Inflating Examples to Obtain Rules

Oscar Luaces,* Antonio Bahamonde†
Artificial Intelligence Center, University of Oviedo at Gijón, Campus de Viesques, 33271, Gijón, Spain

A new machine learning system is presented in this article. It is called INNER and induces classification rules from a set of training examples. The process followed by this system starts with the random selection of a subset of examples that are iteratively inflated in order to cover the surroundings provided that they are inhabited by examples of the same class, thus becoming rules that will be applied by means of a partial matching mechanism. The rules so obtained can be seen as clusters of examples and represent clear evidence to support explanations about their future classifications and may be used to build intelligent advisors. The whole algorithm can be seen as a set of elastic transformations of examples and rules and produces concise, accurate rule sets, as is experimentally demonstrated in the final section of the article. © 2003 Wiley Periodicals, Inc.

1. INTRODUCTION

Let us consider a set of training examples described by attributes with different kinds of values: symbolic (or literal) and numerical. Once we have attached a category label or class to every example, a supervised learning algorithm tries to build a collection of simple classification procedures able to guess the class from the attribute values of a given, unseen example.

A useful means of describing these machine learning tools is to consider the attribute space (the set of all possible examples) from a metric viewpoint. Here, examples are points and classification procedures, usually called *rules,* can be depicted by regions usually limited by borders parallel to the axes with an endowed class label: respectively, the rule body and conclusion. Hence, to classify a new example we need only to observe which region its coordinates belong to; its class label will be the classification returned.

This classification mechanism can, however, be relaxed; when the coordinates of an example do not exactly belong to any rule body, we can use the nearest one instead.[1] In this case, we no longer need to have large rule bodies: certain

*Author to whom all correspondence should be addressed: e-mail: oluaces@aic.uniovi.es.
†e-mail: antonio@uniovi.es.

© 2003 Wiley Periodicals, Inc. Published online in Wiley InterScience (www.interscience.wiley.com). • DOI 10.1002/int.10132

strategically placed decision regions can play the same role.[2–4] We can even have a collection of paradigmatic examples to classify new cases.[5–9]

One of the alleged advantages of exact classification systems (those where points always belong to at least one rule body) such as the paradigmatic C4.5[10] is the explicitness of their solutions, but they may fail when decisions borders are not parallel to the axes. On the other hand, minimum distance classifiers may be adapted to a wavy geometry but may produce heavily implicit solutions.

The RISE system[11] tries to exploit the best of both worlds, and produces a collection of rules whose bodies range from points to large regions parallel to the axes. The classification accuracy of these rules is very high, but the number of rules needed is sometimes too high to tag their solutions as explicit.

In this article, we present a set of elastic transformations of learning objects: training examples and decision rules. These objects are allowed to inflate their frontiers to become more general, explicit, ground classification areas. During this expansion process, some regions may blend together like bubbles to give rise to a single, stronger rule; the maxim is to make implicitness explicit in classification areas.

The achievement of this elasticity is that a set of training examples can lead us to a small, explicit, accurate set of rules to be evaluated by means a nearest-neighbor (or partial matching) strategy. The machine learning system thus obtained is presented here. We call it INNER: an approximate acronym of the process description, given that the system advances by INflating EXamples to obtain RUles. There are a number of techniques[12] concerned with reducing the size of the knowledge induced from a set of training examples. What we would like to emphasize is that in addition to its reduced size, the classification knowledge induced by INNER is highly intuitive because we only allow our rules to grow while they have enough training examples in the surroundings to support their generalization. The reward is that the explanations that can accompany INNER classifications (the mention of the rule used) look very much like a representative evidence cluster. The very small sizes of rule sets obtained add more credibility to the eventual explanations of INNER.

The generalization device employed is heavily inspired by Kohonen's selforganizing maps[13] and is based on the definition of distances between learning objects: examples and rules.[14–17] After a section devoted to giving an overview of our system, we spell out the metrics used throughout the article. We have to deal with numeric as well as symbolic differences. The core idea is that we can learn to measure differences at the same time as we learn how to classify new cases. The result is a metric that takes into account local peculiarities of relevant regions of the attribute space.

In order to illustrate INNER's performance, we close the article with a report of some experimental results obtained from a set of well-known learning datasets[18,19] taken from the UCI repository.[20] In this final section, we compare the accuracy, number and size of rules induced by C4.5, RISE, and our system.

2. OVERVIEW

Our learning system starts from a collection of training examples and is able to induce classification rules described by two parts. The left-hand side or conclu-

```
Function INNER (Examples) : Set Of Rules
Theory=∅
UnCovered=Examples
while not(STOPCRITERION (UnCovered)) do
    Rules=FINDBESTRULES (Examples, UnCovered)
    Theory=Theory ∪ Rules
    Covered=COVERAGE (Theory, Examples)
    UnCovered=Examples\Covered
end while
Theory=POSTPROCESS (Theory, Examples)
return (Theory)
```

Figure 1. INNER's main loop.

sion is given by the name of a class; the right-hand side or body of the rule is a conjunction of conditions or antecedents. We will use the following syntax:

$$C \leftarrow Atr_1 \in [x_1, x_2] \wedge Atr_2 \in \{v_1, v_2, \cdots\} \wedge \cdots$$

where Atr_1 is an attribute with numeric values, while Atr_2 has symbolic values instead. Examples fulfilling these rules should have values of attribute Atr_1 between x_1 and x_2, and a symbolic value of Atr_2 belonging to the set $\{v_1, v_2, \cdots\}$, etc. However, we are going to allow slightly different values to apply the rule and conclude that the class must be C. The tolerance threshold will depend of the remaining available rules, because our evaluation principle is based on the nearest-neighbor rule.

In fact, the evaluation procedure guides INNER's induction mechanism. The overall description of the algorithm (see Fig. 1) can be seen as a variant of the *separate-and-conquer* family.[21] The main difference is that we do not dispose of covered examples to generate new classification rules; instead, we propose a kind of *iterate-and-conquer.* In each cycle of the main loop, we randomly choose a small set of training examples of every class by means of FINDBESTRULES. Considered as point rules, their descriptions are generalized (by inflating them; see Fig. 2) in an attempt to improve their classification accuracy, whereas at same time aiming at making their correct classification areas explicit.

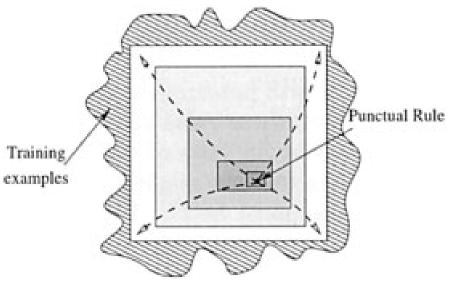

Figure 2. The generalization process consists of inflating rule descriptions as far as possible while the classification quality is maintained or improved.

If e is a training example, its point rule version is simply

$$\textsc{ClassOf}(e) \leftarrow a1 \in [e_{a1}, e_{a1}] \wedge a2 \in \{e_{a2}\} \wedge \cdots$$

where $\textsc{ClassOf}(e)$ is the class of our example, ai are attribute names, and the body of the rule is a conjunction where we have singleton sets formed by the example values attached to each attribute; e_{ai} is the value of example e on attribute ai.

The rules obtained by inflating these point rules are saved and used to compute which training examples are now explicitly covered (this is the function of COVERAGE); the next iteration will inflate a new set of uncovered examples. The stopping criterion is related to the number of training examples covered by the rules generated so far, because the goal is to cover the regions to be classified explicitly. By default, we require at least 95% of the training examples of each class to be covered. However, the maximum number of cycles allowed is 5.

Notice that we use two sets of examples: the original training set, and the collection of uncovered examples. Thus, if we do not obtain high-quality rules in one iteration, we may try again by picking up better starting points whose generalizations may overlap the inappropriate rules. Finally, once we have gathered a set of classification rules in our *Theory,* we try to improve the whole set of rules in a POSTPROCESS step. Here we dispose of unnecessary or erroneous rules and we try to join neighboring, coherent decision areas. In a certain sense, we make rules to play the role of individual examples in the inflating mechanism; see Fig. 11 in Section 5.

3. METRICS USED DURING THE LEARNING PROCESS

Most of INNER's skill is in its inflating procedure, which is based on an original way of measuring distances between learning objects. In fact we are not going to deal with a metric in the mathematical sense of the word. Instead we will have heuristics able to judge the utility of a rule growing in the direction of a given training example.

Instance-based learning systems are usually concerned with distances from examples to examples; a simple generalization leads us to consider distances from rules to examples, and even more so from rules to other rules. In INNER, we will have rules and examples in the inflating stage, but only rules in the final rule set improvement step. In the following, we present and discuss some distance functions to compute distances between examples that will be later generalized to be used with rules.

Our system is able to handle both numeric and symbolic values within the same case, so we need some kind of heterogeneous function measure. An initial candidate is a function similar to the one used by Aha's IBx algorithms[5,6] called the Heterogeneous Euclidean-Overlap Metric[22] defined by

$$\mathrm{HEOM}(x, y) = \sqrt{\sum_{a=1}^{m} d_h(x_a, y_a)^2} \qquad (1)$$

$$d_h(x_a, y_a) = \begin{cases} 1, & \text{if } x_a \text{ or } y_a \text{ are missing} \\ overlap(x_a, y_a), & \text{if } a \text{ is a symbolic attr.} \\ E(x_a, y_a), & \text{if } a \text{ is a continuous attr.} \end{cases} \quad (2)$$

$$overlap(x_a, y_a) = \begin{cases} 0, & \text{if } x_a = y_a \\ 1, & \text{if } x_a \neq y_a \end{cases} \quad (3)$$

$$E(x_a, y_a) = \frac{|x_a - y_a|}{\max_a - \min_a} \quad (4)$$

where x and y are examples described by m attributes, and $\max_a$ and $\min_a$ are respectively, the highest and the lowest values observed in the training set for attribute a.

The main defect of HEOM is the use of *overlap* to decide symbolic differences. Instead, we might try to take advantage of any additional information available on training examples that may become relevant for classification purposes. So, for instance, it may happen that the colors green and red appear in similar proportions in all classes, whereas the occurrence distribution of blue is quite different; then it is reasonable to assume that green and red are near (from a metric viewpoint) values but far from blue cases when we are classifying. This idea is stressed by the so-called virtual difference metric, VDM,[23] later developed into several variants such as HVDM, IVDM or WVDM, introduced in Ref. 22 or SVDM.[11]

Our system incorporates a version of HEOM called $\mathcal{K}$-HEOM, since it is heavily inspired by the ideas involved in Kohonen's selforganizing maps. Given that our training stage has to decide which values to incorporate into rule conditions, we will deal in the symbolic case with a kind of membership function for each rule and condition. Since our symbolically valued attributes have a finite number of possible values, real number vectors can codify the membership functions, where there must be one component for each possible value; we call these vectors *difference tables.*

To give the formulas of our $\mathcal{K}$-HEOM, let us consider an example e (described by m attributes) and a rule with difference table T_a attached to a generic attribute a; for each individual value e_a of a, $T_a[e_a]$ stores the difference from the rule. Then, we define

$$\mathcal{K}\text{-HEOM}(Rule, e) = \sqrt{\sum_{a=1}^{m} d_{\mathcal{K}}(Rule_a, e_a)^2}, \quad (5)$$

where $Rule_a$ is the set attached to attribute a in the Rule's body.

$$d_{\mathcal{K}}(Rule_a, e_a) = \begin{cases} 1, & \text{if } e_a \text{ is missing} \\ D_{\mathcal{K}}(Rule_a, e_a), & \text{if } a \text{ is a symbolic attr.} \\ E_{\mathcal{K}}(Rule_a, e_a), & \text{if } a \text{ is a continuous attr.} \end{cases} \quad (6)$$

$$D_{\mathcal{K}}(Rule_a, e_a) = \begin{cases} 0, & \text{if } T_a[e_a] \leqslant 0 \\ 1, & \text{if } T_a[e_a] \geqslant 1 \\ T_a[e_a], & \text{otherwise} \end{cases} \quad (7)$$

$$E_{\mathcal{H}}(Rule_a, e_a) = \begin{cases} 0, & \text{if } e_a \in Rule_a = [x_1, x_2] \\ \dfrac{\min\{|x_1 - e_a|, |x_2 - e_a|\}}{\max_a - \min_a}, & \text{if } e_a \notin Rule_a = [x_1, x_2] \end{cases} \tag{8}$$

During the training stage, difference tables will be modified in an attempt to adapt their values to a successful classification role. Hence, some individual differences may fall outside the interval [0, 1], as shall be explained in the next section. So, in order to compute distances with $D_{\mathcal{H}}$, we will consider negative values as 0 and values above 1 in the table as 1.

Once we have induced a set of classifying rules, in order to classify a case we will find the nearest rule and return the conclusion class of that rule. In other words, we use a partial matching strategy. The metric used to evaluate test examples is not $\mathcal{H}$-HEOM. Instead, a simple HEOM is sufficient, because all metric knowledge is already included in our rules. However, we did investigate the effects of extending the use of $\mathcal{H}$-HEOM during the evaluation stage, and in Section 6 we discuss the results obtained in this way in a version of our system that we call $\mathcal{H}$-INNER. The difference in accuracy is not significant.

Finally, in the POSTPROCESS step, we need to measure distances from rules. The formulas used are simply a generalization of $\mathcal{H}$-HEOM and can be found in Appendix A.

4. INSTANCE GENERALIZATION

The core stage of INNER is described in this section. Here the task is to obtain a set of reasonable classification rules starting from a collection of randomly chosen training examples called seeds. The algorithm used can be seen in Fig. 3.

First of all we select a small number of uncovered examples of each class. By default, we need 10 per class unless there are classes with few representatives or if we have covered all the available examples of a given class in previous iterations. We then reduce the number of selected uncovered seeds or we choose already covered examples. The cumbersome details about the election of examples to be generalized are explained in Appendix B.

The selected seeds are converted into punctual rules, as mentioned in Section 2. The problem arises when we find missing values in the examples. We must then write some credible values instead. For numerically valued attributes, we use the average of the attribute in the same class as our example. On the other hand, when missing values appear in a symbolically valued attribute a, we need a plausible difference table. Thus, for each possible value v of attribute a, we define

$$T_a[v] = 1 - P(v|C), \tag{9}$$

where $P(v|C)$ is the probability of finding the value v on attribute a in a training example of class C.

To illustrate the generalization mechanism of our system, we generated 1000 examples described by 3 attributes: x with numeric values in [2.0, 8.0]; flavor, symbolic, with possible values {sweet, salty}; and color with values in {red, green,

```
Function FINDBESTRULES (Examples, UnCovered) : Set Of Rules
RuleSet=$\emptyset$
for each class C mentioned in Examples do
   CUnCoveredExs=EXAMPLESOFCLASS (C, UnCovered)
   if CUnCoveredExs $\neq \emptyset$ then
      RuleSet=RuleSet $\cup$ INITEXSASRULES (CUnCoveredExs)
   else
      RuleSet=RuleSet $\cup$ INITEXSASRULES (EXAMPLESOFCLASS (C, Examples))
   end if
end for
RuleSet=GENERALIZEINSTANCES (RuleSet, Examples)
RuleSet=PRUNECONDITIONS (RuleSet, Examples)
return (RuleSet)

Function GENERALIZEINSTANCES (RuleSet, Examples) : Set Of Rules
for each time from 1 to M do
   /* M is the number of times that we repeat the examples; it is
   the minimum integer that guarantees 3000 presentations */
   Examples=RANDOMLYSORT (Examples)
   for each Ej$\in$Examples do
      Rule=NEARESTRULE (RuleSet, Ej)
      Rule=GENERALIZERULE (Rule, Ej)
      if ISTIMETOREGULARIZE (j) then
         RuleSet=REGULARIZE (RuleSet, Examples)
      end if
   end for
end for
return (RuleSet)
```

Figure 3. INNER's instance generalization procedure. Training examples are presented several times (M) to a small set of starting seeds that inflate their descriptions according to the stimuli so received. This is repeated (no more than 5 times) until the rules obtained cover a significant part of the training examples. A final generalization process returns the induced rule set.

blue}. These were artificially labeled with a class tag according to the following rules:

$$\text{class A} \leftarrow x \leq 6.5 \wedge \mathit{flavor} \in \{\mathit{sweet}\} \wedge \mathit{color} \in \{\mathit{red}\}$$

$$\text{class A} \leftarrow x \leq 6.5 \wedge \mathit{flavor} \in \{\mathit{salty}\} \wedge \mathit{color} \in \{\mathit{blue}, \mathit{green}\}$$

$$\text{class B} \leftarrow \text{in all other cases.}$$

Additionally, 58 randomly selected examples were misclassified to add some noise to the training data.

Let us suppose that we start with 4 examples to be generalized: 2 of class A and 2 of class B (see Table I). These are considered as punctual rules and internally represented, endowed with the corresponding difference tables (Table II).

Continuing with the general case, we need to inflate our recently built punctual rules, trying to make explicit their implicit and correct decision areas. To do this iteratively, we present each training example and modify the nearest rule. The influence of the example over the rules depends on the example's class and the distance from the example to the rule. The overall idea is that rules concluding the

Table I. Salad data set.

Class	x	Flavor	Color
A	3.932	salty	red
A	2.582	salty	green
B	2.502	salty	red
B	2.450	sweet	blue

Starting seeds to be generalized by INNER. Observe that according to the given class definition, the first example is a noisy one: its class should be B instead of A, or its flavor should be sweet, or its color should be blue.

same class as the example should grow, trying to shorten their distance, while we should enlarge distances when classes are different. The magnitude of the movement of rule frontiers will be inversely proportional to the distance between example and rule.

To modify the nearest rule, we consider each attribute included in its condition list, changing the respective difference tables for symbolically valued attributes; in the numerical case, we move the extremes of the intervals. For both types of attributes, modifications are made by means of a Kohonen-like formula, where constants are empirically obtained default values; the results are not highly dependent on these values. Equations (10) to (13) show the way we modify the extremes of a numerical antecedent.

$$Ext := Ext \pm h_n(D, t) \cdot d_a \tag{10}$$

Table II. Salad data set.

External representation	Associated difference tables				
$A \leftarrow x \in [3.932,\ 3.932] \wedge$	Flavor		Color		
$\wedge$ flavor $\in$ {salty} $\wedge$ $\wedge$ color $\in$ {red}	Sweet 1	Salty 0	Red 0	Green 1	Blue 1
$A \leftarrow x \in [2.582,\ 2.582] \wedge$	Flavor		Color		
$\wedge$ flavor $\in$ {salty} $\wedge$ $\wedge$ color $\in$ {green}	Sweet 1	Salty 0	Red 1	Green 0	Blue 1
$B \leftarrow x \in [2.502,\ 2.502] \wedge$	Flavor		Color		
$\wedge$ flavor $\in$ {salty} $\wedge$ $\wedge$ color $\in$ {red}	Sweet 1	Salty 0	Red 0	Green 1	Blue 1
$B \leftarrow x \in [2.450,\ 2.450] \wedge$	Flavor		Color		
$\wedge$ flavor $\in$ {sweet} $\wedge$ $\wedge$ color $\in$ {blue}	Sweet 0	Salty 1	Red 1	Green 1	Blue 0

The examples of Table I now considered as punctual rules with their associated difference tables.

$$h_n(D, t) = \alpha_n(t) \cdot S_n(D) \tag{11}$$

$$\alpha_n(t) = 0.75 \cdot \left(1 + \frac{t}{M}\right) \tag{12}$$

$$S_n(D) = \frac{1}{1 + e^{20D-5}}. \tag{13}$$

Here, *Ext* is the nearest extreme to the value of the considered attribute in the example being presented, provided this value is not in the interval; in which case, the numerical interval will not be changed. The extreme will be modified as a percentage of d_a, the distance to the attribute's value of the example. This percentage is computed by (11) depending on the learning rate $\alpha_n(t)$, a value that decreases linearly as long as examples are presented (M is the number of times that we repeat the examples), and the neighborhood function $S_n(D)$, a sigmoid used to smooth changes depending on D, the distance between the rule and the example.

In the symbolic case, we obtain the same effect by modifying the difference tables in a similar way. As Equation (14) shows, for a symbolic antecedent corresponding to attribute a, we will modify the entry in its difference table T_a, whose index is e_a, the attribute's value in the example:

$$T_a[e_a] := T_a[e_a] \pm h_s(D, t) \cdot (T_a[e_a] + 1) \tag{14}$$

$$h_s(D, t) = \alpha_s(t) \cdot S_s(D) \tag{15}$$

$$\alpha_s(t) = 0.675 \cdot \left(1 - \frac{t}{M}\right) \tag{16}$$

$$S_s(D) = \frac{1}{1 + e^{10D-5}} \tag{17}$$

Notice the use of $(T_a[e_a] + 1)$ in (14), which gives the system the ability to modify the difference to an initially covered example (an example for which $T_a[e_a] = 0$). This leads to a selforganizing procedure whose benefits can be appreciated in the salad data set introduced above. Thus, the initial punctual rules (Table II) become the rules represented in Table III, where the first rule has modified the color of its condition de facto from red to blue. Thus, the rule is now responding to the specification of a correct classification of this data set. The unpleasant consequence of this equation is that it can lead us to obtain difference values lower than 0 or higher than 1, as was anticipated in Section 3. The range of differences is actually $[-1, \infty)$, though it is trimmed to $[-1, 2]$.

From time to time (see function IsTimeToRegularize in Fig. 3), we proceed to fix the frontier movements of our rules; we call this process regularization. The criterion is quite simple: we consolidate rule shapes whenever their classification quality has improved since the last control; otherwise, no changes are made. Regularization is applied only once for symbolic antecedents, just when the learning process has finished, to decide which values should be included in the conditions established by every antecedent.

Table III. Salad data set.

External representation	Associated difference tables				
$A \leftarrow x \in [3.93, 3.93]$ $\wedge$	Flavor		Color		
$\wedge$ flavor $\in$ {salty} $\wedge$	Sweet	Salty	Red	Green	Blue
$\wedge$ color $\in$ {blue}	0.92	-1	0.58	1	-1
$A \leftarrow x \in [2.000, 6.122]$ $\wedge$	Flavor		Color		
$\wedge$ flavor $\in$ {sweet, salty}					
$\wedge$	Sweet	Salty	Red	Green	Blue
$\wedge$ color $\in$ {green}	-1	-1	1	-1	1
$B \leftarrow x \in [2.026, 7.970]$					
$\wedge$	Flavor		Color		
$\wedge$ flavor $\in$ {salty} $\wedge$	Sweet	Salty	Red	Green	Blue
$\wedge$ color $\in$ {red}	1.02	-1	-1	1	1
$B \leftarrow x \in [2.055, 7.973]$ $\wedge$	Flavor		Color		
$\wedge$ flavor $\in$ {sweet} $\wedge$	Sweet	Salty	Red	Green	Blue
$\wedge$ color $\in$ {blue}	0	1	1	1	0

The initial punctual rules of Table II, after being inflated. Notice that the first rule now classifies according to the specification of this data set even though the rule seed was a noisy training example.

To quantify the classification quality of rules, we cannot use a minimal distance criterion to apply rules, because we have not yet determined the final rule set. Therefore given a rule R, we must only consider training examples lying inside rule frontiers. Hence, we could have applied the so-called Laplace correction, Lap, to the proportion of successful classifications, as in the CN2[24] version described in Ref. 25. This heuristic is given by the quotient of the number of successes ($\#success$) plus one, and the number of applicable training examples ($n = \#success + \#failures$) plus the number of classes ($\#classes$); in symbols,

$$\textsc{Lap}(R) = \frac{\#success + 1}{\#success + \#failures + \#classes}. \tag{18}$$

This measure faithfully reflects the consistency of a rule, but in our case we are in a nearest-neighbor environment concerned with discovering densely populated regions of examples of the same class within the attribute space. So, we need to add more requirements to our heuristic: it should also include completeness. It may be argued that Domingos used the Laplace correction in RISE,[11] but this algorithm does not necessitate worrying about completeness, because RISE starts with the whole training set as the rule set to be improved (in consistency) if possible.

Hence, in order to quantify the classification quality of rules, we use the impurity level,[4,24] a measure that tries to combine the consistency and the completeness of a rule. It was originally presented with a machine learning system, FAN, designed in our laboratory, and then used in Refs. 9 and 15–17. It is inspired by Aha's criterion for selecting suitable paradigmatic exemplars in IB3.[5,6]

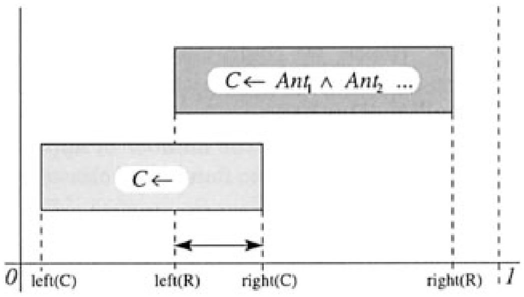

Figure 4. Impurity level. In boxes, we represent confidence intervals of successful classification of rules $C \leftarrow Ant_1 \wedge Ant_2 \cdots$ and the random rule of class C ($C \leftarrow$). The overlapping region of both intervals measures the impurity level of the first rule.

To formulate the impurity level, we first need to compute the confidence interval of success probability of a rule ($p = \#success/n$) by means of Ref. 27 (p. 162):

$$\textsc{ConfidenceInterval}(p, n, z) = \frac{p + \dfrac{z^2}{2n} \pm z\sqrt{\dfrac{p(1-p)}{n} + \dfrac{z^2}{4n^2}}}{1 + \dfrac{z^2}{n}}, \tag{19}$$

where z, according to the confidence level, α (by default 95%), can be found in a normal distribution table.

Finally, given a rule R concluding class C, we first compute the confidence interval of the rule: [left(R), right(R)]. Then we compute this interval for the so-called random rule of a class C; i.e., the rule concluding C without conditions: [left(C), right(C)]. Thus, we define

$$\textsc{ImpurityLevel}(R, Examples) = 100 \cdot \frac{\text{right}(C) - \text{left}(R)}{\text{right}(R) - \text{left}(R)}. \tag{20}$$

In other words, we are measuring the percentage of the confidence interval of the rule classification success that is overlapped by the confidence interval of the random rule of a class (see Fig. 4). So, a negative impurity level means that the confidence of a successful classification is better than the unconditional classification in the same class.

The last function to be applied to a rule set generated by inflating a set of training example seeds is PruneConditions (see Fig. 3). Here each rule description is optimized by deleting those conditions that increase the impurity level of the whole rule.[4,26,28,29]

5. RULE SETS GENERALIZATION

Once we have enough inflated training examples to become rules, Inner starts a pruning process to optimize the whole rule set thus obtained. The overall

```
Function PostProcess (RuleSet, Examples) : Set Of Rules
RuleSet=GeneralizeRuleSet (RuleSet, Examples, without Intersections)
RuleSet=Selection (RuleSet, Examples)
RuleSet=GeneralizeRuleSet (RuleSet, Examples, with Intersections)
if Coverage (RuleSet, Examples) $\neq$ Examples then // Still some uncovered examples
   RuleSet=GeneralizeInstances (RuleSet, Examples)
end if
return (RuleSet)

Function GeneralizeRuleSet (RuleSet, Examples, Intersect?) : Set Of Rules
for each class C do
   let DistancesList be the list of distances of all near pairs of RuleSet
         elements ordered by increasing distance and below a given
         threshold ($S_s(threshold) = 0.95$, see Equation (17)).
   for each [Rp, Rq; Distance] $\in$ DistancesList do
      if Extend (Rp, Rq, Examples, Intersect?) then
         if $Rq \subset Rp$ then RuleSet=RuleSet - {Rq} end if
         DistancesList=Update (DistancesList)
         /* Notice that updating distance lists implies that extended
         rules can be considered for further extensions */
      end if
   end for
end for
return (RuleSet)

Function Extend (Rp, Rq, Examples, Intersect?) : Boolean Value
/* Returns true if Rp can be extended in the direction of Rq, in
which case Rp is modified; otherwise this function returns false */
Extensible=ExtensibleAttributes (Rp, Rq)
if Extensible=$\emptyset$ then return (False)
else/* there are attributes of Rp extensible towards Rq */
   Body=True
   for each antecedent A of Rp do
      if $A \in$ Extensible then Body=Body $\wedge$ A extended towards Rq
      else Body=Body $\wedge$ A
      end if
   end for
   if in the extension rules are allowed to Intersect? OR
         'ClassOf (Rp)$\leftarrow$Body' does not intersect rules of other classes then
      if ImpurityLevel ('ClassOf (Rp)$\leftarrow$Body', Examples)$\leq$
            ImpurityLevel (Rp, Examples) then
         Rp='ClassOf (Rp)$\leftarrow$Body'
         return (True)
      end if
   end if
   return (False)
end if
```

Figure 5. INNER's postprocessing procedure. Inflated training examples are extended trying to cover the body of near rules of the same class. Redundant or ineffective rules are then deleted, and some final touches complete this rule generalization mechanism.

algorithm is described in Fig. 5. The main operation is the extension of one rule toward another of the same class, taking for granted that they are near; the resulting rule improves the classification quality measured by means of the impurity level. The idea is sketched graphically in Fig. 6.

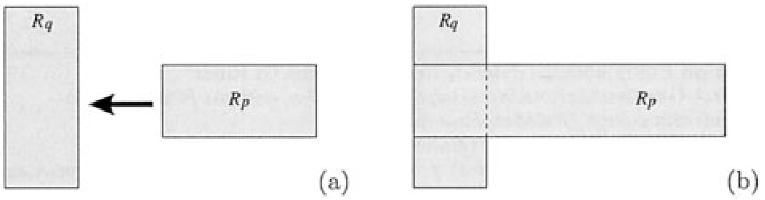

Figure 6. Extension of rules. R_p and R_q are near rules of the same class. In the figure, R_p is extended towards R_q in the direction of the horizontal attribute. This process will be applied whenever the classification quality of the result is improved.

During the inflating cycles, INNER has built a number of draft rules that should be generalized by merging their classification domains. This happens when two close seeds of the same class are fighting to gain classification space in the same iteration or when in different cycles we obtain several versions that are only slightly different from one another (see Fig. 7).

The procedure for making rule extensions can work under two conditions. In the first stage, we do not allow extensions of a rule that would produce new intersections with rules of different classes; we only accept this possibility in the final phase, which implies the definition of application priorities.

To explain the details of rule extensions, let us recall that the formulas used to measure distance between rules are generalizations of $\mathcal{H}$-HEOM and can be found in Appendix A.

But distances cannot be the sole condition to be taken into account for assessing the convenience of a prudent extension of a rule. For instance, Fig. 8 shows three quite different situations of rules at the same distance. Only the leftmost situation should yield an extension; in the other two cases, extensions would be too different from the original rules and so could easily become overgeneralizations. In fact, given two near rules, we will only extend one rule toward another in the direction of some attributes, i.e., dimensions of the attribute

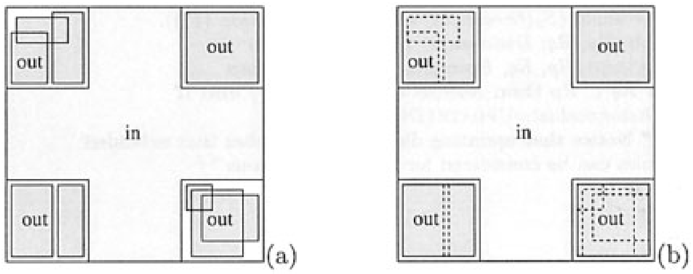

Figure 7. The effect of GENERALIZERULESET. To illustrate the work carried out by this function, let us consider a data set of points of a square classified as *in* or *out* depending on whether they are inside a cross or outside its limits. The lefthand side shows the rules concluding class out after some training examples were inflated. The function GENERALIZERULESET will transform the situation to obtain 4 rules, as depicted on the righthand side of the figure.

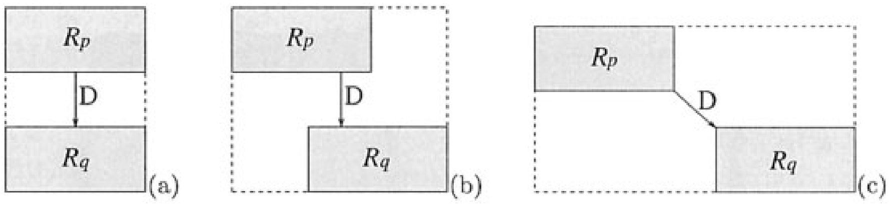

Figure 8. The distance between rules is not enough to determine which rules should be extended. Only the leftmost situation seems adequate to try an extension. In the other cases, the extended rules would be an overgeneralization.

space. To determine the extensible attributes of a rule in the vicinity of another, we use the degree of inclusion of attribute values defined as follows.

Given the rules

$$\begin{aligned} R_p: &\qquad C \leftarrow \cdots \wedge a \in (R_p)a \cdots \\ R_q: &\qquad C \leftarrow \cdots \wedge a \in (R_q)a \cdots, \end{aligned}$$

we are going to define the degree of inclusion Γ or $(R_p)_a$ in $(R_q)_a$. The idea is to measure to what extent $(R_p)_a$ is included in $(R_q)_a$; hence, this is not at all a symmetric measure. On the lefthand side of Fig. 3, the vertical attribute exhibits a complete degree of inclusion of its values in R_p inside those in R_q, but this is not the case of R_q and R_p. Moreover, there is no coincidence of values between these rules in the horizontal attribute. The result is that we will recommend the extension of R_p toward R_q in the direction of the horizontal attribute so as to obtain the situation depicted on the righthand side of Fig. 6. Therefore, if a is a continuous attribute, $(R_p)_a = [x_1^p, x_2^p]$, and $(R_q)_a = [x_1^q, x_2^q]$, we define

$$\Gamma((R_p)_a, (R_q)_a) = \begin{cases} \dfrac{\min\{x_2^p, x_2^q\} - \max\{x_1^p, x_1^q\}}{x_2^p - x_1^q}, & \text{if } (R_p)_a \cap (R_q)_a \neq \varnothing \\ 0, & \text{if } (R_p)_a \cap (R_q)_a = \varnothing. \end{cases} \tag{21}$$

For symbolic attributes we use the difference tables, but first we reduce their values to a number in [0, 1]. To do so, we trim the differences below -0.7 and above 2.0 and then normalize the original values. The thresholds -0.7 and 2.0 were determined experimentally as typical bounds for differences. In symbols, if T_a^p and T_a^q are their respective difference tables, we define

$$\Gamma((R_p)_a, (R_q)_a) = \frac{\sum_{e_a} \mathscr{G}(T_a^p[e_a]) \cdot \mathscr{G}(T_a^q[e_a])}{\sum_{e_a} \mathscr{G}(T_a^p[e_a])}, \tag{22}$$

where

$$\mathscr{G}(x) = \begin{cases} 1, & \text{if } x \leqslant -0.7 \\ 0, & \text{if } x \geqslant 2 \\ \dfrac{2 - x}{2 - (-0.7)}, & \text{otherwise} \end{cases} \tag{23}$$

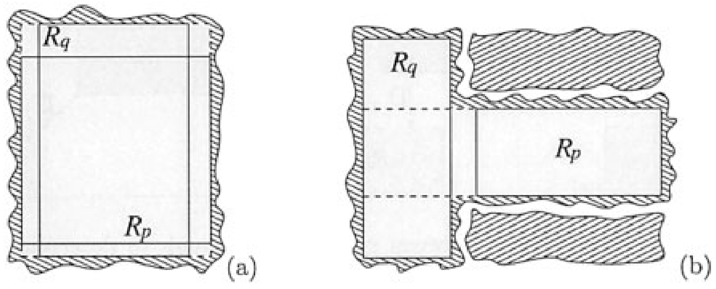

Figure 9. Selection of extensible attributes. The lefthand side shows a typical situation where rules should be merged in all possible directions: all the attributes are extensible. The righthand side shows the case where there is only one attribute; the horizontal should be used to extend R_p towards R_q.

In general, extensible attributes can be either all the attributes of a rule (they all have inclusion degrees higher than 0.9) or only one (the only one that has an inclusion degree below 0.9); see Fig. 9. However, the extended rules can be extended again starting from their new shape.

The key to extending a rule towards another is the extension of one antecedent A toward a rule R. The idea is to compute a kind of union between the set of values of A and the corresponding values in R. Thus, in the numeric case, the extension is the smallest interval containing the union of two intervals. In the symbolic case, we must act on the difference tables of the corresponding attributes: we build the union table with the minimum difference for each individual value.

Once an initial extension of the whole rule set has been carried out as in the previous section with the function PruneConditions, there is a kind of counterpart in the function Selection. It deletes redundant rules from the final rule set as well as those that cause more misclassifications than their absence. Thus, rules are sorted by their impurity level, and we then sequentially consider removing those with a higher impurity level than a certain threshold; the number of classification failures in training examples decides the best rule set.[4,26]

As a part of the final selection process, when all the attributes are symbolic, we make an additional simplification of the rule set. Notice that now, when we have a case at a greater distance than zero from all available rules, the evaluation procedure assigns the class of a rule following a legal but unnatural algorithm. To avoid this, we include a default rule; i.e., a rule to be applied to those cases not explicitly covered by any other rule. This is a rule with no conditions and the lowest priority. Then the minimum distance criterion to apply rules is only formally present: all cases will have some rule at distance zero. Additionally, the rules of the default class are unnecessary if they have no intersection with rules of other classes or have a lower priority than these. This is hence a powerful simplification mechanism in the number of final rules.

Notice that when numeric attributes are present, a default rule would destroy the essence of distance evaluation. In fact, cases at a positive distance from all rules

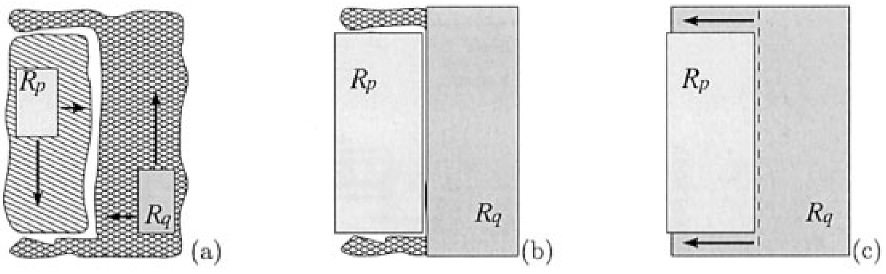

Figure 10. The final inflation of rules allows intersections of rules of different classes. In drawing (a) we see a data set where two rules are being inflated, one for each class; the result being shown in (b). Only when we allow intersections can R_q go below R_p to reach all the examples of its class (c).

would be classified according to the geometrically nearest rule. There is not a privileged class, as happens in the symbolic case. Thus, data sets with some numeric attribute will never include a default rule.

The last two steps in the post process of INNER to produce a final rule set are a new generalization of the whole set of rules and a new inflating if there are still some uncovered examples. However, there is a difference between previous generalization and inflating processes: in these final steps we allow intersections between rules of different classes. The possibility of overlapping rules has been a matter of controversy. In Ref. 3, the authors conclude that a major source of problems in NGE[2] is the creation of overlapping rectangles. We think that these problems may be caused by the peculiar criterion used to resolve priorities. However, INNER's position in this sense is quite different; its algorithm for deciding priorities is more similar to the policy followed in RISE,[11] given that both are based on measurements of the classification quality of the rules. To handle overlapping rules, INNER explicitly assigns relative application priorities to the rules involved. The criterion for defining priorities is based on the impurity level of the intersection.

In the case of the final inflating of rules, the algorithm used is the one previously described in Fig. 3. However, in order to regularize the modified rules, we take into account the effective impurity level. Given a rule R, its effective impurity level is computed as described in Equation (20) but considering only the examples covered by R and not covered by any other rule with higher priority than R. In this way, our rules can reach regions impossible to cover without intersections due to the strict application of the impurity level; see Fig. 10.

Finally, Fig. 11 presents the transformations described in this section for INNER rules induced from a very well-known data set, namely iris flowers, whose examples are described by four numeric attributes: petal and sepal length and width. They are classified into three classes: setosa, virginica, and versicolor. These drawings show only petal attributes, because it is known that these measurements are the most relevant ones for classifying irises. The first snapshot (a) shows the rule set generated by INNER after inflating some iris training examples. Snapshot (b) represents our rule set after the action of PRUNECONDITIONS (see Section 4), the first

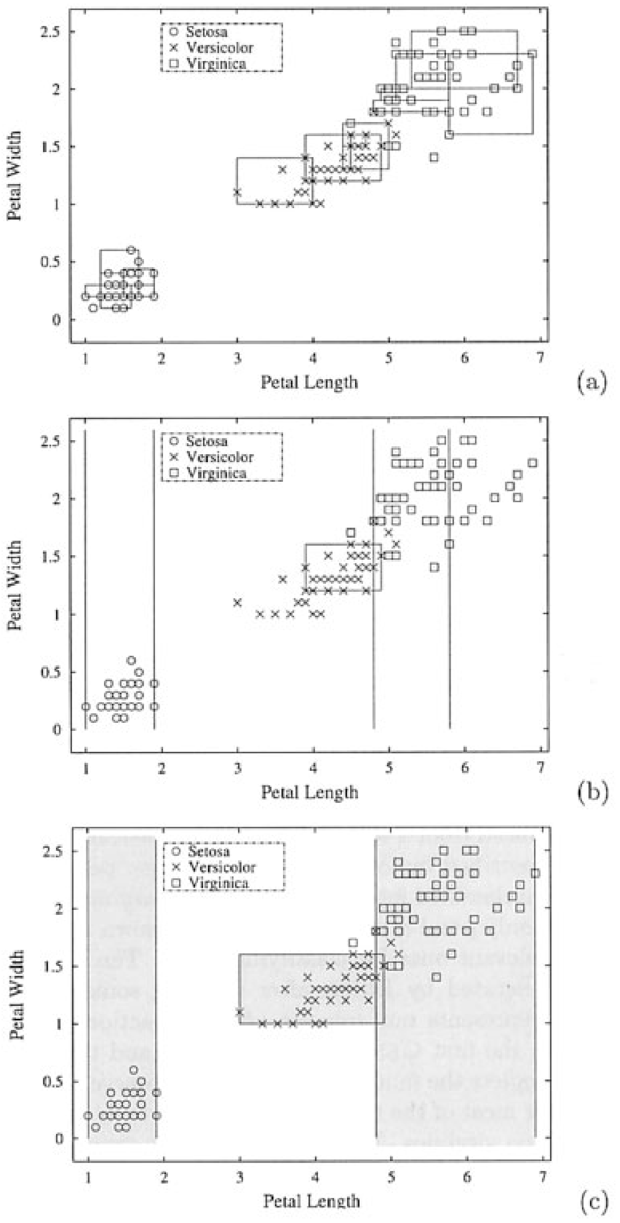

Figure 11. Iris data set. Three snapshots of rule generation. The final rules, represented in (c), are (1) setosa ← Petal Length ∈ [1.0, 1.9]; (2) versicolor ← Petal Length ∈ [3.0, 4.9] ∧ Petal Width ∈ [1.0, 1.6]; and (3) virginica ← Petal Length ∈ [4.8, 6.9].

GeneralizeRuleSet, and the Selection. The last snapshot (c), depicts the final rule set. We can observe how our rules have been inflated to cover most of the training examples of their classes, especially for iris versicolor and iris virginica. Worthy of note is the concise and accurate final set of rules obtained; only three rules are needed, one for each class.

6. EVALUATION OF THE SYSTEM

In this section we present a detailed study of the results obtained with INNER. We shall discuss the accuracy and size of the knowledge induced as well as its usefulness, a subtle term that depends not only on the accuracy achieved but also on the kind or quality of the solution induced. This could be thought of as a weighted combination of accuracy and complexity of the solution. The weightings can be different depending on the problem. Thus, for some problems a less accurate but simpler solution might be preferable to one that is more accurate but humanly unreadable. We will emphasize the ability of learning solutions to endow classifications with a sound explanation of the reasons that led us to conclude a given class.

With respect to the kind of solution, symbolic rules are in general more readable than other representations, but we have to differentiate two main approaches when using symbolic rules to classify:

- Applying a rule whenever an example fulfills the rule antecedents. This approach implies that rules must cover the whole attribute space in order to classify all the examples and unseen cases.
- Applying the closest rule to a given example. The metric used to measure distances is of crucial importance to correctly find the appropriate rule. Rules do not have to cover the whole attribute space.

Although the former approach is a particular case of the latter, where the distance is always 0 from any example to one or more rules, the difference of these approaches is very important when trying to explain a classification decision, as we will discuss later.

Throughout this section we show estimations in different senses of the quality of INNER's solutions, comparing them with those found using two other systems, RISE and C4.5 in its rule generation version (C4.5R); both using their default parameter settings. We used these two systems mainly for the following reasons:

- Both are rule induction systems, like INNER.
- C4.5R exemplifies the paradigm of algorithms producing rule sets that cover the whole attribute space and has a very well-established reputation in machine learning, so we consider it to be an obligatory reference for comparison.
- The core idea used in INNER to induce rules is similar to that used in RISE. In fact, we also unify, in the way stated in Ref. 11, the use of instances and rules. Both INNER and RISE apply their rules using a minimum distance criterion. In addition, RISE has proven itself to be a very accurate algorithm, so its precision is a good reference for any new system.

The first two subsections report the empirical study carried out with benchmark datasets. We then discuss the features of INNER's solutions in order to provide explanations of the classifications. In another subsection, some variants of the original algorithm are studied to clarify how the final results are concerned with the most striking aspects of INNER: the use of $\mathcal{H}$-HEOM, the coverage threshold, and the initial instances selection. Finally, we examine the theoretical time complexity of our system, accompanied by a table of the average running time consumed by the compared algorithms during the empirical study detailed in Section 6.1.

6.1. Experimental Results with Holte's and Monk's Problems

When selecting some datasets for validating a machine learning algorithm, an author may (unconsciously) select suitable problems to exploit some features of his or her algorithm; this selection would probably be inadequate due to its lack of generality. To avoid this pathology, we have used Holte's problems, a well known set of 16 datasets used in Ref. 18 to compare the results of his 1R system versus C4.5 and to justify the fact that the accuracy of systems tested against these datasets will be similar in real-world problems, because their number and diversity indicates that they represent a class of problems that often arises. Because INNER employs an original method to deal with symbolic attributes, we carried out some additional experiments with another well-known set of problems, the Monk's problems.[30] Discussion of the results obtained in these problems will permit us to justify some of the benefits provided by $\mathcal{H}$-HEOM, the metric used by INNER. All datasets used in our experiments were downloaded from the machine learning repository[20] hosted by the University of California at Irvine (UCI).

Experimental results on Holte's problems were obtained by performing a stratified cross-validation with 10 folds. This means that every dataset was randomly divided in 10 partitions, maintaining the same proportion of classes as in the original dataset. Every partition is used as a test set for one experiment in which the system is trained with the other 9 partitions, giving 10 (different) results; this process was repeated 5 times, yielding 50 train/test experiments for each dataset.

In order to avoid undesirable effects due to randomness in the partitioning of each dataset and to make a fair comparison, all the experiments were carried out using tools from MLC++.[31] This library allowed us to obtain exactly the same folds for every system whenever we used the same initial random seed in every cross-validation. Because INNER uses a random generator for some operations, it was also always initialized with a fixed seed. Additionally, all the experiments were carried out on the same machine.

Table IV shows the average error $\pm$ sample standard deviation obtained in the experiments for each system under comparison. In general, INNER's accuracy is similar to that of C4.5R and slightly lower than RISE's.

However, Table V shows a notable difference between INNER and the other systems in the size of the induced knowledge. This table contains the average number of rules and rule conditions obtained by each system in the domains studied.

Domingos[11] reports that RISE can considerably reduce the size of its outputs if rules that do not classify any training examples are discarded, whereas the decrease in accuracy is minimized; in all tables we refer to this release by RISE(p). However, even using this pruning method, the size of the resulting rule sets is still

Table IV. Average error ± standard error in 5 times 10-fold stratified cross-validation experiments.

Domains	C4.5R	RISE	RISE(p)	INNER
BC	29.64 ± 1.10	26.70 ± 0.90	27.25 ± 0.87	26.57 ± 0.90
CH	0.96 ± 0.08	1.95 ± 0.11	2.02 ± 0.12	6.30 ± 0.30
G2	22.72 ± 1.34	20.66 ± 1.43	21.13 ± 1.47	23.53 ± 1.38
GL	30.73 ± 1.27	27.28 ± 1.31	28.21 ± 1.41	34.49 ± 1.22
HD	23.34 ± 1.10	18.92 ± 0.98	18.79 ± 1.02	17.68 ± 0.85
HE	19.41 ± 1.17	21.00 ± 1.48	21.26 ± 1.44	19.88 ± 1.22
HO	16.30 ± 0.78	14.94 ± 0.90	15.42 ± 0.92	15.92 ± 0.92
HY	0.91 ± 0.06	1.84 ± 0.11	1.84 ± 0.10	3.58 ± 0.19
IR	4.93 ± 0.73	4.41 ± 0.75	4.54 ± 0.72	4.40 ± 0.62
LA	14.60 ± 1.92	9.60 ± 2.05	9.54 ± 1.84	11.27 ± 1.55
LY	22.96 ± 1.39	18.23 ± 1.28	18.09 ± 1.40	24.09 ± 1.65
MU	0.03 ± 0.02	0.00 ± 0.00	0.00 ± 0.00	1.47 ± 0.06
SE	2.36 ± 0.11	3.58 ± 0.12	3.74 ± 0.12	7.27 ± 0.25
SO	2.90 ± 1.03	0.00 ± 0.00	0.00 ± 0.00	2.10 ± 0.90
V1	10.34 ± 0.60	11.22 ± 0.53	10.76 ± 0.54	9.88 ± 0.57
VO	4.46 ± 0.45	4.79 ± 0.39	4.88 ± 0.40	4.78 ± 0.43
Average	12.91	11.57	11.72	13.33

much larger than the size of INNER's rule sets, as can be appreciated in the corresponding column of Table V.

The second block of comparisons used the Monk's problems,[19] three datasets that were specially devised to evaluate the performance of different machine learning techniques when they are acting on training examples described by symbolically valued attributes.

These datasets are composed of examples whose attributes represent some characteristics of a robot, for example, the color of its jacket, if it is smiling or not, etc. A robot can be good or bad depending on the relation of attributes and values defined for every problem.

Because Monk's problems have well defined train/test sets, a cross-validation is not appropriate, so we made a simple experiment with C4.5R and RISE. However, taking into account the fact that INNER's results depend on the order of examples presented, we carried out 50 different experiments by giving different initial random seeds to our system. The average results together with the results of C4.5R, RISE and RISE(p) are shown in Tables VI (accuracy) and VII (size of the solutions).

We added this set of problems to the comparative study to show that $\mathcal{H}$-HEOM handles symbolic attributes satisfactorily. There are several datasets in Holte's problems with only symbolic attributes, but the main difference with the Monk's problems is that the correct solution is known in the latter. This knowledge of the domain allows us to notice, for instance, that INNER correctly solves Monk's 1 in the 50 different experiments, yielding the rules that exactly represent the relation to be learned.

In contrast with $\mathcal{H}$-HEOM, the metric used by RISE, SVDM, encounters some problems in solving Monk's 1, a fairly easy problem for most of the rule induction

Table V. Average size of the induced knowledge.

	C4.5R				RISE(p)				INNER			
	Rules		Cond.		Rules		Cond.		Rules		Cond.	
Domains	Av.	St. Dv.	Av.	St. Dv.	Av.	St. Dv.	Av.	St. Dv.	Av.	St. Dv.	Av.	St. Dv.
BC	7.84	2.56	16.18	6.40	53.00	7.33	300.74	50.21	9.20	4.13	23.34	13.10
CH	22.90	4.92	86.70	19.51	95.90	5.02	2649.34	165.64	4.74	1.21	9.48	4.98
G2	8.48	1.58	22.08	4.71	31.24	3.80	281.16	34.23	12.40	2.16	32.62	5.88
GL	12.30	2.01	42.34	7.32	40.62	3.99	365.58	35.95	18.62	3.65	53.12	9.99
HD	11.00	1.97	31.48	6.60	50.94	4.98	518.38	53.60	6.62	2.10	13.98	4.40
HE	5.70	1.35	13.60	4.16	27.54	3.19	308.00	43.10	5.42	1.81	10.16	3.98
HO	9.10	2.14	20.32	6.80	78.20	7.79	746.04	98.46	4.96	1.09	6.68	1.92
HY	8.36	2.06	19.24	6.52	161.68	18.71	3247.30	408.57	1.54	0.61	2.52	1.53
IR	4.12	0.32	6.30	1.22	11.00	2.77	44.00	11.09	4.08	0.89	6.20	1.89
LA	3.92	0.74	6.24	1.24	14.06	3.78	107.84	45.43	5.64	0.93	7.18	1.61
LY	10.00	1.28	23.10	2.92	30.64	5.20	365.50	85.47	7.40	1.57	13.20	3.98
MU	16.48	1.90	25.02	3.56	14.98	0.14	160.00	1.60	4.06	0.24	3.12	0.47
SE	13.92	2.16	43.70	8.27	232.48	17.77	5088.82	422.03	2.70	1.28	6.38	2.98
SO	4.00	0.00	5.24	0.65	4.00	0.00	44.40	0.53	4.90	0.30	4.90	0.30
V1	10.70	1.33	28.06	3.81	50.38	6.01	404.46	71.26	5.56	2.13	14.60	6.53
VO	6.04	0.69	13.64	2.46	35.34	5.19	236.84	44.36	3.06	1.10	6.10	3.47
Average	9.68		25.20		58.25		929.28		6.31		13.35	

The data under RISE(p) correspond to the size of the solutions induced by RISE with the pruning suggested by Domingos,[11] which reduces the sizes to 30% of the original, approximately. Even so, INNER's results are much smaller. Notice that dataset MU, on average, gives rise to 4.06 rules with a total of 3.12 conditions. This means that the default rule, which has zero conditions, is usually present.

algorithms. The relation to be learned in Monk's 1 is jacket_color = RED $\vee$ (head_shape = body_shape). The difficulty for SVDM lies in capturing the equality of shape values, because the probability distribution is the same for every possible value in both attributes. This peculiarity puzzles SVDM when it is searching for the nearest adequate rule to a given example, probably the most important task to learn and classify in distance-based algorithms.

In order to carry out an in-depth analysis of these peculiarities from the point of view of $\mathcal{H}$-HEOM and SVDM, we modified the original Monk's 1 problem. The first modification was the elimination of jacket_color = RED from the defined

Table VI. Error $\pm$ theoretical standard deviation for the Monk's problems.

Domains	C4.5R	RISE	RISE(p)	INNER
Monk's 1	0.00 ± 0.00	13.40 ± 1.64	17.10 ± 1.81	0.00 ± 0.00
Monk's 2	31.94 ± 2.25	30.60 ± 2.22	31.20 ± 2.23	31.53 ± 2.24
Monk's 3	3.70 ± 0.91	7.20 ± 1.25	5.60 ± 1.11	2.34 ± 0.65
Average	11.88	17.07	17.97	11.29

Values for INNER are the average of 50 different experiments varying the initial random seed.

Table VII. Size of the induced knowledge for the Monk's problems.

					INNER			
	C4.5R		RISE(p)		Rules		Cond.	
Domains	Rules	Cond.	Rules	Cond.	Av.	St. Dv.	Av.	St. Dv.
Monk's 1	13.00	25.00	45.00	216.00	5.00	0.00	7.00	0.00
Monk's 2	11.00	27.00	44.00	187.00	26.90	2.71	92.34	11.52
Monk's 3	13.00	25.00	33.00	134.00	3.26	0.44	2.58	1.00
Average	12.33	25.67	40.67	179.00	11.72		33.97	

These results were obtained with a simple train/test experiment, as defined for every problem, except for INNER, the results of which were obtained as the average of 50 experiments with different random seeds. The data under RISE(p) correspond to the size of the solutions induced by RISE with the pruning suggested by Domingos,[11] which reduces the sizes to 30% of the original, approximately.

relation. This simplification does not help SVDM. Instead, the new relation to be learned depends only on the problematic part of the original relation. We made this modification on the original test set that contains all possible (432) combinations. Thus, the experimental method was a typical cross-validation of 10 folds repeated 5 times. The scores obtained with the new problem showed an increase in the error made by RISE from 7.63% up to 11.13%, while INNER's error remained at 0%.

Realizing that when we increase the number of possible values for one attribute we are modifying the probability distribution of its values, we added 5 new values to attribute head_shape. A new cross-validation obtained an error of 1.34% with RISE; INNER's error again remaining at 0%. The last modification was the addition of 5 new values to body_shape, but not coincident with the values added to head_shape; in this case RISE's error decreased to 0.37%, whereas INNER's still remained at 0%.

Returning to the standard datasets, Monk's 2 can be considered a difficult problem because the relation to be learned is quite complex to represent using the rule syntax of these systems. The relation defined in Monk's 3 is similar to that in Monk's 1 but has a 5% of noisy examples, which adds an extra difficulty to the learning task. This difficulty seems to be harder for SVDM than for $\mathcal{H}$-HEOM, according to the scores in Table VI.

To complete the comparative study of this section, we carried out a number of statistical tests to measure the significance of the differences between the means of accuracy and size obtained by the algorithms in the datasets used. Table VIII presents the results of the one-tail t and Wilcoxon tests. There is no significant difference in accuracy between C4.5R and INNER, and the difference is only very low with respect to the RISE versions if we are using the Wilcoxon test. However, the differences in size of the induced knowledge, the number of rules, and the number of conditions are statistically significant in favor of INNER.

Table VIII. Significance tests.

	C4.5R vs. INNER			RISE vs. INNER			RISE(p) vs. INNER		
	Acc.	Rules	Cond.	Acc.	Rules	Cond.	Acc.	Rules	Cond.
t-test	ns	94.5	91.3	ns	99.2	97.3	ns	99.9	99
Wilcoxon	ns	96.5	98.2	90.8	100	100	90.8	100	100

Confidence levels at which the difference with INNER's scores are significant using one-tail t and Wilcoxon tests; values below 90% are considered nonsignificant and marked ns.

6.2. Explanations

As we pointed out at the beginning of this section we consider that the possibility of building a useful explanations device is a consequence of inducing a small set of compact and sound classification rules. INNER offers a kind of solutions suitable to this purpose due to a number of reasons that we shall discuss in what follows.

First of all, the size of INNER's solutions (the number of rules and its conditions) is significantly smaller than those found by other systems, and it is easier to understand a classification mechanism if it can be characterized by a few rules.

On the other hand, the existence of a rule in a certain region of the attribute space depends primarily on the existence of an important mass of examples belonging to the same class. When INNER uses a rule to classify a new case, an explanation follows implicitly from the rule itself: a case belongs to class C because it is in (or close to) a region where training examples usually belong to that class C. Moreover, the region is defined by the antecedents or conditions of the rule and here again, INNER requires fewer conditions to define a good rule, which makes the validation of the classification easier for a human user.

One may argue that a *k-NN* algorithm can explain its classifications the same way, but it cannot show the region that the example belongs to, because *k-NN* does not handle regions, just representative exemplars. This difference can be very important when offering a convincing explanation to a user, as can be appreciated in the following hypothetic problem.

Let us suppose that a kind of industrial processes can be described by two attributes, the integer values of parameters X and Y, and the class, indicating whether the results are GOOD or BAD, where a GOOD result will be obtained if $X \in [55, 100]$ and $Y \in [0, 30]$ and a BAD result will be obtained if $X \leqslant 40$ or if $X \geqslant 55$ and $Y \geqslant 60$.

We built a training set of randomly generated examples according to the preceding specifications, but adding some noise by changing the class values in 4% of the examples. The classification rules generated by C4.5R and INNER are given in Table IX and depicted graphically in Fig. 12 together with the training examples.

Given a case with $X = 61$ units and $Y = 29$ units, a previously trained *k-NN* algorithm could say something like "this process will produce good results since there are k processes in the surroundings of this one which are mostly GOOD." In the same situation, INNER could explain "this process will produce good results

Table IX. Rules for a hypothetical problem.

C4.5 rules	INNER rules
$X > 40 \wedge Y \leq 30 \rightarrow$ GOOD	$X \in [55, 100] \wedge Y \in [0, 30] \rightarrow$ GOOD
$X \leq 40 \wedge Y \leq 30 \rightarrow$ BAD	$X \in [0, 40] \rightarrow$ BAD
$Y > 30 \rightarrow$ BAD	$Y \in [60, 100]$ BAD

From C4.5R, covering the whole attribute space, and from INNER, covering just relevant regions. Although both kind of rules can achieve the same accuracy, INNER's rules describe the regions characterized for belonging to a class better.

since processes with values of parameter X within 55 and 100 units and values of parameter Y within 0 and 30 units produce good results." Notice that INNER's rule ($X \in [55, 100] \wedge Y \in [0, 30] \rightarrow$ GOOD) explicitly shows the conditions required to obtain good results.

Another salient aspect of the solutions provided by our algorithm is that rules are not induced to cover the whole attribute space, INNER just tries to cover relevant and well situated regions. This characteristic allows an assistant to indicate what to do to change a predicted undesirable result in order to get a desirable one. To illustrate this capability, let us assume that we want to predict the result of a running process whose parameters are $X = 38$ and $Y = 45$. C4.5R's rules will predict BAD given that the case fulfills the rule $Y > 30 \rightarrow$ BAD. INNER will predict BAD too, because the case is classified by $X \in [0, 40] \rightarrow$ BAD.

Now, if we want to know what to change in this situation to get a GOOD result, we need to know how the values of X and Y must change to reach a region of GOOD examples. If our explanation assistant uses C4.5R's rules it will recommend varying the value of X and Y (acting on a pressure valve, etc.) till they become $X = 41$ and $Y = 30$, thus satisfying the conditions to be a GOOD example. However, this seems to be a poor recommendation, since the new situation is clearly more similar to BAD examples than to GOOD ones [see Fig. 12(a)]. Advice based on INNER's rules would be more appropriate, because this would recommend changing X and Y to the new values $X = 55$ and $Y = 30$, if possible, as shown in Fig. 12(b).

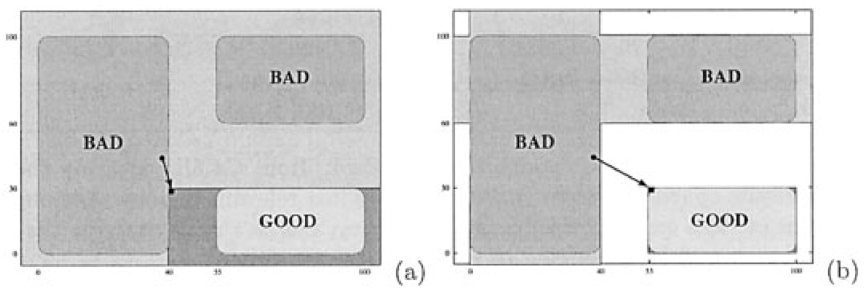

Figure 12. An hypothetic problem to show the explanation ability of INNER's rules against C4.5-like rules covering the whole attribute space. INNER's rules offer a more plausible explanation about the changes needed to vary the predicted class of an example.

The appealing qualities of partial matching for providing useful advice decrease as the number of rules increases. So in this training set, for instance, RISE produces 101 rules, 31 if the pruning process described in Ref. 11 is used, which is much more than the 3 rules needed by C4.5R and INNER. In addition, RISE suffers overfitting because it obtains an error of 3.1%, whereas there is 4% of noisy examples, the error made by C4.5R and INNER.

6.3. Some Variants of INNER

There are some points in our algorithm that can may lead the reader to suspect they have a patent influence on the global behavior of the system. The most important ones are the following:

- The use of difference tables to adapt distances of symbolic values solely during the induction stage, but not in that of classification, when the training has finished.
- The random selection of initial instances to be inflated.
- The coverage threshold used as the stopping criterion in the main loop.

To clarify the importance of these elements, we made some modifications to the original algorithm, giving rise to some variants that were used to conduct a comparative study with the standard INNER.

The first modification is related to the role played by difference tables, giving rise to a variant called $\mathcal{H}$-INNER. In fact, the only difference with respect to the original system is that this release maintains the difference tables obtained during the induction process and uses them to classify. Obviously, this involves the use of $\mathcal{H}$-HEOM instead of HEOM when classifying. Therefore, with this version we check whether keeping the selforganized differences in the classification, using some kind of fuzzy symbolic antecedents instead of crisp ones, would improve accuracy. The results obtained confirm that there is just a slight improvement in some domains with symbolic attributes and, obviously, there is no difference at all in domains described solely by continuous attributes. In turn, induced knowledge is not as explicit as in the original system because the user obtains rules plus the difference tables attached to every symbolic antecedent.

The second modification consists of providing well-situated initial instances to be inflated instead of the random selection mechanism. So, we used a system called BETS[9] that it is capable of selecting paradigmatic training examples from a dataset. This variant, called $\mathcal{B}$-INNER, achieved almost the same scores as the original INNER, which allowed us to conclude that the initial selection of instances is not of crucial importance. The reason is that the iterative mechanism counteracts the lack of precision in discovering good initial instances, and random selection is not as computationally expensive as the use of more sophisticated methods. However, $\mathcal{B}$-INNER usually reaches high coverage values in fewer cycles than INNER.

We conducted a third trial to study how important the coverage threshold is. As stated previously, this parameter stops the main loop when 95% (default value) of training examples of every class have been covered. Taking into account the fact that we are looking for explicit rules covering as many cases as possible, this parameter must obviously have a rather high value. Our experience indicates that the coverage threshold should have values of around 90–100%, but it is not highly dependent on an exact value. In general, smaller values make accuracy decreases notably, as expected.

6.4. Complexity and Running Time

In this subsection we study the time consumption of our system. We will proceed in two steps starting with the estimation order of the theoretical time complexity in the improbable worst case. We then report a table with CPU times used during the experiments discussed in this section.

Let e be the number of training examples, a the number of attributes, and c the number of classes. The algorithm followed by INNER (see Fig. 1) has a while loop that is repeated at most 5 times. The relevant function iterated here is FINDBESTRULES, which has a complexity of the order $\mathcal{O}(c \cdot (e + e \cdot a + e^2 \cdot a^2 + a^2))$.

After the loop, the rule set generated is modified by POSTPROCESS, whose complexity (see Fig. 5) is $\mathcal{O}(e \cdot a \cdot number\_of\_rules^2)$. Given that number_of_rules is bound by $5 \cdot 10 \cdot c \cdot a$, the whole system is $\mathcal{O}(c \cdot (e + e \cdot a + e^2 \cdot a^2 + a^2)) + \mathcal{O}(e \cdot a^3 \cdot c^2)$. If we consider the number of classes as constant, the theoretical time complexity of INNER is $\mathcal{O}(e^2 \cdot a^2) + \mathcal{O}(e \cdot a^3)$.

Let us recall that the total time complexity of RISE[11] is $\mathcal{O}(e^2 \cdot a^2)$ or $\mathcal{O}(e^3 \cdot a^2)$ depending on the stopping criterion used. On the other hand, the decision tree building process of C4.5 is $\mathcal{O}(e \cdot a^2)$ when the examples are described by symbolic attributes, but it is at least quadratic in e when continuous attributes are involved. In general, the final pruning stage is worst than quadratic in e.

Table X shows the average running time of INNER, RISE, RISE(p), and C4.5R for every dataset in the comparative study reported in this section. The values shown in this table are calculated by dividing the total CPU time needed to complete a full cross-validation by the number of single train/test experiments carried out: 50 (5 times a 10-fold cross-validation) for Holte's problems and only one for Monk's. All the experiments were carried out under the same conditions using a 200-MHz Intel Pentium Pro running Linux.

7. CONCLUSIONS AND FUTURE WORK

Simplicity in rule sets is usually upheld as a desirable quality of the induced knowledge synthesized by machine learning algorithms.[18] The algorithm presented in this article, INNER, returns concise rule sets, applicable by means of a minimum distance criterion, with high classificatory accuracy.

In this algorithm, each rule is a representative cluster of training examples of the same class. So, when a rule is used to classify an unseen case, a natural and solid explanation can back the suggested class. This quality is the main achievement of INNER.

Table X. Average running time during the experiments.

	C4.5R	RISE	RISE(p)	INNER
BC	0.96”	0.88”	0.84”	4.09”
CH	10.95”	4’ 38.02”	3’ 20.25”	1’ 42.49”
G2	0.55”	1.29”	1.27”	7.87”
GL	0.96”	1.62”	1.62”	10.46”
HD	0.90”	4.37”	4.35”	7.44”
HE	0.43”	2.24”	2.20”	9.01”
HO	1.26”	13.53”	13.44”	11.8”
HY	8.18”	1h 17’ 31.64”	1h 19’ 38.49”	2’ 43.73”
IR	0.12”	0.63”	0.63”	2.10”
LA	0.11”	0.30”	0.30”	6.70”
LY	0.57”	0.68”	0.67”	8.56”
MU	24.62”	6’ 53.52”	6’ 27.49”	4’ 12.62”
SE	13.88”	56’ 0.85”	55’ 58.28”	3’ 0.85”
SO	0.09”	0.13”	0.13”	2.79”
V1	1.14”	2.70”	2.69”	5.64”
VO	0.67”	2.82”	2.82”	5.05”
Monk’s 1	0.76”	0.84”	0.85”	2.40”
Monk’s 2	0.92”	0.90”	0.84”	5.70”
Monk’s 3	0.79”	0.80”	0.80”	1.86”
Average	3.57”	7’ 39.88”	7’ 40.95”	41.64”
w/o HY & SE	2.70”	42.66”	36.54”	26.27”

These numbers indicate the average time needed to complete every experiment in a 10-fold cross-validation, except those for Monk’s problems, which indicate the time needed to complete the train/test experiment defined for every problem.

The previous section showed the results of a number of experiments conducted to compare the performance of INNER with two state-of-the-art algorithms related to our system: C4.5R and RISE. These results let us conclude that INNER generally induces a smaller set of rules while maintaining a high level of accuracy, which reinforces the idea that simple solutions, i.e., small sets of small rules, can achieve the same (and sometimes even higher) accuracy as more complex ones. Moreover, given that users can more easily understand simpler solutions, the sets of rules induced by our system can be greatly improved with sound, clear explanations attached to their classifications. In this sense, we have included a subsection devoted to spelling out the advantages of using the kind of rules induced by INNER, illustrating how it may become an intelligent advisor by means of the cited explanations mechanism.

Possible future directions for research may be the adaptation of some of the advantages of Kohonen’s SOM philosophy to INNER’s generalization mechanism. We might, for instance, investigate how to extend the selforganizing principle, already used for dealing with symbolic values, to continuous attributes.

Acknowledgments

The authors thank Pedro Domingos, who supplied us with the source code of RISE, Ross Quinlan for publishing C4.5, and those who have contributed to maintaining and enlarging the

University of California at Irvine Machine Learning Repository. This research was funded in part by the Spanish Agency Dirección General de Estudios Superiores, under grants PB96-1457 and PB98-1556.

References

1. Cover TM, Hart PE. IEEE Trans Inf Theory 1967;13(1):21–27.
2. Salzberg S. Learning with nested generalized exemplars. Boston, MA: Kluwer Academic Publishers; 1990.
3. Wettschereck D, Dietterich T. Machine Learning 1995;19:5–27.
4. Ranilla J, Bahamonde A. Int J Hum Comput Studies 2002;56(4):445–474.
5. Aha D. A study of instance-based algorithms for supervised learning tasks: Mathematical, empirical, and psychological evaluations. PhD thesis, University of California at Irvine, 1990.
6. Aha DW, Kibler D, Albert M. Machine Learning 1991;6:37–66.
7. Cost S, Salzberg S. Machine Learning 1993;10:57–78.
8. Wilson DR, Martinez TR. Machine Learning 2000;38(3):257–286.
9. del Coz J, Luaces O, Quevedo J, Alonso J, Ranilla J, Bahamonde A. Self-organizing cases to find paradigms. In: Proc Int Work-Conf on Artificial and Natural Neural Networks (IWANN '99). Lecture Notes in Computer Science, Vol 1606. New York: Springer-Verlag; 1999. pp 527–536.
10. Quinlan JR. C4.5: Programs for Machine Learning. San Mateo, CA: Morgan Kaufmann Publishers; 1993.
11. Domingos P. Machine Learning 1996;24:141–168.
12. Breslow LA, Aha DW. Knowledge Eng Rev 1997;12(1):1–40.
13. Kohonen T. Self-organizing maps. Springer Series in Information Sciences. Berlin: Springer Verlag; 1995.
14. Bahamonde A, de la Cal E, Ranilla J, Alonso J. Self-organizing symbolic learned rules. In: Mira, C. Moreno-Diaz, editors. Biological and artificial computation: From neuroscience to technology. Lecture Notes in Computer Science, Vol 1240. New York: Springer-Verlag; 1997. pp 536–545.
15. Luaces O, Alonso J, de la Cal E, Ranilla J, Bahamonde A. Machine learning usefulness relies on accuracy and self-maintenance. In: Proc 11th Int Conf on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, Lecture Notes in Artificial Intelligence, Vol 1416. New York: Springer-Verlag; 1998. pp 448–457.
16. Luaces O, del Coz JJ, Quevedo JR, Alonso J, Ranilla J, Bahamonde A. Autonomous clustering for machine learning. In: Proc Int Work-Conf on Artificial and Natural Neural Networks (IWANN '99). Lecture Notes in Computer Science, Vol 1606. Alicante, Spain. New York: Springer-Verlag; 1999. pp 497–506.
17. Luaces O, Bahamonde A. Revista Iberoamericana de Inteligencia Artificial 2000;9:38–44.
18. Holte RC. Machine Learning 1993;11:63–91.
19. Thrun S, et al. The monk's problems—a performance comparison of different learning algorithm. Technical Report CS-CMU-91-197. Carnegie Mellon University, 1991.
20. Blake C, Merz C. UCI repository of machine learning databases. [http://www.ics.uci.edu/~mlearn/MLRepository.html]. University of California, Irvine, Department of Information and Computer Sciences, 1998.
21. Fürnkranz J. Artif Intell Rev 1999;13(1):3–54.
22. Wilson DR, Martínez TR. J Artif Intell Res 1997;6(1):1–34.
23. Stanfill C, Waltz D. Commun ACM 1986;29:1213–1228.
24. Clark P, Niblett T. Machine Learning 1989;3(4):261–283.

25. Clark P, Boswell R. Rule induction with CN2: some recent improvements. In: Proc 6th European Working Session on Learning. Porto, Portugal: Springer-Verlag; 1991. pp 151–163.
26. Ranilla J, Mones R, Bahamonde A. Revista Iberoamericana de Inteligencia Artificial 1998;4:4–11.
27. Spiegel MR. Estadística. Atlacomulco, México: McGraw-Hill; 1970.
28. Bahamonde A. Int J Comput Math 1994;54:127–142.
29. Botana F, Bahamonde A. Int J Hum Comput Studies 1995;42:137–155.
30. Wnek J, Sarma J, Wahab A, Michalski R. Comparison learning paradigms via diagrammatic visualization: A case study in single concept learning using symbolic, neural net and genetic algorithm methods. Technical Report, George Mason University, Computer Sciences Department, 1990.
31. Kohavi R, John G, Long R, Manley D. MLC++: A machine learning library in C++. In: Proc 6th Int Conf on Tools with Artificial Intelligence, IEEE Computer Society Press; 1994. pp 740–743.
32. Holte R, Acker L, Porter B. Concept learning and the problem of small disjuncts. In: Proc 11th Int Joint Conf on Artificial Intelligence (IJCAI-89). Detroit, MI, 1989. pp 813–818.

APPENDIX A: DISTANCE BETWEEN RULES

Rule set generalization is mainly a process that involves extension of some rules towards others, but not all the possible combinations are tried; INNER uses some criteria to restrict the possibilities to the most reasonable ones, as was explained in Section 5. One of these criteria, the first one to be satisfied, establishes that a rule should be close enough to another in order to attempt an extension toward it. Hence, a way to measure distances between rules is needed. We use a generalization of $\mathcal{H}$-HEOM called $\mathcal{R}$-HEOM; Equations (24) and (25) reflect how INNER computes the distance from R_p to R_q:

$$\mathcal{R}\text{-HEOM}(R_p, R_q) = \sqrt{\sum_{a=1}^{m} d_{\mathcal{R}}((R_p)_a, (R_q)_a)^2} \tag{24}$$

$$d_{\mathcal{R}}((R_p)_a, (R_q)_a)) = \begin{cases} 0, & \text{if } (R_q)_a \text{ is true} \\ D_{\mathcal{R}}((R_p)_a, (R_q)_a), & \text{if } e_a \text{ is symbolic} \\ E_{\mathcal{R}}((R_p)_a, (R_q)_a), & \text{if } e_a \text{ is continuous.} \end{cases} \tag{25}$$

In the symbolic case, if T_a^p and T_a^q are their respective difference tables:

$$D_{\mathcal{R}}((R_p)_a, (R_q)_a)) = \min_{e_a}\{\mathcal{F}(T_a^p[e_a])/\mathcal{F}(T_a^q[e_a]) = 0\}, \tag{26}$$

where

$$\mathcal{F}(x) = 1 - \mathcal{G}(x). \tag{27}$$

In the continuous case, if $(R_p)_a = [x_1^p, x_2^p]$, and $(R_q)_a = [x_1^q, x_2^q]$, we define

$$E_{\mathcal{R}}((R_p)_a, (R_q)_a)) = \begin{cases} 0, & \text{if } (R_p)_a \cap (R_q)_a \neq \emptyset \\ \dfrac{\min\{|x_1^p - x_2^q|, |x_2^p - x_1^q|\}}{\max_a - \min_a}, & \text{otherwise.} \end{cases} \tag{28}$$

Obviously, the distance between rules depends on the distance between their antecedents, which is calculated by means of d_R. It is quite simple to calculate the distance between two continuous antecedents: the normalized Euclidean distance between the nearest extremes of the intervals is used, giving 0 if they intersect.

However, the distance between symbolic antecedents deserves special attention. The process of extending a rule toward another can be seen as the effort that the rule should make to reach the other one. In this sense, we want to measure the minimum effort needed for a symbolic antecedent to reach its counterpart in the second rule. So we will take the minimum value from its difference table, looking only at the entries associated with symbols included in the antecedent of the second rule (i.e., those whose value is below the threshold -0.7). If we recall, values in the difference tables range from -1 to 2, so we have to re-map the value returned to range from 0 to 1, depending on how far the value is from the threshold of inclusion. The re-mapping is done by function $\mathcal{F}$ used in Equation (26).

In other words, if we want to compute the distance from a symbolic antecedent of R_p to its counterpart in R_q, the algorithm re-maps the difference tables of both antecedents using $\mathcal{F}$. Then it returns the minimum re-mapped value of the antecedent belonging to R_p, taking into account only those entries whose re-mapped value in the antecedent of R_q is 0.

This mechanism is not symmetric, because the effort needed to extend a rule toward another is not necessarily the same as the one needed to extend the second toward the first. In fact, what we are actually measuring is not the distance between rules: we are measuring the distance from the rules to some symbolic values and obviously, one rule may be closer than another to these values.

Additionally, d_R should consider the fact that some antecedents of rule R_p may not be present in R_q, no matter what type of antecedents these are. In this case, the distance returned is 0, because a missing antecedent in R_q is equivalent to an antecedent including all the possible values of an attribute and therefore intersects the antecedent of R_p.

APPENDIX B: INITIAL INSTANCES SELECTION

The instances generalization process starts from a set of uncovered examples taken from the training dataset. By default, 10 randomly selected instances per class are used, but the user can specify a different number, which will be constrained by Equation (29). In fact, 10 is the maximum number permitted; the algorithm may choose a lower number of instances to be generalized, depending on the number of examples in the smaller class ($\#sc$). Figure 13 represents Equation (29) graphically.

$$I_{\max} = \min(10, \max(1, \lfloor e^{\#sc/(50/\log 10)} \rfloor)) \tag{29}$$

The reason for limiting the initial number of instances to a maximum of $I_{\max}$ is related to the part of the main algorithm that drops noisy rules, explained in Section 5. The noise filter uses the number of examples of the smaller class to decide which rules seem to be noisy and should therefore be eliminated.

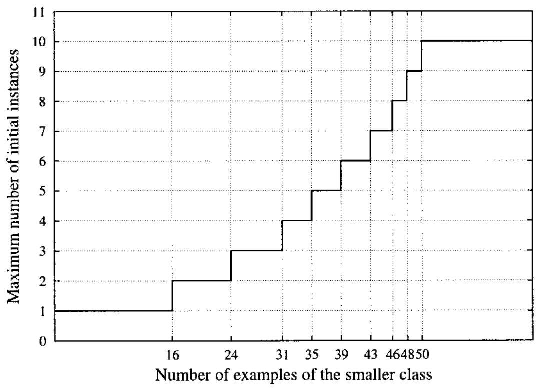

Figure 13. Maximum number of initial instances of every class to be generalized in each cycle.

Considering a uniform distribution of examples in the attribute space and selecting I instances for each class C, we hopefully get a set of generalized instances, i.e., rules, covering each of the N_C/I examples, where N_C is the number of examples belonging to class C. Obviously, as I increases, the number of covered examples per rule decreases and the noise filter may decide to eliminate most of the above mentioned rules, accuracy decreasing at the same time. In short, we are facing the small disjuncts problem already observed by Holte.[32] Taking into account the way that INNER generalizes rules, a large number of these would encumber one another, impeding adequate growth.