第 3 章　关联规则与推荐算法

关联规则和推荐算法都可用来为客户推荐商品等信息。关联规则是根据商品之间的关联性推荐，其算法复杂度较高，适合离线计算，推荐算法根据商品之间的相似性或用户之间的相似性推荐，其算法复杂度低，适合在线计算。

3.1　关联规则挖掘

关联规则(association rules)用来反映一个事务与其他事务之间的相互依存性和关联性，是数据挖掘的一项重要技术，用于从大量数据中挖掘有价值的数据项之间的相关关系。

关联规则挖掘源于购物篮分析，即在超市中用来分析顾客所购买的商品之间的关联性，这有助于决定超市商品的摆放和商品的捆绑销售策略。

3.1.1　基本概念

关联规则挖掘用来发现数据集中项集之间有趣的关联联系。如果两项或多项之间存在关联，就可以根据其中一项推荐相关联的另一项。关联规则的一般表现为蕴含式规则形式：$X \Rightarrow Y$。X 称为关联规则的前提或先导条件，Y 称为关联规则的结果或后继。定义和表示关联规则需要引入置信度(confidence)和支持度(support)两个指标。例如：

```
buys(x, "diapers")==>buys(x, "beers") [0.5%, 60%]
```

表示购买尿布的客户也会购买啤酒，该规则的支持度为 0.5%，置信度为 60%。该规则也可简写为：diapers⇒beers[0.5%，60%]。又如：

```
major(x, "CS") ^ takes(x, "DB")==>grade(x, "A") [1%, 75%]
```

表示专业为计算机选修数据库的学生成绩得 A 的支持度为 1%，置信度为 75%。该规则也可简写为：CS^DB⇒A[1%，75%]。

关联规则中的基本概念如下。

1) 项与项集

数据库中不可分割的最小信息单位(即记录)称为项(或项目)，用符号 i 表示，项的集合称为项集。设集合 $I=\{i_1,i_2,\cdots,i_k\}$ 为项集，I 中项的个数为 k，则集合 I 称为 k-项集。例如，集合{啤酒，尿布，奶粉}是一个 3-项集，而奶粉就是一个项。

2）事务

每一个事务都是一个项集。设 $I=\{i_1,i_2,\cdots,i_k\}$ 是由数据库中所有项构成的全集，则每一个事务 t_i 对应的项集都是 I 的子集。事务数据库 $T=\{t_1,t_2,\cdots,t_n\}$ 是由一系列具有唯一标识的事务组成的集合。例如，如果把超市中的所有商品看成 I，则每个顾客每张小票中的商品集合就是一个事务，很多顾客的购物小票就构成一个事务数据库。

3）项集的频数

包含某个项集的事务在事务数据库中出现的次数称为项集的频数。例如，事务数据库中有且仅有 3 个事务 $t_1=\{$啤酒，奶粉$\}$、$t_2=\{$啤酒，尿布，奶粉，面包$\}$、$t_3=\{$啤酒，尿布，奶粉$\}$，它们都包含了项集 $I_1=\{$啤酒，奶粉$\}$，则称项集 I_1 的频数为 3，项集的频数代表了支持度计数。

4）关联规则

关联规则是形如 $X\Rightarrow Y$ 的蕴含式，其中 X、Y 分别是项集 I 的真子集，并且 $X\cap Y=\varnothing$，X 称为规则的前提，Y 称为规则的结果。关联规则反映了 X 中的项目出现时，Y 中的项目也跟着出现的规律。

5）支持度

关联规则的支持度是事务集中同时包含项 X 和 Y 的事务数与事务集中总事务数的比值。它反映了 X 和 Y 中所包含的项在事务集中同时出现的概率，记为 $\text{support}(X\Rightarrow Y)$，即

$$\text{support}(X\Rightarrow Y)=\text{support}(X\cup Y)=P(XY) \tag{3-1}$$

6）置信度

关联规则的置信度是事务集中同时包含 X 和 Y 的事务数与包含 X 的事务数的比值，记为 $\text{confidence}(X\Rightarrow Y)$。置信度反映了包含 X 的事务中出现 Y 的条件概率，即

$$\text{confidence}(X\Rightarrow Y)=\frac{\text{support}(X\cup Y)}{\text{support}(X)}P(Y\mid X) \tag{3-2}$$

7）最小支持度与最小置信度

通常，支持度与置信度必须都大于（或等于）人为设置的阈值，才表明项与项之间存在关联。支持度的阈值称为最小支持度（min_sup），它描述了关联规则的最低重要程度；置信度的阈值称为最小置信度（min_conf），它反映了关联规则必须满足的最小可靠性。

8）强关联规则

如果某条关联规则 $X\Rightarrow Y$ 的支持度大于或等于最小支持度，置信度大于或等于最小置信度，则称关联规则 $X\Rightarrow Y$ 为强关联规则，否则称 $X\Rightarrow Y$ 为弱关联规则。只有强关联规则才有实际意义，因此通常所说的关联规则都指强关联规则。

9）频繁项集

如果某个项集的支持度大于或等于最小支持度，即项集$\{X,Y\}$的支持度 $\text{support}(X\Rightarrow Y)\geqslant$min_sup，则称该项集为频繁项集。求频繁项集是求强关联规则的第一步。

支持度和置信度的示意图如图 3-1 所示。其中，若 T 代表事务数据库，则 A 的支持度就是 A/T，$A\Rightarrow C$ 或 $C\Rightarrow A$ 的支持度为$(A\cap C)/T$，$A\Rightarrow C$ 的置信度为$(A\cap C)/A$。

【例 3-1】 现有事务数据库 T 如表 3-1 所示，设最小支持度为 50%，最小可信度为

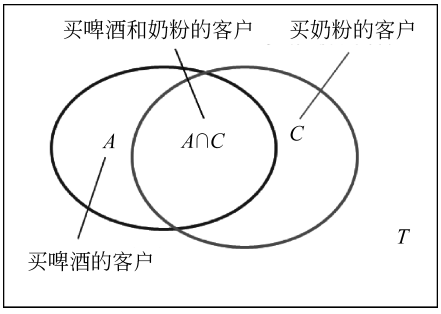

图 3-1 支持度和置信度的示意图

50%，求所有频繁项集(2 项集以上)和强关联规则。

表 3-1 事务数据库 *T*

交易 ID	1001	1002	1003	1004
购买的商品	A,B,C	A,D	A,C	B,E,F

解：事务数据库 T 中共 4 个事务，项集$\{A,C\}$在所有事务中出现了 2 次，因此$\{A,C\}$的支持度为 2/4=50%，不小于最小支持度，其余项集(2 项集以上)均只出现过 1 次，故项集$\{A,C\}$为 T 中唯一的频繁项集。

在频繁项集的基础上求强关联规则，项集$\{A,C\}$可以构成的关联规则有 2 条，即

$A\Rightarrow C$ 和 $C\Rightarrow A$

$\text{confidence}(A\Rightarrow C)=\text{support}(A\cup C)/\text{support}(A)=(2/4)/(3/4)=2/3=66\%$

$\text{confidence}(C\Rightarrow A)=\text{support}(C\cup A)/\text{support}(C)=2/2=100\%$

因为 $A\Rightarrow C$ 和 $C\Rightarrow A$ 的置信度均大于最小置信度，因此它们都是强关联规则。

说明：

① 在同一个事务数据库中，所有项集的支持度的分母都相同，例如本例为 4。因此，在求置信度时，两个项集支持度的分母可以约去，故可直接用项集的频数相除求置信度。

② 为什么用支持度和置信度就能表示关联性呢，这是因为：

假设一个超市一天有 10 000 条销售记录，如果 100 条销售记录中都同时销售了 A 商品和 C 商品，当然可以认为商品 A 和 C 之间具有某种销售关联性，这就是支持度。但是，如果另外 900 条销售记录里也销售了 A 商品，但却没出现 C 商品，这时似乎就不能认为买 A 商品的顾客一定也会买 C 商品，这就是置信度，因此商品之间的关联性与支持度和置信度都有关系。

【练习 3-1】 已知总交易笔数(事务数)为 1000，其中包含某些商品的交易数如下。

包含“牛奶”：50，包含“面包”：80，包含“鸡蛋”：20；

包含“牛奶”和“面包”：15，包含“鸡蛋”和“面包”：10，包含“牛奶”和“鸡蛋”：10，包含“牛奶”“鸡蛋”“面包”：5。

求“牛奶和面包”的支持度，“牛奶、面包和鸡蛋”的支持度；

“牛奶⇒面包”的置信度，“面包⇒牛奶”的置信度；

“牛奶和面包⇒鸡蛋”的置信度,“鸡蛋⇒牛奶和面包”的置信度。

3.1.2 Apriori 算法

关联规则挖掘可分解为两个子问题:第一步是找出事务数据库中所有大于或等于用户指定的最小支持度的数据项集,即频繁项集;第二步是利用频繁项集生成所需要的关联规则,方法是根据用户设定的最小置信度进行取舍,从而得到强关联规则。识别或发现所有频繁项集是关联规则发现算法的核心。

1993 年,Agrawal 等首先提出关联规则概念,1994 年,又提出著名的 Apriori 算法,之后该算法成为关联规则挖掘的经典算法。

1. Apriori 算法的原理和实例

Apriori 算法的基本思想是:通过对事务数据库的多次扫描计算项集的支持度,发现所有的频繁项集,从而生成关联规则。Apriori 算法对数据集进行多次扫描。第一次扫描得到频繁 1-项集的集合 L_1,第 $k(k>1)$ 次扫描首先利用第 $k-1$ 次扫描的结果 L_{k-1} 产生候选 k-项集的集合 C_k,然后在扫描的过程中确定 C_k 中元素的支持度,最后在每一次扫描结束时计算频繁 k-项集的集合 L_k,算法当候选 k-项集的集合 C_k 为空时结束。

可见,Apriori 算法是通过频繁 1-项集求频繁 2-项集,再通过频繁 2-项集生成频繁 3-项集,如此迭代。这样做的理论依据是:频繁项集的任何子集也一定是频繁的;反之,任何一个项集是非频繁的,那么它的超集也一定不是频繁项集。例如,如果$\{A,C\}$不是频繁项集,那么$\{A,B,C\}$也一定不会是频繁项集。

【例 3-2】 现有 A、B、C、D、E 5 种商品的交易记录表(见表 3-2),试找出 3 种商品的关联销售情况($k=3$),设最小支持度 min_sup≥50%,最小置信度 min_conf≥75%。

表 3-2 交易记录表

交易号	101	102	103	104
商品代码	A,C,D	B,C,E	A,B,C,E	B,E

解:因为要找出 3 种商品的关联销售情况,所以要找出所有的频繁 3-项集。那么,必须先找频繁 1-项集,再找频繁 2-项集。

(1) 首先,第 1 次扫描数据库,并计算每个 1-项集的支持度,得到候选 1-项集 C_1。在候选 1-项集中去掉支持度小于最小支持度的项集,得到频繁 1-项集,如图 3-2 所示。

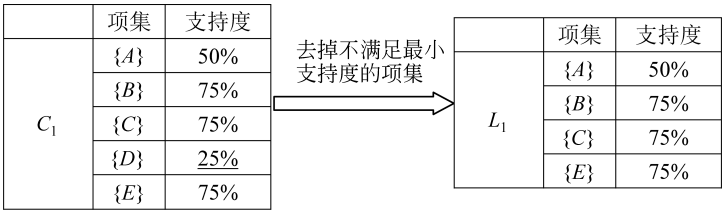

图 3-2 从候选 1-项集中找频繁 1-项集

(2) 第 2 次扫描,为了得到候选 2-项集,算法使用 $C_2=L_1 \infty L_1$,即把 L_1 中的 4 个 1 项集两两组合,得到 6 个候选 2-项集,再在候选 2-项集中去掉不满足最小支持度的项集,得到频繁 2-项集,如图 3-3 所示。

	项集	支持度
C_2	{A,B}	25%
	{A,C}	50%
	{A,E}	25%
	{B,C}	50%
	{B,E}	75%
	{C,E}	75%

去掉不满足最小支持度的项集

	项集	支持度
L_2	{A,C}	50%
	{B,C}	50%
	{B,E}	75%
	{C,E}	75%

图 3-3 从候选 2-项集找频繁 2-项集

(3) 第 3 次扫描,为了得到候选 3-项集,算法使用 $C_3=L_2 \infty L_2$,即把 L_2 中的 4 个 2 项集两两组合,产生的详细项集列表如下。

① $C_3=L_2 \infty L_2=\{\{A,B,C\},\{A,B,C,E\},\{A,C,E\},\{B,C,E\}\}$。

② 使用 Apriori 剪枝算法,因为$\{A,B,C,E\}$是 4 项集,故从 C_3 中删除。另外,频繁项集的所有子集也应该是频繁项集,若某个候选 3-项集的子集中存在非频繁项集,则应该将该候选 3-项集删除,这称为 Apriori 剪枝。在 C_3 中,候选 3-项集$\{A,C,E\}$的子集$\{A,E\}$是非频繁项集,故应将$\{A,C,E\}$删除。$\{A,B,C\}$的子集$\{A,B\}$也是非频繁项集,故应将$\{A,B,C\}$删除,最终得到 C_3 为$\{\{B,C,E\}\}$。

③ 在候选 3-项集 C_3 中去掉不满足最小支持度的项集,得到频繁 3-项集 L_3,如图 3-4 所示。

将 L_2 的项集组合	
{A,C}+{B,C}	{A,B,C}
{A,C}+{B,E}	{A,B,C,E}
{A,C}+{C,E}	{A,C,E}
{B,C}+{B,E}	{B,C,E}
{B,C}+{C,E}	{B,C,E}
{B,E}+{C,E}	{B,C,E}

Apriori剪枝

	项集	支持度
C_3	{B,C,E}	50%

检查支持度

	项集	支持度
L_3	{B,C,E}	50%

图 3-4 从频繁 2-项集得到候选 3-项集的过程

(4) 得到频繁 3-项集后,就可以找出 3 种商品的关联销售情况,方法是:找出频繁 3-项集$\{B,C,E\}$的所有真子集,并计算这些真子集的支持度。再用真子集的支持度除以 L_3 的支持度,就得到关联规则的置信度,置信度大于 min_conf 的就是强关联规则,如图 3-5 所示。

所以,最终得到的强关联规则有 2 条,即 $B \wedge C \Rightarrow E$ [50%,100%]、$C \wedge E \Rightarrow B$[50%,100%],表示购买了商品 B、C 的顾客可能会购买商品 E,购买了商品 C、E 的顾客可能会购买商品 B。

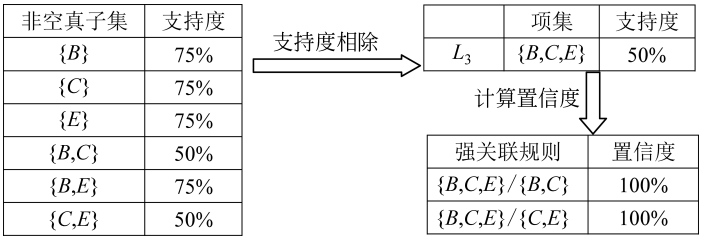

图 3-5　从频繁 3-项集中找强关联规则

2. Apriori 算法的实现

Apriori 算法的主要步骤如下。

(1) 扫描整个事务数据库,产生候选 1-项集的集合 C_1。

(2) 根据最小支持度,由候选 1-项集的集合 C_1 产生频繁 1-项集的集合 L_1。

(3) 设 k 表示 k-项集,对 $k>1$,重复置信步骤(4)～步骤(6)。

(4) 由 L_k 执行连接和剪枝操作,产生候选($k+1$)-项集的集合 C_{k+1}。

(5) 根据最小支持度,由候选($k+1$)-项集的集合 C_{k+1},产生频繁($k+1$)-项集的集合 L_{k+1}。

(6) 若 $L_{k+1}\neq\varnothing$,则 $k=k+1$,跳往步骤(4);否则转到步骤(7)。

(7) 根据最小置信度,由频繁项集产生强关联规则,算法结束。

Apriori 算法求频繁项集的伪代码描述如下。

输入:事务数据库 D,最小支持度 min_sup。

输出:D 中的频繁项集 $L(k)$。

```
L1=find_frequent_1-itemsets(D);          //找出频繁 1-项集
for(k=2;Lk-1≠∅;k++) {
    Ck=apriori_gen(Lk-1);                //产生候选 k-项集
    for each 事务 t in D {               //扫描事务数据库
        Ct=subset(Ck, t);                //得到 t 的子集
        for each candidate c in Ct
                c.count++;
    }
    //返回候选项集中不小于最小支持度的项集
    Lk={c∈Ck | c.count≥min_sup}
}
return L=所有频繁项集 Lk 的并集;
```

在该算法中,候选项集的生成是整个算法的核心,是通过 apriori_gen()函数的连接和剪枝两步生成的。apriori_gen()函数的参数为 L_{k-1},即所有频繁($k-1$)-项集的集合。它返回所有频繁 k-项集的一个超集(superset)。方法是:首先,在连接步,将 L_{k-1} 与 L_{k-1} 自连接,获得一个 k 阶候选项集 C_k,条件 $p[k-1]<q[k-1]$保证不会出现相同的扩展项

集，经过合并运算，$C_k \supseteq L_k$。apriori_gen()函数的伪代码如下。

```
Procedure apriori_gen(Lk-1)
    for each 项集 p in Lk-1
        for each 项集 q in Lk-1
        if((p[1]=q[1]) && (p[2]=q[2]) &&… && (p[k-2]=q[k-2]) && (p[k-1]<
                q[k-1])) {
            c=q 连接 p
            //若 k-1 项集中已经存在子集 c,则进行剪枝
            if has_infrequent_subset(c, Lk-1) then
                delete c;          //剪枝步,删除非频繁候选项集
            else add c to Ck
        }
    return Ck;
```

其中，在剪枝步，对于所有项集 $c \in C_k$，若它的某项$(k-1)$-项集不在 L_{k-1} 中，则将该项集 c 删除。检测是否存在非频繁项集的伪代码如下。

```
Procedure has_infrequent_subset(c, Lk-1)        //检测是否存在非频繁项集
    for each(k-1)-subset s of c
        if(s∉Lk-1) {return true;}
    return false;
```

例如，假设频繁 2-项集 $L_2=\{\{A,B\},\{A,C\},\{A,E\},\{B,C\},\{B,D\},\{B,E\}\}$，则得到候选 3-项集的连接和剪枝过程如下。

连接步：L_2 按照上面的步骤自连接得到$\{\{A,B,C\},\{A,B,E\},\{A,C,E\},\{B,C,D\},\{B,C,E\},\{B,D,E\}\}$。

剪枝步：$\{A,B,C\}$的所有 2 项子集$\{A,B\},\{A,C\},\{B,C\}$都是 L_2 中的元素，因此，保留$\{A,B,C\}$在 C_3 中。$\{B,C,E\}$的 2 项子集中的$\{C,E\}$不是 L_2 中的元素，因此，在 C_3 中删除$\{B,C,E\}$，最终剪枝后的结果是 $C_3=\{\{A,B,C\},\{A,B,E\}\}$。

3. Apriori 算法的优缺点及应用

Apriori 算法的缺点主要表现在计算性能上，其计算开销耗费在两方面：一是会产生巨大的候选集 C_k，算法采用自连接的方式产生候选集，例如，10^4 个频繁 1-项集 i 将生成 10^7个候选 2-项集，如果要找尺寸为 100 的频繁模式，如 $\{a_1,a_2,\cdots,a_{100}\}$，则必须先产生 $2^{100}\approx10^{30}$个候选集。显然，这将耗费巨大的内存空间。二是需要多次扫描事务数据库，每次产生候选集都要扫描一次数据库，如果最长的模式是 n，则需要$(n+1)$次数据库扫描。

Apriori 算法的优点有：它是一个迭代算法；数据采用水平组织方式；可采用 Apriori 优化方法；适合事务数据库的关联规则挖掘；适合稀疏数据集。

Apriori 算法广泛应用于商业，以及消费市场价格分析中，它能很快地求出各种产品

之间的价格关系，以及它们之间的影响。通过数据挖掘，市场商人可以瞄准目标客户，采用最新信息、特殊的市场推广活动或其他一些特殊的信息手段，极大地减少广告预算和增加收入。百货商场、超市和一些零售店也在进行数据挖掘，以便猜测这些年顾客的消费习惯。

Apriori 算法也应用于网络安全领域，如网络入侵检测技术中。早期中大型的计算机系统中都收集审计信息来建立跟踪档，这些审计跟踪的目的多是为了性能测试或计费，因此对攻击检测提供的有用信息比较少。Apriori 算法通过模式的学习和训练可以发现网络用户的异常行为模式，使网络入侵检测系统可以快速发现用户的行为模式，能够快速锁定攻击者，提高了基于关联规则的入侵检测系统的检测性。

3.1.3 Apriori 算法的程序实现

由于 Sklearn 框架中没有提供关联规则分析的功能，因此没有 Apriori 和 Fp-growth 算法，但是其他一些 Python 工具包中提供了 Apriori 算法，可以通过 https://pypi.org 搜索 Python 的工具包。本节选择 efficient-apriori 1.1.1，将其安装文件 efficient_apriori-1.1.1-py3-none-any.whl 下载下来，然后执行“pip install 安装文件路径和文件名”命令即完成安装。

efficient-apriori 模块提供了 apriori 类，该类的构造函数有 3 个参数，分别是数据集、最小支持度和最小置信度，输出是所有的频繁项集和关联规则。

【程序 3-1】 使用 Apriori 算法挖掘事务数据集 data 的频繁项集，并输出关联规则。

```
from efficient_apriori import apriori          #导入模块
    #设置事务数据集 data
data=[('牛奶','面包','香蕉'),
      ('可乐','面包', '香蕉', '啤酒'),
      ('牛奶','香蕉', '啤酒', '鸡蛋'),
      ('面包', '牛奶', '香蕉', '啤酒'),
      ('面包', '牛奶', '香蕉', '可乐')]
    #挖掘频繁项集和频繁规则
itemsets, rules=apriori(data, min_support=0.5, min_confidence=1)
print(itemsets)              #输出频繁项集
print(rules)                 #输出关联规则
```

该程序的输出结果如下。

```
{1: {('香蕉',): 5, ('面包',): 4, ('牛奶',): 4, ('啤酒',): 3},
2: {('牛奶', '面包'): 3, ('牛奶', '香蕉'): 4, ('面包', '香蕉'): 4, ('啤酒',
'香蕉'): 3},
3: {('牛奶', '面包', '香蕉'): 3}}
[{牛奶} ->{香蕉}, {面包} ->{香蕉}, {啤酒} ->{香蕉}, {牛奶, 面包} ->{香蕉}]
```

其中，“1:”表示频繁 1-项集，“('香蕉',): 5”表示香蕉的支持度计数为 5。

3.1.4 FP-Growth 算法

Apriori 算法由于会重复扫描数据库，并且产生巨大的候选集，导致其算法性能较差。2000 年，由韩嘉炜等提出的一种不产生候选项集的算法，称为 FP-Growth（Frequent Pattern Growth，频繁模式树增长）算法，它采用分而治之的思想，将数据库中的频繁项集压缩到一棵频繁模式树中，同时保持项集之间的关联关系。然后，将这些压缩后的频繁模式树分成一些条件子树，每个条件子树对应一个频繁项，从而获得频繁项集，最后挖掘出关联规则。该算法总共需对数据库进行两次扫描，因此能显著加快发现频繁项集的速度。

FP-Growth 算法的主要任务是将数据集存储在 FP-Tree（频繁模式树）中，通过 FP-Tree 可以高效地发现频繁项集，执行速度通常比 Apriori 算法快两个数量级。FP-Growth 算法只给出了高效地发现频繁项集的方法，但不能用于发现关联规则。

1. FP-Growth 算法的原理及实例

FP-Growth 算法的基本思路如下。

（1）遍历一次数据库，找出频繁 1-项集，按递减顺序排序。

（2）建立 FP-Tree。

（3）利用 FP-Tree 为频繁 1-项集的每一项构造条件 FP-Tree。

（4）得到频繁项集。

【例 3-3】 表 3-3 是一个事务数据库，试利用 FP-Growth 算法找出所有含 2 项以上的频繁项集（设最小支持度计数为 2）。

表 3-3 事务数据库

交 易 号	商 品 代 码	交 易 号	商 品 代 码
1	A,B,E	6	B,C
2	B,D	7	A,C
3	B,C	8	A,B,C,E
4	A,B,D	9	A,B,C
5	A,C		

解：（1）扫描事务数据库得到频繁 1-项集，如表 3-4 所示，这是第 1 次扫描数据库。

表 3-4 频繁 1-项集

A	B	C	D	E
6	7	6	2	2

（2）对频繁 1-项集按项集的频数从大到小排序，得到排序后的频繁 1-项集，如表 3-5 所示。

表 3-5　排序后的频繁 1-项集

B	*A*	*C*	*D*	*E*
7	6	6	2	2

(3) 按频繁 1-项集支持度递减的顺序重新排序事务数据库中的项，如表 3-6 所示。

表 3-6　按支持度计数递减排序的事务数据库

交　易　号	商 品 代 码	交　易　号	商 品 代 码
1	*B*,*A*,*E*	6	*B*,*C*
2	*B*,*D*	7	*A*,*C*
3	*B*,*C*	8	*B*,*A*,*C*,*E*
4	*B*,*A*,*D*	9	*B*,*A*,*C*
5	*A*,*C*		

(4) 创建 FP-Tree 的根结点和频繁项目表，FP-Tree 的根结点总是 Null。

(5) 向 FP-Tree 中加入每个事务，这是第 2 次扫描数据库。例如，经排序后的第 1 个事务是{*B*,*A*,*E*}，则按照该排序顺序将 *B*、*A*、*E* 依次添加到 FP-Tree 的一个分支中，并将计数值设为 1，如图 3-6 所示。为了方便遍历，FP-Growth 算法还需要一个称为结点头(Node-head)指针表的数据结构，这是一个用来记录各个元素项的总出现次数的数组，再附带一个指针指向 FP-Tree 中该元素项的第一个结点，这样每个元素项都构成一条单链表。

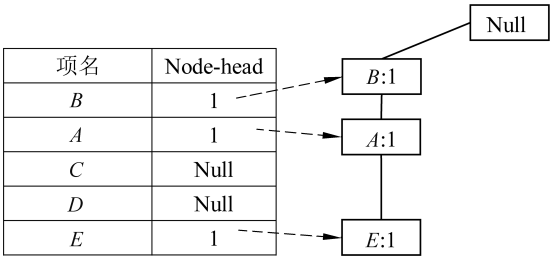

图 3-6　向 FP-Tree 中加入第 1 个事务

(6) 然后依次加入第 2 个事务(见 3-7)和第 3 个事务(见图 3-8)，如果 FP-Tree 中已经有该事务，则将该事务的计数加 1。

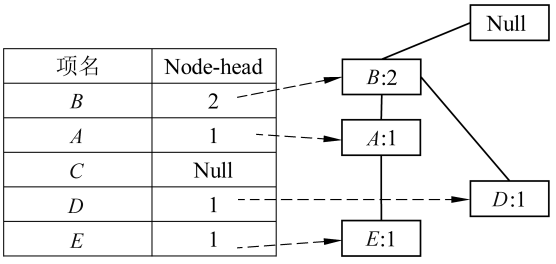

图 3-7　加入第 2 个事务

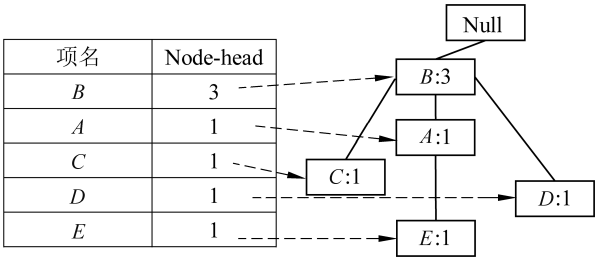

图 3-8 加入第 3 个事务

(7) 按照上述方法加入剩下的第 4～9 个事务。最终生成的 FP-Tree 如图 3-9 所示。

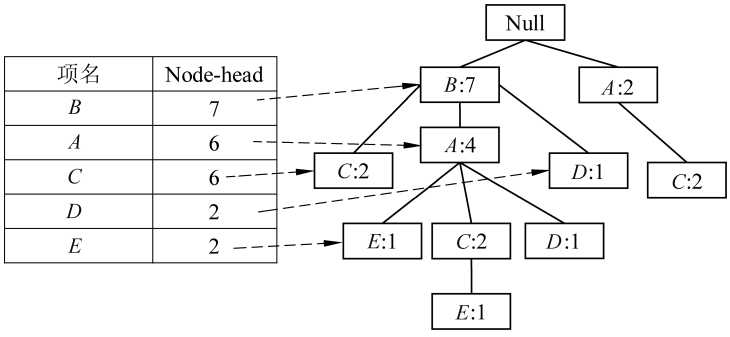

图 3-9 最终生成的 FP-Tree

在 FP-Tree 建立好之后,只要寻找结点的条件模式基(conditional pattern base),就能快速得到频繁项集。条件模式基是以所查找元素项为结尾的路径集合。每一条路径其实都是一条前缀路径。

例如,要找包含 E 的频繁项集,方法是:从元素 E 向上找它的前缀路径,图 3-9 中有 2 个结点 E,因此 E 的前缀路径有 2 条。条件模式基的计数为 E 的计数值(本例均为 1)。得到 E 的条件模式基为

```
<(B,A):1>,<(B,A,C):1>
```

将条件模式基中结点的出现次数合并,得到包含 E 的频繁项集为

```
{{B,E:2},{A,E:2},{A,B,E:2}}
```

同理,对于元素 C,得到的条件模式基为

```
<(B,A):2>,<B:2>,<A:2>
```

得到 C 的频繁项集为

```
{{B,C:4},{A,C:4},{A,B,C:2}}
```

其他结点的频繁项集也是按上述步骤生成的，从而得到所有频繁项集，再根据频繁项集生成关联规则即可完成关联规则的挖掘。

2. FP-Growth 算法的程序实现

FP-Growth 算法的程序实现步骤如下。

(1) 建立头指针：遍历数据集，找出所有的频繁一项集，构成头指针，并根据支持度对一项集排序。

(2) 建立 FP-Tree：定义根结点，遍历数据集，对于每条记录，根据头指针的顺序向树中添加结点。如果记录中上一个结点的子结点中当前结点已存在，则结点支持度＋记录支持度；如果结点不存在，则在上一结点中添加当前子结点，并设置支持度为记录支持度。

(3) 查找条件模式基：根据头指针查找每个一项集的前缀路径，作为条件模式基，且当前一项集作为频繁项基。

(4) 查找频繁项：深度遍历，重复步骤(3)。每次查找完成后，将每一层遍历的频繁项集＋新的头指针中的频繁一项集作为频繁项，重复此步骤，直到 FP-Tree 的头指针为空。

下面给出 FP-Growth 算法的描述。

输入：事务集 D、最小支持度。

输出：FP-Tree、头指针表。

算法步骤如下。

(1) 遍历事务集 D，统计各元素项出现的次数，创建头指针表。

(2) 移除头指针表中不满足最小支持度的元素项。

(3) 第二次遍历数据集，创建 FP-Tree。对每个事务集中的项集。

① 初始化空 FP-Tree；

② 对每个项集进行过滤和重排序；

③ 使用这个项集更新 FP-Tree，从 FP-Tree 的根结点开始：

- 如果当前项集的第一个元素项存在于 FP-Tree 当前结点的子结点中，则更新这个子结点的计数值。
- 否则，创建新的子结点，更新头指针表。
- 对当前项集的其余元素项和当前元素项的对应子结点递归③的过程。

3. FP-Growth 算法的特点

FP-Growth 算法的优点包括不生成候选集，不用候选测试；使用紧缩的数据结构；避免重复数据库扫描；基本操作是计数和建立 FP-Tree。缺点是：实现比较困难，在某些数据集上性能会下降。

3.2 推荐系统及算法

推荐系统在互联网领域有非常广泛的应用，推荐可以满足用户非明确的潜在需求，而搜索用来满足用户主动表达的需求。可见，推荐是搜索功能的重要补充。据统计，在电子

商务网站中，有35%的销售来源推荐。在视频播放网站中，有75%的观看来自推荐。在交友网站中，推荐系统常根据好友的爱好，向用户推荐他可能感兴趣的人。在移动App中，经常根据用户所处的地理位置推荐附近的景点、美食和住宿等信息。可见，推荐在互联网领域无处不在，这是因为互联网上信息过载，用户常被湮没在信息中，而对自己的实际需求不明确；其次，推荐很强地依赖于用户行为，互联网上很难获取用户的真实信息和喜好，只能通过捕获用户的行为获取，如记录用户的浏览历史，点击、收藏、搜索等行为，根据这些行为衡量用户的兴趣。

推荐系统是通过用户与产品之间的二元关系，利用已有的选择过程或相似性关系挖掘每个用户潜在的感兴趣对象，进而进行个性化推荐，其本质是信息过滤。一个完整的推荐系统由3个模块组成：收集用户信息的行为记录模块、分析用户喜好的模型分析模块和推荐算法模块。其中，协同过滤推荐算法是推荐系统最常用的算法，它能分析用户的喜好，并根据推荐算法进行推荐。

3.2.1 协同过滤推荐算法

协同过滤(collaborative filtering，CF)推荐算法是推荐系统中主流的推荐算法。它包括协同和过滤两个操作。所谓协同，就是利用群体的行为做决策(推荐)，而过滤是从可行的决策(推荐)方案(标的物)中将用户喜欢的方案找(过滤)出来。

1. 两种协同过滤推荐算法

协同过滤推荐分为基于用户的协同过滤方法(user-based CF)和基于物品的协同过滤方法(item-based CF)。

基于用户的协同过滤基本假设为：为了给用户推荐感兴趣的内容，可通过找到与该用户偏好相似的其他用户，并将他们感兴趣的内容推荐给该用户。举例来说，如果A、B两个用户都购买了x、y、z三本图书，并且给出了5星好评。那么，A和B就属于相似的用户。可以将A看过的图书w推荐给用户B。

基于物品的协同过滤基本假设为：如果一个用户对某个物品感兴趣，则将与该物品相似的其他物品推荐给该用户。物品与物品之间的相似性根据物品是否被许多用户同时购买来评判，而不会考虑物品本身的属性。例如，有很多购买iPhone手机的用户也同时购买了iPad，则说明iPhone和iPad这两种物品具有相似性，可向购买了iPhone的用户推荐iPad。

2. 基于物品的协同过滤推荐

该算法通过用户对不同物品的评分评测物品之间的相似性，基于物品之间的相似性做出推荐。这里不是利用物品自身属性计算物品之间的相似度，而是通过分析用户的行为记录计算物品之间的相似度。具体而言，通过计算不同用户对不同物品的评分获得物品间的关系，基于物品间的关系对用户进行相似物品的推荐，这里的评分代表用户对商品的态度和偏好。简单来说，如图3-10所示，用户User 1和User 2都购买了Product 1和

Product 3，并给出了 5 星好评，说明商品 Product 1 和 Product 3 比较相似，那么，当用户 User 3 也购买了商品 Product 3 时，可以推断他也有购买 Product 1 的潜在需求，因此可向他推荐 Product 1。

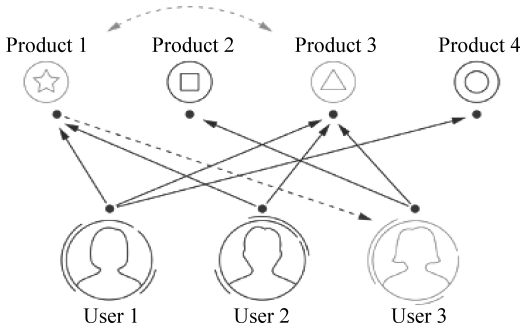

图 3-10　基于物品的协同过滤推荐算法示意图

基于物品的协同过滤算法的实现步骤如下。

(1) 计数物品之间的相似度。在协同过滤算法中，相似度是采用余弦相似度衡量的，余弦相似度表征了两个向量之间夹角的相似度，即如果两个向量的方向相似，它们的余弦相似度值就较大(接近于 1)。

采用余弦相似度，是因为两个用户购买或评价的商品种类可能各不相同，如果采用距离的方法度量，则距离的某些维度将没有值，距离计算将无法进行。其次，每个用户的评分宽严程度不同，有些用户的评分可能总体偏低，此时如果计算距离将差距较大，而计算向量的方向(余弦相似度)则差距很小。

余弦相似度的计算方法如下。

假设两个对象 v_i 和 v_j 对应的向量分别为 $\boldsymbol{X}=(x_{i1},x_{i2},\cdots,x_{in})$和$\boldsymbol{Y}=(x_{j1},x_{j2},\cdots,x_{jn})$，则余弦相似度 $\mathrm{sim}(v_i,v_j)$的计算公式为

$$\mathrm{sim}(v_i,v_j)=\frac{\boldsymbol{X}}{\|\boldsymbol{X}\|}\cdot\frac{\boldsymbol{Y}}{\|\boldsymbol{Y}\|}=\frac{\sum_{k=1}^{n}x_{ik}\times x_{jk}}{\sqrt{\sum_{k=1}^{n}x_{ik}^2}\times\sqrt{\sum_{k=1}^{n}x_{jk}^2}} \tag{3-3}$$

设 x_{ik} 和 x_{jk} 的取值只能是 0 或 1，则式(3-3)可转换成如下形式：

$$w_{ij}=\frac{|N(i)\cap N(j)|}{\sqrt{|N(i)|\cdot|N(j)|}} \tag{3-4}$$

例如，对于物品 a、b、c、d 和用户 A、B、C、D，设 $N(a)=\{A,B\}$表示对物品 a 感兴趣的用户有 A 和 B，$N(b)=\{A,C,D\}$表示对物品 b 感兴趣的用户有 A、C 和 D，各用户对各物品的感兴趣程度均为 1，则物品 a、b 之间的相似度为

$$w_{ab}=\frac{|N(a)\cap N(b)|}{\sqrt{|N(a)|\cdot|N(b)|}}=\frac{|\{A,B\}\cap\{A,C,D\}|}{\sqrt{2\times3}}=\frac{1}{\sqrt{6}}$$

提示：只有当感兴趣程度只能为 1 或 0 时，才能使用简化式(3-4)计算余弦相似度，否则必须使用完整式(3-3)计算。

然后根据 w_{ij} 的大小选出与物品 i 最相似的 K 个物品(K 的大小视情况而定),求这 K 个物品的集合。

(2) 根据物品的相似度和用户的历史行为给用户生成推荐列表。计算用户 u 对物品 j 的感兴趣程度 p_{uj} 的公式如下:

$$p_{uj}=\sum_{i\in S_{jK}\cap N(u)} w_{ij}r_{uj} \tag{3-5}$$

其中,$N(u)$表示用户 u 曾经有过正反馈的物品集合;S_{jK} 表示与物品 j 最相似的 K 个物品的集合;r_{uj} 表示用户 u 对物品 j 的感兴趣程度(用正整数表示)。通过设定阈值决定是否推荐物品,从而生成推荐列表。

【例 3-4】 对于物品 a、b、c、d、e 和用户 A、B、C、D,设 $N(a)=\{A,B\}$,$N(b)=\{A,C\}$,$N(c)=\{D,B\}$,$N(d)=\{A,D\}$,$N(e)=\{C,D\}$。各用户对各物品的感兴趣程度均为1,推荐阈值为 0.9。试使用基于物品的协同过滤算法给用户 A 推荐物品。

解:根据式(3-4)计算物品之间的相似度,有

$$w_{ab}=1/2,\quad w_{ac}=1/2,\quad w_{ad}=1/2,\quad w_{ae}=0/2=0$$

取 $K=3$,而用户 A 对物品 a、b、d 感兴趣($K=3$),剩余可推荐物品只有 c 和 e。先看 c 和 a、b、d 的相似度,$w_{ac}=w_{cd}=1/2$,$w_{bc}=0$,则 $p_{Ac}=1/2\times1+0\times1+1/2\times1=1$。

又因为 $w_{be}=w_{de}=1/2$,$w_{ae}=0$,所以 $p_{Ae}=1/2\times1+0\times1+1/2\times1=1$。

由于阈值为 0.9,因此物品 c 和 e 均可推荐给用户 A。

3. 基于用户的协同过滤推荐

该算法通过不同用户对物品的评分评测用户之间的相似性,然后基于用户之间的相似性做出推荐。具体而言,基于用户的协同过滤算法是通过用户的历史行为数据,发现用户对物品的喜好(如商品购买、收藏、内容评论或分享),并对这些喜好进行度量和打分。根据不同用户对相同商品或内容的态度和偏好程度计算用户之间的关系,在有相同喜好的用户间进行商品推荐。简单地说,如图 3-11 所示,用户 User 1 和 User 3 都购买了 Product 2 和 Product 3,并给出了 5 星好评,那么,User 1 和 User 3 就属于同一类用户,可以将 User 1 买过的物品 Product 1 和 Product 4 推荐给 User 3。

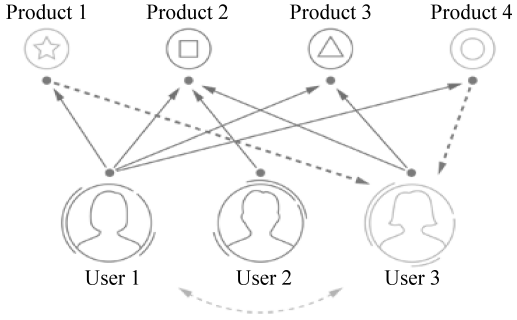

图 3-11 基于用户的协同过滤推荐算法示意图

【例 3-5】 对于用户 A、B、C、D 和物品 a、b、c、d、e,设 $N(A)=\{a,b,d\}$,$N(B)=$

$\{a,c\}$，$N(C)=\{b,e\}$，$N(D)=\{c,d,e\}$。各用户对各物品的感兴趣程度均为 1，推荐阈值为 0.7。试使用基于用户的协同过滤推荐算法给用户 A 推荐物品。

解：根据式(3-4)计算用户之间的相似度，有

$$w_{AB}=\frac{1}{\sqrt{6}},\quad w_{AC}=\frac{1}{\sqrt{6}},\quad w_{AD}=\frac{1}{3}$$

取 $K=3$，因为用户 A 对物品 a、b、d 感兴趣($K=3$)，所以剩余可推荐物品只有 c 和 e。

因为用户 B 和用户 D 对商品 c 感兴趣，而用户 A 和用户 B、用户 D 之间有相似性，故用户 A 对物品 c 的感兴趣程度为

$$p_{Ac}=w_{AB}\cdot r_{Ac}+w_{AD}\cdot r_{Ac}=\frac{1}{\sqrt{6}}\times\frac{1}{3}\times 1\approx 0.742$$

因为用户 C 和用户 D 对商品 c 感兴趣，而用户 A 和用户 C、用户 D 之间有相似性，故用户 A 对物品 e 的感兴趣程度为

$$p_{Ae}=w_{AC}\cdot r_{Ae}+w_{AD}\cdot r_{Ae}=\frac{1}{\sqrt{6}}\times\frac{1}{3}\times 1\approx 0.742$$

由于阈值为 0.7，因此物品 c 和 e 均可推荐给用户 A。

3.2.2 协同过滤推荐算法应用实例

在例 3-3 和例 3-4 中，都是假设用户对商品的感兴趣程度是 1 或 0，而实际上，感兴趣程度是通过用户的行为评估的。通常对用户的各种行为赋予不同的权重值，然后根据权重值判断用户的感兴趣程度。

1. 基于物品的协同过滤推荐算法实例

【例 3-6】 假设在电子商务网站中，用户的行为有以下 4 种。

(1) 点击，用户点击了某个商品页面，设权重值为 1 分。

(2) 搜索，用户在搜索栏搜索某种商品，设权重值为 3 分。

(3) 收藏，用户收藏了某个商品，设权重值为 5 分。

(4) 付款，设权重值为 10 分。

现有如下用户、商品和行为：

用户：A、B、C；

商品：1、2、3、4、5、6；

行为：点击(1 分)、搜索(3 分)、收藏(5 分)、付款(10 分)。

网站记录的用户行为列表如图 3-12(a)所示。

基于物品的协同过滤推荐算法的执行步骤如下。

(1) 根据用户行为列表计算用户、物品的评分矩阵，如图 3-12(b)所示。

(2) 将用户、物品的评分矩阵中的用户行为转换成权重值，如图 3-13(a)所示。显然，评分矩阵中的每个权重值，就代表了用户对物品的喜好程度。

用户	物品	行为
A	1	点击
C	3	收藏
B	2	搜索
B	5	搜索
B	6	收藏
A	2	付款
C	3	付款
C	4	收藏
C	1	收藏
A	1	点击
A	6	收藏
A	4	搜索

(a) 用户行为列表

	A	*B*	*C*
1	点击×2	无	收藏
2	付款	搜索	无
3	无	无	收藏+付款
4	搜索	无	收藏
5	无	搜索	无
6	收藏	收藏	无

(b) 用户、物品的评分矩阵

图 3-12 用户行为列表与用户、物品的评分矩阵

(3) 根据用户、物品的评分矩阵计算物品与物品的相似度矩阵。例如，使用余弦相似度公式计算物品 1 和物品 2 之间的相似度，如图 3-13(b)所示。

	A	*B*	*C*
1	2	0	5
2	10	3	0
3	0	0	15
4	3	0	5
5	0	3	0
6	5	5	0

物品 1: 2 0 5

物品 2: 10 3 0

$$\cos(1,2)=\frac{2\times10+0\times3+5\times0}{\sqrt{2^2+0^2+5^2}\times\sqrt{10^2+3^2+0^2}}=0.36$$

图 3-13 根据用户、物品的评分矩阵计算物品与物品的相似度

按照该方法，对所有物品计算两两之间的相似度值，得到如图 3-14 所示的物品与物品之间的相似度矩阵。显然，相似度矩阵中的相似度值，就代表了物品与物品之间的相似度。

	1	2	3	4	5	6
1	1	0.36	0.93	0.99	0	0.26
2	0.36	1	0	0.49	0.29	0.88
3	0.93	0	1	0.86	0	0
4	0.99	0.49	0.86	1	0	0.36
5	0	0.29	0	0	1	0.71
6	0.26	0.88	0	0.36	0.71	1

图 3-14 物品与物品之间的相似度矩阵

(4) 相似度矩阵(相似程度)×评分矩阵(喜好程度)＝推荐列表，如图 3-15 所示。

得到的推荐列表如图 3-16 所示。

	1	2	3	4	5	6
1	1	0.36	0.93	0.99	0	0.26
2	0.36	1	0	0.49	0.29	0.88
3	0.93	0	1	0.86	0	0
4	0.99	0.49	0.86	1	0	0.36
5	0	0.29	0	0	1	0.71
6	0.26	0.88	0	0.36	0.71	1

×

	A	*B*	*C*
1	2	0	5
2	10	3	0
3	0	0	15
4	3	0	5
5	0	3	0
6	5	5	0

图 3-15　相似度矩阵×评分矩阵

(5) 根据推荐列表,将推荐值最高的若干种物品推荐给用户,如图 3-17 所示。当然,也可先将用户已经购买过的物品推荐值置为 0。

	A	*B*	*C*
1	9.9	2.4	23.9
2	16.6	8.3	4.3
3	4.4	0	24
4	11.7	3.3	22.9
5	6.5	7.4	0
6	15.4	9.8	3.1

图 3-16　得到的推荐列表

	A	*B*	*C*
1	9.9	2.4	23.9
2	0	8.3	4.3
3	4.4	0	0
4	11.7	3.3	22.9
5	6.5	7.4	0
6	15.4	9.8	3.1

图 3-17　推荐权重值最高的物品

例如,对于用户 A 来说,推荐值最高的是物品 6,因此可将物品 6 推荐给 A。

总结:基于物品的协同过滤推荐算法步骤如下。

(1) 根据用户行为列表计算用户、物品的评分矩阵。

(2) 根据用户、物品的评分矩阵计算物品、物品的相似度矩阵。

(3) 物品、物品相似度矩阵×用户、物品评分矩阵=推荐列表。

(4) 将推荐列表中用户之前已经有过购买行为的元素推荐值置为 0。

基于物品的协同过滤推荐算法的优缺点如下。

优点:两个物品之间的距离可能是根据成百上千万用户的评分计算得出的,因此这个评分往往能在一段时间内保持稳定。因此,这种算法可以预先计算距离,其在线部分能更快地生成推荐列表。

缺点:不同领域的最热门物品之间经常具有较高的相似度。这样,可能会给喜欢《算法导论》的读者推荐《哈利波特》。为此,在运行这种算法时可以不纳入最畅销商品。

2. 基于用户的协同过滤推荐算法实例

基于用户的协同过滤推荐算法的基本假设为:和我兴趣相似的人喜欢的商品,我也会喜欢。对于例 3-6,这种推荐算法的主要步骤如下。

(1) 根据用户对各种物品的偏好值的相似程度,在每两个用户之间进行相似度计算,

为每个用户找到与之相似度最高的几个邻居用户，这一步是对用户进行分类。

(2) 将目标用户的邻居对每个物品的偏好值的加权平均作为目标用户偏好值的预测值。把预测值最高的若干商品作为目标用户的推荐列表。

其中，每个邻居用户的权重取决于该邻居用户与目标用户之间的相似度。

算法具体步骤如下。

(1) 根据用户行为列表计算物品、用户的评分矩阵。

(2) 根据用户、物品的评分矩阵计算用户、用户的相似度矩阵。

(3) 用户与用户相似度矩阵 × 评分矩阵＝推荐列表。

(4) 将推荐列表中用户之前已经有过购买行为的元素推荐值置为0。

基于用户的协同过滤推荐算法的缺点是：①形成有意义的邻居集合很难，很多用户两两之间只有很少几个共同评分，而仅有的共同打了分的物品，往往是最热门的商品。②用户之间的距离可能变化得很快，这让离线算法难以瞬间更新推荐结果。

3. 在协同过滤算法中考虑时间和地域的因素

在协同过滤推荐算法中，还应考虑时间和地域的因素。这是因为用户对商品的喜好具有时效性。为此，在基于物品的协同过滤中：①物品之间的相似度可以改为，同一用户在间隔很短的时间内喜欢的两件商品之间，可以给予更高的相似度。②根据当前用户的偏好，推荐相似的物品给他，可以改为，在描述目标用户偏好时，给其最近喜欢的物品赋予较高权重。在基于用户的协同过滤中，计算相似度和描述用户行为时，都给最新的偏好赋予较高权重。

在协同过滤中要考虑到地域因素，因为不同地域的用户对商品的偏好往往是有区别的。为此，在基于物品的协同过滤中，物品之间的相似度可以改为，同一用户在同一地域内喜欢的两件商品之间，可以给予更高的相似度。在基于用户的协同过滤中，把类似地域用户的行为作为推荐的主要依据。

4. 协同过滤推荐算法的特点

协同过滤推荐算法有下列优点。

(1) 不需要根据内容计算物品之间的相似性，使得某些物品(如艺术品、音乐、视频)即使机器无法对其内容进行分析，也能使用协同过滤推荐算法。

(2) 能够基于一些复杂的、难以表达的概念(信息质量、品位)进行过滤。

(3) 推荐结果具有新颖性。

协同过滤推荐算法有下列缺点。

(1) 系统刚使用时，用户对商品的评价非常少，这样，基于用户的评价所得到的用户间(或物品间)的相似性可能不准确(即冷启动问题)。

(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)。

3.3 电影节目推荐实例

目前，电影节目在智能电视和视频网站中日益丰富，使观众迅速从节目匮乏时代进入内容过剩时代。用户在面对繁多的节目时，往往难以找到感兴趣的电影节目，这不仅影响了收视用户的收看感受，也在某种程度上影响到电影节目的收视率。为了给用户推荐个性化的电影节目，本实例采用 MovieLens 数据集作为样本数据，使用两种协同过滤推荐算法向用户推荐相似电影，并比较两种算法的推荐结果。

MovieLens 数据集包含许多用户对很多部电影的评分数据，也包括电影元数据信息和用户属性信息。这个数据集经常用来做推荐系统、机器学习算法的测试数据集。根据这些电影评分数据，就可计算出电影的相似度或用户的相似度，然后根据相似度推荐相似电影给用户。该数据集的下载地址为 http://files.grouplens.org/datasets/movielens/，它有多种版本，对应不同数据量，本例所用的数据为 1MB 的数据集(u.data)。该数据集包含来自 943 个用户以及 1682 部电影的总计 10 万条电影评分记录。

文件里的内容包含了每个用户对每部电影的评分。数据格式如下。

```
userId(用户 id), movieId(电影的 id), rating(用户评分,是 5 星制,按半颗星的规模递增), timestamp(时间戳)
```

例如：{196　242　3　881250949}就是一条评分记录。

【程序 3-2】 使用两种协同过滤推荐算法向用户推荐相似电影，并评估两种算法的推荐结果。

```
import numpy as np
import pandas as pd          #用 pandas 库读取 u.data 文件
#数据文件格式: 用户 id、商品 id、用户评分、时间戳
header=['user_id', 'item_id', 'rating', 'timestamp']
df=pd.read_csv('u.data', sep='\t', names=header)      #读取 u.data 文件
#计算唯一用户和电影的数量
n_users=df.user_id.unique().shape[0]
n_items=df.item_id.unique().shape[0]
print('Number of users='+str(n_users)+' | Number of movies='+str(n_items))
from sklearn.model_selection import train_test_split
train_data, test_data=train_test_split(df, test_size=0.2, random_state=21)
#协同过滤推荐算法
#第一步是创建 user-item 矩阵,这需创建训练和测试两个 user-item 矩阵
train_data_matrix=np.zeros((n_users, n_items))
for line in train_data.itertuples():
    train_data_matrix[line[1]-1, line[2]-1]=line[3]
test_data_matrix=np.zeros((n_users, n_items))
for line in test_data.itertuples():
```