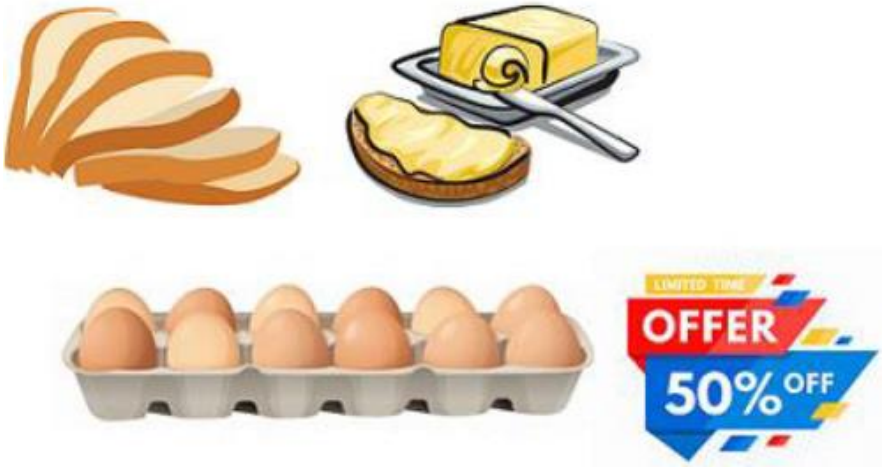
关联分析: 理论与算法
Association Analysis: Theory and Algorithms

关联分析: 理论与算法

- **1. 概述**
- **2. 基本概念与解决方法**
- **3. 经典算法**
  - **Apriori算法**
  - **FP-growth算法**

1 概述—什么是关联规则挖掘？

- 如何发现数据中蕴含的内在规律？
  - 有没有发生过这样的事：你出去买东西，结果却买了比你计划的多得多的东西？这是一种被称为冲动购买的现象，大型零售商利用机器学习和关联规则算法，让我们倾向于购买更多的商品。
  - 购物车分析是大型超市用来揭示商品之间关联的关键技术之一。他们试图找出不同物品和产品之间的关联，这些物品和产品可以一起销售，这有助于正确的产品放置



1 概述—什么是关联规则挖掘？

- 如何发现数据中蕴含的内在规律？
  - 那些产品经常被一起购买？
  - 买了PC之后接着都会买些什么？
  - 哪种DNA对这种新药敏感
- 关联分析(Association Analysis or Association Rule Mining)：用于发现隐藏在大型数据集中令人感兴趣的频繁出现的模式、关联和相关性。
- 关联规则挖掘是数据挖掘中最活跃的研究内容之一，也是无监督学习系统中挖掘本地模式的最普遍形式。

1 概述—什么是关联规则挖掘？

- 最早由R. Agrawal (阿戈沃)等人针对超市购物篮分析(Market Basket Transaction)问题提出的，其目的是为了发现超市交易数据中不同商品之间的关联关系。
  - 一个典型的**关联规则**的例子：70%购买了牛奶的顾客将倾向于同时购买面包。
  - 发现这样的关联规则可以为**市场预测**、**决策**和**策划**等方面提供依据。

关联分析: 理论与算法

- **1. 概述**
- **2. 基本概念与解决方法**
- **3. 经典算法**
  - **Apriori算法**
  - **FP-growth算法**

2 基本概念与解决方法

- 假定某超市销售的商品包括：bread、bear、cake、cream、milk和tea。
- 有如下交易数据：

交易号 TID	顾客购买商品 Items
T1	bread cream milk tea
T2	bread cream milk
T3	cake milk
T4	milk tea
T5	bread cake milk
T6	bread tea
T7	beer milk tea
T8	bread tea
T9	bread cream milk tea
T10	bread milk tea

2 基本概念与解决方法

【定义1】 项与项集

- 设$I=\{i_1, i_2, \ldots, i_m\}$是由$m$个不同项构成的集合，其中的每个$i_k$ (k=1, 2, …, m)被称为一个项(Item)。
  - 例如：在超市的关联规则挖掘中，项就是顾客购买的各种商品，如：bread, milk等。
- 项的集合I被称为项集合(Itemset)，简称项集。
  - I中元素个数称为项集的长度；
    - 例如：超市出售6种商品，即：项集I中包含6个项目，则I的长度为6。
  - 长度为k的项集称为k-项集(k-Itemset)。
    - 例如：对于项集{cake, milk}，可称为2-项集。

2 基本概念与解决方法

【定义2】 **事务**

- 每个**事务**T(Transaction)是项集I上的一个子集，即：$T \subseteq I$，但通常$T \subset I$。
  - 每一个事务有一个唯一的标识：事务号，记作*TID*。
  - 例如：上面数据表中事务号T_3表示第三个事务，它是一个2-项集{milk, cake}，是由6种商品构成的项集的一个子集。
- 事务的全体被称为事务数据集，记作*D*，简称**事务集**。
  - 事务集D中事务的个数记为$|D|$。
  - 例如：上面数据表构成的事务集包含10个事务。

2 基本概念与解决方法

【定义3】 关联规则

- 关联规则(Association Rule)是一个蕴涵表达式：

$$R: A \Rightarrow B\ [s, c]$$

  - 其中$A \subset I$，$B \subset I$，并且$A \cap B = \varnothing$
  - A称为关联规则的条件，B称为关联规则的结果，即：关联规则描述了A中的项出现时，B中的项也随着出现的规律。
  - 规则$A \Rightarrow B$在事务集D中成立，并且具有支持度s和置信度c。
  - 关联规则反映一个事物与其他事物之间的相互依存性和关联性

2 基本概念与解决方法

【定义3】 关联规则

- 关联规则(Association Rule)是一个蕴涵表达式：

$$R: A \Rightarrow B\ [s, c]$$

  - 其中$A \subset I$，$B \subset I$，并且$A \cap B = \varnothing$
  - 例如：规则R_1: {bread} $\Rightarrow$ {milk}；规则R_2: {cream} $\Rightarrow$ {bread, milk}；它们都可能是用户感兴趣的关联规则。
  - 关联规则可以看作是一种IF-THEN关系

- 关联规则的度量：支持度s、置信度c 和提升度l。

2 基本概念与解决方法

【定义4】 **关联规则的支持度**

对于事务集D中的一个关联规则R：$A \Rightarrow B$，

- **关联规则R的支持度**(Support)是事务集D中同时包含A和B的事务数与所有事务数之比：

$$support(A \Rightarrow B) = P(A \cup B)$$

- **关联规则的支持度**反映了A和B中所含的项在事务集中同时出现的频率(概率)。
- 例如：某个商品组合出现的次数与总次数之间的比例，支持度越高表示该组合出现的几率越大。

2 基本概念与解决方法

【定义4】 **关联规则的支持度**

对于事务集D中的一个关联规则R：$A \Rightarrow B$，

- **关联规则R的支持度**(Support)是事务集D中同时包含A和B的事务数与所有事务数之比：

$$support(A \Rightarrow B) = P(A \cup B)$$

- **关联规则的支持度**反映了A和B中所含的项在事务集中同时出现的频率(概率)。
  - 对于关联规则$R1$: {bread} $\Rightarrow$ {milk}，则$support(R1)$ = $support$({bread,milk}) = 0.5。
  - 对于关联规则$R2$: {milk} $\Rightarrow$ {bread}，则$support(R2)$ = $support$({bread,milk}) = 0.5。

2 基本概念与解决方法

【定义5】 **关联规则的置信度**

对于事务集D中的一个关联规则R：$A \Rightarrow B$，

- **关联规则R的置信度**(Confidence)是指D中包含A和B的事务数与包含A的事务数的商：

  $confidence(A \Rightarrow B) = support(A \Rightarrow B) / support(A) = P(AB) / P(A) = P(B \mid A)$

- **关联规则的置信度**反映了如果事务中包含A，则事务中同时出现B的概率。
  - 对于关联规则$R1$: {bread} $\Rightarrow$ {milk}，则$R1$的置信度为 $confidence(R1) = support(\{bread,milk\}) / support(\{bread\}) = 0.5/0.7 = 5/7$。

2 基本概念与解决方法

【定义5】 **关联规则的提升度**

对于事务集D中的一个关联规则R：$A \Rightarrow B$，

- **关联规则R的提升度**(Lift)是指D中A的出现,对B出现概率提升了多少,即“商品A的出现,对商品B的出现概率提升”的程度：

$$lift(A \Rightarrow B) = confidence(A \Rightarrow B) / support(B)$$

$$= \text{support}(A \Rightarrow B) / \text{support}(A) * support(B)$$

$$= P(B \mid A) / P(B)$$

- **关联规则的提升度与置信度同样用于衡量规则的可靠性，可以看作是置信度的一种互补指标**

2 基本概念与解决方法

【定义5】 关联规则的提升度

- 举例来说，我们考虑1000个消费者，发现有500人购买了茶叶，其中有450人同时购买了咖啡，于是规则茶叶→咖啡的置信度高达450/500=90%，由此我们可能会认为喜欢喝茶的人往往喜欢喝咖啡。但当我们来看另外没有购买茶叶的500人，其中同样有450人也买了咖啡，且同样是很高的置信度90%，由此，我们看到不喝茶的人也爱喝咖啡。这样来看，其实是否购买咖啡，与有没有购买茶叶并没有关联，两者是相互独立的，其提升度为90%/((450+450)/1000)=1
- **关联规则的提升度**反映了关联规则中的A与B的相关性，提升度>1且越高表明正相关性越高，提升度<1且越低表明负相关性越高，提升度=1表明没有相关性。

2 基本概念与解决方法

Transaction 1	🍎 🍺 🍚 🍗
Transaction 2	🍎 🍺 🍚
Transaction 3	🍎 🍺
Transaction 4	🍎 🍐
Transaction 5	💉 🍺 🍚 🍗
Transaction 6	💉 🍺 🍚
Transaction 7	💉 🍺
Transaction 8	💉 🍐

置信度：$Confidence = \frac{freq(A,B)}{freq(A)}$

支持度：$Support = \frac{freq(A,B)}{N}$

提升度：$Lift = \frac{Support}{Support(A) \times Support(B)}$

Support {🍎} = $\frac{4}{8}$

Confidence {🍎 → 🍺} = $\frac{\text{Support } \{🍎, 🍺\}}{\text{Support } \{🍎\}}$ =3/4

Lift {🍎 → 🍺} = $\frac{\text{Support } \{🍎, 🍺\}}{\text{Support } \{🍎\} \text{ x Support } \{🍺\}}$

2 基本概念与解决方法

【定义6】 **项集的支持度**

- 项集(Itemset)：包含0个或多个项的集合
- 项集的支持度计数：包含特定项集的事务个数
- 对于项集X，$X \subset I$，设$count(X \subseteq T)$为事务集D中X的支持度计数，则项集X的**支持度**(support)为：

$$support(X) = count(X \subseteq T) / |D|$$

- 项集X的支持度$support(X)$就是项集X在事务集D中出现的概率，能够反映X的重要性。
- 由于关联规则必须由频繁项集产生，因此**关联规则的支持度**就是**频繁项集的支持度**。

2 基本概念与解决方法

交易号 TID	顾客购买商品 Items
T1	bread cream milk tea
T2	bread cream milk
T3	cake milk
T4	milk tea
T5	bread cake milk
T6	bread tea
T7	beer milk tea
T8	bread tea
T9	bread cream milk tea
T10	bread milk tea

- 例如：对于2-项集X={bread, milk}，它出现在T_1, T_2, T_5, T_9和T_{10}中，$support(X)$=5/10=0.5。

2 基本概念与解决方法

【定义7】 **项集的最小支持度与频繁项集**

给定全局项集I和数据库D，

- 用于发现关联规则的项集必须满足的最小支持度的阈值，称为项集的最小支持度(Minimum Support)，记为sup_{min}。
  - 从统计意义来说，sup_{min}表示用户关心的关联规则必须满足的最低重要性。
  - 只有满足最小支持度的项集才能产生关联规则。
- 支持度大于或等于sup_{min}的项集称为**频繁项集**，简称**频繁集**，反之则称为**非频繁集**。
  - k-项集如果满足sup_{min}，可称为k-频繁集，记作L_k。

2 基本概念与解决方法

- 若用户将sup_{min}设为**0.5**，则：
  - 对于2-项集X={bread, milk}，它是一个2-频繁集；
  - 对于2-项集Y={cake, milk}，它不是2-频繁集。

交易号 TID	顾客购买商品 Items
T1	bread cream milk tea
T2	bread cream milk
T3	cake milk
T4	milk tea
T5	bread cake milk
T6	bread tea
T7	beer milk tea
T8	bread tea
T9	bread cream milk tea
T10	bread milk tea

2 基本概念与解决方法

- 若用户将sup_{min}设为**0.1**，则：
  - 对于2-项集X={bread, milk}，它是一个2-频繁集；
  - 对于2-项集Y={cake, milk}，它也是2-频繁集。

交易号 TID	顾客购买商品 Items
T1	bread cream milk tea
T2	bread cream milk
T3	cake milk
T4	milk tea
T5	bread cake milk
T6	bread tea
T7	beer milk tea
T8	bread tea
T9	bread cream milk tea
T10	bread milk tea

2 基本概念与解决方法

【定义8】**关联规则的最小支持度和最小可信度**

- **关联规则的最小支持度**(Minimum Support)表示关联规则需要满足的最低支持度，记为sup_{min}。
- **关联规则的最小可信度**(Minimum Confidence)表示关联规则需要满足的最低可信度，记为$conf_{min}$。

2 基本概念与解决方法

【定义9】 **强关联规则**

- 如果关联规则R：$A \Rightarrow B\ [s, c]$**同时满足**：

  $$support(A \Rightarrow B) = s \geq sup_{min}$$

  $$confidence(A \Rightarrow B) = c \geq conf_{min}$$

  则关联规则R为**强关联规则**，否则为**弱关联规则**。
- 在挖掘关联规则时，产生的关联规则要经过sup_{min}和$conf_{min}$的度量，筛选出来的**强关联规则**才能用于指导决策。

2 基本概念与解决方法

【定义9】 **强关联规则**

若设定$sup_{min} = 0.5$，$conf_{min} = 0.7$，

- 关联规则R_1: {bread} $\Rightarrow$ {milk}是一个强关联规则。
  - $support(R_1) = 0.5$
  - $confidence(R_1) = 5/7$
- 关联规则R_2: {milk} $\Rightarrow$ {bread}是一个弱关联规则。
  - $support(R_2) = 0.5$
  - $confidence(R_2) = 5/8$

2 基本概念与解决方法

- 挖掘关联规则R：$A \Rightarrow B\ [s, c]$的问题可以分两个子问题：
  - 发掘频繁项集：也就是事务支持度s大于预先给定的最小阈值的项的集合。该项集的每一个出现的频繁性 $\geq sup_{min}$
  - 产生关联规则：使用频繁项集来产生数据库中置信度c大于预先给定的最小阈值的关联规则（$\geq conf_{min}$）。

2 基本概念与解决方法

- 主要挑战：会产生大量满足sup_{min}的项集，尤其当sup_{min}设置得低的时候
  - e.g. 一个长度为100的频繁项集$\{a_1, a_2, \ldots, a_{100}\}$包含的频繁项集的总个数为

$$C_{100}^{1} + C_{100}^{2} + \ldots + C_{100}^{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}$$

算力问题：无法存储和计算

寻找

保持了频繁项集的完整信息的简约表示

2 基本概念与解决方法

【定义10】 **闭频繁项集**

对所有项的集合S，非空子集X为S的一个项集或模式，

- 封闭项集：项集X是封闭的，当且仅当找不到X在S中的任何真超项集Y，满足

$$support(Y) = support(X)$$

- 闭频繁项集：项集X 在S中是闭的和频繁的。

在项集X的支持度约束下，你找不到比项集X更大的项集！

2 基本概念与解决方法

【定义11】 **极大频繁项集**

对所有项的集合S，非空子集X为S的一个项集或模式，

- 极大频繁项集（或极大项集）：如果项集X是S中频繁的，并且不存在超集Y，使得Y在S中是频繁的。

闭频繁项集和极大频繁项集

TID	购买的item
1	C,D
2	A,B
3	A,B,C
4	A,B,C

$S = \{A, B, C, D\}$， $sup_{min}=2$，

- 1)频繁项集$\mathcal{F}$:

  $\mathcal{F}$={<A>:3, <B>:3, <C>:3,<A,B>:3,<A,C>:2,<B,C>:2,<A,B,C>:2}

- 2)频繁闭项集C:

  $\mathcal{C}$={<C>:3,<A,B>:3),<A,B,C>:2}

- 3)极大频繁项集M:

  $\mathcal{M}$={<A,B,C>:2)}

- 观察：$\mathcal{M} \subseteq \mathcal{C} \subseteq \mathcal{F}$，然而$\mathcal{C}$可以显著减少模式数量，保持了频繁项集的完整信息，由频繁闭项集可获得所有的频繁项集和它们的支持度，因此挖掘闭项集是一个好的选择。

关联分析: 理论与算法

- 1 概述
- 2 基本概念与解决方法
- 3 经典算法
  - Apriori算法
  - FP-growth算法

关联规则挖掘的过程

- **关联规则挖掘**包含以下两个步骤：
  - 首先，找出所有**频繁集**；
  - 其次，由频繁集产生**强关联规则**。
- R. Agrawal(阿戈沃)等人于1993年提出了挖掘顾客交易数据中项集间的关联规则问题，并且设计了一个算法：**Apriori算法**。
  - Apriori算法通过**多次扫描**数据集，找出所有频繁集，然后用这些频繁集产生强关联规则。

关联规则挖掘的过程

- **关联规则挖掘**包含以下两个步骤：
  - 首先，找出所有**频繁集**；
  - 其次，由频繁集产生**强关联规则**。
- J. Han等人提出了一种不产生候选集来挖掘频繁集的方法——**FP-growth算法**。
  - FP-growth算法在**扫描两遍**数据集之后，利用一棵频繁模式树(FP-tree)来表示频繁集，进而再确定相应的强关联规则。

Apriori算法：频繁项集生成

- **定理1**（*Apriori* 属性1） 如果一个项集X 是频繁的，那么它的所有非空子集一定也是频繁的。

证明：设X是一个项集，事务数据库T 中支持X的元组数为s。对X的任一非空子集为Y，设T中支持Y的元组数为s’。

根据项集支持数的定义，易知：支持X 的元组一定支持Y，所以$s' \geq s$，即$support(Y) \geq support(X)$。

按照假设：项集X 是频繁项集，即$support(X) \geq sup_{min}$，故$support(Y) \geq support(X) \geq sup_{min}$，因此$Y$是频繁项集。 □

Apriori算法：频繁项集生成

- **定理1**（*Apriori* 属性1） 如果一个项集X 是频繁的，那么它的所有非空子集一定也是频繁的。

- 例如：

  假定$\{c, d, e\}$是频繁项集，则任何一个包含项集$\{c, d, e\}$的事务一定包含它的子集$\{c, d\}$，$\{c, e\}$，$\{d, e\}$，$\{c\}$，$\{d\}$和$\{e\}$。即：如果$\{c, d, e\}$是频繁的，则它的所有子集一定也是频繁的。

直观的结果！

Apriori算法：频繁项集生成

- 超集：一个集合*S*2中的每一个元素都在集合*S*1中，且集合*S*1中可能包含*S*2中没有的元素，则集合*S*1就是*S*2的一个超集。

- **定理2**（*Apriori* 属性2）如果一个项集*X*是非频繁的，那么它的所有超集一定也是非频繁的。

  Why?

  证明： <略>。 □

Apriori算法：频繁项集生成

- Apriori算法通过**迭代**来穷举出数据集中的所有频繁集。
- 算法过程：
  - 首先，产生1-频繁集L_1；
  - 其次，在L_1上通过**连接**和**修剪**产生2-频繁集L_2；
  - 依次类推，可在L_k上通过**连接**和**修剪**产生(k+1)-频繁集L_{k+1}；
  - 最后，直到无法产生新的频繁集为止。

Apriori算法：频繁项集生成

输入： 事务数据集D，最小支持度阈值sup_{min}。

输出： 可以产生关联规则的所有频繁集L。

C_k：k-候选频繁集。

L_k：k-频繁集。

(1) L_1 = find_frequent_1_itemset(D); //发现1-频繁集

(2) for(k = 2; $L_{k-1} \neq \emptyset$; k++) {

(3) C_k = apriori_gen(L_{k-1}, sup_{min}); //根据(k-1)-频繁集产生k-候选集

(4) for each $t \in D$ { //扫描数据集，确定每个候选集的支持度

(5) C_t = subset(C_k, t); //获得t所包含的候选集

(6) for each $c \in C_t$ $c.count$++; }

(7) }

(8) $L_k = \{c \in C_k \mid c.count > sup_{min}\}$;

(9) return $L = \cup\{L_k\}$;

Apriori算法：频繁项集生成

- apriori_gen(L_{k-1}, sup_{min})算法

输 入：上次扫描的结果L_{k-1}，最小支持度阈值sup_{min}。

输 出：候选频繁集C_k。

(3.1) for each $l_1 \in L_{k-1}$

(3.2) for each $l_2 \in L_{k-1}$

(3.3) if(($l_1[1]=l_2[1]$)^...^($l_1[k-2]=l_2[k-2]$)^($l_1[k-1]=l_2[k-1]$) {

(3.4) c = $l_1[1] \oplus l_2[1]$; //将只差一项的两个项集连接起来

(3.5) if **has_infrequent_subset(c, L_{k-1})**

(3.6) delete c; //删除不可产生频繁集的候选

(3.7) else $C_k = C_k \cup \{c\}$;

(3.8) }

(3.9) return C_k;

Apriori算法：频繁项集生成

- has_infrequent_subset(c, L_{k-1})算法

输 入：本次扫描产生的C_k的每个子集c，上次扫描产生的L_{k-1}。

输 出：c是否从C_k中删除。

(3.5.1) for each (k-1)-subset s of c

//根据Apriori算法的性质：候选集的子集一定是频繁的。

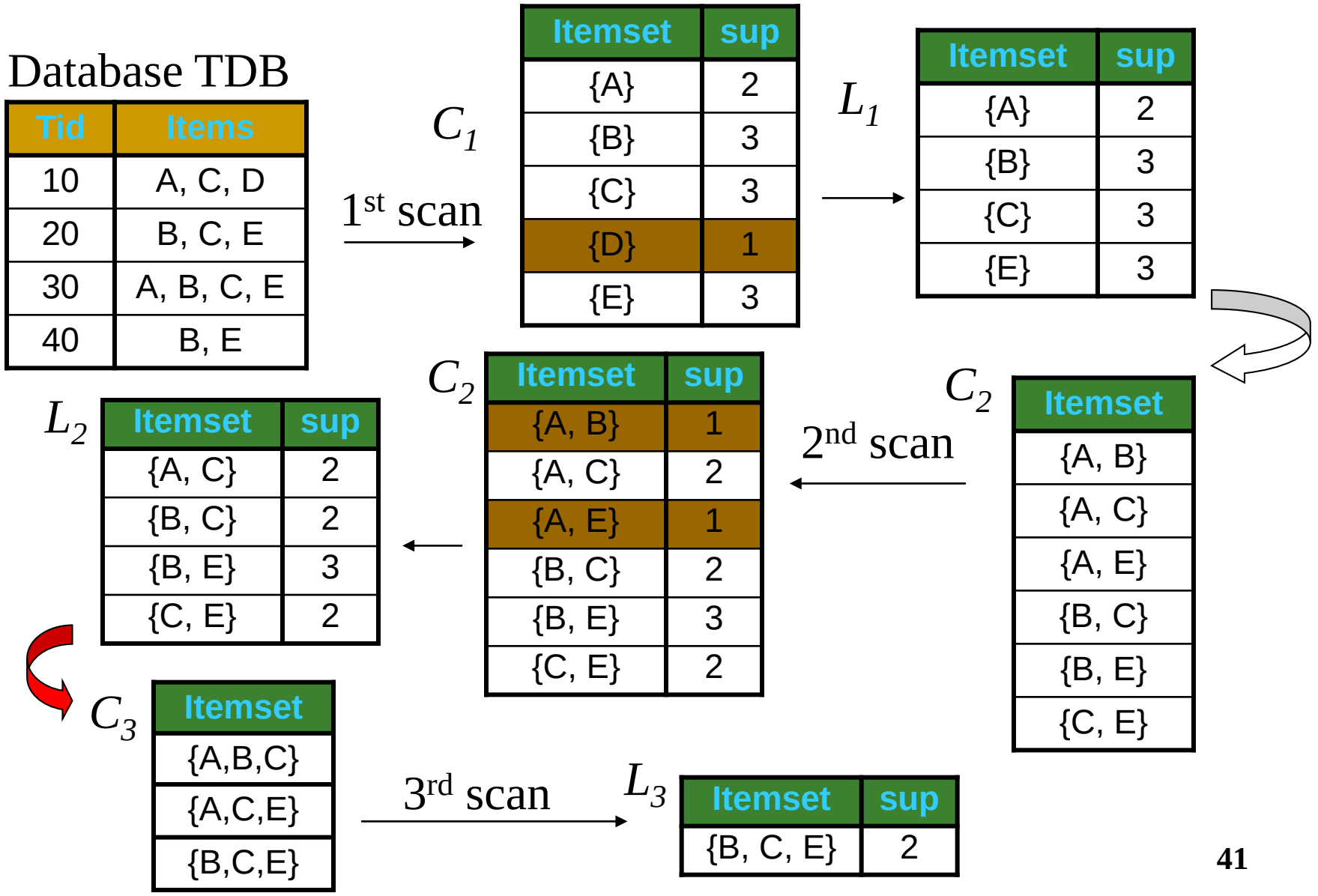
(3.5.2) if $s \notin L_{k-1}$

return TRUE; //删除不是频繁集的子集

else

return FALSE;

Apriori算法：频繁项集生成 (sup_{min}=2)



Apriori算法：频繁项集生成

- **连接**：用 L_{k-1}自连接得到候选k-项集C_k。只相差一个项目的两个项集才能进行连接（集合“并”操作）。
  - 例如：由L_2生成C_3的过程中，L_2中的$\{A,C\}$和$\{B,C\}$只相差一个项目，因此它们可以连接生成$\{A,B,C\}$。
  - L_2中的$\{A,C\}$和$\{B,E\}$无法进行连接。
- **修剪**：去除子集不是频繁集的项集。
  - Apriori算法的性质：**一个k-项集，如果它的一个k-1项集（子集）不是频繁的，那么它本身也不可能是频繁的。**
  - 例如：虽然L_2中的$\{A,C\}$和$\{B,C\}$可以连接生成$\{A,B,C\}$，但是由于$\{A,B,C\}$的子集$\{A,B\}$不是频繁集(不在L_2中)，因此，需要从C_3中删除$\{A,B,C\}$ 。

Apriori算法：规则生成

- 由频繁集产生关联规则的步骤：
  - 首先，设置最小支持度sup_{min}，利用Apriori算法产生所有频繁集；
  - 其次，设置最小可信度$conf_{min}$，并根据不同需要在各频繁集中确定强关联规则（通常是在最高频繁集上确定强关联规则）；
  - 最后，获得强关联规则，用于决策。

Apriori算法：规则生成

- 在得到了所有频繁项集后，可以按照下面的步骤生成关联规则：
  - 对于每一个频繁项集l，生成其所有的非空子集；
  - 对于l的每一个非空子集x，计算$confidence(x \Rightarrow (l - x))$，如果$confidence(x \Rightarrow (l - x)) \geq conf_{min}$，那么规则“$x \Rightarrow (l - x)$”成立。

$$confidence\ (A \Rightarrow B)\ =\ P(B \mid A) = P(AB) / P(A)$$

$$confidence\ (x \Rightarrow l - x)\ =\ P(l - x \mid x) = P(l) / P(x)$$

Apriori算法：规则生成

- 例：对于3-频繁集{*B*,*C*,*E*}来说，它的非空子集包括{{*B*}, {*C*}, {*E*}, {*B*,*C*}, {*B*,*E*}, {*C*,*E*}}。
  - **关联规则{*B*,*C*}→{*E*}**的置信度 *conf*=2/2=100%。
  - **关联规则{*B*,*E*}→{*C*}**的置信度 *conf*=2/3=66.7%。
  - 当最小置信度设置为80%时，关联规则**{*B*,*C*}→{*E*}**是强关联规则，而关联规则**{*B*,*E*}→{*C*}**不是。
  - 当最小置信度设置为50%时，关联规则**{*B*,*C*}→{*E*}**和**{*B*,*E*}→{*C*}**都是强关联规则。

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

Apriori算法：规则生成算法的优化

- **定理3**：设X_s是项集X的一个子集，若规则$X \Rightarrow (l\text{-}X)$不是一条强规则，那么，$X_s \Rightarrow (l\text{-}X_s)$也一定不是强规则。

证明：由于$support(X_s) \geq support(X)$，故

$$confidence(X_s \Rightarrow (l\text{-}X_s)) = support(l)/support(X_s)$$
$$\leq support(l)/support(X) = confidence(X \Rightarrow (l\text{-}X))$$

又，$X \Rightarrow (l\text{-}X)$不是强规则，即

$$confidence(X \Rightarrow (l\text{-}X)) < conf_{min},$$

所以，

$$confidence(X_s \Rightarrow (l\text{-}X_s)) \leq confidence(X \Rightarrow (l\text{-}X)) < conf_{min}$$

即，$X_s \Rightarrow (l\text{-}X_s)$不是强规则。 □

意义：利用已知结果排除一些肯定不是强规则的测试。

如：若$BC \Rightarrow AD$不是强规则，可断定形如$B \Rightarrow *$和$C \Rightarrow *$一定不是强规则。

Apriori算法：规则生成算法的优化

- **定理4**：设X_s是项集X的一个子集，若规则$Y \Rightarrow X$是一条强规则，那么，$Y \Rightarrow X_s$也一定是强规则。

证明：由支持度的定义，$support(X_s \cup Y) \geq support(X \cup Y)$，

故$confidence(Y \Rightarrow X) = support(X \cup Y) / support(Y)$

$\leq support(X_s \cup Y) / support(Y) = confidence(Y \Rightarrow X_s)$

又，$Y \Rightarrow X$是强规则，即

$confidence(Y \Rightarrow X) \geq conf_{min}$，

所以，

$confidence(Y \Rightarrow X_s) \geq confidence(Y \Rightarrow X) \geq conf_{min}$

即，$Y \Rightarrow X_s$也是强规则。 □

意义：利用已知结果避免一些肯定是强规则的测试。

Apriori的性能—够快了吗? 瓶颈

- ***Apriori*算法的核心:**
  - 用频繁的$(k-1)$-项集生成候选的频繁 *k-项集*
  - 用数据库扫描和模式匹配计算候选集的支持度
- ***Apriori* 的瓶颈: 候选集生成**
  - **巨大的候选集:**
    - 10^4 个频繁1-项集要生成 10^7 个候选 2-项集
    - 要找尺寸为100的频繁模式，如 $\{a_1, a_2, \ldots, a_{100}\}$, 你必须先产生$2^{100} \approx 10^{30}$ 个候选集
  - **多次扫描数据库:**
    - 如果最长的模式是*n*的话，则需要 $(n+1)$ 次数据库扫描

提高Apriori效率的方法

- 基于*Hash*的项集计数：在一个*Hash*桶内支持度小于最小支持度的*k*-项集不可能是全局频繁的。
- 减少交易记录：不包含任何频繁*k*-项集的交易也不可能包含任何大于*k*的频繁集，下一步计算时删除这些记录。
- 划分(Partition)：一个项集要想在整个数据库中是频繁的，那么它至少在数据库的一个分割上是频繁的。
- 抽样(Sampling)：使用小的支持度+完整性验证方法。在小的抽样集上找到局部频繁项集，然后在全部数据集找频繁项集。
- 动态项集计数：在添加一个新的候选集之前，先估计一下是不是他的所有子集都是频繁的。

提高Apriori效率的方法

- 划分(Partition)：一个项集要想在整个数据库中是频繁的，那么它至少在数据库的一个分割上是频繁的。
- 方法：把数据库从逻辑上划分为若干个互不相交的块，对每个块应用挖掘算法生成局部的频繁项集，再把这些局部的频繁项集作为候选的全局频繁项集，通过测试它们的支持度来得到最终的全局频繁项集。
- 优势：
  - 合理利用内存空间：数据分割将大数据集分成小的块，为块内数据一次性导入主存提供机会。
  - 支持并行挖掘算法：每个分块的局部频繁项目集是独立生成的，因此提供了开发并行数据挖掘算法的良好机制。

提高Apriori效率的方法：划分(Partition)

- 定理5：设数据集D被划分成分块D_1、D_2、…、D_n，全局最小支持度为*minisupport*，最小支持计数为*minisup_count*。如果一个数据分块D_i的局部最小支持计数记为$minisup_count_i(i=1,2,...,n)$，那么，若局部最小支持计数$minisup_count_i$按如下方法生成:

$$minisup\_count_i = minisup\_count * || D_i || / || D ||, (i=1,2,...,n)$$

则可以保证所有的局部频繁项目集是全局频繁项目集的候选，亦即所有的局部频繁项目集涵盖全局频繁项目集。

证明：仅需证明：如果一个项目集*IS*在所有的数据分块内都不（局部）频繁，那么，项目集*IS*在整个数据集中亦不（全局）频繁。

若*IS*在所有的数据分块内都不（局部）频繁，即

$$sup\_count_i(IS) < minisup\_count_i(i=1,2,...,n)\ ,$$

其中$sup_count_i(IS)$是项目集*IS*在分块D_i的局部支持计数，则

$$\begin{aligned} sup\_count(IS) &= \sum sup\_count_i(IS) \\ &< \sum minisup\_count_i = \sum minisup\_count * || D_i || / || D || \\ &= minisup\_count *(\sum || D_i ||) / || D || \\ &= minisup\_count \end{aligned}$$

因此，*IS*在整个数据集中亦不（全局）频繁。 □

提高Apriori效率的方法：Sampling

- 抽样(Sampling)：使用小的支持度+完整性验证方法。在小的抽样集上找到局部频繁项集，然后在全部数据集找频繁项集。
- 优势：简单，可显著降低扫描数据库的代价。
- 问题：1）如何选取抽样数据？2）如何防止数据扭曲(Data Skew)：结果偏差过大

提高Apriori效率的方法：Sampling

- 从本质上说，抽样可以减少存储量和计算量，提高效率
- 问题：如何抽样以建立能够反映整个数据分布的模型？

若大小为N的总体样本方差为 δ^2，那么，从这个总体样本抽取的大小为n的简单随机样本（不放回抽样）的方差

$$\delta_n^2 = \frac{\delta^2}{n}(1-\frac{n}{N})$$

一般，$n<<N$，故

$$\delta_n^2 \approx \frac{\delta^2}{n}$$

- 关键是找到 δ^2 的估计方法！

$$\delta^2 = \frac{\sum(x(i)-\bar{x})^2}{n-1}$$

提高Apriori效率的方法

- 动态项集计数：在添加一个新的候选集之前，先估计一下是不是它的所有子集都是频繁的。

- 减少交易记录：不包含任何频繁*k*-项集的交易也不可能包含任何大于*k*的频繁集，下一步计算时删除这些记录。

第三章 关联分析: 理论与算法

- **3.1 概述**
- **3.2 基本概念与解决方法**
- **3.3 经典算法**
  - **Apriori算法**
  - **FP-growth算法**

FP-growth算法

- FP-tree： Frequent Pattern Tree
- 频繁模式增长(Frequent Pattern Growth, FP-growth)算法使用一种被称为**FP-tree**的数据结构，并且采用分而治之的(divide and conquer)策略，该算法是对Apriori 方法的改进，无需产生候选频繁集(C_k)就能得到全部的频繁集(L_k)。
  - 在经过第一遍扫描之后，把数据库中的频繁集压缩进一棵频繁模式树（FP-tree），同时依然保留其中的关联信息；
  - 将FP-tree分化成一些条件库，每个库和一个长度为1的频繁集相关，然后再对这些条件库分别进行挖掘。

FP-growth算法

【定义12】 FP-tree

- 频繁模式树FP-tree是一个树形结构。
- 包括：
  - 一个**频繁项**组成的头表；
  - 一个标记为“*null*”的根节点；
  - 子节点为一个**项前缀子树**的集合。

【定义13】 频繁项

- 如果单个项目的支持度大于最小支持度sup_{min}，则称其为频繁项(Frequent Item)。

FP-growth算法

【定义14】 频繁项头表

- 频繁项头表(Head Table)的每个表项都由三个域组成：项目名称item-name、出现次数frequency和指针head。
  - 与倒排索引中的词典类似。
  - 其中，head指向FP-tree中具有与该表项相同item-name的第一个节点。

FP-growth算法

【定义15】 项前缀子树

- 每个项前缀子树(Item Prefix Subtree)的节点有三个域：item-name，count，node_link。
  - item-name记录了该节点所代表的项的名字。
  - count记录了所在路径代表的交易中包含此节点项目的交易个数。
  - node_link指向下一个具有同样的item-name域的节点，要是没有这样一个节点，就为*null*。

FP-growth算法(第一次扫描数据集)

- 根据一个数据集*D*，建立一棵FP-tree。
  - 第一步：扫描一遍数据集*D*，得到1-频繁项集*F*和每个频繁项的支持度。并将*F*按支持度递降排序，存入列表*L*中。

TID	Items
100	{*f, a, c, d, g, i, m, p*}
200	{*a, b, c, f, l, m, o*}
300	{*b, f, h, j, o*}
400	{*b, c, k, s, p*}
500	{*a, f, c, e, l, p, m, n*}

$sup_{min} = 3$

Item	counts
f	4
c	4
a	3
b	3
m	3
p	3

TID	Items	(ordered) frequent items
100	{*f, a, c, d, g, i, m, p*}	{*f, c, a, m, p*}
200	{*a, b, c, f, l, m, o*}	{*f, c, a, b, m*}
300	{*b, f, h, j, o*}	{*f, b*}
400	{*b, c, k, s, p*}	{*c, b, p*}
500	{*a, f, c, e, l, p, m, n*}	{*f, c, a, m, p*}

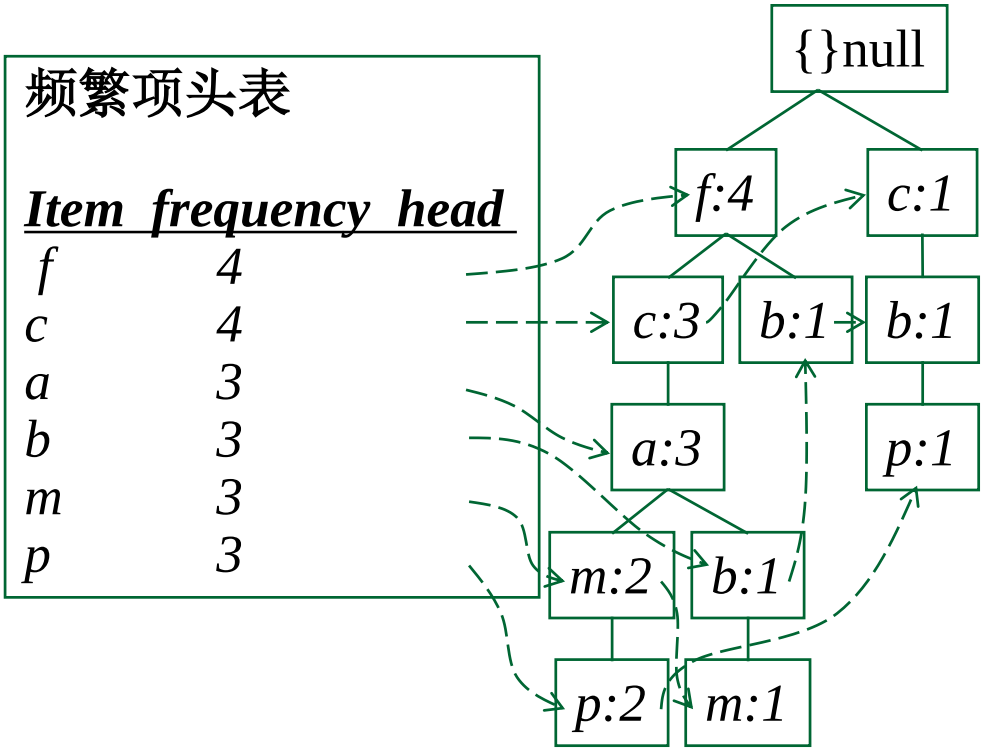
FP-growth算法(第二次扫描数据集)

- 根据一个数据集*D*，建立一棵FP-tree。
  - 第二步：
  1. 创建树的根节点，用*null*标记；
  2. 将每个交易中的项按支持度递减排列，并对每个交易创建一个分枝；
     - **例如：为第一个交易{*f, c, a, m, p*}构建一个分枝。**
  3. 当为一个交易增加分枝时，沿共同前缀上的每个节点的计数加1，为跟随前缀后的项创建节点并连接。
     - **例如：将第二个交易{*f, c, a, b, m*}加到树上时，需要为*f*，*c*，*a*各增计数1，然后为{*b, m*}创建分枝。**
  4. 创建一个**频繁项头表**，以方便遍历，每个项通过一个指针指向它在树中出现的位置。

FP-growth算法——例子

TID	*Items bought*	*(ordered) frequent items*
100	{*f, a, c, d, g, i, m, p*}	{*f, c, a, m, p*}
200	{*a, b, c, f, l, m, o*}	{*f, c, a, b, m*}
300	{*b, f, h, j, o*}	{*f, b*}
400	{*b, c, k, s, p*}	{*c, b, p*}
500	{*a, f, c, e, l, p, m, n*}	{*f, c, a, m, p*}

$sup_{min} = 3$



FP-growth算法—FP树结构的好处

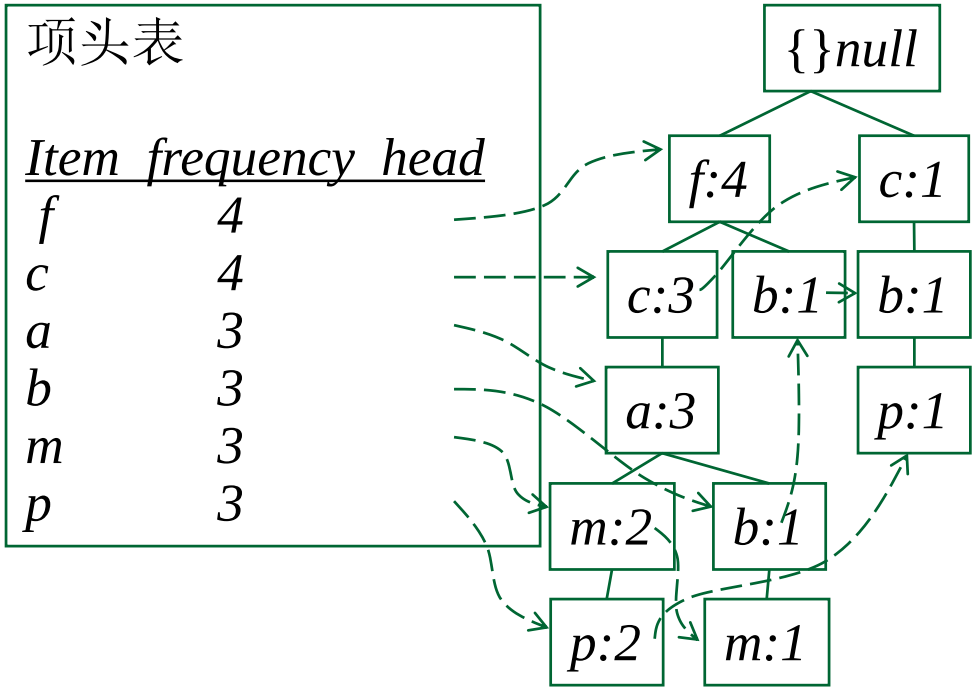
- 完整性
  - 不会打破任何事务数据中的长模式
  - 为频繁模式的挖掘保留了完整的信息
- 紧凑性
  - 减少了不相关的信息—非频繁的项被删除
  - 按频率递减排列—使得更频繁的项更容易在树结构中被共享
  - 数据量比原数据库要小

FP-growth算法— FP树的挖掘

- FP树的挖掘步骤：
  - 由长度为1的频繁模式（初始后缀模式）开始，构造它的条件模式基（一个“子数据库”，由FP树中与后缀模式一起出现的前缀路径集组成）。
  - 构造该初始后缀模式的条件FP树，并递归的在该树上实现挖掘。模式增长通过后缀模式与条件FP树产生的频繁模式连接实现。

FP树挖掘—从FP树到条件模式基

- 从项头表开始挖掘，由频率低的节点开始
- 沿每个（频繁）项的链接来遍历FP树
- 通过积累该项的前缀路径来形成一个条件模式基



Conditional pattern bases

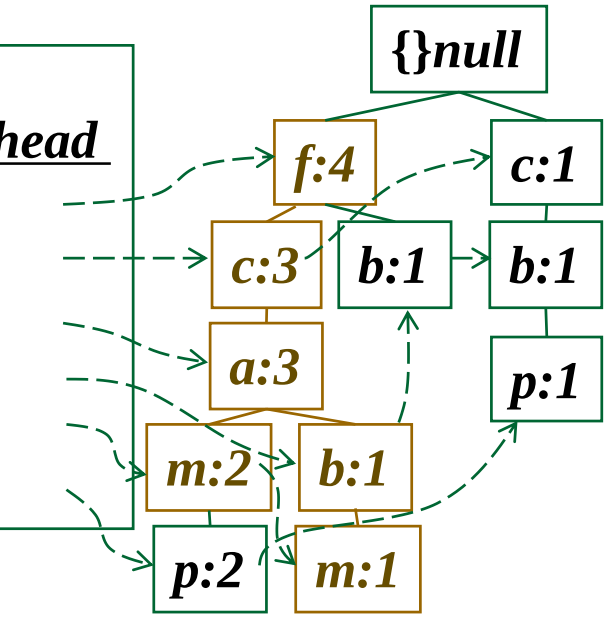
item	*cond. pattern base*
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*

FP树挖掘—构建条件FP树

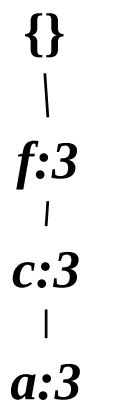
- 对每个条件模式基
  - 为基中的每一项累积计数
  - 为模式基中的频繁项构建FP树

项头表

Item	frequency	head
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



m-条件模式基:
fca:2, fcab:1



所有关于*m*的频繁模式：
m,
fm, cm, am,
fcm, fam, cam,
fcam

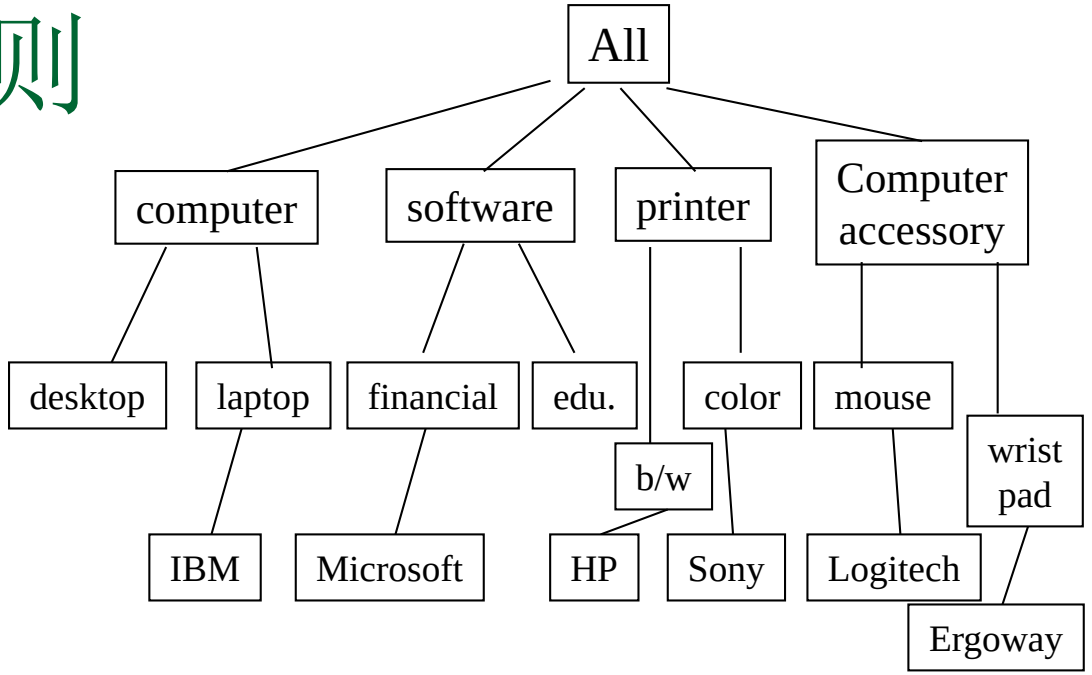
m-条件FP树

总结

- FP-Growth算法的优点
  - 与Apriori算法相比，只需对数据库进行两次扫描
  - 不需要对项目进行配对，因此速度更快
  - 数据库存储在内存中的压缩版本中
  - 对长、短频繁模式的挖掘具有高效性和可扩展性
- FP-Growth算法的缺点
  - FP-Tree比Apriori更麻烦，更难构建
  - 可能很耗资源
  - 当数据库较大时，算法可能不适合共享内存

多层关联规则

- 数据项中经常会形成概念分层
- 底层的数据项，其支持度往往也较低
  - 这意味着挖掘底层数据项之间的关联规则必须定义不同的支持度



TID	Items
T1	{IBM D/C, Sony b/w}
T2	{Ms. edu. Sw., Ms. fin. Sw.}
T3	{Logi. mouse, Ergoway wrist pad}
T4	{IBM D/C, Ms. Fin. Sw.}
T5	{IBM D/C}

多层关联规则

- 在适当的等级挖掘出来的数据项间的关联规则可能是非常有用的
- 事务数据库中的数据一般是根据维和概念分层来进行储存的
  - 这为从事务数据库中挖掘不同层次的关联规则提供了可能
- 在多个抽象层挖掘关联规则，并在不同的抽象层进行转化，是数据挖掘系统应该提供的能力

挖掘多层关联规则的方法

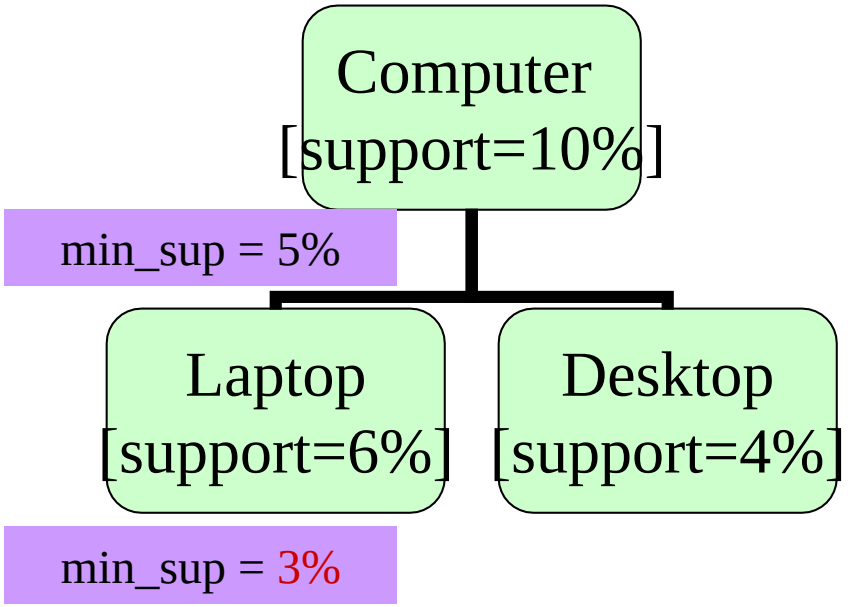
- 通常，多层关联规则的挖掘还是使用置信度－支持度框架，可以采用自顶向下策略
  - 注意：概念分层中，一个节点的支持度肯定不小于该节点的任何子节点的支持度
  - 由概念层1开始向下，到较低的更特定的概念层，对每个概念层的频繁项计算累加计数
  - 每一层的关联规则挖掘可以使用*Apriori*等多种方法
  - 例如：
    - 先找高层的关联规则：*computer* => *printer* [20%, 60%]
    - 再找较低层的关联规则：*laptop* => *color printer* [10%, 50%]

多层关联—一致支持度

- 一致支持度：对所有层都使用一致的最小支持度
  - 优点：搜索时容易采用优化策略，即一个项如果不满足最小支持度，它的所有子项都可以不用搜索
  - 缺点：最小支持度值设置困难
    - 太高：将丢掉出现在较低抽象层中有意义的关联规则
    - 太低：会在较高层产生太多的无兴趣的规则

多层关联—递减支持度

- 使用递减支持度，可以解决使用一致支持度时在最小支持度值上设定的困难
- 递减支持度：在较低层使用递减的最小支持度
  - 每一层都有自己的一个独立的最小支持度
  - 抽象层越低，对应的最小支持度越小



多层关联—基于分组的支持度

- 用户和专家清楚哪些分组比其他分组更加重要，在挖掘多层关联规则时，使用用户指定的基于项或者基于分组的最小支持度阈值
  - 对*laptop_computer*或者*flash_drives*设置特别低的支持度阈值，以便特别关注这类商品的管理模式

冗余的多层关联规则检查

- 挖掘多层关联规则时，由于项间的“祖先”关系，有些发现的规则将是冗余的
  - 例如：
    - *buys(X, “desktop computer”) => buys(X, “HP printer”) [8%, 70%]* (1)
    - *buys(X, “IBM desktop computer”) => buys(X, “HP printer”) [2%, 72%]* (2)
- 规则(1)是规则(2)的“祖先”
- 如果规则(2)中的项用它在概念分层中的“祖先”代替，能得到(1)，而且(1)的支持度和置信度都接近“期望”值，则(2)不是有趣的。