ON MERGING CLASSIFICATION RULES

B. BOUTSINAS
University of Patras, Department of Business Administration,
University of Patras Artificial Intelligence Research Center [UPAIRC],
GR-26500, RIO, GREECE,
e-mail: voutsinas@upatras.gr, tel. +302610997845, fax +302610996236

S. ATHANASIADIS
University of Patras Artificial Intelligence Research Center [UPAIRC],
GR-26500, RIO, GREECE,
e-mail: aa@master.math.upatras.gr

One of the main challenges of nowadays data mining systems is their ability to manage a huge volume of data generated possibly by different sources. On the other hand, inductive learning algorithms have been extensively researched in machine learning using small amounts of judiciously chosen laboratory examples. There is an increasing concern in classifiers handling data that are substantially larger than available main memory on a single processor. One approach to the problem is to combine the results of different classifiers supplied with different subsets of the data, in parallel. In this paper, we present an efficient algorithm for combining partial classification rules. Moreover, the proposed algorithm can be used to match classification rules in a distributed environment, where different subsets of data may have different domains. The latter is achieved by using given concept hierarchies for the identification of matching classification rules. We also present empirical tests that demonstrate that the proposed algorithm has a significant speedup with respect to the analog non-distributed classification algorithm, at a cost of a lower classification accuracy.

Keywords: Data Mining; Classification; Meta-Learning.

1. Introduction
The problem of analyzing, summarizing and extracting 'knowledge' from large sets of stored data is a common concern to databases. The field of data mining has emerged to face the above situations. Data mining consists of particular algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of models over the data. Data mining algorithms may be of various types depending on the specific data mining problem they refer to. Classification is one of the most common problems. Classification becomes a cutting-edge information technology tool in the competitive business world of the "information age". It is applied in numerous business problems, as bankruptcy prediction, credit card portfolio management, etc.
Classification rules can be extracted using supervised learning methods and can be used to classify data into predefined classes, described by a set of attributes. A lot of algorithms have been proposed in the literature for extracting classification rules from large relational databases (e.g. 5, 9, 13, 14, 23, 26, 32).

One of the main challenges of nowadays data mining systems is their ability to manage a huge volume of data. Thus, there is an increasing interest in classifiers handling data that are substantially larger than available main memory on a single processor. This is usually called the scaling-up problem. Moreover, there is an increasing interest in classifiers handling data formed by integrating distributed heterogenous databases, such as databases distributed over various locations and/or vertically fragmented. This is usually called the incompatible schema problem 24.

In this paper, we present an efficient algorithm for combining partial classification rules in order to attack both the scaling-up and the incompatible schema problems. It is based on given concept hierarchies that they are used for the identification of matching classification rules, even if they have extracted from subsets of data with different domains. We also present extensive empirical tests that demonstrate that the proposed algorithm has a significant speed-up with respect to classifying all the data using a single classifier, at a cost of a lower classification accuracy.

In the rest of the paper we first describe related work (Section 2) and some preliminaries (Section 3), such as the concept hierarchy, and then we present the proposed algorithm (Section 4). Finally, we present experimental results (Section 5) that demonstrate the effectiveness and efficiency of the proposed algorithm.

2. Related work

There are several approaches for the scaling-up problem, in the literature. An obvious approach is to parallelize the classification algorithms. This approach requires the transformation of the classification algorithm to an optimized parallel one for a specific architecture (e.g. 42). In 25 a windowing technique is proposed, where the classifier is supplied with a small subset of data, having a fixed size, the window. Iteratively, the window is updated with unclassified data of the remaining set, until all the data are classified correctly. In 10 a technique is proposed, called meta-learning, that uses different classifiers supplied with different subsets of the data, in parallel. Then, the partial results are combined. Obviously, combination of partial results is the most critical step, in order for the classification accuracy to be the same as if one classifier were used, supplied with all the data.

To our knowledge, all the approaches for the scaling-up problem which are based on the meta-learning paradigm need access to the initial data set or to a subset of it (e.g. the validation set in 7, 11, 12). The proposed algorithm is based only on the partial classification rules extracted from subsets in order to synthesize the final set of classification rules.

The above approaches cope with the problem of handling a huge volume of data, artificially partitioning data into homogenous subsets. In a real life distributed en-
environment, classifiers must also face the problem of a physically partitioned volume of data, into heterogeneous subsets (the incompatible schema problem). The incompatible schema problem has already been attacked by extracting classification rules either directly from heterogeneous databases \(^{(15,31)}\) or from their integration \(^{(35)}\). There also approaches that address the incompatible schema problem in the field of database query languages (e.g. \(^{22,30}\)).

3. Preliminaries

The objective of the data analysis using classification algorithms is to induce a classifier from the given training data set. Formally, let \(o_1, \ldots, o_n\) be the input objects of a training data set. Each of them is represented by a \(d\)-tuple \((a_{i1}, \ldots, a_{im})\), where \(1 \leq i \leq d, 1 \leq j \leq m\), denotes the \(j^{th}\) value of the \(i^{th}\) attribute, whose domain \(\text{domain}(a_i) = \{a_{i1}, \ldots, a_{im}\}\) is the set of reals \(R\) for numerical data or a set of labels for categorical ones.

We assume that the models derived from the implementation of the classification algorithms and their application over a data set have the form of classification rules. Of course, in the literature there are transformations of other representation schemes to rules, as for example of decision trees \(^{27}\) and of neural networks \(^{1}\). Basically these classification rules describe the contents of the data set by partitioning them into a set of mutually exclusive classes. A classification rule is an “IF-THEN” rule where the premise, the “IF” part, is a conjunction of attribute-value pairs and the consequent, the “THEN” part, is a class attribute-value pair. An attribute-value pair \(a_i = a_{ik}\) represents the assignment of the value \(a_{ik}\) to the attribute \(a_i\), where \(a_{ik} \in \text{domain}(a_i)\). Formally, a rule \(r\) is defined as:

\[
r \equiv IF \ a_i = a_{ik} \land \ldots \land a_j = a_{jl} THEN \ a_c = a_{cm},
\]

where the attribute \(a_c\) is the class attribute. Every classification rule is associated with a measure of the correctness of the rule, with respect to the training set. We use the term ”coverage” for such measure with respect to a set \(t\). ”Coverage” is related to the probability that an object assigned by the rule to a class actually belongs to this class or, in other words, the percentage of true positives with respect to both true and false positives. Usually the ”coverage” of a rule is extracted from classification algorithms, along with other such measures (e.g. entropy of a rule, in C4.5). Such measures can be used to express the interestingness of a rule. Formally,

\[
\text{coverage}(r, t) = \frac{|s(a_{c_i}) \cap s(a_{c_j})|}{|s(a_{c_j})|},
\]

where \(s(a_{c_i})\) denotes the set of objects in the training set \(t\) belonging to class \(a_{c_i}\) and \(s(a_{c_j})\) denotes the set of objects in the training set \(t\) classified by the rule \(r\) in class \(a_{c_j}\). Also, \(|s|\) denotes the size of set \(s\), i.e. the number of objects in \(s\). In the following, we will also use the error rate defined as:

\[
\text{error}(r, t) = 1 - \text{coverage}(r, t)
\]

The proposed algorithm attempts to combine various sets of classification rules induced from different training sets of the same or different database schemes and
by different classifiers, under the condition that the target class of the classifiers is the same. Thus, the proposed algorithm is based on an artificial or physical partitioning of the initial data set into subsets, processing each subset in parallel and synthesizing the final set of classification rules from the partial results. Such divide-and-conquer process has already been used effectively for classification (e.g. \cite{7,10}), clustering \cite{6}, as well as for association \cite{34}.

Time and space complexity of classification algorithms depend on the size \( n \) of the training data set. Most of the proposed symbolic classification algorithms, and certainly all the popular ones, have significantly worse time complexity than \( O(n) \). For instance, the ID3 algorithm has an \( O(nd^2) \) worst time complexity \cite{40}. Note also that symbolic classification techniques, even with such time complexity, outperform connectionist ones. For instance ID3 algorithm outperforms back propagation neural nets by a factor 500 to 100000 \cite{28}. Thus, the partitioning of the initial data sets reduces significantly the complexity since the subsets of the data set are processed in parallel and each subset has much less objects.

The proposed algorithm tries to induce a final set of classification rules that are of the same classification accuracy as if one classifier were used over the whole data set. This final set of rules could include more general rules in the sense that it covers all possibly heterogenous database schemes which were used for the generation of the input subsets.

Synthesizing the final set of classification rules is based on a given concept hierarchy, that is used for the identification of matching classification rules. We assume that there exists a concept hierarchy tree for each attribute used to describe the data set. A concept hierarchy over an attribute \( a_i \) is a directed acyclic tree \( T_{a_i} = (N, E) \), where \( N \) is the set of nodes and \( E \) is the set of edges such that every \( e_i \in E \) is an ordered pair \((n_k, n_l)\), and \( n_k, n_l \in N \). Every node \( n_i \) represents an attribute value \( a_{ij} \in \text{domain}(a_i) \), thus \( N \equiv \text{domain}(a_i) \).

4. The proposed methodology

In the proposed methodology we presume that two or more classifiers, possibly based on different classification algorithms, are applied on different subsets of the initial data set. The rules extracted from the subsets are processed further, with the aim of generalizing/specializing some of them, or merging the identical ones, or detecting and regulating any kind of conflicts, or deleting the redundant ones.

First, we investigate the scaling-up problem where each artificially made subset is based on the same database schema. To attack the scaling-up problem we introduce three operators to be applied to the union of the sets of classification rules extracted from each subset.

The first operator, the “merge” operator, merges identical rules, which are extracted from different subsets, to one rule. The coverage of the new rule is based on the coverage of the merged ones. Formally, if there are \( m \) identical rules:

\[
\begin{align*}
    r_1 &= \ldots = r_m \\
    \text{IF} & a_i = a_{i_k} \land \ldots \land a_j = a_{j_l} \text{ THEN } a_c = a_{c_x}
\end{align*}
\]
then they are merged to a new rule \( r \), which is also identical to others, such as:

\[
\text{coverage}(r, t) = \frac{|s(a_{c_s}^1) \cap s(a_{c_s}) + \ldots + |s(a_{c_m}^m) \cap s(a_{c_m})|}{|s(a_{c_s}^1)| + \ldots + |s(a_{c_m}^m)|}.
\]

**PROOF.** Since \( r_1 = \ldots = r_m \) are identical rules, the application of \( r \) to all the training set is equivalent to the application of \( r \) to every of the \( m \) disjoint subsets of the training set. Thus, for every subset, the rule \( r \) will assign \(|s(a_{c_s}^1) \cap s(a_{c_s})|\) objects to class \( a_{c_s} \). Thus, in all the training set the rule \( r \) will assign \(|s(a_{c_s}^1) \cap s(a_{c_s})| + \ldots + |s(a_{c_m}^m) \cap s(a_{c_m})|\) objects to class \( a_{c_s} \). In the whole, the rule \( r \) will assign \(|s(a_{c_s}^1)| + \ldots + |s(a_{c_m}^m)|\) objects to some class.

The second operator, the “conflict” operator, removes the conflicting rules. A rule is in conflict with another if they share the same premise but have different consequents. Formally, \( r \neq r_s \)

\[
r_1 \equiv \text{IF } a_i = a_{ik} \land \ldots \land a_j = a_{jl} \text{ THEN } a_c = a_{c_s}
\]

is in conflict with the rule

\[
r_2 \equiv \text{IF } a_i = a_{ik} \land \ldots \land a_j = a_{jl} \text{ THEN } a_c = a_{c_s}.
\]

The “conflict” operator is implemented using the resolve_conflict() procedure presented in the next subsection. After the application of the “conflict” operator either one or both of the conflicting rules are deleted. In the case where one conflicting rule is deleted, suppose that \( r_2 \) is deleted, then coverage of \( r_1 \) is updated as:

\[
\text{coverage}(r_1, t) = \frac{|s(a_{c_s}^1) \cap s(a_{c_s}) + |s(a_{c_s}^1)| \cap s(a_{c_s})|}{|s(a_{c_s}^1)| + |s(a_{c_s}^1)|}.
\]

This can be generalized in the case where there are more than two conflicting rules.

**PROOF.** If \( r_1 \) was applied to subset from which \( r_2 \) is extracted then it will assign to class \( a_{c_s} \) the objects that \( r_2 \) assigned to class \( a_{c_s} \). However, only \(|s(a_{c_s}^1) \cap s(a_{c_s})|\) of them belong to class \( a_{c_s} \). In the whole, the rule \( r_1 \) will assign \(|s(a_{c_s}^1)| + |s(a_{c_s}^1)|\) objects to some class. Thus, for every subset, the rule \( r \) will assign \(|s(a_{c_s}^1) \cap s(a_{c_s})|\) objects to class \( a_{c_s} \). Thus, in all the dataset the rule \( r \) will assign \(|s(a_{c_s}^1) \cap s(a_{c_s})| + \ldots + |s(a_{c_m}^m) \cap s(a_{c_m})|\) objects to class \( a_{c_s} \). In the whole, the rule \( r \) will assign \(|s(a_{c_s}^1)| + \ldots + |s(a_{c_m}^m)|\) objects to a class.

The third operator, the “redundancy” operator, removes the redundant rules. A rule is removed if there exists a more general rule with the same class attribute value. Formally,

\[
r_1 \equiv \text{IF } a_i = a_{ik} \land \ldots \land a_p = a_{p}, \land \ldots \land a_j = a_{jl} \text{ THEN } a_c = a_{c_s}
\]

is redundant if there exists

\[
r_2 \equiv \text{IF } a_i = a_{ik} \land \ldots \land a_p = a_{p}, \text{ THEN } a_c = a_{c_s}.
\]

The redundant rules are deleted. Similarly to the “merge” operator, the coverage of the more general rule is updated based on the coverage of the deleted redundant rules. Formally, if \( r_1 \ldots r_m \) are redundant due to existence of the more general rule \( r \), then the coverage of \( r \) is updated as:

\[
\text{coverage}(r, t) = \frac{|s(a_{c_s}^1) \cap s(a_{c_s})| + |s(a_{c_s}^1)| \cap s(a_{c_s})| \ldots + |s(a_{c_m}^m) \cap s(a_{c_m})|}{|s(a_{c_s}^1)| + |s(a_{c_s}^1)| \ldots + |s(a_{c_m}^m)|}.
\]

The proof is similar to that of identical rules.

It is obvious that the “merge” operator is a special case of the “redundancy” one. However, we define them separately due to implementation reasons, as it will be shown in the next subsection.
Notice that the deletion of a redundant rule $r_2$, due to existence of a more general rule $r_1$, could be considered as a pruning/generalization of $r_2$. However, the motivation is entirely different. The proposed algorithm tries to form the set of rules which would be extracted if classification was applied to all the training set. In that case, $r_2$ would never be extracted. Also, the proposed algorithm does not search for proper conditions to drop from a rule. Instead, it searches for redundant rules extracted from different subsets and deletes the more specific ones. Note also that rule pruning/generalization could be used in the proposed algorithm as a last step, after the application of the “merge”, “conflict” and “redundancy” operators.

The above operators can be applied to rules extracted from subsets based on the same database schema. However, usually, there is a physical partition to subsets based on different database schemas. Note, for instance, the common case of geographically distributed organizations that hold data at different sites under different database schemas. Using centralized classification to discover useful patterns in such organizations’ data is not always feasible, because merging data sets from different sites into a centralized site incurs huge network communication costs. In this case (the incompatible schema problem), when classifying a new object, the “unify” operator must first be applied to it before the above operators. The “unify” operator is based on the given concept hierarchies $T_{a_1}, \cdots, T_{a_d}$ concerning the different attributes.

For each attribute $a_i$, the attribute values that appear in the extracted rules and the new object may be represented by nodes at different levels of the concept hierarchy $T_{a_i}$ concerning this attribute. The “unify” operator changes the attribute values of the new object in order to be represented by the same nodes of the concept hierarchy with respect to some extracted rule. This is achieved by generalizing or specializing the attribute values through choosing a lower or a higher level respectively. Generalization or specialization is determined by identifying the similarity between the new object and each one of the extracted rules. Then, the new object is changed in order to match its most similar rule.

At first, the attributes based on different database schemas across various subsets are identified. Then, the similarity between the new object and each one of the extracted rules is calculated with respect to these attributes. We tested the “unify” operator using two different (dis)similarity measures: the chi-square distance \(d_{\chi^2}(A, B)\) (used e.g. in the $k$-modes clustering algorithm), and the ontology-based measure presented in \cite{8}.

The chi-square distance defines dissimilarity based on the number of mismatches of attributes, taking into account the frequencies of the attribute values with respect to the whole training set. Formally, if $A, B$ are two objects of the training set described by $m$ categorical attributes, then the dissimilarity measure is defined as:

\[
d_{\chi^2}(A, B) = \sum_{j=1}^{m} \frac{|s(a_{jk})| + |s(a_{jl})|}{|s(a_{jk})||s(a_{jl})|} \delta(a_{jk}, a_{jl}),
\]

where

\[
\delta(a_{jk}, a_{jl}) = \begin{cases} 
0, & \text{if } (a_{jk} = a_{jl}); \\
1, & \text{if } (a_{jk} \neq a_{jl}).
\end{cases}
\]
and \(|s(a_{jk})|, |s(a_{jl})|\) is the number of objects in the training set that have the value \(a_{jk}, a_{jl}\), respectively assigned to the attribute \(a_j\)\(^{21}\).

The ontology-based measure is based on user-defined ontologies, which represent domain-specific knowledge. (Dis)similarity is defined with a common to all the ontologies procedure which takes into consideration the structure of each ontology. The similarity between any two attribute values is represented by the distance between the corresponding nodes of the concept hierarchy as defined by the following formula:

\[
d(X, Y) = \frac{1}{fl(X,Y)+1} \times \text{Average}\left(\frac{l(X) - fl(X,Y)}{\max(p(X))}, \frac{l(Y) - fl(X,Y)}{\max(p(Y))}\right) \times (1 + \frac{p(X,Y)}{\max(p(X,Y))})
\]

where \(X\) and \(Y\) represent any two nodes (attribute values) of the concept hierarchy, \(fl(X, Y)\) is the level of the nearest common father node of \(X\) and \(Y\) nodes, i.e. the level of the nearest common predecessor, \(l(X)\) is the level of node \(X\), i.e. the number of nodes in the path from that node up to the root, \(\max(p(X))\) is the length of the maximum path starting from the root and containing node \(X\), \(p(X,Y)\) is the length of the directed path connecting \(X\) and \(Y\). The ontology-based measure is proved to be a metric\(^8\).

Thus, the proposed algorithm is as follows:

**The proposed algorithm.**

**input** data set \(D\{a_1, \ldots, a_n\}\), represented by \(d\)-tuples \(\{a_{1k}, \ldots, a_{dl}, a_{ci}\}\), where \(a_c\) is the class attribute

**input** \(b\) artificially or physically created subsets of \(D\)

for each subset \(D_i, 1 \leq i \leq b\)
do apply a classification algorithm to extract the rules \(R_i\)
set \(R = \bigcup_{i=1}^{b} R_i\)

\(R = \text{merge}(R)\) – apply the “merge” operator
\(R = \text{resolve.conflict}(R)\) – apply the “conflict” operator
\(R = \text{remove.redundancy}(R)\) – apply the “redundancy” operator

while there is a new object \(o_i = \{a_1 = a_{1x}, \ldots, a_d = a_{dx}\}\) to be classified
do apply the “unify” operator to find the most similar rule
\(r_i \in R = IF a_1 = a_{1x}, \ldots, a_d = a_{dx} THEN a_c = a_{cx}\)
classify \(o_i\) as \(a_{cx}\)

In order for the proposed methodology to be valid, operators must be applied in the specific sequence shown above. This is obvious from the next subsection.

### 4.1. Validity of the operators

The “merge”, “conflict” and “redundancy” operators create new rules or delete existing ones. The proof of their validity is reduced to the proof that the created rules could also be extracted by a single classifier supplied with all the data.
In the following, we represent a rule:

\[ r \equiv IF \ a_i = a_{i_1} \land \ldots \land a_j = a_{j_1} \ THEN \ a_c = a_{c_1} \]

as its clausal form: \( \neg a_{i_1} \lor \ldots \lor \neg a_{j_1} \lor a_{c_1} \), which is also equivalent to the form:

\[ a_{i_1} \land \ldots \land a_{j_1} \lor a_{c_1} \]

Also, we define the set of objects \( s(a_i = a_{i_1} \land \ldots \land a_j = a_{j_1} \land a_c = a_{c_1}) \) covered by a rule \( r \equiv IF \ a_i = a_{i_1} \land \ldots \land a_j = a_{j_1} \ THEN \ a_c = a_{c_1} \) as those objects represented (i.e. they have the same attribute-value pairs) by the tuple \( a_i = a_{i_1} \land \ldots \land a_j = a_{j_1} \land a_c = a_{c_1} \).

The “merge” operator creates new rules in the sense of recalculating the coverage of the rule produced by merging identical ones which are extracted from different subsets. The key to validness of the “merge” operator is that any rule extracted from at least one of the subsets can be potentially extracted from a single classifier supplied with all the data.

Clearly, if

\[ r_1 = \ldots = r_m \equiv IF \ a_i = a_{i_1} \land \ldots \land a_j = a_{j_1} \ THEN \ a_c = a_{c_1} \]

are identical rules, then a rule \( r = r_1 = \ldots = r_m \) can be extracted from a single classifier supplied with all the data such that:

\[ \text{coverage}(r, t) \geq \frac{|s(a_{i_1}^r) \cap s(a_{i_1}^c) + \ldots + s(a_{m}^r) \cap s(a_{m}^c)|}{|s(a_{i_1}^r) + \ldots + s(a_{m}^c)|}. \]

The “conflict” operator also deletes extracted from subsets rules. Usually, classification algorithms guarantee the extraction of non conflicting rules or they handle them using special conflict resolving procedures. On the other hand, randomly generated subsets preserve the model of the initial data set and, thus, conflicting rules are not likely to be extracted. Extraction of conflicting rules is usually caused by biased subsets.

A solution to the problem of resolving conflicting rules, as \( r_1 \equiv a_{i_1} \land \ldots \land a_{j_1} \lor a_{c_1} \) and \( r_2 \equiv a_{i_1} \land \ldots \land a_{j_1} \lor a_{c_2} \), is to further process the set of objects \( s(a_{i_1} \land \ldots \land a_{j_1}) \), which are covered by these rules. Consider, for instance, the case of classifiers that extract decision trees using a splitting criterion. The set of objects \( s(a_{i_1} \land \ldots \land a_{j_1}) \) can be further processed by constructing a subtree rooted at the node \( a_{i_1} \land \ldots \land a_{j_1} \) that represents these objects. The tests used in the nodes of this subtree refer to attributes which are not included in the conflicting rules. There is also the case where the test used in the \( a_{i_1} \land \ldots \land a_{j_1} \) must be changed. Consider, as an example, the above conflicting rules \( r_1 \) and \( r_2 \). Without loss of generality we consider that \( \text{domain}(a_j) = \{a_{j_1}\} \) and \( \text{domain}(a_c) = \{a_{c_1}, a_{c_2}\} \). Then, using a splitting criterion based on entropy:

\[ E(a_j) = -f_z \log(f_z) - f_x \log(f_x), \]

where \( f_x = \frac{|s(a_{i_1} \land \ldots \land a_{j_1})|}{|s(a_{i_1} \land \ldots \land a_{j_1})|} \) and \( f_z = \frac{|s(a_{i_1} \land \ldots \land a_{j_1})|}{|s(a_{i_1} \land \ldots \land a_{j_1})|} \).

Clearly, in the subset \( t_1 \) from which \( r_1 \) is extracted,

\[ f_x = |s(a_{i_1} \land \ldots \land a_{j_1})| \ast \text{coverage}(r_1, t_1) \] and

\[ f_z = |s(a_{i_1} \land \ldots \land a_{j_1})| \ast (1 \ast \text{coverage}(r_1, t_1)). \]

Obviously, in order attribute \( a_j \) to be selected as a test, it is true that \( f_x > f_z \).
Similarly, in the subset \( t_2 \) from which \( r_2 \) is extracted,
\[
f_x = |s(a_{i_k} \land \ldots \land a_{p_s})| \ast \text{coverage}(r_2, t_2) < f_z \text{ and}
\]
\[
f_z = |s(a_{i_k} \land \ldots \land a_{p_s})| \ast (1 - \text{coverage}(r_2, t_2)).
\]
Thus, if the two subsets were processed together then it is possible that \( f_x \simeq f_z \Rightarrow E(a_j \gg 0) \) and the attribute \( a_j \) could not be selected as a test.

Note that the above results can also be obtained by different type of classifiers, e.g. by those that extract decision lists. However, such a solution is based on accessing the objects themselves. But, as mention in the introduction, the proposed algorithm must not rely to any access of the initial data set. Thus, conflicts in extracted rules are resolved using the following procedure:

\[
\text{procedure } \text{resolve\_conflict}() \text{ (ruleset)} \\
\text{for } p = 1 \text{ to } d \\
\text{do} \\
\text{for each set } cr = \{r \mid r \equiv a_{i_k} \land \ldots \land a_{p_s} \supset a_{c_i}, a_{c_i} \in \text{domain}(a_c), r \in \text{ruleset} \} \\
\text{do} \\
\text{if } |\text{rank}(r_{max}, D_{max}) - \text{rank}(r_i, D_i)| > g, \forall r_i \text{ rank}(r_{max}, D_{max}) > \text{rank}(r_i, D_i), r_{max}, r_i \in cr \\
\text{then delete } cr - r_{max} \text{ from } \text{ruleset} \\
\text{return } r_{max} \\
\text{else} \\
r_s = r \mid r \equiv a_{i_k} \land \ldots \land a_{p_s} \land \ldots \land a_{j_l} \supset a_{c_i}, a_{c_i} \in \text{domain}(a_c), r \in \text{ruleset} \\
\text{if } \text{ret=resolve\_conflict}(rs) = \text{null} \\
\text{then delete } cr \text{ from } \text{ruleset} \\
\text{else delete from } \text{ruleset} \text{ all rules } r_i \in cr \\
\text{which is in conflict with } \text{resolve\_conflict}(rs) \\
\text{return } \text{ret}
\]

\[
\text{procedure } \text{rank}(r, t) \\
\text{return } \text{coverage}(r, t) + \frac{|s(a_{c_i})|}{|D|}
\]

For every set \( cr \) of conflicting rules, starting from those with less attributes (\( p = 1 \text{ to } d \)), procedure resolve\_conflict() searches for a strong rule among the rules of the set. A strong rule must be stronger than every other rule to the degree that parameter \( g \) indicates. The strength of a rule is calculated (procedure \( \text{rank}() \)) based on both its coverage and the number of objects in the training set it covers (\( |s(a_{c_i}^x)| \)) with respect to the size of the training set \( |D| \). If such rule exists, all the other rules in \( cr \) are deleted. If there is no such strong rule, then procedure resolve\_conflict() tries to find a strong rule among any set containing more specific, possibly conflicting, rules than the conflicting rules of \( cr \). Thus, procedure resolve\_conflict() is a recursive procedure.

It is obvious that the resolve\_conflict() procedure could utilize the more specific
rules of the conflicted rules it tries to resolve. These more specific rules could be considered redundant and deleted if the “redundancy” operator had applied first. Thus, we consider the “redundancy” operator as different from the “merge” operator.

The “redundancy” operator deletes extracted from subsets rules. The validness of the operator is obvious since:

\[ a_{i_k} \land \ldots \land a_{p_s} \land \ldots \land a_j = a_{j_l} \supset a_{c_x} \equiv a_{i_k} \land \ldots \land a_{p_s} \supset a_{c_x} \]

thus, the knowledge there exists in two redundant rules can be represented by the more general of them.

Contrary to the above, the “unify” operator does not create new rules. The proof of its validness is reduced to the proof that generalization/specialization of attribute values is consistent with the knowledge represented by the concept hierarchies. The “unify” operator generalizes (specializes) an attribute value \( a_i = a_{i_k} \) included in the representation of the new object, in order to match the attribute value \( a_i = a_{i_l} \) included in the representation of its most similar rule. On the other hand, edges of the concept hierarchies represent inheritance from more general concepts (represented by nodes in lower levels) to more specific ones (represented by nodes in higher levels). Thus, generalization (specialization) of \( a_i = a_{i_k} \) to \( a_i = a_{i_l} \) is consistent with knowledge represented by the corresponding concept hierarchy only if there is a directed path from the node representing \( a_i = a_{i_k} \) (\( a_i = a_{i_l} \)) to the node representing \( a_i = a_{i_l} \) (\( a_i = a_{i_k} \)).

We guarantee the existence of such a path by applying the tree compression technique presented in \(^{36}\). This technique enables the verification that \( a_{i_k} \) is a subclass of \( a_{i_l} \), inheriting some or all of the defining properties of \( a_{i_l} \), by comparing a pair of numbers, attached to \( a_{i_k} \) in a preprocessing phase, with another such pair attached to \( a_{i_l} \).

---

**Fig. 1. Compressing a concept hierarchy.**
Consider, for example, the concept hierarchy shown in Figure 1. A pair of numbers, which will be referred to as a pair and will be considered as an interval when needed in what follows, is assigned to every node. The first number reflects the relative position of each node in a preorder (depth-first) traversal of the concept hierarchy. The second number is the maximum preorder number of the subtree rooted at that node. It is assumed that the second number assigned to each leaf node is the same as the first number. In this encoding, a node B is a subclass of a node A iff the pair of B is included in or is equal to the pair of A. Notice that this verification is achieved without having to search for a path from B to A. For instance, we can derive the fact that I is a C because the pair [9 9] assigned to the node I, is included in the pair [4 9] assigned to node C. The latter is not true for instance between I and D. The storage required for this compression scheme is $O(n)$ and the time complexity is $O(1)$, i.e. the complexity of the comparison.

5. Empirical results

In order to evaluate the proposed algorithm for the scaling-up problem, we applied it to various test sample databases, which have already been used as test data sets to evaluate genuine classification techniques. More specifically, we evaluate the proposed algorithm over five popular databases supplied by the UCI Machine Learning Repository, namely the "Mushrooms", "tic-tac-toe", "promoter recognition", "DNA splice junctions" and "KRKP" databases. The "Mushrooms" database, having 8124 objects described by 23 attributes, classifies mushrooms as poisonous or edible, in terms of their physical characteristics. The "Mushrooms" database has already been used to evaluate numerous different classification algorithms while the best possible accuracy has already been achieved using some of them. The "tic-tac-toe" database, having 958 objects described by 9 attributes, encodes the complete set of possible board configurations at the end of tic-tac-toe games, where "x" is assumed to have played first. The target concept is "win for x" (i.e., true when "x" has one of 8 possible ways to create a "three-in-a-row"). The "promoter recognition" database, containing 106 examples of DNA sequence, belongs to the domain of Molecular Biology and especially to DNA sequence analysis. The "DNA splice junctions" database contains 3190 examples of DNA sequence also, where each one consists of 60 sequential DNA nucleotide positions. Splice junctions are points on a DNA sequence at which 'superfluous' DNA is removed during the process of protein creation in higher organisms. The problem posed in this database is to recognize, given a sequence of DNA, the boundaries (EI and IE sites) between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out). Each DNA sequence is classified as to contain an EI site, an IE site or neither. The "KRKP" database contains 3196 objects representing board-descriptions for a chess endgame, (King+Rook versus King+Pawn on a7 - the pawn on a7 means it is one square away from queening). Each object is a sequence of 36 attribute values and it is classified as "win" or "nowin".
Moreover, in order to evaluate the proposed algorithm for the incompatible schema problem, we applied it to a real-world database, the “telecom” database. The “telecom” database is a subset of a larger one which includes operational data of a big telecommunications company. Attributes in the “telecom” database represent profit related behavior of the customer base and the class attribute represents a certain customer ranking. We used two of them, i.e. the address and the job attributes, to describe 3116 objects. We were also provided with a concept hierarchy for each one of the two attributes.

We used cross-validation as the testing methodology. More specifically, we used 5-fold cross-validation in which the training set is permuted and divided into 5 disjoint sets. In each evaluation phase, one set, which has never been seen during learning, is used for testing (test set), while the examples of all the remaining 4 sets are used as training set by applying the proposed classification algorithm. Hence, the evaluation process requires 5 training phases.

Empirical tests aim at evaluating the proposed algorithm with respect to an analog non-distributed classification process. More specifically, we have compared the performance of the proposed algorithm with the application of the very popular C4.5 classification algorithm to all the data. Also, we have tested the proposed methodology against varying number of subsets in order to measure the speed-up with respect to the analog non-distributed classification process. Speed-up is the ratio of the execution time of the C4.5 algorithm applied to all the data to the execution time of the proposed algorithm based on the longer execution time for a subset.

In Fig. 2 the classification accuracy is depicted for the five databases, as a function of the number of subsets. The classification accuracy of the analog non-distributed process is indicated as the one subset value in the horizontal axis. The classification accuracy is defined as the average accuracy of the five training phases. Also, the standard deviation of accuracy is shown next to the number of subsets in the horizontal axis. It seems that the classification accuracy decreases as the number of subsets increases.
In Fig. 3 the speedup is depicted as a function of the number of subsets for the five test databases. Time performance of the proposed algorithm is defined as the average maximum (among different subsets) execution time of the five training phases plus the execution time to apply the operators. It seems that speedup is proportional to number of subsets, at a cost of a lower accuracy.

Notice that DNA sequence analysis problems are used as benchmarks for comparing the performance of learning systems. Especially, the “promoter recognition” and “DNA splice junctions” databases have been used for evaluating a great number of different classification algorithms: from decision tree based algorithms and neural network based algorithms to hybrid algorithms based on a domain knowledge. Also, “Mushrooms”, “tic-tac-toe”, “DNA splice junctions” and “KRKP” databases have
already used to empirically evaluate various algorithms for the scaling-up problem based on either windowing \cite{18,16} or partitioning \cite{10,11,16} approach. Thus, one could obtain a straightforward comparison of the proposed algorithm to related ones. Summarizing the empirical results with respect to related algorithms, the conclusion is that the proposed algorithm exhibits a higher speed up while the accuracy is lower for small number of subsets. However, the accuracy of the proposed algorithm for the scaling-up problem is decreased dramatically for large number of subsets, although the speed up increases too much. An example of the latter is shown in Table 1 for both the accuracy and the speed up.

Finally, in order to evaluate the proposed algorithm for the incompatible schema problem, the “telecom” database is partitioned into subsets following two different strategies. The first strategy partitions the “telecom” database randomly. Thus, it is not guaranteed that subsets are based on different database schemas. The
Table 1. Accuracy and speed up for various algorithms.

<table>
<thead>
<tr>
<th></th>
<th>windowing</th>
<th>partitioning</th>
<th>proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Mushrooms&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 subsets</td>
<td>≈ 99.9% (100% max)</td>
<td>99.6% (100% max)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.1 – 2.9</td>
<td>3.37</td>
<td></td>
</tr>
<tr>
<td>10 subsets</td>
<td>≈ 99.5% (100% max)</td>
<td>98.8% (100% max)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 – 9.7</td>
<td>12.13</td>
<td></td>
</tr>
<tr>
<td>&quot;DNA splice junctions&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 subsets</td>
<td>≈ 94% (95% max)</td>
<td>83.8% (90% max)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6.61</td>
<td></td>
</tr>
<tr>
<td>10 subsets</td>
<td>≈ 89 – 93% (95% max)</td>
<td>64.2% (90% max)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>&quot;KRKP&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 subsets</td>
<td>≈ 98.5% (100% max)</td>
<td>93% (99% max)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.9 – 3.4</td>
<td>3.98</td>
<td></td>
</tr>
<tr>
<td>10 subsets</td>
<td>≈ 93% (100% max)</td>
<td>76.6% (99% max)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.5 – 12</td>
<td>19.56</td>
<td></td>
</tr>
</tbody>
</table>

second strategy partitions the “telecom” database into subsets so that each subset includes objects described, among others, by some randomly chosen attribute values that they are not used to describe objects included in the other subsets. Thus, subsets are based on different database schemas simulating the conditions met in the incompatible schema problem.

In Fig. 4 the classification accuracy is depicted for the “telecom” database, as a function of the number of subsets, using both of the above strategies. The test performed without applying the “unify” operator.

![Fig. 4. Accuracy as a function of the number of subsets.](image-url)
In Fig. 5 the classification accuracy is depicted after applying the “unify” operator, using both of the above strategies. The test is performed using the two implementations of the “unify” operator mentioned in Section 4, i.e. the ontology-based and the chi-square distance. It seems that the application of the “unify” operator is very effective in the second strategy which corresponds to the incompatible schema problem. It seems that the implementation of the “unify” operator is critical. It is worth mentioning that the ontology-based “unify” operator is capable to achieve a classification accuracy that is close to the classification accuracy achieved by a single classifier supplied with all the data. This is an important result, since to a single classifier all the information is available while in the incompatible schema problem to each participated classifier only a part of the whole information is available.

Fig. 5. Accuracy as a function of the number of subsets.
6. Discussions and conclusions

The proposed algorithm is a novel algorithm for attacking both the scaling-up and the incompatible schema problems. Empirical results demonstrate that the proposed algorithm has a significant speedup with respect to the analog non-distributed classification algorithm, at a cost of a lower classification accuracy. Also, the proposed algorithm exhibits a higher speed up and a lower accuracy with respect to related algorithms. Notice also that the proposed algorithm does not require a substantial pre-processing phase. Finally, in order to synthesize the final set of classification rules, the proposed algorithm is based only on the partial classification rules extracted from subsets without accessing the initial data set.

There are several variations of the proposed algorithm regarding heuristics for implementing the “merge”, “conflict”, “redundancy” and mainly the “unify” operator. For instance, the “unify” operator could be applied before the “merge”, “conflict” and “redundancy” operators in order to transform subsets of rules to the same generalization or specialization level with respect to the concept hierarchy. At first, for each attribute \( a_i \), all the different attribute values appearing in the extracted rules are identified, forming the set \( V_{a_i} \subseteq \text{domain}(a_i) \). Then, a clustering process is applied to each such set \( V_{a_i} \) in order to identify clusters of attribute values. Then, every attribute value appearing in the extracted rules is substituted by the medoid of the cluster it belongs to. Thus, after the substitution, some rules are now redundant or conflicting or identical. These rules are treated accordingly using the “redundancy”, “conflict” and “merge” operators. Finally, the substitutions are stored, so that these substitutions can be applied later to any new object for classification.

Another variation to the above technique is to find the mean level of nodes representing attribute values that belong to a cluster and transform any attribute value to this level. If an attribute value \( a_{ij} \) is chosen to be generalized, this is achieved by simply following the path of the concept hierarchy \( T_{a_i} \) from the node representing \( a_{ij} \) to its parent node in the mean level. If an attribute value \( a_{ij} \) is chosen to be specialized, this is achieved by following the path of the concept hierarchy \( T_{a_i} \) from the node representing \( a_{ij} \) to its appropriate child node in the mean level. Generally, there are more than one such nodes. One criterion to choose the appropriate node is to choose the one that represents the most common to the training set attribute value.

The above techniques have serious limitations in the case where there are conflicting rules in different generalization/specialization levels. In that case, some of the rules are forced to be lost, since after the substitution the “conflict” operator is allowed to be applied. The overall speed-up and accuracy depends on the variations of the proposed algorithm similar to those describe in the previous paragraphs. However, although such variations could improve the speed-up and/or the accuracy, the presented in this paper basic strategy performs well on numerous different data sets.
References


15. S. Dao and B. Perry, Applying a data miner to heterogeneous schema integration, in *Proc. of the 1st Int. Conf. on Knowledge Discovery and Data Mining*, (Montreal 1995) pp.63–68.


34. A. Savasere, E. Omiecinski and S. Navathe, An Efficient Algorithm for Mining Association Rules in Large Databases, in *Proc. of 21th IEEE Int. Conf. on Very Large Databases*, (Switzerland 1995), pp.432–444.