第 37 卷 第 24 期
Vol.37 No.24

计 算 机 工 程
Computer Engineering

2011 年 12 月
December 2011

• 软件技术与数据库 •　　文章编号：1000—3428(2011)24—0036—02　　文献标识码：A　　中图分类号：TP311

基于多个最小支持度的频繁项目集挖掘算法

陈福集，李福平

(福州大学公共管理学院，福州 350108)

摘　要：传统的关联规则挖掘算法不能在同一事务数据库中连续挖掘多个最小支持度的频繁项目集。为此，提出基于多个最小支持度的频繁项目集挖掘算法。运用集合论定义模型库的概念，将事务数据库转化成模型库，通过检索模型库得到频繁项目集，从而降低频繁项目集的挖掘时间。实验结果表明，该算法的挖掘效率高于 Apriori 算法。

关键词：关联规则；数据挖掘；最小支持度；模型库；频繁项目集

Frequent Itemset Mining Algorithm Based on Multiple Minimum Support Degrees

CHEN Fu-ji, LI Fu-ping

(College of Public Administration, Fuzhou University, Fuzhou 350108, China)

【Abstract】To the demand of a continuous mining frequent itemset in the same transaction database under multiple minimum support degree, this paper proposes frequent itemset mining algorithm based on multiple minimum support degrees. The algorithm uses set theory, leads into model library, converts the transaction database into a model library, and searches model library to obtain frequent itemset. The algorithm reduces the time of frequent itemset mining and improves efficiency of frequent itemset mining. Experimental results show this algorithm is more efficient than Apriori algorithm.

【Key words】association rule; data mining; minimum support degree; model library; frequent itemset

DOI: 10.3969/j.issn.1000-3428.2011.24.012

1　概述

关联规则挖掘是指从事务数据库中挖掘相关属性之间的关联关系。关联规则挖掘是数据挖掘领域的一个重要分支，在现实生活中得到了广泛应用。

文献[1]提出的 Apriori 算法通过对大量候选项目集进行支持度测试得到频繁项目集。该算法虽然易于实现，但需要扫描事务数据库多次。

文献[2]提出基于 FP-tree 结构的 FP-growth 算法，该算法扫描事务数据库 2 次，在 FP-tree 上递归产生条件模式基和条件 FP-tree 来挖掘频繁模式。该算法虽然具有很好的性能，但是存储结构复杂，不适合稠密事务数据库。

文献[3]引入 0-1 矩阵和逻辑运算，提出一种基于矩阵的关联规则挖掘算法。文献[4]引入加权支持度和最小支持期望的概念，提出基于位矩阵的加权频繁 k 项集生成算法。

文献[1-4]分别使用不同的方法实现关联规则挖掘，但当改变最小支持度时，以上算法需要重新扫描事务数据库，进行多次 I/O 操作和运算，浪费了大量时间，降低了挖掘效率。

针对以上问题，本文提出了基于多个最小支持度的频繁项目集挖掘算法(Frequent Itemset Mining algorithm based on multiple Minimum support degrees, FIMM)。

2　FIMM 算法

2.1　定义

设 $I=\{I_1,I_2,\cdots,I_n\}$ 是项目集合，I 的一个非空子集称为一个项目集，含有 k 个项目的项目集称为 k-项目集。事务数据库 $D=\{T_1,T_2,\cdots,T_n\}$，一个事务 T 表示成一个二元组 $T=(TID,X)$，其中，TID 为事务标识符；$X\subseteq I$。设 Y 为一个项目集，当且仅当 $Y\subseteq X$ 时，称事务 T 包含项目集 Y。D 中包含项目集 Y 的事务数称为项目集 Y 在 D 中的支持数，用 $\sigma(Y)$ 表示。Y 的支持数 $\sigma(Y)$ 在 D 中所占的比例称为项目集 Y 在 D 中的支持度，用 $sup(Y)$ 表示，即：

$$sup(Y)=\sigma(Y)/|D|$$

其中，$|D|$ 为 D 中的事务总数。

一个项目集的支持数是该项目集在数据集中的出现次数，而一旦求出项目集的支持数就很容易通过上式计算出其支持度。因此，为了表达方便，使用支持数代表支持度，使用最小支持数代表最小支持度。

定义 1　设 $M=(X,count)$，$X\subseteq I$，$count$ 是 X 在事务数据库 D 中的支持数，则称 M 是 X 的模型。如果 X 是 k-项目集，则称 M 是 k-模型。

定义 2　设 $Model[k]=\{M_1,M_2,\cdots,M_n\}$，$M_i(0<i\leqslant n)$ 是 k-模型，则称 $Model[k]$ 是 k-模型组。

定义 3　设 $Model=\{Model[1],Model[2],\cdots,Model[T_{\max}]\}$，$T_{\max}$ 是事务数据库 D 中事务的最大长度，则 $Model$ 是事务数据库 D 的模型库。

2.2　过程描述

FIMM 算法的具体过程如下：

(1)初始化模型库

对一个含有 i 个项目和 t 个事务的事务数据库 D，设其

基金项目：国家杰出青年科学基金资助项目(70925004)

作者简介：陈福集(1954－)，男，教授、博士、博士生导师，主研方向：数据挖掘，决策支持系统；李福平，硕士研究生

收稿日期：2011-04-12　**E-mail：**79890105@qq.com

事务的最大长度为 $T_{\max}$，初始化模型库：$Model=\{Model[1], Model[2],\cdots, Model[T_{\max}]\}$。

(2)将事务数据库转化为模型库

扫描事务数据库 D，把每条事务的非空子集 T_i^* 的项分别转化为模型 M_i，模型 M_i 的 $count$ =1。

(3)将模型 M_i 加入到模型库 $Model$ 中

设 M_i 是 k-模型，比较 M_i 和 k-模型组 $Model[k]$ 的模型，如果模型组 $Model[k]$ 中已经存在模型 M_i，则 $Model[k]$ 中的模型 M_i 的 $count$ 增加 1，否则，将 M_i 加入模型组 $Model[k]$。重复步骤(2)和步骤(3)，完成模型库 $Model$ 的建立。

(4)检索模型库，输出符合最小支持数的模型

设最小支持数为 $Minsup$，则比较 $Minsup$ 和 k-模型组 $Model[k](0<k\leqslant T_{\max})$ 内模型的次数 $count$ 的大小。如果 $count<Minsup\prec Minsup$，则此模型不是 k-频繁项目，否则，此模型是频繁 k-项目集。

(5)当用户输入多个最小支持数时，对每个最小支持数分别执行步骤(4)。

2.3　算法描述

FIMM 算法的伪代码描述如下：

(1)主函数

```
输入 事务数据库 D，最小支持数集合 Minsups
输出 D 中的所有频繁项目集
Main()
{Creat_Model();//生成事务数据库 D 的模型库
 For each Minsup∈Minsups//遍历最小支持数数组
      Search(Minsup);//在模型库中检索大于或等于最小支持
//数的频繁项集}
```

(2)建立模型库

```
输入 事务数据库 D
输出 事务数据库 D 的模型库 Model
Creat_Model()
{For each Ti∈D//扫描事务数据库
Ti* =Create(Ti);//生成非空子集
For each  t ∈ Ti* //遍历非空子集
M m=new M(t, 1);
Add_Model(m, ti length);//将模型加入模型库}
```

(3)加入模型库

```
输入 模型 m 和模型内集合的长度 n
输出 更新之后的模型库
Add_Model(m, n)
{For each km in Model[n]
If(!Equal(kmX, mX)//查询是否有相同的集合
Add m to Model[n];//把新模式加入模式库
Else
Km.cout;//把模型的支持数加 1}
```

(4)FIMM 算法的检索函数

```
输入 最小支持数 Minsup
输出 满足最小支持数的频繁项集
Search(Minsup)
 {for(k=1; k≤max(Ti length);k++)//遍历从 1-项集到
//(Ti length)-项集
      For each min Model[k]//遍历模型组中的模型
          If(m.cout ≥ Minsup)//比较项集出现的次数与
//Minsup 的大小
              Output  m ;//输出频繁项集}
```

2.4　算法分析

Apriori 算法在运行过程中，需要产生大量的候选项目集，占用大量的内存空间，导致内存频繁的换进换出操作。FIMM 算法的空间复杂度取决于最长事务的非空子集的个数。事务的非空子集的个数至多为 2^L-1（L 为事务的最大长度），所以，FIMM 算法的空间复杂度为 $O(2^L-1)$。

FIMM 算法把事务数据库转化为模型库后，在连续挖掘多个最小支持度的频繁项目集时，不需要产生候选项目集或对候选项目集进行支持度测试，只需要检索模型库即可获得频繁项目集。FIMM 算法在频繁项目集挖掘过程中具有较好的时间特性。

3　案例分析

All Electronics 某分店的事务数据库 D [5]如表 1 所示，其中有 9 条事务，$D=\{T100, T200, T300, T400, T500, T600, T700, T800, T900\}$，对应的项目集 $I=\{I1, I2, I3, I4, I5\}$，事务的最大长度 $T_{\max}$ =4。设最小支持数 $Minsups$ ={2, 4}。

表 1　All Electronics 某分店的事务数据库

事务编号	商品 ID
$T100$	$I1, I2, I5$
$T200$	$I2, I4$
$T300$	$I2, I3$
$T400$	$I1, I2, I4$
$T500$	$I1, I3$
$T600$	$I2, I3$
$T700$	$I1, I3$
$T800$	$I1, I2, I3, I5$
$T900$	$I1, I2, I3$

(1)初始化模型库

模型库 $Model=\{Model[1], Model[2], Model[3], Model[4]\}$。

(2)扫描数据集

逐条扫描事务数据库 D，建立模型库。读取第 1 条事务 $T100$，其非空子集 $T100^*=\{\{I1\}, \{I2\}, \{I5\}, \{I1, I2\}, \{I1, I5\}, \{I2, I5\}, \{I1, I2, I5\}\}$，分别生成 $T100^*$ 内元素的模型，将其加入模型库 $Model$，如表 2 所示。

表 2　加入 $T100$ 后的模型库

模型组	模型
$M[1]$	$(\{I1\}, 1), (\{I2\}, 1), (\{I5\}, 1)$
$M[2]$	$(\{I1, I2\}, 1), (\{I1, I5\}, 1), (\{I2, I5\}, 1)$
$M[3]$	$(\{I1, I2, I5\}, 1)$

然后读取下一条事务 $T200, T300, \cdots, T900$，按 $T100$ 的方法将事务加入模型库中。扫描完事务数据库 D 建立的模型库如表 3 所示。

表 3　加入 $T100$~$T900$ 后的模型库

模型组	模型
$M[1]$	$(\{I1\}, 6), (\{I2\}, 7), (\{I3\}, 6), (\{I4\}, 2), (\{I5\}, 2)$
$M[2]$	$(\{I1, I2\}, 4), (\{I1, I5\}, 2), (\{I2, I5\}, 2), (\{I2, I4\}, 2), (\{I2, I3\}, 4), (\{I1, I4\}, 1), (\{I1, I3\}, 4), (\{I3, I5\}, 1)$
$M[3]$	$(\{I1, I2, I5\}, 2), (\{I1, I2, I4\}, 1), (\{I1, I2, I3\}, 2), (\{I1, I3, I5\}, 1), (\{I2, I3, I5\}, 1)$
$M[4]$	$(\{I1, I2, I3, I5\}, 1)$

(3)检索模型库

检索模型库中满足最小支持数的模型，符合条件的模型即是频繁项目集。

当 $Minsup$ =2 时，检索模型库 $Model$，得到 1-项目集{$I1$, $I2$, $I3$, $I4$, $I5$}、2-项目集{{$I1$, $I2$},{$I1$, $I5$},{$I2$, $I5$},{$I2$, $I4$},{$I2$, I3},{$I1$, $I3$}}、3-项目集{{$I1$, $I2$, $I5$}, {$I1$, $I2$, $I3$}}。

当 $Minsup$ =4 时，重新检索模型库 $Model$，即可得到 1-项目集{$I1$, $I2$, $I3$}和 2-项目集{{$I1$, $I2$}, {$I2$, $I3$}, {$I1$, $I3$}}。

(下转第 41 页)