一、 什么是关联规则

什么我们去沃尔玛超市会发现一个很有趣的现象：货架上啤酒与尿布竟然放在一起售卖，这看似两者毫不相关的东西，为什么会放在一起售卖呢？

原来，在美国，妇女们经常会嘱咐她们的丈夫下班以后给孩子买一点尿布回来，而丈夫在买完尿布后，大都会顺手买回一瓶自己爱喝的啤酒（由此看出美国人爱喝酒）。商家通过对一年多的原始交易记录进行详细的分析，发现了这对神奇的组合。于是就毫不犹豫地将尿布与啤酒摆放在一起售卖，通过它们的关联性，互相促进销售。“啤酒与尿布”的故事一度是营销界的神话。

关联规则挖掘的目的是找出事物之间存在的隐藏的关系，比如经典的案例啤酒和尿布的的故事，用我们人的思维来思考的话，男性在买尿布的时候会买几瓶啤酒，这二者并没有什么因果关系。然而通过对海量数据进行关联分析，却能够发现这个有趣的知识，在超市调整货架后，明显的提升了超市啤酒尿布的销量。

二、 基本概念

对于 A->B

- 支持度：P(AB)，既有 A 又有 B 的概率
- 置信度：P(B|A)，在 A 发生的事件中同时发生 B 的概率$\mathrm{P(B|A)} = \frac{P(AB)}{P(A)}$ 例如购物篮分析：牛奶 ⇒ 面包

  例子：[支持度：3%，置信度：40%]

  支持度 3%：意味着 3%顾客同时购买牛奶和面包

置信度 40%：意味着购买牛奶的顾客 40%也购买面包

- k 项集：如果事件 A 中包含 k 个元素，那么称这个事件 A 为 k 项集，事件 A 满足最小支持度阈值的事件称为频繁 k 项集。
- 同时满足最小支持度阈值和最小置信度阈值的规则称为强规则
- K 维数据项集L_K是频繁项集的必要条件是它所有 K-1 维子项集也为频繁项集，记为$L_K - 1$
- 如果 K 维数据项集 LK 的任意一个 K-1 维子集L_K-1，不是频繁项集，则 K 维数据项集L_K本身也不是最大数据项集。
- L_K是 K 维频繁项集，如果所有 K-1 维频繁项集合L_K-1 中包含L_K的 K-1 维子项集的个数小于 K，则L_K不可能是 K 维最大频繁数据项集。

三、 Apriori 算法简介

Apriori 算法过程分为两个步骤：

- 通过迭代，检索出事务数据库中的所有频繁项集，即支持度不低于用户设定的阈值的项集；
- 利用频繁项集构造出满足用户最小信任度的规则。

具体做法就是：

首先找出频繁 1-项集，记为L_1；然后利用L_1来产生候选项集C_2，对C_2中的项进行判定挖掘出L_2，即频繁 2-项集；不断如此循环下去直到无法发现更多的频繁 k-项集为止。每挖掘一层L_k就需要扫描整个数据库一遍。算法利用了一个性质：

任一频繁项集的所有非空子集也必须是频繁的。也就是说，生成一个 k-itemset 的候选项时 ,如果这个候选项有子集不在(k-1)-itemset(已经确定是 frequent 的)中时，那么这个候选项就不需要和支持度判断，直接删除。具体而言：

1. 连接步

为找出L_k（所有的频繁 k 项集的集合）, 通过将L_{k-1}（所有的频繁 k-1 项集的集合）与自身连接产生候选 k 项集的集合。候选集合记作C_k。设L_1和L_2是L_{k-1}中的成员。记$L_i[j]$表示L_i中的第 j 项。假设 Apriori 算法对事务或项集中的项按字典次序排序 ,即对于(k-1)项集L_i ,$L_i[1]<\ L_i[2]<\ldots\ldots\ldots<\ L_i[k-1]$。将$L_{k-1}$与自身连接 ,如果$(L_1[1]=\ L_2[1])$&&$(\ L_1[2]=\ L_2[2])$&&……&&$(L_1[k-2]=\ L_2[k-2])$&&$(\ L_1[k-1]<\ L_2[k-1])$, 那认为L_1和L_2是可连接。连接L_1和L_2产生的结果是$\{L_1[1],\ L_1[2],\ldots\ldots,\ L_1[k-1],\ L_2[k-1]\}$。

2. 剪枝步

C_k是L_k的超集，也就是说，C_k的成员可能是也可能不是频繁的。通过扫描所有的事务（交易）, 确定C_k中每个候选的计数 ,判断是否小于最小支持度计数，如果不是，则认为该候选是频繁的。为了压缩C_k,可以利用 Apriori 性质：任一频繁项集的所有非空子集也必须是频繁的，反之，如果某个候选的非空子集不是频繁的，那么该候选肯定不是频繁的，从而可以将其从C_k中删除。

四、 Apriori 算法实例

假设有一个数据库 D，其中有 4 个事务记录，如下图所示，预定最小支持度 minSupport=2。

TID	Items
T1	I1,I3,I4
T2	I2,I3,I5
T3	I1,I2,I3,I5
T4	I2,I5

1. 扫描 D，对每个候选项进行支持度计数得到表C_1:

项集	支持度计数
{I1}	2
{I2}	3
{I3}	3
{I4}	1
{I5}	3

2. 比较候选项支持度计数与最小支持度 minSupport，产生 1 维最大项目集L_1：

项集	支持度计数
{I1}	2
{I2}	3
{I3}	3
{I5}	3

3. 由L_1产生候选项集C_2：

项集
{I1,I2}
{I1,I3}
{I1,I5}
{I2,I3}
{I2,I5}
{I3,I5}

4. 扫描 D，对每个候选项集进行支持度计数:

项集	支持度计数
{I1,I2}	1
{I1,I3}	2
{I1,I5}	1
{I2,I3}	2
{I2,I5}	3
{I3,I5}	2

5. 比较候选项支持度计数与最小支持度 minSupport，产生 2 维最大项目集L_2：

项集	支持度计数
{I1,I3}	2
{I2,I3}	2
{I2,I5}	3
{I3,I5}	2

6. 由L_2产生候选项集C_3：

项集
{I2,I3,I5}

7. 比较候选项支持度计数与最小支持度 minSupport，产生 3 维最大项目集L_3：

项集	支持度计数
{I2,I3,I5}	2

8. 算法终止

五、 Apriori 算法优缺点总结

优点

- 适合稀疏数据集。
- 算法原理简单，易实现。
- 适合事务数据库的关联规则挖掘。

缺点

- 可能产生庞大的候选集。
- 算法需多次遍历数据集，算法效率低，耗时。