


# 推荐系统实践-利用用户行为数据（中）



作者

醉起萧寒 (/u/6bb8f84e9703)

+ 关注

2017.02.16 11:25 字数 4410 阅读 120 评论 0 喜欢 0

(/u/6bb8f84e9703)

此文是根据《推荐系统实践》部分整理而来。

## PART 1

基于邻域的算法是推荐系统中最基本的算法，主要分为两大类：

- 基于用户的协同过滤算法
- 基于物品的协同过滤算法

第一部分介绍基于用户的协同过滤算法。在一个在线个性化推荐系统中，当一个用户A需要个性化推荐时，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A没有听说过的物品推荐给A。

主要包括两个步骤

1. 找到和目标用户兴趣相似的用户集合（如果有标签系统能够就会提供很大的帮助）；
2. 找到这个集合中的用户喜欢，且目标用户没有听说过的物品推荐给目标用户。

步骤1的关键就是计算两个用户的兴趣相似度。这里，协同过滤算法主要利用行为的相似度计算兴趣的相似度，给定用户u和用户v，令N(u)表示用户u曾经有过正反馈的物品集合，令N(v)为用户v曾经有过正反馈的物品集合。那么，我们可以通过如下的Jaccard公式简单地计算u和v的兴趣相似度：

$$w_{uv} = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

Paste\_Image.png

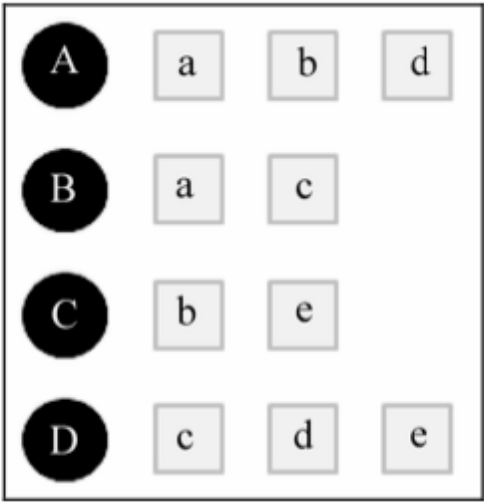
或者通过余弦相似度计算：

$$w_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)| |N(v)|}}$$

Paste\_Image.png

举个例子。用户A对物品{a,b,d}有过行为，用户B对物品{a,c}有过行为，利用余弦相似度公式计算用户A和用户B的兴趣相似度为：

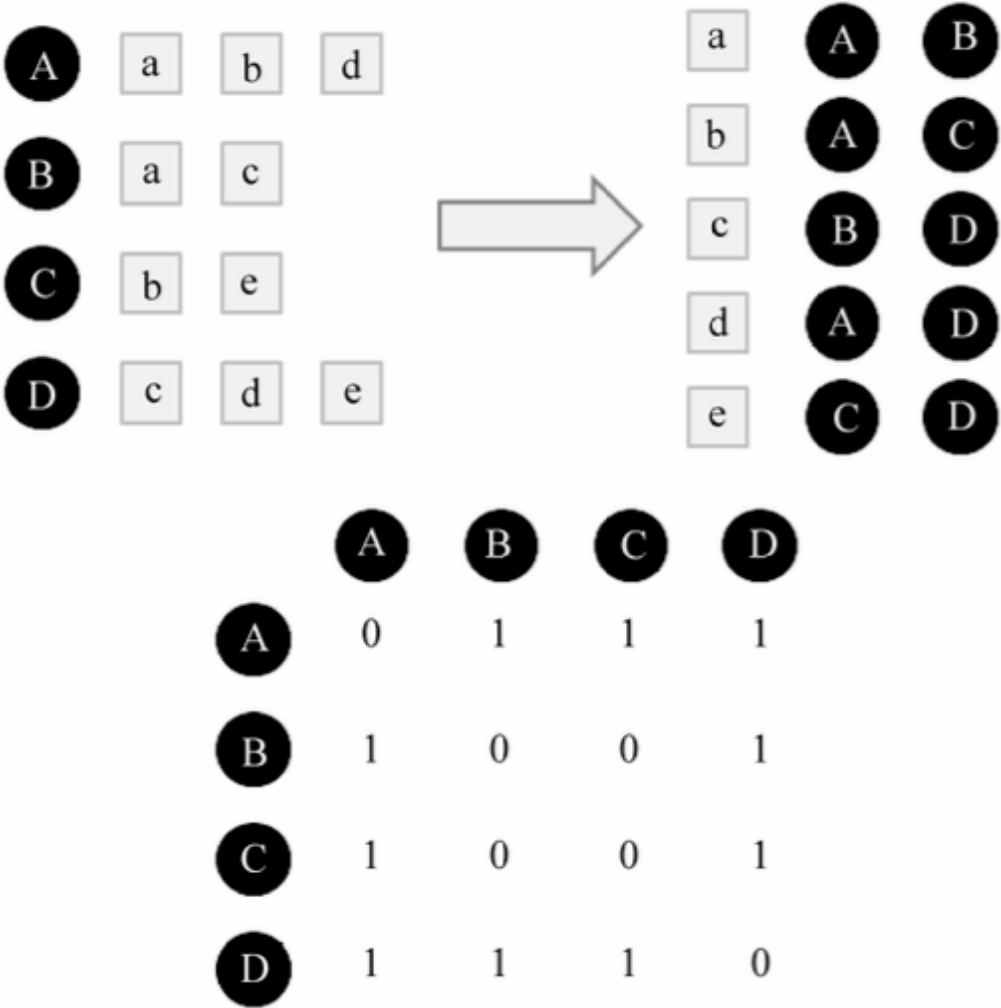
$$w_{AB} = \frac{|\{a,b,d\} \cap \{a,c\}|}{\sqrt{|\{a,b,d\}|} \sqrt{|\{a,c\}|}} = \frac{1}{\sqrt{6}}$$



Paste\_Image.png

如果对所有两两用户都利用余弦相似度计算相似度，这种方法的时间复杂度是  $O(U \times U)$ ，这在用户数大的时候非常耗时。事实上，很多用户相互之间并没有对同样的物品产生过行为，很多的时候  $|N(u) \cap N(v)| = 0$ 。可以首先筛选出  $\neq 0$  的用户对。

为此，可以首先建立物品到用户的倒排表。对于每个物品都保存对该物品产生过行为的用户列表。令稀疏矩阵  $C[u][v] = |N(u) \cap N(v)|$ 。那么，假设用户  $u$  和  $v$  同时属于倒排表中  $K$  个物品对应的用户列表，就有  $C[u][v] = K$ 。从而，可以扫描倒排表中每个物品对应的用户列表，将用户列表中的两两用户对应的  $C[u][v]$  加1。例子展示如下：



Paste\_Image.png

本人再提及一点，在对内容型产品不能直接使用以上算法，因为“物品”数量太多，在这里可以对内容打上标签，由标签来替代物品；同时，如果给用户建立画像，直接给用户的兴趣打上标签，那么会更加容易。

得到用户之间的兴趣相似度后，UserCF算法会给用户推荐和他兴趣最相似的K个用户喜欢的物品。如下的公式度量了UserCF算法中用户  $u$  对物品  $i$  的感兴趣程度：



$$p(u,i)=\sum_{v\in S(u,K)\cap N(i)}w_{uv}r_{vi}$$

Paste\_Image.png

其中，S(u,K)包含和用户u兴趣最接近的K个用户，N(i)是对物品i有过行为的用户集合，wuv是用户u和用户v的兴趣相似度，rvi代表用户v对物品i的兴趣，因为使用的是单一行为的隐反馈数据，所以所有的rvi=1。

根据上图的用户为例，选取K=3，用户A对物品c、 e没有过行为，因此因此可以把这两个物品推荐给用户A。根据UserCF算法，用户A对物品c、 e的兴趣是：

$$p(A,c)=w_{AB}+w_{AD}=0.7416$$
$$p(A,e)=w_{AC}+w_{AD}=0.7416$$

Paste\_Image.png

下表的离线实验来评测基础算法的性能。UserCF只有一个重要的参数K，即为每个用户选出K个和他兴趣最相思的用户，然后推荐那K个用户感兴趣的物品。因此离线实验测量了不同K值下UserCF算法的性能指标：

K	准 确 率	召 回 率	覆 盖 率	流 行 度
5	16.99%	8.21%	51.33%	6.813293
10	20.59%	9.95%	41.49%	6.978854
20	22.99%	11.11%	33.17%	7.10162
40	24.50%	11.83%	25.87%	7.203149
80	25.20%	12.17%	20.29%	7.289817
160	24.90%	12.03%	15.21%	7.369063

Paste\_Image.png

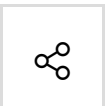
为了反映该数据集上离线算法的基本性能，下图给出两种基本推荐算法的性能。其中，Random算法每次都随机挑选10个用户没有产生过行为的物品推荐给当前用户，MostPopular算法则按照物品的流行度给用户推荐他没有产生过行为的物品中最热门的10个物品。这两个算法都是非个性化的推荐算法：

	准 确 率	召 回 率	覆 盖 率	流 行 度
Random	0.631%	0.305%	100%	4.3855
MostPopular	12.79%	6.18%	2.60%	7.7244

Paste\_Image.png

可以看出，UserCF的准确率和召回率相对MostPopular算法提高了将近1倍。同时，UserCF的覆盖率远远高于MostPopular，推荐结果相对MostPopular不太热门。同时可以发现参数K是UserCF 的一个重要参数，它的调整对推荐算法的各种指标都会产生一定的影响：

- 准确率和召回率。可以看到，推荐系统的精度指标病不和参数K成线性关系。在MovieLens数据集中，选择K=80左右会获得比较高的准确率和召回率。
- 流行度。可以看到，在3个数据集上K越大UserCF推荐结果就越热门。这是因为K决定了UserCF在给你做推荐时参考多少和你兴趣相似的其他用户的兴趣，那么K越大，参考人越多，结果就越来越趋近于全局热门的物品。
- 覆盖率。可以看到，在3个数据集上K越大UserCF推荐结果的覆盖率越低。覆盖率降



低是因为流行度的增加，对长尾物品的推荐越来越少，造成了覆盖率的降低。

上一段中用余弦相似公式计算用户兴趣相似度比较粗略，这段主要介绍下公式的改进。当两个用户对冷门物品采取过同样的行为，更能说明他们兴趣的相似度。因此公式如下：（log1+N(i)什么意思？没有底数）

$$w_{uv} = \frac{\sum_{i \in N(u) \cap N(v)} \frac{1}{\log 1 + |N(i)|}}{\sqrt{|N(u)| |N(v)|}}$$

Paste\_Image.png

可以看到，该公式通过 1/log1+|N(i)| 惩罚了用户u和v共同兴趣列表中热门物品对他们相似度的影响。这种算法作者记为：User-IIF 算法。

下图展示了UserCF和UserCF-IIF的对比推荐性能，在这里选取K=80：

	准 确 率	召 回 率	覆 盖 率	流 行 度
UserCF	25.20%	12.17%	20.29%	7.289817
UserCF-IIF	25.34%	12.24%	21.29%	7.261551

Paste\_Image.png

如图所示，User-IIF 在各方面略优于 User-CF，说明在计算用户兴趣相似度时考虑物品的流行度对提升推荐结果的质量确实有帮助。

本段介绍下实际使用UserCF 推荐算法的实例。最著名的使用者是Digg，系统设计主要通过顶和踩。这种简单的推荐算法的效果在Digg的微博中公布出来：

- 用户反馈增加：顶和踩的行为增加了40%
- 平均你每个用户将从34个具相似兴趣的好友那获得200条推荐结果
- 用户和好友的交互活跃度增加了24%
- 用户评论增加了11%

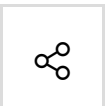
虽然我个人不知道这个效果是因为产品设计，还是推荐算法造成的，但是对推荐系统还是抱有积极的态度。

## PART 2

第二部分介绍基于物品的协同过滤算法。这个算法是目前业界引用最多的算法。基于物品的协同过滤算法给用户推荐那些和他们之前喜欢的物品相似的物品。ItemCF算法并不利用物品的内容属性计算物品直接的相似度，它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B具有很大的相似度是因为喜欢物品A的用户大也都喜欢B。算法主要分为两步：

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。

定义物品的相似度可以用如下公式，其中 N(i) 是喜欢物品 i 的用户数，如下公式表示了喜欢物品 i 的用户中有多少的比例用户也喜欢 j：



$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|}$$

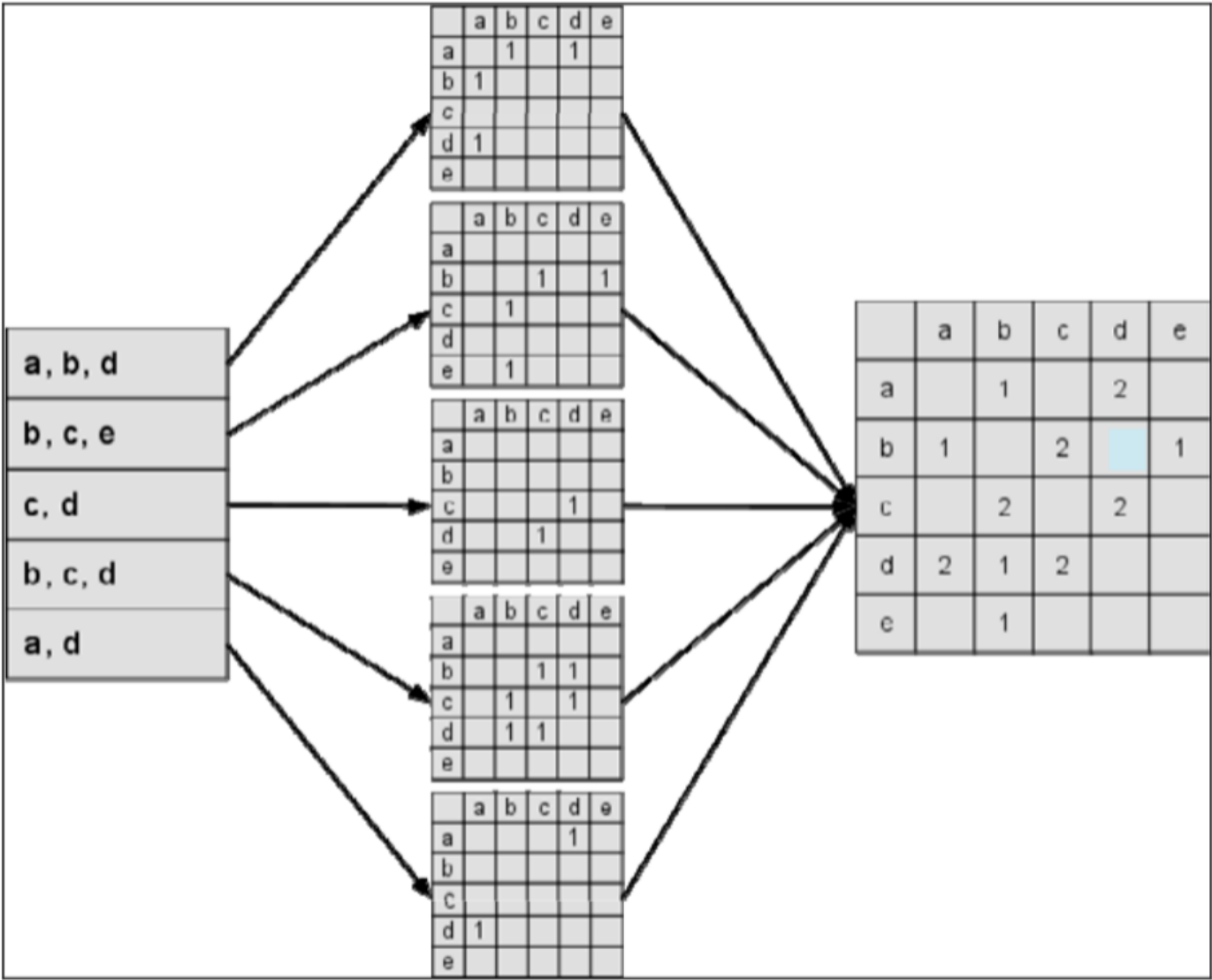
Paste\_Image.png

上面的公式存在一个问题，那么就是如果物品 j 很热门，那么  $w_{ij}$  就会很大，接近1，因此为了避免马太效应，可以用下面的公式：

$$w_{ij} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)| |N(j)|}}$$

Paste\_Image.png

这个公式惩罚了物品 j 的权重，因此减轻了热门物品会和很多物品相似的可能性。和之前的例子一样，下图中最左边是输入的用户行为记录，每一行代表一个用户感兴趣的物品集合。然后对于每个物品集合，我们将里面的物品两两加一，得到一个矩阵。最终将这些矩阵相加得到上面的 C 矩阵。其中  $C[i][j]$  记录了同时喜欢物品 i 和物品 j 的用户数。最后，将 C 矩阵归一化可以得到物品之间的余弦相似度矩阵 W。



Paste\_Image.png

得到物品之间的相似度之后，ItemCF 通过如下公式计算用户 u 对一个物品 j 的兴趣。这里的  $N(u)$  表示用户喜欢的物品的集合， $S(j,k)$  是和物品 j 最相似的 K 个物品的集合， $w_{ij}$  是物品 j 和 i 的相似度， $r_{ui}$  是用户 u 对物品 i 的兴趣。（对于隐反馈数据集，如果用户 u 对物品 i 有过行为，即可令  $r_{ui}=1$ 。）该公式的含义是，和用户历史上感兴趣的物品越相似的物品，越有可能在用户的推荐列表中获得比较高的排名：

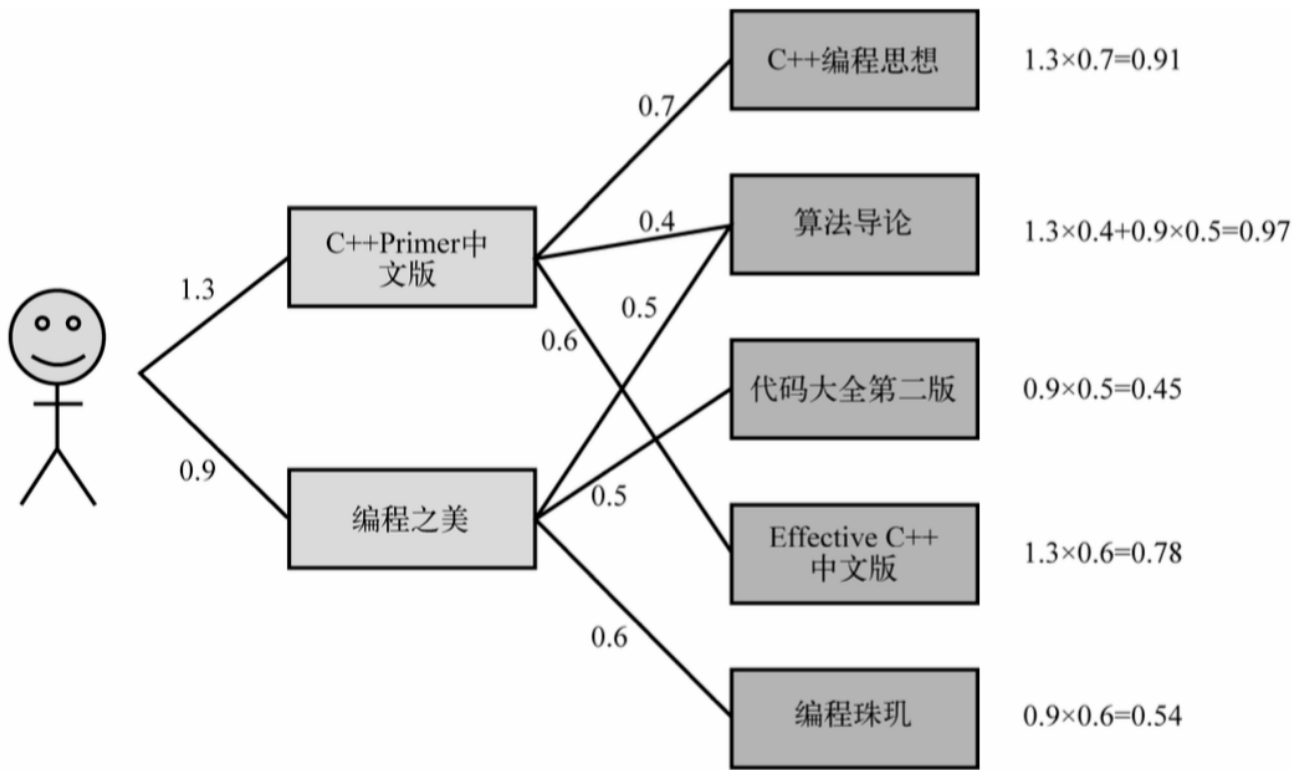




$$p_{uj} = \sum_{i \in N(u) \cap S(j,K)} w_{ji} r_{ui}$$

Paste\_Image.png

举一个简单的例子。用户喜欢《C++ Primer 中文版》和《编程之美》两本书。然后 ItemCF 会为这两本书分别找出和它们最相似的 3 本书，然后根据公式的定义计算用户对每本书的感兴趣程度。



Paste\_Image.png

下表列出了MovieLens数据集上 ItemCF 算法离线实验的各项性能指标的评测结果：

K	准 确 率	召 回 率	覆 盖 率	流 行 度
5	21.47%	10.37%	21.74%	7.172411
10	22.28%	10.76%	18.84%	7.254526
20	22.24%	10.74%	16.93%	7.338615
40	21.68%	10.47%	15.31%	7.391163
80	20.64%	9.97%	13.64%	7.413358
160	19.37%	9.36%	11.77%	7.385278

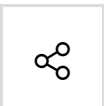
Paste\_Image.png

其中我们可以得出一些结论：

- 精度（准确率和召回率）。不和K成正相关或者负相关，选择合适的K对获得最高精度是非常重要的。
- 流行度。参数K对该算法的推荐结果流行度的影响也不是完全正相关的。
- 覆盖率。K增加会降低系统的覆盖率。

接下来需要进行算法的优化。有一个问题是，是不是每个用户对于物品相似度的贡献都是一样的呢？假设有一个用户买了大批书，但是他买这些书都不是出自自己的兴趣，可能是代购之类；而另一个用户虽然只买了十几本书，但是领域相似，都是该用户确实喜欢的书。

因此用户活跃度对物品相似度也是有影响， John S.Breese 认为活跃用户对物品相似度的贡献小于不活跃的用户，因此提出 IUF 参数，即用户活跃度对数的倒数的参数。修正后的公式如下：



$$w_{ij} = \frac{\sum_{u \in N(i) \cap N(j)} \frac{1}{\log 1 + |N(u)|}}{\sqrt{|N(i)| |N(j)|}}$$

Paste\_Image.png

作者将概述算法记为 ItemCF-IUF，在下表中用离线实验评测这个算法，在这里参数K=10：

	准 确 率	召 回 率	覆 盖 率	流 行 度
ItemCF	22.28%	10.76%	18.84%	7.254526
ItemCF-IUF	22.29%	10.77%	19.70%	7.217326

Paste\_Image.png

如表所示，优化后的算法整体有所提升，明显提升覆盖率，降低流行度。

Karypis在研究中发现如果将ItemCF的相似度矩阵按最大值归一化，可以提高推荐的准确率。其研究表明，如果已经得到了物品相似度矩阵w，那么可以用胡夏公式得到归一化之后的相似度矩阵w'：

$$w'_{ij} = \frac{w_{ij}}{\max_j w_{ij}}$$

Paste\_Image.png

归一化是指，将不同的参考系变成统一的参考系，便于建立标量。假设物品分为两类：A和B，A类之间的物品相似度为0.5，B类之间的为0.6，而A类和B类物品之间的相似度是0.2。在这种情况下，如果一个用户喜欢5个A类和5个B类物品，那么推荐的就都是B类物品，因为B类物品之间的相似度大，但是归一化之后，A类物品之间的相似度变成了1，B类物品之间的相似度也是1，如果用户同样喜欢5个A类5个B类，推荐列表中的A类和B类物品的数目也应该是大致相等的。

一般热门的类内物品的相似度一般比较大，如果不进行归一化，就会推荐比较热门的类里面的物品，而这些物品也是比较热门的，这样推荐的覆盖率就会降低。

下图对比了ItemCF算法和ItemCF-Norm算法的离线实验性能：

	准 确 率	召 回 率	覆 盖 率	流 行 度
ItemCF	22.28%	10.76%	18.84%	7.254526
ItemCF-Norm	22.73%	10.98%	23.73%	7.157385

Paste\_Image.png

实验结果可以看到，归一化确实能提高ItemCF的性能，其中各项指标都有了明显的提高。

### PART 3

这部分综合对比下UserCF和ItemCF。主要从几个角度来说：

- 产品的类型



• 技术实现

从这两个算法的推荐原理，来看，UserCF的推荐结果着重于反映和影虎兴趣相似的小群体的热点，而ItemCF的推荐结果着重于维系用户的历史兴趣。换句话说，UserCF的推荐更社会化，反映了用户所在的小型兴趣群体中的热门程度，而ItemCF的推荐更加个性化，反映了用户自己的兴趣传承。所以选择算法要考虑社会性和个性。

比如新闻类的产品，用户兴趣不是很细化，绝大多数用户都喜欢看热门的新闻，即使是个性化，也是比较粗粒度的。因此，个性化新闻推荐更加强调抓住新闻热点，热门程度和时效性是个性化新闻推荐的重点，而个性化则相对次要。因此，UserCF更加适合。

比如电商网站如亚马逊，用户的兴趣比较固定和持久。一个技术人员可能都是在购买技术书，对书的热门程度不敏感，甚至越来越倾向冷门的物品。因此，这些网站中个性化推荐的任务是帮助用户发现和他研究领域相关的物品，用ItemCF算法较为合适。

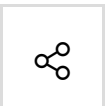
从技术角度来考虑，UserCF需要维护一个用户相似度的矩阵，而ItemCF需要维护一个物品相似度矩阵。如果用户很多，那么需要维护用户兴趣相似度矩阵需要很大的空间；物品同理。不过如果结合标签的话，可以有很大的优化空间。

下图从不同角度比较了两种算法：

性能	适用于用户较少的场合，如果用户很多，计算用户相似度矩阵代价很大	适用于物品数明显小于用户数的场合，如果物品很多（网页），计算物品相似度矩阵代价很大
领域	时效性较强，用户个性化兴趣不太明显的领域	长尾物品丰富，用户个性化需求强烈的领域
实时性	用户有新行为，不一定造成推荐结果的立即变化	用户有新行为，一定会导致推荐结果的实时变化
冷启动	在新用户对很少的物品产生行为后，不能立即对他进行个性化推荐，因为用户相似度表是每隔一段时间离线计算的	新用户只要对一个物品产生行为，就可以给他推荐和该物品相关的其他物品
	新物品上线后一段时间，一旦有用户对物品产生行为，就可以将新物品推荐给对它产生行为的用户兴趣相似的其他用户	但没有办法在不离线更新物品相似度表的情况下将新物品推荐给用户
推荐理由	很难提供令用户信服的推荐解释	利用用户的历史行为给用户做推荐解释，可以令用户比较信服

Paste\_Image.png

下面几张图用之前的离线实验结果对比两种算法：





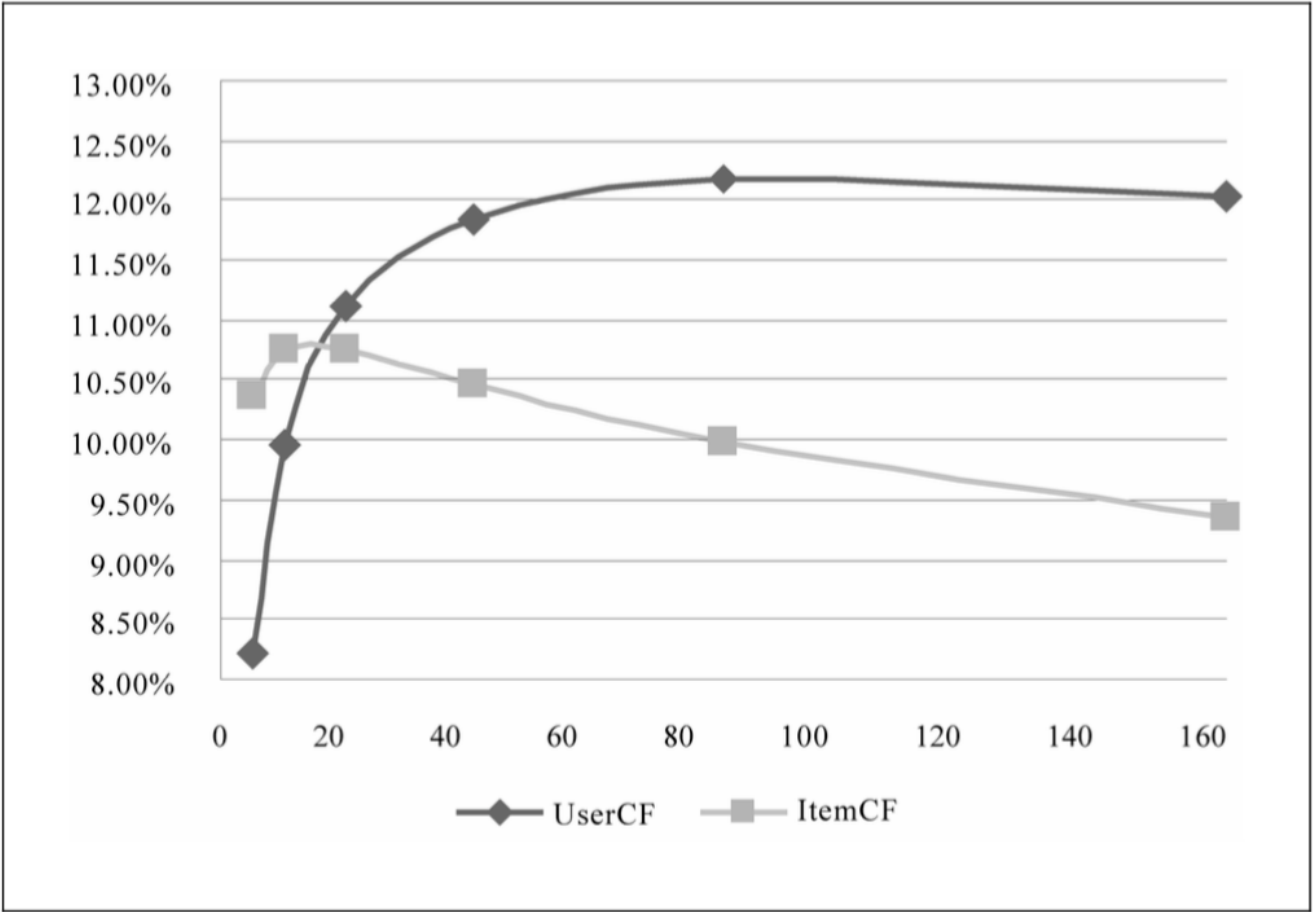


图2-13 UserCF和ItemCF算法在不同K值下的召回率曲线

Paste\_Image.png

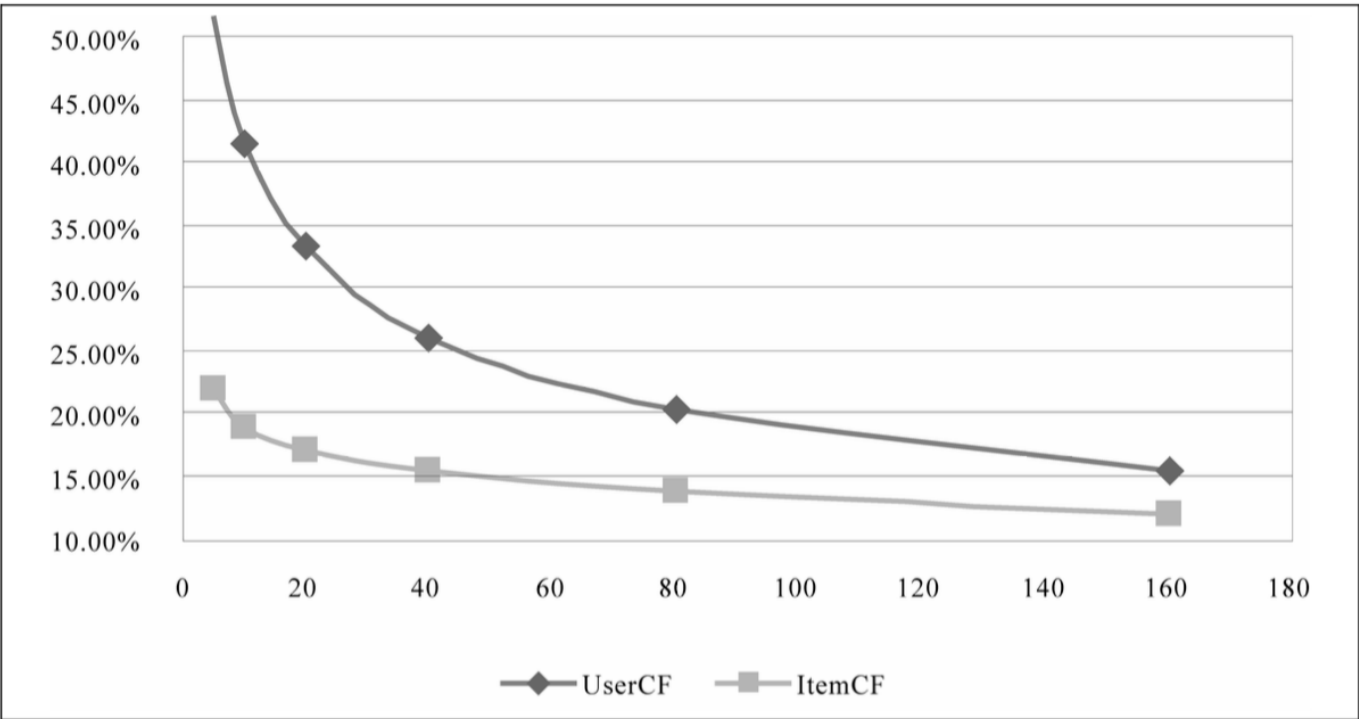


图2-14 UserCF和ItemCF算法在不同K值下的覆盖率曲线

Paste\_Image.png

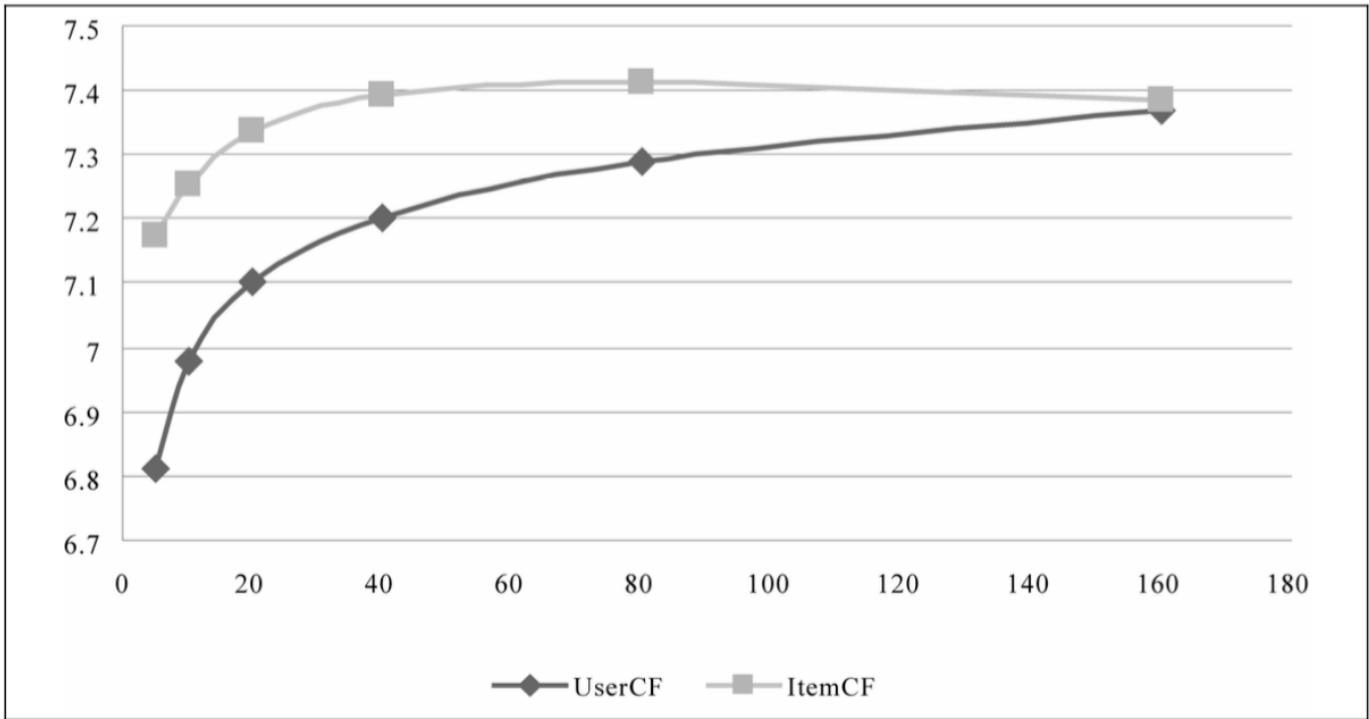
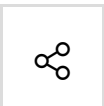


图2-15 UserCF和ItemCF算法在不同K值下的流行度曲线





结果显示，ItemCF算法在各项指标上不如UserCF，特别是覆盖率和新颖度。

需要注意的是，离线实验的性能在选择推荐算法时不起决定性作用，首先应该满足产品的需求，比如需要提供推荐解释；其次，需要看实现代价，比如用户太多就很难计算用户相似度矩阵；最后，离线指标和点击率等在线指标不一定成正比。

“对于稀疏数据集（例如delicious书签，小众群体收藏），基于物品的过滤方法通常要优于基于用户的过滤方法，而对于密集数据集而言，两者的效果几乎是一样的”--摘自《集体智慧编程》

最后补充一点，ItemCF算法的覆盖率和新颖度都不高，是因为两个不同领域的最热门物品之间往往具有比较高的相似度，在这时候仅仅依靠用户行为数据不能解决这个问题。比如说有一本书是最畅销的书，那么无论什么用户都会被推荐给这本书，消除马太效应是ItemCF需要考虑的。

读书 (/nb/8996681)

举报文章    © 著作权归作者所有

醉起萧寒 (/u/6bb8f84e9703)

写了 36004 字，被 13 人关注，获得了 18 个喜欢 (/u/6bb8f84e9703)

+ 关注

丁香园产品经理

如果觉得我的文章对您有用，请随意打赏。您的支持将鼓励我继续创作！

赞赏支持

喜欢 (/sign\_in)

|

0

更多分享

(http://cwb.assets.jianshu.io/notes/images/9033688/weibo/image\_93a16896e05f.jpg)

登录 (/sign\_in)

后发表评论

评论

被以下专题收入，发现更多相似内容

