关联分析及常用算法

一． 基本概念

关联分析又称关联挖掘，就是在交易数据、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。

或者说，关联分析是发现交易数据库中不同商品（项）之间的联系

关联分析是一种简单、实用的分析技术，就是发现存在于大量数据集中的关联性或相关性，从而描述了一个事物中某些属性同时出现的规律和模式。

关联分析是从大量数据中发现项集之间有趣的关联和相关联系。关联分析的一个典型例子是购物篮分析。该过程通过发现顾客放入其购物篮中的不同商品之间的联系，分析顾客的购买习惯。通过了解哪些商品频繁地被顾客同时购买，这种关联的发现可以帮助零售商制定营销策略。其他的应用还包括价目表设计、商品促销、商品的排放和基于购买模式的顾客划分。

可从数据库中关联分析出形如“由于某些事件的发生而引起另外一些事件的发生”之类的规则。如“67%的顾客在购买啤酒的同时也会购买尿布”，因此通过合理的啤酒和尿布的货架摆放或捆绑销售可提高超市的服务质量和效益。又如“‘C 语言’课程优秀的同学，在学习‘数据结构’时为优秀的可能性达 88%”，那么就可以通过强化“C 语言”的学习来提高教学效果

二． 常用关联分析算法

Apriori 算法

Apriori 算法是挖掘产生布尔关联规则所需频繁项集的基本算法，也是最著名的关联规则挖掘算法之一。Apriori 算法就是根据有关频繁项集特性的先验知识而命名的。它使用一种称作逐层搜索的迭代方法，k—项集用于探索（k+1）—项集。首先，找出频繁 1—项集的集合．记做 L1，L1 用于找出频繁 2—项集的集合 L2，再用于找出 L3，如此下去，直到不能找到频繁 k—项集。找每个 Lk 需要扫描一次数据库。

为提高按层次搜索并产生相应频繁项集的处理效率，Apriori 算法利用了一个重要性质，并应用 Apriori 性质来帮助有效缩小频繁项集的搜索空间。

Apriori 性质：一个频繁项集的任一子集也应该是频繁项集。证明根据定义，若一个项集 I 不满足最小支持度阈值 min_sup，则 I 不是频繁的，即 P（I）<min_sup。若增加一个项 A 到项集 I 中，则结果新项集（I∪A）也不是频繁的，在整个事务数据库中所出现的次数也不可能多于原项集 I 出现的次数，因此 P（I∪A）<min_sup，即（I∪A）也不是频繁的。这样就可以根据逆反公理很容易地确定 Apriori 性质成立。

针对 Apriori 算法的不足，对其进行优化：

1）基于划分的方法。该算法先把数据库从逻辑上分成几个互不相交的块，每次单独考虑一个分块并对它生成所有的频繁项集，然后把产生的频繁项集合并，用来生成所有可能的频繁项集，最后计算这些项集的支持度。这里分块的大小选择要使得每个分块可以被放入主存，每个阶段只需被扫描一次。而算法的正确性是由每一个可能的频繁项集至少在某一个分块中是频繁项集保证的。

上面所讨论的算法是可以高度并行的。可以把每一分块分别分配给某一个处理器生成频繁项集。产生频繁项集的每一个循环结束后 .处理器之间进行通信来产生全局的候选是一项集。通常这里的通信过程是算法执行时间的主要瓶颈。而另一方面，每个独立的处理器生成频繁项集的时间也是一个瓶颈。其他的方法还有在多处理器之间共享一个杂凑树来产生频繁项集，更多关于生成频繁项集的并行化方法可以在其中找到。

2）基于 Hash 的方法。Park 等人提出了一个高效地产生频繁项集的基于杂凑（Hash）的算法。通过实验可以发现，寻找频繁项集的主要计算是在生成频繁 2—项集 Lk 上，Park 等就是利用这个性质引入杂凑技术来改进产生频繁 2—项集的方法。

3）基于采样的方法。基于前一遍扫描得到的信息，对它详细地做组合分析，可以得到一个改进的算法，其基本思想是：先使用从数据库中抽取出来的采样得到一些在整个数据库中可能成立的规则，然后对数据库的剩余部分验证这个结果。这个算法相当简单并显著地减少了 FO 代价，但是一个很大的缺点就是产生的结果不精确，即存在所谓的数据扭曲（Dataskew）。分布在同一页面上的数据时常是高度相关的，不能表示整个数据库中模式的分布，由此而导致的是采样 5%的交易数据所花费的代价同扫描一遍数据库相近。

4）减少交易个数。减少用于未来扫描事务集的大小，基本原理就是当一个事务不包含长度为志的大项集时，则必然不包含长度为走 k+1 的大项集。从而可以将这些事务删除，在下一遍扫描中就可以减少要进行扫描的事务集的个数。这就是 AprioriTid 的基本思想。

FP-growth 算法

由于 Apriori 方法的固有缺陷 .即使进行了优化，其效率也仍然不能令人满意。2000 年，Han Jiawei 等人提出了基于频繁模式树（Frequent Pattern Tree，简称为 FP-tree）的发现频繁模式的算法 FP-growth。在 FP-growth 算法中，通过两次扫描事务数据库，把每个事务所包含的频繁项目按其支持度降序压缩存储到 FP—tree 中。在以后发现频繁模式的过程中，不需要再扫描事务数据库，而仅在 FP-Tree 中进行查找即可，并通过递归调用 FP-growth 的方法来直接产生频繁模式，因此在整个发现过程中也不需产生候选模式。该算法克服了 Apriori 算法中存在的问颢．在执行效率上也明显好于 Apriori 算法

算法描述如下：

构造 FP-Tree

挖掘频繁模式前首先要构造 FP-Tree，算法伪码如下：

输入:一个交易数据库 DB 和一个最小支持度 threshold.

输出:它的 FP-tree.

步骤:

1.扫描数据库 DB 一遍.得到频繁项的集合 F 和每个频繁项的支持度.把 F 按支持度递降排序,结果记为 L.

2.创建 FP-tree 的根节点,记为 T,并且标记为'null'.然后对 DB 中的每个事务 Trans 做如下的步骤.

根据 L 中的顺序,选出并排序 Trans 中的事务项.把 Trans 中排好序的事务项列表记为[p|P],其中 p 是第一个元素,P 是列表的剩余部分.调用 insert_tree([p|P],T).

函数 insert_tree([p|P],T)的运行如下.

如果 T 有一个子结点 N,其中 N.item-name=p.item-name,则将 N 的 count 域值增加 1;否则,创建一个新节点 N,使它的 count 为 1,使它的父节点为 T,并且使它的 node_link 和那些具有相同 item_name 域串起来.如果 P 非空,则递归调用 insert_tree(P,N).

FP-Tree 创建的算法流程图

注：构造 FP-Tree 的算法理解上相对简单，所以不过多描述

挖掘频繁模式

对 FP-Tree 进行挖掘，算法如下：

输入:一棵用算法一建立的树 Tree

输出:所有的频繁集

步骤:

调用 FP-growth(Tree,null).

procedure FP-Growth (Tree, x)

```
{
(1)if (Tree 只包含单路径 P) then
(2) 对路径 P 中节点的每个组合（记为 B）
(3) 生成模式 B 并 x，支持数=B 中所有节点的最小支持度
(4) else 对 Tree 头上的每个 ai，do
{
(5) 生成模式 B= ai 并 x，支持度=ai.support；
(6) 构造 B 的条件模式库和 B 的条件 FP 树 TreeB；
(7)if TreeB != 空集
(8)then call FP-Growth ( TreeB , B )
}
}
```