4种序列模式挖掘算法的比较分析

[原创](javascript:;)[fpcheng](http://blog.51cto.com/fpcheng)[2012-04-08 21:35:02](javascript:;)[评论(0)](http://blog.51cto.com/fpcheng/829527#comment)[3457人阅读](javascript:;)

1.     算法简介

AprioriAll算法属于Apriori类算法，其基本思想为首先遍历序列数据库生成候选序列并利用Apriori性质进行剪枝得到频繁序列。每次遍历都是通过连接上次得到的频繁序列生成新的长度加1的候选序列，然后扫描每个候选序列验证其是否为频繁序列。

GSP（generalized sequential pattern）算法是AprioriAll算法的扩展算法，其算法的执行过程和AprioriAll类似，最大的不同在于GSP引入了时间约束、滑动时间窗和分类层次技术，增加了扫描的约束条件，有效地减少了需要扫描的候选序列的数量。此外GSP利用哈希树来存储候选序列，减少了需要扫描的序列数量。

FreeSpan算法是基于模式投影的序列挖掘算法，其基本思想：利用当前挖掘的频繁序列集将序列数据库递归地投影到一组更小的投影数据库上，分别在每个投影数据库上增长子序列。这一过程对数据和待检验的频繁模式集都进行了分割，并且每一次检验限制在与其相符合的更小投影数据库中。

PrefixSpan是FreeSpan的改进算法，即通过前缀投影挖掘序列模式。其基本思想：投影时不考虑所有可能出现的频繁子序列，只检查前缀序列，然后把相应的后缀投影成投影数据库。每个投影数据库中，只检查局部频繁模式，在整个过程中不需要生成候选序列。

2.     算法的定性比较

表1.算法的分类比较

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性 | Apriori类算法 | | 模式增长算法 | |
| AprioriAll | GSP | FreeSpan | PrefixSpan |
| 候选序列 | 产生 | 产生 | 不产生 | 不产生 |
| 数据结构 | Hash树 | Hash树 | Hash树 | WAP树 |
| 数据库分割 | 否 | 否 | 是 | 是 |
| 数据库的扫描次数 | 反复多次 | 反复多次 | 3次 | 2次 |
| 算法执行 | 循环 | 循环 | 递归 | 递归 |

3.     算法的时空执行效率比较

|  |  |
| --- | --- |
| Apriori  All算法 | 这两种算法都属于Apriori类算法，都要产生大量的候选序列，需要有足够的存贮空间。同时还需要反复扫描数据库，需要占用很多运行时间。该类算法的执行效率比较低，特别是在支持度比较低的情况下，其执行效率将会大大下降。和AprioriAll相比，GSP的执行效率比较高，总体来说要比AprioriAll高2~20倍。 |
| GSP算法 |
| FreeSpan算法 | 这两种算法都属于模式增长算法，它们的查找更加集中和有效。由于该类算法不生成大量的候选序列以及不需要反复扫描原数据库，和Apriori类算法相比该类算法要快且更有效，特别是在支持度比较低的情况下更明显。此外，在时空的执行效率上，PrefixSpan比FreeSpan更优。 |
| PrefixSpan算法 |

4.     算法的适用范围分析

通常数据集可分为稠密数据集和稀疏数据集。稠密数据集有大量的长尺度和高支持度的频繁模式，在这样的数据集中，许多事件是相似的，例如DNA分析或者股票序列分析。稀疏数据集主要由短模式组成，长模式也存在，但相应的支持度很小，例如超级市场的交易数据集，用户在网站中的浏览页面序列等。

Apriori类算法在稀疏数据集的应用中比较合适，不适合稠密数据集的应用。对于有约束条件（例如相邻事务的时间间隔约束）序列模式挖掘，GSP更适用。

FreeSpan和PrefixSpan在两种数据集中都适用，而且在稠密数据集中它们的优势更加明显。两者相比，PrefixSpan的性能更好一些。

在实际应用中，在挖掘过程的不同阶段，数据集的特点，数据规模等因素可能不同，如果根据各阶段的特点，选择与之相应的算法，则序列模式挖掘能达到更好的效果。

此外由于Apriori类算法使用较简单，FreeSpan和PrefixSpan虽然效率高，但实现起来难度大。所以，现在大多数应用都是采用Apriori类算法的改进算法，以克服Apriori类算法执行效率不高的缺点。

5.     参考文献

陈卓,杨炳儒,宋威,宋泽锋.序列模式挖掘综述[J].计算机应用研究.2008,25(7).

吕锋,张炜玮.4种序列模式挖掘算法的特性研究[J].武汉理工大学学报.2006,28(2).

GSP：

[**GSP序列模式分析算法**](http://blog.csdn.net/Androidlushangderen/article/details/43699083)

标签： [数据挖掘](http://so.csdn.net/so/search/s.do?q=%E6%95%B0%E6%8D%AE%E6%8C%96%E6%8E%98&t=blog) [机器学习](http://so.csdn.net/so/search/s.do?q=%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0&t=blog) [算法](http://so.csdn.net/so/search/s.do?q=%E7%AE%97%E6%B3%95&t=blog)

2015年02月10日 09:12:016017人阅读 [评论](javascript:void(0);)(0) [收藏](javascript:void(0);) [举报](javascript:void(0);)

https://csdnimg.cn/release/phoenix/images/category_icon.jpg 分类：

算法（45） https://csdnimg.cn/release/phoenix/images/arrow_triangle%20_down.jpg 数据挖掘（32） https://csdnimg.cn/release/phoenix/images/arrow_triangle%20_down.jpg 机器学习（30） https://csdnimg.cn/release/phoenix/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。 http://blog.csdn.net/Androidlushangderen/article/details/43699083

目录[(?)](http://blog.csdn.net/androidlushangderen/article/details/43699083" \o "系统根据文章中H1到H6标签自动生成文章目录)[[+]](http://blog.csdn.net/androidlushangderen/article/details/43699083)

**参考资料：[http://blog.csdn.net/zone\_programming/article/details/42032309](http://blog.csdn.net/zone_programming/article/details/42032309" \t "_blank)**

**更多数据挖掘代码：[https://github.com/linyiqun/DataMiningAlgorithm](https://github.com/linyiqun/DataMiningAlgorithm" \t "_blank)**

**介绍**

GSP算法是序列模式挖掘算法的一种，他是一种类Apriori的一种，整个过程与Apriori算法比较类似，不过在细节上会略有不同，在下面的描述中，将会有所描述。GSP在原有的频繁模式定义的概念下，增加了3个的概念。

1、加入时间约束min\_gap，max\_gap，要求原来的连续变为只要满足在规定的min\_gap到max\_gap之间即可。

2、加入time\_windows\_size，只要在windows\_size内的item，都可以被认为是同一ItemSet。

3、加入分类标准。

以上3点新的中的第一条特征将会在后面的算法中着重展现。

**算法原理**

1、根据所输入的序列，找出所有的单项集，即1频繁模式，这里会经过最小支持度阈值的判断。

2、根据1频繁模式进行连接运算，产生2频繁模式，这里会有进行最小阈值的判断。

3、根据2频繁模式连接产生3频繁模式，会经过最小支持度判断和剪枝操作，剪枝操作的原理在于判断他的所有子集是否也全是频繁模式。

4、3频繁模式不断的挖掘知道不能够产生出候选集为止。

**连接操作的原理**

2个序列，全部变为item列表的形式，如果a序列去掉第1个元素后，b序列去掉最后1个序列，2个序列的item完全一致，则代表可以连接，由b的最后一个元素加入到a中，至于是以独立项集的身份加入还是加入到a中最后1个项集中取决于b中的最后一个元素所属项集是否为单项项集。

**时间约束计算**

这个是用在支持度计数使用的，GSP算法的支持度计算不是那么简单，比如序列判断<2, <3, 4>>是否在序列<(1,5), 2 , <3, 4>, 2>，这就不能仅仅判断序列中是否只包含2,<3, 4>就行了，还要满足时间间隔约束，这就要把2，和<3,4>的所有出现时间都找出来，然后再里面找出一条满足时间约束的路径就算包含。时间的定义是从左往右起1.2,3...继续，以1个项集为单位，所有2的时间有2个分别为t=2和t=4，然后同理，因为<3,4>在序列中只有1次，所以时间为t=3，所以问题就变为了下面一个数组的问题

2  4

3

从时间数组的上往下，通过对多个时间的组合，找出1条满足时间约束的方案，这里的方案只有2-3,4-3,然后判断时间间隔，如果存在这样的方式，则代表此序列支持所给定序列，支持度值加1,这个算法在程序的实现中是比较复杂的。

**算法的代码实现**

测试数据输入(格式：事务ID item数 item1 item2.....):

**[java]** [view plain](http://blog.csdn.net/androidlushangderen/article/details/43699083) [copy](http://blog.csdn.net/androidlushangderen/article/details/43699083)

1. 1 2 1 5
2. 1 1 2
3. 1 1 3
4. 1 1 4
5. 2 1 1
6. 2 1 3
7. 2 1 4
8. 2 2 3 5
9. 3 1 1
10. 3 1 2
11. 3 1 3
12. 3 1 4
13. 3 1 5
14. 4 1 1
15. 4 1 3
16. 4 1 5
17. 5 1 4
18. 5 1 5

最后组成的序列为：

<(1,5) 2 3 4>

<1 3 4 (3,5)>

<1 2 3 4 5>

<1 3 5>

<4 5>

也就是说同一序列都是同事务的。

**算法实现的难点**

1、算法花费了几天的时间，难点首先在于对算法原理本身的理解，网上对于此算法的资料特别少，而且不同的人所表达的意思 都有少许的不同，讲的也不是很详细，于是就通过阅读别人的代码理解GSP算法的原理，我的代码实现也是参考了参考资料的C语言的实现。

2、在实现时间约束的支持度计数统计的时候，调试了一段时间，做时间统计容易出错，因为层级实在太多容易搞晕。

3、还有1个是Sequence和ItemSet的拷贝时的引用问题，在产生新的序列时一定要深拷贝1个否则导致同一引用会把原数据给改掉的。

**GSP算法和Apriori算法的比较**

我是都实现过了GSP算法和Apriori算法的，后者是被称为关联规则挖掘算法，偏向于挖掘关联规则的，2个算法在连接的操作上有不一样的地方，还有在数据的构成方式上，Apriori的数据会简单一点，都是单项单项构成的，而且在做支持度统计的时候只需判断存在与否即可。不需要考虑时间约束。Apriori算法给定K项集，连接到K-1项集算法就停止了，而GSP算法是直到不能够产生候选集为止