A Binary Encoding-Based Improved Apriori Algorithm Postprint

Authors: Hu Shichang, Li Jinhua, Wang Changying

Date: 2018-12-13T00:00:00+00:00

Abstract

The Apriori algorithm necessitates multiple database scans during frequent itemset mining, resulting in inefficiency due to frequent I/O operations. To enhance algorithmic execution efficiency, the BE-Apriori (binay encoded Apriori) algorithm fully exploits the advantages of binary numbers over various programming language data structures in terms of memory consumption and computational speed. It encodes transaction records in binary format before loading them into memory, subsequently employing equivalent binary number operations to replace set operations. The algorithm' s performance is analyzed, and experimental validation is conducted on the BE-apriori algorithm using the poisonous mushroom dataset from the UCI repository. The results demonstrate that BE-Apriori can correctly mine frequent itemsets and achieves superior performance compared to the Apriori algorithm.

Full Text

Preamble

Vol. 37 No. 2
Application Research of Computers
ChinaXiv Partner Journal

An Improved Apriori Algorithm Based on Binary Encoding

Hu Shichang, Li Jinhua†, Wang Changying
(School of Data Science & Software Engineering, Qingdao University, Qingdao, Shandong 266071, China)

Abstract: The Apriori algorithm requires multiple database scans when mining frequent itemsets, resulting in inefficiency due to frequent I/O operations. To improve its execution efficiency, the BE-Apriori (Binary Encoded Apriori)

algorithm leverages the advantages of binary numbers over various data structures in programming languages in terms of memory usage and computational speed. It loads transaction records into memory after binary encoding, then replaces set operations with equivalent binary number operations. The algorithm' s performance is analyzed, and experimental validation is conducted using the mushroom dataset from the UCI repository. Results demonstrate that BE-Apriori can correctly mine frequent itemsets and achieves better performance compared to the original Apriori algorithm.

Keywords: frequent itemsets; set operations; binary; Apriori algorithm
Classification: TP301.6
DOI: 10.19734/j.issn.1001-3695.2018.07.0519

0 Introduction

With the rapid development of social economy and the Internet, information resources have completely broken through previous geographical and temporal constraints, spreading and evolving rapidly across networks. Storage units have grown from gigabytes to petabytes and even exabytes. However, as information volume increases, so does the amount of invalid information, making it increasingly difficult to mine specific information from massive datasets. Consequently, technologies for extracting valuable information from large volumes of data have become increasingly important, giving rise to data mining techniques and algorithms. Association rule mining represents a crucial research direction in this field with wide-ranging applications.

The Apriori algorithm, proposed by American scholar Agrawal in 1993, is the most classical algorithm for association rule mining. It performs well across various domains, effectively analyzing correlations among massive transactions and providing valuable support for decision-making. The Apriori algorithm consists of two main steps: (1) obtaining frequent itemsets from transaction records, and (2) deriving association rules from these frequent itemsets. The first step —acquiring all frequent itemsets—plays a decisive role in the algorithm' s performance. Subsequent Apriori-based improvements have primarily focused on enhancing this phase. One matrix-based Apriori improvement converts transaction records into matrix form to reduce database traversal frequency, then obtains frequent itemsets through matrix operations. However, matrix operations are time-consuming. Another approach uses vector matrix optimization to reorder frequent itemset items by their individual frequency in ascending order, reducing candidate generation and improving efficiency. This method suffers from unstable performance, as sorting reduces candidate generation but cannot filter out all impossible candidates. A probabilistic Apriori improvement estimates the probability of itemset co-occurrence, but requires additional time to set parameters a and b and risks missing frequent itemsets. A cross-linked list-based Apriori algorithm maps transaction records to a cross-linked

structure, effectively organizing transaction sequences to reduce database and record scanning frequency. An efficient association rule mining algorithm based on predictive screening first samples to estimate frequent itemsets, then calculates them from original data, introducing damping and compensation factors to correct prediction errors, thereby improving efficiency at the cost of certain misjudgment and omission rates. While these algorithms propose improvements to Apriori, their results remain suboptimal. To further enhance execution efficiency, we propose the BE-Apriori algorithm based on binary encoding.

---

1 Apriori Algorithm

The Apriori algorithm obtains frequent itemsets iteratively layer by layer. It uses Properties 1 and 2 to reduce the number of candidate itemsets, avoiding excessive candidate generation and unnecessary database scans for support calculation.

1.1 Properties and Definitions

Property 1: Any non-empty subset of a frequent itemset is also frequent.
Property 2: Any superset of an infrequent itemset is also infrequent.

Support refers to the proportion of transactions in the record set that contain a given itemset, where "contain" means the itemset is a subset of the transaction.

1.2 Algorithm Description

This paper improves Apriori' s efficiency in the frequent itemset mining step. The Apriori algorithm proceeds as follows:

a) Scan the database to obtain support for all items and generate frequent 1-itemsets.

b) For two frequent k-itemsets A:{A[1],A[2],⋯,A[k-1],A[k]} and B:{B[1],B[2],⋯,B[k-1],B[k]}, if they satisfy A[1]=B[1], A[2]=B[2], ⋯, A[k-1]=B[k-1], A[k]!=B[k], then itemsets A and B can be joined to form a candidate frequent (k+1)-itemset {A[1],A[2],⋯,A[k],B[k]}.

c) **Pruning:** According to Property 2, if any k-item subset of a candidate (k+1)-itemset is not a frequent k-itemset, then the candidate cannot be a frequent (k+1)-itemset. This eliminates invalid candidates.

d) **Support counting:** Traverse the database to count support for each candidate (k+1)-itemset, filtering out those below minimum support. During this process, each candidate (k+1)-itemset requires (k+1) database traversals to complete support statistics.

e) Repeat steps b)-d) until no new candidate frequent itemsets can be generated.

1.3 Apriori Algorithm Defects

The main drawbacks of Apriori (illustrated through generating frequent (k+1)-itemsets from frequent k-itemsets) are:

a) **Inefficient join and prune operations:** After identifying that two frequent k-itemsets share (k-1) identical items, the algorithm must verify that all k subsets of the generated candidate (k+1)-itemset are frequent k-itemsets.

b) **Excessive I/O operations:** Each candidate (k+1)-itemset requires (k+1) database traversals to verify if it is frequent, and I/O operations are extremely inefficient.

c) **Pattern matching overhead:** Scanning the database requires pattern matching between candidate itemsets and transactions, which is time-consuming.

---

2 BE-Apriori Algorithm

As is well known, I/O access costs are several orders of magnitude higher than memory access. The proposed BE-Apriori algorithm improves all three steps of Apriori to enhance frequent itemset mining efficiency. It requires only two database scans: the first to obtain frequent 1-itemsets, and the second to encode transaction records based on these itemsets and load them into memory. All subsequent computations are performed in memory, effectively avoiding time loss due to inefficient I/O operations.

2.1 Basic Concepts

If the number of frequent 1-itemsets is n, all itemsets can be encoded as binary numbers with length $\leq n$. Each position in an n-bit binary number represents an item. If an item exists in the itemset, its position is set to 1; otherwise, it is 0. Encoded binary numbers may be shorter than n bits when leading items are absent. This encoding applies to all transactions, frequent itemsets, and candidate itemsets.

Property 3: When joining frequent k-itemsets A and B to obtain candidate (k+1)-itemset C, if the itemset {A[k],B[k]} in C is not frequent, then C cannot be frequent.

Proof: By Property 2, any superset of an infrequent itemset is infrequent. Since candidate C' s subset {A[k],B[k]} is not frequent, C cannot be frequent.

Property 4: For encoded itemsets a and b, the result of $a\&b$ represents the intersection of a and b. For example, if frequent 1-itemsets are {A,B,C}, itemset a={A,B} encodes to 110, itemset b={A,B,C} encodes to 111, and itemset c={A,C} encodes to 101. Then $a\&b=a$ indicates {A,B} is a subset of {A,B,C}, while $a\&c\neq a$ indicates {A,B} is not a subset of {A,C}.

Property 5: For encoded itemsets a and b, the result of $a\hat{}b$ represents the difference between a and b. For example, with frequent 1-itemsets {A,B,C}, itemset a={A,B} encodes to 110 and b={A,B,C} encodes to 111, so $a\hat{}b$=001, indicating the itemset not shared by both is {C}.

2.2 Example Illustration

Consider the transaction records in Table 1 with minimum support set to 0.4.

Table 1 Transaction Record List of Items
A,B,C,D
A,C,D
C,D,E

The algorithm proceeds as follows:

a) Scan the database to obtain support for each 1-itemset, filter out those below minimum support, and generate frequent 1-itemsets with their encodings as shown in Table 2 .

Table 2 Frequent 1-Itemsets and Encoded Results
Frequent 1-Itemsets

b) Encode transaction records based on frequent 1-itemsets. With 4 frequent 1-itemsets, the encoding length is $ $4 bits. The encoded transaction records are shown in Table 3 .

Table 3 Encoded Transaction Records

c) Generate candidate 2-itemsets by traversing frequent 1-itemsets. No pruning is possible at this stage, resulting in a relatively large candidate set. The candidate 2-itemsets and their encodings are shown in Table 4 .

d) Calculate support for each candidate 2-itemset. The AND operation results between candidate 2-itemset 1100 and each transaction are shown in Table 5 .

According to Property 4, the number of AND operations between candidate 2-itemset 1100 and transaction records that equal the candidate itself is 2, yielding a support of 0.4 for 1100.

Similarly, 1010 has support 0.4, 1001 has support 0, 0110 has support 0.8, 0101 has support 0.4, and 0011 has support 0.2. Thus, the frequent 2-itemsets are {1100, 1010, 0110, 0101}.

Table 4 Candidate 2-Itemsets and Encoded Results
Candidate 2-Itemsets

Table 5 AND Operation Results Between Candidate 2-Itemset 1100 and Encoded Transaction Records
AND Results with Transaction Records

e) Generate candidate 3-itemsets. By Property 5, since 1100^{1010}=1100, the items not shared by frequent 2-itemsets 1100 and 1010 form 1100, which belongs to frequent 2-itemsets. Therefore, 1100+1100&1010 becomes a candidate 3-itemset. No other candidates exist, yielding candidate 3-itemset 1110.

f) Obtain frequent 3-itemsets. The AND operation results between 1110 and each transaction are shown in Table 6 .

Table 6 AND Operation Results Between Candidate 3-Itemset 1110 and Encoded Transaction Records
AND Results with Transaction Records

The support of 1110 is 0.4, making it a frequent 3-itemset. Since only one such itemset exists, no candidate 4-itemsets can be generated, and the algorithm terminates.

2.3 BE-Apriori Algorithm Description

Let L denote frequent k-itemsets, C denote candidate frequent k-itemsets, and D denote transaction records. The BE-Apriori pseudocode is as follows:

Input: Transaction records D, minimum support *minsup*
Output: Frequent itemsets L

```
transactions = encode(D)
for (k=2; L_{k-1}$\neq$ ; k++) {
    if (k==2) C_k = apriori-gen-1(L_{k-1})
    else C_k = apriori-gen-2(L_{k-1})
    for each transaction t D {
        for each candidate c C_k {
            if (t & c == c) c.count++
        }
    }
    L_k = {c C_k | c.count $\geq$ minsup}
}
Answer =  _k L_k
```

apriori-gen-1:
Insert into C
Select $p.item_1$, $p.item_2$, ⋯, $p.item$, $q.item$
From *L_{k-1} p, L_{k-1} q*

Where $p.item_1 = q.item_1, \cdots, p.item\_\{k\text{-}1\} = q.item\_\{k\text{-}1\}, p.item\_k < q.item\_k$

apriori-gen-2:
Insert into C
Select $p.item_1, p.item_2, \cdots, p.item , q.item$
From $L\_\{k\text{-}1\}$ p, $L\_\{k\text{-}1\}$ q
Where $p.item_1 = q.item_1, \cdots, p.item\_\{k\text{-}1\} = q.item\_\{k\text{-}1\}, p.item\_k < q.item\_k$ and $p \frown q$ is frequent

2.4 BE-Apriori Performance Analysis

a) **Reduced database scanning:** The algorithm scans the database only twice—first to obtain frequent 1-itemsets, then to encode transaction records. The encoded transaction set replaces the original database, enabling all subsequent computations in memory. This significantly reduces time spent on I/O operations from frequent database scanning.

b) **Efficient join and prune operations:** In Apriori, the join operation requires checking whether two frequent k-itemsets share (k-1) identical items, which demands sorted itemsets with time complexity *O(log k)*. The prune operation must verify all k subsets of a candidate (k+1)-itemset, requiring *O(k)* time. Thus, join and prune together cost *O(log k + k)*. BE-Apriori completes both operations in one step by checking whether the XOR of two encoded frequent k-itemsets is a frequent 2-itemset, achieving *O(1)* constant time complexity.

c) **Optimized set operations:** After Property 1, most operations reduce to set operations. Encoded itemsets leverage computer-supported binary operations to replace set operations directly. While set operations in programming languages require conversion to specific data structures, binary-encoded itemsets eliminate this intermediate step, substantially improving BE-Apriori' s execution efficiency.

---

3 Experimental Results and Analysis

Experiments were conducted on a system with 8GB of 1867 MHz DDR3 memory, macOS 10.13.5, using Python 3.6. We implemented the baseline Apriori algorithm, VM_{Apriori}, and BE-Apriori. The dataset was the mushroom data from the Frequent Itemset Mining Dataset Repository, containing 8,124 transactions of length 23 with 120 total items.

Table 7 shows memory consumption for the three algorithms with support=0.2 and transaction counts of 1,000, 2,000, 4,000, and 8,000. **Figure 1** [Figure 1: see original paper] compares runtime for supports of 0.12, 0.15, 0.17, and 0.2.

Table 7 Memory Consumption (Bytes) of Three Algorithms

Algorithm	Transactions	Memory Usage
Apriori	···	···
VM_{Apriori}	···	···
BE-Apriori	···	···

Figure 1 [Figure 1: see original paper] Runtime of Three Algorithms

Table 7 demonstrates that for the same transaction volume, BE-Apriori's binary-encoded records occupy significantly less memory than Apriori's original storage and VM_{Apriori}' s matrix storage, making memory loading feasible.

When support is small, more frequent 1-itemsets are generated, increasing computational load in subsequent iterations and causing runtime to decrease as support increases. Figure 1 clearly shows BE-Apriori' s runtime is substantially lower than the other two algorithms at small support values and remains superior at larger support values.

Experimental results demonstrate that the algorithm significantly improves both time and space efficiency compared to Apriori.

---

4 Conclusion

The evolution from Apriori to BE-Apriori involves no complex derivations and remains simple to understand. The algorithm comprehensively addresses Apriori' s defects in frequent itemset mining with corresponding solutions. It innovatively proposes using binary-encoded itemsets as an in-memory representation, enabling efficient equivalent set operations. Experimental comparisons confirm that the algorithm effectively improves Apriori' s execution efficiency and space utilization.

However, when frequent 1-itemsets are numerous, encoding length increases, potentially consuming more memory. Future work may explore alternative encoding methods to address this limitation.

References

[1] Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large database [C]// Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1993: 207-216.

[2] Lei Xuefeng. Mining data analysis of mine monitoring based on association rules [J]. Coal Technology, 2017(11): 289-291.

[3] Qu Rui, Zhang Tianqiao. Improved Apriori algorithm based on matrix compression [J]. Computer Engineering and Design, 2017,38(8): 2127-2131.

[4] Tu Ming, Zhang Gongrang, Cheng Yeyuan. An efficient incremental Updating Algorithm for mining association rules [J]. Microelectronics & Computer, 2010(9): 56-60.

[5] Qu Meng, Zou Shurong, Fang Rui. An improved Apriori algorithm based on matrix [J]. Information Technology, 2018(3): 150-154, 158.

[6] Cao Ying, Miao Zhigang. Improved Apriori algorithm based on vector matrix optimization frequent items [J]. Journal of Jilin University: Science Edition, 2016, 54(2): 349-353.

[7] Cheng Jiangping, Fu Zhongliang, Xu Zhihong. An improved algorithm of Apriori [J]. Geomatics and Information Science of Wuhan University, 2003,28(1): 94-99.

[8] Huang JianMing, Zhao Wenjing, Wang Xingxing. improved Apriori algorithm based on across linker [J]. Computer Engineering, 2009,35(2): 37-38, 41.

[9] Cui Guanxun, Li Liang, Wang Keke, et al. Research and improvement on Apriori algorithm of association rule mining [J]. Journal of Computer Applications, 2010, 30(11): 2952-2955.

[10] Ramalho L. Fluent Python [M]. An Dao, Wu Ke, trans. Beijing: Posts & Telecon Press, 2017: 797-798.

[11] Li Dechen, Lyu Yifan, Zhao Xuejian. A frequent item-set mining algorithm based on prejudgment and screening [J]. Computer Technology and Development, 2018,28(5): 99-102.

Note: Figure translations are in progress. See original paper for figures.