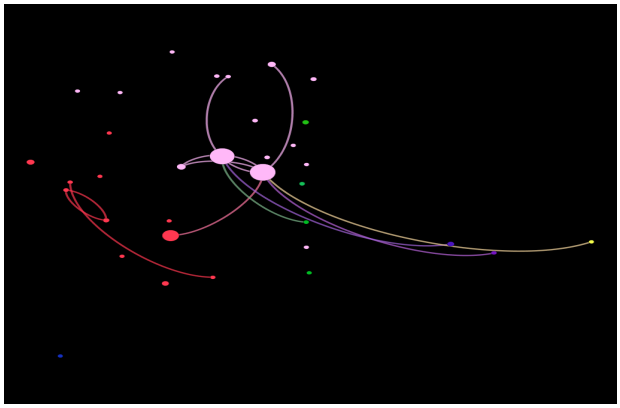

Figure 24: *Network Graph of people whose food preference is "Indian cuisine" (nodes: 34; edges: 14)*

6.4 Power Law of In-degree

In Openrice, users can follow other users and the more followers one have, we may regard him as more popular. In most social network, popularity is a phenomenon featured by extreme imbalance: most users only achieve limited followers, a few users get wider visibility and very few users could attain global name recognition. We would like to test whether the power law of in-degrees, which states k number of in-degree (number of followers) will decrease as k to some fixed power, would approximately hold within Openrice network.

Estimation Method:

(Date set: 300627060)

Denote k as number of in-degree, f(k) be the number of users who has value k. The target model is $f(k) = a/k^2$. After taking logarithms of both sidewe get $logf(k) = log(a) - clog(k)$. This indicates that if the power law of in-degree holds, we should observe a linear relationship between log(k) and log f(k). Figure 26 shows the plot of log f(k) against log(k). After applying the simple linear regression, we could obtain the least-square estimator of the intercept and coefficient of the line, which is 5.1807 and -1.118 respectively. The Adjusted R-squared is around 0.8124, and the p-value for F-test is less than 2.2e16, which suggest a high goodness-of-fit. Thus the estimated power law function is approximately in following manner:

$logf(k) = log(177.8127) - 1.118 * k$

or equivalently, $f(k) = 177.8127/k^{1.118}$

Plot the estimated power law function along with the raw data (Figure 25), we could find the approximation is quite good, hence we may conclude that the power law of in-degree holds in Openrice user network.

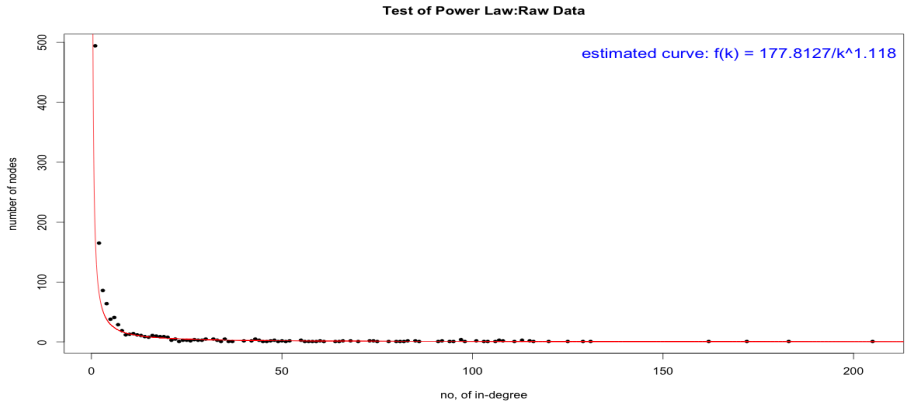

Figure 25: *Test of Power Law: Raw Data*

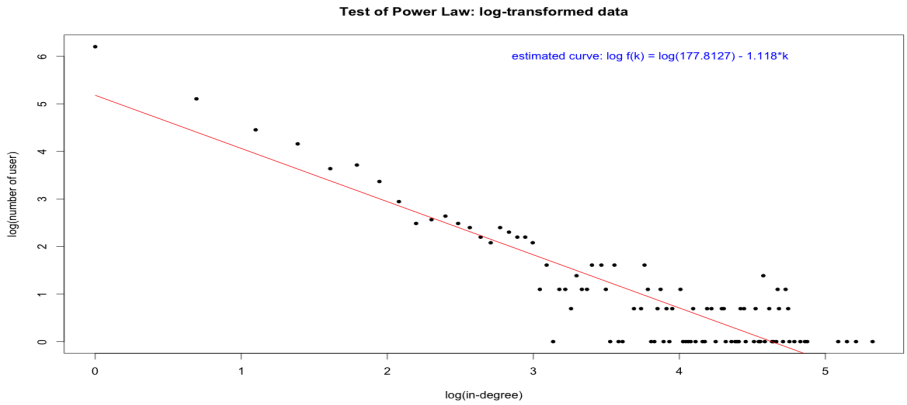

Figure 26: *Test of Power Law: log-transformed Data*

7 CONCLUSION

In this project, we designed algorithm in Python script to ectract data from Openrice.com to form various user network graphs. Some basic analysis and verification of social networks are implemented with the assistance of open-source softwares Gephi and R.

Homophily feature seems not held by our network obviously for the reason that people with their own location and food preference are also willing to focus others with quite different preference out of their curiosity. Thus, people in the same cluster may have abundant taste preference.

Small world phenomenon and power law of in-degree were briefly verified in Openrice user network, which is particularly relevant for restaurant who would like to promote themselves through Openrice, since they can utilize the core users to maximize the information diffusion effect.

It is appropriate to discuss some of the limitation within our study. First, all our analysis were performed on some partial graph of the entire Openrice user network since it is infeasible to extract the whole user network due to its enormous size. Second, our extraction algorithm would extract users' imfomation around the starting user url level by level and the total number of extracted nodes is 1200, which is still small portion of the whole user network so that it is inclined to construct a close-knit cluster rather than several clusters. Thus maybe there exist some deviations when applying our conclusions to the whole user network.

Due to the some technical limitation, it is difficult to extract more detail data regarding users' dynamic actions from Openrice. Further study of Openrice might focus on the dynamic structure of user network with emphasis on information cascade through the network. Further, the relationship between users and restaurants is also a interesting topic to work on. What is more, analysis on some particular users' behaviour mode and the effect they have on other nodes also deserves researching.

A Construct network graph in Gephi

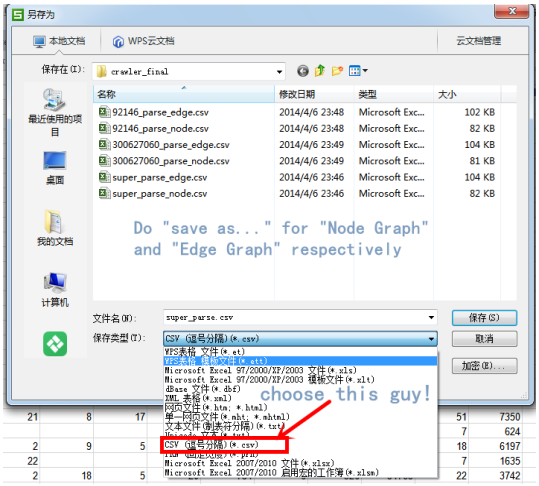

Figure 27: *Save Node Graph and Edge Graph as csv format*

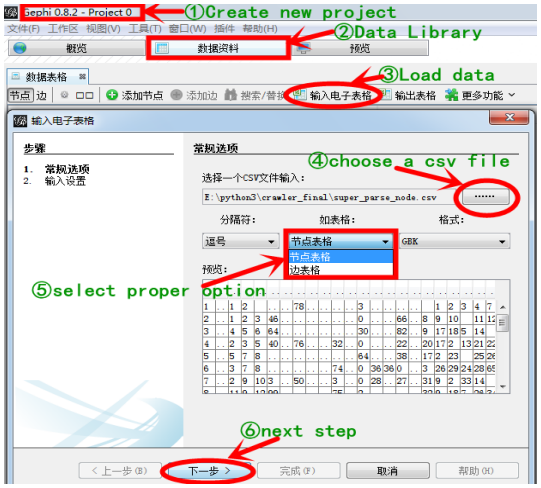

Figure 28: *Load data into Gephi*

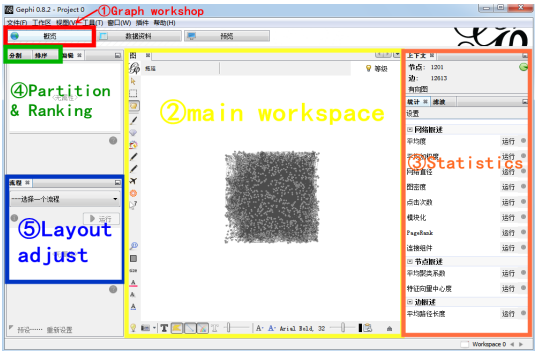

Figure 29: *Gephi user interface*

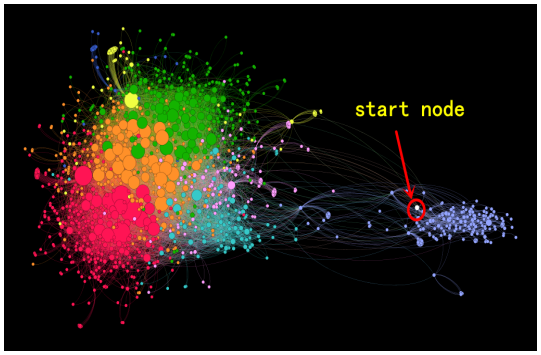

Figure 30: *One of our network graph constructed and polished by using Gephi*

B Source Code crawler.py

```
# -*- coding: utf-8 -*-
from bs4 import BeautifulSoup
import httplib2
import urllib
import re
import time

h = httplib2.Http('.cache')
#start node
url = "http://www.openrice.com/restaurant/userinfo.htm?userid=813584"

#nodes need to be visited
urls = [url]
#nodes has been visited
visited = [url]
#current node "guanzhu" page need to be visited
pageurl = []
#information about known nodes
network = []
count = 0
```

```
#url pattern
pattern1 = re.compile(r'http://\S*.openrice.com\$')
pattern2 = re.compile(r'http://www.openrice.com/restaurant/userinfo')
pattern3 = re.compile(r'//\S*.openrice.com\$')
pattern4 = re.compile(r'/restaurant/userinfo.htm')

#info pattern
location = re.compile(u'最常於\D*出沒')
preference = re.compile(u'最鍾意\D*。')
shouweishipin = re.compile(u'\d*次成為首位寫食評會員')
shipinshumu = re.compile(u'食評數目\d*')
bianjituijie = re.compile(u'編輯推介數目\d*')
huiyuantuijie = re.compile(u'會員推介次數\d*')
renqizhishu = re.compile(u'人氣指數\d*')
liuyanshumu = re.compile(u'留言數目\d*')
shangzaixiangpian = re.compile(u'上載相片\d*')
shangzaiyingpian = re.compile(u'上載影片\d*')
tuijieshipin = re.compile(u'推介的食評\d*')
canting = re.compile(u'我的餐廳\d*')
guanzhu = re.compile(u'關注\d*')
fensi = re.compile(u'粉絲\d*')

while len(urls)>0:
    try:
        print("count="+str(count))
        print("urls[0]:" + urls[0])
        if (pattern1.match(urls[0])):
            string = urllib.parse.urljoin(urls[0], "/home/bookmarkuser.htm")
        if (pattern2.match(urls[0])):
            string = re.sub('restaurant/userinfo', 'gourmet/bookmarkuser', urls[0])
        print("str:" + string)
        while True:
            try:
                response, htmltext = h.request(string, headers={'cache-control':'no-cache'})
                break
            except:
                time.sleep(5)
                continue
        print(response.status)
        print("len_text="+str(len(htmltext)))
        while (len(htmltext) < 20000):
            time.sleep(5)
            while True:
                try:
                    response, htmltext = h.request(string, headers={'cache-control':'no-cache'})
                    break
                except:
                    time.sleep(5)
                    continue
            print(response.status)
            print("len_text="+str(len(htmltext)))
        print("hi")
    except:
        print("fail")

    soup = BeautifulSoup(htmltext)
    info_group = soup.find_all("div", "info")
    if len(info_group) > 1:
        temp = "".join(info_group[1].get_text().split())
```

```
    info = string+" "+info_group[0].get_text() + temp #get current node information
else:
    if len(info_group) == 1:
        temp = "".join(info_group[0].get_text().split())
        info = string+" "+temp #get current node information
    else:
        info = string
print(info)
network.append({'info': info, 'follow':set()}) #update network

#get current node's "focus"
pages = soup.find_all("a", "number")
if len(pages)==0:
    pageurl.append(string)
else:
    pageurl.append(string)
    for i in range(len(pages)):
        tempstr = urllib.parse.urljoin(urls[0], pages[i].get('href'))
        if tempstr not in pageurl:
            pageurl.append(tempstr)

print("#pageurl")
print(pageurl)

follows = []
while len(pageurl)>0:
    while True:
        try:
            response, htmltext = h.request(pageurl[0], headers={'cache-control':'no-cache'})
            break
        except:
            time.sleep(5)
            continue

    print(response.status)
    print("len_text="+str(len(htmltext)))
    while (len(htmltext) < 20000):
        time.sleep(5)
        while True:
            try:
                response, htmltext = h.request(pageurl[0], headers={'cache-control':'no-cache'}
                                                                    )
                break
            except:
                time.sleep(5)
                continue
        print(response.status)
        print("len_text="+str(len(htmltext)))

    soup = BeautifulSoup(htmltext)
    for tag in soup.find_all("span", "menpiccomment"):
        follow = tag.find_previous_sibling('a').get('href')
        if (pattern3.match(follow)):
            follow= urllib.parse.urljoin("http:", follow)
        if (pattern4.match(follow)):
            follow= urllib.parse.urljoin("http://www.openrice.com", follow)

        print(follow)
        follows.append(follow)
```

```
            if follow not in visited:
                urls.append(follow)
                visited.append(follow)
                network[count]['follow'].add(len(visited))
            else:
                index = visited.index(follow)+1
                network[count]['follow'].add(index)

        pageurl.pop(0)

    urls.pop(0)
    count = count+1

    if count>1200: #control the number of nodes
        break

from tempfile import TemporaryFile
from xlwt3 import Workbook
book = Workbook()
book.add_sheet('Node Graph')
book.add_sheet('Edge Graph')
edge_row = 0
for i in range(len(network)):
    #write "Edge Graph" sheet
    sheet1 = book.get_sheet(0)
    for k in range(25):
        sheet1.col(k).width = 5000

    sheet1.write(i+1, 0, i+1)
    m = re.search(location, network[i]['info'])
    if m != None:
        loca = m.group(0)[3:-2].split(', ')
        for j in range(len(loca)):
            sheet1.write(i+1, j+1, loca[j])
    else:
        print("no location")

    m = re.search(preference, network[i]['info'])
    if m != None:
        food = m.group(0)[3:-1].split(', ')
        print(food)
        for z in range(len(food)):
            sheet1.write(i+1, z+5, food[z])
    else:
        print("no preference")

    m = re.search(shouweishipin, network[i]['info'])
    if m!= None:
        number = m.group(0)[:-10]
        print(number)
        sheet1.write(i+1, 30, number)
    else:
        print("error")

    m = re.search(shipinshumu, network[i]['info'])
    if m!= None:
        number = m.group(0)[4:]
        print(number)
        sheet1.write(i+1, 31, number)
    else:
```

```
        print("error")

m = re.search(bianjituijie, network[i]['info'])
if m!= None:
        number = m.group(0)[6:]
        print(number)
        sheet1.write(i+1, 32, number)
else:
        print("error")

m = re.search(huiyuantuijie, network[i]['info'])
if m!= None:
        number = m.group(0)[6:]
        print(number)
        sheet1.write(i+1, 33, number)
else:
        print("error")

m = re.search(renqizhishu, network[i]['info'])
if m!= None:
        number = m.group(0)[4:]
        print(number)
        sheet1.write(i+1, 34, number)
else:
        print("error")

m = re.search(liuyanshumu, network[i]['info'])
if m!= None:
        number = m.group(0)[4:]
        print(number)
        sheet1.write(i+1, 35, number)
else:
        print("error")


m = re.search(shangzaixiangpian, network[i]['info'])
if m!= None:
        number = m.group(0)[4:]
        print(number)
        sheet1.write(i+1, 36, number)
else:
        print("error")


m = re.search(shangzaiyingpian, network[i]['info'])
if m!= None:
        number = m.group(0)[4:]
        print(number)
        sheet1.write(i+1, 37, number)
else:
        print("error")

m = re.search(tuijieshipin, network[i]['info'])
if m!= None:
        number = m.group(0)[5:]
        print(number)
        sheet1.write(i+1, 38, number)
else:
        print("error")
```

```
    m = re.search(canting, network[i]['info'])
    if m!= None:
        number = m.group(0)[4:]
        print(number)
        sheet1.write(i+1, 39, number)
    else:
        print("error")

    m = re.search(guanzhu, network[i]['info'])
    if m!= None:
        number = m.group(0)[2:]
        print(number)
        sheet1.write(i+1, 40, number)
    else:
        print("error")

    m = re.search(fensi, network[i]['info'])
    if m!= None:
        number = m.group(0)[2:]
        print(number)
        sheet1.write(i+1, 41, number)
    else:
        print("error")

    m = re.search("bookmarkuser.htm\?userid=\d*", network[i]['info'])
    if m!=None:
        ID = m.group(0)[24:]
        print(ID)
        sheet1.write(i+1, 42, ID)
    else:
        m = re.search(":[\s\S]*.openrice", network[i]['info'])
        if m!=None:
            ID = m.group(0)[3:-9]
            print(ID)
            sheet1.write(i+1, 42, ID)
        else:
            print("error")

    #write "Edge Graph" sheet
    sheet2 = book.get_sheet(1)
    for e in network[i]['follow']:
        if e <= len(network):
            sheet2.write(edge_row, 0, i+1)
            sheet2.write(edge_row, 1, e)
            edge_row += 1

book.save('network_infi.xls')
book.save(TemporaryFile())
```