Neighborhood Methods Movie Recommender

实验要求：

根据一个涉及影评者及其几部影片评分情况的字典，实现 collaborative filtering，并为 Toby 进行推荐。

实验思路：

1. 使用 collaborative filtering 中的 neighborhood methods，首先找出与目标用户口味相近的用户，根据相近度对其他用户排序。**相近度**通过两者共同评价过的电影的评分情况来衡量。
2. 考虑到每个用户的评分标准的差异，直接使用某个用户对某部电影的评分可能会造成较大误差（例如，A 评分及其苛刻，对一部较好的电影打了较低的分，而没有对一部较差的电影评分过；B 评分十分宽松，对这部较差的电影打了较高的分，却没有看过较好的那部电影，这样的情况可能造成较差的电影评分比较好的电影要高），重新对字典中的电影评分加权赋值。赋值方法为：计算某个用户所有电影评分的平均值，新值=旧值/平均分，因此新的分值会在 1 左右，我们称它为“**综合评分**”。
3. 把字典中目标用户没有看过的电影作为 keys 放入一个新的字典中，综合推荐指数作为每个 key 的 value。**综合推荐指数**的计算方式为：对每一部目标用户没评分过但其他用户评分过的电影，求其他用户 **相近度 * 综合评分** 的和，再除以这部电影被其他用户评分的总次数。
4. 以综合推荐指数为依据，从高到低对候选推荐电影排序，放入一个列表中，作为推荐电影的推荐顺序。

核心代码片段：

计算用户之间的相近度：

```python
# 计算用户之间的相近度，client1是被推荐的用户
def distance(client1,client2):
    sum = 0
    #计算所有距离的和（差值的平方和）
    #if they have no rating in common, return 0
    for item in critics[client1]:
        if item in critics[client2]:
            sum += pow(critics[client1][item] - critics[client2][item], 2)
    # sum越大，差异越大
    # 为了与用户相近度呈正相关，取欧氏距离的倒数，分母加一是防止分母为0的情况
    return 1/(1+sqrt(sum))
```

根据相似度排序用户：

```python
# 根据相似度排序用户，这个函数可以不用到
def rankDistance(client,n=6):
    rank_list = []
    for item in critics:
        if item != client:
            rank_list.append((distance(client,item),item))
    #排序
    rank_list.sort()
    rank_list.reverse()
    return rank_list[0:n]
```

计算综合评分、综合推荐指数：

```python
# 计算综合评分、综合推荐指数
def getRecommendations(client):
    rec_dic = {} # 存放推荐的电影和推荐指数
    movieNum_dic = {} # 存放推荐的电影在其他用户中出现的总次数
    # 计算综合评分
    for other_client in ave_critics:
        movieNum = 0 # 统计某个用户评价过的电影总数
        scoreSum = 0 # 统计某个用户评价过的所有电影总分数
        if other_client != client:
            for movie in ave_critics[other_client]:
                movieNum += 1
                scoreSum += ave_critics[other_client][movie]

            averageScore = scoreSum/movieNum # 某个用户对电影的平均打分
            for movie in ave_critics[other_client]:
                # 将每个用户对某部电影的评分除以他的平均打分
                ave_critics[other_client][movie] = ave_critics[other_client][movie]/averageScore
                # 把符合被推荐资格的电影放入推荐字典中
                if movie not in ave_critics[client] and movie not in rec_dic:
                    rec_dic[movie] = 0
                    movieNum_dic[movie] = 0
    # print("计算平均后的用户-电影字典：",ave_critics)
    # 计算被推荐电影的综合推荐指数
    for other_client in ave_critics:
        if other_client != client:
            for movie in ave_critics[other_client]:
                if movie in rec_dic:
                    # 用户近似度 * 用户平均评分
                    rec_dic[movie] += distance(client,other_client) * ave_critics[other_client][movie]
                    movieNum_dic[movie] += 1
    for movie in rec_dic:
        rec_dic[movie] /= movieNum_dic[movie]
    print("推荐电影被其他用户评分的次数：",movieNum_dic)
    print("推荐电影综合推荐指数：",rec_dic)
    print("推荐电影顺序：",sorted(rec_dic.keys(), reverse=True))
```

实验结果：

Toby 用户相近度排序： [(0.4, 'Mick LaSalle'), (0.38742588672279304, 'Michael Phillips'), (0.3567891723253309, 'Claudia Puig'), (0.3483314773547883, 'Lisa Rose'), (0.2674788903885893, 'Jack Matthews'), (0.25824569976124334, 'Gene Seymour')]

```
推荐电影被其他用户评分的次数： {'Lady in the Water': 5, 'Just My Luck': 4, 'The Night Listener': 6}
推荐电影综合推荐指数： {'Lady in the Water': 0.29341619553631737, 'Just My Luck': 0.263923508015639, 'The Night Listener': 0.3604461845787418
推荐电影顺序： ['The Night Listener', 'Lady in the Water', 'Just My Luck']
```

即若要给用户 Toby 推荐电影，最佳的是 'The Night Listener'，其次是 'Lady in the Water'，最后是 'Just My Luck'。

实验总结：

这次的实验目标是实现一个简易的电影推荐系统，样本的数据也很小，偶然性较大。正是如此，更加要小心极端情况的出现，正如我之前所写，可能会存在这样的情况：A 评分及其苛刻，对一部较好的电影打了较低的分，而没有对一部较差的电影评分过；B 评分十分宽松，对这部较差的电影打了较高的分，却没有看过较好的那部电影，这样的情况可能造成较差的电影评分比较好的电影要高。我认为最终的推荐指数的计算方式应该有多种，我用的只是自己想出来的其中一种，但是这些方法的共性就在于一定要考虑到玩家之间评分标准的差异以及极端情况的存在，尽量让最后的分数能够均衡、公正。我认为在数据集更大的情况下，结果也会更加准确，也可能会有更多的算法能符合要求。

关于 Collaborative Filtering，课上讲到了两种方法，第一种是 Neighborhood Methods，第二种是 Latent Factor Methods。由于第二种方法需要有每个样本的 properties，再根据 properties 放到二维坐标系中去比较邻近关系，考虑到数据样本很小，加上这几个电影我们都不熟悉，更不知道它们的 properties，如果强行加上 properties 很可能推荐结果较差，因此选择了第一种方法——Neighborhood Methods，也就是根据每个用户的评分情况分析它们的口味、喜好，再找喜好接近的用户，根据它们的评分情况计算综合推荐指数来进行推荐。其中计算用户之间的相近度时用到了欧氏距离。