第 10 章　推荐系统

推荐系统起源于人的依赖性：人们在做事情作决策的时候总是依赖于别人的建议。推荐系统正是将这种“我们大脑转换建议的过程”由计算机来实现。本章介绍推荐系统的基本概念和基本方法。

2　推荐系统[Charu C. Aggarwal,2016]

2.1　推荐系统概述

2.1.1　引言

作为电子和商务交易的媒介，Web 如今扮演着越来越重要的角色并推动了推荐系统技术的发展。其中一个重要的作用是能够让用户轻松地提供他喜欢或不喜欢的反馈。例如，Netflix（一家在线影片租赁提供商）的用户只需要简单地动动鼠标就能提供反馈。评分是一种提供反馈的典型方法。在某个特定的评分系统（例如五星评分系统）中，用户可以选择不同大小的数值来说明自己对不同物品的满意程度。

其他形式的反馈不像评分一样清晰明了，但却更容易采集。例如，可将用户在网上购买或是浏览一件物品的行为视为对该物品的认可。获取这类反馈形式的数据十分容易，这种方法被 Amazon.com 等网上商家采用。推荐系统的基本思想是利用这些不同来源的数据来推断顾客的喜好。推荐系统面向的对象称为用户（user），推荐的产品称为物品（item）。由于用户曾经的兴趣喜好通常预示着未来的选择，因此推荐分析也通常是基于先前用户与物品之间的关系。但仍有一个特例——基于知识的推荐系统是根据用户指定的需求面非用户的历史记录进行推荐。

本书将会讲述所有基本类型的推荐系统，包括协同系统、基于内容的系统和基于知识的系统。

2.1.2　推荐系统的目标

在讨论推荐系统的目标之前，我们先介绍推荐问题的几种不同表述方式。下面是两种主要的模型：

1)　预测模型：第一种方法是对用户—物品组合的评分值进行预测。该方法假设描述用户对物品喜好的训练数据是可用的。对于 m 个用户和 n 件物品，这个训练集相当于一个$m \times n$的不完全矩阵，矩阵中的已知值（或观测值）被用来训练。矩阵中的缺失值（或未观测值）则通过这个训练模型进行预测。因为是根据不完整的数值矩阵用学习算法预测出剩余的未知值，所以这个问题又被称作矩阵补全问题。

2)　排名模型：实际上，对用户做推荐并不需要预测出用户对具体物品的评分。商家可能希望向特定的用户推荐前 k 种（top-k）物品或者是为某个指定物品确定前 k 个（top-k）感兴趣的用户。虽然这两种算法极其类似，但是对 top-k 物品的计算比确定 top-k 用户要应用普遍，因此本书中我们只讨论对 top-k 物品的计算。这个问题也被叫作 top-k 推荐问题，它是推荐问题的排名模型。

在第二种情况下，对评分的准确值的预测并不重要。由于排名模型可以由第一种预测模型得出结果后再排序得到，所以第一种模型的使用更加普遍。

推荐系统毕是商家用来提高利润的，所以其主要目的是增加产品销量。通过把仔细筛选后的物品推荐给用户，推荐系统能使相关物品得到用户的关注，从而达到增加销量、提高利润的目的，尽管主要目的是盈利，但要实现其功能，方法并不是所想的那么直观。为了实现商业性盈利，一般推荐系统操作上和技术上的目标如下：

1)　相关性：推荐系统最重要的操作目标是推荐与用户相关的物品，用户更可能消费那些他们觉得有趣的物品。尽管相关性是推荐系统的主要操作目标，但并不充分。因此我们下面会讨论一些不如相关性重要但仍具有很大影响力的其他操作目标。

2)　新颖性：如果所推荐的物品是用户从没见过的，那么推荐系统确实很有用。例如，用户喜欢类型的

流行电影很少会让用户眼前一亮。反复推荐受欢迎的物品也可能导致销售的多样性降低[D. M. Fleder and K. Hosanagar,2007]。

3) 意外性：意外性是指所推荐的物品出乎意料[N. Good et al.,1999]，幸运的发现相比于明显的建议要温和得多。意外性不同于新颖性的地方在于其能真正让用户感到惊喜，而不是简单地推荐一些之前没见过的东西。通常情况下，用户可能只是消费一类特定的物品，然而并不排除同时存在着使他们惊喜的物品。和新颖性不同，意外性注重于发现这类推荐物品。

4) 提高推荐系统的多样性：推荐系统通常列出一个物品的 top-k 推荐列表。当所有推荐的物品都非常相似时，用户一个都不喜欢的风险也随之而来。另一方面，当推荐列表包含不同类型的物品时，用户在这些物品中至少看上一个的可能性就变得很大。多样性确保用户不会对相似的物品反复推荐感到厌烦。

2.1.2.1 推荐系统应用范围

接下来，我们将给出不同推荐系统中实际应用目标的概述。各个推荐系统所推荐的产品和目标如表 2.1 所示。大多数推荐系统都集中于传统的电子商务应用，推荐包括书籍、电影、视频、旅行、其他产品和服务。[J. Schafer et al.,1999] 讨论了推荐系统在电子商务领域更广泛的应用。然而，推荐系统已经超越了传统的产品推荐领域。值得注意的是，表 2.1 中的一些系统不推荐具体的产品。比如 Google Search 应用程序，可以与搜索结果一同打出产品广告。这就涉及计算广告学，这是一个完全不同的领域，但毋置疑的是它与推荐系统密切相关。同样，脸书网推荐朋友，在线招聘网站向雇主和求职者推荐彼此。在线招聘网系统也被称作相互推荐系统。其中一些推荐算法的模型与传统的推荐系统大不相同。这本书将仔细研究这些区别。

表 2.1 现实中不同推荐系统推荐产品的样例

系统	产品目标	系统	产品目标
Amazon.com	书籍和其他产品	Google Search	广告
Netflix	DVD，视频流	Facebook	朋友，广告
Jester	笑话	Pandora	音乐
GroupLens	新闻	YouTube	在线视频
MovieLens	电影	Tripadvisor	旅行产品
last.fm	音乐	IMDb	电影
Google News	新闻		

2.2 基于近邻的协同过滤

2.2.1 引言

基于近邻的协同过滤算法，也被称为基于内存的算法(memory-based algorithm)，是最早的为协同过滤而开发的算法之一。这类算法是基于相似的用户以相似的行为模式对物品进行评分，并且相似的物品往在获得相似的评分这一事实。基于近邻的算法分为以下两个基本类型：

1） 基于用户的协同过滤：这种类型中，把与目标用户 A 相似的用户的评分用来为 A 进行推荐。这些“同组群体”对每件物品的评分的加权平均值将用来计算用户 A(对物品)的预计评分。

2） 基于物品的协同过滤：为了推荐目标物品 B，首先确定一个物品集合 S，使 S 中的物品与 B 相似度最高。然后，为了预测任意一个用户 A 对 B 的评分，需要确定 A 对集合 S 中物品的评分，这些评分的加权平均值将用来计算用户 A 对物品 B 的预计评分。

2.2.2 评分矩阵的关键性质

根据具体应用的不同，评分可以分为如下几类：

1) 连续评分：这种评分是连续变量，分值对应着对眼前物品的喜恶程度。比如 Jesterjoke 推荐引擎就是使用这种评分的一个例子，这种引擎允许评分从-10～10 连续变化。其缺点是为用户带来了要从无穷多个数中想出一个的负担，因此采用这种方式的相对稀少。

2) 间隔评分：这种评分通常采用 5 分制或 7 分制，当然，也可能是 10 分制或 20 分制。其实例可以是 1～5，-2～2 或者 1～7 的整数。一个重要的假设是令分值明确定义评分之间的差距，并且通常情况下分值是等距的。

3) 顺序评分：顺序评分与间隔评分十分相近，唯一的不同是顺序评分使用有序的分类值，例如“强烈反对”“反对”“保留意见”“赞同”“强烈赞同”。顺序评分与间隔评分主要的区别在于：顺序评分不要求相邻评分等距。然而，这仅仅是理论情况，实际上这些不同的分类值常常被赋予等距的实用的数值。比如令“强烈反对”为 1 分，“强烈赞同”为 5 分。在这种情况下，顺序评分几乎等同于间隔评分。通常来说，为避免偏差，正面评价与负面评价的数目是相等的。当设置偶数种评价时，不提供“保留意见”选项，这种方法就是强迫选择法，因为你必须表明立场。

4) 二元评分：在二元评分中，仅提供两个选项，分别对应正面与负面的评价。二元评分可以看作是间隔评分与顺序评分的特殊情况。例如， Pandora 网络广播站让用户能够选择喜欢或不喜欢特定的音乐曲目。二元评分迫使用户做出选择，以防止用户因持中立态度而总是不做出评价。

5) 一元评分：这种系统允许用户对某件物品选择一个正面的选项，但不提供负面选项。这往往是许多真实世界中的设置，比如 Facebook 中使用的“喜欢”按钮。更进一步，一元评分可以从顾客的操作中导出。例如，顾客购买某物品的行为可被视为对该物品的一项正面投票。另一方面，顾客没有购买某件物品并不一定意味着顾客不喜欢这件物品。一元评分很特别，因为它简化了用于设定评分的专业模型的开发过程。

值得一提的是，从客户操作中推导一元评分也被称为隐式反馈(implicit feedback)。

物品评分的分布常常满足现实世界中的长尾(long-tail)属性。根据这一属性可知，只有一小部分的物品被频繁地评价，这类物品被称为热门物品。而绝大多数的物品很少被评价。这导致了分布的高度偏斜。图 2.1 阐述了一个评分偏斜分布的例子。X 轴代表物品的序号，按被评价的频率降序排列，Y 轴代表物品被评价的频率。显然，大多数物品的评价次数很少。这样的评分分布对推荐过程有着重要意义。

1) 在许多情况下，高频物品倾向于利润低的、竞争相对激烈的物品，另一方面，低频物品的利润率更大。这种情况下，推荐低物品对商家来说是有利的。事实上，分析表明[C. Anderson, 2006]，许多公司，比如 Amazon.com，通过销售长尾部分的物品使得利润最大化。

2) 由于长尾部分的物品评价较少，对长尾部分提供健壮的评分预测通常更加困难。实际上，许多推荐算法倾向于推荐热门物品而非冷门物品[P. Cremonesi et al., 2010]。这种现象制约了物品推荐的多样性，用户可能常常对相同的推荐感到厌倦。

3) 长尾分布意味着经常被评价的物品数量较少，这一事实对基于近邻的协同过滤算法有着重要影响，因为近邻的定义常常是基于这些经常被评价的物品。在很多情况下，热门物品的评价并不能代表冷门物品的评价，因为这两类物品在评分模式上有着本质区别。故，预测过程可能产生具有误导性的结果。这种现象也能造成推荐算法的误导性评价。

推荐过程中，需要考虑评分的诸如稀疏性和长尾性这样重要的特性。通过调整推荐算法，考虑这种现实属性，就能够获得更有意义的预测。

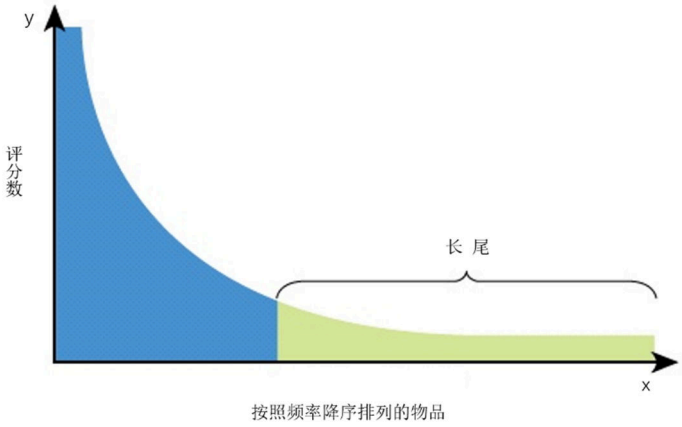


图 2.1 评分频率的长尾性质

2.2.3 通过基于近邻的方法预测评分

基于近邻的方法的基本思想是，利用用户—用户的相似性或物品—物品的相似性从评分矩阵中获得推荐结果。近邻这一概念说明我们需要确定相似的用户或相似的物品来预测评分。接下来，我们将讨论如何使用基于近邻的方法来预测特定的用户—物品组合的评分。基于近邻的模型有如下两个基本原则：

1) 基于用户的模型：相似的用户对相同的物品有相似的评价。因此，如果 Alice 和 Bob 在过去对电影有相似的评价，我们就能利用 Alice 对电影《Terminator》的已知评价去预测 Bob 对这部电影的评价。

2) 基于物品的模型：同一位用户对相似的物品评价是相似的。因此，Bob 对《Alien》和《Predator》这类科幻电影的评价可以用来预测他对相似电影《Terminator》的评价。

2.2.3.1 基于用户的近邻模型

对于拥有 m 位用户和 n 件物品的 m×n 的评分矩阵$R = [r_{uj}]$，I_u表示已被用户(行)u 评价的物品的序号之集。例如，如果用户(行)u 对第一、第三、第五件物品(列)的评价是已知的(观测到的)，其余未知，那么我们$I_u = \{1,3,5\}$。因此，用户 u 和 v 均评价过的物品集合就是$I_u \cap I_v$。例如，如果用户 v 对前 4 件物品做出了评价，那么$I_v = \{1,2,3,4\}$，$I_u \cap I_v = \{1,3,5\} \cap \{1,2,3,4\} = \{1,3\}$。$I_u \cap I_v$有可能(并且常常)是空集，因为评分矩阵通常是稀疏的。集合$I_u \cap I_v$定义了两位用户均已知的评分，利用这个集合，我们可以计算第 u 位和第 v 位用户的相似度，得到的相似度则用于计算近邻。

Pearson 相关系数(Pearson correlation coefficient)可以用来衡量用户 u 和用户 v 之间评分向量的相似程度$\mathrm{Sim}(u,v)$。鉴于$I_u \cap I_v$代表了用户 u 和用户 v 均做出评分的物品序号集合，仅在这个集合中的物品上计算相关系数。第一步是利用每位用户 u 的评分计算每位用户的平均评分μ_u：

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|} \quad \forall u \in \{1 \cdots m\} \tag{2.1}$$

接下来，行(用户)u 和 v 之间的 Pearson 相关系数定义如下：

$$\mathrm{Sim}(u,v) = Pearson(u,v)$$

$$= \frac{\sum_{k\in I_u\cap I_v}(r_{uk}-\mu_u)\times(r_{vk}-\mu_v)}{\sqrt{\sum_{k\in I_u\cap I_v}(r_{uk}-\mu_u)^2}\times\sqrt{\sum_{k\in I_u\cap I_v}(r_{vk}-\mu_v)^2}} \quad (2.2)$$

我们计算目标用户和其他每一位用户之间的 Pearson 系数。一种定义目标用户的同组群体的方法是选择前 k 个 Pearson 系数最高的用户。然而，这样挑选出的同组群体对各个物品的评分情况有明显差别，因此需要对每个要预测评分的物品都单独找出前 k 位最相似的用户，这样对每个物品都有 k 个用户做出评分。这些评分的加权平均值能作为对那件物品的预测评分。在这里，每一个评分都以做出评分的用户与目标用户之间的 Pearson 系数作为权重。

这种方法的主要问题是，不同的用户可能以不同的尺度做出评价。某位用户可能对所有物品做出高度评价，而另一位用户也许对所有物品给出消极评价。因此，在确定同组群体的(加权)平均评分之前，用户的评分需要进行均值中心化。用户 u 对物品 j 按均值中心化后的评分s_{uj}被定义为原始评分r_{uj}减去其平均评分。

$$s_{uj} = r_{uj} - \mu_u \quad \forall u \in \{1 \cdots m\} \quad (2.3)$$

正如之前所说，目标用户 u 的前 k 个同组群体对一件物品的均值中心化评分的加权平均值被用于提供一个均值中心化(mean-centered)的预测。再把目标用户评分的平均值加上这个预测，得到用户 u 对物品 j 的一个评分预测$\hat{r}_{uj}$。 r_{uj}顶部的帽形记号表示一个预测评分，与原始评分矩阵中已知的评分相对。令$P_u(j)$表示与目标用户 u 最相近的 k 位对物品 j 做出评分的用户集合。作为一种启发式的改进，与目标用户 u 相关性很低或者负相关的用户有时会从$P_u(j)$中剔除。于是，整体的基于近邻的预测函数表示如下：

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v\in P_u(j)} \mathrm{Sim}(u,v) \times s_{vj}}{\sum_{v\in P_u(j)} |\mathrm{Sim}(u,v)|}$$

$$= \mu_u + \frac{\sum_{v\in P_u(j)} \mathrm{Sim}(u,v)\times(r_{uj}-\mu_u)}{\sum_{v\in P_u(j)} |\mathrm{Sim}(u,v)|} \quad (2.4)$$

这种泛化的方法可以允许定义不同类型的相似度函数或预测函数，以及物品被淘汰的策略。

基于用户的算法实例

考虑表 2.1 中的例子，表中展示了 1…5 这 5 位用户对标号为 1…6 的 6 件物品的评分。每一项评分从{1…7}中取值。假设目标用户是 3 号用户，我们希望基于表 2.1 中的评分进行物品预测。为决定最优的推荐物品，我们需要计算用户 3 对物品 1 和物品 6 的预测评分$\hat{r}_{31}$和$\hat{r}_{36}$。

第一步要计算用户 3 与其他用户的相似度。我们在这一表格的后两列展示了两种计算相似度的可行方法。倒数第二列展示了评分之间基于余弦的相似度，而最后一列展示了基于 Pearson 相关系数的相似度。例如，$\mathrm{Cosine}(1,3)$和$\mathrm{Pearson}(1,3)$的值计算如下：

$$\mathrm{Cosine}(1,3) = \frac{6*3+7*3+4*1+5*1}{\sqrt{6^2+7^2+4^2+5^2}\times\sqrt{3^2+3^2+1^2+1^2}} = 0.956$$

$$\mathrm{Pearson}(1,3) = \frac{(6-5.5)*(3-2)+(7-5.5)*(3-2)+(4-5.5)*(1-2)+(5-5.5)*(1-2)}{\sqrt{1.5^2+1.5^2+(-1.5)^2+(-0.5)^2}\times\sqrt{1^2+1^2+(-1)^2+(-1)^2}} = 0.894$$

表 2.1 的后两列展示了用户 3 与其他用户的余弦相似度和 Pearson 相似度。需要注意到， Pearson 相关系数更具有说服力，并且其符号的正负代表了相似或相异。根据两种相似度的计算方法，与用户 3 最相近的两位用户是用户 1 和用户 2。根据用户 1 和用户 2 的原始评分以 Pearson 系数为权重的加权平均值，

对用户 3 未评分的物品 1 和 6 得到如下预测：

$$\hat{r}_{31} = \frac{7 * 0.894 + 6 * 0.939}{0.894 + 0.939} \approx 6.49$$

$$\hat{r}_{36} = \frac{4 * 0.894 + 6 * 0.939}{0.894 + 0.939} = 4$$

表 2.1 用户 3 和其他用户间的用户-用户相似度计算

物品 ID 用户 ID	1	2	3	4	5	6	平均评分	$\text{Cosine}(i,3)$ (用户-用户)	$\text{Pearson}(i,3)$ (用户-用户)
1	7	6	7	4	5	4	5.5	0.956	0.894
2	6	7	?	4	3	4	4.8	0.981	0.939
3	?	3	3	1	1	?	2	1.0	1.0
4	1	2	2	3	3	4	2.5	0.789	-1.0
5	1	?	1	2	3	3	2	0.645	-0.817

因此，物品 1 应该先于物品 6 被推荐给用户 3。更进一步，预测结果显示，对用户 3 来说，与她评价过的任何电影相比，她或许会更喜欢电影 1 和电影 6。然而，这是由于同组群体{1，2}远比目标用户 3 要乐观，他们给出更多积极的评分。现在，让我们验证均值中心化的评分对预测带来的影响。表 2.2 展示了均值中心化的评分，与之对应的均值中心化的预测公式如下所示：

$$\hat{r}_{31} = 2 + \frac{1.5 * 0.894 + 1.2 * 0.939}{0.894 + 0.939} \approx 3.35$$

$$\hat{r}_{36} = 2 + \frac{-1.5 * 0.894 - 0.8 * 0.939}{0.894 + 0.939} \approx 0.86$$

于是，均值中心化的预测结果仍然认为物品 1 应该优先于物品 6 被推荐给用户 3。然而，一个与之前的推荐至关重要的不同是，在这种情况下，物品 6 的预测评分只有 0.86，低于用户 3 评价过的任何物品。这与之前物品 6 的预测评分高于所有已知评分的情况有天壤之别。直接观察表 2.1(或者表 2.2)，显然，用户 3 理应给予物品 6 很低的评分(和她的其他物品相比)，因为与她最相近的用户(用户 1 和 2)对物品 6 的评分都比他们对其他物品的评分低。由此可见，均值中心化能够就已知的评分提供更好的相对预测。在许多情况下，这么做也会影响到所预测物品的相对顺序。这种结果唯一的缺点是物品 6 的预测评分 0.86 超出了允许的评分范围。这样的评分可以用来排名，也可以将预测值修正到最近的允许值。

表 2.2 物品 1 和物品 6 与其他物品的调整后的余弦相似度

物品 ID 用户 ID	1	2	3	4	5	6
1	1.5	0.5	1.5	-1.5	-0.5	-1.5
2	1.2	2.2	?	-0.8	-1.8	-0.8
3	?	1	1	-1	-1	?
4	-1.5	-0.5	-0.5	0.5	0.5	1.5
5	-1	?	-1	0	1	1
$\text{Cosine}(1,j)$ (物品-物品)	1	0.735	0.912	-0.848	-0.813	-0.990

$\text{Cosine}(6,j)$（物品-物品）	-0.990	-0.622	-0.912	0.829	0.730	1

2.2.3.2 基于物品的近邻模型

在基于物品的模型中，以物品而不是用户构建同组群体。因此，需要计算物品(即评分矩阵的列)之间的相似度。在计算列之间的相似度之前，每行的评分被以均值为零点中心化。和基于用户的模型一样，每一件物品的评分都被减去该物品的平均评分以得到一个均值中心化的矩阵。这一过程与之前计算均值中心化评分s_{uj}一样(见公式(2.3))。令U_i表示已对物品 i 做出评价的用户集合。因此，如果第一、第三、第四位用户对物品 i 的评价已知，那么我们有$\mathrm{U}_i = \{1,3,4\}$。

于是，物品 i 与物品 j 的调整余弦相似度定义如下：

$$\text{AdjustedCosine}(i,j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} \times s_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} s_{ui}^2} \times \sqrt{\sum_{k \in I_u \cap I_v} s_{uj}^2}} \tag{2.5}$$

这种相似度之所以被称为调整过的，是因为在计算相似度数值之前，评分被均值中心化了。虽然在基于物品的方法中仍然可以使用 Pearson 相关系数，但调整余弦通常会产生更好的结果。

假设我们需要确定用户 u 对物品 t 的评分。第一步是通过之前提到的调整余弦相似度确定与物品 t 最相似的 k 件物品。用$Q_t(u)$表示用户 u 已评价且与 t 最相似的 k 件物品。这些(原始)评分的加权平均值即是预测结果。物品 j 与目标物品 t 的调整余弦相似度即为其权重。因此，用户 u 对目标物品 t 的预测评分表示如下：

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} \text{AdjustedCosine}(j,t)\ r_{uj}}{\sum_{j \in Q_t(u)} |\text{AdjustedCosine}(j,t)|} \tag{2.6}$$

基本思想是在最终预测阶段中利用用户自己对相似物品的评价。例如，在一个电影推荐系统中，物品的同组群体通常是同类型的电影。一位用户对这些电影的历史评价在预测这位用户的兴趣时是十分可靠的因素。

前一节讨论了许多基于用户的协同过滤的基本方法的变形。由于基于物品的算法与基于用户的算法十分相似，所以在基于物品的方法中，也可以在相似度函数和预测函数中设计类似的变形。

基于物品的算法实例

我们仍然利用表 2.1 的例子来说明基于物品的算法。我们将使用基于物品的算法来预测用户 3 的未知评分。因为用户 3 对物品 1 和物品 6 的评分是未知的，因此我们需要计算物品 1 和物品 6 与其他列(物品)的相似度。

首先，需要计算均值中心化以后的物品相似度。表 2.2 展示了均值中心化之后的矩阵。表的最后两行展示了物品 1 和物品 6 与其他物品对应的调整余弦相似度，物品 1 和 3 之间的调整余弦相似度$\text{AdjustedCosine}(1,3)$计算如下：

$$\text{AdjustedCosine}(1,3) = \frac{1.5 * 1.5 + (-1.5) * (-0.5) + (-1) * (-1)}{\sqrt{1.5^2 + (-1.5)^2 + (-1)^2} \times \sqrt{1.5^2 + (-0.5)^2 + (-1)^2}} = 0.912$$

其他的物品—物品相似度以类似的方法计算，其结果在表 2.2 的最后两行。显然，物品 2 和物品 3 与物品 1 最为相似，物品 4 和物品 5 与物品 6 最相似。因此用户 3 对物品 2 和物品 3 的原始评分的加权平均值被用来预测她对物品 1 的评分$\hat{r}_{31}$；同理，她对物品 4 和物品 5 的原始评分的加权平均值被用来预测她对物品

6 的评分$\hat{r}_{36}$：

$$\hat{r}_{31} = \frac{3 * 0.735 + 3 * 0.912}{0.735 + 0.912} = 3$$

$$\hat{r}_{36} = \frac{1.* 0.829 + 1 * 0.730}{0.829 + 0.730} = 1$$

可见，基于物品的方法也表明，用户 3 可能更倾向于选择物品 1 而不是物品 6。然而由于此次预测利用用户 3 自己的评分，所以预测的结果与该用户对其他物品的评分有较高的一致性。在这个实例中，值得注意的一点是，与基于用户的方法不同，对物品 6 的预测评分并没有超出允许的评分范围。基于物品的方法的主要优势在于它具有更高的预测准确度。在某些情况下，基于物品的方法和基于用户的方法，虽然它们的推荐列表大致会相同，但可能会产生不同的前 k 个推荐物品。

2.2.3.3 基于近邻方法的优劣势

基于近邻的方法的简单和直观为其带来了优势。由于简单直观，它们容易实现与调试。通常很容易说明一件物品为什么会被推荐，基于物品的方法的可解释性尤其明显。在之后将要讨论的许多基于模型的方法中，做出这种解释并非易事。除此以外，在用户与物品增加时，其推荐结果相对稳定。同时，这些方法可以对增量做出预估。

这类方法的主要缺点在于，其离线阶段在大规模数据上变得无法实现。基于用户的方法，其离线阶段要求至少$O(m^2)$的时间和空间。在桌面计算系统下，当 m 达到千万量级时，其计算会变得很慢或空间不足。不过近邻方法的在线阶段总是很高效。这些方法最致命的缺点是由于稀疏性导致的覆盖度不足。例如，如果 John 的最近邻没有评价过《Terminator》，那么 John 对《Terminator》的评价就无法预测。另一方面，我们在大多数的推荐中只关心前 k 件物品。如果 John 的最近邻都没有评价过《Terminator》，那这部电影显然不是一个好的推荐。当两位用户共同评价过的物品很少时，稀疏性同样也对相似度计算的健壮性带来挑战。

2.2.4 降维与近邻方法

降维方法能够同时提高近邻方法的质量和效率。尤其是在稀疏矩阵中很难健壮地计算每对之间的相似度的情况下，降维也能够根据潜在因子提供稠密的低维表示。因此，这样的模型被称为潜在因子模型。

基于用户的协同过滤方法的基本思想是利用主成分分析法将 m×n 的矩阵 R 转化到更低维度的空间中。得到的矩阵R'是一个 m×d 的矩阵，且 d≪n。因此，代表用户评分的每一个(稀疏的)n 维向量被转化为低维的 d 维向量。而且，与原始评分向量不同，每一个 d 维向量都是完全确定的。当表示每位用户的 d 维向量都确定之后，我们就用降维后的向量来计算目标用户和其他用户的相似度。在降维表示上的相似度计算更具有健壮性，因为新的低维向量是完全确定的。而且由于低维向量维度较低，相似度的计算也更加高效。在低维空间中，简单的余弦或点积就足以计算相似度。

接下来要说明如何计算每个数据的低维表示。可以通过类 SVD 方法或类 PCA 方法计算低维表示。下面我们说明类 SVD 方法。

第一步是填充 m×n 不完全矩阵 R 中的未知项。以对应行的平均值(即对应用户的平均评分)作为未知项的估值。另一种方法是用列的平均值(即对应物品的平均评分)作为估值。结果表示为R_f。接下来，我们计算 n×n 的物品相似度矩阵 S，$S = R_f^T \cdot R_f$。这个矩阵是半正定的。为了确定 SVD 的控制基向量，我们对相似度矩阵 S 施行如下的对角化：

$$S = P\Delta P^T \tag{2.7}$$

这里，P 是一个 n×n 的矩阵，其列包含 S 的正交特征向量。Δ是一个对角矩阵，其对角线上是 S 的非负特征向量。令P_d为 n×d 的矩阵，仅包含 P 的最大的 d 个特征向量对应的列。那么，矩阵之积$R_f P_d$就是R_f的低维表示。注意，由于R_f是 m×n 的矩阵，P_d是 n×d 的矩阵，所以降维表示$R_f P_d$的维度为 m×d。因此这时 m 个用户每个都能够在 d 维空间内表示。这样的表示被用于决定每位用户的同组群体。一旦确定了用户的同组群体，便可以利用公式(2.4)预测评分。这样的方法也能被用于基于物品的协同过滤，只需用R_f的转置矩阵替代R_f。

先前提到的方法可被看作评分矩阵R_f的奇异值分解(Singular Value Decomposition，SVD)。很多其他方法使用主成分分析(Principal Component Analysis，PCA)而不是 SVD，但是总体结果非常相似。在 PCA 方法中，使用R_f的协方差阵替代相似度矩阵$R_f^T R_f$。对于列均值中心化的数据来说，这两种方法是一样的。

值得注意的是，矩阵R_f是由不完全矩阵 R 以行或列的均值填入未知项而得到的。这种方法很可能会引起偏差。为了理解偏差的性质，我们假设使用 PCA 进行降维，因此需要估计协方差阵。我们假设未知值用列的均值代替。

2.2.4.1 极大似然估计

概念重构法(conceptual reconstruction method)提出使用概率技术，比如 EM 算法来估计协方差矩阵。我们假设数据符合生成模型，即把已知项看成是生成模型的输出。对协方差矩阵的估计可以看作是生成模型参数估计的一部分。接下来，我们提供一种该方法的简化。这种简化的方法计算协方差矩阵的最大似然估计。每对物品之间的协方差仅使用已知项进行估计。换句话说，只有对某对物品做出评分的用户被用来估计协方差。当没有用户在一对物品上做出共同评价时，协方差被估计为 0。

为了进一步减少表示的偏差，可以直接将不完全矩阵 R 投射到降维矩阵P_d上，而不是将填充过的矩阵R_f投射到P_d。其基本思想是计算每个已知评分对投影到P_d中每个潜在向量的贡献，然后计算贡献的平均值。平均贡献计算如下。令$\vec{e_i}$代表P_d的第 i 列(特征向量)，其中第 j 项为e_{ji}。令r_{uj}为 R 中用户 u 对物品 j 的已知评分。则用户 u 对投影到潜在向量的贡献为$r_{uj}e_{ji}$。设集合I_u代表用户 u 已评分的物品集合。用户 u 在第 i 个潜在向量上的平均贡献计算如下：

$$a_{ui} = \frac{\sum_{j \in I_u} r_{uj} e_{ji}}{|I_u|} \tag{2.8}$$

这种均值归一化的方法在不同的用户做出不同数量的评价时尤其有用。得到的 m×d 矩阵$A = [a_{ui}]_{m \times d}$便是原始评分矩阵的降维表示。在基于用户的协同过滤中，这个降维矩阵被用来计算目标用户的近邻。同样，也可以将此方法用于 R 的转置矩阵，来降低用户的维度(而不是物品的维度)。在基于物品的协同过滤中利用这样的方法来计算物品的近邻是很有用的。

2.2.4.2 不完全数据的直接矩阵分解

虽然前面的方法能够在某些情况下修正协方差估计产生的偏差，但是当评分矩阵的稀疏程度很高时并不十分有效。这是因为协方差估计要求物品之间足够多的已知评分来进行健壮的估计。当矩阵稀疏时，协方差的估算在统计学上来说是不可靠的。

一种更直接的方法是使用矩阵分解方法。像奇异值分解之类的方法从本质上说就是矩阵分解方法。我们暂时假设 m×n 的矩阵 R 是完全已知的。在线性代数[G. Strang , 2009]中，一个众所周知的事实是，任何(完全已知)矩阵 R 都能分解成如下形式：

$$R = Q\sum P^T \quad (2.9)$$

这里，Q 是一个 m×m 的含有RR^T的 m 个正交特征向量的矩阵。P 是一个 n×n 的含有R^TR的 n 个正交特征向量的矩阵。Σ是一个 m×n 的对角矩阵，其中只有对角线项是非零值，并且包含R^TR(或RR^T)的非零特征值的平方根。值得注意的是，R^TR和RR^T的特征向量并不相同并且当 m≠n 时维度不同。但是，它们总是拥有相同数量的非零特征值，且值相等。在Σ对角线上的值也被叫作奇异值。

2.2.5 基于近邻的协同过滤实例[zhuanlan,2019]

为了使说明过程更清楚，这里使用自己的数据。每一行记录着某用户对某本书的评分，评分区间为 1 至 5。

```
import pandas as pd

data_url = 'https://gist.githubusercontent.com/guerbai/3f4964350678c84d3
        59e3536a08f6d3a/raw/f62f26d9ac24d434b1a0be3b5aec57c8a08e7741
        /user_book_ratings.txt'
df = pd.read_csv(data_url,
          sep = ',',
          header = None,
          names = ['user_id', 'book_id', 'rating'])
print (df.head())
user_count = df['user_id'].unique().shape[0]
book_count = df['book_id'].unique().shape[0]
print ('user_count: ', user_count)
print ('book_count: ', book_count)

'''
运行以上程序可以得到输出:
      user_id    book_id   rating
0   user_001   book_001         4
1   user_001   book_002         3
2   user_001   book_005         5
3   user_002   book_001         5
4   user_002   book_003         4
user_count:   6
book_count:   6
```

（1） 生成用户物品关系矩阵

现在根据加载进来的数据生成推荐系统中至关重要的用户物品关系矩阵。可以理解为数据库中的一张表，一本书为一列，一行对应一个用户，当用户看过某本书并进行评分后，在对应的位置填入分数，其他位置均置为 0，表示尚未看过。

需要注意的是，矩阵取值用下标表示，比如 matrix[2][2]对应的是第三个用户对第三本书的评分情况，所以这里要做一个 user_id，book_id 到该矩阵坐标的对应关系，使用 pandas 的 Series 表示。

```
user_id_index_series = pd.Series(range(user_count),
    index=['user_001', 'user_002', 'user_003', 'user_004', 'user_005', 'user_006'])
book_id_index_series=pd.Series(range(book_count),
    index=['book_001', 'book_002', 'book_003', 'book_004', 'book_005', 'book_006'])
import numpy as np

def construct_user_item_matrix(df):
    user_item_matrix = np.zeros((user_count, book_count), dtype=np.int8)
    for row in df.itertuples():
        user_id = row[1]
        book_id = row[2]
        rating = row[3]
        user_item_matrix[user_id_index_series[user_id],book_id_index_series
        [book_id]]book_id_index_series[book_id]] = rating
    return user_item_matrix

user_book_matrix = construct_user_item_matrix(df)
print (user_book_matrix)

'''
运行以上程序可以得到输出:
 [[4 3 0 0 5 0]
  [5 0 4 0 4 0]
  [4 0 5 3 4 0]
  [0 3 0 0 0 5]
  [0 4 0 0 0 4]
  [0 0 2 4 0 5]]
```

（2） 计算相似度矩阵

所谓相似度，我们这里使用余弦相似度，其他的还有皮尔逊相关度、欧氏距离、杰卡德相似度等，个中差别暂不细表。计算公式为：

$$sim(x,y)=\frac{xy}{||x||\times||y||}=\frac{\sum x_i y_i}{\sqrt{\sum x_i^2}\sqrt{\sum y_i^2}} \tag{2.10}$$

现在已经拿到了 user_book_matrix，每个用户、每个物品都可以对应一个向量，比如 user_book_matrix[2]为代表 user_003 的向量等于[4,0,5,3,4,0]，而 user_book_matrix[:,2]代表了 book_003：[0,4,5,0,0,2]。

这样基于用户和基于物品便分别可以计算用户相似度矩阵与物品相似度矩阵。

以用户相似度矩阵为例，计算后会得到一个形状为（user_count,user_count）的矩阵，比如 user_similarity_matrix[2][3]的值为 0.5，则表示 user_002 与 user_003 的余弦相似度为 0.5。此矩阵为对称矩阵，相应地，user_similarity_matrix[3][2] 亦为 0.5，而用户与自己自然是最相似的，遂有 user_similarity_matrix[n][n]总等于 1。

```
def cosine_similarity(vec1, vec2):
    return round(vec1.dot(vec2)/(np.linalg.norm(vec1)*np.linalg.norm(vec2)), 2)

def construct_similarity_matrix(user_item_matrix, dim='user'):
    if dim == 'user':
        similarity_matrix = np.zeros((user_count, user_count))
        count = user_count
    else:
        similarity_matrix = np.zeros((book_count, book_count))
        count = book_count
    get_vector = lambda i: user_item_matrix[i]
                    if dim == 'user' else user_item_matrix[:,i]
    for i in range(user_count):
        i_vector = get_vector(i)
        similarity_matrix[i][i] = cosine_similarity(i_vector, i_vector)
        for j in range(i, count):
            j_vector = get_vector(j)
            similarity = cosine_similarity(i_vector, j_vector)
            similarity_matrix[i][j] = similarity
            similarity_matrix[j][i] = similarity
    return similarity_matrix
user_similarity_matrix = construct_similarity_matrix(user_book_matrix)
book_similarity_matrix = construct_similarity_matrix(user_book_matrix, dim='book')
print ('user_similarity_matrix:')
print (user_similarity_matrix)
print ('book_similarity_matrix:')
print (book_similarity_matrix)

'''
运行以上程序可以得到输出:
 user_similarity_matrix:
 [[1.   0.75 0.63 0.22 0.3  0.  ]
  [0.75 1.   0.91 0.   0.   0.16]
  [0.63 0.91 1.   0.   0.   0.4 ]
  [0.22 0.   0.   1.   0.97 0.64]
  [0.3  0.   0.   0.97 1.   0.53]
  [0.   0.16 0.4  0.64 0.53 1.  ]]
 book_similarity_matrix:
 [[1.   0.27 0.79 0.32 0.98 0.  ]
  [0.27 1.   0.   0.   0.34 0.65]
  [0.79 0.   1.   0.69 0.71 0.18]
  [0.32 0.   0.69 1.   0.32 0.49]
  [0.98 0.34 0.71 0.32 1.   0.  ]
  [0.   0.65 0.18 0.49 0.   1.  ]]
```

（3） 推荐

有了相似度矩阵，可以开始进行推荐。首先可以为用户推荐与其品味相同的用户列表，这在知乎、豆瓣、网易云音乐这样具有社交属性的产品中很有意义。

做法很简单，要为用户 A 推荐 k 位品味相似的用户（此处 K 取 3），则将 user_similarity_matrix 中关于 A 的那一行的值排序从最高往下取出 K 位即可。

```
def recommend_similar_items(item_id, n=3):
    item_index = book_id_index_series[item_id]
    similar_item_index = pd.Series(book_similarity_matrix[item_index]).
                drop(index=item_index).sort_values(ascending=False).index[:n]
    return np.array(similar_item_index)
print ('recommend item_indexes %s to book_001' % recommend_similar_items
     ('book_001'))
'''
```

同时在物品维度，类似的推荐也是很有用的，比如 QQ 音乐给用户正在听的音乐推荐相似的歌曲，还有亚马逊中对用户刚购买的物品推荐相似的物品。 代码与推荐相似用户相同，无需做其他处理。

接下来是为用户推荐书籍，首先选出与该用户最相似的 K 个用户，然后找出这 K 个用户评过分的书籍的集合，再去掉该用户已经评过分的部分。 在剩下的书籍中，根据下面的公式，计算出该用户为某书籍的预计评分，将评分从高到低排序输出即可。

$$P_{u,i} = \frac{\sum_j^n (sim_{u,j} * R_{j,i})}{\sum_j^n sim_{u,j}} \tag{2.11}$$

```
def recommend_item_to_user(user_id):
    user_index = user_id_index_series[user_id]
    similar_users = recommend_similar_users(user_id, 2)
    recommend_set = set()
    for similar_user in similar_users:
        recommend_set = recommend_set.union(np.nonzero(
                        user_book_matrix[similar_user])[0])
    recommend_set = recommend_set.difference(np.nonzero(
                    user_book_matrix[user_index])[0])
    predict = pd.Series([0.0]*len(recommend_set), index=list(recommend_set))
    for book_index in recommend_set:
        fenzi = 0,fenmu = 0
        for j in similar_users:
            if user_book_matrix[j][book_index] == 0:
                continue #  相似用户未看过该书则不计入统计.
            fenzi += user_book_matrix[j][book_index] * user_similarity_matrix
                [j][user_index]
            fenmu += user_similarity_matrix[j][user_index]
        if fenmu == 0:
            continue
        predict[book_index] = round(fenzi/fenmu, 2)
    return predict.sort_values(ascending=False)
'''
```

```
print(recommend_item_to_user('user_005'))

'''
运行以上程序可以得到输出:
3    4.0
2    2.0
dtype: float64
```

以上是利用用户相似度矩阵来为用户推荐物品，同样也可以反过来为利用物品相似度矩阵来为用户推荐书籍。 做法是，找出该用户读过的所有书，为每本书找出两本与该书最相似的书籍，将找出来的所有书去掉用户已读过的，然后为书籍预测被用户评分的分值。

这里的确有些绕，容易与上文缠在一起搞乱掉，遂举例如下：

比如 user_001 读过书 book_001，book_002，book_005，找到的书籍集合再去掉用户已读过的结果为{'book_003', 'book_006'}，要为 book_003 预测分数，需要注意到它同时被 book_001 与 book_005 找出，要根据它们、用户对 book_001 与 book_005 的评分以及相似度套用至上文公式，来得出对 book_003 的分数为：(4*0.79+5*0.71)/(0.79+0.71)=4.47。

则基于物品为用户推荐物品的函数为：

```
def recommend_item_to_user_ib(user_id):
    user_index = user_id_index_series[user_id]
    user_read_books = np.nonzero(user_book_matrix[user_index])[0]
    book_set = set()
    book_relation = dict()
    for book in user_read_books:
        relative_books = recommend_similar_items(book, 2)
        book_set = book_set.union(relative_books)
        book_relation[book] = relative_books
    book_set = book_set.difference(user_read_books)
    predict = pd.Series([0.0]*len(book_set), index=list(book_set))
    for book in book_set:
        fenzi = 0
        fenmu = 0
        for similar_book, relative_books in book_relation.items():
            if book in relative_books:
                fenzi += book_similarity_matrix[book][similar_book] * user_book
                    _matrix[user_index][similar_book]
                fenmu += book_similarity_matrix[book][similar_book]
        predict[book] = round(fenzi/fenmu, 2)
    return predict.sort_values(ascending=False)

'''
```

```
print(recommend_item_to_user_ib('user_001'))

'''
运行以上程序可以得到输出:
2       4.47
5       3.00
dtype: float64
```

（4） 总结

以上是基于领域的协同过滤的运作机制介绍，只用了两个简单的数学公式，加上各种代码处理，便可以为用户做出一些推荐。就给用户推荐物品而言，基于用户与基于物品各有特点。

基于用户给出的推荐结果，更依赖于当前用户相近的用户群体的社会化行为，考虑到计算代价，它适合于用户数较少的情况，同时，对于新加入的物品的冷启动问题比较友好，然而相对于物品的相似性，根据用户之间的相似性做出的推荐的解释性是比较弱的，实时性方面，用户新的行为不一定会导致结果的变化。

基于物品给出的推荐结果，更侧重于用户自身的个体行为，适用于物品数较少的情况，对长尾物品的发掘好于基于用户，同时，新加入的用户可以很快得到推荐，并且物品之间的关联性更易懂，是更易于解释的，而且用户新的行为一定能导致结果的变化。

显然，基于物品总体上要优于基于用户，历史上，也的确是基于用户先被发明出来，之后 Amazon 发明了基于物品的算法，现在基于用户的产品已经比较少了。

2.3 基于模型的协同过滤

上一章提到的基于近邻的协同过滤方法可以被看作是机器学习中常用的 k-近邻分类方法的泛化，它们都是基于案例的方法。这些方法必须是高效的，因为除了一些可选的预处理环节，我们不会预先建立模型以用于预测。基于近邻的方法是基于案例的学习方法或懒惰学习的泛化，其预测方法针对实例的预测。例如，在基于用户的近邻方法中，要确定等价的目标用户群体才能实施预测。

基于模型的推荐系统很多情况下优于基于近邻的推荐系统：

1) 节省空间：一般情况下，学习得到的模型的大小远小于原始的评分矩阵，所以空间需求通常较低。另一方面，基于用户的近邻算法可能需要$O(m^2)$的空间复杂度，其中 m 是用户数目。基于物品的近邻算法则需要$O(n^2)$的空间复杂度。

2) 训练和预测速度快：基于近邻的方法的一个问题在于预处理环节需要用户数或物品数的平方级别时间，而基于模型的系统在建立训练模型的预处理环节需要的时间往往要少得多。在大多数情况下，压缩和总结模型可以被用来加快预测。

3) 避免过拟合：过拟合在很多机器学习算法中是非常严重的问题，在这些算法中预测结果往往被一些随机因素影响。此类问题在分类和回归模型中同样存在。在基于模型的方法中，运用总结方法有助于避免过拟合。除此之外，还可以运用正则化方法使得这些模型更具健壮性。

虽然基于近邻的方法是最早被提出的协同过滤方法之一，且由于其简洁性被应用得非常广泛，但就目前的情况而言，它们并非总是最精确的方法。事实上，通常最精确的协同过滤方法都是基于模型的，尤其是潜在因子模型。

2.3.1 基于规则的协同过滤

关联规则[C. Aggarwal and J. Han, 2014]和协同过滤的关系非常自然，这是因为关联规则问题最早的提

出背景是为了发现超市数据之间的关联关系。尽管通过将分类和数值型的数据转化为二元数据，关联规则可以被扩展到多种类型的数据上，但从本质上说，关联规则是定义在二元数据上的。为了讨论方便，我们将假设数据是一元的。一元数据在超市交易数据和隐式反馈数据集合中非常常见。

考虑一个交易数据库$T = \{T_1 \cdots T_m\}$，其包含定义在 n 个项（物品）构成的集合 I 上的 m 个事务。也就是说，I 是物品的全集，事务T_1是 I 的一个子集。挖掘关联规则即是要找出交易数据中那些相关性较高的物品。为此，可以定义支持度（support）和置信度（confidence）来度量物品的相关度。

定义 3.1 (支持度). 物品集$X \subseteq I$的支持度定义为 T 中包含 X 的事务所占的百分比。

如果某物品集的支持度大于一个预先定义的阈值 s，则称物品集是频繁的。该阈值被称为最小支持度。支持度不小于阈值的物品集被称为是频繁项集或频繁模式。频繁项集可以为用户购买行为的关联性提供重要线索。

定义 3.2(置信度). 规则$X \Longrightarrow Y$的置信度是含 X 的事务中同时包含 Y 的条件概率 P(Y|X)，因此，其置信度等于$X \cup Y$的置信度除以 X 的置信度。

注意$X \cup Y$的置信度一定不大于 X 的置信度。因为如果一个事务包含$X \cup Y$，那么它一定包含 X。不过反过来不一定成立，因此一条规则的置信度的值总是位于(0，1)区间内的。置信度越高则规则越强。例如，如果规则$X \Longrightarrow Y$为真，那么商家只要知道客户购买了 X 中的物品，就可以推断出客户会购买 Y 中的物品。基于最小支持度 s 和最小置信度 c 可以定义如下关联规则：

定义 3. 3(关联规则). 规则$X \Longrightarrow Y$被称为是最小支持度 s 和最小置信度 c 下的关联规则，如果下述两个条件同时被满足：

1. $X \cup Y$的支持度不小于 s；
2. $X \Longrightarrow Y$的置信度不小于 c。

寻找关联规则的算法分为两步。首先，确定所有满足最小支持度阈值 s 的物品集。然后，对其中任一物品集 Z，用所有可能的二路划分(X，$Z - X$)产生候选规则$X \Longrightarrow Z - X$。候选中满足最小置信度的规则被保留。第一步中确定频繁项集需要很大的计算量，当数据库非常大的时候该问题尤为严重。当前已经有很多高效的频繁项集发现算法被提出用于提高这一步的效率。这些算法在数据挖掘里属于专门的领域，因此不在本书的讨论范围之内，感兴趣的读者可以阅读文献[C. Aggarwal and J. Han, 2014]来获取频繁项集算法的细节。在本书中，我们将展示如何在协同过滤中使用这些算法。

基于规则的协同过滤的第一步是在一个预先确定好的最小支持度和最小的置信度取值下发现所有的关联规则。最小支持度和最小置信度可以看作是能被调整以使得预测准确度最大化的参数。只有后件包含单个物品的规则会被保留。该规则集合就是可以被用来为特定用户提供推荐的模型。给定需要获取相关物品推荐的用户 A，首先要确定 A 触发的关联规则。如果一条关联规则的前件表示的物品集包含于 A 喜欢的物品集合，则称该规则是 A 能够触发的。所有触发规则会被按照置信度排序，排好序后的规则的前 k 个后件即是要被推荐给 A 的物品。上述方法是文献[B. Sarwar et al., 2001]中算法的简化版本。许多基于此方法的变形常被应用于推荐系统中，如采用降维来处理稀疏性的方法等[B. Sarwar et al., 2001]。

2.3.2 潜在因子模型

潜在因子模型在推荐系统中被认为是一种最先进的方法。这些模型利用了一些著名的降维方法来填充缺失项。降维方法在数据分析的其他领域中经常被用来得到原始数据的低维表示。其基本思想是旋转坐标系，以使得维度之间的两两相关性被去除，得到的降推和旋转后的完整数据表示可以有效地近似原始的不完整矩阵。一旦获得了完整的数据表示，则我们可以再反向旋转回原始坐标系以得到完整的数据表示[C. Aggarwal and S. Parthasarathy, 2001]。在内部，降维利用了行和列之间的相关性来得到完整的数据表示。无论在基于近邻的还是基于模型的协同过滤算法中，相关性发挥的作用都是至关重要的。例如，基于用户的近邻方法利用了用户之间的相关性，基于物品的近邻方法则利用了物品之间的相关性。矩阵因子分解给出了一种优雅的同时利用行列相关性来估计整个数据矩阵的方法。这种方法的复杂性使其成为协同过滤中最

先进的方法。

2.3.2.1 低秩解释和矩阵分解原理

首先，让我们考虑简单的情形----评分矩阵中所有的项的值都是已知的。关键的想法在于任意 m×n 且秩k ≪ min{m, n}的矩阵 R 可以表示成如下 k 个因子的乘积：

$$R = UV^T \tag{2.12}$$

其中，U 是一个 m×k 的矩阵，V 是一个 n×k 的矩阵。注意，R 的行空间和列空间的秩都是 k。U 的每一列可以被看作 R 的 k 维列空间的 k 个基向量之一，V 的第 j 行包含相应的系数，将这些基向量合并到 R 的第 j 列中。又或者，我们可以将 V 的列看作是 R 的行空间的基向量，将 U 的列看作是相应的系数。这里，秩为 k 的矩阵的因子分解基于线性代数的一些基础知识[G. Strang, 2009]，对于不同的基向量集合的因子分解可能有无穷多种。SVD 是此类因子分解的一个例子，其中 U 的列（以及 V 的列）表示的基向量是正交的。

即使矩阵 R 的秩大于 k，其也可以近似表示为 k-秩因子的乘积。

$$R \approx UV^T \tag{2.13}$$

和之前一样，U 是 m×k 的矩阵，V 是 n×k 的矩阵。该近似的误差等于$||R - UV^T||^2$，其中$||\cdot||^2$表示剩余矩阵$(R - UV^T)$的项的平方和。这个量也称为剩余矩阵的 Frobenius 范数。剩余矩阵主要表示评分矩阵的无法用低秩因子建模的噪声。为了简化讨论，我们来考虑 R 完全已知的简单情况。我们首先考虑因子分解过程的内在含义，然后讨论矩阵缺失项时该含义的引申意义。

如图 2.2 所示的评分矩阵。该图表示了一个 7×6 的评分矩阵，有 7 个用户和 6 个物品。所有的评分都取自集合{1，一 1，0}，这些分值分别表达了喜欢、不喜欢和中立三个观点。被评分的物品是电影，分别属于爱情和历史两个分类。其中一个名为《Cleopatra》的电影同时属于两个分类。由于电影分类的特性，用户也在评分方面表现出明显的倾向性。例如，用户 1～3 明显喜欢历史电影但对爱情电影持中立态度。用户 4 对两类电影都喜欢。用户 5～7 喜欢爱情电影，但不喜欢历史电影。注意，该矩阵中用户和物品之间有很强的关联性。尽管两类电影的评分看起来是相对独立的。因此，该矩阵可以近似用 2-秩因子分解，如图 2.2a 所示。矩阵 U 是一个 7×2 的矩阵，表示了用户对于两个分类的倾向性，V 是一个 6×2 的矩阵，表示了电影的分类归属。换言之，矩阵 U 提供了列空间的基，矩阵 V 提供了行空间的基。例如，矩阵 U 表明用户 1 喜欢历史电影，而用户 4 两类电影都喜欢。

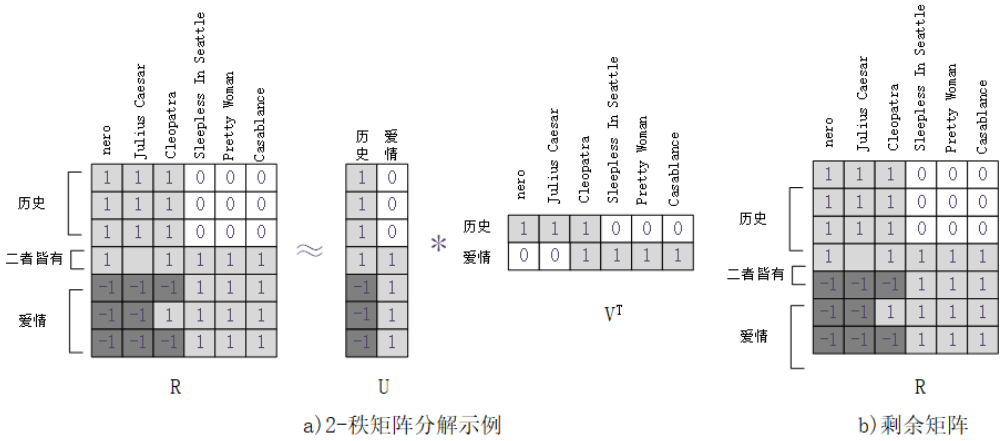


图 2.2 矩阵因子分解与剩余矩阵示例

2.3.2.2 无约束矩阵分解

在讨论不完整矩阵的分解之前，我们先来看看分解完整矩阵的问题。如何确定因子矩阵 U 和 V，使得完整矩阵 R 尽可能接近UV^T？可以针对矩阵 U 和 V 形式化优化问题，以实现这一目标：

$$\text{Minimize J} = \frac{1}{2}||R - \mathrm{UV}^T||^2$$

满足：U 和 V 上无约束 （2.14）

这里，$||\cdot||^2$表示矩阵陈的平方 Frobenius 范数，其等于矩阵项的平方和。因此，目标函数等于剩余矩阵$(R - \mathrm{UV}^T)$中项的平方和。目标函数越小，因子分解$\mathrm{R} \approx \mathrm{UV}^T$的质量越好。这个目标函数可以被看作是一个二次损耗函数，它通过使用低秩分解来量化估计矩阵 R 的精度损失。可以使用各种梯度下降方法为该分解提供最优解。

然而，在具有缺失值的矩阵的上下文中，只有 R 的值的子集是已知的。因此，如上所述，目标函数也是不确定的。毕竟，在一些值缺失的情况下，人们无法计算矩阵的 Frobenius 范数。因此，为了学习 U 和 V，目标函数需要仅基于观察到的值重写。关于这个过程的很好的部分是，一旦潜在因子 U 和 V 被学习出来，整个评分矩阵可以使用UV^T被一次性重建出来。

令 S 表示在 R 中已知的所有用户—物品对(i，j)构成的集合。其中，$i \in \{1 \cdots m\}$是用户的索引，$j \in \{1 \cdots n\}$是物品的索引。因此，已知的用户—物品对的集合 S 定义如下：

$$\mathrm{S} = \{(i,j): r_{ij} \text{是已观测的}\} \quad (2.15)$$

如果我们可以将不完整矩阵 R 分解为完全指定矩阵$\mathrm{V} = [v_{js}]_{n \times k}$和$\mathrm{U} = [u_{is}]_{m \times k}$的近似乘积$\mathrm{UV}^T$，则也可以预测 R 中的所有值。具体地，可以如下预测矩阵 R 的(i，j)位置的值：

$$\hat{r}_{uj} = \sum_{s=1}^{k} u_{is} \cdot v_{js} \quad (2.16)$$

注意左侧评级上的“帽子”符号（即回旋）表示它是预测值而不是观测值。指定条目(i，j)的观测值和预测值之间的差由$e_{ij} = r_{ij} - \hat{r}_{ij} = \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}\right)$给出。然后，使用 S 中的已知值和修改后的针对不完整矩阵的目标函数做如下计算：

$$\text{Minimize J} = \frac{1}{2}\sum_{(i,j) \in S} e_{ij}^2 = \frac{1}{2}\sum_{(i,j) \in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}\right)^2$$

满足：U 和 V 上无约束 （2.17）

注意，上述目标函数仅在 S 中的已知值上对误差求和。此外，每个项$\left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}\right)^2$是(i，j)观测值和预测值之间的平方误差$e_{ij}^2$。这里，$u_{is}$和$v_{js}$是未知变量，需要学习以最小化目标函数。这可以简单地用梯度下降方法来实现。因此，需要计算相对于决策变量u_{iq}和v_{jq}的 J 的偏导：

$$\left[\frac{\partial J}{\partial u_{iq}}\right] = \sum_{j:(i,j) \in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}\right)(-v_{jq}) \quad \forall i \in \{1 \cdots m\}, q \in \{1 \cdots k\}$$

$$= \sum_{j:(i,j) \in S} e_{ij}(-v_{jq}) \quad \forall i \in \{1 \cdots m\}, q \in \{1 \cdots k\} \quad (2.18)$$

$$\left[\frac{\partial J}{\partial v_{jq}}\right]=\sum_{i:(i,j)\in S}\left(r_{ij}-\sum_{s=1}^{k}u_{is}\cdot v_{js}\right)\left(-u_{iq}\right)\quad\forall j\in\{1\cdots n\},q\in\{1\cdots k\}$$

$$=\sum_{i:(i,j)\in S}e_{ij}\left(-u_{iq}\right)\quad\forall j\in\{1\cdots n\},q\in\{1\cdots k\}\qquad(2.19)$$

注意，全部偏导向量向矩阵 U 和 V 中的(m·k+n·k)个决策变量向量提供梯度。令$\overline{\nabla J}$表示这个梯度向量，且令$\overline{\mathrm{VAR}}$表示 U 和 V 中(m·k+n·k)决策变量的向量，则可以用$\overline{\mathrm{VAR}}\Leftarrow\overline{\mathrm{VAR}}-\alpha\cdot\overline{\nabla J}$ 更新整个决策变量向量。这里，a＞0 是步长，可以使用非线性规划中的标准数值方法来选择。在许多情况下，步长设置为很小的常数，迭代一直执行到收敛。上述方法被称为梯度下降。梯度下降的算法框架下所示。值得注意的是，中间变量u_{iq}^{+}和v_{jq}^{+}用于确保对 U 和 V 中条目的所有更新都同时执行。

```
Algoritjm GD(Ratings Matrix: R, Learning Rate:  α)
begin
    Randomly initialize matrices U and V;
    S = {(i, j) :  r_ij  is  observed};
    while not(convergence) do
    begin
        Compute each error  e_ij ∈ S  as the observed entries of  R − UV^T;
        for each user-component pair(i , q) do  u_iq^+ ⇐ u_iq + α · Σ_j:(i,j) e_ij · v_jq;
        for each user-component pair(j , q) do  v_jq^+ ⇐ v_jq + α · Σ_j:(i,j) e_ij · u_iq;
        for each user-component pair(i , q) do  u_iq = u_iq^+;
        for each user-component pair(j , q) do  v_jq = v_jq^+;
        Check convergence condition;
    end
end
'''
```

上述方法被称为批量更新方法。一个重要的观察是，更新是评分矩阵的已知值的错误的线性函数。可以以其他方式通过将更新分解为与单个已知值（非所有已知值）的错误相关联的较小分量来执行更新。根据（随机选择的）已知值(i，j)中的误差，该更新可以随机地近似如下：

$$u_{iq}\Leftarrow u_{iq}-\alpha\cdot\left[\frac{\partial J}{\partial u_{iq}}\right](i,j)\text{的贡献量}\quad\forall q\in\{1\cdots k\}\qquad(2.20)$$

$$v_{jq}\Leftarrow v_{jq}-\alpha\cdot\left[\frac{\partial J}{\partial v_{jq}}\right](i,j)\text{的贡献量}\quad\forall q\in\{1\cdots k\}\qquad(2.21)$$

可以一次（按随机顺序）循环遍历 R 中的已知值，并仅更新因子矩阵中的 2 · k 个值的相关集合，而不是因子矩阵中的所有(m·k+n·k)个值。在这种情况下，特定于值(i，j) ∈ S 的 2 · k 个更新如下：

$$u_{iq}\Leftarrow u_{iq}+\alpha\cdot e_{ij}\cdot v_{jq}\quad\forall q\in\{1\cdots k\}\qquad(2.22)$$

$$v_{jq}\Leftarrow v_{jq}+\alpha\cdot e_{ij}\cdot u_{iq}\quad\forall q\in\{1\cdots k\}\qquad(2.23)$$

对于每个已知的评分r_{ij}，使用误差e_{ij}来更新 U 的行 i 中的 k 个值和 V 的行 j 中的 k 个值。注意，$e_{ij}\cdot v_{jq}$是 J 相对于u_{iq}的偏导数的分量，其特定于单个值(i，j)。为了提高效率，k 个值中的每一个都可以用向量化的形式同时更新。令$\overline{u_i}$表示 U 的第 i 行，可表示$\overline{v_j}$的第 j 行。那么，上述更新可以用 k 维向量化形式重写如

下：

$$\overline{u_i} \Leftarrow \overline{u_i} + \alpha \cdot e_{ij} \cdot \overline{v_j} \quad (2.24)$$

$$\overline{v_j} \Leftarrow \overline{v_j} + \alpha \cdot e_{ij} \cdot \overline{u_i} \quad (2.25)$$

我们遍历所有已知值多次（即多次迭代）直到达到收敛。这种方法被称为随机梯度下降，其中梯度用矩阵中单个随机选择的值的误差来近似。随机梯度下降法的伪码如下所示。值得注意的是，临时变量u_{iq}^{+}和v_{jq}^{+}用于在更新过程中存储中间结果，以使得 2 · k 个更新不会相互影响。这是一个通用的方法，尽管我们可能不会明确说明，但它应该在本书中讨论的所有面向群组的更新中使用。

```
Algoritjm SGD(Ratings Matrix: R, Learning Rate:  α
begin
    Randomly initialize matrices U and V;
    S = {(i, j) :  r_ij  is  observed};
    while not(convergence) do
    begin
        Randomly shuffle observed entries in S;
        for each (i , j)  ∈  S in shuffled order do
        begin
            e_ij ⇐ r_ij − Σ_{s=1}^{k} u_is v_is;
            for each q  ∈  {1⋯k} do  u_iq^+ ⇐ u_iq + α · e_ij · v_jq;
            for each q  ∈  {1⋯k} do  v_jq^+ ⇐ v_jq + α · e_ij · u_iq;
            for each q  ∈  {1⋯k} do  u_iq = u_iq^+  and  v_jq = v_jq^+;
        end
        Check convergence condition;
    end
end
‘’’
```

实际上，与批次更新方法相比，随机梯度下降法获得的收敛速度更快，尽管后者的收敛性更加平滑。这是因为在后一种情况下，使用所有已知值而非单个随机选择的值，U 和 V 的值被同时更新。随机梯度下降的这种噪声近似有时会影响解的质量和收敛的平滑度。通常，当数据大小非常大并且计算时间是主要瓶颈时，随机梯度下降更好。其他“折中”方法会使用更小的批次，用已知值的子集来构建更新。这些不同的方法提供了解决方案质量和计算效率之间的不同权衡。

2.3.2.3 奇异值分解

奇异值分解（SVD）是矩阵分解的一种形式，其中 U 和 V 的列被限定为相互正交的。相互正交性的优点在于，概念可以完全独立于彼此且可以在散点图中进行几何解释。然而，这种分解的语义解释通常更加困难，因为这些隐向量包含正数和负数，并且受其与其他概念的正交性约束。对于完全已知的矩阵，使用特征分解方法执行 SVD 是比较容易的。

考虑评分矩阵完全已知的情况。可以通过使用秩k ≪ min{m, n}的截断 SVD 近似分解评分矩阵 R。截断 SVD 计算如下：

$$R \approx Q_k \sum_k P_k^T \quad (2.26)$$

这里，Q_k、$\sum_k$、P_k分别是 m×k、k×k、n×k 的矩阵。矩阵Q_k和P_k分别包含R^TR和RR^T的 k 个最大特征向量，而（对角线）矩阵$\sum_k$包含沿其对角线的任一矩阵的 k 个最大特征值的（非负）平方根。值得注意的是，R^TR和RR^T的非零特征值是相同的，即使当 m≠n 时它们将包含不同数量的零特征值。矩阵P_k包含R^TR的顶部特征向量，它是行空间降维所需的简化基本表示。这些特征向量包含关于评分的物品—物品相关性的方向性信息，因此它们能够在旋转坐标系中用较少的维度表示每个用户。

从公式(2.19)可以很容易地看出，SVD 被定义为矩阵分解。当然，这里的因子分解是要分解为三个矩阵而不是两个矩阵。然而，对角矩阵$\sum_k$可以被用户因子Q_k或物品因子P_k吸收。按惯例，用户因子和物品因子定义如下：

$$\mathrm{U} = Q_k \sum_k \tag{2.27}$$

$$\mathrm{V} = P_k \tag{2.28}$$

如前所述，评分矩阵 R 的因子分解被定义为$\mathrm{R} = \mathrm{UV}^T$。只要用户和物品因子矩阵具有正交的列，就很容易将得到的因子分解转换成满足 SVD 的形式。因此，分解过程的目标是用正交列发现矩阵 U 和 V，故 SVD 可以表示为矩阵 U 和 V 上的优化问题：

$$\text{Minimize } \mathrm{J} = \frac{1}{2}||R - \mathrm{UV}^T||^2$$

满足：

U 的列相互正交

V 的列相互正交 （2.29）

很容易看出，与无约束因子分解的情况的唯一区别是存在正交性的约束。换句话说，与无约束矩阵分解相比，是在更小的解空间上优化相同的目标函数。尽管人们可能会认为约束的存在会增加近似误差 J，但是事实证明，如果矩阵 R 完全已知且未使用正则化，在 SVD 和非约束矩阵分解的情况下 J 的最优值是相同的。因此，对于完全已知的矩阵，SVD 的最优解是无约束矩阵分解的替代最优解之一。在 R 不完全已知且目标函数 $\mathrm{J} = \frac{1}{2}||R - \mathrm{UV}^T||^2$仅在已知值上计算的情况下，这不一定正确。此时，无约束矩阵分解通常在已知值上能够保证较低的误差。然而，由于不同模型的可泛化的程度不同，对于未知值来说其性能是不可预测的。

2.3.2.4 非负矩阵分解

非负矩阵分解（NMF）可用于非负的评分矩阵。这种方法的主要优点不一定是准确性，而是在理解用户和物品的交互中提供的高可解释性。与其他形式矩阵分解的主要区别在于 U 和 V 因子必须是非负的。因此，非负矩阵分解的优化公式如下:

$$\text{Minimize } \mathrm{J} = \frac{1}{2}||R - UV^T||^2$$

$$\text{满足：} \mathrm{U} \geq 0;\ \mathrm{V} \geq 0 \tag{2.30}$$

虽然非负矩阵分解可以用于任何非负评分矩阵（例如评分从 1～5 的情况），但是其最大的可解释性优点出现在有机制令用户表达“喜欢”但没有机制让用户表达“不喜欢”的情况。这样的矩阵包括一元评分矩阵或用矩阵的非负值表示动作率的情况。这些数据集也称为隐式反馈数据集。

算法案例：餐馆菜肴推荐系统［李锐等］

现在我们就开始构建一个推荐系统，该推荐系统关注的是餐馆食物的推荐，假设一个人在家决定外出吃饭，但是他并不知道该到哪儿去吃饭，该点什么菜。我们这个推荐系统可以帮助他做到这两点。

首先我们构建一个基本的系统，他能够寻找用户没有尝过的菜肴。然后，通过 SVD 来减少特征空间并提高推荐的效果。这之后，将程序打包并通过用户可读的人机界面提供给人们使用。最后，我们介绍在构建推荐系统时面临的一些问题。

（1） 餐馆菜肴评级

下面给出了一个矩阵，它是由餐馆的菜和品菜师对这些菜的意见构成的。品菜师可以采用 1 到 5 之间的任意一个整数来对菜评级。如果品菜师没有尝过某道菜，则评级为 0。

	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	手撕猪肉
Ed	0	0	0	2	2
Peter	0	0	0	3	3
Tracy	0	0	0	1	1
Fan	1	1	1	0	0
Ming	2	2	2	0	0
Pachi	5	5	5	0	0
Jocelyn	1	1	1	0	0

图 2.3 餐馆的菜及其评级的数据

（2） 相似度计算

```
from numpy import *
from numpy import linalg as la

def ecludSim(inA,inB):
    return 1.0/(1.0 + la.norm(inA - inB))

def pearsSim(inA,inB):
    if len(inA) < 3 : return 1.0
    return 0.5+0.5*corrcoef(inA, inB, rowvar = 0)[0][1]

def cosSim(inA,inB):
    num = float(inA.T*inB)
    denom = la.norm(inA)*la.norm(inB)
    return 0.5+0.5*(num/denom)
'''
```

程序中的 3 个函数分别是欧氏距离、皮尔逊相关系数和余弦相似度三种计算相似度的方法为了便于理解，NumPy 的线性代数工具箱 linalg 被作为 la 导入，函数中假定 inA 和 inB 都是列向量。pearsSim()函数会检查是否存在 3 个或更多的点。如果不存在，该函数会返回 1.0，这是因为此时两个向量完全无关。

在这里我们计算了两个餐馆菜肴之间的距离，这称为基于物品的相似度。

（3） 推荐未尝过的菜肴

推荐系统的工作过程是：给定一个用户，系统会为此用户返回 N 个最好的推荐菜。为了实现这一点，则需要我们做到：

1) 寻找用户没有评级的菜肴，即在用户—物品矩阵中的 0 值

2) 在用户没有评级的所有物品中，对每个物品预计一个可能的评级分数。这就是说，我们认为用户可能会对物品的打分（这就是相似度计算的初衷）；

3）对这些物品的评分从高到低进行排序，返回前 N 个物品。

好了，接下来我们用代码实现基于物品相似度的推荐引擎：

```
def standEst(dataMat, user, simMeas, item):
    n = shape(dataMat)[1]
    simTotal = 0.0; ratSimTotal = 0.0
    for j in range(n):
        userRating = dataMat[user,j]
        if userRating == 0: continue
        overLap = nonzero(logical_and(dataMat[:,item].A>0, dataMat[:,j].A>0))[0]
        if len(overLap) == 0: similarity = 0
        else: similarity = simMeas(dataMat[overLap,item],dataMat[overLap,j])
        print('the %d and %d similarity is: %f' % (item, j, similarity))
        simTotal += similarity
        ratSimTotal += similarity * userRating
    if simTotal == 0: return 0
    else: return ratSimTotal/simTotal

def recommend(dataMat, user, N=3, simMeas=cosSim, estMethod=standEst):
    unratedItems = nonzero(dataMat[user,:].A==0)[1]#find unrated items
    if len(unratedItems) == 0: return 'you rated everything'
    itemScores = []
    for item in unratedItems:
        estimatedScore = estMethod(dataMat, user, simMeas, item)
        itemScores.append((item, estimatedScore))
    return sorted(itemScores, key=lambda jj: jj[1], reverse=True)[:N]
    '''
```

上述程序包含了两个函数。第一个函数是 standEst()，用来计算在给定相似度计算方法的条件下，用户对物品的估计评分值。第二个函数是 recommend()，也就是推荐系统，它会调用 standEst()函数。我们先讨论 standEst()函数，然后讨论 recommend()函数。

函数 standEst()的参数包括数据矩阵、用户编号、物品编号和相似度计算方法。假设这里的数据矩阵为图 2.3 的形式，即行对应用户、列对应物品。那么，我们首先会得到数据集中的物品数目，然后对两个后面用于计算估计评分值的变量进行初始化。接着，我们遍历行中的每个物品。如果某个物品评分值为 0，就意味着用户没有对该物品评分，跳过了这个物品。该循环大体上是对用户评过分的每个物品进行遍历，并将它和其他物品进行比较。变量 overLap 给出的是两个物品当中已经被评分的那个元素。如果两者没有任何重合元素、则相似度为 0 且中止本次循环。但是如果存在重合的物品，则基于这些重合物品计算相似度。随后，相似度会不断累加，每次计算时还考虑相似度和当前用户评分的乘积。最后，通过除以所有的

评分总和、对上述相似度评分的乘积进行归一化。这就可以使得最后的评分值在 0 到 5 之间、而这些评分值则用于对预测值进行排序。

函数 recommend()产生了最高的 N 个推荐结果。如果不指定 N 的大小、则默认值为 3。该函数另外的参数还包括相似度计算方法和估计方法。我们可以使用上述程序中的任意一种相似度计算方法。此时我们能采用的估计方法只有一种选择，但是在下一小节中会增加另外一种选择。该函数的第一件事就是对给定的用户建立一个未评分的物品列表。如果不存在未评分物品、那么就退出函数；否则，在所有的未评分物品上进行循环。对每个未评分物品，则通过调用 standEst()来产生该物品的预测得分。该物品的编号和估计得分值会放在一个元素列表 itemScores 中。最后按照估计得分，对该列表进行排序并返回。该列表是从大到小逆序排列的，因此其第一个值就是最大值。

将上述代码保存在 svdRec.py 文件中，接下里我们来看看运行结果。首先调入一个矩阵实例，可以对本章前面给出的矩阵稍加修改后使用。这里先调入原始矩阵：

```
import svdRec
import numpy as np
myMat = np.mat(svdRec.loadExData())
'''
```

该矩阵对于展示 SVD 的作用非常好，但是它本身不是十分有趣，因此我们要对其中的一些值进行改进，得到的矩阵如下：

```
myMat[0,1]=myMat[0,0]=myMat[1,0]=myMat[2,0]=4
myMat[3,3]=2
print(myMat)
'''
运行以上程序可以得到输出：
[[4 4 0 2 2]
 [4 0 0 3 3]
 [4 0 0 1 1]
 [1 1 1 2 0]
 [2 2 2 0 0]
 [5 5 5 0 0]
 [1 1 1 0 0]]
```

现在已经可以做推荐了。先尝试一下默认的推荐：

```
print(svdRec.recommend(myMat,2))
'''
运行以上程序可以得到输出：
the 1 and 0 similarity is: 1.000000
the 1 and 3 similarity is: 0.928746
the 1 and 4 similarity is: 1.000000
the 2 and 0 similarity is: 1.000000
the 2 and 3 similarity is: 1.000000
the 2 and 4 similarity is: 0.000000
[(2, 2.5), (1, 2.0243290220056256)]
```

这表明用户 2（由于我们从 0 开始计数，因此这对应了矩阵的第 3 行）对物品 2 的预测评分值为

2.5，对物品 1 的预测评分值为 2.05。下面我们就利用其他的相似度计算方法来进行推荐：

```
print(svdRec.recommend(myMat,2,simMeas=svdRec.ecludSim))
print(svdRec.recommend(myMat,2,simMeas=svdRec.pearsSim))
'''
运行以上程序可以得到输出：
the 1 and 0 similarity is: 1.000000
the 1 and 3 similarity is: 0.309017
the 1 and 4 similarity is: 0.333333
the 2 and 0 similarity is: 1.000000
the 2 and 3 similarity is: 0.500000
the 2 and 4 similarity is: 0.000000
[(2, 3.0), (1, 2.8266504712098603)]
the 1 and 0 similarity is: 1.000000
the 1 and 3 similarity is: 1.000000
the 1 and 4 similarity is: 1.000000
the 2 and 0 similarity is: 1.000000
the 2 and 3 similarity is: 1.000000
the 2 and 4 similarity is: 0.000000
[(2, 2.5), (1, 2.0)]
```

我们可以对多个用户进行尝试，或者对数据集做些修改来了解其给预测结果带来的变化。

这个例子给出了如何利用基于物品相似度和多个相似度计算方法来进行推荐的过程，下面我们介绍如何将 SVD 应用于推荐。

（4） 利用 SVD 提高推荐的效果

实际的数据集会比我们用于展示 recommand()函数功能的 myMat 矩阵稀疏得多。图 2.4 就给出了一个更真实的矩阵的例子。

	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	三文鱼汉堡	鲁宾三明治	印度烧鸡	麻婆豆腐	宫保鸡丁	印度奶酪咖喱	俄式汉堡
Brett	2	0	0	4	4	0	0	0	0	0	0
Rob	0	0	0	0	0	0	0	0	0	0	5
Drew	0	0	0	0	0	0	0	1	0	4	0
Scott	3	3	4	0	3	0	0	2	2	0	0
Mary	5	5	5	0	0	0	0	0	0	0	0
Brent	0	0	0	0	0	0	5	0	0	5	0
Kyle	4	0	4	0	0	0	0	0	0	0	5
Sara	0	0	0	0	0	4	0	0	0	0	4
Shaney	0	0	0	0	0	0	5	0	0	5	0
Brendan	0	0	0	3	0	0	0	0	4	5	0
Leanna	1	1	2	1	1	2	1	0	4	5	0

图 2.4 一个更大的用户-菜肴矩阵

我们可以将矩阵输入到程序中 loadExData2()函数中，下面我们计算该矩阵 SVD 来了解其到底需要多少维特征。

```
from numpy import linalg as la

U,Sigma,VT = la.svd(np.mat(svdRec.loadExData2()))
print(Sigma)
'''
运行以上程序可以得到输出：
[15.77075346 11.40670395 11.03044558  4.84639758  3.09292055  2.58097379
  1.00413543  0.72817072  0.43800353  0.22082113  0.07367823]
```

接下来我们看看到底有多少个奇异值能达到总能量的 90%。首先，对 Sigma 中的值求平方：

```
sig2 = Sigma**2
'''
```

再计算一下总能量：

```
print(sum(sig2))
'''
运行以上程序可以得到输出：
541.9999999999995
```

再计算总能量的 90%：

```
print(sum(sig2)*0.9)
'''
运行以上程序可以得到输出：
487.7999999999996
```

然后，计算前两个元素所包含的能量：

```
print(sum(sig2[:2]))
'''
运行以上程序可以得到输出：
378.8295595113579
```

该值低于总能量的 90%，于是计算前三个元素所包含的能量：

```
print(sum(sig2[:3]))
'''
运行以上程序可以得到输出：
500.50028912757926
```

该值高于总能量的 90%，这就可以了。于是，我们可以将一个 11 维的矩阵转换成一个 3 维的矩阵。下面对转换后的三维空间构造出一个相似度计算函数。我们利用 SVD 将所有的菜肴映射到一个低维空间中去。在低维空间下、可以利用前面相同的相似度计算方法来进行推荐。我们会构造出一个类似于上面列出的 standEst()函数。打开 svdrec.py 文件并加入如下代码：

```
def svdEst(dataMat, user, simMeas, item):
    n = shape(dataMat)[1]
    simTotal = 0.0; ratSimTotal = 0.0
    U,Sigma,VT = la.svd(dataMat)
    Sig4 = mat(eye(4)*Sigma[:4]) #arrange Sig4 into a diagonal matrix
    xformedItems = dataMat.T * U[:,:4] * Sig4.I    #create transformed items
    for j in range(n):
        userRating = dataMat[user,j]
        if userRating == 0 or j==item: continue
        similarity = simMeas(xformedItems[item,:].T,\
                             xformedItems[j,:].T)
        print('the %d and %d similarity is: %f' % (item, j, similarity))
        simTotal += similarity
        ratSimTotal += similarity * userRating
    if simTotal == 0: return 0
    else: return ratSimTotal/simTotal
'''
```

上述程序中包含有一个函数 svdEst()。在 recommend()中，这个函数用于替换对 standEst()的调用，该函数对给定用户给定物品构建了一个评分估计值。如果将该函数与 standEst()函数进行比较，就会发现很多行代码都很相似。该函数的不同之处就在于它在第 3 行对数集进行了 SVD 分解。在 SVD 分解之后，我们只利用包含了 90%能量值的奇异值，这些奇异值会以 NumPy 数组的形式得以保存。因此如果要进行矩阵运算，那么就必须要用这些奇异值构建出一个对角矩阵。然后，利用 U 矩阵将物品转换到低维空间中。

对于给定的用户、for 循环在用户对应行的所有元素上进行遍历。这和 standEst()函数中的 for 循环的目的一样，只不过这里的相似度计算是在低维空间下进行的。相似度的计算方法也会作为一个参数传通给该函数。然后，我们对相似度求和，同时对相似度及对应评分值的乘积求和。这些值返回之后则用于估计评分的计算。for 循环中加入了一条 print 语句，以便能够了解相似度算的进展情况。如果觉得这些输出很累赘，也可以将该语句注释。

接下来看看程序的执行效果。

```
print(svdRec.recommend(myMat,1,estMethod=svdRec.svdEst))
'''
运行以上程序可以得到输出：
the 1 and 0 similarity is: 0.498142
the 1 and 3 similarity is: 0.498131
the 1 and 4 similarity is: 0.509974
the 2 and 0 similarity is: 0.552670
the 2 and 3 similarity is: 0.552976
the 2 and 4 similarity is: 0.217301
[(2, 3.4177569186592387), (1, 3.330717154558564)]
```

下面再尝试另外一种相似度计算方法：

```
print(svdRec.recommend(myMat,1,estMethod=svdRec.svdEst,simMeas=svdRec.pearsSim))
'''
运行以上程序可以得到输出：
the 1 and 0 similarity is: 0.626075
the 1 and 3 similarity is: 0.672793
the 1 and 4 similarity is: 0.614375
the 2 and 0 similarity is: 0.429334
the 2 and 3 similarity is: 0.387057
the 2 and 4 similarity is: 0.043539
[(2, 3.4992661245386794), (1, 3.327232428061366)]
```

我们还可以再用其他多种相似度计算方法尝试一下。感兴趣的读者可以将这里的结果和前面的方法（不做 SVD 分解）进行比较，看看到底哪个性能更好。

（5） 构建推荐系统面临的挑战

本节的代码很好地展示出了推荐系统的工作流程以及 SVD 将数据映射为重要特征的过程。在撰写这些代码时，尽量保证它们的可读性，但是并不保证代码的执行效率。一个原因是，我们不必在每次估计评分时都做 SVD 分解。对于上述数据集，是否包含 SVD 分解在效率上没有太大的区别。但是在更大规模的数据集上，SVD 分解会降低程序的速度。SVD 分解可以在程序调入时运行一次。在大型系统中，SVD 每天运行一次或者频率更低，并且还要离线运行。

推荐系统中还存在其他很多规模扩展性的挑战性问题，比如矩阵的表示方法。在上面给出的例子中有很多 0，实际系统中 0 的数目更多。也许，我们可以通过只存储非零元素来节省内存和计算开销？另一个潜在的计算资源浪费则来自于相似度得分。在我们的程序中，每次需要一个推荐得分时，都要计算多个物品的相似度得分，这些得分记录的是物品之间的相似度。因此在需要时这些记录可以被另一个用户重复使用。在实际中，另一个普遍的做法就是离线计算并保存相似度得分。

推荐系统面临的另一个问题就是如何在缺乏数据时给出好的推荐。这称为冷启动（cold-start）问题，处理起来十分困难。这个问题的另一个说法是，用户不会喜欢一个无效的物品。而用户不喜欢的物品又无效。如果推荐只是一个可有可无的功能，那么上述问题倒也不大。但是如果应用的成功与否和推荐的成功与否密切相关，那么问题就变得相当严重了。

冷启动问题的解决方案，就是将推荐看成是搜索问题。在内部表现上、不同的解决办法虽有所不同，但是对用户而言却都是透明的。为了将推荐看成是搜索问题，我们可能要使用所需要推荐物品的属性。在餐馆菜肴的例子中，我们可以通过各种标签来标记菜肴，比如素食、美式 BBQ、价格很贵等。

2.4 基于内容的推荐系统

前几章讨论的协同系统使用用户评分模式的相关性来给出推荐。而另一方面，这些方法并不使用物品的属性来计算预测评分，这似乎相当浪费。毕竟，如果 John 喜欢科幻电影《Terminator》，那么他很可能会喜欢属于同类别的《Aliens》。在此情况下，根据其他用户的评分无法做出有意义的推荐。基于内容的系统设计时尝试使用描述性的属性集来描述物品。在这种情况下，用户自己的评分和对其他电影的评分动作足以帮助我们发现有意义的推荐。当某个物品是新的并且对其评分很少时，这种方法非常有用。

在基础层面上，基于内容的系统依赖于两个数据来源：

1) 第一个数据来源是根据以内容为中心的属性对各种物品的描述，例如制造商对物品的文本描述。

2) 第二个数据源是用户画像，其根据用户对各种物品的反馈而生成。用户可能有显式或隐式的反馈，显式反馈可以对应于评分，而隐式反馈可以对应于用户动作，其中评分通过与协同系统类似的方式收集。

基于内容的系统主要适用于手里有大量可用的属性信息的场景。在许多情况下，这些属性是从产品描述中提取的关键词。事实上绝大多数基于内容的系统从底层对象中提取文本属性。因此，基于内容的系统特别适合在文本丰富和非结构化数据中提供建议。使用这种系统的一个典型例子是网页推荐，如可以利用用户之前的浏览行为来创建基于内容的推荐系统。然而，这种系统的使用不仅仅限于 Web。产品说明中的关键词用于创建物品和用户画像，从而为其他电子商务场景提供推荐。

2.4.1 基于内容的系统的架构

基于内容的系统具有一些基本组件，这些组件在这种系统的不同实例中保持不变。由于基于内容的系统具有关于用户的各种物品描述和知识，因此必须将这些不同类型的非结构化数据转换为标准化描述。在大多数情况下，优先的选择是将物品的描述转换为关键词。因此，基于内容的系统在很大程度上（但并不仅仅）被用来操作文本数据，其常见的应用场景也是以文本为中心的。例如，新闻推荐系统通常是基于内容的系统，它们也是以文本为中心的系统。一般说来，文本分类和回归建模方法广泛用来建立基于内容的推荐系统。

基于内容的系统的主要组件包括（离线）预处理部分、（离线）学习部分和在线预测部分。离线部分用于创建汇总模型（一般是分类或回归模型），然后将该模型用于在线生成给用户的推荐。基于内容的系统的各个组成部分如下：

1) 预处理和特提取：基于内容的系统广泛应用于各种领域，如网页、产品描述、新闻、音乐功能等。在大多数情况下，特征提取自这些不同的数据来源，并被转换成基于关键词的向量空间表示。这是所有基于内容的推荐系统的第一步，它是与领域高度相关的。然而，正确提取最具信息性的特征对于有效运行基于内容的推荐系统来说至关重要。

2) 基于内容的用户画像学习：如前所述，基于内容的模型与给定的用户密切相关。因此，需要构建特定于用户的模型，进而根据他们过去的购买或评分历史来预测用户对物品的兴趣。为了实现这一目标，需要利用用户反馈，这可以通过当前已知评分（显式反馈）或用户活动（隐式反馈）的形式表现。这些反馈与物品的属性一起使用以建立训练数据，从而构建学习模型。本阶段通常与分类或回归建模基本相同，这取决于反馈是分类（例如是否选择物品的二元操作）还是数字（例如评分或购买频率）类型的。因为所得到的模型在概念上将用户兴趣（评分）与物品属性关联起来，所以该模型被称为用户画像。

3) 过滤和推荐：在此步骤中，使用上一步学习的模型对特定用户给出推荐的物品。因为推荐需要实时进行，所以这个步骤的效率是非常重要的。

2.4.2 预处理和特征提取

所有基于内容的模型的第一阶段是提取用于表示物品的鉴别性特征。鉴别性特征是能在预测用户兴趣时发挥巨大作用的特征。这一阶段与具体应用高度相关，如网页推荐系统与产品推荐系统就是截然不同的。

2.4.2.1 特征提取

在特征提取阶段，不同物品的描述会被提取出来。尽管我们可以使用任意一种表示，例如多维数据表示，但最常见的方法是从底层数据中提取关键词。做出这种选择是因为非结构化文本描述通常在各种领域中广泛使用，并且它们仍然是最自然的描述物品的方式。在许多情况下，可以用多个字段来描述物品的各个方面。例如，书商可能会提供书籍的文字描述，以及描述内容、标题和作者的关键词。在一些情况下这些描述会被转换成一系列的关键词。而在其他情况下，可以直接使用多维（结构化）表示。当属性包含数值（如价格）或从某个较小的域中取出的值（如颜色）时，后者更为适用。

为了便于在分类过程中使用，各字段需要适当加权。特征加权与选择密切相关，因为前者是后者的“软”

版本。在后一种情况下，特征基于其相关性被选择或被放弃；而在前一种情况下，特征则基于其重要性被赋予不同的权值。

考虑一个电影推荐网站 IMDb，其提供个性化的电影推荐。每部电影通常与电影描述相关联，例如其简介、导演、演员、类型等。在 IMDb 网站上对《Shrek》的简短描述如下：

“在沼泽地充满神奇的生物之后，怪物史瑞克同意为邪恶国王弗瓜拯救菲欧娜公主，以便挽回他的沼泽地。”

许多其他属性（如用户标签）也可以使用，可将其视为内容为中心的关键词。

在《Shrek》的例子里，可以简单地连接各个字段中的所有关键词来创建文本描述，其主要问题在于推荐过程中各关键词的重要性可能并不相同。例如，特定的演员在推荐中的重要性可能要超过摘要中的某个词。这可以通过两种方式解决：

1) 领域相关的知识可以用来决定关键词的重要性。例如，电影名和主演可能比描述中的单词更重要。许多时候这个过程是以试探性的启发式方式进行的。

2) 在许多情况下，自动学习各特征的重要性是可能的。该过程被称为特征加权，与特征选择密切相关。特征加权和特征选择都将在后文中进行描述。

2.4.2.2 特征表示和清洗

此过程在使用非结构化表示时显得尤为重要。特征提取阶段能够从产品或网页的非结构化描述中得到一系列单词。然而，这些表示需要被清洗并以适当的格式表示以便处理。清洗过程包含以下几个步骤：

1) 停止词删除：从物品的自由描述中提取的大部分文本将包含许多与物品相关性不强的常用词。这样的词通常是高频词。例如，“a”“an”和“the”这样的词对于正在处理的物品来说没什么作用。在电影推荐应用中，通常会在剧情介绍中找到这样的词。一般来说，冠词、介词、连词和代词被视为停止词。在大多数情况下，各种语言都有停止词的标准化列表。

2) 词干提取：词干提取过程合并了同一个词的不同变形。例如，同一单词的单复数或不同时态会被合并。在一些情况下，会从各种词汇中提取共同的词根。例如，“hoping”和“hope”这样的词汇被合并成了共同的词根 “hop”。当然，词干提取有时会产生副作用，因为类似于“hop”这样的词可能具有多种不同的含义。

3) 短语提取：这一步工作是检测出文档中频繁同时出现的单词。例如，“hot dog”这样的短语具有与组成它的单词不同的含义。短语提取可以基于手动定义的字典进行，也可以使用一些自动化的方法。

执行这些步骤后，关键词被转换为向量空间表示。每个单词也称为项。在向量空间表示中，文档被表示为一组单词及它们出现的频率。尽管使用单词出现的原始频率可能是诱人的想法，但这通常不可取。因为经常出现的词通常在统计学上差异较小，所以这些词经常被降低权重。这与停止词的原理相似，只不过采用的是“软”的权重打折的方式，而不是完全剔除。

如何对单词打折？这可以基于逆文档频率的概念来实现。第 i 项的逆文档频率id_i是其出现过的文档数量n_i的递减函数。

$$\mathrm{id}_i = log\left(\frac{n}{\mathrm{n}_i}\right) \tag{2.31}$$

其中集合中的文档总数由 n 表示。

2.4.2.3 收集用户的偏好

除了关于物品的内容之外，还需要为推荐过程收集用户喜欢和不喜欢的相关数据。数据收集离线完成，而推荐过程则在用户与系统交互时在线完成。我们将在任何给定时间执行预测的用户称为活动用户。在在

线阶段，需要将用户自己的偏好与内容相结合以提供偏好预测。有关用户喜欢和不喜欢的数据可以采取以下形式表示：

1) 评分：在这种情况下，用户通过评分来表示他们对物品的偏好。评分可以是二元的、基于区间的或者基于顺序的。在极少数情况下，评分甚至可以是实数。评分的性质对学习用户画像的模型有重要的影响。

2) 隐式反馈：隐式反馈是指用户的行为，例如购买或测览物品。在大多数情况下，只能捕获用户的正面偏好和隐式反馈，而难以获得明确的负面偏好信息。

3) 文本观点：在很多情况下，用户可以用文本的形式表达观点。在这种情况下，可以从这些观点中提取隐含的评分。这种形式的评分提取与意见挖掘和情感分析有关。

4) 案例：用户可以指定他们感兴趣的物品的示例（或案例），这可以用作最近邻或 Rocchio 分类器的隐式反馈。然而，当相似性检索与精心设计的效用函数结合使用时，这些方法与基于案例的推荐系统更紧密相关。基于案例的系统是基于知识的推荐系统的子类，其使用领域知识来发现匹配物品，而不是学习算法。

在所有上述情况下，用户对物品的喜欢或不喜欢最终被转换为一元、二元、基于区间的或实数评分。获取评分的过程也可以被看作是提取要用于学习的类标签或因变量的过程。

2.4.2.4 监督特征选择和加权

特征选择和加权的目标是确保在向量空间表示中只保留提供信息最多的词。事实上，许多著名的推荐系统明确提出，应该限制关键词数量。在文献[M. Pazzani and D. Billsus, 1999]中，多个领域的实验结果表明提取的词数应在 50～300 之间。基本思想是噪声单词常常导致过拟合，因此应该提前删除。当考虑到可用于学习特定用户画像的文档数量通常不是很大时，这点尤为重要，因为当可用于学习的文档数量很少时，模型会更倾向于过拟合。因此，减小特征空间至关重要。

向文档表示中引入特征信息量可以从两个不同的方面来考虑。一个是特征选择，对应于删除单词。第二个是特征加权，这涉及词的重要程度。停止词删除和使用逆文档频率分别是特征选择和特征加权的例子。然而，这些是无监督的特征选择和加权方式，用户反馈对其来说不重要。在本节中，我们将研究特征选择的有监督方法，结合用户评分来评估特征的信息量。大多数这些方法评估因变量对特征的敏感性，从而评估其信息量。

特征信息量度量一方面可以用于特征的硬性选择，另一方面也可以使用关于信息的函数来启发式地对特征加权。针对用户评分是被视为数字或类别值的不同情况，特征信息量的度量也不同。例如，在二元评分（或具有少量离散值的评分）的场景下，使用类别而不是数字是更有意义的。

基尼指数是特征选择最常用的度量之一。这是一个简单直观的度量，很容易理解。基尼指数本质上适用于二元评分、顺序评分或分布在少量区间中的评分值。最后一种情况有时可以通过离散化评分来获得。评分的顺序被忽略，评分的每个可能值均被视为一个类别值。这可能看起来像是一个缺点，因为它丢失了关于评分相对顺序的信息。然而，在实践中，可能的评分数通常很小，因此整体上不会损失精度。

令 t 为评分的可能值的总数。在包含特定单词 w 的文档中，令$p_1(w)\cdots p_t(w)$表示在这 t 个可能值中的每一个相关的文档数目。那么，单词 w 的基尼指数定义如下：

$$\text{Gini}(w) = 1 - \sum_{i=1}^{t} p_i(w)^2 \tag{2.32}$$

$\text{Gini}(w)$的值总是位于范围$(0，1-\frac{1}{t})$中，较小的值表示更大的区分能力。例如，当单词 w 的存在总是导致文档被评为第 j 个可能的评分值（即$p_j(w)$=1）时，这个单词对于评分预测是非常有区分能力的。相应地，在这种情况下，基尼指数的值为$1-1^2=0$。当$p_j(w)$的每个值都等于$\frac{1}{t}$时，基尼指数取最大值$1-\sum_{i=1}^{t}\left(\frac{1}{t^2}\right)=1-\frac{1}{t}$。

熵在原理上与基尼指数非常相似，除了使用信息理论原理来设计度量。与以前的情况一样，令 t 为评分的可能值的总数，$p_1(w)\cdots p_i(w)$表示包含特定词 w 的文档中与 t 个可能评分相关的文档的数目。那么，词 w 的熵定义如下：

$$\text{Entropy}(w) = -\sum_{i=1}^{t} p_i(w) log\big(p_i(w)\big) \tag{2.33}$$

$\text{Entropy}(w)$的值总是位于范围(0，1)中，值越小区分能力越强。很容易看出熵具有和基尼指数相似的特性。事实上，尽管它们具有不同的概率解释，但这两个度量往往产生非常相似的结果。基尼指数更容易理解，而熵度量有更坚实的信息论的数学基础。

特征加权可以被视为特征选择的软版本。在本章前面部分的特征表示中，已经讨论了如何使用逆文档频率等度量来加权文档。然而，逆文档频率是不依赖于用户偏好的无监督度量，还可以使用有监督度量来进一步对向量空间表示进行加权，以表达对单词的不同重要程度。例如，在电影推荐的应用中，描述电影类型或演员名的关键词比从电影概要中选择的词更重要。另一方面，简介中的词也能一定程度上表达用户的偏好。因此，它们也不能被删除。特征加权是一种更精细的通过使用权重而非“硬”二元决策来表示词语区分能力的方法。特征加权的最简单方法是采取任何特征选择度量并使用它们来导出权重。例如，可以使用基尼指数或熵的倒数。在许多情况下，启发式函数可以进一步应用于选择度量，以控制加权过程的灵敏度。例如，考虑词 w 的加权函数$g(w)$，其中 a 是大于 1 的参数。

$$g(w) = a - Gini(w) \tag{2.34}$$

所得到的权重$g(w)$将始终位于(a-1，a)范围内。通过改变 a 的值，可以控制加权过程的灵敏度。a 取值越小则灵敏度越大。然后将向量空间表示中的每个单词 w 的权重乘以$g(w)$。可以基于熵和归一化偏差来定义类似的加权函数。

2.4.3 学习用户画像和过滤

用户画像的学习与分类和回归建模问题密切相关。当评分被视为离散值（例如“赞”或“踩”）时，问题类似于文本分类。另一方面，当评分被视为一组数值时，问题类似于回归模型。此外，在结构化和非结构化领域中都可以定义学习问题，由于问题的同质性，我们将假设物品的描述是文档的形式。然而，该方法可以很容易地被推广到任何类型的多维数据，因为文本是一种特殊类型的多维数据。

在每种情况下，假设我们有一个训练文档的集合D_L，这些文档由特定用户标记。当用户从系统中获得建议时，这些用户也称为活动用户，训练文档对应于物品的描述，在预处理和特征选择阶段被提取出来。此外，训练数据还包含活动用户对这些文档的评分。这些文档用于构建训练模型。请注意，在训练过程中不使用其他用户（非活动用户）分配的标签。因此，训练模型特定于给定用户，而无法用于任意用户。这与传统的用矩阵分解在所有用户上建立模型的协同过滤方法不同。这里特定用户的训练模型代表用户画像。

文档上的标签对应于数值、二元或一元评分。假设D_L中的第 i 个文档具有评分c_i。我们还有一组测试文档D_U，它们是未标记的。请注意，D_L和D_U都是专门针对（活动）用户的。测试文档对应于可能推荐给用户但尚未被用户购买或评分的物品的描述。在新闻推荐等领域中，D_U中的文档对应于要推荐给活动用户的候选 Web 文档。D_U的精确定义取决于当前正在处理的领域，其中的单个文档以与D_L中类似的方式提取。D_L上的训练模型用于从D_U中选取要推荐给活动用户的物品。和协同过滤的情况类似，该模型可用于提供评分预测值或 top-k 推荐的排名列表。

很明显，这个问题类似于文本领域的分类和回归建模。在下文中，我们将讨论一些常见的学习方法。

2.4.3.1 最近邻分类

最近邻分类器是最简单的分类技术之一，它可以以相对直接的方式实现。第一步是定义一个将在最近邻分类器中使用的相似度函数。最常用的相似度函数是余弦函数，令$\bar{X} = (x_1 \cdots x_d)$和$\bar{Y} = (y_1 \cdots y_d)$是一对文档，其中第 i 个单词的归一化频率分别由两个文档中的x_i和y_i给出。请注意，需要使用无监督 tf-idf 加权或有监督方法对这些频率进行归一化或加权。然后，基于这些归一化频率来定义余弦测量：

$$\text{Cosine}(\bar{X}, \bar{Y}) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} {x_i}^2} \sqrt{\sum_{i=1}^{d} {y_i}^2}} \tag{2.35}$$

余弦相似性经常在文本领域中使用，因为它能够调整底层文档的长度。当该方法用于其他类型的结构化和多维数据时，会使用其他相似度/距离函数，例如欧几里得距离和曼哈顿距离。对于具有分类属性的关系数据，可以使用各种基于匹配的相似性度量[C. Aggarwal, 2015]。

该相似度函数在对用户偏好未知的物品（文档）进行预测时很有用。对于D_U中的每个文档，使用余弦相似度函数来确定D_L中的 k 个最近邻居。确定D_U中每个物品的 k 个邻居的评分均值。该均值是D_U中对应物品的预测评分。额外可以使用的启发式增强是可以使用相似度对每个评分进行加权。在评分被视为类别的情况下，需要确定每个评分值的投票数，并将票数（频次）最高的值作为预测的评分值。然后根据评分的预测值对D_U中的文档进行排名，并向用户推荐分值最高的物品。

使用这种方法的主要挑战是其高计算复杂度。注意，需要确定D_L中每个文档的最近邻，并且每个最近邻确定所需的时间与D_L的大小呈线性关系。因此，计算复杂度等于$|\mathrm{D}_L| \times |\mathrm{D}_U|$。使方法更快的一种方法是使用聚类来减少$\mathrm{D}_L$中训练文档的数量。对于评分的每个不同的值，$\mathrm{D}_L$中对应的文档子集被聚成$p \leq |D_L|$组。因此，如果有 s 个不同的评分值，则组的总数是$p \cdot s$。通常，使用快速基于质心（即 k 均值）的聚类来创建每组 p 个簇。

2.4.3.2 基于回归的模型

基于回归的模型其优点在于可用于各种类型的评分，如二元评分、基于区间的评分或数值型评分。诸如线性模型、逻辑回归模型和 ordered probit 模型之类的大类回归模型可用于对各种类型的评分进行建模。在这里，我们将描述最简单的线性回归模型。

令D_L是一个 n×d 矩阵，表示基于大小为 d 的词典标注过的 n 个文档的训练集D_L。类似地，令$\bar{\mathrm{y}}$是包含训练集中 n 个文档的活动用户的评分的 n 维列向量，线性回归的基本思想是假设评分可以被建模为单词频率的线性函数。令$\overline{\mathrm{W}}$表示将词频与评分相关联的线性函数中的每个单词的系数的 d 维行向量。然后，线性回归模型假设训练矩阵D_L中的单词频率与评分向量相关如下：

$$\bar{\mathrm{y}} \approx \mathrm{D}_L \overline{\mathrm{W}}^T \tag{2.36}$$

因此，$\mathrm{D}_L \overline{\mathrm{W}}^T - \bar{\mathrm{y}}$是预测误差的 n 维向量。为了最大化预测的质量，必须最小化该向量的平方范数。此外，为了减少过拟合，可以将正则化项$\lambda||\overline{\mathrm{W}}||^2$加到目标函数中。这种正则化形式也被称为 Tikhonov 正则化。这里，$\lambda > 0$ 是正则化参数。因此，目标函数 O 可以表示如下：

$$\text{Minimize } \mathrm{O} = ||\mathrm{D}_L \overline{\mathrm{W}}^T - \bar{\mathrm{y}}||^2 + \lambda ||\overline{\mathrm{W}}||^2 \tag{2.37}$$

通过将该目标函数的梯度相对于$\overline{\mathrm{W}}$设置为 0 可以解决问题。这导致以下条件：

$$D_L^T(\mathrm{D}_L\overline{\mathrm{W}}^T-\bar{\mathrm{y}})+\lambda\overline{\mathrm{W}}^T=0 \quad (2.38)$$

$$(D_L^T\mathrm{D}_L+\lambda\mathrm{I})\overline{\mathrm{W}}^T=D_L^T\bar{\mathrm{y}} \quad (2.39)$$

矩阵$(D_L^T\mathrm{D}_L+\lambda\mathrm{I})$为正定的，因此可逆，故我们可以直接求解权重向量$\overline{\mathrm{W}}$如下：

$$\overline{\mathrm{W}}^T=(D_L^T\mathrm{D}_L+\lambda\mathrm{I})^{-1}D_L^T\bar{\mathrm{y}} \quad (2.40)$$

这里，I 是一个 d×d 的单位矩阵。因此，$\overline{\mathrm{W}}^T$总是存在封闭解。对于来自未标记集合D_U的任何给定文档向量（物品描述）$\bar{\mathrm{X}}$，其评分可以被预测为$\overline{\mathrm{W}}$和$\bar{\mathrm{X}}$之间的点积。Tikhonov 正则化使用L_2正则化项$\lambda||\mathrm{W}||^2$。也可以使用L_1正则化，其中该项被$\lambda||\mathrm{W}||$代替。所得到的优化问题没有封闭解，并且必须使用梯度下降方法。这种正则化形式也被称为 Lasso[T. Hastie et al.,2009]，其可以用于特征选择。这是因为这种方法具有选择$\overline{\mathrm{W}}$的稀疏系数向量的趋势，其中$\overline{\mathrm{W}}$的大多数分量取值为 0。这样的特征可以被丢弃。因此，L1 正则化方法为推荐过程的重要功能子集提供了高度可解释的结果。这些模型的详细讨论可以在文献[C. Aggarwal, 2015]中找到。

线性模型是适用于实数型评分的回归模型的一个例子。在实践中，评分可能是一元的、二元的、基于区间的或分类的（少量的序数值）。目前已经为不同类型的目标类变量设计了各种线性模型，如逻辑回归、probit 回归、ordered probit 回归和非线性回归。一般评分通常被视为二元评分，其中未标记的物品被视为负实例。然而，对于这种情况，存在专门的 positive-unlabeled（PU）模型[B. Liu, 2007]。ordered probit 回归对于基于区间的评分特别有用。此外，在特征和目标变量之间的依赖是非线性的情况下，可以使用诸如多项式回归和核回归等非线性回归模型。当特征数量大，训练样本数量小时，线性模型通常表现相当好，实际上可能优于非线性模型。这是因为线性模型不太容易过拟合。表 2.3 显示了各种回归模型与目标变量（评分）的特点之间的映射关系。

表 2.3 回归模型及适用的评分类型

回归模型	评分特点（目标变量）	回归模型	评分特点（目标变量）
线性回归	实数	多路逻辑回归	分类、顺序
多项式回归	实数	probit	一元、二元
和回归	实数	multiway probit	分类、顺序
二元逻辑回归	一元、二元	ordered probit	顺序、区间

2.4.4 基于内容的推荐算法实例（网络电视节目的精准营销推荐）

（1） 简化说明

例子来自 2018 年的泰迪杯的 B 题，但这里已经过大量简化，仅通过三个用户的收视数据来实现我们基于内容的推荐算法。

（2） 目标

基于每位用户的观看记录以及节目信息，对每位用户实行节目的个性化推荐。

（3） 输入数据

本次实例需要三个数据文件：

1） 节目及其所属标签类型的 01 矩阵：temp_movies_01mat.csv，具体格式如下：

	A	B	C	D	E	F	G	H
1	节目名称	剧情	西部	家庭	惊悚	动画	爱情	情色
2	少年狮王	1	0	0	0	0	0	0
3	大宅门	1	0	1	0	0	1	0
4	父母爱情	0	0	1	0	0	1	0
5	非诚勿扰	0	0	0	0	0	1	0

2） 用户—节目评分矩阵：temp_user_scores_mat.csv，具体格式如下：

	A	B	C	D	E	F	G	H
1	user_id	剧场	王牌对王	最强大脑	金牌调解	七十二家房	飞哥大英雄	倚天屠龙记
2	A	0.4	0.1	0.4	0.6	0.2	0.3	0.6
3	B	0	0	0	0	0	0	0
4	C	0.6	0	0	0	0.1	0.2	0

3） 用户观看的节目—标签 01 矩阵：temp_users_movies_01mat.csv，具体格式如下：

	A	B	C	D	E	F	G	H
1	节目名	剧情	西部	家庭	惊悚	动画	爱情	情色
2	剧场	1	0	0	0	0	1	0
3	王牌对王	1	0	0	1	0	0	0
4	最强大脑	0	0	0	0	0	0	0
5	金牌调解	0	0	0	0	0	0	0

数据文件的全部内容请参考附录，在此不一一列举。

（4） 创建节目画像

```
# 参数说明：
# items_profiles = {item1:{'label1':1, 'label2': 0, 'label3': 0, ...}, item2:{...}...}
def createItemsProfiles(data_array, labels_names, items_names):
    items_profiles = {}
    for i in range(len(items_names)):
        items_profiles[items_names[i]] = {}
        for j in range(len(labels_names)):
            items_profiles[items_names[i]][labels_names[j]] = data_array[i][j]
    return items_profiles
'''
```

（5） 创建用户画像：参数 data_array 存储了所有用户对于其所看过的节目的评分矩阵；用户 user1 对于类型 label1 的隐性评分：user1_score_to_label1；用户 user1 对于其看过的含有类型 label1 的节目 item i 的评分：score_to_item i；用户 user1 对其所看过的所有节目的平均评分：user1_average_score；用户 user1 看过的节目总数：items_count；它们的关系公式为：user1_score_to_label1 = Sigma(score_to_item i - user1_average_score)/items_count。该节目含有特定标签 labels_names[j]

```
# users_profiles = {user1:{'label1':1.1, 'label2': 0.5, 'label3': 0.0, ...}, user2:{...}...}
def createUsersProfiles(data_array, users_names, items_names, labels_names, items_profiles):
    users_profiles = {}
    users_average_scores_list = []   # 计算每个用户对所看过的所有节目的平均隐性评分
    items_users_saw = {}   # 统计每个用户所看过的节目（不加入隐性评分信息）
    items_users_saw_scores = {}   # 统计每个用户所看过的节目及评分
    for i in range(len(users_names)):
        items_users_saw_scores[users_names[i]] = []
        items_users_saw[users_names[i]] = []
        count = 0
        sum = 0.0
        for j in range(len(items_names)):
            if data_array[i][j] > 0: # 用户对该节目隐性评分为正，表示真正看过该节目
                items_users_saw[users_names[i]].append(items_names[j])
                items_users_saw_scores[users_names[i]].append([items_names[j], data_array[i][j]])
                count += 1
                sum += data_array[i][j]
        if count == 0:
            users_average_scores_list.append(0)
        else:
            users_average_scores_list.append(sum / count)

    for i in range(len(users_names)):
        users_profiles[users_names[i]] = {}
        for j in range(len(labels_names)):
            count = 0
            score = 0.0
            for item in items_users_saw_scores[users_names[i]]:
                if items_profiles[item[0]][labels_names[j]] > 0:
                    score += (item[1] - users_average_scores_list[i])
                    count += 1
            if abs(score) < 1e-6:     # 如果求出的值太小，直接置 0
                score = 0.0
            if count == 0:
                result = 0.0
            else:
                result = score / count
            users_profiles[users_names[i]][labels_names[j]] = result
    return (users_profiles, items_users_saw)
'''
```

（6） 计算用户画像向量与节目画像向量的距离（相似度）

```
    def calCosDistance(user, item, labels_names):
        sigma_ui = 0.0
        sigma_u = 0.0
        sigma_i = 0.0
        for label in labels_names:
            sigma_ui += user[label] * item[label]
            sigma_u += (user[label] * user[label])
            sigma_i += (item[label] * item[label])

        if sigma_u == 0.0 or sigma_i == 0.0:    # 若分母为 0，相似度为 0
            return 0
        return sigma_ui/math.sqrt(sigma_u * sigma_i)
'''
```

（7） 代码实现基于内容的推荐算法：借助特定的某个用户 user 的画像 user_profile 和备选推荐节目集的画像 items_profiles，通过计算向量之间的相似度得出推荐节目集。参数说明： user_profile 是某一用户 user 的画像；items_profiles 是备选推荐节目集的节目画像；items_names 是备选推荐节目集中的所有节目名；labels_names 是所有类型名；items_user_saw 是用户 user 看过的节目。

```
def contentBased(user_profile, items_profiles, items_names, labels_names, items_user_saw):
    # 对于用户 user 的推荐节目集为 recommend_items = [[节目名，该节目画像与该用户画像的
      相似度], ...]
    recommend_items = []
    for i in range(len(items_names)):
        # 从备选推荐节目集中的选择用户 user 没有看过的节目
        if items_names[i] not in items_user_saw:
            recommend_items.append([items_names[i], calCosDistance(user_profile,
                                    items_profiles[items_names[i]], labels_names)])
    # 将推荐节目集按相似度降序排列
    recommend_items.sort(key=lambda item: item[1], reverse=True)
    return recommend_items
'''
```

（8） 输出推荐给该用户的节目列表：其中 max_num 为最多输出的推荐节目数

```
def printRecommendedItems(recommend_items_sorted, max_num):
    count = 0
    for item, degree in recommend_items_sorted:
        print("节目名：%s， 推荐指数：%f" % (item, degree))
        count += 1
        if count == max_num:
            break
'''
```

（9） 主程序的设计

```
if __name__ == '__main__':
    all_users_names = ['3', '13', '23']
    # 按指定顺序排列所有的标签
    all_labels = ['剧情', '西部', '家庭', '惊悚', '动画','爱情', '情色', '运动', '音乐', '灾难','悬疑', '儿童', '短
                片', '历史', '动作', '科幻', '传记', '同性', '冒险', '歌舞', '脱口秀', '真人秀', '新闻', '恐怖', '
                奇幻', '犯罪', '喜剧', '纪录片', '战争', '古装', '武侠', '综艺', '电视剧', '邵氏', '电影']
    labels_num = len(all_labels)

    df1 = pd.read_csv('temp_user_scores_mat.csv', sep=',', encoding='gbk',
                      header='infer', error_bad_lines=False)  # 读取用户--节目评分矩阵
    (m1, n1) = df1.shape
    data_array1 = np.array(df1.iloc[:m1 + 1, 1:])  # 所有用户对其看过的节目的评分矩阵
    # 按照"所有用户对其看过的节目的评分矩阵"的列序排列的所有用户观看过的节目名称
    items_users_saw_names1 = df1.columns[1:].tolist()
    # 读取用户观看过的节目及其所属类型的 01 矩阵
    df2 = pd.read_csv('temp_users_movies_01mat.csv', sep=',', encoding='gbk',
                      header='infer', error_bad_lines=False)
    (m2, n2) = df2.shape,   data_array2 = np.array(df2.iloc[:m2 + 1, 1:])
    # 按照"所有用户看过的节目及所属类型的 01 矩阵"的列序排列的所有用户观看过的节目名称
    items_users_saw_names2 = np.array(df2.iloc[:m2 + 1, 0]).tolist()
    # 为用户看过的节目建立节目画像
    items_users_saw_profiles = createItemsProfiles(data_array2, all_labels,items_users_saw_names2)
    # 建立用户画像 users_profiles 和用户看过的节目集 items_users_saw
    (users_profiles, items_users_saw) = createUsersProfiles(data_array1, all_users_names,
                                        items_users_saw_names1,all_labels, items_users_saw_profiles)
    # 读取备选节目及其所属类型的 01 矩阵
    df3 = pd.read_csv('temp_movies_01mat.csv', sep=',', encoding='gbk',
                      header='infer', error_bad_lines=False)
    (m3, n3) = df3.shape, data_array3 = np.array(df3.iloc[:m3 + 1, 1:])
    # 按照"备选推荐节目集及所属类型 01 矩阵"的列序排列的所有用户观看过的节目名称
    items_to_be_recommended_names = np.array(df3.iloc[:m3 + 1, 0]).tolist()
    # 为备选推荐节目集建立节目画像
    items_to_be_recommended_profiles = createItemsProfiles(data_array3, all_labels,
                                            items_to_be_recommended_names)
    for user in all_users_names:    # 开始推荐
        print("给用户 id 为 %s 的推荐节目如下：" % user)
        recommend_items = contentBased(users_profiles[user], items_to_be_recommended_profiles,
                                items_to_be_recommended_names, all_labels, items_users_saw[user])
        printRecommendedItems(recommend_items, 5)
        print('该用户推荐任务完成。')
        print()
'''
```

（10） 运行结果

```
运行以上程序可以得到输出：
给用户 id 为 3 的推荐节目如下：
节目名：父母爱情， 推荐指数：0.215453
节目名：奔跑吧， 推荐指数：0.171533
节目名：一站到底， 推荐指数：0.171533
节目名：平民英雄， 推荐指数：0.171533
节目名：向往的生活， 推荐指数：0.171533
该用户推荐任务完成。

给用户 id 为 13 的推荐节目如下：
节目名：奔跑吧兄弟， 推荐指数：0.630852
节目名：逃学英雄传， 推荐指数：0.630852
节目名：非诚勿扰， 推荐指数：0.565759
节目名：生财有道， 推荐指数：0.472502
节目名：家庭幽默录像， 推荐指数：0.446079
该用户推荐任务完成。

给用户 id 为 23 的推荐节目如下：
节目名：档案， 推荐指数：0.327783
节目名：欢乐颂， 推荐指数：0.293158
节目名：温暖的弦， 推荐指数：0.293158
节目名：繁星四月， 推荐指数：0.293158
节目名：人间至味是清欢， 推荐指数：0.293158
该用户推荐任务完成。
```

2.4.5 基于内容的推荐与协同推荐

与协同方法相比，基于内容的方法具有一些优缺点。基于内容的方法的优点如下：

1) 当一个新物品被添加到评分矩阵中时，它没有任何来自用户的评分。基于记忆和基于模型的协同过滤方法都不会推荐这样的物品，因为没有足够的评分可以用于推荐。但是，基于内容的方法是利用用户之前评分的物品来给出推荐，因此，只要不是新用户，就可以通过与其他物品比较来以一种公平的方式对新物品做出有意义的推荐。协同系统对于新用户和新物品都具有冷启动问题，而基于内容的系统仅对新用户具有冷启动问题。

2) 基于内容的方法在物品的特征方面提供了解释，但协同推荐通常没有办法给出这样的解释。

3) 基于内容的方法通常可以与现成的文本分类器一起使用。此外，每个特定于用户的分类问题通常规模不会像协同系统中那么大。因此，它们在相对较少的工程量下比较容易使用。

另一方面，基于内容的方法也具有协同推荐所没有的缺点。

1) 基于内容的系统倾向于找到与用户迄今为止所看过的类似的物品。这个问题被称为过度特化(overspecialization)，在推荐中总应该有一定的新颖性和偶然性。新颖性指的是该物品与用户在过去看到的不一样，偶然性意味着用户想要发现他们可能没有发现的令人惊讶的相关物品。这是基于内容的系统的问题，其中基于属性的分类模型倾向于推荐非常相似的物品。例如，如果用户从未听过或评价过古典音乐，那么基于内容的系统通常不会向她推荐这样的物品，因为古典音乐将通过与用户迄今为止所评估的属性值非常不同的属性值进行描述。另一方面，协同系统可以利用其同组群体的兴趣来推荐这些物品。例如，协

同系统可能会自动发现某些流行歌曲和古典音乐之间令人惊讶的关联，并将相应的古典音乐推荐给流行音乐爱好者。过度特化和缺乏偶然性是基于内容的推荐系统面临的两个最重要的挑战。

2) 即使基于内容的系统有助于解决新物品的冷启动问题，它们也无法帮助新用户解决冷启动问题。事实上，对于新用户而言，基于内容的系统中的问题可能更为严重，因为文本分类模型通常需要足够数量的训练文档来避免过拟合。只利用一个用户特定的（小）训练数据集，而丢弃所有其他用户的训练数据似乎是相当浪费的。

2.5 基于知识的推荐系统

基于内容的系统和协同系统都要求关于过去购买和评分的大量历史数据。例如，协同系统需要合理范围内大量的评分矩阵，以用于未来的推荐。在数据有限的情况下，推荐系统要么很差，要么无法覆盖用户物品的所有组合。这个问题也被称作冷启动问题。关于这个问题，不同系统的敏感度不同。例如，协同系统具有最高的敏感度，它不能很好地处理新增物品和新增用户。基于内容的推荐系统在某种程度上能够更好地处理新增物品，但是仍然无法为新增用户提供推荐。

此外，一般情况下，这些方法不适用于产品高度定制的领域。如不动产、汽车、旅游产品、金融服务或昂贵的奢侈品。这些物品很少被购买，不能得到充分的评分。

如何处理这种个性化配置和评分缺失呢？基于知识的推荐系统很少依赖明确的用户请求。然而，在这样复杂的领域，用户很难清晰阐明甚至清楚了解其需求如何与产品匹配。例如，用户可能根本不知道汽车的燃料效率和马力之间的对应关系。因此，这些系统利用与用户的交互反馈，允许用户探索内在复杂的产品空间以及学会在多种选择之间进行折中。知识库描述了物品域中不同特征的效用及其折中，从而促进检索和探索过程。知识库在实现检索和探索的过程中十分重要，这样的推荐系统被称为基于知识的推荐系统。

基于知识的推荐系统尤其适用于非定期买的物品推荐。此外，在这样的物品域中，用户在明确其需求上是活跃的。用户很愿意在提供较少输入的情况下接受电影的推荐，但是不愿意在没有指定物品特征的具体信息的情况下接受房子或汽车的推荐。因此，基于知识的推荐系统适用的物品是不同于协同过滤和基于内容的推荐系统的。通常说来，基于知识的推荐系统适合如下情况：

1) 用户想要明确描述其需求。因此，系统中必须有交互组件。协同过滤和基于内容的系统都不允许这种类型的用户反馈。

2) 由于物品的类型和选项所导致的物品域的复杂性，某些特定类型的物品的评分可能很难获得。

3) 在一些领域中（例如计算机），评分可能是对时间敏感的。陈旧的汽车或计算机的评分在推荐系统中是没有用的，因为这些物品会随着用户需求的变化进行更新换代。

在基于知识的推荐系统中，一个重要的部分是用户在推荐的过程中拥有更强的控制权。这种更强的控制权是在复杂问题域中将需求细化的直接结果。表 2.4 给出三种推荐系统概念层次上的比较。注意在不同的推荐系统中，输人的数据也是明显不同的。协同系统和基于内容的推荐系统主要基于历史数据，而基于知识的推荐系统是基于用户描述的需求说明。基于知识的推荐系统的一个显著特征就是在特定领域中的高度个性化定制。通过知识库来实现个性化的定制，知识库中以域或相似性度量的方式嵌入领域知识。除了利用物品属性，某些基于知识的推荐系统也会利用查询时指定的用户属性（如人口统计属性）。在这种情况下，领域知识也可以包含用户属性和物品属性之间的关系。然而，这些属性的应用在基于知识的推荐系统中并不普遍，因为在此类推荐系统中主要关注的是用户需求。

表 2.4 不同推荐系统的概念目标

方法	概念目标	输入
协同	基于协同的方法利用用户本人或同伴的评分和活动，给出推荐结果	用户评分+社区评分
基于内容	基于用户过去对内容（属性）的评分和活	用户评分+物品属性

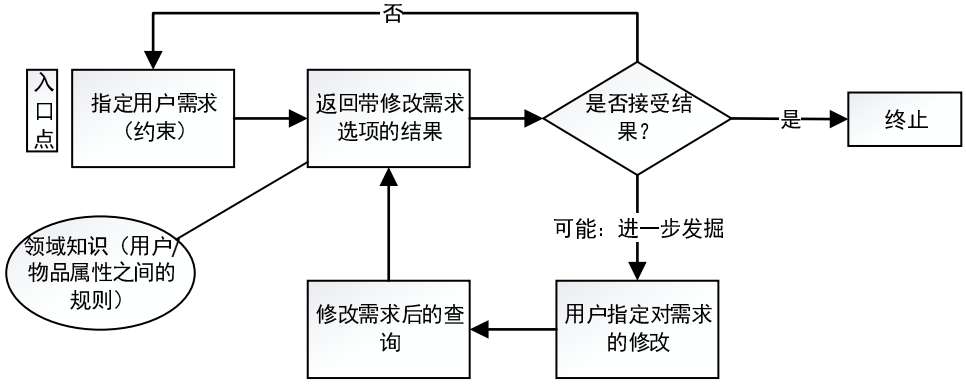
	动，给出推荐结果	
基于知识	基于用户明确的内容（属性）需求说明，给出推荐结果	用户需求说明+物品属性+领域知识

根据用户的交互方法和辅助用户交互的相关知识，基于知识的推荐系统可以被分为两类：

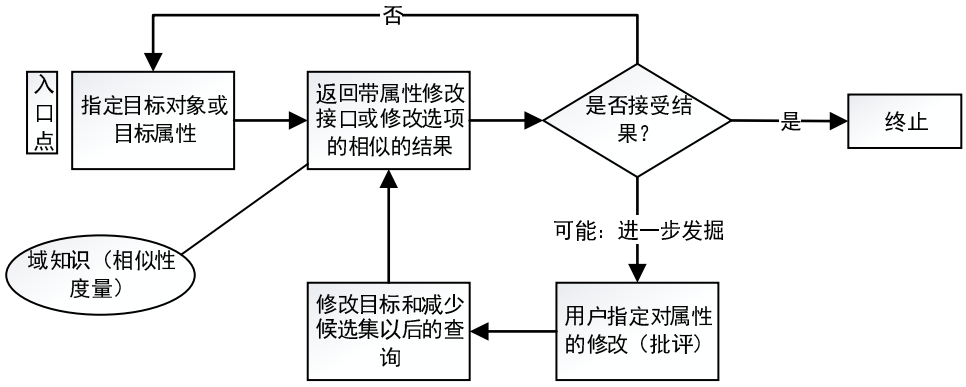
1) 基于约束的推荐系统：在基于约束的推荐系统中，用户指定需求或约束（如物品属性的下限或上限），然后利用特定领域的规则匹配用户需求或物品属性。这些规则代表了系统中使用的领域知识。这些规则使用物品属性上的特定域约束条件（如“1970 年以前的汽车没有恒速操纵器”）。此外，基于约束的系统建立关于用户属性和物品属性关系的规则（如“较年长的投资者不会投资超高风险的物品”）。在这些情况下，也可以在搜索过程中指定用户属性。基于返回结果的数量和类型，用户有机会修改最初的需求。例如，当返回结果较少时，可以适当放松约束条件的限制，返回结果多时，可以增加一些约束条件。这个交互式的搜索过程不断重复，直到用户达到他期望的目的才终止。

2) 基于案例的推荐系统：在基于案例的推荐系统中，由用户来指定特定实例作为目标或锚点。在物品属性上定义相似性度量，以便检索与目标相似的物品。相似性度量通常以领域相关的方式谨慎地被定义。因此，相似性度量就构成了该系统中的领域知识。返回的结果通过与用户交互被当作新的目标实例。例如，当用户看到一个与其预期结果很相近的返回结果时，他就会再发出一个查询，通过修改一些属性，找到他最想得到结果。与此同时，有针对性的批评可以对一些属性值大于或小于特定值的物品进行剪枝。这个交互过程会引导用户得到最终的推荐结果。

基于约束推荐和基于案例推荐的交互过程分别在图 2.5a 和图 2.5b 中给出说明。总体的交互过程是相似的。这两种情况的最大不同在于用户通过特定查询以及与系统交互，来对后面的结果进行提炼。在基于约束的系统中，用户指定特定需求（约束）。在基于案例的系统中，用户指定特定的目标（案例）。相应地，在两个系统中使用不同类型的交互过程和领域知识。在基于约束的系统中，原始用户设置的查询需求通过增加、删除、修改、放松等操作进行修改。在基于案例的系统中，要么通过用户交互修改目标，要么通过用户有方向的引导对搜索结果剪枝，用户只需要简单地说明在搜索结果中是否需要通过某种方法增加、删除、改变一个特定的属性。这样的方式比普通的修改目标方式显示出更明显的对话特点。这些系统假设用户在复杂物品域中无法清晰陈述其需求。在基于约束的系统中，利用知识库规则把用户需求映射到物品属性来解决这个问题；在基于案例的系统中，通过批评的会话方式来解决这个问题。这两个系统都存在交互的过程，帮助用户在复杂物品域中发现符合需求的物品。



a) 基于约束的定义



b) 基于案例的交互

图 2.5 基于知识的推荐系统的交互过程

2.5.1 基于约束的推荐系统

基于约束的推荐系统允许用户在物品属性上指定需求或约束，并利用规则集匹配用户需求和物品的属性。然而，顾客不可能永远基于物品的同一属性描述查询。因此，需要增加一个规则的额外集合，将顾客需求与物品属性相关联。

因此，基于约束的推荐系统有三种类型的基本输入：

1) 第一种类型的输入由属性描述用户的固有性质（如人口统计或风险报告），以及产品的特定需求（如最小卧室数）。这些属性中的一些可以直接和产品属性关联起来，而另一些必须通过知识库才能与产品属性关联。大多数情况下，在交互环节说明顾客性质和需求，但是这些需求在不同的环节可能会不一样。因此，如果另一个用户在一个环节中指定了相同的需求集合，他们就会得到相同的结果。这与其他类型的推荐系统不同，它们的个性化推荐是基于历史数据的，不会发生改变。

2) 第二种类型的输入由知识库表示，把顾客属性/需求映射到不同的产品属性上。映射可以通过直接或间接方法得到：

- 直接：规则将顾客需求与硬性产品属性需求相关联。下面给出示例：

  Suburban-or-rural = Suburban $\Longrightarrow$ Locality = List of relevant localities

  Min-Bedrooms$\geqslant$3 $\Longrightarrow$ Price$\geqslant$100000

  这样的规则也被看作过滤条件。
- 间接：规则将顾客属性/需求与期望的典型产品需求相关联。因此，这类规则可以被看作关联顾客属性与产品属性的间接途径。
- 最后，产品目录包含具备相应的物品属性的所有产品。

2.5.1.1 返回相关结果

返回相关结果的问题可以被看作是约束可满足问题的一个实例，把目录中的每个物品看作是属性中的约束，用析取范式表达目录。表达式与知识库中的规则相结合，判断物品空间是否存在相互一致的区域。

更简单地说，规则和需求的设置可以被看作是日志中的过滤任务。所有顾客需求和与顾客相关的活跃规则被用于构建数据库选择查询。创建过滤查询的步骤如下：

1） 对于用户在用户接口中指定的请求（个人属性），检查其是否匹配知识库中规则的前件。如果存在匹配，规则的结果就被看作有效的选择条件。

2） 在析取范式中，用扩展需求构建数据库查询。这种方法本质上将顾客属性约束和请求属性约束映射到物品域的约束上。

3） 用选择查询检索与用户请求相关的目录中的实例。

值得注意的是，大多数基于约束的系统允许用户在自己的部分对需求或属性（如偏好、人口统计信息）进行说明。换句话说，指定的信息并不是一成不变的；如果另一个用户也提出了相同的需求，他们将得到相同的结果。对于大多数基于知识的系统，这种情况是普遍存在的。

满足约束的物品结果列表提供给用户，本节后面将讨论排序物品的方法。用户可以修改请求，获取更准确的推荐。探索和提炼的总过程让顾客发现意想不到的推荐。

2.5.1.2 交互方法

用户和推荐系统之间的交互进行过程如下：

1) 用户利用交互界面说明初始偏好。一种常见的方法是利用网页表格，其中可以填入需要属性的值。图 2.6 给出一个房屋购买的例子。用户会被问到一系列问题，来明确他们的初始偏好。例如，汽车推荐网页 Edmunds.com 给用户提出一些问题，关于一些特定的特征让用户说明其偏好。第一个界面的答案可能影响到下一个界面的问题。

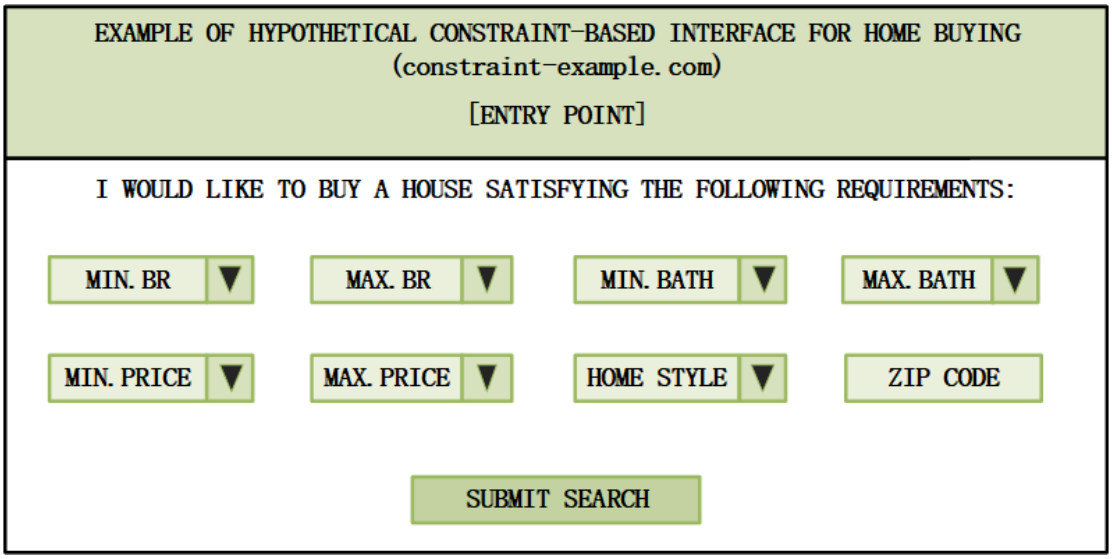


图 2.6 一个基于约束的推荐系统中初始用户界面的假设示例(constraint-example.com)

2) 给用户提供匹配物品的排序列表，并解释一下物品返回的原因。一些情况下可能没有满足用户需求的物品。此时，可以对请求进行适当的放松。例如，在图 2.7 中，查询没有返回结果，建议采取可能的放松条件。在某些情况下，返回的物品太多，需要增加一些约束（需求）。例如，在图 2.8 中，返回了过多的结果，因此给查询增加了一些可能的约束条件。

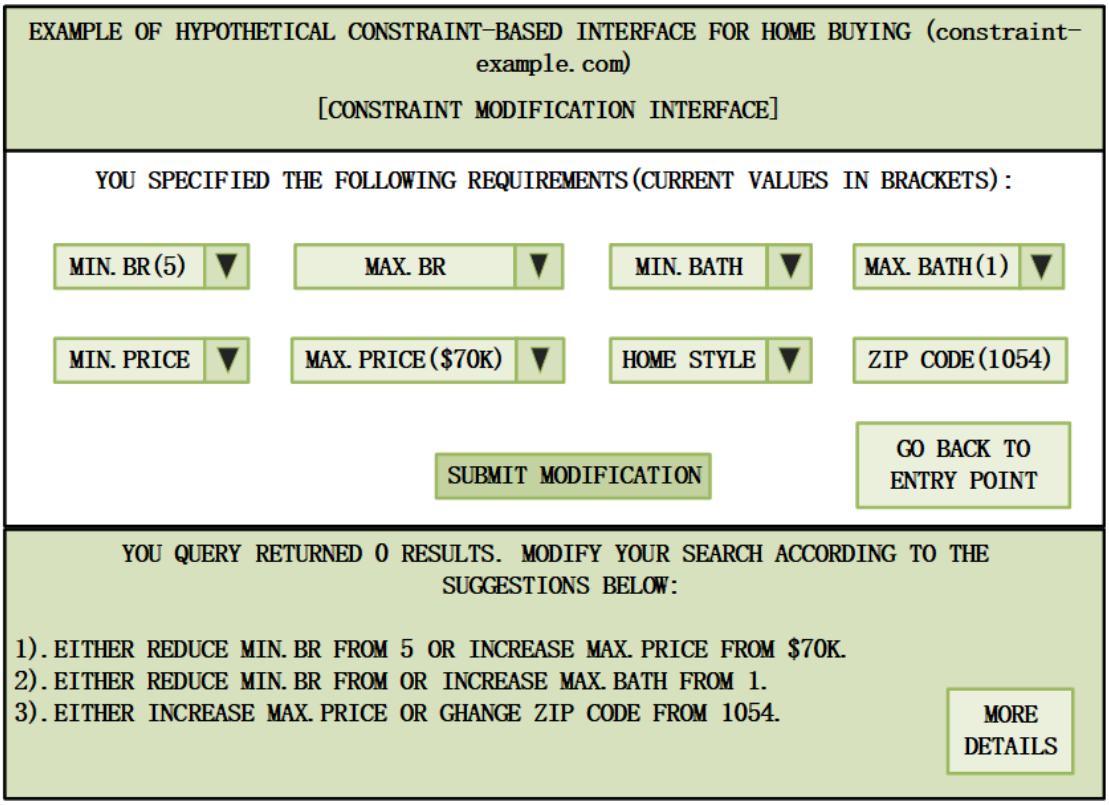


图 2.7 在一个基于约束的推荐系统中用于处理空查询结果的一个用户界面的假设示例(constraint-example.com)

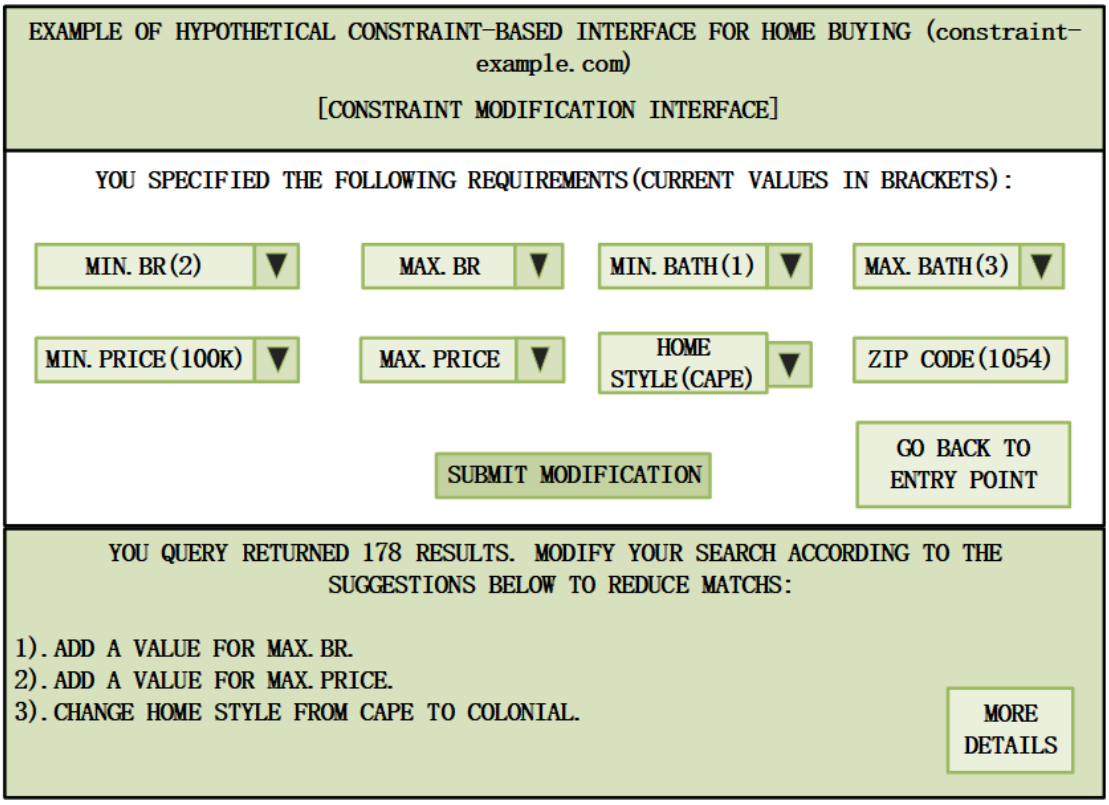


图 2.8 在一个基于约束的推荐系统中用于处理大量查询结果的一个用户界面的假设示例(constraint-example.com)

3) 用户根据返回结果重新定义其需求，可以增加额外的需求或移除某些需求。例如，当返回的结果为空时，就需要放松某些需求。用约束可满足方法识别需要放松的可能候选约束集。因此，系统可以帮助用

户更智能有效地修改查询。

因此，该整体方法使用一个选代反馈循环来帮助用户做出有意义的决策。设计一个系统，其可以引导用户以达到提高她对可用选择的认识的需求，是至关重要的。

这种交互有几个方面，需要显式计算以帮助用户。例如，一位用户通常将不会为所有产品的属性指定期望值。具体来说，在我们的购房示例中，用户可能仅对卧室的数量指定约束，但不对价格指定任何约束。在这种情况下有几种解决方案是可行的：

a) 系统可以使其他属性不受约束，并且仅基于已指定的约束来检索结果。例如，可以考虑所有可能的价格范围，以便向用户提供第一组响应。

b) 在某些情况下，可能会为用户建议一些默认值，以提供向导。默认值只用于指导用户选择值，或者如果用户没有为该属性选择任何值（包括默认值），则默认值将实际用于查询。可以认为，在查询中包括默认值（没有明确指定）可能会导致推荐系统中的显著偏差，特别是在默认值没有被充分研究时。一般来说，默认值只能作为给用户的一条建议。这是因为默认值的主要目的应朝着自然值的方向引导用户，而不是替代未指定的选项。

在大多数情况下，需要以领域相关的方式选择默认值。在发出查询后，系统会提供一个目录中可能匹配的排名列表。因此，能够有意义地对这些匹配进行排序是很重要的，并且如果需要，还要提供推荐结果的说明。在返回的匹配结果集太小或太大的情况下，可以放宽或收紧需求以向用户提供进一步的指导。值得注意的是，提供解释也是指导用户进行更有意义的查询优化的一种聪明的方法。接下来，我们将讨论交互式用户指导的各个方面。

2.5.1.3 排序匹配的物品

存在一些根据用户需求对物品进行排序的自然方法。最简单的方法是允许用户指定一个单一的数字属性，根据该属性对这些物品进行排序。例如，在购房应用中，系统可能向用户提供基于住宅价格、卧室数量或与特定邮政编码的距离来对物品进行排序的选项。事实上，这种方法用于许多商业界面。

使用单个属性会削弱其他属性的重要性。一个常用的方法是使用效用函数来排列匹配的物品。令$\overline{\mathrm{V}} = (v_1 \cdots v_d)$是定义匹配产品属性的值的向量。因此，内容空间的维度为 d。效用函数可以被定义为各个属性的效用的加权函数。每个属性具有一个分配给它的权重w_j，以及一个由函数$f_j(v_j)$定义、依赖于四配的属性值v_j的贡献值。然后，匹配物品的效用$U(\overline{\mathrm{V}})$由下式给出：

$$U(\overline{\mathrm{V}}) = \sum_{j=1}^{d} w_j \cdot f_j(v_j) \tag{2.41}$$

显然，需要实例化w_j和$f_j(v_j)$的值才能学习效用函数。有效的效用函数的设计通常需要特定领域的知识，或者从过去的用户交互中学习数据。例如，当v_j是数值时，可以假设函数$f_j(v_j)$在v_j中是线性的，然后通过抽出来自各种用户的反馈来学习线性函数和w_j的系数。通常，来自某些用户的训练数据被抽取出来，这些用户被赋予对一些样本物品进行排序的任务。然后使用这些排序结合使用回归模型来学习上述模型。这种方法与联合分析的方法相关。联合分析定义了人们如何评估构成一个个人产品或服务的不同属性的正式研究的统计方法。

2.5.1.4 处理不可接受的结果或空集

在许多情况下，一条特定的查询可能返回一个空结果集。在其他情况下，返回的结果集可能不足以满足用户的要求。在这种情况下，用户有两个选择。如果认为不存在修改约束的直接方式，她可以选择从入口点重新开始。或者，她可以决定改变或放宽下一次交互式迭代的约束。

用户如何在是否放宽约束上做出一个有意义的选择，以及该以何种方式去做？在这种情况下，向用户

提供放宽当前需求的指导通常是有帮助的。这些建议被称为修补建议。这个想法是能够确定最小的不一致约束集，并将它们呈现给用户。用户更容易接受最小的不一致约束集，并找到放宽这些集合中的一个或多个约束的方法。

2.5.1.5 添加约束

在某些情况下，返回结果的数量可能非常大，用户可能需要建议添加到查询中的可能的约束。这时可以使用各种方法来给用户建议约束以及可能的默认值。经常通过挖掘历史会话日志来选择此类约束的属性。历史会话日志可以在所有用户上定义，也可以在当前特定的用户上定义。后者提供更具个性化的结果，但对于不经常购买的物品（例如汽车或房屋）来说是不可获得的。值得注意的是，基于知识的系统通常被设计为不精确地使用这种持久的和历史的信息，因为它们需要在环境中工作。尽管如此，这样的信息通常可以在改善用户体验的时候变得非常有用。

如何使用历史会话数据呢？其思想是选择受欢迎的约束。例如，如果一位用户已经对一组物品属性指定了约束，则其他包含这些属性中的一个或多个的会话就会被识别。

在许多情况下，用户在过去指定约束的时间顺序是可用的。在这种情况下，通过将约束看作一个有序集而不是无序集，也可以使用用户指定约束的顺序。实现此目标的一个简单方法是确定在先前会话中遵循当前指定约束属性集的最频繁属性。序列模式挖掘可用于确定这种频繁属性。可以基于约束在数据库中的选择性或基于在过去会话中用户的平均规定来建议约束。

2.5.2 基于案例的推荐系统

在基于案例的推荐系统中，使用相似性度量来检索与指定目标（或案例）相似的示例。

在如何改进结果方面，基于约束的推荐系统和基于案例的推荐系统之间也存在重大差异。基于约束的系统使用需求放宽、修改和收紧来改进结果。最早的基于案例的系统主张重复修改用户查询的要求，直到找到合适的解决方案。因此，“批评”方法被提出。批评方法的基本思想是，用户可以选择一个或多个检索结果，并指定以下形式的进一步查询：

“给我更多像 X 的物品，但根据指南 Z，它们的属性 Y 要不同。”

关于是否选择一个或多个属性进行修改以及如何给定修改属性的指南有很多方法。批评的主要目标是支持物品空间的交互式浏览，其中用户逐渐通过已检索的示例来了解更多的可用选项。物品空间的交互式浏览具有以下优点：它是迭代查询制定过程中的一个用户学习过程。通常，通过重复和互动的探索，用户可能会达到一开始就无法达到的物品。

通过反复批评，用户有时会得到与初始查询指定的完全不同的最终结果。毕竟，用户通常很难在一开始就表达出他们所期望的所有特征。例如，在查询过程开始时，用户可能不会意识到期望的房屋特征的一个可接受的价格。这种交互方法弥合了她的初步理解与物品可用性之间的差距。正是这种辅助浏览的功能使得基于案例的方法在提高用户意识方面变得如此强大。用户有时也可能通过重复地减少候选集而得到一组空的候选集。这样的会话可以被视为无结果的会话，并且在这种情况下，用户必须在入口点从头重新开始。请注意，这与基于约束的系统不同，在后者中用户还可以选择放宽当前的要求集来增大结果集。产生这种差异的原因是基于案例的系统通常将候选结果的数量从一个周期减少到下个周期，而基于约束的系统则不会。

为了使基于案例的推荐系统有效运行，系统的两个关键方面必须进行有效的设计：

1) 相似性度量：相似性度量的有效设计在基于案例的系统中非常重要，以便检索相关结果。各种属性的重要性必须适当地纳入相似度函数中，使系统有效地工作。

2) 批评方法：使用批评方法来支持物品空间的交互探索。各种不同的批评方法可用于支持不同的探索目标。

在本节中，我们将讨论基于案例的推荐系统设计的所有这些重要的方面。

2.5.2.1 相似性度量

考虑由 d 个属性来描述产品的应用。我们想确定在 d 个属性的领域的子集 S（$|S| = s \leq d$）上定义的两个部分属性向量之间的相似度值。令$\overline{X} = (x_1 \cdots x_d)$和$\overline{T} = (t_1 \cdots t_d)$表示可能部分指定的两个 d 维向量，其中$\overline{T}$表示目标。假设在两个向量中至少指定属性子集$S \subseteq \{1 \cdots d\}$。请注意，我们使用部分属性向量，因为这些查询通常仅在用户指定的一小部分属性上定义。例如，在上述房地产示例中，用户可能仅指定查询特征的一个小的集合，例如卧室或浴室的数量。然后，两组向量之间的相似度函数$f(\overline{T}, \overline{X})$定义如下：

$$f(\overline{T}, \overline{X}) = \frac{\sum_{i \in S} w_i \cdot Sim(t_i, x_i)}{\sum_{i \in S} w_i} \tag{2.42}$$

这里，$Sim(t_i, x_i)$表示值x_i和y_i之间的相似度。权重w_i表示第 i 个属性的权重，它规定了该属性的相对重要性。那么相似度函数$Sim(t_i, x_i)$和属性重要性w_i是如何被学的呢？

首先，我们将讨论如何确定相似度函数$Sim(t_i, x_i)$。请注意，这些属性可能是定量的或分类的，这进一步增加了这种系统的异构性和复杂性，此外，属性可以根据较高或较低的值是对称的或不对称的。例如，对于一个属性，如相机的分辨率，用户可能更期望找到更大的分辨率，但这种倾向可能不如考虑价格那么强烈。其他属性可能是完全对称的。在这种情况下，用户想要属性值准确地定在目标值t_i上。一个对称度量的示例如下：

$$Sim(t_i, x_i) = 1 - \frac{|t_i - x_i|}{max_i - min_i} \tag{2.43}$$

这里，max_i和min_i表示属性 i 的最大或最小的可能值。或者，可以使用标准差σ_i（在历史数据上）来设置相似度函数：

$$Sim(t_i, x_i) = max\left\{0, 1 - \frac{|t_i - x_i|}{3\sigma_i}\right\} \tag{2.44}$$

请注意，在对称度量的情况下，相似度完全由两个属性之间的差异定义。在一个非对称属性的情况下，可以额外添加一个非对称的奖励，这取决于目标属性值是更小还是更大。对于属性值较大是更好的情况，可能的相似度函数的示例如下：

$$Sim(t_i, x_i) = 1 - \frac{|t_i - x_i|}{max_i - min_i} + \alpha_i \cdot I(x_i > t_i) \cdot \frac{|t_i - x_i|}{max_i - min_i} \tag{2.45}$$

这里，$\alpha_i \cdot I(x_i > t_i) \cdot \frac{|t_i - x_i|}{max_i - min_i}$为非对称奖励，$\alpha_i \geq 0$是一个用户定义的参数，$I(x_i > t_i)$是一个指示函数，如果$x_i > t_i$，则取值为 1，否则为 0。注意，只有当属性值$x_i$（例如，相机分辨率）大于目标值$t_i$时，奖励才生效。对于属性值较小是较好的情况（例如价格），奖励函数是类似的，只是该情况使用如下指标函数进行奖励：

$$Sim(t_i, x_i) = 1 - \frac{|t_i - x_i|}{max_i - min_i} + \alpha_i \cdot I(x_i < t_i) \cdot \frac{|t_i - x_i|}{max_i - min_i} \quad (2.46)$$

α_i的值以一个高度领域特定的方式选择。对于$\alpha_i > 1$ 的值，“相似度”实际上随着与目标的距离增加而增大。在这种情况下，将$Sim(t_i, x_i)$视为一个效用函数而不是一个相似度函数是很有帮助的。

设计相似度函数的第二个问题就是不同属性之间的相关度的测定。第 i 个属性的相关重要性用公式（2.32）中的参数w_i来规定。对于领域专家来说，通过实验和经验来硬编w_i的值是可能的。别的可能是使用用户的反馈来了解w_i的值。可以把成对的目标对象呈现给用户，而且用户可能会被要求给这些目标对象的相似度进行评分。可以使用这些反馈结合一个线性回归模型来确定w_i的值。通常，由用户指定相关排名，而不是给成对的对象去指定明确的相似度，要更加简单。

2.5.2.2 批评方法

批评的动机是基于这样的一个事实，那就是在最初的查询中用户往往不能够准确地声明他们的需求。在某些复杂的领域，他们甚至没有发现以一种语义上有意义的方式把他们的需要翻译成产品领域的属性值是非常困难的。只有在看过一个查询的结果之后，这个用户可能才意识到她已经有点难以表达她的查询。批评就是被设计用来在这个事实之后提供给用户这样的能力。

在一个简单批评中，对于推荐中的某个物品的某一个特征，用户具体说明一个单独的改变。简单批评方法的主要问题就是它难以导航。如果推荐的物品包含太多需要改变的属性，那么就会产生一个特别长的连续链。而且，当其中的一个属性改变的时候，推荐系统可能需要自动改变一些其他取决于物品的可用性的属性值。大多数情况下，在一个给定的循环中，保持其他的属性值在一个明确的常数值是不可能的。因此，当用户已经把一些属性值修改成他们的期望值的时候，他们可能发现其他的属性值已经变得不能被接受。推荐循环的值越大，用户对于在别的早期迭代中可接受的属性值的变化的控制将会越少。这种问题往往是因为用户不了解问题领域中的自然权衡。比如，某个用户可能不理解马力和燃料效率之间的权衡，从而尝试去寻找一个大马力且有着 50 英里（1 英里≈1.61 千米）高能源效率的汽车。很多批评界面的主要问题就是推荐物品的下一个集合是基于最近被批评的物品，而且没有一种方式能够导航回早期的物品。因此，简单批评的长循环有时候可能会导致没有结果。

复合批评的发展是为了减少推荐循环的长度。在这种情况下，用户能具体说明如何修改一个单一循环中的多种特征。例如，一个汽车导航系统允许用户去具体说明多种隐藏在用户能够理解的（比如优等的、宽敞的、便宜的、漂亮的）非正式描述信息之后的修改。例如，领域专家可能认为“优等的”暗示某个有着更高价值和精致的内部构造的型号。当然，用户也能够直接去修改需要的产品特征，但是这会增加其负担。常见的批评是一个用户可能需要一个“优等的”汽车，但是就产品特征而言，如汽车的内部构造，他们可能没法简单具体地去表述。另一方面，一个类似“优等的”判定是更直观的，而且就产品特征而言，它能够被一个领域专家编码。之所以设计这种交互过程，就是为了帮助他们以一种直观的方式来了解复杂的产品空间。

复合批评的主要优势在于为了发出一个新的查询或者用之前的查询来修剪查询结果，用户可以改变目标推荐的多种特征。因此，这种方法允许在产品特征空间上大的跳跃，而且用户经常可以对评论过程有更好的掌控。这对于减少推荐循环的数量和做出更有效率的探索过程都是有帮助的。但是，在帮助用户了解产品空间上，尚不清楚复合批评是否总是优于简单批评；短的批评循环也能减少用户了解产品空间的特征之间不同的权衡和相关性的可能性。另一方面，某个用户有时也可能通过简单批评的缓慢和艰苦的过程来更多地了解产品空间。

尽管复合批评允许在导航空间上有大的跳跃，但是就其不依赖于检索结果这个意义而言，其缺点在于呈现给读者的反馈是静态的。例如，如果用户在浏览汽车，而且她已经浏览了很多大马力的昂贵汽车，那么增加马力和价格的选项还是会在推荐界面中。很显然，具体说明这些选项将会导致一个无意义的结果。这是因为用户对于复杂的产品空间的固有权衡的了解往往不足。

在动态批评中，目标就是在检索结果中使用数据挖掘来决定探索过程中最有成效的路径然后把它们推荐给用户。因此，动态批评在定义上是复合批评，因为它们大部分总是代表呈现给用户的变化的结合体。主要的区别在于基于当前的检索结果，呈现最相关的可能性的那部分子集。因此，动态批评的设计是为了在搜索过程中给用户提供更好的指导。

动态批评的一个重要的方面就是发现产品特征变化的频繁结合。

2.5.2.3 批评的解释

给出批评过程的解释是有用的，因为这能够帮助用户更好地理解信息空间。有几种用来提高批评质量的解释方式，以下是一些例子：

1) 在简单批评中，对于用户来说以一种无结果的方式进行导航很常见，因为其不了解物品空间中固有的权衡。例如，一个用户可能成功地增加了马力，增加了每加仑的里程数，然后尽力去减少预期的价格。这时系统可能无法给用户展示一个可以接受的结果，而且用户也将必须重新开始导航过程。在会话的最后，系统需要自动确定无效对话的内在权衡。使用相关性和同现统计来决定这样的权衡往往是可行的。然后用户可以获得他们之前输入的批评上冲突的见解。

2) 文献[J. Reilly et al., 2005]已经展示了在一段会话内解释是如何与动态的复合批评相结合的。例如，Qwikshop 系统提供了满足每个复合批评的那部分实例的信息。这就可以在用户做出批评选择之前，给用户提供一个有关其将要探索的空间大小的直观展示。在这段会话中给用户提供更好的解释可以提高获取有意义结果的可能性。

基于批评系统的主要风险就是，用户以无目的的方式漫游知识空间，从而无法成功找到他们想要寻找的内容。通过给界面增加解释可以减少这种风险发生的可能性。

2.6 推荐系统评估

为了对多种多样的推荐算法的有效性有一个充分的了解，设计一个良好的评价系统是至关重要的。我们在本章的后续部分也会发现，推荐系统的评估是多方面的，所以一个单一标准不可能实现设计者的全部设计目标。一个不正确的试验评价系统会导致一个特别的算法或模型的精确性要么被过分高估，要么被过分低估。

特别要说的是，从为推荐系统设计评估方法的角度来看，下面几点也很重要。

1) 评估目的：当我们用推荐精确性来评价推荐系统的时候，但其实这种方法带来的用户体验往往并不是那么好。尽管精确性确实是整个评估过程最重要的一部分，但是还有很多次要的指标，比如新颖性、信任度、覆盖率和惊喜度等对用户体验也很重要。这是因为这些指标对转换率有着或多或少的影响。尽管如此，对这些因子的定量经常是相当主观的，而且通常也没有硬性措施来提供一个数值指标。

2) 实验设计因素：尽管精确性被用来当评估标准，设计实验以确保精确性不要被高估或者低估是至关重要的。例如，如果同一个具体的评价同时被用来建模和评估精确性，那么精确性必然会被高估。在这种背景下，认真设计实验就尤其重要。

3) 精确性：如果抛开那些次要标准不谈，精确性确实依然会是评估过程最重要的部分。推荐系统将会被以预测精确性或者排名精确性来评估。因此，很多像平均对误差和均方误差这样的普通指标也会被广泛使用。排名评估可以使用各种方法进行，如基于效用的计算、秩相关系数以及受试者操作特征曲线（ROC 曲线）。

2.6.1 评估范例

本节仔细研究了推荐系统的三种不同类型的评估方法：用户调查、在线评估和用历史数据集的离线评

估。前两种类型都是和用户有关的，尽管它们使用的方法略微不同。前两种方法的主要区别在于如何召集用户来做调查。尽管在线评估方法对推荐算法的真实作用有着深刻的研究，但是在它的发展过程中依然面临着几个障碍。接下来，我们将讨论这几种不同的评估类型。

2.6.1.1 用户调查

在用户调查中，测试对象被动态要求执行一些具体的需要与系统进行交互的任务。我们可以在交互之前或者之后来收集用户给出的反馈信息，而且系统会收集有关用户和推荐系统交互的信息。然后，这些信息被用来推断用户喜欢什么，不喜欢什么。例如，用户可能被要求和一个使用了推荐方法的产品页面交互，并且在之后给出对这个推荐系统的反馈。这样的方法可以用来评价推荐系统中算法的有效性。或者，用户可能被要求去听几首歌曲，然后通过给这些歌曲评分的方式来给出反馈。

2.6.1.2 在线评估

在线评估也要利用用户调查的方法，只可惜这些调查对象往往很少是已部署或者商业系统的真实用户。这种方法很少受到招募过程误差的影响，因为在通常情况下，用户往往会直接使用这个系统。这种系统经常被用来评估多个算法的不同表现[R. Kohavi et al.,2009]。通常，用户被随机抽样，并对每一个随机选择的用户来测试各种不同的算法。转化率是一个典型的用来测量推荐系统对用户有效性的指标。转化率度量某个用户选择系统推荐给他的物品的频率。例如，在一个新闻推荐系统中，可以计算用户选择系统推荐的文章的次数。如果需要，预计的花费或者利润会被添加到每个物品中，使得测试对物品的重要性敏感。这些方法同时也适用于 A/B 测试，并且会测量推荐系统在用户端的直接影响。这些方法的基本思路就是如下比较两种算法：

1) 把用户分割成 A 和 B 两个小组；

2) 用两个算法分别在两个小组里运行一段时间，在这期间控制两个小组的其他所有条件（比如每个小组成员的选择过程）尽可能相似；

3) 测试过程结束后，比较两个小组的转化率（或者其他回报指标）。

一个观测结果是，如果每次和推荐系统交互的回报都能单独测试出来，那么就没有必要严格地把用户分成各个小组。在这种情况下，可以随机地把其中的一个算法呈现给同一个用户，并且这种具体交互带来的回报也是可以测量的，这些评估推荐系统的方法还可以被推广到那些更有效的推荐算法的发展中，从而得到被称为多臂博机(multi-arm bandit)的算法。

2.6.1.3 使用历史数据集进行离线评估

离线测试使用的是评分等历史数据。在某些情况下，时间信息也可能和评分相关，比如在每个用户已经评分过的物品的时间戳。Netflix Prize 数据集是一个众所周知的历史数据集。其最初发布在一个在线测试的文本中，并由此成为衡量很多算法的基准。使用历史数据集的主要优点是不用要求大量的用户参与。一旦获得了一个数据集，就能拿来作为一个通过一系列的设置来比较多种算法的标准化基准。而且，来自不同领域（如音乐电影、新闻等）的多样化的数据集可以被用来测试推荐系统的普适性。

2.6.2 评估设计的总体目标

2.6.2.1 精确性

精确性评估的要素如下：

1) 设计精确性评估：所有我们在评分矩阵中观察到的数据条目不能既用来训练模型又用来评估精确性。这样做的目的是为了避免因为过拟合造成的过分高估。

2) 精确性指标：精确性指标被用来评估指定的用户—物品组合的评分预测精确性评测或者由推荐系提供的前 k 个排名的精确性。具体来说，评分矩阵中集合 E 的物品的评分是隐藏的，并且精确性评估是在这些隐藏项上进行。

2.6.2.2 覆盖率

即使在一个推荐系统已经高度准确的情况下，它甚至也经常不能对其中某一部分物品或者用户做出推荐。这就涉及覆盖率。推荐系统的这种局限性是评分矩阵稀疏所造成的。例如，在一个每行每列只含有一个数据元素的评分矩阵中，那么，几乎所有的推荐算法都不能够给出有意义的推荐。尽管如此，不同的推荐系统在提供覆盖率上还是有不同的倾向性。然而在实际的设置中，因为在那些不能够准确预测的矩阵里使用了系统默认值，所以系统经常有百分百的覆盖率，当某个具体的用户—物品组合不能被预测的时候，这个默认值的例子将会设为所有用户对物品的评分的均值。因此，精确性和覆盖率的折中往往需要被纳入整个评估过程当中。目前存在两种类型的覆盖率，一种指的是用户空间覆盖率，另一种指的是物品空间覆盖率。

用户空间覆盖度量可预测至少 k 个评分的用户占比。k 的值应设置为推荐列表的期望大小。当用户可以预测少于 k 个评分时，不可能向用户呈现大小为 k 的有意义的推荐列表。这种情况适用于当一个特定用户很少有和其他用户一样评分的项时。

物品空间覆盖率的概念和用户空间覆盖率的概念类似。物品空间覆盖率测量那些至少被 k 个用户评分过的物品。在实践中，这个概念根少被使用，因为推荐系统主要是为了给用户提供推荐清单，而不是仅仅为了给物品推荐用户。

物品空间覆盖率的另一种定义形式是类别覆盖，适用于推荐列表。注意前一个定义是为评分值预测量身定做的。设想一个场景，其中矩阵里面的每一个评分值都可以被一个算法预测，但是推荐给每一个用户的前 k 项物品往往是相同的。因此，尽管之前关于物品空间覆盖率的定义意味着更好的表现，但是实际对用户来说覆盖率是非常有限的。换句话说，对用户的推荐并不是多样化的，也不能完全覆盖所有类别。令 T_n 代表推荐给用户 $\mathrm{u} \in \{1 \cdots m\}$ 的前 k 项。类别覆盖率 CC 被定义为至少推荐给一个用户的物品的占比。

$$\mathrm{CC} = \frac{|U_{u-1}^{m} \mathrm{T}_n|}{n} \tag{2.47}$$

其中 n 表示物品的总数。这个值可以很容易通过实验来获取。

2.6.2.3 置信度和信任度

对评分的估计是一个不准确的过程，因为这个过程会随着手中特定数据集的改变而显著地变化。而且，算法采用的方法可能也会对预测的分数产生重要的影响。这就往往造成用户质疑预测的精确性。鉴于此，许多推荐系统会为系统评分赋予一个置信度估计。例如，系统提供一个预测分数的置信度区间。一般来讲，那些能准确地推荐更小置信度区间的推荐系统是更加可取的，因为这样会增强用户对于系统的信任。对于两个用同样方法来赋予置信度的算法而言，测量预测误差落在置信度区间的程度是可行的。例如，如果两个推荐系统能为每个评分提供 95%的置信度区间，我们可以测量两个算法赋予的区间的绝对宽度。其中有更小置信度区间宽度的算法将会胜出，尽管两个算法都正确（比如，都在指定的区间内）地在至少 95%的时间在隐藏的评分中。如果其中有一个算法低于要求的 95%精确性，那么这个算法必然输了。遗憾的是，如果一个系统使用 95%的置信度区间而另外一个使用 99%的置信度区间，完全有意义地比较它们是不可能的。因此，只有在两种情况下设置同样的置信度，才有可能对系统进行比较。

信任度经常不是为了达到和效用一样服务于推荐系统的目标。例如，如果一个推荐系统推荐了一些用户已经喜欢或者知道的物品给这个用户，那么可以认为这样的推荐对用户来说可用性很低。另一方面，这样的推荐却可以增加用户对系统的信任。所以说在推荐已经被用户知道的前提下，信任度和类似新颖度这样的指标是矛盾的，这样的情况是不受欢迎的。因此在推荐系统中，指标对立是很常见的。最简单的测量信任度的方法就是在用户对结果信任明确怀疑的情况下，采用用户调查实验。这样的实验同时也适用于在线实验。总的来说，通过离线实验来测量信任度是很难的。

2.6.2.4 新颖度

推荐系统中的新颖度用来评估推荐系统向用户推荐他们不知道的物品或者他们之前没见过的物品的可能性。有关新颖度概念的讨论可参考文献[J. Konstan et al.,2006]，没有见过的推荐物品往往会增加用户发掘他们之前并不知道的喜好的能力。这比发掘那些用户已经知道的但是还没有评分的物品显得重要多了。在多种类型的推荐系统中，比如基于内容的方法，推荐的物品更倾向于那些用户显然喜欢的，因为系统的特性是推荐期望的物品。在底层系统中，一小部分这样的推荐可以提高用户的信任度，然而就提高转化率而言它们往不是很有用。最自然的评测新颖度的方法是在线实验，在实验中明确地询问用户他们是否之前就已经熟悉这个推荐物品了。

2.6.2.5 惊喜度

“惊喜度”这个单词逐字翻译的意思就是“幸运地发现”。因此，惊喜度是用来评测成功推荐的惊讶级别。换句话说，推荐应该是意想不到的。相比之下，新颖度仅仅要求用户之前不熟悉推荐的物品。惊喜度比新颖度的要求更为严格。

推荐系统中有几种评测惊喜度的方法。这个概念也会在信息检索应用的上下文中出现[Y. Zhang et al., 2002]，评估惊喜度的在线和离线的方法如下所示：

1) 在线方法：推荐系统收集用户对推荐的有用性和显著性的反馈信息。那部分既有用又不显著的推荐被作为惊喜度的测量标准。

2) 离线方法：也可以使用一种原始的推荐来自动产生有关一个推荐的显著性的信息。那些原始的推荐系统可以选用基于内容的推荐系统，这种推荐系有推荐显著物品的习惯。然后，可以确定被推荐的推荐列表里面正确的前 k 个（比如，很高的隐藏评分），且没有被之前的推荐系统推荐的那些物品。这个占比可以用来度量惊喜度。

值得关注的是，仅仅评测那部分不显著的物品是不够的，因为一个系统可能推荐不相关的物品。因此，惊喜度的评价往往是和物品的有用性紧密结合的。惊喜度对于提高推荐系统的转化率有着长期的影响，尽管其有时会违背最大化精确度这一目的。

2.6.2.6 多样性

多样性的概念意味着含有单值推荐列表的被推荐集合应该尽可能是多种多样的。例如，考虑这样一种情况，有三个电影被推荐给用户在列表的前三项。如果这三个电影是同一题材而且有相似的演员，那么她有很大概率全部都不喜欢。呈现给用户多样化类型的电影能够提高用户选择其中之一的概率。注意到多样性经常是在一个推荐集合中来测量的，而且和新颖度及惊喜度密切相关。保证更复杂的多样性往往能够提高推荐列表的新颖度和惊喜度。而且，推荐的多样性还可以提高系统的销售多样性和类别覆盖率。

多样性能依据两两用户的内容相关的语意相似度来度量。物品的空间向量表示被用来进行相似度的计算。例如，如果有一个含有 k 个物品的列表推荐给用户，然后计算物品的列表里每两个物品的相似度。所有对的平均相似度就是多样性。平均相似度越低，就表明多样性越高。与使用精确性度量的情境相比，多

样性经常可以提供多种不同的结果。

2.6.2.7 健壮性和稳定性

如果推荐系统不会受到“假评分攻击”或“模式随时间显著变化”等情况的影响，那么可以认为推荐系是稳定和健壮的。通常情况下，利益驱动等动机会使得一些用户提供虚假评分。例如，在亚马逊购物网站上，一本书的作者或者出版商可能会提供与本书有关的虚假的正面评分，或者会给竞争对手的图书提供一些虚假的负面评分。相应的评估系统健壮性和稳定性的方法会被采用来抵御攻击。

2.6.2.8 可扩展性

最近几年，从许多用户那里收集大量的评分和隐式反馈信息变得越来越容易。在这种情况下，随着时间的推移，数据集的大小也在持续增加。因此，设计出能够有效且高效地处理大量数据的推荐系统也变得越来越重要。一些方法可以用来决定系统的可扩展性。

1) 训练时间：大多数推荐系统要求一个训练阶段，这个阶段是独立于测试阶段的。例如，一个基于近邻的协同过滤算法可能需要对一个用户的同类群体进行预计算，而一个矩阵分解系统需要确定潜在因子。被要求用来训练模型的总体时间也是一种评测方法。在大多数情况下，训练是在离线状态下进行的。因此，只要训练时间达到几小时的程度，大部分设置都是能接受的。

2) 预测时间：一旦一个模型已经训练形成，它会被用来确定对用户最合适的推荐。短的预测时间至关重要，因为它决定着用户得到响应的等待时间。

3) 存储需求：当评分矩阵非常大的时候，如何在内存中存储整个矩阵也是一项挑战。在某些情况下，设计一个最小化存储需求的算法是很有必要的。当存储要求变得非常高的时候，在大规模和实际设置中使用系统是有困难的。

由于大数据范式日益增长的重要性，近年来，可伸缩性的重要性也变得尤其重要。

2.7 习题

1 推荐系统的功能是什么？

2 讨论推荐系统的结构组成。

3 推荐系统常用于哪些领域？举例说明。

4 推荐系统常用的方法有哪些？这些方法分别适用什么场合？

5 基于内容推荐的基本思想是什么？

6 什么是冷启动问题？如何解决？

7 推荐系统的性能如何评价？

参考文献

李锐等(2013).机器学习实战.人民邮电出版社出版.

B. Liu. (2007). Web data mining: exploring hyperlinks, contents, and usage data Springer. New York.

B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. (2001). Item-based collaborative filtering recommendation algorithms. World Wide Web Confernce, 285-295.

C. Aggarwal. (2015). Data mining: the textbook. Springer, New York.

C. Aggarwal and J. Han. (2014). Frequent pattern mining. Springer, New York.

C. Aggarwal and S. Parthasarathy.(2001). Mining massively incomplete data sets by conceptual reconstruction. ACM KDD Conference, 227-232.

C. Anderson. (2006).The long tail: why the future of business is selling less of more Hyperion.

Charu C. Aggarwal. (2016). Recommender Systems: The Textbook. Springer Publishing Company, Incorporated.

D. M. Fleder and K. Hosanagar. (2007). Recommender systems and their impact on sale diversity. ACM Conference on Electronic Commerce,192-199.

G. Strang. (2009). An introduction to linear algebra. Wellesley Cambridge Press.

J. Konstan, S. McNee, C. Ziegler, R Torres, N. Kapoor, and J. Riedl. (2006). Lessons on applying automated recommender systems to information-seeking tasks. AAAI Conference, 1630-1633.

J. Reilly, K. MeCarthy, L. McGinty, and B. Smyth. (2005). Explaining compound critiques. Artifcial Intelligence Review, 24(2), 190-220.

J. Schafer, J. Konstan, and J. Riedl. (1999). Recommender systems in e-commerce. ,ACM Conference on Electronic Commerce,158.

M. Pazzani and D. Billsus. (1997). Learning and revising user profiles: The identification of interesting Web sites. Machine learning,27(3),313-331.

N. Good, J. Schafer, J Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl.(1999). Combining collaborative filtering with personal agents for better recommendations. National Conference on Artificial Intelligence (AAAI/IAA1), 439-446.

P. Cremonesi, Y. Koren, and R. Turrin. (2010). Performance of recommender algorithms on top-n recommendation tasks. RecSys, 39-46.

R. Devooght, N. Kourtellis, and A. Mantrach. (2015). Dynamic matrix factorization with on unknown values. ACM ADD Conference.

R. Kohavi, R. Longbotham. D. Sommerfield, R. Henne. (2009). Controlled experiments on the Web: survey and practical guide. Data Mining and Knowledge Discovery, 18(1).140-181.

T. Hastie, R. Tibshirani, and J. Friedman .(2009). The elements of statistical learning. Springer.

Y. Koren. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. ACM KDD Conference, 426-434.

Y. Koren and R. Bell. (2011). Advances in collaborative filtering. Recommender Systems Handbook, Springer. 145-186. (Extended version in 2015 edition of hand-book).

Y. Zhang, J. Callan, and T. Minka. (2002). Novelty and redundancy detection in adaptive filtering. ACM SIGIR Conference, 81-88.

Zhuanlan(2019). https://zhuanlan.zhihu.com/p/56997926