

基于一次性数据库访问策略的关联规则挖掘算法的研究

刘培奇, 卢 麟, 廖福燕, 宋 阳

(西安建筑科技大学 信息与控制工程学院, 陕西 西安 710055)

摘 要: 针对大型数据库挖掘中需要多次访问数据库和效率较低的局限性, 提出一次性数据库访问策略, 设计了基于 Apriori 的 Apriori_ADO 算法. 经过实验分析, Apriori_ADO 算法降低了算法的时间和空间开销, 提高了关联规则挖掘效率. 算法具有很强的实用性, 已用于超市中顾客消费知识的挖掘, 并取得了满意的结果.

关键词: 关联规则; Apriori 算法; 频繁项集; 支持度

中图分类号: TP301.6

文献标识码: A

文章编号: 1000-7180(2010)12-0022-04

The Research on the Mining Association Rules Algorithm Based on the Strategy Accessing to Database Once

LIU Pei-qi, LU Lin, LIAO Fu-yan, SONG Yang

(School of Information and Control Engineering, Xi'an University of Architecture and Technology, Xi'an 710055, China)

Abstract: This article analyzes the Apriori algorithm, and points out the limitations that it requires many accesses in the large-scale database mining and its efficiency is low. Based on the strategy accessing to DB once, it designs Apriori_ADO algorithm based on the Apriori. This algorithm avoids cutting frequently. Experimental analysis shows that the new improved Apriori_ADO algorithm reduces the time and space, and improves the mining efficiency of association rules. Apriori_ADO algorithm has strong practicality; it has been used for supermarkets in customer consumer knowledge mining, and has achieved satisfying results.

Key words: association rules; Apriori algorithm; frequent itemsets; support

1 引言

数据挖掘^[1]是从数据库、数据仓库或海量数据中挖掘有趣知识的过程. 随着实际应用中数据量的急剧增加, 许多专业人士对数据挖掘的相关知识越来越感兴趣, 其中关联规则挖掘更是大家关注的焦点. 关联规则挖掘是从大量数据中发现项集间有趣强规则的一种挖掘方法.

Apriori 算法是一种经典关联规则挖掘算法^[2-3]. 它使用逐层搜索的迭代方法, 由 k -项频繁集用于探索 $(k+1)$ -项集. 首先, 计算所有候选 1-项集^[4] C_1 , 找出所有频繁 1-项集 L_1 ; 再由 L_1 确定候选 2-项集 C_2 , 并从 C_2 中找出频繁 2-项集 L_2 . 重复上述过程, 直到找不到频繁 k -项集 L_k 为止. 需要注意的是, 寻找每个

L_k 需扫描数据库一次, 对大型数据库时间开销较大.

在许多情况下, Apriori 算法的候选集的产生-检查方法很大程度上压缩了候选项集的大小, 能产生很好的性能. 然而, Apriori 算法在计算时可能产生大量的候选集, 尽管应用了一些剪枝策略, 也需要多次扫描数据库. 这两种时间开销也很大^[5].

根据上述分析, 可从减少候选集数量和数据库扫描次数这两方面来改进 Apriori 算法. 基于这一思想, 本文提出了基于一次性访问数据库的关联规则挖掘算法.

2 Apriori_ADO 算法设计与分析

2.1 算法改进思想

在传统 Apriori 算法中, 每产生一个 C_k , 都要访

收稿日期: 2009-12-13; 修回日期: 2010-01-13

基金项目: 国家自然科学基金项目(60673170); 陕西省教育厅自然科学基金专项(08JK318); 校人才基金项目(RC0618)

问一次数据库. 频繁地访问数据库, 严重影响了挖掘效率. 经研究分析, 在 Apriori 访问数据库中, 每次仅用了其中很少信息, 而大部分信息被丢弃. 因此, 可在访问数据库后, 一次性提取记录所有信息, 避免再次访问数据库. 如果 k 项集的任意子集是非频繁集, 则 k 项集一定是非频繁项集. 在产生候选集的时候, 利用 Apriori 算法的这一性质, 可知如果第 i 条记录中项 ab 为频繁集, 则对 i 后所有记录该项也是频繁集. 随着扫描过程中进行比较的项变少, 可以极大减少 $C_k (k > 1)$ 数量. 在挖掘中有两个明显的性质^[6]:

性质 1 若项集 x 是频繁项集, 则 x 的子集也为频繁项集;

性质 2 若项集 x 在第 i 条记录时为频繁项集, 则 x 在第 j 条 ($j \geq i$) 记录时也为频繁项集.

这两条基本性质奠定了新改进算法的基本思想.

2.2 Apriori_ADO 算法设计

根据以上两条基本性质和基于一次性扫描数据库的特点, 设计了 Apriori_ADO 算法. 该算法详细步骤如下:

- (1) 打开数据库 DB, 令 $i = 1$, 设候选集 $C = \Phi$, 频繁集 $L = \Phi$
- (2) 若 $i > |DB|$, 则转(10)
- (3) 令 $t = R_i$ 并计算 $j = |t|$ // R_i 为第 i 条记录
- (4) 若 $j = 0$, 则转(9)
- (5) $C_j = \{x | x \subseteq t \wedge |x| = j\}$. 若 $x \in C_j \wedge x \in L$, 则 $C_j = C_j - x$
- (6) $C = C \cup C_j \wedge x.$ count $= x.$ count $+ 1$ // $x.$ count 为候选项 x 的支持度计数
- (7): 若 $x \in C \wedge x.$ count $= \min_sup$, 则 $L = L \cup \{y | y \subseteq x\} \wedge C = C - x$
- (8) $j = j - 1$, 转(4)
- (9) $i = i + 1$, 转(2)
- (10) 关闭 DB, 生成关联规则
- (11) 算法结束

Apriori_ADO 算法只需扫描数据库一次, 降低了传统 Apriori 算法频繁扫描数据库的开销. 同时, 根据性质 1 和性质 2, 利用传统 Apriori 算法的剪枝技术尽快删除非频繁项, 不会产生支持计数为 0 的候选项^[5], 减少了候选项的数量. 另外, Apriori_ADO 算法所花费的时间开销几乎与频繁项集的层次无关.

2.3 算法复杂性分析

2.3.1 时间复杂度

时间复杂度是评价一个算法性能的重要标准.

在对算法 Apriori_ADO 进行时间复杂度分析时主要参照 Apriori 算法时间复杂度分析方法. Apriori 算法时间复杂度主要体现在对事务项集的搜索中, 当自底向上搜索时其访问事务项集的复杂性为^[6] $f(k) = |DB| \sum_{i=1}^k |C_i| + |L_i| + \sum_{j=2}^{k-1} |L_{j-1}^3|$, 其时间复杂度 $F(k) = O(k^3)$.

然而在 Apriori_ADO 算法中, 访问事务项集的复杂性 $f'(k)$ 包括:

- (1) 产生第一轮数据 R_1 的 k 项候选集 $|R_1|$. $\sum_{i=1}^k |C_i|$ 和频繁集 $|R_1|$. $\sum_{j=1}^k |L_j|$. 此时只需要扫描第一项数据, 扫描次数 $M_1 = 1$;
- (2) 产生新一轮的各项候选集. 在扫描第二项数据 R_2 的同时, 还需扫描旧的所有 k 项候选集和频繁集, 扫描次数 $M_2 = 1 + |R_1| \sum_{i=1}^k (|C_i| + |L_i|)$;
- (3) 产生第三轮数据的各项候选集, 扫描次数 $M_3 = 1 + |R_2| \sum_{i=1}^k (|C_i| + |L_i|)$;
- (4) 产生最后的候选集, 并由 $\min_sup$ 得到所有频繁项集 L_k , 扫描次数 $M_{|DB|} = 1 + |R_j| \sum_{i=1}^k (|C_i| + |L_i|)$, 其中 $j = |DB| - 1$.

由以上分析可得

$$f'(k) = |DB| + |R_1| \sum_{i=1}^k (|C_i| + |L_i|) + |R_2| \sum_{i=1}^k (|C_i| + |L_i|) + \dots + |R_j| \sum_{i=1}^k (|C_i| + |L_i|) = |DB| + (|R_1| + |R_2| + \dots + |R_j|) \sum_{i=1}^k (|C_i| + |L_i|)$$

其中 $|R_1| + |R_2| + \dots + |R_j|$ 为 j 项数据记录扫描之和, 由上推理过程可知每项数据记录只扫描一次且 $j = |DB| - 1$, 可得 $|R_1| + |R_2| + \dots + |R_j| = |DB| - 1$.

因此 $f'(k) = |DB| + (|DB| - 1) \sum_{i=1}^k (|C_i| + |L_i|)$, 其时间复杂度为 $F'(k) = O(k)$.

显然 $F'(k) < F(k)$. 也就是说, Apriori_ADO 算法的时间复杂度小于 Apriori 算法的时间复杂度.

2.3.2 空间复杂度

空间复杂度可简单地认为是算法执行中需要的存储量的开销. Apriori 算法将整个数据库数据读到内存中, 所以每产生一层候选集用来存储数据库记录需要 $|DB|$ 个临时变量. 当产生第 1 层候选集和频繁集时, 需要 $M_1 = |DB| + 2 \sum_{i=1}^k (|C_i| + |L_i|)$ 个存储空间, 当产生第 n 层候选集和频繁集时需要 $M_n = |DB| + 2 \sum_{i=1}^k (|C_i| + |L_i|)$ 个存储空间. 所

以 Apriori 算法的空间复杂度为

$$\begin{aligned} S(k) &= \max\{M_1, M_2, M_3, \dots, M_n\} \\ &= |DB| + 2\text{Max}_{j=1}^n \sum_{i=1}^k (|C_i| + |L_i|) \\ &= |DB| + 2\sum_{i=1}^k (|C_i| + |L_i|) \end{aligned}$$

式中, C_i 为第 i 层候选集, L_i 为第 i 层频繁集. 而 Apriori_ADO 算法执行所需存储量的开销包括:

(1) 产生第一轮数据 R_1 的 k 项候选集 $|R_1|$, $\sum_{i=1}^k |C_i|$ 和频繁集 $|R_1|$, $\sum_{j=1}^k |L_j|$: 用临时变量 t 存储当前记录 R_1 , 用 $|R_1| [\sum_{i=1}^k (|C_i| + |L_i|)]$ 个存储变量存储 R_1 的候选集和频繁集, 需要 $|R_1| [\sum_{i=1}^k (|C_i| + |L_i|)]$ 个存储变量存储 R_1 的候选集和频繁集的支持度计数. 因此, 产生第一项数据需存储空间 $N_1 = 1 + 2|R_1| [\sum_{i=1}^k (|C_i| + |L_i|)]$;

(2) 产生第二轮的各项候选集: 用临时变量 t 存储当前记录 R_2 , 用 $|R_2| [\sum_{i=1}^k (|C_i| + |L_i|)]$ 个存储变量来存储第二项数据的候选集和频繁集, 用 $|R_2| [\sum_{i=1}^k (|C_i| + |L_i|)]$ 个存储变量来存储第一项数据的候选集和频繁集的支持度计数. 因此, 产生第二轮数据需存储空间 $N_2 = 1 + 2|R_2| [\sum_{i=1}^k (|C_i| + |L_i|)]$;

(3) 同理, 产生第三轮数据的各项候选集需存储空间 $N_3 = 1 + 2|R_3| [\sum_{i=1}^k (|C_i| + |L_i|)]$;

(4) 产生最后的候选集, 并由 $\min_sup$ 得到所有频繁项集 L_k 需存储空间 $N_{|DB|} = 1 + 2|R_{|DB|} [\sum_{i=1}^k (|C_i| + |L_i|)]$.

由上分析可得

$$\begin{aligned} S'(k) &= \max\{N_1, N_2, N_3, \dots, N_{|DB|}\} \\ &= 1 + 2\text{Max}_{j=1}^{|DB|} [|R_j| \sum_{i=1}^k (|C_i| + |L_i|)] \\ &= 1 + 2\text{Max}_{j=1}^{|DB|} |R_j| \cdot \sum_{i=1}^k (|C_i| + |L_i|) \\ &= 1 + 2\sum_{j=1}^k (|C_j| + |L_j|). \end{aligned}$$

式中, C_j 为第 j 条记录的候选集, L_j 为第 j 条记录的频繁集, R_j 用一个存储空间.

虽然两者产生的频繁项是相同的, 即 $\max(\sum_{i=1}^k |L_i|) = \max(\sum_{j=1}^k |L_j|)$, 但在实际

情况中, (1) Apriori 算法会产生许多支持度为 0 的候选项, 它们需要通过重复扫描数据来删除; (2) Apriori_ADO 算法不但不会产生支持度为 0 的候选项, 而且候选项的数目会随着数据库读取记录的增多而减少, 从而使得 $\max(\sum_{j=1}^k |C_j|)$ 远远小于 $\max(\sum_{i=1}^k |C_i|)$. 因此, $S'(k) < S(k)$.

综上所述, Apriori_ADO 算法与传统的 Apriori 算法相比, 既减少了数据库扫描次数和比较次数, 又降低了算法的时间复杂度和空间复杂度. 当数据库记录及其所含字段值较多时, 可以大幅减少候选项的数量.

3 简单实例分析

假设有一个 4 个记录的简单事务数据库 (表 1), 最小支持度计数 $\min_sup = 2$, 用 Apriori_ADO 算法对数据库的挖掘过程详见表 2. 表 3 对 Apriori 算法和 Apriori_ADO 算法在本例中执行情况进行了对比.

表 1 一个简单的事务数据库模型

TID	项
001	ABC
002	BCE
003	ABCD
004	BE

表 2 Apriori_ADO 算法运算过程

C_1 /计数	C_3 /计数	C_2 /计数	C_1 /计数	L_3	L_2	L_1
ABC	ABC/1	AB/1	A/1			
		AC/1	B/1			
		BC/1	C/1			
BCE	BCE/1	BC/2	E/1		BC	B
		BE/1				C
		CE/1				
ABCD	ABCD/1	ABC/2	AD/1	D/1	ABC	AB
		ABD/1	BD/1		AC	B
		ACD/1	CD/1		BC	C
		BCD/1				
BE		BE/2		ABC	AB	A
					AC	B
					BC	C
					BE	E

表 3 Apriori 算法和 Apriori _ ADO 算法在简单实例中的执行情况对比说明

项目 算法	支持度计数=1			支持度计数=2			支持度计数=3			支持度计数=4		
	比较 次数	候选项 数目	DB 扫描 次数	比较 次数	候选项 数目	DB 扫描 次数	比较 次数	候选项 数目	DB 扫描 次数	比较 次数	候选项 数目	DB 扫描 次数
Apriori	>100	>100	4	25	22	4	15	8	4	13	5	4
Apriori _ ADO	4	4	1	22	22	1	25	22	1	25	22	1

4 实验分析与应用

关联规则挖掘在超市数据库中应用非常广泛,决策者可以通过找到顾客购物之间的联系来决定超市的批货以及货物的摆放.我们用 C# 语言实现了传统 Apriori 算法和 Apriori _ ADO 算法,并通过处理部分超市数据来比较两者处理大量数据时的时间开销.同时,超市决策者利用该信息可以掌握商品的销售情况,优化贸易活动.

实验数据集 test. csv 是包含 10 800 条购物数据的 Excel 表,基本项为面包、牛奶、花生、鸡蛋、奶油,0 表示未购买,1 表示购买.部分数据见表 4.

表 4 部分实验数据

序号	面包	牛奶	花生	奶油	鸡蛋	序号	面包	牛奶	花生	奶油	鸡蛋
1	1	1	0	0	1	11	0	1	0	1	0
2	0	1	0	1	0	12	0	1	1	0	0
3	0	1	1	0	0	13	1	1	0	1	0
4	1	1	0	1	0	14	1	0	1	0	0
5	1	0	1	0	0	15	0	1	1	0	0
6	0	1	1	0	0	16	1	0	1	0	0
7	1	0	1	0	0	17	1	1	1	0	1
8	1	1	1	0	1	18	1	1	1	0	0
9	1	1	1	0	0	19	1	1	0	0	1
10	1	1	0	0	1	20	0	1	0	1	0

在程序运行中,对两种算法的运行时间进行了比较.实验结果可以看出,Apriori _ ADO 算法比传统的 Apriori 算法缩短了执行时间,提高了处理效率,并具有较强的稳定性.在最小可信度为 60% 时发现了一些强关联规则(表 5).

表 5 强关联规则挖掘

频繁集	关联规则	强关联规则
面包		
牛奶		
花生	面包→牛奶	面包→牛奶
面包 牛奶	面包→花生	面包→花生
面包 花生	牛奶→花生	
牛奶 花生		

表 5 说明该超市出售最多的是面包、花生和牛奶,而且大多数顾客会同时购买面包和牛奶、牛奶和花生或者面包和花生.这些规则是有趣的,它对于改善超市销售水平,提高资金周转率,加强超市管理都有一定的指导意义.

5 结束语

本文设计实现了 Apriori _ ADO 算法,并以超市消费知识挖掘为例对两种算法进行了比较:相对于传统的 Apriori 算法,Apriori _ ADO 算法更加稳定,由于其只需对数据库进行一次访问,所以不增加候选项,不需要进行剪枝筛选,因而具有较高的效率.当数据库的记录非常多而每条记录中包含字段值又较多时,该算法的优越性尤为明显.

参考文献:

[1] Han Jiawei, Kamber M. Data mining: concepts and techniques[M]. 北京:机械工业出版社, 2001.

[2] 陆丽娜,陈亚萍,魏恒义,等.挖掘关联规则中的 Apriori 算法的研究[J]. 小型微型计算机系统, 2000, 21(9): 940—943.

[3] Han J, Jian P, Yiwen Y. Mining frequent pattens without candidate generation [C]// Proceedings of the 2000 ACM SIGMOD International Conference Management of Data. Dallas: ACM, 2000: 1—12.

[4] 芦洁,刘志镜.挖掘关联规则中对 Apriori 算法的一个改进[J]. 微电子学与计算机, 2006, 23(2): 10—12.

[5] 冯浩,陶宏才.快速挖掘最大频繁项集[J]. 微电子学与计算机, 2007, 24(5): 123—126.

[6] 袁鼎荣,严小卫. Apriori 算法的复杂性研究[J]. 广西科学, 2005, 12(2): 115—122.

作者简介:

刘培奇 男, (1959—), 副教授, 博士. 研究方向为机器学习、模式识别和网络安全.

卢 麟 男, (1985—), 硕士研究生. 研究方向为知识发现方法和网络安全.