A greedy classification algorithm based on association rule

F.A. Thabtah a,*, P.I. Cowling b

a Department of Computing and Engineering, University of Huddersfield, Huddersfield HD1 3DH, UK
b MOSAIC Research Centre, Bradford University, Bradford BD7 1DP, UK

Received 8 May 2006; received in revised form 8 August 2006; accepted 17 October 2006
Available online 28 November 2006

Abstract

Classification and association rule discovery are important data mining tasks. Using association rule discovery to construct classification systems, also known as associative classification, is a promising approach. In this paper, a new associative classification technique, Ranked Multilabel Rule (RMR) algorithm is introduced, which generates rules with multiple labels. Rules derived by current associative classification algorithms overlap in their training objects, resulting in many redundant and useless rules. However, the proposed algorithm resolves the overlapping between rules in the classifier by generating rules that does not share training objects during the training phase, resulting in a more accurate classifier. Results obtained from experimenting on 20 binary, multi-class and multi-label data sets show that the proposed technique is able to produce classifiers that contain rules associated with multiple classes. Furthermore, the results reveal that removing overlapping of training objects between the derived rules produces highly competitive classifiers if compared with those extracted by decision trees and other associative classification techniques, with respect to error rate.

1. Introduction

Classification is an important task in data mining, which aims to predict the classes of future data objects. In classification, one builds a model (set of rules) from a group of classified training data objects in order to forecast the classes of previously unseen data objects. The data set used to learn the model is known as the training data set and the data set used to measure the quality of the model is known as the test data set. Association rule discovery is another important task in data mining, which aims to find the correlations among items in a transactional database. These correlations are represented as simple rules, i.e. \( X \rightarrow Y \), which can be used to guide strategic decisions related to product shelving, sales promotions, marketing, and planning [1,8,21]. The main difference between association rule discovery and classification is that there is no class to predict in the former, whereas in the latter, the ultimate goal is to predict the class labels of test data objects.

Classification techniques such as rule induction and covering [3,4], derive the most obvious class correlated to a rule and simply ignore the other classes. They consider building the classifier in a heuristic way. Once a rule is created, all training data objects associated with it are removed, and thus a training object is covered by only one rule. Hence, these approaches usually produce small sized classifiers. In the last few years a new hybrid approach that uses association rule discovery methods to build classifiers called associative classification (AC), has been proposed [9,10,17]. AC is a promising classification approach, which has been shown to build more accurate set of rules than traditional classification approaches [2,9,10,15,20].

Association rule discovery approaches consider the correlation among all possible items in a transactional database and therefore, rules generated share training objects. In other words, a single transactional data object is allowed to be used in the production of multiple association rules. Since, AC approach uses association rule discovery methods to discover rules, the derived rules in the classifier share training data objects as well. This results in rules dependency where some rules depend upon other higher ranked rules in the classifier. Another more important problem that could result from rules sharing training objects is that classes associated with many rules learned during the training phase are not the most accurate ones. To explain

* Corresponding author.
E-mail address: f.thabtah@hud.ac.uk (F.A. Thabtah).

1568-4946/$ – see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2006.10.008
such a problem, consider Table 1, which represents a partial training data set where “Att1” and “Att2” columns are attributes, and the last column is the class attribute. Assume that there are two potential rules extracted from Table 1, e.g. $r_1$: $a \land b \rightarrow c_1$ and $r_2$: $b \rightarrow c_1$, and assume that $r_1$ precedes $r_2$. Now, once $r_1$ is evaluated and inserted into the classifier using an AC technique such as CBA [10] or CMAR [9], all training data objects associated with $r_1$ are removed, i.e. rows (1–3) using the database coverage pruning [10]. The database coverage heuristic removes training objects covered by every evaluated rule and ensures that each rule must cover at least one training object to be part of the final classifier. In cases where the evaluated rule covers no training objects, it will be then discarded. This paper argues that the removal of the training objects associated with $r_1$ may influence other potential rules that share training objects with $r_1$ such as $r_2$. Consequently, after inserting $r_1$ into the classifier, the “statistically fittest” class $c_1$ of rule $r_2$ that was learned during the training phase would not be the fittest class any more; rather a new class at that point becomes the fittest class, $c_2$, because it has the largest representation among the remaining $r_2$ rows in the training data. We call this problem, which has resulted from rule pruning, the “fittest class problem”. To the best of the author’s knowledge, there are no AC methods which take the fittest class problem into consideration. This paper investigates the impact of removing the training data objects associated with each evaluated rule during pruning on other candidate rules that share training data. This may involve applying new class labels for some of the rules during the prediction phase rather than the class labels learned for these rules during the training phase.

The fittest class problem is a serious problem because it may lead to the creation of many useless rules (rules that predict classes that have low actual representation and sometimes no representation at all in the training data (see Section 4.3)). In the training phase, rules generated by AC methods are associated with the largest frequency classes, however, the influence of the deleted training objects for each evaluated rule during pruning is not considered on other potential rules that use these deleted objects, thus there could be many redundant rules in the final classifier. A post-pruning method like [9], which discards detailed specific rules with lower confidence values than general rules can be applied to reduce rules redundancy. But, in the example demonstrated above, the specific rule $(r_1)$ has occurred only with class $c_1$ in the training data, i.e. $(3/3)\ 100\%\ confidence,$ whereas the general rule $(r_2)$ has $(5/8)\ 62.50\%\ confidence$ and therefore, this pruning method cannot do much in this case. If the effect of removal of training objects for the evaluated rules during pruning is considered on other potential rules in the training phase, then a more realistic classifier that assigns the true class fitness to each rule will result.

A new AC algorithm called Ranked Multilabel Rule (RMR) algorithm, which uses the fittest class associated with a rule in the prediction step rather than the largest frequency class learned for that rule, is presented. The fittest class is assigned to each rule by a pruning heuristic, which ensures training data objects cannot be shared by rules. This pruning heuristic discards the training data objects for each selected rule and these discarded objects become unavailable for the remaining lower ranked candidate rules. This means even though a training object has been used in learning multiple rules, the pruning method proposed ensures that the final rules in the classifier do not share training objects similar to rule induction approaches (Section 4.3 gives further detail). The pruning heuristic may change the class for some of the rules due to removing the overlapping among rules training data objects.

Basic concepts in AC and a demonstrated example are presented in Section 2. We survey some common AC approaches (Section 3). The proposed algorithm and experimental results are discussed in depth in Sections 4 and 5, respectively. Finally, conclusions are presented in Section 6.

### 2. Associative classification problem and its basic concepts

In associative classification, the training data set $T$ has $m$ distinct attributes $A_1, A_2, \ldots, A_m$ and $C$ is a list of classes. The number of rows in $T$ is denoted by $|T|$. An attribute may be categorical (where each attribute takes a value from a finite set of possible values) or continuous where each attribute takes a value from an infinite set (e.g. reals or integers). For categorical attributes, all possible values are mapped to a set of positive integers to represent them. The actual occurrence of a rule $r$ in $T$ is the number of rows in $T$ that match $r$’s itemset.

<table>
<thead>
<tr>
<th>RowID</th>
<th>Att1</th>
<th>Att2</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
<td>$c_1$</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>b</td>
<td>$c_1$</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>b</td>
<td>$c_1$</td>
</tr>
<tr>
<td>4</td>
<td>e</td>
<td>b</td>
<td>$c_1$</td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>b</td>
<td>$c_1$</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>b</td>
<td>$c_2$</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>b</td>
<td>$c_2$</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>b</td>
<td>$c_2$</td>
</tr>
<tr>
<td>9</td>
<td>e</td>
<td>f</td>
<td>$c_3$</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Definition 6. The support count ($\text{suppcount}$) of ruleitem \( r = (\text{cond}, c) \) is the number of rows in \( T \) that matches \( r \)'s itemset, and belongs to a class \( c \).

Definition 7. The occurrence ($\text{occitm}$) of an itemset \( I \) in \( T \) is the number of rows in \( T \) that match \( I \).

Definition 8. An itemset \( I \) passes the minimum support (\( \text{minsupp} \)) threshold if \( \text{occitm}(I)/|T| \geq \text{minsupp} \). Such an itemset is called a frequent itemset.

Definition 9. A ruleitem \( r \) passes the \( \text{min supp} \) threshold if, \( \text{suppcount}(r)/|T| \geq \text{minsupp} \). Such a ruleitem is said to be a frequent ruleitem.

Definition 10. A ruleitem \( r \) passes the minimum confidence (\( \text{minconf} \)) threshold if \( \text{suppcount}(r)/\text{actocc}(r) \geq \text{minconf} \).

Definition 11. A rule in the RMR rule based system is represented in the form: \( \text{cond} \rightarrow c_1 \lor c_2 \lor \cdots \lor c_r \), where the left-hand side of the rule (antecedent) is an itemset and the right-hand side of the rule (consequent) is a ranked list of class labels.

The proposed algorithm classifiers are of the form \( H: A_1 \times A_2 \times \cdots \times A_n \rightarrow Y \), where \( A_i \) is a set of possible values of attribute \( i \) and \( Y \) is a list of ranked class labels. The aim of the presented AC approach is to produce all frequent \( \text{ruleitem} \) and to choose a subset of them which passes the \( \text{minconf} \) threshold to form the classifier. In other words, the goal is to find a classifier \( h \) that maximises the probability that \( h(a) = y \) for each test case \( (a, y) \).

Most AC algorithms work in two phases, phase 1 involves the discovery of frequent \( \text{ruleitem} \) (attribute values that occur with a class label above the user specified support threshold). In the second phase, a set of rules (classifier) is built from the frequent \( \text{ruleitem} \) found in phase 1. To explain the discovery of rules and building the classifier in AC mining, consider the training data set shown in Table 2, which represents whether or not a person is likely to buy a new car. Assume that \( \text{minsupp} = 2 \) and \( \text{minconf} = 50\% \). Frequent \( \text{ruleitem} \) discovered in phase 1 along with their relevant support and confidence values are shown in Table 3. In phase 2 and before constructing the classifier, most AC algorithms including [9,10,15,20] sort the rules discovered in phase 1 according to their confidence and support values and then apply pruning heuristics to discard redundant and useless rules. For more comprehensive studies on pruning and sorting techniques in AC mining refer to [16] and [14], respectively.

### Table 2

<table>
<thead>
<tr>
<th>Age</th>
<th>Income</th>
<th>has a car</th>
<th>Buy/class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Senior</td>
<td>Middle</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Youth</td>
<td>Low</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Junior</td>
<td>High</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Youth</td>
<td>Middle</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Senior</td>
<td>High</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Junior</td>
<td>Low</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Senior</td>
<td>Middle</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

A popular pruning method in AC mining, which was proposed in [10], for building the classifier, is the database coverage. This method tests the set of ranked rules against the training data. For each rule starting with the top ranked rule, the database coverage heuristic tests if the selected rule covers correctly any training data object. In other words, it examines if the selected rule antecedent matches any training data objects. If the test turns to be true and both the selected rule and the training data object have a common class, then such a rule is considered a candidate rule in the classifier. If no training data objects match the selected rule or there was a match but no common class was found, then the selected rule will be discarded. \( \text{Ruleitem} \) in bold within Table 3 cover all the training data objects of Table 2, and represent the classifier after applying database coverage pruning.

### Table 3

<table>
<thead>
<tr>
<th>Itemset</th>
<th>AC ruleitem</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>[low]</td>
<td>No</td>
<td>2/7</td>
</tr>
<tr>
<td>[high]</td>
<td>Yes</td>
<td>2/7</td>
</tr>
<tr>
<td>[senior, no]</td>
<td>Yes</td>
<td>2/7</td>
</tr>
<tr>
<td>[middle]</td>
<td>Yes</td>
<td>2/7</td>
</tr>
<tr>
<td>[senior]</td>
<td>Yes</td>
<td>2/7</td>
</tr>
<tr>
<td>[y]</td>
<td>Yes</td>
<td>2/7</td>
</tr>
<tr>
<td>[n]</td>
<td>Yes</td>
<td>2/7</td>
</tr>
<tr>
<td>[n]</td>
<td>No</td>
<td>2/7</td>
</tr>
</tbody>
</table>

3. Literature review

Classic AC algorithms like CBA and ADT [18] are based on the step-wise a priori algorithm of association rule discovery [1], which discovers frequent itemsets by making multiple passes over the training data set. In the first scan, it counts the support of the one-itemsets and determines whether or not they are frequent, then in each subsequent pass, starting with itemsets found to be frequent in the previous pass produces new candidate itemsets. The generation of rules is straightforward once the frequent itemsets and their support are known. However, the problem of finding frequent itemsets is a harder problem that requires extensive computation capacity especially when the expected number of candidate itemsets is large [8,22].

Li et al. [9] investigated the problem of generating a large number of rules in AC. They developed an algorithm that adopts the FP-growth association rule discovery approach [8] to efficiently find frequent \( \text{ruleitem} \). Their proposed algorithm, named CMAR, differs from other associative methods, since, it uses more than one rule to assign a class to each test object, and it stores the rules in an efficient prefix tree data structure, known as a CR-tree. Experimental studies on 26 data sets from Ref. [11] showed that CMAR has slightly higher accuracy than CBA on 13 benchmark problems.

To exploit all possible knowledge that can be extracted from a training data set, a lazy pruning approach based on a two level...
of classification rules, called \( L^3 \), has been proposed in Ref. [2]. \( L^3 \) differs from other AC techniques because it does not discard rules that have not been used in the training phase, aiming to improve upon classification accuracy. During the learning process, \( L^3 \) divides the rules into two levels, a primary level that contains rules that correctly cover at least one training object, and a secondary level that contains rules, which do not cover any training objects. In classifying a test object, all primary rules are first tested, in the case none of them is able to cover the test object; the secondary rules are then evaluated, until the test object is covered. The \( L^3 \) algorithm generates rules using the FP-growth [8] approach, and ranks the rules similarly to CBAs rule ranking method. The \( L^3 \) algorithm often extracts very large classifiers, sometimes having of the order of tens of thousands of rules, this limits its use.

A recently developed associative classification algorithm called MCAR [15] employs tid-list intersections to quickly find the rules. This algorithm consists of two main phases: rules generation and a classifier builder. In the first phase, the training data set is scanned once to discover the potential rules of size 1, and then MCAR intersects the potential rules tid-lists of size 1 to find potential rules of size 2 and so forth. This rules discovery method does no require passing over the training data multiple times. In the second phase, potential rules that cover a certain number of training objects will be kept in the final classifier. Unlike the MCAR algorithm which produces rules associated

---

**Fig. 1.** The RMR algorithm.

---

```
Input: Training data \( \mathcal{D}, \minsupp \) and \( \minconf \) thresholds
Output: A classifier \( R \) and a list \( \mathcal{Tp} \) of covered rows
A rule \( r \) in \( R \) has the following properties: items, class, rowIDs (tid-list)

Step 1: Rule Discovery

Scan \( \mathcal{D} \) for the set \( S_1 \) of frequent one-itemset
\[ R \leftarrow S_1 \]
\[ i \leftarrow 1 \]
while (\( S_i \) ≠ 0)
\[ S_{i+1} \leftarrow \text{produce}(S_i) \] //see Figure 2 for produce function
\[ R \leftarrow R \cup S_{i+1} \]
\[ i \leftarrow i + 1 \]

Step 2: Evaluation Step (Pruning) for each candidate rules produced during an iteration
\[ R' = \{r_1\} \]
\[ \mathcal{Tp} = \{r_1{\text{rowIDs}}\} \]
For \( i = 2 \) to \( |R| \)
\[ \text{Temp} = (r_i{\text{rowIDs}} - \mathcal{Tp}) \]
if (\(|\text{Temp}| > 0\))
Update \( r_i \) class by considering only rowIDs which are in \( \text{Temp} \)
\[ R' \leftarrow R' \cup r_i \]
\[ \mathcal{Tp} \leftarrow \mathcal{Tp} \cup \text{Temp} \]
end if
end for
return \( R', \mathcal{Tp} \)

Step 3: Multi-label rules:
Do
\[ i \leftarrow 1 \]
\[ T' \leftarrow T \]
\[ R \leftarrow \Phi \]
\[ T' \leftarrow T' - \mathcal{Tp} \]
\[ (R', \mathcal{Tp}) = \text{RMRScan}(T', \minsupp, \minconf) \] // Repeat step 1 on \( T' \) until no further frequent rules are found
\[ R \leftarrow R \cup R' \]
While (\( R' \neq \Phi\))
return \( R \)
```
with the most obvious class in the training data, the RMR algorithm generates rules with more than one class as we will discuss in Section 4.4, extracting useful knowledge ignored by most of the existing AC algorithms.

4. The RMR algorithm

The algorithm presented in this paper consists of three main phases: rules discovery, repeated learning and classification. In the first phase, RMR scans the training data to discover the potential rules. During rules discovery, RMR utilises a pruning heuristic that ensures the orders of the classes in the potential rules that share-training objects are updated to minimise rules redundancy. In addition, each evaluated rule’s training data objects are removed from the training data (see Section 4.3). In the second phase, the algorithm proceeds to discover more rules that pass the minsupp and minconf thresholds from the remaining unclassified objects, until no further frequent ruleitems can be found. At that stage, the rules sets derived during each iteration will be merged to form a multi-label classifier. In the third phase, the multi-label classifier is evaluated against a test data set. Training attributes can be categorical or continuous. Continuous attributes are discretised using the multi-interval method of Ref. [7]. Pseudocode for the algorithm is given in Figs. 1 and 2, which will be explained in depth in Sections 4.1–4.3, respectively.

4.1. Frequent ruleitems discovery

The proposed algorithm employs a method that extends the tid-list intersection methods of Ref. [22] to handle classification benchmark problems. The tid-list (transaction identifier list) for an itemset holds the training row IDs where that itemset occurs in the training data set. Using the tid-list for an itemset in association rule discovery is a good approach, since the cardinality of the itemset tid-list divided by the total number of the transactions gives the support for that itemset. In addition, each evaluated rule’s training data objects are removed from the training data (see Section 4.3). In the second phase, the algorithm proceeds to discover more rules that pass the minsupp and minconf thresholds from the remaining unclassified objects, until no further frequent ruleitems can be found. At that stage, the rules sets derived during each iteration will be merged to form a multi-label classifier. In the third phase, the multi-label classifier is evaluated against a test data set. Training attributes can be categorical or continuous. Continuous attributes are discretised using the multi-interval method of Ref. [7]. Pseudocode for the algorithm is given in Figs. 1 and 2, which will be explained in depth in Sections 4.1–4.3, respectively.

4.1. Frequent ruleitems discovery

The proposed algorithm employs a method that extends the tid-list intersection methods of Ref. [22] to handle classification benchmark problems. The tid-list (transaction identifier list) for an itemset holds the training row IDs where that itemset occurs in the training data set. Using the tid-list for an itemset in association rule discovery is a good approach, since the cardinality of the itemset tid-list divided by the total number of the transactions gives the support for that itemset. However, tid-list intersection method presented in association rule discovery needs to be modified in order to deal with classification problems, where the classes associated with each itemset are considered when computing the support.

**Function produce**

Input: A set of itemsets $S$

Output: set of $S'$ produced itemsets

$S' \leftarrow \emptyset$

Do

For each pair of disjoint items $I_1, I_2$ in $S$ Do

If ($\langle I_1 \cup I_2, c \rangle$) passes the minsupp threshold

and ($\langle I_1 \cup I_2, c \rangle$) passes the minconf threshold

$S' \leftarrow S' \cup \langle \langle I_1 \cup I_2, c \rangle \rangle$

end if

end

Return $S'$

Fig. 2. Rule discovery algorithm of RMR.

The frequent ruleitems discovery method employed by RMR has been adapted from a recently developed AC algorithm called MCAR, where during the first training data scan, frequent one-itemsets are determined, and their occurrences in the training data (rowIDs) are stored inside an array. Also, classes and their frequencies are stored in an array. Any itemset that fails to pass the support threshold is discarded. We denote the tid-list entities for itemset $I$ as $I \times$ rowIDs. We use the (produce function) shown in Fig. 2 to find frequent itemsets of size $k$ by appending disjoint frequent itemsets of size $k - 1$ and intersecting their rowIDs. The result of a simple intersection between rowIDs of two itemsets gives a set, which holds the rowIDs where both itemsets occur together in the training data. This set along with the class array, which holds the class labels frequencies can be used to compute the support and confidence of the new ruleitem resulted from the intersection (see Section 4.2, for support and confidence calculation).

The (produce function) is invoked iteratively on each set of frequent itemsets produced at iteration $K$ in order to generate possible frequent itemsets at iteration $K + 1$. To show how a frequent ruleitem is determined, consider, for instance itemsets ($\langle A_1, x_1 \rangle$) and ($\langle A_2, y_1 \rangle$) in Table 4, following the two sets represent the rowIDs in which they occur $\{1, 2, 3, 4, 8\}$ and $\{1, 3, 5, 6, 10\}$. One can determine the support of the itemset ($\langle A_1, x_1 \rangle$, $\langle A_2, y_1 \rangle$) by intersecting the rowID sets for itemsets ($\langle A_1, x_1 \rangle$) and ($\langle A_2, y_1 \rangle$). The cardinality of the resulting set $\{1, 3\}$ represents the support for itemset ($\langle A_1, x_1 \rangle$, $\langle A_2, y_1 \rangle$), i.e., $2/10$. If it passes the minsupp threshold, then one can proceed by checking whether there is some class $c$ in the class array such that ($\langle A_1, x_1 \rangle$, $\langle A_2, y_1 \rangle$, $c$) passes the minconf threshold, otherwise we prune it. The support and confidence for a frequent ruleitem is calculated by iterating over the tid-list of its itemset and locating the largest class that occurs with it in the class array as we will discuss in the next section.

The main operation used in the rule discovery algorithm of RMR is simple intersections between rowIDs (tid-lists) of frequent itemsets. There are no multiple database scans or candidate generation step, rather and during each iteration, only tid-lists of frequent itemsets produced at the previous iteration are kept for further intersections.

<table>
<thead>
<tr>
<th>RowID</th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x_1$</td>
<td>$y_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>2</td>
<td>$x_1$</td>
<td>$y_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>3</td>
<td>$x_3$</td>
<td>$y_1$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>4</td>
<td>$x_4$</td>
<td>$y_2$</td>
<td>$c_4$</td>
</tr>
<tr>
<td>5</td>
<td>$x_3$</td>
<td>$y_1$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>6</td>
<td>$x_2$</td>
<td>$y_3$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>7</td>
<td>$x_2$</td>
<td>$y_3$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>8</td>
<td>$x_1$</td>
<td>$y_3$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>9</td>
<td>$x_2$</td>
<td>$y_4$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>10</td>
<td>$x_3$</td>
<td>$y_1$</td>
<td>$c_1$</td>
</tr>
</tbody>
</table>
Given two rules, \( r_a \) and \( r_b \), \( r_a \) precedes \( r_b \) if:

1. The confidence of \( r_a \) is greater than that of \( r_b \).
2. The confidence values of \( r_a \) and \( r_b \) are the same, but the support of \( r_a \) is greater than that of \( r_b \).
3. Confidence and support values of \( r_a \) and \( r_b \) are the same, but \( r_a \) has fewer conditions in its left hand side than that of \( r_b \).
4. The confidence, support and conditions cardinality values of \( r_a \) and \( r_b \) are the same but \( r_a \) is associated with more frequently occurring class than that of \( r_b \).
5. All above criteria are identical for \( r_a \) and \( r_b \) but \( r_a \) was generated from items that occur earlier in the training data than that of \( r_b \).

Fig. 3. Rule ranking method used by RMR.

4.2. Support and confidence computation and rule generation

In this section we briefly explain how support and confidence for ruleitems are found. To find the support for a ruleitem, we use the rowIDs of its itemset to locate classes associated with it in the class array and select the class with the largest frequency. Then by taking the cardinality of the set of the rowIDs where the itemset and its largest class occur and dividing it by the size of the training data set, one can obtain the ruleitem support. The calculation of the confidence is done similarly except that the denominator of the fraction is the size of the set of the rowIDs of the ruleitem condition (its itemset) instead of the size of the whole training data set. It should be noted that every time a frequent ruleitem is found, only the rule with the largest confidence is considered. In the case that a ruleitem is associated with two classes with identical confidence, the choice of the rule is arbitrary.

4.3. Classifier builder and pruning

The RMR algorithm shown in Fig. 1 derives a set of potential rules during each iteration and stores them in a descending order based on the ranking procedure shown in Fig. 3 (step 1). Then a pruning step is performed to measure the effectiveness of each rule and allow the removal of training objects shared by rules. In the rule pruning as shown in Fig. 1, the highest ranked potential rule is inserted into the classifier and its associated rowIDs are put into a temporary array, Temp. For each other potential lower ranked rule \( r \), one can check if it covers any training objects by performing simple set difference operation between \( r \) rowIDs set and Temp. The resulting set will represent training objects row numbers of \( r \), excluding the training objects row numbers used by any higher ranked rules than \( r \) (rules that already have been evaluated and inserted into the classifier). This set (\( T_p \)) will be used by the RMR algorithm as the new rowIDs set of \( r \), where class labels in this set are the only ones to be considered for rule \( r \) in the prediction step of test data objects. Consequently, after ignoring all training row numbers used by any higher ranked rules that share some training objects with \( r \), the statistically fittest class associated with \( r \) may not be the fittest class learned during the training phase any more. The RMR algorithm checks if rule \( r \) covers any training objects, if so \( r \) is then inserted into the classifier as well as its updated set of rowIDs (\( T_p \)) into Temp and continuously repeats the same process on the remaining potential rules.

The procedure described above is performed at each iteration by the RMR algorithm and it results in a rule set, which contains rules that do not overlap in their training data objects. To clarify, suppose rules such as \( r_1 \) and \( r_2 \) share training objects and the sets \{1, 2, 5, 6\} and \{1, 2, 7, 10\} indicate where their conditions (items) occur in the training data, respectively. Assume that \( r_1 \) precedes \( r_2 \), once \( r_1 \) is evaluated and inserted into the classifier, this involves discarding \( r_1 \)'s objects (putting \( r_1 \)'s rowIDs into array such as Temp). In evaluating \( r_2 \), RMR considers classes for items that only occur in rowIDs \{7, 10\}, since the result of the difference between \( r_2 \) rowIDs and the Temp, i.e. array \( T_p \), does not contain \( r_2 \) rowIDs that have been used to generate \( r_1 \). The RMR classifier builder ensures that each training instance is covered by a single rule and it prunes any rule that fails to cover at least one training data object.

4.4. Multiple labels rules generation

The proposed algorithm learns all possible classes associated with each candidate itemset in the training data set to extract the multi-label classifier. For a given multi-class training data \( T \), existing AC algorithms learn and derive a single label rule set and form a default class for the remaining unclassified objects in \( T \). On the other hand, RMR utilises a recursive learning phase similar to the MMAC algorithm [17] in order to derive more than one rule set and merges them to form a multi-label classifier. The main difference between the proposed algorithm and the MMAC one is that MMAC does not employ the pruning step described in Section 4.3 and thus rules produced do overlap in their training data objects similar to regular AC algorithms. The proposed algorithm produces a first-pass rule set in which each rule is associated with the most obvious class. Once this initial rule set is generated, all training objects associated with it are discarded and the remaining unclassified objects become a new training data set, \( T' \).

The process of discarding the training data objects of the first rule set is performed using the rowIDs of the rules conditions. According to the multi-label step described in Fig. 1, the highest ranked potential rule rowIDs are inserted in a temporary array, Temp, and we iterate over the remaining potential rules. For each lower ranked rule, one can perform a simple difference operation between the rowIDs set of such rule and Temp, if the result of the difference is an empty set, meaning, the current rule training objects are already covered by other higher ranked rules, the rule is discarded. If the resulting set contains values, those values are inserted into Temp and the rule is put into the classifier. The same process is repeated on the remaining rules. Once all rules are tested, the resulting Temp holds all training objects row numbers for rules that have been inserted into the classifier.

Data set \( T' \) is generated from the difference between \( T \) and Temp. The RMR algorithm checks whether there are still more frequent ruleitems remaining undiscovered in \( T' \) (rules derived
from \( T \) which may be associated with more than one class). If so, a new set of rules will be generated from \( T' \), and the remaining unclassified objects in \( T' \) will form new training data, and so on. The algorithm proceeds with learning until no more frequent rule items are discovered in a pass. At that stage, any remaining unclassified objects will form a default class. This process results in learning from several subsets of the original training data and generating a number of rule sets. It should be noted that frequent rule items discovered during all iterations that come after the initial iteration pass the \( \text{minsupp} \) and \( \text{minconf} \) thresholds over the original training data and not on the subset of data, which they have been generated from.

4.5. Sorting of rules procedure

Since, the number of rules generated by AC can be large [2,9], selecting an appropriate rule set to form the classifier is important. Rule ranking in AC is an important step that helps in choosing the most effective rules for prediction. For instance, CBA and CMAR algorithms use the database coverage pruning to build their classifiers, whereas using this pruning, rules are tested according to their ranks. Generally, rule ranking in AC is based on support, confidence and length/cardinality of the rule's antecedent. This sequence of parameters was introduced in Ref. [14]. Several AC techniques developed after CBA, including [9,18] rank rules based on its ordering. When several rules have identical confidence, support and cardinality, these techniques choose one of the rules randomly, which possibly in some cases may degrade accuracy. This random selection happens frequently in mining classification data sets where certain attribute values occur frequently.

In order to ensure a subset of effective rules form the classifier, RMR uses a sorting method shown in Fig. 3 that considers the class distribution frequency parameter after confidence, support and rule antecedent cardinality to distinguish among rules with similar confidence and/or support in the rule ranking process. This parameter proved to be effective in reducing rule random selection for dense data sets according to Ref. [14]. The RMR algorithm invokes the above rule ranking method during each iteration on the potential generated rules set in order to ensure that only high confidence rules are kept for the prediction stage.

4.6. Prediction of test objects

In classification, let \( R \) be the set of generated rules and \( T_s \) be the set of test data objects. The basic idea of the classification procedure used by the proposed algorithm as shown in Fig. 4 is to choose the best rule among a set of high confidence, representative and general rules in \( R \) to predict \( T_s \). In classifying a test object (line 1), the classification procedure uses a simple approach, which states that the first rule in the set of ranked rules that matches the test object condition classifies it (line 5).

In a case where no rule fully matching the test object condition, the RMR prediction procedure seeks for the highest precedence rule that matches part of the test object and applies it (line 7). This is the first rule in \( Tr \), which its body matches any attribute value of the test object and not necessarily the one which matches the maximum number of attribute values in the test object condition. In cases where no rule matches the test object condition, the default class will be assigned to the test object (line 8). This classification process ensures that only the highest ranked rules classify test data objects.

5. Experimental results

Experiments have been performed on 20 data sets from the UCI Machine Learning Repository [11] and real world optimisation data [5]. Three popular classification techniques have been compared to RMR in terms of classification accuracy and the number of rules derived on the 10 binary and multi-class data sets (C4.5 [12], RIPPER [4], CBA [10]) in order to evaluate the predictive power of the proposed method. We also compared RMR with a recently developed multi-label AC technique called MMAC on a real world multi-label data. The MMAC algorithm uses overlapping rules in building the classifier, whereas RMR removes the overlapping of rules training objects while building the classifier.

To run the experiments, stratified 10-fold cross-validation [19] was used. The experiments of C4.5 and RIPPER algorithms were conducted using the Weka software system [24]. CBA experiments were conducted using a VC++ implementation version provided by Ref. [23] and (MMAC and RMR) algorithms were implemented in Java. The \( \text{minsupp} \) has been set to 2% in CBA, MMAC and RMR experiments since more extensive experiments reported in Refs. [2,10,17] suggested that it is one of the rates that achieve a good balance between accuracy and the size of the classifiers. The confidence

---

**Fig. 4.** RMR prediction algorithm.
threshold, on the other hand, has a smaller impact on the behaviour of any AC method and it has been set to 30%.

5.1. Binary and multi-class data results

The classification accuracy and the number of rules produced using stratified 10-fold cross-validation of C4.5, RIPPER, CBA and RMR against 10 binary and multi-class classification data sets from Ref. [11] are shown in Table 5. Since, RMR produces rules with multiple labels, only the top ranked class for each rule is considered to obtain the accuracy for fair comparison. RMR achieved higher classification accuracy than C4.5 on 7 out of 10 benchmark problems. RMR also achieved higher accuracy than CBA on seven benchmark problems with one further tied result. On average, the RMR algorithm achieved +3.11%, +2.91% and +1.70% above C4.5, RIPPER and CBA learning methods, respectively. The accuracy figures provide evidence for the hypothesis that in general AC techniques are more accurate than traditional classification ones such as rule induction and decision trees. This supports other research findings conducted in AC such as [9,10,20].

The number of rules generated by the proposed method and CBA are often larger than those derived by C4.5 and RIPPER for the majority of the data sets. This is not surprising, since traditional methods (RIPPER, C4.5) use extensive pruning such as global optimisation [4] and post-pruning [12] to cut down the number of rules. In fact C4.5 algorithm utilises two pruning methods to simplify the decision trees constructed, these are sub-tree replacement and pessimistic error [12]. Sub-tree replacement may be performed when a sub-tree has an expected error larger than its replacement leaf. At that point, the decision tree will be pruned by replacing a whole sub-tree by a leaf node. The expected error can be calculated on the training data or on a hold out subset that has been never used in building the decision tree. Furthermore, using pessimistic error, once the decision tree is built, rules can be directly read from the tree, where the antecedent of the rule is constructed from nodes and arcs values and the consequent is the majority values in the leaf node. For each rule, \( r: P \rightarrow C \), we can obtain a new rule \( r' \) by eliminating some items from \( r \) antecedent. Then we can replace \( r \) with \( r' \), if it has a lower error rate on the training data. These two pruning heuristics cause the generation of fewer rules for C4.5, if compared with the proposed algorithm.

Moreover, the heuristic method employed by RIPPER to discover the rules produces its classifiers locally and in a greedy fashion, which explains its limited sized classifiers. The derived rules for RIPPER are local because when a rule is discovered, all training data objects associated with it are discarded and the process continues until the rule found has unacceptable error rate. This means rules are discovered from partitions of the training data and not from the whole training data once. The search process for the rules is greedy, since RIPPER looks for the rule condition that maximises a statistical measure called Foli-gain [13]. On the other hand, AC algorithms such as CBA and RMR utilise association rule discovery strategies to find the rules. These strategies normally explore all possible associations between attribute values in the training data which explains the exponential growth of rules.

5.2. Real optimisation data results

Data from several runs generated by a Peckish hyperheuristic for a personnel-scheduling problem were provided by the authors of Ref. [5]. Data from 10 solution runs in separate text files were used. Each row in the data represents an effective local search neighbourhood (low-level heuristic) for a set of 10 possible local search neighbourhoods developed for a personnel-scheduling problem. Each file consists of data from 5000 local search applications. The term hyperheuristic was introduced by Ref. [6] as a general method that manages the choice of which neighbourhood search technique to apply during the construction process of a solution for optimisation problems. The main difference between hyperheuristic and metaheuristic approaches such as tabu search and simulated annealing is that hyperheuristic requires limited knowledge about the domain of the problem under investigation and does not search directly for the solution, and thus it can be applied to different problem without making any significant changes. The Peckish hyperheuristic, which has been used to produce the data used in my experiments, selects and applies the low-level heuristic that leads to the largest improvement on the objective function (if one exists), and this is the class we want to find.
Table 6 represents part of a solution run (data) generated by Peckish hyperheuristic where columns LLH_2 and LLH_1 represent the low-level heuristics applied at the previous two iterations. Column LLH represents the current low-level heuristic that improved the objective function and column Imp represents the improvement on the objective function value. Finally, column apply represents whether or not the selected low-level heuristic has been applied by the hyperheuristic. The data generated by the hyperheuristic have multiple labels, since at each iteration there could be more than one low-level heuristic that improves the objective function. For example, at the first iteration in Table 6, there are three low-level heuristics (LLH 2, LLH 43, LLH 74) that improve the objective function. Thus, there are three class labels associated with the instance \((LLH_2, 1), (LLH_1, 1)\). Generally, each training instance in the optimisation data may associate with more than one class label.

Figs. 5 and 6 show the relative prediction accuracy of RMR with respect to that of MMAC and CBA. In other words, how much better or worse RMR performs in term of accuracy with respect to MMAC and CBA learning algorithms. The relative prediction accuracy numbers shown in the figures are estimated using the formula: \((\text{accuracy}_{\text{RMR}}/\text{accuracy}_{\text{MMAC}})/\text{accuracy}_{\text{MMAC}}\) for MMAC, and \((\text{accuracy}_{\text{RMR}}/\text{accuracy}_{\text{CBA}})/\text{accuracy}_{\text{CBA}}\) for CBA.

The graphs show that the proposed algorithm consistently produces more accurate classifiers than that of MMAC and CBA methods for most data sets. Since, RMR beats MMAC with regards to accuracy, the effectiveness of removing the training objects for evaluated rules from other potential rules on

<table>
<thead>
<tr>
<th>Iteration</th>
<th>LLH_2</th>
<th>LLH_1</th>
<th>LLH</th>
<th>Imp</th>
<th>Apply</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>987</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>43</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>74</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>92</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>981</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>58</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>
accuracy is relatively high. Fig. 5 reveals that using rules with non-overlapping training objects is a more accurate approach when compared to AC approaches that use overlapping rules. Particularly, the proposed algorithm outperformed MMAC on 8 out of 10 multi-label data sets.

Fig. 7 shows the number of rules extracted by RMR from each data set categorised by the number of classes. Our proposed algorithm is able to extract rules that are associated with up to four class labels. This can be effective for improving the classification accuracy within applications. Fig. 7 also demonstrates that the majority of the rules created from each scheduling data set are associated with one or two labels. It turns out that this is due to the fact that during each iteration, usually only one or two low-level heuristics results in an improvement of the objective function in the scheduling problem. Thus, each training instance often corresponds to just one or two classes.

6. Conclusions

A new associative algorithm for multi-class and multi-label classification problems was proposed. This algorithm has the following characteristics:

- RMR generates rules that do not overlap in their training objects and ensures that a training instance is allowed to be covered by only a single rule, solving problems inherited from association rule discovery in AC.
- Other associative methods apply the largest frequency class associated with the rules in the training data in the prediction step, whereas RMR removes the training objects for each evaluated rule from any lower ranked rules that share these objects with it. This removal results in updating the rank of class labels associated with some of the impacted rules in the classifier.

Performance studies on 20 data sets indicated that the proposed method is highly competitive when compared with the state-of-the-art AC and traditional techniques in term of classification accuracy. The results on the real data revealed that using non-overlapping multi-label classifier (RMR) extract more accurate classification systems if compared with classifiers that contain rules that overlap in their training objects CBA and MMAC. In near future, we intend to develop a post-pruning method to cut down the number of rules generated by the proposed technique.

References