**基于内存的推荐算法**

**靳远远**

基于内存的算法分为两大类：基于用户的协同过滤算法与基于物品的协同过滤算法。协同过滤是指参考其他人的选择来帮助自己在众多要选择的物品中过滤掉与自己的兴趣不相关的，保留自己较感兴趣的物品，要达到这个目的，人们通常向与自己兴趣相仿的人征求建议。比如想买一本名人传记，人们问问自己周围的朋友，了解一下他们最近看的这方面的书籍，从中选几个跟自己兴趣相仿的人给出的书单再进行购买。协同过滤算法的核心思想就在于此。

协同过滤一般是在海量的用户中发掘出一小部分和你兴趣比较类似的成为你的“邻居”，然后根据反映他们喜好的用户行为组织出一个排好序的推荐目录给你。要编写算法来自动化完成这个推荐任务，需要给出以下问题的答案：

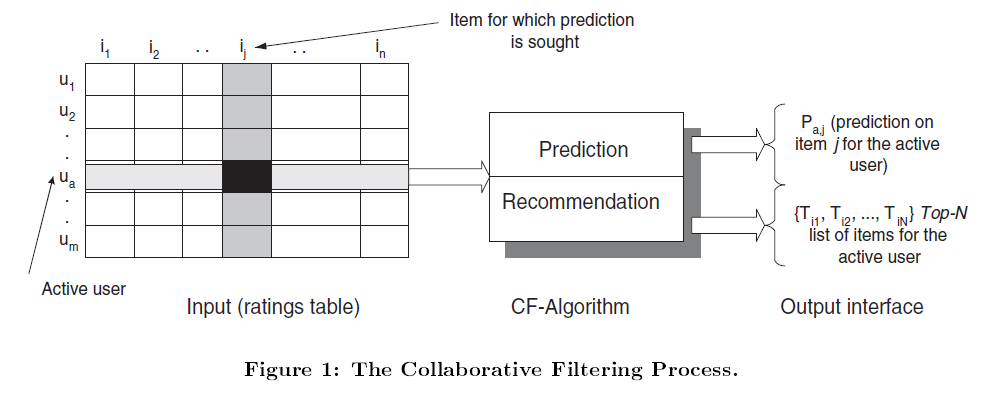
* 找出与你兴趣相仿的用户，即你的“邻居”
* 把邻居们喜欢的物品根据某种规则进行排序，给出最终的推荐列表

**协同过滤的核心**

完成协同过滤的一般步骤如下

* 对用户行为数据进行预处理
* 找到相似的用户或物品，预测目标用户对自己没有访问过的商品的感兴趣程度
* 对推荐候选集中的物品进行排序，给出推荐列表

论文[7]中的图表描述了得到用户偏好后进行推荐的流程，其中的prediction和recommendation对应上述的后两步：



Rating table是一个的打分矩阵，这里的active user表示当前在对这个用户进行推荐。其中的每一项 表示第个用户对第个物品的偏好分数，每个打分值在一定的数值范围内，若用户对该商品没有行为，则值为0。可以看到最后在recommendation阶段给出的推荐列表中含有top-N个物品，通常是根据物品的推荐指数排序后取前N个推荐给目标用户，这种推荐称为Top-N推荐。

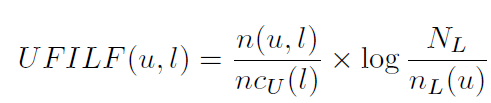
**收集用户行为数据**

用户的兴趣反映在用户和系统交互时的各种行为中，先要从这些数据中挖掘出用户偏好，才能基于此做出推荐。所以预处理用户行为数据的方式是影响推荐效果最基础的决定因素。用户的网上行为有多种类型，比如在亚马逊网站上，一个用户可能留下浏览网页，购买书籍，点击某一商品查看详情，评价已经收到的商品，给出评分等行为。这些行为可分为显式反馈行为（explicit feedback）和隐式反馈行为（implicit feedback）两种，显式行为指能明显表示用户喜好的行为，比如豆瓣中的评分系统和有些网站的“顶”和“踩”的按钮来收集用户的偏好，隐式反馈行为指不能明确反应用户偏好的行为，最常见的是页面浏览行为，相比显式行为，隐式行为虽然不明确，但数据量更大。按照反馈的情感态度来分，这些行为又可以细分为正反馈和负反馈，正反馈指用户的行为倾向于指用户喜欢该物品，而负反馈则是反映用户不喜欢该物品。下面用图表总结一下常见的用户行为类型：

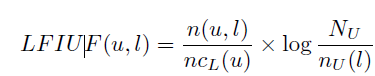
|  |  |  |  |
| --- | --- | --- | --- |
| 用户行为 | 类型 | 特征 | 作用 |
| 评分 | 显式 | 整数量化的偏好，可能的取值是 [0, n]；n 一般取值为 5 或者是 10 | 通过用户对物品的评分，可以精确的得到用户的偏好 |
| 投票 | 显式 | 布尔量化的偏好，取值是 0 或 1 | 通过用户对物品的投票，可以较精确的得到用户的偏好 |
| 保存书签 | 显式 | 布尔量化的偏好，取值是 0 或 1 | 通过用户对物品的投票，可以精确的得到用户的偏好。 |
| 评论 | 显式 | 一段文字，需要进行文本分析，得到偏好 | 通过分析用户的评论，可以得到用户的情感：喜欢还是讨厌 |
| 页面停留时间 | 隐式 | 一组时间信息，噪音大，需要进行去噪，分析，得到偏好 | 用户的页面停留时间一定程度上反映了用户的注意力和喜好，但噪音偏大，不好利用。 |
| 购买 | 隐式 | 布尔量化的偏好，取值是 0 或 1 | 用户的购买是很明确的说明这个项目它感兴趣。 |

收集了用户行为数据后，还需要对数据进行一定的预处理。因为现实世界中的数据大都是不完整，不一致的脏数据，无法直接用于后续的数据分析工作，所以产生了数据预处理技术。在数据挖掘领域，数据预处理与后续的特征工程占据了很大的工作量。常用的数据预处理方法有：数据清理，数据集成，数据变换，数据规约等。比如数据清理，通过填补缺失的值，识别或删除离群点并解决不一致性来清理数据，主要是为了纠正错误，清除重复数据。而数据变换则通过平滑聚集，数据概化，规范化等方式将数据转换为适用于数据挖掘的形式。比如减噪和归一化。归一化主要用于将不同类型用户行为的数据统一在一个相同的取值范围内，从而使得加权求和得到的用户的偏好更加精确。基础的归一化处理有按最大值归一化，将取值限制在[0,1]范围内；或者把单个样本的取值除以所有样本的取值之和同样达到这一目的。 [5]中采用了两种方式来利用用户签到数据：

* 将签到数据转化为二元变量，如果用户有在某地的签到记录则为1，否则为0
* 使用权重模式来衡量用户偏好：FIF(frequency-inverse frequency)，该方法把签到次数转化为一个归一化的权重值。该方法来源于信息检索领域经常用到的TFIDF的思想，包含两部分：UFILF(User Frequency-Inverse Location Frequency)，用来衡量一个用户的访问对于某一个POI的重要程度；LFIUF(Location Frequency-Inverse User Frequency)，用来衡量一个POI对某个用户的重要程度。公式如下：



其中是用户在POI处的签到次数，是所有用户在POI处的签到总次数，是用户签过到的POI总数。



其中是用户在POI处的签到次数，是用户的签到总数，，是访问过POI 的用户总数。

至此可得



预处理后可以得到一个用户偏好的二维矩阵，每一行代表一个用户，每一列代表一个物品，矩阵中的每个值是用户对物品的偏好。

**找到相似的用户或物品**

经过数据预处理后，我们得到了表示用户兴趣的user-item矩阵，其中每一行记录了一个用户对所有item的行为，每一列记录了所有的用户对这个item的行为。接下来可以基于此计算不同的用户或不同的商品之间的相似度，也就是协同过滤中的两个分支：use-based CF和item-based CF。这一步的关键是相似度的计算，下面详细介绍几种相似度的计算方法。

**相似度的计算**

常用的相似度计算方法都是将用户对所有物品的行为建模成一个向量，即user-item-preference二维矩阵中的一行，或者将所有用户对某一物品的行为建模成一个向量，即对应user-item-preference矩阵中的一列，这样就把相似度的计算转化为两个向量之间距离的计算，距离越近表示相似度越大。 以下图为例，要计算

用户和用户之间的相似度即是计算向量与向量之间的距离；同理要计算物品和物品之间的相似度转化为计算向量与向量之间的距离:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 1 | 0 |
|  | 1 | 1 |

这里1表示用户对物品有过行为，0表示用户对物品没有过行为。

下面列出了几种常用的计算相似度的方法：

**皮尔逊相关系数（Pearson Correlation Coefficient）**

一般用于计算两个变量间联系的紧密程度，取值在[-1,+1]之间，负值表示两个变量呈负相关，正值表示两个变量呈正相关。

其中表示对打过分的用户集合，，表示物品受到的用户打分的均值。

从计算公式上来看，如果共同评价过某两个物品的用户集合大小比共同评论过其他电影的用户集合大小要小得多，那么这两个物品的相似度的可靠性就比较低。而现在待处理的推荐系统的规模越来越大，也带来了数据稀疏性的问题，这会大大加强相似度的不可靠性，为了提高结果的可靠性，可以根据共同评价过两个商品的用户集合的大小对相似度进行压缩：

其中为用户指定的压缩系数，根据上面的式子可以看出当越大时压缩程度就越大，用户集合大小越小，乘上的系数值也就越小，也就是把计算得到的皮尔森相似度根据用户集合大小乘上相应的系数，用户集合越大，相似度越可靠，乘上的系数就越大，用户集合越小，相似度越不可靠，乘上的系数就越小。

**Cosine相似度**

其中表示用户曾经有过正反馈的物品集合，表示用户曾经有过正反馈的物品集合。

**Jaccard 系数**

与的含义同上。

**Adjusted Cosine**

修正的余弦相似性是对余弦相似度的变体。余弦相似度没有考虑到用户的评分尺度问题，如在评分区间[1-5]的情况下，对用户甲来说评分3以上就是自己喜欢的，而对于用户乙，评分4以上才是自己喜欢的。通过减去用户对项的平均评分，修正的余弦相似性度量方法一定程度解决了以上问题。

**相似邻居的计算**

**常用的挑选邻居的原则为取TOP-K个最相近的邻居。**

**K-neighborhoods**

**不论邻居的远近，只取最近的K个，作为其邻居，如下图，若要计算点1的5个邻居，我们取最近的5个点，分别是点2，点3，点4，点7和点5。**

4

3

1

7

2

5

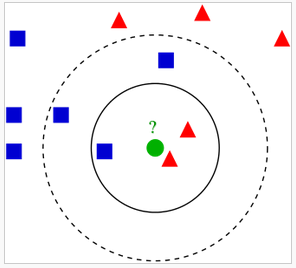
8

6

这种方法实际上借鉴了KNN模型的核心思想，KNN模型是k近邻分类(k-nearest neighbor classification)算法，是最简单的机器学习算法之一，后面要介绍的item-based和user-based的算法是将该算法的核心思想应用于推荐系统得到的。

KNN算法最早由[15]提出，后来又经过了一些改进。最近邻算法是KNN算法在K=1时的特例，即寻找最近的邻居。KNN算法的核心思想是若要判断一个样本点的类别，参考和它比较相近的样本点的类别就可以了。其中用来寻找邻居的样本集合都是经过正确分类的。由于KNN方法主要靠周围有限的临近样本做出类别决策，所以当样本集的类域交叉或重叠较多时，KNN方法相较与基于类域的判别方法更有优势。K-NN可以看成：有已经分好类的训练数据，然后来了一个新数据时，就开始跟训练数据里的每个点求距离，然后挑离这个训练数据最近的K个点看看这几个点属于什么类型，然后用少数服从多数的原则，给新数据归类。

下面使用维基百科上的一个例子来认识KNN分类算法：



在上图中为确定绿色的点是属于正方形这个分类还是属于三角形这个分类，分别选取k=3（此时该绿色点的邻居圈在实线圆形内），可以看出邻居中有两个三角形和一个正方形，所以该绿色点应该属于三角形这一类；当k=5时（此时绿色点的邻居圈在虚线圆形内），同理此时该绿色点属于正方形这一类。可以看出，最终的分类结果与k的取值有关。

在这里我们不用KNN来实现分类，而是使用KNN最原始的算法思路，即为每个内容寻找K个与其最相似的内容，并推荐给用户。将这一思想应用于推荐场景中就是要预测用户M对一首歌的评分，只需要找出K个与用户M相似的且对这首歌评过分的用户，根据这K个用户的评分来预测用户M对这首歌的评分，这种方法叫做user-based KNN；要评价一个未知的物品U,只需找k个与U相似的已知的物品，并通过已有的用户对这k个物品的评分来对U进行评价，这种方法叫做item-based KNN。

**补充另外一种方法（不常用）**

基于相似度门槛的邻居：Threshold-based neighborhoods . 与计算固定数量的邻居的原则不同，基于相似度门槛的邻居计算是对邻居的远近进行最大值的限制，落在以当前点为中心，距离为 K 的区域中的所有点都作为当前点的邻居，这种方法计算得到的邻居个数不确定，但相似度不会出现较大的误差。如图 1 中的 B，从点 1 出发，计算相似度在 K 内的邻居，得到点 2，点 3，点 4 和点 7，这种方法计算出的邻居的相似度程度比前一种优，尤其是对孤立点的处理。

**用户对物品的感兴趣程度计算（打分预测）**

**参照讨论班的书**

得到用户或物品之间的相似度后，接下来要计算目标用户对之前自己没有接触过的商品的感兴趣程度，在显式数据集中可以称为打分预测（rating prediction）问题。最基本的方法是通过计算邻居们对同一件物品的打分加权均值，用目标用户与邻居之间的相似度作为权重。

令表示用户之间的相似度，表示用户的K个最相似的邻居，对于基于用户的协同过滤的打分预测公式如下

同理在基于商品的协同过滤推荐中，要预测对的打分值，可以通过对在与最相似的K个物品的打分值进行加权平均，权重为与邻居商品的相似度。令表示之间的相似度，表示打过分的商品中，与最相近的K个商品。基于商品的打分预测公式如下：

上述方法可以扩展为包含无偏修正项。例如可以使用复合基线来实现对每一个特定用户的调节作用：

这里的可以是用户对所有物品打分的均值。

另外在基础预测方法中，是的权重值。可以定义一个更一般的形式：

*Neighborhood relationship model*[1,2]中将其建模为优化问题：

其中表示与最相近的K个商品，而且同时被打过分。

**推荐系统的实验方法和评价指标**

评测推荐算法的效果的实验大概分为3类：

**离线实验**（offline experiment）

离线实验在数据集上进行，并不需要一个实际的系统来验证，只需要有从实际系统中得到的数据集即可，很多论文在检验文中提出的推荐算法时大都采用现有的公开数据集，比如MovieLents,Lastfm,或者从一些网站上爬取自己所需要的数据，用于离线实验，优点是可以方便快速地来检测自己的算法效果。步骤如下：

1. 将用户行为数据整理成一个标准的数据集
2. 将数据集分成测试集和训练集
3. 在训练集上训练算法模型，在测试集上进行预测
4. 通过评价指标来评测算法效果

**用户调查（user study）**

相比上面的方法，这种方法需要真实用户的参与，用户要在待测的推荐系统上完成一些实验，在这些实验的过程中收集用户志愿者的行为来评测系统的性能。在心理学研究领域，这种实验比较常见，通过招募志愿者来参与一些问卷调查,然后对问卷调查结果进行统计分析。显然这是一种抽样调查，那么样本的选取需要保证志愿者的分布和真实用户的分布相同，比如性别，年龄，职业等分布要尽量符合真实用户群的分布情况。其缺点主要是成本较高，所以一般只能小规模的进行。

**在线实验**（online experiment）

直接上线测试是最好的方法，因为它是在真实环境中测试算法。这里介绍一种常见的在线评测算法的实验方法：AB测试。它通过一定的规则将用户随机分成几组，并对不同组的用户采用不同的算法，然后通过统计不同组用户的各种不同的评测指标比较不同算法，比如统计不同组用户的转化率和点击率来比较不同算法的性能。AB测试本质上是个分离式组间实验，即为同一目标制定两个方案（这里是两种不同的推荐算法），让一部分用户使用A方案，另一部分用户使用B方案，记录下用户的使用情况，看哪个方案更符合设计。AB测试的缺点是周期较长，经过长期的实验才到得到比较可信的实验结果，所以一般使用AB测试来测试在离线实验和用户调查中表现比较好的算法。

所以这三个步骤实际上是一环套一环的：

1. 首先通过离线实验说明新提出的推荐算法优于现有的算法
2. 然后通过用户调查确定其用户满意度不低于现有的算法
3. 最后才是通过在线的AB测试来确定它优于现有的算法

在推荐系统实验中常用的评价指标主要有：召回率，精度，覆盖率，流行度等。下面详细介绍各个指标的含义和计算方法：

令表示给目标用户的推荐列表，是用户在测试集上的行为列表

**准确率**（Precision）

由定义可以看出准确率描述最终的推荐列表中有多少比例是发生过的用户-物品评分记录。

**召回率**（Recall）

由定义可以看出召回率描述的是有多少比例的用户-物品评分记录包含在最终的推荐列表中。

**覆盖率（Coverage）**

覆盖率描述一个推荐系统对物品长尾的发掘能力，即对不那么热门的商品的发掘能力。覆盖率最简单的定义是推荐系统能够推荐出来的物品占总物品集合的比例。假设用户集合为U,每个用户会被推荐一个长度为N的物品列表，是总物品集合。覆盖率的计算公式如下：

和上面的两个指标不同，这个指标是网站的内容供应商会关心的指标，比如亚马逊上家用电器的供应商会很关心他们的电器是否推荐给用户过。极端情况下，覆盖率为100%的推荐系统可以将每个物品都推荐给至少一个用户。

**新颖性**

给用户推荐他们以前没有听说过的物品，那可以直接从推荐列表中把用户之前已经有过行为的物品过滤掉来达到这一目的，但实际上仅过滤掉用户已经有过行为的物品并不能完全实现新颖性，因为用户完全可能通过该网站之外的媒介看过推荐列表中剩余的物品。计算新颖性最简单的方法是利用推荐结果中物品的平均流行度，因为越不热门的物品越可能让用户觉得新颖[14]。

下面给出计算物品的平均流行度的伪码：

def popularity(train,test,N):

Item\_popularity=dict()

for user,items in train.items():

for item in items.keys()

If item not in item\_popularity:

Item\_popularity[item]=0

Item\_popularity[item]+=1

ret=0

n=0

for user in train.keys():

rank=GetRecommendation(user,N)

for item,pui in rank:

rat+=math.log(1+item\_popilarity[item])

n+=1

ret/=n\*1.0

return ret

代码中的第一个for循环是对每个物品出现的次数进行计数，第二个循环是对推荐列表中的所有商品的出现次数取对数后进行累加，然后再除以推荐列表中的物品总数，从而得到推荐结果中物品的平均流行度。这里物品的流行度用物品在训练集中的出现次数来衡量。

这里对流行度取对数的原因是物品的流行度分布服从长尾分布，在取对数后，流行度的平均值更加稳定。这里提到的长尾分布是互联网上的很多数据都满足的一种分布，比如互联网上歌曲和软件的下载，网页的点击，网上店铺的销售等。一般形式为,以歌曲下载量为例，把曲目按照下载量排序，可近似地得到一条递减曲线。在曲线的头部，热门曲目被大量下载。接下来，随着流行程度的降低（对应为序号的增大），曲线徒然下降。但有趣的是，在尾部曲线并没有迅速坠落到零，而是极其缓慢地贴近于横轴，粗看上去几乎与横轴平行延伸（这说明很不热门的曲目仍然保持着一定的下载率）。

下面将对基于用户和基于商品这两种算法进行深入介绍，解决这两个问题。

* **基于用户的协同过滤算法**（基础方法的变体还没找到）

基于用户的协同过滤算法是推荐系统中最古来的算法，该算法在1992年被提出，应用于Tapestry这一邮件过滤系统[3], 主要是解决Xerox公司在Palo Alto的研究中心资讯过载的问题；1994年的GroupLens主要应用于新闻筛选。它的基本思想与我们在日常生活中寻求他人推荐的过程十分相似，比如我们要找一些小说来看，会先通过找到跟自己兴趣比较接近的人，看看他们最近都读了那些小说，再看看这些小说中有哪些是自己比较感兴趣的，加入自己的待读书单。

1. 基础算法

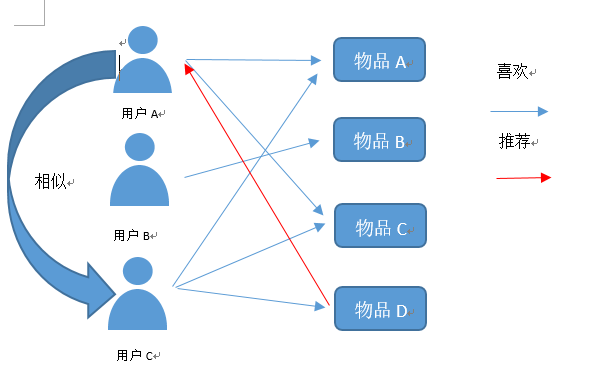
使用基于用户的协同过滤算法为用户做出推荐时，首先找到与目标用户偏好相似的其他用户，然后把这些用户喜欢的，而且目标用户之前没有接触过的物品推荐给目标用户。

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
2. 找到这个集合中用户喜欢的，且之前没有听说过的物品推荐给目标用户

下面举例说明使用UserCF来计算用户兴趣相似度。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用户/物品 | 物品A | 物品B | 物品C | 物品D |
| 用户A | √ |  | √ | 推荐 |
| 用户B |  | √ |  |  |
| 用户C | √ |  | √ | √ |



利用余弦相似度计算用户A和用户C的兴趣相似度为：

余弦相似度的实现可以利用如下伪码：

def UserSimilarity(train):

W=dict()

For u in train.keys():

For v in train.keys():

If u==v:

Continue

W[u][v]=len(train[u]&train[v])

W[u][v]/=math.sqrt(len(train[u])\*len(train[v])\*1.0)

Return W

注意到该代码有两个for循环，也就是会计算所有用户对之间的相似度，而在现实生活中，只有一小部分用户对同样的商品产生过行为，有很多计算是没有必要的。因此我们可以只计算那些对同样的商品有过行为的用户对的相似度，即找出的用户对来计算相似度。

首先可以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]=。则假设用户和用户同时属于倒排表中K个物品对应的用户列表，就用C[=K。可以通过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[加1，最终得到所有用户之间不为0的C[。相应的伪码如下：

def UserSimilarity(train):

/\*为物品建立倒排表\*/

Item\_users=dict()

For u ,items in train.items():

For I in items.keys():

If I not in item\_users:

Item\_users[i]=set()

Item\_users[i].add(u)

/\*计算用户之间的共有商品数\*/

C=dict()

N=dict()

for I ,users in item\_users.items():

For u in users:

N[u]+=1

For v in users:

If u==v:

Continue

C[u][v]+=1

/\*计算最终的相似度矩阵W\*/

W=dict()

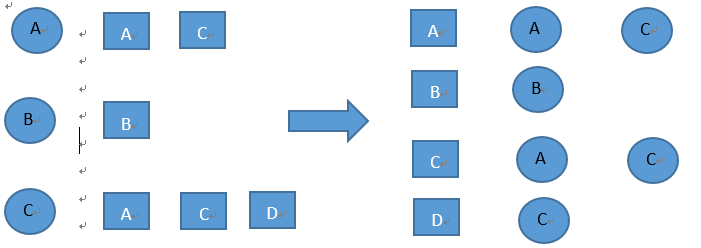
For u,related\_users in C.items():

For v,cuv in related\_users.items():

W[u][v]=cuv/math.sqrt(N[u]\*N[v])

Return W

同样以上图为例来说明，首先建立物品-用户的倒排表，然后建立一个的用户相似度矩阵W,对于物品A,将W[A][C]和W[C][A]加1，以此类推。



|  |  |  |  |
| --- | --- | --- | --- |
|  | A | B | C |
| A | 0 | 0 | 1 |
| B | 0 | 0 | 0 |
| C | 1 | 0 | 0 |

得到用户之间的相似度后，接下来会推荐给用户和他兴趣最相似的K个用户喜欢的物品，接下来计算用户u对物品i的感兴趣程度：

其中是和用户u兴趣最相似的K个用户，是对物品i有过行为的用户集合，是用户v对物品i的兴趣，若使用隐式反馈数据，则均为1，显式数据随所使用的指标不同而有不同的定义。

以下伪码用户实现上述UserCF算法：

Def Recommend(user,train,W):

Rank=dict()

Interacted\_items=train[user]

For v,wuv in sorted(W[u].items,key=itemgetter(1),

Reverse=True)[0:K]:

For I ,rvi in train[v].items():

If I in interacted\_items[v].items():

Continue

Rank[i]+=wuv\*rvi

Return rank

根据上面的例子，用户A与用户C比较相似，选择K=1，用户A对物品D没有过行为，可以把D推荐给A,用户A对D的兴趣是：

1. 用户相似度计算的改进

上面是最基础的余弦相似度计算方法，而在有些情况下需要对其进行调整。比如有两个用户都购买过《牛津英语词典》这一工具书，并不能据此说明两个人的兴趣相似，因为这是高中英语必备的工具书。但如果有两个人都买了日本推理作家东野圭吾的推理小说《白夜行》，则可以认为两人都对推理悬疑类的文学作品感兴趣。所以可见若两个用户都对比较冷门的商品有过行为更能说明他们的兴趣相似，也就是计算时需要对热门商品进行惩罚。John S.Breese在论文[4]中提出了如下计算用户相似度的公式：

该公式通过分子惩罚了用户和用户共同兴趣列表中热门商品对相似度的影响。伪码如下：

Def UserSimilarity(train):

/\*为物品建立倒排表\*/

Item\_users=dict()

For u ,items in train.items():

For I in items.keys():

If I not in item\_users:

Item\_users[i]=set()

Item\_users[i].add(u)

/\*计算用户之间的共有商品数\*/

C=dict()

N=dict()

For I ,users in item\_users.items():

For u in users:

N[u]+=1

For v in users:

If u==v:

Continue

C[u][v]+=1/math.log(1+len(users))

/\*计算最终的相似度矩阵W\*/

W=dict()

For u,related\_users in C.items():

For v,cuv in related\_users.items():

W[u][v]=cuv/math.sqrt(N[u]\*N[v])

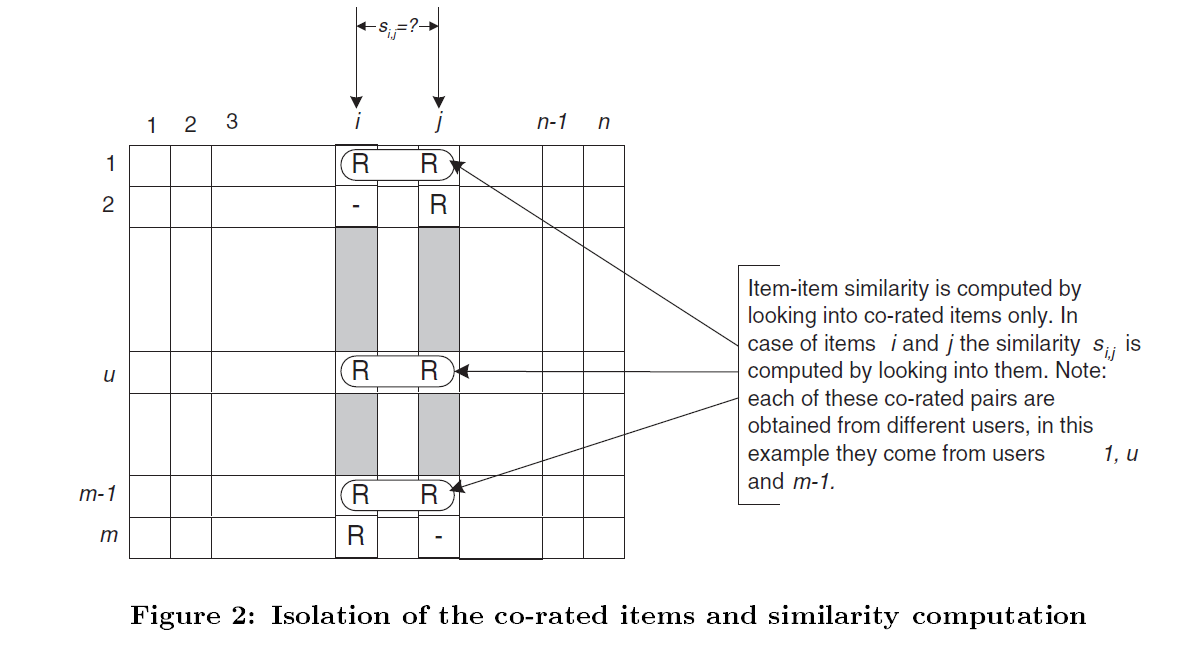
Return W

论文[5]中实现了基础的基于用户的协同过滤算法，进行POI的推荐。使用皮尔森相似度计算用户之间的相似度，打分预测使用公式(3)所示的方法，其中取值为用户对所有物品打分的均值。

* **基于物品的协同过滤算法**

基于用户的协同过滤算法有一些缺点，主要是该算法的可扩展性较差。随着用户数目的增大，计算用户相似度矩阵的时间和空间复杂度呈平方关系增长，而且难以对推荐结果做出解释。因此Amazon于2003年提出了基于物品的协同过滤算法 [6]。该算法通过寻找与目标用户喜欢的商品相似的商品来做出推荐，由于商品之间的关系相对稳定，则比起基于用户的协同过滤算法，可以在更小的在线计算量的前提下给出同样质量的推荐。这种推荐方法对应到我们日常的生活场景中可能就是我刚刚看完一本日本作家东野圭吾的推理小说《嫌疑人X的献身》，很喜欢这个作家的写作风格，可能会自己上网来搜这个作家的其他推理类作品比如《白夜行》接着看。那么现在由推荐系统来帮我们做这个事，它就会先根据我的阅读历史，然后找跟我读过的书风格，内容相似的书推荐给我，比起基于用户的推荐需要计算我与其他上千甚至上万个用户的相似度，这里只需要计算我读过的书与其他书的相似度，而商品的个数是小于用户的，增长速度也不如用户那么快，从而会大大降低计算的时间和复杂度。而且当我看到亚马逊给我推荐了《白夜行》，会很容易想到系统是根据我之前读过的东野圭吾的作品来推荐的，从而一定程度上解决了推荐结果的解释问题，使得推荐结果更容易被用户所接受。

要计算商品和 之间的相似度，首先需要找出对这两个商品都打过分的用户，然后计算。论文[7]描述了这一过程：



其中矩阵中的每一行表示一个用户，每一列表示一个商品。

计算物品之间相似度的方法上面已经介绍过，这里以Cosine Similarity为例：

其中是喜欢物品的用户数，是喜欢物品的用户数。可以看出要预测目标用户对新物品的偏好程度，依赖于用户对历史记录中与新物品相似的物品的行偏好程度。在实际生活中，大多数人的兴趣范围是有限的，可能只局限于几个领域，所以可以推测出如果两个物品同时出现在一个用户的历史兴趣列表中，可能这两个物品就在有限的几个领域中，所以会有比较高的相似度。

具体的计算中仍需先建立用户-物品倒排表（即对每个用户建立一个包含他喜欢的物品的列表），然后对于每个用户，将他物品列表中的物品两两在共现矩阵C中加1，伪码如下：

Def ItemSimilarity(train):

C=dict()

N=dict()

For u ,items in train.items():

For I in items”

N[i]+=1

For j in items:

If i==j:

Continue

C[i][j]+=1

W=dict()

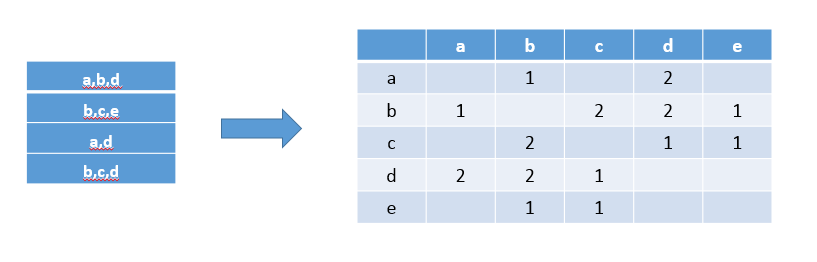
For I related\_items in C.items():

For j ,cij in related\_items.items():

W[i][j]=cij/math.sqrt(N[i]\*N[j])

Return W

下面给出一个具体的例子。下图最左边是输入的用户感兴趣的物品列表，每一行代编一个用户的感兴趣物品集合，由此可以得到共现矩阵C，其中C[i][j]记录了同时喜欢物品i和物品j的用户数。



算出物品之间的相似度后，基本的ItemCF使用如下公式来计算用户对物品的偏好程度：

其中是用户喜欢的物品集合，是与最相似的K个物品，是物品之间的相似度，是用户对物品的兴趣。伪码实现如下：

Def Recommendation(train , user\_id , W , k):

rank=dict()

ru=train[user\_id]

for i, pi in ru.items():

for j,wj in sorted(W[i].items(),/

key=itemgetter(1),reverse=True)[0:K]:

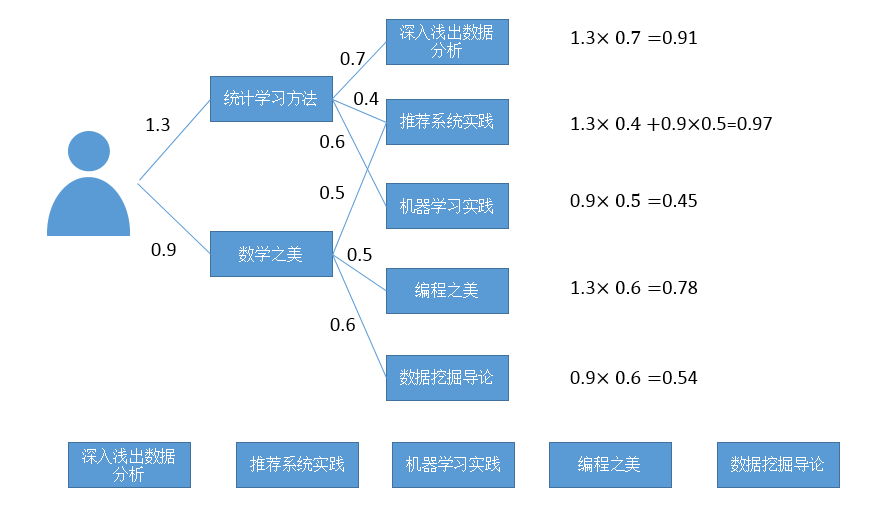
if j in ru:

continue

rank[j]+=pi\*wj

return rank

下图是基于物品的协同过滤算法用于推荐的简单例子：



可以看出，ItemCF的优点是可以对推荐结果做出解释，使用用户之前喜欢的物品来为推荐结果做出解释。给出解释的ItemCF算法伪码如下：

Def Recommendation(train,user\_id,W,K):

rank=dict()

ru=train[user\_id]

for I,pi in ru.items():

for j,wj in sorted(W[i].items),/

key=itemgetter(1).reverse=True)[0:K]:

if j in ru:

continue

rank[j].weight+=pi\*wj

rank[j].reason[i]=pi\*wj

return rank

**用户活跃度对物品相似度的影响**

由上面的计算公式可以看出相似的物品是因为它们在很多用户的历史行为列表中同时出现过，那是不是每个用户的贡献都是一样的？

假设有个用户是开饰品店的，他从淘宝网上买了很多包准备自己来卖，假设他的购物记录中包含淘宝网50%的包，假设淘宝网上有80万个包，那么就会因为这一个用户，产生一个40万乘上40万的稠密矩阵。但我们可以发看出这个用户买这些包并不是出于自己的想去，那么这个用户对于他所购买包的两两相似度的贡献应该小于只买了几个自己喜欢的包的用户。

论文[8]中，John S.Breese提出了一种方法：IUF(Inverse User Frequence)，即用户活跃度对数的倒数的参数，用来惩罚活跃用户，通过增加IUF参数来修正物品相似度的公式为：

而在实际应用中，为了避免相似度矩阵过于稠密，一般直接忽略过于活跃的用户的兴趣列表。实验表明这种方法明显提高了推荐结果的覆盖率，降低了推荐结果的流行度。这个改进也告诉我们通常一种基本的方法是难以应对真实问题中的多种情境的，如果用来解决所有的情况，可能会出现较大的偏差，所以需要我们多发现一些不同于普通情境的例子，针对此进行公式或方法上的微调。

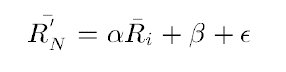
**物品相似度的归一化**

[9]中提出将ItemCF的相似度矩阵按照最大值归一化可以提高推荐的准确率。若已经得到了物品相似度矩阵w，归一化公式可以表示为：

归一化是常用的一种处理计算结果的方式，意为通过某种方式把结果转化为[0,1]之间的值。正式的定义为是一种无量纲处理手段，使物理系统数值的绝对值变成某种相对值关系。归一化处理物品之间的相似度后，还可以提高推荐结果的多样性，以下面这个例子来看，一般同一类物品之间的相似度要高于不同类物品之间的相似度，假设有两种类型的书：名人传记和推理小说，名人传记类图书之间的相似度均为0.4，而推理小说类图书之间的相似度为0.6，他们之间的相似度为0.1，如果不做归一化进行推荐，则推荐列表中都是推理小说类书籍，而进行组内按最大值归一化后可得名人传记类书籍之间的相似度均为1，而推理小说类书籍之间相似度也为1，这样推荐列表中就会包含这两类书籍，从而提高了推荐结果的多样性。

**使用回归模型进行评分预测**

[7]中提到了使用回归模型来得到打分的近似值。这种方法不用用户u对与i相似的商品打分的真实值，而是通过回归模型来得到打分的近似值。这是为了克服在使用余弦相似度或correlation相似度时，可能会出现两个打分向量的欧式距离很远，但计算得到的相似度却很大这种误差。所以使用线性回归模型来得到近似值。本文使用和来表示与目标商品i及相似商品N相关的打分向量：



在实际的应用中，考虑到商品的特征及属性一般比较固定，会预计算商品之间的相似度，由于实际推荐中只用到较小比例的相似商品来给出推荐，预计算过程只计算与每个商品最相似的前K个商品。与之相关的推荐流程是要预测用户u对商品i的打分，首先检索与i相似的前k个商品，然后计算这k个商品与用户u的购买记录的交集，接下来就使用基础的基于用户的协同过滤算法得到推荐结果。

* **加入POI地理特征的POI推荐**

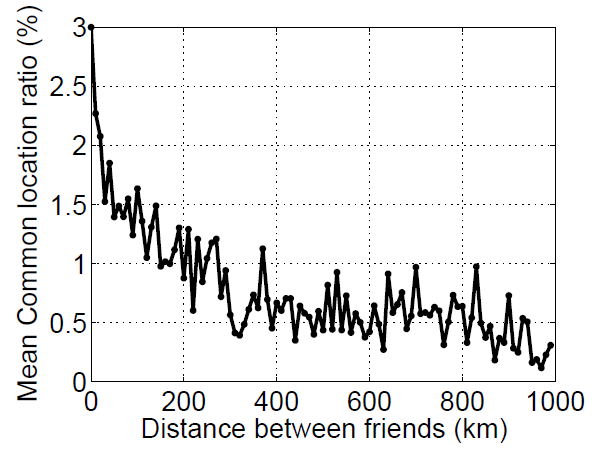
在针对POI的推荐中，POI的地理位置是一个重要的特征，融合该特征的基于内存的协同过滤推荐框架利用POI的地理位置通常有两张方式：

1. [10]中通过对用户行为数据的分析得到：用户倾向于访问离自己比较近的POI，称为proximity constraint。基于这一发现，在寻找POI候选集时

设定阀值来过滤掉离用户当前位置距离超出的候选POI。可以根据应用的需求来调整，避免最后的推荐结果中出现离用户当前位置很远的POI。通过提前划定推荐候选集的范围，减少了后续的计算量。

2．[11]中利用朋友之间所处POI的距离来计算用户之间共同访问过的POI比例。

这是基于朋友之间共同访问过的POI的比例是随朋友间距离变化呈长尾分布的特征。这里朋友之间共同访问过的POI比例随朋友之间的距离变化的曲线为：

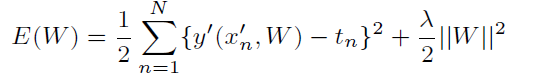


其中横轴表示朋友之间的距离，纵轴表示用户访问过的POI重合度。POI重合度定义为：，表示用户。可以看出朋友之间的距离越近，则共同访问同一个POI的几率就越大。随着朋友间距离的增大，他们之间的POI重合度下降。但尾部曲线并没有迅速坠落到零，而是贴近于横轴，近似服从长尾分布。

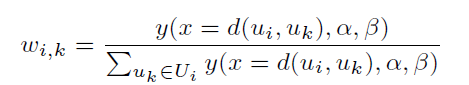
设该长尾分布为,其中表示朋友之间的距离，表示共同访问的POI比例。据此构建一个线性模型



其中, ,与是线性系数，为防止过拟合，本文使用最小二乘方误差的方法来确定权重系数：



其中N代表输入数据集的基数，是与相对应的ground truth,是正则化项。则两个朋友和之间的相似度就可以表示为：



用计算朋友之间地理位置的距离代替扫描用户和他的朋友访问过的所有地点来计算相似度，节省了计算用户之间相似度的时间。

* **加入时间特征的POI推荐（现在只有1篇论文）**

日常生活中人们的行为受时间的影响很大[12]，比如中午12点左右是大多数人的就餐时间，而周五的晚上和周末人们大多会选择娱乐活动，人们的行为不仅随着一天中的不同时段有所不同，在工作日和休息日也会呈现不同的规律。在POI签到数据中通过不同时段用户的签到数据可以反映出一天中的不同时段人们的活动呈现不同的规律，挖掘每个用户的随时间的活动规律对POI的推荐有很大的帮助。比如签到记录显示大多数用户在午夜时分选择去酒吧，只有极少数的用户选择去图书馆，那么在午夜为用户推荐的POI列表中酒吧应该具有更高的优先级。[13]为了探索用户在不同时间的POI偏好，将用户每天的签到记录按小时划分为24个时间段内。则用户在某一特定时间的行为用该时刻用户的签到记录来表示。为了利用时间这一维度，把时间这一维加入到传统的user-POI矩阵中，即user-time-POI(UTP)cube来表示带时间的签到记录。在这个立方体中，每个元素表示用户在时间对POI 的签到行为，若用户访问过该地，，否则值为0。

这里通过扩展余弦相似度来计算用户和之间的相似度：

接下来计算在时刻，用户会访问地点的可能性，即推荐指数：

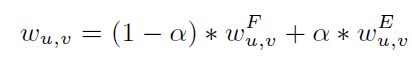
* **加入社交因素的POI推荐 （与地理因素部分共用两篇论文）**

日常生活中人们的行为受朋友的影响很大，人们周末外出的娱乐活动很少会单独行动，多会和朋友一起外出，所以有时候通过某个人的朋友的位置可以推测出这个人的位置，或者可以认为一个人的朋友所在地对这个人来说也是很有吸引力的。现在社交网络十分受用户青睐，活跃用户数量较大，用户在发布状态时通常会附带当前所在地，也为这类研究提供了丰富的数据来源。诸多研究工作也表明在POI的推荐中考虑社交因素，在参考相似用户外，考虑参考朋友的访问记录通常会得到更好的效果。

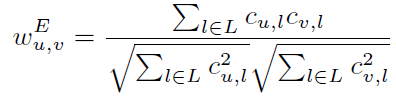
[10]和[11]都考虑到了目标用户的朋友对推荐的贡献，但利用的方式有所不同，[10]在计算用户之间相似度时同时考虑了朋友和相似用户的影响，而[11]中用来推荐的用户只有目标用户的朋友。

下面详细介绍一下两者利用社交关系的方式：

[10]提出了UPS-CF方法，使用user-based CF相似的计算公式，不同之处是的计算，这里的融合了相似用户和朋友的影响，使用参数来平衡相似用户的权重和朋友的权重,定义如下：

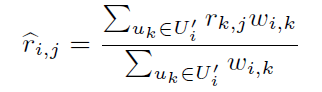


其中



而用户和用户之间的友谊使用来表示，若用户和用户之间是朋友关系，则，否则。定义=。

[11]提出了 FCF(Friend-based Collaborative Filtering)模型，该模型中只考虑朋友对推荐的贡献。FCF在计算用户与他的朋友之间的相似度时使用余弦相似度。用来表示用户对POI的打分。则打分预测值计算公式如下：



其中是用户的top-m个最相似的朋友。

**User-based CF vs Item-based CF**

下面是user-CF和item-CF的优缺点对比：

|  |  |  |
| --- | --- | --- |
|  | UserCF | ItemCF |
| 性能 | 适用于用户较少的场合，若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 |
| 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 | 新用户对很少的物品产生行为后，不能立即给出个性化推荐，因为用户相似度表示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，就能给他推荐和该物品相关的其他物品 |
| 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出推荐结束，可以令用户比较信服 |

UserCF的推荐结果着重于反映和用户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，即UserCF的推荐更社会化，ItemCF的推荐更加人性化，反映了用户自己的兴趣传承。

UserCF需要维护一个用户相似度的矩阵，而ItemCF需要维护一张物品相关度的表，若用户很多，那么维护用户兴趣相似度矩阵需要很大的空间，同理如果物品很多，那么维护物品相似度矩阵代价较大。现在的研究主要在于如何进行预计算，在线的计算量，从而提升实时推荐的效果。

后续调整格式

[1]讨论班P163 Modeling relationships at multiple scales to improve accuracy of large recommender systems.

[2]Scalable collaborative filtering with jointly derived neighborhood interpolation weights.

[3] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D.(1992). Using Collaborative Filtering to Weave anInformation Tapestry. Communications of the ACM.December.

[4]Empirical Analysis of predictive algorithms for collaborative filtering

[5] A study of recommending locations on location-based social network by collaborative filtering

[6]Amazon.com Recommendations : Item-to-Item Collaborative filtering

[7] Item-Based Collaborative Filtering Recommendation Algorithms

[8]Empirical Analysis of Predictive Algorithms for Collaborative filtering

[9]Evaluation of item-based Top-N Recommendation algorithms

[10] Location Recommendation for Out-of-Town Users in

Location-Based Social Networks 被引2

[11] Location Recommendation for Location-based

Social Networks 被引122

[12] A. Noulas, S. Scellato, C. Mascolo, and M. Pontil. An empirical study of geographic user activity patterns in foursquare. In ICWSM, 2011.

[13] Time-aware Point-of-interest Recommendation 被引50

[14] Celma Ò. Music Recommendation and Discovery in the Long Tail[J]. Ceedings of International Congress on Electron Microscopy Methods Enzymol–89, 2010, 11(1):7-8.

[15] T.M. Cover, ”Rates of Convergence for Nearest Neighbor Procedures,” *Proc. Hawaii Int’l Conf. Systems Sciences,* pp. 413-415, Western Periodicals, Honolulu, 1968