Improved Apriori Algorithm Based on Spark

ZHANG Yuntao
ID:20551081

LIANG Shuang
ID: 20559095

Abstract

Apriori is one of classical algorithms for mining frequent item sets and association rules. In this project, we try to compensate the shortcomings of the traditional Apriori algorithm on the implementation in Spark by designing special data structures and modifying the algorithm to process data in parallel.

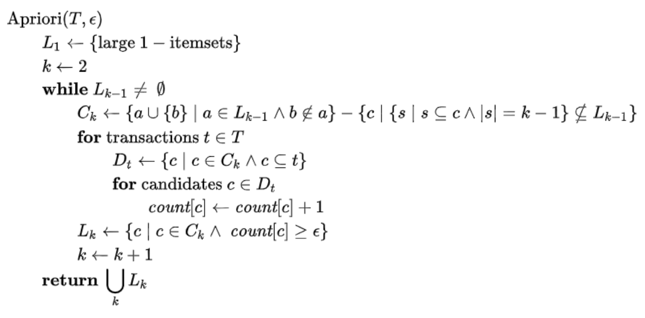
1 Introduction

Big data is one of the hottest topics in recent years and association rule mining is an important research direction in data mining. Association rule aims to discover interesting relations between variables in large databases. For example, people often buy a mouse when they buy a new computer, thus there is some association between mouse and computer, which indicates that the shop can recommend mice while finding customers who have bought new computers. Apriori algorithm is one basic algorithm in association rule mining.

2 Apriori Algorithm

The main idea for Apriori algorithm is to find high-dimensional frequent item sets by low-dimensional frequent item sets. The pseudo code is given in Figure 1. where T is a transaction database, ϵ is a support threshold, C_k is the candidate set for level k. L_k denotes frequent item sets of length k and D_t denotes candidates which are contained in the transaction t.

After frequent item sets generation, Apriori algorithm breaks each frequent item sets into left-side item set and right-side item set as candidate association rules. Then calculate the confidence based on the ratio between the support of right-side item set and the original frequent item set for each possible candidate. Finally, candidates with confidence higher than the threshold are selected as association rules.



```
Apriori($T, \epsilon$)
    $L_1 \leftarrow \{\text{large } 1 - \text{itemsets}\}$
    $k \leftarrow 2$
    while $L_{k-1} \neq \emptyset$
        $C_k \leftarrow \{a \cup \{b\} \mid a \in L_{k-1} \wedge b \notin a\} - \{c \mid \{s \mid s \subseteq c \wedge |s| = k-1\} \nsubseteq L_{k-1}\}$
        for transactions $t \in T$
            $D_t \leftarrow \{c \mid c \in C_k \wedge c \subseteq t\}$
            for candidates $c \in D_t$
                $count[c] \leftarrow count[c] + 1$
        $L_k \leftarrow \{c \mid c \in C_k \wedge \ count[c] \geq \epsilon\}$
        $k \leftarrow k + 1$
    return $\bigcup_k L_k$
```

Figure 1: apriori algorithm

3 Improve Apriori Based on Spark

From the process above we can find that apriori algorithm needs to scan the whole transactions for each time when obtaining support of candidate frequent item sets, which leads to heavy I/O burden and reduces the efficiency especially when the number of candidate frequent item sets in the transactions is large.

Our project aims to improve the Apriori algorithm in the field of big data. Spark allows users to cut the dataset into a few data blocks and remembers the actions on them, which helps save time. And we will introduce data matrix to save information.

3.1 Design of Data Structure

We plan to map the transactions onto a special data structure as shown in Figure 2. Note that key I_i denotes distinct item in the transactions, d_{ij} denotes whether the transaction j contains item I_i and s_j denotes the number of items which are contained in the transaction j.

$$d_{ij} = \{ \begin{matrix} 0, I_i \in T_j \\ 1, I_i \notin T_j \end{matrix}, s_j = \sum_{i=1}^{|n|} d_{ij} or |T_j|$$

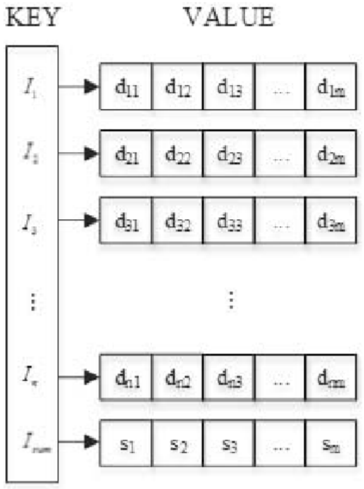
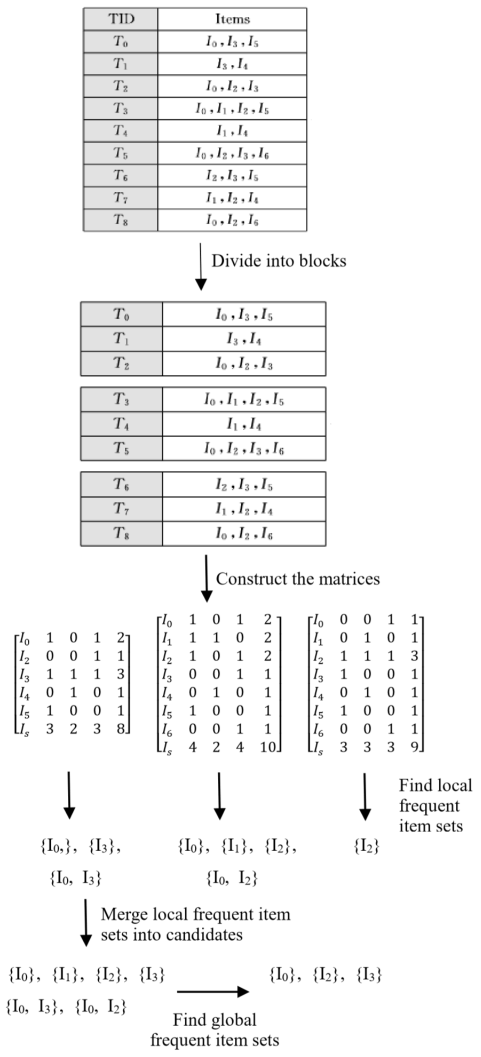


Figure 2: data structure

3.2 Implementation of Apriori

To implement Apriori algorithm in the distributed environment, we need to distribute computing to each node. The main steps are shown as following:

1) Divide the transaction into n blocks horizontally. Apply Apriori algorithm on each block to get the local frequent item sets with support larger than minimum support.
2) Merge all the local frequent item sets as candidate global frequent item sets. Then obtain support for each item set by analyzing the previously mentioned data structure.
3) Filter the candidates by comparison with minimum support, the remain will be global frequent item sets. A detailed example is shown in Figure 3, suppose minimum support = 0.5.



TID	Items
T_0	I_0, I_3, I_5
T_1	I_3, I_4
T_2	I_0, I_2, I_3
T_3	I_0, I_1, I_2, I_5
T_4	I_1, I_4
T_5	I_0, I_2, I_3, I_6
T_6	I_2, I_3, I_5
T_7	I_1, I_2, I_4
T_8	I_0, I_2, I_6

T_0	I_0, I_3, I_5
T_1	I_3, I_4
T_2	I_0, I_2, I_3

T_3	I_0, I_1, I_2, I_5
T_4	I_1, I_4
T_5	I_0, I_2, I_3, I_6

T_6	I_2, I_3, I_5
T_7	I_1, I_2, I_4
T_8	I_0, I_2, I_6

$$\begin{bmatrix} I_0 & 1 & 0 & 1 & 2 \\ I_2 & 0 & 0 & 1 & 1 \\ I_3 & 1 & 1 & 1 & 3 \\ I_4 & 0 & 1 & 0 & 1 \\ I_5 & 1 & 0 & 0 & 1 \\ I_s & 3 & 2 & 3 & 8 \end{bmatrix} \quad \begin{bmatrix} I_0 & 1 & 0 & 1 & 2 \\ I_1 & 1 & 1 & 0 & 2 \\ I_2 & 1 & 0 & 1 & 2 \\ I_3 & 0 & 0 & 1 & 1 \\ I_4 & 0 & 1 & 0 & 1 \\ I_5 & 1 & 0 & 0 & 1 \\ I_6 & 0 & 0 & 1 & 1 \\ I_s & 4 & 2 & 4 & 10 \end{bmatrix} \quad \begin{bmatrix} I_0 & 0 & 0 & 1 & 1 \\ I_1 & 0 & 1 & 0 & 1 \\ I_2 & 1 & 1 & 1 & 3 \\ I_3 & 1 & 0 & 0 & 1 \\ I_4 & 0 & 1 & 0 & 1 \\ I_5 & 1 & 0 & 0 & 1 \\ I_6 & 0 & 0 & 1 & 1 \\ I_s & 3 & 3 & 3 & 9 \end{bmatrix}$$

Find local frequent item sets
{I0,}, {I3}, {I0, I3}
{I0}, {I1}, {I2}, {I0, I2}
{I2}
Merge local frequent item sets into candidates
{I0}, {I1}, {I2}, {I3} {I0, I3}, {I0, I2}
Find global frequent item sets
{I0}, {I2}, {I3}

Figure 3: process example

Finally we will test the performance of the algorithm by comparing the running time.