



第八章:

关联分析技术

杨建武

北京大学计算机科学技术研究所

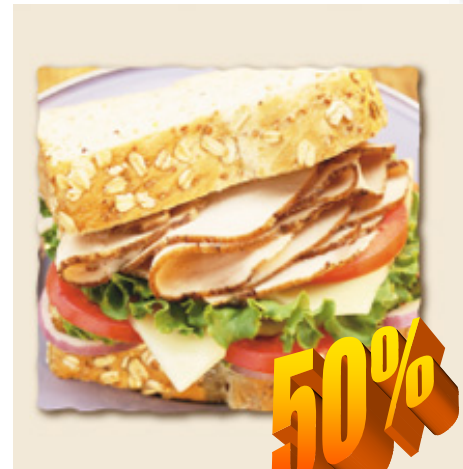
Email:yangjianwu@icst.pku.edu.cn

关联分析



- 关联分析：关联规则挖掘
- 关联规则挖掘的典型示例：**购物篮问题**
- 在商场中拥有大量的商品（项目），如：牛奶、面包等，客户将所购买的商品放入到自己的购物篮中。

A Example



50%





关联分析

- 通过发现顾客放入购物篮中的不同商品之间的联系，分析顾客的购买习惯
 - ❖ 哪些物品经常被顾客购买？
 - ❖ 同一次购买中，哪些商品经常会被一起购买？
 - ❖ 一般用户的购买过程中是否存在一定的购买时间序列？
- 具体应用：利润最大化
 - ❖ 商品货架设计：更加适合客户的购物路径
 - ❖ 货存安排：实现超市的零库存管理
 - ❖ 用户分类：提供个性化的服务

关联规则挖掘



- 简单的说，关联规则挖掘就是发现大量数据中项集之间有趣的关联
 - ❖ 在交易数据、关系数据或其他信息载体中，查找存在于项目集合或对象集合之间的频繁模式、关联、相关性或因果结构。
- 应用
 - ❖ 购物篮分析、交叉销售、分类设计等
 - ❖ 两种策略：
 - 商品放近，增加销量
 - 商品放远，增加其他商品的销量



关联规则挖掘形式化定义

设 $I = \{i_1, i_2, \dots, i_m\}$ 是项 (item) 的集合。

- 若干项的集合，称为**项集** (Item Sets)。
- 记 D 为交易 (transaction, 事务) T 的集合，其中，交易 T 是项的集合，并且 $T \subseteq I$ 。
- 对应每一个交易有唯一的标识，如交易号，记作 TID 。
- 设 X 是一个 I 中项的集合，如果 $X \subseteq T$ ，那么称交易 T 包含 X 。
- 寻找：有趣的关联规则 (**强规则**)。

关联规则



- 所有形如 $X \Rightarrow Y$ 蕴涵式的称为关联规则，这里 $X \subset I$ ， $Y \subset I$ ，并且 $X \cap Y = \Phi$ 。
- 关联规则是有趣的，如果它满足最小支持度阈值与最小置信度阈值，并称之为**强规则**

支持度与置信度



为了描述关联规则的有用性和确定性。

➤ 关联规则的**支持度**

- ❖ 如果交易数据库D中s的交易包含 $A \cup B$ ，则称规则 $A \Rightarrow B$ 在事务集D上的支持度为s。

$$\text{Support}(A \Rightarrow B) = P(A \cup B)$$

➤ 关联规则的**置信度**

- ❖ 如果交易数据库D中，包含A的交易中有c(%)的交易同时也包含B，称规则的置信度为c。

(条件概率)

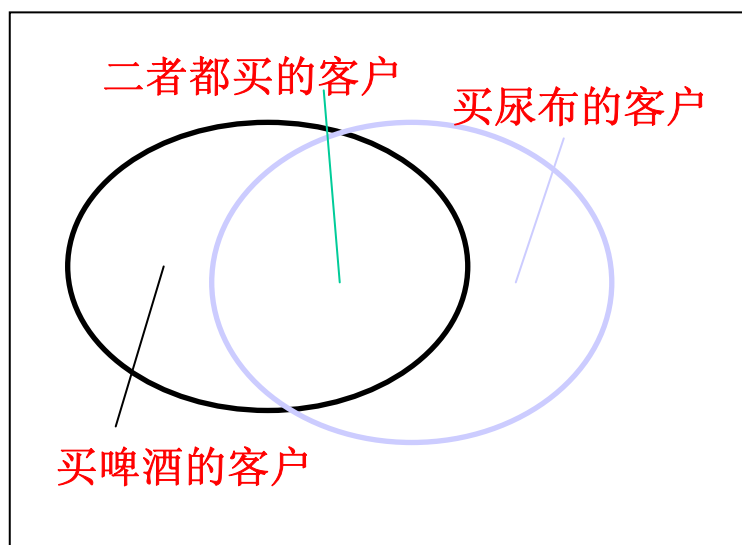
$$\begin{aligned} \text{Confidence}(A \Rightarrow B) &= P(B | A) \\ &= \text{support}(\{A\} \cup \{B\}) / \text{support}(\{A\}) \end{aligned}$$

(注：这里的U是指同时在交易中同时出现 $\{A\}$ 和 $\{B\}$ 。)



关联规则例子

- 查找所有的规则 $A \Rightarrow C$ 具有最小支持度和可信度
 - ❖ 支持度, s , 一次交易中包含{A、C}的可能性
 - ❖ 置信度, c , 包含{A}的交易中也包含C的条件概率



交易ID	购买的商品
10	A,B,C
20	A,C
30	A,D
40	B,E,F



关联规则例子

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

Min. support 50%
Min. confidence 50%

Frequent pattern		Support
{A}	3	75%
{B}	2	50%
{C}	2	50%
{A, C}	2	50%

❖ rule $A \Rightarrow C$:

- $\text{support} = \text{support}(\{A\} \cup \{C\}) = 50\%$
- $\text{confidence} = \text{support}(\{A\} \cup \{C\}) / \text{support}(\{A\}) = 66.7\%$

❖ rule $C \Rightarrow A$ (50%, 100%)



挖掘关联规则方法



频繁项集

➤ 项集(Itemset): a set of items

❖ 例如 $acm = \{a, c, m\}$, $sup = 3$

➤ 频繁项集（高频项集）

❖ 如果项集满足最小支持度，则称之为频繁项集

❖ 如果 $min_sup = 3$, 则 acm 是频繁项集

➤ 如果频繁项集中包含 K 个项，则称为频繁 K 项集

TID	Items bought
100	f, a, c, d, g, l, 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 ₂

关联规则挖掘



- 关联规则的挖掘步骤
 - ❖ 发现频繁项集
 - ❖ 由频繁项集生成满足最小支持度和最小置信度的关联规则
- 发现频繁项集直接的方法
 - ❖ 产生所有可能的项集,并测试它们的支持度
 - ❖ 100个项 $\rightarrow 2^{100}-1$ 可能项集

Apriori性质



Agrawal & Srikant 1994, Mannila, et al. 1994

- Apriori性质：一个频繁项集中任一非空子集也应是频繁项集。
 - ❖ 如果一项交易包含 {牛奶, 面包, 汽水}, 那么它一定包含 {牛奶, 面包}
 - ❖ {牛奶, 面包, 汽水}是频繁的 \rightarrow {牛奶, 面包} 一定也是频繁的
- 也就是说, 任何非频繁项集的超集一定也是非频繁的
 - ❖ 非频繁项集的超集可以不用进行测试
 - ❖ 许多项之间的组合可以去掉 (不满足频繁条件)



Apriori方法

- 寻找最大频繁集
- 逐层搜索的迭代方法。
- 用 k -项集探求 $(k+1)$ -项集。
- 具体地：
 - ❖ 首先找出频繁1-项集，该集合记为 L_1 ；
 - ❖ 用 L_1 找出频繁2-项集的集合 L_2 ；
 - ❖ 如此继续下去，直到找到最大频繁项集
- 该方法，主要有连接和剪枝两步构成。



The Apriori Algorithm

- C_k : Candidate itemset of size k
- L_k : frequent itemset of size k
- $L_1 = \{\text{frequent items}\};$
- for ($k = 1; L_k \neq \emptyset; k++$) do
 - ❖ $C_{k+1} = \text{candidates generated from } L_k;$
 - ❖ for each transaction t in database do
 - increment **the count** of all candidates in C_{k+1} that are contained in t
 - ❖ $L_{k+1} = \text{candidates in } C_{k+1} \text{ with } \text{min_support}$
- return $\cup_k L_k;$

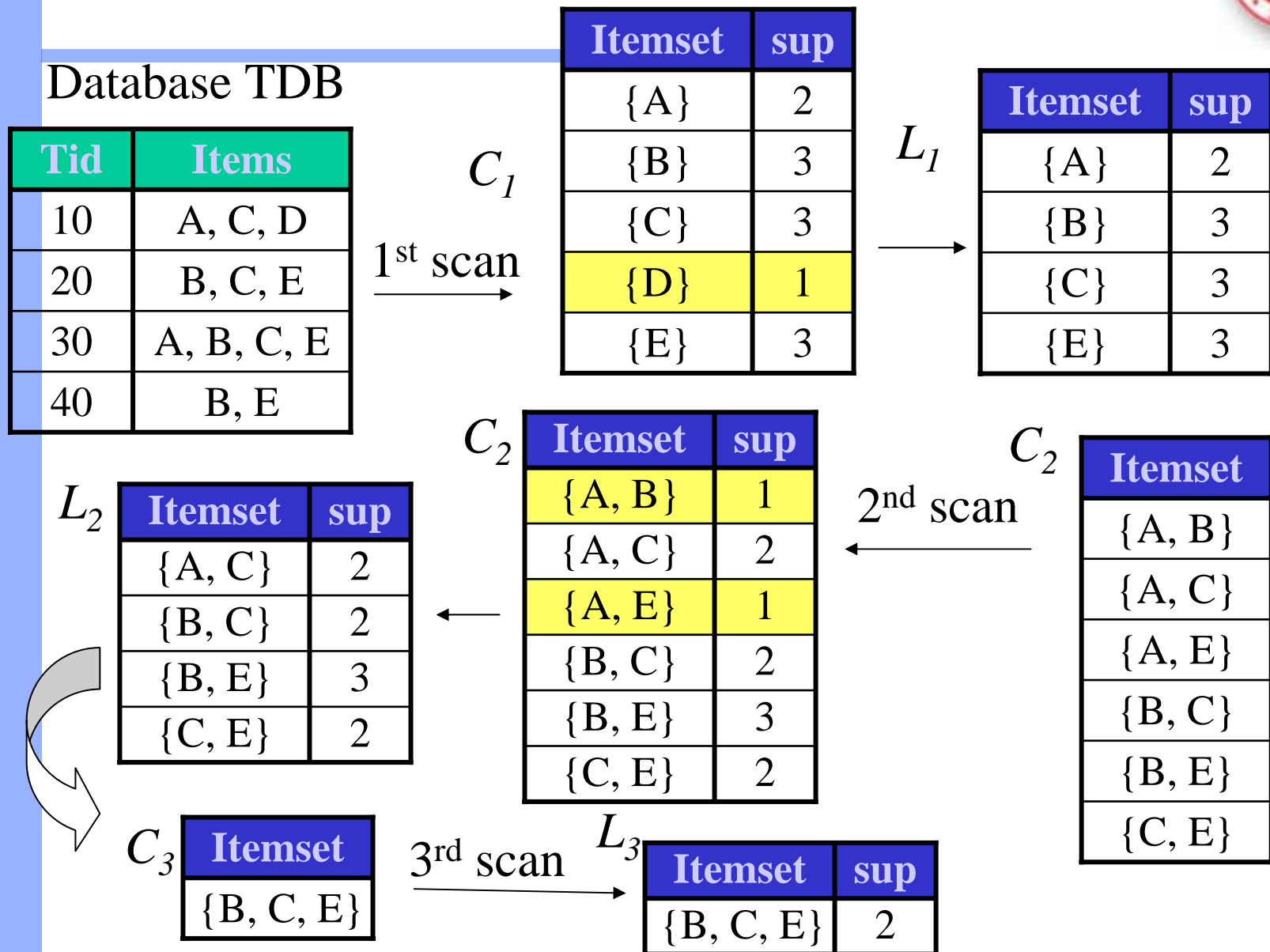
产生候选集



- $L_3 = \{abc, abd, acd, ace, bcd\}$
- **Self-joining:** $L_3 * L_3$
 - ❖ $abcd \leftarrow abc * abd$
 - ❖ $acde \leftarrow acd * ace$
- **Pruning:**
 - ❖ For each itemset c in C_k do
 - For **each $(k-1)$ -subsets** s of c do if (**s is not in L_{k-1}**) then delete c from C_k
 - ❖ $acde$ is removed because ade is not in L_3
- $C_4 = \{abcd\}$



The Apriori Algorithm—An Example



Apriori 性能瓶颈



- 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: candidate-generation-and-test
 - ❖ 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 $\{a_1, a_2, \dots, a_{100}\}$, 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
- Can we avoid candidate generation?

Mining Frequent Patterns Without Candidate Generation



➤ 模式增长的特征

❖ 令 α 为DB的一个频繁集, B 为 α 的条件模式库, β 是 B 中的一个项, 要使 $\alpha \cup \beta$ 是DB中的频繁集, 当且仅当 β 是 B 的频繁项.

➤ 例: "*abcdef*" 是频繁集, 当且仅当

❖ "*abcde*" 是频繁集, 且

❖ "*f*" 在包含 "*abcde*" 的事务中是频繁的。

❖ (注: 支持度按个数算, 而不是百分比)

频繁模式增长算法



- frequent-pattern growth (FP-增长)
 - ❖ 基于FP-tree (频繁模式树)
 - ❖ 将提供频繁集的数据库压缩到一棵FP-tree, 但仍保留项集关联信息;
 - ❖ 将压缩后的数据分成一组条件数据库 (一种特殊类型的投影数据库), 每个关联到一个频繁项集;
 - ❖ 分别挖掘每个条件数据库



构造FP-tree

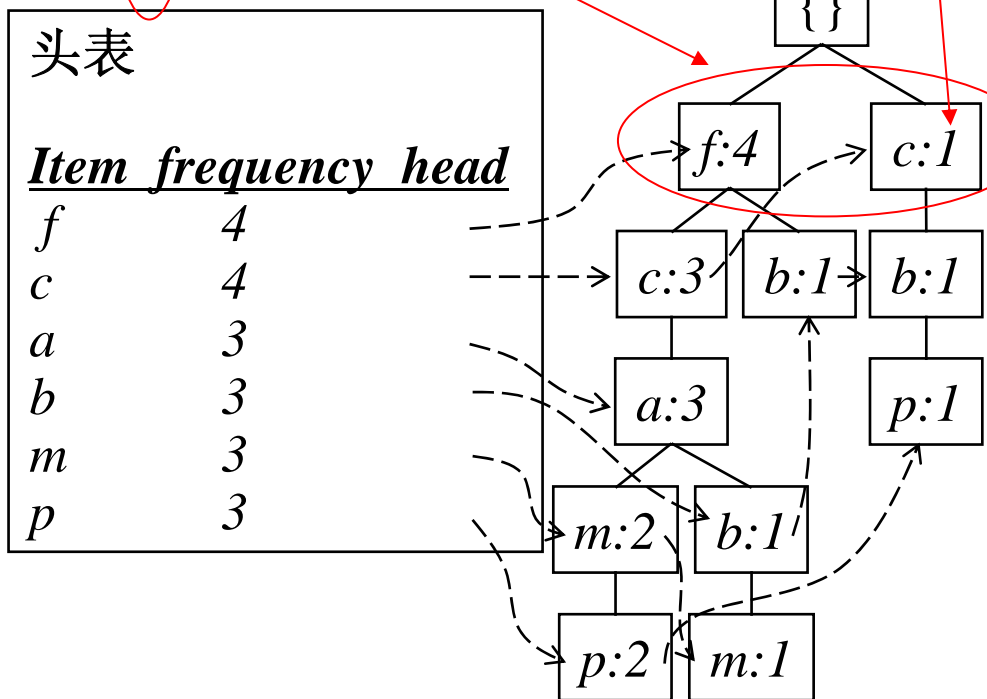
<i>TID</i>	<i>Items bought</i>	<i>(ordered) frequent items</i>
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

$min_support = 3$

次数

步骤:

1. 扫描数据库一次,
得到频繁1-项集
2. 把项按支持度递减
排序
3. 再一次扫描数据库
, 建立FP-tree



基于FP-tree的频繁集挖掘算法



- 基本思想 (分而治之)
 - ❖ 用FP-tree递归增长频繁集
- 方法
 - ❖ 对每个项，生成它的条件模式库
 - ❖ 用条件模式库构造对应的条件FP-tree
 - ❖ 对每个新生成的条件FP-tree，重复这个步骤
 - ❖ 直到结果FP-tree为空，或只含唯一的一个路径 (此路径的每个子路径对应的相集都是频繁集)



步骤1: 从 FP-tree 到条件模式库

- 从FP-tree的头表开始
- 按照每个频繁项的连接遍历 FP-tree
- 列出能够到达此项的**所有前缀路径**，得到条件模式库

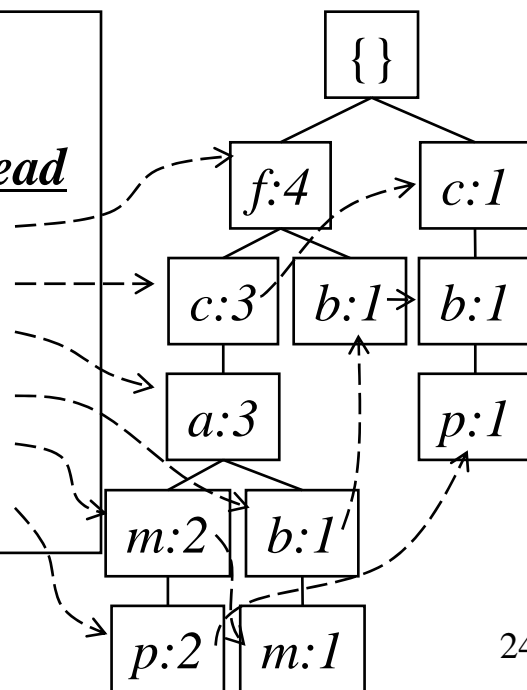
条件模式库

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

头表

Item frequency head

<i>f</i>	4
<i>c</i>	4
<i>a</i>	3
<i>b</i>	3
<i>m</i>	3
<i>p</i>	3



FP-tree支持条件模式库构造的属性



➤ 节点链接

- ❖ 任何包含 a_i 的可能频繁集，都可以从FP-tree头表中的 a_i 沿着 a_i 的节点链接得到

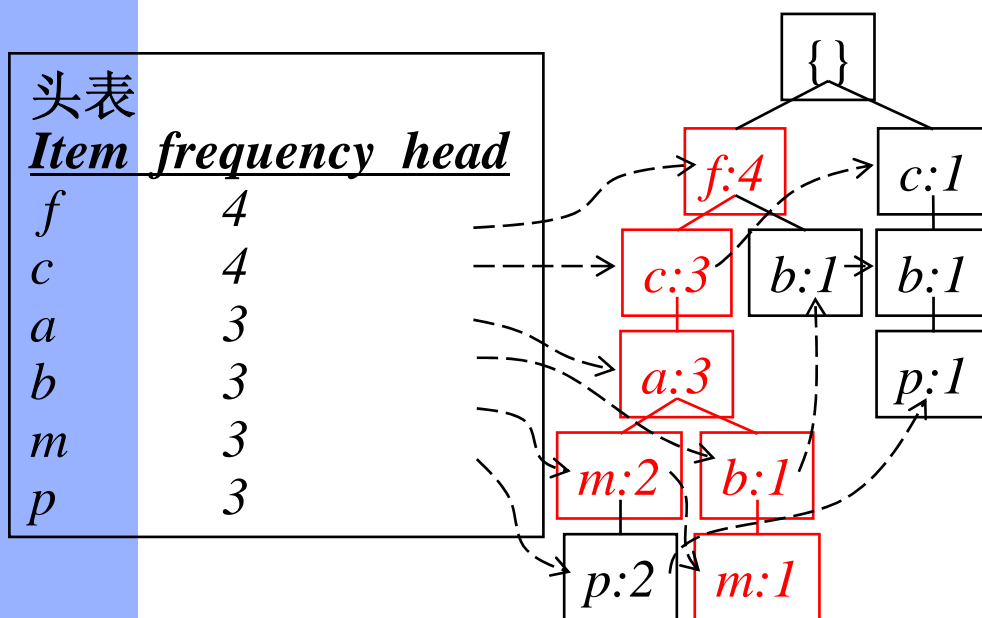
➤ 前缀路径

- ❖ 要计算路径 P 中包含节点 a_i 的频繁集，只要考察到达 a_i 的**路径前缀**即可，且其支持度等于节点 a_i 的**支持度**

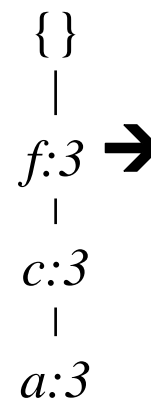


步骤2: 建立条件 FP-tree

- 对每个条件模式库
 - 计算库中每个项的支持度
 - 用模式库中的频繁项建立该条件FP-tree



m-条件模式库:
fca:2, fcab:1



All frequent
patterns relate to
m

m,
fm, cm, am,
fcm, fam, cam,
fcam

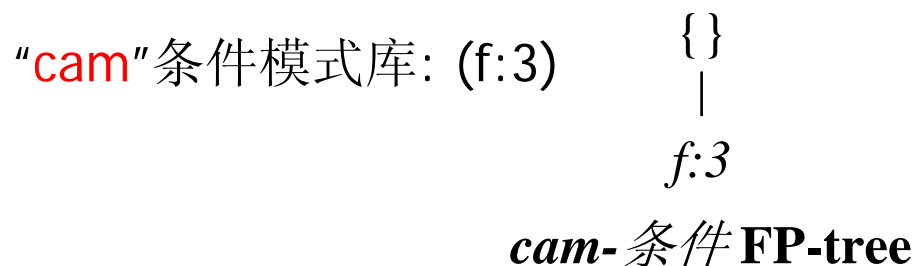
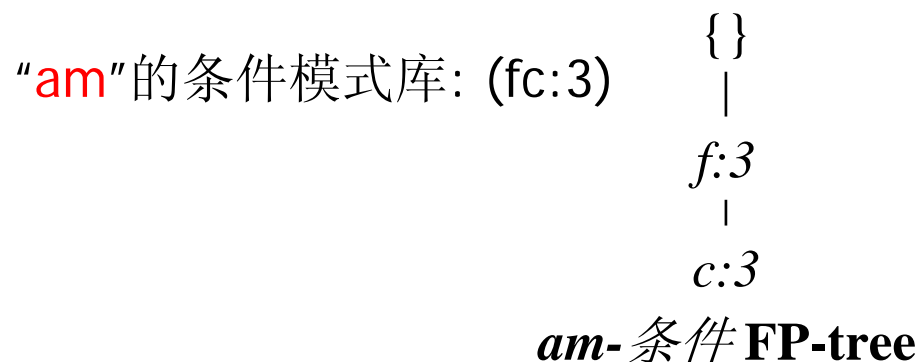
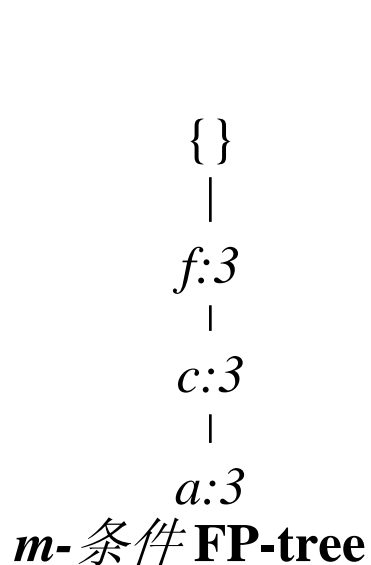
m-conditional FP-tree

通过建立条件模式库得到频繁集



项	条件模式库	条件FP-tree
p	{(fcam:2), (cb:1)}	{(c:3)} p
m	{(fca:2), (fcab:1)}	{(f:3, c:3, a:3)} m
b	{(fca:1), (f:1), (c:1)}	Empty
a	{(fc:3)}	{(f:3, c:3)} a
c	{(f:3)}	{(f:3)} c
f	Empty	Empty

第3步: 递归挖掘条件FP-tree





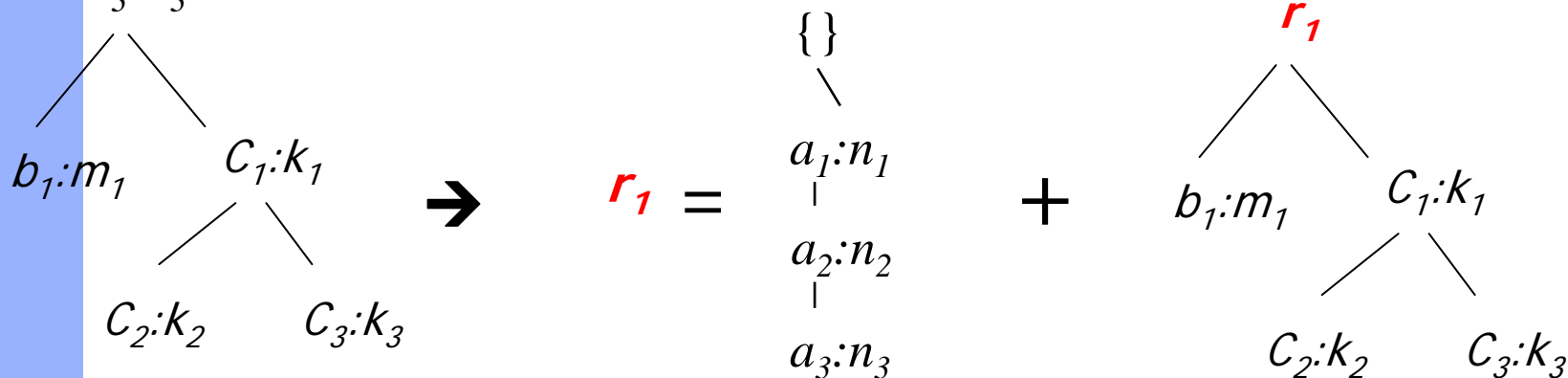
特例: FP-tree 中的唯一前缀路径

- ▶ 假定一个 (条件) FP-tree T 有一个共享唯一前缀路径 P

- $\{\}$ ▶ 挖掘可分解为如下两个步骤

❖ 用一个节点代替此前缀路径 P

❖ 分别计算这两个部分的结果





FP-tree 结构的好处

➤ 完备:

- ❖ 不会打破交易中的任何模式
- ❖ 包含了序列模式挖掘所需的全部信息

➤ 紧密

- ❖ 去除不相关信息—不包含非频繁项
- ❖ 支持度降序排列: 支持度高的项在FP-tree中共享的机会也高
- ❖ 决不会比原数据库大 (如果不计算树节点的额外开销)



性能比较

➤ 性能研究显示

- ❖ FP-growth 比Apriori快一个数量级。

➤ 原因

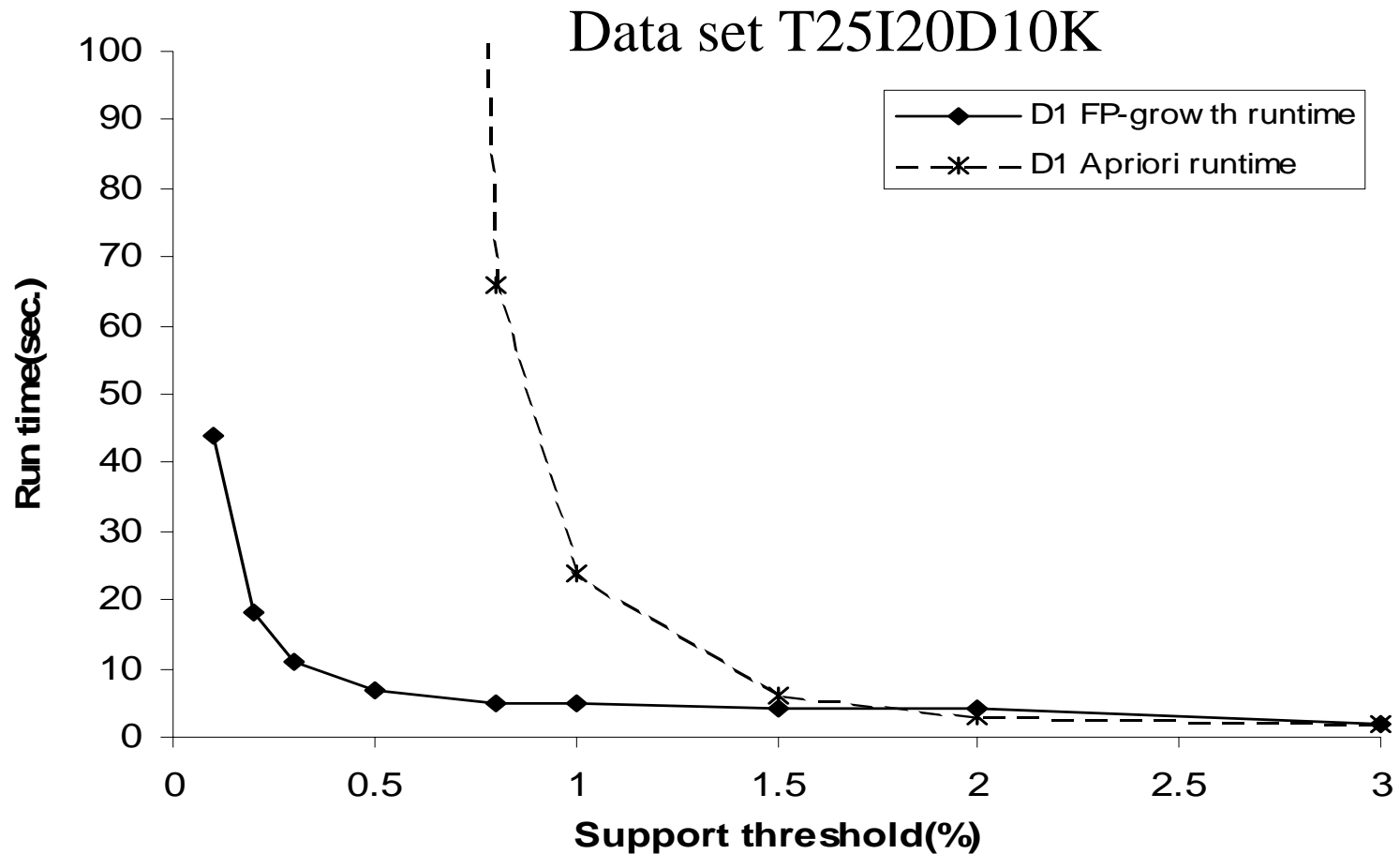
- ❖ 不生成候选集，不用候选测试。

- ❖ 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 树

FP-growth vs. Apriori: 相对于支持度的扩展性



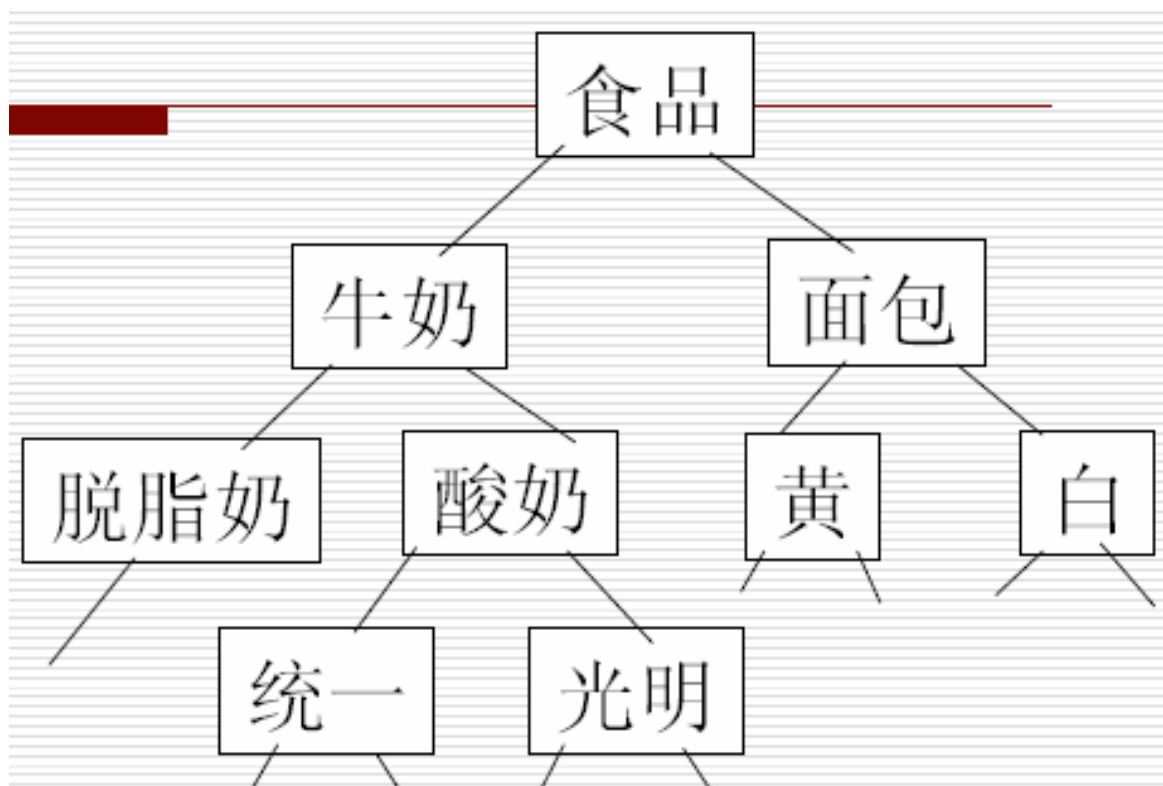


挖掘多层关联规则

➤ 项通常具有层次

❖ 牛奶 \Rightarrow 面包[20%, 60%].

❖ 酸奶 \Rightarrow 黄面包[6%, 50%]





挖掘多层关联规则

➤ 自上而下，深度优先的方法：

❖ 先找高层的“强”规则：

牛奶 \Rightarrow 面包[**20%, 60%**].

❖ 再找他们底层的“弱”规则：

酸奶 \Rightarrow 黄面包[**6%, 50%**].

➤ 多层关联规则的变种：

层次交叉的关联规则：

- 酸奶 \Rightarrow 北大面包房黄面包

基于关联规则的文本分类



- 文档中的每个单词看作是一个项目 (Item)
- 每篇文档看作是一个事务 (Transaction), 即: 项目的集合
- 在不同文档中频繁项集 (文档集的单词共同出现, 简称“共现”) 用于产生分类的规则



小结

- 关联规则概念
- 关联分析的基本方法
- 基于关联规则的文本分类

课程作业



如何获得数据？

➤ 1. 公开评测的数据集

❖ 如：SEWM评测（北大网络实验室）

- 自动分类：
- <http://www.cwirf.org/SharedRes/DataSet/cct.html>
- Web检索：
- <http://www.cwirf.org/SharedRes/DataSet/cwt.html>

➤ 网页抓取工具

❖ 如：Webzip等

➤ 元搜索

❖ 通过Google, Baidu获得检索结果，然后把检索结果抓下来。



Any Question?