**Task2、协同过滤**

3.1协同过滤算法介绍

协同过滤（Collaborative Filtering）推荐算法是最经典、最常用的推荐算法。

根据用户之前的喜好以及其他兴趣相近的用户的选择来给用户推荐物品(基于对用户历史行为

数据的挖掘发现用户的喜好偏向， 并预测用户可能喜好的产品进行推荐)，一般是仅仅基于用户的行为数据（评价、购 买、下载等）,

而不依赖于项的任何附加信息（物品自身特征）或者用户的任何附加信息（年龄， 性别等）。

基于邻域的方法：目前应用较广泛

1. **基于用户**的协同过滤算法(UserCF): 给用户**推荐**和他**兴趣相似的其他用户喜欢的**产品

2. **基于物品**的协同过滤算法(ItemCF): 给用户**推荐**和他**之前喜欢的物品相似的**物品

算法思想是计算用户和用户或者物品和物品之间的相似度。

3.2 相似性度量方法

1. 杰卡德(Jaccard)相似系数

**衡量两个集合的相似度**

两个用户交互商品交集的数量占这两个用户交互商品并集的数量的比例

2. 余弦相似度

**度衡量了两个向量的夹角**，夹角越小越相似。

从向量的角度进行描述，令矩阵 为用户-商品交互矩阵(因为是TopN推荐并不需要用户对物品的评分，只需要知道

用户对商品是否有交互就行)，即矩阵的每一行表示一个用户对所有商品的交互情况，有交互的商品值为1没有交互的

商品值为0，矩阵的列表示所有商品。若用户和商品数量分别为 的话，交互矩阵 就是一个 行 列的矩

阵。

上述用户-商品交互矩阵在现实情况下是非常的稀疏了，**为了避免存储这么大的稀疏矩阵，在计算用户相似度的时候一般会采用集合的方式进行计算。**

从集合的角度描述余弦相似度，相比于Jaccard公式来说就

是**分母有差异，不是两个用户交互商品的并集的数量，而是两个用户分别交互的商品数量的乘积，**

具体实现的时候， 可以使用 **cosine\_similarity** 进行实现：

from **sklearn.metrics.pairwise** import **cosine\_similarity**

i = [1, 0, 0, 0]

j = [1, 0.5, 0.5, 0]

**consine\_similarity([a, b])**

3. 皮尔逊相关系数

皮尔逊相关系数的公式与余弦相似度的计算公式非常的类似，首先对于上述的余弦相似度的计算公式**写成求和的形式,**

相比余弦相似度，皮尔逊相关系数**通过使用用户的平均分对各独立评分进行修正，减小了用户评分偏置的影响。**

具体实现：

from **scipy.stats** import **pearsonr**

i = [1, 0, 0, 0]

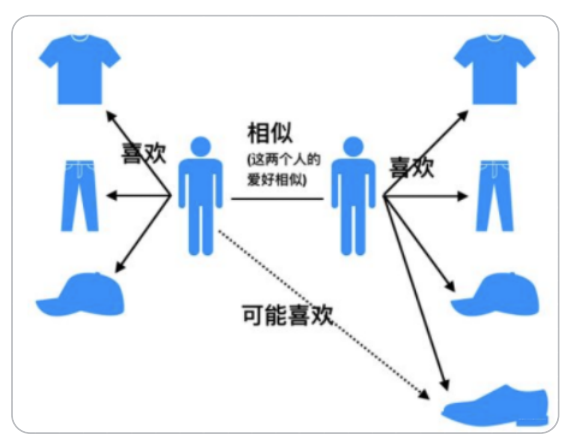
j = [1, 0.5, 0.5, 0]

**pearsonr(i, j)**

3.3 UserCF(基于用户的)协同过滤

思想：

当一个用户A需要个性化推荐的时候， 我们可以**先找 到和他有相似兴趣的其他用户**， **然后把那些用户喜欢的， 而用户A没有听说过的物品推荐给A。**

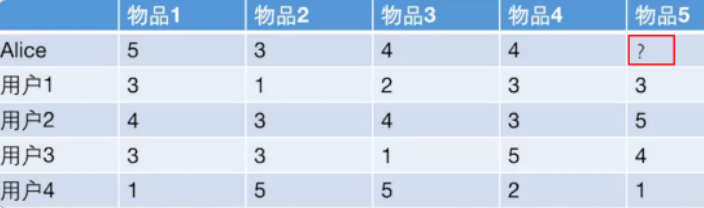
****

步骤：

1. **找到和目标用户兴趣相似的集合**

2. **找到这个集合中的用户喜欢的， 且目标用户没有听说过的物品推荐给目标用户。**

**举例：**



给用户推荐物品的过程可以**形象化为一个猜测用户对商品进行打分的任务**，上面表格里面是**5个用户对于5件物品**的一个**打分**情况，就**可以理解为**用户对物品的**喜欢程度**

1. 首先根据**前面这四个物品的这些打分情况**(或者说**已有**的用户**向量如用户1[3,1,2,3]用户2[4,3,4,3]……**）**计算一下Alice和用户1， 2， 3， 4的相似程度**， **找出**与Alice

**最相似的n个用户**

2. **根据这n个用户对物品5的评分**情况**和与Alice的相似程度**会**猜测**出**Alice对物品5的评分**， 如果**评分比较高**的话， 就把**物品5推荐**给用户Alice， **否则不推荐**。

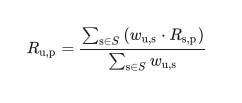
第一个步骤：**计算两个用户相似性**；

第二个步骤：**最终结果的预测**，

关于第一个步骤， 上面已经给出了计算两个用户相似性的方法， 这里不再过多赘述， 这里主要解决第二个问题， 如何产生最终结果的预测

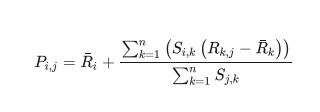
根据上面**计算**出**向量**之间的**相似程度**， 也就是可以计算出Alice和其他用户的相近程度， 这时候我们就可以**选出与Alice最相近的前n个用户**， **基于他们对物品5的评价猜测出Alice的打分**值：

利用**用户相似度**和**相似用户的评价加权**平均获得用户的评价预测

公式一：

用上例解释，w是某用户与Alice的相似度，R是该用户对某物品的评分。

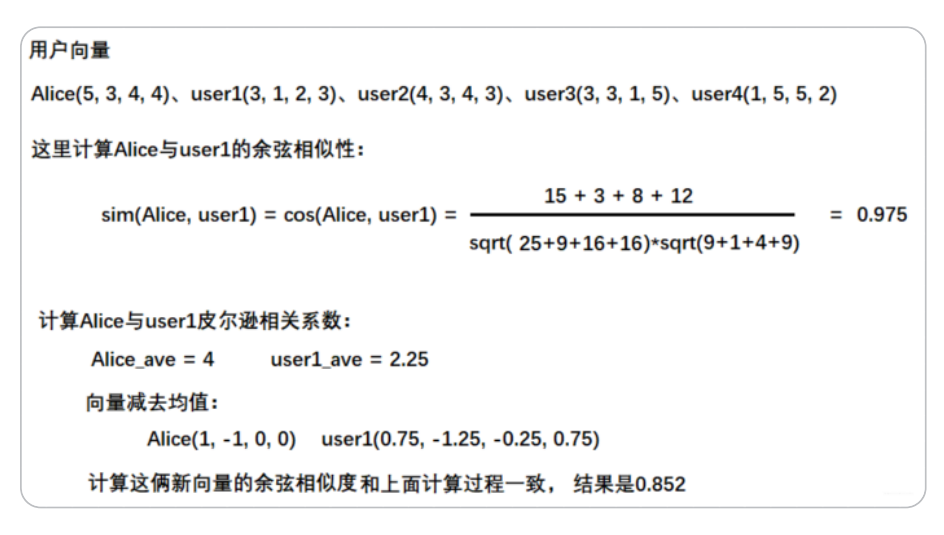
还有一种方法，更全面，依然是用户相似度作为权值，但公式后面R不单纯是其他用户对物品的评分， 而是**相似用户对该物品的评分与此用户的所有评分之差进行加权平均， 这时候考虑到了有的用户内心的评分标准不一的情况**， 即有的用户喜欢打高分， 有的用户喜欢打低分的情况。消除了不同用户之间的内心量纲差异。

公式二：

在**获得用户对不同物品的评价预测后**， **排序**，得到**推荐列表**。

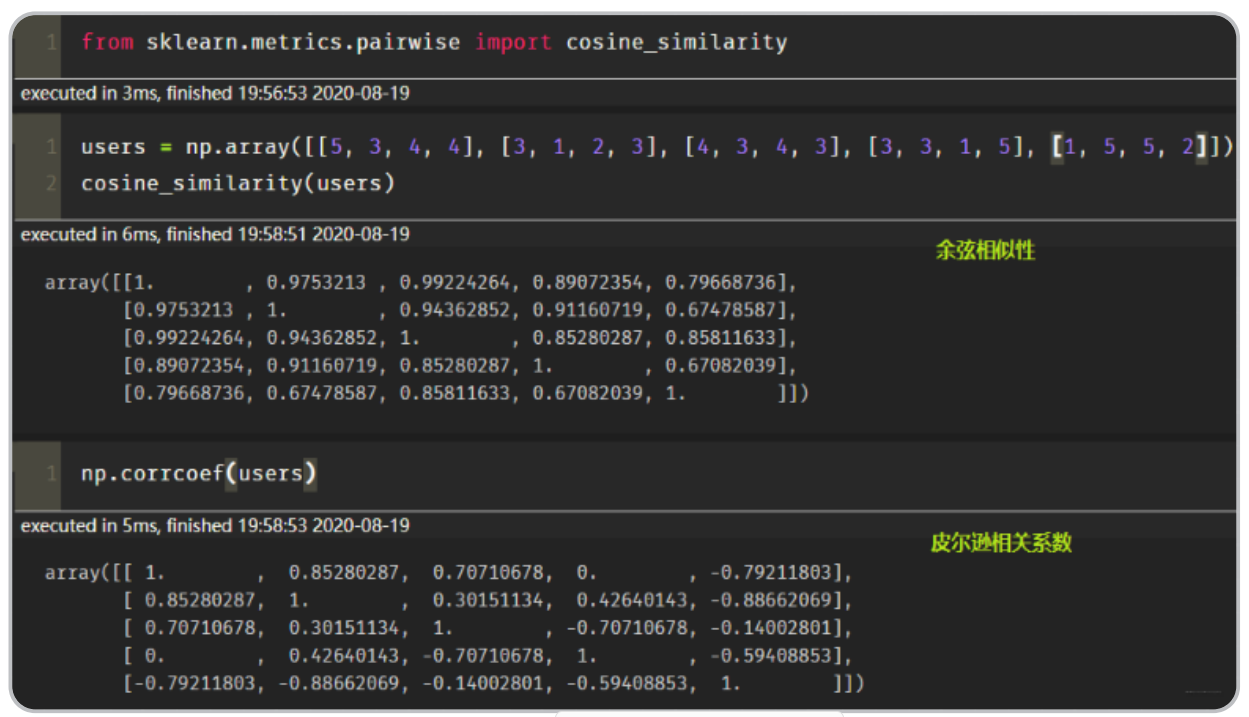
计算过程示例如下：

1. 计算Alice与其他用户的相似度（这里使用皮尔逊相关系数），如下图：



这里我们使用**皮尔逊相关系数**， 也就是Alice与用户1的相似度是0.85。同

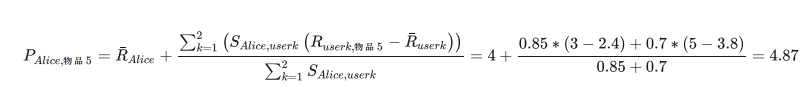
也可以使用**numpy的相似度函数from sklearn.metrics.pairwise import cosine\_similarity**得到用户的相似性矩阵,如下图：



从这里看出, Alice用户和用户2,用户3,用户4的相似度是0.7，0, -0.79。 所以如果n=2， 找到与Alice最相近的两个用户是用户1， 和Alice的相似度是0.85， 用户2， 和Alice相似度是0.7

2. 根据相似度用户计算Alice对物品5的最终得分

用户1对物品5的评分是3， 用户2对物品5的打分是5， 代入上面的计算公式， 可以计算出Alice对物品5的最终得分如下图：



3. 根据用户评分对用户进行推荐

这时候， 我们就得到了Alice对物品5的得分是4.87， 根据Alice的打分对物品排个序从大到小：

物品1 > 物品5 > 物品3 = 物品4 > 物品2

这时候，如果要向Alice推荐2款产品的话， 我们就可以**推荐物品1和物品5给Alice**

至此， 基于用户的协同过滤算法原理介绍完毕。

3.4 UserCF**编程实现**（基于物品的）

三块：1、计算用户相似性矩阵、2、得到前n个相似用户（排序）、3、计算最终得分。

1、

# 定义数据集， 也就是那个表格， 注意这里我们采用字典存放数据， 因为实际情况中数据是非常稀疏的， 很少有情况

是现在这样

def loadData():

items={'A': {1: 5, 2: 3, 3: 4, 4: 3, 5: 1},

'B': {1: 3, 2: 1, 3: 3, 4: 3, 5: 5},

'C': {1: 4, 2: 2, 3: 4, 4: 1, 5: 5},

'D': {1: 4, 2: 3, 3: 3, 4: 5, 5: 2},

'E': {2: 3, 3: 5, 4: 4, 5: 1}

}

users={1: {'A': 5, 'B': 3, 'C': 4, 'D': 4},

2: {'A': 3, 'B': 1, 'C': 2, 'D': 3, 'E': 3},

3: {'A': 4, 'B': 3, 'C': 4, 'D': 3, 'E': 5},

4: {'A': 3, 'B': 3, 'C': 1, 'D': 5, 'E': 4},

5: {'A': 1, 'B': 5, 'C': 5, 'D': 2, 'E': 1}

}

return items,users

items, users = loadData()

item\_df = pd.DataFrame(items).T

user\_df = pd.DataFrame(users).T

2、

"""计算用户相似性矩阵"""

similarity\_matrix = pd.DataFrame(np.zeros((len(users), len(users))), index=[1, 2, 3, 4, 5],

columns=[1, 2, 3, 4, 5])

# 遍历每条用户-物品评分数据

for userID in users:

for otheruserId in users:

vec\_user = []

vec\_otheruser = []

if userID != otheruserId:

for itemId in items: # 遍历物品-用户评分数据

itemRatings = items[itemId] # 这也是个字典 每条数据为所有用户对当前物品的评分

if userID in itemRatings and otheruserId in itemRatings: # 说明两个用户都对该物品评过分

vec\_user.append(itemRatings[userID])

vec\_otheruser.append(itemRatings[otheruserId])

# 这里可以获得相似性矩阵(共现矩阵)

similarity\_matrix[userID][otheruserId] = np.corrcoef(np.array(vec\_user),

np.array(vec\_otheruser))[0][1]

#similarity\_matrix[userID][otheruserId] = cosine\_similarity(np.array(vec\_user),

np.array(vec\_otheruser))[0][1]

3、

"""计算前n个相似的用户（排序）"""

n = 2

similarity\_users = similarity\_matrix[1].sort\_values(ascending=False)[:n].index.tolist() # [2, 3] 也就是用户1和用户2

4、

"""计算最终得分"""

base\_score = np.mean(np.array([value for value in users[1].values()]))

weighted\_scores = 0.

corr\_values\_sum = 0.

for user in similarity\_users: # [2, 3]

corr\_value = similarity\_matrix[1][user] # 两个用户之间的相似性

mean\_user\_score = np.mean(np.array([value for value in users[user].values()])) # 每个用户的打分平均值

weighted\_scores += corr\_value \* (users[user]['E']-mean\_user\_score) # 加权分数

corr\_values\_sum += corr\_value

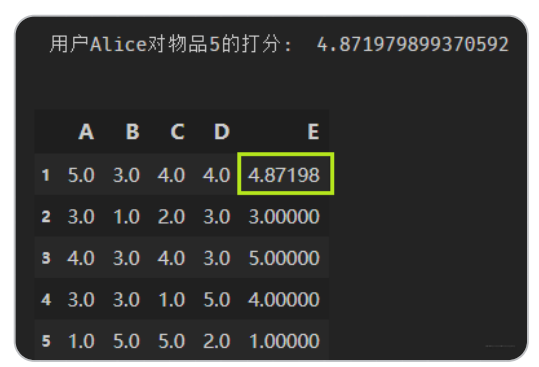
final\_scores = base\_score + weighted\_scores / corr\_values\_sum

print('用户Alice对物品5的打分: ', final\_scores)

user\_df.loc[1]['E'] = final\_scores

user\_df

结果：



至此， 我们就用代码完成了上面的小例子， 有了这个评分， 我们其实就可以对该用户做推荐了。 这其实就是微型版的

UserCF的工作过程了。

以上代码待敲，详见task2附带py文件

**3.5 UserCF优缺点**

问题：

1、数据稀疏性。

电子商务推荐系统一般有非常多的物品，用户可能买的其中不到1%的物品，不同用户之间买的物品重叠

性较低，导致算法无法找到一个用户的邻居，即偏好相似的用户。

这导致**UserCF**不适用于那些正反馈获取较困难的应用场景(如酒店预订， 大件商品购买等低频应用)

2. 算法扩展性。

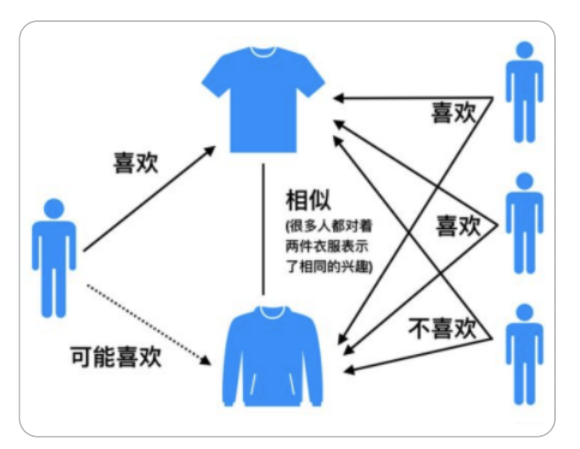
协同过滤需要维护用户相似度矩阵以便快速的找出Topn相似用户， 该矩阵的存储开销非常大，存储空间随着用户数量的增加而增加，不适合用户数据量大的情况使用。

所以，很多电商平台并没有采用这种算法， 而是采用了ItemCF算法实现最初的推荐系统。

**3.6 ItemCF(基于物品的)协同过滤**

思相：计算物品之间的相似性（基于用物的历史偏好数据，事先计算），推荐类似物品。

即，他不计算物品内容属性的相似度，是通过分析用户行为记录计算物品之间的相似度，比如物品a和c相似是因为喜欢a的用户大都喜欢c。

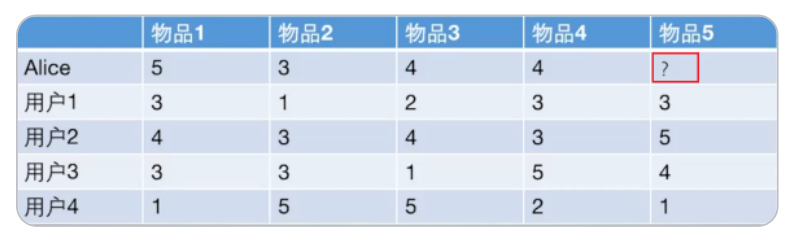


步骤：

1. 计算物品之间的相似度

2. 根据物品的相似度和用户的历史行为给用户生成推荐列表（购买了该商品的用户也经常购买的其他商品）

例：



想知道Alice对物品5打多少分， 基于物品的协同过滤算法会这么做：

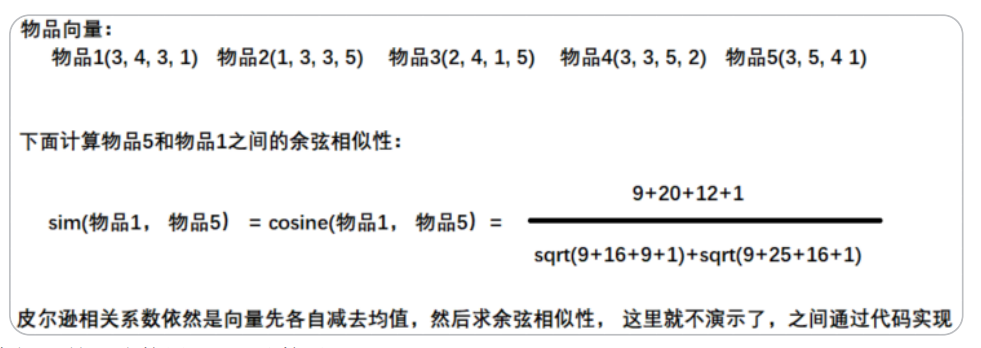
1. 首先计算一下物品5和物品1， 2， 3， 4之间的相似性(它们也是向量的形式， 每一列的值就是它们的向量表示， 因为ItemCF认为物品a和物品c具有很大的相似度是因为喜欢物品a的用户大都喜欢物品c， 所以就可以基于每个用户对该物品的打分或者说喜欢程度来向量化物品)

2. 找出与物品5最相近的n个物品

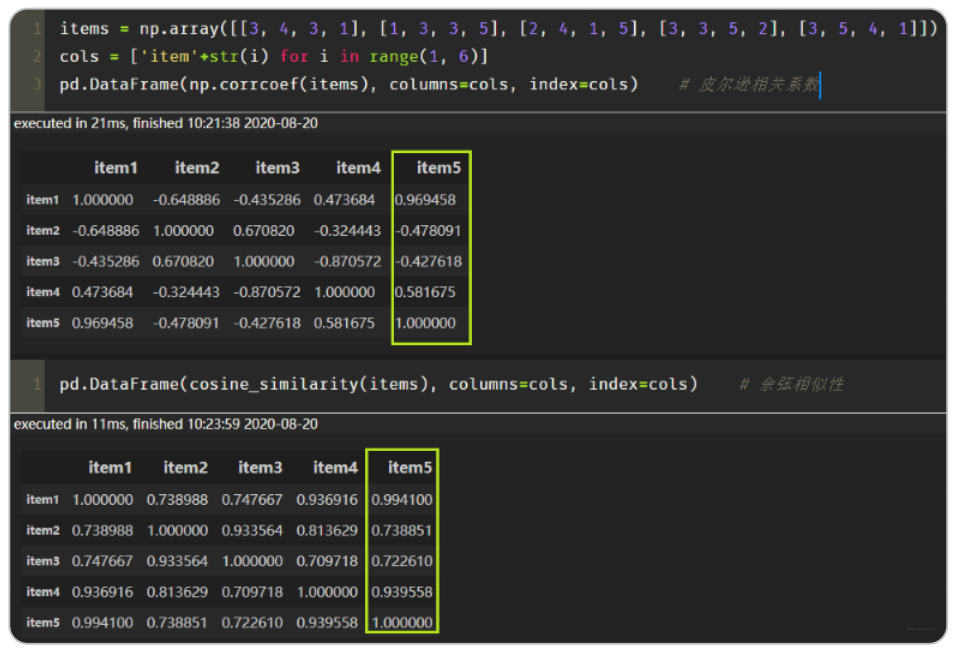
3. 根据Alice对最相近的n个物品的打分去计算对物品5的打分情况

计算示例

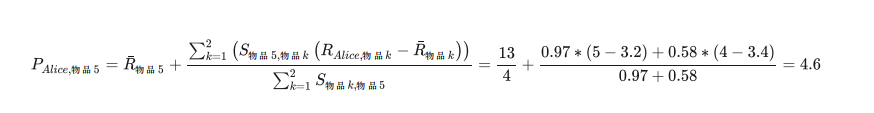
皮尔逊相关系数



Python np.corrcoef



根据皮尔逊相关系数， 可以找到与物品5最相似的2个物品是item1和item4(n=2)， 下面基于上面的公式计算最终得分：



编程实现：

1、

"""计算物品的相似矩阵"""

similarity\_matrix = pd.DataFrame(np.ones((len(items), len(items))), index=['A', 'B', 'C', 'D',

'E'], columns=['A', 'B', 'C', 'D', 'E'])

# 遍历每条物品-用户评分数据

1

2

3

4这里就是物品的相似度矩阵了， 大概长下面这个样子：

然后也是得到与物品**5**相似的前**n**个物品， 计算出最终得分来。

for itemId in items:

for otheritemId in items:

vec\_item = [] # 定义列表， 保存当前两个物品的向量值

vec\_otheritem = []

#userRagingPairCount = 0 # 两件物品均评过分的用户数

if itemId != otheritemId: # 物品不同

for userId in users: # 遍历用户-物品评分数据

userRatings = users[userId] # 每条数据为该用户对所有物品的评分， 这也是个字典

if itemId in userRatings and otheritemId in userRatings: # 用户对这两个物品都评过分

#userRagingPairCount += 1

vec\_item.append(userRatings[itemId])

vec\_otheritem.append(userRatings[otheritemId])

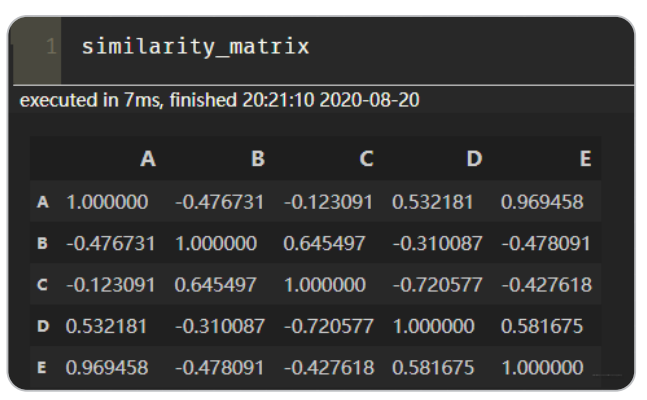
# 这里可以获得相似性矩阵(共现矩阵)

similarity\_matrix[itemId][otheritemId] = np.corrcoef(np.array(vec\_item),np.array(vec\_otheritem))[0][1]

#similarity\_matrix[itemId][otheritemId] = cosine\_similarity(np.array(vec\_item),

np.array(vec\_otheritem))[0][1]

相似度矩阵结果：



2、

"""得到与物品5相似的前n个物品（排序）"""

n = 2

similarity\_items = similarity\_matrix['E'].sort\_values(ascending=False)[:n].index.tolist()

# ['A', 'D']

3、

"""计算最终得分"""

base\_score = np.mean(np.array([value for value in items['E'].values()]))

weighted\_scores = 0.

corr\_values\_sum = 0.

for item in similarity\_items: # ['A', 'D']

corr\_value = similarity\_matrix['E'][item] # 两个物品之间的相似性

mean\_item\_score = np.mean(np.array([value for value in items[item].values()])) # 每个物品的打分平均值

weighted\_scores += corr\_value \* (users[1][item]-mean\_item\_score) # 加权分数

corr\_values\_sum += corr\_value

final\_scores = base\_score + weighted\_scores / corr\_values\_sum

print('用户Alice对物品5的打分: ', final\_scores)

user\_df.loc[1]['E'] = final\_scores

user\_df

结果如下：



代码待敲

**3.7 算法评估**

1、准确率

用户u推荐的物品列表与在测试集上真喜欢的物品集合交集/推荐物品集

这个意思再说， 在我推荐的所有物品中， 用户真正看的有多少， 这个考察的是我模型推荐的一个准确性。 查准率。

2. 召回率

用户真实购买或者看过的影片里面， 我模型真正预测出了多少，，这个考察的是模型推荐的一个全面性，查全率

用户u推荐的物品列表与在测试集上真喜欢的物品集合交集/真喜欢的物品集

3.覆盖率

覆盖率越高， 说明推荐算法越能将长尾中的物品推荐给用户。

该覆盖率表示最终的推荐列表中包含多大比例的物品。如果所有物品都被给推荐给至少一个用户， 那么覆盖率是100%。

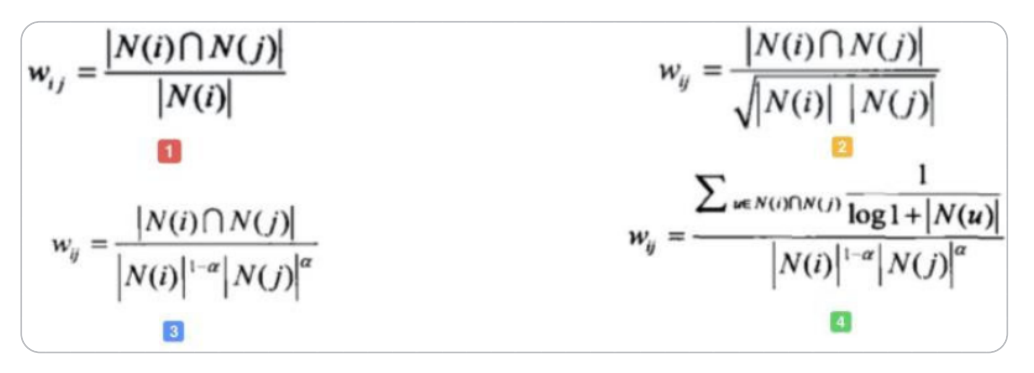
4、新颖度

推荐列表中物品的平均流行度度量推荐结果的新颖度。

很热门， 说明推荐的新颖度较低。

由于物品的流行度分布呈长尾分布， 所以为了流行度的平均值更加稳定， 在计算平均流行度时对每个物品的流行度取对数。

**3.8 协同过滤算法的权重改进**



2中加N,对热门物品的惩罚

3 对极热门商品加大惩罚用a参数

4 对活跃用户的惩罚 ：一个活跃用户（专门做刷单）可能买了非常多的物品

**3.9 协同过滤算法的问题分析**

泛化能力弱， 即协同过滤无法将两个物品相似的信息推广到其他物品的相似性上

热门物品具有很强的头部效应， 容易跟大量物品产生相似， 而尾部物品由于特征向量稀疏， 导致很少被推荐。

协同过滤的天然缺陷：推荐系统头部效应明显， 处理稀疏向量的能力弱

为了解决这个问题， 同时增加模型的泛化能力，2006年，**矩阵分解技术(Matrix Factorization,MF)**被提出， 该方法在协同过滤共现矩阵的基础上， 使用更稠密的隐向量表示用户和物品， 挖掘用户和物品的隐含兴趣和隐含特征， 在一定程度上弥补协同过滤模型处理稀疏矩阵能力不足的问题。 具

**3.10 课后思考**

1.什么时候使用**UserCF**，什么时候使用**ItemCF**？为什么？

UserCF：

由于是基于用户相似度进行推荐， 所以具备更强的社交特性， 这样的特点非常适于**用户少， 物品多， 时效性较强的场合**， 比如新闻推荐场景， 因为新闻本身兴趣点分散， 相比用户对不同新闻的兴趣偏好， 新闻的及时性，热点性往往更加重要， 所以正好适用于发现热点，跟踪热点的趋势。 另外还具有推荐新信息的能力， 更有可能发现惊喜, 因为看的是人与人的相似性, 推出来的结果可能更有惊喜，可以发现用户潜在但自己尚未察觉的兴趣爱好。

ItemCF：

这个更适用于兴趣变化较为稳定的应用， 更接近于个性化的推荐， 适合**物品少，用户多，用户兴趣固定持久， 物品更新速度不是太快的场合**， 比如推荐艺术品， 音乐， 电影。

2.协同过滤在计算上有什么缺点？有什么比较好的思路可以解决（缓解）？

较差的稀疏向量处理能力 ，泛化能力弱

尾部物品由于特征向量稀疏，相似物品少，导致很少被推荐。

解决方法：矩阵分解技术**(Matrix Factorization,MF**)

该方法在协同过滤共现矩阵的基础上， 使用更稠密的隐向量表示用户和物品， 挖掘用户和物品的隐含兴趣和隐含特征。

**3.**上面介绍的相似度计算方法有什么优劣之处？

余弦相似度cosine相似度比较常用的，效果也不会太差，但用户乱评分不能避免。

而实际上， 如果看这个商品喜好的一个趋势的话，不一样。

所以对于这种用户评分偏置的情况， 余弦相似度就不是那么好了， 可以考虑使用皮尔逊相关系数。

4.协同过滤还存在其他什么缺陷？有什么比较好的思路可以解决（缓解）？

协同过滤的特点就是完全没有利用到物品本身或者是用户自身的属性， 仅仅利用了用户与物品的交互信息就可以实现推荐，比较简单高效。

但这也是它的短板所在，无法有效的引入用户年龄， 性别，商品描述，商品分类，当前时间，地点等一系列用户特征、物品特征和上下文特征， 这就造成了有效信息的遗漏，不能充分利用其它特征数据。

为了解决这个问题， 在推荐模型中**引用更多的特征**，推荐系统慢慢的**从以协同过滤为核心**到了**以逻辑回归模型为核心**， 提出了能够综合不同类型特征的机器学习模型。

