Research on CM_HI_pApriori Algorithm Based on Hyperthyroidism

Renjian Hu and Yuanbo Shi

School of Computer and Communication Engineering, Liaoning Shihua University, Liaoning, China
Email: 1444044709@qq.com; syb2011@yeah.net

Abstract. Aiming at the problems of the existing mining algorithm processing data scale, the time complexity of frequent item set storage, and the limitation of the generated association rules, the CM_HI_pApriori (parallel Apriori algorithm of the compressed matrix based on HashMap and Interest) algorithm is proposed. First of all, the parallel data partition method is used to solve the problem of low serialization efficiency. Secondly, HashMap is used to store and query frequent item sets, which makes the time complexity of storing frequent item sets change from linear growth to constant growth. Finally, the interest model is used to solve the problem that association rules do not conform to the actual situation. Experiments show that the CM_HI_pApriori algorithm not only has high scalability and stability in processing large-scale data but also has high accuracy in generating association rules.

Keywords. Data mining; Hadoop; Apriori algorithm; data partition parallelization; HashMap; interest.

1. Introduction

With the development of the hospital information system, the data of medical activities and medical research were stored, which has formed massive medical big data. Recently, “Nature Reviews Endocrinology” has published a review titled “Global Epidemiology of Hyperthyroidism and Hypothyroidism” [1]. Although people have a full understanding of hyperthyroidism, the diagnosis and treatment of hyperthyroidism cannot be completely controlled due to the different populations and stages, and other complex environments. Therefore, it is necessary to accurately mine the medical big data of hyperthyroidism to find out the association rules of potential value, so that it is possible to realize the precise forecast and treatment of hyperthyroidism. At present, the main method of mining big data is the NCM_Apriori_1 algorithm [2] based on the compressed matrix. However, the NCM_Apriori_1 algorithm based on the compressed matrix has the following three problems: First, large-scale data processing is inefficient; Second, the time complexity of querying and storing frequent item sets is too high; Third, this algorithm generates a large number of association rules with low precision. In order to improve the shortcomings of the NCM_Apriori_1 algorithm, researchers have proposed improvement measures on different aspects. Reference [3] proposed an effective association rule mining method based on the compressed matrix. The method realized association rules mining by compressing the matrix, improving the storage structure of the matrix, and sorting the item sets. Reference [4] proposed a matrix-based parallel optimization algorithm, which simplified the “self-connection” process and “pruning” process of the Apriori algorithm. Reference [5] solved the problems that the Apriori algorithm needs to scan the transaction database multiple times and generate a large number of irrelevant candidate item sets. It was proposed to classify and count the number of transaction items and support items in the
matrix during matrix operation, and delete infrequent item sets to form a new matrix structure, and improve storage space efficiency.

Given the above problems, this paper proposes: (1) Using data partitioning and parallel processing to overcome the low efficiency of the traditional serial method; (2) Using HashMap structure to store and query frequent item sets to solve the problem of high time complexity; (3) Using the fusion of interest to improve the limitation of association rules.

2. Basic Concepts

2.1. Data Partition Parallelization
The idea adopts the data partition processing method which runs in parallel on the Hadoop platform [6-7]. The steps are as follows: firstly, the data set is divided into n blocks to n nodes; secondly, by calculating each node, n local strong association rules $R'_n$ are obtained; finally, they are merged into a global strong association rule $R'$. The specific data partition method can use the FTDV strategy [8], which considers the content characteristics of data and the scale of the data comprehensively. The idea transforms a single serial processing mode into parallelization of multiple data blocks, which are calculated by each node separately, thus the idea can solve the I/O bottleneck problem, reduce the running time greatly and improve the execution efficiency.

2.2. Store and Query Frequent Item Sets Based on the HashMap
The idea proposes that the HashMap is used to store and query frequent item sets and its support. HashMap structure can be used to store key-value pairs and HashMap structure is the Entry < key, value >. The implementation of HashMap storage is to hash each key to obtain the hash value of each key. In this idea, key stands for the support of frequent item set, and value stands for frequent item set.

On one hand, it is assumed that the traditional Apriori algorithm adopts a dynamic array structure, and the time complexity of storing frequent item sets is $O(n)$, while that of the HashMap structure is $O(1)$. In contrast, the time complexity of storing changes from linear to constant, and the time consumption is greatly reduced; On the other hand, the time complexity of querying of dynamic array structure is $O(1)$, that of linked list structure is $O(n)$, and that of HashMap structure is $O(1)$ $\sim O(n)$ [9]. Therefore, considering comprehensively, it is more appropriate to use HashMap to store and query frequent item sets.

2.3. Interest Model
The idea proposes the interest model which not only considers association rules $x \Rightarrow y$, but also considers association rules $\bar{x} \Rightarrow y$. The expression is shown in equation (1).

$$Int(x \Rightarrow y) = \frac{\text{conf}(x \Rightarrow y)}{\text{conf}(\bar{x} \Rightarrow y)}$$

$\text{conf}(x \Rightarrow y)$ is the probability that $y$ appears when $x$ exists; $\text{conf}(\bar{x} \Rightarrow y)$ is the probability that $y$ appears when $\bar{x}$ does not exist. According to equation (1), it can be divided into three situations: (1) When the $Int(x \Rightarrow y) > 1$, it means that $x$ and $y$ have a positive correlation. (2) When $Int(x \Rightarrow y) = 1$, it means that $x$ and $y$ do not exist correlation; (3) When $0 < Int(x \Rightarrow y) < 1$, it means that $x$ and $y$ have a negative correlation.

3. CM_HI_pApriori Algorithm

3.1. Algorithm Idea
First of all, the pre-processed data needs to be partitioned and parallelized, so as to reflect the running state of the data set efficiently and improve the running efficiency quickly.

Secondly, the NCM_Apriori_1 algorithm is still used in the stage of generating frequent item sets. However, the HashMap structure can be used to store and query frequent item sets. And the HashMap structure optimizes the time complexity of storing and querying frequent item sets.
Finally, the interest model is added, which can be used to generate strong association rules which users are interested in.

Therefore, the algorithm can be named CM_HI_pApriori algorithm (parallel Apriori algorithm of the compressed matrix based on HashMap and Interest).

3.2. CM_HI_pApriori Implementation on MapReduce

On the Hadoop platform, the CM_HI_pApriori algorithm is implemented by MapReduce design. Firstly, the pre-processed data set is imported into the HDFS; then the calculation runs on the Hadoop platform; finally, the output result is returned to the HDFS. As can be seen from figure 1, the MapReduce design of the algorithm mainly includes four steps: (1) Generation of frequent item sets \( L_1 \). (2) Generation of frequent item sets \( L_k \). (3) Generation of initial association rules \( R \). (4) Generation of strong association rules \( R' \) which users are interested in.

![Figure 1. CM_HI_pApriori algorithm implementation on MapReduce.](image)

The pseudo-code of the specific steps is as follows:

**Algorithm step 1 Generation of \( L_1 \)**

**Input:** Transaction data set \( D \), \( \text{min supp} \)

**Output:** Frequent item set \( L_1 \) and its support

1. **Function** Map
   1. \( \text{Scan } D \);  
2. \( \text{foreach } D_i \ (i=1..n) \in D \ do \)
   (Select data block \( D_i \) as an example)
3. \( A = \text{Translate}_\text{(} D_i \text{)}; \)
4. \( \text{foreach } \text{item } I_j \in \text{itemsets do} \)
5. \( \text{Output } <\text{Item, support}>; \)
6. **Function** Reduce
   1. \( \text{Reducer (key, list } [\text{sup } p_1, \text{sup } p_2, \ldots, \text{sup } p_j ]; \)
7. \( \text{sup } p_{-1} = 0; \)
8. \( \text{sup } p_{-1} += \text{list } [\text{sup } p_j ]; \)
9. \( \text{if } \text{sup } p_{-1} \geq \text{min supp} \) then
10. \( \text{Output } <L_1, \text{sup } p_{-1} >; \)
11. \( A_1 = \text{Update}_\text{(} A \text{)} \)
Algorithm step 2 Generation of $L_k$

**Input:** The candidate set $C_{k-1}$, matrix $A_{k-1}$, $min\_supp$

**Output:** Frequent item set $L_k$ and its support

1. **Function** Map
2. foreach vector $k$ in $A_{k-1}$ do
3. Output $<\text{item, sup}_{-k}>$;
4. $A_k = \text{Update } (A_{k-1})$;
5. **Function** Reduce
6. foreach $sup$ in list $[sup_{-k}]$ do
7. \[ sup = \text{count} += sup; \]
8. if $sup - \text{count} \geq min\_supp$ then
9. Output $<L_k, sup - \text{count}>$

Algorithm step 3 Generation of initial association rules $R$

**Input:** Frequent item set $L_1 \ldots L_k$, $min\_conf$

**Output:** Initial association rule $R$

1. **Function** Map
2. foreach frequent item set $x, y \in L_1 \ldots L_k$ do
3. calculate $conf_{-k} = \text{Conf}(x \Rightarrow y) = \frac{p(xy)}{p(x)}$;
4. Output $<(x \Rightarrow y), conf_{-k}>$;
5. **Function** Reduce
6. Reducer (key, list $[conf_{-k}]$)
7. $R = \emptyset$;
8. foreach $conf$ in list $[conf_{-k}]$ do
9. Output $conf$;
10. if $(conf \geq min\_conf)$ then
11. Output $R = R \cup \{(x \Rightarrow y)\}$

Algorithm step 4 Generation of strong association rules $R'$

**Input:** Initial association rule $R$, Frequent item set $L_1 \ldots L_k$, $min\_int_1 \ (min\_int_1 > 1)$ and $min\_int_2 \ (0 < min\_int_2 < 1)$

**Output:** The strong association rule $R'$, the positive association rule $R_+$, the negative association rule $R_-$

1. **Function** Map
2. foreach frequent item set $x, y \in L_1 \ldots L_k \in R$ do
3. calculate $int_{-k} = \text{Int}(x \Rightarrow y) = \frac{conf(x = y)}{conf(x = y)}$;
4. Output $<(x \Rightarrow y), int_{-k}>$
5. **Function** Reduce
6. Reducer (key, list $[int_{-k}]$);
7. $R_+ = \emptyset$, $R_- = \emptyset$;
8. foreach $int$ in list $[int_{-k}]$ do
9. if $(int > 1)$ then
10. if $(int \geq min\_int_1)$ then
11. Output $R_+ = R_+ \cup \{(x \Rightarrow y)\}$;
12. if $(0 < int < 1)$ then
13. if $(0 < int \leq min\_int_2)$ then
Output $R_\neg = R_\neg \cup \{(x \Rightarrow y)\}$

if $(\text{int}=1)$ then

Output $R' = R_+ \cup R_\neg$

### 4. Experimental Evaluations

The experimental data set was collected by “Global Epidemiology of Hyperthyroidism and Hypothyroidism” [1], and the data sizes are 100MB, 300MB, 500MB, 700MB, 900MB. Each data set includes the following attributes \{AGE, SEX, WEIGHT, WBC, RBC, HGB, LYMPH, TBIL, TSH, FT4, TT3, FT3, TGAB, TMAB, SYMPTOM, RESULT, ACCOMPANY SYMPTOM, etc.\}. The primary keys include \{AGE, SEX, WEIGHT, SYMPTOM, and RESULT\}.

In this experiment, 6 machines are used to build a Hadoop distributed platform, one of which is the master, five of which are slaves. Each machine has an Intel Core i5-2400 processor with 4 3.10GHz processor cores, 8GB of RAM. All running Ubuntu Desktop operating system 14.04 LTS, Hadoop 2.4.1.

#### 4.1. Comparison of $\text{min\_supp}$ on Two Algorithms and Comparison of $\text{min\_conf}$ on Two Algorithms

This experiment uses the CM_HI_pApriori algorithm to compare with the NCM_Apriori_1 algorithm. The experiment of comparison of $\text{min\_supp}$ is performed under the condition that $\text{min\_conf}$ is 0.7 and $\text{min\_supp}$ is 0.3, 0.55, 0.7. The experiment of comparison of $\text{min\_conf}$ is performed under the condition that $\text{min\_supp}$ is 0.3 and $\text{min\_conf}$ is 0.8, 0.85, 0.9. Each run is performed five times, and the average running time is selected as the evaluation metric.

As can be seen from figures 2-7, from a longitudinal comparison, under the same $\text{min\_conf}$, the runtime of the two algorithms continues to decrease with the increase of $\text{min\_supp}$; under the same $\text{min\_supp}$, the runtime of the two algorithms also decreases with the increase of $\text{min\_conf}$. From a horizontal comparison, the runtime of the CM_HI_pApriori algorithm is longer than that of the NCM_Apriori_1 algorithm when the data size is less than 500MB. This is because the CM_HI_pApriori algorithm needs data partitioning which consumes more communication and partition time. However, when the data size is greater than or equal to 500MB, the advantage of the parallel processing of the CM_HI_pApriori algorithm is reflected, so the runtime is much shorter than that of the NCM_Apriori_1 algorithm.

**Figure 2.** Runtime for algorithms for $\text{min\_supp}$ is 0.3, $\text{min\_conf}$ is 0.7.

**Figure 3.** Runtime for algorithms for $\text{min\_supp}$ is 0.55, $\text{min\_conf}$ is 0.7.
Figure 4. Runtime for algorithms for min_supp is 0.7, min_conf is 0.7.

Figure 5. Runtime for algorithms for min_supp is 0.3, min_conf is 0.8.

Figure 6. Runtime for algorithms for min_supp is 0.3, min_conf is 0.85.

Figure 7. Runtime for algorithms for min_supp is 0.3, min_conf is 0.9.

4.2. Comparison of min_int on Two Algorithms

The experiment is performed under the condition that min_int is 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75 and min_supp is 0.3, and min_conf is 0.7, and data size is 500MB. The number of strong association rules is shown in figure 8.

From a longitudinal comparison, under the same interest, the number of association rules generated by the CM_HI_pApriori algorithm is less than that of the NCM_Apriori_1 algorithm; from a horizontal comparison, the number of strong association rules will also rapidly decrease when the interest decreases from 1.0 to both sides. This proves that interest setting has a certain effect on reducing the number of association rules and eliminating redundant association rules; it also has a certain effect on improving the accuracy of association rules and eliminating irrelevant association rules.

4.3. SpeedUp, SizeUp and ScaleUp

In this experiment, SpeedUp, SizeUp, and ScaleUp are used as evaluation criteria [10-11] to test the performance of the two algorithms, respectively:

\[
\text{SpeedUp} = \frac{T_1}{T_p}
\] (2)
SizeUp \( (data, m) = \frac{T_m}{T_k} \) \hspace{1cm} \text{(3)}

ScaleUp \( (data, n) = \frac{T_k}{T_{nk}} \) \hspace{1cm} \text{(4)}

The graphic of SpeedUp experiments is shown in figure 9. The SpeedUp of the two algorithms keeps a stable linear ratio with the number of computing nodes. Under the same node, the SpeedUP of the CM_HI_pApriori algorithm is higher than that of the NCM_Apriori_1 algorithm. With the increase of node and data size, the acceleration performance is getting better and better. This proves that the multi-node computing cluster can reduce computing time well.

The graphic of SizeUp experiments is shown in figure 10. With the increase of data size, the SizeUp of the two algorithms increases gradually, which proves that the efficiency of the two algorithms in processing large-scale data is getting higher and higher. Under the same data size, the SizeUp of the CM_HI_pApriori algorithm is higher than that of the NCM_Apriori_1 algorithm. This fully explains that the CM_HI_pApriori algorithm can well solve the scalability problem in dealing with big data.

The graphic of ScaleUp experiments is shown in figure 11. with the increase of computing nodes, the ScaleUp of the two algorithms decreases continuously and the ScaleUp is always less than 1. Under the same conditions, the ScaleUp of the CM_HI_pApriori algorithm is smaller than that of the NCM_Apriori_1 algorithm. This indicates that the CM_HI_pApriori algorithm has good stability and scalability.
5. Conclusion
In this paper, the CM_HI_pApriori algorithm is proposed based on the NCM_Apriori_1 algorithm, and the CM_HI_pApriori algorithm is run on the Hadoop platform. At the same time, experimental comparison and analysis show that the algorithm has greatly improved the way of dealing with data scale, the time complexity of storing and querying frequent item sets, and the quantity and quality of association rules. Furthermore, the CM_HI_pApriori algorithm can be effectively applied to mining association rules of hyperthyroidism big data.

References
[1] Taylor P N, Albrecht D, Scholz A, et al. 2018 Global epidemiology of hyperthyroidism and hypothyroidism Nature Reviews Endocrinology (14) 301-316.
[2] Luo D and Li T S 2013 Improved apriori algorithm based on compressed matrix Computer Science 40 (12) 75-80.
[3] Pan J H and Zhang Q 2019 An efficient algorithm for mining association rules based on compressed matrix Computer and Digital Engineering 47 (11) 2819-2823.
[4] Liang Y, Yuan D, et al. 2018 Optimized implementation of association rules algorithm based on spark platform Computer Engineering and Design 384 (12) 100-107.
[5] Liu F and Wu G C 2018 An improved apriori algorithm based on the compressed matrix Journal of Shandong University (Engineering Edition) 48 (6) 82-88.
[6] Ahmed E and Mohamed F.M 2016 Spatial Hadoop: A MapReduce Framework for Spatial Data IEEE International Conference on Data Engineering.
[7] Rao B T and Reddy L S S 2012 Survey on improved scheduling in Hadoop MapReduce in cloud environments International Journal of Computer Applications 34 (S2) 96-98.
[8] Ying J J and Wang L 2010 Parallel k-means algorithm for data partition optimization Computer Engineering and Application 46 (15) 127-131.
[9] Alhanjouri M, et al. 2012 A new method of query over encrypted data in database using HashMap International Journal of Computer Applications 41 (4) 46-51.
[10] Castro E P S, Maia T D, Pereira M R, et al. 2018 Review and comparison of Apriori algorithm implementations on Hadoop-MapReduce and spark Knowledge Engineering Review 33 (e9) 1-19.
[11] Zhang S F, Dong Y Y and Cheng X B 2018 Research and design of HKM clustering algorithm based on Hadoop cloud computing platform Journal of Applied Sciences 36 (003) 524-534.