Transactions on Information and Communications Technologies vol 19, © 1997 WIT Press, www.witpress.com, ISSN 1743-3517

The RULES-4 incremental inductive learning algorithm

D.T. Pham & S.S. Dimov

Intelligent Systems Research Laboratory

Systems Engineering Division, School of Engineering

University of Wales Cardiff, Cardiff CF2 1XH, UK

EMail: PhamDT@cf.ac.uk, Dimov@cf.ac.uk

Abstract

This paper describes RULES-4, a new algorithm for incremental inductive learning from the "RULES" family of automatic rule extraction systems. This algorithm is the first incremental learning system in the family. It has a number of advantages over well known non-incremental schemes. It allows the stored knowledge to be updated and refined rapidly when new examples are available. The induction of rules for two bench mark pattern classification problems is employed to test the algorithm. The results obtained have shown that the accuracy of the extracted rule sets is commensurate with the accuracy of the rule set obtained using a non-incremental algorithm.

1 Introduction

Methods for inducing concept descriptions from examples can help to ease the knowledge acquisition "bottleneck" in the development of knowledge-based systems. Thus far, the majority of concept induction systems are non-incremental. That is they require all training examples from the beginning of the induction process. The advantages of using incremental knowledge acquisition systems, which are systems that accept examples as and when they

become available, are well described by a number of authors [1, 2, 3] and can be summarised as follows:

- the stored knowledge may be rapidly updated as each new example is encountered. In some systems, the updating is based only on the new example thus avoiding the need to retain past examples [1];
- the system can gradually refine the knowledge base from an initial set of user-supplied rules [2];
- the incremental approach is resource-constrained, that is an incremental algorithm will operate within a specified computational budget and will perform better if this budget is increased [3].

A number of incremental learning algorithms have been proposed. These include ID4 [1], ID5 [4] and AQ15 [3]. The first two algorithms are from the well known ID3 family [5] and the third is from the AQ family [6].

ID4 incrementally builds a decision tree. Instead of forming a decision tree from a set of examples, the algorithm updates a tree based on each individually observed example. The main ingredients of ID4 are a series of tables located at each node of the tree. The tables consist of entries for the values of all non-test attributes and the numbers of positive and negative examples for each value. If the information measure for a non-test attribute at a given node (which is a function of these numbers) is lower than that of the current test attribute, the algorithm removes the subtrees below that node and changes the test attribute. This change discards information acquired over previous examples and wastes training effort [4].

ID5 builds on the idea of maintaining the counts of positive and negative examples for each attribute that could become a test attribute [4]. The difference between ID5 and ID4 is in their methods of replacing the test attributes. Instead of discarding the subtrees below the old test attribute, ID5 reshapes the tree by adding new nodes or re-ordering the positions of existing nodes. The benefit of doing this is that the numbers of positive and negative examples can be recalculated during manipulation of the tree, without reprocessing the examples. A potential drawback of this algorithm is that the portion of each example description not explicit in the decision tree has to be stored at each leaf so that the re-ordering process is possible.

Like the AQ algorithm on which it is based [6**,** 7], AQ15 generates decision rules from a set of examples. When building a decision rule, AQ15 performs a heuristic search through a space of logical expressions to determine those that cover only positive examples and no negative examples. Then the preferred rule is selected from the set of complete and consistent expressions according to a rule preference criterion that is usually defined by the user to reflect the needs of the application domain. AQ15 performs incremental learning with perfect memory. In this type of learning, the system stores all examples that it has used to form rules as well as the rules formed. A feature of the algorithm is the possibility to start learning with initial rules supplied by the

user. A problem with AQ15 is its complexity which is probably the reason for its not being widely used.

This paper describes RULES-4 (RULe Extraction System - Version 4), a simple incremental learning algorithm belonging to the RULES family. Previous versions in the family [8, 9, 10, 11] are algorithms for nonincremental inductive learning. RULES-4 and its predecessors have been employed for extraction of classification rules for solving different manufacturing engineering problems, for example the recognition of design form features in CAD models for computer-aided process planning [12] and the mapping of manufacturing information to design features [12].

RULES-4 shares some common features with its immediate predecessor, RULES-3 Plus. For ease of referencing, the paper first summarises the rule forming procedure of RULES-3 Plus. It then describes the new algorithm. Finally, the paper presents the results of applying the algorithm to two bench mark classification problems.

2 Rule Forming Procedure

RULES-3 Plus extracts classification rules for a collection of objects, each belonging to one of a number of given classes. The objects constitute the set of examples from which the algorithm has to induce general rules. An object is described by a fixed set of attributes, each with its own set of possible values. An attribute-value pair constitutes a condition. If the number of attributes is n_a, a rule may contain between one and n_a conditions, each of which must be a different attribute-value pair as only conjunction of conditions is permitted. For each example considered, the rule forming procedure may require at most n_a passes, the first pass producing rules with one condition and the second pass resulting in rules with two conditions, etc. A set called the *"SET of Attributes and Values" (SETAV)* is constructed, the elements of which are attribute-value pairs associated with the example under consideration. The total number of elements in the set is equal to the number of attributes in the example. In the first iteration, each element of the set is examined to decide whether it can form a rule with that element as the condition. For the whole set of examples, if a given element applies to only one class, then it is a candidate for forming a rule. If it pertains to more than one class, it is added to a set called *Partial Rules SET (PRSET)*. The maximum number of expressions in PRSET is specified by the user and this parameter determines the number of alternatives that will be examined in the next pass when each expression in PRSET is specialised by appending to it a single condition from SETAV. The appended condition has to differ from the conditions already included in the expression. In this way, the rule forming procedure carries out a "general-to-specific" pruned search [13, 14].

Step 1. Take an unclassified example and form array SETAV.
Step 2. Initialise arrays PRSET and T_PRSET (PRSET and T_PRSET will consist of m_{PRSET} expressions with null conditions and zero H measures) and set $n_{CO} = 0$.
Step 3. IF $n_{CO} < n_a$
THEN $n_{CO} = n_{CO} + 1$ and set m = 0;
ELSE the example itself is taken as a rule and STOP.
Step 4. DO
m = m + 1;
Specialise expression m in PRSET by appending to it a condition from SETAV that differs from the conditions already included in the expression;
Compute the H measure for the expression;
IF its H measure is higher than the H measure of any expression in T_PRSET
THEN replace the expression having the lowest H measure with the newly formed expression;
ELSE discard the new expression;
WHILE m <= m_{PRSET} .
Step 5. IF there are consistent expressions in T_PRSET
THEN choose as a rule the expression that has the highest H measure and discard the others;
ELSE copy T_PRSET into PRSET;
initialise T_PRSET and go to Step 4.

n_{CO} - number of conditions; n_a - number of attributes;
m_{PRSET} - number of expressions stored in PRSET (m_{PRSET} is user-provided);
T_PRSET - a temporary array of partial rules of the same dimension as PRSET.

Figure 1: Rule forming procedure of RULES-3 Plus and RULES-4

To assess the information content of the expressions during the rule forming process, RULES-3 Plus uses a metric called the *H measure* [15] which, for a given expression, is defined as:

$$H = \sqrt{\frac{E^c}{E}}\left[2 - 2\sqrt{\frac{E_i^c}{E^c}\frac{E_i}{E}} - 2\sqrt{\left(1-\frac{E_i^c}{E^c}\right)\left(1-\frac{E_i}{E}\right)}\right] \qquad (1)$$

where E^c is the number of examples covered by the expression, E is the total number of examples, E_i^c is the number of examples covered by the expression and belonging to the target class *i*, and E_i is the number of instances in the training set belonging to the target class *i*. In equation (1), the first term

$$G = \sqrt{\frac{E^c}{E}} \tag{2}$$

pertains to the generality of the rule and the second term

$$A = 2 - 2\sqrt{\frac{E_i^c}{E^c}\frac{E_i}{E}} - 2\sqrt{\left(1 - \frac{E_i^c}{E^c}\right)\left(1 - \frac{E_i}{E}\right)} \tag{3}$$

relates to its accuracy [15]. During the rule forming process, the expressions in PRSET are ordered according to their H measures. If the number of rules extracted in a given iteration is greater than one, then only the rule that has the highest H measure is retained.

When using the rule set formed by RULES-3 Plus to classify a new example, there are three possible outcomes. They are:

(i) only one rule covers the new example and the example is correctly classified by that rule;

(ii) more than one rule covers the example, in which case, the rule with the largest H measure will be used to classify the example;

(iii) no rules can classify the example or the example is misclassified. In the latter case, the example should be added to the training set and the induction process re-initiated.

RULES-3 Plus can deal with attributes having numerical values by quantising them. The ranges of values of these attributes and number of quantisation levels for each range are specified by the user.

Figure 1 presents a summary of the rule forming procedure of RULES-3 Plus. This procedure is also employed in RULES-4.

3 Induction Procedure of RULES-4

RULES-4 is summarised in Figure 2. The algorithm extracts rules incrementally by processing one example at a time.

The input information needed to start the induction process includes:

(a) A set of examples examined in some of the previous iterations and retained in a short-term memory (STM). The size of the memory has to be predefined by the user. Initially, the STM is empty.

(b) The set of rules extracted so far and stored in a long-term memory (LTM). At the beginning, the LTM can be empty or can contain initial rules defined by the user. Apart from the rules themselves, statistical information is stored regarding the numbers of examples correctly classified and misclassified by each rule in the LTM. Using this information, a decision is made as to which rules to prune from the LTM.

(c) Ranges of values for the numerical attributes. As with RULES-3 Plus, the new algorithms can also deal with numerical attributes by quantising them. The ranges of values for these attributes are updated before processing every new example. The number of quantisation levels for each range is specified by the user. When a new example is processed, RULES-4 forms a

corresponding description in which the values of all numerical attributes are represented by appropriate quantisation levels. Induction is then carried out with the new description, the quantisation levels being treated as any other values.

Input: Short-Term Memory (STM), Long-Term Memory (LTM), ranges of values of the numerical attributes, frequency distribution of examples among classes, a prespecified Noise Level (NL), one new example.
Output: Short-Term Memory (STM), Long-Term Memory (LTM), updated ranges of values of the numerical attributes, updated frequency distribution of examples among classes.

Step 1. Update the frequency distribution of examples among classes.
Step 2. Test whether the values of the numerical attributes are within their existing ranges and if not update the ranges.
Step 3. Test whether there are rules in the LTM that classify or misclassify the new example and simultaneously update their accuracy measures (A measures) and H measures. Also, the noise threshold T for each rule in LTM is updated.
Step 4. Prune the LTM by removing the rules for which the A measure is lower than their noise threshold T.
Step 5. IF the number of examples in the STM is less than a prespecified limit
THEN add the example to the STM
ELSE
IF there are no rules in the LTM that classify the new example
THEN replace an example from the STM that belongs to the class with the largest number of representatives in the STM by the new example.
Step 6. For each example in the STM that is not covered by the LTM, form a new rule by applying the rule forming procedure summarised in Figure 1 or of RULES-3 Plus. Add the rule to the LTM. Repeat this step till there are no examples uncovered by the LTM.

Figure 2: Incremental induction procedure in RULES-4

(d) A prespecified Noise Level (NL). The *NL* parameter specified by the user allows the extracted rules to handle a degree of inconsistency in the data and thus makes the rule set produced more robust.

Noise is considered inconsistencies or non-systematic errors in attribute values or class information. As most data obtained from real domains contain noise, an incremental inductive learning system that cannot handle noise is of limited value. In particular, learning algorithms that are to handle numerical

attributes must be able to cope with some noise in the values of attributes or class information.

(e) The number of examples found in each class so far.

The algorithm comprises six main steps.

In Step 1, the number of examples in the class to which the new example belongs is increased by one.

In Step 2, the value of each numerical attribute is tested to determine if it falls in the range formed so far and if not, a new range is formed for that attribute.

In Step 3, the LTM is inspected for rules that correctly classify or misclassify the new example. Then the A and H measures for each rule are updated using equations (3) and (1), respectively. Also, in this step, a noise threshold for each rules in the LTM is updated. The noise threshold T is defined as

$$T = 2 - 2\sqrt{(1-NL)\frac{E_i}{E}} - 2\sqrt{NL\left(1-\frac{E_i}{E}\right)} \qquad (4)$$

where NL is a prespecified noise level. Notice that equation (4) is obtained from equation (3) by replacing the ratio $\frac{E_i^c}{E^c}$ with $1-NL$. When the accuracy measure A for a rule falls below the threshold T, the rule is removed from the LTM.

In Step 4, the LTM is examined for rules where the A measure is below the noise threshold T.

The STM management strategy implemented in RULES-4 is simple. At the beginning all new examples are stored in the STM till it is full. When the STM is full, one of the examples belonging to the class with the largest number of representatives in the STM is replaced by the new example.

In Step 6, for each example in the STM that is not covered by the rules in the LTM, a new rule is formed by applying the rule forming procedure described in section 2. The newly formed rule is then added to the LTM.

4 Experimental Results

Experiments were performed on two data set: IRIS and Tic-Tac_Toe endgame data sets (obtained from Machine Learning repository in California University, USA). These data sets are commonly used to bench mark machine learning and pattern classification algorithms. The results are discussed below.

4.1 Iris Flower Data

The IRIS data set was used to compare RULES-4 and RULES-3 Plus. The data set contains 3 classes (Setosa, Versicolor and Virginica) of 50 examples each.

A class is described in terms of four numerical attributes: Sepal-Length (SL), Sepal-Width (SW), Petal-Length (PL) and Petal-Width (PW).

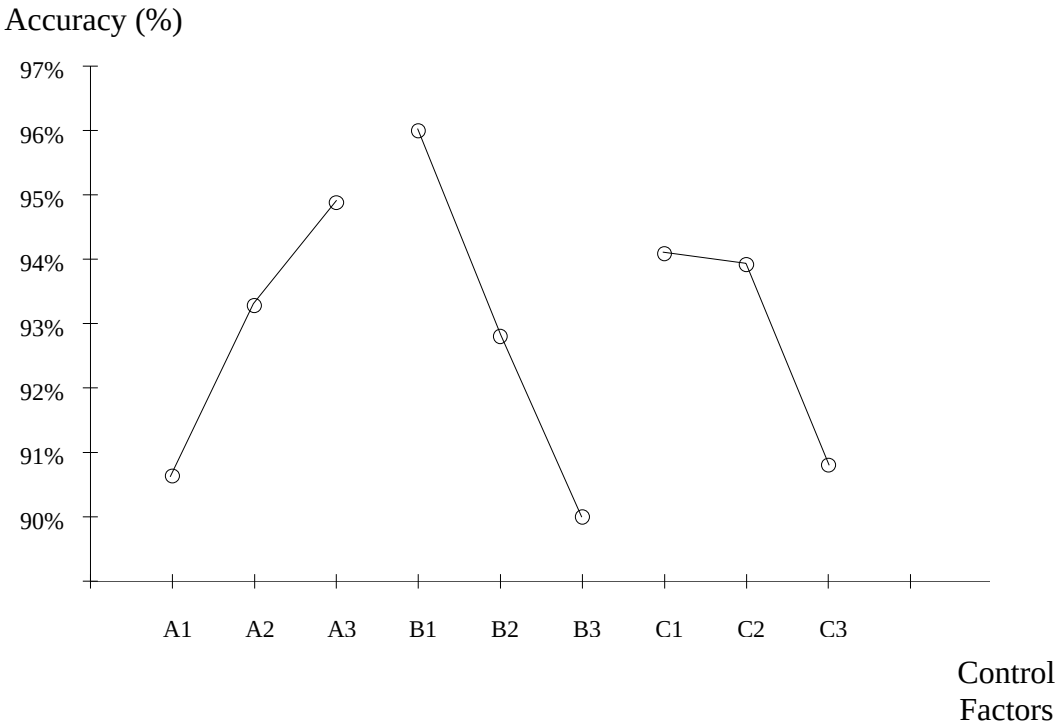
Table 1. Experiment design and results for RULES-4.
(C.: correctly classified examples; MisC.: misclassified examples)

Control Factors			**Training**		**Test**		**Number of**	**Accuracy**
STM_SIZE	INT	NL	C.	MisC.	C.	MisC.	**Rules**	**(%)**
3	6	20%	97	3	47	3	10	96%
			96	2	48	0	10	96%
			92	5	47	2	9	92.67%
3	7	10%	91	4	48	2	12	92.67%
			93	2	46	1	12	92.67%
			93	1	45	3	12	92%
3	8	0%	90	5	43	2	14	88.67%
			81	8	38	6	12	79.33%
			92	4	43	4	14	90%
9	6	10%	97	2	49	1	12	97.33%
			98	2	49	1	14	98%
			95	2	48	1	11	95.33%
9	7	0%	91	7	47	2	17	92%
			88	6	46	2	13	89.33%
			92	4	47	1	14	92.67%
9	8	20%	96	4	48	2	16	96%
			96	4	46	2	13	94.67%
			89	3	41	2	12	86.67%
15	6	0%	98	2	49	1	16	98%
			96	1	49	1	16	96.67%
			96	1	46	1	17	94.67%
15	7	20%	96	2	47	1	14	95.33%
			97	2	48	1	15	96.67%
			95	1	47	1	13	94.67%
15	8	10%	94	5	44	3	17	92%
			96	2	45	2	18	94%
			94	4	46	1	14	93.33%
Average accuracy of the extracted rules:								93.23%

A Taguchi experimental design procedure was conducted to determine the values of the following control parameters which were required to initialise RULES-4:

- maximum number of examples that can be stored in the STM (STM-SIZE);
- number of quantisation levels (INT) for numerical attributes;
- prespecified noise level (NL).

An L9 orthogonal array was adopted in the Taguchi experiment [16]. Three levels were used for each control factor as follows: STM_SIZE {3, 9, 15}; INT {6, 7, 8}; NL {20%, 10%, 0%}. 100 examples were chosen at random for rule extraction and 50 for testing the extracted rules. The results of the different tests in the experiment (classification accuracy and number of rules) are given in Table 1. Figure 3 plots the responses to each of the control factors. Three tests were conducted for every combination of factors and the results obtained averaged. The reason for this was the different rule accuracy in each test due to the random choice of example to be replaced in the STM in Step 5. The graphs in Figure 3 show that, within the range of control factor levels considered, the best result would be obtained with the following combination of factors for RULES-4: STM-SIZE = 15; INT = 6; NL = 20%.



Factor A (STM_SIZE): A1 (3), A2 (9), A3 (15);
Factor B (INT): B1 (6), B2 (7), B3 (8);
Factor C (NL): C1 (0%), C2 (10%), C3 (20%)

Figure 3. Response graphs of the control factors

The rule set produced by RULES-4 using the above factor levels is given in Table 2. The classification accuracy achieved when applying this rule set to the test data is presented in Table 3. In the test, the first 70 examples were chosen for rule extraction and the remaining 80 for testing the extracted rules. The results show that 98.75% accuracy was obtained. Experiments conducted with STM-SIZE equal to 30 and 45 showed that increasing the size of the memory did not improve the performance of the algorithm.

Table 2. Rule set obtained by RULES-4 (IRIS data set; STM-SIZE = 15; INT = 6; A = 20%)

Rule 1:	[1.3<=PL<2.08333] ⇒ Iris_Setosa	H=0.454018
Rule 2:	[1.35<=PW<1.73333] ⇒ Iris_Versicolor	H=0.142753
Rule 3:	[5.21667<=PL<6] ⇒ Iris_Virginica	H=0.322622
Rule 4:	[2.3<=SW<2.5] ⇒ Iris_Versicolor	H=0.144281
Rule 5:	[0.966667<=PW<1.35] ⇒ Iris_Versicolor	H=0.322622
Rule 6:	[1.73333<=PW<2.11667] ⇒ Iris_Virginica	H=0.204505
Rule 7:	[5.71667<=PL<6.6] ⇒ Iris_Virginica	H=0.249902
Rule 8:	[1.1<=PL<2.01667] ⇒ Iris_Setosa	H=0.370704
Rule 9:	[2.1<=PW<=2.5] ⇒ Iris_Virginica	H=0.249902
Rule 10:	[1<=PL<1.98333] ⇒ Iris_Setosa	H=0.242683

Table 3. Classification accuracies for IRIS data problem

Algorithm	No. of training exam.	No. of test exam.	No. of rules	No. of unclass. examples*	No. of misclass. examples*	Accuracy (%)*
RULES-3 Plus	70	80	10	0 (0)	2 (2)	97.5 (98.67)
RULES-4	70	80	10	0 (1)	1 (4)	98.75 (96.67)

* results for the complete set of examples (training and test examples) are given in brackets.

The accuracy of the classification rules extracted by RULES-3 Plus for the same training and test data is also given in Table 3 and the rule set in Table 4. It can be seen that accuracies are very similar for RULES-3 Plus and RULES-4.

Note the compactness of the rule sets obtained by both of the RULES algorithms with only ten rules in each case. To reach the same accuracy, which

has been regarded as the best achievable with the IRIS data set, 25 rules were required by [15] using a different inductive algorithm.

Table 4. Rule set obtained by RULES-3 Plus (IRIS data set; INT = 6)

Rule	Rule description	H-measure
1	[1<=PL<1.98333] ⇒Iris_Setosa	0.485366
2	[3.95<=PL<4.93333] [1.3<PW<1.7] ⇒ Iris_Versicolor	0.395129
3	[2.1<=PW<=2.5] ⇒ Iris_Virginica	0.338369
4	[4.93333<=PL<5.91667] ⇒ Iris_Virginica	0.381731
5	[2.4<=SW<2.8] [1.7<=PW<2.1] ⇒ Iris_Virginica	0.228128
6	[2.96667<=PL<3.95] ⇒ Iris_Versicolor	0.228128
7	[5.91667<=PL<=6.9] ⇒ Iris_Virginica	0.269925
8	[0.9<=PW<1.3] ⇒ Iris_Versicolor	0.228128
9	[3.2<=SW<3.6] [3.95<=PL<4.93333] ⇒ Iris_Versicolor	0.204044
10	[2.8<=SW<3.2] [1.7<=PW<2.1] ⇒ Iris_Virginica	0.228128

4.2 Tic-Tac-Toe Endgame Data

This data set was employed to compare RULES-4 against other algorithms tested by Aha [17], in particular NewID, CN2, MBRtalk, IB1, IB3 and IB3-CI. The data encodes the complete set of possible board configurations at the end of the tic-tac-toe game. The learning problem is a recognition task with two possible outcomes, a win for either player X or O. The target concepts are configurations in which one of the two players has three consecutive squares vertically, horizontally or diagonally. The data set contains 958 examples of which 65.3% describe winning configurations for player X. Nine attributes are used in this game domain to describe board configurations. Each attribute corresponds to one tic-tac-toe square. The possible values of these attributes are: X (player X has taken a square), O (player O has taken a square) and B (blank square).

Table 5. Classification accuracy for Tic-Tac-Toe endgame data problem

Algorithm	Accuracy (%)*
NewID	84.0
CN2	98.1
MBRtalk	88.4
IB1	98.1
IB3	82.0
IB3-CI	99.1
RULES-3 Plus	99.5
RULES-4	98.0

Table 6. Rule set obtained by RULES-4 (Tic-Tac-Toe endgame data set; STM-SIZE = 66; A = 8%)

Rule	Rule description	H-meas.
1	[top_left_square=x] [top_middle_square=x] [top_right_square=x] ⇒ X	2.46065
2	[top_left_square=o] [middle_left_square=o] [bottom_left_square=o] ⇒ O	4.23301
3	[top_right_square=o] [middle_right_square=o] [bottom_right_square=o] ⇒ O	4.13997
4	[top_middle_square=o] [middle_middle_square=o] [bottom_middle_square=o] ⇒ O	4.41322
5	[bottom_left_square=o] [bottom_middle_square=o] [bottom_right_square=o] ⇒ O	4.41322
6	[top_left_square=o] [top_middle_square=o] [top_right_square=o] ⇒ O	3.9473
7	[top_right_square=x] [middle_middle_square=x] [middle_right_square=o] [bottom_left_square=x] ⇒ X	1.93029
8	[top_left_square=x] [middle_left_square=x] [bottom_left_square=x] ⇒ X	2.50752
9	[top_right_square=x] [middle_middle_square=x] [bottom_left_square=x] ⇒ X	2.6651
10	[middle_left_square=o] [middle_middle_square=o] [middle_right_square=o] ⇒ O	4.13997
11	[middle_left_square=x] [middle_middle_square=x] [middle_right_square=x] ⇒ X	2.4842
12	[top_left_square=o] [middle_middle_square=o] [bottom_right_square=o] ⇒ O	4.83444
13	[bottom_left_square=x] [bottom_middle_square=x] [bottom_right_square=x] ⇒ X	2.28905
14	[top_middle_square=x] [middle_middle_square=x] [bottom_middle_square=x] ⇒ X	2.33936
15	[top_left_square=x] [middle_middle_square=x] [bottom_right_square=x] ⇒ X	2.38862
16	[top_right_square=x] [middle_right_square=x] [bottom_right_square=x] ⇒ X	2.2376
17	[top_right_square=o] [middle_middle_square=o] [bottom_left_square=o] ⇒ O	3.84736

For all algorithms, 70% of the examples were selected for training and the remaining 30% for testing. Each test was repeated 10 times with randomly chosen training examples. The final results were the average of the 10

experiments (Table 5). The control parameters used for RULES-4 are STM-SIZE = 66 (10% of the training set size) and NL = 8%. The rule set extracted by RULES-4 contains between 17 and 24 rules for different tests. The best rule set extracted is given in Table 6. To reach a similar accuracy, more than 128 rules were required by RULES-3 Plus. This proves again that the rule sets extracted by RULES-4 are more compact. The compactness of the rule set can be attributed to the good generalisation capability of the algorithm.

5. Conclusion

This paper has presented RULES-4, a new algorithm for incremental inductive learning. Although only two data classification tasks were discussed, the algorithm has been applied to different problems as mentioned in the introduction section. The results obtained have shown that in all cases the accuracy of the extracted rule sets is commensurate with that of the rule set obtained using nonincremental algorithm. As expected, in general, RULES-4 requires more examples, especially in problems involving numerical attributes, to converge on a stable rule set, and may sacrifice the optimality of the rules.

Further work is needed to improve the performance of RULES-4. First, new strategies have to be sought for managing the STM that would allow a better representation of the problem domain in the memory. Second, an approach requires to be developed for adaptive determination of the STM size.

Acknowledgement

The research described in this paper was carried out within two EC INCO-COPERNICUS research projects: N 960231 “Intelligent Product Manuals” and N 964438 “Handling of Non-rigid Materials with Robots”.

Keywords: Rule induction, Expert systems, Knowledge acquisition

References

1. Schlimmer, J.C. and Fisher, D. A case study of incremental concept induction. Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, PA:, Morgan Kaufmann, San Mateo, CA, 1986, pp. 496-501.
2. Michalski, R.S., Mozetic, I., Hong, J. and Lavrac, N. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, PA:, Morgan Kaufmann, San Mateo, CA, 1986, pp. 1041-1045.
3. Quinlan, J.R. C4.5: programs for machine learning, 1993 (Morgan Kaufmann, San Mateo, CA).

4. Utgoff, P.E. ID5: An incremental ID3. Proceedings of the Fifth International Conference on Machine Learning, The Univ. of Michigan, Ann Arbor, MI, 1988, pp. 107-120.
5. Quinlan, J.R. Learning efficient classification procedures and their application to chess end games. Machine Learning An Artificial Intelligence Approach, (Eds R.S. Michalski, J.G. Carbonell and T.M. Mitchell), Tiago, Palo Alto, CA, 1983, pp. 463-482.
6. Michalski, R.S. On the quasi-minimal solution of the general covering problem. Proceedings of the Fifth International Symposium on Information Processing (FCIP 69), Vol. A3, Bled, Yugoslavia, 1969, pp. 125-128.
7. Michalski, R.S. and McCormick, B.H. Interval Generalisation of Switching Theory. Report No. 442, 1971 (Dept. of Computer Science, Univ. of Illinois, Urbana).
8. Pham, D.T. and Aksoy, M.S. An algorithm for automatic rule induction. Artificial Intelligence in Engineering, 1993, 8, 277-282.
9. Pham, D.T. and Aksoy, M.S. RULES: A simple rule extraction system. Expert Systems With Applications, 1995, 8(1), 59-65.
10. Pham, D.T. and Aksoy, M.S. A new algorithm for inductive learning. J. Systems Eng., 1995, 5(2), 115-122.
11. Pham, D.T. and Dimov, S.S. The RULES-3 Plus inductive learning algorithm. Proceedings of the 3rd World Congress on *Expert Systems*, Seoul, Korea, 1996, 2, pp. 917-924.
12. Pham, D. T. and Dimov, S.S. Intelligent support systems for concurrent product and process design. In Integrated Product, Process and Enterprise Design, (Ed.: B. Wang), 1997, (Chapman & Hall, London), (in press).
13. Clark, P. and Niblett, T. The CN2 induction algorithm. Machine Learning 3, 1989, pp. 261-283 (Kluwer Academic, The Netherlands).
14. Michalski, R.S. A theory and methodology of inductive learning. Reading in Machine Learning (Eds: J.W. Shavlik and T.G. Dietterich), 1990, pp. 70-95 (Morgan Kaufman, San Mateo, CA).
15. Lee, C. Generating classification rules from databases. Proceedings of the Ninth Conference on Application of AI in Eng., Pennsylvania, USA, 1994, pp. 205-212.
16. Roy, R.K. A primer on the Taguchi method. 1990 (Van Nostrand Reinhold, New York).
17. Aha, D. W Incremental constructive induction. Proceedings of the Eighth International Workshop on Machine Learning, 1991, pp. 117-121 (Morgan Kaufman, San Mateo, CA).