第五章 相关性与关联规则

内容提要

- 关联和相关性
- 高效的可扩展的频繁项集挖掘方法
- 挖掘的几种关联规则
- 从关联挖掘到相关性分析
- 基于约束的关联挖掘

关联规则是什么

- 关联规则（Association Rules）是反映一个事物与其他事物之间的相互依存性和关联性，如果两个或多个事物之间存在一定的关联关系，那么，其中一个事物就能通过其他事物预测到。关联规则是数据挖掘的一个重要技术，用于从大量数据中挖掘出有价值的数据项之间的相关关系。
- 典型案例：啤酒与尿布。

购物篮问题

- 一个大的项目集，例如，超市里售卖的东西.
- 一个大的篮子集，每个篮子都是一个小的项目集，例如，一个顾客一天买的东西.
- 最简单的问题：找到这个篮子里经常出现的项目集.
- 项目集I的支持数= 含有I里所有项目的篮子数量.
- 给定一个最小支持数门槛s，在≥ s个篮子里出现的项目的集合，称为频繁项集

购物篮问题

- Items={milk, coke, pepsi, beer, juice}.
- Support s= 3 baskets.

$B_1 = \{m, c, b\}$ $B_2 = \{m, p, j\}$

$B_3 = \{m, b\}$ $B_4 = \{c, j\}$

$B_5 = \{m, p, b\}$ $B_6 = \{m, c, b, j\}$

$B_7 = \{c, b, j\}$ $B_8 = \{b, c\}$

- Frequent itemsets: {m}, {c}, {b}, {j}, {m, b}, {c, b}, {j, c}.

潜在应用

- **真正的市场篮子：连锁店保存有关客户同时购买的海量信息。**
  - 讲述了典型的客户浏览商店，让他们定位诱人的项目.
  - 建议搭配的“招数”，例如，尿布的打折销售并提高啤酒的价格.
  - 篮子数据分析，交叉营销，目录设计，销售活动分析

潜在应用

- “Baskets” = 评论; “items” = 那些从互联网抓取信息的话.
  - 让我们找到那些经常一起出现的热点评论，也就是，评论情感分析.
- “Baskets” = 信用卡账单, “items” = 商业银行数据库的交易.
  - 经常一起出现的项目可以表明顾客的消费模式，也就是，顾客行为分析.
- “Baskets” = 网页; “items” = 浏览的页面.
  - 经常一起出现的项目可以表明网友的浏览模式，即互联网行为分析.

重要提示

- “市场篮子”是将任何两个概念之间的多对多关系模型化的一个抽象：“项目”和“篮子。”
  - 项目不需要被包含在篮子里.
- 唯一不同的是，我们数与一个篮子相关的同时出现的项目，而不是相反.
- 规模问题
  - 沃尔玛卖100,000个项目并且储存上亿个篮子.
  - 网络有超过100,000,000单词和上亿网页

什么是频繁模式分析?

- 频繁模式:在数据集中经常出现的模式（项目集，子序列，子结构等
- 最先由 Agrawal, Imielinski, 和 Swami [AIS93] 在文章频繁项集和关联规则挖掘中提出
- 揭示了数据集的一个内在的重要的特性
- 形成了许多重要的数据挖掘任务的基础
  - 关联，相关和因果关系分析
  - 顺序，结构（例如，子图）模式
  - 时空，多媒体，时间序列，数据流模式分析
  - 分类：关联分类
  - 聚类分析：频繁模式聚类
  - 数据仓库：冰山立方体和立方梯度
  - 语义数据压缩：成簇

频繁模式与关联规则-事务数据库

- 设I={i_1，i_2，…，i_m}是一个项目集合，事务数据库D={t_1，t_2，…，t_n}是由一系列具有唯一标识TID的事务组成，每个事务t_i（i=1，2，…，n）都对应I上的一个子集。
- 一个事务数据库可以用来刻画：
  - 购物记录： I是全部物品集合， D是购物清单，每个元组t_i是一次购买物品的集合（它当然是I的一个子集）。
  - 其它应用

支持度与频繁项目集

- **定义（项目集的支持度）**.给定一个全局项目集I和数据库D，一个项目集$I_1 \subseteq I$在D上的**支持度（Support）**是包含I_1的事务在D中所占的百分比：$support(I_1) = \|\{t \in D \mid I_1 \subseteq t\}\| / \|D\|$。
- **定义（频繁项目集）**.给定全局项目集I和数据库D，D中所有满足用户指定的最小支持度（Minsupport）的项目集，即大于或等于minsupport的I的非空子集，称为**频繁项目集（频集：Frequent Itemsets）**或者大项目集（Large litemsets）。在频繁项目集中挑选出所有不被其他元素包含的频繁项目集称为**最大频繁项目集（最大频集：Maximum Frequent Itemsets）**或最大大项目集（Maximum Large litemsets）。

可信度与关联规则

- **定义（关联规则与可信度）**.给定一个全局项目集I和数据库D，一个定义在I和D上的关联规则形如$I_1 \Rightarrow I_2$，并且它的**可信度**或**信任度**或**置信度**（Confidence）是指包含I_1和I_2的事务数与包含I_1的事务数之比，即

  $$\text{Confidence}(I_1 \Rightarrow I_2) = \text{support}(I_1 \cup I_2) / \text{support}(I_1),$$

其中I_1，$I_2 \subseteq I$，$I_1 \cap I_2 = \Phi$。

- **定义（强关联规则）**.D在I上满足最小支持度和最小信任度（Minconfidence）的关联规则称为**强关联规则**（Strong Association Rule）。

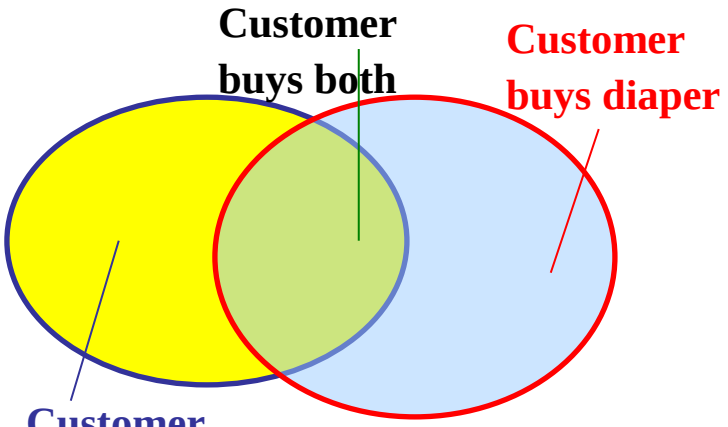
可信度与关联规则

- *Let supmin = 50%,*
- *confmin = 50%*
- *Freq. Pat.:*

{A:3, B:3, D:4, E:3, AD:3}

- Association rules:
  - $A \rightarrow D \quad (60\%, \quad 100\%)$
  - $D \rightarrow A \quad (60\%, \quad 75\%)$

Transaction-id	Items bought
10	A, B, D
20	A, C, D
30	A, D, E
40	B, E, F
50	B, C, D, E, F



可信度与关联规则

- 支持度揭示了A与B同时出现的概率。如果A与B同时出现的概率小，说明A与B的关系不大；如果A与B同时出现的非常频繁，则说明A与B总是相关的。
- 置信度揭示了A出现时，B是否也会出现或有多大概率出现。如果置信度度为100%，则A和B可以捆绑销售了。如果置信度太低，则说明A的出现与B是否出现关系不大。
- 示例：某销售手机的商场中，70%的手机销售中包含充电器的销售，而在所有交易中56%的销售同时包含手机和充电器。则在此例中，支持度为56%，置信度为70%。

关联规则挖掘基本过程

- 关联规则挖掘问题可以划分成两个子问题：
  - 1. **发现频繁项目集**:通过用户给定Minsupport ，寻找所有频繁项目集或者最大频繁项目集。
  - 2. **生成关联规则**:通过用户给定Minconfidence ，在频繁项目集中，寻找关联规则。
- 第1个子问题是近年来关联规则挖掘算法研究的重点。

频繁模式挖掘方法

- Agrawal等人建立了用于事务数据库挖掘的项目集格空间理论（1993, Apriori 属性）。
- **定理（Apriori 属性1）.如果项目集X是频繁项目集，那么它的所有非空子集都是频繁项目集。**
  - 例如：如果 **{beer, diaper, nuts}** 是频繁的，那么 **{beer, diaper}**也是
- **定理（Apriori 属性2）.如果项目集X是非频繁项目集，那么它的所有超集都是非频繁项目集。**

经典的发现频繁项目集算法

- 算法apriori中调用了apriori-gen（L_{k-1}），是为了通过（k-1）-频集产生K-侯选集

（1） FOR all itemset p∈ L_{k-1} DO
（2） FOR all itemset q∈L_{k-1} DO
（3） IF $p.item_1=q.item_1$，…，$p.item_{k-2}=q.item_{k-2}$，$p.item_{k-1} < q.item_{k-1}$ THEN BEGIN
（4） c= p∞q;//把q的第k-1个元素连到p后
（5） IF *has_infrequent_subset*（c，L_{k-1}） THEN
（6） delete c;//删除含有非频繁项目子集的侯选元素
（7） ELSE add c to C_k;
（8） END
（9） Return C_k;

- *has_infrequent_subset*（c，L_{k-1}），判断c是否加入到k-侯选集中。

apriori-gen过程

- 1994年，Agrawal 等人提出了著名的Apriori 算法。
- 算法3-1 Apriori（发现频繁项目集）

```
（1）   L1 = {large 1-itemsets}; //所有1-项目频集
（2）   FOR （k=2; Lk-1≠Φ; k++） DO BEGIN
（3）     Ck=apriori-gen（Lk-1）; // Ck是k-候选集
（4）     FOR all transactions t∈D DO BEGIN
（5）       Ct=subset（Ck，t）; // Ct是所有t包含的候选集元素
（6）       FOR all candidates c∈ Ct DO
（7）         c.count++;
（8）     END
（9）     Lk={c∈Ck |c.count≥minsup_count}
（10）  END
（11）  L= ∪Lk;
```

Apriori算法例子

- Minsupport=50%

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D →

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

→

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D ←

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

←

L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

Scan D →

L_3

itemset	sup
{2 3 5}	2

Apriori算法作业

13. 给定如表 A3-1 所示的一个事务数据库，写出 Apriori 算法生成频繁项目集的过程(假设 MinSuport=50%)。

表 A3-1 事务数据库示例 1

TID	Itemset
1	a,c,d,e,f
2	b,c,f
3	a,d,f
4	a,c,d,e
5	a,b,d,e,f

14. 给定如表 A3-2 所示的一个事务数据库，写出 Apriori 算法生成频繁项目集的过程(假设 MinSuport=40%)。

表 A3-2 事务数据库示例 2

TID	Itemset
1	1,3,4
2	2,3,4,5
3	1,3,5,7
4	2,5
5	1,2,4,6,7
6	2,4,6

关联规则的生成问题

- 根据上面介绍的关联规则挖掘的两个步骤，在得到了所有频繁项目集后，可以按照下面的步骤生成关联规则：
  - 对于每一个频繁项目集*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Conference（x），如果Confidence（x）≥minconfidence，那么“x⇨（*l*-x）”成立。
- 算法3-4 从给定的频繁项目集中生成强关联规则

Rule-generate（L，minconf）
（1） FOR each frequent itemset l_k in L
（2） *genrules*（l_k，l_k）；

- 算法3-4的核心是genrules递归过程，它实现一个频繁项目集中所有强关联规则的生成。

算法-递归测试一个频集中的关联规则

genrules（l_k: frequent k-itemset，x_m: frequent m-itemset）

（1）X={（m-1）-itemsets x_{m-1} | x_{m-1} in x_m}；

（2）FOR each x_{m-1} in X BEGIN

（3）　conf = support（l_k）/support（x_{m-1}）；

（4）　IF （conf ⩾　minconf） THEN BEGIN

（5）　　print the rule "x_{m-1}⇨（l_k-x_{m-1}），with support = support（l_k），confidence=conf";

（6）　　IF （m-1 > 1） THEN //generate rules with subsets of x_{m-1} as antecedents

（7）　　genrules（l_k，x_{m-1}）；

（8）　END

（9）END；

Rule-generate算法例子

- Minconfidence=80%

序号	l_k	x_{m-1}	confidence	support	规则（是否是强规则）
1	235	23	100%	50%	23⇨5（是）
2	235	2	67%	50%	2⇨35（否）
3	235	3	67%	50%	3⇨25（否）
4	235	25	67%	50%	25⇨3（否）
5	235	5	67%	50%	5⇨23（否）
6	235	35	100%	50%	35⇨2（是）

Apriori算法的性能瓶颈

- Apriori作为经典的频繁项目集生成算法，在数据挖掘中具有里程碑的作用。
- Apriori算法有两个致命的性能瓶颈：
  - 1．多次扫描事务数据库，需要很大的I/O负载
    - 对每次k循环，侯选集C_k中的每个元素都必须通过扫描数据库一次来验证其是否加入L_k。假如有一个频繁大项目集包含10个项的话，那么就至少需要扫描事务数据库10遍。
  - 2．可能产生庞大的侯选集
    - 由L_{k-1}产生k-侯选集C_k是指数增长的，例如10^4个1-频繁项目集就有可能产生接近10^7个元素的2-侯选集。如此大的侯选集对时间和主存空间都是一种挑战。

提高Apriori算法效率的技术

- 改进的 Apriori:
  - 减少交易数据库的扫描
  - 减少候选的数量
  - 提高候选频繁项支持度计算效率
- 主要的改进方法有：
  - **基于数据分割（Partition）的方法**：基本原理是“在一个划分中的支持度小于最小支持度的k-项集不可能是全局频繁的”。
  - **基于散列（Hash）的方法**：基本原理是“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”。
  - **基于采样（Sampling）的方法**：基本原理是“通过采样技术，评估被采样的子集中，并依次来估计k-项集的全局频度”。
  - **事务压缩**：动态删除没有用的事务，“不包含任何L_k的事务对未来的扫描结果不会产生影响，因而可以删除”。

基于散列的方法

- 1995，Park等发现寻找频繁项目集的主要计算是在生成2-频繁项目集上。因此，Park等利用了这个性质引入杂凑技术来改进产生2-频繁项目集的方法。
- 例子： 桶地址 =（10x+y）mod 7；minsupport_count=3

TID	Items
1	I1，I2，I5
2	I2，I4
3	I2，I3
4	I1，I2，I4
5	I1，I3
6	I2，I3
7	I1，I3
8	I1，I2，I3，I5
9	I1，I2，I3

桶地址	0	1	2	3	4	5	6
桶计数	2	2	4	2	2	4	4
桶内	{I1，I4}	{I1，I5}	{I2，I3}	{I2，I4}	{I2，I5}	{I1，I2}	{I1，I3}
	{I3，I5}	{I1，I5}	{I2，I3}	{I2，I4}	{I2，I5}	{I1，I2}	{I1，I3}
			{I2，I3}			{I1，I2}	{I1，I3}
			{I2，I3}			{I1，I2}	{I1，I3}

L2={{I2，I3}，{I1，I2}，{I1，I3}}

频繁模式挖掘的瓶颈

- 多次数据库扫描开销太大
- 挖掘长模式需要多次扫描，并产生大量的候选者
  - To find frequent itemset $i_1 i_2 \cdots i_{100}$
    - # of scans: 100
    - # of Candidates: $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{1}{1}\binom{0}{0}\binom{0}{0} = 2^{100}-1 = 1.27*10^{30}$!
- 瓶颈：候选生成和检验
- 我们能避免候选生成么？
- FP-tree

FP-tree算法的基本原理

- 进行2次数据库扫描：一次对所有1-项目的频度排序；一次将数据库信息转变成紧缩内存结构。
- 不使用侯选集，直接压缩数据库成一个频繁模式树，通过频繁模式树可以直接得到频集。
- 基本步骤是：
  - 两次扫描数据库，生成频繁模式树FP-Tree：
    - 扫描数据库一次，得到所有1-项目的频度排序表T；
    - 依照T，再扫描数据库，得到FP-Tree。
  - 使用FP-Tree，生成频集：
    - 为FP-tree中的每个节点生成条件模式库；
    - 用条件模式库构造对应的条件FP-tree；
    - 递归挖掘条件FP-trees同时增长其包含的频繁集：
      - 如果条件FP-tree只包含一个路径，则直接生成所包含的频繁集。

生成频繁模式树FP-Tree

TID	Original Items	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{c, f, a, m, p}
200	{a, b, c, f, l, m, o}	{c, f, 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}	{c, f, a, m, p}

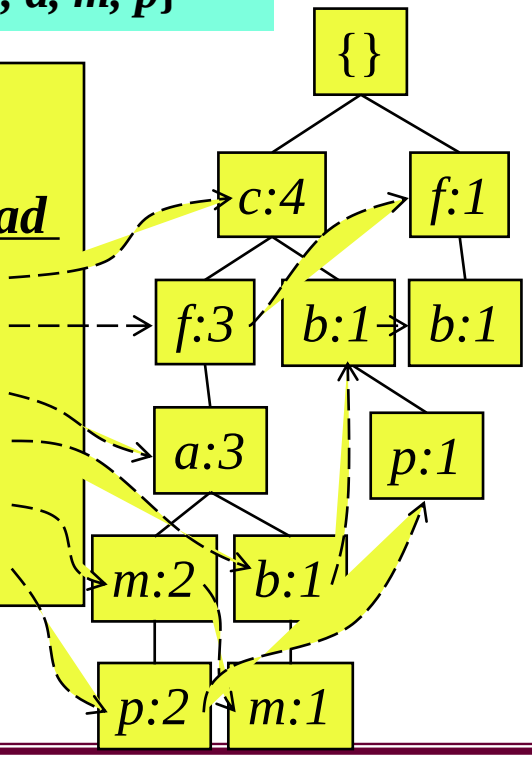
min_support = 0.5

扫描DB一次，找到频繁1项集

- 用频率降序排序选择频繁项，f-list
- 再一次扫描DB，构建FP树

T

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



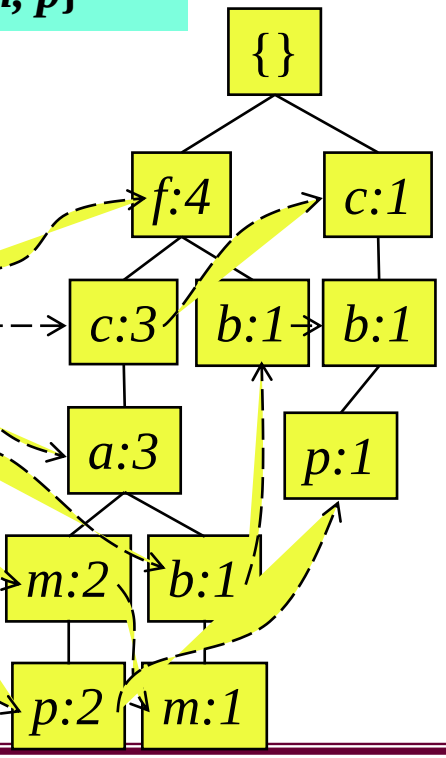
生成频繁模式树FP-Tree

TID	Original 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}

min_support = 0.5

T

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

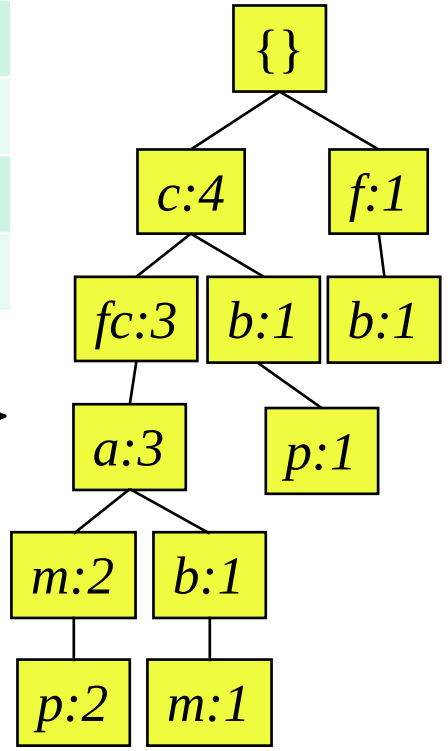


生成频繁模式树FP-Tree

min_support = 0.5

项	模式	条件	频繁集
p	{(cfam:2),(cbp:1)}	<c:3>	cp:3
m	{(cfa:2),(cfab:1)}	<cfa:3>	cfam:3
b	{(cfa:1),(c:1),(f:1)}	ø	ø
a	{(cf:3)}	<cf:3>	cfa:3
F	{(c:3)}	<c:3>	cf:3
C	ø	ø	ø

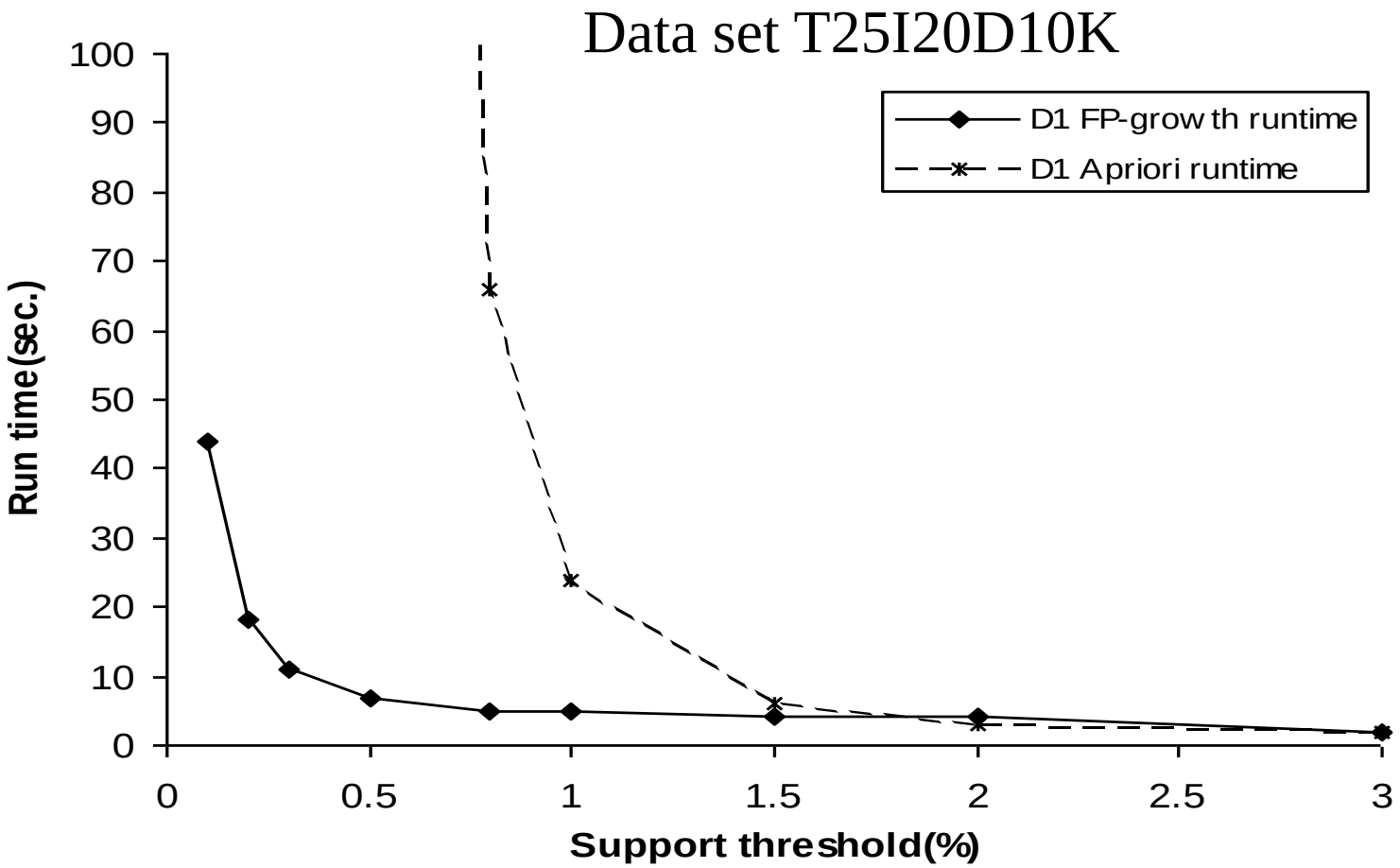
- 最大频繁集为：{{c,p}，{c,f,a,m}}



FP-Tree结构的优点

- **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 vs. Apriori



FP-Tree

- 按照步骤画出FP-Tree，并求出最大频繁集。
- 最小支持度为40%

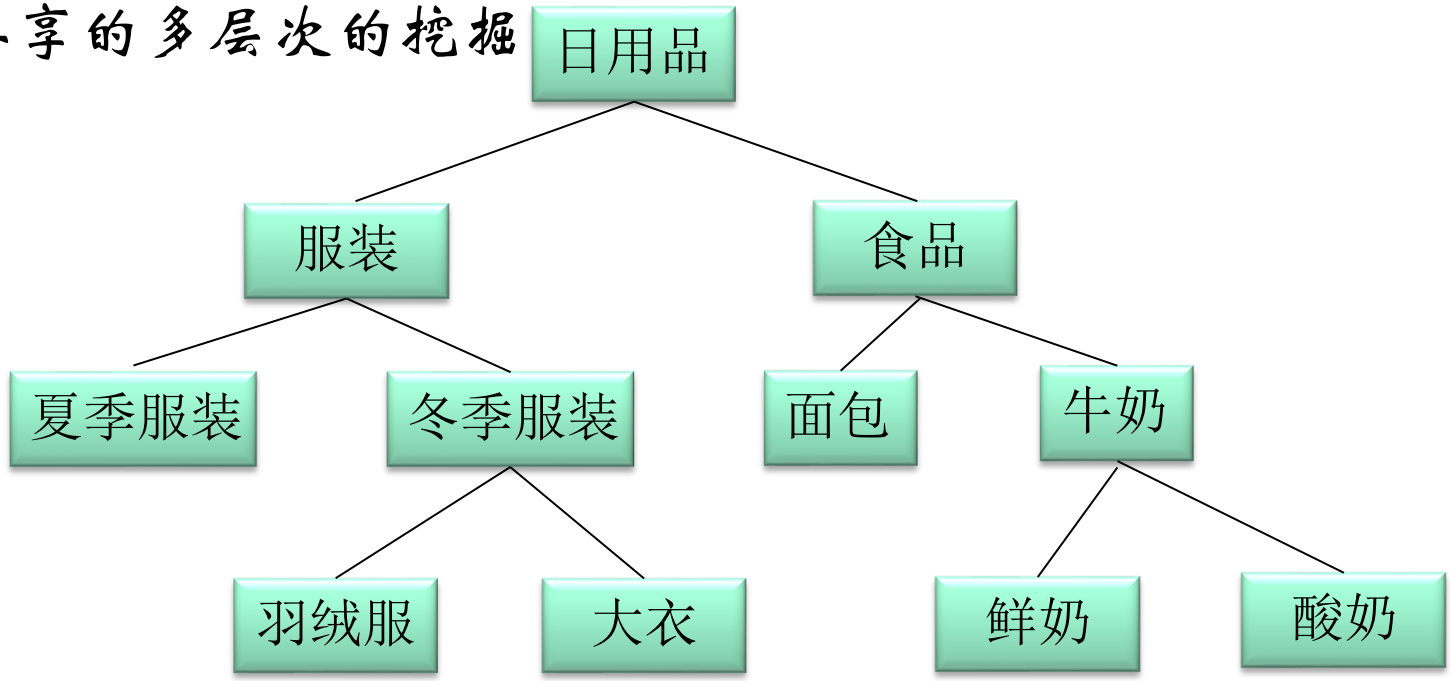
TID	Itemset
1	a,b,c,f
2	b,c,d,e
3	a,c,e
4	b,c,d
5	b,c,d,e

不同层次的关联规则

- 挖掘多层次关联
  - 概念层次
- 挖掘概念层次多维度关联
  - 年龄，项目，职业
- 挖掘定量关联
- 有趣的相关模式挖掘

多层次关联规则

- 项目通常形成层级，多层次关联规则挖掘的度量方法可以沿用“支持度-可信度”的框架
- 灵活的支持度设置：在较低层级的项目上，预计将有较低的支持度
- 探索共享的多层次的挖掘



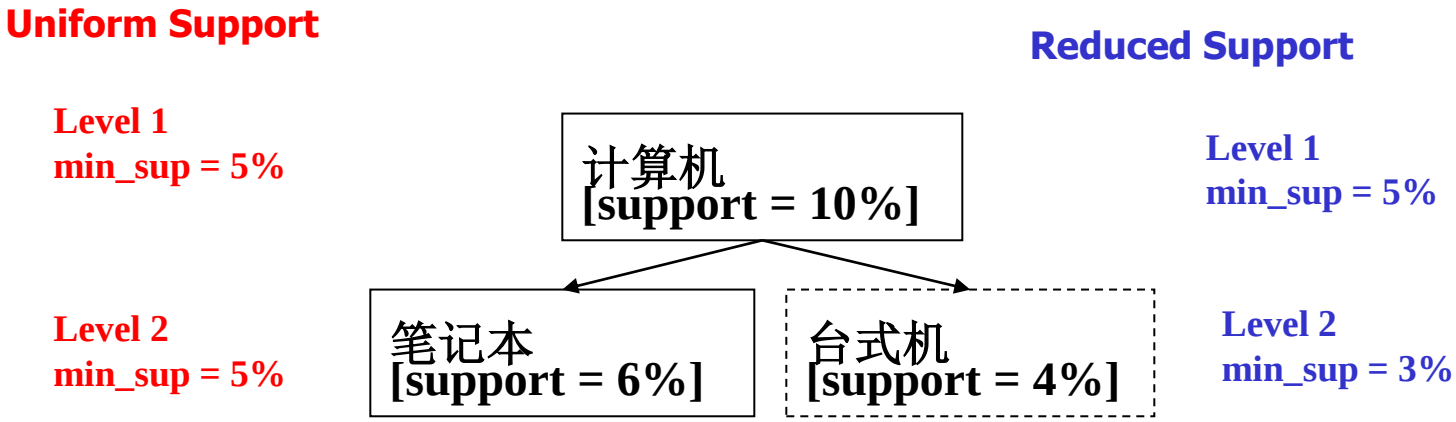
多层次关联规则支持度策略

- 多层次关联规则挖掘有两种基本的设置支持度的策略：
  - 统一的最小支持度：

弊端：不同层次可能考虑问题的精度不同，产生了过多不感兴趣的规则或有用信息丢失

  - 不同层次使用不同的最小支持度：

灵活性高，但需要解决：不同层次间的支持度关联问题，层间的关联规则挖掘也是必须解决的问题



多层次关联规则挖掘策略

- 多层次关联规则挖掘可以使用三种方法：
  - 自上而下的方法

    顶层-》下一层-》下一层……

    每个层次支持度可以一致也可以动态生成

    例如：首先挖掘高层次的频繁项：牛奶（15%），面包（10%）
    然后挖掘他们的较低级别的“弱”的频繁项集:Nest milk（5%），小麦面包（4%）
  - 自下而上的方法

    底层-》上一层-》上一层……
  - 在一个固定层次上的挖掘

多层次关联规则问题

- **产生冗余**
  - 由于项目间先前的关系，一些规则可能是冗余的

Desktop computer ⇒ b/w printer [support = 8%, confidence = 70%]

IBM Desktop computer ⇒ b/w printer [support = 2%, confidence = 72%]

  - 我们说的第一条规则是第二条规则的祖先.
  - 根据规则的祖先，如果它的支持度是接近“预期”的值，规则就是多余的.
- **规则包含、规则合并**
- **选择合适的挖掘策略**

多维度关联

- **单一维度规则：**

  Milk ⇒ bread 即 buys(X, “milk”) ⇒ buys(X, “bread”)

- **多维关联规则可以分为：**
  - **维内的关联规则：**

    年龄（X，20~30）^职业（X，学生）=>购买（X，笔记本电脑）

    这里我们就涉及到三个维：年龄、职业、购买。
  - **混合维关联规则：**这类规则允许同一个维重复出现。

    年龄（X，20~30）^购买（X，笔记本PC）=>购买（X，打印机）

    由于同一个维“购买”在规则中重复出现，因此为挖掘带来难度。但是，这类规则更具有普遍性，具有更好的应用价值，因此近年来得到普遍关注。

挖掘量化关联规则

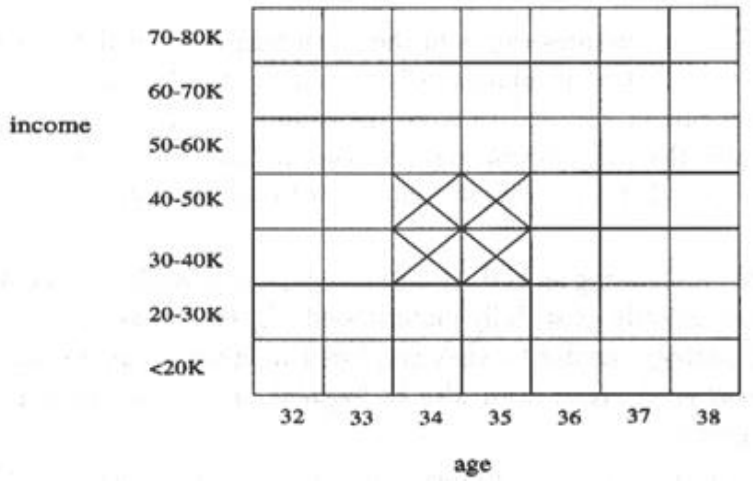
- Age(X, “30…39”) ∧ income(X, “42K…48K”) =>buys (X, “high resolution TV”)
- 使用量化属性的静态离散
  - 通过使用预定义的概念层次，对定量属性进行静态离散化
- 量化关联规则
  - 在数据分布的基础上，量化属性动态的离散化成“箱”.
  - 分箱策略：等宽、等深、基于同质
- 基于距离的关联规则
  - 考虑数据点之间的距离，这是一个动态的离散化过程中

挖掘量化关联规则

- 静态离散化
  - 用概念分层在挖掘前离散化
  - 数值被替换范围
- 相近关联规则聚类
  - 使用2-D栅格将得到的关联规则转化为更一般的规则.
- Example:

age(X,”30-34”) ∧ income(X,”24K - 48K”)

⇒ buys(X,”high resolution TV”)

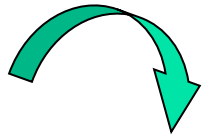


挖掘基于距离的关联规则

- 装箱的方法没有考虑区间数据的语义

Price($)	Equi-width (width $10)	Equi-depth (depth 2)	Distance-based
7	[0,10]	[7,20]	[7,7]
20	[11,20]	[22,50]	[20,22]
22	[21,30]	[51,53]	[50,53]
50	[31,40]		
51	[41,50]		
53	[51,60]		

- 基于距离分隔，更有意义的离散化方法：
  - 每个间隔中点的密度/数量
  - 每个间隔中点的紧密度



从关联分析到相关分析

- 相关分析
  - 正相关
  - 负相关
  - 不相关
- 所有发现的关联规则都是有趣的吗?

Computer game => video

支持度：4000/10000 = 40%

置信度：4000/6000 = 66%

75% > 66%

	数量
Computer game	6000
video	7500
both	4000
total	10000

- 实际是负相关，寻求支持度-置信度的代替

挖掘高度关联的模式

- A=>B[support, confidence,correlation]
- 相关性度量方法：

$$lift = \frac{P(A \cup B)}{P(A)P(B)}$$

>1,正相关
<1,负相关
=1,不相关

	数量
Computer game	6000
video	7500
both	4000
total	10000

P(Cg∪V) = 40%
P(Cg) = 60%
P(V) = 75%
Lift = 0.4/(0.6*0.75)= 0.89

- 其他相关性度量方法：全置信度和余弦等

$$all\_conf = \frac{\sup(X)}{\max\_item\_\sup(X)}$$

基于约束的关联挖掘

- 自动的找到数据库里的所有模式？—不现实
  - 模式太多并且不集中！
- 数据挖掘应该是一个互动的过程
  - 用一种挖掘查询语言（或者图形用户界面），用户可以指示挖掘什么
- 基于约束的挖掘
  - 用户灵活性：为挖掘的内容提供约束
  - 系统优化：为高效的挖掘 探寻约束—基于约束的挖掘

数据挖掘中的约束

- 知识类型约束
  - 分类，关联，等.
- 数据约束—用像SQL的查询
  - 找到12月2号芝加哥成双销售的产品
- 维度/层次约束
  - 在相关地区，价格，品牌，客户分类
- 规则（或模式）约束
  - 小的销售引发(price ＜ $10) 大的销售 (sum ＞ $200)
- 兴趣度约束
  - 强规则: min_support ≥ 3%, min_confidence ≥ 60%
- 基于规则约束的挖掘使得挖掘有效和高效

关联规则的元规则制导挖掘

- 制定他们在挖掘中感兴趣的规则的**语法形式**.
- 元规则可以用来作为约束，以帮助提高挖掘过程的效率.
- 元规则是基于分析师的实验，期望，或有关数据的直觉

元规则

$P1(X, W) \wedge P2(X, V) \Rightarrow buys(X, \text{“educational software”})$

匹配的规则：

$age(X, \text{“30..39”}) \wedge income(x, \text{“42..48K”})$

$\Rightarrow buys(X, \text{“educational software”})$

判断变量和属性变量

- 元规则模板： $P_1 \wedge P_2 \wedge \cdots \wedge P_l \Rightarrow Q_1 \wedge Q_2 \wedge \cdots \wedge Q_r$

基于约束的模式生成：剪枝

- 我们怎么利用规则约束去修正搜索空间？
- 什么样的约束规则可以被“推入”挖掘过程，并且还能保证挖掘查询返回的回答是完整的？
- 规则约束的分类
  - Antimonotonic（反单调的）
  - Monotonic（单调的）
  - Succinct（简洁的）
  - Convertible（可转变的）
  - Inconvertible（不可转变的）

规则约束中的反单调性和单调

- **反单调的**
  - *当一个项集s违反了约束，那么它的任何超集也是*
  - $sum(S.Price) \leq v$ 是反单调的
  - $sum(S.Price) \geq v$ 不是反单调的
- **单调的**
  - *当一个项集s满足了约束，那么它的任何超集也是*
  - $sum(S.Price) \geq v$ is 单调的
  - $min(S.Price) \leq v$ is 不是单调的

规则约束中的简洁和可转变

- **简洁:**
  - 可以列出并且仅仅列出所有满足该限制的集合
  - 不看交易数据库，根据项目的选择，可以决定一个项集S是否满足约束C，精确产生满足其的集合
  - $min(S.Price) \leq v$ 是简洁的
  - $sum(S.Price) \geq v$ 不是简洁的
- **可转变约束**
  - 通过恰当的命令条目，将硬约束转换成反单调的或单调的
  - $avg(S.profit) \geq v$
- **不可转变约束**

Apriori + Constraint

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D →

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

→

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
~~{5}~~	~~3~~

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D ←

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

←

L_2

itemset	sup
{1 3}	2
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

C_3

itemset
{2 3 5}

Scan D →

L_3

itemset	sup
~~{2 3 5}~~	~~2~~

Constraint: Sum{S.price} < 5

Apriori + Constraint

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D →

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
~~{5}~~	~~3~~

→ L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
~~{5}~~	~~3~~

C_2

itemset
{1 2}
{1 3}
~~{1 5}~~
{2 3}
~~{2 5}~~
~~{3 5}~~

Scan D ←

C_2

itemset	sup
{1 2}	1
{1 3}	2
~~{1 5}~~	~~1~~
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

← L_2

itemset	sup
{1 3}	2
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

C_3

itemset
~~{2 3 5}~~

Scan D →

L_3

itemset	sup
~~{2 3 5}~~	~~2~~

Constraint: Sum{S.price} < 5

Apriori + Constraint

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D →

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

→

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
~~{2 3}~~
~~{2 5}~~
~~{3 5}~~

Scan D ←

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

←

L_2

itemset	sup
{1 3}	2
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

C_3

itemset
~~{2 3 5}~~

Scan D →

L_3

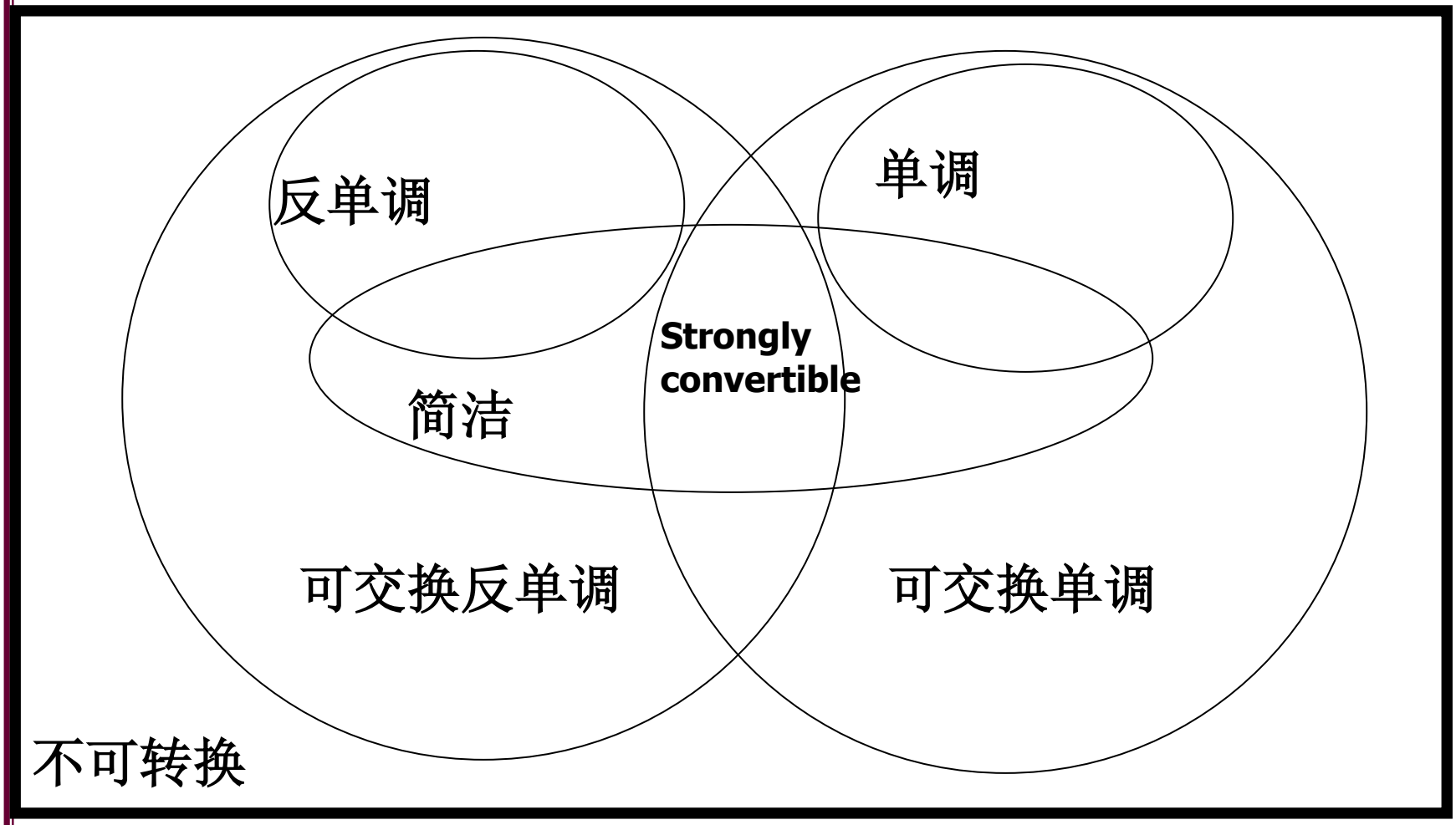
itemset	sup
~~{2 3 5}~~	~~2~~

Constraint:
min{S.price } <= 1

基于约束的挖掘

Constraint	Antimonotone	Monotone	Succinct
$v \in S$	no	yes	yes
$S \supseteq V$	no	yes	yes
$S \subseteq V$	yes	no	yes
$\min(S) \leq v$	no	yes	yes
$\min(S) \geq v$	yes	no	yes
$\max(S) \leq v$	yes	no	yes
$\max(S) \geq v$	no	yes	yes
$\text{count}(S) \leq v$	yes	no	weakly
$\text{count}(S) \geq v$	no	yes	weakly
$\text{sum}(S) \leq v$ ($a \in S, a \geq 0$)	yes	no	no
$\text{sum}(S) \geq v$ ($a \in S, a \geq 0$)	no	yes	no
$\text{range}(S) \leq v$	yes	no	no
$\text{range}(S) \geq v$	no	yes	no
$\text{avg}(S)\ \theta\ v, \theta \in \{=, \leq, \geq\}$	convertible	convertible	no
$\text{support}(S) \geq \xi$	yes	no	no
$\text{support}(S) \leq \xi$	no	yes	no

约束的分类



Thank you !!!