The Knowledge Engineering Review, Vol. 22:1, 37–65. © 2007, Cambridge University Press
doi:10.1017/S0269888907001026 Printed in the United Kingdom

A review of associative classification mining

FADI THABTAH

Department of Computing and Engineering, University of Huddersfield, HD1 3DH, UK;
e-mail: f.thabtah@hud.ac.uk

Abstract

Associative classification mining is a promising approach in data mining that utilizes the association rule discovery techniques to construct classification systems, also known as associative classifiers. In the last few years, a number of associative classification algorithms have been proposed, i.e. CPAR, CMAR, MCAR, MMAC and others. These algorithms employ several different rule discovery, rule ranking, rule pruning, rule prediction and rule evaluation methods. This paper focuses on surveying and comparing the state-of-the-art associative classification techniques with regards to the above criteria. Finally, future directions in associative classification, such as incremental learning and mining low-quality data sets, are also highlighted in this paper.

1 Introduction

Associative classification (AC) is a branch of a larger area of scientific study known as data mining. Fayyad *et al*. (1998) define data mining as one of the main phases in knowledge discovery from databases (KDD), which extracts useful patterns from data. AC integrates two known data mining tasks, association rule discovery and classification, to build a model (classifier) for the purpose of prediction. Classification and association rule discovery are similar tasks in data mining, with the exception that the main aim of classification is the prediction of class labels, while association rule discovery describes correlations between items in a transactional database. There are other differences discussed further in Section 2.2. In the last few years, association rule discovery methods have been successfully used to build accurate classifiers, which have resulted in a branch of AC mining (Ali *et al*., 1997; Liu *et al*., 1998). Several studies (Li *et al*., 2001; Yin & Han, 2003; Thabtah *et al*., 2004) have provided evidence that AC algorithms are able to extract classifiers competitive with those produced by decision trees (Quinlan, 1993, 1998), rule induction (Quinlan & Cameron-Jones, 1993; Cohen, 1995) and probabilistic (Duda & Hart, 1973) approaches.

Rule induction approaches, such as IREP (Furnkranz & Widmer, 1994) and RIPPER (Cohen, 1995), derive local sets of rules in a greedy manner. The derived rules 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 an unacceptable error rate. This means rules are discovered from partitions of the training data set and not from the whole training data set once. The search process for the rules is greedy as most rule induction algorithms normally look for the rule that maximizes a statistical measure. For example, the IREP rule induction algorithm constructs the rules based on first-order inductive learner (FOIL)-gain measure (Quinlan & Cameron-Jones, 1993). This means that the attribute value in the training data set with the best FOIL-gain is chosen first as a member of the current rule left-hand side (antecedent) and the process is repeated until a stopping condition is met. In contrast to rule induction approaches, which greedily and locally derive rules, AC explores the complete training data set and aims to construct a global classifier.

To build a classifier using an AC algorithm, the complete set of class association rules (CARs) is first discovered from the training data set and a subset is selected to form the classifier. The selection of such a subset can be accomplished in many ways, for example in the classification by association rule (CBA; Liu *et al*., 1998) and classification based on multiple rules (CMAR; Li *et al*., 2001) algorithms, the selection of the classifier is made using the database coverage heuristic (Liu *et al*., 1998), which evaluates the complete set of CARs on the training data set and considers rules that cover a certain number of training data objects. However, the live-and-let-live (L^3; Baralis & Torino, 2002) algorithm uses a lazy pruning approach to build the classifier. Once the classifier is constructed, its predictive power is then evaluated on test data objects to forecast their class labels.

Several AC techniques have been proposed in recent years, including Dong *et al*. (1999), Li *et al*. (2001), Baralis & Torino (2002), Yin & Han (2003) and Thabtah *et al*. (2004, 2005). These techniques use several different approaches to discover frequent itemsets (these are attribute values that pass a user-defined threshold known as minimum support), extract rules, rank rules, store rules, prune redundant or 'harmful' rules (rules that lead to incorrect classification) and classify new test objects. The goal of this paper is to survey and compare the state-of-the-art AC techniques with reference to the above criteria.

The rest of the paper is organized as follows. The AC problem and an example to demonstrate its main steps are given in Section 2. Different methods used to discover potential rules and pruning methods are surveyed in Sections 3 and 4, respectively. Common rule sorting techniques and procedures that predict test data objects are discussed in Sections 5 and 6, respectively. Sections 7 and 8 are devoted to common evaluation measures for associative classifiers and future directions in AC mining. Finally, conclusions are given in Section 9.

2 Associative classification

2.1 Associative classification problem

AC is a special case of association rule discovery in which only the class attribute is considered in the rule's right-hand side (consequent); for example, in a rule such as $X \rightarrow Y$, Y must be a class attribute. One of the main advantages of using a classification based on association rules over classic classification approaches is that the output of an AC algorithm is represented in simple if–then rules, which makes it easy for the end-user to understand and interpret it. Moreover, unlike decision tree algorithms, one can update or tune a rule in AC without affecting the complete rules set, whereas the same task requires reshaping the whole tree in the decision tree approach. Let us define the AC problem, where a 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 $|T|$. Attributes can be categorical (meaning they take a value from a finite set of possible values) or continuous (where they are real or integer). In the case of categorical attributes, all possible values are mapped to a set of positive integers. For continuous attributes, a discretization method is used.

Definition 1 A row or a training object in T can be described as a combination of attribute names A_i and values a_{ij}, plus a class denoted by c_j.

Definition 2 An item can be described as an attribute name A_i and a value a_i, denoted $\langle (A_i, a_i) \rangle$.

Definition 3 An itemset can be described as a set of disjoint attribute values contained in a training object, denoted $\langle (A_{i1}, a_{i1}), \ldots, (A_{ik}, a_{ik}) \rangle$.

Definition 4 A *ruleitem* r is of the form $\langle itemset, c \rangle$, where $c \in C$ is the class.

Definition 5 The actual occurrence (*actoccr*) of a *ruleitem* r in T is the number of rows in T that match the itemset of r.

Definition 6 The support count (*suppcount*) of *ruleitem* r is the number of rows in T that match the itemsets of r, and belong to the class c of r.

Definition 7 The occurrence of an itemset i (*occitm*) in T is the number of rows in T that match i.

Definition 8 An itemset i passes the *minsupp* threshold if $(occitm(i)/|T|) \geq minsupp$.
Definition 9 A *ruleitem* r passes the *minsupp* threshold if $(suppcount(r)/|T|) \geq minsupp$.
Definition 10 A *ruleitem* r passes the *minconf* threshold if $(suppcount(r)/actoccr(r)) \geq minconf$.
Definition 11 Any *itemset* i that passes the *minsupp* threshold is said to be a *frequent itemset*.
Definition 12 Any *ruleitem* r that passes the *minsupp* threshold is said to be a *frequent ruleitem*.
Definition 13 A CAR is represented in the form: $(A_{i1}, a_{i1}) \wedge \ldots \wedge (A_{ik}, a_{ik}) \rightarrow c$, where the left-hand side (antecedent) of the rule is an itemset and the consequent is a class.

A classifier is a mapping form $H : A \rightarrow Y$, where A is a set of itemsets and Y is the set of classes. The main task of AC is to construct a set of rules (model) that is able to predict the classes of previously unseen data, known as the test data set, as accurately as possible. In other words, the goal is to find a classifier $h \in H$ that maximizes the probability that $h(a) = y$ for each test object.

2.2 Solution scheme and example

An AC task is different from association rule discovery. The most obvious difference between association rule discovery and AC is that the latter considers only the class attribute in the rules consequent. However, the former allows multiple attribute values in the rules consequent. Table 1 shows the main important differences between AC and association rule discovery, where overfitting prevention is essential in AC, but not in association rule discovery as AC involves using a subset of the discovered set of rules for predicting the classes of new data objects. Overfitting often occurs when the discovered rules perform well on the training data set and badly on the test data set. This can be due to several reasons such as a small amount of training data objects or noise.

The problem of constructing a classifier using AC can be divided into four main steps, as follows.

- Step 1: The discovery of all frequent *ruleitems*.
- Step 2: The production of all CARs that have confidences above the *minconf* threshold from frequent *ruleitems* extracted in Step 1.
- Step 3: The selection of one subset of CARs to form the classifier from those generated at Step 2.
- Step 4: Measuring the quality of the derived classifier on test data objects.

Figure 1 shows the general steps used in the AC approach, in which the first step is computationally expensive because it is similar to the discovery of frequent itemsets in association rule discovery, which is a challenging problem (Agrawal & Srikant, 1994; Savasere *et al*., 1995; Zaki *et al*., 1997; Han *et al*., 2000; Lim *et al*., 2000). Methods that find the complete set of frequent *ruleitems*, generally separate *ruleitems* that are potentially frequent and then work out their frequencies in the training data set (Step 1). Once all frequent *ruleitems* are identified, for each *ruleitem* that passes the *minconf* threshold, a single rule is generated of the form, $X \rightarrow C$, where C is the largest frequency class associated with itemset X in the training data set (Step 2).

The problem of generating the classification rules is straightforward, once all frequent *ruleitems* are identified, as no support counting or scanning of the training data set is required. In Step 3, a selection of an effective ordered subset of rules is accomplished using various methods discussed in this paper, and in the final step the quality of the selected subset is measured on an independent test data set.

Let us explain the discovery of frequent *ruleitems* and the construction of the classifier in AC using an example. Consider the training data set shown in Table 2, which represents three attributes, A_1 (a_1, b_1, c_1), A_2(a_2, b_2, c_2), and A_3(a_3, b_3, c_3), and two classes (y_1, y_2). Assuming $minsupp = 20\%$ and $minconf = 80\%$, the frequent one, two and three *ruleitems* for Table 1 are shown in Table 3, along with the relevant supports and confidences. In cases where a *ruleitem* is associated with multiple classes, only the class with the largest frequency is considered by most current AC methods. Frequent *ruleitems* in bold in Table 3 represent those that pass the confidence and support thresholds, which are converted into rules. Finally, the classifier is constructed using an ordered subset of these rules.

Table 1 The main differences between AC and association rule discovery

Association rule discovery	Associative classification
No class attribute involved (unsupervised learning)	A class must be given (supervised learning)
The aim is to discover associations between items in a transactional database	The aim is to construct a classifier that can forecast the classes of test data objects
There could be more than one attribute in the consequent of a rule	There is only attribute (class attribute) in the consequent of a rule
Overfitting is usually not an issue	Overfitting is an important issue

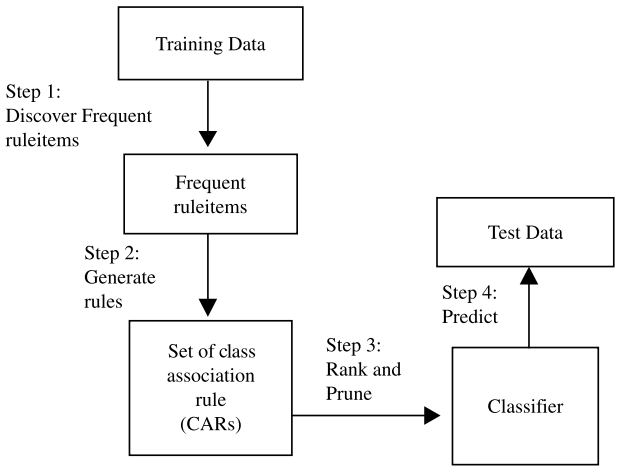

Figure 1 Associative classification steps

3 Discovery techniques for frequent ruleitems

The step of finding frequent *ruleitems* in AC is a hard step that necessitates large amounts of computation (Liu *et al*., 1998, 2000; Li *et al*., 2001). Several different approaches to discover frequent ruleitems from a data set have been adopted from association rule discovery. For example, some AC methods (Liu *et al*., 1998; Meretakis & Wüthrich, 1999; Zaïane & Antonie, 2002) employ the Apriori candidate generation method (Agrawal & Srikant, 1994). Other AC methods (Li *et al*., 2001; Baralis & Torino, 2002; Baralis *et al*., 2004) adopt the FP-growth approach (Han *et al*., 2000) and algorithms such as CPAR (Yin & Han; 2003) use a greedy strategy presented in FOIL (Quinlan & Cameron-Jones, 1993). Finally, AC algorithms (Thabtah *et al*., 2004, 2005) extend tid-lists intersections methods of vertical association rule data layout (Zaki *et al*., 1997; Zaki & Gouda, 2003) to solve classification benchmark problems. This section focuses on these different approaches to discover *ruleitems*.

3.1 Candidate generation

In the Apriori association rule discovery algorithm (Agrawal & Srikant, 1994), the discovery of frequent itemsets is accomplished in levels, where in each level Apriori uses itemsets found to be frequent in the previous level to produce new candidate itemsets. Apriori utilizes the 'downward-closure' property with the aim of speeding up the search process by reducing the number of candidate itemsets at any level. The 'downward-closure' property ensures that all subsets of a frequent itemset must be frequent as well. If an itemset is infrequent at any level, it will be removed because any addition of items to it will not make it frequent. Apriori uses this property to prune candidate itemsets that have infrequent subsets before counting their support at any

Table 2 Training data set

rowids	A_1	A_2	A_3	class
1	a_1	a_2	b_3	y_1
2	a_1	a_2	c_3	y_2
3	a_1	b_2	b_3	y_1
4	a_1	b_2	b_3	y_2
5	b_1	b_2	a_3	y_2
6	b_1	a_2	b_3	y_1
7	a_1	b_2	b_3	y_1
8	a_2	a_2	b_3	y_1
9	c_1	c_2	c_3	y_2
10	a_1	a_2	b_3	y_1

Table 3 Potential classifier for Table 2

Frequent ruleitems			
Antecedent	Consequent/class	Supp	Conf
$\langle a_2, b_3\rangle$	y_1	4/10	4/4
$\langle a_1, a_2b_3\rangle$	y_1	3/10	3/3
$\langle b_3\rangle$	y_1	6/10	6/7
$\langle a_1, b_3\rangle$	y_1	5/10	5/6
$\langle a_2\rangle$	y_1	4/10	4/5
$\langle a_1, a_2\rangle$	y_1	3/10	3/4
$\langle a_1\rangle$	y_1	5/10	5/7

level. This should reduce the time to produce and compute the support for all items combinations in the transactional database.

The CBA algorithm was one of the first AC algorithms that employed an Apriori candidate generation step to find the rules. After its introduction, many other algorithms adopted its approach, see, for example, Meretakis and Wüthrich (1999), Liu *et al*. (2001), Zaïane & Antonie (2002), Antonie & Zaïane (2004) and Xu *et al*. (2004). The bottleneck of Apriori candidate generation is the task of finding frequent itemsets from all possible candidate itemsets at each level. AC techniques that use the Apriori candidate generation step to discover frequent *ruleitems*, generally achieve good performance when the size of the candidate *ruleitems* is small. In circumstances involving highly correlated classification data sets with many attributes, and a very low support threshold, the potential number of candidate *ruleitems* at each level can be enormous and these algorithms may consume considerable CPU time and storage.

For instance, the CBA algorithm requires a complete pass over the training data set to find candidate *ruleitems* at any level and therefore, to find candidate *ruleitems* at any level, a merge of all possible combinations of frequent *ruleitems* found in the previous level and a complete scan of the training data set to update frequencies of candidate *ruleitems* at the current level are performed. This process, of repeatedly scanning the database, is costly with regards to processing time.

Furthermore, AC algorithms often experience exponential growth in the number of rules. This is because of the association rule discovery approaches used, which explore all possible associations between attribute values in a database. This particular problem is clearly seen when comparing the size of rule induction classifiers with those produced by AC methods. This is made apparent by various researches (Liu *et al*., 2000; Li *et al*., 2001), which point out that the increased number of rules may cause serious problems such as overfitting the training data set and misleading classification (occurring when multiple rules in the classifier cover a single test object and have different class labels), as well as high CPU and memory requirement.

3.2 Frequent pattern tree

To improve the efficiency of the Apriori candidate generation step, at least three AC methods, CMAR, L^3 and L^3G (Baralis *et al*., 2004), use approaches based on the frequent pattern (FP)-growth method (Han *et al*., 2000) to discover rules. The FP-growth method builds a highly dense FP-tree for the training data set, where each training object is represented by at most one path in the tree. As a result, the length of each path is equal to the number of the frequent items in the transaction representing that path. This type of representation is very useful for the following reasons. (1) All of the frequent itemsets in each transaction of the original database are given by the FP-tree, and because there is much sharing between frequent items, the FP-tree is smaller in size than the original database. (2) The FP-tree construction requires only two database scans, where in the first scan, frequent itemsets along with their support in each transaction are produced, and in the second scan, the FP-tree is constructed.

Once the FP-tree is built, a pattern growth method is used to find the rules by using patterns of length one in the FP-tree. For each frequent pattern, all other possible frequent patterns co-occurring with it in the FP-tree (using the pattern links) are generated and stored in a conditional FP-tree. The mining process is performed by concatenating the pattern with those produced from the conditional FP-tree. The mining process used by the FP-growth algorithm is not Apriori-like in that there is no candidate rule generation. One primary weakness of the FP-growth method is that there is no guarantee that the FP-tree will always fit in the main memory, especially in cases where the mined database is dimensionally large.

The CMAR, L^3 and L^3G algorithms store rules in a prefix tree data structure known as a CR-tree. The CR-tree is used to store rules in descending order according to the frequency of attribute values appearing in the antecedent. Once a rule is generated, it will be inserted into the CR-tree as a path from the root node, and its support, confidence and class are stored at the last node in the path. When a candidate rule that has common features with an already existing rule in the tree is inserted, the path of the existing rule in the tree is extended to reflect the addition of the new rule.

Algorithms that use the CR-tree consider the common attribute values contained in the rules, which use less storage if compared with CBA. In addition, rules can be retrieved efficiently as CR-tree indices rules. Experimental tests reported in Li *et al*. (2001) on different data sets from the University of California (UCI) data collection (Merz & Murphy, 1996) show that the classifiers generated by CMAR are more accurate than those of C4.5 (Quinlan, 1993) and the CBA on 50% of the benchmark problems considered. Furthermore, the results revealed that 50–60% space can be saved in the main memory using the CR-tree when compared with the CBA.

3.3 Greedy approach

For each class in the training data set, the FOIL algorithm (Quinlan & Cameron-Jones, 1993) builds rules heuristically from training literals, also known as items, using the FOIL-gain method. The FOIL-gain method measures the information gained from adding a condition to the current rule. For class c, assume that $|P'|$ positive data objects and $|N'|$ negative data objects in the training data set are associated with it. Positive data objects for c are those training objects that contain c, whereas c negative data objects are those training objects where class c never occurs. FOIL starts constructing a rule for each class (c) by adding a literal (l) into its antecedent. After adding l, there will be $|P|$ positive and $|N|$ negative training data objects that match the current rule, $l \rightarrow c$:

$$\text{FOIL} - \text{gain}(l) = |P|\left(\log\frac{|P|}{|P|+|N|} - \log\frac{|P'|}{|P'|+|N'|}\right). \tag{3.1}$$

The FOIL algorithm seeks the literal that yields the largest FOIL-gain for a particular class in the training data set. Once that literal is identified, all training objects associated with it are discarded and the process is repeated until positive data objects for the current class are covered. At that point, another class is selected and the same process is repeated for it, and so forth.

For multi-class classification problems, FOIL divides the training data set according to class labels. It may be argued that FOIL generates local classifiers rather than global ones, as rules are greedily extracted from different local parts of the training data set instead of the whole set (Yin & Han, 2003). Predictive quality is potentially limited too, as selecting only a single literal may not always be the best choice because several literals could have close FOIL-gain values.

An AC technique called CPAR has been proposed, which improves upon the FOIL strategy for generating rules. CPAR differs from FOIL in that it does not remove all data objects associated with the literal once it is determined; instead, weights of data objects associated with the literal are reduced by a multiplying factor and the process is repeated until all positive data objects for the current class are covered. This weighted version of FOIL extracts more rules, as a training object is allowed to be covered by more than just a single rule, similar to association rule discovery approaches.

The search process for the best rule condition is a time-consuming process for CPAR, because the FOIL-gain for every possible literal needs to be calculated in order to determine the best literal gain. As a result, CPAR uses a PNArray (Gehrke *et al*., 1998) data structure to store the necessary information for the process of calculating each literal FOIL-gain. PNArray stores the numbers of positive and negative data objects that match the current rule's condition before adding the literal and then after appending the literal to the rule. Finally, CPAR derives not only the best FOIL-gain literal, but also similar ones close to it, solving the problem of multiple literals with similar FOIL-gain measurements. Empirical studies conducted using three popular classification algorithms (C4.5, RIPPER, CBA) in Yin & Han (2003) against data sets from UCI data collection have shown that CPAR achieves on average +2.24%, +1.83% and +0.48% higher classification accuracies than RIPPER, C4.5 and CBA, respectively.

3.4 Confidence rules

The support threshold is the key to success in both association rule discovery and AC approaches to data mining. However, for certain application data, some rules with large confidence are ignored simply because they do not have enough support. Classic AC algorithms use one support threshold to control the number of rules derived and may be unable to capture high confidence rules that have a low support. In order to explore a large search space and to capture as many high confidence rules as possible, such algorithms commonly tend to set a very low support threshold, which may give rise to problems such as overfitting, generation of statistically low support rules and a large number of candidate rules with high CPU time and storage requirements.

In response to this issue, an approach that suspends the support threshold and uses only the confidence threshold to discover rules has been proposed in Wang *et al*. (2000, 2001). This confidence-based approach aims to extract all rules in a data set that pass the *minconf* threshold. Without the support threshold, the candidate generation step is no longer applicable and the 'downward-closure' property (Agrawal *et al*., 1993) that has been heavily employed by support-based techniques is also invalid. This is because the downward-closure property depends on the support threshold to prune infrequent candidate itemsets and to avoid unnecessary support computation, and because the support threshold is suspended, thus the downward-closure property cannot be used. It is necessary to introduce an analogous property to downward-closure, in order to keep the mining process efficient and scalable.

As a result, a new property called 'existential upward-closure' has been introduced in the AC approach based on a decision tree called the association-based decision tree algorithm (ADT; Wang *et al*., 2000). The best way to explain the existential upward-closure property is by using an example. Consider the following three rules:

R_1, income is high then credit-card = yes;
R_2, income is high and age $>$ 55 then credit = yes;
R_3, income is high and age $\leq$ 55 then credit = yes.

Assume that R_1 has a 70% confidence. A rule $\langle(X,x),(A_i, a_i)\rangle \rightarrow c$ is an A_i-specialization of $(X,x) \rightarrow c$ if a_i is a value of A_i. Because R_2 and R_3 are specializations of R_1 and are mutually exclusive, then if one condition implies negative confidence, the other condition must imply positive confidence. Thus, at least one rule of R_2 and R_3 has as much confidence as R_1. In other words, if an attribute A_i is not in a rule R_i: $x \rightarrow c$ and R_i is confident, so is some A_i-specialization of R_i.

The ADT performs a level-wise search to find candidate confident rules where at each level the database is completely scanned. Starting from K-rules where K represents the number of non-class attributes in the database, the existential upward-closure property is used as follows. A candidate $(K-1)$ rule, R_i, is produced if for every attribute A_i not in R_i, some A_i-specialization of R_i is confident. This rule generation phase is implemented using expressions in relational algebra. No more information is given on how these expressions have been implemented.

The ADT uses pessimistic error pruning (Quinlan, 1987), which constructs a decision-tree-like structure, known as an ADT-tree, using the generated CARs and places general rules at the higher levels and specific rules at the lower levels of the tree. In the prediction step, the ADT selects the highest ranked rule that matches a test object; a procedure that ensures each object has only one covering rule. The bottleneck occurs during the extraction of the huge numbers of possible candidate confidence rules, especially for high-dimensional training data sets. Several experiments conducted in Wang *et al.* (2000) against 21 data sets from the UCI data collection indicate that confidence-based AC algorithms such as the ADT scale well in terms of classification accuracy when compared with the C4.5 and CBA algorithms. In particular, the ADT accomplished +2.8 and +1.7 higher accuracy rates than C4.5 and the CBA, respectively, on the data sets considered.

3.5 Multi-support approach

In some classification data, class labels are unevenly distributed, causing the generation of many rules for the dominant classes and few and sometimes no rules for the minority classes. Using one global support threshold may be ineffective for data sets with uneven class frequencies because when the user sets the *minsupp* threshold above some class label frequencies, there will be no rules retrieved for such classes, and some high confidence rules may be discarded.

To treat such a shortcoming, extensions to some of the existing AC approaches, such as the CBA and CMAR, have been developed. These extensions have resulted in a new approach that considers the class distribution frequencies in a data set, and assigns a different support threshold to each class. For instance, the CBA (2) multiple support algorithm (Liu *et al.*, 2000) modifies the original CBA algorithm to employ multiple class supports by assigning a different support threshold to each class in the training data set based on the classes frequencies. This assignment is done by distributing the global support threshold to each class corresponding to its number of occurrences in the training data set, and thus considers the generation of rules for class labels with low frequencies in the training data set.

Another iterative multi-support approach, which analyses the result of the previous rule extraction cycle and updates the support threshold for each class, has been proposed in Baralis & Torino (2002). This iterative approach initially assigns a support threshold to each available class as in Liu *et al.* (2000). Then by analysing the number of rules generated at specific rule generation cycles, supports for class labels with a low number of rules are lowered by a multiplying factor during the next cycle. Consequently, this ensures that sufficient numbers of rules are produced for each class.

These multiple support AC techniques use either Apriori or FP-growth methods to find frequent *ruleitems* and are quite similar to an earlier developed association rule discovery algorithm called MSapriori (Liu *et al.*, 1999). In fact, MSapriori was one of the first association rule algorithms that introduced the idea of using multiple supports to solve the problem of discarding rare items in databases. An empirical study (Liu *et al.*, 2000) using 34 classification benchmarks from the UCI data collection revealed that, on average, the error rate of CBA (2)

is lower than that of the CBA and C4.5. In particular, the CBA (2) won–loss–tied records against the CBA and C4.5 are 19–13–2 and 20–12–2, respectively.

3.6 Multipass atomic rules using dynamic support and adaptive confidence

A recently proposed approach called classification based on atomic association rules (CAAR; Xu *et al*., 2004) mines only atomic CARs from image block data sets. An atomic rule takes the form of $I \rightarrow C$, where the antecedent contains a single item. CAAR has been designed for image block classification data sets, although its authors claim that it could be adopted to other classification data sets, which were not supported in the experimental tests. CAAR builds the classifier in multiple passes, where in the first pass, it scans the data set to count the potential atomic rules (*ruleitems* of length 1), which are then hashed into a table. The algorithm generates all atomic rules that pass the initial support and confidence thresholds given by the end-user.

A pruning step similar to database coverage heuristic (see Section 4.3) is used after the generation of all rules in the first scan in order to keep only atomic rules that cover at least one training data object. All training objects that were covered by the discovered atomic rules in the first pass are removed and the uncovered training objects are put to a new cycle. At that point, the support threshold becomes $minsupp_i = minsupp \times (D_1/D_0)$, where D_0 and D_1 correspond to the numbers of training objects not covered by the produced rules at iterations 0 and 1, respectively. Furthermore, CAAR employs a self-adaptive confidence, $minconf = \text{COEF} \times MaxConf$, where COEF is a user-defined threshold (the authors have used COEF = 0.92 in the experiments) and *MaxConf* is the maximum confidence of all potential atomic rules in each pass. The algorithm continuously discovers potential atomic rules and prunes them in each iteration until the original training data set D_0 becomes empty. At that point, all atomic rules are combined to form the classifier.

A comparison test of CAAR, CBA and C4.5 on a specific image block data set showed that CAAR is able to classify test data objects using cross-validation more accurately than the CBA and C4.5 algorithms.

3.7 The intersections of training objects locations

In general, there are two common representations of a target database in association rule discovery; these are the horizontal (Agrawal & Srikant, 1994) and vertical (Zaki *et al*., 1997) layouts. In the horizontal layout, the database consists of a group of transactions, where each transaction has an identifier followed by a list of items contained in that transaction. Table 4 illustrates a horizontal layout representation for a transactional database. In searching for frequent itemsets in the horizontal layout, the database is scanned multiple times, once during each iteration, to perform support counting and pattern matching for candidate itemsets at each level. Furthermore, computational overheads occur during support counting of candidate itemsets. According to Zaki *et al*. (1997), for each transaction with length l, during an iteration n, one needs to produce and evaluate whether all $\binom{l}{n}$ n-subsets of the transaction are contained in the current candidate list. In the vertical layout however, the database consists of a group of items where each item is followed by its tid-list (Savasere *et al*., 1995) as shown in Table 5, which is a vertical representation of Table 4. A tid-list of an item is the transaction numbers (tids) in the database that contain that item.

Supports of frequent itemsets are computed in the vertical layout by simple intersections of the tids. For instance, the supports of candidate itemsets of size k can be easily obtained by intersecting the tid-lists of any two $(k-1)$ subsets. The tid-lists that hold all the information related to items in the database are a relatively simple and easy to maintain data structure, and thus there is no need to scan the database during each iteration to obtain the supports of new candidate itemsets, saving I/O time (Zaki *et al*., 1997; Zaki & Gouda, 2003).

To the best of the author's knowledge, the majority of current AC algorithms adopt the horizontal data format and the first AC algorithm to utilize the vertical data layout to perform simple intersections among frequent itemsets tid-lists is MCAR (Thabtah *et al*., 2005). MCAR improves

Table 4 Horizontal representation of database

Tid	Items			
1	bread	milk	juice	
2	bread	juice	milk	
3	milk	beer	bread	juice
4	milk	eggs	bread	
5	beer	basket	bread	juice

Table 5 Vertical tid-list representation of database

Basket	Beer	Bread	Eggs	Juice	Milk
5	3	1	4	1	1
	5	2		2	2
		3		3	3
		4		5	4
		5			

the efficiency of the rule discovery phase by employing a method that extends the tid-list intersection methods of Savasere *et al.* (1995) and Zaki *et al.* (1997) to handle classification benchmark problems. Using the tid-list for an itemset in association rule discovery is a good approach as the cardinality of the itemset tid-list divided by the total number of the transactions gives the support for that itemset. However, the tid-list intersection methods presented in association rule discovery need to be modified in order to treat classification problems, where classes associated with each itemset (rule antecedent) are considered when computing the support.

The frequent itemsets discovery method employed by MCAR scans the training data set to count the occurrences of one-itemset, from which it determines those that hold enough support. During the scan, frequent one-itemsets are determined, and their occurrences in the training data set (tids) are stored inside an array in a vertical format. Also, classes and their frequencies are stored in an array. Any one-itemset that fails to pass the support threshold is discarded. The result of a simple intersection between the tid-lists of two itemsets gives a set, which holds the tids where both itemsets occur together in the training data set. This set along with the class array, which holds the class label frequencies and was created during the first scan, can be used to compute the support and confidence of the new $\langle$*itemset, class*$\rangle$ (*ruleitem*) resulting from the intersection.

To show how a frequent *ruleitem* is determined, consider for example itemsets $\langle(A_1, x_1)\rangle$ and $\langle(A_2, y_1)\rangle$ in Table 6; the following two sets represent the tids in which they occur, {1, 2, 3, 4, 8} and {1, 3, 5, 6, 10}. We can determine the support of the itemset $\langle(A_1, x_1), (A_2, y_1)\rangle$ by intersecting the tids 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 $\langle(A_1, x_1), (A_2, y_1)\rangle$, i.e. 2/10. If it passes the *minsupp* threshold, then we proceed by checking whether there is some class c in the class array such that $\langle(A_1, x_1), (A_2, y_1), c\rangle$ passes the *minsupp* threshold; otherwise we prune it.

4 Pruning techniques

AC algorithms normally derive a large set of rules (Topor & Shen, 2001; Li *et al.*, 2001) because (1) classification data sets are typically highly correlated and (2) association rule discovery methods are used for rule discovery. As a result, there have been many attempts to reduce the size of classifiers produced by AC approaches, mainly focused on preventing rules that are either redundant or misleading from taking any role in the prediction process of test data objects. The removal of such rules can make the classification process more effective and accurate.

Table 6 Training data 1

Rowid	A_1	A_2	Class
1	x_1	y_1	c_1
2	x_1	y_2	c_2
3	x_1	y_1	c_2
4	x_1	y_2	c_1
5	x_2	y_1	c_2
6	x_2	y_1	c_1
7	x_2	y_3	c_2
8	x_1	y_3	c_1
9	x_2	y_4	c_1
10	x_3	y_1	c_1

Several pruning methods have been used effectively to reduce the size of associative classifiers, some of which have been adopted from decision trees, such as pessimistic estimation, others from statistics such as chi-square (χ^2) testing (Snedecor & Cochran, 1989). These pruning techniques are utilized during the construction of the classifier; for instance, a very early pruning step, which eliminates *ruleitems* that do not pass the support threshold, may occur in the process of finding frequent *ruleitems*. Another pruning approach such as χ^2 testing may take place when generating the rules, and late pruning methods, such as database coverage, may be carried out while building the classifier. Throughout this section, pruning techniques used in AC are discussed.

4.1 χ^2 testing

χ^2 testing is a well-known discrete data hypothesis testing method from statistics, which evaluates the correlation between two variables and determines whether they are independent or correlated (Snedecor & Cochran, 1989). The test for independence, when applied to a population of subjects, determines whether they are positively correlated or not, i.e.

$$\chi^2 = \sum_{i=1}^{k} \frac{(f_i - e_i)^2}{e_i} \tag{4.1}$$

where e_i represents the expected frequencies and f_i represents the observed frequencies. When the expected frequencies and the observed frequencies are notably different, the hypothesis that they are correlated is rejected.

This method has been used in AC to prune negatively correlated rules. For example, a test can be carried out on every discovered rule, such as $R : x \rightarrow c$, to find out whether the condition x is positively correlated with the class c. If the result of the test is larger than a particular constant, there is a strong indication that x and c of R are positively correlated, and therefore R will be stored as a candidate rule in the classifier. If the test result indicates negative correlation, R will not take any part in the later prediction step and is discarded. The CMAR algorithm adopts the χ^2 testing in its rules discovery step. When a rule is found, CMAR tests whether its body is positively correlated with the class. If a positive correlation is found, CMAR keeps the rule, otherwise the rule is discarded.

4.2 Redundant rule pruning

In AC, all attribute value combinations are considered in turn as a rule's condition. Therefore, rules in the resulting classifiers may share training items in their conditions, and for this reason there could be several specific rules containing many general rules. Rule redundancy in the classifier is unnecessary and in fact could be a serious problem, especially if the number of discovered rules is extremely large.

```
Give a set of generated rules R, and the training data set T, the
database coverage process works as follows:

Rank R in descending order based on a ranking procedure (see
Section 5.3)

For each rule r1 in R Do
    Find all applicable data instances in T that match r1•s
condition
    If r1 correctly classifies a training instance in T
        Mark r1 as a candidate rule
        Remove all instances in T covered by r1
    end if
    If r1 cannot correctly cover any training instance in T
          Remove, r1 from R
    end if
end
```

Figure 2 Database coverage method

A pruning method that discards specific rules with fewer confidence values than general rules, called redundant rule pruning, has been proposed in CMAR. The redundant rule pruning method works as follows. Once the rule generation process is finished and rules are sorted, an evaluation step is performed to prune all rules such as $I' \rightarrow c$ from the set of generated rules, where there is some general rule $I \rightarrow c$ of a higher rank and $I \subseteq I'$. This pruning method may reduce the size of the resulting classifiers and minimizes rule redundancy.

Algorithms, such as those in Li *et al.* (2001), Antonie *et al.* (2003) and Antonie & Zaïane (2004), have used redundant rule pruning. They perform such pruning immediately after a rule is inserted into the compact data structure, the CR-tree. When a rule is added to the CR-tree, a query is issued to check if the inserted rule can be pruned or some other already existing rules in the tree can be removed.

4.3 Database coverage

The database coverage method, which is illustrated in Figure 2, is a pruning technique, used in AC (Liu *et al.*, 1998), and usually invoked after rules have been created and sorted. This method tests the generated rules against the training data set, and only rules that cover at least one training object not considered by a higher ranked rule are kept for later classification. For each ordered rule starting from the top ranked one (r_1), a pass over the training data set to find all objects that match the r_1 body is performed. Once r_1 training objects are located, then they will be removed and r_1 will be inserted into the classifier. The process is repeated for the remaining ordered rules until all training objects are covered or all ordered rules have been considered. If an ordered rule is unable to cover at least a single training object, then it will be discarded. The database coverage method was created by the CBA and then latterly used by other algorithms, including those in Liu *et al.* (2001), Li *et al.* (2001) and Thabtah *et al.* (2005).

4.4 Pessimistic error estimation

In general, there are two pruning strategies in decision trees, pre-pruning and post-pruning (Witten & Frank, 2000). The latter, also called backward pruning, is more popular and has been used by many decision tree algorithms such C4.5 and See5 (Quinlan, 1998). In performing backward pruning, the tree is first completely constructed, then at each node a decision is made whether to replace a node and its descendents with a single leaf or to leave the node unchanged. The decision whether to replace a node or not is made by calculating the estimated error using the pessimistic error estimation measure (Quinlan, 1987) of a particular node and comparing it with its potential replacement leaf. The method of replacing a sub-tree with a leaf node is called sub-tree replacement. The error is estimated using a pessimistic error estimation measure based on the training objects.

The probability of error at a node v is

$$q(v) = \frac{N_v - N_{v,c} + 0.5}{N_v} \tag{4.2}$$

where N_v is the number of training data objects at node v and $N_{v,c}$ is the number of training data objects associated with the majority class at node v. The error rate at a sub-tree T is

$$q(T) = \frac{\sum_{l \in leafs(T)} N_l - N_{l,c} + 0.5}{\sum_{l \in leafs(T)} N_l}. \tag{4.3}$$

The sub-tree T is pruned if $q(v) \leq q(T)$.

In addition to using the pessimistic error rate in decision tree algorithms, it can be also used in AC by comparing the estimated error of a new rule, r_i, resulting from the deletion of one item in the condition of the original rule, r_j. If the expected error of r_i is lower than that of r_j, then r_j will be replaced by r_i. AC algorithms, including those in Liu *et al*. (1998) and Wang *et al*. (2000), have used pessimistic error estimation to effectively cut down the number of extracted rules.

4.5 *Lazy pruning*

Some AC techniques (Baralis & Torino, 2002; Baralis *et al*., 2004) raise the argument that pruning classification rules should be limited to only 'negative' rules (those that lead to incorrect classification). In addition, it is claimed that database coverage pruning often discards some useful knowledge, as the ideal support threshold is not known in advance. Because of this, these algorithms have used a late database coverage-like approach, called lazy pruning, which discards rules that incorrectly classify training objects and keeps all others.

Lazy pruning occurs after rules have been created and stored, where each training object is taken in turn and the first rule in the set of ranked rules applicable to the object is assigned to it. The training object is then removed and the correctness of class labels assigned to the object is checked. Once all training objects have been considered, only rules that wrongly classified training objects are discarded and their covered objects are put into a new cycle. The process is repeated until all training objects are correctly classified. The results are two levels of rules: the first level contains rules that classified at least one single training object correctly and the second level contains rules that were never used in the training phase. The main difference between lazy pruning and database coverage pruning is that the second level rules that are held in the memory by lazy pruning are completely removed by the database coverage method during the rule pruning step. Furthermore, once a rule is applied to the training objects, all objects covered by the rule are removed (negative and positive) by the database coverage method.

Experimental tests reported in Baralis *et al*. (2004) using 26 different data sets from the UCI data collection have shown that methods that employ lazy pruning, such as L^3 and L^3G, may improve classification accuracy on average by 1.63% over other techniques that use database coverage pruning. However, lazy pruning may lead to very large classifiers, which makes it difficult for a human to understand or interpret. In addition, the experimental tests indicate that lazy pruning algorithms consume more memory than other AC techniques and, more importantly, they may fail if the support threshold is set to a very low value as a result of the very large number of potential rules.

4.6 *Conflicting rules*

For highly dense classification data sets and other data where there could be multiple class labels associated with each training object, it is possible to produce rules with the same body that predict different classes. For example, given two rules such as $x \rightarrow c_1$ and $x \rightarrow c_2$, Antonie & Zaine (2003) proposed a pruning method that considers these two rules as conflicting. Their method removes conflicting rules and disallows them from taking any role in classifying test data objects. However,

a recent proposed algorithm called MMAC (Thabtah *et al.*, 2004) showed using experiments that such rules represent useful knowledge as they pass support and confidence requirements. Thus, domain experts can profit from them. MMAC developed a recursive learning phase that combines so-called conflicting rules into one multi-label rule. For the above example, MMAC combines the two rules into the following multi-label rule, $x \rightarrow c_1 \vee c_2$ and assigns an appropriate weight to each class label in the rule consequent according to its frequency with the rule antecedent (itemset) in the training data set.

4.7 Impact of pruning on classifiers

In association rule discovery and AC mining, a transaction or a training object can be used to generate many rules; therefore, there are tremendous numbers of potential rules. Without adding constraints on the rule discovery and generation phases or imposing appropriate pruning, the very large numbers of rules, usually of the order of thousands or even tens of thousands, make humans unable to understand or maintain the outcome. Pruning noisy and redundant rules becomes an important task.

AC algorithms that use pruning methods such as database coverage and redundant rule prefer general rules over specific ones; thus, they produce smaller classifiers than other techniques that adopt lazy pruning. Experiments were conducted on the 'german' and 'wine' data sets downloaded from Weka (2000) to compare the lazy pruning algorithm, L^3, and the database coverage approach of the CBA with reference to the number of rules and accuracy. The L^3 results on both data sets have been derived using a *minsupp* of 1% and a *minconf* 0.0% and as a result the experiments of the CBA were run using the same support and confidence thresholds for fair comparison.

The numbers of rules produced by L^3 on the 'german' and 'wine' data sets are 175 365 and 40 775, respectively, with prediction accuracies of 72.50% and 95.00%, respectively. By comparison, the CBA derives only 325 and 12 rules from the same data sets, with prediction accuracies of 73.58% and 98.33%, respectively. These results provide direct, if limited, evidence that techniques that use database coverage pruning tend to choose general rules and simpler classifiers, which sometimes are more accurate on test datasets when compared with lazy pruning methods such as L^3. Further results can be found in a recent paper, which studies the effect of pruning on different AC algorithms (Thabtah, 2006).

Overall, techniques that derive smaller classifiers are generally preferred by human experts because of their ease of manual maintenance and interpretability. For instance, if general practitioners used their patient data to build a rule-based diagnosis system, they would prefer the resulting number of rules to be small and simple. As such, they may even slightly decrease accuracy in exchange for a more concise set of rules, which human experts can understand. Smaller classifiers do however suffer from some drawbacks, including their sensitivity to low-quality data (data sets that contain redundant information and missing values) and their inability to cover the whole training data set (Hu & Li, 2005).

However, approaches that produce very large numbers of rules, such as L^3, usually give slightly improved predictive power, but spend a long time in training and during the prediction of test objects, as they must pass over a very large number of rules when classifying the test data set. In the L^3 algorithm, rules which cover no training data objects are known as spare or secondary rules. Holding a very large number of spare rules to cover a limited number of test objects missed by the primary rules is inefficient. There should be a trade-off between the size of the classifiers and the predictive accuracy, especially where slightly lower accuracy can be tolerated in exchange for a more concise set of rules.

5 Approaches to rule ranking

Rule sorting before building the classifier plays an important role in the classification process because the majority of AC algorithms, such as those in Wang *et al.* (2000), Liu *et al.* (1998),

Baralis & Torino (2002), Li *et al*. (2001), Baralis *et al*. (2004) and Thabtah *et al*. (2004, 2005) utilize rule ranking procedures as the basis for selecting the classifier during pruning. In particular, the CBA and CMAR algorithms, for example, use database coverage pruning to build their classifiers, where this pruning tests rules according to the rule ranking procedure. Therefore, the highest-order rules are generally evaluated first and are then inserted into the classifier and later used for predicting test data objects.

The precedence of the rules is normally determined according to several parameters, including confidence, support and rule antecedent length. This section highlights the different constraints considered by current algorithms to discriminate between rules in the rule ordering process and also discusses the impact they have on the quality of the resulting classifiers.

5.1 Support, confidence and cardinality method

One of the common rule sorting techniques, which favours rules with large support and confidence values, was introduced in the CBA, and is shown in Figure 3. The ranking technique employed by the CBA considers principally confidence and support to order rules, and when two rules have identical support and confidence, the choice is based on that generated earlier. This means that the CBA selects rules with lower antecedent cardinality first as it employs the Apriori step-wise algorithm. The Apriori algorithm generates rules starting from those that have length 1, then 2, and so on.

The CBA sorting method fails to break many ties for highly correlated classification data sets, where the expected number of the produced rules is relatively large. For example, for the 'vote' data set downloaded from Weka (2000) and using *minsupp* of 2% and *minconf* of 40%, there are 8208 potential rules with identical confidence, of which 6975 have the same support. Also, from the 6975 potential rules with identical confidence and support, there are 5204 that have the same length and only 1164 potential rules which have different lengths. These numbers of potential rules have been produced without using any pruning. The remaining 4040 rules are ranked randomly if we use the CBA rule sorting method, where many rule ranking decisions may be sub-optimal, reducing the quality of the resulting classifier. Additional tie breaking conditions have the potential to improve classification accuracy over the (support, confidence, cardinality) method.

The majority of AC algorithms developed after the introduction of the CBA, including those in Wang *et al*. (2000), Li *et al*. (2001) and Antonie & Zaïane (2004) have used the (support, confidence, cardinality) rule ranking method. These algorithms tend to prefer general rules (those with very small numbers of attribute values in their antecedent) as they occur more frequently in the training data set. However, such rules may suffer from large error rates. Generally speaking, specific rules (rules with high cardinality) are supersets of some general rules and cover smaller numbers of training objects. Thus, their chance of misclassification on the training data set is usually smaller than that of general rules.

5.2 L^3 rule ranking method

A rule ranking method, which favours specific rules, especially those that have large confidence, over general rules, was developed within the L^3 algorithm. The main reason for giving specific

Give two rules, r_a and r_b, r_a precedes r_b ($r_a \rangle r_b$) 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 was generated earlier than r_b

Figure 3 CBA rule ranking method

rules higher ranking than general rules is to reduce the chance of misclassification and to try specific rules first in the prediction step; then if they fail to cover a test object, rules with a smaller number of attributes are considered.

5.3 Support, confidence, cardinality and class distribution method

In general, rule ranking in AC is based on support, confidence and cardinality of the rule's antecedent (see Section 5.1). When several rules have identical confidence, support and cardinality, AC techniques choose one of the rules randomly, which possibly in some cases may degrade accuracy. This random selection occurs frequently in mining classification data sets where certain attribute values occur frequently. Thabtah (2006) argued that ranking of rules should not be limited to support and confidence parameters because the possibility of two or more rules with similar confidence and support is relatively high. There should be other parameters to consider in favouring one rule over another in order to limit rule random selection. Thus, Thabtah (2006) proposed a rule ranking procedure shown in Figure 4 that takes into account the class distribution frequency of each rule after considering confidence, support and rule antecedent length.

This rule ranking procedure adds upon previous rule ranking approaches by looking at the class distribution frequencies in the training data set, and prefers rules that are associated with dominant classes (class labels that occur more frequently in the training data set). For example, if two rules, r_1 and r_2, have the same support, confidence and cardinality, but r_2 is associated with a class that occurred more frequently in the training data set than that of r_1, this procedure favours r_2 over r_1 in the rule ranking process. In cases where two or more rules also have identical class frequencies, then it selects one randomly. The rule random selection shown in Figure 4 considers the rule's items row ordering in the training data set and prefers rules that have a higher order. Empirical evaluations using 12 classification data sets from the Weka data collection (Weka, 2000) have revealed that adding more constraints to discriminate between rules slightly improves the accuracy of the resulting classifiers. Particularly, the rule ranking method of Thabtah (2006) improved the accuracy for the 12 classification data sets on average +0.62% and +0.40% over the (support, confidence) and (support, confidence, lower cardinality) rule ranking approaches, respectively. Moreover, the results show that the class distribution parameter has been used often for breaking ties among rules.

5.4 Impact of global ranking of rules on classifiers

Every rule-based AC technique performs global sorting on the rules before using them in the predication step. This sorting can be considered a first step toward pruning noisy and redundant rules and explains the reason why these algorithms sort rules before pruning. Sorting aims to give good quality classification rules the chance to be selected first in the process of building the classifier. Many algorithms (Liu *et al.*, 1998; Wang *et al.*, 2000; Li *et al.*, 2001; Thabtah *et al.*, 2004,

Give two rules, r_a and r_b, precedes r_b(r_a〉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 of r_b
4. Confidence, support and cardinality of r_a and r_b are the same, but r_a is associated with a more representative 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 have higher order in the training data than that of r_b.

Figure 4 Rule ranking method presented in Thabtah (2006)

2005) use the rule sorting procedure to select high confidence rules to build their classifiers. This makes rule sorting an important step, which influences the predictive quality of the classifier. As shown previously in this section, the measures used to discriminate between rules are confidence, support, rule antecedent cardinality and class distribution frequency. However, the question still remains, which rule ranking method is the most effective?

It is the firm belief of the author that if more effective conditions can be imposed to break ties during the ranking process, a better quality classifier will result. This is because the majority of AC algorithms use pruning methods such as database coverage, which consider the rule sorting procedure as the basis for selecting rules in the classifier.

6 Classification methods of test objects

Predicting the class labels of test objects is the primary aim for classification in data mining. In general, predicting the class labels of test objects in AC can be categorized into two main groups: one that makes the prediction based on the highest precedence single rule applicable to the test object and one that makes the prediction based on multiple rules. In this section, the different prediction methods used are discussed.

6.1 Maximum likelihood based prediction

When using the maximum likelihood prediction and given a classifier with a set of rules R and a test object t, only the highest precedence rule in R that matches the test object is considered. In cases where there is no applicable rule in R to cover t, then t is covered by the default class label (the dominant class label in the training data set).

There are several AC algorithms that utilize the maximum likelihood matching rule for prediction (e.g. Liu *et al*., 1998; Wang *et al*., 2000; Baralis & Torino, 2002; Baralis *et al*., 2004; Thabtah *et al*., 2005). Using the largest confidence and support rule for classification is considered an effective and simple prediction method because first, only a single rule is used for classification, and second, the highest ranked rules play the major parts in classifying test objects. Moreover, confidence can be considered as a probability measure indicating the likelihood that a test data object belongs to the right class. However, this approach has been criticized, as it is possible that there could be more than one rule applicable to a test object with similar confidence. In addition, the highest confidence rule may be ineffective, especially for data sets that have unbalanced distribution of class labels (Li *et al*., 2001; Liu *et al*., 2003). Thus, grouping a small subset of rules to make a decision is claimed to be more appropriate than using just one (Li *et al*., 2001; Zaïane & Antonie, 2002). As shown in the next section, there are different ways to make a prediction from a group of rules.

A recently developed algorithm, L^3G, modifies the maximum likelihood prediction approach. In order to limit the use of the default rule in prediction, which frequently causes misclassifications, two levels of rules have been introduced. In classifying a test object, the first level is checked and, if there is no rule applicable to the test object, instead of taking on the default class as the CBA and ADT algorithms do, the second level is checked. This process reduces the utilization of the default rule, but proves costly in processing time.

6.2 Multiple rules based prediction

In the classification process of a test object, there can be multiple applicable rules for a test object and these rules may have very close confidences, making the decision to assign only a single rule questionable. In this section, the techniques in AC that employ multiple rules prediction are reviewed.

6.2.1 Score based prediction methods

The CMAR algorithm exploits a prediction method that selects a subset of high confidence rules applicable to a test object and analyses the correlation among them, to make the prediction

decision. The correlation is measured using a weighted χ^2 analysis (Li, 2001), which examines the strength of a rule based on its support and class frequency in the training data set.

Given a classifier R and a test object t, CMAR picks up the subset of rules, R_k, in R that matches t. If all rules in R_k predict the same class, then that class will be assigned to t. In the case that rules in R_k have different class labels, CMAR then splits them into groups according to the class labels and compares the strength of each group. The support and correlation between the rules in a particular group determines the strength of that group and the class belonging to the group with the largest strength is assigned to t. The correlation is estimated for each group using weighted (χ^2) analysis in order to evaluate how positive rules are in each group. Only rules with χ^2 values above a significant level threshold are stored for later use. For a rule $R_k: P \rightarrow c$, let $Supp(c)$ denote the number of training objects associated with class label c and let $Supp(P)$ denote the number of training objects associated with itemset P. Also assume that $|D|$ denotes the total number of rows in the training data set. The weighted χ^2 denoted $Max\ \chi^2$ of R_k is defined as

$$Max\,\chi^2 = \left(\min\{\mathrm{Supp(P)}, \mathrm{Supp(c)}\} - \frac{\mathrm{Supp(p)Supp(c)}}{\mathrm{|D|}}\right)^2 |D|u \tag{6.1}$$

where

$$u = \frac{1}{Supp(P)Supp(c)} + \frac{1}{Supp(P)(|D| - Supp(c))} + \frac{1}{(|D| - Supp(P))Supp(c)} + \frac{1}{(|D| - Supp(P))(|D| - (Supp(c)))}. \tag{6.2}$$

The weighted χ^2 for each group of classes can be computed using

$$\sum \frac{\chi^2\chi^2}{Max\chi^2}. \tag{6.3}$$

A prediction method closely related to this approach has been developed in Zaïane & Antonie (2002), where the class of the subset of rules in R_k with the dominant class label is assigned to the test object t.

6.2.2 *Laplace based prediction method*

Laplace accuracy (Clark & Boswell, 1991) is mainly used in classification to estimate the expected error of a rule. This expected accuracy for a given rule, r, is given by

$$\mathrm{Laplace}(r) = \frac{(p_c(r) + 1)}{(p_{tot}(r) + m)} \tag{6.4}$$

where m is the number of class labels in the domain, $p_{\mathrm{tot}}(r)$ is the number of objects matching the r antecedent and $p_c(r)$ is the number of objects covered by r that belong to class c.

Unlike the score-based prediction procedures, the comparison between groups is performed using the Laplace expected error, where the expected accuracy for each rule is calculated before classifying a test object. The Laplace expected error has been successfully used by the CPAR algorithm, where the expected accuracy for each rule is calculated before the classification of test objects. To classify a test object t using classifier R, CPAR selects all rules in R whose antecedent satisfies t and from these the best rule for each class is determined. Then, CPAR compares the average expected accuracy of the best rules for class labels and selects the class of the rule with the highest expected accuracy to cover t. This ensures that the best expected accuracy rules for each class participate in the prediction.

6.2.3 *Discussion*

The main advantage of using multiple rules for prediction is that there is more than one rule contributing to the final decision, which greatly limits the chance of favouring a single rule to predict

all test objects satisfying its condition. However, algorithms that are based around the use of multiple rules for classifying test objects, such as CMAR and CPAR, do not consider the independence of the rules (Clark & Boswell, 1991; Hu & Li, 2005), as training objects are allowed to be covered by several rules with different class labels. This may cause rule dependency and conflicts. In other words, when a training object t is used to generate many rules during the rule discovery, then it is possible that in the prediction step, more than one rule with different class labels could be applicable to a test data object similar to t. In addition, there is rule dependency because once a rule has classified a training data object during the rule evaluation phase and that object is deleted, all other rules that have used this object during the learning phase are affected. Multiple label prediction algorithms have not addressed this issue.

The rule dependency can be explained as follows. Assume that two potential rules $r_1 : (A, a) \wedge (B, b) \rightarrow c_1$ and $r_2 :(B, b) \rightarrow c_1 \vee c_2$ can be generated from a training data set and r_1 precedes r_2, i.e. r_1 has a higher rank than r_2. Furthermore, assume that r_1 is associated with class c_1 three times, whereas r_2 is associated with class labels c_1 and c_2 five and three times, respectively. Also, assume that within the five times c_1 is associated with r_2, it has $(A,a) \wedge (B,b)$ as antecedent three times. Most of the current AC techniques, such as CBA and CMAR, generate only one class per rule; therefore r_1 will be generated regularly and all training instances associated with it will be discarded using the database coverage heuristic. In addition, only label c_1 will be considered by r_2 as it has more frequency than c_2 when associated with r_2 in the training data set. Yet because r_1 precedes r_2 and they share $(A,a,) \wedge (B,b)$ as an antecedent, the three training objects that are related to r_2 are already classified by r_1 and removed (this has been done by the database coverage heuristic). Consequently, after inserting r_1 into the classifier, class c_1 of rule r_2 would no longer be the fittest class; a new class at that point becomes the fittest class, i.e. c_2, as it has the largest representation among r_2 class labels in the training data. To the best of the author's knowledge, none of the available associative algorithms that utilize database coverage heuristic or lazy pruning considers rule dependency problems.

7 Evaluation methods

Measuring the quality of the classifiers on a test data set is an essential task in classification as this shows how effective the results are. If the results produced from the training data set accurately predict the class labels of test data objects, we simply accept them, whereas if there are several misclassifications, then we reject them. So the important question is, how can we measure the effectiveness of classification results in data mining?

There are many evaluation methods proposed in classification such as error-rate (Witten & Frank, 2000), recall-precision (Van Rijsbergan, 1979) and others. In addition and recently, new evaluation methods have been proposed in AC such as any-label and label-weight (Thabtah *et al*., 2004). In this section, these evaluation methods are discussed and compared.

7.1 General evaluation measures in classification

AC techniques use an error-rate method (Witten & Frank, 2000) to evaluate the effectiveness of their classifiers. Using this method, the classifier simply predicts the class of a test data object; if it is correct, this will be counted as a success, otherwise it will be counted as an error. The number of error cases divided by the total number of cases in a test data set gives the overall error on this data. The error-rate of a classifier on a test data set measures its predictive accuracy.

Another evaluation method in classification applications such as text categorization is precision, which has been developed with another method called recall in the information retrieval (IR) field by Van Rijsbergan (1979). Precision and recall work as follows. One starts with a collection of objects/documents and has a query. Some of the objects pertain to the query and others do not. When objects are retrieved based on the query, one may make two types of mistake: false-positives and false-negatives. Precision measures the proportion of correct answers from all those

Table 7 Possible sets of documents based on a query in IR

Iteration	Relevant	Irrelevant
Documents retrieved	X	Y
Documents not retrieved	Z	W

that were retrieved. Recall measures the proportion of correct answers retrieved from the set of all correct answers.

In general and with respect to a given query, documents can be divided into four different sets as shown in Table 7. According to Table 7,

$$\text{precision} = \frac{|X|}{|X \cup Y|} \tag{7.1}$$

and

$$\text{recall} = \frac{|X|}{|X \cup Z|}. \tag{7.2}$$

For example, consider if someone has five blue and seven red tickets in a set and he submitted a query to retrieve the blue tickets. If he retrieves six tickets where four are blue and two are red, this means that they obtained four out of five blue (one false-negative) and two red (two false-positives). Based on these results, precision = 4/6 (four blue out of six retrieved tickets), and recall = 4/5 (four blue out of five in the initial set).

For classification problems in data mining, we can look at a precision class by class or globally. For each class we can divide the number of correct classifications by the number of objects classified in that class to obtain precision. Globally, precision is the number of correct classifications divided by the total number of objects in the test set. Recall is better seen class by class; for a given class, one can divide the correct classifications by the number of objects that should have been classified in that class to obtain recall.

For multi-class and multi-label problems, methods such as precision and recall need to be combined in order to measure the performance of all classes. Therefore, a hybrid method, called $F1$ (Van Rijsbergan, 1979), which measures the average effect of both precision and recall together, has been used in IR and text categorization. Given precision (P) and recall (R) estimates for a given class in a problem

$$F1(P, R) = \frac{2 \times P \times R}{(P + R)}. \tag{7.3}$$

$F1$ is computed for each class independently and then the mean or the average of the results is computed on the test data set as a whole using one of two different methods called micro-averaging and macro-averaging (Yang *et al.*, 2002) in order to reflect the quality of the classifier. Macro-averaging represents the average of precision or recall for all classes and micro-averaging accumulates the decisions for all classes (sum of all true-positives, false-positives and false-negatives), then precision and recall are calculated using the global values.

For binary classification where there are two classes in the training data set, a common method called a confusion matrix, which takes into account the cost of wrong prediction, was proposed by Provost *et al.* (1997). A confusion matrix is similar to precision and recall methods where it contains information about actual and predicted classifications made by the classifier. The performance of the resultant classifier is commonly evaluated using the data in the matrix.

Table 8 represents a confusion matrix, which contains information about the actual and predicted classifications made by a classifier. In Table 8, W corresponds to so-called true-positive and represents the number of cases in which an object is positive. Z represents the number of incorrect predictions that an object is negative, X represents the number of correct predictions

Table 8 Confusion matrix

		Predicted	
		Negative	Positive
Actual	Negative	X	Y
	Positive	Z	W

that an object is negative and finally Y represents the number of incorrect predictions that an object is positive. Based on Table 8, the following equations represent accuracy, true-positive, true-negative, false-positive and false-negative.

The accuracy is the proportion of cases in the test data set that were correct.

$$\text{Accuracy} = (X + W)/(X + Y + Z + W). \tag{7.4}$$

The true-positive rate (TP) is the proportion of positive cases that were correctly classified:

$$TP = W/(Z + W). \tag{7.5}$$

The false-positive rate (FP) is the proportion of negatives cases that were incorrectly classified as positive:

$$FP = Y/(X + Y). \tag{7.6}$$

The true-negative rate (TN) is defined as the proportion of negatives cases that were classified correctly:

$$TN = X/(X + Y). \tag{7.7}$$

The false-negative rate (FN) is the proportion of positives cases that were incorrectly classified as negative:

$$FN = Z/(Z + W). \tag{7.8}$$

Overall, AC algorithms, (e.g. Liu *et al.*, 1998; Baralis & Torino, 2000; Li *et al.*, 2001; Yin & Han, 2003; Antonie & Zaïane, 2004) use an error-rate method to obtain the effectiveness of their classifiers. Using an error-rate method to validate the predictive strength of a classifier is not the optimum choice for multi-label classifiers, as only one class per rule contributes to the overall effectiveness of the classifier. However, more than one class with very close frequencies can occur with a rule, making the contribution of a single class and ignoring the rest, questionable. In the next section, recently proposed evaluation methods specifically designed for binary, multi-class and multi-label AC problems are shown.

7.2 Associative classification evaluation measures

The multi-label rules approach has been investigated mostly in specific multi-label classification problems such as gene classification in bioinformatics (Clare & King, 2001) and scene classification (Boutell *et al.*, 2003), and therefore there has been very little work conducted on developing evaluation measures for multi-label classifiers. There are no standard evaluation techniques applicable to the multi-label rules approach. Moreover, the right method is often problematic and depends heavily on the features of the conducted problem (Schapire & Singer, 2000; Boutell *et al.*, 2003). In this section, two evaluation measures called any-label and label-weight (Thabtah *et al.*, 2004) are reviewed, which are suitable for the majority of single and multi-label associative classifiers. Let D denote a test data set with m rows $d_1, d_2, \ldots, d_m$, and let C denote the set of classes. Row d has class $c(d)$ in the test data set. A (possibly multi-label) classifier is a multi-function h: $D \rightarrow 2^C$, where for $d \in D, h(d) = \langle h^1(d), h^2(d), \ldots, h^{k(d)}(d)\rangle$. The frequencies of $h^1(d), h^2(d), \ldots, h^{k(d)}(d)$ in the training data set are denoted $f^1(d), f^2(d), \ldots, f^{k(d)}(d)$, respectively.

7.2.1 Any-label

This lightly optimistic evaluation method measures how many times any of the predicted class labels assigned by a multi-label rule to a test data object matches the actual class in all cases of that object in the test data set. If any of the predicted class labels of a test data object d matches the true class y, then the classification is correct. The any-label is

$$|\{d \in D : \exists i \text{ with } h^i(d) = c(d)\}|/m. \tag{7.9}$$

7.2.2 Label-weight

This method enables each class for a multi-label rule to play a role in classifying a test object based on its ranking. An object may belong to several class labels, each associated with it by the number of occurrences in the training data set. Each class can be assigned a weight according to how many times that class has been associated with the object. The label-weight is

$$\sum_{d \in D} \sum_{\{i:h^i(d)=c(d)\}} \left(f^i(d) / \sum_{j=1}^{k(d)} f^j(d) \right). \tag{7.10}$$

For example, if an itemset (A, a) is associated with class labels c_1, c_2 and c_3, seven, five and three times, respectively, in the training data set, and that itemset becomes a rule $(A,a) \rightarrow c_1 \vee c_2 \vee c_3$, then each class associated with such a rule is assigned a weight, i.e. 7/15, 5/15 and 3/15, respectively, for class labels c_1, c_2 and c_3, based on their frequencies in the training data set. This technique assigns the predicted class weight to the test object if the predicted class matches the true test object class. For example, if class c_2 of the rule $(A,a) \rightarrow c_1 \vee c_2 \vee c_3$ matches a test object in the test data set that has c_2 as its actual class, then that test object is considered a correct classification, and the value of 5/15 will be assigned to it. For multiple label classification data sets where there are more than one class associated with a test data object, the label-weight method considers the summation of test object label-weights in the prediction step. For the above example and assuming that the test object is associated with class labels c_1 and c_2, the label-weight assigns $(7/15 + 5/15)$ to the test data object.

7.3 Discussion

The error-rate method considers only one class for each rule in computing the correct predictions and thus it can be criticized for favouring only one class. Alternatively, the label-weight assigns a value for each possible class in a rule according to its frequency in the training data set, and therefore the top-ranked class in a rule achieves the highest weight. For example, for a CAR, $r : (x \wedge y) \rightarrow c_1 \vee c_2 \vee c_3$ and for each test object that contains items x and y, the error-rate method considers only class c_1 as a correct classification in the prediction stage. All other applicable test data objects having either c_2 or c_3 are considered a misclassification. However, class labels c_2 and c_3 can contribute to the final decision because they pass support and confidence requirements.

The label-weight method gives a value for each test data object that contains itemset (x, y) in its body and has either class c_1, c_2 or c_3 based on the class label frequencies in the training data set. The summation of class label-weights for a rule is equal to one. This method does not favour any class, no matter what its ranking in a rule; instead, it reflects the true distribution frequency for each class when associated with a particular itemset.

8 Future directions

8.1 Incremental learning

Existing AC algorithms mine the training data set as a whole in order to produce the outcome. When data operations (adding, deleting and editing) occur on the training data set, current algorithms have to scan the complete training data set one more time in order to reflect the changes made. Furthermore, as data are collected in most application domains on a daily, weekly

or monthly basis, training data sets can grow rapidly. As a result of this, the cost of the repetitive scan each time a training data set is modified in order to update the set of rules is costly with regards to I/O and CPU times. Incremental AC algorithms, which can keep the last mining results and only consider data records that have been updated, are a more efficient approach, which can lead to a huge saving in computational time.

To explain the incremental mining problem more precisely within the AC context, assume a training data set T exists. The following operations may occur on T:

(1) T can be incremented by T^+ records;
(2) T^- records can be removed from T;
(3) T^+ records can be added to T and T^- records can be removed from T.

The result of any of the operations described above on T is an updated training data set T'. It should be noted that the third operation (update) is more difficult than insert/delete operations as it combines both. The question now is how the outcome (rules) of the original data set T can be updated to reflect changes made to T without having to perform extensive computations. This problem can be divided into sub-problems according to the possible *ruleitems* contained in T after performing data operations such as insert, update or edit. For example, *ruleitems* in T can be divided into the following groups after inserting new records (T^+):

(a) *ruleitems* that are frequent in T and T^+;
(b) *ruleitems* that are frequent in T and not frequent in T^+;
(c) *ruleitems* that are frequent in T^+ and not frequent in T;
(d) *ruleitems* that are not frequent in either T^+ or T.

The *ruleitems* in groups (a) and (b) can be identified in a straightforward manner. For example, if *ruleitem* Y is frequent in T, then its support count in the updated training data set (T'), $Y'_{\text{count}} = Y_{\text{count}} + Y^+_{\text{count}}$, where Y_{count} is known and Y^+_{count} can be obtained after scanning T^+. The challenging problem is to find frequent *ruleitems* that are not frequent in T but frequent in T^+ as these *ruleitems* are not determined after scanning T or T^{+-}. Once all above frequent *ruleitems* are determined, the generation of the incremental rules is easy because no database scan is involved.

There has been some research work on incremental association rule discovery algorithms (i.e. Tsai *et al.*, 1999; Zhou & Ezeife, 2001; Valtchev *et al.*, 2002). These can be considered as a starting point for incremental AC mining. For example, an incremental association rule discovery algorithm called maintenance association rule with Apriori property (MAAP; Zhou & Ezeife, 2001) has been proposed. This algorithm generates incremental rules from an updated database using an Apriori candidate generation step (Agrawal & Srikant, 1994). MAAP computes high-level frequent n-itemsets and then starts producing all lower-level $n-1, n-2, \ldots, 1$ frequent itemsets. This approach decreases the processing overheads for generating some of the low-level frequent itemsets that have no chance of being frequent in the updated database. Thus, the key feature of MAAP is the downward-closure property presented in Apriori.

Another incremental association rule discovery algorithm, which extends the fast-update (FUP) algorithm (Cheung *et al.*, 1996) to handle editing and deleting operations on transactional databases, was proposed by Tsai *et al.* (1999). The FUP incremental algorithm deals only with the dynamic insertion of records into the database and uses the Apriori approach (Agrawal & Srikant, 1994) for discovering frequent itemsets. The algorithm proposed by Tsai *et al.* (1999) improves upon the FUP algorithm with reference to processing time by storing not only frequent itemsets discovered from the original database but also itemsets that are not frequent but may become frequent after updating the original database. These potential frequent itemsets may reduce the search time for candidate itemsets in the updated database.

Incremental AC mining is a challenging problem in data mining, which has not been carefully studied. Furthermore, the key success to solving this problem is to determine the frequent

ruleitems that overlap between the original training data set and the records, which have been updated regardless of whether the operation is insert, delete or edit.

8.2 Missing data

Roughly speaking, a classifier in data mining is constructed from labelled data records, and later is used to forecast classes of previously unseen data as accurately as possible. Training and test data sets may contain noise, including missing or incorrect values inside records. It is necessary to think carefully about the importance of missing or incorrect values in training or test data sets. As a result, only human experts in the application domains used to generate the data sets can make an implicit assumption about the significance of missing or invalid values.

In data mining and machine learning communities, several classification algorithms have been proposed, where most of these produce classifiers with an acceptable error rate. However, most of these algorithms assume that all records in the test data collection are complete and no missing data are present. When test data sets suffer from missing attribute values or incomplete records, classification algorithms often produce poor classifiers with reference to prediction accuracy. This is because these algorithms tend to tailor the training data set too much.

In real-world applications, it is common that a training data set or test data set contains attributes with missing values. For instance, the 'labor' and 'hepatid' data sets published in the UCI data repository contain many missing records. Thus, it is imperative to build classifiers that are able to predict accurately the classes for test data sets with missing attribute values. These classifiers are normally called robust classifiers. Unlike traditional classifiers, which assume that the test data set is complete, robust classifiers deal with existing and non-existing values in the test data set.

There have been some solutions to avoid noise in the training data sets. The naïve Bayes algorithm, for example, ignores missing values during the computation of probabilities, and thus missing values have no effect on the prediction because they have been omitted. However, omitting missing values may not be the ideal solution as these unknown values may provide a good deal of information. Other classification techniques such as CBA assume that the absence of missing values may be of some importance, and therefore they treat them as other existing known values in the training data set. However, if this is not the case, then the missing values should be treated in a special way rather than just considering them as other possible values that the attribute might take (Witten & Frank, 2000). Decision tree algorithms such as C4.5 and C5 deal with missing values using probabilities, which are calculated from the frequencies of the different values for an attribute at a particular node in the decision tree (Quinlan, 1993 provides further details).

The problem of dealing with unknown values inside test data sets has not yet been explored well in traditional classification or AC approaches. One possible simple solution for this problem is to select the common value of the attribute that contains missing values from the training data set. The common value could be selected from the attribute objects that occur with the same class to which the missing value belongs. Then, each missing value for that attribute and its corresponding class in the test data set is substituted with the common value. The common value represents the value that has the highest frequency with the attribute in the training data set. We could also use common values from the test data set in the same way as described above to substitute attributes with missing values. Another possible solution for missing values in the test data set is to utilize weights or probabilities similar to the C4.5 algorithm.

8.3 The exponential growth of rules

Association rule discovery considers all associations between items in a transactional database, and thus the expected numbers of association rules are of the order of thousands or even tens of thousands. The AC approach employs association rule discovery to discover CARs, and therefore the numbers of rules produced are normally large, especially when a very low support threshold is used. The key element, which controls the number of rules produced in both

association rule discovery and AC, is the support threshold. If the support is set to a large value, normally the number of extracted rules is very limited, and many rules with high confidence will be missed. This may lead to discarding important knowledge that could be useful in the classification step. To overcome this problem, it is necessary to set the support threshold to a very small value. However, this usually involves the generation of massive numbers of rules, where many are useless as they hold low support and confidence values. This large number of rules may cause severe problems, such as overfitting.

There have been attempts to control the number of rules extracted by AC algorithms. One promising direction is to use multiple support thresholds during the mining process. Using this approach, high-frequency classes are given larger support thresholds than low-frequency classes. The user sets one support threshold, which is distributed to the available classes in the training data set according to their frequencies. Using this approach, low-frequency classes are given smaller support than high-frequency classes, and therefore they will be balanced among classes with reference to the number of generated rules.

The multiple support AC approach ensures the production of rules for low coverage classes. Yet, the numbers of produced rules are still large, which makes it difficult for human experts to profit from the outcome. Pruning heuristics presented in machine learning and statistics are another potential tool that can be utilized to cut down further the number of extracted rules. Some of the current AC algorithms utilize various early and late pruning methods to decrease the number of produced rules (see Section 4). Mainly, they use support and confidence as early pruning to discard infrequent *ruleitems* from taking any part in the classifier. Furthermore, pruning heuristics such as database coverage and pessimistic error estimation are used to discard rules that do not cover a sufficient number of training objects or rules that have unacceptable error rate on an independent test data set.

One possible future direction to produce moderate-sized classifiers with an acceptable error rate is to ensure that every data object in the training data set is used only once during the rule generation phase. This approach is similar to rule induction and covering approaches in classification. In fact, rule induction and covering algorithms frequently produce small-sized classifiers when compared with popular classification approaches such as decision trees and AC (Freitas, 2000; Hu & Li, 2005). This is because they employ heuristic strategies to discover the rules and utilize extensive pruning such as global optimization (Cohen, 1995). Consequently, if an AC algorithm marks the training data objects covered by each discovered rule to ensure that these objects are no longer available for other potential rules, there is a high possibility that a smaller-sized classifier will result.

It is the firm belief of the author that at least two directions can be used as starting points to produce moderate-sized classifiers with good predictive accuracy. The first direction is the utilization of multiple support thresholds with post-pruning methods such as database coverage and/or pessimistic error estimation. The second direction is to produce classifiers based on association rule discovery techniques, where each training object is used only once during the training phase.

9 Conclusions

In this paper, we have surveyed different AC approaches, as shown in Table 9, and discussed the approaches used to find rules, rank rules, prune rules, predict test objects and evaluate the effectiveness of their classifiers. Research work on AC to date has been devoted to general classification problems where the aim is to build a classifier that contains single label rules. Many AC algorithms have been successfully used to build accurate classifiers, such as CBA, MCAR, CMAR and CPAR, where only the most obvious class correlated to a rule is created and other classes are simply discarded. Sometimes the ignored class labels have frequencies in the training data set above certain user thresholds, making their presence in the classifier important. In addition, these ignored class labels may represent useful knowledge discarded by current AC algorithms, which domain experts may benefit from.

Table 9 Summary of AC algorithms

Name	Data layout	Rule discovery	Ranking	Pruning	Prediction method	Reference
CBA	Horizontal	Apriori candidate generation	Support, confidence, rules generated first	Pessimistic error, database coverage	Maximum likelihood	Liu *et al.* (1998)
CMAR	Horizontal	FP-growth approach	Support, confidence, rules cardinality	Chi-square, database coverage, redundant rule	CMAR multiple label	Li *et al.* (2001)
CPAR	Horizontal	Foil greedy	Support, confidence, rules cardinality	Laplace expected error estimate	CPAR multiple label	Yin & Han (2003)
ADT	Horizontal	Upward cluster property	Support, confidence, rules cardinality, items lexicographical	Pessimistic error, redundant rule	Maximum likelihood	Wang *et al.* (2000)
Negative rules	Horizontal	Apriori candidate generation	Support, confidence	Correlation coefficient	Dominant factor multiple label	Antonie & Zaïane (2004)
ARC-AC	Horizontal	Apriori candidate generation	Support, confidence	Redundant rule	Dominant factor multiple label	Zaïane & Antonie (2002)
ARC-BC	Horizontal	Apriori candidate generation	Support, confidence, cardinality	Redundant rule	Dominant factor multiple label	Antonie *et al.* (2003)
L^3	Horizontal	FP-growth approach	Support, confidence, rules cardinality, items lexicographical	Lazy pruning	Maximum likelihood	Baralis & Torino (2000)
L^3G	Horizontal	FP-growth approach	Support, confidence, rules cardinality, items lexicographical	Lazy pruning	Maximum likelihood	Baralis *et al.* (2004)
CAEP	Horizontal	Emerging Pattern	growth rare, support	Redundant rules	Aggregate score	Dong *et al.* (1999)
MCAR	Vertical	Tid-list intersections	Support, confidence, cardinality	Database coverage	Exact maximum likelihood	Thabtah *et al.* (2005)
MMAC	Vertical	Tid-list intersections and recursive learning	Support, confidence, cardinality, class distribution frequency	Database coverage	Exact maximum likelihood	Thabtah *et al.* (2004)
CBA (2)	Horizontal	Apriori candidate generation	Support, confidence, rules generated first	Pessimistic error, database coverage	Maximum likelihood	Liu *et al.* (2003)
CAAR	Horizontal	Multipass Apriori	Support, confidence, rules generated first	Database coverage similar method	Not known	Xu *et al.* (2004)
Partial classifier	Horizontal	Apriori candidate generation	N/A	N/A	N/A	Ali *et al.* (1997)

For classification data sets that are correlated, there is a need for additional constraints beside support, confidence and cardinality in the rule ranking process, in order to break ties between similar rules and to minimize random selection. Also, pruning can be used to cut down the number of rules produced and to avoid overfitting. Furthermore, most existing AC techniques use the horizontal layout presented in Apriori to represent the training data set. This approach suffers from drawbacks, including multiple database scans and the use of complex data structures in order to hold all potential rules during each level, requiring large CPU times and memory size. However, the vertical data format may require only a single database scan, although the number of tid-list intersections may become large, consuming considerable CPU time. Efficient rule discovery methods that avoid going through the database multiple times and do not perform a large number of computations can avoid some of these problems.

Single and multiple label prediction procedures have been surveyed and it has been found that algorithms which use more than one rule to predict class labels for a test data set do not consider the independence of the rules, where they allow a training data object to be covered by several rules. However, algorithms that utilize one rule in the prediction step may produce good classifiers, but assume that the highest precedence rule is able to predict the majority of test data objects satisfying its body. Finally, there is a need for new associative algorithms that deal with low-quality training and test data sets, i.e. data sets which contain missing values and noise, as well as data sets that are updated frequently.

References

Agrawal, R. & Srikant, R. 1994 Fast algorithms for mining association rule. In *Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufmann, Santiago, Chile*, pp. 487–499.

Agrawal, R., Amielinski, T. & Swami, A. 1993 Mining association rule between sets of items in large databases. In Buneman, P. & Jajodia, S. (eds.), *Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, DC*, pp. 207–216.

Ali, K., Manganaris, S. & Srikant, R. 1997 Partial classification using association rules. In Heckerman, D., Mannila, H., Pregibon, D. & Uthurusamy, R. (eds.), *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, Newport Beach, CA*, pp. 115–118.

Antonie, M. & Zaïane, O. 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, France: ACM Press, pp. 64–69.

Antonie, M., Zaïane, O. & Coman, A. 2003 Associative classifiers for medical images. *Mining Multimedia and Complex Data (Lecture Notes in Artificial Intelligence, 2797)*. Berlin: Springer, pp. 68–83.

Baralis, E. & Torino, P. 2002 A lazy approach to pruning classification rules. *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02), Maebashi City, Japan*, p. 35.

Baralis, E., Chiusano, S. & Graza, P. 2004 On support thresholds in associative classification. In *Proceedings of the 2004 ACM Symposium on Applied Computing.* Nicosia, Cyprus: ACM Press, pp. 553–558.

Boutell, M., Shen, X., Luo, J. & Brown, C. 2003 Multi-label semantic scene classification. Technical Report 813, Department of Computer Science, University of Rochester, NY and Electronic Imaging Products R & D, Eastern Kodak Company.

Cheung, D. W., Ng, V. T. & Tam, B. W. 1996 Maintenance of discovered knowledge: A case in multi-level association rules. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, Portland, OR: AAAI Press, pp. 307–310.

Clare, A. & King, R. 2001 Knowledge discovery in multi-label phenotype data. In De Raedt, L. & Siebes, A. (eds.), *Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'01) (Lecture Notes in Artificial Intelligence, 2168)*. Berlin: Springer, pp. 42–53.

Clark, P. & Boswell, R. 1991 Rule induction with CN2: Some recent improvements. In *Proceedings of the 5th European Working Session on Learning.* Berlin, Germany: Springer Verlag, pp. 151–163.

Cohen, W. 1995 Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning, Morgan Kaufmann, CA*, pp. 115–123.

Dong, G., Zhang, X., Wong, L. & Li, J. 1999 CAEP: Classification by aggregating emerging patterns. In *Proceedings of the 2nd Imitational Conference on Discovery Science.* Tokyo, Japan: Springer Verlag, pp. 30–42.

Duda, R. & Hart, P. 1973 *Pattern Classification and Scene Analysis*. New York: Wiley.

Fayyad, U., Piatetsky-Shapiro, G., Smith, G. & Uthurusamy, R. 1998 *Advances in Knowledge Discovery and Data Mining*. Menlo Park, CA: AAAI Press.

Freitas, A. 2000 Understanding the crucial difference between classification and association rule discovery. *ACM SIGKDD Explorations Newsletter* **2**, 65–69.

Furnkranz, J. & Widmer, G. 1994 Incremental reduced error pruning. In *Proceedings of the 11th International Machine Learning Conference, New Brunswick, NJ*, pp. 70–75.

Gehrke, J., Ramakrishnan, R. & Ganti, V. 1998 *RainForest*: A Framework for fast decision tree construction of large datasets. In *Proceedings of the International Conference on very Large Data Bases, New York, NY*, pp. 416–427.

Han, J., Pei, J. & Yin, Y. 2000 Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data.* Dallas, TX: ACM Press, pp. 1–12.

Hu, H. & Li, J. 2005 Using association rules to make rule-based classifiers robust. In *Proceedings of the 16th Australasian Database Conference, Newcastle, Australia*, pp. 47–54.

Li, W. 2001 Classification based on multiple association rules. MSc thesis, Simon Fraser University, BC, Canada, April 2001.

Li, W., Han, J. & Pei, J. 2001 CMAR: Accurate and efficient classification based on multiple-class association rule. In *Proceedings of the International Conference on Data Mining (ICDM'01), San Jose, CA*, pp. 369–376.

Lim, T., Loh, W. & Shih, Y. 2000 A comparison of prediction accuracy, complexity and training time of thirty-three old and new classification algorithms. *Machine Learning* **40**, 203–228.

Liu, B., Hsu, W. & Ma, Y. 1998 Integrating classification and association rule mining. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining.* New York, NY: AAAI Press, pp. 80–86.

Liu, B., Hsu, W. & Ma, Y. 1999 Mining association rules with multiple minimum supports. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* San Diego, CA: ACM Press, pp. 337–341.

Liu, B., Ma, Y. & Wong, C.-K. 2000 Improving an association rule based classifier. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, Lyon, France*, pp. 504–509.

Liu, B., Ma, Y., & Wong, C.-K. 2001 Classification using association rules: Weakness and enhancements. In Vipin Kumar, *et al.* (eds), *Data Mining for Scientific Applications,* 2001.

Meretakis, D. & Wüthrich, B. 1999 Extending naïve Bayes classifiers using long itemsets. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* San Diego, CA: ACM Press, pp. 165–174.

Merz, C. & Murphy, P. 1996 UCI repository of machine learning databases. Irvine, CA: University of California, Department of Information and Computer Science.

Provost, F., Fawcett, T. & Kohavi, R. 1997 The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the 15th International Conference on Machine Learning, Madison, WI*, pp. 445–453.

Quinlan, J. 1987 Simplifying decision trees. *International Journal of Man–Machine Studies* **27**, 221–248.

Quinlan, J. 1998 Data mining tools See5 and C5.0. Technical Report, RuleQuest Research.

Quinlan, J. 1993 *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Quinlan, J. & Cameron-Jones, R. 1993 FOIL: A midterm report. In *Proceedings of the European Conference on Machine Learning.* Vienna, Austria: Springer Verlag, pp. 3–20.

Savasere, A., Omiecinski, E. & Navathe, S. 1995 An efficient algorithm for mining association rules in large databases. In *Proceedings of the 21st conference on Very Large Databases (VLDB'95), Zurich, Switzerland*, pp. 432–444.

Schapire, R. & Singer, Y. 2000 BoosTexter: A boosting-based system for text categorization. *Machine Learning* **39**(2/3), 135–168.

Snedecor, W. & Cochran, W. 1989 *Statistical Methods*, 8th edn. Iowa City, IA: Iowa State University Press.

Thabtah F. 2006 Pruning techniques in associative classification: Survey and comparison. *Journal of Digital Information Management* **4**, 202–205.

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 (ICDM'04), Brighton, UK*, pp. 217–224.

Thabtah, F., Cowling, P. & Peng, Y. 2005 MCAR: Multi-class classification based on association rule approach. In *Proceeding of the 3rd IEEE International Conference on Computer Systems and Applications, Cairo, Egypt*, pp. 1–7.

Topor, R. & Shen, H. 2001 Construct robust rule sets for classification. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Edmonton, Alberta, Canada: ACM Press, pp. 564–569.

Tsai, P., Lee, C. & Chen, A. 1999 An efficient approach for incremental association rule mining. In *Proceedings of the 3rd Pacific–Asia Conference on Methodologies for Knowledge Discovery and Data Mining.* London, UK: Springer Verlag, pp. 74–83.

Valtchev, P., Missaoui, R., Godin, R. & Meridji, M. 2002 A framework for incremental generation of frequent closed itemsets using galois (Concept) lattice theory. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI), Special Issue on Concept Lattice based Theory, Methods and Tools for Knowledge Discovery in Databases*, **14**, 115–142.

Van Rijsbergan, C. 1979 *Information Retrieval*, 2nd edn. London: Buttersmiths.

Wang, K., Zhou, S. & He, Y. 2000 Growing decision tree on support-less association rules. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Boston, MA: ACM Press, pp. 265–269.

Wang, K., He, Y. & Cheung, D. 2001 Mining confidence rules without support requirements. In *Proceedings of the 10th International Conference on Information and Knowledge Management.* Atlanta, GA: ACM Press, pp. 89–96.

Weka, 2000 Data mining software in Java. `http://www.cs.waikato.ac.nz/ml/weka`.

Witten, I. & Frank, E. 2000 *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco, CA: Morgan Kaufmann.

Xu, X., Han, G. & Min, H. 2004 A novel algorithm for associative classification of images blocks. In *Proceedings of the 4th IEEE International Conference on Computer and Information Technology, Lian, Shiguo, China*, pp. 46–51.

Yang, Y., Slattery, S. & Ghani, R. 2002 A study of approaches to hypertext categorization. *Journal of Intelligent Information Systems* **18**, 149–241.

Yin, X. & Han, J. 2003 CPAR: Classification based on predictive association rule. In *Proceedings of the SIAM International Conference on Data Mining.* San Francisco, CA: SIAM Press, pp. 369–376.

Zaïane, O. & Antonie, A. 2002 Classifying text documents by associating terms with text categories. In *Proceedings of the 13th Australasian Database Conference (ADC'02), Melbourne, Australia*, pp. 215–222.

Zaki, M. & Gouda, K. 2003 Fast vertical mining using diffsets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Washington, DC: ACM Press, pp. 326–335.

Zaki, M., Parthasarathy, S., Ogihara, M. & Li, W. 1997 New algorithms for fast discovery of association rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining.* Menlo Park, CA: AAAI Press, pp. 283–286.

Zhou, Z. & Ezeife, C. 2001 A low-scan incremental association rule maintenance method based on the Apriori property. In *Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence.* London, UK: Springer-Verlag, pp. 26–35.