推荐系统之协同过滤（CF）算法详解和实现

21CTO 2015-07-15

1 集体智慧和协同过滤

1.1 什么是集体智慧（社会计算）？

集体智慧 (Collective Intelligence) 并不是 Web2.0 时代特有的，只是在 Web2.0 时代，大家在 Web 应用中利用集体智慧构建更加有趣的应用或者得到更好的用户体验。集体智慧是指在大量的人群的行为和数据中收集答案，帮助你对整个人群得到统计意义上的结论，这些结论是我们在单个个体上无法得到的，它往往是某种趋势或者人群中共性的部分。

Wikipedia 和 Google 是两个典型的利用集体智慧的 Web 2.0 应用：

- Wikipedia 是一个知识管理的百科全书，相对于传统的由领域专家编辑的百科全书，Wikipedia 允许最终用户贡献知识，随着参与人数的增多，Wikipedia 变成了涵盖各个领域的一本无比全面的知识库。也许有人会质疑它的权威性，但如果你从另一个侧面想这个问题，也许就可以迎刃而解。在发行一本书时，作者虽然是权威，但难免还有一些错误，然后通过一版一版的改版，书的内容越来越完善。而在 Wikipedia 上，这种改版和修正被变为每个人都可以做的事情，任何人发现错误或者不完善都可以贡献他们的想法，即便某些信息是错误的，但它一定也会尽快的被其他人纠正过来。从一个宏观的角度看，整个系统在按照一个良性循环的轨迹不断完善，这也正是集体智慧的魅力。
- Google：目前最流行的搜索引擎，与 Wikipedia 不同，它没有要求用户显式的贡献，但仔细想想 Google 最核心的 PageRank 的思想，它利用了 Web 页面之间的关系，将多少其他页面链接到当前页面的数目作为衡量当前页面重要与否的标准；如果这不好理解，那么你可以把它想象成一个选举的过程，每个 Web 页面都是一个投票者同时也是一个被投票者，PageRank 通过一定数目的迭代得到一个相对稳定的评分。Google 其实利用了现在 Internet 上所有 Web 页面上链接的集体智慧，找到哪些页面是重要的。

1.2 什么是协同过滤？

协同过滤是利用集体智慧的一个典型方法。要理解什么是协同过滤 (Collaborative Filtering, 简称 CF)，首先想一个简单的问题，如果你现在想看个电影，但你不知道具体看哪部，你会怎么做？大部分的人会问问周围的朋友，看看最近有什么好看的电影推荐，而我们一般更倾向于从口味比较类似的朋友那里得到推荐。这就是协同过滤的核心思想。

协同过滤一般是在海量的用户中发掘出一小部分和你品位比较类似的，在协同过滤中，这些用户成为邻居，然后根据他们喜欢的其他东西组织成一个排序的目录作为推荐给你。当然其中有一个核心的问题：

- 如何确定一个用户是不是和你有相似的品位？
- 如何将邻居们的喜好组织成一个排序的目录？

协同过滤相对于集体智慧而言，它从一定程度上保留了个体的特征，就是你的品位偏好，所以它更多可以作为个性化推荐的算法思想。可以想象，这种推荐策略在 Web 2.0

的长尾中是很重要的，将大众流行的东西推荐给长尾中的人怎么可能得到好的效果，这也回到推荐系统的一个核心问题：了解你的用户，然后才能给出更好的推荐。

2 深入协同过滤的核心

前面作为背景知识，介绍了集体智慧和协同过滤的基本思想，这一节我们将深入分析协同过滤的原理，介绍基于协同过滤思想的多种推荐机制，优缺点和实用场景。

首先，要实现协同过滤，需要一下几个步骤

- 收集用户偏好
- 找到相似的用户或物品
- 计算推荐

2.1 收集用户偏好

要从用户的行为和偏好中发现规律，并基于此给予推荐，如何收集用户的偏好信息成为系统推荐效果最基础的决定因素。用户有很多方式向系统提供自己的偏好信息，而且不同的应用也可能大不相同，下面举例进行介绍：

表 1 用户行为和用户偏好

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
转发	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。 如果是站内，同时可以推理得到被转发人的偏好（不精确）
保存书签	显示	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。
标记标签(Tag)	显示	一些单词，需要对单词进行分析，得到偏好	通过分析用户的标签，可以得到用户对项目的理解，同时可以分析出用户的情感：喜欢还是讨厌
评论	显示	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

点击流（查看）	隐式	一组用户的点击，用户对物品感兴趣，需要进行分析，得到偏好	用户的点击一定程度上反映了用户的注意力，所以它也可以从一定程度上反映用户的喜好。
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。

以上列举的用户行为都是比较通用的，推荐引擎设计人员可以根据自己应用的特点添加特殊的用户行为，并用他们表示用户对物品的喜好。

在一般应用中，我们提取的用户行为一般都多于一种，关于如何组合这些不同的用户行为，基本上有以下两种方式：

- 将不同的行为分组：一般可以分为“查看”和“购买”等等，然后基于不同的行为，计算不同的用户 / 物品相似度。类似于当当网或者 Amazon 给出的“购买了该图书的人还购买了 ...”，“查看了图书的人还查看了 ...”
- 根据不同行为反映用户喜好的程度将它们进行加权，得到用户对于物品的总体喜好。一般来说，显式的用户反馈比隐式的权值大，但比较稀疏，毕竟进行显示反馈的用户是少数；同时相对于“查看”，“购买”行为反映用户喜好的程度更大，但这也因应用而异。

收集了用户行为数据，我们还需要对数据进行一定的预处理，其中最核心的工作就是：减噪和归一化。

- 减噪：用户行为数据是用户在使用应用过程中产生的，它可能存在大量的噪音和用户的误操作，我们可以通过经典的数据挖掘算法过滤掉行为数据中的噪音，这样可以是我们的分析更加精确。
- 归一化：如前面讲到的，在计算用户对物品的喜好程度时，可能需要对不同的行为数据进行加权。但可以想象，不同行为的数据取值可能相差很大，比如，用户的查看数据必然比购买数据大的多，如何将各个行为的数据统一在一个相同的取值范围中，从而使得加权求和得到的总体喜好更加精确，就需要我们进行归一化处理。最简单的归一化处理，就是将各类数据除以此类中的最大值，以保证归一化后的数据取值在 [0,1] 范围中。

进行的预处理后，根据不同应用的行为分析方法，可以选择分组或者加权处理，之后我们可以得到一个用户偏好的二维矩阵，一维是用户列表，另一维是物品列表，值是用户对物品的偏好，一般是 [0,1] 或者 [-1, 1] 的浮点数值。

2.2 找到相似的用户或物品

当已经对用户行为进行分析得到用户喜好后，我们可以根据用户喜好计算相似用户和物品，然后基于相似用户或者物品进行推荐，这就是最典型的 CF 的两个分支：基于用户的 CF 和基于物品的 CF。这两种方法都需要计算相似度，下面我们先看看最基本的几种计算相似度的方法。

相似度的计算

关于相似度的计算，现有的几种基本方法都是基于向量（Vector）的，其实也就是计算两个向量的距离，距离越近相似度越大。在推荐的场景中，在用户 - 物品偏好的二维矩阵中，我们可以将一个用户对所有物品的偏好作为一个向量来计算用户之间的相似度，或者将所有用户对某个物品的偏好作为一个向量来计算物品之间的相似度。下面我们详细介绍几种常用的相似度计算方法：

- 欧几里德距离（Euclidean Distance）

最初用于计算欧几里德空间中两个点的距离，假设 x，y 是 n 维空间的两个点，它们之间的欧几里德距离是：

$$d(x,y)=\sqrt{(\sum(x_i-y_i)^2)}$$

可以看出，当 n=2 时，欧几里德距离就是平面上两个点的距离。

当用欧几里德距离表示相似度，一般采用以下公式进行转换：距离越小，相似度越大

$$sim(x,y)=\frac{1}{1+d(x,y)}$$

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

皮尔逊相关系数一般用于计算两个定距变量间联系的紧密程度，它的取值在 [-1，+1] 之间。

$$p(x,y)=\frac{\sum x_iy_i-n\overline{x}\overline{y}}{(n-1)s_xs_y}=\frac{n\sum x_iy_i-\sum x_i\sum y_i}{\sqrt{n\sum x_i^2-(\sum x_i)^2}\sqrt{n\sum y_i^2-(\sum y_i)^2}}$$

sx，sy是 x 和 y 的样品标准偏差。

- Cosine 相似度（Cosine Similarity）

Cosine 相似度被广泛应用于计算文档数据的相似度：

$$T(x,y)=\frac{x\bullet y}{\|x\|^2\times\|y\|^2}=\frac{\sum x_iy_i}{\sqrt{\sum x_i^2}\sqrt{\sum y_i^2}}$$

- Tanimoto 系数（Tanimoto Coefficient）

Tanimoto 系数也称为 Jaccard 系数，是 Cosine 相似度的扩展，也多用于计算文档数据的相似度：

$$T(x,y)=\frac{x\bullet y}{\|x\|^2+\|y\|^2-x\bullet y}=\frac{\sum x_iy_i}{\sqrt{\sum x_i^2}+\sqrt{\sum y_i^2}-\sum x_iy_i}$$

相似邻居的计算

介绍完相似度的计算方法，下面我们看看如何根据相似度找到用户 - 物品的邻居，常用的挑选邻居的原则可以分为两类：图 1 给出了二维平面空间上点集的示意图。

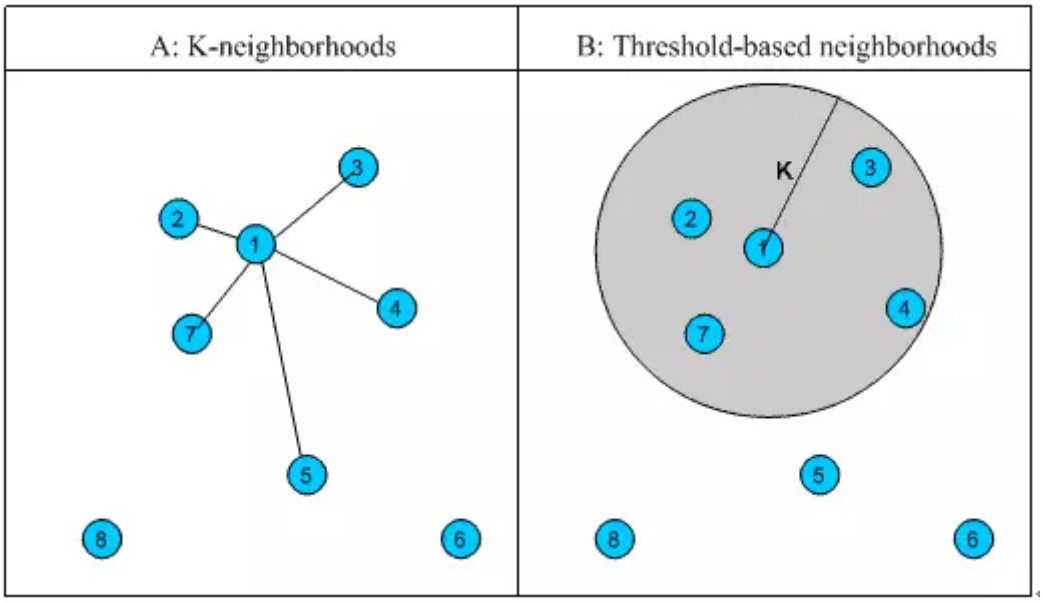
- 固定数量的邻居：K-neighborhoods 或者 Fix-size neighborhoods

不论邻居的“远近”，只取最近的 K 个，作为其邻居。如图 1 中的 A，假设要计算点 1 的 5- 邻居，那么根据点之间的距离，我们取最近的 5 个点，分别是点 2，点 3，点 4，点 7 和点 5。但很明显我们可以看出，这种方法对于孤立点的计算效果不好，因为要取固定个数的邻居，当它附近没有足够多比较相似的点，就被迫取一些不太相似的点作为邻居，这样就影响了邻居相似的程度，比如图 1 中，点 1 和点 5 其实并不是很相似。

- 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。

图 1.相似邻居计算示意图



回到顶部

2.3 计算推荐

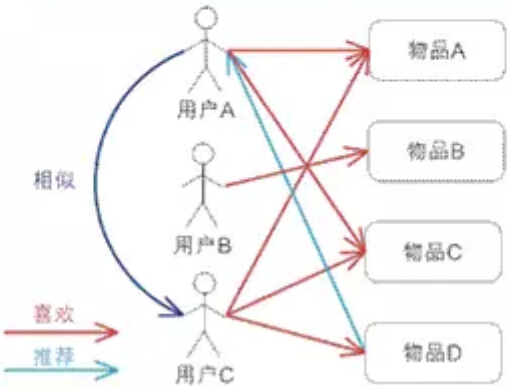
经过前期的计算已经得到了相邻用户和相邻物品，下面介绍如何基于这些信息为用户进行推荐。本系列的上一篇综述文章已经简要介绍过基于协同过滤的推荐算法可以分为基于用户的 CF 和基于物品的 CF，下面我们深入这两种方法的计算方法，使用场景和优缺点。

基于用户的 CF（User CF）

基于用户的 CF 的基本思想相当简单，基于用户对物品的偏好找到相邻邻居用户，然后将邻居用户喜欢的推荐给当前用户。计算上，就是将一个用户对所有物品的偏好作为一个向量来计算用户之间的相似度，找到 K 邻居后，根据邻居的相似度权重以及他们对物品的偏好，预测当前用户没有偏好的未涉及物品，计算得到一个排序的物品列表作为推荐。图 2 给出了一个例子，对于用户 A，根据用户的历史偏好，这里只计算得到一个邻居 - 用户 C，然后将用户 C 喜欢的物品 D 推荐给用户 A。

图 2.基于用户的 CF 的基本原理

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	√		√	√

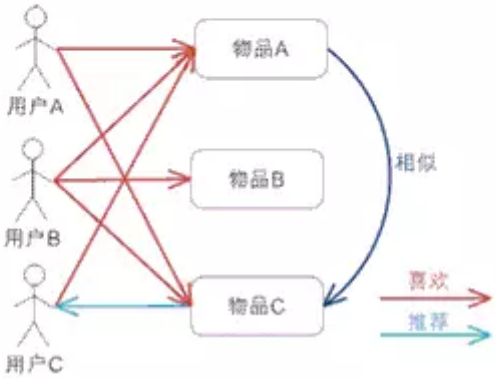


基于物品的 CF（Item CF）

基于物品的 CF 的原理和基于用户的 CF 类似，只是在计算邻居时采用物品本身，而不是从用户的角度，即基于用户对物品的偏好找到相似的物品，然后根据用户的历史偏好，推荐相似的物品给他。从计算的角度看，就是将所有用户对某个物品的偏好作为一个向量来计算物品之间的相似度，得到物品的相似物品后，根据用户历史的偏好预测当前用户还没有表示偏好的物品，计算得到一个排序的物品列表作为推荐。图 3 给出了一个例子，对于物品 A，根据所有用户的历史偏好，喜欢物品 A 的用户都喜欢物品 C，得出物品 A 和物品 C 比较相似，而用户 C 喜欢物品 A，那么可以推断出用户 C 可能也喜欢物品 C。

图 3.基于物品的 CF 的基本原理

用户/物品	物品A	物品B	物品C
用户A	√		√
用户B	√	√	√
用户C	√		推荐



User CF vs. Item CF

前面介绍了 User CF 和 Item CF 的基本原理，下面我们分几个不同的角度深入看看它们各自的优缺点和适用场景：

- 计算复杂度

Item CF 和 User CF 是基于协同过滤推荐的两个最基本的算法，User CF 是很早以前就提出来了，Item CF 是从 Amazon 的论文和专利发表之后（2001 年左右）开始流行，大家都觉得 Item CF 从性能和复杂度上比 User CF 更优，其中的一个主要原因就是对于一个在线网站，用户的数量往往大大超过物品的数量，同时物品的数据相对稳定，因此计算物品的相似度不但计算量较小，同时也不必频繁更新。但我们往往忽略了这种情况只适应于提供商品的电子商务网站，对于新闻，博客或者微内容的推荐系统，情况往往是相反的，物品的数量是海量的，同时也是更新频繁的，所以单从复杂度的角度，这两个算法在不同的系统中各有优势，推荐引擎的设计者需要根据自己应用的特点选择更加合适的算法。

- 适用场景

在非社交网络的网站中，内容内在的联系是很重要的推荐原则，它比基于相似用户的推荐原则更加有效。比如在购书网站上，当你看一本书的时候，推荐引擎会给你推荐相关的书籍，这个推荐的重要性远远超过了网站首页对该用户的综合推荐。可以看到，在这种情况下，Item CF 的推荐成为了引导用户浏览的重要手段。同时 Item CF 便于为推荐做出解释，在一个非社交网络的网站中，给某个用户推荐一本书，同时给出的解释是某某和你有相似兴趣的人也看了这本书，这很难让用户信服，因为用户可能根本不认识那个人；但如果解释说是因为这本书和你以前看的某本书相似，用户可能就觉得合理而采纳了此推荐。

相反的，在现今很流行的社交网络站点中，User CF 是一个更不错的选择，User CF 加上社会网络信息，可以增加用户对推荐解释的信服程度。

- 推荐多样性和精度

研究推荐引擎的学者们在相同的数据集合上分别用 User CF 和 Item CF 计算推荐结果，发现推荐列表中，只有 50% 是一样的，还有 50% 完全不同。但是这两个算法确有相似的精度，所以可以说，这两个算法是很互补的。

关于推荐的多样性，有两种度量方法：

第一种度量方法是从单个用户的角度度量，就是说给定一个用户，查看系统给出的推荐列表是否多样，也就是要比较推荐列表中的物品之间两两的相似度，不难想到，对这种度量方法，Item CF 的多样性显然不如 User CF 的好，因为 Item CF 的推荐就是和以前看的东西最相似的。

第二种度量方法是考虑系统的多样性，也被称为覆盖率 (Coverage)，它是指一个推荐系统是否能够提供给所有用户丰富的选择。在这种指标下，Item CF 的多样性要远远好于 User CF, 因为 User CF 总是倾向于推荐热门的，从另一个侧面看，也就是说，Item CF 的推荐有很好的新颖性，很擅长推荐长尾里的物品。所以，尽管大多数情况，Item CF 的精度略小于 User CF， 但如果考虑多样性，Item CF 却比 User CF 好很多。

如果你对推荐的多样性还心存疑惑，那么下面我们再举个实例看看 User CF 和 Item CF 的多样性到底有什么差别。首先，假设每个用户兴趣爱好都是广泛的，喜欢好几个领域的东西，不过每个用户肯定也有一个主要的领域，对这个领域会比其他领域更加关心。给定一个用户，假设他喜欢 3 个领域 A,B,C，A 是他喜欢的主要领域，这个时候我们来看 User CF 和 Item CF 倾向于做出什么推荐：如果用 User CF, 它会将

A,B,C 三个领域中比较热门的东西推荐给用户；而如果用 ItemCF，它会基本上只推荐 A 领域的东西给用户。所以我们看到因为 User CF 只推荐热门的，所以它在推荐长尾里项目方面的能力不足；而 Item CF 只推荐 A 领域给用户，这样他有限的推荐列表中就可能包含了一定数量的不热门的长尾物品，同时 Item CF 的推荐对这个用户而言，显然多样性不足。但是对整个系统而言，因为不同的用户的主要兴趣点不同，所以系统的覆盖率会比较好。

从上面的分析，可以很清晰的看到，这两种推荐都有其合理性，但都不是最好的选择，因此他们的精度也会有损失。其实对这类系统的最好选择是，如果系统给这个用户推荐 30 个物品，既不是每个领域挑选 10 个最热门的给他，也不是推荐 30 个 A 领域的给他，而是比如推荐 15 个 A 领域的给他，剩下的 15 个从 B,C 中选择。所以结合 User CF 和 Item CF 是最优的选择，结合的基本原则就是当采用 Item CF 导致系统对个人推荐的多样性不足时，我们通过加入 User CF 增加个人推荐的多样性，从而提高精度，而当因为采用 User CF 而使系统的整体多样性不足时，我们可以通过加入 Item CF 增加整体的多样性，同样同样可以提高推荐的精度。

- 用户对推荐算法的适应度

前面我们大部分都是从推荐引擎的角度考虑哪个算法更优，但其实我们更多的应该考虑作为推荐引擎的最终使用者 -- 应用用户对推荐算法的适应度。

对于 User CF，推荐的原则是假设用户会喜欢那些和他有相同喜好的用户喜欢的东西，但如果一个用户没有相同喜好的朋友，那 User CF 的算法的效果就会很差，所以一个用户对的 CF 算法的适应度是和他有多少共同喜好用户成正比的。

Item CF 算法也有一个基本假设，就是用户会喜欢和他以前喜欢的东西相似的东西，那么我们可以计算一个用户喜欢的物品的自相似度。一个用户喜欢物品的自相似度大，就说明他喜欢的东西都是比较相似的，也就是说他比较符合 Item CF 方法的基本假设，那么他对 Item CF 的适应度自然比较好；反之，如果自相似度小，就说明这个用户的喜好习惯并不满足 Item CF 方法的基本假设，那么对于这种用户，用 Item CF 方法做出好的推荐的可能性非常低。

3. 基于KNN的协同过滤推荐算法MATLAB实现

邻居模型通常也被称为k-最近邻模型，或者简称为kNN。KNN 模型可以获得精确的推荐结果并为结果给出合理的解释，它们是CF 推荐系统中最早被使用也是直至目前最流行的一类模型。

PS：以下公式和图片转自博主自己的CSDN博客。

为了获得用户对产品的评分预测值，kNN 模型一般包括以下三步：

1. 计算相似度

这步中计算每对产品之间的相似度（similarity）。一些被广泛使用的相似度测度包括：

Pearson correlation：

$$s_{mn}^{p}=\frac{\sum_{v\in\mathcal{P}mn}\left(r_{v,m}-\bar{r}^{m}\right)\left(r_{v,n}-\bar{r}^{n}\right)}{\sqrt{\sum_{v\in\mathcal{P}mn}\left(r_{v,m}-\bar{r}^{m}\right)^{2}\sum_{v\in\mathcal{P}mn}\left(r_{i,n}-\bar{r}^{n}\right)^{2}}}$$

其中¯rm 和¯rn 分别表示电影m 和n 获得的评分平均值，而Pmn 表示对电影m 和n 都提供了评分的用户集合，也即Pmn = Pm ∩ Pn 。

Cosine：

$$s^{c}_{mn} = \frac{\sum_{v \in P^{mn}} r_{v,m} r_{v,n}}{\sqrt{\sum_{v \in P^{mn}} r^{2}_{v,m} \sum_{v \in P^{mn}} r^{2}_{v,n}}}$$

Adjusted Cosine：

$$s^{ac}_{mn} = \frac{\sum_{v \in P^{mn}} (r_{v,m} - \bar{r}_{v})(r_{v,n} - \bar{r}_{v})}{\sqrt{\sum_{v \in P^{mn}} (r_{v,m} - \bar{r}_{v})^{2} \sum_{v \in P^{mn}} (r_{v,n} - \bar{r}_{v})^{2}}}$$

其中¯rv 表示用户v 的评分平均值。

2. 选择邻居

为了预测用户u 对电影m 的评分值，我们首先从Pu 中选取与电影m有最高相似度的特定数量的电影，这些电影形成u−m 对的邻居（neighborhood），记为N(m; u) 。

3. 产生预测值

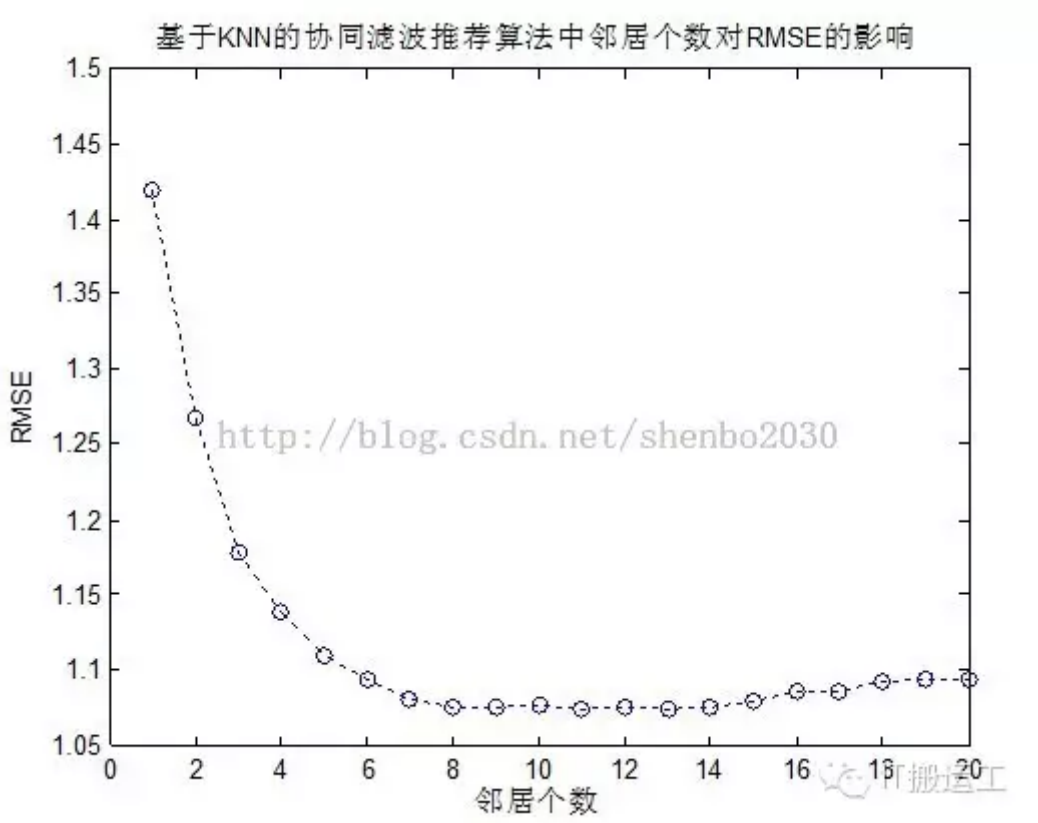
用户u 对电影m 的评分预测为上步获得的邻居N(m; u) 中评分的加权平均值：

$$\hat{r}_{u,m} = b_{u,m} + \frac{\sum_{n \in N(m;u)} s_{mn}(r_{u,n} - b_{u,n})}{\sum_{n \in N(m;u)} s_{mn}}$$

其中bu;n 为用户u 对电影n 的基准预测评分。这里的基准模型可以是任何可以产生预测评分的模型。

按照上述过程，在MATLAB仿真环境下得到的RMSE=1.0776，这里取得邻居个数为10。

下图为邻居个数的选取（0~20）对RMSE的影响曲线：




```
%% 载入训练数据
 load g:\matlab\协同过滤做推荐\dataset\Movielens\u1.base%% 数据预处理
%  提取数据的前三列，即用户序号、被该用户评价电影序号、评价分值
[m,n]=size(u1);
test=zeros(m,3);for i=1:3
    test(:,i)=u1(:,i);
end
%% 建立评分矩阵
number_user=max(test(:,1));
number_movies=max(test(:,2));
score_matrix=zeros(number_user,number_movies);%评分矩阵943*1682维for i=1
    score_matrix(test(i,1),test(i,2))=test(i,3);
end
Sim_matrix=zeros(number_movies,number_movies);%相似度矩阵1642*1642维
tic;
%计算评分矩阵for i=1:number_movies-1
    for j=i+1:number_movies
        Sim_matrix(i,j)=Similarity_ab(score_matrix,i,j);
    end
end
toc;
%% 建立相似度矩阵
% function Neibor=neibor_select(Sim_matrix,a,n)
```

```
neibor_num=10;%邻居的大小
Sim_matrix=Sim_matrix'+Sim_matrix;%求完整的相似度矩阵%neibor_sim_matrix_
%neibor_matrix_temp各个相似度所对应的电影，也就是我们要找的邻居
value_1_index=find(Sim_matrix>=0.9999);%找出Sim_matrix矩阵中所有相似度为1
                                       %因为可能是错误值，后期选择邻居不应
%为什么不是value_1_index=find(Sim_matrix==1)这样有部分1不能正确找出，可以
Sim_matrix(value_1_index)=0;%将所有相似度为1的值用0代替
% [neibor_sim_matrix_temp,neibor_matrix_temp]=sort(Sim_matrix,2,'descend
% neibor_sim_matrix=zeros(number_movies,neibor_num);
% neibor_matrix=zeros(number_movies,neibor_num);
% for i=1:neibor_num
%     neibor_sim_matrix(:,i)=neibor_sim_matrix_temp(:,i);%每个邻居对应的
%     neibor_matrix(:,i)=neibor_matrix_temp(:,i);%邻居
% end
%% 载入测试集
load g:\matlab\协同过滤做推荐\dataset\Movielens\u1.test
%% 作预测
[m,n]=size(u1);
test=zeros(m,3);for i=1:3
    test(:,i)=u1(:,i);
end
Predict_score=zeros(m,1);for j=1:m
P_u=find(score_matrix(test(j,1),:)~=0);%找出该用户评价的电影集合
[~,num]=size(P_u);%计算该用户评价的电影个数
%%%%%%%%%%计算邻居%%%%%%%%%%%%
neibor_num=10;%最大为4
P_u_sim=Sim_matrix(test(j,2),P_u);
[temp,index]=sort(P_u_sim,2,'descend');
[~,num1]=size(index);if num1>=neibor_num
neibor=(P_u(index(1:neibor_num)));else
    neibor=(P_u(index));
    neibor_num=num1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sum_score=sum(score_matrix(test(j,1),:),2);%该用户对所有电影的总评分
aver_score=sum_score/num;%该用户对电影的平均评分
sum1=0;
sum2=0;for i=1:neibor_num
    sum1=sum1+Sim_matrix(test(j,2),neibor(i))*(score_matrix(test(j,1),ne
    sum2=sum2+Sim_matrix(test(j,2),neibor(i));
endif sum2==0
```

```
    Predict_score(j,1)=round(aver_score);%排除分母为零的情况elsePredict_s
%确保预测值为1~5的评分数if Predict_score(j,1)>5
    Predict_score(j,1)=5;
elseif Predict_score(j,1)<1
    Predict_score(j,1)=1;
end
end
end
%% 计算RMSE
Eval=zeros(m,3);
Eval(:,1)=test(:,3);
Eval(:,2)=Predict_score(:,1);
Eval(:,3)=abs(test(:,3)-Predict_score(:,1));
RMSE=sqrt(Eval(:,3)'*Eval(:,3)/m);
```

协同过滤的缺点是:

（1）用户对商品的评价非常稀疏，这样基于用户的评价所得到的用户间的相似性可能不准确（即稀疏性问题）；

（2）随着用户和商品的增多，系统的性能会越来越低;

（3）如果从来没有用户对某一商品加以评价，则这个商品就不可能被推荐（即最初评价问题）。

4. 总结

Web2.0 的一个核心思想就是“集体智慧”，基于协同过滤的推荐策略的基本思想就是基于大众行为，为每个用户提供个性化的推荐，从而使用户能更快速更准确的发现所需要的信息。从应用角度分析，现今比较成功的推荐引擎，比如 Amazon，豆瓣，当当等都采用了协同过滤的方式，它不需要对物品或者用户进行严格的建模，而且不要求物品的描述是机器可理解的，是中领域无关的推荐方法，同时这个方法计算出来的推荐是开放的，可以共用他人的经验，很好的支持用户发现潜在的兴趣偏好。基于协同过滤的推荐策略也有不同的分支，它们有不同的实用场景和推荐效果，用户可以根据自己应用的实际情况选择合适的方法，异或组合不同的方法得到更好的推荐效果。

5. 参考资料

- Collective Intelligence in Action: 详细介绍了如何利用集体智慧构建智能应用。
- Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. Adomavicius, G. and Tuzhilin 在 2005 年发表的一片文章，详细总结了推荐引擎的发展和存在的问题
- Collaborative_Filtering：Wikipedia 上对于协同过滤的介绍和相关论文。
- Item-based collaborative filtering recommendation algorithms：Amazon 最早提出 Item CF 的推荐策略的论文