



西安电子科技大学

XIDIAN UNIVERSITY

数据挖掘 课程实验 报告

姓名：赵宇盛 学号：19049200010
班级：1903013 计算机科学与技术专业

任课教师 马小科 教授

2021 年 12 月 6 日

目录

0	简介	1
1	实验一分类技术——二分网络上的链路预测	2
1.1	实验内容	2
1.2	分析及设计	2
1.3	详细实现	3
1.4	实验结果	6
1.5	心得体会	9
2	实验二聚类技术——复杂网络社团检测	11
2.1	实验内容	11
2.2	分析及设计	11
2.3	详细实现	12
2.4	实验结果	16
2.5	心得体会	20
3	附录A 部分实验一代码	20
4	附录B 部分实验二代码	21

0 简介

本实验报告记录了数据挖掘课程的实验任务的完成过程。在老师布置的实验任务中，我分别选择了实验一分类技术——二分网络上的链路预测和实验三聚类技术——复杂网络社团检测。另外，实验代码在Jupyter Notebook上实现，源代码文件以.ipynb结尾，使用编程语言为python。

1 实验一 分类技术—二分网络上的链路预测

1.1 实验内容

基于网络结构的链路预测算法被广泛的应用于信息推荐系统中。算法不考虑用户和产品的内容特征，把它们看成抽象的节点，利用用户对产品的选择关系构建二部图。为用户评估它从未关注过的产品，预测用户潜在的消费倾向。以下我们应用基于网络结构的链路预测算法对ml-1m文件夹中的“用户—电影”打分数据做一个分类任务。

首先我们需要观察分类任务依赖打分数据的数据结构，建立一个用户-电影的关系矩阵，其中实验要求我们需要着重考虑到用户对电影的打分。

然后，我们关注到电影之间存在着相似性，实验要求我们透过我们建立的关系矩阵来抽出电影之间的“距离矩阵”。

以上我们初步建立好了针对用户的电影推荐系统。从而我们可以对给定用户，按照其喜欢程度，对电影进行排名，进行电影推荐。

我们建立的用户-电影的关系矩阵的用户-电影关系随机分为两部分，其中90%归为训练集（train set），10%归为测试集（test set）。将测试集透过训练集得到的用户-电影推荐系统来量化评估实验中建立的电影推荐系统模型的预测精确度。

最后，实验要求画出ROC曲线来度量我们建立的预测方法的准确性。

1.2 分析及设计

读取含有打分数据的`rating.dat`文件，首先了解其数据结构。总共1000209个打分记录，数据量较多。纵向看，属性包括“UserID”、“MovieID”、“Rating”以及“Timestamp”。

	UserID	MovieID	Rating	Timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291
...
1000204	6040	1091	1	956716541
1000205	6040	1094	5	956704887
1000206	6040	562	5	956704746
1000207	6040	1096	4	956715648
1000208	6040	1097	4	956715569
1000209 rows × 4 columns				

图 1: df_ratings打分数据

数据预处理：“Timestamp”时间戳列数据在本实验任务中属于冗余项，故要将其清洗掉。“Rating”

项的值离散分布，取值范围 $\in \{1,2,3,4,5\}$ ；实验任务内容要求筛选出Rating值 > 3 的行元组，据此我们对原数据进行二分类：用户喜欢的（Rating > 3 ）记为1，用户不喜欢的（others）记为0，则用户-电影的关系矩阵 A 为：

$$A_{n \times m} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}, a_{ij} = \begin{cases} 1 & a_{ij} > 3 \\ 0 & \text{others} \end{cases}$$

我们把用户不喜欢的元组也清洗掉。最后我们可以得个用户、电影、好评打分的三元组集合。事实上，在处理后的数据中电影都是“被喜欢的”，Rating列也可以被清洗掉。由于实验要求量化精确度，我们把这数据按9:1随机分成训练集和测试集。以下建模中相关指标的运算依靠训练集数据，测试集只参与量化精确度计算。

受到用户对产品的选择关系中资源配置的启发，一个用户选择过的产品都有向该用户推荐其他产品的能力；我们不如把电影看作是产品，电影之间的相似度意味着不同产品间的资源配额。在此之前我们需要统计出 n 位用户各自评价了几部电影，以及 m 部电影各自被多少位用户评价，把这两个指标分别记为 k^n 和 k^m 。

计算电影之间相似度矩阵 W ，它也可称之为资源配额矩阵，其中元素 w_{ij} 表示电影 j 愿意分配给电影 i 的资源配额，它表示这两部电影之间的相似程度。

$$W_{m \times m} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \dots & \dots & \dots & \dots \\ w_{m1} & w_{m2} & \dots & w_{mm} \end{bmatrix}, w_{ij} = \frac{1}{k_j^m} \sum_{l=1}^n \frac{a_{li} a_{lj}}{k_l^n}$$

对给定用户，按照其喜欢程度，对电影进行排名，进行电影推荐。最后得到结果的资源分配矩阵 F 为

$$F_{n \times m} = (W_{m \times m} A_{n \times m}^T)^T$$

将 n 位用户所有没看过的电影按照矩阵 F 中对应项的得分进行排序，推荐排序靠前的电影给对应用户。

精确度检验：总共 n 位用户中，对给定用户 i ，假设其有 L_i 个产品是未选择的，如果在测试集中用户 i 选择的电影 j ，而电影 j 依据向量被排在第 R_{ij} 位，则计算其相对位置为

$$r_{n \times m} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2m} \\ \dots & \dots & \dots & \dots \\ r_{n1} & r_{n2} & \dots & r_{nm} \end{bmatrix}, r_{ij} = \frac{R_{ij}}{L_i}$$

对矩阵 r 求行平均值得到每位用户的精确度矩阵 r^n ，最后需要得到所有用户 r_n 的平均值 \bar{r} ，当 \bar{r} 越小，表明精确度越高。

正确率计算：以ROC曲线为指标，我们需要计算真阳率（TPR）、假阳率（FPR）以及ROC曲线与水平轴围成的面积AUC指数。通过设置不同阈值，对于每一位用户计算其TPR和FPR，取平均作为该阈值下的点值；对于绘制出的ROC曲线，我们通过调用sklearn库相关接口来直接计算AUC。

1.3 详细实现

在完成该实验的主要步骤前，我们需要进行数据预处理。

1. 读入数据

```

1  ## 定义文件路径和属性组
2  movies_path = './movies.dat'
3  users_path = './users.dat'
4  ratings_path = './ratings.dat'
5  movies_column = ['MovieID', 'Title', 'Genres']
6  users_column = ['UserID', 'Gender', 'Age', 'Occupation', 'Zip-code']
7  ratings_column = ['UserID', 'MovieID', 'Rating', 'Timestamp']
8
9  ## 读取dat 文件
10 def read_dat(filePath, column):
11     f = open(filePath, encoding='unicode_escape')
12     datlist = []
13     for line in f:
14         l = line.strip().split('::')
15         datlist.append(l)
16     return pd.DataFrame(datlist, columns=column)

```

2. 将数据集随机按9: 1比例分成训练集和测试集

```

1  ## 把数据资料分成两部分：训练集90% 测试集10%
2  def splitData(df_data):
3      mask = np.random.rand(len(df_data)) < .9
4      train_data = df_data[mask]
5      test_data = df_data[~mask]
6      return train_data, test_data

```

3. 筛选数据，扔掉冗余项

```

1  ## 用户与好评电影关系(DataFrame)
2  def users_prefer_movies(ratings):
3      ratings['prefer'] = ratings['Rating'] > '3'
4      ratings['prefer'] = ratings['prefer'].astype(int)
5      user_movie = pd.DataFrame(ratings, columns=['UserID', 'MovieID', 'prefer'])
6      user_movie_prefer = user_movie[user_movie['prefer']==True]
7      return user_movie_prefer

```

4. 计算电影、用户总数，统计 k^n 和 k^m 两指标

```

1  ## 用户数量n; 电影数量m
2  n = len(df_users.UserID.unique())
3  m = np.max(np.array(df_movies.MovieID, dtype=np.int64))

```

```

4
5 def k_counts_mat(ratings):
6     mat = np.zeros((n,m))
7     for _,triple in ratings.iterrows():
8         mat[int(triple[0])-1,int(triple[1])-1] = 1
9     return mat
10
11 k_counts = k_counts_mat(df_ratings)
12 user_count = k_counts.sum(axis=1)
13 movie_count = k_counts.sum(axis=0)

```

如上我们得到了只含有好评和只有非好评的用户电影关系组，以及统计出 n 位用户各自评价了几部电影，以及 m 部电影各自被多少位用户评价，这两个指标分别记为 k^n 和 k^m 。在主要实验步骤中，首先对 n 个用户 m 部电影的推荐系统建立一个“0-1邻接矩阵A”，在`rating.dat`数据中当rating大于3分时记为1；我们还需要建立一个非好评的“0-1邻接矩阵dis_A”，当Rating ≤ 3 时记为1。

```

1 ## 建立用户电影矩阵A
2 def create_matA(ratings):
3     mat_A = np.zeros((n,m))
4     dis_mat_A = np.zeros((n,m))
5     for _,triple in ratings.iterrows():
6         if int(triple[2]) > 3:
7             mat_A[int(triple[0])-1,int(triple[1])-1] = 1
8         else:
9             dis_mat_A[int(triple[0])-1,int(triple[1])-1] = 1
10    return dis_mat_A,mat_A

```

然后计算矩阵W，我们把公式中的遍历运算转化为矩阵运算，可以加快运算速度。

```

1 ## 建立矩阵W
2 def create_matW(matA,user_cnt,movies_cnt):
3     temp = (matA / user_cnt.reshape([-1,1])) / movies_cnt
4     return np.dot(matA.T,temp)

```

通过矩阵W计算资源配置矢量矩阵F。

```

1 ## 建立矩阵F
2 def create_matF(matW,matA):
3     temp = np.dot(matW,matA.T).T
4     return temp

```

我们要对给定用户，按照其喜欢程度，对电影进行排名，进行电影推荐。将用户所有没看过的电影按照F中对应项的得分进行排序，推荐排序靠前的电影给该用户。

```

1 ## 经过排序的矩阵F

```

```
2 sort_matF = m - np.argsort(mat_F,axis=1)
```

所得排序矩阵应该是 $n \times m$ 维的，每一行针对每一个用户具有一个推荐电影序列。

至此，我们建立的电影推荐系统模型基本完成。

接下来对分类模型的验证包括两方面的实现：精确度与准确率。

1. 精确度 r 的量化

公式在1.2中给出，按照以下函数得到每位用户的平均 r^n

```
1     ## 创建矩阵r
2 def create_matr(sortF,matA):
3     matL = m - k_users_cnt      #所有电影数 - 该用户已选择电影数
4     R = np.zeros((n,m))
5     R = sortF * matA
6     return np.average(R / matL.reshape([-1,1]))
```

放入测试集得到相对位置差矩阵，然后对所有用户求平均 \bar{r} ,

```
1 ## 每位用户的推荐电影序列；测试集
2 mat_r = create_matr(sort_matF,test_a)
3
4 ## 求所有用户的平均值记为精确度r
5 avg_r = np.average(mat_r)
```

2. 准确率验证：TPR、FPR以及ROC曲线

以FPR为x轴，TPR为y轴，在(0, 1)间分割出许许多多阈值，画出ROC曲线，通过sklearn.auc接口计算出ROC 与水平轴围成的面积AUC。此部分代码放在附录A。

1.4 实验结果

数据预处理结果：

1.数据读取，例子见图1。

2.将数据集随机按9：1比例分成训练集和测试集

```
## 把数据资料分成两部分：训练集90% 测试集10%
def splitData(df_data):
    ...mask = np.random.rand(len(df_data))<.9
    ...train_data = df_data[mask]
    ...test_data = df_data[~mask]
    ...return train_data,test_data

train_ratings,test_ratings = splitData(df_ratings)

train_ratings.shape,test_ratings.shape
((900394, 4), (99815, 4))
```

图 2: 数据随机分割结果

如上图，训练集和测试集分别为 $train_ratings$ 和 $test_ratings$ ，打印两者的shape，比例约为9: 1。
3.筛选数据。如下图，



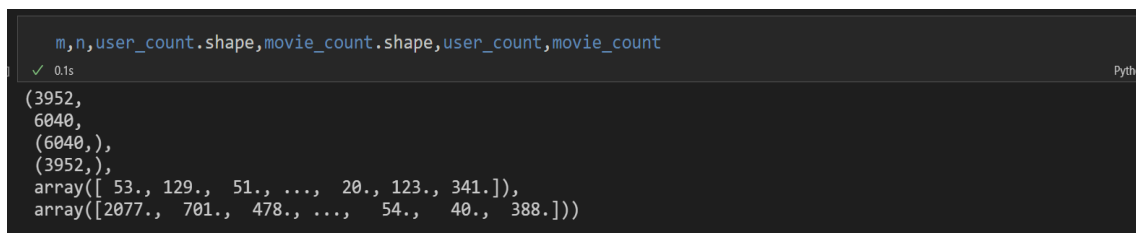
```
users_prefer_movies(train_ratings)
```

UserID	MovieID	prefer
0	1	1193
3	1	3408
4	1	2355
6	1	1287
7	1	2804
...
1000202	6040	1089
1000205	6040	1094
1000206	6040	562
1000207	6040	1096
1000208	6040	1097

518013 rows × 3 columns

图 3: $train_ratings$ 清洗结果

4.用户、电影数量以及统计值 k^n 和 k^m ，截图如下，



```
m,n,user_count.shape,movie_count.shape,user_count,movie_count
```

```
(3952,
 6040,
 (6040,),
 (3952,),
 array([ 53., 129., 51., ..., 20., 123., 341.]),
 array([2077., 701., 478., ..., 54., 40., 388.]))
```

图 4: 统计数据结果

邻接矩阵A



```
dis_train_a,train_a = create_matA(train_ratings)
dis_test_a,test_a = create_matA(test_ratings)
```

```
train_a,train_a.shape,test_a,test_a.shape
```

```
(array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]]),
 (6040, 3952),
 array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]]),
 (3952, 6040))
```

图 5: 训练集和测试集：用户-好评电影邻接矩阵A

资源配置矩阵W


```

mat_W = create_matW(train_a,user_count,movie_count)
✓ 6.1s Python

mat_W[np.isnan(mat_W)] = 0
mat_W,mat_W.shape
✓ 3.4s Python

(array([[5.67646468e-03, 7.27938753e-04, 5.01448535e-04, ...,
5.36094922e-04, 3.78810301e-04, 8.17176994e-04],
[2.45683710e-04, 2.00312838e-03, 7.40925880e-05, ...,
1.17360635e-04, 0.00000000e+00, 1.15537859e-04],
[1.15403177e-04, 5.05224780e-05, 2.03524652e-03, ...,
4.62494121e-05, 0.00000000e+00, 5.07494471e-05],
...,
[1.39379517e-05, 9.04061950e-06, 5.22482899e-06, ...,
2.64074220e-03, 3.41718381e-04, 1.29214217e-04],
[7.29533560e-06, 0.00000000e+00, 0.00000000e+00, ...,
2.53124727e-04, 4.70519232e-03, 1.77869518e-04],
[1.52655115e-04, 6.39496282e-05, 4.11941119e-05, ...,
9.28428077e-04, 1.72533433e-03, 5.32970951e-03]],
(3952, 3952))

```

图 6: 中间量: 资源配置矩阵 W

用户—电影最终资源配置矢量 F 以及各用户推荐电影序列 F'

```

mat_F = create_matF(mat_W,train_a)
✓ 1.4s Python

mat_F[np.isnan(mat_F)] = 0
mat_F,mat_F.shape
✓ 0.2s Python

(array([[5.87459534e-02, 8.40352276e-03, 3.87879099e-03, ...,
7.13884177e-04, 6.70916313e-04, 6.31429436e-03],
[7.58351453e-02, 1.01403037e-02, 5.32209060e-03, ...,
1.34386898e-03, 1.06241469e-03, 1.15457310e-02],
[3.91602532e-02, 7.18254503e-03, 3.01094412e-03, ...,
6.40282257e-04, 2.25952610e-04, 4.05791249e-03],
...,
[1.60522443e-02, 1.70137289e-03, 8.51845464e-04, ...,
2.04810050e-04, 1.01840970e-04, 1.77649245e-03],
[8.27151756e-02, 1.12212010e-02, 4.91039064e-03, ...,
1.52228701e-03, 1.06261405e-03, 1.04209540e-02],
[1.91688909e-01, 2.17506361e-02, 8.23963576e-03, ...,
3.97770903e-03, 4.70378336e-03, 3.57399424e-02]],
(6040, 3952))

```

图 7: 最终资源配置矢量 F

```

## 将用户所有没看过的电影按照f'中对应项的得分进行排序, 推荐排序靠前的电影给用户。
sort_matF = m - np.argsort(mat_F,axis=1)
sort_matF
✓ 2.4s Python

array([[2101, 1699, 2425, ..., 1925, 3693, 1095],
[3311, 393, 3230, ..., 3360, 1925, 1095],
[2376, 3156, 1718, ..., 2755, 2757, 3693],
...,
[2793, 181, 1108, ..., 3693, 2743, 1095],
[2243, 2501, 3374, ..., 3095, 1095, 3693],
[1733, 1720, 1723, ..., 3693, 3345, 1095]], dtype=int64)

```

图 8: 各用户推荐电影序列 F'

由 F' 矩阵首尾一些信息我们大概可以推测出, 在我们构建的针对 n 位用户的电影推荐系统下, $MovieID$ 为 1095、3693、1925等电影是烂片或者不存在(总之是不推荐的), 但是每位用户对好评电影有各自的看法, 这体现出人审美的个性化差异。

分类模型的验证包括两方面的实现：精确度与准确率，结果展示。

1. 精确度 r 的量化

```
mat_r = create_matr(sort_matF, test_a)
✓ 1.8s

avg_r = np.average(mat_r)
avg_r
✓ 0.2s
0.0013145790966581548
```

图 9: 精确度平均值 r

显示 $r \approx 0.0013$ ，由于 r 越小，表明模型精确度越高，此处验证了我所建立的电影推荐模型表现相当不错。

2. 准确率验证：TPR、FPR以及ROC曲线

选择阈值范围为 $[0, 1, 100]$ ，即在 $(0, 1)$ 范围内以0.01为间隔，得到以下图

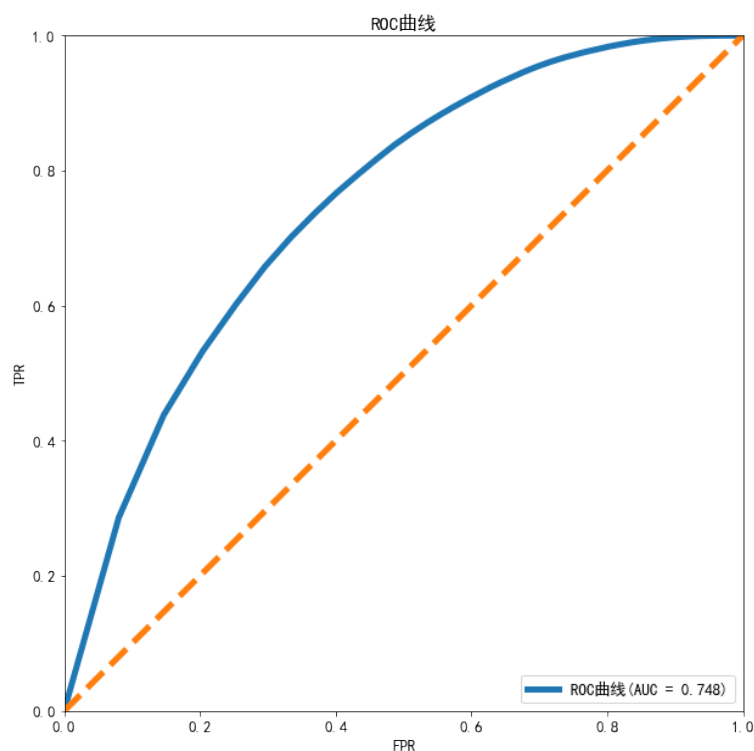


图 10: ROC曲线图

其中面积AUC为0.748，表现不错；如果阈值划分的再细腻些，AUC可以更靠近1。

1.5 心得体会

1. 本实验比较简单，主要在于数据比较干净，没有要求冗繁的数据预处理；而且实验步骤也描述的比较清晰，方便笔者书写程序。

2. 实验中涉及到许多遍历求和等等运算，利用python中numpy库转化为矩阵、向量运算会大大加快运算速度。

3.不足之处在于以上内容基本完全按照实验步骤执行，没有衍生或者改良内容，考虑到实验给了3个数据集，其中`users.dat`以及 `movie.dat`几乎完全没有用到，如果尝试其他方法或者多加改良，我们考虑到用户间的相似度，电影题材的影响等等因素，我们可以构建更为精确、鲁棒的电影推荐系统模型。

2 实验二 聚类技术—复杂网络社团检测

2.1 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。

我们拥有一个真实数据集：由34个节点组成的跆拳道俱乐部数据，由于管理上的分歧，俱乐部分解成两个社团。本实验要求我们在该数据集上做聚类任务，将34个节点分成两个社团。

首先我们用邻接矩阵来存储gml文件中的网络。

然后，我们从网络结构中刻画节点之间的相似关系，给出节点相似性度量指标。

接下来，我们用贪婪算法（Greedy Algorithm）来提取网络结构中的模块。

最后，可视化聚类结果。

2.2 分析及设计

1. 从gml文件导入网络数据

利用邻接矩阵A来存储网络，其中 A_{ij} 表示第i个节点与第j个节点的是否有边相互链接，1表示有，0表示没有。因此，我使用python的networkx库来处理gml文件，

通过调用read_gml()函数和adjacency_matrix().todense()给出该网络数据的邻接矩阵。

2. 根据网络结构特征给出节点相似性度量指标

给定节点 i ，其邻居节点定义为与该节点相链接的所有节点组成的集合，即 $N(i) = \{j | A_{ij} = 1, j = 1, 2, \dots, n\}$ ，给定一对节点 (i, j) ，其相似性定义为这个两个节点的公共邻居节点与邻居节点的并，即：

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

其中 $|N(i) \cap N(j)|$ 表示集合 $N(i) \cap N(j)$ 中元素的个数。

相似度值落在 $[0, 1]$ 之间，节点和自身相似度为1，相似度为0则表明两节点间无公共节点。

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 $C = C \cup v$ 。持续该过程知道当前社团的密度小于某个阈值。

已得知，任务要求用部分贪心来对以上所得网络数据进行二聚类（或者说分类）。对于这一部分，首先我们需要定义如何计算社团密度，我的策略是，从总指标中提取出当前社团的邻接矩阵和相似度矩阵。由于一般认为社团内部的点之间的连接相对稠密，而不同社团的点之间的连接相对稀疏。因此，表达社团密度大小的其中之一指标就是该社团邻接矩阵的0-1成分；

其次，相似度矩阵值表达了两个节点间的距离，所以社团内部相似度越大，表明社团内的关系越稠密。除此之外，社团内节点数量越多，或者说节点度数总和和节点数量之比越小，则说明社团内部稀疏风险越大。因此，我们总结出邻接矩阵的总度数、相似度矩阵以及节点数量作为社团密度计算

(目标函数)的参数。这里使用最简单的形式：根据情况取正反比。

$$Dens(deg, sim, community^{cnt}) = \frac{\overline{sim} \times \sum deg}{community^{cnt}}$$

其中, \overline{sim} 是社团内部相似度的平均值, $\sum deg$ 是社团内部的总度数(或者说是边条数), $community^{cnt}$ 则是社团成员(节点)个数。

在得到社团密度的计算表达后, 然后需要设计函数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 $C = C \cup v$; 不断重复该过程直到低于某个阈值, 聚类结束。

4. 可视化聚类结果

此处采用networkx库绘制出经聚类结果的可视化网络图。

2.3 详细实现

1. 读取数据

通过调用networkx库的read_gml()函数和adjacency_matrix().todense()给出该网络数据的邻接矩阵。

```

1  ## 读取gml 文件; 画出网络结构图; 返回一个邻接矩阵
2  import networkx as nx
3  import numpy as np
4  data = 'karate.gml'
5  def read_gml(filePath):
6      G = nx.read_gml(filePath, label='id')
7      # nx.draw(G, with_labels=True)
8      # plt.show()
9      return np.array(nx.adjacency_matrix(G).todense())
10 G = nx.read_gml(data, label='id')
11 mat_A = np.array(nx.adjacency_matrix(G).todense())

```

由此, 我们得到了网络图G和该图的邻接矩阵mat_A。

2. 定义交、并, 返回节点个数

```

1  ## 节点间交运算
2  def set_cap(li, lj):
3      l = len(li)
4      cnt = 0
5      for i in range(0, l):
6          if li[i]==1 and lj[i]==1: cnt+=1
7      return cnt
8
9  ## 节点间并运算
10 def set_cup(li, lj):

```

```

11     l = len(li)
12     cnt = 0
13     for i in range(0,l):
14         if li[i]==1 or lj[i]==1:cnt+=1
15     return cnt

```

3. 相似度计算，具体描述见上一节。

该函数设计应用上面节点交并运算函数。

```

1  ## 节点之间的相似度度量
2  def Similarity(mat_A):
3      m,n = mat_A.shape
4      sim = np.zeros((m,n))
5      for i in range(0,m):
6          for j in range(i,n):
7              sim[i,j] = set_cap(mat_A[i],mat_A[j]) /
8                  set_cup(mat_A[i],mat_A[j])
9              if i==j:sim[i,j] = 0
10     return sim

```

算法复杂度 $O(n^2)$ ，相似度矩阵值等于节点之间的公交节点和共并节点之比。节点和自身相似度为1，相似度为0则表明两节点间无公交节点。

4. 贪婪算法提取模块

(a) 计算社团密度

```

1  ## 矩阵提取函数
2  def com_mat(mat,community):
3      temp = mat[community]
4      return temp[:,community]
5
6  ## 计算社团内部的密度
7  def cal_density(mat_A,sim,community):
8      ## 从小到大排序
9      community.sort()
10
11     ## 矩阵提取sim,a
12     com_sim = com_mat(sim,community)
13     com_a = com_mat(mat_A,community)
14
15     if len(community)==1:return 1
16     if np.sum(com_a)==0:return 0

```

```

17         else:
18             return np.average(com_sim) * np.sum(com_a)
19                 / len(community)

```

按照第二节所述计算社团密度。

(b) 在未加入社团当中找到使社团密度降低最少的节点

```

1  ## 找到社团密度降低最小的那个节点 resNode节点: density计算好的密度:
2  def findBestdensity(community, nextNode, mat_A, sim, isIncommunity):
3      density_list = np.zeros(mat_A[nextNode].shape[0])
4      for node in range(0, mat_A[nextNode].shape[0]):
5          copyCom = community.copy()
6          if node == nextNode: continue
7          if mat_A[nextNode][node] == 1 and isIncommunity[node] == 0:
8              copyCom.append(node)
9              density_list[node] = cal_density(mat_A, sim, copyCom)
10             copyCom.clear()
11         #按密度大小排序, 返回节点索引序列, 取最大值#
12         resNode = np.argsort(density_list)[::-1][0]
13         density = density_list[resNode]
14         return resNode, density

```

遍历社团外的节点, 调用定义的cal_density()来计算每个节点加入社团后的社团密度, 从大到小排序, 取到排序第一个, 函数返回。

(c) 用贪婪/贪心算法提取模块

局部贪心做聚类任务, 需要取到一个合适的阈值threshold, 在社团密度还未低于阈值时, 调用上面定义的findBestdensity()函数, 把函数返回的节点加入到社团当中, 同时社团密度更新。向量isIncommunity(初始化为0)用来标记社团成员和非社团成员。

最终结果返回选择出来的社团community和此时的社团密度。

```

1  def greedy_cluster(mat_A, sim, threshold, begin_node):
2      ## 社团
3      density = 1
4      community = []
5      isIncommunity = np.zeros(mat_A.shape[0])
6      ## 初始节点
7      nextNode = begin_node
8      # print(nextNode)
9      community.append(nextNode)
10     isIncommunity[nextNode] = 1
11
12     while density > threshold:

```

```

13         nextNode, density = findBestdensity(community, nextNode, mat_A,
14             sim, isIncommunity)
15         if density < threshold: break
16         # print(nextNode, density)
17         community.append(nextNode)
18         isIncommunity[nextNode] = 1
19     return community, density

```

如上，起始节点`begin_node`被这样定义：

```

1 begin_node = np.random.randint(mat_A.shape[0])

```

保证起始节点是[0,33]的随机正整数（因为邻接矩阵从0开始记）

5. 可视化聚类结果

由于任务要求聚类完成分为两个社团，以下函数`create_com()`返回聚类后的两社团。

```

1 def create_com(res_cluster):
2     com1 = list((np.array(res_cluster) + 1))
3     com2 = [i for i in G.nodes if i not in com1]
4     return com1, com2
5
6 def draw_res(com1, com2):
7     all_nodes = com1 + com2
8     elist = [e for e in G.edges]
9     G3 = nx.Graph()
10    for n in all_nodes:
11        G3.add_node(n)
12    for from_loc, to_loc in elist:
13        G3.add_edge(from_loc, to_loc)
14
15    pos = nx.spiral_layout(G3)
16    nx.draw(G3, pos, edge_color='k', with_labels=True,
17        font_weight='light', node_size= 280, width= 0.9)
18
19    nx.draw_networkx_nodes(G3, pos, nodelist=com1, node_color='g', alpha
20        =0.7)
21    nx.draw_networkx_nodes(G3, pos, nodelist=com2, node_color='r', alpha
22        =0.7)
23    plt.show()

```

用`networkx`库的`draw()`函数来可视化聚类结果。

2.4 实验结果

1. 读取数据将初始读到的网络数据可视化，如下图

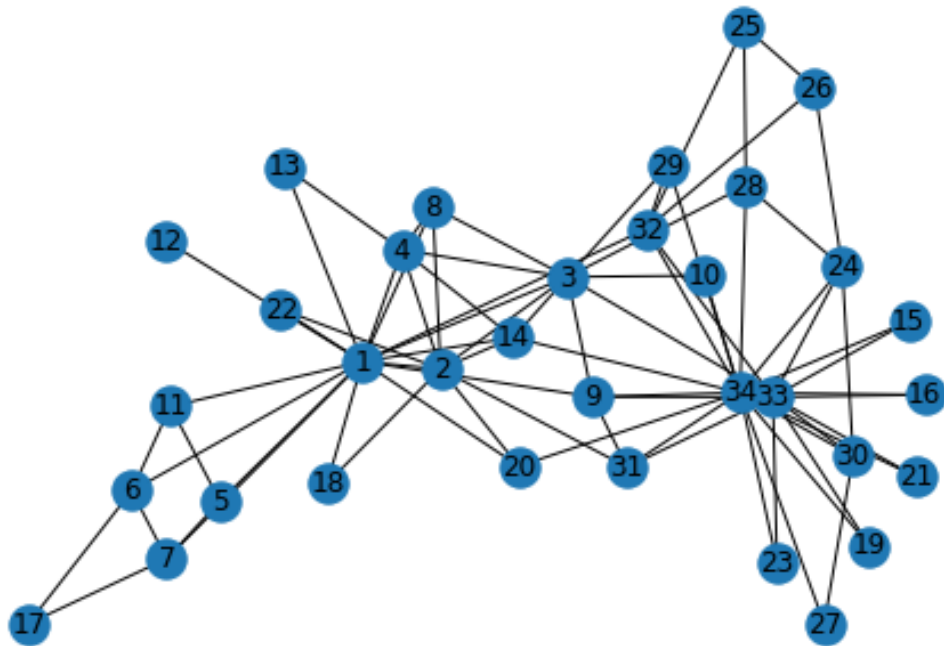


图 11: 网络数据可视化

由于节点数量比较少，节点关系似乎还是比较明朗。我得到如下的邻接矩阵A:

```

import numpy as np
mat_A = np.array(nx.adjacency_matrix(G).todense())
mat_A, mat_A.shape
[184] ✓ 0.3s Python
... (array([[0, 1, 1, ..., 1, 0, 0],
          [1, 0, 1, ..., 0, 0, 0],
          [1, 1, 0, ..., 0, 1, 0],
          ...,
          [1, 0, 0, ..., 0, 1, 1],
          [0, 0, 1, ..., 1, 0, 1],
          [0, 0, 0, ..., 1, 1, 0]], dtype=int32),
      (34, 34))

```

图 12: 网络数据邻接矩阵

2. 定义交、并；进行相似度计算相似度矩阵结果如下：

```

def Similarity(mat_A):
    m,n = mat_A.shape
    sim = np.zeros((m,n))
    for i in range(0,m):
        for j in range(i,n):
            sim[i,j] = set_cap(mat_A[i],mat_A[j])/set_cup(mat_A[i],mat_A[j])
            if i==j:sim[i,j] = 1
    return sim
sim = np.zeros(mat_A.shape)
sim = Similarity(mat_A)
sim,sim.shape

[187] ✓ 0.4s Python
... (array([[1.          , 0.38888889, 0.23809524, ..., 0.          , 0.12          ,
            0.13793103],
            [0.          , 1.          , 0.26666667, ..., 0.07142857, 0.10526316,
            0.13043478],
            [0.          , 0.          , 1.          , ..., 0.23076923, 0.04761905,
            0.28571429],
            ...,
            [0.          , 0.          , 0.          , ..., 1.          , 0.05882353,
            0.0952381 ],
            [0.          , 0.          , 0.          , ..., 0.          , 1.          ,
            0.52631579],
            [0.          , 0.          , 0.          , ..., 0.          , 0.          ,
            1.          ],
            [1.          , 1.          , 1.          , ..., 1.          , 1.          ,
            1.          ]]),
(34, 34))

```

图 13: 网络数据邻接矩阵

3. 贪婪算法提取模块

由于节点数量只有34个，所以我在阈值threshold=0.5情形下，得到了每个节点（从0开始计数）作为初始节点的得到的社团（聚类）结果。

```

1      0 [0, 1, 3, 7, 2, 13, 33, 19]
2      1 [1, 0, 3, 7, 2, 13, 33, 19]
3      2 [2, 3, 7, 1, 0, 13, 33, 19]
4      3 [3, 7, 1, 2, 0, 13, 33, 19]
5      4 [4, 10, 0, 5, 6, 16]
6      5 [5, 6, 16]
7      6 [6, 5, 16]
8      7 [7, 3, 1, 2, 0, 13, 33, 19]
9      8 [8, 30, 32, 33, 22]
10     9 [9, 33]
11    10 [10, 4, 0, 5, 6, 16]
12    11 [11, 0]
13    12 [12, 3, 0, 7, 1, 2, 13, 33, 19]
14    13 [13, 3, 1, 2, 0, 7]
15    14 [14, 32, 33, 22]
16    15 [15, 32, 33, 22]
17    16 [16, 5, 6, 0, 4, 10]
18    17 [17, 1, 0, 21]
19    18 [18, 32, 33, 22]
20    19 [19, 1, 0, 13, 3, 2, 7]
21    20 [20, 32, 33, 22]

```

```

22      21 [21, 1, 0, 17]
23      22 [22, 32, 33, 15]
24      23 [23, 29, 33, 32, 14]
25      24 [24, 25, 31, 28, 33, 27, 23, 32, 29, 26]
26      25 [25, 24, 31, 28, 33, 27, 23, 32, 29, 26]
27      26 [26, 29, 33, 23, 32, 14]
28      27 [27, 23, 33, 32, 29, 26]
29      28 [28, 31, 33, 32, 8, 30, 1, 2, 0, 13, 3, 7]
30      29 [29, 23, 33, 32, 14]
31      30 [30, 8, 32, 33, 22]
32      31 [31, 28, 33, 32, 8, 30, 1, 2, 0, 13, 3, 7]
33      32 [32, 33, 8, 30, 1, 2, 13, 0, 3, 7]
34      33 [33, 32, 8, 30, 1, 2, 13, 0, 3, 7]

```

截图如下：

```

for i in range(0,34):
    res_cluster,density = greedy_cluster(mat_A,sim,0.5,i)
    print("节点:",i,res_cluster)

```

```

节点: 0 [0, 1, 3, 7, 2, 13, 33, 19]
节点: 1 [1, 0, 3, 7, 2, 13, 33, 19]
节点: 2 [2, 3, 7, 1, 0, 13, 33, 19]
节点: 3 [3, 7, 1, 2, 0, 13, 33, 19]
节点: 4 [4, 10, 0, 5, 6, 16]
节点: 5 [5, 6, 16]
节点: 6 [6, 5, 16]
节点: 7 [7, 3, 1, 2, 0, 13, 33, 19]
节点: 8 [8, 30, 32, 33, 22]
节点: 9 [9, 33]
节点: 10 [10, 4, 0, 5, 6, 16]
节点: 11 [11, 0]
节点: 12 [12, 3, 0, 7, 1, 2, 13, 33, 19]
节点: 13 [13, 3, 1, 2, 0, 7]
节点: 14 [14, 32, 33, 22]
节点: 15 [15, 32, 33, 22]
节点: 16 [16, 5, 6, 0, 4, 10]
节点: 17 [17, 1, 0, 21]
节点: 18 [18, 32, 33, 22]
节点: 19 [19, 1, 0, 13, 3, 2, 7]
节点: 20 [20, 32, 33, 22]
节点: 21 [21, 1, 0, 17]
节点: 22 [22, 32, 33, 15]
节点: 23 [23, 29, 33, 32, 14]
节点: 24 [24, 25, 31, 28, 33, 27, 23, 32, 29, 26]

```

图 14: 各个节点作为初始节点聚类结果

显然看到，例如索引0, 1, 2, 3（对应节点为1, 2, 3, 4）在设计聚类算法下肯定属于同一社团（他们的结果甚至一模一样！）

4. 可视化聚类结果

当随机初始节点为26时，得到以下聚类结果：

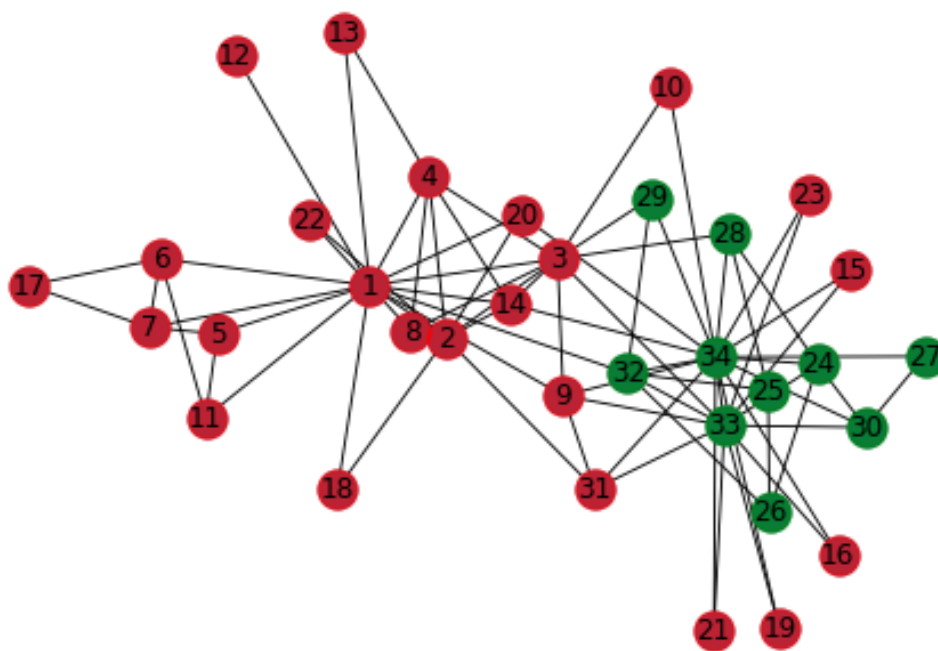


图 15: 始节点为26时聚类结果

当随机初始节点为11时，得到以下聚类结果：

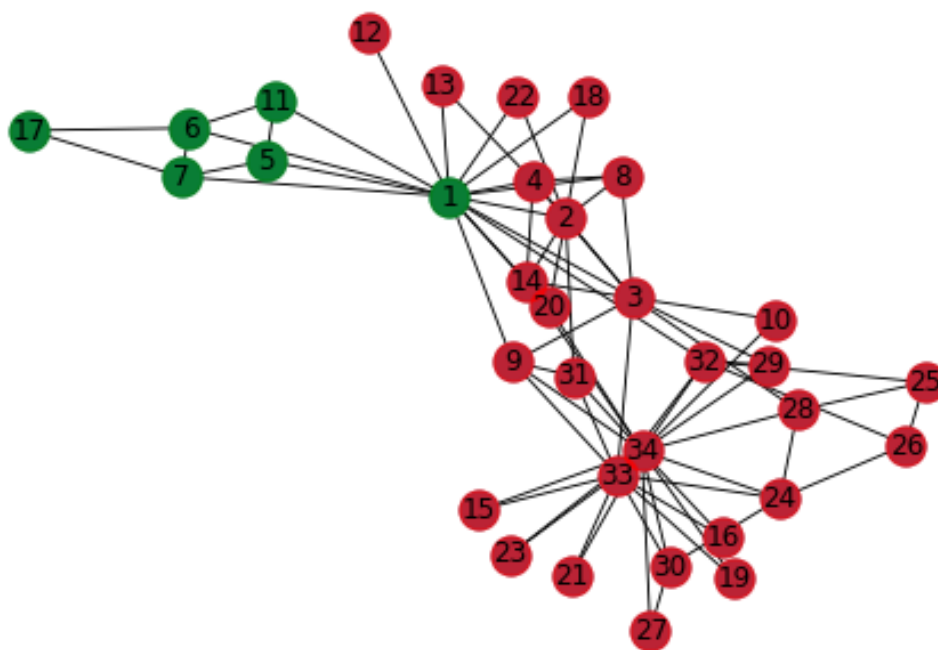


图 16: 始节点为11时聚类结果

由于篇幅有限，不再呈现其他聚类结果图，所有节点聚类后社团如上所示。事实上，有些节点聚类效果不佳，有些节点有很明显的聚类效应（比如11和17聚类结果是一样的）如果说能把所有聚类结

果进行统计分析，应该可以得到更合理的二聚类结果。

2.5 心得体会

- 事实上，实验效果还可以，但未到达预期，我总结了两点主要原因：
 - 一方面该任务仅要求将34个节点划分为两个社团，实际上根据网络数据，划分为三至四个社团，效果更显著。
 - 另一方面，该任务要求使用贪心算法划分社团，局部贪心有很大的局限性。
 - 除此之外，社团密度（也可以称之为模块度量）在本任务中要求学生自主设计，该目标函数的设计极大的影响了最终结果。我的完成使用了可以说是最直观最简单的一种目标函数，结果差强人意。
- 在完成该实验的进度中，正如上所说，社团密度的计算方法（目标函数）的设计是最为困难的，而我几乎没有什么独到的进展，我领会到数学知识在数据挖掘领域是多么重要。

3 附录A 部分实验一代码

ROC曲线的实现

```

1 def roc_pic(f_mat, user_count, mat_rat, mat_dislike, num):
2     threshold_rate = np.linspace(0, 1, num)
3
4     sort_result = np.argsort(-f_mat, axis=1)
5     th_fprs = np.zeros(num)
6     th_tprs = np.zeros(num)
7     for i, threshold in enumerate(threshold_rate):
8         recommond_num = int(mat_rat.shape[1] * threshold)
9         fprs = np.zeros(user_count.shape[0])
10        tprs = np.zeros(user_count.shape[0])
11        for user in range(user_count.shape[0]):
12            recommond_movie = sort_result[user, 0:recommond_num]
13            user_like = np.where(mat_rat[user, :] == 1)[0]
14            user_dislike = np.where(mat_dislike[user, :] == 1)[0]
15            like = np.intersect1d(recommond_movie, user_like)
16            dis_like = np.intersect1d(recommond_movie, user_dislike)
17            if len(user_dislike) == 0: fprs[user] = 0
18            else: fprs[user] = len(dis_like) / len(user_dislike)
19            if len(user_like) == 0: tprs[user] = 0
20            else: tprs[user] = len(like) / len(user_like)
21        th_fprs[i] = fprs.mean()
22        th_tprs[i] = tprs.mean()
23    roc_auc = auc(th_fprs, th_tprs)

```

```
24     lw = 5
25     plt.rc('font', family='SimHei', size=13)
26     plt.figure(figsize=(10, 10))
27     plt.plot(th_fprs, th_tprs, lw=lw, label='曲线ROC(AUC=_%0.3f)' % roc_auc)
28     plt.plot([0, 1], [0, 1], lw=lw, linestyle='—')
29     plt.xlim([0.0, 1.0])
30     plt.ylim([0.0, 1.0])
31     plt.xlabel('FPR')
32     plt.ylabel('TPR')
33     plt.title('曲线ROC')
34     plt.legend(loc="lower_right")
35     plt.show()
```

4 附录B 部分实验二代码

聚类结果绘图代码

```
1  ## 随机初始节点
2  begin_node = np.random.randint(mat_A.shape[0])
3
4  ## 邻接矩阵, 关系度矩阵, 阈值, 随机起始节点
5  res_cluster, density = greedy_cluster(mat_A, sim, 0.5, begin_node)
6
7  ## 两个社团: 聚类结果
8  com1, com2 = create_com(res_cluster)
9
10 ## 绘图
11 draw_res(com1, com2)
12 print("起始节点:", begin_node+1)
```