高维稀疏数据频繁项集挖掘算法的研究

闫 珍[1] 皮德常[1] 吴文昊[2]
(南京航空航天大学信息科学与技术学院 南京 210016)[1]
(复旦大学计算机科学与技术学院 上海 200433)[2]

摘 要 传统挖掘算法不适用于挖掘高维稀疏数据集。提出了一种针对高维稀疏数据的频繁项集挖掘算法 FIHS。FIHS 引入了一种新的数据结构用来存储频繁项集,该结构不但可以减少存储空间,而且可以降低计数代价。该算法只需扫描一次数据集,通过优化连接剪枝操作避免产生非频繁的候选项集,基于 K-频繁项集使用"与"、"或"操作产生 $K+1$-频繁项集,且数据结构易于维护。理论分析和实验表明,该算法用于高维稀疏数据集上具有挖掘速度快,存储空间少等优点。

关键词 高维数据,稀疏数据,频繁项集,存储结构

中图法分类号 TP312 **文献标识码** A

Research on Frequent Itemsets Mining Algorithm Based on High-dimensional Sparse Dataset

YAN Zhen[1] PI De-chang[1] WU Wen-hao[2]
(College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)[1]
(School of Computer Science, Fudan University, Shanghai 200433, China)[2]

Abstract The traditional mining algorithms are not applicable to mine high-dimensional sparse dataset, a new frequent itemsets mining algorithm based on high-dimensional sparse dataset named FIHS (Frequent Itemsets mining algorithm based on High-dimensional Sparse dataset) was proposed in this paper. FIHS adopts a new data structure to store frequent itemsets, using this structure can reduce the storage space and the cost of counting. FIHS can avoid generating infrequent candidate itemsets through optimizing the operation of connection and pruning, which requires scan the dataset once. What's more, just by applying AND-OR operation, frequent $K+1$-itemsets can be created according to frequent K-itemsets, and the maintenance of the data structure is simple. According to theoretical analysis and experiments, the improved algorithm enjoys many advantages aiming at high-dimensional sparse dataset, such as quick mining, less memory space, etc.

Keywords High-dimensional data, Sparse data, Frequent itemsets, Data structure

1 引言

2000年,Hipp提出关于当前频繁项集挖掘算法的分类方法[5]。根据算法对搜索空间计数的方式不同,分为广度优先搜索方式(BFS)和深度优先搜索方式(DFS)。根据频繁项集挖掘算法计数的方式不同,分为直接计数(counting)和交叉计数(intersecting)。代表性算法是 Apriori[1], FP-Growth[2] 和 dEclat[3] 算法,其中 Apriori 是 BFS, counting, 候选集-计数模型类算法, FP-Growth 是 DFS, counting, 模式增长类算法, dEclat 是 DFS, intersecting, 候选集-计数模型类算法。

广度优先搜索算法发展最早,研究相对较成熟。这种迭代循环、逐渐查找频繁项集的思路比较符合人的直观思维。深度优先搜索则是后来发展的,通过引入相对复杂的概念和结构来提高算法的效率。下面分析了几个代表性算法的优缺点。

Apriori 算法:需要多次扫描数据库,产生庞大的候选集,计算候选集的成本相当高。相对来讲,Apriori 算法在稀疏数据集和短模式下性能表现得更好。

FP-Growth 算法:只需扫描两次数据库,同时采用了模式增长方式,以避免产生大量的候选集。然而现实数据集多为稀疏的,这种模式压缩的方法仍显得不够紧凑。FP-Growth 算法在数据集密集时,FP-Tree 的密度是相当高的,构造 FP-Tree 的有效性抵销了其维护成本,性能较好。

dEclat 算法:项的事务列表采用与事务记录号对应的 0 和 1 序列表示,交叉计数本身相当高效。但当事务列表较长时,交叉计算的"逻辑与"计算时间就相当长。

根据研究,得出一些频繁项集挖掘算法的总结论断:

1)DFS 所计算的候选集数目比 BFS 计算的数目多;

2)运用于稀疏数据集,模式增长型算法占用内存很大;

3)交叉计算的效率较高。

从 2005 年至今,国内外关于高维稀疏数据频繁项集挖掘算法的研究如文献[6-12]。算法的搜索方式和计数方式是算

到稿日期:2010-07-06 返修日期:2010-09-29 本文受国防技术基础研究和国家高技术研究发展计划(863 计划)项目(2007AA01Z404)资助。
闫 珍(1987-),女,硕士生,主要研究方向为零售业数据挖掘,E-mail:ahyanzhen@yahoo.com.cn;皮德常(1971-),男,教授,博士生导师,主要研究方向为数据挖掘和数据库技术;吴文昊(1985-),男,硕士生,主要研究方向为数据挖掘、图形图像处理。

法的核心部分，算法使用的数据结构和检索技术是为算法的核心部分服务的。频繁项集挖掘算法有多种优化方式，这些优化无非是针对以下目标中的一个或者多个进行的：扫描数据集次数、产生候选集的数量、计算每个候选集支持度的时间、维护数据结构的成本。针对零售业数据集高维稀疏的特点，结合目前国内外研究现状的分析，设计了一种广度优先搜索、交叉计数、候选集-计数模型类的算法。算法定义了一种新的数据结构，用来存储频繁项集，采用垂直数据表示方法和交叉计数方式，针对以上的4个优化目标进行改进。

2 零售业数据的简单描述

目前，大部分零售行业已经部署了POS系统。从零售商的角度出发，如果可以从大量的销售事务记录中发现有趣的规则，就可以帮助它们制定决策，如促销分析、交叉购物等；如果可以对顾客的购买行为进行相关分析，商家就可以充分利用顾客的购买习惯，从而增加商品的销售量，借以提高商品的利润。

然而，零售业数据具有以下几方面的特点，使得其不再适合使用传统的关联规则挖掘方法进行处理。

1)流数据。每天每时都会产生大量的交易记录，数据库会动态地更新。

2)数据维数高。通常一条POS记录会有数十甚至上百个数据项。

3)数据量大。像Wal. Mart每天都会产生2亿条左右的交易数据。

4)数据稀疏。大型超市商品种类会达到上万种，但一般的购买项最多也是上百个。

因此，在做分析时应当注意两点：

1)交易数据集的特点是高维、稀疏、庞大。进行数据挖掘时必须考虑到交易数据的固有特点。

2)交易数据库是动态更新的，新的数据积累可能会导致以前发现的规则失效，这些规则需要动态维护和及时更新。针对这样的情况，须基于先前的挖掘结果维护更新的数据而不是重新挖掘所有的数据。

本文只是针对静态数据集的特点做研究，暂不考虑时态的问题。

3 基本定义与有关性质

定义1(事务集-二进制矩阵关系)　给定$I=\{i_1,i_2,\cdots,i_m\}$为项集，事务数据库$D=\{T_1,T_2,\cdots,T_n\}$，事务$T_i\in D$且$T_i\subseteq I, i=1,2,\cdots,n$。我们称$f:T_i\times I\to b_1b_2\cdots b_{m-1}b_m$为$T_i$-二进制数关系，并称$b_{Ti}=b_1b_2\cdots b_{m-1}b_m$为事务$T_i$所对应的二进制数，记为$B_{Ti}$，其中$b_j\in\{0,1\}$。如果$i_j\in T_i$，则$b_j=1$，否则$b_j=0, j=1,2,\cdots,m$。根据事务-二进制数关系，将每个事务都转换为二进制数，构成事务集-二进制矩阵$B_{n\times m}$。

例1　给定事务数据库$D=\{T_1,T_2,T_3,T_4\}$，项集$I=\{1,2,3,4,5\}$，则事务-二进制数关系如表1所列。

表1　事务-二进制数关系

事务 T_i	二进制数
$T_1=\{1,3,4\}$	$b_{T1}=10110$
$T_2=\{2,3,5\}$	$b_{T2}=01101$
$T_3=\{1,2\}$	$b_{T3}=11000$
$T_4=\{1,2,5\}$	$b_{T4}=11001$

从而，得到事务集-二进制矩阵为$\begin{bmatrix}10110\\01101\\11000\\11001\end{bmatrix}$。

定义2(二进制向量-1项集关系)　给定$I=\{i_1,i_2,\cdots,i_m\}$为项集，事务数据库根据定义1得到的二进制矩阵$B_{n\times m}$。令$B=[B_1\ B_2\cdots\ B_m]$，其中$B_j=b_1b_2\cdots b_{n-1}b_n$为$B_{n\times m}$的第$j$列元素构成的向量。我们称$f:i_j\to B_j$为$i_j$对应的二进制向量-1项集关系。若项目$i_j$在数据库中第$m$条事务中出现，则$b_m=1$，否则$b_m=0, m=1,2,\cdots n$。

例2　给定二进制矩阵$B_{n\times m}$，$I=\{1,2,3,4,5\}$。则I对应的二进制向量-1项集为

1:1011
2:0111
3:1100。
4:1000
5:0101

定义3(频率，Frequent)　给定n位二进制数$B=b_1b_2\cdots b_{n-1}b_n$，我们称$B$中1的个数为$B$的频率，记为$Frequent(B)$。

例3　已知，二进制数$B=0111$，则$Frequent(B)=3$。

定义4(存储项集的结构体)　定义一种Struct，用于存放K-频繁项集。

```
Struct FI_K//存放K频繁项集
{
  Arraylist Item;
  Bittype Item_Index;
}
```

对于上述存储结构，做以下几点说明：

给定$I=\{i_1,i_2,\cdots,i_m\}$为项集，事务数据库$D=\{T_1,T_2,\cdots,T_n\}$，事务$T_i\in D$且$T_i\subseteq I, i=1,2,\cdots n$。

1)项集包含K个元素并且项集在数据集中出现的频率大于或者等于最小支持度与D中事务总数的乘积时，项集才是频繁项集。利用本结构体存储，表示为$|Item|=K$并且$Frequent(Item_Index)>S\times|D|$。

2)一般地，$1<<K<<m$。因此，采用Arraylist结构存储K-频繁项集的元素。Arraylist表示数组列表，零售业数据一般都为数值型矩阵，采用整型列表即可。

3)一般地，事务数据库D很庞大，因此采用Bittype结构来存储K-频繁项集出现在数据库中的位置。Bittype表示二进制串，$Item_Index$的每一个二进制位的存储代价为$n/32$个字节。

定义5(K频繁项集-$K+1$频繁项集关系)　根据定义2以及定义3将事务数据库转化为1-频繁项集，利用定义4提出的结构体存储1-频繁项集，即FI_1。

定义操作“$*$”为$FI_P_i * FI_P_j\to FI_Q_m$($P=1,2,\cdots$; $i\neq j$)，其中FI_Q_m为$K+1$-频繁项集，FI_P_i和FI_P_j为K-频繁项集。FI_Q_m的$Item$元素为FI_P_i和FI_P_j的$Item$元素相“并”后得到，FI_Q_m的$Item_Index$元素为FI_P_i和FI_P_j的$Item_Index$元素相“与”后得到。

例4　利用定义4中给出的Struct存储例2中的二进制向量-1项集元素，设最小支持度$S=50\%$。

FI_1如表2所列。

表 2　1频繁项集 FI_1

Item	Item_Index
1	1 0 1 1
2	0 1 1 1
3	1 1 0 0
5	0 1 0 1

根据定义 5，得到 FI_2 如表 3 所列。

表 3　2频繁项集 FI_2

Item	Item_Index
1 2	0 0 1 1
2 5	0 1 0 1

性质 1　对于任意两个不同的 K-频繁项集 X 和 Y，如果 X 和 Y 的 *Item* 元素只有 $K-1$ 个相同，项集 $X\cup Y$ 有可能成为 $K+1$ 频繁项集。

证明：由于项集 X 和 Y 的 *Item* 只有 $K-1$ 个元素相同，1 个元素不同，这样 X 和 Y 相“并”后，将会有 $K-1+2=K+1$ 个元素，即 $|X.Item \text{ or } Y.Item|=K+1$。因此，项集 $X\cup Y$ 有可能成为 $K+1$ 频繁项集。

推论 1　对于任意两个不同的 K-频繁项集 X 和 Y，如果 X 和 Y 的 *Item* 元素不仅有 $K-1$ 个相同，项集 $X\cup Y$ 不可能成为 $K+1$-频繁项集。

证明：如果项集 X 和 Y 的 *Item* 元素不是仅有 $K-1$ 个相同，分以下两种情况讨论。

1)项集 X 和 Y 的 *Item* 元素有小于 $K-1$ 个相同。

设其相同的元素有 $n(n<K-1)$ 个，$K-n$ 个元素不同。X 和 Y 相“并”后，将会有大于 $K+1$ 个元素：

$$n+2\times(K-n)=2K-n>2K-(K-1)=K+1$$

因此，项集 $X\cup Y$ 不可能成为 $K+1$ 频繁项集。

2)项集 X 和 Y 的 *Item* 元素有大于 $K-1$ 个相同。

X 和 Y 都为 K-频繁项集，则 X，Y 至多有 K 个元素相同。因此，若 X 和 Y 有 K 个元素相同，则 X 和 Y 为同一项集。与条件 X 和 Y 不同相矛盾，不成立。

性质 2　给定 n 位二进制数 P 和 Q，若 $Frequent(PandQ)=m$，表示 P 和 Q 有 m 个二进制位在相同的位置同为 1。

定义 6　对于任意两个不同的 K-频繁项集 X 和 Y，若 X 和 Y 的 *Item* 元素只有 $K-1$ 个相同，*Item_Index* 相“与”后得到 n 位二进制数 p，且 $Frequent(p)>S\times|D|$，则项集 $X\cup Y$ 为 $K+1$ 频繁项集。

证明：根据性质 1，若 X 和 Y 的 *Item* 元素只有 $K-1$ 个相同，项集 $X\cup Y$ 有可能成为 $K+1$ 频繁项集。根据定义 4，$X.Item_Index$ 和 $Y.Item_Index$ 相“与”，表示 $X.Item\cup Y.Item$ 出现在事务数据库中位置。因此，当其频率大于最小支持度与 D 中事务总数的乘积时，$X\cup Y$ 是频繁的。

定义 7(稀疏矩阵)　设矩阵 $m\times n$，有 t 个元素不为零，其中 m 表示数据库中的事务数，n 表示数据库中的项目数，t 表示数据库中所有事务中含有项目数的总和，则称 $\delta=t/(m\times n)$ 为矩阵的稀疏因子[4]。

通常，当矩阵满足稀疏因子 $\delta\leqslant 0.05$ 时为稀疏矩阵。这样的矩阵一般采用三元组形式存储。

另一种情况，若矩阵中非零元素占全部元素的百分比较大(大于 50%)，且分布很有规律，这种矩阵仍可称为稀疏矩阵。这样的矩阵一般采用压缩存储的方式。

本文研究的零售业数据符合第一种稀疏矩阵的形式。

4　基于高维稀疏数据的频繁项集挖掘算法 FIHS

4.1　FIHS 算法的流程

FIHS 算法的流程描述如下：

1)扫描一次数据库，将数据集转化为 1-频繁项集存放。

2)通过递归方法产生频繁项集。具体描述为比较两个 K-频繁项集，如果两个项集的 *Item* 元素只有一个不同，并且两个项集的 *Item_Index* 相“与”后的频率大于最小支持度与 D 中事务总数的乘积，连接操作后即可生成 $K+1$ 频繁项集。递归生成所有的频繁项集。

下面，对 FIHS 的优势进行简单分析。

1)只需扫描一次数据库。扫描一次数据库后即可得到 1-频繁项集。

2)减少了候选项集的产生。Apriori 算法生成 $K+1$ 频繁项集时，需要进行连接和剪枝两个操作。而连接操作一次，相应的剪枝操作要生成 C_{K+1}^{K} 个候选子集，因此会产生大量多余的候选集。FIHS 算法，内存中始终保存最新的 K-频繁项集，$K+1$ 项集的生成是直接根据 K 频繁项集相“与”得到的，无需先连接再剪枝。因此，采用这样的方法优化连接和剪枝，频繁项集产生的效率可以得到大大提高。

3)耗费存储空间少。FIHS 采用二进制数的方式记录频繁项集出现在数据库的位置，存储空间会大大减少。

4.2　FIHS 算法伪代码描述

算法 1　基于高维稀疏数据的频繁项集挖掘算法 FIHS

```
输入:数据集 D,最小支持度 S
输出:频繁项集 FI
//第一步:扫描数据集 D
01. FIHS (Dataset D)
02. BEGIN
    //通过读取数据文件产生 1-频繁项集
03.   FI_1=Read_DataFile();
04.   FIHS_Gen(FI_1);
05. END
//第二步:递归产生所有频繁项集
06. FIHS_Gen (Frequent Itemsets FI_K)
07. BEGIN
08.   IF |FI_K|>1 THEN //判断 FI_K 是否为空。
09.     IF | FI_Ki. Item or FI_Kj. Item|=K+1 THEN
10.       IF Frequent (FI_Ki. Item_Index and
          FI_Kj. Item_Index)>S×|D| THEN
11.         FI_K+1. Add(FI_Ki * FI_Kj);
12.       END IF
13.     END IF //所有的 K 频繁项集判断结束
14.     FI=FI∪FI_K; //保存 FI_K
15.     FI_K. Dispose; //释放 FI_K
16.     FIHS_Gen(FI_K+1); //递归
17.   END IF //IF |FI_K|>1
18. END
```

5　实验对比和性能分析

5.1　实验对比

实验在 Windows XP 系统下实现，通过 C# 开发程序，采用 Pentium4 1.0G 处理器，内存大小为 512MB。采用 FIMI'03 (http://fimi.cs.helsinki.fi/data/) 的两个数据集 T10I4D100K，Retail (数据集的属性对比如表 4) 作为数据输入集，

分别对 FIHS 算法、Apriori 算法和 dEclat 算法生成频繁项集的时间进行对比。

表 4 数据集属性对比

数据集	数据集大小	事务数	项目数	稀疏因子
T10I4D100K	4MB	100000	1000	0.01
Retail	4MB	88162	16470	0.0006

为了验证 FIHS 算法的性能，实验旨在比较算法的运行效率。对于同样的数据集，在实验相同的环境、不同的支持度下比较 3 个算法的运行效率。

图 1 显示在数据集 T10I4D100K 下 3 个算法的运行结果。T10I4D100K 是一个稀疏因子为 0.01、平均事务长度为 10 的人工数据库，包含大量的短模式，比较稀疏。从图 1 中可以看出，Apriori 算法的执行效率是优于 dEclat 算法的，而 FIHS 算法时间消耗是最少的。

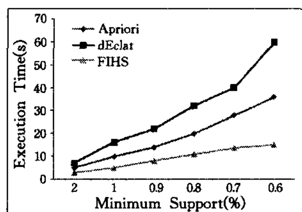

图 1 不同支持度下运行时间对比(T10I4D100K)

图 2 显示在数据集 Retail 下两个算法的运行结果。Retail 是比利时一家零售商店过去某段时间的顾客购买记录，它比 T10I4D100K 更具有稀疏性。从图 2 中可以看出，随着支持度的变化，FIHS 的运行耗时较平稳且较低，而 Apriori 算法在支持度为 0.02 时，有迅速增长的趋势。因此，当事务数据库具有更高的稀疏性时，FIHS 算法的效果更好。

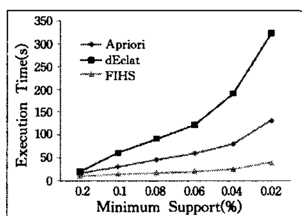

图 2 不同支持度下运行时间对比(Retail)

5.2 性能分析

算法效率的度量有两个指标：算法的执行时间和算法的存储空间需求。下面从时间复杂度和空间复杂度两个方面分析 Apriori 和 FIHS 算法。

假设数据库记录数为 N，平均记录长为 M。不考虑 I/O 操作的代价，假设算法都可以在内存中运行。

(1)时间复杂度分析

判断一个 K-候选项集($Item_K$)是否频繁时，两个算法的时间复杂度分别为：

Apriori 的代价：$C_M^K \times N$。产生每一条事务的所有 K 项集，判断 $Item_K$ 是否在其中。

FIHS 的代价：N。相"与"两个 $K-1$-频繁项集的 $Item_Index$ 部分，计算 $Item_K$ 的频率大小。

一般地，稀疏数据集包含较多短模式($1 << K << M$，$1 < C_M^K$，$N < C_M^K \times N$)。同时，FIHS 避免了产生非频繁的候选项集。理论和实验都可以证明，FIHS 的时间代价较小。

(2)空间复杂度分析

Apriori 算法存储整个数据集以及候选项集，FIHS 动态存储频繁项集，内存中始终仅保存最新的 K-频繁项集。算法的空间复杂度分别为：

Apriori 的代价：$N \times M + K' \times K''$。其中，$K''$ 表示候选 K 项集的个数。

FIHS 的代价：$K' \times (K + N/32)$。其中，K' 表示 K-频繁项集的个数，$Item_Index$ 的存储代价为 $N/32$。

一般地，针对稀疏数据集，$K' << K''$。因此，多数情况下，FIHS 的空间代价是较小的。

结束语 针对零售业数据集稀疏和高维的特点，在 4 个优化目标的基础上提出了基于高维稀疏数据的频繁项集挖掘算法 FIHS。FIHS 运用在高维稀疏数据集上，性能有较大的提高，但是始终没有跳出 Apriori 算法的框架，所以在性能提高方面总是会受到一些限制。以后研究的方向：

1)新的挖掘模式，完全脱离 Apriori 算法的框架。

2)零售业数据的动态更新特征，考虑高效的增量挖掘方法。

参考文献

[1] Agrawal R, Srikant R. Fast Algorithm for Mining Association rules[C]//Proceedings of the 20th International Conference on VLDB. Santiago, Chile, 1994: 487-499

[2] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation [J]. ACM-SIGMOD International Conference on Management of Data, 2000, 29(2): 1-12

[3] Zaki M J. Fast vertical mining using diffsets[R]. 01-1. Troy, New YorkDepartment of Computer Science, Rensselaer Polytechnic Institute, 2001

[4] Im E-J, Yelick K, Vuduc R. Sparsity: Optimization framework for sparse matrix kernels[J]. International Journal of High Performance Computing Applications, 2004, 18(1): 135-158

[5] Hipp J, Guntzer U, Nakhaeizadeh G. Algorithms for association rule mining-A general survey and comparison[J]. SIGKDD Explorations, 2000, 2(1): 58-64

[6] 陈耿，朱玉全. 关联规则挖掘中若干关键技术的研究[J]. 计算机研究与发展, 2005, 42 (10): 1785-1789

[7] Dong J, Han M. BitTableFI: An efficient mining frequent itemsets algorithm [J]. Knowledge-Based Systems, 2007, 20(4): 329-335

[8] Zheng X Y, Sun J Z, Zheng X Y. Finding Frequent Item Sets from Sparse Matrix[C]//International Conference on Electronic Computer Technology. 2009: 615-619

[9] Song M J, Rajasekaran S. A transaction mapping algorithm for frequent itemsets mining [J]. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(4): 472-481

[10] Lucchese C, Orlando S, Perego R. Fast and memory efficient mining of frequent closed itemsets[J]. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(1): 21-36

[11] Kunkle D, Zhang D, Cooperman G. Mining frequent generalized itemsets and generalized association rules without redundancy [J]. Journal of Computer Science and Technology, 2008, 23(1): 77-102

[12] Hahsler M, Grün B, Hornik K. Arules-A Computational Environment for Mining Association Rules and Frequent Item Sets [J]. Journal of Statistical Software, 2005, 14(15): 1-25