Appl Intell (2015) 42:707–721
DOI 10.1007/s10489-014-0614-1

Updating mined class association rules for record insertion

Loan T. T. Nguyen · Ngoc-Thanh Nguyen

Published online: 13 December 2014

Abstract Mining class association rules is an interesting problem in classification and prediction. Some recent studies have shown that using classifiers based on class association rules resulted in higher accuracy than those obtained by using other classification algorithms such as C4.5 and ILA. Although many algorithms have been proposed for mining class association rules, they were used for batch processing. However, real-world datasets regularly change; thus, updating a set of rules is challenging. This paper proposes an incremental method for mining class association rules when records are inserted into the dataset. Firstly, a modified equivalence class rules tree (MECR-tree) is created from the original dataset. When records are inserted, nodes on the tree are updated by changing their information including *Obidset* (a set of object identifiers containing the node's itemset), *count,* and *pos*. Secondly, the concept of pre-large itemsets is applied to avoid re-scanning the original dataset. Finally, a theorem is proposed to quickly prune nodes that cannot generate rules in the tree update process. Experimental results show that the proposed method is more effective than mining entire original and inserted datasets.

Keywords CAR-Miner · Incremental dataset · Class association rule · Pre-large itemset

L. T. T. Nguyen
Division of Knowledge and System Engineering for ICT and Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh, Vietnam
e-mail: nguyenthithuyloan@tdt.edu.vn; nthithuyloan@gmail.com

L. T. T. Nguyen · N.-T. Nguyen (✉)
Department of Information Systems, Faculty of Computer Science and Management, Wroclaw University of Technology, Wroclaw, Poland
e-mail: Ngoc-Thanh.Nguyen@pwr.edu.pl

1 Introduction

Pattern mining has widely used applications in a lot of areas such as association rule mining [4, 13, 18, 24], sequence mining [19, 21], and others [3, 14]. Association rule mining is to mine relationships among items in a transaction database. An association rule has form $X \rightarrow Y$ in that X, Y are itemsets. A class association rule is an association rule whose right hand side (Y) is a class label.

Class association rule mining was first proposed by Liu et al. [12]. After that, a large number of methods related to this problem, such as Classification based on Multiple Association Rules (CMAR) [10], Classification based on Predictive Association Rules (CPAR) [27], Multi-class, Multi-label Associative Classification (MMAC) [20], ECR-CARM [23], and CAR-Miner [16] have been proposed. Results of classification based on association rule mining are often more accurate than those obtained based on ILA and decision tree [9, 12, 22].

All above studies simply focused on solving the problem of class association rule (CAR) mining based on batch processing approaches. In reality, datasets typically change due to operations such as addition, deletion, and update. Algorithms for effectively mining CARs from incremental datasets are thus required. The naïve method is to re-run the CAR mining algorithm on the updated dataset. The original dataset is often very large, whereas the updated portion is often small. Thus, this approach is not effective because the entire dataset must be re-scanned. In addition, previous mining results cannot be reused. Therefore, an efficient algorithm for updating the mined CARs when some rows are inserted into the original dataset need to be developed to solve this problem.

This work focuses on solving the problem of CAR mining from an incremental dataset (i.e., new records are added to the original dataset).

Main contributions are as follows:

1. The CAR-Miner algorithm [16] is used to build the MECR-tree for the original dataset.

   The concept of pre-large itemsets (i.e. itemsets which do not satisfy the minimum support threshold, but satisfy the lower minimum support threshold) is applied to avoid re-scanning the original dataset [5, 11].
2. When a new dataset is inserted, only information of the nodes on the MECR-tree including *Obidset, count,* and *pos*, need to be updated. During the update process, nodes which are frequent or pre-large in the original dataset but are not frequent in the updated dataset are pruned by simply processing each node. However, this task is time-consuming if many nodes on a given branch of the tree need to be removed. Therefore, a theorem is developed to eliminate such nodes.

The rest of the paper is organized as follows. Section 2 presents basic concepts of CAR mining. Section 3 presents problems related to CAR mining and frequent itemset mining from incremental datasets. Section 4 presents the proposed algorithm while Section 5 provides an example to illustrate its basic ideas. Section 6 shows experimental results on some standard datasets. Conclusions and future work are described in Section 7.

2 Basic concepts

Let D be a training dataset which includes n attributes $A_1, A_2, \ldots, A_n$ and $\mid D \mid$ objects. Let $C = \{c_1, c_2, \ldots, c_k\}$ be a list of class labels in D. An itemset is a set of pairs, denoted by $\{(A_{i1}, \mathrm{a}_{i1}), (A_{i2}, \mathrm{a}_{i2}), \ldots, (A_{\mathrm{im}}, \mathrm{a}_{\mathrm{im}})\}$, where A_{ij} is an attribute and a_{ij} is a value of A_{ij}.

A class association rule r has the form $\{(A_{\mathrm{i1}}, \mathrm{a}_{\mathrm{i1}}), \ldots, (A_{\mathrm{im}}, \mathrm{a}_{\mathrm{im}})\} \rightarrow c$, where $\{(A_{\mathrm{i1}}, \mathrm{a}_{\mathrm{i1}}), \ldots, (A_{\mathrm{im}}, \mathrm{a}_{\mathrm{im}})\}$ is an itemset and $c \in C$ is a class label. The actual occurrence of rule r in D, denoted ActOcc(r), is the number of records in D that match the left-hand side of r. The support of a rule r, denoted Sup(r), is the number of records in D that match r's left-hand side and belong to r's class.

Object Identifier (OID): OID is an object identifier of a record in D.

Example 1 Consider rule $r = \{(B, b1)\} \rightarrow y$ from the dataset shown in Table 1. ActOcc(r) = 3 and Sup(r) = 2 because there are three objects with $B = b1$, where 2 objects belong to y.

Table 1 Example of a training dataset

OID	*A*	*B*	*C*	*class*
1	a1	b1	c1	y
2	a1	b1	c2	n
3	a1	b2	c2	n
4	a1	b3	c3	y
5	a2	b3	c1	n
6	a1	b3	c3	y
7	a2	b1	c3	y
8	a2	b2	c2	n

3 Related works

3.1 Mining class association rules

This section introduces existing algorithms for mining CARs in static datasets (Table 2), namely CBA [12], CMAR [10], ECR-CARM [23], and CAR-Miner.

The first study of CAR mining was presented by [12]. The authors proposed CBA-RG, an Apriori-like algorithm, for mining CARs. To build a classifier based on mined CARs, an algorithm, named CBA-CB, was also proposed. This algorithm is based on heuristic to select the strongest rules to form a classifier. Li, Han, and Pei proposed a method called CMAR in 2001 [10]. CMAR uses the FP-tree to mine CARs and uses the CR-tree to store the set of rules. The prediction of CMAR is based on multiple rules. To predict a record with an unlabeled class, CMAR obtains the set of rules R that satisfies that record and divides them into l groups corresponding to l existing classes in R. A weighted $\chi 2$ is calculated for each group. The class with the highest weighted $\chi 2$ is selected and assigned to this record. [20] proposed the MMAC method. MMAC uses multiple labels for each rule and multiple classes for prediction. Antonie and Zaïane proposed an approach which uses both positive and negative rules to predict classes of new samples [1]. Vo and Le [23] presented the ECR-CARM algorithm for quickly mining CARs. CAR-Miner, an improved version of ECR-CARM proposed by Nguyen et al. in 2013 [16], has a significant improvement in execution time compared to ECR-CARM. Nguyen et al. [17]

Table 2 Summary of existing algorithms for mining CARs

Algorithm	Year	Approach
CBA	1998	Apriori-like
CMAR	2001	FP-tree structure-base
ECR-CARM	2008	Equivalence class rule tree
CAR-Miner	2013	Improved equivalence class rule tree

proposed a parallel algorithm for fast mining CARs. Several methods for pruning and sorting rules have also been proposed [10, 12, 15, 20, 23, 27].

These approaches are used for batch processing only; i.e., they are executed on the integration of original and inserted datasets. In reality, datasets often change via the addition of new records, deletion of old records, or modification of some records. Mining knowledge contained in the updated dataset without re-using previously mined knowledge is time-consuming, especially if the original dataset is large. Mining rules from frequently changed datasets is thus challenging problem.

3.2 Mining association rules from incremental datasets

One of the most frequent changes on a dataset is data insertion. Integration datasets (from the original and inserted) for mining CARs may have some difficulties of execution time and storage space. Updating knowledge which has been mined from the original dataset is an important issue to be considered. This section reviews some methods relating to frequent itemset mining from incremental datasets.

Cheung et al. [2] proposed the FUP (Fast UPdate) algorithm. FUP is based on Apriori and DHP to find frequent itemsets. The authors categorized an itemset in the original and inserted datasets into two categories: frequent and infrequent. Thus, there are four cases to consider, as shown in Table 3.

In cases 1 and 4, the original dataset does not need to be considered to know whether an itemset is frequent or infrequent in the updated dataset. For case 2, only the new support count of an itemset in the inserted dataset needs to be considered. In case 3, the original dataset must be re-scanned to determine whether an itemset is frequent since supports of infrequent itemsets are not stored.

Although FUP primarily uses the inserted data, the original dataset is still re-scanned in case 3, which requires a lot of effort and time for large original datasets. In addition, FUP uses both frequent and infrequent itemsets in the inserted data, so it can be difficult to apply popular frequent itemset mining algorithms. Thus, a large number of itemsets must be mined for comparison with previously mined frequent itemsets in the original dataset. In order to minimize the number of scans of the original dataset and the large number of generated itemsets from the new data, Hong et al. [5] proposed the concept of pre-large itemsets. A pre-large itemset is an infrequent itemset, but its support is larger than or equal to a lower support threshold. In the concept of pre-large itemsets, two minimum support thresholds are used. The first is upper minimum support S_U (is also the minimum support threshold) and the second is the lower minimum support S_L. With these two minimum support thresholds, an itemset is placed into one of three categories: frequent, pre-large, and infrequent. Thus, there are 9 cases when considering an itemset in 2 datasets (original and inserted), as shown in Table 4.

To reduce the number of re-scans of the original dataset, the authors proposed the following safe threshold formula f (i.e., if the number of added records does not exceed the threshold, then the original dataset does not need to be considered):

$$f = \left\lfloor \frac{(S_U - S_L) \times \mid D \mid}{1 - S_U} \right\rfloor \tag{1}$$

where $\mid D \mid$ is the number of records in the original dataset.

In 2009, Lin et al. proposed the Pre-FUFP algorithm for mining frequent itemsets in a dataset by combining the FP-tree and the pre-large concept [11]. They proposed an algorithm that updates the FP-tree when a new dataset is inserted using the safety threshold f. After the FP-tree is updated, the FP-Growth algorithm is applied to mine frequent itemsets in the whole FP-tree (created from the original dataset and inserted data). The updated FP-tree contains the entire resulting dataset, so this method does not reuse information of previously mined frequent itemsets and thus has to re-mine frequent itemsets from the FP-tree. Some effective methods for mining itemsets in incremental datasets based on the pre-large concept have been proposed, such as methods based on Trie [7] and the IT-tree [8], method

Table 3 Four cases of an itemset in the original and inserted datasets [2]

Case	Original dataset – Inserted dataset	Updated dataset
1	frequent – frequent	Frequent
2	frequent – infrequent	frequent/infrequent
3	infrequent – frequent	frequent/infrequent
4	infrequent – infrequent	Infrequent

Table 4 Nine cases of an itemset in the original and inserted datasets when using the pre-large concept

Case	Original dataset – Inserted dataset	Updated dataset
1	frequent–frequent	frequent
2	frequent–pre-large	frequent/pre-large
3	frequent–infrequent	frequent/pre-large/infrequent
4	pre-large–frequent	frequent/pre-large
5	pre-large–pre-large	pre-large
6	pre-large–infrequent	pre-large/infrequent
7	infrequent–frequent	scan the original dataset to check the itemset
8	infrequent–pre-large	infrequent/pre-large
9	infrequent–infrequent	infrequent

Table 5 Summary of algorithms for incremental mining

Algorithm	Year	Approach
FUP	1996	Apriori-based
Pre-large itemset	2001	Apriori-based and prelarge concept
Pre-FUFP	2009	FP-tree-based and prelarge concept
Pre-FUT	2011	Trie-based and prelarge concept
Pre-FUIT	2012	IT-based and prelarge concept
Frequent closed itemset lattice	2013	Lattice and prelarge concept
TMPFIL & DMPFIL	2014	Lattice and prelarge concept

for fast updating frequent itemset lattice [25], and method for fast updating frequent closed itemset lattice [6, 26]. Summary of algorithms for incremental mining is shown in Table 5.

4 A novel method for updating class association rules in incremental dataset

Algorithms presented in Section 3.2 are applied for mining frequent itemsets. It is difficult to modify them for the CAR mining problem because they are simply applied to the 1st phase of association rule mining (updating frequent itemsets when new transactions are inserted into the dataset), with the 2nd phase still based on all frequent itemsets to generate association rules.

Unlike association rule mining, CAR mining has only one phase to calculate the information of nodes, in which each node can generate only one rule whose support and confidence satisfy thresholds. Updated information of each node (related to *Obidset*, *count*, and *pos*) is much more complex than information of an itemset (related to only the support). This section presents a method that mines class association rules based on the concept of pre-large itemsets. The proposed method uses CAR-Miner to build the MECR-tree in the original dataset with few modifications; the new algorithm is called **Modified-CAR-Miner** (Fig. 1). The MECR-tree is generated with the lower minimum support threshold and then the safety threshold f is calculated using (1). A function of tree traversal for generating rules satisfying the upper minimum support threshold is then built. The number of rows of the inserted dataset is compared with the safety threshold f. If the number of rows does not exceed the safety threshold f, the tree is updated by changing the information of nodes. Otherwise, **Modified-CAR-Miner** is called to rebuild the entire tree based on the original dataset and the inserted dataset. A theorem for pruning tree nodes is also developed to reduce the execution time and storage space of nodes.

4.1 Modified CAR-Miner algorithm for incremental mining

a) MECR-tree structure

Each node in the MECR-tree contains an itemset (att, values) that includes the following information [16]:

i. *Obidset*: a set of object identifiers containing the itemset
ii. (#c1, #c2,, #ck): a set of integer numbers where $\#c_i$ is the number of objects that belong to class c_i
iii. *pos*: an integer number that stores the position of class c_i such that $\#c_i = \max_{i\in[1,k]}\{\#c_i\}$, i.e., $pos = \arg\max_{i\in[1,k]}\{\#c_i\}$, the maximum position is underlined in black.

More details about the MECR-tree can be found in Nguyen et al. [16].

b) Modified CAR-Miner algorithm

Input: Original dataset D, two minimum support thresholds S_U and S_L, and minimum confidence threshold *minConf*

Output: Class association rules mined from D that satisfy S_U and *minConf*

Figure 1 shows a modified version of CAR-Miner for incremental mining. The main differences compared to CAR-Miner are on lines 3, 19, and 22. When the procedure **GENERATE-CAR** is called to genarate a rule (line 3), the input for this function must be S_U. Therefore, lines 22 and 24 consider whether the support and confidence of the current node satisfy S_U and *minConf*, respectively. If the conditions hold, a rule is generated (line 25). Line 19 considers whether the support of a new node satisfies S_L. If so, this node is added into the tree.

4.2 Algorithm for updating the MECR-tree in incremental datasets

Theorem 1 *Given two nodes l_1 and l_2 in the MECR-tree, if l_1 is a parent node of l_2 and Sup($l_1.itemset \rightarrow c_{l_1.pos}$) < minSup, then Sup ($l_2.itemset \rightarrow c_{l_2.pos}$) < minSup.*

Proof Because l_1 is a parent node of l_2, it implies that l_1.itemset $\subset$ l_2.itemset $\Rightarrow$ all Obidsets containing l_1.itemset also contain l_2.itemset or $l_1.Obidset \supseteq l_2.Obidset \Rightarrow \forall i, l_1.count_i \geq l_2.count_i$ or $\max\{l_1.count_i\}_{i=1}^{k} \geq \max\{l_2.count_i\}_{i=1}^{k} \Rightarrow$ Sup($l_1.itemset \rightarrow c_{l_1.pos}$) $\geq$ Sup($l_2.itemset \rightarrow c_{l_2.pos}$). Because Sup ($l_1.itemset \rightarrow c_{l_1.pos}$) < minSup $\Rightarrow$ Sup($l_2.itemset \rightarrow c_{l_1.pos}$) < minSup. □

Fig. 1 Modified CAR-Miner algorithm for incremental mining

```
Modified-CAR-Miner(L_r)
  Begin
1.  CARs = ∅;
2.  for all l_i ∈ L_r.children do
    Begin
3.    GENERATE-CAR(l_i, S_U, minConf)
4.    P_i = ∅;
5.    for all l_j ∈ L_r.children, with j > i do
6.      if l_i.att ≠ l_j.att then
        Begin
7.        O.att = l_i.att ∪ l_j.att;
8.        O.values = l_i.values ∪ l_j.values;
9.        O.Obidset = l_i.Obidset ∩ l_j.Obidset;
10.       if |O.Obidset| = |l_i.Obidset| then
          Begin
11.         O.count = l_i.count;
12.         O.pos = l_i.pos;
          Endif
13.       else if |O.Obidset| = |l_j.Obidset| then
            Begin
14.           O.count = l_j.count;
15.           O.pos = l_j.pos;
            Endif
16.         Else
            Begin
17.           O.count = {count(x ∈ O.Obidset | class(x) = c_i, ∀i∈[1,k]};
18.           O.pos = arg max_{i∈[1,k]} {O.count_i};
19.           if O.count[O.pos] ≥ S_L × |D| then
20.             P_i = P_i ∪ O;
            End
        Endif
    Endfor
21.   Modified-CAR-Miner(P_i);
  End

GENERATE-CAR(l, S_U, minConf)
  Begin
22. if l.count[l.pos] ≥ S_U × |D| then
23.   conf = l.count[l.pos] /|l.Obidset|;
24.   if conf ≥ minConf then
25.     CARs = CARs ∪ {l.itemset → c_pos (l.count[l.pos], conf)};
  End
```

Based on Theorem 1, infrequent nodes from the MECR-tree are pruned to reduce updating time.

Input: The MECR-tree built from original dataset D in which L_r is the root node, inserted dataset D', two minimum support thresholds S_U and S_L, and *minConf*

Output: Class association rules that satisfy S_U and *minConf* from $D + D'$

Figure 2 shows the algorithm for updating the MECR-tree when dataset D' is inserted. Firstly, the algorithm checks whether the MECR-tree was created by considering the number of rows in the original dataset. If the number of rows is 0 (line 1), it means that the tree was not created, so **Modified-CAR-Miner** is called to create the MECR-tree for dataset D' (line 2) and the safety threshold f is computed using (1) (line 3). If the number of rows in dataset D' is larger than the safety threshold f, then the algorithm calls **Modified-CAR-Miner** to generate rules in the entire dataset $D + D'$ (lines 4 and 5), and then computes the safety threshold f based on the integrated dataset $D + D'$ (line 6). If the number of rows in dataset D' is not greater than f, then the algorithm simply updates the MECR-tree as follows. First, all Obidsets of nodes on the tree (line 8) are deleted to ensure that the algorithm works on the inserted dataset only. Second, the **UPDATE-TREE** procedure with root node L_r is called to update the information of nodes on the tree (line 9). Third, the procedure **GENERATE-RULES** with L_r is called to generate rules whose supports and confidences satisfy S_U and *minConf* (line 10). The

Fig. 2 Algorithm for updating the MECR-tree for an incremental dataset

Output: Class association rules that satisfy S_U and *minConf* from $D + D'$

CAR-Incre()
Begin
1. if $|D| = 0$ then
Begin
2. Call the procedure **Modified-CAR-Miner** to mine CARs in D' using S_L.
3. Compute $f = \left\lfloor \frac{(S_U - S_L) \times |D'|}{1 - S_U} \right\rfloor$
Endif
4. else if $|D'| > f$ then
Begin
5. Call the procedure **Modified-CAR-Miner** to mine CARs in $D + D'$ using S_L.
6. Compute $f = \left\lfloor \frac{(S_U - S_L) \times (|D| + |D'|)}{1 - S_U} \right\rfloor$
Endif
7. Else
Begin
8. Clear the *Obidset* of each node in the MECR-tree
9. Call the procedure **UDATE-TREE** to update the MECR-tree
10. Call the procedure **GENERATE-RULE** to generate CAR from the MECR-tree
11. $f = f - |D'|$
End
12. $D = D + D'$
End

UPDATE-TREE(L_r)
Begin
13. Update information of nodes in the first level of L_r including *Obidset*, *count*, and *pos* and mark them
14. for all $l_i \in L_r$.children do
15. if l_i is not marked then
16. **TRAVERSE-TREE-TO-CHECK**(l_i);
17. else if l_i.count[l_i.pos] < $S_L \times (|D| + |D'|)$ then
18. **DELETE-TREE**(l_i); // by theorem 1, delete l_i and its child nodes
19. else
20. for all $l_j \in L_r$.children, with j > i do
21. if l_i.att ≠ l_j.att and l_j is marked then
Begin
22. Let O be a direct child node of l_i and l_j;
23. if O has existed in the tree then
24. $O.Obidset = l_i.Obidset \cap l_j.Obidset$;
25. if $|O.Obidset| > 0$ then
Begin
26. Update $O.count$ based on $O.Obidset$;
27. $O.\text{pos} = \arg\max_{i \in [1,k]} \{O.count_i\}$;
28. if O.count[O.pos] ≥ $S_L \times (|D| + |D'|)$ then
29. Mark O;
Endif
Endif
30. **UPDATE-TREE**(l_i);
End

TRAVERSE-TREE-TO-CHECK(l)
Begin
31. if l.count[l.pos] < $S_L \times (|D| + |D'|)$ then
32. **DELETE-TREE**(l);
33. else
34. for all $l_i \in l$.children do
35. **TRAVERSE-TREE-TO-CHECK**(l_i);
End

DELETE-TREE(l)
Begin
36. for all $l_i \in l$.children do
37. **DELETE-TREE**(l_i);
38. delete l;
End

GENERATE-RULES(L_r, S_U, *minConf*)
Begin
39. for all $l \in L_r$.children do
Begin
40. if l.count[l.pos] ≥ $S_U \times |D|$ then
Begin
41. conf = l.count[l.pos] / $\sum_{i=1}^{k} l.count[i]$;
42. if conf ≥ *minConf* then
43. CARs = CARs ∪ {l.itemset → c_{pos} (l.count[l.pos], conf)};
Endif
44. **GENERATE-RULES**(l, S_U, *minConf*);
Endfor
End

Table 6 Inserted dataset

OID	*A*	*B*	*C*	class
9	a1	b1	c2	y

safety threshold f is reduced to f - $|D'|$ (line 11). Finally, the original dataset is supplemented by D' (line 12).

Consider procedure **UPDATE-TREE**. First, this procedure changes the information of nodes in the first level of the MECR-tree whose itemsets are contained in the inserted dataset and marks them (line 13). Line 15 checks whether each child node l_i of the root node L_r is unmarked (i.e., it is not changed from the original dataset). Its child nodes are then checked using Theorem 1 (line 16). If the support of l_i does not satisfy S_L, then l_i and all its child nodes are deleted by Theorem 1 (lines 17 and 18). Otherwise, l_i is marked and its support satisfies S_L. Then, information of *Obidset*, *count*, and *pos* of all child nodes of l_i is updated (lines 19-27). If the support of O satisfies S_L, then it is marked (lines 28 and 29). After all the child nodes of l_i have been checked, the procedure **UPDATE-TREE** is recursively called to update all child nodes of l_i (line 30).

Consider procedure **TRAVERSE-TREE-TO-CHECK**. This procedure checks whether the support of l satisfies

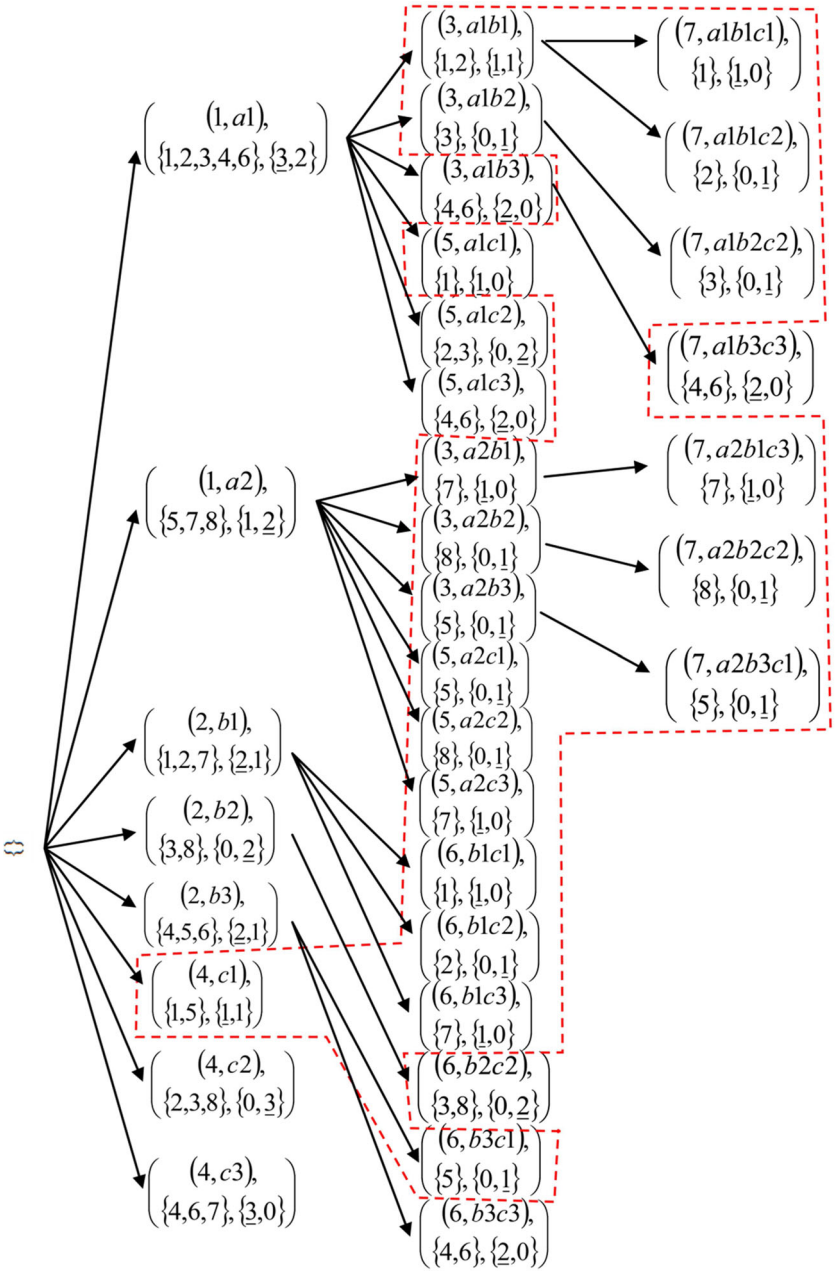


Fig. 3 Results of **Modified-CAR-Miner** with $S_U = 25\ \%$ and $S_L = 12.5\ \%$

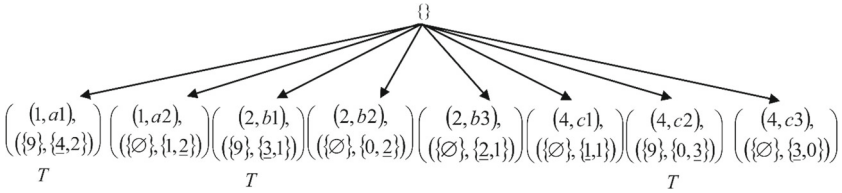


Fig. 4 Results of updating level 1 of the MECR-tree

S_L. If not, then l and all its child nodes are deleted using Theorem 1. Otherwise, the child nodes of l are checked in the same way as l and the procedure is called recursively until there are no more nodes. Procedure **DELETE-TREE** deletes this node with all its child nodes. Procedure **GENERATE-RULES** checks each child node l of the root node L_r to generate a rule r, if the support of r satisfies *minSup* (line 40), this procedure checks the confidence of r (line 42), if the confidence of r saisfies *minConf* then r is added into the set of rules (CARs). After that, this procedure is called recursively to generate all rules from the sub-tree l.

5 Example

Assume that the dataset in Table 1 is the original dataset and the inserted dataset has one row, as shown in Table 6.

With $S_U = 25\ \%$ and $S_L = 12.5\ \%$, the process of creating and updating the MECR-tree is illustrated as follows.

Figure 3 shows the results of **Modified-CAR-Miner** obtained using S_L for the dataset in Table 1. Because $25\ \% \times 8 = 2$ and $12.5\ \% \times 8 = 1$, nodes whose supports are greater than or equal to 2 are frequent and those whose supports are equal to 1 are pre-large. Consequently, nodes enclosed by the dashed line contain pre-large itemsets.

The safety threshold f is computed as follows: $f = \left\lfloor \frac{(0.25-0.125)\times 8}{1-0.25} \right\rfloor = 1$.

Consider the inserted dataset. Because the number of rows is 1, $|D'| = 1 \leq f = 1$, the algorithm updates the information of nodes in the tree without re-scanning original dataset D.

The process of updating the MECR-tree is as follows. The first level of the MECR-tree is updated. The results are shown in Fig. 4.

The new row (row 9) contains items (A, a1), (B, b1), and (C, c2), so only three nodes in the MECR-tree are changed (marked by T in Fig. 4).

- Consider node $l_i = \begin{pmatrix} (1, a1), \\ (\{9\}, \{\underline{4}, 2\}) \end{pmatrix}$. Because it has been changed, it needs to be checked with its following nodes (only changed nodes) in the same level to update information:
  - With node $l_j = \begin{pmatrix} (2, b1), \\ (\{9\}, \{\underline{3}, 1\}) \end{pmatrix}$, the node created from these two nodes (after update) is $\begin{pmatrix} (3, a1b1), \\ (\{9\}, \{\underline{2}, 1\}) \end{pmatrix}$. This node has *count*[*pos*] $= 2 \geq S_L \times (8+1) = 1.125$, so it is marked as a changed node.
  - With node $l_j = \begin{pmatrix} (4, c2), \\ (\{9\}, \{1, \underline{3}\}) \end{pmatrix}$, the node created from these two nodes (after update) is $\begin{pmatrix} (5, a1c2), \\ (\{9\}, \{1, \underline{2}\}) \end{pmatrix}$. This node has *count*[*pos*]

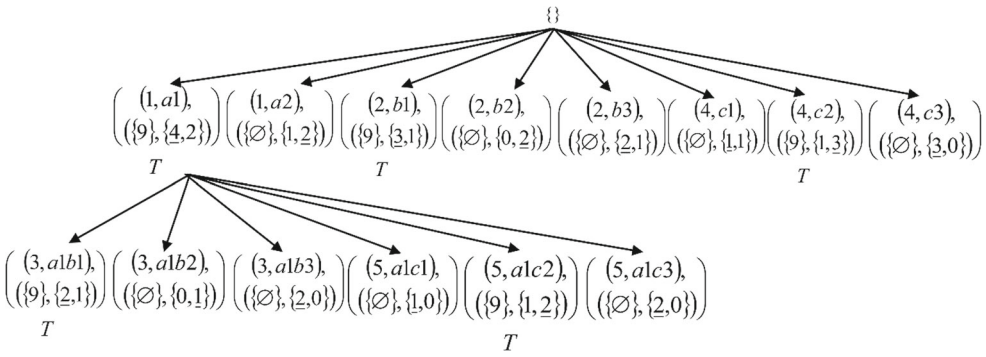
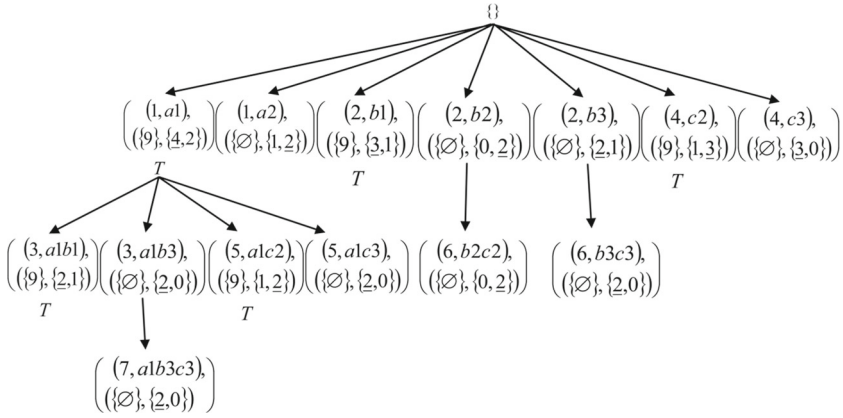


Fig. 5 Results obtained after considering node $\begin{pmatrix} (1,a1) \\ (\{9\},\{\underline{4},2\}) \end{pmatrix}$

Fig. 6 Updated MECR-tree



$= 2 \geq S_L \times(8+1) = 1.125$, so it is marked as a changed node.

Results obtained after considering node $\left(\begin{array}{c}(1,a1),\\(\{9\},\{\underline{4},2\})\end{array}\right)$ are shown in Fig. 5.

After considering node $\left(\begin{array}{c}(1,a1),\\(\{9\},\{\underline{4},2\})\end{array}\right)$, the algorithm is called recursively to update all its child nodes.

- Consider node $\left(\begin{array}{c}(3,a1b1),\\(\{9\},\{\underline{2},1\})\end{array}\right)$. Because $count[pos] \geq S_L \times(8+1)$, it will check with node $\left(\begin{array}{c}(5,a1c2),\\(\{9\},\{1,\underline{2}\})\end{array}\right)$. The node created from these two nodes (after update) is $\left(\begin{array}{c}(7,a1b1c2),\\(\{9\},\{\underline{1},1\})\end{array}\right)$. This node is deleted because $count[pos] = 1 < S_L \times(8+1)$. All its child nodes are deleted because their supports are smaller than $S_L \times(8+1)$.
- Consider node $\left(\begin{array}{c}(3,a1b2),\\(\{\emptyset\},\{0,\underline{1}\})\end{array}\right)$. Because $count[pos] < S_L \times(8+1)$, $\left(\begin{array}{c}(3,a1b2),\\(\{\emptyset\},\{0,\underline{1}\})\end{array}\right)$ is deleted. All its child nodes are also deleted by using Theorem 1.
  - Similarly, node $\left(\begin{array}{c}(5,a1c1),\\(\{\emptyset\},\{\underline{1},0\})\end{array}\right)$ is also deleted.
- Consider node $\left(\begin{array}{c}(1,a2),\\(\{\emptyset\},\{1,\underline{2}\})\end{array}\right)$. This node is not deleted. All its child nodes are deleted by checking support and using Theorem 1.

Do the same process for nodes $\left\{\left(\begin{array}{c}(2,b1),\\(\{9\},\{\underline{3},1\})\end{array}\right), \left(\begin{array}{c}(2,b2),\\(\{\emptyset\},\{0,\underline{2}\})\end{array}\right), \left(\begin{array}{c}(2,b3),\\(\{\emptyset\},\{\underline{2},1\})\end{array}\right), \left(\begin{array}{c}(4,c1),\\(\{\emptyset\},\{\underline{1},1\})\end{array}\right), \left(\begin{array}{c}(4,c2),\\(\{9\},\{1,\underline{3}\})\end{array}\right), \left(\begin{array}{c}(4,c3),\\(\{\emptyset\},\{\underline{3},0\})\end{array}\right)\right\}$, the MECR-tree after all updates is shown in Fig. 6.

The number of nodes in Fig. 6 is significantly less than that in Fig. 3 (14 versus 33). The MECR-tree can thus be efficiently updated.

Note that after the MECR-tree is updated, the safety threshold f is decreased by $1 \Rightarrow f = 0$, which means that if a new dataset is inserted, then the algorithm rebuilds the MECR-tree for the original and inserted datasets. $D = D + D'$ includes nine rows.

Table 7 Characteristics of experimental datasets

Dataset	# of attributes	# of classes	# of distinct values	# of itemsets
Breast	11	2	737	699
German	21	2	1,077	1,000
Led7	8	10	24	3,200
Vehicle	19	4	1,434	846
Lymph	18	4	63	148
Chess	37	2	75	3,196
Poker-hand	11	10	95	1,000,000

6 Experimental results

Experiments were conducted on a computer with an Intel Core i3 2.53-GHz CPU and 2 GB of RAM running Windows 7. Algorithms were coded in C# 2010.

Experimental datasets were obtained from the UCI Machine Learning Repository (http://mlearn.ics.uci.edu). Table 7 shows the characteristics of the experimental datasets.

The experimental results from Figs. 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, to 20 show that CAR-Incre is more efficient than CAR-Miner in most cases, especially in large datasets or large *minSup*. Examples are Poker-hand (a large number of records) or Chess (*minSup* is large).

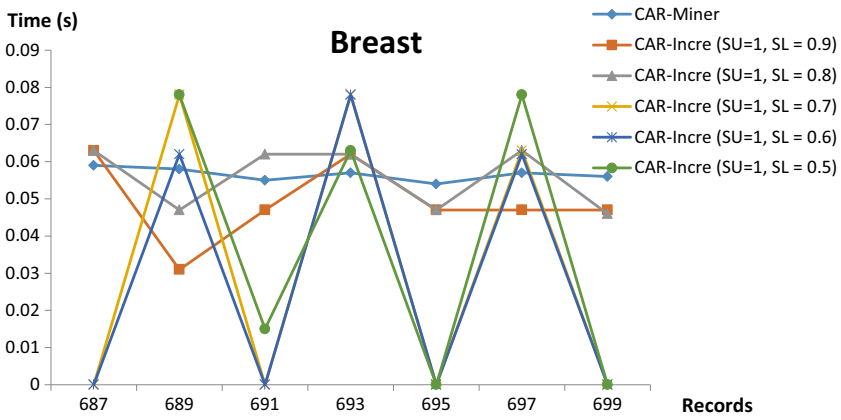


Fig. 7 Run times for CAR-Miner and CAR-Incre for Breast dataset ($S_U = 1$ %,$S_L =$ 0.9 %, 0.8 %, 0.7 %, 0.6 %, 0.5 %) for each inserted dataset (two records for each insert)

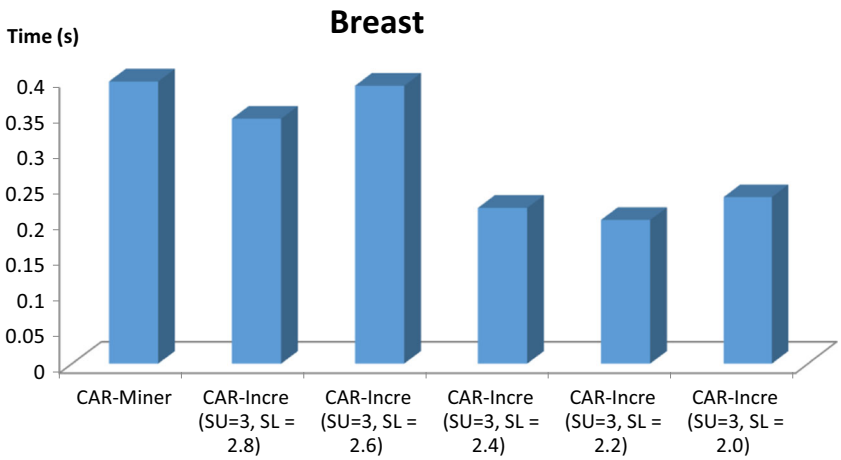


Fig. 8 Total runtime for CAR-Miner and CAR-Incre for Breast dataset ($S_U = 1$ %,$S_L = 0.9$ %, 0.8 %, 0.7 %, 0.6 %, 0.5 %)

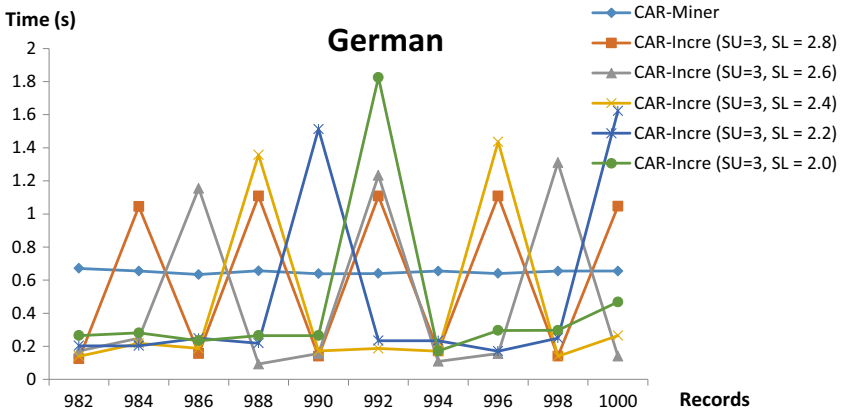


Fig. 9 Run times for CAR-Miner and CAR-Incre for German dataset ($S_U = 3$ %,$S_L =$ 2.8 %; 2.6 %; 2.4 %; 2.2 %; 2.0 %) for each inserted dataset (two rows for each insert)

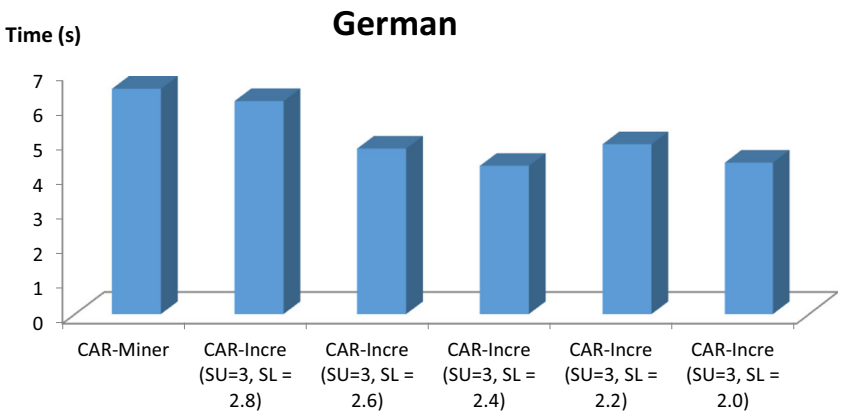


Fig. 10 Total runtime for CAR-Miner and CAR-Incre for German dataset (S_U = 3 %,S_L = 2.8 %; 2.6 %; 2.4 %; 2.2 %; 2.0 %)

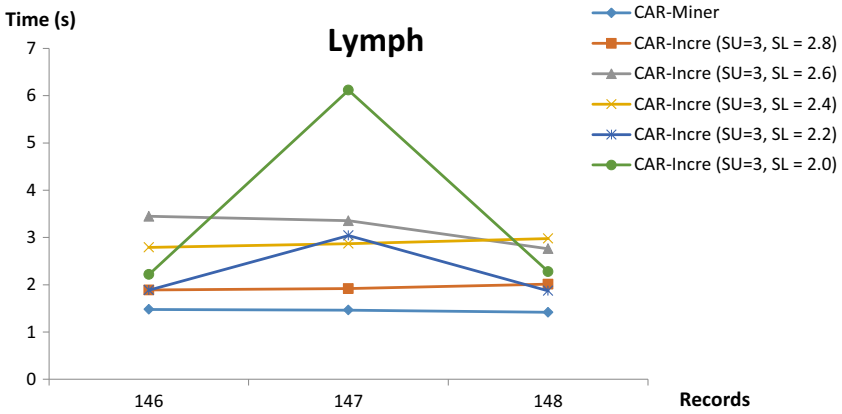


Fig. 11 Run times for CAR-Miner and CAR-Incre for Lymph dataset (S_U = 3%,S_L = 2.8 %; 2.6 %; 2.4 %; 2.2 %; 2.0 %) for each inserted dataset (one record for each insert)

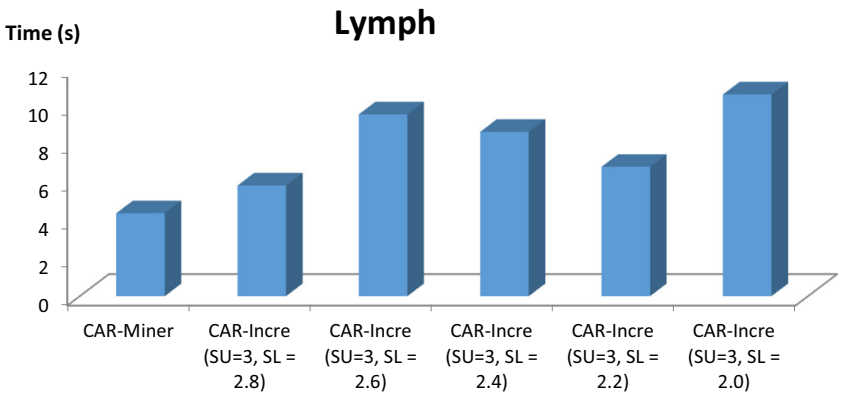


Fig. 12 Total runtime for CAR-Miner and CAR-Incre for Lymph dataset (S_U = 3 %, S_L = 2.8 %; 2.6 %; 2.4 %; 2.2 %; 2.0 %)

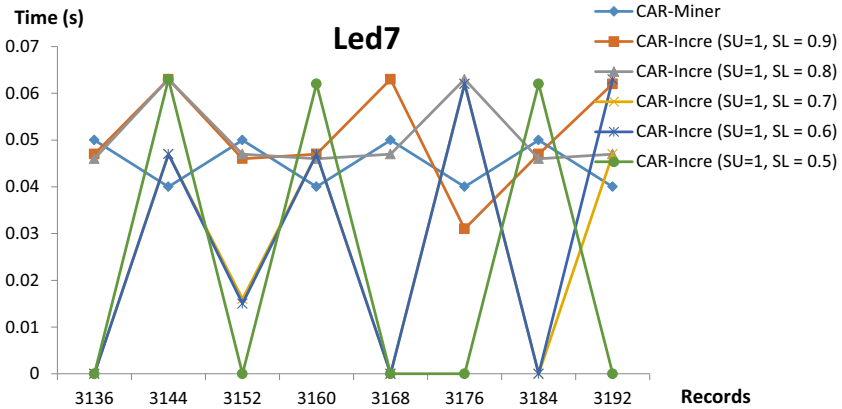


Fig. 13 Run time for CAR-Miner and CAR-Incre for Led7 dataset (S_U = 1 %,S_L = 0.9 %; 0.8 %; 0.7 %; 0.6 %; 0.5 %) for each inserted dataset (two records for each insert)

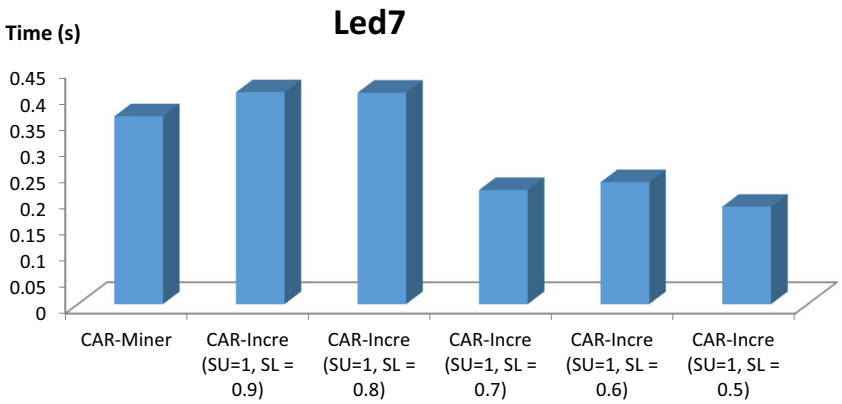


Fig. 14 Total runtime for CAR-Miner and CAR-Incre for Led7 dataset ($S_U = 1$ %, $S_L = 0.9$ %; 0.8 %; 0.7 %; 0.6 %; 0.5 %)

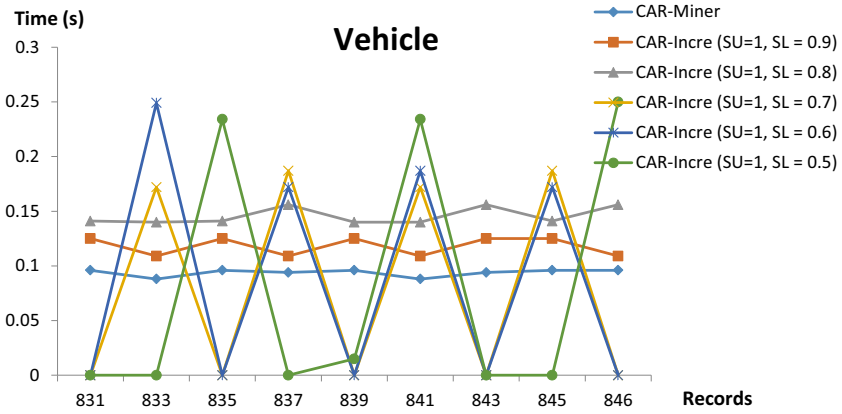


Fig. 15 Run times for CAR-Miner and CAR-Incre for Vehicle dataset ($S_U = 1$ %,$S_L = 0.9$ %; 0.8 %; 0.7 %; 0.6 %; 0.5 %) for each inserted dataset (two records for each insert)

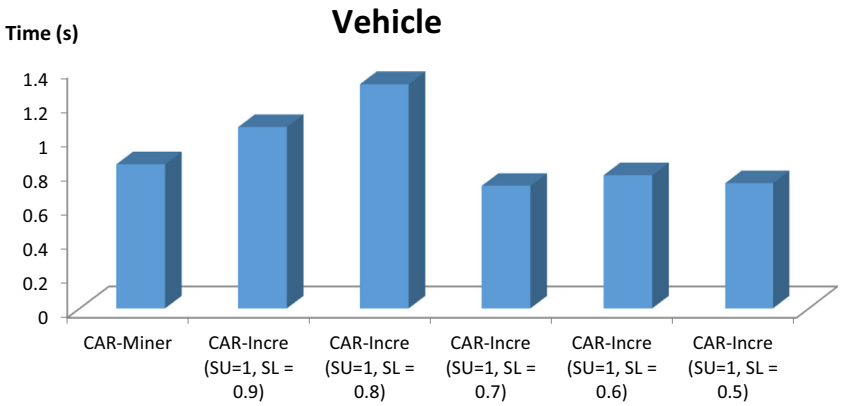


Fig. 16 Total runtime for CAR-Miner and CAR-Incre for Vehicle dataset ($S_U = 1$ %,$S_L = 0.9$ %; 0.8 %; 0.7 %; 0.6 %; 0.5 %)

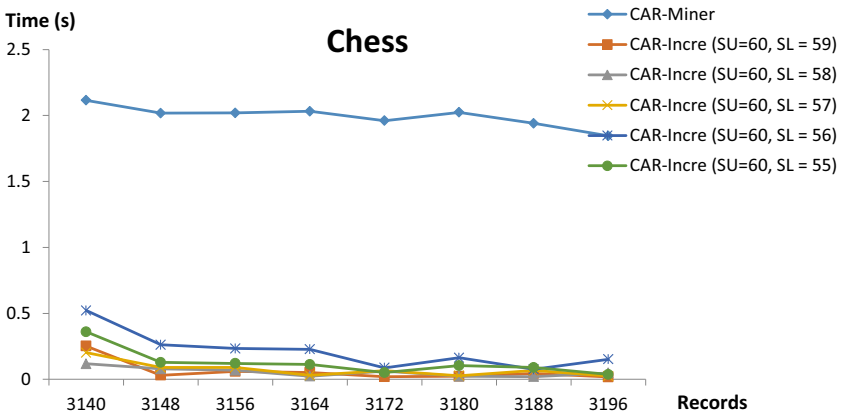


Fig. 17 Run times for CAR-Miner and CAR-Incre for Chess dataset ($S_U = 60$ %,$S_L = 59$ %; 58 %; 57 %; 56 %; 55 %) for each inserted dataset (8 records for each insert)

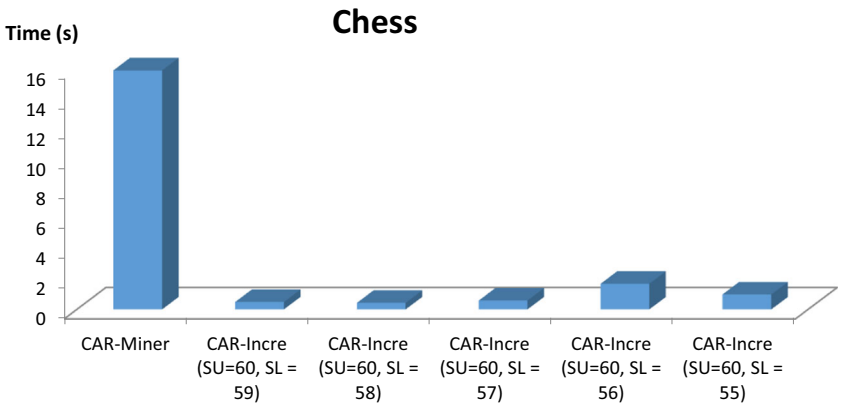


Fig. 18 Total runtime for CAR-Miner and CAR-Incre for Chess dataset ($S_U = 60\ \%, S_L = 59\ \%; 58\ \%; 57\ \%; 56\ \%; 55\ \%$)

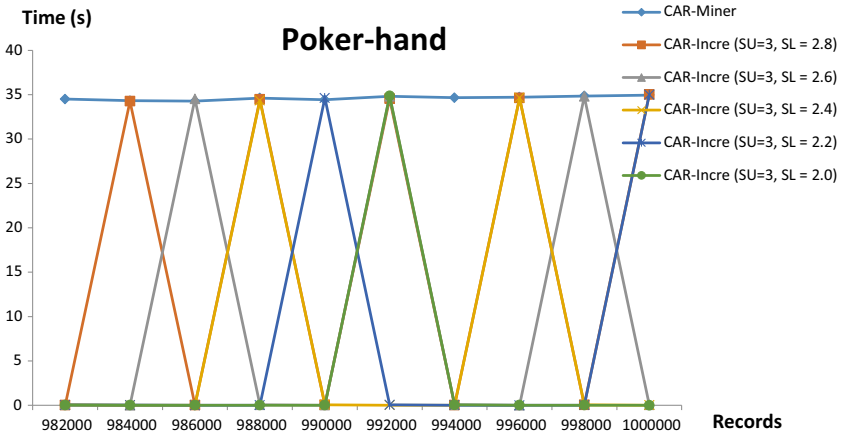


Fig. 19 Run time for CAR-Miner and CAR-Incre for Poker-hand dataset ($S_U = 3\ \%, S_L = 2.8\ \%; 2.6\ \%; 2.4\ \%; 2.2\ \%; 2.0\ \%$) for each inserted dataset (2,000 records for each insert)

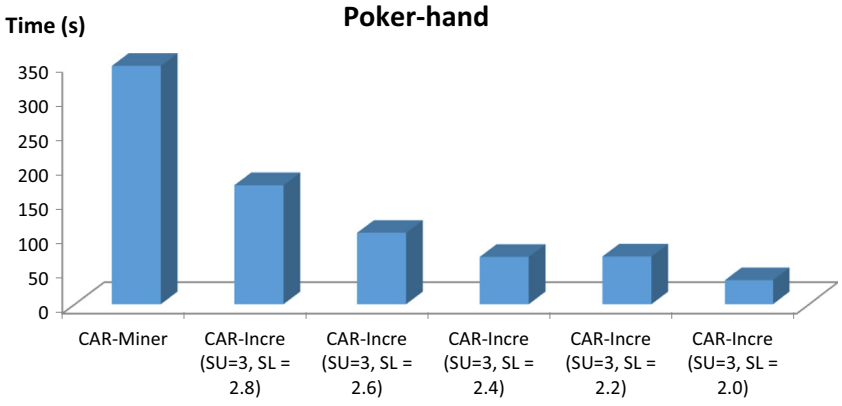


Fig. 20 Total runtime for CAR-Miner and CAR-Incre for Poker-hand dataset ($S_U = 3\ \%, S_L = 2.8\ \%; 2.6\ \%; 2.4\ \%; 2.2\ \%; 2.0\ \%$)

6.1 The impact of number of records

CAR-Incre is very efficient when the number of records in the original dataset is large. For example, consider the Poker-hand dataset. Updating 2,000 data rows on the MECR-tree built from 980,000 rows of the original dataset takes about 0.09 seconds, and mining rules using the batch process on 982,000 rows takes about 34 seconds (Fig. 19). Figure 20 shows a comparison of total run time of two algorithms (CAR-Miner and CAR-Incre) in some S_L ($S_U = 3$).

However, when we compare the run times of two algorithms in a small dataset such as Lymph, we can see that CAR-Miner is faster than CAR-Incre with all thresholds.

6.2 The impact of S_{L}

The most important issue in CAR-Incre is how to choose a suitable S_{L}. If S_{L} is large then f must be small. In this case, the algorithm needs to rescan the original dataset many times, which is time-consuming. If S_{L} is small, many frequent and pre-large itemsets are generated and the tree must be updated. This is also very time-consuming. To the best of our knowledge, there is not any method for choosing a suitable S_{L} value. Therefore, we conducted experiments with different S_{L} values to determine the influence of S_{L} values on the run time.

Consider Breast dataset with $S_U = 1$. The total time of 7 runs of CAR-Miner is 0.369s. We change $S_L = \{0.9, 0.8, 0.7, 0.6, 0.5\}$ and the run times are {0.344, 0.390, 0.219, 0.202, 0.234}respectively, the best threshold is 0.6.

Consider German dataset with $S_U = 3$. The total time of 10 runs of CAR-Miner is 6.5s. We change $S_L = \{2.8, 2.6, 2.4, 2.2, 2.0\}$and the run times are {6.147, 4.774, 4.274, 4.898, 4.368}respectively, the best threshold is 2.4.

Similarly, the best threshold of Led7 is 0.5 ($S_U = 1$), that of Vehicle is 0.7 ($S_U = 1$), that of Chess is 59 ($S_U = 60$), that of Poker-hand is 2.0 ($S_U = 3$).

6.3 The impact of *minSup*

The safety threshold f is proportional to *minSup* (S_U). If *minSup* is large, safety threshold f is also large. Therefore, the number of inserted records is small, we do not need to rescan the original dataset; we update only information of nodes on the tree with new data. For example, consider Chess dataset with $S_U = 60$ %, and $S_L = 59$ %. The original dataset is inserted eight times with eight rows each time but the safety threshold f is still satisfied ($f = ((0.6 - 0.59) \times 3132)/(1 - 0.6) = 78$ records).

7 Conclusions and future work

This paper proposed a method for mining CARs from incremental datasets. The proposed method has several advantages:

- The MECR-tree structure is used to generate rules quickly.
- The concept of pre-large itemsets is applied to CAR mining to reduce the number of re-scans on the original dataset.
- A theorem for quickly pruning infrequent nodes in the tree is developed to improve the process of updating the tree.

One of weaknesses of the proposed method is that it must rebuild the MECR-tree for the original and inserted datasets when the number of rows in the inserted dataset is larger than the safety threshold f. This approach is not appropriate for large original datasets. Thus, the algorithm is being improved to avoid re-scanning the original dataset. In addition, a lattice structure helps to identify redundant rules quickly. It is possible to update the lattice when a dataset is inserted will thus be studied in the future.

References

1. Antonie ML, Zaïane OR (2004) An associative classifier based on positive and negative rules. In: Proceedings of the 9th ACM SIGMOD workshop on research issues in data mining and knowledge discovery, Paris, pp 64–69
2. Cheung DW, Han J, Ng VT, Wong CY (1996) Maintenance of discovered association rules in large databases:An incremental updating approach. In: Proceedings of the twelfth IEEE international conference on data engineering, New Orleans, pp 106–114
3. Duong TH, Nguyen NT, Jo GS (2010) Constructing and mining a semantic-based academic social network. J Intell Fuzzy Syst 21(3):197–207
4. Grahne G, Zhu J (2005) Fast algorithms for frequent itemset mining using fptrees. IEEE Trans Knowl Data Eng 17(10):1347–1362
5. Hong TP, Wang CY, Tao YH (2001) A new incremental data mining algorithm using pre-large itemsets. Int Data Anal 5(2):111–129
6. La PT, Le B, Vo B (2014) Incrementally building frequent closed itemset lattice. Expert Syst Appl 41(6):2703–2712
7. Le TP, Hong TP, Vo B, Le B (2011) Incremental mining frequent itemsets based on the trie structure and the prelarge itemsets. In: Proceedings of the 2011 IEEE international conference on granular computing, Kaohsiung, pp 369-373
8. Le TP, Hong TP, Vo B, Le B (2012) An efficient incremental mining approach based on IT-tree. In: Proceedings of the 2012 IEEE

international conference on computing & communication technologies, research, innovation, and vision for the future, Ho Chi Minh, pp 57–61
9. Lee MS, Oh S (2014) Alternating decision tree algorithm for assessing protein interaction reliability. Vietnam J Comput Sci 1(3):169–178
10. Li W, Han J, Pei J (2001) CMAR:Accurate and efficient classification based on multiple class-association rules. In: Proceedings of the 1st IEEE international conference on data mining, San Jose, pp 369–376
11. Lin CW, Hong TP (2009) The Pre-FUFP algorithm for incremental mining. Expert Syst Appl 36(5):9498–9505
12. Liu B, Hsu W, Ma Y (1998) Integrating classification and association rule mining. In: Proceedings of the 4th international conference on knowledge discovery and data mining, New York, pp 80–86
13. Lucchese B, Orlando S, Perego R (2006) Fast and memory efficient mining of frequent closed itemsets. IEEE Trans Knowl Data Eng 18(1):21–36
14. Nguyen NT (2000) Using consensus methods for solving conflicts of data in distributed systems. In: Proceedings of SOFSEM 2000, Lecture Notes in Computer Science 1963, pp 411–419
15. Nguyen TTL, Vo B, Hong TP, Thanh H. C (2012) Classification based on association rules: a lattice-based approach. Expert Syst Appl 39(13):11357–11366
16. Nguyen TTL, Vo B, Hong TP, Thanh H. C (2013) CAR-Miner: an efficient algorithm for mining class-association rules. Expert Syst Appl 40(6):2305–2311
17. Nguyen D, Vo B, Le B (2014) Efficient strategies for parallel mining class association rules. Expert Syst Appl 41(10):4716–4729
18. Pei J, Han J, Mao R (2000) CLOSET: An efficient algorithm for mining frequent closed itemsets. In Proceedings of the 5th ACM-SIGMOD workshop on research issues in data mining and knowledge discovery, pp. 11–20
19. Pham TT, Luo J, Hong TP, Vo B (2014) An efficient method for mining non-redundant sequential rules using attributed prefix-trees. Eng Appl Artif Intell 32:88–99
20. Thabtah F, Cowling P, Peng Y (2004) MMAC: a new multi-class, multi-label associative classification approach. In: Proceedings of the 4th IEEE international conference on data mining, Brighton, pp 217–224
21. Van TT, Vo B, Le B (2014) IMSR_PreTree: an improved algorithm for mining sequential rules based on the prefix-tree. Vietnam J Comput Sci 1(2):97–105
22. Veloso A, Meira Jr. W, Goncalves M, Almeida HM, Zaki MJ (2011) Calibrated lazy associative classification. Inf Sci 181(13):2656–2670
23. Vo B, Le B (2008) A novel classification algorithm based on association rule mining. In: Proceedings of the 2008 pacific rim knowledge acquisition workshop (Held with PRICAI'08), LNAI 5465, Ha Noi, pp 61–75
24. Vo B, Hong TP, Le B (2013) A lattice-based approach for mining most generalization association rules. Knowl-Based Syst 45:20–30
25. Vo B, Le T, Hong TP, Le B (2014) An effective approach for maintenance of pre-large-based frequent-itemset lattice in incremental mining. Appl Intell 41(3):759–775
26. Yen SJ, Lee YS, Wang CK (2014) An efficient algorithm for incrementally mining frequent closed itemsets. Appl Intell 40(4):649–668
27. Yin X, Han J (2003) CPAR: Classification based on predictive association rules. In: Proceedings of SIAM international conference on data mining (SDM'03), San Francisco, pp 331–335

Loan T. T. Nguyen is currently a PhD Student at Institute of Informatics, Wroclaw University of Technology, Poland. She received BSc degree in 2002 from University of Science, Vietnam National University of Ho Chi Minh, Vietnam and MSc in 2007 degree from the University of Information Technology, National University of Ho Chi Minh, Vietnam. Her research interests include association rules, classification, and incremental mining.

Ngoc Thanh Nguyen is currently a full professor of Wroclaw University of Technology, Poland, and is the chair of Information Systems Department in the Faculty of Computer Science and Management. His scientific interests consist of knowledge integration methods, intelligent technologies for conflict resolution, inconsistent knowledge processing, multiagent systems, collective intelligence and E-learning methods. He has edited 20 special issues in international journals and 10 conference proceedings. He is the author or editor of 14 books and more than 200 other publications. He is the Editor-in-Chief of two international journals: "LNCS Transactions on computational Collective Intelligence" and "International Journal of Intelligent Information and Database". He is the Associate Editor of 4 prestigious international journals and a member of Editorial Boards of several other prestigious international journals. He is a Senior Member of IEEE. Prof. Nguyen has been nominated by ACM of title "Distinguished Scientist".