doi :10.3969/j.issn.1671-1122.2015.11.013

数据挖掘中一种增强的 Apriori 算法分析

胡雪[1]，封化民[1,2]，李明伟[1]，丁钊[3]

（1. 北京电子科技学院，北京 100070；2. 西安电子科技大学通信工程学院，陕西西安 710071；
3. 西安电子科技大学计算机学院，陕西西安 710071）

摘　要：在当今这个信息极度发达的社会，网络数据急剧膨胀，激增的数据背后隐藏着许多重要的信息，所以对大量数据进行分析是必要的。Apriori 算法是一种挖掘关联规则的频繁项集算法，其核心思想是通过候选集生成和情节的向下封闭检测两个阶段来挖掘频繁项集。可能产生大量的候选集，以及可能需要重复扫描数据库是 Apriori 算法的两大缺点。文中提出了一种需要更少的扫描时间的 Apriori 算法，在剪枝候选项集的同时也在消除冗余的子项集的产生。改进的 Apriori 算法通过消除数据库中不需要记录的传输有效减少了 I/O 所花费的时间，Apriori 算法的效率得到了极大的优化。文章给出了算法实现思想及证明，并对传统的和改进的 Apriori 算法进行比较和分析。

关键词：数据挖掘；关联规则；频繁项集；事务数；支持计数

中图分类号：TP309　　**文献标识码：**A　　**文章编号：**1671-1122（2015）11-0077-07

中文引用格式：胡雪，封化民，李明伟，等．数据挖掘中一种增强的 Apriori 算法分析 [J]. 信息网络安全，2015，（11）：77-83

英文引用格式：HU X, FENG H M, LI M W, et al. Analysis of An Enhanced Apriori Algorithms in Data Mining[J]. Netinfo Security, 2015, (11):77-83.

Analysis of An Enhanced Apriori Algorithms in Data Mining

HU Xue[1], FENG Hua-min[1,2], LI Ming-wei[1], DING Zhao[3]

(1. *Beijing Electronic Science and Technology Institute, Beijing* 100070, *China*; 2. *Communication Engineering Institute, Xidian University, Xi'an Shanxi* 710071, *China*; 3. *School of Computer Science and Technology, Xidian University, Xi'an Shanxi* 710071, *China*)

Abstract: In the highly developed information society, network data expand rapidly and much important information hide behind the surge of data. So it is necessary that analyze a large amounts of data. Apriori algorithm is a frequent item set algorithm for mining association rules. Its core idea is to excavate frequent item sets through two stages including generating candidate sets and closed down testing of plot. May generate a large number of candidate sets and may need to repeat scanning database are the two major drawbacks of Apriori algorithm. By eliminating unnecessary transmission of records in the database, the improved Apriori algorithm effectively reduces the time spent on I/O, greatly optimizes the efficiency of the algorithm, proves and gives the algorithm implementation thought. In this paper, an enhanced Apriori algorithm is proposed which takes less scanning time. It is achieved by eliminating the redundant generation of sub-items during pruning the candidate item sets. Both traditional and enhanced Apriori algorithms are compared and analyzed in this paper.

Key words: data mining; association rule; frequent item sets; transaction number; support counting

收稿日期：2015-09-01

基金项目：国家自然科学基金 [61103210]；中央高校基本科研业务费专项资金 [2015XS1-LB，38201541]

作者简介：胡雪（1990-），男，山东，硕士研究生，主要研究方向：计算机网络、数据挖掘算法研究；封化民（1963-），男，陕西，教授，博士，主要研究方向：信息安全、网络安全、密码学；李明伟（1991-）男，安徽，硕士研究生，主要研究方向：数据挖掘、机器学习算法研究与应用；丁钊（1992-）男，山东，硕士研究生，主要研究方向：数据挖掘算法研究与应用、信息安全。

通讯作者：胡雪 zchuxue@163.com

0 引言

随着计算机技术、网络技术和通信技术的发展，以及各行各业业务操作流程的自动化，企业内积累了大量业务数据，这些数据的数量需以TB计算[1]。这些数据对于企业来说一种财富，但由于无法对其进行有效的处理而变成一种负担。这主要是因为产生的这些数据数据量极其庞大，传统的处理方法在处理大数据时显得力不从心。在这种情况下，迫切需要一种新的数据处理和整合方法[2]。数据挖掘就是从大量数据中挖掘出隐含的、未知的、对决策有潜在价值的关系、模式和趋势的一种技术，并利用这些知识和规则建立用于决策支持和安全防御的模型，提供预测性决策支持。将数据挖掘技术用于网络信息安全领域等方面也逐步受到各方的重视，特别是将数据挖掘技术运用于网络安全审计的研究。经过二十多年的科研攻关，许多前沿的技术被应用于安全审计领域，许多创新性的方法也在安全审计领域被提出，这些方法为进一步研发更高性能的分析技术提供了重要依据。在信息安全技术处于领先地位的美国，多项基于数据挖掘的网络安全审计研究得到了许多安全和科研部门的重视，包括国防部高级研究计划署、国家自然科学基金等。然而，只有德国的Inspect分布式网络安全审计系统、Purdue大学的NASHID系统等在数据安全审计领域取得了一些成绩，其他科研机构还没有研究出十分成熟的网络安全审计系统。检测网络中的异常行为是这些系统的主要目的[3]。而且将数据挖掘技术运用于网络安全审计的研究工作很大一部分还停留在理论研究阶段，总体上处于初级阶段，离实际应用还有一定的距离[4]。系统输入通常为收集到的审计数据，输出为经过系统挖掘分析后的审计结果。虽然中国在安全审计领域的研究也在紧锣密鼓地进行，但是应该清楚地看到我们离世界先进水平还很远，远不如国际上的研究深刻和广泛。

关联规则算法作为一种广泛应用的数据挖掘算法，其目的是在一个数据集中找出各项之间的相关关系，是Agrawal等人在1993年首先提出的，于此同时Agrawal还给出了相应的挖掘算法AIS，但算法性能不是很高[5]。1994年，Agrawal和Srikant共同提出了著名的基于逐层迭代思想的Apriori算法，至今Apriori算法仍然被广泛应用于社会生产的各行各业，以后许多数据挖掘领域的研究人员对关联规则的挖掘问题进行了大量的研究。Savasere等人[6]提出了一种基于划分思想的频繁项集发现算法；Tiovonen[7]提出了基于抽样的思想算法，该策略实际上是以牺牲精度为代价来换取算法的执行效率，适合以效率为重的挖掘任务；Crestana-Jensen等人[8]提出的JSApriori算法采用逐层迭代的思想在多关系挖据中挖掘关联规则。

在网络安全领域Apriori算法取得了广泛应用，将关联规则算法用于入侵检测技术中也是当前研究的热点[7]。网络发展日新月异使得攻击手段不断多样化，传统的仅靠单一的基于模式匹配方法的入侵检测技术难以适用于目前的网络状况。与其他网络安全产品不同的是，智能化的需求在入侵检测系统中格外迫切，它需要具有快速、有效分析数据的能力，并以友好的方式给出有用的结果。一个合格的入侵检测系统可以最大限度地减轻网络管理员的工作，大大提高系统检测的准确性，保证网络系统安全可靠地运行。采用作用度的Apriori算法削弱了Apriori算法的挖掘结果规则，使网络入侵检测系统可以精确地发现用户的行为模式，能够高效地分析出网络攻击者，提高了基于关联规则的入侵检测系统的检测性能。

关联规则挖掘技术能从大量数据中挖掘出有价值、反映数据项之间潜在联系的知识信息。随着数据不断的积累，存储在数据库中的数据规模越来越大，从这些数据中挖掘相应的关联知识信息显得尤为重要，这些关联知识信息可以为决策者做出前沿判断提供有力支撑[9]。

1 传统的Apriori算法

关联规则挖掘是从事务集合中挖掘出满足支持度和置信度最低阈值要求的所有关联规则，这样的关联规则也称强关联规则[10]。常用的关联规则算法如表1所示。

表1 常用的关联规则算法

算法名称	算法描述
Apriori	一种最有影响的挖掘布尔关联规则频繁项集算法，其核心是基于两阶段频集思想的递推算法。
FP-tree	一种不产生候选挖掘频发项集的关联规则算法。
灰色关联法	分析和确定各因素之间的影响程度或若干个子因素（子序列）对主因素（母序列）的贡献程度。
HotSpot	挖掘通过树状结构显示的感兴趣的目标最大化/最小化的一套规则，最大化/最小化的利益目标变量/值。

关联规则挖掘是从事务集合中挖掘出支持度和置信度大于最低阈值(*minsup,minconf*)的关联规则，该阈值是由用户根据公式(1)、公式(2)指定。

$$支持度 = (X,Y)_{.count}/T_{.count} \quad \cdots\cdots(1)$$

$$置信度 = (X,Y)_{.count}/X_{.count} \quad \cdots\cdots(2)$$

其中，$T_{.count}$ 表示所有事务的集合个数，$(X,Y)_{.count}$ 表示 T 中同时包含 X 和 Y 的事务的个数，$X_{.count}$ 表示 T 中包含 X 的事务的个数。

关联规则挖掘总体过程主要包括生成频繁项目集和产生强关联规则两步[11]。

1）以事先设定的最小支持度为根据，找出事务集合 D 中所有的频繁项目集。最终目的是根据 K- 频繁项目集生成需要的 $(K+1)$- 频繁项目集[12]。

2）由频繁项目集和最小支持度产生强关联规则。一种比较容易的办法就是遍历所有的频繁项目集，然后从每个项目集中依次取 1,2,...,K 个元素作为后件，该项目集中的其他元素作为前件，通过计算该规则的置信度筛选出强关联规则。但是这种方法的效率不是很高[13]。

Apriori 关联规则挖掘的两步中，步骤 1）根据最小支持度找出事务集合 D 中所有的频繁项目集是关联规则挖掘的中心问题，能否迅速高效地找出集合 D 中所有频繁项目集会对算法效率产生很大影响。步骤 2）根据频繁项目集和最小支持度产生强关联规则求解相对简单和直接，但也需要根据用户和应用场景的不同来设置求解指标(如兴趣度的度量标准)。

1.1 Apriori算法计算复杂度

Apriopri 算法的计算复杂度主要由支持度阈值、项数、事物数 3 个方面因素决定[14]。

1.1.1 支持度阈值

降低支持度阈值导致的后果就是频繁项集的增多，这会大大增加算法的计算复杂度，因为支持度阈值降低产生的候选项集大量增加并要统计候选项集的个数。此外，支持度阈值的降低还将增加频繁项集的最大长度，频繁项集最大长度的增加带来的后果就是算法需要更多次数的扫描数据集。所以，支持度阈值会对算法的计算复杂度产生很大影响。

1.1.2 项数(维度)

Apriori 算法项数的增加导致的一个直接结果就是支持度计数项的增多，为了存储支持度计数项就需要更大的存储空间。如果频繁项集的数目也随着数据维度增加而增长，那么算法产生的候选项集将会变得十分庞大，因此，计算量和 I/O 开销将明显增加。

1.1.3 事务数

Apriori 算法的一个特性就是运行时间随着事务数增加而增加，这是因为 Apriori 算法需要反复扫描数据集。Apriori 算法采用逐层搜索的迭代方法，其核心思想是利用 k 项集来探索 $(k+1)$ 项集[15]。首先，通过扫描数据库，记录每个项的个数，从而产生频繁 1- 项集，同时收集满足最小支持度的这些项，得到的集合记为 L_1。接下来，用 L_1 产生频繁 2- 项集 L_2，然后用 L_2 来产生频繁 3- 项集 L_3，依此类推，直到没有新的频繁 k- 项集产生，即 $L_k=\varnothing$。每产生一个 L_k 都需要把数据库全部扫描一遍。为提高逐层迭代产生频繁项集的效率，应用先验原理可以减少搜索空间，提高逐层迭代产生频繁项集的效率[16]。

1.2 先验性质

先验性质(apriori property)要求频繁项集的所有非空子集也一定是频繁的。先验性质是一种用于压缩空间的重要性质，先验性质的应用可以提高频繁项集逐层产生的效率。根据定义，如果项集 I 不满足最小支持度阈值 min_sup，则 I 不是频繁的，即 $P(I)<min\_sup$[17]。如果把项 A 添加到项集 I 中，则结果项集(即 $I\cup A$)不可能比 I 更频繁出现，因此，$I\cup A$ 也不是频繁的，即 $P(I\cup A)<min\_sup$。例如，如果 {1,2} 出现的次数小于最小支持度(非频繁的)，那么超集 {0,1,2} 的组合肯定也是非频繁的。频繁项集的产生需要合并和剪枝两个步骤。

1.3 经典的Apriori算法

下面将给出 Apriori 算法产生频繁项集的部分伪代码。事务集合 D 和最小支持度阈值(min_sup)作为算法输入。D 中的频繁项集 L 作为算法输出。具体算法如下。

```
L1=find_frequent_1-itemsets(D);// 从事务集合 D 中发现所有的频繁 1- 项集，定义为 L1
for(k=2; L_{k-1} ≠ Φ; k++){
  C_k=apriori_gen(L_{k-1}, min_sup); // 产生候选项集
  for each log t D{// 扫描数据库，并计数
    C_t=subset(Ck,t); // 获得 t 所包含的候选项集
```

for each log t C_t// 循环遍历每个候选项集 c∈C_t

c.count++;}

L_k={ c∈C_k | c.count ⩾ min_sup }// 提取频繁 k- 项集

return L=∪$_k L_k$;

apriori_gen(L_{k-1}:frequent(k-1)-itemsets)

// 产生 k- 项候选项集

for each itemset l_1∈L_{k-1}{ // 循环遍历每个项

for each itemset l_2∈L_{k-1}{

if(l_1 [1]= l_2 [1])^(l_1 [2]= l_2 [2]) ^...^(l_1 [k-1] = l_2 [k-1])^(l_1 [k-1]< l_2 [k-1]) {

c=l_1 ∞ l_2; // l_1 和 l_2 做笛卡尔积

if has_infrequent_subset(c,L_{k-1})

delete c; // 删除非频繁的候选项

else add c to C_k ; }

return C_k;

has_infrequent_subset (c: candidate k-itemset; L_{k-1}:frequent(k-1)-itemsets) // 使用先验知识

for each itemset (k-1)-subset s of c {

if s ∉L_{k-1}

return true;

return false; }

上述算法中的 apriori-gen 函数通过候选项集的产生和候选项集的剪枝操作产生候选项集。

1.4 使用传统Apriori算法产生频繁项集

表 2 是一个日志审计数据库的具体例子。在这个数据库中有 9 条交易记录，也就是 |*D*|=9。T_1,T_2,...,T_9 表示一条日志记录项集，*a*,*b*,*c*... 表示一条日志记录中的项。

表2 某日志审计数据库

TID	*item_ID* 的列表
T_1	*a*,*b*,*e*
T_2	*b*,*d*
T_3	*b*,*c*
T_4	*a*,*b*,*d*
T_5	*a*,*c*
T_6	*b*,*c*
T_7	*a*,*b*,*c*,*d*
T_8	*a*,*b*,*c*,*e*
T_9	*a*,*b*,*c*

下面的步骤用来解释候选项集和频繁项集的产生，*Itemset* 表示项集，*Sup.count* 表示支持度计数。表 1 的事务数据最小支持计数为 3。

1）扫描事务集合 *D*，统计每个候选项的支持度计数，如表 3 所示。

表3 候选1-项集选项的支持度计数

Itemset	*Sup.count*
[*a*]	6
[*b*]	8
[*c*]	6
[*d*]	3
[*e*]	2

2）比较候选项支持度计数与最小支持度计数，删掉不符合条件的候选，产生频繁 1- 项集 L_1，如表 4 所示。

表4 频繁1-项集L_1

Itemset	*Sup.count*
[*a*]	6
[*b*]	8
[*c*]	6
[*d*]	3

3）从 L_1 中产生候选 2- 项集 C_2，如表 5 所示

表5 候选2-项集C_2

Itemset
[*a*,*b*]
[*a*,*c*]
[*a*,*d*]
[*a*,*e*]
[*b*,*c*]
[*b*,*d*]
[*b*,*e*]
[*c*,*d*]
[*c*,*e*]
[*d*,*e*]

4）扫描事务集合 *D*，对每个候选项计数，如表 6 所示。

表6 候选2-项集候选项计数

Itemset	*Sup.count*
[*a*,*b*]	5
[*a*,*c*]	4
[*a*,*d*]	2
[*a*,*e*]	2
[*b*,*c*]	4
[*b*,*d*]	3
[*b*,*e*]	2
[*c*,*d*]	0
[*c*,*e*]	1
[*d*,*e*]	0

5）将候选项支持度计数与最小支持度比较，产生频繁 2- 项集 L_2，如表 7 所示。

表7 频繁2-项集L_2

Itemset	*Sup.count*
[*a*,*b*]	5
[*a*,*c*]	4
[*a*,*d*]	2
[*a*,*e*]	2
[*b*,*c*]	4
[*b*,*d*]	3
[*b*,*e*]	2

6）从 L_2 中产生候选 3- 项集 C_3，如表 8 所示。

表8 候选3-项集C_3

Itemset
[a,b,c]
[a,b,e]
[a,b,d]

7）扫描事务集合 D，对每个候选项计数，如表 9 所示。

表9 候选3-项集候选项计数

Itemset	Sup.count
[a,b,c]	3
[a,b,e]	2
[a,b,d]	2

8）将候选项支持度计数与最小支持度比较，删掉不符合条件的候选，产生频繁 3- 项集 L_3，如表 10 所示。

表10 频繁3-项集L_3

Itemset	Sup.count
[a,b,c]	3

1.5 Apriori算法的改进方法

1.5.1 基于动态项集计数的改进方法

基于动态项集计数的优化方法将数据库划分为标记开始点的块，算法可以在任何开始点添加新的候选项集。该方法动态地评估已被计数的所有项集的支持度，如果一个项集的所有子集已被确定为频繁的，则添加它作为新的候选项集。

1.5.2 基于哈希（Hash）表技术的改进方法

利用 Hash 表技术可以帮助有效减少候选 k- 项集 $C_k(k \geqslant 1)$ 所占用的空间。利用这样 Hash 表技术可以有效减少需要检查的候选项集数目。

1.5.3 基于采样的优化方法

基于采样的优化方法是在一个给定数据库 D 的随机样本 S 中进行挖掘，这种方法牺牲了精确性换取有效性，内存的大小决定了样本 S 的大小。样本 S 中的频繁项集不一定是数据库 D 中的频繁项集，而且，数据库 D 中的频繁项集不一定出现在样本 S 的频繁项集中。

2 改进的 Apriori 算法

对于非常大的数据库，传统 Apriori 算法产生庞大的候选项集，所以它需要更大的计算代价。为了避免这个缺点，本文介绍一种改进的 Apriori 算法来减少候选集数量。本文提出的系统变量 *transaction_size(TS)* 包括个人交易项目的数量，如果 *transaction_size* 小于阈值，这一交易将从数据库中删除。

2.1 算法描述

改进的 Apriori 算法通过单遍扫描数据集确定每个项的支持度，进而就得到所有的频繁 1- 项集的集合 L_1[17]。依照此方法，使用上一次迭代发现的频繁 $(k-1)$- 项集，产生新的候选 k- 项集[18]。*apriori-gen* 函数负责实现候选项的产生，算法需要再次扫描数据集，计算出候选项的支持度。计算候选项的支持度计数之后，算法将删除支持度计数小于 *min_sup* 的所有候选项集。更新数据库，通过扫描事务集合 D、L_{k-1}、L_k，将存在 L_{k-1} 中，不存在 L_k 中且未在频繁 1- 项集中出现的元素从数据库中删除。最后再一次扫描事务集合 D，计算每个事物中的元素个数，若元素个数小于 k 则删除该条事物。当没有新的频繁项集产生，即 $L_k=\varnothing$ 时算法结束[19,20]。

下面给出了改进的 Apriori 算法产生频繁项集的部分伪代码。事务集合 D 和最小支持度阈值（*min_sup*）作为算法输入，D 中的频繁项集 L 作为算法输出。具体实现算法如下。

```
L1 = find_frequent_1-itemsets(D);
    for(k=2; Lk-1 ≠ Ø; k++){
        Ck = apriori_gen(Lk-1,min_sup);
        if(k=2){
            delete_datavalue(Dk-1,Lk,Lk-1); }// 只在生成频繁 2- 项集时进行删项操作
        if(k>=2){
            delete_datarow (Dk-1,Lk); } // 删除数据库中不合符条件的记录
        for each log t∈D k-2{
            Ct = subset(Ck,t);
            for each candidate c∈Ct
                c.count++; }
        Lk = { c∈Ck||c.count ⩾ min_sup };
        return L=∪kLk};
Procedure apriori_gen(Lk-1: frequent(k-1)-itemsets)
    for each itemset l1∈ Lk-1 {
    for each itemset l2∈ Lk-1 {
        if(l1 [1]= l2 [1])^(l1 [2]= l2 [2]) ^...^ (l1 [k-1]= l2[k-1]) ^ (l1 [k-1]<l2 [k-1]) then {
            c=l1 ∞ l2;
            for each itemset l1∈ Lk-1 {
            for each candidate c∈ Ck {
```

if l1 is the subset of c then

c.num++; }}}}}

C'_k={ c∈C_k |c.num=k};

return C'_k;

Procedure delete_datavalue (

D_{k-1}:Database; Lk: frequent (k)-itemsets; L_{k-1}: frequent(k-1) – itemsets) // 删除

D_{k-1} 中项的元素

for each itemset i∈L_{k-1} and i ∉L_k){

for each log t∈D_{k-1}){

for each datavalue∈t{

if (datavalue=i)

update datavalue=null; }}}

for each log t∈D_{k-1}){

for each datavalue∈t{

if(datavalue!=null || datavalue!=0)

delete datavalue; }} // 删除记录中不满足支持度的元素

Procedure delete_datarow

(D_{k-1}: Database; Lk:frequent(k) – itemsets)

for each log t∈D_{k-1}){

if(TS.count <k){

delete datarow;} }// 删除 TS 计数小于所求频繁项集数的记录

2.2 算法实例

以下步骤是使用改进的 Apriori 算法产生表 11 的频繁项集，*TS* 表示事务元素个数。

1）扫描表 11 每个 1– 项集的计数，生成候选 1– 项集，即表 12，删除候选 1– 项集中不满足最小支持度的项集，生成频繁 1– 项集，即表 13。

表11 原始日志审计数据库实验数据

TID	*item_ID*	*TS*
T_1	*a,b,e*	3
T_2	*b,d*	2
T_3	*b,c*	2
T_4	*a,b,d*	3
T_5	*a,c*	2
T_6	*b,c*	2
T_7	*a,b,c,d*	4
T_8	*a,b,c,e*	4
T_9	*a,b,c*	3

表12 用改进后算法产生的候选1–项集

Itemset	*Sup.count*
[*a*]	6
[*b*]	8
[*c*]	6
[*d*]	3
[*e*]	2

表13 用改进后算法产生的频繁1–项集

Itemset	*Sup.count*
[*a*]	6
[*b*]	7
[*c*]	6
[*d*]	3

2）执行更新数据库的操作，删除数据库中未在频繁 1– 项集中出现的项，同时更新数据库中 *TS* 的计数。检查新数据库中每项事务 *TS* 是否小于 *k*，若小于 *k*，则在数据库中删除该项事务（*k* 为候选项的个数），完成数据库更新，如表 14 所示。根据已产生的频繁 1– 项集，进行连接操作，生成候选 2– 项集，如表 15 所示。扫描数据库记录候选 2– 项集支持度计数，删除候选 2– 项集中不满足最小支持度计数的项，生成频繁 2– 项集，如表 16 所示。

表14 更新一次后的日志审计数据库

TID	*item_ID*	*TS*
T_1	*a,b*	2
T_2	*b,c*	2
T_3	*b,c*	2
T_4	*a,b,d*	3
T_5	*a,c*	2
T_6	*b,c*	2
T_7	*a,b,c,d*	4
T_8	*a,b,c*	3
T_9	*a,b,c*	3

表15 用改进后算法产生的候选2–项集

Itemset	*Sup.count*
[*a,b*]	5
[*a,c*]	4
[*b,c*]	5

表16 用改进后算法产生的频繁2–项集

Itemset	*Sup.count*
[*a,b*]	5
[*a,c*]	4
[*b,c*]	5

3）再次执行更新数据库的操作，此时不再进行删项操作，只检查新数据库中每项事务 *TS* 是否小于 *k*，若小于 *k*，则在数据库中删除该项事务（*k* 为候选项的个数），完成数据库更新，新数据库如表 17 所示。根据已产生的频繁 2– 项集，进行连接操作，生成候选 3– 项集，如表 18 所示。扫描更新后的数据库记录候选 3– 项集支持度计数，删除候选 3– 项集中不满足最小支持度计数的项来生成频繁 3– 项集，如表 19 所示。

表17 更新两次后的日志审计数据库

TID	item_ID	TS
T_4	a,b,d	3
T_7	a,b,c,d	4
T_8	a,b,c	3
T_9	a,b,c	3

表18 用改进后算法产生的候选3-项集

Itemset	Sup.count
[a,b,c]	3

表19 用改进后算法产生的频繁3-项集

Itemset	Sup.count
[a,b,c]	3

3 传统 Apriori 和改进的 Apriori 算法性能比较

本文在不同的实例集和置信度上分别比较了两种算法的性能，如图 1 所示。

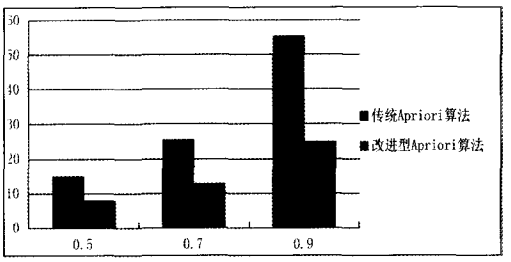

图1 算法性能比较

图 1 中，纵轴代表时间，横轴代表置信度。

从图 1 可以看出，对于 3 个不同的置信度执行改进的 Apriori 算法都比传统的 Apriori 算所花的时间少。因此，改进的 Apriori 算法是一个高效、可扩展的完整的频繁模式挖掘方法。

4 结束语

本文通过分析 Apriori 算法，结合文献中已有的经典 Apriopri 算法，提出了一种新的更高效的算法，并给出了算法实现思想。改进的 Apriori 算法通过消除数据库中不必要记录的传输有效减少了 I/O 所花费的时间，Apriori 算法的效率得到了极大的优化，从海量数据中挖掘关联信息将会更快。

评估这两种算法的性能是基于它们产生关联规则所用的时间。经过分析可以得出，改进的 Apriori 算法的性能比传统的 Apriori 算法更好。

但本文提出的改进 Apriori 算法除了具有 Apriori 普遍的缺点——产生大量的频繁项集外，当数据库中的频繁项集中元素个数较多时，算法执行时并没有太多冗余项可删除，这就造成了无谓的扫描数据库。因此该算法在有些情况下也会增加扫描数据库的次数，如何避免这种情况将是进一步研究的问题。（责编 潘海洋）

参考文献：

[1] 陈晓，赵晶玲．大数据处理中混合型聚类算法的研究与实现 [J]. 信息网络安全，2015，(4)：45-49.

[2] 刘步中．基于频繁项集挖掘算法的改进与研究 [J]. 计算机应用研究，2012，29 (2)：475-477.

[3] 崔贯勋，李梁，王柯柯，等．关联规则挖掘中 Apriori 算法的研究与改进 [J]. 计算机应用，2010,30 (11)：2952-2955.

[4] 吴旭，郭芳毓，颉夏青，等．面向机构知识库结构化数据的文本相似度评价算法 [J]. 信息网络安全，2015，(5)：16-20.

[5] AGRAWAL R, IMIELIŃSKI T, SWAMI A. Mining association rules between sets of items in large databases [J]. ACM SIGMOD Record, 1993, 22(2): 207-216.

[6] SAVASERE A, OMIECINSKI E, NAVATHE S B. An Efficient Algorithm for Mining Association Rules in Large Databases[C]//21th International Conference on Very Large Data Bases, San Francisco, CA, USA, 1995: 432-444.

[7] TIOVONEN H. Sampling Large Databases for Association Rules[C]//22th International Conference on Very Large Data Bases, San Francisco, CA, USA , 1996: 134-145.

[8] CRESTANA-JENSEN V, SOPARKAR N. Frequent Itemset Counting Across Multiple Tables[C]//4th Pacific-Asia Conference, PAKDD 2000, Kyoto, Japan, 2000: 49-61.

[9] 许为，林柏钢，林思娟，等．一种基于用户交互行为和相似度的社交网络社区发现方法研究 [J]. 信息网络安全，2015，(7)：77-83.

[10] BAY S D, PAZZANI M J. Detecting Group Differences: Mining Contrast Sets[J]. Data Mining & Knowledge Discovery, 2002, 5(3): 213-246.

[11] AGARWAL R C, AGGARWAL C C, PRASAD V V V. A Tree Projection Algorithm For Generation Of Frequent Itemsets[J]. Journal of Parallel & Distributed Computing, (Special Issue on High Performance Data Mining), 2000, 61(3): 350-371.

[12] ZHANG G L, LEI J S, WU X H. An Improved Apriori Algorithm for Mining Association Rules[J]. Microelectronics & Computer, 2010, 23(2): 10-12.

[13] 吴坚，沙晶．基于随机森林算法的网络舆情文本信息分类方法研究 [J]. 信息网络安全，2014，(11)：36-40.

[14] JR R J B. Efficiently mining long patterns from databases[C]//The ACM International Conference on Management of Data, Washington, 1998, 27:85-93.

[15] 高聪．Deep Web 下不确定数据处理的研究 [D]. 沈阳：东北大学，2008.

[16] AGRAWAL R, SRIKANT R. Fast Algorithms for Mining Association Rules in Large Databases[C]//The 20th International Conference on Very Large Data Bases, CA, USA, 1994: 487-499.

[17] 孙兴东，李爱平，李树栋．一种基于聚类的微博关键词提取方法的研究与实现 [J]. 信息网络安全，2014，(12)：27-31.

[18] HAN J, PEI J, YIN Y, et al. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach[J]. Data Mining & Knowledge Discovery, 2004, 8(1):53-87.

[19] HUNYADI D. Performance Comparison of Apriori and FP-Growth Algorithms in Generating Association Rules[J]. 5th European conference on European computing conference, Wisconsin, 2011: 376-381.

[20] GOSWAMI D N, CHATURVEDI A, RAGHUVANSHI C S. Efficient Algorithm for Frequent Pattern Mining Based On Apriori[J].