# Python 根据打分数据对某用户进行推荐

编写程序，生成数据模拟（也可以使用网上爬取的真实数据）多人对多部定影的打分（1~5分），然后根据这些数据对某用户A进行推荐。

推荐规则为：在已有的数据中选择与该用户A的爱好最相似的用户B，然后从最相似的用户B已看过但用户A还没看过的电影中选择B打分最高的电影推荐给用户A。其中，相似度的计算标准：

（1）两个用户共同打分过的电影越多，越相似；

（2）两个用户对共同打分的电影的打分越接近，越相似。

## 实现

### 生成数据模拟

import random  
  
from openpyxl import Workbook  
  
user = ['用户1', '用户2', '用户3', '用户4', '用户5', '用户6', '用户7', '用户8',  
 '用户9', '用户10', '用户11', '用户12', '用户13', '用户14', '用户15']  
# 实例化  
wb = Workbook()  
# 激活 worksheet  
ws = wb.active  
# 设置表头  
  
ws.append(['用户', '电影1', '电影2', '电影3', '电影4', '电影5', '电影6', '电影7', '电影8', '电影9', '电影10'])  
  
# i 为用户数 j为电影数  
for i in range(15):  
 numscore = []  
 numscore.append(user[i])  
 for j in range(10):  
 # 生成电影j的分数  
 numscore.append(random.randint(0, 5))  
 # 生成第i行数据  
 ws.append(numscore)  
  
wb.save('电影用户评价信息.xlsx')

通过这种方式，我们可以一个随机出来的打分表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 用户 | 电影1 | 电影2 | 电影3 | 电影4 | 电影5 | 电影6 | 电影7 | 电影8 | 电影9 | 电影10 |
| 用户1 | 0 | 4 | 4 | 0 | 5 | 2 | 0 | 3 | 1 | 5 |
| 用户2 | 1 | 1 | 1 | 4 | 5 | 4 | 1 | 3 | 3 | 0 |
| 用户3 | 3 | 4 | 2 | 3 | 2 | 4 | 5 | 2 | 2 | 2 |
| 用户4 | 3 | 5 | 5 | 0 | 4 | 2 | 2 | 1 | 2 | 2 |
| 用户5 | 2 | 0 | 1 | 2 | 1 | 3 | 0 | 5 | 5 | 1 |
| 用户6 | 1 | 0 | 0 | 0 | 0 | 1 | 5 | 1 | 4 | 5 |
| 用户7 | 5 | 1 | 0 | 0 | 4 | 3 | 1 | 2 | 1 | 1 |
| 用户8 | 0 | 5 | 2 | 1 | 3 | 2 | 4 | 5 | 5 | 4 |
| 用户9 | 3 | 3 | 2 | 3 | 0 | 0 | 0 | 4 | 5 | 3 |
| 用户10 | 2 | 2 | 3 | 1 | 0 | 5 | 3 | 4 | 3 | 4 |
| 用户11 | 2 | 0 | 1 | 1 | 5 | 3 | 1 | 0 | 5 | 3 |
| 用户12 | 5 | 1 | 2 | 1 | 3 | 3 | 4 | 4 | 3 | 4 |
| 用户13 | 5 | 2 | 5 | 5 | 4 | 3 | 3 | 1 | 0 | 3 |
| 用户14 | 2 | 4 | 5 | 0 | 3 | 2 | 0 | 2 | 4 | 0 |
| 用户15 | 2 | 1 | 4 | 2 | 5 | 1 | 0 | 5 | 4 | 4 |

### 数据分析推荐

#### 读取数据

* 读取excel表格数据，并存放于一个字典中

from openpyxl import load\_workbook  
  
# 打开文件及表  
wb = load\_workbook('电影用户评价信息.xlsx')  
ws = wb["Sheet"]  
  
'''使用演员作为键，使用包含该演员参演电影名称的集合作为“值”'''  
fimeDict = dict()  
i = 0  
# 遍历Excel文件中的所有行  
for index, row in enumerate(ws.rows):  
 # 跳过表头，对于每一行有效数据，获取每一行的电影名称和演员清单，  
 if index == 0:  
 continue  
 # 获取电影名称和演员列表  
 user = row[0].value  
 fimescore = [row[i].value for i in range(1, 11)]  
 # 得到评分字典，评价用户作为键，评分列表作为值  
 fimeDict[user] = fimescore  
  
print("评分字典：{}".format(fimeDict))

评分字典：{'用户1': [0, 4, 4, 0, 5, 2, 0, 3, 1, 5], '用户2': [1, 1, 1, 4, 5, 4, 1, 3, 3, 0],...

##### 用户A读入

# 假设来了个新用户  
username = "用户A"  
user = [4, 0, 4, 4, 0, 0, 2, 0, 0, 5]  
print(f"新来的用户为{username}:{user}")

#### 共同打分过的电影多的用户

* 先找出第一个最高分用户

filmSamenum = 0 # 共同打分的数目  
filmSameNumDict = {}  
for key, value in fimeDict.items():  
 # 共同打分的电影分析  
 filmiswatch = []  
 for i in range(len(user)):  
 # 如果双方都看了 ，便为True  
 filmiswatch.append(bool(value[i]) & bool(user[i]))  
 # 统计True的个数，即双方都看的个数  
 filmSamenum = filmiswatch.count(True)  
 # 将此用户名用户名与共同观看个数放入字典  
 filmSameNumDict[key] = filmSamenum  
print(filmSameNumDict)  
# 调用max函数找出最大值对应的键 但此方法只会找到返回一个值  
keyName = max(filmSameNumDict, key=filmSameNumDict.get)  
print("第一个最高匹配人为：{}".format(keyName))  
  
# 再次遍历查看有没有其他的用户观看次数一样  
userSame = []  
maxScore = filmSameNumDict[keyName]  
for key, value in filmSameNumDict.items():  
 # 如果观看次与最大值相同添加进列表  
 if value == maxScore:  
 userSame.append(key)  
print("共同打分过的电影多的用户为：{}".format(userSame))

第一个最高匹配人为：用户3

* 根据最高分找出其他观看次数一样的用户

# 再次遍历查看有没有其他的用户观看次数一样  
userSame = []  
maxScore = filmSameNumDict[keyName]  
for key, value in filmSameNumDict.items():  
 # 如果观看次与最大值相同添加进列表  
 if value == maxScore:  
 userSame.append(key)  
print("共同打分过的电影多的用户为：{}".format(userSame))

共同打分过的电影多的用户为：['用户3', '用户10', '用户11', '用户12', '用户13']

#### 打分越接近的用户

# 开始对相同次数的用户进行第二轮推荐  
userScoreDifferentDict = {}  
for username in userSame:  
 score = 0  
 for index, userscore in enumerate(fimeDict[username]):  
 # 计算两个用户的打分差距  
 if userscore != 0 and user[index] != 0:  
 score += abs(userscore - user[index])  
 # 将计算出的  
 userScoreDifferentDict[username] = score  
print("相似用户的得分字典为：{}".format(userScoreDifferentDict))

相似用户的得分字典为：{'用户3': 10, '用户10': 8, '用户11': 11, '用户12': 9, '用户13': 6}

#### 转换出电影名

# 统计出其观看电影的清单fimeName = ['电影1', '电影2', '电影3', '电影4', '电影5', '电影6', '电影7', '电影8', '电影9', '电影10']  
selectefimeName = []  
selecteName = min(userScoreDifferentDict, key=userScoreDifferentDict.get)  
for index, fime in enumerate(fimeDict[selecteName]):  
 if fime != 0:  
 selectefimeName.append(fimeName[index])  
print("最高匹配人为：{0}，他所观看的电影为{1}".format(selecteName, selectefimeName))

最高匹配人为：用户13，他所观看的电影为['电影1', '电影2', '电影3', '电影4', '电影5', '电影6', '电影7', '电影8', '电影10']

#### 找出用户未看过的电影

# 求出用户观看的电影列表  
userfimeName = []  
for index, fime in enumerate(user):  
 if fime != 0:  
 userfimeName.append(fimeName[index])  
  
# 转成集合求差集  
selectefimeName = set(selectefimeName)  
userfimeName = set(userfimeName)  
finallFimeName = selectefimeName.difference(userfimeName)  
print("推荐的电影如下：{}".format(finallFimeName))

推荐的电影如下：{'电影6', '电影5', '电影8', '电影2'}

#### 推荐评分最高电影

# 求出这些电影中评分最高的一个  
finallScore = 0  
finallFime = ""  
for fime in finallFimeName:  
 # 找到这个电影的索引  
 index = fimeName.index(fime)  
 # 根据索引找到分数  
 score = fimeDict[selecteName][index]  
 # 比较哪那个高  
 if finallScore < score:  
 finallScore = score  
 finallFime = fime  
 # 如果是最高评分则无需继续查找了  
 if score == 5:  
 break  
print("最终推荐的电影为：{}".format(finallFime))

最终推荐的电影为：电影5

本次模拟的推荐算法是当今社会常见的算法应用，平台的大数据推荐在我们生活中屡见不鲜。本次我在计算标准的指引下，也亲身实际设计了一次大数据推荐算法，本次生成的数据先保存至excel表格，用户作为行，电影名称作为列，实现了评分的存储。在数据读取时，通过同样的方式，以用户名作为键，评分列表作为值进行读取，这也与我们实际应用中的表单传值使用的json数据有异曲同工之处，都是字典搭配列表。读取时首先要统计其他用户与用户A共同打分过的电影数，这里我采用的bool的方式，进行&运算，便得到了一个是否都打过分的布尔列表，然后统计True的个数便可以得出。接下来第二轮比较评分差异时，便通过双循环算出各用户与用户A打分之差，统计时需要将有人未看的电影去除。在最后的找出要推荐的电影阶段，先是通过索引找出对应的电影名称，再将列表转为集合求出差集，最后再次根据索引去查找分数，找出其打分最高的那个电影。