基于 Aprior & FP-growth 算法的研究①

晏 杰[1], 亓文娟[2]

[1](武夷学院 团委, 武夷山 354300)
[2](武夷学院 数学与计算机系, 武夷山, 354300)

摘 要: 关联规则挖掘在数据挖掘中占有极其重要的地位, Apriori 算法和 FP-growth 算法是当前关联规则中两大主要频繁项集发现算法. 研究了这两种算法的基本思想, 指出了算法各自的优缺点并通过具体的实例说明发现频繁项集的方法, 最后通过实验对算法进行了性能上的比较.

关键词: 关联规则; Apriori 算法; FP-growth 算法

Research Based on Aprior & FP-growth Algorithm

YAN Jie[1], QI Wen-Juan[2]

[1](Youth League Committee, Wuyi University, Wuyi Shan 354300, China)
[2](Mathematics and Computer Science department, Wuyi University, Wuyi Shan 354300, China)

Abstract: Mining of association rules in data mining plays an important role, Apriori algorithm and FP-growth algorithm are the two major association rules frequent itemsets discovery algorithm.study of the two kinds of the basic idea of the algorithm, points out the advantages and disadvantages of the algorithm through specific examples of frequent itemsets found method, finally through the experiment to the algorithm for the performance comparison.

Key words: association rules; Apriori algorithm;FP-growth algorithm

数据挖掘(Data Mining)就是从数据库中发现知识(KDD)、数据分析、数据融合(Data Fusion)以及决策支持等. 关联规则挖掘(Association Rule Mining)是数据挖掘领域成果颇丰而且比较活跃的研究分支, 是用于发现隐藏在大型数据集中令人感兴趣的联系. 本文针对频繁项集发现算法 Apriori 算法和 FP-growth 算法进行了研究, 旨在对关联规则挖掘算法的扩展和改进奠定基础.

1 关联规则概念和理论

● 设 I=[i1,i2,i3,… in]项的集合. 设任务相关的数据集 D 是事务数据库的集合, 其中每个事务 T 是项目的集合, 使得 T⊆I. 每一个事务有一个表示符, 称作 TID. 事务 T 包含一个项目集 A 当且仅当 A⊆T, 一个关联规则就是形如 A⇒B 的逻辑蕴涵式, 其中 A⊂I, B⊂I 并且 A∩B=∅ [1].

● 支持度: 即 A 和 B 这两个项集的并集 A∪B 在所有事务 D 中出现的概率. Support(A⇒B)=P(A∪B)=Support(A∪B)=S.

● 置信度: 即在出现了项集 A 的事务 D 中, 项集 B 也同时出现的概率. Confidence(A⇒B)=P(B|A)=Support(A∪B)/Support(A)=C.

关联规则的挖掘算法分为两个步骤:

● 产生频繁项集: 发现满足最小支持度阈值的所有项集, 即频繁项集.

● 产生规则: 根据上一步发现的频繁项集中提取大于置信度阈值的规则, 即强规则.

2 Apriori算法

2.1 Apriori 算法基本思想

Apriori 算法是布尔关联规则挖掘频繁项集的原创性算法. 该算法利用逐层搜索的迭代方法找出数据库

① 收稿时间:2012-10-24;收到修改稿时间:2012-11-28

中项集的关系, 以形成规则, 其过程由连接(类矩阵运算)与剪枝(去掉那些没必要的中间结果)组成. Apriori 算法寻找最大项目集的基本思想是: 算法需要进行多步处理数据集.第一步,简单统计所有含一个元素项目集出现的频率,并找出不小于最小支持度的项目集, 即一维最大项目集. 从第二步开始循环处理直到再没有最大项目集生成. 循环周期是: 第 k 步中, 根据第 k-1 步生成的(k-1)维最大项目集产生 k 维侯选项目集,然后对数据库进行搜索, 得到侯选项目集的项集支持度, 与最小支持度比较, 直到没有候选项集为止, 最终找到 k 维最大项目集.

2.2 Apriori 算法应用实例

已知事务数据库 D, 如表 1 所示. 该数据库中有 10 个事务, 设最小支持度是 3, 即 min_sup=30%

表 1 事务数据库 D

TID	Items
T1	e b c
T2	d c
T3	b a c
T4	b d
T5	d f c b
T6	e a c g
T7	d g c
T8	a e b
T9	b c d
T10	e c b d

频繁项集发现过程如下:

1) 扫描数据集中的所有事务, 对每个项的出现次数计数;

2) 最小事务支持度计数为 3, 确定频繁 1-项集的集合 L_1, 由大于或等于支持度计数的 1-项集组成, L_1={{a},{b},{c},{d},{e}};

3) 为发现频繁 2-项集 L_2, 算法连接 L_1 产生候选 2-项集的集合 C_2={{a,b},{a,c},{a,d},{a,e},{b,c},{b,d},{b,e},{c,d},{c,e},{d,e}};

4) 扫描 D 中事务, 计算 C_2 中每个候选项集的支持计数. 如果某个事务包含该候选项集, 则该候选项集的支持计数加 1;

5) 确定频繁 2-项集的集合 L_2, 它由 C_2 中大于或等于支持度计数的 2-项集组成, L_2={{b,c},{b,d},{b,e},{c,d},{c,e}};

6) 候选 3-项集的集合 C_3 由频繁 2-项集产生. C_3={{b,c,d},{b,c,e}};

7) 扫描 D 中事务, 计算 C_3 中每个候选项集的支持计数. 统计出 C_3 中候选项集{b,c,d}的支持计数为 3, 所以{b,c,d}为频繁 3-项集 L_3.

频繁项集发现的详细过程如图 1 所示.

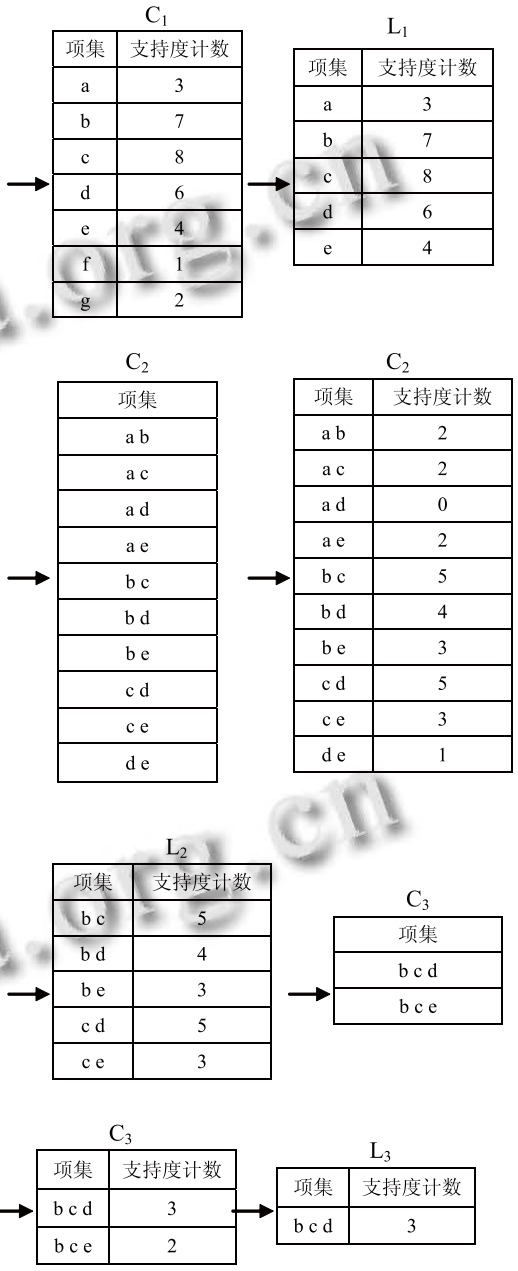

图 1 频繁项集的发现过程

2.3 Apriori 算法优缺点

Apriori 算法使用 Apriori 性质来生成候选项集的方法, 在数据量较小的情况下大大压缩了候选频繁项集的大小, 取得了很好的性能. 但当频繁项集数据量

很大的时候, 它有两个方面的开销可能是巨大的: (1)产生大量的候选项集; (2)重复扫描事务数据库, 数据要在外存与内存之间转换处理, 开销很大, 同时运行效率也较低, 在频繁项长度变大的情况下, 运算时间显著增加.

为了提高 Apriori 算法的效率, 国内外的许多文献提出了 Apriori 算法的优化, 如划分、抽样、事务压缩、散列技术、动态项集计数等[2]. 本文提出了一种不产生候选项集算法—FP-growth 算法, 可以有效地解决上述问题.

3　FP-growth算法

3.1 FP-growth 算法基本思想

FP-growth 算法不同与 Apriori 算法生成候选项集再检验是否频繁的“产生-测试”方法, 而是使用一种称为频繁模式树(FP-tree)的紧凑数据结构组织数据, 并直接从该结构中提取频繁项集. FP-tree 是一种输入数据的压缩表示, 通过逐个读入事务, 并把每个事务映射到 FP-tree 中的一条路径来构造. 由于不同的事务可能会有若干个相同的项, 因此它们的路径可能部分重叠. 路径相互重叠越多, 使用 FP-tree 结构获得的压缩效果越好. 如果 FP-tree 足够小, 能够存放在内存中, 就可以直接从这个内存的结构提取频繁项集, 而不必重复地扫描放在硬盘上的数据, 从而提高处理的效率.

FP-growth 算法[3]包含两个步骤:一是构造频繁模式树 FP-Tree, 二是调用 FP-growth 算法进行频繁项集挖掘.

3.2 FP-growth 算法应用实例

该实例采用图 1 中的事务数据库 D, 设最小支持度是 3, 即 min_sup=30%

FP-growth 算法执行过程如下:

1) 首先扫描一次数据库D, 找出频繁项的列表L1, 按照它们的支持度计数递减排序, 即 L={{c},{b},{d},{e},{a}}.

2) 再次扫描数据库, 利用每个事务中出现的频繁项构造 FP-tree. 创建树的根节点 null. 处理每个事务时按照 L 中的顺序将事务中出现的频繁项添加到 FP-tree 中的一个分支.

① 扫描第一个事务“e b c”, 按照 L 中的顺序将事务排序为“c b e”,构造 FP-tree 的第一个分支<(c:1), (b:1), (e:1)>;

② 第二个事务“d c”排序后为“c d”,与第一个分支共享节点<c>,因此将树中节点<c>的计数加 1, 同时在<c>节点上创建分支<d:1>;

③ 第三个事务“b a c”排序后为“c b a”,与分支<c b e>共享节点<c b>,一次将树中节点<c>和<b>计数加 1, 同时在<b>节点上创建分支<a:1>;

④ 以此类推, 将数据库中的事务都添加到 FP-tree 中. 为便于遍历树, 创建一个头结点表, 使得每个项通过一个结点链指向它在树中的出现, 相同的项目链接在一个链表中. 构造好的 FP-tree 如图 2 所示. 这样数据库频繁模式的挖掘问题就转换成挖掘 FP-tree 问题.

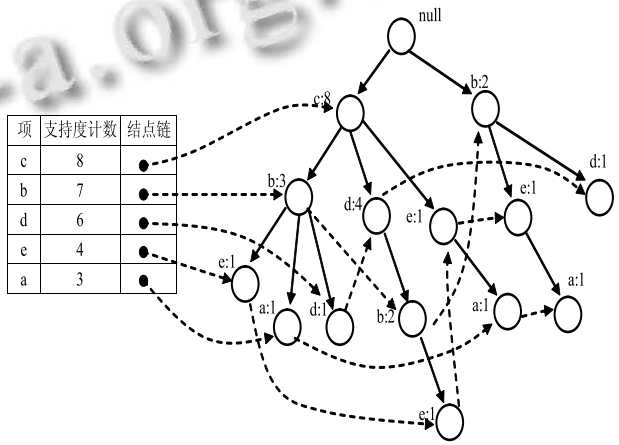

图 2　FP-tree 的构造

3) 挖掘 FP-tree. 挖掘 FP-tree 采用自底向上的迭代方式[4], 首先查找以“a”为后缀的频繁项集, 然后依次是“e”, “d”, “b”, “c”. 查找以“a”为后缀的频繁项集, 利用头节点表和树节点的链接, 找出包含“a”的三个分支构造条件模式基. 条件模式基可以看作是一个“子数据集”, 由 FP-tree 中与后缀模式一起出现的前缀路径组成. “a”的条件模式基为{(c b:1),(c e:1),(b e:1)},利用条件模式基构造条件 FP-树, 因为所有路径的支持度都小于最小支持度 3, 所以 a 的条件模式树不含任何路径. 同理“e”的条件模式基为{(c b:1),(c:1),(b:1),(c d b:1)},构造条件 FP-树如图 3 所示.

根据条件 FP-树, 挖掘出频繁模式, 如表 2 所示.

3.3 FP-growth 算法优缺点

FP-growth 算法将事务数据库 D 有效地压缩成小存储空间的数据结构, 克服了 Apriori 算法中多次扫描事务数据库的缺陷, 只需对事务数据库进行二次扫描, 将发现长频繁模式的问题转化递归模式增长的策略, 避免产生的大量候选集, 大大降低了算法的时间复杂

度. 但也存在两个方面的缺陷: (1)算法在挖掘大型数据集时如果由原数据库得到的 FP-tree 的分支很多, 而且分支长度很长时, 该算法将需要构造出数量巨大的条件 FP-tree, 不仅费时而且要占用大量的空间. (2)算法在挖掘频繁模式过程中存在性能瓶颈: 由于该算法要递归生成条件数据库和条件 FP-tree, 而条件 FP-tree 需要自顶向下生成, 频繁模式的挖掘需要自底向上处理, 在挖掘时需要反复地搜索 FP-tree, 这就需要更多的指针, 所以内存开销大.

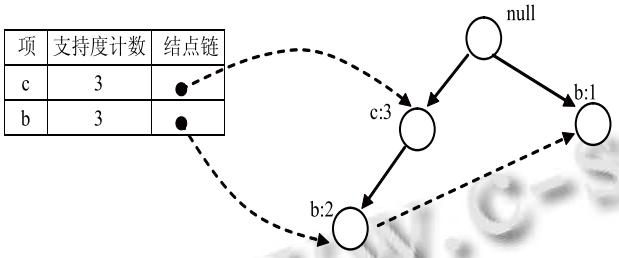

图 3 节点 e 的条件 FP-树

表 2 条件 FP-树产生的频繁模式

项	条件模式基	条件 FP-树	产生的频繁模式
a	{(cb:1),(ce:1), (be:1)}	空	空
e	{(cb:1),(c:1),(b:1), (cdb:1)}	<c:3,b:2> <b:1>	ce:3, be:3
d	{(cb:1),(c:4),(b:1)}	<c:5>	cd:5
b	{(c:3),(cd:2) }	<c:5>	cb:5

4 算法性能分析

为了验证算法的效率, 采用 VC++6.0 分别实现 Apriori 算法和 FP-growth 算法算法. 在 CPU 为 2GHZ, 内存为 1G, 操作系统为 Windows XP 的计算机上测试. 测试数据包含事务数为 8536, 每个事务包含 132 个属性. 分别对支持度为 0.1, 0.2,0.3,0.4, 0.5 时进行测试. 测试结果如图 4 所示.

从图 4 可以看出, 在支持度较大的情况下, Apriori 算法和 FP-growth 算法执行时间差别不大, 但当支持度较小的情况下, FP-growth 算法的性能明显优于 Apriori 算法, 这是由于 Apriori 算法在支持度较小的情况下, 产生的候选项集相应的增加, 影响了算法的效率, 而 FP-growth 算法不产生候选项集, 相比之下挖掘的效率更高.

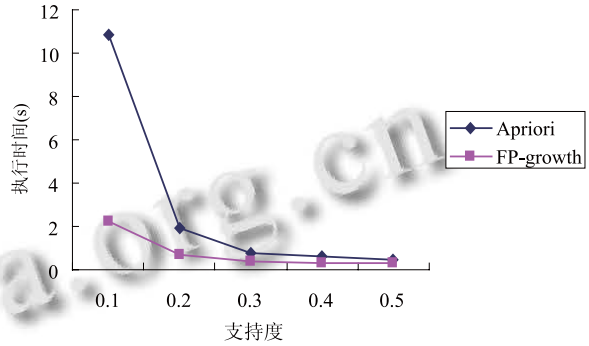

图 4 算法在不同支持度下执行时间比较

5 结语

本文在深入研究关联规则经典算法 Apriori 算法和改进算法 FP-growth 算法思想的基础上, 提出了算法各自的优缺点, 针对每种算法给出了具体的实例来分析, 并通过实验进行了性能上的比较, 结果表明在支持度较小的情况下, FP-growth 算法较 Apriori 算法有着明显的优势.

参考文献

1 Han JW,Kamber Mi.范明,孟小峰译.数据挖掘概念与技术.北京:机械工业出版社,2006.

2 周怡,王世伟.医学数据挖掘－SQL Server 2005 案例分析.北京:中国铁道出版社,2008.

3 冯志新,钟诚.基于 FP-Tree 的最大频繁模式挖掘算法.计算机工程,2004,6(11):123−126.

4 蒋盛益,李霞,郑琪.数据挖掘原理与实践.北京:电子工业出版社,2011.