聚类技术——复杂网络社团检测

实验内容

> 复杂网络是描述复杂系统的有力工具，其中每个实体定义成一个节点，实体间的交互关系定义为边。复杂网络社团结构定义为内紧外松的拓扑结构，即一组节点的集合，集合内的节点交互紧密，与外界节点交互松散。复杂网络社团结构检测广泛的应用于信息推荐系统、致癌基因识别、数据挖掘等领域。
>
> 本实验利用两类数据：模拟数据与真实数据。模拟数据有著名复杂网络学者Mark Newmann所提出，该网络包括128个节点，每个节点的度为16，网络包含4个社团结构，每个社团包含32个节点，每个节点与社团内部节点有k1个节点相互链接，与社团外部有k2个节点相互链接 （k1+k2=16）。通过调节参数k2 (k2=1,2,3,4,5,6,7,8)增加社团构建检测难度。http://www-personal.umich.edu/~mejn/
>
> 真实数据集：跆拳道俱乐部数据由34个节点组成，由于管理上的分歧，俱乐部分解成两个社团。

该实验度量网络的节点相似度，使用贪婪算法提取模块，并采用 Cytoscape 工具，可视化聚类结果

分析及设计

- 步骤1：导入网络数据

  实验中有两类数据，模拟数据和真实数据，真实数据的存储格式为 gml 格式，gml 文件中有节点的定义，和连接的边的定义，在 Mark Newmann 的个人主页中，他介绍了python的解析 gml 文件的工具：networkx，使用 networkx 包，可以很方便的解析 gml 文件。

- 步骤2：计算节点相似度
  - 两个节点的相似度与其公共邻居节点数量有关，

    $$S_{ij} = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|}$$

    其中 $|N(i) \cap N(j)|$ 表示 $N(i) \cap N(j)$ 集合中元素的个数。
  - 求两个节点的相似度，关键是要求这两个节点的邻居节点的交集和并集。节点的邻居节点可以使用python的内置数据类型 set 进行表示，交集使用 set 的 intersection 方法，并集使用 set 的 union 方法，两个节点的相似度就是交集和并集的元素数量之比。

- 步骤3：采用贪婪算法提取模块

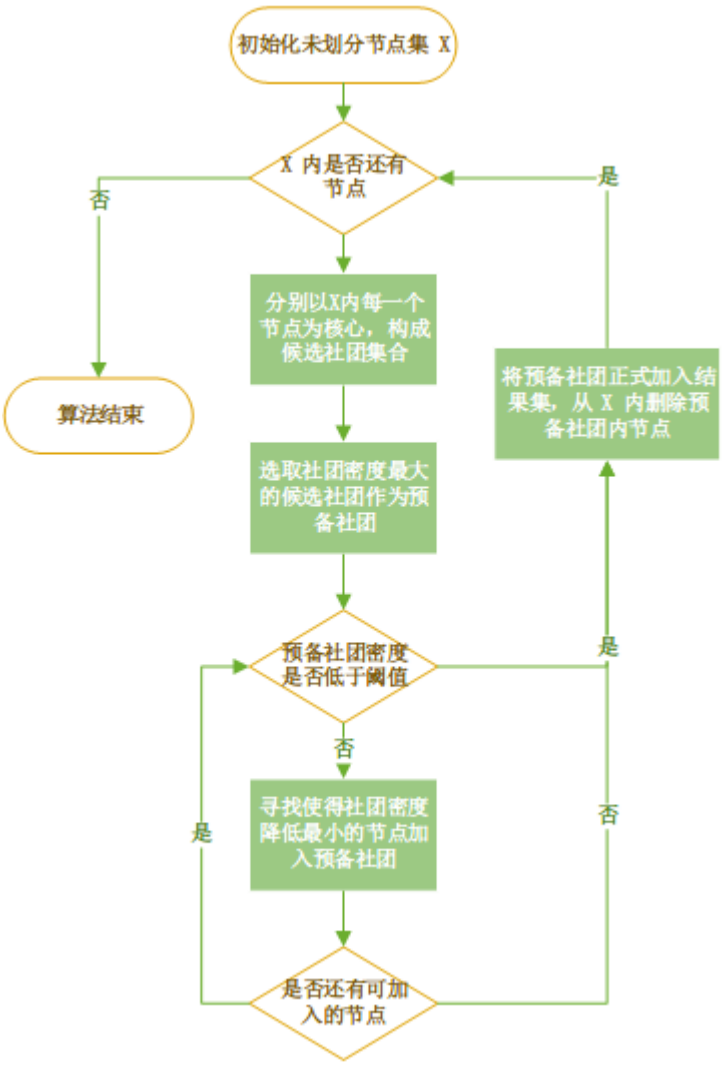
  > 随机选择一个未聚类的节点作为当前社团C，提取出社团C所有未聚类的邻居节点 N(C)。选择使得社团密度降低最小的那个节点v添加到社团C，更新当前社团 为。持续该过程直到当前社团的密度小于某个阈值。

  首先需要定义社团密度的计算公式：

  $$d = \frac{2|E|}{|V|^2}$$

  其中 E 表示社团内的边，$2|E|$ 代表社团内边数，V 表示社团中的点，$|V|^2$ 表示 邻接矩阵行数和列数的乘积

算法流程如图所示：



1. 初始化未划分节点集 X 为所有节点。
2. 判断未划分节点集中是否还有节点，若无节点，输出聚类结果，算法结束
3. 将所有未划分的节点都作为一个社团的核心节点，并将每一个核心节点的邻节点加入该核心节点所在的社团，构成候选社团集合 $c(c_1, c_2, \ldots, c_n)$
4. 根据社团密度公式计算候选社团集合 c 中每个社团的社团密度，选取社团密度最大的一个候选社团 c_i 作为预备社团。并将预备社团中的节点从未划分节点集中删除。
5. 判断预备社团的社团密度是否低于阈值，若低于，将当前预备社团加入聚类结果，转到步骤 2
6. 分别将既不属于候选社团，又处于未划分集合的节点逐个尝试加入当前社团，寻找使得当前社团密度降低最小的节点加入当前社团。
7. 判断当前是否还有既不属于候选社团，又处于未划分集合的节点，若有，转到步骤 5 ，否则，将当前预备社团加入聚类结果，转到步骤 2

- 步骤四（自己添加的额外步骤）：使用不同的随机网络，在不同的社团网络密度阈值下进行聚类，使用 NMI 指标对聚类效果进行评价，绘制不同聚类难度的随机网络的 NMI 曲线。

  NMI (Normalized Mutual Information) 即归一化互信息。

$$NMI(\Omega, C) = \frac{I(\Omega; C)}{(H(\Omega) + H(C)/2)}$$

  其中, I 表示互信息(Mutual Information), H 为熵，当 log 取 2 为底时，单位为 bit，取 e 为底时单位为 nat。

$$I(\Omega;C)=\sum_k\sum_j P(w_k\cap c_j)\log\frac{P(w_k\cap c_j)}{P(w_k)P(c_j)}$$
$$=\sum_k\sum_j\frac{|w_k\cap c_j|}{N}\log\frac{N|w_k\cap c_j|}{|w_k||c_j|}$$

其中, $P(w_k)$, $P(c_j)$, $P(w_k\cap c_j)$ 可以分别看作样本 (document) 属于聚类簇 w_k, 属于类别 c_j, 同时属于两者的概率。第二个等价式子则是由概率的极大似然估计推导而来。

$$H(\Omega)=-\sum_k P(w_k)\log P(w_k)$$
$$=-\sum_k\frac{|w_k|}{N}\log\frac{|w_k|}{N}$$

互信息 $I(\Omega;C)$ 表示给定类簇信息C 的前提条件下,类别信息 Ω 的增加量,或者说其不确定度的减少量。直观地,互信息还可以写出如下形式:

$$I(\Omega;C)=H(\Omega)-H(\Omega|C)$$

- 互信息的最小值为 0, 当类簇相对于类别只是随机的, 也就是说两者独立的情况下, Ω 对于 C 未带来任何有用的信息;
- 如果得到的 Ω 与 C 关系越密切, 那么 $I(\Omega;C)$ 值越大. 如果 Ω 完整重现了 C, 此时互信息最大

$$I(\Omega;C)=H(\Omega)=H(C)$$

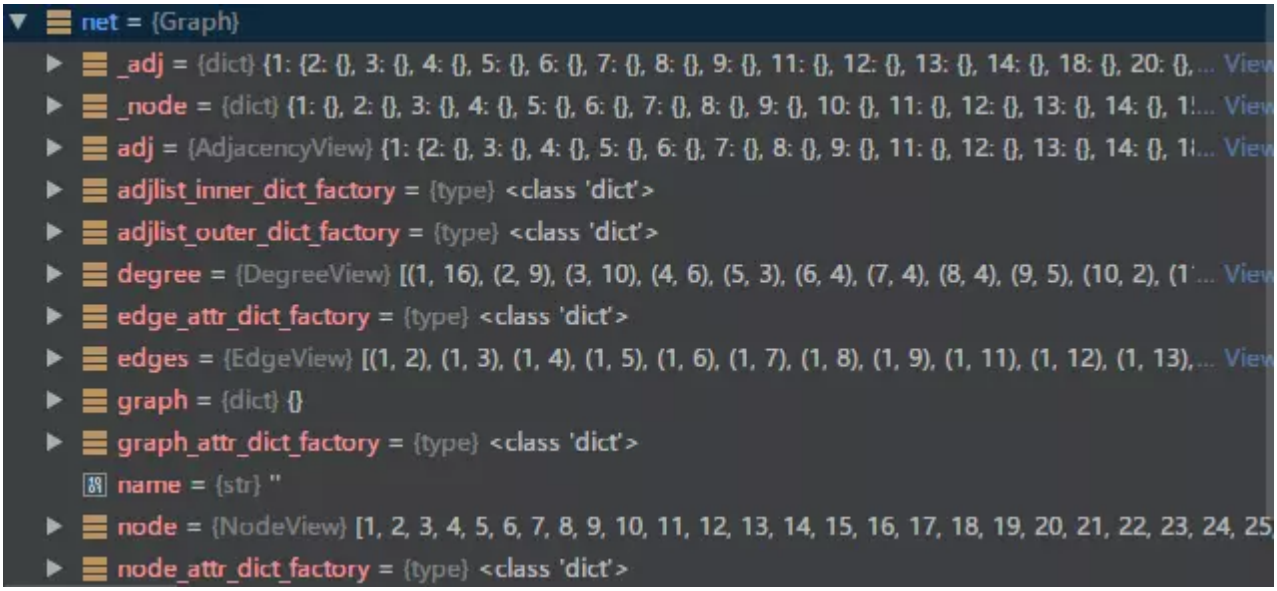
详细实现

- 导入网络数据

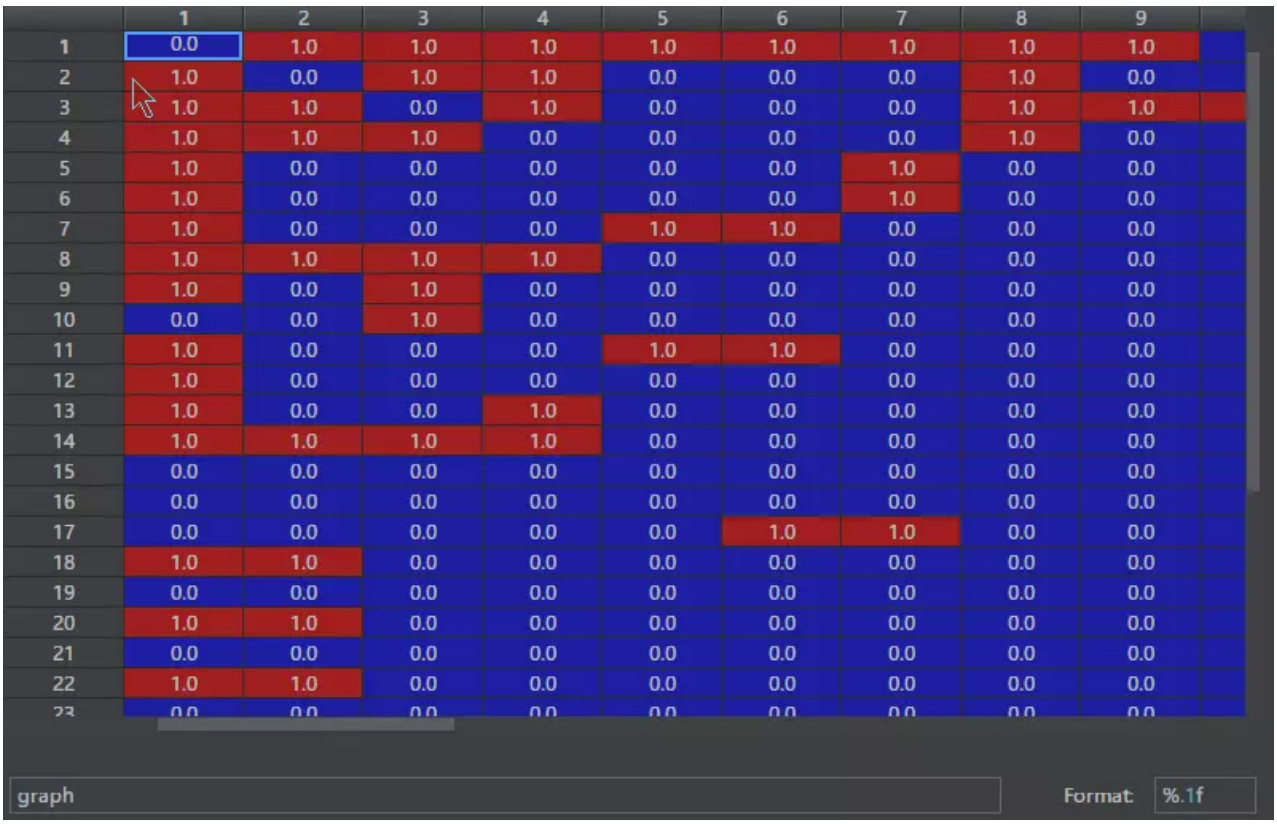
```
net = networkx.read_gml('karate.gml', label='id')  #读取网络社团数据
graph = np.zeros((len(net.node)+1, len(net.node)+1)) #用全 0 初始化邻接矩阵
SimilarityGraph = np.zeros((len(net.node)+1, len(net.node)+1)) #用全 0 初始化相似度矩阵

for e in net.edges:     #对网络中的边进行遍历，邻接矩阵相应位置置1
    graph[e[0]][e[1]] = 1
    graph[e[1]][e[0]] = 1
```

使用 networkx 中的 `read_gml` 函数，来解析 gml 文件，可以得到一个 Graph 类型的对象，保存了所有的节点和边



对网络中的边进行遍历，在对应位置写入邻接矩阵



	1	2	3	4	5	6	7	8	9
1	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0
3	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0
4	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
5	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
6	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
7	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
8	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
9	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
11	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
12	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
14	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
17	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
18	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
21	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
22	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

- 根据网络结构特征给出节点相似性度量指标

  一对节点的相似性定义为如下：

$$S_{ij} = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|}$$

  要对这对节点的邻居节点取交集和并集，因此，定义函数 `getNeighbor`：

```
def getNeighbor(i:int)->list:
    """
    获取节点 i 所有的邻居节点
    :param i: 节点序号
    :return: list 所有邻居节点的序号
    """
    result = []
    for j in range(len(net.node)+1):
        if graph[i][j]:
            result.append(j)
    return result
```

函数以 list 的形式把节点 i 的所有邻居节点给出。

定义函数 `getSimilarity`:

```
def getSimilarity(i:int, j:int)->float:
    """
    计算节点 i 和节点 j 的相似性
    :param i: 节点 i 序号
    :param j: 节点 j 序号
    :return: float 相似性
    """
    Ni = getNeighbor(i)
    Nj = getNeighbor(j)
    intersection = set(Ni).intersection(set(Nj))
    union = set(Ni).union(set(Nj))
    try:
        return len(intersection)/len(union)
    except Exception :
        return 0
```

在函数内将邻居节点由 list 形式转换为 set 形式，可以直接计算其交集并集，通过交集并集的元素个数来计算相似性

调用 `getSimilarity` 函数，可以得到相似性矩阵

```
for i in range(len(net.node)+1):
    for j in range(len(net.node)+1):
        print(getSimilarity(i, j))
        SimilarityGraph[i][j] = getSimilarity(i,j)
        SimilarityGraph[j][i] = SimilarityGraph[i][j]
```

	1	2	3	4	5
1	1.0	0.3888888888888889	0.23809523809523808	0.29411764705882354	0.11764705882352
2	0.3888888888888889	1.0	0.26666666666666666	0.36363636363636365	0.09090909090909
3	0.23809523809523808	0.26666666666666666	1.0	0.3333333333333333	0.08333333333333
4	0.29411764705882354	0.36363636363636365	0.3333333333333333	1.0	0.125
5	0.11764705882352941	0.09090909090909091	0.08333333333333333	0.125	1.0
6	0.1111111111111111	0.08333333333333333	0.07692307692307693	0.1111111111111111	0.75
7	0.1111111111111111	0.08333333333333333	0.07692307692307693	0.1111111111111111	0.16666666666666
8	0.17647058823529413	0.3	0.2727272727272727	0.42857142857142855	0.16666666666666
9	0.05	0.2727272727272727	0.15384615384615385	0.2222222222222222	0.14285714285714
10	0.058823529411764705	0.1	0.0	0.14285714285714285	0.0
11	0.11764705882352941	0.09090909090909091	0.08333333333333333	0.125	0.2
12	0.0	0.1111111111111111	0.1	0.16666666666666666	0.33333333333333
13	0.058823529411764705	0.2222222222222222	0.2	0.14285714285714285	0.25
14	0.16666666666666666	0.2727272727272727	0.25	0.375	0.14285714285714
15	0.0	0.0	0.09090909090909091	0.0	0.0
16	0.0	0.0	0.09090909090909091	0.0	0.0
17	0.125	0.0	0.0	0.0	0.25
18	0.058823529411764705	0.1	0.2	0.3333333333333333	0.25
19	0.0	0.0	0.09090909090909091	0.0	0.0
20	0.05555555555555555	0.09090909090909091	0.18181818181818182	0.2857142857142857	0.2

- **采用社团密度算法进行聚类**

  定义 BDA 算法函数，对给定的社团网络 graph 进行聚类。Graph 为自定义的类，类中有网络的邻接矩阵，并且可以从 Graph 类中直接获取任意节点的邻居节点，还可以直接计算两个节点之间的相似性。该类的定义见源代码，在此不再赘述。同时还自定义了 group 类，该类会自动计算 group 的密度。

```
def BDA(graph: Graph, treshold):
    """
    基于社团密度的社团发现算法
    :param graph: 社团网络
    :param treshold: 聚类社团密度阈值
    :return:
    """

    nodeNum = graph.G.shape[0]  # 获取社团节点个数
    isInGroup = np.zeros(len(graph.G), dtype=bool) #初始化未划分节点集为所有节点
    result = []

    while False in isInGroup: #判断是否有节点还未划分
        grouptestList = []  # 初始化候选社团集合
        for i in range(nodeNum):
            if not isInGroup[i]:
                iandNeighbor = [i]
                for j in graph.getNeighbor(i):
                    if not isInGroup[j]:
                        iandNeighbor.append(j)
                grouptestList.append(group(iandNeighbor, graph))
        grouptestList.sort(key=lambda x: x.density, reverse=True) #根据候选社团的密度进行排序

        groupselect = grouptestList[0]  #选取社团密度最大的社团作为预备社团
        isInGroup[groupselect.node] = True
        while groupselect.density >= treshold: #判断社团密度是否小于阈值
```

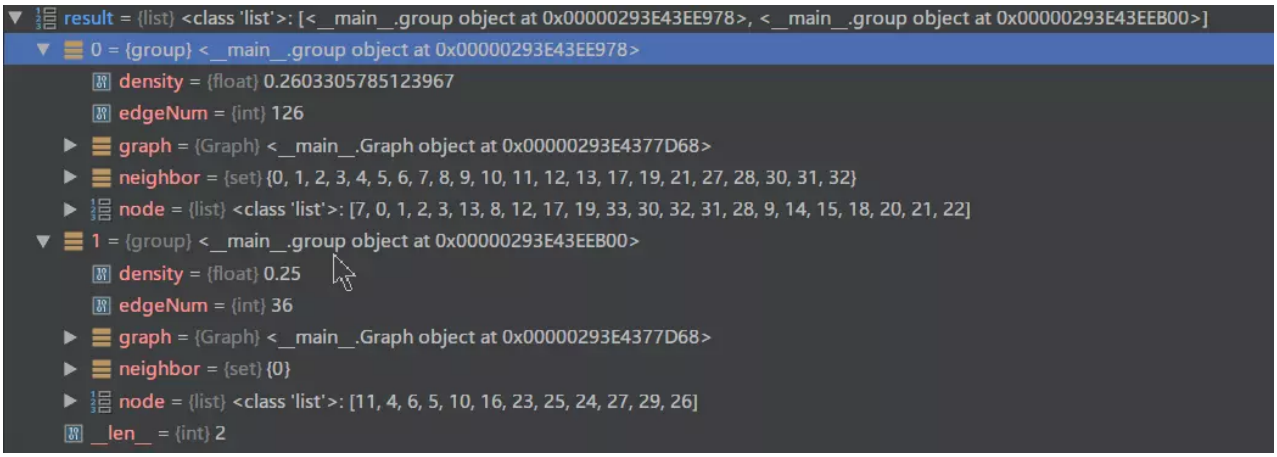
```
            max_density = 0
            max_node = 0
            for i in range(nodeNum):
                if i not in groupselect.node and not isInGroup[i]:
                    #分别将既不属于候选社团，又处于未划分集合的节点逐个尝试加入当前社团
                    trydensity = groupselect.tryAddNode(i)
                    if trydensity > max_density:
                        max_density = trydensity
                        max_node = i
            if max_density != 0:
                #寻找使得当前社团密度降低最小的节点加入当前社团
                groupselect.addNode(max_node)
                isInGroup[max_node] = True
            else:
                break
        result.append(groupselect) #将当前社团确定为正式社团加入结果集
    return result

```

使用该函数时，直接把待聚类的网络作为参数传输函数，并设定社团密度阈值 （0<t<1）

```
rs = BDA(karateGraph, 0.27)
print("网络共被分为 ",len(rs), " 个类")
```

处于同一类的节点 id 会在同一个 group 类中返回，得到聚类结果



实验结果

真实数据实验结果：

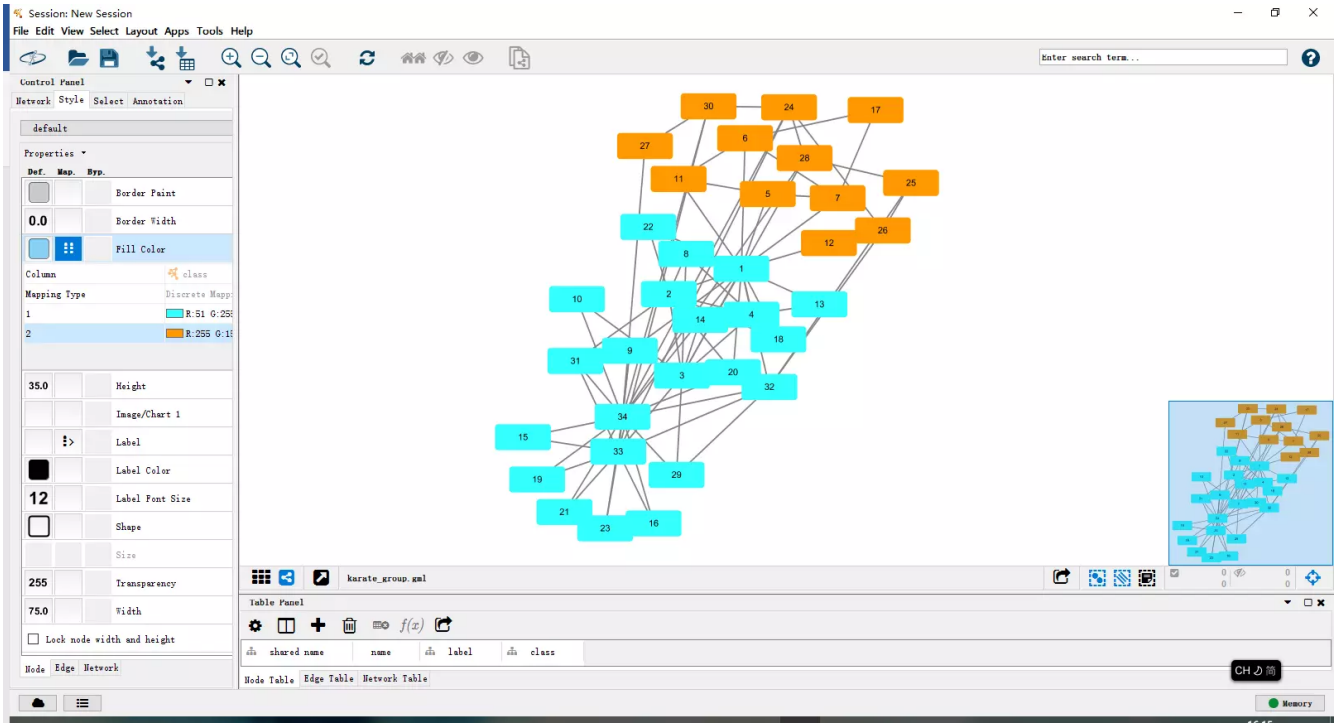
设定社团密度阈值为 0.25，使用社团密度算法进行聚类，得到的聚类结果如下：

可以看到，整个社团被分为两类，

第一类：8 1 2 3 4 14 9 13 18 20 34 31 33 32 29 10 15 16 19 21 22 23

第二类：12 5 7 6 11 17 24 26 25 28 30 27

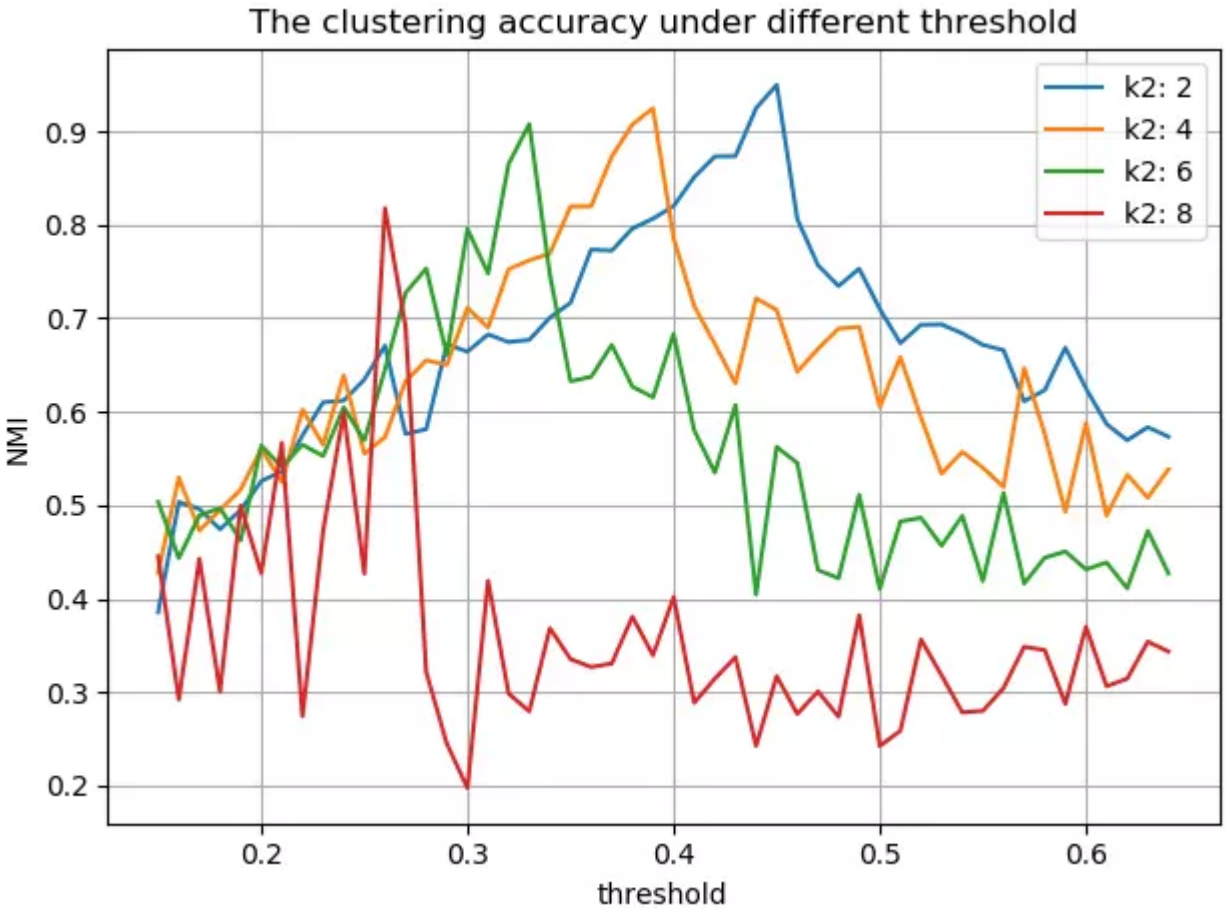
使用Cytoscape 进行绘图并着色，可视化聚类结果：



模拟数据实验结果：

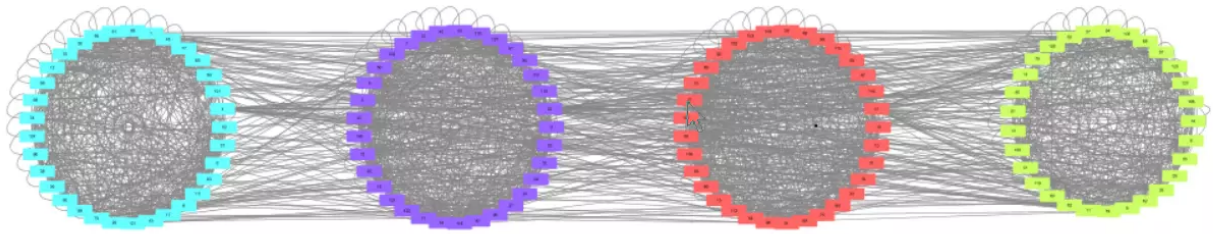
> 该网络包括128个节点，每个节点的度为16，网络包含4个社团结构，每个社团包含32个节点，每个节点与社团内部节点有k1个节点相互链接，与社团外部有k2个节点相互链接 （k1+k2=16）。通过调节参数k2(k2=1,2,3,4,5,6,7,8)增加社团构建检测难度

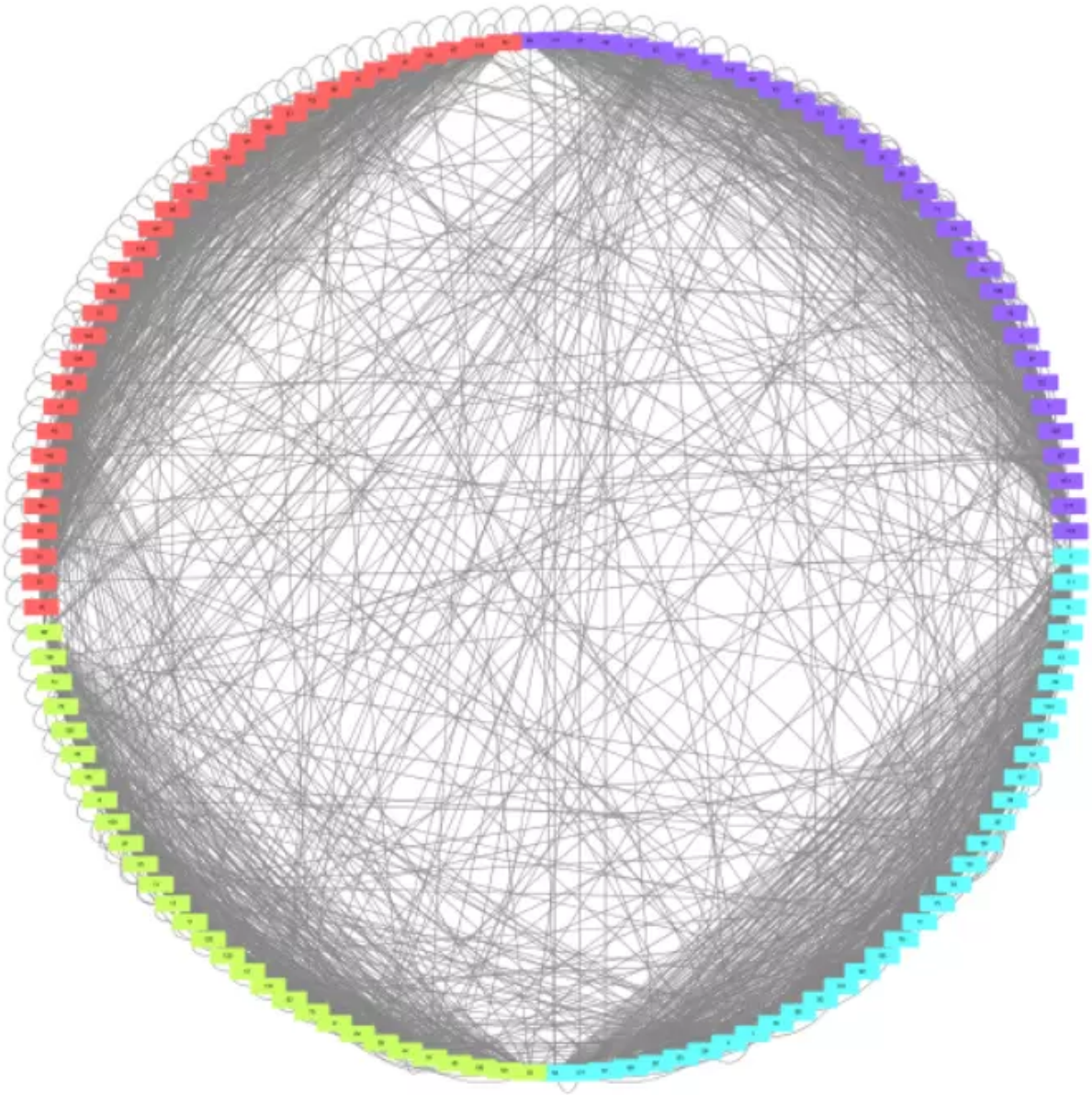
在这里，分别选取 k2 的值为 2、4、6、8，对于每个 k2 值，连续等间隔取 50 个不同的社团密度阈值，进行聚类，使用 NMI 指标对聚类结果进行衡量（越接近 1 聚类结果越好），得到如下结果



从图中可以看出，随着社团外部链接数的增加，NMI 峰值逐渐降低，且峰值时的聚类密度阈值越来越低，聚类难度不断增加

可视化模拟数据的聚类结果：





心得体会

通过本次实验，我设计了基于社团密度的社团发现算法，对给定的真实数据（跆拳道俱乐部人际网络数据）和生成的不同聚类难度的网络数据进行了聚类验证。但我认为基于密度的社团发现算法还有很大的不足。在实验过程中发现，算法的聚类效果受阈值选取的影响非常大，当给定的阈值过大时，网络会被切分为许多的孤立节点。当阈值选取过小时，网络又会产生包含节点数目过多的「超级社团」。并且在大量的模拟数据的测试过程中，基于社团密度的社团发现算法的聚类效果产生了很大的波动，该算法的聚类效果具有较大的随机性。

因此，我认为，该算法还有改进的空间，判断社团是否聚类完毕，还应该引入一些新的评价指标，使得社团在到达一定规模之后能更快停止增长。对社团密度的定义也较为简单，除了社团内部的边数外和社团的节点数外，还应该考虑一些别的因素，避免一些过小社团的产生。