豆瓣TOP250电影数据爬取分析

1. **内容简介**

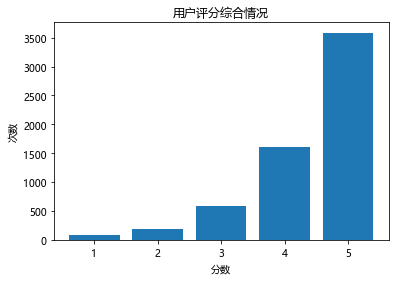
爬取内容：

* 豆瓣电影TOP250的电影序号、ID、名称、上映时间、导演、主演（前四名）、类型、国家/地区、时长、评分、评价人数
* 每部电影的前100条短评：用户ID、用户评分、短评内容

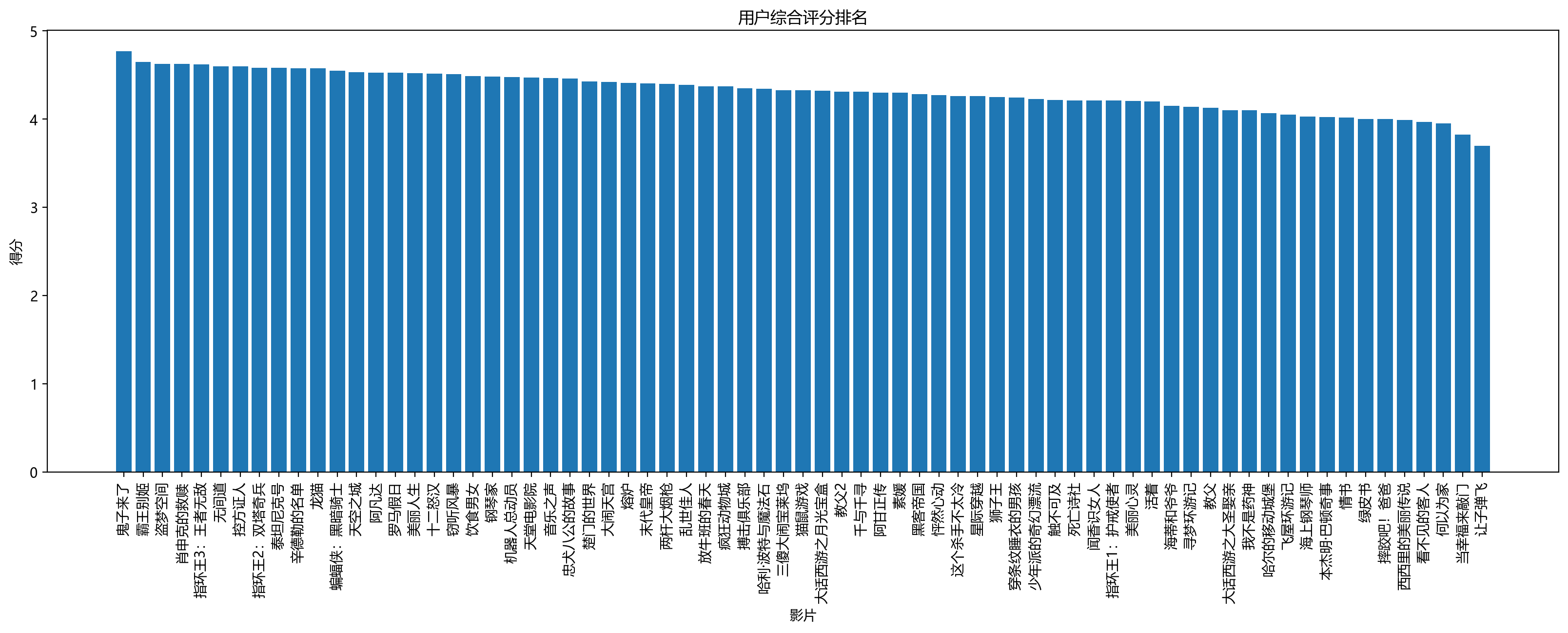
分析内容：

* 用户评分综合统计
* 用户综合评分排名
* 用户评论次数统计
* 用户评论分析

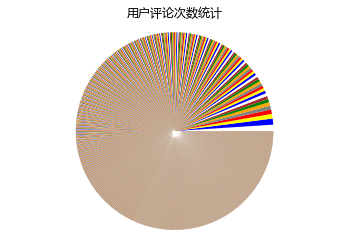
1. **数据处理与分析成果**
2. 用户评分综合统计



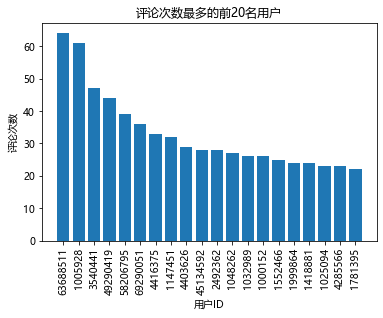
1. 用户综合评分排名



1. 用户评论次数统计



1. 前20名热爱评论的用户



1. 对用户的评论分析



1. 对电影千与千寻的评论分析



1. 详细代码
2. 爬虫程序

import csv

import os

import re

import time

import mongo

import pandas as pd

import requests

import text\_clean

from bs4 import BeautifulSoup

from fake\_useragent import UserAgent

# 基于proxy\_pool项目的代理池

def get\_proxy():

    return requests.get("http://127.0.0.1:5010/get?type=https").json()

def delete\_proxy(proxy):

    requests.get("http://127.0.0.1:5010/delete/?proxy={}".format(proxy))

# 爬取豆瓣电影TOP250的电影序号、ID、名称、上映时间、导演、主演（前四名）、类型、国家/地区、时长、评分、评价人数

# 爬取每个电影的信息页面

def scrape\_movie\_list(ua, cookies):

    url = 'https://movie.douban.com/top250?start=125&filter='

    headers = {

        'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7',

        'User-Agent': ua.random,

        'Connection': 'keep-alive',

        'Cookie': cookies,

        'Host': 'movie.douban.com',

        'Referer': 'https://www.bing.com/'

    }

    movie\_list = []

    while url:

        count = 0

        if count < 25:

            retry\_flag = True

        while retry\_flag:

            proxy = get\_proxy().get('proxy')

            print(proxy)

            try:

                time.sleep(20)

                response = requests.get(url, headers=headers, proxies={

                                        "http": "http://{}".format(proxy), "https": "https://{}".format(proxy)}, timeout=20)

                # response = requests.get(url, headers=headers, timeout=20)

                print(response)

                soup = BeautifulSoup(response.text, 'html.parser')

                items = soup.find\_all('div', class\_='item')

                movie\_list = scrape\_movie\_info(movie\_list, items, ua, cookies)

                count += 1

                retry\_flag = False

            except Exception as e:

                print('Exception:', e)

                print('获取新代理...')

                delete\_proxy(proxy)

        next\_page = soup.find('span', class\_='next').find('a')

        if next\_page:

            url = 'https://movie.douban.com/top250'+next\_page['href']

            # 爬取完一页后输入新的Cookies

            cookies = input('请更新Cookies:')

        else:

            url = None

    return movie\_list

# 爬取电影基本信息

def scrape\_movie\_info(movie\_list, items, ua, cookies):

    for item in items:

        retry\_flag = True

        movie\_url = item.find('a')

        movie\_num = item.find('em').text

        movie\_id = movie\_url['href'].split('/')[-2]

        detail\_headers = {

            'User-Agent': ua.random,

            'Host': 'movie.douban.com',

            'Connection': 'keep-alive',

            'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7',

            'Referer': 'https://movie.douban.com/top250',

            'Cookies': cookies

        }

        # 测试接口获取的代理

        while retry\_flag:

            proxy = get\_proxy().get('proxy')

            print(proxy)

            try:

                # 5秒延迟

                time.sleep(5)

                movie\_detail = requests.get(

                    url=movie\_url['href'], headers=detail\_headers, proxies={"http": "http://{}".format(proxy), "https": "https://{}".format(proxy)}, timeout=20)

                movie\_detail = requests.get(

                    url=movie\_url['href'], headers=detail\_headers, timeout=20)

                print(movie\_detail)

                # 调用 process\_info\_html()

                movie\_list = process\_info\_html(

                    movie\_num, movie\_id, movie\_detail, movie\_list)

                retry\_flag = False

            except Exception as e:

                print(e)

                print('获取新代理...')

                delete\_proxy(proxy)

    return movie\_list

# 对电影信息页面进行处理

def process\_info\_html(movie\_num, movie\_id, movie\_detail, movie\_list):

    soup\_detail = BeautifulSoup(movie\_detail.text, 'html.parser')

    # 提取上映年份

    year = soup\_detail.find(

        'span', class\_='year').text.replace('(', '').replace(')', '')

    # 提取名称

    title = soup\_detail.find(

        'span', attrs={'property': 'v:itemreviewed'}).text.split(' ')[0]

    # 提取导演信息

    director = soup\_detail.find(

        'a', attrs={'rel': 'v:directedBy'}).text

    # 提取演员信息（前四名）若无演员表则设置为空

    try:

        actors = []

        actors\_box = soup\_detail.find('span', class\_='actor').find(

            'span', class\_='attrs').find\_all('a')

        for a in actors\_box[:4]:

            actors.append(a.text)

    except AttributeError:

        actors = []

    # 提取类型信息

    genre\_spans = soup\_detail.find\_all(

        'span', attrs={'property': 'v:genre'})

    genres = [span.get\_text() for span in genre\_spans]

    # 提取制片国家信息

    country\_span = soup\_detail.find('span', string='制片国家/地区:')

    country = country\_span.next\_sibling.strip()

    # 提取时长信息

    duration = soup\_detail.find(

        'span', attrs={'property': 'v:runtime'}).text.replace('分钟', '')

    # 提取评分

    score = soup\_detail.find(

        'strong', attrs={'property': 'v:average'}).text

    # 提取评价人数

    rater\_num = soup\_detail.find(

        'span', attrs={'property': 'v:votes'}).text

    movie\_list.append(

        [movie\_num, movie\_id, year, title, director, actors, genres, country, duration, score, rater\_num])

    print(f'获取到第{movie\_num}部电影：{title}')

# 爬取电影评论的用户ID和评分值

def scrape\_movie\_reviews(movie\_id, ua, cookies):

    url = f'https://movie.douban.com/subject/{movie\_id}/comments?start=0&limit=20&status=P&sort=new\_score'

    print(url)

    time.sleep(5)

    headers = {

        'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7',

        'User-Agent': ua.random,

        'Connection': 'keep-alive',

        'Cookie': cookies,

        'Host': 'movie.douban.com',

        'Referer': f'https://www.douban.com/misc/sorry?original-url=https://movie.douban.com/subject/{movie\_id}/comments?status=P',

    }

    review\_list = []

    wrong\_time = 0

    while url and len(review\_list) < 100:

        retry\_flag = True

        while retry\_flag:

            proxy = get\_proxy().get('proxy')

            time.sleep(20)

            try:

                # 代理

                response = requests.get(url, headers=headers, proxies={

                                        "http": "http://{}".format(proxy), "https": "https://{}".format(proxy)}, timeout=20)

                # response = requests.get(url, headers=headers, timeout=20)

                print(response)

                response.raise\_for\_status()

                soup = BeautifulSoup(response.text, 'html.parser')

                comments = soup.find\_all('div', class\_='comment-item')

                review\_list = scrape\_comment\_info(

                    movie\_id, comments, review\_list)

                retry\_flag = False

            except (requests.exceptions.RequestException, Exception) as e:

                print('Exception:', e)

                wrong\_time += 1

                if wrong\_time == 5:

                    if(len(review\_list) > 70):

                        return review\_list

                    return -1

                print('获取新代理...')

                delete\_proxy(proxy)

        next\_page = soup.find('a', class\_='next')

        if next\_page:

            url = f'https://movie.douban.com/subject/{movie\_id}/comments' + \

                next\_page['href']

        else:

            url = None

    return review\_list

# 爬取每条评价

def scrape\_comment\_info(movie\_id, comments, review\_list):

    for comment in comments:

            # 获取用户id

        user\_id = re.search(

            r'u(\d+)-', comment.find('div', class\_='avatar').find('img')['src']).group(1)

        # 获取评分，若无评分就设置为0分

        try:

            rating = comment.find('span', class\_='rating')['class'][0][-2]

        except:

            rating = '0'

        # 获取短评内容

        short = comment.find('span', class\_='short').text

        review\_list.append([movie\_id, user\_id, rating, short])

    return review\_list

# 保存数据到CSV文件

def save\_to\_csv(data, filename):

    with open(filename, 'a', encoding='utf-8-sig', newline='') as file:

        writer = csv.writer(file)

        writer.writerows(data)

# 对CSV文件去重

def remove\_duplicates(csv\_file):

    print('开始去重...')

    df = pd.read\_csv(csv\_file, header=None)

    df.drop\_duplicates(inplace=True)

    df.to\_csv(csv\_file, index=False, header=False)

    print('去重完成')

if \_\_name\_\_ == '\_\_main\_\_':

    ua = UserAgent()

    # 开始运行前打开对应的网页设置好cookies

    # 根据经验这几个网页的cookies都一样，设置为变量直接传参

    # ps：这个cookies好像是不会很快改变但是我仍然设置了阶段性更新cookie

    cookies = input('请查看页面是否正常并更新cookie:')

    movie\_data\_file = '../数据分析/movie\_data.csv'

    review\_data\_file = '../数据分析/movie\_reviews.csv'

    # 爬取电影列表

    movie\_list = scrape\_movie\_list(ua, cookies)

    # 保存电影列表数据到CSV文件

    save\_to\_csv(movie\_list, movie\_data\_file)

    # 对电影列表去重

    remove\_duplicates(movie\_data\_file)

    # 爬取电影评论数据并保存到CSV文件

    # 爬取评论列表，直接读文件里爬取好的id去爬评论

    with open(movie\_data\_file, 'r', encoding='utf-8') as file:

        reader = csv.reader(file)

        movie\_list = list(reader)

    # 存储检查点的文件名

    checkpoint\_file = './checkpoint.txt'

    # 断点初始化为1

    last\_save\_index = 1

    for movie in movie\_list:

        if os.path.exists(checkpoint\_file):

            with open(checkpoint\_file, 'r') as checkpoint:

                duandian = checkpoint.read().strip()

                last\_save\_index = int(duandian)

        else:

            duandian = '1'

        reviews = []

        movie\_num = movie[0]

        try:

            if duandian == movie\_num:

                time.sleep(5)

                movie\_id = movie[1]

                print(f'开始抓取第{movie\_num}部电影的评论')

                movie\_reviews = scrape\_movie\_reviews(movie\_id, ua, cookies)

                print(f"获取到第{movie\_num}部电影的{len(movie\_reviews)}条评论")

                if(len(movie\_reviews) < 50):

                    break

                reviews.extend(movie\_reviews)

                # 断点更新

                last\_save\_index += 1

                # 更新检查点为last\_save\_index,即当前电影的movie\_num

                with open(checkpoint\_file, 'w') as checkpoint:

                    checkpoint.write(str(last\_save\_index))

                # 补充评论列表数据到CSV文件

                save\_to\_csv(reviews, review\_data\_file)

                print('抓取完成，数据已保存')

        except Exception as e:

            print(e)

            break

    # 对内容进行去重

    remove\_duplicates(review\_data\_file)

    # 对评论进行清理

    text\_clean.process\_csv(review\_data\_file)

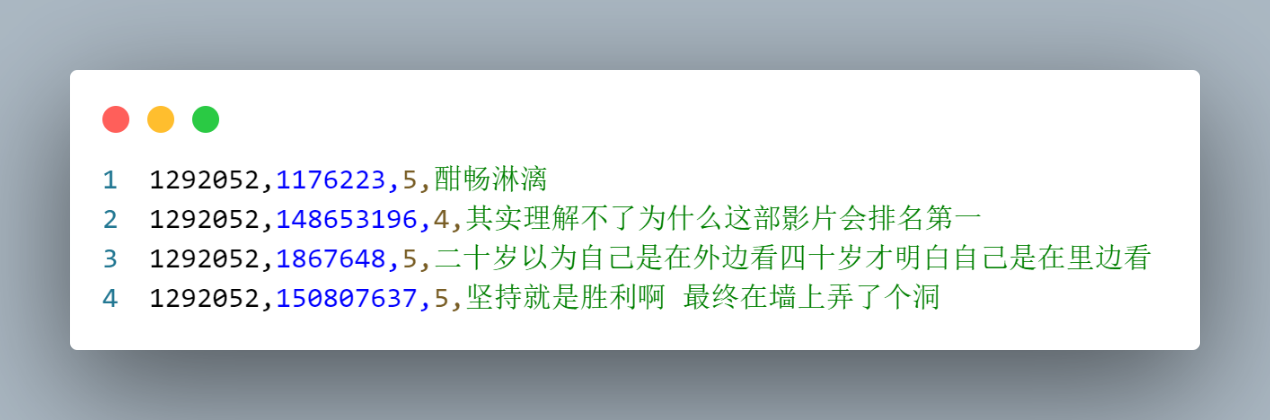
mongo.runMongo()

这部分执行完成后会将所有爬取到的数据存入本地文件和mongodb中

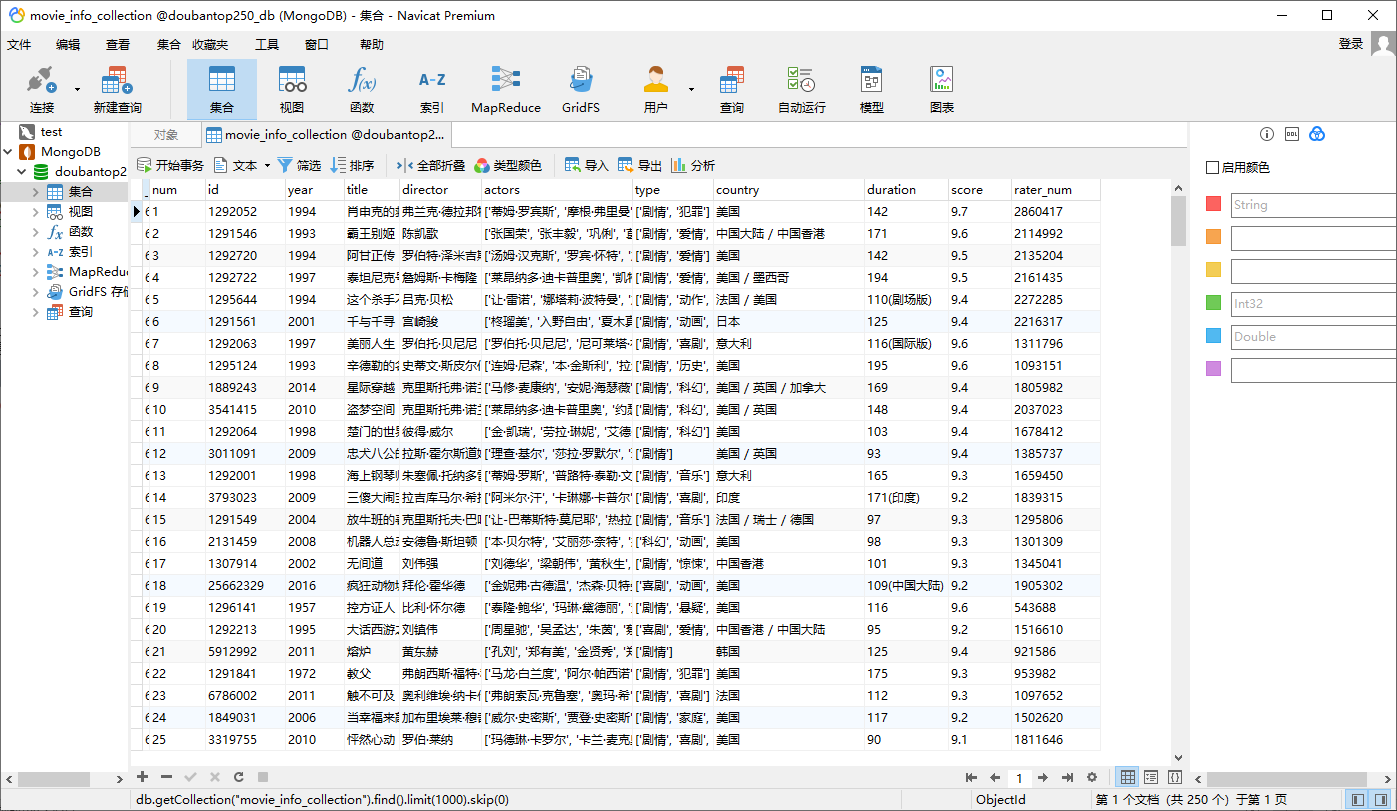
电影信息movie\_data.csv：

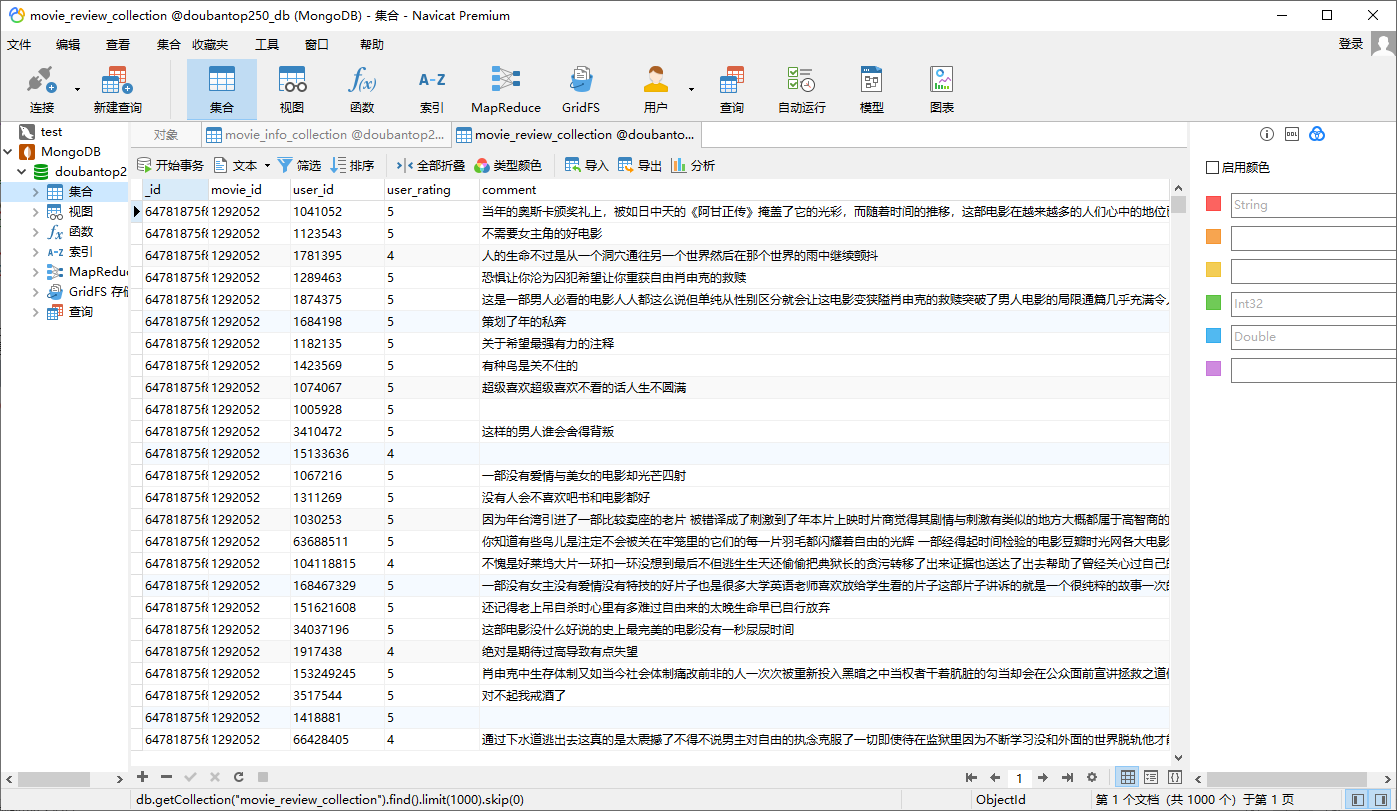


评论信息movie\_reviews.csv：



Mongodb中：





1. 数据分析

**对所有评分进行汇总，找出评分特征：**

ratings=movie\_review\_dataframe[movie\_review\_dataframe['user\_rating'] != 0]['user\_rating']

ratings\_counts=ratings.value\_counts()

print(ratings\_counts)

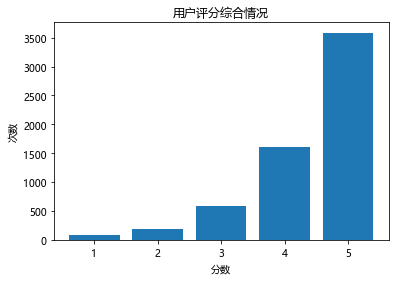
plt.bar(ratings\_counts.index,ratings\_counts.values)

plt.title('用户评分综合情况')

plt.xlabel('分数')

plt.ylabel('次数')

plt.show()



**计算每部电影的用户评分并对电影排序**

average\_ratings=movie\_review\_dataframe.groupby('title')['user\_rating'].mean()

sorted\_movies=average\_ratings.sort\_values(ascending=False)

print(sorted\_movies)

#根据用户评分绘制条形图

plt.bar(sorted\_movies.index,sorted\_movies.values)

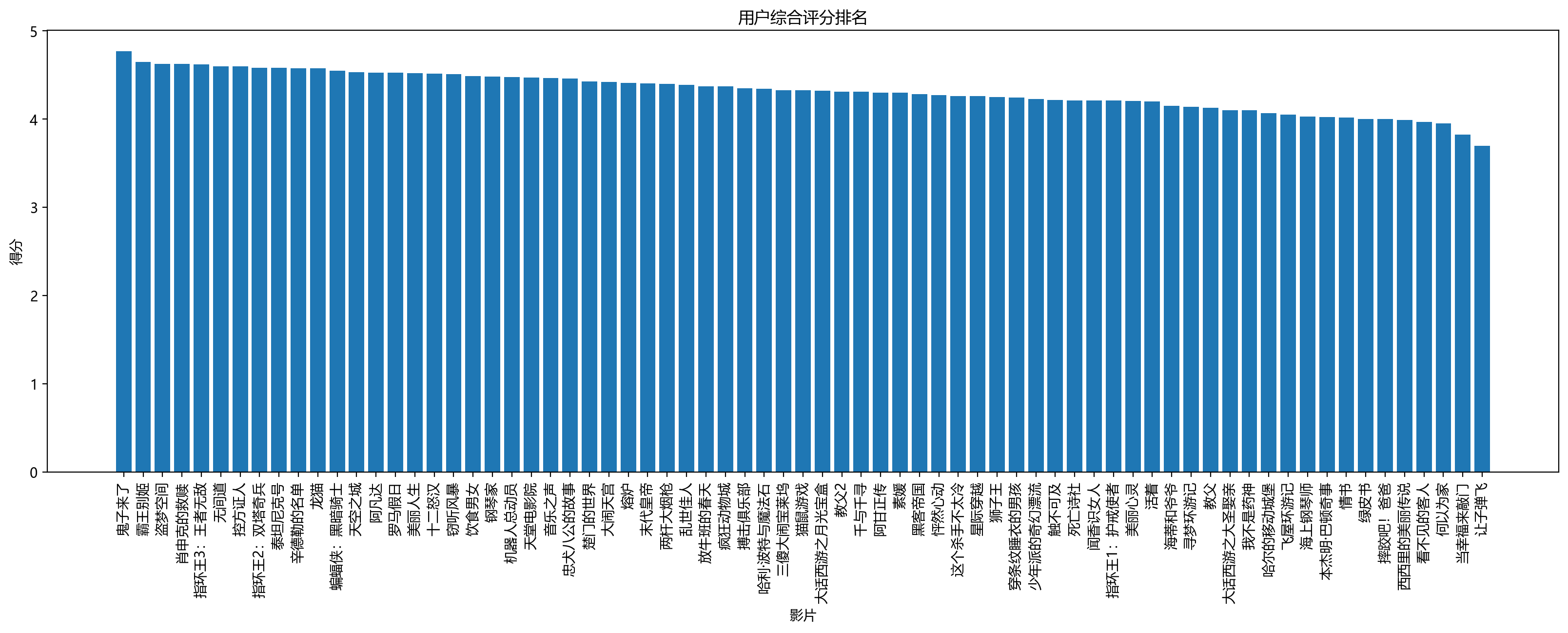
plt.title('用户综合评分排名')

plt.xlabel('影片')

plt.ylabel('得分')

plt.xticks(rotation=90)

plt.show()



**查找出评论次数最多的前二十名用户**

#所有用户评论次数统计

users=movie\_review\_dataframe[['user\_id','comment']]

users\_count=users['user\_id'].value\_counts()

color=['white','blue','yellow','red','gray','orange','green','brown']

plt.pie(users\_count,colors=color)

plt.axis('equal')

plt.title('用户评论次数统计')

plt.show()

top20\_user=users\_count.nlargest(20)

plt.bar(top20\_user.index,top20\_user.values)

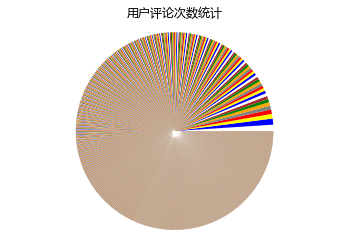
plt.xlabel('用户ID')

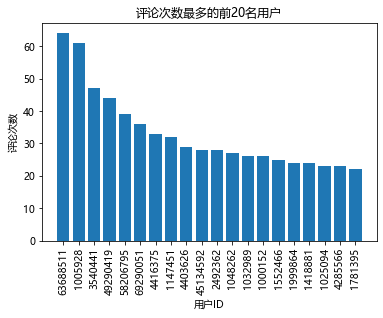
plt.ylabel('评论次数')

plt.title('评论次数最多的前20名用户')

plt.xticks(rotation='90')

plt.show()





**对这些用户的评论进行词云分析**

for user in top20\_user.index:

  user\_data=users[users['user\_id']==user]

  text=' '.join(user\_data['comment'])

  print('用户：',user)

  wc(text)



comment\_dataframes=movie\_review\_dataframe.groupby('title')['comment'].apply(lambda x:' '.join(x))

for title,comment in comment\_dataframes.head(10).items():

  print(f"电影: {title}")

  wc(comment)



**词云生成方法：**

def wc(text):

  seg\_list=jieba.cut(text)

  filtered\_words=[word for word in seg\_list if word not in stopwords]

  filtered\_text=' '.join(filtered\_words)

  wordcloud=WordCloud(font\_path='./msyh.ttf',width=800,height=400,background\_color='white').generate(filtered\_text)

  plt.figure(figsize=(10,5))

  plt.imshow(wordcloud,interpolation='bilinear')

  plt.axis('off')

  plt.show()

1. 基于物品的协同过滤简单电影推荐

import math

import random

from operator import itemgetter

class ItemBasedCF():

    # 初始化参数

    def \_\_init\_\_(self):

        # 找到相似的20部电影，为目标用户推荐10部电影

        self.n\_sim\_movie = 20

        self.n\_rec\_movie = 10

        # 将数据集划分为训练集和测试集

        self.trainSet = {}

        self.testSet = {}

        # 用户相似度矩阵

        self.movie\_sim\_matrix = {}

        self.movie\_popular = {}

        self.movie\_count = 0

        print('计划查找相似电影数 = %d' % self.n\_sim\_movie)

        print('计划推荐电影数 = %d' % self.n\_rec\_movie)

    # 读文件得到“用户-电影”数据

    def get\_dataset(self, filename, pivot=0.875):

        trainSet\_len = 0

        testSet\_len = 0

        for line in self.load\_file(filename):

            user, movie, rating, timestamp = line.split(',')

            if(random.random() < pivot):

                # 相当于trainSet.get(user)，若该键不存在，则设trainSet[user] = {}，典中典

                self.trainSet.setdefault(user, {})

                self.trainSet[user][movie] = rating

                trainSet\_len += 1

            else:

                self.testSet.setdefault(user, {})

                self.testSet[user][movie] = rating

                testSet\_len += 1

        print('训练集测试集划分完成!')

        print('训练集 = %s' % trainSet\_len)

        print('测试集 = %s' % testSet\_len)

    # 读文件，返回文件的每一行

    def load\_file(self, filename):

        with open(filename, 'r', encoding='utf-8') as f:

            for i, line in enumerate(f):

                if i == 0:  # 去掉文件第一行的title

                    continue

                yield line.strip('\r\n')

        print('加载 %s 成功!' % filename)

    # 计算电影之间的相似度

    def calc\_movie\_sim(self):

        for user, movies in self.trainSet.items():  # 循环取出一个用户和他看过的电影

            for movie in movies:

                if movie not in self.movie\_popular:

                    self.movie\_popular[movie] = 0

                self.movie\_popular[movie] += 1  # 统计每部电影共被看过的次数

        self.movie\_count = len(self.movie\_popular)  # 得到电影总数

        print("电影总数 = %d" % self.movie\_count)

        # 得到矩阵C，C[i][j]表示同时喜欢电影i和j的用户数

        for user, movies in self.trainSet.items():

            for m1 in movies:

                for m2 in movies:

                    if m1 == m2:

                        continue

                    self.movie\_sim\_matrix.setdefault(m1, {})

                    self.movie\_sim\_matrix[m1].setdefault(m2, 0)

                    # ItemCF-IUF改进，惩罚了活跃用户 22.00 10.65 14.98

                    self.movie\_sim\_matrix[m1][m2] += 1 / \

                        math.log(1 + len(movies))

        print("建立用户相似度矩阵成功!")

        # 计算电影之间的相似性

        print("计算电影相似度矩阵 ...")

        for m1, related\_movies in self.movie\_sim\_matrix.items():  # 电影m1，及m1这行对应的电影们

            for m2, count in related\_movies.items():  # 电影m2 及 同时看了m1和m2的用户数

                # 注意0向量的处理，即某电影的用户数为0

                if self.movie\_popular[m1] == 0 or self.movie\_popular[m2] == 0:

                    self.movie\_sim\_matrix[m1][m2] = 0

                else:

                    # 计算出电影m1和m2的相似度

                    self.movie\_sim\_matrix[m1][m2] = count / math.sqrt(

                        self.movie\_popular[m1] \* self.movie\_popular[m2])

        print('计算电影相似度矩阵成功!')

        # 添加归一化

        maxDict = {}

        max = 0

        for m1, related\_movies in self.movie\_sim\_matrix.items():

            for m2, \_ in related\_movies.items():

                if self.movie\_sim\_matrix[m1][m2] > max:

                    max = self.movie\_sim\_matrix[m1][m2]

        for m1, related\_movies in self.movie\_sim\_matrix.items():  # 归一化

            for m2, \_ in related\_movies.items():

                self.movie\_sim\_matrix[m1][m2] = self.movie\_sim\_matrix[m1][m2] / max

    # 针对目标用户U，找到K部相似的电影，并推荐其N部电影

    def recommend(self, user):

        K = self.n\_sim\_movie  # 找到相似的20部电影

        N = self.n\_rec\_movie  # 为用户推荐10部

        rank = {}

        watched\_movies = self.trainSet[user]  # 该用户看过的电影

        for movie, rating in watched\_movies.items():  # 遍历用户看过的电影及对其评价

            # 找到与movie最相似的K部电影,遍历电影及与movie相似度

            for related\_movie, w in sorted(self.movie\_sim\_matrix[movie].items(), key=itemgetter(1), reverse=True)[:K]:

                if related\_movie in watched\_movies:  # 如果用户已经看过了，不推荐了

                    continue

                rank.setdefault(related\_movie, 0)

                rank[related\_movie] += w \* float(rating)  # 计算用户对该电影的兴趣

        # 返回用户最感兴趣的N部电影

        return sorted(rank.items(), key=itemgetter(1), reverse=True)[:N]

    # 产生推荐并通过准确率、召回率和覆盖率进行评估

    def evaluate(self):

        print('开始评估 ...')

        N = self.n\_rec\_movie  # 要推荐的电影数

        # 准确率和召回率

        hit = 0

        rec\_count = 0

        test\_count = 0

        # 覆盖率

        all\_rec\_movies = set()

        for i, user in enumerate(self.trainSet):

            test\_moives = self.testSet.get(user, {})  # 测试集中用户喜欢的电影

            rec\_movies = self.recommend(user)  # 得到推荐的电影及计算出的用户对它们的兴趣

            for movie, w in rec\_movies:  # 遍历给user推荐的电影

                if movie in test\_moives:  # 测试集中有该电影

                    hit += 1  # 推荐命中+1

                all\_rec\_movies.add(movie)

            rec\_count += N

            test\_count += len(test\_moives)

        precision = hit / (1.0 \* rec\_count)

        recall = hit / (1.0 \* test\_count)

        coverage = len(all\_rec\_movies) / (1.0 \* self.movie\_count)

        print('准确率=%.4f\t召回率=%.4f\t覆盖率=%.4f' %

              (precision, recall, coverage))

if \_\_name\_\_ == '\_\_main\_\_':

    rating\_file = '../movie\_dataset/ratings.csv'

    itemCF = ItemBasedCF()

    itemCF.get\_dataset(rating\_file)

    itemCF.calc\_movie\_sim()

    itemCF.evaluate()

# 对id为 1 的用户推荐结果

    user\_id = '1'

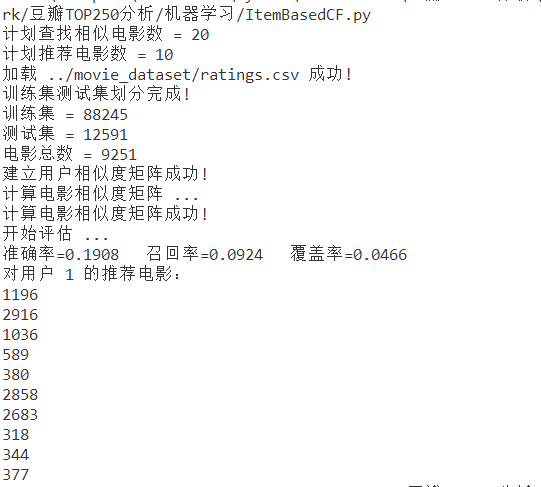
    recommended\_movies = itemCF.recommend(user\_id)

    print(f"对用户 {user\_id} 的推荐电影：")

    for movie, \_ in recommended\_movies:

        print(movie)

**结果：**



1. 学习收获

在这次实验中，我采用了多种技术来完成对豆瓣TOP榜单的数据爬取，并实现了数据分析和机器学习进行电影推荐的功能。这个过程中涉及到了以下几个关键技术：

1. 常规爬虫技术：通过编写爬虫程序，我成功地从豆瓣TOP榜单页面中获取到所需的电影数据。这包括电影的名称、评分、导演、演员等信息。

2. 代理池技术：为了应对反爬虫机制，我使用了代理池技术来随机切换爬虫程序的IP地址，提高爬取数据的稳定性和匿名性。

3. 断点续传：考虑到爬取过程中可能会出现网络中断等问题，我实现了断点续传功能，即在爬取过程中保存已经获取的数据，以便在下次运行时可以从断点处继续爬取，提高数据获取的效率。

4. 异常捕获与处理：在爬取过程中，我使用异常捕获与处理机制来处理可能出现的异常情况，例如网络请求超时、页面解析错误等，以确保程序的稳定性和可靠性。

5. MongoDB存储：我选择使用MongoDB作为数据存储的解决方案。将爬取到的电影数据存储在MongoDB数据库中，方便后续的数据分析和机器学习处理。

6. 词云分析：通过对电影名称和评论文本进行文本分析和词频统计，我生成了词云图，直观展示了电影名称和评论中出现频率较高的关键词，为后续的推荐算法提供了参考。

7. 停用词处理：为了提高词云分析的效果，我使用了停用词列表来过滤掉常见的无意义词语，如冠词、介词等。这样可以更准确地反映出电影名称和评论中的关键信息。

8. 基于物品的协同过滤：最终，我实现了基于物品的协同过滤推荐算法，根据用户的历史行为和电影之间的相似度，为用户推荐可能感兴趣的电影。通过这一算法，我为用户提供了个性化的电影推荐体验。

通过这次实验，我学到了很多关于数据爬取、处理和分析的实际技术。我不仅熟悉了常规爬虫技术，还了解了代理池的应用和断点续传的实现。同时，我学会了如何处理异常情况，保证程序的鲁棒性。使用MongoDB进行数据存储的经验也为我今后的项目开发提供了参考。此外，通过词云分析和基于物品的协同过滤算法，我深入了解了数据分析和机器学习在电影推荐中的应用。

这次实验让我深刻认识到数据的重要性和多样性，同时也感受到了技术的强大和创造力。我意识到，通过合理运用各种技术手段，我们可以从海量的数据中发现有价值的信息，并为用户提供更好的体验和服务。我对数据科学和机器学习的前景充满信心，并期待在未来的项目中继续探索和应用这些技术。