

# 豆瓣电影 Top250 影评人共现图的制作与分析

## 1 引言

图数据集是一种以节点和边的形式呈现对象之间复杂关系的结构化数据，广泛应用于社交网络、推荐系统、金融分析、生物信息等领域。在社交网络中，可以通过图数据分析用户关系网络，识别意见领袖并发现社交群体；在推荐系统中，基于用户行为构建的图可以提升推荐的精准度；在生物领域，通过分析基因或蛋白质相互作用图，可以揭示重要的生物机制。因此，图数据集的分析能够为现实生活中的复杂问题提供有效的解决方案，是数据挖掘与应用的重要研究方向。

本项目通过爬取豆瓣电影 Top250 的所有影评人，对数据进行清洗后构建影评人共现图，对影评人之间的关系网络进行分析，旨在：

挖掘意见领袖：通过图分析方法（如加权度数中心性、加权介数中心性等），找出网络中具有重要影响力的影评人。

发现社区结构：利用社区发现算法（如 Louvain 算法），探索影评人群体中的兴趣分布及其关联特征。

## 2 数据集描述

### 2.1 数据爬取

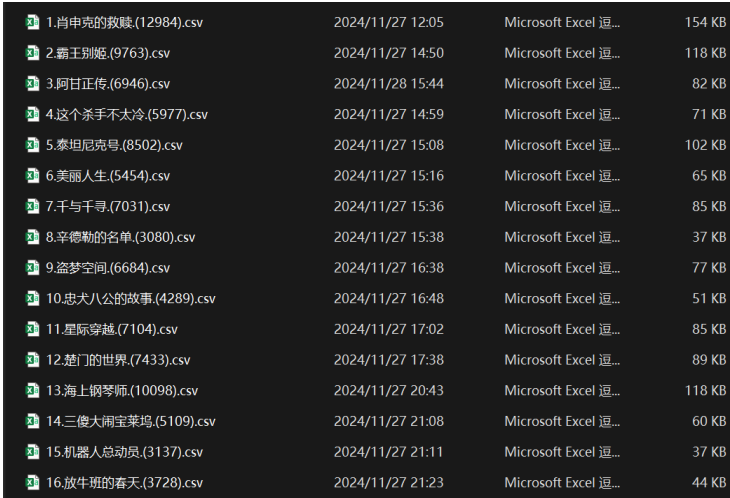
豆瓣是中国知名的社交媒体与内容平台，主要涵盖图书、电影、音乐等领域的评分和评论功能。豆瓣电影 Top250 是用户根据评分和评价筛选出的经典电影榜单，反映了观众的偏好和热度。通过分析影评人评论关系网络，可以更深入地理解影评人与电影的互动模式。

本项目的数据来源于豆瓣电影 Top250 主页面的各个分页面，为了获取影评人之间的共现关系，需从豆瓣的网页内容中爬取电影的影评人数据。数据爬取使用 Python 的 requests 库和 lxml 解析库，同时结合 BeautifulSoup 提取所需信息。为了实现对多页内容的处理，还采用了动态请求和分页控制。

考虑到豆瓣对爬虫的限制，代码中设计了以下功能：

1. 动态 User-Agent 切换：通过 random 模块随机选择请求头，模拟真实用户访问；
2. 反爬检测及重试：若返回的页面中包含提示“访问豆瓣的方式有点像机器人程序”，自动暂停后重试；
3. 分页控制：自动识别影评总页数，并依次爬取所有页面内容。

每部电影的影评人存储为独立的 CSV 文件，文件名格式为 {电影编号}{电影名}{影评人数}.csv，如图 2.1 所示。所有电影的影评人数信息汇总为 reviewers\_count.csv 文件，格式为 title,link,reviewers\_count，便于后续分析。



1.肖申克的救赎(12984).csv	2024/11/27 12:05	Microsoft Excel 逗...	154 KB
2.霸王别姬(9763).csv	2024/11/27 14:50	Microsoft Excel 逗...	118 KB
3.阿甘正传(6946).csv	2024/11/28 15:44	Microsoft Excel 逗...	82 KB
4.这个杀手不太冷(5977).csv	2024/11/27 14:59	Microsoft Excel 逗...	71 KB
5.泰坦尼克号(8502).csv	2024/11/27 15:08	Microsoft Excel 逗...	102 KB
6.美丽人生(5454).csv	2024/11/27 15:16	Microsoft Excel 逗...	65 KB
7.千与千寻(7031).csv	2024/11/27 15:36	Microsoft Excel 逗...	85 KB
8.辛德勒的名单(3080).csv	2024/11/27 15:38	Microsoft Excel 逗...	37 KB
9.盗梦空间(6684).csv	2024/11/27 16:38	Microsoft Excel 逗...	77 KB
10.忠犬八公的故事(4289).csv	2024/11/27 16:48	Microsoft Excel 逗...	51 KB
11.星际穿越(7104).csv	2024/11/27 17:02	Microsoft Excel 逗...	85 KB
12.楚门的世界(7433).csv	2024/11/27 17:38	Microsoft Excel 逗...	89 KB
13.海上钢琴师(10098).csv	2024/11/27 20:43	Microsoft Excel 逗...	118 KB
14.三傻大闹宝莱坞(5109).csv	2024/11/27 21:08	Microsoft Excel 逗...	60 KB
15.机器人总动员(3137).csv	2024/11/27 21:11	Microsoft Excel 逗...	37 KB
16.放牛班的春天(3728).csv	2024/11/27 21:23	Microsoft Excel 逗...	44 KB

图 2.1 爬取的影评人文件

## 2.2 数据预处理

在完成豆瓣电影 Top250 的影评人数据爬取后，需对这些分散的影评数据进行整合和处理，为后续分析构建一个结构化的影评人-电影矩阵。本部分主要描述数据预处理的步骤。

将爬取到的影评人数据整合成一个矩阵化的形式。其中，矩阵的列为 250 部电影的名称。矩阵的行为所有影评人（去重）。矩阵的标志位为二值化标记（1 表示该影评人评论过该电影，0 表示未评论）。

如图 2.2 所示，此矩阵能够清晰展示每位影评人对 Top250 电影的评论分布，为后续的图构建与社区分析提供基础。数据整合步骤如下：

### 1. 读取影评人数据

从 reviewers\_count.csv 文件中逐一读取每部电影的评论信息文件（如 {电影编号}{电影名}{影评人数}.csv），提取影评人的用户名。通过构建影评人字典，将每位影评人评论过的电影进行标记。

### 2. 影评人去重及统一记录

将分散在多个电影文件中的影评人列表去重。每位影评人被唯一标识，同时记录其评论的电影。在构建影评人字典时，如果影评人已存在，则为其添加对应电影的标志位。

### 3. 缺失值填充

对于影评人未评论的电影，矩阵中补充标志位 0。

### 4. 矩阵构建与保存

构建一个包含影评人和电影评论标志的 DataFrame，其中，第一列为影评人的索引编号，第二列为影评人的用户名，其余列为 250 部电影的名称，对应的值为影评人是否评论该电影的标志位。

最终，矩阵保存为 reviewersAndMoviesMatrix.csv 文件，供后续分析使用。

index	name	肖申克的救赎	霸王别姬	阿甘正传	这个杀手不太冷静	泰坦尼克号	美丽人生
0	大头绿豆	1	0	0	0	0	0
1	隱居雲上	1	0	0	0	0	0
2	冷十三	1	0	0	0	0	0
3	方枪枪	1	0	1	0	0	0
4	aratana	1	0	0	0	0	0
5	養笠翁	1	0	0	0	0	0
6	油爆虾	1	0	0	0	0	0
7	附离	1	0	0	0	0	0
8	刘泽宇	1	0	0	0	0	0
9	無腳之鳥	1	0	0	0	0	0
10	伊谢尔伦白	1	0	0	0	0	0
11	仰山雪	1	0	1	1	1	0
12	彩虹	1	0	1	1	0	0
13	三二	1	0	0	1	0	0
14	本来老六	1	1	0	0	0	0
15	Vic	1	0	0	0	0	0
16	布灵布灵	1	0	0	0	0	0
17	杨大志°	1	1	0	1	0	0
18	hghh	1	0	0	0	0	0
19	美神经	1	1	1	1	1	0

图 2.2 影评人-电影矩阵

2.3 数据集特性

本项目的数据集以影评人和电影之间的评论关系为基础，构建了一个无向加权图，图中的节点和边具有如下特性：

节点：每个节点表示一位影评人，其编号为唯一标识。

边：如果两个影评人评论了至少一部相同的电影，则在这两个节点之间添加一条边。

权值：边的权值表示两位影评人共同评论的电影数量。权值越高，说明两位影评人具有更大的共同兴趣或关注范围。

由于每部电影的评论人数往往较多，导致图中的节点之间连接较为密集。如果对未剪枝的整个影评人-电影矩阵的原始图进行分析，图的规模会非常大。由于影评电影数较少的影评人可能在图中并不重要，可以采用剪枝策略来减少图的规模并突出重要节点。表 2.1 展示了不同“影评电影数”阈值下生成图的顶点数量和更新边权值的次数（附录代码 2）。

阈值	顶点数	更新次数
1	278280	1217216650
2	86473	485972075
3	45160	287880419
5	19941	142635591

10	5546	41685024
15	3222	23496676
20	1906	13149019
25	1266	8147233
30	894	5377098

表 2.1 不同阈值下生成图的预估规模

在后续实验中采用的阈值为 30，即对所有影评电影数大于等于 30 的影评人生成的共现图进行分析。该共现图是一个顶点数为 894 和边数为 398866 的无向图。

3 共现图分析

3.1 中心性分析

中心性分析用于识别共现图中具有重要影响力的节点（影评人），我们选取加权度数中心性和加权介数中心性两种指标。

加权度数中心性反映节点直接连接的边的权重之和，表示节点与其他节点之间的联系强度。在影评人共现图中，度数中心性高的影评人评论过更多电影，与其他影评人具有更大的共现联系，体现出较强的活跃度和影响力。表 3.1 展示了加权度数中心性排名前 Top10 的顶点（附录代码 4）。

排名	评论人	加权度数中心性
1	273	45481
2	2	45360
3	464	45349
4	622	44009
5	435	43837
6	154	42502
7	407	40741
8	749	39568
9	179	37343
10	323	37008

表 3.1 加权度数中心性排名前 top10 的顶点

可以看出，排名前 10 的影评人具有较高的活跃度，说明这些影评人不仅评论了大量电影，还与其他影评人有广泛的共现联系，是网络中的重要节点。为了凸显结果的差异，图 3.1 是基于顶点的加权度数值来构建的可视化。

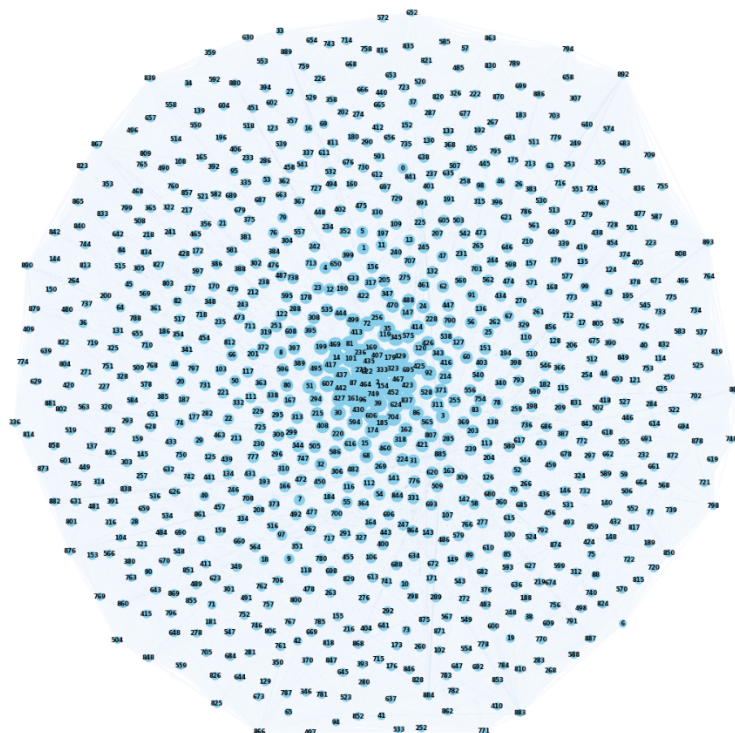


图 3.1 基于顶点的加权重数值来构建的可视化

加权介数中心性用于衡量节点在网络中的桥梁作用，表示节点在最短路径中出现的次数。加权介数中心性较高的影评人，在网络中的信息传递中起到重要的中介作用。表 3.2 展示了加权介数中心性排名前 Top10 的顶点（附录代码 4）。

排名	评论人	加权介数中心性
1	892	0.12003883789107435
2	794	0.07253072698261352
3	883	0.06811636130150814
4	888	0.052240856308842516
5	798	0.04923122132502038
6	839	0.04346473054511763
7	771	0.042477652559872764
8	748	0.038919854503999346
9	852	0.027340639582949283
10	867	0.02730167962984898

表 3.2 加权介数中心性排名前 Top10 的顶点

## 3.2 社区检测

社区发现旨在挖掘共现图中的兴趣群体。本项目使用 Louvain 算法进行社区

划分。Louvain 算法基于模块度最大化，能够高效地将影评人网络划分为多个内部连接密集、外部连接稀疏的子群体。

通过 Louvain 算法，网络被划分为 3 个主要社区，各社区的节点数如表 3.3，基于社区划分共现图如图 3.2 所示。

社区	人数
0	261
1	293
2	340

表 3.3 各社区的影评人人数

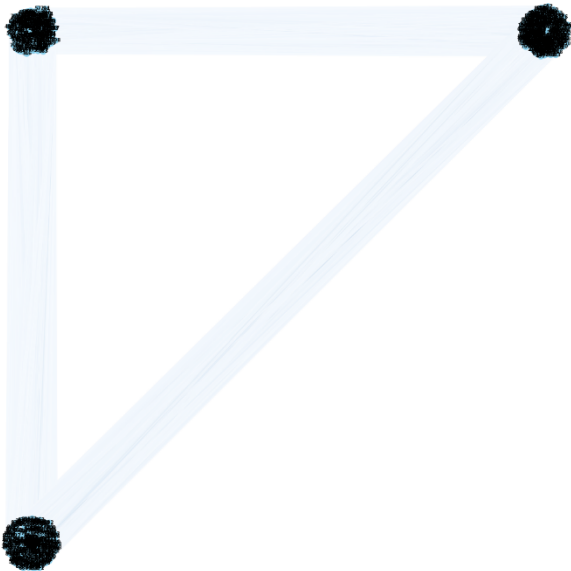


图 3.2 社区划分结果

4 结论

本项目通过分析豆瓣电影 Top250 影评人共现图，揭示影评人之间的关系网络和兴趣分布。通过中心性分析，识别出网络中的意见领袖，这些影评人具有较强的活跃度和桥梁作用，是关键节点。社区发现则揭示了影评人群体的兴趣共性，为理解用户行为和推荐系统提供了依据。

整体而言，共现图分析有效挖掘了影评人网络中的核心节点和兴趣结构，为社交网络分析和应用场景（如推荐系统）提供了重要支持。

## 5 附录

### 1 代码 1：爬取数据代码

```
import requests

from bs4 import BeautifulSoup

import pandas as pd

import time

from lxml import etree

import os # 引入 os 模块, 用于创建文件夹

import random

import json

agents = [

    "Mozilla/5.0 (Linux; U; Android 2.3.6; en-us; Nexus S Build/GRK39F) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1",

    "Avant Browser/1.2.789re11 (http://www.avantbrowser.com)",

    "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/532.5 (KHTML, like Gecko) Chrome/4.0.249.0 Safari/532.5",

    "Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US) AppleWebKit/532.9 (KHTML, like Gecko) Chrome/5.0.310.0 Safari/532.9",

    "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/534.7 (KHTML, like Gecko) Chrome/7.0.514.0 Safari/534.7",

    "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/534.14 (KHTML, like Gecko) Chrome/9.0.601.0 Safari/534.14",

    "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.14 (KHTML, like Gecko) Chrome/10.0.601.0 Safari/534.14",

    "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.20 (KHTML, like Gecko) Chrome/11.0.672.2 Safari/534.20",

    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.27 (KHTML, like Gecko) Chrome/12.0.712.0 Safari/534.27",

    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/13.0.782.24 Safari/535.1",

    "Mozilla/5.0 (Windows NT 6.0) AppleWebKit/535.2 (KHTML, like Gecko) Chrome/15.0.874.120 Safari/535.2",

    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.36 Safari/535.7",

    "Mozilla/5.0 (Windows; U; Windows NT 6.0 x64; en-US; rv:1.9pre) Gecko/2008072421 Minefield/3.0.2pre",

    "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.10) Gecko/2009042316 Firefox/3.0.10",

    "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-GB; rv:1.9.0.11) Gecko/2009060215 Firefox/3.0.11 (.NET CLR 3.5.30729)",

    "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6 GTB5",

    "Mozilla/5.0 (Windows; U; Windows NT 5.1; tr; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8 ( .NET CLR 3.5.30729; .NET4.0E)",
```

```

"Mozilla/5.0 (Windows NT 6.1; rv:2.0.1) Gecko/20100101 Firefox/4.0.1",
"Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1",
"Mozilla/5.0 (Windows NT 5.1; rv:5.0) Gecko/20100101 Firefox/5.0",
"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:6.0a2) Gecko/20110622 Firefox/6.0a2",
"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:7.0.1) Gecko/20100101 Firefox/7.0.1",
"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0b4pre) Gecko/20100815 Minefield/4.0b4pre"]

# 提取电影链接函数
def get_movie_links(url):
    """从 CSV 文件中读取电影链接和名称"""
    try:
        # 读取 CSV 文件
        df = pd.read_csv('Top250.csv')

        # 确保第一列是电影名称，第二列是电影链接
        if len(df.columns) < 2:
            print("CSV 文件格式错误，请确保第一列为电影名称，第二列为电影链接。")
            return []

        # 提取名称和链接
        movies = []

        for index, row in df.iterrows():
            title = row.iloc[0] # 使用 iloc 明确表示按位置访问
            link = row.iloc[1] # 使用 iloc 明确表示按位置访问
            movies.append({'title': title.strip(), 'link': link.strip()})

        return movies
    except Exception as e:
        print(f"读取 CSV 文件失败，错误: {e}")
        return []

# 修改后的影评人提取函数，支持分页抓取
def get_movie_data(link):
    """获取单部电影的所有影评人，支持代理切换"""
    reviewers = [] # 用于存储影评人名称

    # 获取影评的总页数
    total_pages = 1
    while total_pages <= 1:
        try:
            # 设置请求头
            headers = {
                'User-Agent': random.choice(agents),
                'cookie': ''bid=CnGq1bCceRY; _pk_id.100001.4cf6=ea78ed9f70cee281.1732094671.;

```



```
__utmt=30149280.1732094672.1.1.utmcsr=accounts.douban.com|utmccn=(referral)|utmcmd=referral|utmcct=/
;

__utmt=223695111.1732094672.1.1.utmcsr=accounts.douban.com|utmccn=(referral)|utmcmd=referral|utmcct=/
/; ll="118283"; __vwo_uuid_v2=D20818C0069160E6F0D77CA7AF135609C|9d0c7905d992fdabcbdaabba518534e8;
__utmc=30149280; __utmc=223695111; __yadk_uid=pM7Ei1UkY09Kob1KPtXe9xpuaaTvEoT;
dbcl2="284939293:uziaY2CNoq8"; ck=84n1; frodotk_db="16c5f016bcc998dc4451d044365cca94";
push_noty_num=0; push_doumail_num=0; _ga=GA1.2.348907611.1732094672;
_gid=GA1.2.598864478.1732591024; _ga_PRH9EWN86K=GS1.2.1732591024.1.0.1732591024.0.0.0;
__utmv=30149280.28493;
__pk_ref.100001.4cf6=%5B%22%22%2C%22%22%2C1732627899%2C%22https%3A%2F%2Faccounts.douban.com%2F%22%5D;
__utma=30149280.348907611.1732094672.1732589232.1732627899.7;
__utma=223695111.131605596.1732094672.1732589232.1732627899.5''

}

response = requests.get(link + '/reviews', headers=headers, timeout=5)

if response.status_code == 200:

    txt = response.text

    # 检查是否包含指定的子字符串

    if "访问豆瓣的方式有点像机器人程序" not in txt:

        html = etree.HTML(txt)

        total_pages = html.xpath('//span[@data-total-page]/@data-total-page')

        total_pages = int(total_pages[0]) if total_pages else 1 # 默认至少有 1 页

        # break

    else:

        print("当前访问已被 ban!")

        time.sleep(5)

else:

    print(f"请求失败, 状态码: {response.status_code}")

except Exception as e:

    print(f"当前请求不可用, 错误: {e}, 再次请求")

# 遍历每一页, 提取影评人名称

for page in range(total_pages):

    page_url = f"{link}/reviews?start={page * 20}"

    while True:

        # 设置请求头

        headers = {

            'User-Agent': random.choice(agents),

            'cookie': ''bid=CnGq1bCceRY; __pk_id.100001.4cf6=ea78ed9f70cee281.1732094671.;
__utmt=30149280.1732094672.1.1.utmcsr=accounts.douban.com|utmccn=(referral)|utmcmd=referral|utmcct=/
;

__utmt=223695111.1732094672.1.1.utmcsr=accounts.douban.com|utmccn=(referral)|utmcmd=referral|utmcct=
/; ll="118283"; __vwo_uuid_v2=D20818C0069160E6F0D77CA7AF135609C|9d0c7905d992fdabcbdaabba518534e8;
__utmc=30149280; __utmc=223695111; __yadk_uid=pM7Ei1UkY09Kob1KPtXe9xpuaaTvEoT;
```

```

dbc12="284939293:uziaY2CNoq8"; ck=84n1; frodotk_db="16c5f016bcc998dc4451d044365cca94";
push_noty_num=0; push_doumail_num=0; _ga=GA1.2.348907611.1732094672;
_gid=GA1.2.598864478.1732591024; _ga_PRH9EWN86K=GS1.2.1732591024.1.0.1732591024.0.0.0;
__utmv=30149280.28493;
__pk_ref.100001.4cf6=%5B%22%22%2C%22%22%2C1732627899%2C%22https%3A%2F%2Faccounts.douban.com%2F%22%5D;
__utma=30149280.348907611.1732094672.1732589232.1732627899.7;
__utma=223695111.131605596.1732094672.1732589232.1732627899.5'''
    }
    try:
        response = requests.get(page_url, headers=headers, timeout=5)
        # time.sleep(5)
        if response.status_code == 200:
            txt = response.text
            # 检查是否包含指定的子字符串
            if "访问豆瓣的方式有点像机器人程序" not in txt:
                html = etree.HTML(txt)
                page_reviewers = html.xpath('//div[@class="main review-
item"]//a[@class="name"]/text()')
                reviewers_num = len(page_reviewers)
                if reviewers_num == 0 and (page+1) != total_pages:
                    continue
                reviewers.extend(page_reviewers)
                print(f"已抓取 {page + 1}/{total_pages} 页, 当前页影评人数: {reviewers_num}")
                break
            else:
                print("当前访问已被 ban!")
                time.sleep(5)
        else:
            print(f"请求失败, 状态码: {response.status_code}")
    except Exception as e:
        print(f"请求不可用, 错误: {e}")

    return reviewers

# 主函数, 整合抓取流程并保存结果
def main():
    # 豆瓣 TOP250 电影的第一页链接
    url = 'https://movie.douban.com/top250?start=0&filter='
    movies = get_movie_links(url)
    movies_num = len(movies)

    # 创建一个保存 CSV 文件的文件夹
    output_dir = "movie_reviewers"
    os.makedirs(output_dir, exist_ok=True)

```

```

# 创建一个影评人数 CSV 文件
reviewers_count_file = os.path.join(output_dir, "reviewers_count.csv")
if not os.path.exists(reviewers_count_file):
    # 如果文件不存在，则创建并写入表头
    with open(reviewers_count_file, 'w', encoding='utf-8-sig') as f:
        f.write("title,link,reviewers_count\n")

# 遍历每部电影的链接和名称
print(f"总电影数:{movies_num}")
i = 0
while i < 250:
    movie = movies[i]
    title = movie['title']
    link = movie['link']
    print(f"正在爬取: {title} ({link})")

    try:
        reviewers = get_movie_data(link)
        reviewers_num = len(reviewers)
        print(f"总影评人数{reviewers_num}")

        # 保存评论人到独立的 CSV 文件，文件名基于电影名
        csv_filename = os.path.join(output_dir,
f"{i+1}.{title}.({reviewers_num}).csv".replace("/", "-")) # 替换文件名中的特殊字符
        df = pd.DataFrame({'reviewer': reviewers})
        df.to_csv(csv_filename, index=False, encoding='utf-8-sig')
        print(f"电影《{title}》的评论人已保存到 {csv_filename}")

        # 保存影评人数到汇总文件
        with open(reviewers_count_file, 'a', encoding='utf-8-sig') as f:
            f.write(f'"{title}", "{link}", {reviewers_num}\n')

    except Exception as e:
        print(f"爬取失败: {title} ({link}), 错误: {e}")
        time.sleep(2)
        i += 1

if __name__ == "__main__":
    main()

```

## 2 代码 2：转换影评人-电影矩阵

```
import pandas as pd
import os
import sys
from tqdm import tqdm
from six import moves

# 读取 reviewers_count.csv
reviewers_count_df = pd.read_csv('movie_reviewers/reviewers_count.csv', encoding='utf-8-sig') # 替换
为实际文件路径

# 创建一个空的字典来存储影评人的名字和相关的电影标记
reviewers_dict = {}

# 遍历每部电影的信息
moves_num = 0
for index, row in reviewers_count_df.iterrows():
    title = row['title']
    link = row['link']
    reviewers_count = row['reviewers_count']
    print(f"当前处理电影{moves_num+1}:{title},影评人数:{reviewers_count}")

    # 构建电影对应的 CSV 文件路径
    movie_csv_filename = f"movie_reviewers\\{index + 1}.{title}.({reviewers_count}).csv".replace("/",
    "-") # 电影的 csv 文件名

    # 判断电影的 CSV 文件是否存在
    if not os.path.exists(movie_csv_filename):
        # 文件不存在，停止代码
        print(f"文件不存在: {movie_csv_filename}")
        sys.exit() # 停止程序执行

    try:
        # 读取电影的 CSV 文件
        movie_reviewers_df = pd.read_csv(movie_csv_filename, encoding='utf-8-sig')
        reviewers = movie_reviewers_df['reviewer'].tolist() # 获取影评人列表

        # 遍历影评人，并为每个影评人添加电影的标记
        for reviewer in tqdm(reviewers, desc="Processing reviewers", unit="reviewer"):
            if reviewer not in reviewers_dict:
                reviewers_dict[reviewer] = {'name': reviewer} # 初始化字典中的影评人

        # 给影评人添加电影标记（1 表示该影评人在该电影下有评论）
        reviewers_dict[reviewer][title] = 1
```

```
except Exception as e:
    print(f"读取电影文件失败: {movie_csv_filename}, 错误: {e}")
    sys.exit() # 停止程序执行
moves_num += 1

# 将所有电影的名字作为列, 构建每个影评人的数据
all_movie_titles = reviewers_count_df['title'].tolist()
for reviewer in reviewers_dict.values():
    # 对于每个影评人, 添加缺少的电影标记 (如果该影评人没有评论某部电影, 标记为 0)
    for title in all_movie_titles:
        if title not in reviewer:
            reviewer[title] = 0

# 转换为 DataFrame 并保存
reviewers_data = []
for idx, reviewer in enumerate(reviewers_dict.values()):
    row = [idx] # 第一列为索引号
    row.append(reviewer['name']) # 第二列为名字
    row.extend([reviewer.get(title, 0) for title in all_movie_titles]) # 后续列为电影的评论标记
    reviewers_data.append(row)

# 创建 DataFrame
df_reviewers_and_movies = pd.DataFrame(reviewers_data, columns=['index', 'name'] + all_movie_titles)

# 保存为 CSV 文件
df_reviewers_and_movies.to_csv('reviewersAndMoviesMatrix.csv', index=False, encoding='utf-8-sig')
print("保存成功!")
```

### 3 代码 3：使用 NetworkX 创建图数据结构

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from tqdm import tqdm

# 读取 reviewersAndMovies.csv 文件
file_num = 30
df = pd.read_csv(f'reviewersAndMovies\\reviewersAndMovies_filtered{file_num}.csv')

# 提取影评人之间的评论关系（从第三列开始）
# 我们遍历每部电影，若两个影评人都在该电影中评论，则它们之间建立一条边
G = nx.Graph()

# 遍历每一部电影
moves_num = 0
for col in df.columns[2:]: # 获取电影名
    # 获取评论该电影的影评人索引（即评论该电影的影评人对应的索引号，从 0 开始的索引号，非 'index' 列）
    reviewers = df[df[col] == 1].index.tolist() # 选取评论该电影的影评人
    reviewers_num = len(reviewers)

    # 计算该电影创建/更新边权值的次数，公式为  $C(n, 2) = n * (n - 1) / 2$ 
    edges_for_movie = reviewers_num * (reviewers_num - 1) // 2

    # 输出当前电影的信息
    print(f"当前处理电影 {moves_num + 1}: {col}, 影评人数: {reviewers_num}, 需要创建/更新边权值的次数: {edges_for_movie}")

    # 为每两个评论该电影的影评人之间添加一条边（双重循环遍历影评人索引列表）
    with tqdm(total=edges_for_movie, desc=f"Processing {col}") as pbar_movie:
        for i in range(reviewers_num):
            for j in range(i + 1, reviewers_num):
                # 检查边是否已经存在，若存在则增加权重，否则创建新边
                if G.has_edge(reviewers[i], reviewers[j]):
                    G[reviewers[i]][reviewers[j]]['weight'] += 1 # 如果已有边，权重+1
                else:
                    G.add_edge(reviewers[i], reviewers[j], weight=1) # 如果没有边，创建并赋初值为 1
                pbar_movie.update(1) # 每添加一条边，更新进度条

    moves_num += 1

# 获取图的边数和顶点数
num_edges = G.number_of_edges()
num_nodes = G.number_of_nodes()
```

```
# 输出边数和顶点数
print(f"图的顶点数: {num_nodes}")
print(f"图的边数: {num_edges}")

# 保存图的节点和边数据到本地
nx.write_gml(G, f'reviewers_network{file_num}({num_nodes})({num_edges}).gml')

# 从 GML 文件加载图
# G = nx.read_gml('graph.gml')
```

## 4 代码 4：中心性分析

```
import pandas as pd
import networkx as nx
from tqdm import tqdm
import matplotlib.pyplot as plt

# 读取图数据（假设已经创建了加权图）
G = nx.read_gml('reviewers_network.gml')

# 1. 加权度数 (Weighted Degree): 节点的度数乘以其连接的边的权重总和
# 加权度数计算，节点大小由加权度数决定
weighted_degrees = dict(G.degree(weight='weight')) # 根据边权重计算度数
sorted_weighted_degrees = sorted(weighted_degrees.items(), key=lambda x: x[1], reverse=True)

# 输出前 10 个度数最多的评论人及其加权度数
print("Top 10 评论人（加权度数）:")
for reviewer, degree in sorted_weighted_degrees[:10]:
    print(f"评论人: {reviewer}, 加权度数: {degree}")

# 2. 加权介数中心性 (Weighted Betweenness Centrality): 加权介数中心性考虑了边的权重，边的权重越大，它对最
# 短路径的影响就越大。
betweenness = nx.betweenness centrality(G, weight='weight')
sorted_betweenness = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)

# 输出前 10 个介数中心性最高的评论人
print("\nTop 10 评论人（介数中心性）:")
for reviewer, centrality in sorted_betweenness[:10]:
    print(f"评论人: {reviewer}, 介数中心性: {centrality}")

# 使用加权度数来调整节点大小，并设置最大值（根据加权度数的平方根或线性关系调整）
max_node_size = 400
node_size = [min(max(sorted_weighted_degrees [node] // 1000 * 8 , 10), max_node_size) for node in
G.nodes()] # 节点大小比例

# 绘制图形
# font_weight = 'normal': 常规字体，默认的字体的粗细。 'bold': 粗体字。 'light': 细体字。 'ultrabold':
# 超粗体字。 'ultralight': 超细体字。
nx.draw(G, pos, with_labels=True, node_size=node_size, edge_color=edge_weight, width=1.0,
        edge_cmap=plt.cm.Blues, font_size=6, font_weight='bold', node_color='skyblue')

# 绘制边的颜色和宽度
plt.title('影评人评论关系图')

# 保存图形
```



```
plt.savefig('reviewers_network.png', format='PNG')  
plt.close()
```

## 5 代码 5：社区检测

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import community as community_louvain
from networkx.algorithms import community

# 从 GML 文件加载图
G = nx.read_gml('reviewers_network.gml')

# 使用社区检测算法（例如 Louvain 算法）检测社区
communities = community_louvain_communities(G)
# 统计每个社区的节点数
community_sizes = {i: len(comm) for i, comm in enumerate(communities)}

# 打印每个社区的节点数
print("社区发现结果:")
for comm_id, size in community_sizes.items():
    print(f"Community {comm_id} has {size} nodes.")

# 为每个节点分配社区标签
node_community = {}
for i, comm in enumerate(communities):
    for node in comm:
        node_community[node] = i # 将节点与其对应的社区编号关联

# 绘制图形并保存
plt.figure(figsize=(12, 12))

# 使用 spring_layout 进行布局
pos = nx.spring_layout(G, seed=42, k=0.9, iterations=100)

# 调整节点位置，使得同一社区的节点尽可能靠近图的相同角落
# 这里我们手动将节点的位置信息修改为社区编号
corner_positions = {
    0: [-1, 1], # 左上角
    1: [1, 1], # 右上角
    2: [-1, -1], # 左下角
    3: [1, -1], # 右下角
}

# 根据社区调整节点的位置
for node, comm in node_community.items():
    x, y = pos[node] # 获取原始位置
```

```

corner_x, corner_y = corner_positions[comm % 4] # 根据社区划分到不同角落
pos[node] = [corner_x + x * 0.1, corner_y + y * 0.1] # 细微调整位置, 避免重叠

# 调整边的宽度和节点大小, 边的宽度基于权重, 节点大小基于加权度数
edges = G.edges(data=True)
edge_width = [d['weight'] for u, v, d in edges] # 使用权重来调整边的宽度

# 加权度数计算, 节点大小由加权度数决定
weighted_degrees = dict(G.degree(weight='weight')) # 根据边权重计算度数
# 使用加权度数来调整节点大小, 并设置最大值 (根据加权度数的平方根或线性关系调整)
max_node_size = 400
node_size = [min(max(weighted_degrees[node] // 1000 * 8, 10), max_node_size) for node in G.nodes()]
# 节点大小比例

# 绘制图形
# font_weight = 'normal': 常规字体, 默认的字体系数。 'bold': 粗体字。 'light': 细体字。 'ultrabold':
超粗体字。 'ultralight': 超细体字。
nx.draw(G, pos, with_labels=True, node_size=node_size, edge_color=edge_width, width=1.0,
        edge_cmap=plt.cm.Blues, font_size=6, font_weight='bold', node_color='skyblue')

# 绘制边的颜色和宽度
plt.title('Community')

# 保存图形
plt.savefig('reviewers_Community.png', format='PNG')
plt.close()

```