

# AI上推荐 之 协同过滤

原创

翻滚的小@强

于 2020-08-21 11:30:02 发布

12045

收藏 227

版权

分类专栏：

推荐系统学习笔记


文章标签：

推荐系统

协同过滤

UserCF

ItemCF

 推荐系统学习笔记

专栏收录该内容

528 订阅

23 篇文章

订阅专栏

## 1. 前言

随着信息技术和互联网的发展， 我们已经步入了一个信息过载的时代， 这个时代， 无论是信息消费者还是信息生产者都遇到了很大的挑战：

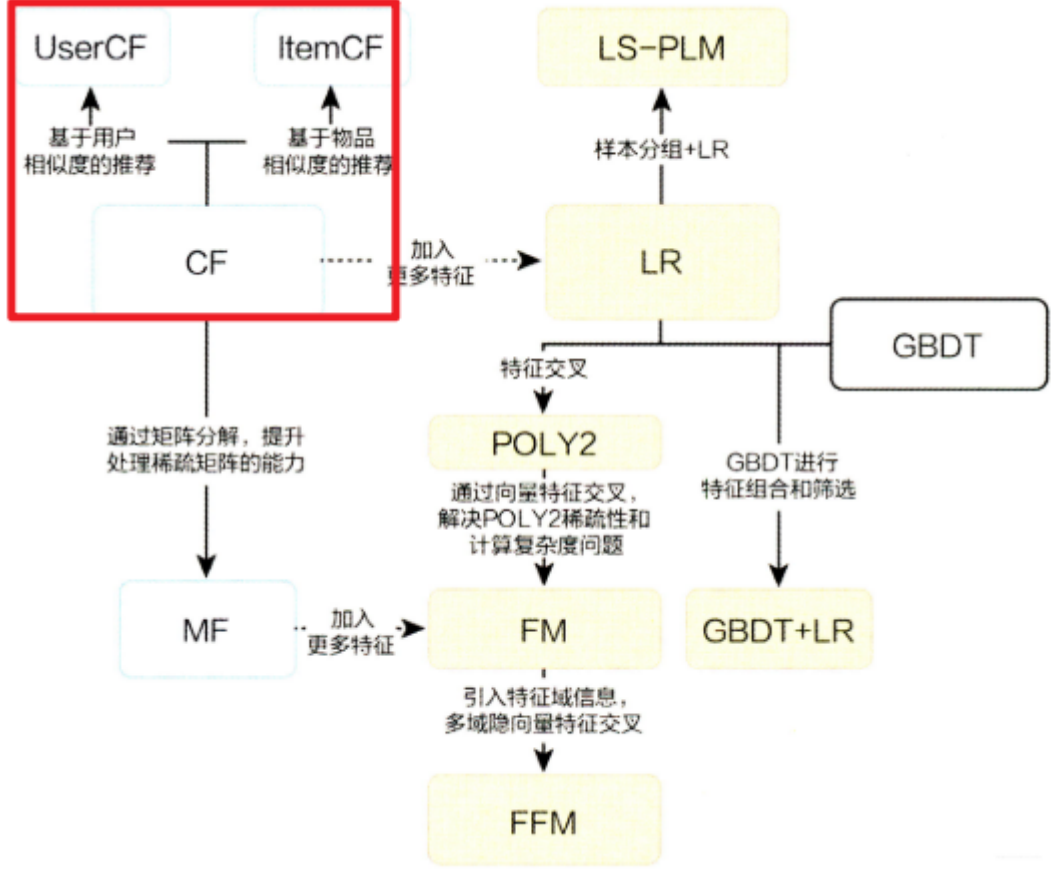
- 信息消费者： 如何从大量的信息中找到自己感兴趣的信息？
- 信息生产者： 如何让自己生产的信息脱颖而出， 受到广大用户的关注？

为了解决这个矛盾， 推荐系统应时而生， 并飞速前进， 在用户和信息之间架起了一道桥梁， 一方面帮助用户发现对自己有价值的信息， 一方面让信息能够展现在对它感兴趣的用户前面。 推荐系统近几年有了深度学习的助推发展之势迅猛， 从前深度学习的传统推荐模型( [协同过滤](#) )， 矩阵分解， LR, FM, FFM, GBDT)到深度学习的浪潮之巅(DNN, Deep Crossing, DIN, DIEN, Wide&Deep, Deep&Cross, DeepFM, AFM, NFM, PNN, FNN, DRN), 现在正无时无刻不影响着大众的生活。

推荐系统通过分析用户的历史行为给用户的兴趣建模， 从而主动给用户推荐给能够满足他们兴趣和需求的信息， 能够真正的“懂你”。 想上网购物的时候， 推荐系统在帮我们挑选商品， 想看资讯的时候， 推荐系统为我们准备了感兴趣的新闻， 想学习充电的时候， 推荐系统为我们提供最合适的课程， 想消遣放松的时候， 推荐系统为我们奉上欲罢不能的短视频..., 所以当我们淹没在信息的海洋时， 推荐系统正在拨开一层层波浪， 为我们追寻多姿多彩的生活！

这段时间刚好开始学习推荐系统， 通过王喆老师的《深度学习推荐系统》已经梳理好了知识体系， 了解了当前推荐系统领域各种主流模型架构和技术。 所以接下来的时间就开始对这棵大树开枝散叶， 对每一块知识点进行学习总结。 所以接下来一块目睹推荐系统的风采吧！

这次整理重点放在推荐系统的模型方面， 先从传统推荐模型开始， 然后到深度学习模型。 传统模型的演化关系拿书上的张图片， 便于梳理传统推荐模型的进化关系脉络， 对知识有个宏观的把握：



今天是第一篇， 我们从最经典的协同过滤算法开始， 虽然这个离我们比较久远， 但它是对业界影响力最大， 应用最广泛的一种经典模型， 从1992年一直延续至今， 尽管现在协同过滤差不多都已经融入到了深度学习， 但模型的基本原理依然还是基于经典协同过滤的思路， 或者是在协同过滤的基础上进行演化， 所以这个算法模型依然有种“宝刀未老”的感觉， 掌握和学习非常有必要。

所谓协同过滤(Collaborative Filtering)算法， 基本思想是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）。目前应用比较广泛的协同过滤算法是基于邻域的方法， 而这种方法主要有下面两种算法：

- 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
- 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

所以本篇文章的核心就是这两个算法， 所以分成两块， 每一块都是先对理论部分进行补充， 包括算法的工作原理和计算方式， 相似度的度量方式等。 然后通过一个例子进行解释加深理解， 最后基于学习的理论知识进行代码实战， 完成一个推荐小任务， 这样可以对协同过滤的思想有个更深的理解。

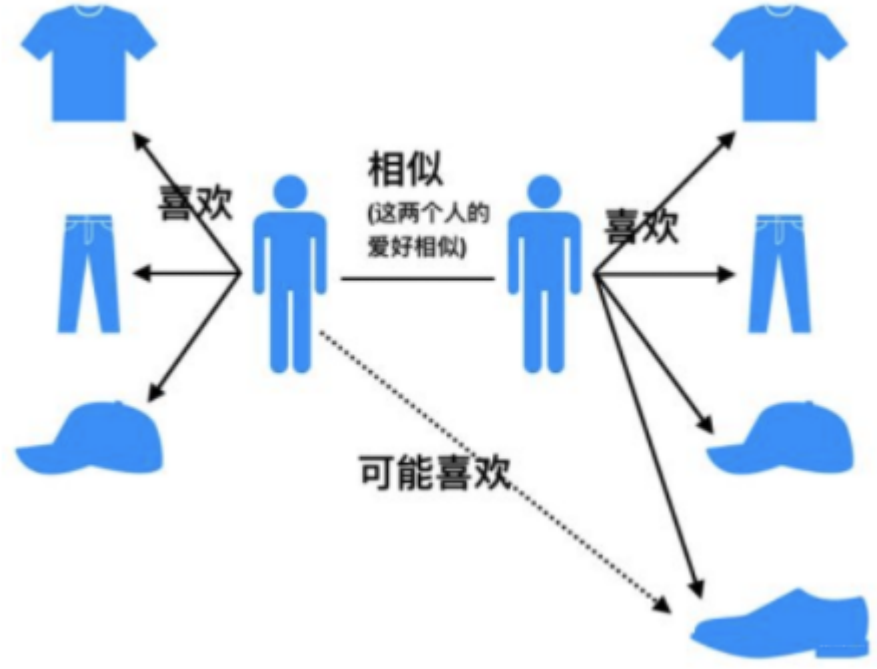
大纲如下：

- 基于用户的协同协同过滤
- 基于物品的协同过滤算法
- 应用场景及存在问题分析
- 总结

Ok, let's go!

## 2. 基于用户的协同过滤

基于用户的协同过滤(UserCF)可以追溯到1993年， 可以说是非常早的一种算法了， 这种算法的思想其实比较简单， 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。



所以基于用户的协同过滤算法主要包括两个步骤：

- 找到和目标用户兴趣相似的集合
- 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用户。

如果这样说比较抽象的话， 我们可以看一个例子， 了解一下给用户推荐物荐到底是怎么推荐的， 看下面这个表格：

	物品1	物品2	物品3	物品4	物品5
Alice	5	3	4	4	?
用户1	3	1	2	3	3
用户2	4	3	4	3	5
用户3	3	3	1	5	4
用户4	1	5	5	2	1

其实， 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(这里多说一句， 这种表格在实际情况中就是根据用户的行为进行统计出来的， 比如用户购买了某个物品， 那直接量化为5分， 用户收藏了某个物品， 量化为4分， 用户看某个物品很久量化为3分等， 通过这样的量化就相当于把每个用户对物品的行为刻画成了向量的形式， 我们就可以计算相似程度了)， 我们的任务是判断到底该不该把物品5推荐给用户Alice呢？





如果是基于用户的协同过滤算法， 根据上面的算法步骤， 其实它会这么做：

1. 首先根据前面的这些打分情况(或者说已有的用户向量) 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。

所以这个过程相信很容易理解了吧， 下面我们就尝试解决这个问题， 但是在解决之前， 还得先补充两个知识点： 第一个就是用户之间的相似性怎么衡量？ 第二个就是选出了topn个与Alice最相似的用户来之后， 如果根据他们的相似程度和对物品5的分数计算Alice对物品5的分数？

### 2.1 计算两个向量之间的相似程度

计算相似度需要根据特点的不同选择不同的相似度计算方法， 比较常用的有下面几种, 更多的关于向量的相似性度量，可以参考这篇文章：

1. 杰卡德(Jaccard)相似系数  
这个是衡量两个集合的相似度一种指标。两个集合A和B的交集元素在A，B的并集中所占的比例，称为两个集合的杰卡德相似系数，用符号J(A,B)表示。

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}$$

在项亮老师的《推荐系统实战》中， 计算相似度出现过这个公式。所以在这里简单的说一下。

2. 余弦相似度  
余弦相似度这个很熟悉了吧。 它衡量了用户向量i和j之间的向量夹角的大小， 夹角越小， 说明相似度越大， 两个用户越相似。 公式如下：

$$\text{sim}(i,j)=\cos(i,j)=\frac{i\cdot j}{\|i\|\cdot\|j\|}$$

这里面是向量表示的， 如果不太明白的话， 可以换成具体的数值表示， 假设两个用户的向量是n维的， 分别是 $i(x_{11},x_{12},\dots,x_{1n})$ 、 $j(x_{21},x_{22},\dots,x_{2n})$ ， 那么余弦相似度为：

$$\cos(\theta)=\frac{\sum_{k=1}^n x_{1k}x_{2k}}{\sqrt{\sum_{k=1}^n x_{1k}^2}\sqrt{\sum_{k=1}^n x_{2k}^2}}$$

这个在具体实现的时候， 可以使用 `cosine_similarity` 进行实现：

```
1 from sklearn.metrics.pairwise import cosine_similarity
2 i = [1, 0, 0, 0]
3 j = [1, 0.5, 0.5, 0]
4 consine_similarity([a, b])
```

cosine相似度还是比较常用的， 一般效果也不会太差， 但是对于评分数据不规范的时候， 也就是说， 存在有的用户喜欢打高分， 有的用户喜欢打低分情况的时候， 有的用户喜欢乱打分的情况， 这时候consine相似度算出来的结果可能就不是那么准确了， 比如下面这种情况：

	x	y	z
d	4	4	5
e	1	1	2
f	4	1	5

这时候， 如果用余弦相似度进行计算， 会发现用户d和用户f比较相似， 而实际上， 如果看这个商品喜好的一个趋势的话， 其实d和e比较相近， 只不过e比较喜欢打低分， d比较喜欢打高分。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， 可以考虑使用下面的皮尔逊相关系数。

3. 皮尔逊相关系数  
这个也是非常常用的一种计算相似度的一种方式， 相比余弦相似度， 皮尔逊相关系数通过使用用户平均分对个独立评分进行修正， 减少了用户评分偏置的影响。简单的说， 其实pearson做的就是将两个向量都减去他们的均值， 然后再计算consine值。用pearson来计算用户相似进行推荐的话， 效果还是好于consine的。公式如下：

$$\text{sim}(i,j)=\frac{\sum_{p\in P} (R_{i,p}-\bar{R}_i)(R_{j,p}-\bar{R}_j)}{\sqrt{\sum_{p\in P} (R_{i,p}-\bar{R}_i)^2}\sqrt{\sum_{p\in P} (R_{j,p}-\bar{R}_j)^2}}$$

这个式子里面其实就是每个向量先减去了它的平均值， 然后在计算余弦相似度， 其中 $R_{i,p}$ 代表用户i对物品p的评分。 $\bar{R}_i$ 代表用户i对所有物品的平均评分,P代表所有物品的集合， p表示某个物品。待会计算上面例子的时候就很容易明白了。具体实现， 我们也是可以调包， 这个计算方式很多， 下面是其中的一种：

```
1 from scipy.stats import pearsonr
2
3 i = [1, 0, 0, 0]
4 j = [1, 0.5, 0.5, 0]
5 pearsonr(i, j)
```

常用的就是这几种， 其余方法， 例如欧式距离， 曼哈顿距离， 马氏距离等参考上面给出的那篇文章， 在协同过滤这里衡量相似性往往喜欢用余弦相似度或者皮尔逊相关系数。第一个问题解决。

谈到距离， 这里再补充一个问题为什么在**一些场景中要使用余弦相似度而不是欧式距离呢**？

有些时候， 我们其实并不关心两个向量的绝对大小， 而是关注向量之间的夹角大小， 此时就要用余弦相似度。比如， 当一对文本相似度的长度差距很大， 但是内容相似时， 如果是使用词频或者词向量作为特征， 它们在特征空间中的欧式距离非常大， 而使用余弦相似度时， 可能很小， 相似度高， 这正是我们想要的。此外， 在文本， 图像， 视频领域， 研究对象的特征维度往往很高， 余弦相似度在高维情况仍然保持“相同时为1， 正交时为0， 相反时-1”的性质， 而欧式距离的数值则受维度的影响， 范围不固定， 并且含义也比较模糊。不过， 向量的模长归一化了之后， 就成了皮尔逊相关系数了， 这时候欧式距离与余弦相似度就有着单调关系了。

总体的来说， 欧式距离体现数值上的绝对差异， 而余弦距离体现方向上的相对差异。具体使用中， 看需求决定。

- 如果要统计两部剧的用户观看行为， 用户A的观看向量(0,1)， 用户B为(1,0), 此时二者的余弦距离很大， 而欧式距离很小。我们分析两个用户对于不同视频的偏好， 更关注相对差异， 显然应当用余弦距离。
- 而当我们分析用户活跃度， 以登录次数和平均观看时长作为特征时， 余弦距离会认为(1,10)和(10,100)两个用户距离很近， 但显然这两个用户活跃度是有着极大差异的。此时我们关注的是数值绝对差异， 应当使用欧式距离。

### 2.2 最终结果的预测

根据上面的几种方法， 我们可以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， 那么是怎么计算的呢？

这里常用的方式之一是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， 用下面式子表示：

$$R_{u,p}=\frac{\sum_{s\in S}(w_{u,s}\cdot R_{s,p})}{\sum_{s\in S}w_{u,s}}$$

这个式子里面， 权重 $w_{u,s}$ 是用户u和用户s的相似度，  $R_{s,p}$ 是用户s对物品p的评分。

还有一种方式如下， 这种方式考虑的更加全面， 依然是用户相似度作为权值， 但后面不单纯的是其他用户对物品的评分， 而是该物品的评分与此用户的所有评分的差值进行加权平均， 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。

$$P_{i,j}=\bar{R}_i+\frac{\sum_{k=1}^n(S_{i,k}(R_{k,j}-\bar{R}_k))}{\sum_{k=1}^nS_{j,k}}$$



所以这一种计算方式更为推荐。下面的计算将使用这个方式。这里的 $S_{j,k}$ 是用户i和用户k的相似度， $w_{u,s}$ 代表的是一个意思。

在获得用户u对不同物品的评价预测后，最终的推荐列表根据预测评分进行排序得到。至此，基于用户的协同过滤算法的推荐过程完成。

### 2.3 拿一个例子来具体看一下

下面我们解决上面的问题，把图拿过来：

	物品1	物品2	物品3	物品4	物品5
Alice	5	3	4	4	?
用户1	3	1	2	3	3
用户2	4	3	4	3	5
用户3	3	3	1	5	4
用户4	1	5	5	2	1

猜测Alice对物品5的得分：

计算Alice与其他用户的相似度（这里使用皮尔逊相关系数）

用户向量

Alice(5, 3, 4, 4)、user1(3, 1, 2, 3)、user2(4, 3, 4, 3)、user3(3, 3, 1, 5)、user4(1, 5, 5, 2)

这里计算Alice与user1的余弦相似性：

$$\text{sim}(\text{Alice}, \text{user1}) = \cos(\text{Alice}, \text{user1}) = \frac{15 + 3 + 8 + 12}{\sqrt{(25 + 9 + 16 + 16)} * \sqrt{(9 + 1 + 4 + 9)}} = 0.975$$

计算Alice与user1皮尔逊相关系数：

Alice\_ave = 4      user1\_ave = 2.25

向量减去均值：

Alice(1, -1, 0, 0)    user1(0.75, -1.25, -0.25, 0.75)

计算这俩新向量的余弦相似度和上面计算过程一致，结果是0.852

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。同样的方式，我们就可以计算与其他用户的相似度，这里可以使用numpy的相似度函数得到用户的相似性矩阵：

```
1 from sklearn.metrics.pairwise import cosine_similarity

executed in 3ms, finished 19:56:53 2020-08-19

1 users = np.array([[5, 3, 4, 4], [3, 1, 2, 3], [4, 3, 4, 3], [3, 3, 1, 5], [1, 5, 5, 2]])
2 cosine_similarity(users)

executed in 6ms, finished 19:58:51 2020-08-19

                                余弦相似性
array([[1.          , 0.9753213 , 0.99224264, 0.89072354, 0.79668736],
       [0.9753213 , 1.          , 0.94362852, 0.91160719, 0.67478587],
       [0.99224264, 0.94362852, 1.          , 0.85280287, 0.85811633],
       [0.89072354, 0.91160719, 0.85280287, 1.          , 0.67082039],
       [0.79668736, 0.67478587, 0.85811633, 0.67082039, 1.          ]])

1 np.corrcoef(users)

executed in 5ms, finished 19:58:53 2020-08-19

                                皮尔逊相关系数
array([[ 1.          , 0.85280287, 0.70710678, 0.          , -0.79211803],
       [ 0.85280287, 1.          , 0.30151134, 0.42640143, -0.88662069],
       [ 0.70710678, 0.30151134, 1.          , -0.70710678, -0.14002801],
       [ 0.          , 0.42640143, -0.70710678, 1.          , -0.59408853],
       [-0.79211803, -0.88662069, -0.14002801, -0.59408853, 1.          ]])
```

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的打分为5，那么根据上面的计算公式，可以计算出Alice对物品5的最终得分是

$$P_{\text{Alice,物品5}} = \bar{R}_{\text{Alice}} + \frac{\sum_{k=1}^2 (S_{\text{Alice,userk}} (R_{\text{userk,物品5}} - \bar{R}_{\text{userk}}))}{\sum_{k=1}^2 S_{\text{Alice,userk}}} = 4 + \frac{0.85 * (3 - 2.4) + 0}{0.85 + 0}$$

#### 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。

### 2.4 编程实现

这里简单的通过编程实现上面的案例计算吧。梳理一下上面的过程其实就是三步：计算用户相似性矩阵、得到前n个相似用户、计算最终得分。

所以我们下面的程序也是分为这三步：

#### 1. 首先，先把数据表给建立起来

这里我采用了字典的方式，之所以没有用pandas，是因为上面举得这个例子其实是个个例，在真实情况中，我们知道，用户对物品的打分情况并不会这么完整，会存在大量的空值，所以矩阵会很稀疏，这时候用DataFrame，会有大量的NaN。故这里用字典的形式存储。用两个字典，第一个字典是物品-用户的评分映射，键是物品1-5，用A-E来表示，每一个值又是一个字典，表示的是每个用户对该物品的打分。第二个字典是用户-物品的评分映射，键是上面的五个用户，用1-5表示，值是該用户对每个物品的打分。

```
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，
2 def loadData():
3     items={'A': {1: 5, 2: 3, 3: 3, 4: 3, 5: 1},
4             'B': {1: 3, 2: 1, 3: 3, 4: 3, 5: 5},
5             'C': {1: 4, 2: 2, 3: 3, 4: 4, 5: 5},
6             'D': {1: 4, 2: 3, 3: 3, 4: 5, 5: 2},
7             'E': {2: 3, 3: 5, 4: 4, 5: 1}
8         }
9     users={1: {'A': 5, 'B': 3, 'C': 4, 'D': 4},
10            2: {'A': 3, 'B': 1, 'C': 2, 'D': 3, 'E': 3},
11            3: {'A': 4, 'B': 3, 'C': 4, 'D': 3, 'E': 5}}
```

#### 2. 计算用户相似性矩阵

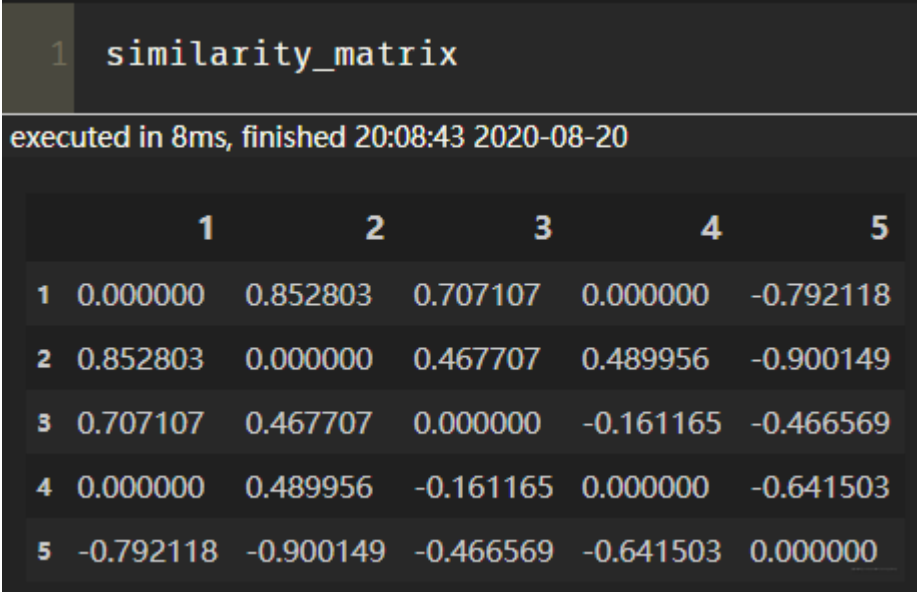
这个是一个共现矩阵，5\*5，行代表每个用户，列代表每个用户，值代表用户和用户的相关性，这里的思路是这样，因为要求用户和用户两两的相关性，所以需要双层循环遍历用户-物品评分数据，当不是同一个用户的时候，我们要去遍历物品-用户评分数据，在里面去找这两个用户同时对该物品评过分的数

```
1 """计算用户相似性矩阵"""
2 similarity_matrix = pd.DataFrame(np.zeros((len(users), len(users))), index=[1, 2, 3, 4, 5], columns=[1, 2, 3, 4, 5])
3
4 # 遍历每条用户-物品评分数据
5 for userID in users:
6     for otheruserID in users:
7         vec_user = []
8         vec_otheruser = []
9         if userID != otheruserID:
10             for itemId in items: # 遍历物品-用户评分数据
11                 itemRating = items[itemId][userID] # 这个是个字典，每个数据字典右有E
```





这里的similarity\_matrix就是我们的用户相似性矩阵， 张下面这样：



有了相似性矩阵， 我们就可以得到与Alice最相关的前n个用户。

3. 计算前n个相似的用户

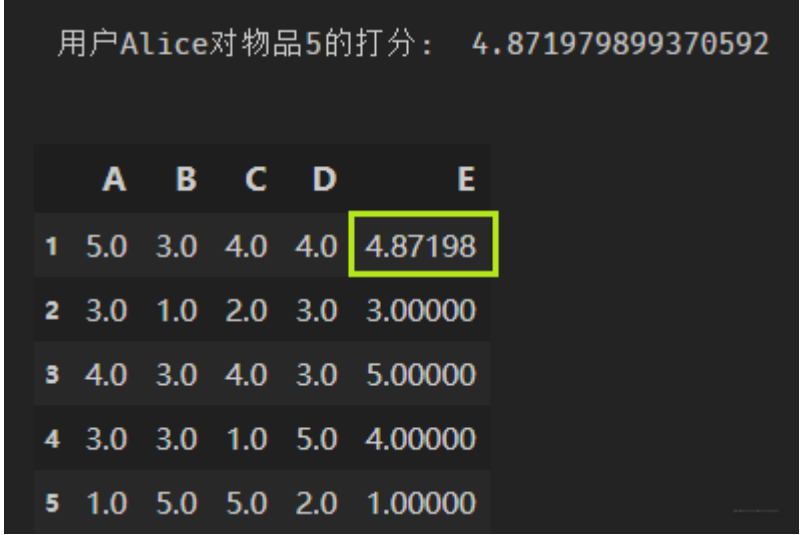
```
1 """计算前n个相似的用户"""
2 n = 2
3 similarity_users = similarity_matrix[1].sort_values(ascending=False)[:n].index.
```

4. 计算最终得分

这里就是上面的那个公式了。

```
1 """计算最终得分"""
2 base_score = np.mean(np.array([value for value in users[1].values()]))
3 weighted_scores = 0.
4 corr_values_sum = 0.
5 for user in similarity_users: # [2, 3]
6     corr_value = similarity_matrix[1][user] # 两个用户之间的相似性
7     mean_user_score = np.mean(np.array([value for value in users[user].values()]))
8     weighted_scores += corr_value * (users[user]['E']-mean_user_score) # 加
9     corr_values_sum += corr_value
10 final_scores = base_score + weighted_scores / corr_values_sum
11 print('用户Alice对物品5的打分: ', final_scores)
12 user_df.loc[1]['E'] = final_scores
13 user_df
```

结果如下：



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。

这里其实本来想把项亮老师推荐系统实践里面的那个UserCF例子放过来， 但是那个可能不是太好理解， 因为那里做的是TopN的推荐， 那里的任务是预测用户会不会对某个物品进行评分， 而不是预测用户在准备为物品打分的情况下打多少分。所以那一个类似于找到用户可能打分的物品然后对用户推荐， 而这里是预测出用户对物品打多少分， 然后根据分数推荐物品。这俩还是有区别的。所以那一个实战的例子放在了GitHub， 后面会给出链接。

2.5 优缺点

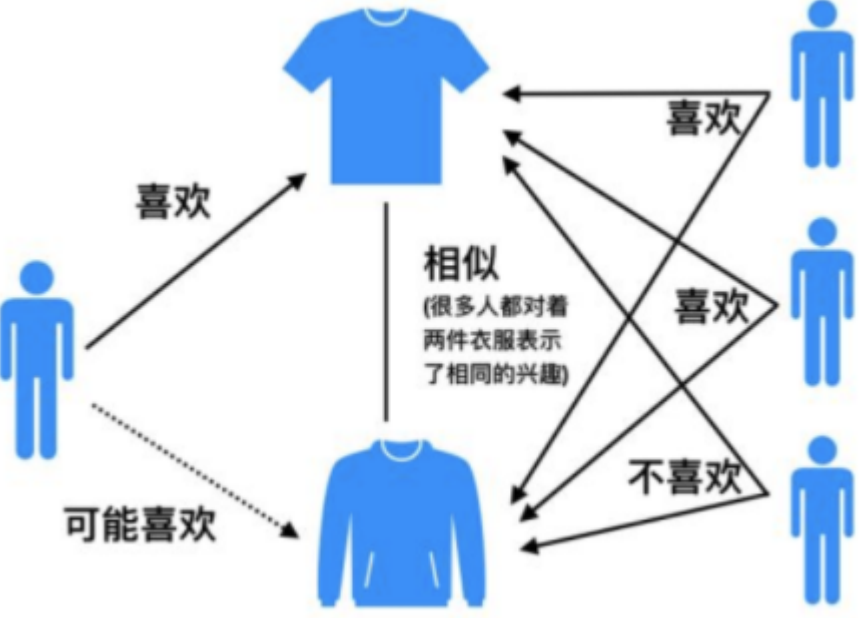
User-based算法存在两个重大问题：

1. 数据稀疏性。  
一个大型的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。  
基于用户的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**。

由于UserCF技术上的两点缺陷， 导致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. 基于物品的协同过滤

基于物品的协同过滤(ItemCF)的基本思想是预先根据所有用户的历史偏好数据计算物品之间的相似性，然后把与用户喜欢的物品相类似的物品推荐给用户。比如物品a和c非常相似，因为喜欢a的用户同时也喜欢c，而用户A喜欢a，所以把c推荐给用户A。**ItemCF算法并不利用物品的内容属性计算物品之间的相似度， 主要通过分析用户的行为记录计算物品之间的相似度， 该算法认为， 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。**



基于物品的协同过滤算法主要分为两步：

- 计算物品之间的相似度
- 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）

3.1 还是前面的例子

基于物品的协同过滤算法和基于用户的协同过滤算法很像， 所以我们这里直接还是拿上面Alice的那个例子来看。

	物品1	物品2	物品3	物品4	物品5
Alice	5	3	4	4	?
用户1	3	1	2	3	3
用户2	4	3	4	3	5
用户3	3	3	1	5	4
用户4	1	5	5	2	1

如果想知道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



举报

下面我们就可以具体计算一下， 首先是步骤1：

物品向量：  
物品1(3, 4, 3, 1) 物品2(1, 3, 3, 5) 物品3(2, 4, 1, 5) 物品4(3, 3, 5, 2) 物品5(3, 5, 4, 1)

下面计算物品5和物品1之间的余弦相似性：

sim(物品1, 物品5) = cosine(物品1, 物品5) = 
$$\frac{9+20+12+1}{\sqrt{9+16+9+1}+\sqrt{9+25+16+1}}$$

皮尔逊相关系数依然是向量先各自减去均值，然后求余弦相似性， 这里就不演示了， 之间通过代码实现

由于计算比较麻烦， 这里直接用python计算了：

```
1 items = np.array([[3, 4, 3, 1], [1, 3, 3, 5], [2, 4, 1, 5], [3, 3, 5, 2], [3, 5, 4, 1]])
2 cols = ['item'+str(i) for i in range(1, 6)]
3 pd.DataFrame(np.corrcoef(items), columns=cols, index=cols) # 皮尔逊相关系数
```

executed in 21ms, finished 10:21:38 2020-08-20

	item1	item2	item3	item4	item5
item1	1.000000	-0.648886	-0.435286	0.473684	0.969458
item2	-0.648886	1.000000	0.670820	-0.324443	-0.478091
item3	-0.435286	0.670820	1.000000	-0.870572	-0.427618
item4	0.473684	-0.324443	-0.870572	1.000000	0.581675
item5	0.969458	-0.478091	-0.427618	0.581675	1.000000

```
1 pd.DataFrame(cosine_similarity(items), columns=cols, index=cols) # 余弦相似性
```

executed in 11ms, finished 10:23:59 2020-08-20

	item1	item2	item3	item4	item5
item1	1.000000	0.738988	0.747667	0.936916	0.994100
item2	0.738988	1.000000	0.933564	0.813629	0.738851
item3	0.747667	0.933564	1.000000	0.709718	0.722610
item4	0.936916	0.813629	0.709718	1.000000	0.939558
item5	0.994100	0.738851	0.722610	0.939558	1.000000

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：

$$P_{\text{Alice,物品5}} = \bar{R}_{\text{物品5}} + \frac{\sum_{k=1}^2 (S_{\text{物品5,物品k}} (R_{\text{Alice,物品k}} - \bar{R}_{\text{物品k}}))}{\sum_{k=1}^2 S_{\text{物品k,物品5}}} = \frac{13}{4} + \frac{0.97 * (5 - 3.2) + 0.}{0.97 + 0.}$$

这时候依然可以向Alice推荐物品5。

至此， 基于物品的协同过滤算法原理介绍完毕。

### 3.2 编程实现

这个编程实现有了上面的那个就比较简单了， 照着葫芦画瓢的问题了， 所以不再过多的赘述， 直接给出代码， 和上面的差不多：

```
1 """计算物品的相似矩阵"""
2 similarity_matrix = pd.DataFrame(np.ones((len(items), len(items))), index=['A', 'B']
3
4 # 遍历每条物品-用户评分数据
5 for itemId in items:
6     for otherItemId in items:
7         vec_item = [] # 定义列表, 保存当前两个物品的向量值
8         vec_otheritem = []
9         #userRagingPairCount = 0 # 两件物品均评过分的用户数
10        if itemId != otherItemId: # 物品不同
11            for userId in users: # 遍历用户-物品评分数据
```



这里就是物品的相似度矩阵了， 张下面这个样子：

1 similarity\_matrix

executed in 7ms, finished 20:21:10 2020-08-20

	A	B	C	D	E
A	1.000000	-0.476731	-0.123091	0.532181	0.969458
B	-0.476731	1.000000	0.645497	-0.310087	-0.478091
C	-0.123091	0.645497	1.000000	-0.720577	-0.427618
D	0.532181	-0.310087	-0.720577	1.000000	0.581675
E	0.969458	-0.478091	-0.427618	0.581675	1.000000

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。

```
1 """得到与物品5相似的前n个物品"""
2 n = 2
3 similarity_items = similarity_matrix['E'].sort_values(ascending=False)[:n].index.to
4
5 """计算最终得分"""
6 base_score = np.mean(np.array([value for value in items['E'].values()]))
7 weighted_scores = 0.
8 corr_values_sum = 0.
9 for item in similarity_items: # ['A', 'D']
10     corr_value = similarity_matrix['E'][item] # 两个物品之间的相似性
11     mean_item_score = np.mean(np.array([value for value in items[item].values()]))
```



结果如下：

用户Alice对物品5的打分： 4.6					
	A	B	C	D	E
1	5.0	3.0	4.0	4.0	4.6
2	3.0	1.0	2.0	3.0	3.0
3	4.0	3.0	4.0	3.0	5.0
4	3.0	3.0	1.0	5.0	4.0
5	1.0	5.0	5.0	2.0	1.0

同样的， 这里还是实现了上面例子中的那个过程。 项亮老师书里面的那个基于物品的协同过滤放在了GitHub。

### 3.3 优缺点分析

上面说道UserCF存在两个问题， 数据稀疏性和算法扩展性问题， 而ItemCF算法因为物品直接的相似性相对比较固定， 所以可以预先在线下计算好不同物品之间的相似度， 把结果存在表中， 当推荐时进行查表， 计算用户可能的打分值， 可以同时解决上面两个问题。在Item-to-Item论文中， 作者得出结论：

- Item-based算法的预测结果比User-based算法的质量要高一点。
- 由于Item-based算法可以预先计算好物品的相似度， 所以在线的预测性能要比User-based算法的高。
- 用物品的一小部分子集也可以得到高质量的预测结果。

至于存在的问题， 应该是CF存在的共性问题了， 放在下面一块。







- 协同过滤算法(collaborative filtering)

论文:

- Using collaborative filtering to weave an information tapestry, 1992
- Item-to-Item collaborative filtering, 2003

整理这篇文章的同时， 也刚建立了一个GitHub项目， 准备后面把各种主流的推荐模型复现一遍，并用通俗易懂的语言进行注释和逻辑整理，协同过滤算法是一个开头，主要是把项亮推荐系统实践里面的协同过滤算法(UserCF和ItemCF)实现一遍， 并进行解释。 由于也是小白起步， 学习起来比较吃力， 但后面会坚持进行更新， 所以对推荐系统感兴趣的， 欢迎一块探讨和交流， 如果有幸遇到大神， 也希望能带一下 🤝。

筋斗云: <https://github.com/zhongqiangwu960812/AI-RecommenderSystem>

