CAR-Miner: An Efficient Algorithm for Mining
Class-association Rules

Abstract. Building a high accuracy classifier for classification is a problem in real applications. One high accuracy classifier used for this purpose is based on association rules. In the past, some researches showed that classification based on association rules (or class-association rules – CARs) has higher accuracy than that of other rule-based methods, such as ILA and C4.5. However, mining CARs consumes more time than methods based on ILA or C4.5 because it mines a complete rule set. Therefore, improving the execution time for mining CARs is one of the main problems with this method that needs to be solved. In this paper, we propose a new method for mining class-association rule using a tree structure. Firstly, we design a tree structure for the storage of frequent itemsets of datasets. Some theorems for pruning nodes and computing information in the tree are developed after that, and then, based on the theorems, we propose an efficient algorithm for mining CARs. Experimental results show that our approach is more efficient than those used previously.

Keywords: accuracy, classification, class-association rules, data mining, tree structure.

1 Introduction

Classification plays an important role in decision support systems. A lot of methods for mining classification rules have been developed in recent years, including C4.5 and ILA. These methods
are, however, based on heuristics and greedy approaches to generate rule sets that may be either
too general or too overfitting for a given dataset, thus often yielding high error ratios. A new
method for classification from data mining, called Classification Based on Associations (CBA),
has thus been proposed for mining Class-Association Rules (CARs). This method can easily
remove noise and gets higher accuracy. It generates a more complete rule set than C4.5 and ILA.
For association-rule mining, the target attribute (or class attribute) is not pre-determined.
However, the target attribute must be defined in classification problems. Thus, some algorithms
for mining classification rules based on association-rule mining have been proposed. Examples
include classification based on redtive association rules (Yin and Han, 2003), classification
based on multiple association rules (Li et al., 2001), classification based on associations (CBA,
Liu et al., 1998), multi-class, multi-label associative classification (Thabtah et al., 2004), multi-
class classification based on association rules (Thabtah et al., 2005), associative classifier based
on maximum entropy (Thonangi and Pudi, 2005), Noah (Giuffrida et al., 2000), and the use of
the equivalence classrule tree (Vo and Le, 2008). Some researches have also reported that
classifiers based on classassociation rules are more accurate than some traditional methods such
as C4.5 (Quinlan, 1992) and ILA (Tolun et al., 1998; Tolun et al., 1999) both theoretically
(Veloso et al., 2006) and with regard to experimental results (Liu et al., 1998). Veloso et al.
proposed the lazy associative classification (Veloso et al., 2006; Veloso et al., 2007; Veloso et
al., 2011), which differed from CARs in that it used rules mined from the projected dataset of an
unknown object for predicting the class instead of using the ones mined from the whole dataset.
Genetic algorithms have also been applied recently for mining CARs, and several approaches
have been proposed. For example, Chien and Chen (2010) proposed a GA-based approach to
build the classifier for numeric datasets and applied it to stock trading data. Kaya (2010)
proposed a Pareto-optimal genetic approach for building autonomous classifiers. Qodmanan et al. (2011) proposed a GA-based mining method without requiring minimum support or minimum confidence thresholds. Yang et al. (2011) proposed an evolutionary approach to rank class-association rules. These algorithms were mainly based on heuristics in order to build classifiers. All the above methods focused on the design of the algorithms for mining CARs or for building classifiers, but did not discuss much with regard to their mining time. Therefore in this paper, we aim to propose an efficient algorithm for mining CARs based on a tree structure.

In the past, Vo and Le (2008) proposed a method for mining CARs using the ECR-tree (Equivalence Class-Rule tree). An efficient mining algorithm, named ECR-CARM, was proposed in their study. ECR-CARM scanned the dataset only once and was based on object identifiers to quickly determine the support of itemsets. However, it was quite time-consuming for generating and testing candidates because all values with the same attributes are grouped into one node in the tree. When joining two nodes \( l_i \) and \( l_j \) to create a new node, ECR-CARM had to consider each element of \( l_i \) with each element of \( l_j \) to check whether they had the same prefix or not. Nguyen et al. (2012) then proposed an algorithm for pruning redundant rules using the lattice structure. In the method of using multiple rules for prediction, we need mine all CARs for this purpose. In this paper, we modify the ECR-tree to mine CARs in a more efficient way. Each node in the tree contains only a value, instead of all the values, of an attribute. With this tree, some theorems are also designed and based on them, an algorithm is proposed for mining CARs.

The rest of this paper is organized as follows. In Section 2, we introduce some works related to mining CARs. Section 3 presents preliminary concepts. The main contributions are presented in Section 4, in which a modification data structure for the ECR-tree is developed and some theorems for pruning candidates fast are derived. Based on the tree and these theorems, an
algorithm is also proposed for mining CARs efficiently in that section. In Section 5, we show and discuss the experimental results. The conclusions and future work are presented in Section 6.

2 Related work

Mining CARs is discovery of all classification rules that satisfy the minimum support ($\text{minSup}$) and the minimum confidence ($\text{minConf}$) thresholds. The first method for mining CARs was proposed by Liu et al. (1998). It generates all candidate 1-ruleitems and then calculates their supports for finding frequent ruleitems that satisfy $\text{minSup}$. It then generates all the candidate 2-ruleitems from the frequent 1-ruleitems by the same checking way. The above procedure is then repeated until no more candidates can be obtained. The authors also proposed a heuristic for building the classifier. That approach, however, generates a lot of candidates and scans the dataset many times, thus quite time-consuming. Therefore, the proposed algorithm uses a threshold $K$ and only generates $k$-ruleitems with $k \leq K$. In 2000, an improved algorithm for solving the problem of imbalanced datasets was proposed (Liu et al., 2000). The latter has higher accuracy than the former because it uses a hybrid approach for prediction.

Li et al. then proposed a method based on the FP-tree (Li et al., 2001). The advantage of this method is that it scans the dataset only two times and uses an FP-tree to compress the dataset. It also uses the tree-projection technique to find frequent itemsets. To predict unseen data, this method finds all rules that satisfy this data and uses a weighted $\chi^2$ measure to determine the class.

Vo and Le then proposed another approach based on the ECR-tree (Vo and Le, 2008). This approach develops a tree structure called the ECR-tree (Equivalence Class- Rules tree), and proposes an algorithm called ECR-CARM for mining CARs. The algorithm scans the dataset only once. It is based on the intersection of object identifications to quickly compute the supports.
of itemsets. Nguyen et al. (2012) then proposed a new method for pruning redundant rules based on a lattice. Thabtah et al. (2004) proposed a multi-class, multi-label associative classification approach for mining CARs. This method used the rule form \((A_{i1}, a_{i1}), (A_{i2}, a_{i2}), \ldots, (A_{im}, a_{im})\) \(\rightarrow c_{i1} \lor c_{i2} \lor \ldots \lor c_{il}\), where \(a_{ij}\) is a value of attribute \(A_{ij}\), and \(c_{ij}\) is a class label.

Some other classification association rule mining approaches have been presented in the work of Coenen et al. (2007), Guiffrida et al. (2000), Lim and Lee (2010), Liu et al. (2008), Priss (2002), Sun et al. (2006), Thabtah et al. (2005), Thabtah et al. (2006), Thonangi and Pudi (2005), Yin and Han (2003), Zhang et al. (2011), and Zhao et al. (2010).

3 Preliminary concepts

Let \(D\) be the set of training data with \(n\) attributes \(A_1, A_2, \ldots, A_n\) and \(|D|\) objects (cases). Let \(C = \{c_1, c_2, \ldots, c_k\}\) be a list of class labels. A specific value of an attribute \(A_i\) and a class \(C\) are denoted by the lower-case letters \(a\) and \(c\), respectively.

**Definition 1:** An itemset is a set of some pairs of attributes and specific values, denoted \(\{(A_{i1}, a_{i1}), (A_{i2}, a_{i2}), \ldots, (A_{im}, a_{im})\}\).

**Definition 2:** A class-association rule \(r\) is of the form \(\{(A_{i1}, a_{i1}), \ldots, (A_{im}, a_{im})\} \rightarrow c\), where the condition \(\{(A_{i1}, a_{i1}), \ldots, (A_{im}, a_{im})\}\) is an itemset, and the conclusion \(c \in C\) is a class label.

**Definition 3:** The actual occurrence \(ActOcc(r)\) of a rule \(r\) in \(D\) is the number of rows (transactions) of \(D\) that match \(r\)’s condition.

**Definition 4:** The support of a rule \(r\), denoted \(Sup(r)\), is the number of rows (transactions) of \(D\) that match \(r\)’s condition and belong to \(r\)’s class.

<table>
<thead>
<tr>
<th>OID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>y</td>
</tr>
</tbody>
</table>
For example, consider the rule $r$: {$A, a1$} $\rightarrow$ $y$ from the dataset in Table 1. We have $ActOcc(r) = 3$ and $Sup(r) = 2$ because there are three objects with $A = a1$, and two objects among them have the same class $y$ as the rule.

4 Mining Class-Association Rules

4.1. Tree structure

In this paper, we modify the ECR-tree structure (Vo and Le, 2008) into the MECR-tree structure (M stands for Modification) as follows. In the ECR-tree, all itemsets with the same attributes were arranged into one group and put in one node. Itemsets in different groups were then joined together to form itemsets with more items. This led to the consumption of much time for generating and checking itemsets. In our work, each node in the tree contains only one itemset along with the following information:

a) $Obidset$: a set of object identifiers that contain the itemset.

b) $(c_1, c_2, \ldots, c_k)$ – a list of integers, where $c_i$ is the number of records in $Obidset$ which belong to class $c_i$, and

c) $pos$ – a positive integer storing the position of the class with the maximum count, i.e., $pos = \arg \max_{i \in [1,k]} \{c_i\}$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>a1</td>
<td>b2</td>
<td>c1</td>
<td>n</td>
</tr>
<tr>
<td>3</td>
<td>a2</td>
<td>b2</td>
<td>c1</td>
<td>n</td>
</tr>
<tr>
<td>4</td>
<td>a3</td>
<td>b3</td>
<td>c1</td>
<td>y</td>
</tr>
<tr>
<td>5</td>
<td>a3</td>
<td>b1</td>
<td>c2</td>
<td>n</td>
</tr>
<tr>
<td>6</td>
<td>a3</td>
<td>b3</td>
<td>c1</td>
<td>y</td>
</tr>
<tr>
<td>7</td>
<td>a1</td>
<td>b3</td>
<td>c2</td>
<td>y</td>
</tr>
<tr>
<td>8</td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>n</td>
</tr>
</tbody>
</table>
In the ECR-tree, the authors did not store \( c_i \) and \( pos \), thus needing to compute them for all nodes. However, these values are not calculated in the proposed approach here with the MECR-tree by using theorems presented in Section 4.2.

For example, consider a node containing the itemset \( X = \{(A, a3), (B, b3)\} \) from Table 1. Because \( X \) is contained in objects 4 and 6, all of them belong to class \( y \). Therefore, a node \( \{(A,a3),(B,b3)\} \) or more simply as \( 3 \times a3b3 \) is generated in the tree. The pos is 1 (underlined at position 1 of this node) because the count of class \( y \) is at a maximum (2 as compared to 0). The latter is another representation of the former for saving memory when we use the tree structure to store itemsets. We use bit presentation for storage of the itemset’s attributes. For example, \( AB \) can present as 11 in bit presentation, and therefore, the value of these attributes is 3. With this presentation, we can use bitwise operations to make itemsets join faster.

4.2. Proposed algorithm

In this section, some theorems for fast mining CARs are designed. Based on these theorems, we propose an efficient algorithm for mining CARs.

**Theorem 1:** Given two nodes \( att \times values_1 \) and \( att \times values_2 \), if \( att_1 = att_2 \) and \( values_1 \neq values_2 \), then \( Obidset_1 \cap Obidset_2 = \emptyset \).

**Proof:** Since \( att_1 = att_2 \) and \( values_1 \neq values_2 \), there exist a \( val_1 \in values_1 \) and a \( val_2 \in values_2 \) such that \( val_1 \) and \( val_2 \) have the same attributes but different values. Thus, if a record with \( OID_i \) contains \( val_1 \), it cannot contain \( val_2 \). Therefore, \( \forall OID \in Obidset_1 \), and it can be inferred that \( OID \notin Obidset_2 \). Thus, \( Obidset_1 \cap Obidset_2 = \emptyset \).

In this theorem, we divide the itemset into form \( att\times values \) for ease of use. Theorem 1 infers
that if two itemsets $X$ and $Y$ have the same attributes, they don’t need to be combined into the itemset $XY$ because $\text{Supp}(XY) = 0$. For example, consider the two nodes $\frac{1 \times a_1}{127(2,1)}$ and $\frac{1 \times a_2}{38(1,1)}$, in which $\text{Obidset}((\{A, a1\})) = 127$, and $\text{Obidset}((\{A, a2\})) = 38$. $\text{Obidset}((\{A, a1\})) \cap \text{Obidset}((\{A, a2\})) = \emptyset$. Similarly, $\text{Obidset}((\{A, a1\}; (B, b1))) = 1$, and $\text{Obidset}((\{A, a1\}; (B, b2))) = 2$. It can be inferred that $\text{Obidset}((\{A, a1\}; (B, b1))) \cap \text{Obidset}((\{A, a1\}; (B, b2))) = \emptyset$ because both of these two itemsets have the same attributes $AB$ but with different values.

**Theorem 2:** Given two nodes $\text{Obidset}_i(c_{i1}, ..., c_{ik})$ and $\text{Obidset}_2(c_{21}, ..., c_{2k})$, if itemset$_1 \subseteq$ itemset$_2$ and $|\text{Obidset}_1| = |\text{Obidset}_2|$, then $\forall i \in [1,k]: c_{1i} = c_{2i}$.

**Proof:** We have itemset$_1 \subseteq$ itemset$_2$. This means that all records containing itemset$_2$ also contain itemset$_1$, and therefore, $\text{Obidset}_2 \subseteq \text{Obidset}_1$. Additionally, according to theory, we have $|\text{Obidset}_1| = |\text{Obidset}_2|$. This means that we have $\text{Obidset}_2 = \text{Obidset}_1$, or $\forall i \in [1,k]: c_{1i} = c_{2i}$.

From Theorem 2, when we join two parent nodes into a child node, then the itemset of the child node is always a superset of the itemset of each of the parent nodes. Therefore, we will check their cardimations, and if they are the same, we need not compute the count for each class and the pos of this node because they are the same as the parent node.

Using these theorems, we develop an algorithm for mining CARs efficiently. By theorem 1, we need not join two nodes with the same attributes, and by theorem 2, we need not compute the information for some child nodes.
First of all, the root node of the tree ($L_r$) contains child nodes such that each node contains a single frequent itemset. After that, procedure CAR-Miner will be called with the parameter $L_r$ to mine all CARs from the dataset $D$.

The CAR-Miner procedure (Figure 1) considers each node $l_i$ with all the other node $l_j$ in $L_r$, with $j > i$ (Lines 2 and 5) to generate a candidate child node $l$. With each pair $(l_i, l_j)$, the algorithm checks whether $l_i.att \neq l_j.att$ or not (Line 6, using Theorem 1). If they are different, it computes
the three elements *att, values, Obidset* for the new node *O* (Lines 7-9). Line 10 checks if the number of object identifiers of *l* is equal to the number of object identifiers of *O* (by Theorem 2). If this is true, then, by Theorem 2, the algorithm can copy all information from node *l* to node *O* (Lines 11-12). Similarly, in the event that the result of Line 10 is false, the algorithm checks *l* with *O*, and if the numbers of their object identifiers are the same (Line 13), the algorithm can copy all information from node *l* to node *O* (Lines 14-15). Otherwise, the algorithm computes the *O*.count by using *O*.Obidset and *O*.pos (Lines 17-18). After computing all of the information for node *O*, the algorithm adds it to *P* ( *P* is initialized empty in Line 4) if *O*.count[*O*.pos] ≥ *minSup* (Lines 19-20). Finally, **CAR-Miner** will be recursively called with a new set *P* as its input parameter (Line 21).

The procedure **ENUMERATE-CAR**(l, *minConf*) generates a rule from node *l*. It first computes the confidence of the rule (Line 22), if the confidence of this rule satisfies *minConf* (Line 23), then it adds this rule into the set of CARs (Line 24).

### 4.3. An example

In this section, we use the example in Table 1 to describe the **CAR-Miner** process with *minSup* = 10% and *minConf* = 60%. Figure 2 shows the results of this process.
The MECR-tree was built from the dataset in Table 1 as follows: First, the root node \( L_r \) contains all frequent 1-itemsets such as:

\[
\begin{align*}
&1 \times a1, \\
&1 \times a2, \\
&1 \times a3, \\
&2 \times b1, \\
&2 \times b2, \\
&2 \times b3, \\
&4 \times c1, \\
&4 \times c2.
\end{align*}
\]

After that, procedure CAR-Miner is called with the parameter \( L_r \). We use node \( l_i = \frac{1 \times a2}{38(0,2)} \) as an example for illustrating the CAR-Miner process. \( l_i \) joins with all nodes following it in \( L_r \):

- With node \( l_j = \frac{1 \times a3}{456(2,1)} \): They \( (l_i \) and \( l_j \) have the same attribute and different values. Don’t make any thing from them.

- With node \( l_j = \frac{2 \times b1}{15(1,1)} \): Because their attributes are different, three elements are computed such as \( O.\text{att} = l_i.\text{att} \cup l_j.\text{att} = 1 \cup 2 = 3 \) or \( 11 \) in bit presentation; \( O.\text{itemset} = l_i.\text{values} \cup l_j.\text{values} = a2 \cup b1 = a2b1 \), and \( O.\text{Obidset} = l_i.\text{Obidset} \cap l_j.\text{Obidset} = \{3,8\} \cap \{1,5\} = \emptyset \). Because the \( O.\text{count}[O.\text{pos}] = 0 < \text{minSup} \), \( O \) is not added to \( P_i \).
With node $l_j = \frac{2 \times b_2}{238(3,0)}$: Because their attributes are different, three elements are computed such as $O.\text{att} = l_i.\text{att} \cup l_j.\text{att} = 1 | 2 = 3$ or 11 in bit presentation; $O.\text{itemset} = l_i.\text{values} \cup l_j.\text{values} = a_2 \cup b_2 = a_2b_2$, and $O.\text{Obidset} = l_i.\text{Obidset} \cap l_j.\text{Obidset} = \{3,8\} \cap \{2,3,8\} = \{3,8\}$. Because of $l_i.\text{Obidset} = \emptyset$, the algorithm copies all information for $l_i$ to $O$. This means that $O.\text{count} = l_i.\text{count} = (2,0)$, and $O.\text{pos} = 1$. Because $O.\text{count}[O.\text{pos}] = 2 > \min\text{Sup}$, add $O$ to $P_i \Rightarrow P_i = \{ \frac{3 \times a_2 b_2}{38(0,2)} \}$.

With node $l_j = \frac{2 \times b_3}{467(3,0)}$: Because their attributes are different, three elements are computed such as $O.\text{att} = l_i.\text{att} \cup l_j.\text{att} = 1 | 2 = 3$ or 11 in bit presentation; $O.\text{itemset} = l_i.\text{values} \cup l_j.\text{values} = a_2 \cup b_3 = a_2b_3$, and $O.\text{Obidset} = l_i.\text{Obidset} \cap l_j.\text{Obidset} = \{3,8\} \cap \{4,6,7\} = \emptyset$. Because the $O.\text{count}[O.\text{pos}] = 0 < \min\text{Sup}$, $O$ is not added to $P_i$.

With node $l_j = \frac{4 \times c_1}{12346(3,2)}$: Because their attributes are different, three elements are computed such as $O.\text{att} = l_i.\text{att} \cup l_j.\text{att} = 1 | 4 = 5$ or 101 in bit presentation; $O.\text{itemset} = l_i.\text{values} \cup l_j.\text{values} = a_2 \cup c_1 = a_2c_1$, and $O.\text{Obidset} = l_i.\text{Obidset} \cap l_j.\text{Obidset} = \{3,8\} \cap \{1,2,3,4,6\} = \{3\}$. The algorithm computes additional information including $O.\text{count} = \{0,1\}$ and $O.\text{pos} = 2$. Because the $O.\text{count}[O.\text{pos}] = 1 \geq \min\text{Sup}$, $O$ is added to $P_i \Rightarrow P_i = \{ \frac{3 \times a_2 b_2 \times 5 \times a_2 c_1}{38(0,2)} , \frac{3 \times a_2 c_2}{38(0,2)} \}$.

With node $l_j = \frac{4 \times c_2}{578(1,2)}$: Because their attributes are different, three elements are computed such as $O.\text{att} = l_i.\text{att} \cup l_j.\text{att} = 1 | 4 = 5$ or 101 in bit presentation; $O.\text{itemset} = l_i.\text{values} \cup l_j.\text{values} = a_2 \cup c_2 = a_2c_2$, and $O.\text{Obidset} = l_i.\text{Obidset} \cap l_j.\text{Obidset} = \{3,8\} \cap \{1,2,3,4,6\} = \emptyset$. The algorithm computes additional information including $O.\text{count} = \{0,1\}$ and $O.\text{pos} = 2$. Because the $O.\text{count}[O.\text{pos}] = 1 \geq \min\text{Sup}$, $O$ is added to $P_i \Rightarrow P_i = \{ \frac{3 \times a_2 b_2 \times 5 \times a_2 c_1}{38(0,2)} , \frac{3 \times a_2 c_2}{38(0,2)} \}$.
The algorithm computes additional information including $O.\text{count} = \{0,1\}$ and $O.\text{pos} = 2$. Because the $O.\text{count}[O.\text{pos}] = 1 \geq \text{minSup}$, $O$ is added to $P_i \Rightarrow P_i = \{3\times a2b2, 5\times a2c1, 5\times a2c2 \over 38(0.2), 8(0.1), 8(0.1)\}$.

- After $P_i$ is created, the CAR-Miner is called recursively with parameters $P_i$, $\text{minSup}$, and $\text{minConf}$ to create all children nodes of $P_i$. Consider the process to make children nodes of node $l_i = \{3\times a2b2 \over 38(0.2)\}$:
  - With node $l_j = \{5\times a2c1 \over 3(0.1)\}$: Because their attributes are different, three elements are computed such as $O.\text{att} = l_i.\text{att} \cup l_j.\text{att} = 3 \cup 5 = 7$ or 111 in bit presentation; $O.\text{itemset} = l_i.\text{values} \cup l_j.\text{values} = a2b2 \cup a2c1 = a2b2c1$, and $O.\text{Obidset} = l_i.\text{Obidset} \cap l_j.\text{Obidset} = \{3,8\} \cap \{3\} = \{3\} = l_j.\text{Obidset}$. The algorithm copies all information of $l_j$ to $O$, it means that $O.\text{count} = l_j.\text{count} = (0,1)$ and $O.\text{pos} = 2$. Because the $O.\text{count}[O.\text{pos}] = 1 > \text{minSup}$, $O$ is added to $P_i \Rightarrow P_i = \{7\times a2b2c1 \over 3(0.1)\}$.
  - Using the same process for node $l_j = \{5\times a2c2 \over 8(0.1)\}$, we have the result $P_i = \{7\times a2b2c1, 7\times a2b2c2 \over 3(0.1), 8(0.1)\}$.

Rules are easily to generate in the same step for traversing $l_i$ (Line 3) by calling procedure $\text{ENUMERATE-CAR}(l_i, \text{minConf})$. For example, when traversing node $l_i = \{1\times a2 \over 38(0.2)\}$, the procedure computes the confidence of the candidate rule, $\text{conf} = l_i.\text{count}[l_i.\text{pos}] / |l_i.\text{Obidset}| = 2/2 = 1$. Because $\text{conf} \geq \text{minConf}$ (60%), add rule $\{(A,a2)\} \rightarrow \text{n} (2,1)$ into the rule set CARs. The meaning of this rule is “If $A = a2$, then class = $n$” (support = 2 and confidence = 100%).
To show the efficiency of Theorem 2, we can see that the algorithm need not compute the information of some itemsets, such as \{3\times a2b2, 7\times a1b1c1, 7\times a1b2c1, 7\times a1b3c1, 7\times a2b2c1, 7\times a2b2c2, 7\times a3b1c2, 7\times a3b3c1\}.

5 Experimental results

5.1. Characteristics of experimental datasets

The algorithms used in the experiments were coded on a personal computer with C#2008, Windows 7, Centrino 2\times 2.53 GHz, and 4MBs RAM. The experimental results were tested in the datasets obtained from the UCI Machine Learning Repository (http://mlearn.ics.uci.edu). Table 4 shows the characteristics of the experimental datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#attrs</th>
<th>#classes</th>
<th>#distinct values</th>
<th>#Objs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast</td>
<td>12</td>
<td>2</td>
<td>737</td>
<td>699</td>
</tr>
<tr>
<td>German</td>
<td>21</td>
<td>2</td>
<td>1077</td>
<td>1000</td>
</tr>
<tr>
<td>Lymph</td>
<td>18</td>
<td>4</td>
<td>63</td>
<td>148</td>
</tr>
<tr>
<td>Led7</td>
<td>8</td>
<td>10</td>
<td>24</td>
<td>3200</td>
</tr>
<tr>
<td>Vehicle</td>
<td>19</td>
<td>4</td>
<td>1434</td>
<td>846</td>
</tr>
</tbody>
</table>

The experimental datasets had different features. The Breast, German and Vehicle datasets had many attributes and distinctive (values) but had very few numbers of objects (or records). The Led7 dataset had only a few attributes, distinctive values and number of objects.

5.2. Numbers of rules of the experimental datasets

Figures 3 to 7 show the numbers of rules of the datasets in Table 4 for different minimum support thresholds. We used a \( \text{minConf} = 50\% \) for all experiments.
Figure 3. Numbers of CARs in the Breast dataset for various minSup values

Figure 4. Numbers of CARs in the German dataset for various minSup values

Figure 5. Numbers of CARs in the Lymph dataset for various minSup values
The results from Figures 3-7 show that some datasets had a lot of rules. For example, the Lymph dataset had 4,039,186 rules with a minSup = 1%. The German dataset had 752,643 rules with a minSup = 1%, etc.

5.3. Execution time

Experiments were then made to compare the execution time between CAR-Miner and ECR-CARM (Vo and Le, 2008). The results are shown in Figures 8 to 12.
Figure 8. The execution time for CAR-Miner and ECR-CARM in the Breast dataset

Figure 9. The execution time for CAR-Miner and ECR-CARM in the German dataset

Figure 10. The execution time for CAR-Miner and ECR-CARM in the Lymph dataset
Figure 11. The execution time for CAR-Miner and ECR-CARM in the Led7 dataset

Figure 12. The execution time for CAR-Miner and ECR-CARM in the Vehicle dataset

Results from Figures 8 to 12 show CAR-Miner to be more efficient than ECR-CARM in all of the experiments. For example: Consider the Breast dataset with a \( \text{minSup} = 0.1\% \). The mining time for the CAR-Miner was 1.517 second, while that for the ECR-CARM was 17.136 second. The ratio was \( \frac{1.517}{17.136} \times 100\% = 8.85\% \).
6 Conclusions and future work

This paper proposed a new algorithm for mining CARs using a tree structure. Each node in the tree contained some information for fast computation of the support of the candidate rule. In addition, using Obidset, we were able to compute the support of itemsets quickly. Some theorems were also developed. Based on these theorems, we did not need to compute the information for a lot of nodes in the tree. With these improvements, the proposed algorithm had better performance relative to the previous algorithm in regard to all results.

Mining itemsets from incremental databases has been developed in recent years (Gharib et al., 2010; Hong and Wang, 2010; Hong et al., 2009; Hong et al., 2011; Lin et al., 2009). It can be seen that it saves a lot of time and memory when compared with mining from integration databases. Therefore, in the future, we will study how to use this approach for mining CARs.

References


Li, W., Han, J., Pei, J. (2001). CMAR: Accurate and efficient classification based on multiple class-association rules. *The 1st IEEE International Conference on Data Mining*, San Jose, California, USA, pp. 369-376.


