Social Network 社交网络分析

2193211134 吴博成 *Instructor:* 刘跃文

§ 基于“Facebook Government”数据的社交网络分析 §

Problem 1: 数据集介绍

(1) 数据来源 本文选用 networkrepository 提供的 **FB-PAGES-GOVERNMENT** 数据 [2]。该数据集收集了截至 2017 年 11 月带有政府认证标识 (蓝色标识) 的 Facebook 账号的相互关注数据。其中节点代表账号，边是它们之间的相互关注关系，为有向边。

图 1: Facebook 政府认证账号示例

(2) 数据特征 该社交网络数据有 7057 个节点，89455 条有向边；即共收集了 7057 个政府账号的 89455 个关注关系。该图是弱连通图不是强连通图。

网络较为稀疏，密度约为 0.001796:

$$density = \frac{\text{图的边数}}{\text{图的节点数 (图的节点数-1)}}$$

节点间联系很紧密，平均路径长度为 0.8156:

$$a = \sum_{s,t \in V} \frac{d(s,t)}{n(n-1)}$$

其中 v 是 G 中的节点集合，$d(s\,t)$ 是从 s 到 t 的最短路径，n 是 G 中的节点数。

由于图并非强连通图，因此转为无向图计算，直径为 10。

该图的平均聚类系数为 0.2054，是网络中所有节点的聚类系数的平均值，这表明朋友之间互为朋友的平均概率较低:

$$C = \frac{1}{N}\sum_{i=1}^{N} C_i$$

$$C_i = \frac{E_i}{\frac{1}{2}k_i\,(k_i-1)} = \frac{2E_i}{k_i\,(k_i-1)}$$

其中某节点 i 的聚类系数为其邻节点中实际相连的边数除以该节点的邻节点相互连接的最大可能边数。

该社交网络图的平均度为 12.676，节点的度分布如下图，呈现幂律分布。

该网络度同配系数为 0.0645，网络呈现同配性结构，度大的节点倾向于和度大的节点相连。

(3) 数据可视化

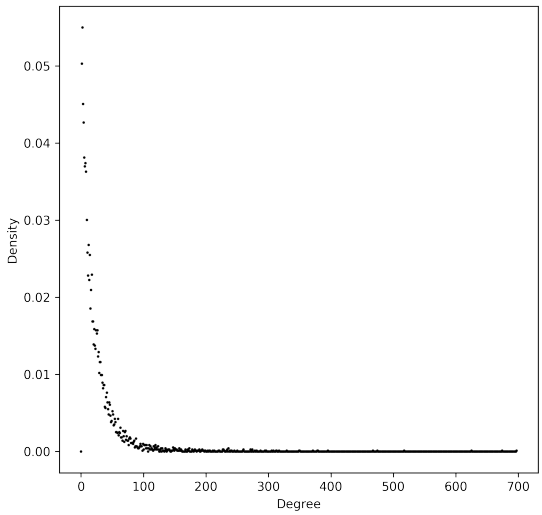

图 2: 节点度分布图

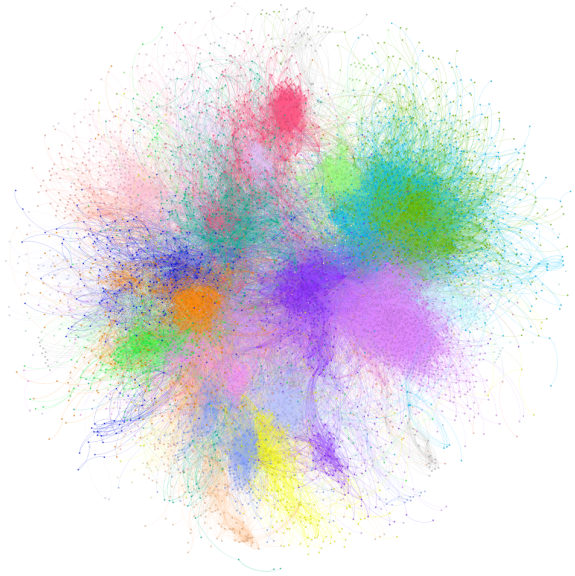

图 3: FB-PAGES-GOVERNMENT 社交网络图

Problem 2: 节点重要性度量

(1) 度中心性 Degree Centrality 节点的度为一个节点连边的多少，体现了该节点的局部重要性。在一个包含 N 个节点的网络中，节点最大的可能度为 N1，因此归一化的度即度中心性为:

$$DC_i = \frac{k_i}{N-1}$$

(2) 介数中心性 Betweeness Centrality 节点 v 的介数中心性是指图中任意两个节点对之间的最短路径当中, 其中经过 v 的最短路径的所占的比例。即经过 v 的最短路径越多, 节点 v 越重要。假如介数中心性高的点消失了，那么其他点之间的交流会变得困难，甚至可能断开。

$$BC_i = \sum_{s \neq i \neq t} \frac{n_{st}^i}{g_{st}}$$

其中, g_{st} 为从节点 s 到节点 t 的最短路径的数目, n_{st}^i 为从节点 s 到节点 t 的 g_{st} 条最短路径中经过节点 i 的最短路径的数目。

(3) 接近中心性 Closeness Centrality 对于网络中的每一个节点 i, 计算与其它所有节点的平均距离为:

$$d_i = \frac{1}{N} \sum_{j=1}^{N} d_{ij}$$

节点 i 的接近中心性 CC_i : 节点 i 与其它所有节点的平均距离 d_i 的倒数。一个点的接近中心度较高, 说明该点到网络中其他各点的距离总体来说较近, 反之则较远。

(4) k-壳分解方法 K-shell Decomposition Method k-壳分解方法 (k-shell decomposition method): 和度不同, 每一步反复循环, 直到度等于 n 的节点全部被移除。k-壳分解算法将网络划分成从核心到边缘的层次。

(5) 特征向量中心性 Eigenvector Centrality 特征向量中心性的基本思路: 一个节点的重要性既取决于其邻居节点的数量 (度), 也取决于邻居节点的重要性。如果节点 i 的重要性度量为 x_i, 那么有

$$x_i = c\sum_{j=1}^{N} a_{ij}x_j$$

其中 c 为一个比例常数。

(6) PageRank 算法 PageRank 的核心思想：如果一个网页被很多其他网页链接到的话说明这个网页比较重要，也就是 PageRank 值会相对较高。如果一个 PageRank 值很高的网页链接到一个其他的网页，那么被链接到的网页的 PageRank 值会相应地因此而提高。

(7) 节点重要性指标比较 分别取出各个节点重要性指标中最重要的三个节点（见表 1）进行对比，发现不同指标测度下，最重要的三节点并不相同，表明在不同节点重要性指标下，节点有着不同的相对重要性。

表 1: 不同指标测度下最重要的三个节点

Rank	DC	BC	CC	KC	EC	PR
first	5320 (697)	5320 (0.0988)	6615 (0.291)	5320 (46)	7018 (0.651)	6615 (0.00769)
second	6615 (674)	6615 (0.0955)	4028 (0.283)	6615 (46)	7045 (0.587)	6966 (0.00555)
third	2141 (625)	2141 (0.0886)	6872 (0.268)	2344 (46)	7016 (0.330)	4028 (0.00541)

Problem 3: 节点相似度及好友推荐

(1) AA-index 进行好友推荐 社会网络分析中三元闭包（Triadic closure）原则指出, 如果两个人 A 和 B 拥有一个共同的朋友 C, 那么这两个人今后也很有可能成为朋友, 从而使三个节点构成一个闭合的三角形 ABC 。对于一般的网络, 这一原则推广如下: 如果两个节点的共同邻居的数量越多, 这两个节点就越相似, 从而更倾向于互相连接。计算公式如下:

$$s_{xy}^{CN} = |\Gamma(x) \cap \Gamma(y)|$$

Adamic-Adar(AA) 指标是对共同好友进行加权后的指标, 计算公式如下:

$$s_{xy}^{AA} = \sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\log k(z)}$$

由于本网络为有向图，因此将原图转化为无向图后进行计算，推荐结果如下表:

(2) 节点相似度指标设计 基于三元闭包原则，对两个节点的共同好友进行加权后，可以构建不同的节点相似度指标。在现有的对共同邻居进行加权的节点相似指标中，其对应的实际意义可以归纳为两个假设：(1) 两个节点之间的共同邻居越多，则它们的节点相似度越高。(2) 如果一个共同邻居连接的除节点 x 和节点 y 的其它节点数越少，则节点的相似度越高。这等价于：如果一个共同邻居连接的节点数越少，则节点的相似度越高。

表 2: 抽取 24 个结点基于 AA-index 的好友推荐结果

userID	ffID	AA-index	userID	ffID	AA-index
310	390	0.417	741	2851	11.895
780	1642	2.069	918	3168	1.018
1301	5187	4.729	1320	4104	1.570
2264	4220	2.452	2314	2011	1.884
2363	322	4.340	2746	4806	1.796
2763	2258	0.910	2984	690	3.131
3560	6563	3.166	3627	2001	4.488
3716	3740	1.897	4017	68	5.327
4377	1473	2.117	4444	1406	7.884
4560	5814	2.981	4770	2083	1.833
4911	769	0.910	4995	5676	5.442
5056	386	12.757	5103	5676	8.504

如果一个网络中共同邻居的作用受到节点数影响很大，在节点相似度的计算中对共同邻居进行加权时，应令节点数较少的共同邻居的权重很大，而节点数较多的共同邻居的权重非常小，也就是说共同好友数的权重随着其连接节点数的增加趋近于 0 的速度应该很快。

WBC-index 是对共同好友进行加权后的指标，在 AA 指标基础上进行改进，计算公式如下:

$$s_{xy}^{AA} = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{k(z) \log k(z)}$$

其中 $k(z)$ 表示 z 节点 (共同好友) 的好友数。

在共同邻居所连接的好友数大于等于 2 时，由于该函数极限为 0，且一阶导数趋近于 0 的速度非常快，使用这种函数定义的共同好友数的权重随着其连接节点数的增加趋近于 0 的速度也非常快。

(3) 指标验证 随机抽取 30 个用户，并随机删除与之相关的 100 条边。通过 WBC-index 预测得到的好友准确率约为 61%，通过 WBC-index 预测得到的好友准确率约为 73%，效果有所提升，能较好地完成推荐任务。

Problem 4: 社团识别

由于原数据集太大，社团识别时画图太过困难，因此以结点度为概率随机抽样 150 个节点，改成无向图后，以最大连通子图作为新的社交网络 FB-PAGES-GOVERNMENT-SMALL，新社交网络节点数 121，新社交网络边数 777。

(1) 聚类算法——Kmeans 聚类 K 均值聚类是最基础的聚类算法。它的基本思想是，通过迭代寻找 K 个簇 (Cluster) 的一种划分方案，使得聚类结果对应的损失函数最小。损失函数可以定义为各个样本距离所属簇中

表 3: 抽取 24 个结点基于 WBC-index 的好友推荐结果

userID	ffID	WBC-index	userID	ffID	WBC-index
310	390	0.218	741	1046	2.441
780	1642	0.803	918	3168	0.463
1301	5187	1.553	1320	4104	0.791
2264	4220	1.140	2314	2011	0.774
2363	322	1.765	2746	4806	0.553
2763	2258	0.366	2984	690	1.418
3560	6563	1.386	3627	2001	1.610
3716	3740	0.505	4017	68	1.264
4377	3275	0.768	4444	1406	2.553
4560	5814	0.877	4770	2083	0.544
4911	769	0.366	4995	5676	1.866
5056	386	4.232	5103	1120	3.212

心点的误差平方和:

$$J(c,\mu) = \sum_{i=1}^{M} \|x_i - \mu_{c_i}\|^2$$

其中 x_i 代表第 i 个样本，c_i 是 x_i 所属的簇，μ_{c_i} 代表簇对应的中心点，M 是样本总数。

(2) 基于边分裂的社团识别算法——GN 算法 GN 算法：Girvan Newman 算法通过从原始图中逐步去除边缘来检测社区。该算法在每一步中去掉中心性最高的边（比如最短路径边介数，即每条边被多少条最短路径经过）。当图分解成小块时，紧密连接的社区结构就显露出来了。

(3) 基于 Q 函数的贪婪算法 因为 Q 函数可以反应网络社团划分的优劣，通过最大化 Q 函数运用贪婪学习算法可以进行社团划分。初始时，将每个节点视为一个社团，即 n 个节点，n 个社团。之后两两合并社团，计算社团合并所产的 Q 值的变化量，选择使得 Q 值增加最大（或减小最少）的方式进行合并。网络的最优划分就是 Q 函数最大值所对应的划分方式。

(4) LPA 社区发现算法 LPA 社区发现算法 [1] 是一种标签传播社区检测算法，通过图中一个点所在的圈子来判断该点的特征性质，连接紧密的点被划分到同一个社区中。

在一个由点和边组成的图结构中，通过一个点 V 的所有邻居对 V 进行投票，每个邻居把自己社区标签投给 V，票数占多数的标签作为 V 的社区标签（如果最大票数中多个标签持平，则随机选择一个）。使用上述方法对所有的点迭代一轮，则每个点都会被分配到一个社区中（打上了标签）。迭代多轮后、或者在某一轮迭代后社区保持稳定时，连接紧密的点将有共同的社区标签，算法结束。

(5) 社团识别算法对比 以上四种社团识别方法中 GN 算法划分社团数量最少，用时最长；LPA 社区发现算法用时最短，发现社团最多。此外，我们发现 Kmeans 算法划分的社团相对集中，社团之间规模相差较大；LPA

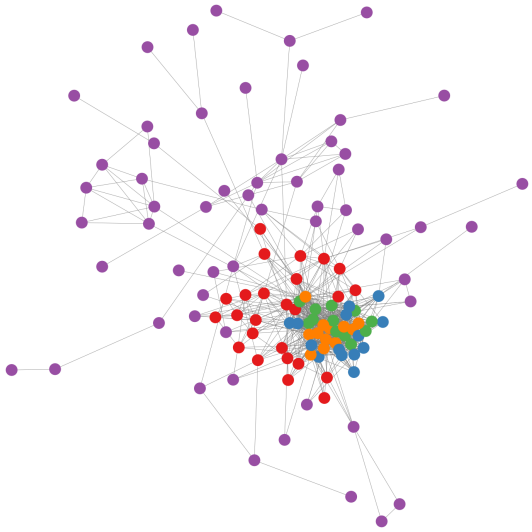

图 4: Kmeans 聚类算法社团识别结果

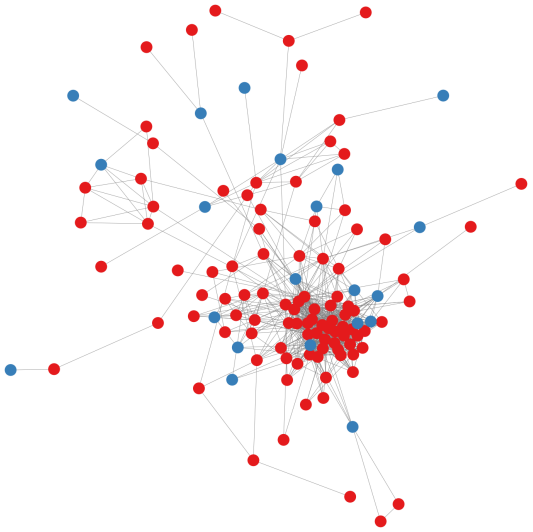

图 5: GN 算法社团识别结果

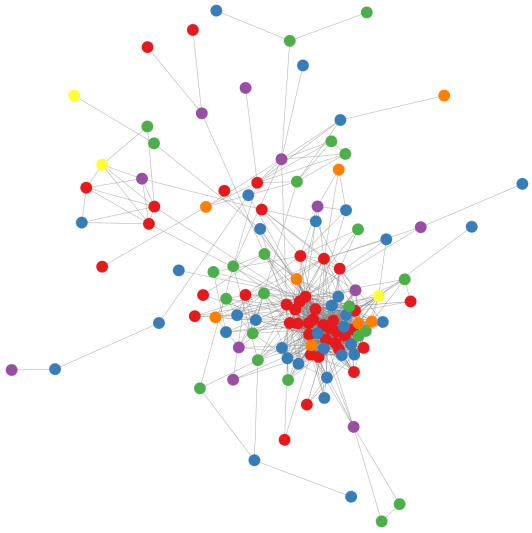

图 6: 基于 Q 函数的贪婪算社团识别结果

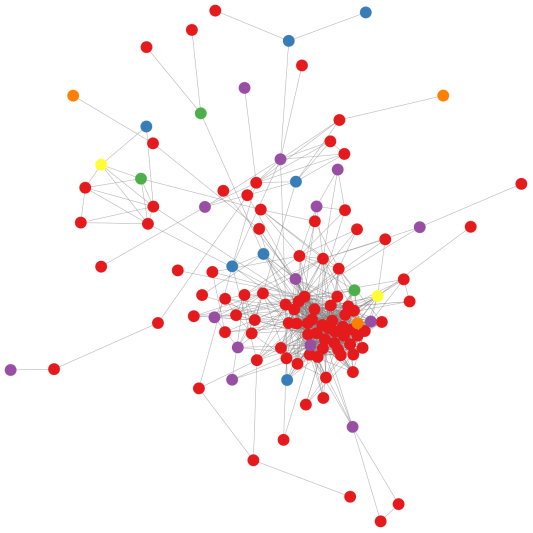

图 7: LPA 社区发现识别算法社团识别结果

社区发现算法识别出的社团规模差距很大，可以发现一个巨大社团和许多零散社团。Kmeans 聚类算法需要提前给出发现的社团数量。

表 4: 社团识别算法对比

	Kmeans	GN 算法	基于 Q 函数的贪婪算法	LPA 社区发现
划分社团数量	5	2	6	6
用时（s）	0.145	0.947	0.099	0.052

(6) 社团识别算法设计——凝聚的层次聚类 Kmeans 聚类对于数据的分层结构识别效果并不理想，因此，采用凝聚的层次聚类方法进行社团识别。初始时将每个样本点当做一个类簇，所以原始类簇的大小等于样本点的个数，然后计算两两类簇之间的距离（后边会做介绍），找到距离最小的两个类簇 c1 和 c2；并合并这两个簇，以此类推直到达到设定的分类数目。发现社团数为 5，用时 0.005s, 用时最短，在大样本数据集上有较好的表现。

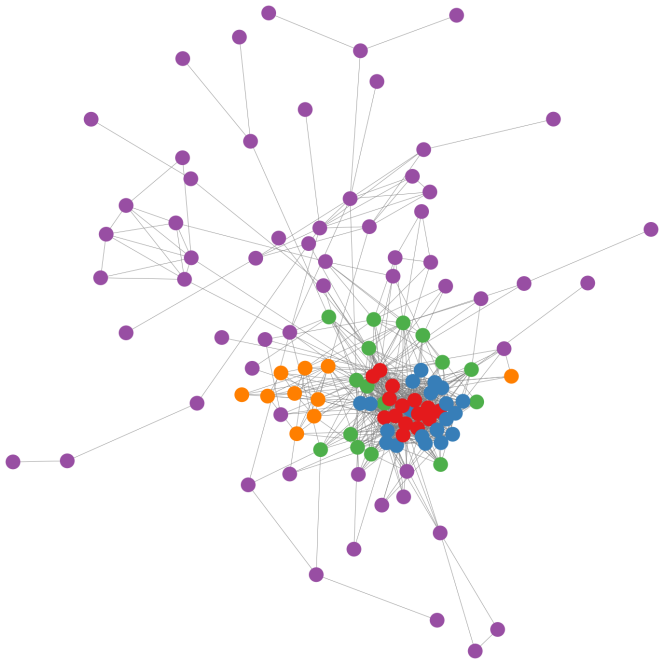

图 8: 凝聚的层次聚类算法社团识别结果

Problem 5: 代码整理

(1) 网络总体特征

社交网络图读取及预处理

```
import networkx as nx
import pandas as pd
import numpy as np
# 读取图
G=nx.DiGraph()
nodes = pd.read_csv("data/fb_nodes.csv")
edges = pd.read_csv("data/fb-pages-government.csv")
G.add_nodes_from(nodes["Id"])
G.add_edges_from(edges.apply(lambda x: tuple(x), axis=1).values.tolist())
# 转化无向图
UG = G.to_undirected()
UG.remove_edges_from(nx.selfloop_edges(UG))
```

社交网络整体指标计算

```
from matplotlib import pyplot as plt
# 计算网络拓扑结构
#判断是否是连通图
print('FB-GOVERN网络图是否是弱连通图:',nx.is_weakly_connected(G))
print('FB-GOVERN网络图是否是强连通图:',nx.is_strongly_connected(G))
#计算密度
print('FB-GOVERN网络图的密度:',nx.density(G))
# 富人俱乐部系数(转化成无向图)
# print('FB-GOVERN网络图的富人俱乐部系数:',nx.rich_club_coefficient(UG))

# 平均路径长度
print('FB-GOVERN网络图的平均路径长度:',nx.average_shortest_path_length(G))
# 直径(无向图,因为图不是强连通,所以有向图的直径为正无穷)
print('FB-GOVERN网络图的直径:',nx.diameter(UG))
# 平均聚类系数
print('FB-GOVERN网络图的平均聚类系数:',nx.average_clustering(G))
# 平均度
degree = pd.DataFrame(G.degree(),columns=["nodes","degree"])
print('FB-GOVERN网络图的平均度:',np.mean(degree["degree"].values)/2)
# 度分布图
degree=nx.degree_histogram(G)
x=range(len(degree))#生成X轴序列,从1到最大度
y=[z/float(sum(degree))for z in degree]#将频次转化为频率,利用列表内涵
plt.figure(figsize=(7,7))
plt.scatter(x,y,s=1,color=(0,0,0))#在双对坐标轴上绘制度分布曲线
plt.xlabel("Degree")
plt.ylabel("Density")
```

```
plt.savefig("degree_distribution.png",dpi = 300)#显示图表

#度同配系数
print('FB-GOVERN网络图的度同配系数:',nx.degree_assortativity_coefficient(G))
```

(2) 节点重要性度量

节点重要性度量

```
G.remove_edges_from(nx.selfloop_edges(G)) #删除自连接边
DC = nx.degree(G) #度中心性
BC = nx.degree_centrality(G) #介数中心性
CC = nx.closeness_centrality(G) #接近中心性
KC = nx.core_number(G) # k核值(k-core)
EC = nx.eigenvector_centrality(G,max_iter = 500) #特征向量中心性
PR = nx.pagerank(G) #PageRank算法
def get_top(dic,num,name): # 通过name算法,获取的dic中value最大的num个(结点,中心度)对
    sorted_dic = sorted(dict(dic).items(), key=lambda x: x[1],reverse=True)[:num]
     ↪ #将dic强制转化为字典格式并按照value由大到小排序, 取出前num个
    data = pd.DataFrame(sorted_dic,columns=['nodes',name]) #转为dataframe格式
    return data.set_index(["nodes"]) #将结点编号作为index
result = pd.concat([get_top(DC,3,"DC"),get_top(BC,3,"BC"),get_top(CC,3,"CC"),
 ↪ get_top(KC,3,"KC"),get_top(EC,3,"EC"),get_top(PR,3,"PR")])
result
```

(3) 节点相似度指标及好友推荐

构建无向图的边列表

```
data = pd.DataFrame(UG.edges,columns = ["Source","Target"])
Uedges = pd.concat([data,pd.DataFrame(data.loc[:,["Target","Source"]].values,columns
 ↪ = ["Source","Target"])],axis=0)
```

AA 指标计算

```
from numpy import random
random.seed(1)
targetUsers = pd.DataFrame(random.choice(nodes["Id"], 30), columns=[
                           "Id"])  # 篇幅限制, 只给30个随机用户推荐好友
# 关联二度好友
df1 = pd.merge(targetUsers, Uedges, left_on=['Id'], right_on=['Source'])
df2 = pd.merge(df1, Uedges, left_on=['Target'], right_on=['Source'])
friendNum = Uedges.groupby(['Source'])['Target'].count().reset_index()
friendNum.rename(columns={'Target': 'friendNum'}, inplace=True)
```

```
df3 = pd.merge(df2, friendNum, left_on='Target_x', right_on='Source')
df3.drop(columns=['Source_x', 'Source_y', 'Source'], inplace=True)
# 将结算结果改名为userID, ffID, CN
df3.rename(columns={'Id': 'userID', 'Target_y': 'ffID',
           'Target_x': 'CN_id', 'friendNum': 'CN_friendNum'}, inplace=True)
# df3包含了二度好友表和共同好友的好友数

# 计算AA_index
df3["AA_index"] = 1/np.log(df3["CN_friendNum"])
df4 = df3.groupby(['userID', 'ffID'])['AA_index'].sum().reset_index()
# 关联好友表, 用于标记二度好友是否为好友,已经是好友的删除, 推荐自己的删除
df6 = pd.merge(df4, Uedges, how='left', left_on=[
               'userID', 'ffID'], right_on=['Source', 'Target'])
df6 = df6.loc[df6['Target'].isnull(), ['userID', 'ffID', "AA_index"]]
df6 = df6[df6['userID'] != df6['ffID']]
#  按照AAindex排序推荐
df6['rn'] = df6.groupby('userID')['AA_index'].rank(
    ascending=False, method='first')
result = df6.loc[df6['rn'] == 1]
result.drop(columns=['rn'], inplace=True)
```

节点相似度指标设计

```
from numpy import random
random.seed(1)
targetUsers = pd.DataFrame(random.choice(nodes["Id"], 30), columns=[
                           "Id"])  # 篇幅限制, 只给30个随机用户推荐好友
# 关联二度好友
df1 = pd.merge(targetUsers, Uedges, left_on=['Id'], right_on=['Source'])
df2 = pd.merge(df1, Uedges, left_on=['Target'], right_on=['Source'])
friendNum = Uedges.groupby(['Source'])['Target'].count().reset_index()
friendNum.rename(columns={'Target': 'friendNum'}, inplace=True)
df3 = pd.merge(df2, friendNum, left_on='Target_x', right_on='Source')
df3.drop(columns=['Source_x', 'Source_y', 'Source'], inplace=True)
# 将结算结果改名为userID, ffID, CN
df3.rename(columns={'Id': 'userID', 'Target_y': 'ffID',
           'Target_x': 'CN_id', 'friendNum': 'CN_friendNum'}, inplace=True)
# df3包含了二度好友表和共同好友的好友数

# 计算WBC_index
df3["WBC_index"] = 1/df3["CN_friendNum"]*np.log(df3["CN_friendNum"])
df4 = df3.groupby(['userID', 'ffID'])['WBC_index'].sum().reset_index()
# 关联好友表, 用于标记二度好友是否为好友,已经是好友的删除, 推荐自己的删除
```

```
df6 = pd.merge(df4, Uedges, how='left', left_on=[
                'userID', 'ffID'], right_on=['Source', 'Target'])
df6 = df6.loc[df6['Target'].isnull(), ['userID', 'ffID', "WBC_index"]]
df6 = df6[df6['userID'] != df6['ffID']]
#  按照WBCindex排序推荐
df6['rn'] = df6.groupby('userID')['WBC_index'].rank(
    ascending=False, method='first')
result = df6.loc[df6['rn'] == 1]
result.drop(columns=['rn'], inplace=True)
```

(4) 社团识别

抽样获得小数据集

```
import networkx as nx
import pandas as pd
import numpy as np
import time

# 抽样获得小数据集
G_small=nx.Graph()
nodes = pd.read_csv("data/fb_nodes.csv")
edges = pd.read_csv("data/fb-pages-government.csv")
df1 = pd.merge(nodes,edges,left_on="Id",right_on="Source")
df2 = df1.groupby(['Source'])['Target'].count().reset_index()
df2 = pd.DataFrame(df2.values,columns=["Id","friendNum"])
df2["rn"] = df2["friendNum"].rank(ascending=False, method='first')
# nodes = df2.loc[df2["rn"]<=100,["Id"]]
nodes = pd.DataFrame(np.random.choice(df2["Id"],150,replace=False,p =
 ↪ (1/df2["rn"])/sum(1/df2["rn"])),columns=["Id"])
edges = pd.merge(nodes,edges,left_on="Id",right_on="Source",how =
 ↪ "inner").loc[:,["Source","Target"]]
edges = pd.merge(nodes,edges,left_on="Id",right_on="Target",how =
 ↪ "inner").loc[:,["Source","Target"]]
G_small.add_nodes_from(nodes["Id"])
G_small.add_edges_from(edges.apply(lambda x: tuple(x), axis=1).values.tolist())
# 删除自连接
G_small.remove_edges_from(nx.selfloop_edges(G_small))
# 找出最大连同子图
largest = max(nx.connected_components(G_small),key=len)
G_small = G_small.subgraph(largest)
print("新社交网络节点数",G_small.number_of_nodes())
print("新社交网络边数",G_small.number_of_edges())
```

社团识别算法

```
from matplotlib import cm # 节点颜色
def draw_plot(G,y,title):
    i = 0
    colors = []
    for node in G.nodes:
        colors.append(cm.Set1(y[i]))
        i+=1
    plt.figure(figsize=(8,8))
    nx.draw(G,pos=nx.spring_layout(G,k = 0.3,seed =1,iterations =
     ↪ 50),node_size=100,node_color=colors,width = 0.3,edge_color =
     ↪ "grey",with_labels=False)
    plt.savefig("{title}.png".format(title = title),dpi = 200)
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
data = nx.adjacency_matrix(G_small).todense()
# Kmeans聚类
start = time.time()
kmeans = KMeans(n_clusters=5, random_state=1).fit(data)
y = kmeans.predict(data)
end = time.time()
draw_plot(G_small,y,"Kmeans")
print("划分的社团数",len(set(y)))
print("Kmeans聚类耗时:%.3f s"%(end-start))

# GN算法
from networkx.algorithms import community
start = time.time()
comp = community.girvan_newman(G_small)# 显示不同划分层级的节点
y = []
for keys,values in dict(enumerate(next(comp))).items():
    for value in values:
        y.append(keys)
end = time.time()
draw_plot(G_small,y,"GN_model")
print("划分的社团数",len(set(y)))
print("GN算法耗时:%.3f s"%(end-start))

# 基于Q函数的贪婪算法
from networkx.algorithms import community
start = time.time()
```

```
comp = community.greedy_modularity_communities(G_small)# 显示不同划分层级的节点
y = []
for keys,values in dict(enumerate(comp)).items():
    for value in values:
        y.append(keys)
end = time.time()
draw_plot(G_small,y,"Q_model")
print("划分的社团数",len(set(y)))
print("基于Q函数的贪婪算法耗时:%.3f s"%(end-start))

# LPA社区发现算法
from networkx.algorithms import community
start = time.time()
comp = community.asyn_lpa_communities(G_small)
y = []
for keys,values in dict(enumerate(comp)).items():
    for value in values:
        y.append(keys)
end = time.time()
draw_plot(G_small,y,"LPA")
print("划分的社团数",len(set(y)))
print("LPA社区发现算法耗时:%.3f s"%(end-start))

from sklearn.cluster import FeatureAgglomeration
# 分层聚类
start = time.time()
agg = FeatureAgglomeration(n_clusters=5).fit(data)
y = agg.labels_
end = time.time()
draw_plot(G_small,y,"Feature")
print("划分的社团数",len(set(y)))
print("分层聚类耗时:%.3f s"%(end-start))
```

References

[1] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

[2] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.