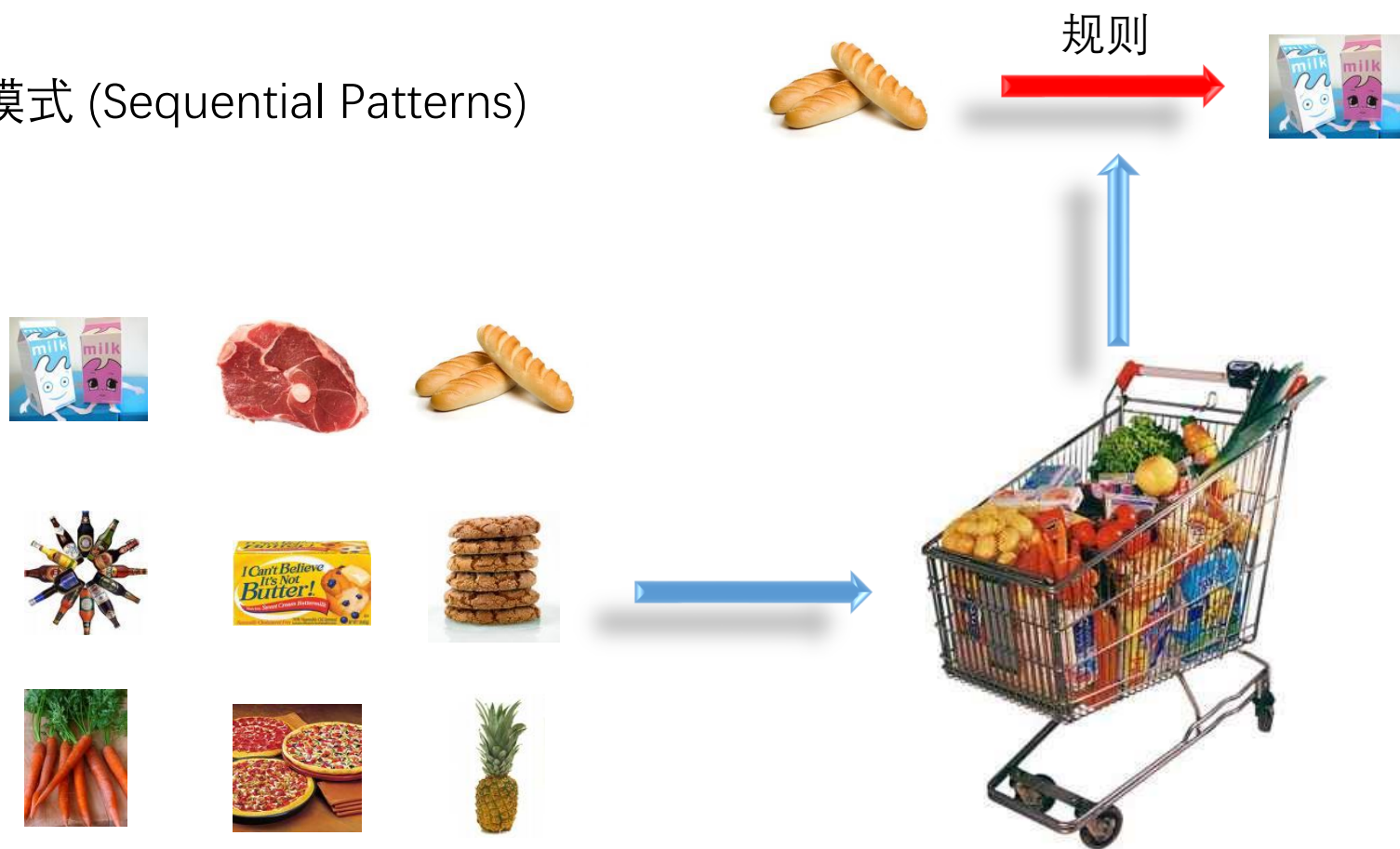


关联分析

参考：《数据挖掘导论》(中文版, 第2版), 陈封能等著, 2019. 第5章

概述

- 频繁项集 (Frequent Itemsets)
- 关联规则 (Association Rules)
- 序列模式 (Sequential Patterns)



关联规则的例子

❑ 沃尔玛，请把蛋挞与飓风用品摆在一起

- 通过对历史交易记录这个庞大数据库进行观察，沃尔玛注意到，每当季节性飓风来临之前，不仅手电筒销量增加，而且美式早餐含糖零食蛋挞销量也增加了。
- 因此每当季节性飓风来临时，沃尔玛就会把蛋挞与飓风用品摆放在一起，从而增加销量。

❑ 尿不湿和啤酒

- 超级商业零售连锁巨无霸沃尔玛公司(Wal Mart)拥有世上最大的数据仓库系统之一。为了能够准确了解顾客在其门店的购买习惯，沃尔玛对其顾客的购物行为进行了购物篮关联规则分析，从而知道顾客经常一起购买的商品有哪些。
- 跟尿不湿一起购买最多的商品竟是啤酒”! 这是数据挖掘技术对历史数据进行分析的结果，反映的是数据的内在规律。
- 那么这个结果符合现实情况吗?是否是一个有用的知识? 是否有利用价值?

尿不湿和啤酒

- 经过大量实际调查和分析，他们揭示了一个隐藏在“尿不湿与啤酒”背后的美国消费者的一种行为模式：
- 在美国，到超市去买婴儿尿不湿是一些年轻的父亲下班后的日常工作，而他们中有30%~40%的人同时也会为自己买一些啤酒。产生这一现象的原因是：美国的太太们常叮嘱她们的丈夫不要忘了下班后为小孩买尿不湿，而丈夫们在买尿不湿后又随手带回了他们喜欢的啤酒。另一种情况是丈夫们在买啤酒时突然记起他们的责任，又去买了尿不湿。既然尿不湿与啤酒一起被购买的机会很多，那么沃尔玛就在他们所有的门店里将尿不湿与啤酒并排摆放在一起，结果是得到了尿不湿与啤酒的销售量双双增长。
- 按常规思维，尿不湿与啤酒风马牛不相及，若不是借助数据挖掘技术对大量交易数据进行挖掘分析，沃尔玛是不可能发现数据内这一有价值的规律的。

关联规则在电商中的应用

亚马逊
amazon.cn
免费试读Prime

全部分类

浏览

全部商品分类

Z秒杀

礼品卡

全球开店

海外购

帮助

In English

- 不仅是传统零售业，在电商中同样有**交叉销售** (Cross Selling)
- 比如，根据用户已经购买的商品，进行商品推荐，或者把两种商品捆绑销售（2件9折，3件8折）

**商品已加入购物车**
库存中仅剩 1 件。

购物车小计 (6 件商品): ¥ 2,054.86
您订单中的部分商品可享受**免费配送**，请在提交订单时选择快速送货上门。详情

购物车

进入结算中心 (6 件商品)

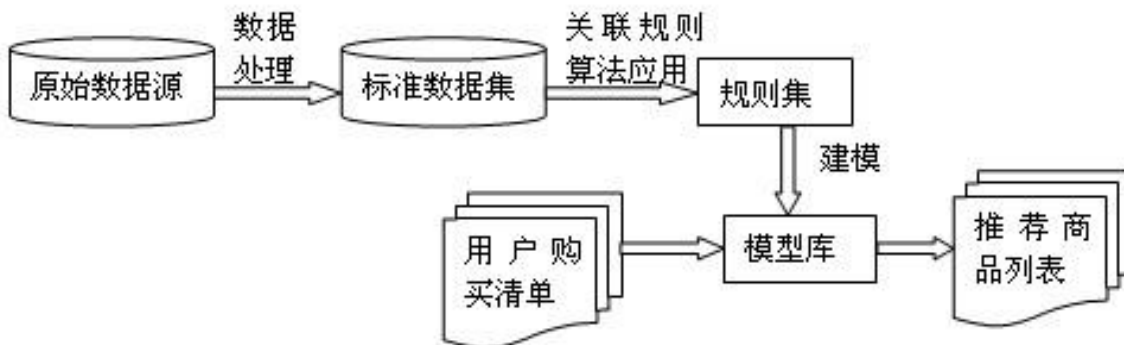
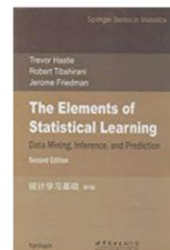
经常与经典原版书库：数据挖掘导论(英文版)一起购买的商品

频繁项集



购买经典原版书库：数据挖掘导论(英文版)的客户还购买了：

关联规则



超市会发现很多关联规则

- 超市分析了很多商品之间的关联性。
- 尿布加啤酒是所发现的关联规则中的一个。

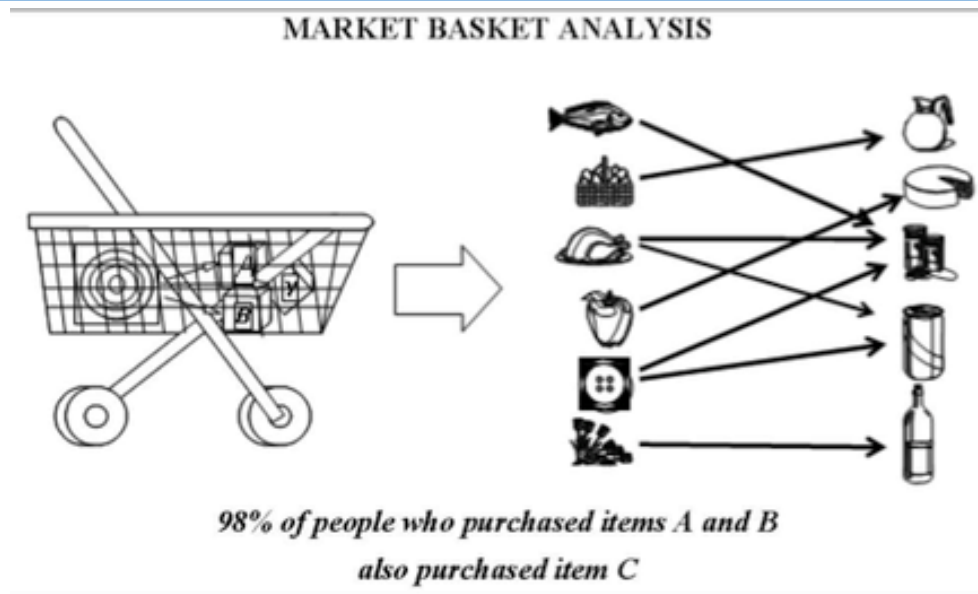
事务 (Transactions)	项 (Items)
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

发现的规则:

$\{\text{Milk}\} \rightarrow \{\text{Coke}\}$

$\{\text{Diaper, Milk}\} \rightarrow \{\text{Beer}\}$

购物篮问题 (Market-Based Problems)

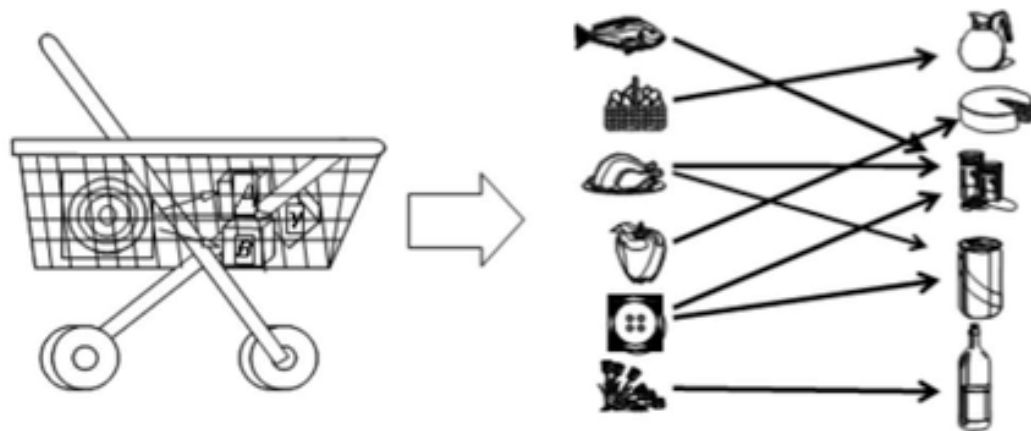


事务 (Transactions)	项 (Items)
1	Bread, Jelly, Peanut, Butter
2	Bread, Butter
3	Bread, Jelly
4	Bread, Milk, Butter
5	Chips, Milk
6	Bread, Chips
7	Bread, Milk
8	Chips, Jelly

- **项** (item): 购物篮中的每一个商品
 - 如 Bread, Milk, Chocolate, Butter ...
- **项集** (itemset): 购物篮中多个项组成的集合
 - 如 {Bread, Jelly, Peanut, Butter}, {Bread, Butter} ...
 - 每个顾客购买的所有商品都是一个项集。其中，项集中项的个数称为项集的长度
 - 含有 k 个项的项集成为 **k -项集**。例如，{milk, bread, Butter} 是一个 3-项集

购物篮问题 (Market-Based Problems)

MARKET BASKET ANALYSIS



标识 (TID)	项 (Items)
1	Bread, Jelly, Peanut, Butter
2	Bread, Butter
3	Bread, Jelly
4	Bread, Milk, Butter
5	Chips, Milk
6	Bread, Chips
7	Bread, Milk
8	Chips, Jelly

- 定义 I 为所有项(商品)的集合, 在上述例子中 $I = \{\text{Bread, Jelly, Peanut, Butter, Milk, Chips}\}$ 。
- 事务**(又称**交易**, transaction), 包含一个标识TID和购买的商品的集合。
- I 的每个非空子集都称为一个**事务** (至少购买一件商品)。
- 所有事务构成事务数据库 (记为 D)。
- 每一个事务(交易)有唯一的标识, 记作TID (Transaction Identifier)
- 关联规则**的主要目的是找出事务数据库中多次重复出现的项(item)之间的“关联”(association)

基于购物篮的问题(Market-Based Problems)

- 交叉销售(Cross Selling)
 - 向现有客户销售其他产品或服务.
- 捆绑折扣 (Bundle Discount)
- 商店布局设计
 - 最小距离 vs. 最大距离
 - 牙膏和牙刷的货架摆放远一些, 可增加顾客在商场的停留时间
- 其他应用:
 - “篮子”和“项”: 句子和单词(“Baskets” & “Items”: Sentences & Words)
 - 把文档（例如网页、推特）当做购物篮, 把词汇当做item。我们可以发现哪些词汇之间共同出现的频率较高。



关联规则基本概念

- 设 $I = \{i_1, i_2, \dots, i_m\}$ 是所有项(Item) i_j ($j=1, 2, \dots, m$) 的集合
- T 是所有事务构成的集合: $T = \{t_1, t_2, \dots, t_N\}$, T 为 I 的一个非空子集, 也称为事务标识符集. 每一个事务 t_i 是一个项集, 即 $t_i \subset I$
- **关联规则形式化为** $P \rightarrow Q$ 的蕴含式, 其中 $P \subset I, Q \subset I$, 且 $P \cap Q = \emptyset$
- 数据库 D 包含所有事务的集合
- 项集可以看作是项的合取(conjunction)
(一组共同出现的商品)

标识 (TID)	项 (Items)
1	Bread, Jelly, Peanut, Butter
2	Bread, Butter
3	Bread, Jelly
4	Bread, Milk, Butter
5	Chips, Milk
6	Bread, Chips
7	Bread, Milk
8	Chips, Jelly

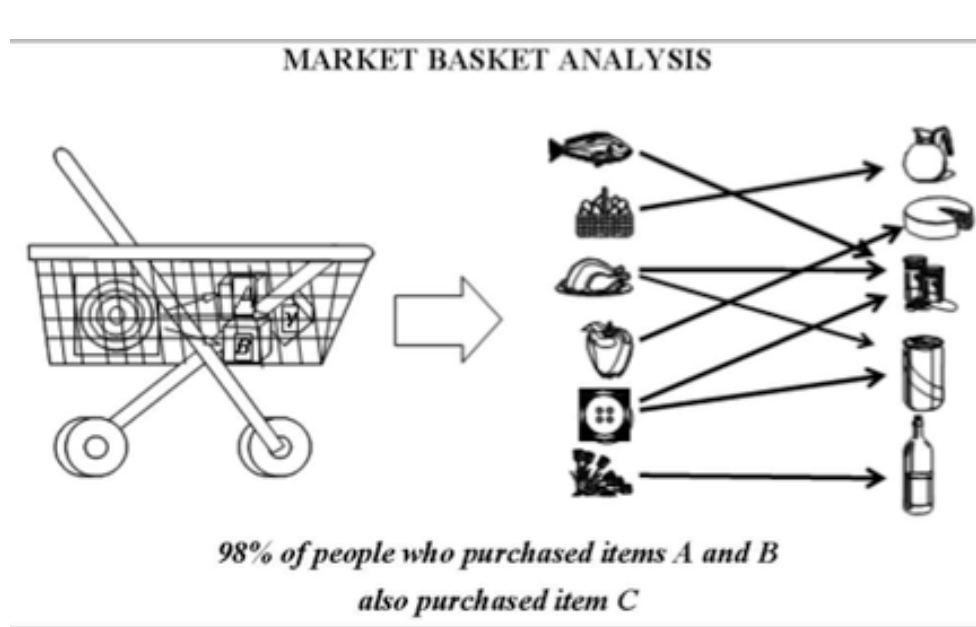
事务(Transactions)

标识 (TID)	项 (Items)
1	Bread, Jelly, Peanut, Butter
2	Bread, Butter
3	Bread, Jelly
4	Bread, Milk, Butter
5	Chips, Milk
6	Bread, Chips
7	Bread, Milk
8	Chips, Jelly



目的在于挖掘出如下形式的关联规则: **Bread** → **Butter**

超市处理不了过多的关联规则



标识 (TID)	项 (Items)
1	Bread, Jelly, Peanut, Butter
2	Bread, Butter
3	Bread, Jelly
4	Bread, Milk, Butter
5	Chips, Milk
6	Bread, Chips
7	Bread, Milk
8	Chips, Jelly

- 大型超市的数据库中可能有10万个项(item), 几百万个购物篮(可以是一笔交易, 即一张购物小票的形式)
- 必须限定关联规则的数量, 否则如果给超市经理提供一百万条关联规则, 他们根本阅读不过来。
- 规则兴趣度(有效性)的两种度量: **支持度**(support)和**置信度**(confidence)

项（或项集）的支持度计数(Support Number)

- 项（或项集）X 的支持度计数(频度或计数)为数据库 *D* 中包含项（或项集）X 的事务的数目

标识 (TID)	项 (Items)
1	Bread, Jelly, Peanut, Butter
2	Bread, Butter
3	Bread, Jelly
4	Bread, Milk, Butter
5	Chips, Milk
6	Bread, Chips
7	Bread, Milk
8	Chips, Jelly

项 (Item)	支持度计数 (Support Number)	项集 (Itemset)	支持度计 数(Support Number)
Bread	6	Bread, Butter	3
Butter	3	...	
Chips	2	Bread, Butter, Chips	0
Jelly	3	...	
Milk	3	Bread, Butter, Chips, Jelly	0
Peanut	1	...	
		Bread, Butter, Chips, Jelly, Milk	0
		...	
		Bread, Butter, Chips, Jelly, Milk, Peanut	0

项（或项集）的支持度(support)

- 项（或项集） X 的**支持度 (sup)** 是包含项（或项集） X 的事务数与总事务数 ($\#D$) 的比值 (有时项（或项集） X 的支持度也记为 $\text{sup}(X, D)$, 简写为 $\text{sup}(X)$)

$$\text{sup}(X) = \frac{\#X}{\#D}$$

标识 (TID)	项 (Items)
1	Bread, Jelly, Peanut, Butter
2	Bread, Butter
3	Bread, Jelly
4	Bread, Milk, Butter
5	Chips, Milk
6	Bread, Chips
7	Bread, Milk
8	Chips, Jelly

项 (Item)	支持度 (Support)	项集 (Itemset)	支持度 (Support)
Bread	6/8	Bread, Butter	3/8
Butter	3/8	...	
Chips	2/8	Bread, Butter, Chips	0/8
Jelly	3/8	...	
Milk	3/8	Bread, Butter, Chips, Jelly	0/8
Peanut	1/8	...	
		Bread, Butter, Chips, Jelly, Milk	0/8
		...	
		Bread, Butter, Chips, Jelly, Milk, Peanut	0/8

支持度单调不增，当项集越大时，支持度越小

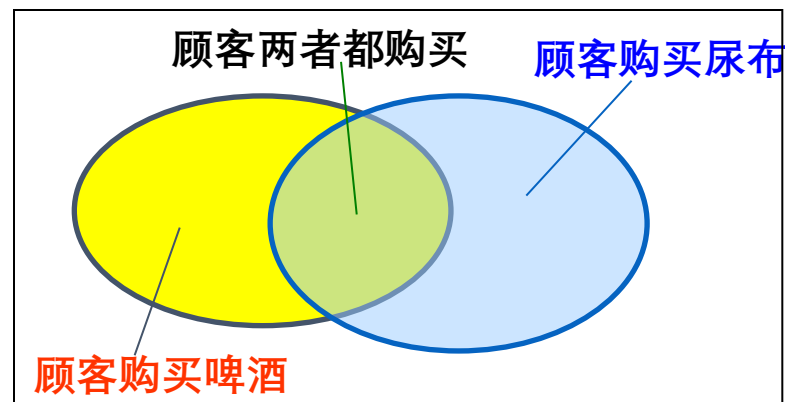
规则的支持度(support)

- 关联规则 $X \rightarrow Y$ 的支持度计数为同时包含项集X和Y的事务数。
- 规则的支持度(sup)是数据库 D 中包含项集 X 和项集 Y 的事务数与数据库 D 中的事务数的比

$$\text{sup}(X \rightarrow Y) = P(X \cup Y) = \frac{\#(X \cup Y)}{\#D}$$

- 其中 $\#(X \cup Y)$ 表示 X 、 Y 两个项集同时发生的事务个数, $\#D$ 表示事务数据库 D 的总数。(即同时购买 X 和 Y 的**频率**)
- 描述了 X 和 Y 所含的项在所有的事务中同时出现的概率有多大, 即概率 $P(XY)$:

$$\text{sup}(X \rightarrow Y) = P(X \cup Y) = \frac{\#(X \cup Y)}{\#D}$$



规则的置信度(confidence)

- **规则的置信度(conf)**指在数据库 D 中同时包含两个项集 $\{X, Y\}$ 的事务数与包含项集 X 的事务数的比率:

$$conf(X \rightarrow Y) = \frac{\#(X \cup Y)}{\#(X)}$$

其中 $\#(X)$ 表示事务数据库 D 中包含项集 X 的事务个数。

- **条件概率**: 在出现了项集 X 的事务 T 中, 项集 Y 也同时出现的概率, 即条件概率 $P(Y|X)$, 即

$$conf(X \rightarrow Y) = \mathbf{P(Y|X)} = \frac{\#(X \cup Y)}{\#(X)}$$

- 上式置信度可以等价地表示为

$$conf(X \rightarrow Y) = \frac{sup(X \cup Y)}{sup(X)}$$

关联规则的支持度和置信度

- **支持度 (Support)** 度量了规则在事务数据集中出现的频率.
- **置信度 (Confidence)** 度量了规则的强度.

标识 (TID)	项 (Items)
1	Bread, Jelly, Peanut, Butter
2	Bread, Butter
3	Bread, Jelly
4	Bread, Milk, Butter
5	Chips, Milk
6	Bread, Chips
7	Bread, Milk
8	Chips, Jelly

Bread → Milk

Support: 2/8

Confidence: 2/6=1/3

Milk → Bread

Support: 2/8 (相当于项集本身的支持度)

Confidence: 2/3

买牛奶的人会去买面包的规则比买面包的人会再买牛奶的规则更强

频繁项集和强规则(Frequent Itemsets and Strong Rules)

- 人们一般只对满足一定的支持度和置信度的关联规则感兴趣
- 为了筛选出那些具有较高支持度和置信度的规则，需要给这两个指标分别设定一个阈值。达到了阈值的规则才是有效规则。
- 支持度和置信度受到阈值的限制:
 - 最小支持度 σ (或记为 $minsup$)
 - 最小置信度 Φ (或记为 $minconf$)
- **频繁项集**是支持度大于 $minsup$ 的项集，即满足 $sup(X) > minsup$ 的项集。
- 频繁 k 项集的集合通常记为 L_k
- **强关联规则(强规则)**是一种同时满足最小支持度($minsup$)和最小置信度($minconf$)的规则 (或者说**条件概率**大于 $minconf$), 即满足

$$sup(X \rightarrow Y) > minsup \text{ 且 } conf(X \rightarrow Y) > minconf$$

的规则

- 关联规则问题

- 给定所有商品 I , 数据库 D , 两个阈值 σ 和 Φ , 求出满足 $X \rightarrow Y$ 的所有强规则, 即为关联规则的挖掘.
- 数据挖掘主要就是对强规则的挖掘

- 挖掘关联规则的一种原始方法是: 暴力法(Brute-force approach):

- 计算每个可能规则的支持度和置信度
- 这种方法计算代价过高, 因为可以从数据集提取的规则的数量达指数级
- 从包含 d 个项的数据集提取的可能规则的总数 $R=3^d-2^{d+1}+1$, 如果 d 等于 6, 则 $R=602$

- 大多数关联规则挖掘算法通常采用的一种策略是，将关联规则挖掘任务分解为如下两个主要的子任务：

1. 生成频繁项集 (Frequent Itemset Generation)

- 其目标是发现满足最小支持度阈值的所有项集，这些项集称作频繁项集。

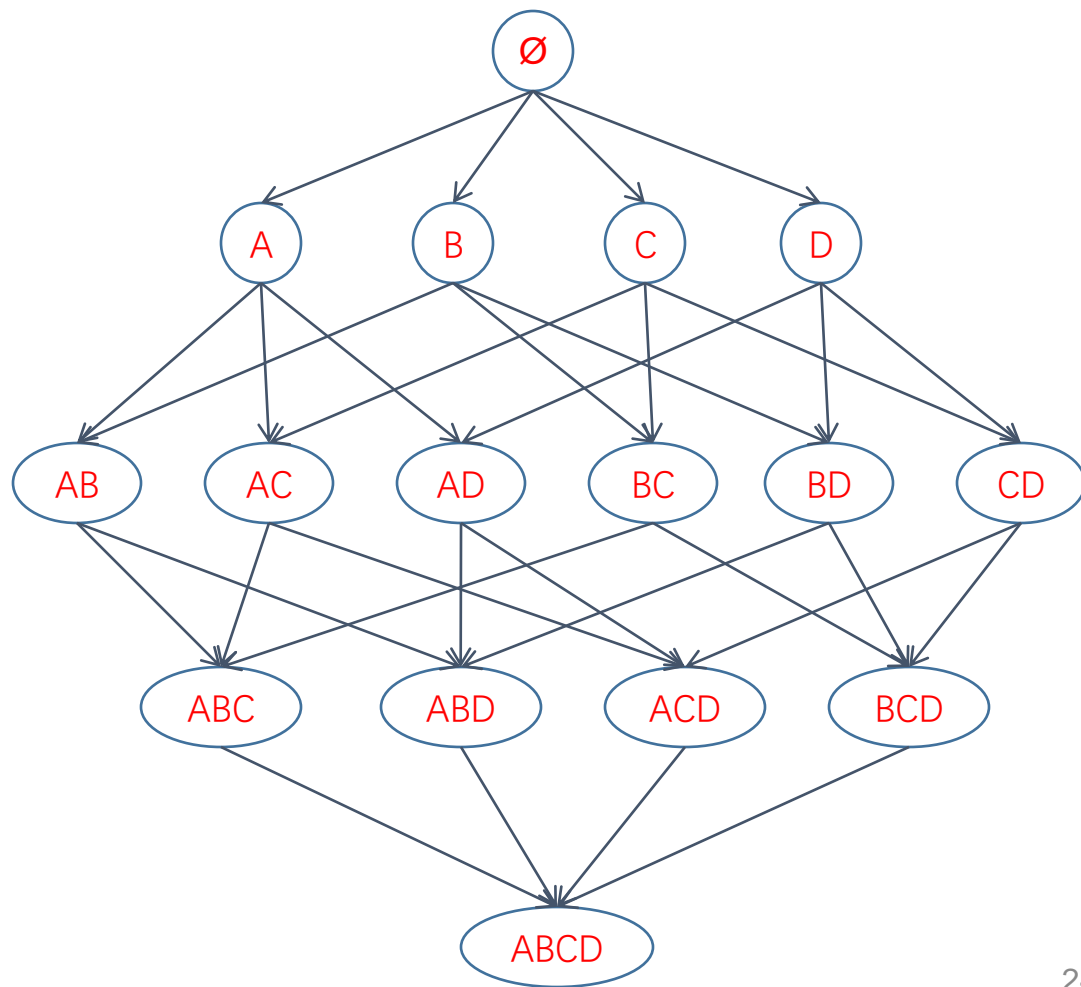
2. 生成规则 (Rule Generation)

- 其目标是从上一步发现的频繁项集中提取所有高置信度的规则，这些规则称作强规则 (strong rule) 。

项集生成 (Itemset Generation)

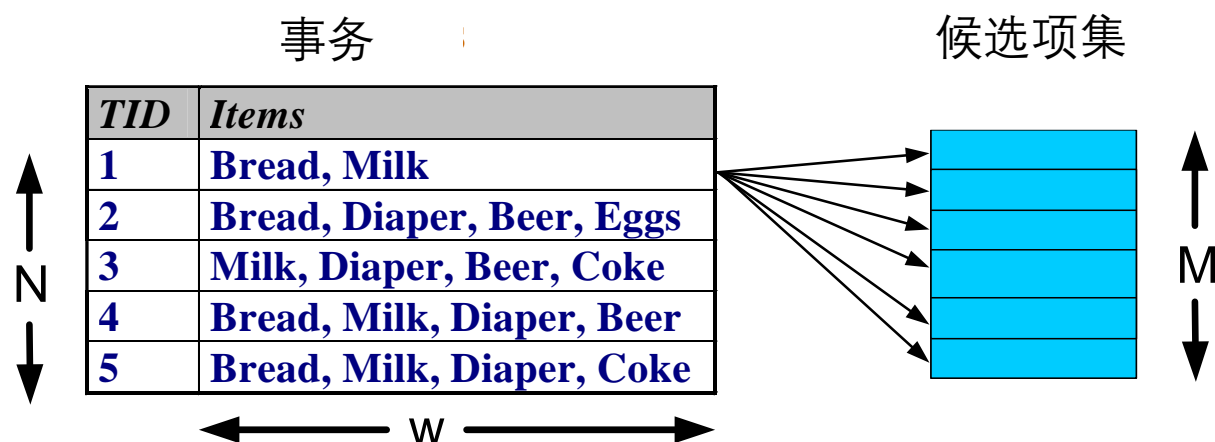
- 假设有 4 种商品 A, B, C, D, 可组合出很多种项集 (15个), 总数很大
- 如果商品超过100种, 网络的规模将非常庞大, 无法通过原始的暴力(brute force) 方法计算项集
- 一个包含 k 个项的数据集可能产生不包括空集在内的 $2^k - 1$ 个频繁项集

格结构 (lattice structure) 用来枚举所有可能的项集



项集生成 (Itemset Calculation)

- 原始暴力方法 (Brute-force 方法):
 - 把格结构中每个项集作为候选项集
 - 将每个候选项集和每个事务进行比较, 确定每个候选项集的支持度计数。



- 时间复杂度 $\sim O(NMW)$, 这种方法的开销可能非常大。
- 如果有 k 种商品, 所有可能的项集有 $M = 2^k - 1$ 次方个候选项集 (因为项集是非空子集, 故需要 -1), 需要探查的项集搜索空间是指数规模的

降低产生频繁项集计算复杂度的方法

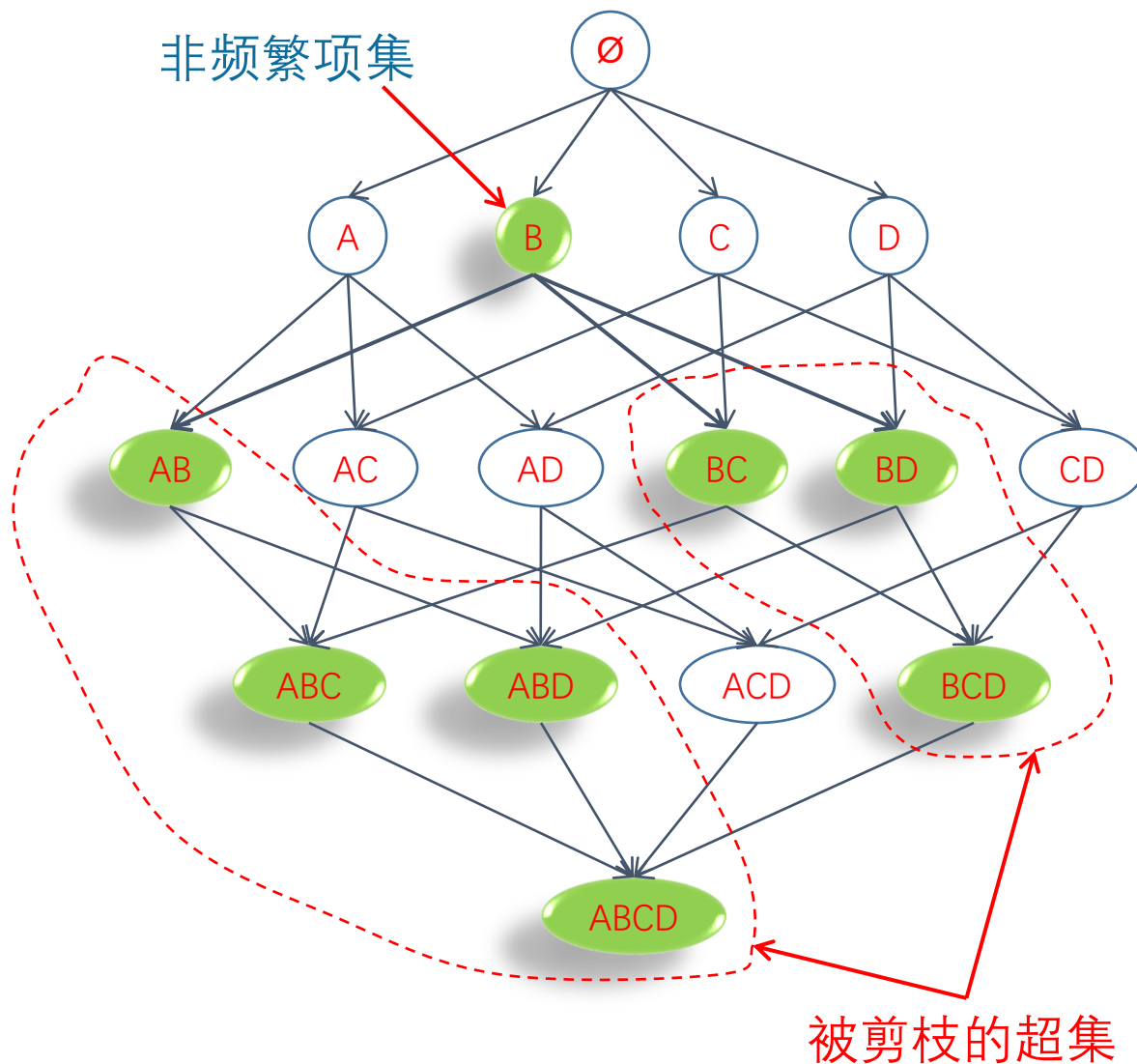
- 减少候选项集的数量 (M)
 - Apriori算法-先验原理(Apriori principle)
- 减少比较的次数 (NM)
 - 替代将每个候选项集与每个事务相匹配，可以使用更高级的数据结构，或存储候选项集或压缩数据集，来减少比较次数

Apriori 算法

- 数据挖掘中最著名的算法之一
- 一种发现频繁项集的基本算法
- 关键思路 (基于先验原理 (Apriori principle)):
 - 如果一个项集 X 是频繁的, 则它的所有非空子集 $Y \subset X$ 一定也是频繁的
 - {Milk, Bread} 是频繁的 \rightarrow {Milk}, {Bread} 是频繁的
 - 如果一个项集 X 是非频繁的, 则它的所有超集 $Y \supset X$ 也一定是非频繁的
 - {Coke} 是非频繁的 \rightarrow {Milk, Coke} 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 (support-based pruning)
 - 这种剪枝策略依赖于支持度度量的一个关键性质, 即一个项集的支持度决不会超过它的子集的支持度。这个性质也称为支持度度量的反单调性(anti-monotone), 即 $\text{support}(X \cup Y) \leq \text{support}(X)$

候选修剪(Candidate Pruning)

- 如果 **B** 是非频繁项集, 那么包含 **B** 的项集(超集)都是非频繁的



Apriori 算法一般过程 (General Procedure)

- 产生一个特定大小的项集 (如:每一个商品为一个项集).
- 扫描数据库一次以查看它们中的哪些是频繁的(去除非频繁的).
- 使用频繁项集来生成 $\text{size} = \text{size} + 1$ 的候选项集(单个商品两两组合).
- 迭代地找到基数从1到k的频繁项集.
- 避免生成已知不频繁的候选项集.
- 缺点: 需要多次扫描数据库.
- 高效的索引技术 (如Hash函数和位图) 可能会有所帮助.



Apriori 算法

C_k : 大小为 k 的候选项集

L_k : 大小为 k 的频繁项集

$L_1 \leftarrow \{\text{频繁项集}\}$ (找出所有单个项的频繁项集)

for ($k=1$; $L_k \neq \emptyset$; $k++$)

$C_{k+1} \leftarrow \text{candidate}(L_k)$ 连接步

for 每一个事务 t (扫描数据库中的每条记录)

$Q \leftarrow \{c \mid c \in C_{k+1} \wedge c \subseteq t\}$

$\text{count}[c] \leftarrow \text{count}[c] + 1, \quad \forall c \in Q$

end for

$L_{k+1} \leftarrow \{c \mid c \in C_{k+1} \wedge \text{count}[c]/N \geq \sigma\}$

end for

return $\bigcup_k L_k$

候选(candidates)

计数(counting)

过滤(filtering)

Apriori 算法总结

- **连接步**: 候选 k -项集 C_k 由频繁 $(k-1)$ -项集 L_{k-1} 与其自身连接生成
- **剪枝步**: 任意非频繁 $(k-1)$ -项集不可能是一个频繁 k -项集的子集
- Apriori 算法的频繁项集产生的部分有两个重要的特点:
 - 它是一个逐层算法。即从频繁**1-项集**到最长的频繁项集，它每次遍历项集格中的一层
 - 它使用产生-测试策略来发现频繁项集。**在每次迭代，新的候选项集由前一次迭代发现的频繁项集产生**，然后对每个候选的支持度进行计数，并与最小支持度阈值进行比较。
 - 该算法需要的总迭代次数是 $k_{\max}+1$ ，其中 k_{\max} 是频繁项集的最大长度

$L_k \rightarrow C_{k+1}$ 候选项集生成方法实例分析

频繁项集 $L_1 = \{\textcolor{red}{1}, \textcolor{red}{2}, \textcolor{red}{3}, \textcolor{red}{4}, \textcolor{red}{5}\}$ $L_2 = \{\{\textcolor{red}{1}, \textcolor{red}{2}\}, \{\textcolor{red}{2}, \textcolor{red}{3}\}\}$

C_3 候选项集产生方法1: 频繁1-项集与频繁2-项集进行连接

$$\{X \cup p \mid X \in L_k, p \in L_1, p \notin X\}$$

候选项集 $C_3 = \{\{\textcolor{red}{1}, \textcolor{red}{2}, \textcolor{red}{3}\}, \{\textcolor{red}{1}, \textcolor{red}{2}, \textcolor{red}{4}\}, \{\textcolor{red}{1}, \textcolor{red}{2}, \textcolor{red}{5}\}, \{\textcolor{red}{2}, \textcolor{red}{3}, \textcolor{red}{4}\}, \{\textcolor{red}{2}, \textcolor{red}{3}, \textcolor{red}{5}\}\}$ 不紧凑

↑
非频繁
{1,3}不在 L_2 中

↑
非频繁
{1,5}不在 L_2 中

C_3 候选项集产生方法2: 频繁2-项集与其自身进行连接

$$\{X \cup Y \mid X, Y \in L_k, |X \cap Y| = k - 1\}$$

条件: X 和 Y 只有一位不同

候选项集 $C_3 = \{\{\textcolor{red}{1}, \textcolor{red}{2}, \textcolor{red}{3}\}\}$ 不紧凑

↑
非频繁
(因{1,3}非频繁)



$L_k \rightarrow C_{k+1}$ (Apriori 算法真正采用的候选项集生成策略)

(1) Apriori 算法假设事务或项集里的各项已经预先按字典顺序进行排列

(2) 只对 L_k 中那些 X 和 Y 前 $k-1$ 项相同且第 k 项不同的频繁项集进行连接, 生成 C_{k+1} 的候选项集, 连接方法如下:

候选项集产生 $\{X \cup Y | X, Y \in L_k, X_i = Y_i, \forall i \in [1, k-1], X_k \neq Y_k\}$ 有序列表

$$L_2 = \{\{1, 2\}, \{2, 3\}\}$$

$$C_3 = \{\}$$

$$L_2 = \{\{1, 3\}, \{2, 3\}\}$$

$$C_3 = \{\}$$



$$L_2 = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$$

$$C_3 = \{\{1, 2, 3\}\}$$

$$L_2 = \{\{1, 2\}, \{1, 3\}\}$$

$$C_3 = \{\{1, 2, 3\}\}$$



不频繁

Apriori 算法例子,挖掘频繁项集, 要求最小支持度=50%(即支持度计数 ≥ 2)

数据库 D

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

扫描 D

C_1 项集	支持度计数
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1 项集	支持度计数
{1}	2
{2}	3
{3}	3
{5}	3

L_2 项集	支持度计数
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_2 项集	支持度计数
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

扫描 D

C_2 项集
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

C_3 项集
{2 3 5}

扫描 D

L_3 项集	支持度计数
{2 3 5}	2

注意{1,2,3}, {1,2,5}, {1,3,5} 不在 C_3 中

Apriori 算法例子, 由频繁项集构造强规则

数据库 D

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

要求规则最小支持度=50%(即支持度计数≥2), 置信度≥70%

频繁项集	支持度计数
{1}	2
{2}	3
{3}	3
{5}	3
{1,3}	2
{2,3}	2
{2,5}	3
{3,5}	2
{2,3,5}	2

{1,3}产生规则: **1→3(sup=2/4=50%, conf=2/2=100%)**

3→1(sup=2/4=50%, conf=2/3≈66.7%)

{2,3}产生规则: **2→3(sup=2/4=50%, conf=2/3≈66.7%)**

3→2(sup=2/4=50%, conf=2/3≈66.7%)

{2,5}产生规则: **2→5(sup=3/4=75%, conf=3/3=100%)**

5→2(sup=3/4=75%, conf=3/3=100%)

{3,5}产生规则: 3→5(sup=2/4=50%, conf=2/3≈66.7%)

5→3(sup=2/4=50%, conf=2/3≈66.7%)

{2,3,5}产生规则: 2→3∪5(sup=2/4=50%, conf=2/3≈66.7%)

3→5∪2(sup=2/4=50%, conf=2/3≈66.7%)

5→2∪3(sup=2/4=50%, conf=2/3≈66.7%)

2∪3→5(sup=2/4=50%, conf=2/2=100%)

2∪5→3(sup=2/4=50%, conf=2/3≈66.7%)

3∪5→2(sup=2/4=50%, conf=2/3=100%)

强关联规则:

1→3(50%, 100%)

2→5(75%, 100%)

5→2(75%, 100%)

2∪3→5(50%, 100%)

3∪5→2(50%, 100%)

数据库 D

TID	项集
10	A B C
20	A C
30	A D
40	B E F

- (1) 根据给定的数据库D计算所有的频繁项集
- (2) 根据频繁项集给出满足最小支持度和最小置信度的强关联规则

频繁项集最小支持度=50%(即支持度计数 ≥ 2)

关联规则支持度 $\geq 50\%$ (即支持度计数 ≥ 2),
置信度 $\geq 70\%$

Apriori 算法例子,挖掘频繁项集, 要求最小支持度=50%(即支持度计数 ≥ 2)

数据库 D

TID	项集
10	A B C
20	A C
30	A D
40	B E F

扫描 D

C_1

项集	支持度计数
{A}	3
{B}	2
{C}	2
{D}	1
{E}	1
{F}	1

L_1

项集	支持度计数
{A}	3
{B}	2
{C}	2

L_2

项集	支持度计数
{A C}	2

C_2

项集	支持度计数
{A B}	1
{A C}	2
{B C}	1

扫描 D

C_2

项集
{A B}
{A C}
{B C}

频繁项集	支持度
{A}	75%
{B}	50%
{C}	50%
{A, C}	50%

要求最小支持度=50%, 最小置信度=70%

对于规则 $A \rightarrow C$:

支持度: $\text{Support}(A \cup C) = 50\%$

置信度: $\text{Support}(A \cup C) / \text{Support}(A) = 66.6\%$

对于规则 $C \rightarrow A$:

支持度: $\text{Support}(C \cup A) = 50\%$

置信度: $\text{Support}(C \cup A) / \text{Support}(C) = 100\%$

Apriori-Gen Algorithm – Clothing Example

- Given: 20 clothing transactions; **supp**=20%, **conf**=50%
- Generate association rules using the Apriori algorithm

Transaction	Items	Transaction	Items
t_1	Blouse	t_{11}	TShirt
t_2	Shoes, Skirt, TShirt	t_{12}	Blouse, Jeans, Shoes, Skirt, TShirt
t_3	Jeans, TShirt	t_{13}	Jeans, Shoes, Shorts, TShirt
t_4	Jeans, Shoes, TShirt	t_{14}	Shoes, Skirt, TShirt
t_5	Jeans, Shorts	t_{15}	Jeans, TShirt
t_6	Shoes, TShirt	t_{16}	Skirt, TShirt
t_7	Jeans, Skirt	t_{17}	Blouse, Jeans, Skirt
t_8	Jeans, Shoes, Shorts, TShirt	t_{18}	Jeans, Shoes, Shorts, TShirt
t_9	Jeans	t_{19}	Jeans
t_{10}	Jeans, Shoes, TShirt	t_{20}	Jeans, Shoes, Shorts, TShirt

- Scan1: Find all 1-itemsets. Identify the frequent ones.**

Candidates: ~~Blouse~~, Jeans, Shoes, Shorts, Skirt, Tshirt

Support: ~~3/20~~ 14/20 10/20 5/20 6/20 14/20

Frequent (Large): Jeans, Shoes, Shorts, Skirt, Tshirt

Join the frequent items – combine items with each other to generate candidate pairs

Clothing Example – cont.1

Jeans, Shoes, Shorts, Skirt, Tshirt

- Scan2: 10 candidate 2-itemsets were generated. Find the frequent ones.**

~~{Jeans, Shoes}:7/20~~ ~~{Shoes, Short}:4/20~~ ~~{Short, Skirt}: 0/20~~ {Skirt, TShirt}: 4/20

~~{Jeans, Short} :5/20~~ ~~{Shoes, Skirt}: 3/20~~ {Short, TShirt}: 4/20

~~{Jeans, Skirt} :3/20~~ {Shoes, TShirt}: 10/20

{Jeans, TShirt}:9/20 4/20

7 frequent itemsets are found out of 10.

Scan	Candidates	Large Itemsets
1	{Blouse}, {Jeans}, {Shoes}, {Shorts}, {Skirt}, {TShirt}	{Jeans}, {Shoes}, {Shorts} {Skirt}, {Tshirt}
2	{Jeans, Shoes}, {Jeans, Shorts}, {Jeans, Skirt}, {Jeans, TShirt}, {Shoes, Shorts}, {Shoes, Skirt}, {Shoes, TShirt}, {Shorts, Skirt}, {Shorts, TShirt}, {Skirt, TShirt}	{Jeans, Shoes}, {Jeans, Shorts}, {Jeans, TShirt}, {Shoes, Shorts}, {Shoes, TShirt}, {Shorts, TShirt}, {Skirt, TShirt}
3	{Jeans, Shoes, Shorts}, {Jeans, Shoes, TShirt}, {Jeans, Shorts, TShirt}, {Shoes, Shorts, TShirt},	{Jeans, Shoes, Shorts}, {Jeans, Shoes, TShirt}, {Jeans, Shorts, TShirt}, {Shoes, Shorts, TShirt}
4	{Jeans, Shoes, Shorts, TShirt}	{Jeans, Shoes, Shorts, TShirt}
5	∅	∅

Everyone is combined with each other

2 sets are joined if they have 1 item in common (i.e. 1 item different)

2 sets are joined if they have 2 item in common (i.e. 1 item different)

Clothing Example – cont.2

- The next step is to use the large itemsets and generate association rules

- **conf=50%**

- The set of large itemsets is

$L = \{\{\text{Jeans}\}, \{\text{Shoes}\}, \{\text{Shorts}\}, \{\text{Skirt}\}, \{\text{TShirt}\}, \{\text{Jeans, Shoes}\}, \{\text{Jeans, Shorts}\}, \{\text{Jeans, TShirt}\}, \{\text{Shoes, Shorts}\}, \{\text{Shoes, TShirt}\}, \{\text{Shorts, TShirt}\}, \{\text{Skirt, TShirt}\}, \{\text{Jeans, Shoes, Shorts}\}, \{\text{Jeans, Shoes, TShirt}\}, \{\text{Jeans, Shorts, TShirt}\}, \{\text{Shoes, Shorts, TShirt}\}, \{\text{Jeans, Shoes, Shorts, TShirt}\}\}$

- We ignore the first 5 as they do not consists of 2 nonempty subsets of large itemsets. We test all the others, e.g.:

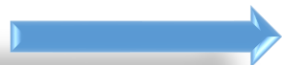
$$\text{conf}(\text{Jeans} \rightarrow \text{Shoes}) = \frac{\text{sup}(\{\text{Jeans, Shoes}\})}{\text{sup}(\{\text{Jeans}\})} = \frac{7/20}{14/20} = 50\% \geq \text{conf}$$

购买 *Shoes* 的先验概率 $P(\text{Shoes}) = \frac{10}{20} = 50\%$ 上述规则 $\text{Jeans} \rightarrow \text{Shoes}$ 无意义

有效推荐

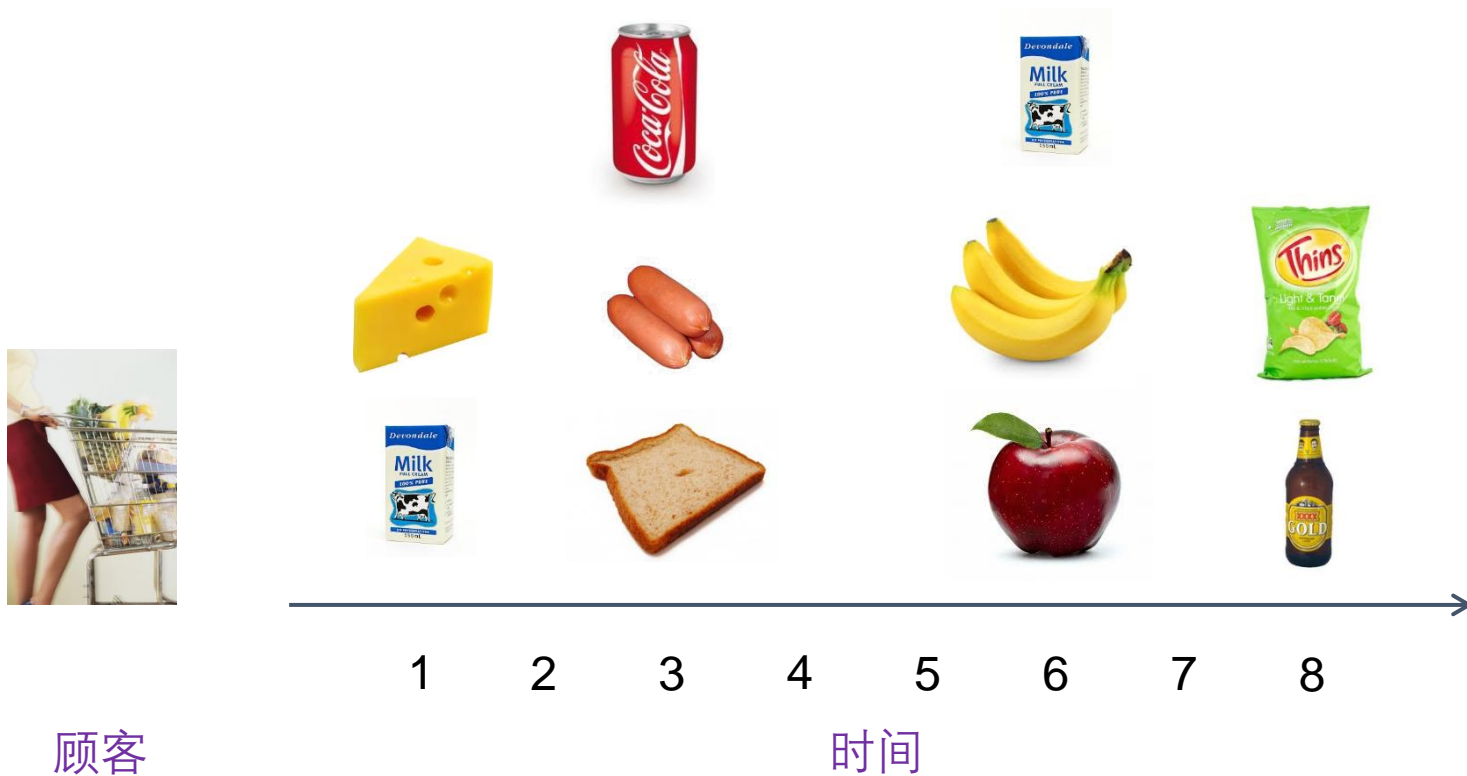


这孩子, 没救了..



序列模式(Sequential Pattern)挖掘

序列模式寻找的是事务(项)之间时间上的相关性。



序列(Sequence)模式挖掘

- 关联规则的挖掘不考虑事务的顺序.
- 购物篮数据中包含商品何时被顾客购买的时间信息
- 在许多应用中，这种排序很重要
 - 在购物篮分析中，知道人们是否按时间顺序购买一些物品很有意思
 - 例如先买床，然后再买床单
 - 客户购买行为模式预测
 - 通过使用时间信息，将顾客在一段时间内的购物拼接成实物序列
 - 在Web使用挖掘中，从用户的页面访问序列中查找网站中的用户的导航模式是有用的
- 市场营销
- 客户流量分析
- 股票价格变动
- 宏观经济预测

基本概念

- 设 $I = \{i_1, i_2, \dots, i_m\}$ 是所有项 (Item) i_j ($j=1, 2, \dots, m$) 的集合。
- **序列**: 元素(项集, 事务)的有序列表
- 元素/项集: 非空项集 $X \subset I$, 表示序列 s 为 $s = \langle s_1 \ s_2 \ \dots \ s_n \rangle$, 其中 s_i 是一个或多个项(**事件**)的集族, 也叫作 s 的一个元素 (element)
- 序列的一个元素(或项集)表示为 $\{x_1, x_2, \dots, x_k\}$, 其中 $x_j \in I$ 是一个项
- 不失一般性, 假设序列中元素的项是按**字典顺序**排列的
- **大小**(size): 序列的大小是序列中元素(或项集)的个数
- **长度**: 一个序列的长度是序列中所有项的个数
 - 长度为 k 的序列称为 **k -序列**
- 称 $t = \langle t_1 \ t_2 \ \dots \ t_m \rangle$ 是 $s = \langle s_1 \ s_2 \ \dots \ s_n \rangle$ 的一个**子序列**如果存在整数 $1 \leq j_1 < j_2 < \dots < j_m \leq n$ 使得 $t_1 \subseteq s_{j_1}, t_2 \subseteq s_{j_2}, \dots, t_m \subseteq s_{j_m}$. 我们也称 s 是 t 的**超序列**, 或 s **包含** t , 记为 $t \subset s$

例子

- 设 $I = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- 序列 $\langle \{3\} \{4,5\} \{8\} \rangle$ 包含在 $\langle \{6\} \{3,7\} \{9\} \{4,5,8\} \{3,8\} \rangle$ (或者说前者是后者的子序列)
 - $\{3\} \subseteq \{3,7\}, \{4,5\} \subseteq \{4,5,8\}, \{8\} \subseteq \{3,8\}$
 - 但是 $\langle \{3\}, \{8\} \rangle$ 并不包含在 $\langle \{3,8\} \rangle$ 中, 反之也成立
 - 序列 $\langle \{3\} \{4,5\} \{8\} \rangle$ 的大小是3, 序列长度是4
- 如果一个序列只有一个项集, 则括号可以省略
- 看 t 是否为 s 的子序列

t	s	Y/N
$\langle \{2\} \{3, 6\} \{8\} \rangle$	$\langle \{2, 4\} \{3, 6, 5\} \{8\} \rangle$	Yes
$\langle \{2\} \{8\} \rangle$	$\langle \{2, 4\} \{3, 6, 5\} \{8\} \rangle$	Yes
$\langle \{1\} \{2\} \rangle$	$\langle \{1, 2\} \{3, 4\} \rangle$	No
$\langle \{2\} \{4\} \rangle$	$\langle \{2, 4\} \{2, 4\} \{2, 5\} \rangle$	Yes

子序列-例子

- 假定没有时限约束，列举包含在下面的数据序列中的所有4-子序列

$\langle \{1,3\}, \{2\}, \{2,3\}, \{4\} \rangle$

- 列举如下：
- $\langle \{1, 3\} \{2\} \{2\} \rangle$, $\langle \{1, 3\} \{2\} \{3\} \rangle$, $\langle \{1, 3\} \{2\} \{4\} \rangle$,
- $\langle \{1, 3\} \{2, 3\} \rangle$, $\langle \{1, 3\} \{3\} \{4\} \rangle$, $\langle \{1\} \{2\} \{2, 3\} \rangle$,
- $\langle \{1\} \{2\} \{2\} \{4\} \rangle$, $\langle \{1\} \{2\} \{3\} \{4\} \rangle$, $\langle \{1\} \{2, 3\} \{4\} \rangle$,
- $\langle \{3\} \{2\} \{2, 3\} \rangle$, $\langle \{3\} \{2\} \{2\} \{4\} \rangle$, $\langle \{3\} \{2\} \{3\} \{4\} \rangle$,
- $\langle \{3\} \{2, 3\} \{4\} \rangle$, $\langle \{2\} \{2, 3\} \{4\} \rangle$

子序列-例子

- 假定没有时限约束，列举包含在下面的数据序列中的所有3个元素 (项集) 的子序列

$\langle \{1,3\}, \{2\}, \{2,3\}, \{4\} \rangle$

- 列举如下：


- $\langle \{1, 3\} \{2\} \{2, 3\} \rangle$, $\langle \{1, 3\} \{2\} \{4\} \rangle$
- $\langle \{1, 3\} \{3\} \{4\} \rangle$, $\langle \{1, 3\} \{2\} \{2\} \rangle$
- $\langle \{1, 3\} \{2\} \{3\} \rangle$, $\langle \{1, 3\} \{2, 3\} \{4\} \rangle$
- $\langle \{1\} \{2\} \{2, 3\} \rangle$, $\langle \{1\} \{2\} \{4\} \rangle$
- $\langle \{1\} \{3\} \{4\} \rangle$, $\langle \{1\} \{2\} \{2\} \rangle$
- $\langle \{1\} \{2\} \{3\} \rangle$, $\langle \{1\} \{2, 3\} \{4\} \rangle$
- $\langle \{3\} \{2\} \{2, 3\} \rangle$, $\langle \{3\} \{2\} \{4\} \rangle$
- $\langle \{3\} \{3\} \{4\} \rangle$, $\langle \{3\} \{2\} \{2\} \rangle$
- $\langle \{3\} \{2\} \{3\} \rangle$, $\langle \{3\} \{2, 3\} \{4\} \rangle$

序列挖掘的目标

- 设 S 是包含一个或多个数据序列的数据集
- **数据序列**是指与单个数据对象相关联的事件的有序列表
- **序列 s 的支持度**是包含 s 的所有数据序列所占的比例
- 给定输入数据序列集(或序列数据库) S , 挖掘序列模式的问题是求满足用户指定最小支持度的所有序列, 即该子序列在序列集中的出现频率大于或等于用户指定的最小支持度阈值
- **频繁序列 (或序列模式)**: 如果序列 s 在序列数据库 S 中的支持度大于或等于用户指定的最小支持度阈值, 则称序列 s 为**频繁序列 (或序列模式)**
- **序列的支持度计数**是 S 中包含该序列的总数据序列个数
- 长度为1的序列模式记为1-模式
- 系统规定: 由于同一个元素中的项集之间排列没有顺序, 为了表达的唯一性, 我们将同一个元素内部的不同项集按照字典顺序排列

序列的支持度-例子

区分不同的客户



CID	时间戳	项(事件)
A	1	1, 2, 4
A	2	2, 3
A	3	5
B	1	1, 2
B	2	2, 3, 4
C	1	1, 2
C	2	2, 3, 4
C	3	2, 4, 5
D	1	2
D	2	3, 4
D	3	4, 5
E	1	1, 3
E	2	2, 4, 5

数据集 S 含5个数据序列,
A、B、C、D、E各一个

支持度	
$\langle \{1, 2\} \rangle$	60%
$\langle \{2, 3\} \rangle$	60%
$\langle \{2, 4\} \rangle$	80%
$\langle \{3\} \{5\} \rangle$	80%
$\langle \{1\} \{2\} \rangle$	80%
$\langle \{2\} \{2\} \rangle$	60%
$\langle \{1\} \{2, 3\} \rangle$	60%
$\langle \{2\} \{2, 3\} \rangle$	60%
$\langle \{1, 2\} \{2, 3\} \rangle$	60%

$$\text{sup}(\langle \{1, 2\} \rangle) = \frac{\# \langle \{1, 2\} \rangle}{\#S} = \frac{3}{5} = 60\%$$

$$\text{sup}(\langle \{1\}, \{2\} \rangle) = \frac{\# \langle \{1\}, \{2\} \rangle}{\#S} = \frac{4}{5} = 80\%$$

候选空间 (Candidate Space)

- 给定: 1-项集 {Milk} {Bread}
- 生成 2-项集: {Bread, Milk}
- 生成 2-序列:
 - $\langle \{Bread, Milk\} \rangle$
 - $\langle \{Bread\} \{Milk\} \rangle$
 - $\langle \{Milk\} \{Bread\} \rangle$
 - $\langle \{Bread\} \{Bread\} \rangle$
 - $\langle \{Milk\} \{Milk\} \rangle$
- 顺序在序列中很重要, 但对于项集则不然.
- 对于1000 个项: $1000 \times 1000 + \frac{1000 \times 999}{2} = 1499500$ 个2-序列
- 序列搜索空间比以前项集的大得多.
- 怎样有效生成候选序列?

候选序列空间 (Candidate Space)

- 候选序列的个数比候选项集的个数大得多。产生更多候选序列的原因有下面两个
 - 一个项在项集中最多出现一次，但一个事件可以在序列中出现多次。给定两个项 i_1 和 i_2 ，只能产生一个候选2-项集 $\{i_1, i_2\}$ ，但却可以产生许多候选2-序列，如 $\langle \{i_1, i_2\} \rangle$, $\langle \{i_1\}i_2 \rangle$, $\langle \{i_2, i_1\} \rangle$, $\langle \{i_1, i_1\} \rangle$ 。
 - 次序在序列中是重要的，但在项集中不重要。例如， $\{i_1, i_2\}$ 和 $\{i_2, i_1\}$ 表示同一个项集，而 $\langle \{i_1\}i_2 \rangle$ 和 $\langle \{i_2\}i_1 \rangle$ 对应于不同的序列，因此必须分别产生。
- 先验原理 (Apriori 算法) 对序列数据成立
 - 如果一个 k -序列是频繁的，则它的所有 $(k-1)$ -子序列也一定是频繁的。

序列模式发现的 Apriori 算法

算法 序列模式发现的类 Apriori 算法

```
1:  $k = 1$ .
2:  $F_k = \{i \mid i \in I \wedge \sigma(\{i\})/N \geq \text{minsup}\}$ .      {找出所有的频繁 1-序列。}
3: repeat
4:    $k = k + 1$ .
5:    $C_k = \text{apriori-gen}(F_{k-1})$ .      {产生候选  $k$ -序列。}
6:   for 每个数据序列  $t \in T$  do
7:      $C_t = \text{subsequence}(C_k, t)$ .      {识别包含在  $t$  中的所有候选。}
8:     for 每个候选  $k$ -序列  $c \in C_t$  do
9:        $\sigma(c) = \sigma(c) + 1$ .      {支持度计数增值。}
10:    end for
11:  end for
12:   $F_k = \{c \mid c \in C_k \wedge \sigma(c)/N \geq \text{minsup}\}$ .      {提取频繁  $k$ -序列。}
13: until  $F_k = \emptyset$ .
14:  $\text{Answer} = \bigcup F_k$ .
```

候选产生

- 一对频繁(k-1)-序列合并，产生候选k-序列。
- 为了避免重复产生候选，传统的Apriori算法仅当前k-1项相同且第k项不同时才合并一对频繁k-项集。类似的方法可以用于序列。
- 例子
 - $\langle \{1\} \{2\} \{3\} \{4\} \rangle$ 是通过合并 $\langle \{1\} \{2\} \{3\} \rangle$ 和 $\langle \{2\} \{3\} \{4\} \rangle$ 得到。由于事件3和事件4属于第二个序列的不同元素，它们在合并后序列中也属于不同的元素。
 - $\langle \{1\} \{5\} \{3,4\} \rangle$ 通过合并 $\langle \{1\} \{5\} \{3\} \rangle$ 和 $\langle \{5\} \{3,4\} \rangle$ 得到。由于事件3和事件4属于第二个序列的相同元素，4被合并到第一个序列的最后一个元素中。

序列合并过程

序列 $s^{(1)}$ 与另一个序列 $s^{(2)}$ 合并, 仅当从 $s^{(1)}$ 中去掉第一个事件得到的子序列与从 $s^{(2)}$ 中去掉最后一个事件得到的子序列相同。结果候选是序列 $s^{(1)}$ 与 $s^{(2)}$ 的最后一个事件的连接。 $s^{(2)}$ 的最后一个事件可以作为最后一个事件合并到 $s^{(1)}$ 的最后一个元素中, 也可以作为一个不同的元素, 取决于如下条件:

(1) 如果 $s^{(2)}$ 的最后两个事件属于相同的元素, 则 $s^{(2)}$ 的最后一个事件在合并后的序列中是 $s^{(1)}$ 的最后一个元素的一部分。

(2) 如果 $s^{(2)}$ 的最后两个事件属于不同的元素, 则 $s^{(2)}$ 的最后一个事件在合并后的序列中成为连接到 $s^{(1)}$ 的尾部的单独元素。

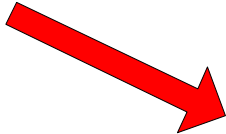
候选生成 (Candidate Generation)

- 候选剪枝
 - 一个候选 k -序列被剪枝，如果它的 $(k-1)$ -序列最少有一个是非频繁的。
 - 例如，假设 $\langle \{1\} \{2\} \{3\} \{4\} \rangle$ 是一个候选4-序列。我们需要检查 $\langle \{1\} \{2\} \{4\} \rangle$ 和 $\langle \{1\} \{3\} \{4\} \rangle$ 是否是频繁3-序列。由于它们都不是频繁的，因此可以删除候选 $\langle \{1\} \{2\} \{3\} \{4\} \rangle$ 。
- 支持度计数
 - 在支持度计数期间，算法将枚举属于一个特定数据序列的所有候选 k -序列。
 - 计数之后，算法将识别出频繁 k -序列，并可以丢弃其支持度计数小于最小支持度阈值 minsup 的候选。

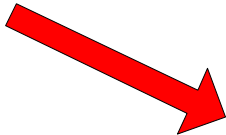
候选生成(Candidate Generation) (GSP算法)

- 当且仅当通过丢弃 s_1 中的第一项获得的子序列与通过丢弃 s_2 中的最后一项而获得的子序列相同时，序列 s_1 与另一序列 s_2 合并

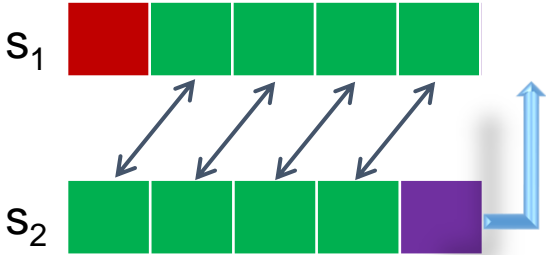
频繁3-序列
<{1} {2} {3}>
<{1} {2, 5}>
<{1} {5} {3}>
<{2} {3} {4}>
<{2, 5} {3}>
<{3} {4} {5}>
<{5} {3, 4}>



候选序列
<{1} {2} {3} {4}>
<{1} {2, 5} {3}>
<{1} {5} {3, 4}>
<{2} {3} {4} {5}>
<{2, 5} {3, 4}>



剪枝
<{1} {2, 5} {3}>



候选序列生成

■考虑以下频繁3-序列: $\langle \{1, 2, 3\} \rangle$, $\langle \{1, 2\}\{3\} \rangle$, $\langle \{1\}\{2, 3\} \rangle$,
 $\langle \{1, 2\}\{4\} \rangle$, $\langle \{1, 3\}\{4\} \rangle$, $\langle \{1, 2, 4\} \rangle$, $\langle \{2, 3\}\{3\} \rangle$, $\langle \{2, 3\}\{4\} \rangle$,
 $\langle \{2\}\{3\}\{3\} \rangle$, 和 $\langle \{2\}\{3\}\{4\} \rangle$

- (1) 列举出候选生成步骤产生的所有候选4-序列

所有候选4-序列列举如下:

$\langle \{1, 2, 3\}\{3\} \rangle$, $\langle \{1, 2, 3\}\{4\} \rangle$, $\langle \{1, 2\}\{3\}\{3\} \rangle$, $\langle \{1, 2\}\{3\}\{4\} \rangle$,
 $\langle \{1\}\{2, 3\}\{3\} \rangle$, $\langle \{1\}\{2, 3\}\{4\} \rangle$

- (2) 列出候选剪枝步骤剪掉的所有候选4-序列(假定没有时限约束)。
如果没有时间限制, 则所有候选子序列都必须频繁。因此, 经过修剪掉的候选子序列(非频繁)为:

$\langle \{1, 2, 3\}\{3\} \rangle$, $\langle \{1, 2\}\{3\}\{3\} \rangle$, $\langle \{1, 2\}\{3\}\{4\} \rangle$,
 $\langle \{1\}\{2, 3\}\{3\} \rangle$, $\langle \{1\}\{2, 3\}\{4\} \rangle$

剪枝后的候选序列为: $\langle \{1, 2, 3\}\{4\} \rangle$

- 教科书

- 陈封能等, 数据挖掘导论, Chapter 5-6, 机械工业出版社, 2019
- 韩家炜等, 数据挖掘: 概念与技术, Chapter 6, Morgan Kaufmann.

- 核心论文

- J. Han, J. Pei, Y. Yin and R. Mao (2004) “Mining frequent patterns without candidate generation: A frequent-pattern tree approach”. *Data Mining and Knowledge Discovery*, Vol. 8(1), pp. 53-87.
- R. Agrawal and R. Srikant (1995) “Mining sequential patterns”. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE)*, pp. 3-14.
- R. Agrawal and R. Srikant (1994) “Fast algorithms for mining association rules”. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pp. 487-499.
- R. Agrawal, T. Imielinski, and A. Swami (1993) “Mining association rules between sets of items in large databases”. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp. 207-216.