第 3 章

关联规则挖掘

假如你是一个超市促销员，正在和一位购买了可乐和面包的顾客交谈。此时你会向他推荐一些什么东西呢？你可能会凭直觉推荐一些他有可能购买的东西。那么，这些东西是否真的是他需要的呢？频繁模式和关联规则在很大程度上解决了这类问题。超市促销员根据超市的历史销售记录，查找出频繁出现的可乐和面包的组合销售记录。这些组合就形成了频繁购物模式，利用这些频繁出现的购物模式信息，促销员就可以有效地针对顾客的需求给出一些合理的推荐。

频繁模式和关联规则反映了一个事物与其他事物同时出现的相互依存性和关联性，常用于实体商店或在线电商的推荐系统。通过对顾客的历史购买记录数据进行关联规则挖掘，发现顾客群体购买习惯的内在共性，根据挖掘结果，可以调整货架的布局陈列，设计促销组合方案，从而实现销量的提升。

本章介绍频繁模式和关联规则的基本概念，讲解关联规则挖掘的核心挖掘算法——Apriori 算法和 FP-Growth 算法。

3.1 基 本 概 念

设 $I=\{i_1,i_2,\cdots,i_m\}$ 是项(item)的集合，D 是事务(transaction)的集合(事务数据库)，事务 T 是项的集合，并且 $T\subseteq I$。每一个事务具有唯一的标识，称为事务号，记作 TID。设 A 是 I 中的一个项集，如果 $A\subseteq T$，那么事务 T 包含 A。

例如，一个商店所售商品的集合 $I=$｛可乐，薯片，面包，牛奶，尿布，啤酒｝。假设商店某段时间的事务数据库 D 如表 3.1 所示，该数据库有 5 个事务，$D=$｛｛可乐，薯片｝，｛可乐，面包｝，｛可乐，面包，牛奶｝，｛尿布，啤酒｝，｛可乐，面包，啤酒｝｝。其中，事务｛可乐，面包，牛奶｝包含了事务｛可乐，面包｝。

表 3.1 某商店事务数据库

事务号(TID)	购买商品列表
100	可乐，薯片
200	可乐，面包
300	可乐，面包，牛奶
400	尿布，啤酒
500	可乐，面包，啤酒

定义 1：关联规则。

关联规则是形如 $A \rightarrow B$ 的逻辑蕴含式，其中 $A \neq \varnothing$，$B \neq \varnothing$，且 $A \subset I$，$B \subset I$，并且 $A \cap B = \varnothing$。

定义 2：关联规则的支持度。

规则 $A \rightarrow B$ 具有支持度 S，表示 D 中事务包含 $A \cup B$ 的百分比，它等于概率 $P(A \cup B)$，也叫相对支持度。

$$S(A \rightarrow B) = P(A \cup B) = \frac{|A \cup B|}{|D|}$$

另外，还有绝对支持度，又叫支持度计数、频度或计数，是事务在事务数据库中出现的次数，表示为 $|A \cup B|$。

例如，对于表 3.1 所示的商店事务数据库，顾客购买可乐和薯片有 1 笔，顾客购买可乐和面包有 3 笔，那么可乐和薯片的关联规则的支持度 S(可乐→薯片)$=\frac{1}{5}=20\%$，可乐和面包的关联规则的支持度 S(可乐→面包)$=\frac{3}{5}=60\%$。

定义 3：关联规则的置信度。

规则 $A \rightarrow B$ 在事务数据库中具有置信度 C，它表示包含项集 A 的同时也包含项集 B 的概率，即条件概率 $P(B|A)$。因为事务数据库 D 的规模是一定的。所以

$$C(A \rightarrow B) = \frac{S(A \cup B)}{S(A)} = \frac{|A \cup B|}{|A|}$$

其中 $|A|$ 表示事务数据库中包含项集 A 的事务个数。

例如，对于表 3.1 所示的商店事务数据库，顾客购买可乐有 4 笔，顾客购买可乐和薯片有 1 笔，顾客购买可乐和面包有 3 笔，那么顾客购买可乐和薯片的置信度 C(可乐→薯片)$=\frac{1}{4}=25\%$，购买可乐和面包的置信度 C(可乐→面包)$=\frac{3}{4}=75\%$。这说明买可乐和买面包的关联性比买可乐和买薯片的关联性强，在营销上可以作为组合策略销售。

定义 4：阈值。

为了在事务数据库中找出有用的关联规则，需要由用户确定两个阈值：最小支持度阈值(min_sup)和最小置信度阈值(min_conf)。

定义 5：强关联规则。

同时满足最小支持度阈值(min_sup)和最小置信度阈值(min_conf)的规则称为强关联规则，即，当 $S(A \rightarrow B) >$ min_sup 且 $C(A \rightarrow B) >$ min_conf 成立时，规则 $A \rightarrow B$ 称为强关联规则。

规则的支持度和置信度是规则价值的两种度量。它们分别反映了规则的有用性和确定性。支持度很低的规则只是偶然出现。支持度通常用于排除那些无意义的规则。置信度体现了规则推理的可靠性，对于给定的规则，置信度越高，其发生的概率越大。在典型情况下，如果关联规则满足最小支持度阈值和最小置信度阈值，则通常认为它是有价值的。

例如，假设表 3.1 的事务最小支持度阈值 min_sup 为 50%，最小置信度阈值 min_

conf 也为 50%。容易看出：关联规则(可乐→面包)的支持度 S(可乐→面包)=60%大于最小支持度阈值；置信度 C(可乐→面包)=75%，大于最小可信度阈值。所以，(可乐→面包)是强关联规则。而关联规则(可乐→薯片)不是强关联规则。所以关联规则(可乐→面包)是有价值的规则。

定义 6：频繁模式。

频繁模式是频繁地出现在数据集中的模式(如项集、子序列或子结构)。

项的集合称为项集(itemset)，包含 k 个项的集合称为 k-项集。项集的出现频度是包含项集的事务数，简称为项集的频度、支持度计数或计数。如果项集频繁地出现在交易数据集中，同时其支持度大于或等于最小支持度阈值，则称为频繁项集。

例如，在表 3.1 的事务数据库中，项集{可乐，面包}的支持度为 60%，大于规定的最小支持度阈值，所以{可乐，面包}是频繁 2 项集。项集{可乐，薯片}的支持度为 20%，小于规定的最小支持度阈值，所以{可乐，薯片}不是频繁 2 项集。

一个子序列，例如首先购买 PC，然后购买数码相机，最后购买内存卡，如果它频繁地出现在事务数据库中，则称它为频繁子序列。

一个子结构可能涉及不同的结构形式，如子图、子树或子格，它可能与项集或子序列结合在一起。如果一个子结构频繁地出现在事务数据库中，则称它为频繁子结构。

频繁项集模式挖掘的一个典型例子是购物篮分析。购物篮分析通过发现顾客放入“购物篮”中的商品之间的关联，分析顾客的购物习惯。这种关联的发现可以帮助零售商了解哪些商品频繁地被顾客同时购买，从而帮助零售商制订更好的营销策略。例如，可以将顾客经常同时购买的商品摆放在一起，以便促进这些商品的销售。换一个角度，也可以把顾客经常同时购买的商品摆放在商店的两端，以诱导同时购买这些商品的顾客一路挑选其他商品。当然，购物篮分析也可以帮助零售商决定将哪些商品降价出售。例如，表 3.1 的事务数据库显示，顾客经常同时购买可乐和面包，{可乐，面包}是频繁项集，则可乐的降价出售可能既促进可乐销售，又促进面包销售。

如果项集的全域是商店中的商品的集合，每种商品有一个对应的布尔变量，表示该商品是否被顾客购买，则每个购物篮都可以用一个布尔向量表示。可以通过分析布尔向量得到反映商品频繁关联或同时购买的模式，这些模式可以用关联规则的形式表示。例如，购买可乐也趋向于同时购买面包的顾客信息可以用以下的关联规则表示：

$$\text{可乐}\rightarrow\text{面包}\ [\text{support}=60\%,\ \text{confidence}=75\%]$$

3.2 Apriori 算法

关联规则挖掘的一种自然的、原始的方法是穷举所有可能的规则，计算每个规则的支持度和置信度，但是这种方法代价很高。提高性能的方法是拆分支持度和置信度。因为规则的支持度主要依赖于规则前件和后件项集的支持度，因此大多数关联规则挖掘算法通常采用的策略是分为两阶段：第一阶段从事务数据库中找出所有大于或等于用户指定的最小支持度的频繁项集；第二阶段利用频繁项集生成需要的关联规则，根据用户设定的最小置信度进行取舍，最后得到强关联规则。由于第二阶段的开销远低于第一阶段，因此

挖掘关联规则的总体性能取决于第一阶段的频繁项集产生算法。

关联规则挖掘的第一阶段的任务是从事务数据库中找出所有的频繁项集,也就是找出所有大于或等于最小支持度阈值(min_sup)的项集。算法一般首先统计事务数据库中各个项,产生频繁 1 项集,从频繁 1 项集中产生频繁 2 项集……从频繁 $k-1$ 项集中产生频繁 k 项集,直到无法再找到更长的频繁项集为止。

关联规则挖掘的第二阶段的任务是由频繁项集产生关联规则。若一个规则的置信度满足最小置信度阈值,称此规则为强关联规则。首先根据选定的频繁项集找到它的所有非空子集,找到所有可能性的关联规则。例如,频繁项集为{1,2,3},它的所有非空子集则为{1,2},{1,3},{2,3},{1},{2},{3}。可能的关联规则为:{1,2}→3,{1,3}→2,{2,3}→1,1→{2,3},2→{1,3},3→{1,2}。最后计算所有可能的关联规则的置信度,找到符合最小置信度的规则,它们就是强关联规则。

3.2.1 Apriori 算法详解

1. 第一阶段:产生频繁项集

从大型数据集中挖掘频繁项集的主要挑战是:挖掘过程中常常产生大量满足最小支持度阈值的项集,特别是当最小支持度阈值 min_sup 设置得很低时尤其如此。这是因为,如果一个项集是频繁的,则它的每个子集也是频繁的。

格结构(lattice structure)常常被用来表示所有可能的项集。从中可以看出频繁项集的搜索空间是指数搜索空间,随着事务数据库中项的增加,候选项集和比较次数都呈指数级增长,计算复杂度很高。图 3.1 为项集 $I=\{a,b,c,d,e\}$ 的格。

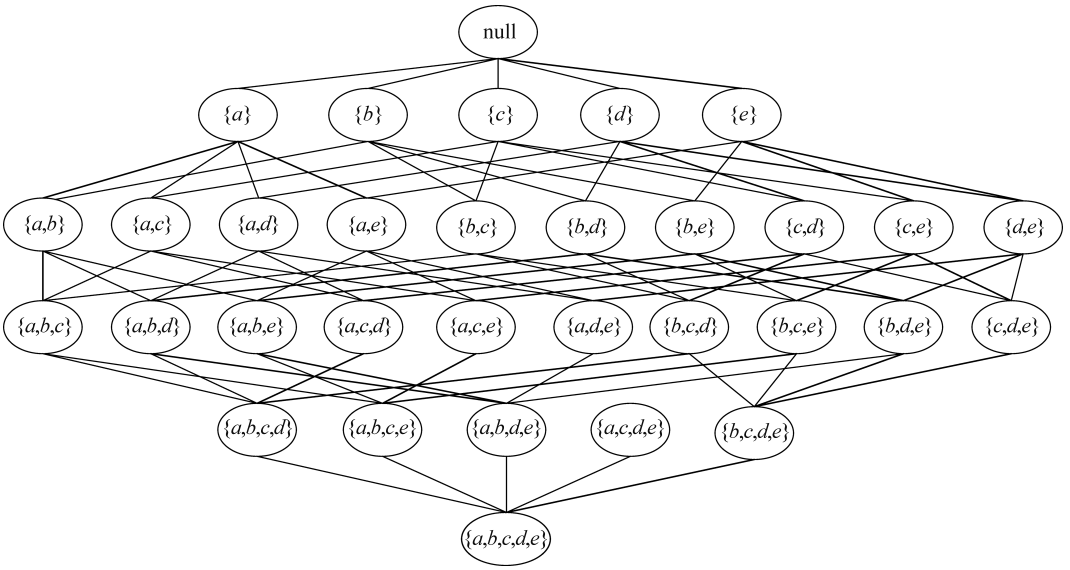


图 3.1 项集 $I=\{a,b,c,d,e\}$ 的格

发现频繁项集的一个朴素的方法是确定格结构中每个候选项集的支持度,但是工作量比较大。降低产生频繁项集的计算复杂度的方法有以下两个:

(1) 减少候选项集的数目。例如,根据先验性质原理,可以不用计算支持度,因而可以删除某些候选项集。

(2) 减少比较次数。利用更高级的数据结构,或者存储候选项集,或者压缩数据集,以减少比较次数。

定理 1:先验性质。频繁项集的所有非空子集也一定是频繁的。

这个性质很容易理解。例如,一个项集$\{I_1,I_2,I_3\}$是频繁的,那么这个项集的支持度大于最小支持度阈值 min_sup。显而易见,它的任何非空子集(如$\{I_1\}$、$\{I_2,I_3\}$等)的支持度也一定比最小支持度阈值 min_sup 大,因此一定都是频繁的。

如图 3.2 所示,如果$\{c,d,e\}$是频繁的,则它的所有子集也是频繁的。反过来,如果项集 I 是频繁的,那么给这个项集再添加新项 A,则这个新的项集$\{I \cup A\}$至少不会比 I 更加频繁,因为增加了新项,所以新项集中所有项同时出现的次数一定不会增加。如果项集 I 是非频繁的,给项集 I 增加新项 A 后,这个新的项集$\{I \cup A\}$一定还是非频繁的。这种性质叫反单调性。

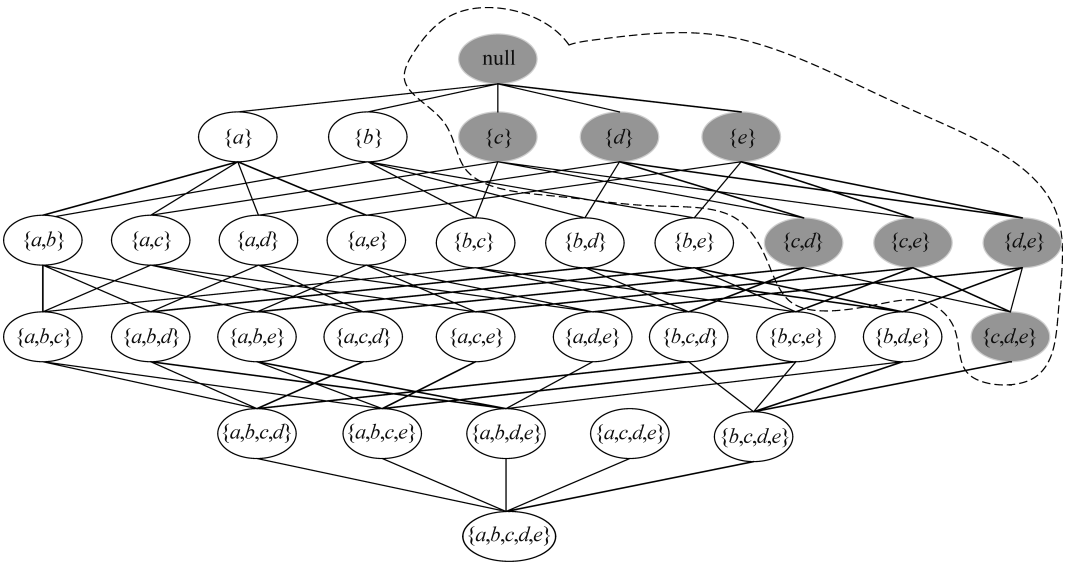


图 3.2　先验性质图示

定理 2:反单调性。在一个项集中,如果有至少一个非空子集是非频繁的,那么这个项集一定是非频繁的。即,如果一个项集是非频繁的,则它所有的超集都是非频繁的。

这种基于支持度度量修剪指数搜索空间的策略称为基于支持度的剪枝。如图 3.3 所示,如果$\{a,b\}$是非频繁项集,则它的所有超集也是非频繁的,其超集在搜索过程中都可以剪掉。

Apriori 算法利用定理 1 和定理 2,通过逐层搜索的模式,由频繁 $k-1$ 项集生成频繁 k 项集,从而最终得到全部的频繁项集。

由定理 1 和定理 2 可知:如果一个项集是频繁 k 项集,那么它的任意非空子集一定是频繁的,所以,频繁 k 项集一定是由频繁 $k-1$ 项集组合生成的。

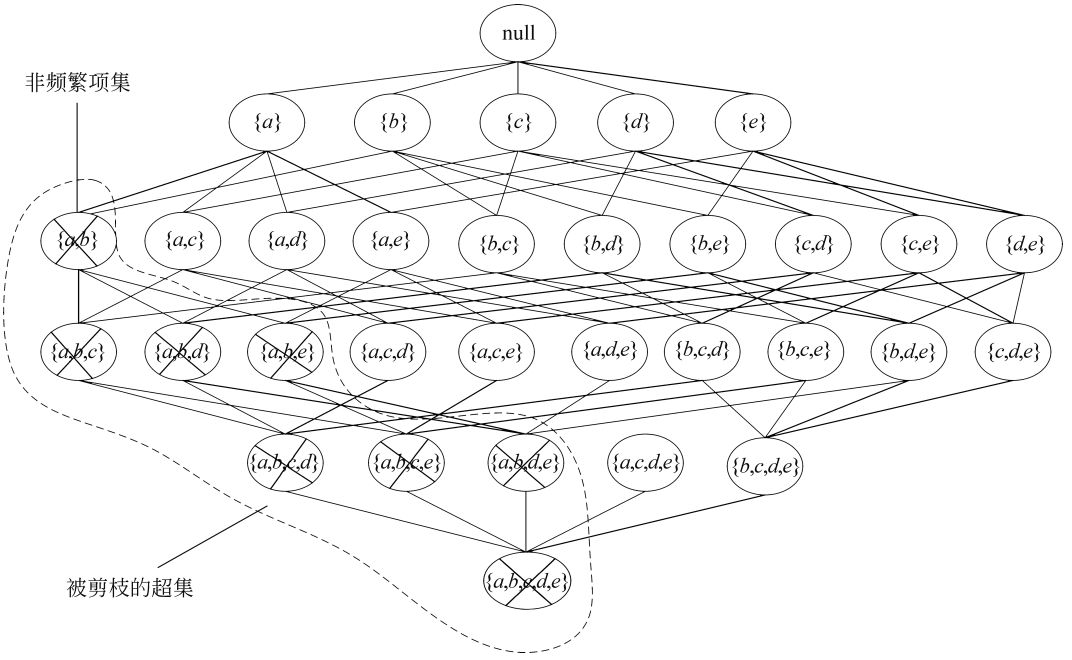


图 3.3　基于支持度的剪枝图示

Apriori 算法的核心是通过频繁 $k-1$ 项集生成频繁 k 项集。定理 1 和定理 2 可以简洁地描述为定理 3 和定理 4。

定理 3：任何频繁 k 项集都是由频繁 $k-1$ 项集组合生成的。

定理 4：频繁 k 项集的所有 $k-1$ 项子集一定都是频繁 $k-1$ 项集。

Apriori 算法使用一种称为逐层搜索的迭代算法，其中 k 项集用于探索 $k+1$ 项集。首先，通过扫描数据库，累计每个项的个数，并收集满足最小支持度的项，找出频繁 1 项集的集合，该集合记为 L_1。然后，使用 L_1 找出频繁 2 项集的集合 L_2，使用 L_2 找出 L_3，如此迭代进行下去，直到不能再找到频繁 $k-1$ 项集。找出每个 L_k 需要对数据库进行一次完整扫描。

Apriori 算法的步骤如下：

(1) 扫描全部数据，产生候选项 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) 若 $k>1$，重复步骤(4)、(5)和(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}，筛选产生频繁 $k+1$ 项集的集合 L_{k+1}。

(6) 若 $L\neq\varnothing$，则 $k=k+1$，跳往步骤(4)；否则，跳往步骤(7)。

(7) 根据最小置信度，由频繁项集产生强关联规则，结束。

Apriori 算法可以描述为算法 3.1：

算法 3.1 Apriori 算法的频繁项集产生

```
1  k=1
2  F_k={i | i∈I ∧ ∂({i})≥N * minsup}        {发现所有的频繁 1 项集}
3  repeat
4    k=k+1
5    C_k=apriori-gen(F_k-1)                  {产生候选项集}
6    for 每个事务 t∈T do
7      C_t=subset(C_k,t)                     {识别属于 t 的所有候选项集}
8      for 每个候选项集 c∈C_t do
9        ∂(c)=∂(c)+1                         {支持度计数增值}
10     end for
11   end for
12   F_k={c | c∈C_k ∧ ∂(c)≥N * minsup}       {提取频繁 k 项集}
13 until F_k=∅
14 Result=∪F_k
```

Apriori 算法产生频繁项集的过程有两个重要的特点：

(1) 逐层进行。从频繁 1 项集到最长的频繁 k 项集，每次遍历项集格中的一层。

(2) 它使用产生和测试(generate-and-test)策略发现频繁项集，每次迭代后的候选项集都由上一次迭代发现的频繁项集产生。算法总迭代次数为 $k_{\max}+1$，其中 $k_{\max}$ 为频繁项集最大长度。

此过程中最重要的环节是如何从频繁 $k-1$ 项集产生频繁 k 项集，即如何从 L_{k-1} 找出 L_k。该环节可由连接、剪枝和扫描筛选 3 步组成，首先通过连接和剪枝产生候选项集，然后通过扫描筛选来实现进一步的删除小于最小支持度阈值的项集。

(1) 连接。

为找出 L_k，通过将 L_{k-1} 与自身连接产生候选 k 项集的集合。连接的作用就是用两个频繁 $k-1$ 项集组成一个 k 项集。该候选 k 项集的集合记为 C_k。具体来说，分为两步：

① 判断两个频繁 $k-1$ 项集是否是可连接的：对于两个频繁 $k-1$ 项集 I_1 和 I_2，先将项集中的项排序(例如按照字典排序或者人为规定的其他序列)，如果 I_1、I_2 的前 $k-2$ 项都相等，则 I_1 和 I_2 可连接。

② 如果两个频繁 $k-1$ 项集 I_1 和 I_2 可连接，则用它们生成一个新的 k 项集：$\{I_1[1], I_1[2], \cdots, I_1[k-2], I_1[k-1], I_2[k-1]\}$，也就是用相同的前 $k-2$ 项加上 I_1 和 I_2 不同的末尾项，这个过程可以用 $I_1 \times I_2$ 表示。只需找到所有的 C_n^2 个两两组合，n 为 L_{k-1} 的长度，挑出其中可连接的，就能生成所有可能是频繁项集的 k 项集，也就是候选频繁 k 项集，将这些候选频繁 k 项集构成的集合记为 C_k。

说明：经过上述方法连接起来的 k 项集，至少有两个 $k-1$ 子集是频繁的，由定理 4 可知，这样的 k 项集才有可能是频繁的。这种连接方法一开始直接排除了大量不可能的组合，所以不需要通过找出所有项的 k 组合来生成候选频繁 k 项集 C_k。

(2) 剪枝。

Apriori 算法使用逐层搜索技术。为了压缩 C_k，利用定理 1 给出的先验性质(任何非

频繁的 $k-1$ 项集都不是频繁 k 项集的子集)来剪掉非频繁的候选 k 项集。对给定候选 k 项集 C_k,只需检查它们的 $k-1$ 项的所有子集是否频繁即可。

说明:因为在连接之后所有的候选频繁 k 项集都在 C_k 中,所以现在的任务是对 C_k 进行筛选,剪枝是初步的筛选。具体过程是:对于每个候选 k 项集,找出它的所有 $k-1$ 项子集,检测其是否频繁,也就是看其是否都在 L_{k-1} 中,只要有一个子集不在其中,那么这个 k 项集一定不是频繁的。剪枝的原理就是定理 4。经过剪枝,C_k 进一步缩减。这个过程也叫子集测试。

(3) 扫描筛选。

因为当前候选频繁 k 项集 C_k 中仍然可能存在支持度小于最小支持度阈值 min_sup 的项集,所以需要做进一步筛选。扫描事务数据库 D,得出在当前 C_k 中 k 项集的计数,这样能统计出目前 C_k 中所有项集的频数,从中删去小于 min_sup 的项集,就得到了频繁 k 项集组成的集合 L_k。

说明:在候选频繁 k 项集的产生过程中要注意以下 3 点。

(1) 应当避免产生太多不必要的候选项集。如果一个候选项集的子集是非频繁的,则该候选项集肯定是非频繁的。

(2) 确保候选项集的集合完整性,即在产生候选项集的过程中没有遗漏任何频繁项集。

(3) 不应当产生重复的候选项集。

2. 第二阶段:由频繁项集产生关联规则

首先,对于每一个频繁项集产生关联规则。计算频繁项集的所有非空真子集,计算所有可能的关联规则的置信度,如果其置信度大于最小置信度阈值,则该规则为强关联规则,输出该规则。即,对于选定的频繁项集 I 的每个非空子集 s,如果 $\mathrm{Conf}(S\rightarrow I-S)=\dfrac{\mathrm{Support}(I)}{\mathrm{Support}(S)}=\dfrac{|I|}{|S|}\geqslant \mathrm{min\_conf}$,则输出规则 $S\Rightarrow(I-S)$。其中,min_conf 是最小置信度阈值。

说明:由于规则由频繁项集产生,每个规则都自动满足最小支持度阈值,这里不需要再计算支持度的满足情况。

例如,频繁项集为{1,2,3},则其非空真子集为{1,2},{1,3},{2,3},{1},{2},{3}。

可能的关联规则为{1,2}→3, {1,3}→2, {2,3}→1, 1→{2,3},2→{1,3},3→{1,2}。

最后,计算所有可能的关联规则的置信度,找到满足最小置信度阈值的规则,它们就是强关联规则。

图 3.4 给出了从项集{0,1,2,3}产生的所有关联规则,其中阴影区域给出的是低可信度的规则。可以发现,如果{0,1,2}→{3}是一条低可信度规则,那么所有其他以 3 作为后件(箭头右部包含 3)的规则均为低可信度的规则。

可以观察到,如果某条规则并不满足最小可信度阈值要求,那么该规则的所有子集也不会满足最小可信度阈值要求。以图 3.4 为例,假设规则{0,1,2} → {3}并不满足最小可信度阈值要求,那么任何左部为{0,1,2}子集的规则也不会满足最小可信度阈值要求。可以利用关联规则的上述性质来减少需要测试的规则数目,类似于 Apriori 算法求解频繁

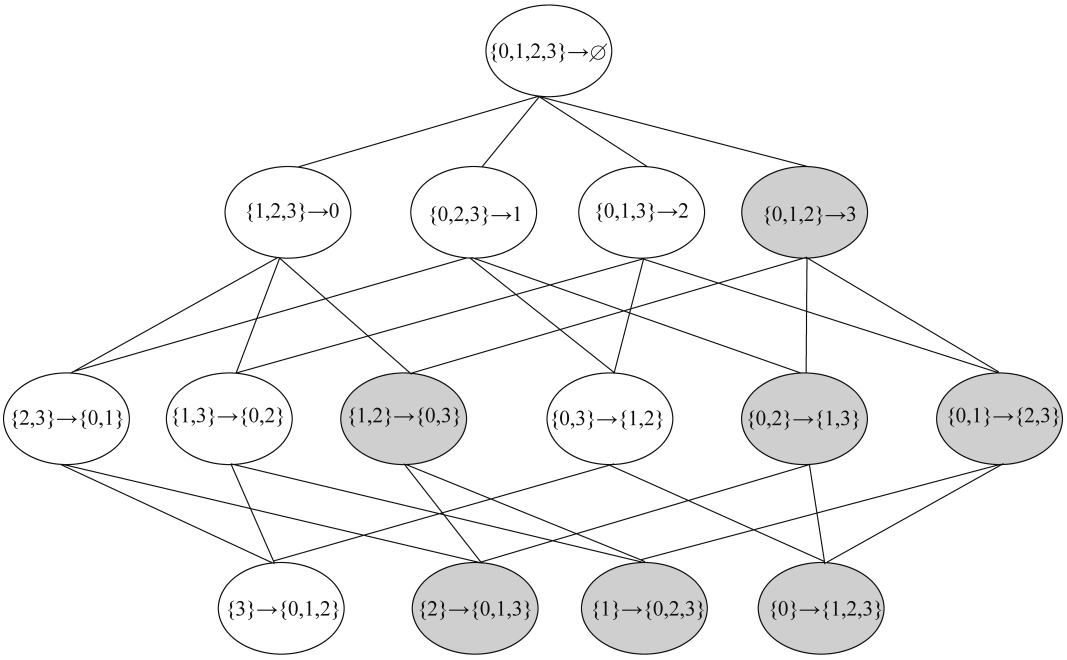


图 3.4　频繁项集{0,1,2,3}的关联规则

项集时采用的方法。

3.2.2　Apriori 算法的例子

本例子依据的交易数据如表 3.2 所示(假设给定的最小支持度阈值为 2,最小置信度阈值为 0.6)。

针对本例,最简单的办法是穷举法,即把每个项集都作为候选项集,统计它在数据集中出现的次数,如果其出现次数大于最小支持度计数,则为频繁项集,如图 3.5 所示,但该方法开销很大。

表 3.2　某商店交易数据

交 易 号 码	商　　品
100	Cola,Egg,Ham
200	Cola,Diaper,Beer
300	Cola,Diaper,Beer,Ham
400	Diaper,Beer

采用 Apriori 算法挖掘规则的过程如下:

第 1 步:生成 1 项集的集合 C_1。将所有事务中出现的项组成一个集合,记为 C_1,C_1 可以看作由所有的 1 项集组成的集合。在本例中,所有可能的 1 项集 C_1 为{E}、{C}、{D}、{B}、{H},分别代表 Egg、Cola、Diaper、Beer、Ham。

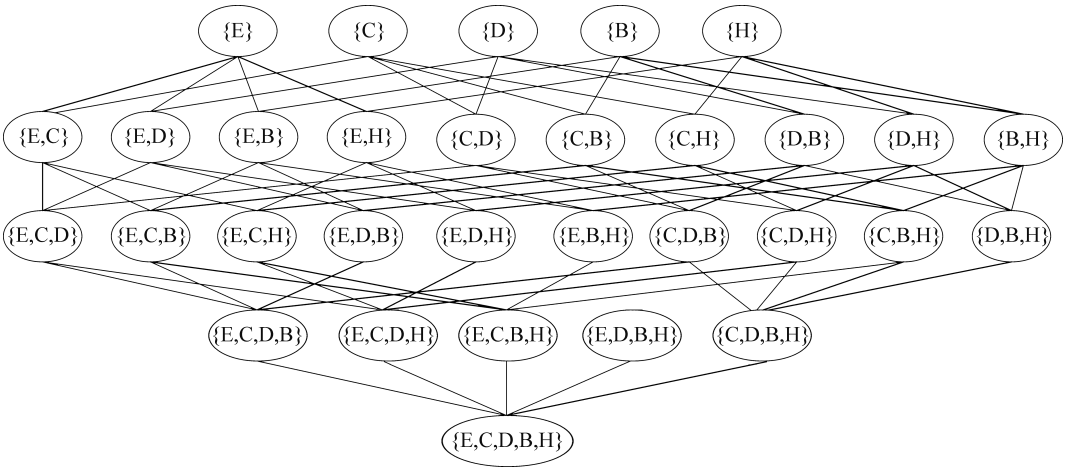


图 3.5　项集格

第 2 步：寻找频繁 1 项集。统计 C_1 中所有元素出现的次数，再与最小支持度阈值 min_sup 比较，筛除小于 min_sup 的项集，剩下的都是频繁 1 项集。将这些频繁 1 项集组成的集合记为 L_1。在本例中，设置 min_sup＝2。经过这一步的筛选，项集{E}被淘汰。{E}的超集都不可能是频繁项集，这样，在项集空间中就剪掉了一个分枝。经过剪枝后的图如图 3.6 所示。

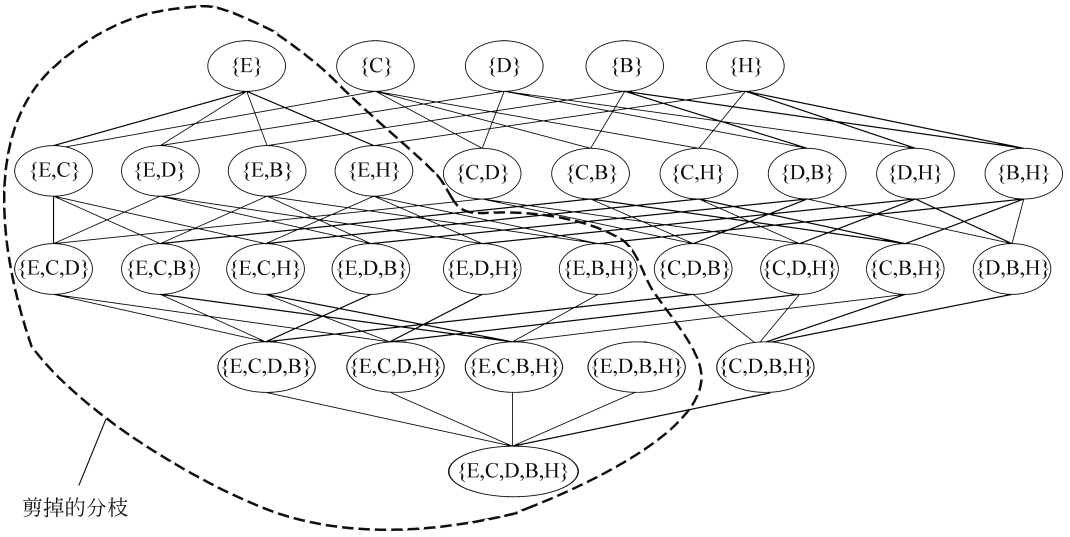


图 3.6　剪枝后的项集

频繁 1 项集为{C}、{D}、{B}、{H}。

频繁 1 项集的发现过程如图 3.7 所示。

按照算法流程，在得到频繁 1 项集后，通过连接和剪枝得到候选 2 项集 C_2，通过扫描事务数据库并进行计数筛选得到频繁 2 项集 L_2。同样，使用 L_2 找出 L_3。如此继续下去，直到不能再找到频繁项集时为止。

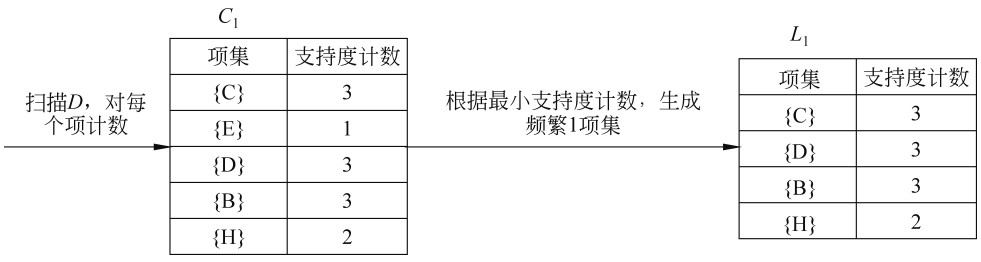


图 3.7　频繁 1 项集发现过程

具体过程如图 3.8 所示。

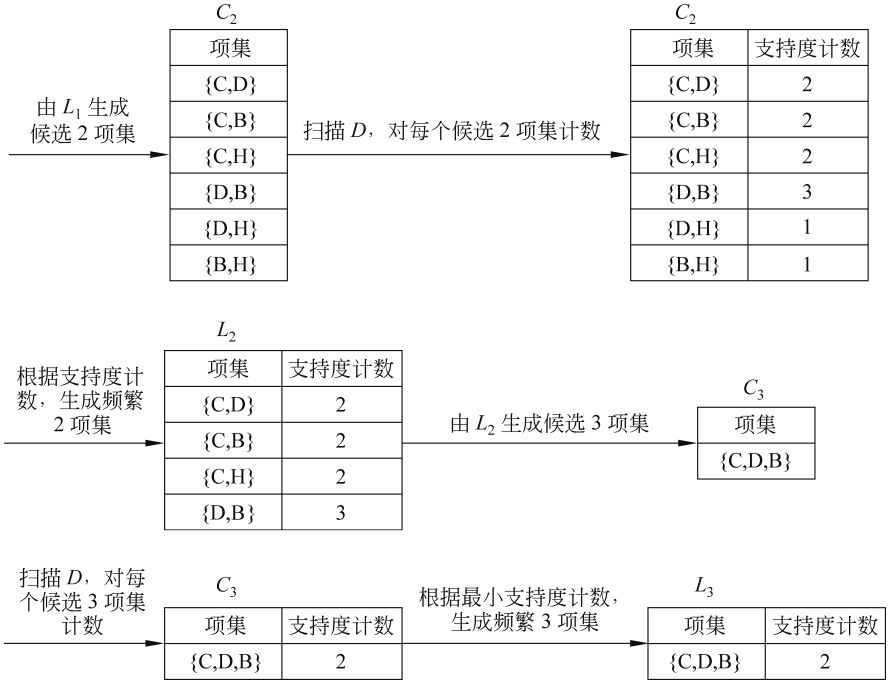


图 3.8　频繁项集生成过程

从上述过程可以看到，频繁 2 项集有{C,D}，{C,B}，{C,H}，{D,B}，最后得到的最大频繁项集为频繁 3 项集：{C,D,B}。

在第二阶段，由频繁项集产生关联规则。

频繁 2 项集{C,D}生成强关联规则的过程如下：

{C,D}的非空真子集为{C}、{D}。P(C→D)＝2/3，P(D→C)＝2/3，都大于最小置信度阈值，所以规则 C→D 和 D→C 都是强关联规则。

同理，可以计算频繁 2 项集{C,B}生成的强关联规则：P{C→B}＝2/3，P{B→C}＝2/3，都大于最小置信度阈值，所以规则 C→B 和 B→C 都是强关联规则。

同理，可以计算频繁 2 项集{C,H}生成的强关联规则：P{C→H}＝2/3，P{H→C}＝1，都大于最小置信度阈值，所以规则 C→H 和 H→C 都是强关联规则。

同理,可以计算频繁 2 项集{D,B}生成的强关联规则:$P\{D\rightarrow B\}=1$,$P\{B\rightarrow D\}=1$,都大于最小置信度阈值,所以规则 D→B 和 B→D 都是强关联规则。

频繁 3 项集{C,D,B}生成强关联规则的过程如下:

最终生成的{C,D,B}可能的非空真子集为{C}、{D}、{B}、{C,D}、{D,B}、{B,C}。

分别计算以下概率:

$$P(C,D|B)=\frac{P(C,D,B)}{P(B)}=\frac{2}{3}$$

$$P(B,D|C)=\frac{P(B,D,C)}{P(C)}=\frac{2}{3}$$

$$P(B,C|D)=\frac{P(B,D,C)}{P(D)}=\frac{2}{3}$$

$$P(C,D\rightarrow B)=\frac{2}{2}=1$$

$$P(B,D\rightarrow C)=\frac{2}{3}$$

$$P(B,C\rightarrow D)=\frac{2}{2}=1$$

因此,这些规则都为强关联规则。

3.2.3 Apriori 算法总结

关联分析是用于发现大数据集中各项间有价值的关系的一种方法。可以采用两种方式来量化这些有价值的关系:第一种方式是使用频繁项集,它会给出经常在一起出现的项;第二种方式是关联规则,关联规则意味着项之间存在“如果……那么……”关系。

发现项的不同组合是十分耗时的任务,不可避免地需要消耗大量的计算资源,这就需要采用一些智能的方法在合理的时间范围内找到频繁项集。Apriori 算法是一个经典的算法,它使用 Apriori 原理来减少在数据库上进行检查的集合的数目。Apriori 算法从 1 项集开始,通过组合满足最小支持度阈值要求的项集来形成更大的集合。每次增加频繁项集的大小,Apriori 算法都会重新扫描整个数据集。当数据集很大时,这会显著降低频繁项集的发现速度。

3.3 FP-Growth 算法

在关联分析中,Apriori 算法是挖掘频繁项集常用的算法。Apriori 算法是一种先产生候选项集再检验是否频繁的“产生并测试”方法。它使用先验性质来压缩搜索空间,以提高逐层产生频繁项集的效率。尽管 Apriori 算法非常直观,但需要进行大量计算,包括产生大量候选项集和进行支持度计算,需要频繁地扫描数据库,运行效率很低。特别是对于海量数据,Apriori 算法的时间和空间复杂度都不容忽视,每计算一次 C_k 就需要扫描一遍数据库。

Jiawei Han 等人在 2000 年提出了 FP-Growth 算法(Frequent Pattern Growth,频繁

模式增长),它可以挖掘出全部频繁项集,而无须经历 Apriori 算法代价昂贵的候选项集产生过程。FP-Growth 算法是基于 Apriori 原理提出的关联分析算法,FP-Growth 算法巧妙地将树状结构引入算法,采取分治策略:将提供频繁项集的数据库压缩为一棵频繁模式树(Frequent Pattern Tree,FP-Tree),并保留项集的关联信息。该算法和 Apriori 算法最大的不同有两点:第一,该算法不产生候选集;第二,该算法只需要扫描两次数据库,大大提高了效率。在 FP-Growth 算法中经常用到以下几个概念。

(1) 频繁模式树(以下简称 FP 树)。将事务数据表中的各个事务数据项按照支持度排序后,把每个事务数据项按降序依次插入一棵以 NULL 为根节点的树中,同时在每个节点处记录该节点的支持度。

(2) 条件模式基。包含在 FP 树中与后缀模式一起出现的前缀路径的集合。

(3) 条件树。将条件模式基按照 FP 树的构造原则形成的一棵新的 FP 子树。

FP-Growth 算法发现频繁项集的过程如下:

(1) 构建 FP 树。将提供频繁项集的数据库压缩为一棵 FP 树,并保留项集的关联信息。

(2) 从 FP 树中挖掘频繁项集。把这种压缩后的数据库划分成一组条件数据库,每个条件数据库关联一个频繁项或模式段,并分别挖掘每个条件数据库。具体步骤如下:

① 扫描原始事务数据集,根据最小支持度阈值条件得到频繁 1 项集,对频繁 1 项集中的项按照频度降序排序。然后,删除原始事务数据集中非频繁的项,并将事务按项集中降序排列。

② 第二次扫描,根据频繁 1 项集创建频繁项头表(从上往下降序)。

③ 构建 FP 树。读入排序后的数据集,插入 FP 树。插入时按照排序后的顺序,排序靠前的是祖先节点,而靠后的是子孙节点。如果有共同的祖先节点,则对应的共同祖先节点计数加 1。插入节点后,如果有新节点出现,则频繁项头表对应的节点会通过节点链表链接新节点。直到所有的数据都插入 FP 树后,FP 树的构建完成。

④ 从 FP 树中挖掘频繁项集。从频繁项头表的底部依次从下向上找到所有包含该项的前缀路径,即其条件模式基(Conditional Pattern Base,CPB),从条件模式基递归挖掘得到频繁项头表项的频繁项集。递归调用树结构构建 FP 子树时,删除小于最小支持度阈值的项。如果条件模式基(FP 子树)最终呈现单一路径的树结构,则直接列举所有组合;否则继续调用树结构,直到形成单一路径时为止。

说明:FP-Growth 算法通过两次扫描数据库,将原始数据集压缩为一个树状结构;然后找到每个项的条件模式基,递归挖掘频繁项集。

3.3.1　FP-Growth 算法详解

1. FP 树数据结构

为了减少 I/O 次数,FP-Growth 算法使用一种称为频繁模式树(FP 树)的数据结构来存储数据。FP 树是一种特殊的前缀树,由频繁项头表和项前缀树构成。

FP-Growth 算法基于以上的树状结构来加快整个挖掘过程。这个数据结构包括 3 部分,如图 3.9 所示。

原始数据集

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

频繁项头表

Item	Pointer
B	8
A	7
C	7
D	5
E	3

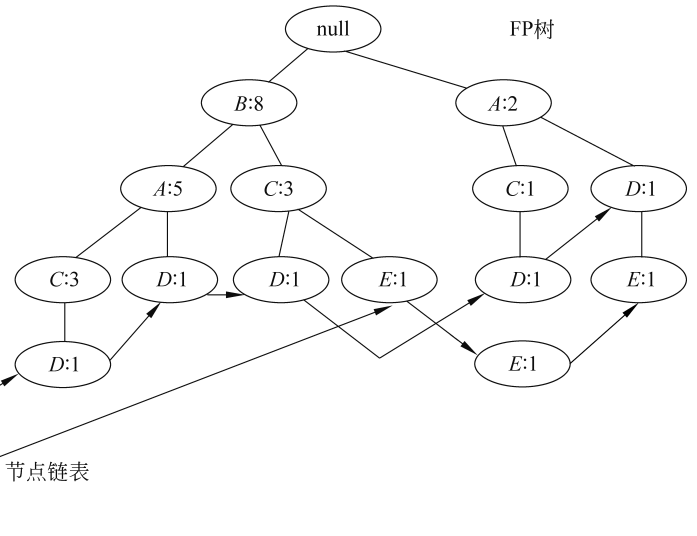


图 3.9 FP 树的数据结构

第一部分是频繁项头表。它记录了所有的频繁 1 项集出现的次数,按照次数降序排列。例如,在图 3.9 中,B 在所有 10 组数据中出现了 8 次,出现次数最多,因此排在第一位;E 出现了 3 次,出现次数最少,排在最后一位。

第二部分是 FP 树。原始数据集被映射到内存中的一棵 FP 树,它保留了项集的关联信息。

第三部分是节点链表。频繁项头表里的每个频繁 1 项集都是一个节点链表的头,它指向 FP 树中该频繁 1 项集出现的位置,构成该项集在树中出现的节点的节点链表。这样做主要是为了方便频繁项头表和 FP 树之间的联系的查找和更新。

下面分别讨论频繁项头表和 FP 树的建立过程。

2. 频繁项头表的建立

FP 树的建立需要依赖频繁项头表的建立。首先要建立频繁项头表。

第一次扫描原始数据集,得到所有频繁 1 项集的计数。然后删除小于最小支持度阈值的项,将频繁 1 项集放入频繁项头表,并按照支持度降序排列。第二次扫描原始数据集,剔除原始数据中的非频繁 1 项集,并按照支持度降序排列。

例如,如图 3.10 所示,假设最小支持度阈值是 20%,事务数据集中有 10 条数据。第一次扫描数据并对 1 项集进行计数,发现 O、I、L、J、P、M、N 都只出现一次,支持度低于 20%的阈值,因此不会出现在频繁项头表中;剩下的 A、C、E、G、B、D、F 按照支持度的大小降序排列,组成了频繁项头表。

接着第二次扫描数据,剔除非频繁 1 项集,并按照支持度降序排列。例如,在数据项 $ABCEFO$ 中,O 是非频繁 1 项集,因此被剔除,只剩下 $ABCEF$;然后按照支持度降序排序,变成了 $ACEBF$。其他的数据项以此类推。对原始数据集里的数据项进行排序是为了在后面构建 FP 树时可以尽可能地共用祖先节点。

通过两次扫描，频繁项头表已经建立，也得到了排序后的数据集，如图 3.10 所示。接下来就可以建立 FP 树了。

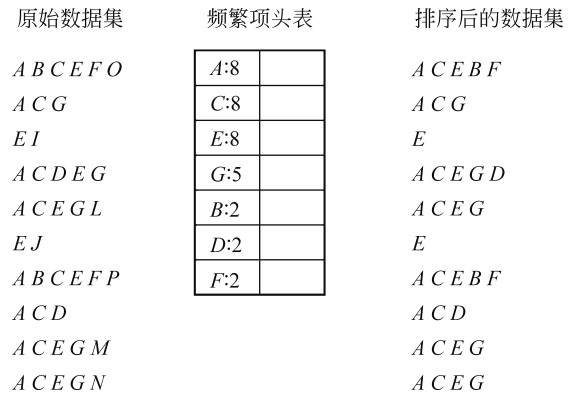


图 3.10　建立项头表

3. FP 树的建立

有了项头表和排序后的数据集，就可以开始 FP 树的建立了。开始时 FP 树没有数据。在建立 FP 树时，一条一条地读入排序后的数据集，并按照排序后的顺序插入 FP 树中。

下面用图 3.10 的例子说明 FP 树的建立过程。

首先，插入第一条数据 *ACEBF*，如图 3.11 所示。此时 FP 树没有节点，因此 *ACEBF* 是一个独立的路径，所有节点计数为 1。频繁项头表通过节点链表链接对应的新增节点，如图 3.11 所示。

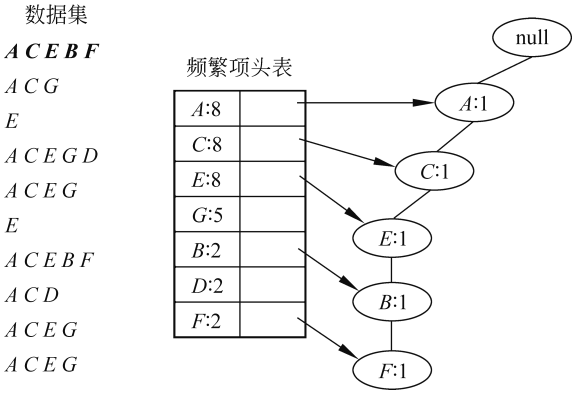


图 3.11　插入数据 *ACEBF*

接着插入数据 *ACG*，如图 3.12 所示。由于 *ACG* 和现有的 FP 树可以有共同的祖先节点序列 *AC*，因此只需要增加一个新节点 *G*，其计数为 1。同时 *A* 和 *C* 的计数加 1，成为 2。当然，*G* 的节点链表要更新。

用同样的方法插入数据 *E*，如图 3.13 所示。需要注意的是，由于插入 *E* 后多了一个节点，因此需要通过节点链表链接新增的节点 *E*。

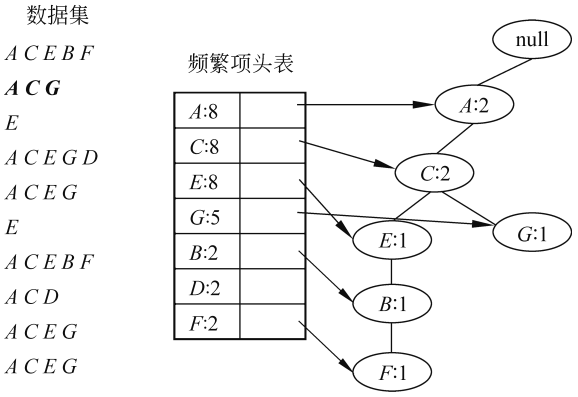


图 3.12　插入数据 *ACG*

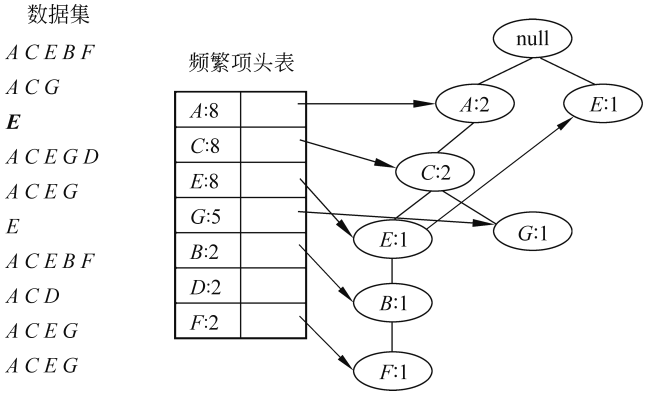


图 3.13　插入数据 *E*

采用同样的方法更新其余 7 条数据，如图 3.14～图 3.20 所示。由于原理类似，这里就不再一一讲解了。

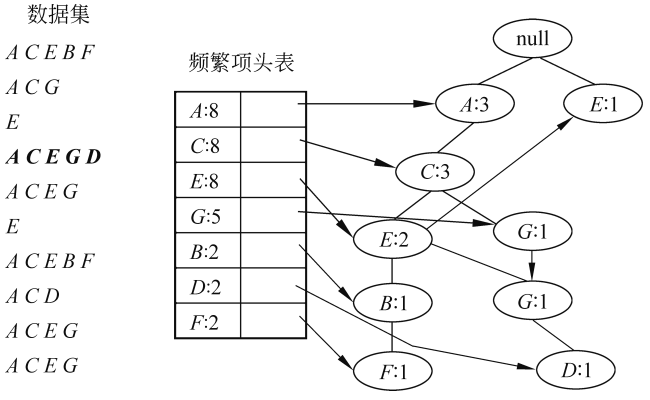


图 3.14　插入数据 *ACEGD*

数据集

A C E B F
A C G
E
A C E G D
A C E G
E
A C E B F
A C D
A C E G
A C E G

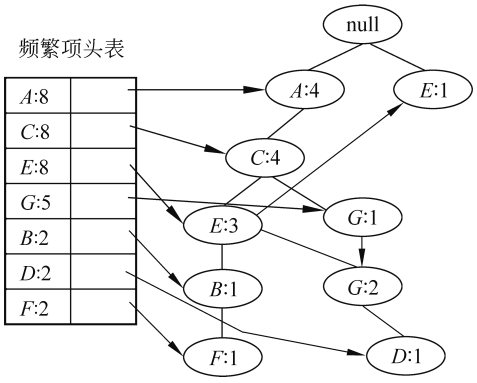


图 3.15　插入数据 *ACEG*

数据集

A C E B F
A C G
E
A C E G D
A C E G
E
A C E B F
A C D
A C E G
A C E G

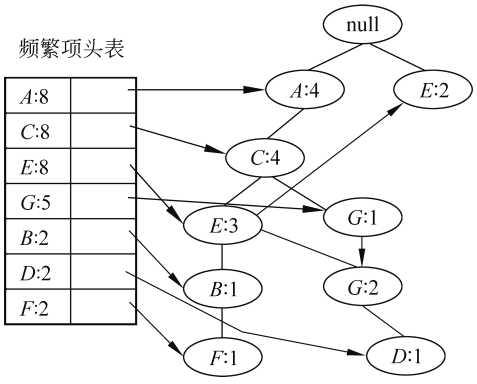


图 3.16　插入数据 *E*

数据集

A C E B F
A C G
E
A C E G D
A C E G
E
A C E B F
A C D
A C E G
A C E G

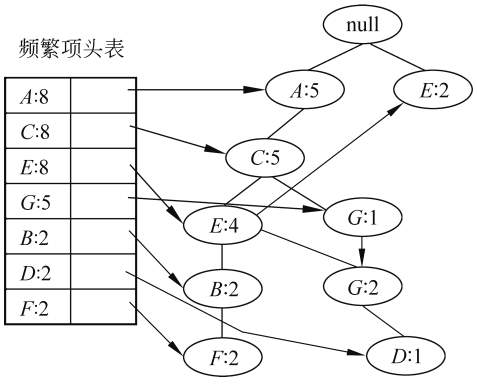


图 3.17　插入数据 *ACEBF*

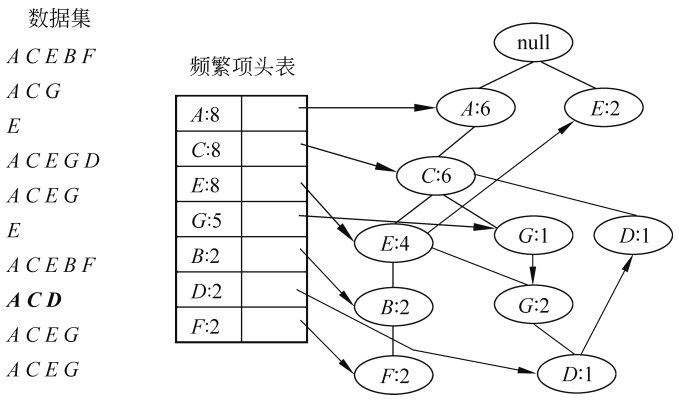


图 3.18　插入数据 *ACD*

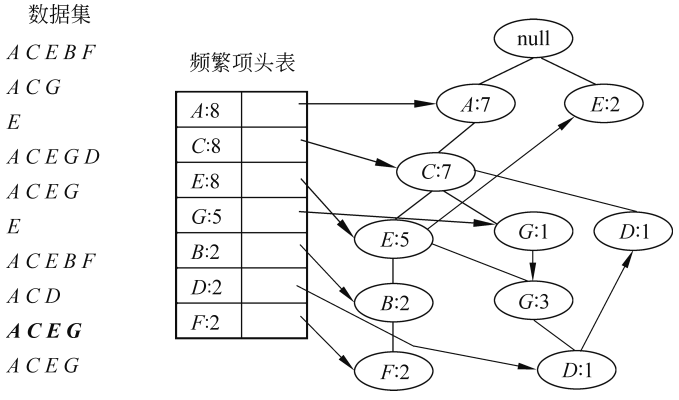


图 3.19　插入数据 *ACEG*

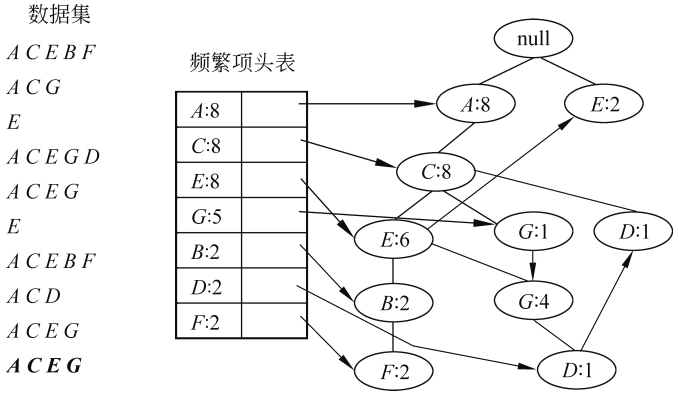


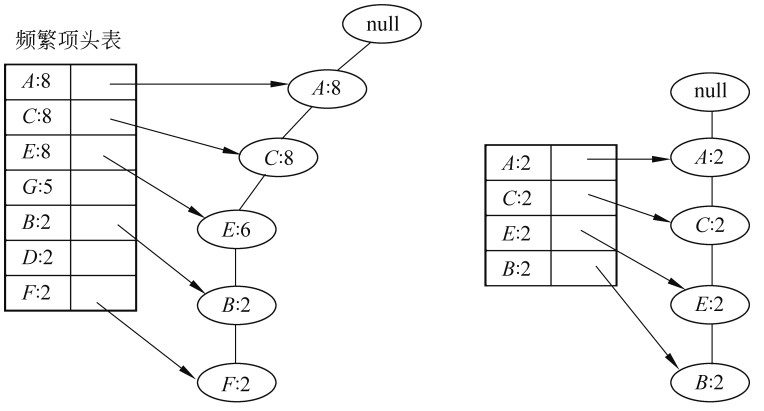
图 3.20　插入数据 *ACEG*

4. 挖掘频繁项集

至此,已经把 FP 树建立起来了,并得到了项头表以及节点链表。那么,如何从 FP 树里挖掘频繁项集呢?

首先要从项头表的底部项依次向上挖掘。对于项头表对应于FP树的每一项，要找到它的条件模式基。所谓条件模式基，是以要挖掘的节点作为叶子节点的FP子树。得到这棵FP子树后，将该子树中每个节点的计数设置为叶子节点的计数，并删除计数低于支持度计数阈值的节点。如果条件模式基(FP子树)最终呈现单一路径的树结构，则直接列举所有组合；否则继续递归调用树结构，直到形成单一路径时为止。从条件模式基(FP子树)出发，就可以通过递归挖掘得到频繁项集了。

下面以图3.20中的FP树为例，介绍从FP树中挖掘频繁项集的过程。首先从最底下的F节点开始，先寻找F节点的条件模式基。由于F在FP树中只有一个节点，因此候选就只有图3.21(a)所示的一条路径，对应$\{A:8,C:8,E:6,B:2,F:2\}$。接着将所有的祖先节点计数设置为叶子节点的计数，即FP子树变成$\{A:2,C:2,E:2,B:2,F:2\}$。一般条件模式基可以不写叶子节点，因此最终F的条件模式基如图3.21(b)所示。



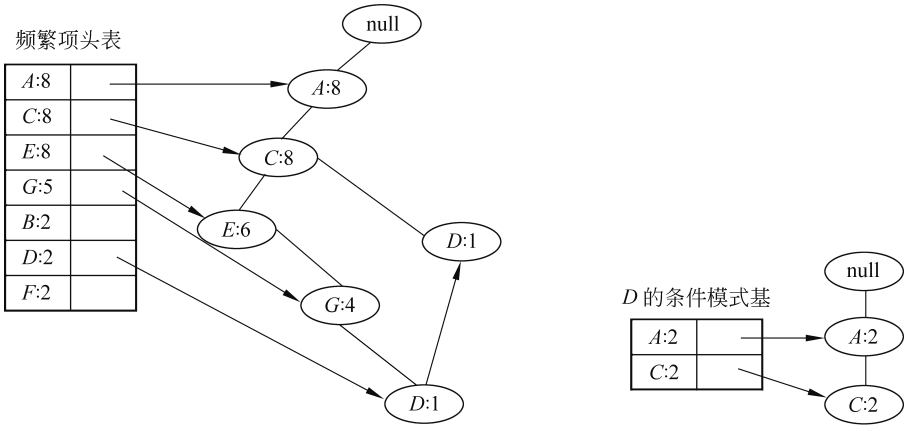
(a) F的FP子树　　(b) F的条件模式基

图3.21　建立F的条件模式基

因为该条件模式基呈现单一路径树结构，可以直接列举其所有组合，很容易得到F的4个频繁2项集：$\{A:2,F:2\}$、$\{C:2,F:2\}$、$\{E:2,F:2\}$、$\{B:2,F:2\}$。递归合并频繁项集，得到的频繁3项集有6个：$\{A:2,C:2,F:2\}$、$\{A:2,E:2,F:2\}$、$\{A:2,B:2,F:2\}$、$\{C:2,E:2,F:2\}$、$\{C:2,B:2,F:2\}$、$\{E:2,B:2,F:2\}$；频繁4项集有4个：$\{A:2,C:2,E:2,F:2\}$、$\{A:2,C:2,B:2,F:2\}$、$\{A:2,E:2,B:2,F:2\}$、$\{C:2,E:2,B:2,F:2\}$；最大的频繁项集为频繁5项集，有1个：$\{A:2,C:2,E:2,B:2,F:2\}$。

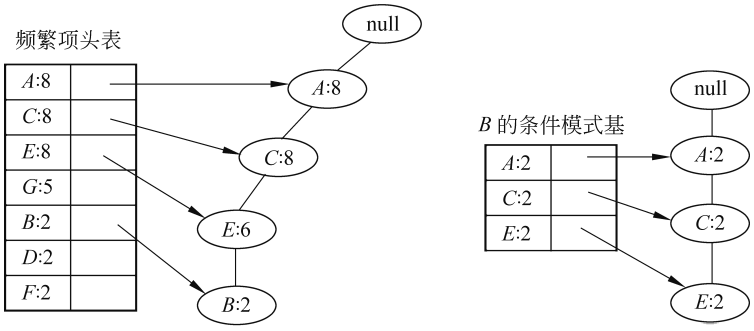
F节点频繁集挖掘完后，开始挖掘D节点的频繁项集。D节点比F节点复杂一些，因为它有两个叶子节点，因此首先得到的FP子树如图3.22(a)所示。接着将所有的祖先节点计数设置为叶子节点的计数，即变成$\{A:2,C:2,E:1,G:1,D:1\}$，此时E节点和G节点由于在条件模式基中的支持度低于阈值，被删除。最终，D的条件模式基为$\{A:2,C:2\}$。很容易得到D的频繁2项集为$\{A:2,D:2\}$、$\{C:2,D:2\}$。合并频繁2项集，得到的频繁3项集为$\{A:2,C:2,D:2\}$，是D对应的最大的频繁项集。

用同样的方法可以得到B的FP子树和条件模式基，如图3.23所示。递归挖掘得到的B的最大频繁项集为频繁4项集：$\{A:2,C:2,E:2,B:2\}$。



(a) D 的 FP 子树　　(b) D 的条件模式基

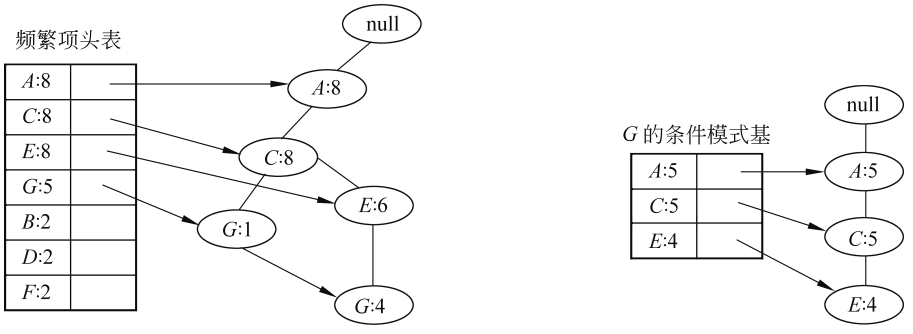
图 3.22　建立 *D* 的条件模式基



(a) B 的 FP 子树　　(b) B 的条件模式基

图 3.23　建立 *B* 的条件模式基

继续挖掘 G 的频繁项集。G 的 FP 子树和条件模式基如图 3.24 所示。递归挖掘得到的 G 的最大频繁项集为频繁 4 项集：$\{A:5, C:5, E:4, G:4\}$。



(a) G 的 FP 子树　　(b) G 的条件模式基

图 3.24　建立 *G* 的条件模式基