从图中可以很清楚的看到，首先，系统对每个用户都有一个用户 Profile 的建模，其中包括用户的基本信息，例如用户的年龄，性别等等；然后，系统会根据用户的 Profile 计算用户的相似度，可以看到用户 A 的 Profile 和用户 C 一样，那么系统会认为用户 A 和 C 是相似用户，在推荐引擎中，可以称他们是“邻居”；最后，基于“邻居”用户群的喜好推荐给当前用户一些物品，图中将用户 A 喜欢的物品 A 推荐给用户 C。

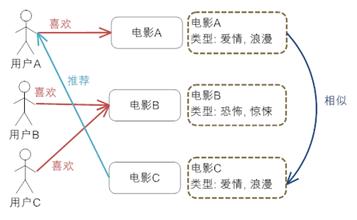
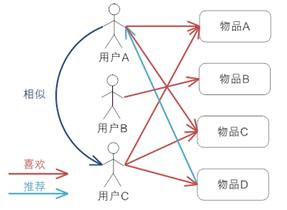
**图 3. 基于内容推荐机制的基本原理**  
 

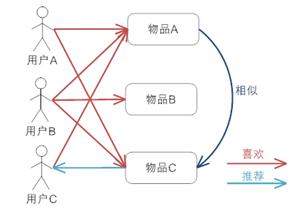
图 3 中给出了基于内容推荐的一个典型的例子，电影推荐系统，首先我们需要对电影的元数据有一个建模，这里只简单的描述了一下电影的类型；然后通过电影的元数据发现电影间的相似度，因为类型都是“爱情，浪漫”电影 A 和 C 被认为是相似的电影（当然，只根据类型是不够的，要得到更好的推荐，我们还可以考虑电影的导演，演员等等）；最后实现推荐，对于用户 A，他喜欢看电影 A，那么系统就可以给他推荐类似的电影 C。

**图 4. 基于用户的协同过滤推荐机制的基本原理**  
 

上图示意出基于用户的协同过滤推荐机制的基本原理，假设用户 A 喜欢物品 A，物品 C，用户 B 喜欢物品 B，用户 C 喜欢物品 A ，物品 C 和物品 D；从这些用户的历史喜好信息中，我们可以发现用户 A 和用户 C 的口味和偏好是比较类似的，同时用户 C 还喜欢物品 D，那么我们可以推断用户 A 可能也喜欢物品 D，因此可以将物品 D 推荐给用户 A。

假设用户 A 喜欢物品 A 和物品 C，用户 B 喜欢物品 A，物品 B 和物品 C，用户 C 喜欢物品 A，从这些用户的历史喜好可以分析出物品 A 和物品 C 时比较类似的，喜欢物品 A 的人都喜欢物品 C，基于这个数据可以推断用户 C 很有可能也喜欢物品 C，所以系统会将物品 C 推荐给用户 C。

与上面讲的类似，基于项目的协同过滤推荐和基于内容的推荐其实都是基于物品相似度预测推荐，只是相似度计算的方法不一样，前者是从用户历史的偏好推断，而后者是基于物品本身的属性特征信息。

**图 5. 基于项目的协同过滤推荐机制的基本原理**  


**混合的推荐机制**

在现行的 Web 站点上的推荐往往都不是单纯只采用了某一种推荐的机制和策略，他们往往是将多个方法混合在一起，从而达到更好的推荐效果。关于如何组合各个推荐机制，这里讲几种比较流行的组合方法。

1. 加权的混合（Weighted Hybridization）: 用线性公式（linear formula）将几种不同的推荐按照一定权重组合起来，具体权重的值需要在测试数据集上反复实验，从而达到最好的推荐效果。
2. 切换的混合（Switching Hybridization）：前面也讲到，其实对于不同的情况（数据量，系统运行状况，用户和物品的数目等），推荐策略可能有很大的不同，那么切换的混合方式，就是允许在不同的情况下，选择最为合适的推荐机制计算推荐。
3. 分区的混合（Mixed Hybridization）：采用多种推荐机制，并将不同的推荐结果分不同的区显示给用户。其实，Amazon，当当网等很多电子商务网站都是采用这样的方式，用户可以得到很全面的推荐，也更容易找到他们想要的东西。
4. 分层的混合（Meta-Level Hybridization）: 采用多种推荐机制，并将一个推荐机制的结果作为另一个的输入，从而综合各个推荐机制的优缺点，得到更加准确的推荐。

Amazon 利用可以记录的所有用户在站点上的行为，根据不同数据的特点对它们进行处理，并分成不同区为用户推送推荐：

* 今日推荐 (Today's Recommendation For You): 通常是根据用户的近期的历史购买或者查看记录，并结合时下流行的物品给出一个折中的推荐。
* 新产品的推荐 (New For You): 采用了基于内容的推荐机制 (Content-based Recommendation)，将一些新到物品推荐给用户。在方法选择上由于新物品没有大量的用户喜好信息，所以基于内容的推荐能很好的解决这个“冷启动”的问题。
* 捆绑销售 (Frequently Bought Together): 采用数据挖掘技术对用户的购买行为进行分析，找到经常被一起或同一个人购买的物品集，进行捆绑销售，这是一种典型的基于项目的协同过滤推荐机制。
* 别人购买 / 浏览的商品 (Customers Who Bought/See This Item Also Bought/See): 这也是一个典型的基于项目的协同过滤推荐的应用，通过社会化机制用户能更快更方便的找到自己感兴趣的物品。

值得一提的是，Amazon 在做推荐时，设计和用户体验也做得特别独到：

Amazon 利用有它大量历史数据的优势，量化推荐原因。

* 基于社会化的推荐，Amazon 会给你事实的数据，让用户信服，例如：购买此物品的用户百分之多少也购买了那个物品；
* 基于物品本身的推荐，Amazon 也会列出推荐的理由，例如：因为你的购物框中有 \*\*\*，或者因为你购买过 \*\*\*，所以给你推荐类似的 \*\*\*。

另外，Amazon 很多推荐是基于用户的 profile 计算出来的，用户的 profile 中记录了用户在 Amazon 上的行为，包括看了那些物品，买了那些物品，收藏夹和 wish list 里的物品等等，当然 Amazon 里还集成了评分等其他的用户反馈的方式，它们都是 profile 的一部分，同时，Amazon 提供了让用户自主管理自己 profile 的功能，通过这种方式用户可以更明确的告诉推荐引擎他的品味和意图是什么。

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

**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。 |

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。
* 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：

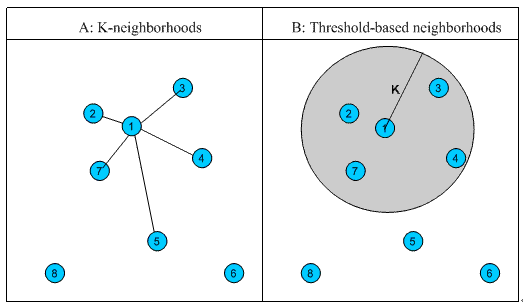
Figure xxx. Requires a heading

* 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**  


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。

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。

GenericPreference preference = new GenericPreference(123, 456, 3.0f);

这其中， 123 是用户 ID，long 型；456 是物品 ID，long 型；3.0f 是用户偏好，float 型。

Apache Mahout

文本的向量空间模型就是将文本信息建模为一个向量，其中每一个域是文本中出现的一个词的权重。关于权重的计算则有很多中：

* 最简单的莫过于直接计数，就是词在文本里出现的次数。这种方法简单，但是对文本内容描述的不够精确。
* 词的频率 (Team Frequency, TF)：就是将词在文本中出现的频率作为词的权重。这种方法只是对于直接计数进行了归一化处理，目的是让不同长度的文本模型有统一的取值空间，便于文本相似度的比较，但可以看出，简单计数和词频都不能解决“高频无意义词汇权重大的问题”，也就是说对于英文文本中，“a”，“the”这样高频但无实际意义的词汇并没有进行过滤，这样的文本模型在计算文本相似度时会很不准确。
* 词频 - 逆向文本频率 (Term Frequency – Inverse Document Frequency, TF-IDF)：它是对 TF 方法的一种加强，字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在所有文本中出现的频率成反比下降。举个例子，对于“高频无意义词汇”，因为它们大部分会出现在所有的文本中，所以它们的权重会大打折扣，这样就使得文本模型在描述文本特征上更加精确。在信息检索领域，TF-IDF 是对文本信息建模的最常用的方法。