基于二进制编码的 Apriori 增量更新算法研究

罗章铭,唐 杰,黄逸奇,张 锦*
(湖南师范大学 信息科学与工程学院,湖南 长沙 410006)

摘 要:针对经典 Apriori 算法在迭代过程中频繁扫描数据库,且动态数据更新后需要重新处理数据的不足,提出一种基于二进制编码的增量更新改进 CBEF-Apriori 算法。该算法的核心思想是将添加增量后的项集、事务转换成二进制编码,从而将计算项集支持度转化为项集与事务数据库的二进制编码位运算过程。改进算法筛选原数据库生成的频繁项集与增量数据库新生成的候选项集,有效减少了候选项集的规模,提高算法效率的同时更符合现实需要。实验结果表明,相比于经典 Apriori 算法和 CBE-Apriori 算法,改进算法在挖掘出正确频繁项集的数量不降低的情况下,明显提升了计算效率,在小数据规模下相比经典 Apriori 算法最高提升 3.6 倍,相比 CBE-Apriori 算法最高提升 1.4 倍。在较大数据规模下相比经典 Apriori 算法最高提升 10.41 倍,相比 CBE-Apriori 算法最高提升 11.53 倍。

关键词:数据挖掘;Apriori 算法;关联规则;二进制;增量更新
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2022)01-0047-07
doi:10.3969/j.issn.1673-629X.2022.01.009

Research on Apriori Incremental Update Improved Algorithm Based on Binary Code

LUO Zhang-ming, TANG Jie, HUANG Yi-qi, ZHANG Jin*
(School of Information Science and Engineering, Hunan Normal University, Changsha 410006, China)

Abstract: Aiming at the defect that the classic Apriori algorithm frequently scans the database during the iterative process, and the data needs to be reprocessed after the dynamic data is updated, an improved CBEF-Apriori algorithm based on the incremental update of binary coding is proposed. The core idea of the improved algorithm is to convert the added itemsets and transactions into binary codes, so as to convert the calculation itemsets support into the binary code bit operation process of the itemsets and the transaction database. The improved algorithm filters the frequent itemsets generated by the original database and the candidate item sets newly generated by the incremental database, which effectively reduces the size of the candidate item sets, improves the efficiency of the algorithm, and meets actual needs. The experiment shows that compared with the classic Apriori algorithm and the CBE-Apriori algorithm, the improved algorithm mines the number of correct frequent itemsets without reducing, and its computational efficiency is significantly improved, which is 3.6 times higher than the classic Apriori algorithm at a small data scale. Under a larger data scale, it is up to 10.41 times higher than the classic Apriori algorithm, and up to 11.53 times higher than the CBE-Apriori algorithm.

Key words: data mining; Apriori algorithm; association rules; binary; incremental update

0 引 言

数据挖掘是从有噪音的、不完整的、不清晰的、随机的数据中提取出有效的、潜在的、有用的知识[1],在人工智能、医学、电子商务、工业等各领域得到广泛应用[2]。关联规则是目前最为常用,应用领域最为广泛的数据挖掘技术。作为挖掘重要关联知识与规则的分析手段,关联规则用于发现存在于数据库中的海量数据之间的关联性,而这些关联性或许能满足人们对其中一些属性同时出现在一个事务上的规律和方式的探寻,以提升经济、管理等方面的效益。

Agrawal 等[3]在 1994 年通过候选项集的连接和剪枝得到频繁项集,首次提出了关联规则中最经典也是使用最为频繁的 Apriori 算法,2000 年 Han 等[4]在此基础上提出了频繁模式增长算法,即 FP-growth 算法。

收稿日期:2021-03-04 **修回日期**:2021-07-06
基金项目:军委装发预研项目(31511010105);国防科工局国防基础科研计划项目(WDZC20205500119);湖南省交通运输厅科技进步与创新计划项目(201927);湖南省研究生培养创新实践基地项目(湘教通〔2019〕248 号);湖南省科技厅创新引领计划(2020GK2009)
作者简介:罗章铭(1995-),男,硕士生,CCF 会员(A4211G),研究方向为数据挖掘;通讯作者:张 锦(1979-),男,博士,教授,CCF 会员(06153S),研究方向为智能软件工程、人工智能等。

近三十年来关于关联规则的改进算法层出不穷,主要工作集中在解决关联规则算法多次迭代中频繁扫描数据库这一问题。程远[5]对剪枝连接函数 Apriori_Gen 进行优化;刘智[6]提出 V-Apriori 算法,即单次扫描数据库后,事务数据库被转换为布尔矩阵,将事务数据库的扫描转化为向量运算等,通过类似的优化手段改进 Apriori 算法。改进后的关联规则可以用于医疗领域中的病症规律研究[7]、消费领域的经济发展研究[8]、教育信息化研究[9]等。然而实际应用时,数据库经常处于不断变化状态,数据经常性增加或更新,因此关联规则发现的规则具有时效性,仅反映数据库某一时刻的状态。为了使发现的规则稳定可靠,应在相当长的一段时间内收集大量数据。如果要更新规则,就不得不重新使用算法对数据库进行挖掘,这样做除了效率低下,还浪费了大量已经挖掘出的信息资源。

在此背景下,对于动态数据的增量式关联规则挖掘成为新的研究方向,力图发现快速地更新已有规则的算法。在这一研究方向上,Cheung 等[10]首次提出了增量更新算法(incremental updating,IU),通过对增量数据库与原数据库历史结果的关系进行处理,提出了在数据增加这一情况下的关联规则增量算法(stands for fast update,FUP)。该算法对历史频繁项集以及新产生的候选项集进行筛选来得到新的频繁项集。该算法计算支持度的方式仍然沿用 Apriori 算法的思想,需要多次扫描数据库。陈劲松等[11]对小增量下的情况进行了研究。李宝东等[12]和黄德才等[13]则是着重对数据集的扫描次数减少问题进行优化,以此提高算法的效率。Hong 等[14]将增量更新算法思想应用到 FP-growth 算法中。王诚等[15]与朱晓峰等[16]则对算法进行了并行化改进,以此来提高数据处理效率。

该文从项集支持度的计算过程入手,通过二进制编码位运算在项集支持度计算方面的优势,减少增量更新算法扫描数据库的次数和时间资源消耗,通过模拟实验对比分析了不同算法的性能,验证了所提出算法的有效性。

1 相关知识

1.1 Apriori 算法

关联规则有多种算法,但大部分都是基于经典 Apriori 的改进算法。根据支持度阈值设置的不同,挖掘结果也会有所不同。首先需要说明以下概念,包含某项集的事务数在整个数据库中的比例称为该项集的支持度,包含的意思指的就是事务的子集是该项集。最小支持度指的就是人为根据经验设置的最小支持度阈值。从关联规则诞生以来,最基本也是使用率最高的算法就是 Apriori 算法。

其主要有两条基本性质:

性质 1:频繁项目集的所有非空子集也是频繁项目集。

性质 2:非频繁项集的超集一定是非频繁项集。

该算法的特点是在每一次的迭代过程中都对项集进行连接和剪枝操作,以 k 项集来产生($k+1$)项集,这种迭代方法被称之为“逐层搜索”。因为 Apriori 算法对数据集进行扫描的次数与最长频繁模式的长度有关,所以当数据集很大且频繁模式很长时,该算法会大大降低执行效率。Apriori 的优点在于算法思想较为简单,但为了生成频繁项集,在此期间反复扫描庞大的数据库将会消耗大量时间;另一个原因是在 Apriori_Gen 生成候选项目集的过程中,随着频繁项集的增加,候选项集变得更多。这无疑将增加算法的计算量。例如,频繁 1-项集若为 5 项,则将有 10 个候选项集;频繁 1-项集若为 10 000 项,则候选项集将达到 $c^2_{10\,000}$,此时计算量将会变得非常庞大。以此类推,当频繁项集变得更多时,计算量将无止尽地增长。

1.2 CBE-Apriori 算法

Apriori 算法的痛点在于每一次迭代中生成频繁 $k-$ 项集时都需要对数据库进行一次扫描,用以对项集支持度进行计数,直接影响了算法的运行效率。在进行连接操作时,还需判断项集得前($k-1$)项是否相等。基于上述 Apriori 算法存在的问题,近年来研究者从许多方面对此算法进行改进,包括矩阵化处理,引入并行机制等等。而胡世昌[17]、谷鹏[18]分别提出了 BE-Apriori(binary encode,BE)和 CBE-Apriori(compressed binary encode,CBE)算法,通过对项集和事务二进制编码进行位运算,将扫描数据库的过程转化为内存中的位运算。

与经典 Apriori 需要重复扫描数据库来计算项集的支持度不同,现假设某数据库有频繁 1-项集 a,b,c,事物数据库有三项数据(a,b,c),(a,c),(a,b)。首先根据频繁 1-项集的个数对频繁 1-项集进行二进制编码,a,b,c 分别被编码位 100,010,001,即 $2^2,2^1,2^0$。编码后的二进制数可能小于 n,这是由于二进制数前面部分位置若为 0 代表的项均不存在。然后根据频繁 1-项集的编码结果对事务数据库进行编码。可知数据库事务的编码对应 111,101,110。对于任意项集例如(a,b)的支持度计算被转化为了二进制的位与运算。若项集的二进制编码与事务数据库的二进制编码位与的结果等于项集二进制编码其本身,则证明该项集为该条事务的子集,即支持度计数理应加 1。即若 $a\&b=a$,说明 a 是 b 的子集,若 $a\&b!=a$,则说明 a 不是 b 的子集。如表 1 所示,上述例子中 2-项集(a,b)的支持度应为 2。

表 1　CBE-Apriori 算法的支持度计算过程

操作	位运算	结果
(a,b)&(a,b,c)	110&111=110	(a,b)为(a,b,c)的子集,支持度计数+1
(a,b)&(a,c)	110&101=100	(a,b)不为(a,c)的子集,支持度计数不变
(a,b)&(a,b)	110&110=110	(a,b)为(a,b)的子集,支持度计数+1

CBE-Apriori 算法的主要流程如下:

(1)扫描数据库获取频繁 1-项集,对其进行二进制编码。

(2)根据 1-项集的编码结果对事务数据库进行编码。

(3)频繁 1-项集自连接产生候选 2-项集。

(4)通过位运算计算支持度,得到频繁 2-项集。

(5)若每一项频繁 2-项集相异或,结果仍是频繁项集,则求得这两项频繁 2-项集的并集作为候选 3-项集。

(6)通过位运算计算支持度,得到频繁 3-项集。

重复步骤(5)~(6),直到没有新的频繁项集产生。

BE-Apriori、CBE-Apriori 算法的优势在于只需扫描两次数据库,即可得到全部频繁项集。原 Apriori 算法连接和剪枝操作的时间复杂度为 $O(k^2\log k)$。而改进后算法通过编码后两个频繁项集异或结果是否为频繁 2-项集,就可以完成连接和剪枝操作,时间复杂度为 $O(1)$,常数时间即可完成。正是因为二进制的基本运算代替了集合之间的运算,因此可以有效提高算法执行效率。

2　CBEF-Apriori 算法

基于已有研究,该文提出了基于二进制编码的 Apriori 增量更新算法 CBEF-Apriori(compressed binary encode fast update Apriori)。该算法吸收了二进制编码算法的优点,即只需要通过两次对数据库的遍历,第一次遍历的目的是为了获取频繁 1-项集,第二次遍历则是根据频繁 1-项集的结果对数据库进行二进制编码,将结果存入内存中,在内存中计算代替之后的"逐层搜索"过程,能有效避免 I/O 效率低下的时间消耗。结合算法性质,其主要优势体现在:无需重新对新数据和旧数据一起进行挖掘,根据增量数据和上一次挖掘得到的结果,可以生成新的频繁项集;在 k 次迭代过程中,候选项集的产生大大减少。

2.1　算法的定义和性质

假设 DB 为原数据库, L 为原数据库中频繁项集的集合, s 为人为设定的最小支持度阈值, D 为原数据库中的事务数。假定对于每个 Item $\in L$,其支持计数为 Item.support(包含 Item 的原数据库中的事务支持度),新事务数据增量 db 添加到原始数据库 DB 中, d 是 db 中的事务数。对于相同的最小支持度 s,如果 DB∪db 中 Item 的支持度不小于 s,即 Item.support $\geqslant s*(D+d)$,则项集 Item 在更新的数据库 DB∪db 中仍频繁。下文中 Item.$\text{support}_{\text{DUd}}$ 代表项集 Item 在 DB∪db 中的支持度,即全局支持度;Item.$\text{support}_{\text{D}}$ 代表项集 Item 在原数据库 DB 中的支持度;Item.$\text{Support}_{\text{d}}$ 代表项集 Item 在增量数据库 db 中的支持度。对于第一次迭代主要有两条重要性质:

性质 3:当且仅当 Item.$\text{Support}_{\text{DUd}} < s*(D+d)$ 时,原始的频繁 1-项集 L_1 中的项 Item 在更新后的数据库 DB∪db 中成为失败者(即不是新的频繁 1-项集)。

性质 4:仅当 Item.$\text{Support}_{\text{d}} > s*d$ 时,不是原频繁 1-项集 L_1 中的项 Item 才可能成为更新数据库 DB∪db 的赢家(即可能是新的频繁 1-项集)。

下列性质则可用于得出更新数据库后的频繁 k-项集(其中 $k>1$)。

性质 5:若$\{\text{Item}_1,\cdots,\text{Item}_{k-1}\}$在第($k-1$)次迭代中是失败者,即该项在 L_{k-1} 中却不在 L'_{k-1} 中时,包含此项的 k-项集不可能成为第 k 次迭代的赢家,即不可能是新的频繁 k-项集。

性质 6:对于在原频繁 k-项集 L_k 中的 k-项集$\{\text{Item}_1,\cdots,\text{Item}_k\}$,当且仅当$\{\text{Item}_1,\cdots,\text{Item}_k\}$.$\text{support}_{\text{DUd}} < s*(D+d)$时,在更新后的数据库 DB∪db 中成为失败者。即不可能是新的频繁 k-项集。

性质 7:对于不在原频繁 k-项集 L_k 中的 k-项集$\{\text{Item}_1,\cdots,\text{Item}_k\}$,只有在$\{\text{Item}_1,\cdots,\text{Item}_k\}$.$\text{support}_{\text{d}} \geqslant s*d$时,在更新后的数据库 DB∪db 才有可能成为赢家,即可能是新的频繁 k-项集。

2.2　算法流程

基于性质 3、性质 4,第一次迭代的流程见图 1。在更新的数据库 DB∪db 中找到新的频繁 1-项集 L'_1 的过程概述如下:

(1)扫描增量数据库 db,更新原 L_1 项集中的所有项的支持度 Item.$\text{support}_{\text{DUd}}$,找到 L_1 中的所有失败者。若 Item.$\text{support}_{\text{DUd}} < s*(D+d)$,将其删除,剩余的项则考验成功,加入新的频繁 1-项集 L'_1。

(2)在同一扫描中,创建一个集合 C_1,每个 Item ∈ db 但不在原频繁 1-项集 L_1 中的大小为 1 的项集被加入 C_1。这些项成为候选 1-项集,它们在 db 中的支持度可以在扫描中统计得到。更重要的是,根据性质 2,如果 Item ∈ C_1 且 Item. $\text{support}_d < s * d$,则 Item 在 DB ∪db 中永远不可能频繁。因此,删除 C_1 中所有在 db 数据库支持度计数小于 $s * d$ 的项。这样在生成新的频繁 1-项集的过程中,减少了大量不可能在更新数据库后成为频繁 1-项集的项。

(3)然后在 DB∪db 上进行扫描以更新每个 Item ∈ C_1 的支持计数。通过统计检查其支持计数,可以找到 C_1 中的新频繁项并将其加入到 L_1' 中,生成新的频繁 1-项集 L_1'。

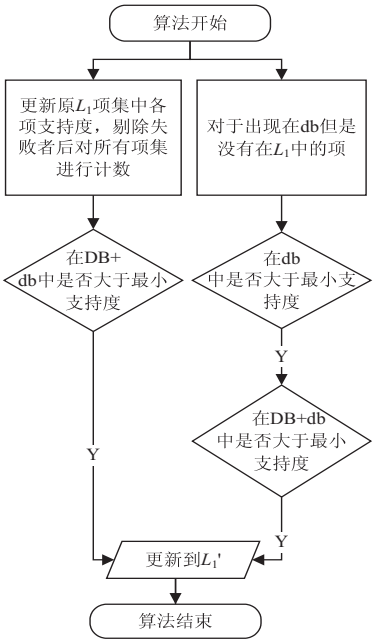

图 1 CBEF-Apriori 算法新 1-项集生成流程

第一次迭代后,对所有的频繁 1-项集进行二进制编码,并根据 1-项集的编码结果对事务数据库进行编码。基于性质 5、性质 6、性质 7,在更新的数据库 DB ∪db 中找到新的频繁 2-项集 L_2' 的过程见图 2。

(1)与第一次迭代类似,L_2 中的失败者将在对 db 的扫描中被筛除掉。筛除过程分为两个步骤。首先,根据性质 3 所述,可以将 L_2 中的某些失败者筛除掉。失败者的集合 $L_1 - L_1'$ 已在第一次迭代中确定。因此,具有子集 Y 使得 $Y \in L_1 - L_1'$ 的任何集合 Item ∈ L_2 不可能频繁,不需要对 db 进行扫描就可以从 L_2 中滤除。

(2)与第一次迭代类似,此次迭代的第二部分是找到新频繁 2-项集。通过自连接创建 C_2 时将排除原 L_2 中的集合。通过二进制运算统计其支持计数来修剪 C_2 中的项目集。基于性质 4,所有被筛除的集合在 DB ∪db 中不可能频繁。对于所有 Item ∈ C_2,如果 Item. $\text{support}_d < s * d$,则从 C_2 中删除。

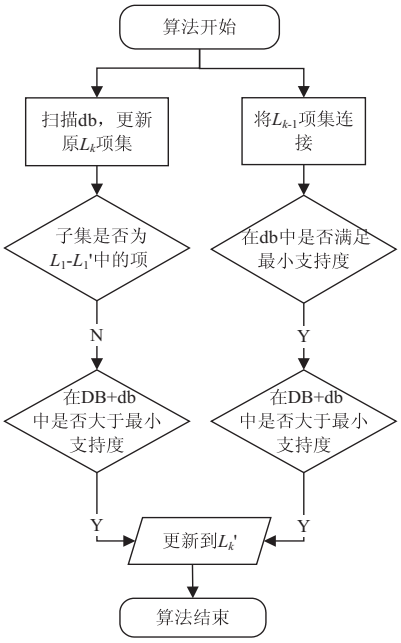

图 2 CBEF-Apriori 算法新 k-项集生成流程

(3)最后一步是二进制运算以更新 C_2 中所有项集的支持数。其支持计数 Item. $\text{support}_{D \cup d} \geqslant s * (D + d)$ 的集合 Item ∈ C_2 被标识为新的频繁 2-项集。集合 L_2' 包含从上面的 L_2 和 C_2 识别的所有频繁项集。将相同的算法应用于以后的迭代,直到没有找到新的频繁项集。第 k 次迭代的流程见图 2。

在 k 次迭代中,无需再对数据库进行扫描来连接和剪枝候选项集,而是通过二进制运算来判断项集是否为事务的子集。其伪代码如下:

输入:增量数据 db,原数据库 DB,支持度 minsup,原频繁项集 $L_1, L_2, \cdots, L_k$

输出:增量更新后的频繁项集 L_k'

```
(1)forall transactions t ∈ db do begin
    forall item ∈ t do begin
        db_item. count ++
    end
end
(2)forall t ∈ DB and db do begin
    forall item ∈ t do begin
        DB_and_db_item. count ++
    end
end
(3) C1' = {item ∈ db and ∉ DB | db_item. count ≥ minsup}//对候选 1-项集进行第一次考验
(4) C1' = {item ∈ C1 | DB_and_db_item. count ≥ minsup}//对候选 1-项集进行第二次考验
(5) L1' = C1' ∪ {item ∈ L1 | DB _ and _ db _ item. count ≥ minsup}//将原频繁 1-项集通过全局考验以及候选 1-项集通过两次考验加入新的频繁 1-项集
(6)for( k =2; Lk' ≠∅; k ++) do begin
    if k ==2 then
```

C_k = {item ∈ (apriori-gen1(L_{k-1}) - L_2) | db_item. count ≥ minsup}//对候选-2 项集进行第一次考验

$C_k^{'}$ = {item ∈ C_k | DB_and_db_item. count ≥ minsup}//对候选 2-项集进行第二次考验

$L_2^{'}$ = $C_k^{'}$ ∪ {item ∈ L_2 | DB_and_db_item. count ≥ minsup}//将原频繁 2-项集通过全局考验以及候选 2-项集通过两次考验加入新的频繁 2-项集

else

C_k = {item ∈ (apriori-gen2(L_{k-1}) - L_{k-1}) | db_item. count ≥ minsup}//对候选- k 项集进行第一次考验

$C_k^{'}$ = {item ∈ C_k | DB_and_db_item. count ≥ minsup}//对候选 k -项集进行第二次考验

$L_k^{'}$ = $C_k^{'}$ ∪ {item ∈ L_k | DB_and_db_item. count ≥ minsup}//将原频繁 k -项集通过全局考验以及候选 k -项集通过两次考验加入新的频繁 k -项集

end

(7) answer = ∪ L_k

(8) apriori-gen-1//获取候选 2-项集

INSERT INTO C_k

SELECT p. $item_1$, q. $item_1$

from L_{k-1}p, L_{k-1}q

(9) apriori-gen-2//获取候选 k -项集(k >2)

INSERT INTO C_k

SELECT p. $item_1$, p. $item_2$, . . . , p. $item_{k-1}$, q. $item_{k-1}$

from L_{k-1}p, L_{k-1}q

where p^q ∈ L_2

3　性能评估

3.1　实验环境

实验环境如下:处理器为 3.0 GHz AMD Ryzen5 4600H,内存为 16 GB DDR4,操作系统为 Windows 10,选用 Python 3.6 作为开发语言。模拟实验对 Apriori 算法、CBE-Apriori 算法以及 CBEF-Apriori 算法进行了对比分析。

3.2　数据集介绍

实验采用的数据集为数据挖掘实验中常用的数据集,即 mushroom 数据集[19]和 T10I4D100K[20],数据集信息如表 2 所示。mushroom 数据集为稠密型数据,总共包含 120 种不同的属性,任意一条事务的长度均为 23,样本总数则为 8 124,总共包含 120 个项。T10I4D100K 为模拟稀疏数据集,其中 T 表示事务平均长度, I 表示频繁项集的平均长度, D 表示数据集中的事务总数,意味着该数据集事务平均长度为 10,频繁项集的平均长度为 4,事务总数为 100 000 条。

表 2　数据集简介

数据集	类型	样本个数	空间维数	类别分布	属性总数
Mushroom	稠密	8 124	22	4208,3916	126
T10I4D100K	稀疏	100 000	10	/	870

对于文中所考虑的增量更新问题,最为直接的办法就是将 Apriori 算法与 CBE 算法重新运行一遍,与文中提出的增量更新算法 CBEF 进行性能比较。增量从原数据库中随机提取。

下文的实验结果以 5 次取值的平均值的方式给出,减少实验的偶然性。

3.3　实验分析

(1) mushroom 数据集。为验证 CBEF 算法结果的有效性,实验中分别比较了三种算法在不同支持度约束下生成的最终频繁项集数目,如表 3 所示。从表 3 中可以看出,CBEF 算法在结果上与其他两种算法无异。

表 3　不同增量与支持度下的频繁选项集个数

增量规模	最小支持度	Apriori	CBE	CBEF-
1 125	0.5	153	153	153
1 125	0.4	565	565	565
1 125	0.3	2 735	2 735	2 735
1 125	0.25	5 545	5 545	5 545
2 125	0.5	153	153	153
2 125	0.4	565	565	565
2 125	0.3	2 735	2 735	2 735
2 125	0.25	5 545	5 545	5 545

图 3 和图 4 分别给出了在增量为 1 125 以及 2 125 时,三个算法各自的运行时间以及候选项集个数。CBEF 算法生成的候选项集数均小于原 Apriori 算法以及 CBE 算法生成的候选项集数,算法运行时间也明显少于其他两种算法。在支持度接近 0.5 时,三个算法生成的候选项集个数十分接近,其主要原因是在 0.5 支持度下,筛除掉的必不可能成为频繁项集的数目变得越来越少,因此候选项集的个数趋向于相等,耗费时

间的优势变小。总体上,其运行效率与 Apriori 相比提升达到 2.2 至 3.6 倍,与 CBE 算法相比提升也有 1.1 至 1.4 倍。以上研究结果表明,在较小数据规模数据集上 CBEF 算法的时间效率较 Apriori 算法以及 CBE 算法是有明显提升的。

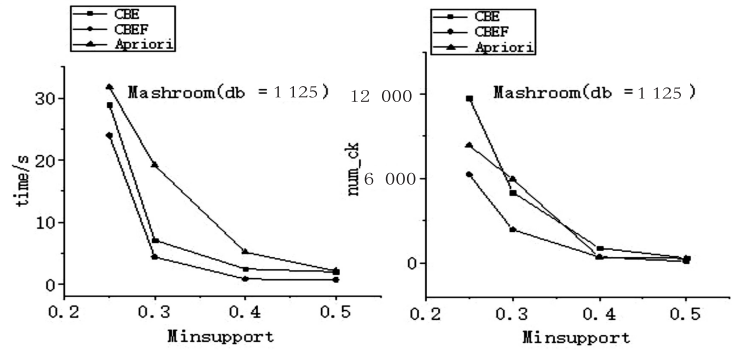

图 3 1 125 增量下算法时间消耗与候选项集生成数对比

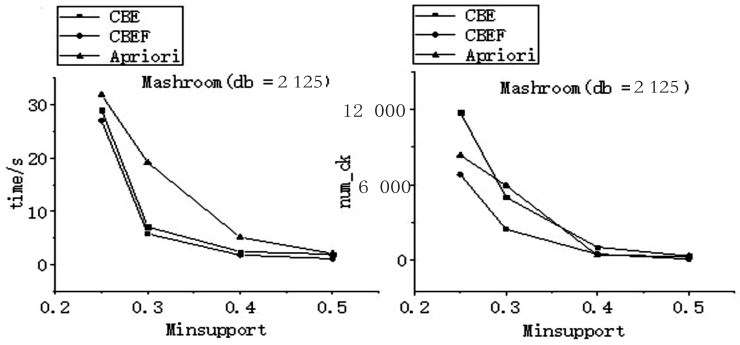

图 4 2 125 增量下算法时间消耗与候选项集生成数对比

(2) T10I4D100K 数据集。为了探究该算法在较大数据集上的运行情况,使用 IBM 合成数据产生器生成的 T10I4D100K 数据集进行实验分析。并分别提取出 10%、40%、60%、80% 作为增量数据条件下,模拟 CBEF 算法数据增量更新过程所需时间。

在表 4 中可以看出,CBE 算法在较大数据量的数据集中已经不再适用。在大量数据集中使用 CBE 算法,会产生大量的候选项集以及事务的编码数据,该过程产生的开销已经超过了在每一次迭代中重新扫描数据库计算最小支持度的开销。

表 4 不同增量与支持度下的运行时间 s

最小支持度	Apriori	CBE	10% 增量下 CBEF	40% 增量下 CBEF	60% 增量下 CBEF	80% 增量下 CBEF
0.06	26.99	23.82	6.83	14.6	20.43	25.62
0.04	28.86	25.91	10.25	13.88	22.78	26.87
0.02	174.84	192.2	15.32	25.69	48.55	42.99
0.01	932.23	1 084.83	132.75	513.68	824.56	1 056
0.007 5	1 043.8	1 569.67	205.18	632.49	1 218.69	1 880.02

随着支持度阈值的增大,各算法的运行时间大大减少。原因是符合条件的候选项集数量在阈值增大后减少,在逐层搜索的迭代过程中,产生的频繁项集个数也大大减少,在某些参数下没有迭代过程,只产生频繁 1-项集。

10% 增量下与 40% 增量下的 CBEF 算法在运行效率上都有着显著的提升,在 10% 增量下的提升尤为明显。在最小支持度阈值设置为 0.02,0.01,0.007 5 时,算法相比 Apriori 算法分别有 10.41 倍,6.02 倍,4.08 倍提升,相比 CBE 算法分别有 11.54 倍,7.17 倍,6.65 倍提升。

通过分析,在 10% 增量下提升明显的原因是,10% 的增量对于原数据 100 000 条来说是较小的数据量,对关联规则更新的影响程度也较小,例如在 0.02 的支持度参数下,频繁 1-项集只增加 3 项,且只有 1 次迭代过程,即只有频繁 1-项集的产生。因此产生的候选项集个数相比于重新计算过程大大减少,这正是增量更新算法最大的优势所在,即若增量数据对关联规则产生的影响较小,此算法可以节约大量时间快速给出新的结果。而在 60% 增量下 CBEF 算法的优势

已经不再明显,虽然在 0.01 以上支持度上仍然优于其他两种算法,但是在 0.007 5 支持度阈值以及 80% 增量的条件下,由于候选项集的个数显著增多,候选项集的编码开销基本已经超过了 Apriori 算法。

在数据集较大的情况下,由以上分析结果可以说明:(1)在数据增量较小时,CBEF 算法相比于其他两种算法有明显的提升,这是因为较小的增量对于规则的影响不大,与增量前的数据结果出入较小,因此产生的开销也小。CBEF 算法在增量较小的情况下的规则更新效果优势明显。(2)在增量到达一定规模时,例如上文的 80%,此时运用增量更新的算法开销已经接近或者大于使用 Apriori 或者 CBE 算法,可以考虑直接使用其他算法重新计算。

4　结束语

针对 Apriori 算法性能提升需求,通过二进制位运算过程计算项集支持度,结合增量更新思想,该文提出了一种基于二级制编码的 Apriori 增量更新算法 CBEF-Apriori。通过对算法的分析以及实验证明,相比经典 Apriori 算法以及普通的二进制编码算法 CBE-Apriori,该算法不仅可以结合历史的规则生成结果来更快地给出新的频繁项集,并且在计算支持度以及筛除候选项集上都有着明显的优势,解决了 Apriori 算法以及 CBE 算法各自的问题。后续可以考虑在最小支持度阈值更新的情况下,如何更快生成新的频繁项集的情况。并且在较大规模较大增量的情况下通过多进程并行计算等提高算法效率。

参考文献:

[1] 范　明,孟小峰. 数据挖掘:概念与技术[M]. 北京:机械工业出版社,2001.

[2] 李　青. 基于数据挖掘的工业互联网入侵检测方法研究[D]. 成都:电子科技大学,2020.

[3] AGRAWAL R,IMIELINSKI T,SWAMI A. A mining association rules between sets of items in large databases[J]. SIGMOD Rec. ,1993,22(2):207-216.

[4] HAN Jiawei,PEI Jian,YIN Yiwen. Mining frequent patterns without candidate generation[J]. Data Mining and Knowledge Discovery,2004,8(1):53-87.

[5] 程　远. 关联规则挖掘在疾病数据处理中的应用研究[D]. 重庆:重庆医科大学,2010.

[6] 刘　智. 关联规则挖掘方法及其在冠心病中医诊疗中的应用研究[D]. 大连:大连海事大学,2012.

[7] 李雨洁,郑锐龙,杨旭明. 基于数据挖掘技术的冠心病诊断预测模型[J]. 医学信息,2020,33(24):14-17.

[8] 陈　星. 基于数据挖掘的地区经济关联发展研究[J]. 科技经济导刊,2020,28(35):239-240.

[9] 陈维洁,岳　明,田金涛. 大数据视野下高职院校教育信息化建设研究[J]. 科技经济导刊,2021,29(1):125-127.

[10] CHEUNG D,HAN J,NG V,et al. Maintenance of discovered association rules in large databases:an incremental updating technique[C]//International conference on data engineering. New Orleans:IEEE,1996:106-114.

[11] 陈劲松,施小英. 一种关联规则增量更新算法[J]. 计算机工程,2002,28(7):106-107.

[12] 李宝东,宋瀚涛. 关联规则增量更新算法研究[J]. 计算机工程与应用,2002,38(23):6-8.

[13] 黄德才,张良燕,龚卫华,等. 一种改进的关联规则增量式更新算法[J]. 计算机工程,2008,34(10):38-39.

[14] HONG T P,LIN J W,WU Y L. A fast updated frequent pattern tree[C]//IEEE international conference on systems,man and cybernetics. Taipei:IEEE,2007:2167-2172.

[15] 王　诚,赵申屹. 一种改进的并行关联规则增量更新算法研究[J]. 计算机技术与发展,2018,28(7):48-52.

[16] 朱晓峰,李玲娟,徐小龙,等. 基于 MapReduce 的关联规则增量更新算法[J]. 计算机技术与发展,2012,22(4):115-118.

[17] 胡世昌,李劲华,王常颖. 基于二进制编码的 Apriori 改进算法[J]. 计算机应用研究,2020,37(2):398-400.

[18] 谷　鹏,肖建于,徐成振. Apriori 算法的压缩二进制编码改进[J]. 宜宾学院学报,2020,20(6):54-58.

[19] 杨　志. 基于粒子群的粗糙聚类算法分析与研究[D]. 长沙:长沙理工大学,2014.

[20] GOPAGONI P K,MOHAN R S K. Distributed elephant herding optimization for grid-based privacy association rule mining[J]. Data Technologies and Applications,2020,54(3):365-382.