ACKNOWLEDGMENT

The authors would like to thank the Editor, the two referees, and Dr. S. Kaneko and Dr. L. Anthony for providing many useful comments which have led to improvement in this paper.

REFERENCES

[1] R. Lippmann, "Pattern classification using neural networks," *IEEE Commun. Mag.*, pp. 47–64, Nov. 1989.
[2] M. Obaidat, "Artificial neural networks to systems, man and cybernetics: Characteristics, structures, and applications," *IEEE Trans. Syst., Man, Cybern. B*, vol. 28, pp. 489–495, Aug. 1998.
[3] L. Y. Cai and H. K. Kwan, "Fuzzy classifications using fuzzy inference networks," *IEEE Trans. Syst., Man, Cybern. B*, vol. 28, pp. 334–347, June 1998.
[4] L. Chen, D. Cooley, and J. Zhang, "Possibility-based fuzzy neural networks and their application to image processing," *IEEE Trans. Syst., Man, Cybern. B*, vol. 29, pp. 334–347, Feb. 1999.
[5] S. V. Aleshin, *Recognition of Dynamical Object* (in Russian). Moscow, Russia: Moscow Univ. Press, 1996.
[6] I. Chegis and S. Yablonsky, "Logical methods for controlling electric circuit function" (in Russian), in *Proc. V.A. Steklov Inst. Math.*, vol. 51, 1958.
[7] Y. Zhuravlov, A. Dmitriev, and F. Krendelev, "Mathematical principles of the classification of objects and scene" (in Russian), *Discrete Analyse*, vol. 7, 1966.
[8] E. Djukova, "On asymptotically optimal terminal test detection algorithm" (in Russian), *Dokl. Akad. Nauk SSSR*, vol. 233, no. 4, pp. 527–530, 1977.
[9] A. Andreev, "On terminal and minimal tests" (in Russian), *Dokl. Akad. Nauk SSSR*, vol. 256, no. 3, pp. 521–524, 1981.
[10] A. Kibkalo, "T-algorithms that use short tests," Ph.D. dissertation (in Russian), Moscow State Univ., Moscow, Russia, 1988.
[11] F. Blayo *et al.*, "Enhanced learning for evolutive neural architecture," ESPRIT Basic Res. Project no. 6891, 1995.
[12] K. Woods, W. Kegelmeyer, and K. Bowyer, "Combination of multiple classifiers using local accuracy estimates," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, pp. 405–410, Apr. 1997.
[13] D. Wilson and T. Martinez, "Reduction techniques for instance-based learning algorithms," *Mach. Learn.*, vol. 38, pp. 257–286, Nov. 2000.
[14] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. New York: Academic, 1990.
[15] J. Sammon, "A nonlinear mapping for data structure analysis," *IEEE Trans. Comput.*, vol. C-18, pp. 401–409, May 1969.
[16] T. Kohonen, "The self organizing map," *Proc. IEEE*, vol. 78, pp. 1464–1480, 1990.
[17] J. Mao and A. Jain, "Artificial neural networks for feature extraction and multivariate data projection," *IEEE Trans. Neural Netw.*, vol. 6, pp. 296–317, Apr. 1995.
[18] V. Lashkia, S. Kaneko, and S. Aleshin, "Textual region location in complex images using test feature classifiers," *Can. J. Elect. Eng.*, vol. 24, no. 2, pp. 65–71, 1999.
[19] W. Wolberg and O. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology," *Proc. Nat. Acad. Sci. (USA)*, vol. 87, pp. 9193–9196, 1990.
[20] O. Mangasarian, R. Setiono, and W. H. Wolberg, "Pattern recognition via linear programming: Theory and application to medical diagnosis," in *Large-Scale Numerical Optimization*, T. Coleman and Y. Li, Eds. Philadelphia, PA: SIAM, 1990, pp. 22–30.

SAFARI: A Structured Approach for Automatic Rule Induction

M. A. Wani

***Abstract*—This paper describes a new algorithm for obtaining rules automatically from training examples. The algorithm is applicable to examples involving both objects with discrete and continuous-valued attributes. The paper explains a new quantization procedure for continuous-valued attributes and shows how appropriate ranges of values of various attributes are obtained. The algorithm uses a decision-tree-based approach for obtaining rules, but unlike other tree-based algorithms such as ID3, it allows more than one attribute at a node which greatly improves its performance. The ability of the algorithm to obtain a measure of partial match further enhances its generalization characteristic. The algorithm produces the same rules irrespective of the order of presentation of training examples. The algorithm has been demonstrated on classification problems. The results have compared favorably with those obtained by existing inductive learning algorithms.**

***Index Terms*—Machine learning, pattern classification, rule induction, system identification.**

I. INTRODUCTION

Knowledge-based expert systems consist of two essential components: a knowledge base and an inference mechanism. While the inference mechanism might be provided as a standard ready-made component of an expert system development tool, obtaining the necessary knowledge to form a customized knowledge base is the first task of the process of building an expert system.

Knowledge can be acquired through protracted interactions between a domain specialist and a knowledge engineer. However, it may be difficult for a domain specialist or a knowledge engineer to define relevant knowledge in an explicit form for complex systems. This is because these systems generate large volumes of data the analysis of which is time-consuming and error-prone.

Inductive learning is a process which can extract rules/concepts from large volumes of data such that the information of interest contained in the rules/concepts is similar to that contained in the original data. Essentially, inductive learning summarizes massive amounts of data to support fast decision making. This makes inductive learning a useful tool for a number of engineering applications [17].

There are two main types of inductive learning algorithms. They are represented by the ID3 algorithm [14] and the AQ algorithm [11]. However, as explained in the next section, both of these algorithms have drawbacks. This has motivated the author to develop the algorithm presented in this paper.

The body of the paper is organized as follows. Section II gives a critical review of the main existing inductive learning algorithms. The new algorithm is detailed in Section III. Section IV gives a simple example to illustrate the operation of the algorithm and describes its application to a standard classification problem and a system identification problem.

II. REVIEW OF INDUCTION ALGORITHMS

Inductive learning algorithms can be classified as divide-and-conquer algorithms and covering algorithms. The popular ID3 algorithm is

Manuscript received July 27, 1995; revised January 9, 2001. This paper was recommended by Associate Editor R. Popp.

The author is with the School of Computing and Mathematics, University of Teesside, Middlesbrough TS1 3BA, U.K. (e-mail: a.wani@tees.ac.uk).

Publisher Item Identifier S 1083-4419(01)04856-7.

a direct descendant of the CLS family of algorithms by Hunt [8]. The algorithm deals with discrete attributes and produces a multi-branch decision tree. ID3 starts with a window, which is a randomly chosen subset of the training set. A decision tree is constructed from this window using a recursive procedure. This tree correctly classifies the cases in the window. All the other cases in the training set are classified using this tree. If any classification error occurs, then a selection of the misclassified cases is added to the window and the process continues until no classification error remains. In this way, a decision tree can be formed after a number of iterations. The core procedure is the recursive process to form a decision tree from the window. This process employs entropy measures to select attributes as nodes of the tree to partition the window. The attributes are selected to minimize the entropy of the partition and therefore maximize the information gain at each stage.

CID3 [5] is a modified version of ID3 which handles continuous-valued attributes. It discretizes continuous-valued variables before applying the standard ID3 tree forming procedure.

Fayyad and Irani [6] present a learning algorithm that generates decision trees using the information entropy minimization heuristic of ID3 for discretizing continuous-valued attributes. The behavior of the information entropy heuristic possesses desirable properties that improve the efficiency of evaluating continuous-valued attributes for cut value selection. Brodley and Utgoff [2] discuss construction of multivariate decision trees and present several algorithms for forming decision trees.

One of the merits of the ID3 algorithm is that its tree forming procedure is very simple. Another advantage with this approach is that a given example can be classified quickly as searching is faster using a tree structure than a sequential list of rules. However, as pointed out by Cheng *et al.* [4] the ID3 approach has the following problems.

1) *Irrelevant value problem:* When ID3 chooses an attribute for branching out of a node, it creates a branch for each value of that attribute that appears in the examples. Some of the values of that attribute may be relevant to the classification, yet the rest may not be. The subtrees generated by such irrelevant values will result in over specialized classification rules, that is rules that check for unnecessary or irrelevant preconditions.

   Cheng *et al.* [4] avoid branching on irrelevant values of an attribute by choosing only those values which appear to be relevant according to the information measure. However, in their approach, a user has to define a tolerance level (TL). Setting $\mathrm{TL} = \infty$ results in a behavior that exactly matches that of ID3. With $\mathrm{TL} = 1.0$, only attribute-value pairs whose entropy measure is equal to the minimum entropy measure will be branched on. The rest are grouped together in a "default" branch. There is no systematic way of determining the proper setting of TL for a given set of data and different trees may be generated for different values of TL.
2) *Missing branches problem:* The missing branches problem is essentially a reduction in the inductive capacity of the tree. It is due to the fact that some of the reduced subsets at the nonleaf nodes do not necessarily contain examples of every possible value of the branching attribute.
3) *Bias problem:* The formula used by ID3 for choosing an attribute for further partitioning a node is heavily biased in favor of attributes with larger ranges of values [15]. To compensate for the bias, a measure of the randomness of the distribution of the learning examples, over the values of the chosen attribute, was introduced by Quinlan [15]. However, this randomness measure does not take into account the classes of the learning examples [4].

Representing the second main type of inductive learning algorithms, the AQ algorithm [11] is provided with two preclassified sets of examples. The set of events for which rules are under construction is called the positive class and the set of events not belonging to the desired concept is the negative class. AQ begins with the most general description and specializes the concept by adding one or more selectors from the seed (a selected positive event) through a process known as "extend against." The specialized concept covers fewer negative examples than the more general concept. When a concept is consistent (covers no negative events) it is saved in a concept set C. A new seed is selected from the positive events that are not covered by any of the concepts in C. The process continues until the concepts in C cover all the positive events. Each of the conjunctive concepts in C is used to define a rule for the positive class. This mechanism ensures that all the positive events are covered by the concept set.

CAQ [17] is a modified version of AQ which can handle both continuous and discrete valued attributes. The developers of CAQ have shown that handling continuous attributes as real numbers instead of forcing them into a discrete representation leads to more efficient concept formulation. However, even this approach may not obtain appropriate ranges for the values of an attribute. This is because the limits will be influenced by the current examples, although they should be based on the overall information contained in all the training examples.

The AQ algorithm is free from the irrelevant condition and bias problems seen in the ID3 algorithm. However, AQ has the following weak points.

1) The chosen seeds may not lead to the extraction of the optimal set of rules. Different sets of rules may be produced if different seeds are employed.
2) The inferencing is slower with this approach compared to a decision tree approach.
3) The algorithm is more difficult to implement compared to ID3.

Another algorithm also belonging to the second type but much simpler than AQ has recently been developed by the authors' group. The algorithm has been referred to as the rule extraction system (RULES) [12]. RULES can handle discrete and continuous-valued attributes. The range of values is specified by the user who can also determine the minimum number of conditions for each rule. The rule extraction procedure involves examining combinations of attribute-value pairs taken from a training example that has not been classified to see if they could serve as the conditions of a rule. The number of attribute-value pairs in a combination must be at least equal to the minimum number of conditions specified by the user. A given combination is deemed a suitable candidate for a rule if it is not found in any training examples of a different class from that of the current example. For a given training example, several candidate rules could be generated. The selected candidate is the one that classifies the most training examples. The procedure terminates once all training examples have been classified.

RULES is free from the problems observed in the ID3 algorithm. However, it has the following drawbacks.

1) The order of presentation of learning examples is important. Different rules may be produced if the order is changed.
2) There is no systematic method for defining the minimum number of conditions in a rule. With that number chosen as one (which is the default value), the accuracy of classification with such rules is usually not good. On the other hand, the arbitrary choice of a higher number of conditions may produce more rules than desired. Also, the induction time increases quickly as the minimum number of conditions is increased. Another problem is that different rules will be produced for the same system if the minimum number of conditions in a rule is changed.
3) As with AQ, the inferencing is slower compared to a decision tree approach.

4) A continuous valued attribute is discretized using fixed intervals which is not natural. The problem of fixed-interval discretization of a continuous valued attribute is that the attribute values may not be divided at natural boundaries. Therefore, the rules extracted with fixed-interval discretized attributes will not be very effective. To achieve a compact set of accurate rules, it is desirable that the limits of an attribute should be as close to the natural boundaries within an attribute as possible.

Another class of algorithms use information theory (IT) for extracting rules from training data. Smyth and Goodman [16] discuss an algorithm falling in this class which maintains a list of R best rules that it finds as it searches the rule space. It considers each of the possible first-order rules for each possible right-hand side, calculates their J-measure, and includes them in the rule list if their information content is greater than that of the Rth best rule found so far. The J-measure calculations are made based on estimates from the data of the various probabilities involved. A decision is then made about whether it is worth specializing the rule further, which consists of adding extra conditions to the left-hand side. The key efficiency of the algorithm lies in the fact that it uses information-theoretic bounds to determine how much information can be gained by further specialization. If the upper bound on attainable information is less than the information of the Rth rule on the list, the algorithm can safely ignore all specializations of this rule and backs up from this point. In this manner, it continues to search and apply bounds until it has covered the entire space of possible rules.

The algorithm is free from problems observed in ID3 algorithm, however, it has the following drawbacks.

1) The algorithm requires a user to specify values of the parameters R (number of rules) and W (the maximum size of the conjunctions allowed in the rules). It is desirable that the value of R should be based on training data rather being specified by the user. This is because the extracted rules may not optimally represent the training data if the value of R is arbitrarily chosen by the user. Same is true with the value of W unless it is set to the maximum value, which will increase the computational effort.
2) The algorithm has the drawback that it uses a brute-force technique for obtaining rules. This involves searching of entire rule space (all possible combinations of various values of all parameters) which makes it less efficient compared to the induction process which is data driven instead of rule driven. In data driven induction process rules are extracted by using one best value of a parameter without exploring the entire rule space.
3) As pointed out by the authors, the algorithm is not directly suitable for domains characterized by continuous variables with regular functional relationships, for example, polynomial relations between real valued variables.
4) The terminology used in information theoretic approach is not semantically explicit. For example, it would be more appropriate to discuss the explicit terminology like significance of a symbol (or attribute value) for classification rather than implicit terminology of information gain in bits.
5) As with AQ and RULES, the inferencing is slower compared to a decision tree-based approach.

Kabakcioglu [9] present an algorithm for construction of binary decision tree using informatic theoretic approach. They construct chain of nodes (CN) from training examples. Each CN forms a production rule for classification purposes. A CN is obtained by iteratively partitioning the training examples until a partition containing examples of one class only is obtained. A certainty measure is attached to a rule if the corresponding CN identifies examples of more than one class.

This algorithm has certain advantages compared to the above one. The production rule induction problem is solved as a series of two class problems which allows the algorithm to find optimal partitions in linear time in s, where s is the number of distinct values of various features. The algorithm does not require a user to specify the values of input parameters like S and W. However, the algorithm does not discuss how to handle the continuous valued variables.

III. New Induction Algorithm

Compared to the algorithms reviewed in the previous section, the new algorithm, called structured approach for automatic rule induction (SAFARI), possesses the following advantageous features.

1) It can employ more than one attribute at each node for greater efficiency (ID3 uses only one attribute per node).
2) It selects attributes by considering information at both a local and a global level for higher accuracy (ID3 and AQ only employ local information).
3) It generates a decision tree for each class of examples for greater inferencing speed and readability (ID3 generates a single decision tree for the complete set of training examples; AQ and RULES produce individual production rules).
4) It handles continuous valued attributes in a more natural way, allowing the ranges of values to correspond closely to the natural boundaries within an attribute.

These advantages will be discussed in detail later in this section following a description of the operation of the proposed algorithm.

A. Definitions

The value of an attribute will be either one of a number of discrete levels or a numeric value within a continuous range. For a continuous-valued attribute, the algorithm first finds a set of possible quantization levels before the extraction of rules begins. The discrete levels and quantization levels are given *symbolic names* which are used in the extraction of rules. Each *symbolic name* (or *symbol*) corresponding to a continuous-valued attribute will have a range of values associated with it. If a symbol covers examples which form a subset of examples covered by another symbol then the former symbol is called a *subset symbol*.

Let $\mathbf{A}$ be a set of m attributes $\{A_1, A_2, A_3, \ldots A_m\}$, $\mathbf{C}$ a set of p classes $\{C_1, C_2, C_3, \ldots C_p\}$, $\mathbf{E}$ the entire set of training examples, and $\mathbf{E1}$ a set of examples at some node. The set of possible symbols of attribute A_i is represented by

$$\mathbf{S_i} = \{S_{i1}, S_{i2}, S_{i3}, \ldots S_{i1_i}\}$$

where 1_i is the number of symbols of attribute A_i. Each example in $\mathbf{E}$ is an $(m+1)$-tuple of the form: $\langle V_1, V_2, \ldots, V_m, C_k\rangle$, where $V_i \in \mathbf{S_i}$, and $C_k \in \mathbf{C}$. N is the total number of training examples and n_{ij} is the number of examples in set $\mathbf{E1}$ which contain symbol S_{ij} (i.e., the number of examples in $\mathbf{E1}$ covered by symbol S_{ij}). N_k is the number of examples in set $\mathbf{E}$ which belong to class k, and $n_{(ij)k}$ is the number of examples in $\mathbf{E}$ containing symbol S_{ij} and belonging to class k.

A measure of the reliability of classifying an example containing a specific symbol S_{ij} at a node is obtained by computing a *local probability* LP_{ij}, defined as the probability of occurrence of a single class in those examples of set $\mathbf{E1}$ which contain symbol S_{ij}.

A measure of the reliability of classifying an example containing a specific symbol S_{ij} using the entire set of training examples is obtained by computing a *global probability* GP_{ij}, defined as the probability of occurrence of a single class in those examples of set $\mathbf{E}$ which contain symbol S_{ij}.

Note that at the root node $\mathbf{E1} = \mathbf{E}$ and the size of set $\mathbf{E1}$ decreases as the number of nodes in a branch of the decision tree increases. As the value of LP_{ij} is computed using set $\mathbf{E1}$, this causes LP_{ij} to have a value which is initially based on global information (i.e., information

contained in set $\mathbf{E}$). As the amount of global information in $\mathbf{E1}$ decreases gradually and information becomes more localized so does the value of LP_{ij}. Therefore, the information used for computing LP_{ij} has a pyramid structure. That is, in the beginning, it is global and toward the end it is highly localized.

The values of LP_{ij}, n_{ij} and GP_{ij} are used to find the *best symbol* for formation of a branch from a node in the decision tree. Generally, parameter LP_{ij} should be sufficient to find the best symbol. However, it is possible that two symbols may have the same LP_{ij} value. In this situation, the symbol having the higher value of n_{ij}/N (proportion of training examples in set $\mathbf{E1}$ containing symbol S_{ij}) is taken as the best symbol. In some cases, the two symbols may have the same value of LP_{ij} and n_{ij}. For such cases, the symbol having the higher value of GP_{ij} is the best symbol.

Mathematically, the best symbol is the one that maximizes the function

$$F = \mathrm{LP}_{ij} + (n_{ij}/N)/\alpha_1 + \mathrm{GP}_{ij}/\alpha_2 \qquad \forall\, i = 1 \text{ to } m \text{ and } j = 1 \text{ to } 1_i$$

where α_1 and α_2 are weighting factors and should be chosen such that it makes LP_{ij} the most significant term, n_{ij}/N a less significant term and GP_{ij} the least significant term in the above expression which is in accordance with the explanation given above. So the value of α_1 will depend on the number of decimal places present in the term LP_{ij}. The value of LP_{ij} can lie between (1 and 0.1) or (1 and 0.01) or (1 and 0.001) for N equals 10 or 100 or 1000, respectively. Therefore, the value of α_1 should be chosen 10 or 100 or 1000 if N has a value between (1 and 10) or (11 and 100) or (101 and 1000), respectively. Same is the case with the term (n_{ij}/N), therefore, the value of α_2 should be chosen $10\alpha_1$ or $100\alpha_1$ or $1000\alpha_1$ if N has a value between (1 and 10) or (11 and 100) or (101 and 1000), respectively.

A measure of how well a given symbol represents a class is obtained by finding the *maximum-local-probability* MLP_{ij}, defined as the maximum probability of occurrence of a class in those examples containing a given symbol. MLP_{ij} gives an indication of the amount of overlap (presence of other classes) in a symbol S_{ij}.

$$\mathrm{MLP}_{ij} = \max(n_{(ij)k}/n_{ij}) \qquad \text{for } k = 1 \text{ to } p.$$

A measure of the *significance* of a symbol, SIG_{ij}, is obtained by finding the maximum ratio of the number of examples of a class containing the symbol to the number of examples of the same class in set $\mathbf{E}$. SIG_{ij} indicates the importance of symbol S_{ij} when only examples of class k are present.

$$\mathrm{SIG}_{ij} = \max(n_{(ij)k}/N_k) \qquad \text{for } k = 1 \text{ to } p.$$

A measure of how well the symbols obtained through automatic quantization of the range of values of a continuous-valued attribute represent that attribute is determined by finding the average of the product of the maximum-local-probability and the significance measure of all symbols in an attribute

$$\text{quantization\_}m = 1/1_i(\Sigma(\mathrm{MLP}_{ij} * \mathrm{SIG}_{ij})) \qquad \text{for } j = 1 \text{ to } 1_i.$$

B. Operation of SAFARI

The algorithm operates by quantizing continuous valued attributes at natural boundaries which are obtained by minimizing the disorder measure defined in Section III-D. The key in quantization process is to identify the limits of contization levels at places where week concentrations of specialized concepts are present. The quantization process is carried out such that the most of the examples in a quantization level belong to one class only. The quantized (or discrete) levels of various attributes are mapped onto a symbol table. This is a one-to-one mapping where each attribute level is assigned one symbol name. The symbol table is then used to obtain one decision tree corresponding to each class.

A decision tree is grown by branching at all nodes. The key in branching is to identify a symbol which best represents the examples of one class (i.e., class of the decision tree). The next best symbol at a node is identified corresponding to examples which are not covered by the selected symbol. Further symbols are identified until all examples at a node are covered. Each identified symbol gives rise to a new branch at a node. This process continues until a node contains examples of one class only (i.e., when the F value of the node equals one or until the F value at a node cannot be improved).

C. Branching at a Node

Branching at a node takes place if

$$\exists \text{ a symbol: } F_{\text{symbol}} \geq F_{\text{leaf-node}} \qquad \text{where } F_{\text{symbol}} = \max(F_s) \;\; \forall s \text{ present in } \mathbf{E1}.$$

Let a list of k such symbols cover all the examples present in $\mathbf{E1}$, and $\mathbf{E1i}$ represent sets of examples covered by ith symbol at the current node then

$$\mathbf{E1} = \{\mathbf{E11}, \mathbf{E12}, \mathbf{E13}, \ldots \mathbf{E1k}\}.$$

A symbol s_i is a subset symbol if

$$\mathbf{E1i} \subset \mathbf{E1j} \text{ for } j = 1 \text{ to } k \text{ and } i \neq j.$$

All subset symbols are deleted from the symbol list. A branch is formed at the current node corresponding to each of the remaining symbols. The algorithm to carry out branching at a node is given below.

$\mathbf{E1}$ is first initialized with those learning examples which satisfy all the conditions along the shortest path from the root to the current node. The following steps are then performed for branching at a node.

1) Set the number of new branches nb to zero at the current node. Set F to zero if the current node is a root node.
2) Select that symbol which maximizes function F in set $\mathbf{E1}$. If the value of F corresponding to this symbol is less than or equal to the F value at the current node, then go to step 5). Otherwise, form a subset of set $\mathbf{E1}$ containing only those training examples not covered by the selected symbol. Increment nb by 1 and store the selected symbol. Replace set $\mathbf{E1}$ by the new subset.
3) Repeat step 2) until $\mathbf{E1}$ becomes an empty set.
4) If more than one symbol is obtained, then delete subset symbols if any. Decrement nb by 1 for each subset symbol deleted.
5) If $\mathbf{E1}$ is an empty set go to step 6). Else, add a terminating branch (a terminating branch is a branch directly leading to a leaf node) at the current node. The F value of the leaf node is the F value of its parent node.
6) Corresponding to each symbol stored, add one branch at the current node. The F value of the node at the end of the new branch is the F value of the symbol forming the branch.

D. Quantization of Continuous Data

Each quantization level of an attribute is represented by a symbol. Quantization of continuous data can be carried out by comparing consecutive values of an attribute arranged in an ascending order. An appreciable difference exceeding a predefined threshold should ideally lead to the next quantization level. However, it is difficult to define the optimal threshold. The following is a procedure which generates suitable threshold values for quantization, based on statistical properties associated with the values of an attribute in the training examples. The

optimal threshold value for quantization of a variable is obtained by minimizing a disorder measure defined as

$$\text{disorder\_}m = 1 - \text{quantization\_}m.$$

The steps for quantizing a continuous-valued attribute are the following.

1) Obtain an initial threshold value by finding the average difference between two consecutive values of an attribute arranged in ascending order, namely

$$\text{initial\_threshold} = 1/(M-1)\Sigma(val_i - val_{i-1}) \quad \forall\, i = 2 \text{ to } M$$

where M is the number of consecutive attribute values and val_i is the ith attribute value.
2) Obtain the disorder measure using the initial threshold value.
3) Repeat step 2) with different values in a close neighborhood of the current threshold value. Adopt the value that minimizes the disorder measure as the value of the threshold.
4) Obtain the quantization levels by performing the following steps.
   a) *Initial quantization:* Arrange attribute values in an ascending order. Initially assign the next higher quantization level to an attribute value if the difference between it and the one immediately below it exceeds the threshold.
   b) *Quantization level merging:* Merge two consecutive quantization levels if they cover examples of the same class.
   c) *Final quantization:* Repeat b) until no more merging is possible, which gives the final quantization levels for the attribute of interest.

The use of disorder_m (a function of MLP_{ij} and SIG_{ij}) for quantizing attributes reduces the overlap in the quantization levels. In situations where various classes share similar values of an attribute, it is likely that a large number of quantization levels of small sizes will be obtained. This will result in a large number of rules. In such situations, it may be desirable to have fewer rules for fast inferencing, which is possible if fewer quantization levels are obtained by allowing a higher degree of overlap. However, this will reduce the accuracy of classification. SAFARI gives the choice of having a small number of rules, which may produce low accuracy.

E. Inferencing

Inferencing involves searching decision trees, which is faster than searching a set of rules sequentially, as already mentioned above. The inferencing procedure is as follows.

1) Search a decision tree with an example to be classified. If a match is found then store the F value of the leaf node of the matching branch.
2) If a match is not found or if the F value obtained is less than 1, then search another decision tree. If the search is successful, then compare the F value of the matching branch with that of the previously matched decision tree. Retain the decision tree corresponding to the higher F value.
3) Repeat step ii) until either the F value becomes greater than or equal to 1 or all the decision trees have been searched.
4) Take the class to which the decision tree belongs as the class of the given example.

F. Time Complexity and Classification Accuracy Comparison

Here time complexity is discussed using N training examples having m v-array attributes (that is, attributes that can take on v number of values). The critical component in SAFARI is the process of selecting a symbol on which to branch. Each such choice involves computation of F value for each symbol, taking time $O(N)$. Once the best symbol is found, the examples are divided into two sets, this takes time $O(N)$. Therefore, the overall time for a single symbol choice is $O(N)$. However, the time complexity of the entire induction process would be different. Under worst case conditions a root node can have z branches (where z is total number of symbols) taking time $O(N.z)$, and a tree can have depth of m levels taking time $O(N.m)$, but it is impossible that both these worst case conditions can occur simultaneously. For example, in a situation where a root node has z branches, the number of depth levels of the tree cannot be more than 1. Similarly in a situation where a tree has m depth levels the maximum number of branches at a node cannot be more than v. Therefore, SAFARI partitions a total of N examples with an overall time complexity of $O(N.m.v)$.

TABLE I
SYMBOL TABLE FOR CAR ACCELERATION PROBLEM

Size		Engine		Max_speed	
symbol	value	symbol	value	symbol	value
$s_{1,1}$	full	$s_{2,1}$	diesel	$s_{3,1}$	high
$s_{1,1}$	full	$s_{2,2}$	propane	$s_{3,1}$	high
$s_{1,2}$	compact	$s_{2,3}$	gas	$s_{3,1}$	high
$s_{1,1}$	full	$s_{2,3}$	gas	$s_{3,1}$	high
$s_{1,3}$	medium	$s_{2,1}$	diesel	$s_{3,2}$	low
$s_{1,2}$	compact	$s_{2,3}$	gas	$s_{3,2}$	low
$s_{1,3}$	medium	$s_{2,3}$	gas	$s_{3,3}$	average
$s_{1,3}$	medium	$s_{2,1}$	diesel	$s_{3,3}$	average

The important component in ID3 is the process of selecting a test attribute on which to branch. Each such choice involves calculation of entropy function for various attributes. The time complexity of the entire induction process of ID3 in worst case conditions involves sorting a total of N examples among m attributes for each level of the tree, giving an overall time complexity of $O(N.m.m)$ as the tree depth is bounded by m.

In AQ, the basic operation is the specialization of concepts in a star. The algorithm generates specialization that cause a negative example to be uncovered by concepts in AQs star. The set of selectors that distinguish the negative example from the seed are found, this takes time $O(m)$. Each concept in the star is specialized by intersection with this set of selectors, taking time $O(m.st)$ where st represents maximum star size. The resulting concepts are evaluated, which takes time $O(N.m.st)$. Therefore, the time complexity of the entire induction process which involves inducing N rules of length m, under worst case conditions is $O(N.N.m.m.st)$, Chan [3].

In RULES, various combinations of attribute values are used for searching of possible rules. Pham and Aksoy [13] report that the total number of searches in worst case conditions for RULES algorithm is $\sum_{i=1}^{m} z!/(z-i)!i!$. This gives an overall time complexity of $O(\sum_{i=1}^{m} z!/(z-i)!i!)$.

The algorithm based on information theory searches the entire rule space which involves examining all combinations of attribute values. Smyth and Goodman [16] report that for m v-array attributes the number of possible rules to be examined by the algorithm equals $m.v((2v+1)^{m-1}-1)$. This gives an overall time complexity of $O(m.v((2v+1)^{m-1}-1))$.

A comparison of classification accuracy of various inductive learning algorithms on Iris data set is given in Table VI. The results shown against the RULES2 and RULES3 algorithms are adopted from [12], [13] and that shown against the information theory algorithm are adopted from [10]. The classification accuracy results of the ID3 row were obtained by implementing the algorithm described in [15] and of the AQ row were obtained by implementing the algorithm summarized in [17]. Except for the information theory algorithm,

TABLE II
RULE SETS OBTAINED BY SAFARI (CAR ACCELERATION PROBLEM) (a) CLASS "GOOD." (b) CLASS "EXCELLENT." (c) CLASS "POOR"

1. IF Max_speed is "low" THEN acceleration is "good"
2. IF size is "full" AND engine is "propane" THEN acceleration is "good"
3. IF size is "full" and engine is "diesel" THEN acceleration is "good"
4. IF engine is "gas" and Max_speed is "high" THEN acceleration is "excellent"
5. IF engine is "gas" and Max_speed is "average" THEN acceleration is "excellent"
6. IF Max_speed is "average" and engine is "diesel" THEN acceleration is "poor"

TABLE III
RULE SET OBTAINED BY RULES-3 (CAR ACCELERATION PROBLEM)

1. IF Size IS full AND Engine IS diesel THEN Acceleration IS good
2. IF Engine IS propane THEN Acceleration IS good
3. IF Size IS compact AND Max_speed IS high THEN Acceleration IS excellent
4. IF Size IS full AND Engine IS gas THEN Acceleration IS excellent
5. IF Max_speed IS low THEN Acceleration IS good
6. IF Engine IS gas AND Max_speed IS average THEN Acceleration IS excellent
7. IF Engine IS diesel AND Max_speed IS average THEN Acceleration IS poor

the results shown in the table were obtained by using 100 random examples for extracting rules and the full Iris data set of 150 examples for testing the extracted rules.

G. Merits of SAFARI

SAFARI has the following strong points.

1) Inherent Ability to Generalize: Generalization is the ability in a system to classify unseen examples. Generalizing requires a degree of resemblance between the unseen examples and the existing rules. SAFARI assesses similarity using the partial match G of an unseen example with the decision trees of various classes. The class of the decision tree corresponding to which G is highest is the class of the unseen example. Partial match search will be required only if the given example cannot be classified using the decision trees because the example lacks some of the attribute symbols contained in the trees. The measure of partial match G is defined as

$$G = F - \sum_{i=1}^{n_{ms}} \Delta F_i.$$

Here, n_{ms} is the number of missing symbols in the test example, and ΔF_i is the gain in F value associated with the missing symbol. The value of ΔF_i is obtained by subtracting the F value of those two nodes in the decision tree which correspond to the missing symbol.

2) Insensitivity to Ordering of Examples: Due to its globalized operation, SAFARI produces the same rules irrespective of the order of the training examples. Furthermore, it does not require a seed and is not influenced by any input parameters or initialization processes.

3) Robustness to Noise: SAFARI is suitable for noisy conditions as the rule generation process, through the use of the maximum F value, selects only those symbols which are least affected by noise. Unseen noise patterns can be handled through the partial matching procedure described earlier.

4) No Irrelevant Conditions: SAFARI addresses the irrelevant condition problem by selecting a symbol (which forms a part of an attribute) instead of an attribute for branching out. Each symbol gives rise to one branch only, therefore, eliminating the problem of irrelevant conditions.

5) No Missing Branches: SAFARI uses information from all the training examples present at a node instead of information from examples present in a current window. The branches obtained by the algorithm at a given node cover all the training examples present at that node, therefore avoiding the missing branches problem.

6) No Bias: The bias problem is addressed by using the measure of the significance of the symbols of an attribute instead of a single significance measure for the entire attribute. The significance measure of a symbol is not biased in favor of attributes with a larger range of values.

7) Optimal Rules: SAFARI keeps adding symbols to a tree (i.e., conditions to a rule) until the F value of the last symbol (i.e., the last condition) of a rule becomes one or until it cannot be improved further. Therefore, the procedure obtains the appropriate number of conditions in a rule, based on the information contained in the training examples, instead of requiring the user to set it arbitrarily.

8) High Speed: SAFARI produces one decision tree for each class. It was shown in Section III-F that tree generation process is linear in N. Further, as mentioned earlier, inferencing by searching a tree structure is faster than sequentially scanning a set of rules. SAFARI uses decision trees for inferencing and therefore is faster than algorithms such as AQ and RULES. The decision trees of various classes can be searched in parallel if parallel processor hardware is available.

The generation of separate decision tree for each class was also proposed by Kabakcioglu [9], but there is a fundamental difference between the two algorithms in obtaining a decision tree. Kabakcioglu's algorithm extracts next CN when a preceding CN has been completely obtained which is unlike the approach discussed in SAFARI where branches at a node are grown simultaneously instead of obtaining one complete path first before starting extracting the next.

9) Efficient Discretization: SAFARI discretizes continuous-valued attributes by minimizing the disorder of discretized data. The minimizing of disorder pushes the range of symbols closer to the desired (natural) boundaries within an attribute, yielding more efficient rules.

IV. ILLUSTRATIVE EXAMPLES

A. Car Acceleration Problem

The simple car acceleration problem adapted from [18] is used to illustrate the operation of SAFARI. The problem involves three attributes: Size, Engine, and Max_speed. Size has three values: full, compact, and medium. Engine has three values: diesel, propane, and gas. Max_speed has three values: high, average, and low. There are three classes of acceleration: good, excellent, and poor. The example set for the problem is given in Table I.

As the problem only involves discrete attributes, there is no need to perform quantization. In step i), SAFARI identifies the symbols associ-

TABLE IV
SYMBOL TABLE FOR IRIS DATA

Sepal_length			Sepal_width			Petal_length			Petal_width		
symbol	value		symbol	value		symbol	value		symbol	value	
	Min.	max.		min.	max.		min.	max.		min.	max.
$s_{1,1}$	4.300	4.850	$s_{2,1}$	2.000	2.100	$s_{3,1}$	1.000	2.450	$s_{4,1}$	0.100	0.800
$s_{1,2}$	4.850	4.950	$s_{2,2}$	2.100	2.250	$s_{3,2}$	2.450	4.450	$s_{4,2}$	0.800	1.350
$s_{1,3}$	4.950	5.050	$s_{2,3}$	2.250	2.350	$s_{3,3}$	4.450	4.550	$s_{4,3}$	1.350	1.450
$s_{1,4}$	5.050	5.150	$s_{2,4}$	2.350	2.450	$s_{3,4}$	4.550	4.750	$s_{4,4}$	1.450	1.550
$s_{1,5}$	5.150	5.250	$s_{2,5}$	2.450	2.550	$s_{3,5}$	4.750	4.850	$s_{4,5}$	1.550	1.650
$s_{1,6}$	5.250	5.350	$s_{2,6}$	2.550	2.650	$s_{3,6}$	4.850	4.950	$s_{4,6}$	1.650	1.750
$s_{1,7}$	5.350	5.450	$s_{2,7}$	2.650	2.750	$s_{3,7}$	4.950	5.050	$s_{4,7}$	1.750	1.850
$s_{1,8}$	5.450	5.550	$s_{2,8}$	2.750	2.850	$s_{3,8}$	5.050	5.150	$s_{4,8}$	1.850	2.500
$s_{1,9}$	5.550	5.650	$s_{2,9}$	2.850	2.950	$s_{3,9}$	5.150	6.900			
$s_{1,10}$	5.650	5.750	$s_{2,10}$	2.950	3.050						
$s_{1,11}$	5.750	5.850	$s_{2,11}$	3.050	3.150						
$s_{1,12}$	5.850	5.950	$s_{2,12}$	3.150	3.250						
$s_{1,13}$	5.950	6.050	$s_{2,13}$	3.250	3.350						
$s_{1,14}$	6.050	6.150	$s_{2,14}$	3.350	3.450						
$s_{1,15}$	6.150	6.250	$s_{2,15}$	3.450	3.550						
$s_{1,16}$	6.250	6.350	$s_{2,16}$	3.550	3.650						
$s_{1,17}$	6.350	6.450	$s_{2,17}$	3.650	3.750						
$s_{1,18}$	6.450	6.550	$s_{2,18}$	3.750	3.850						
$s_{1,19}$	6.550	6.650	$s_{2,19}$	3.850	4.400						
$s_{1,20}$	6.650	6.750									
$s_{1,21}$	6.750	6.850									
$s_{1,22}$	6.850	6.950									
$s_{1,23}$	6.950	7.050									
$s_{1,24}$	7.050	7.900									

TABLE V
RULE SETS OBTAINED BY SAFARI (IRIS DATA) (a) CLASS "SETOSA." (b) CLASS "VERSICOLOR." (c) CLASS "VIRGINICA"

1. IF 1.0 <= Petal_length < 2.45 THEN class is Setosa
2. IF 2.45 <= Petal_length < 4.45 THEN class is Versicolor
3. IF 4.55 <= Petal_length < 4.75 THEN class is Versicolor
4. IF 0.8 <= Petal_width < 1.35 THEN class is Versicolor
5. IF 1.45 <= Petal_width < 1.55 AND 4.45 <= Petal_length < 4.55 THEN class is Versicolor
6. IF 4.75 <= Petal_length < 4.85 AND 1.35 <= Petal_width < 1.45 THEN class is Versicolor
7. IF 1.55 <= Petal_width < 1.65 AND 5.95 <= Sepal_length < 6.05 THEN class is Versicolor
8. IF 1.65 <= Petal_width < 1.75 AND 6.65 <= Sepal_length < 6.75 THEN class is Versicolor
9. IF 1.45 <= Petal_width < 1.55 AND 4.85 <= Petal_length < 4.95 THEN class is Versicolor
10. IF 4.75 <= Petal_length < 4.85 AND 5.85 <= Sepal_length < 5.95 THEN class is Versicolor
11. IF 5.15 <= Petal_length < 6.9 THEN class is Virginica
12. IF 1.85 <= Petal_width < 2.5 THEN class is Virginica
13. IF 1.75 <= Petal_width < 1.85 THEN class is Virginica
14. IF 5.05 <= Petal_length < 5.15 THEN class is Virginica
15. IF 4.95 <= Petal_length < 5.05 THEN class is Virginica
16. IF 1.65 <= Petal_width < 1.75 THEN class is Virginica

ated with each attribute. In step ii), the F values of the various symbols are computed. In this problem, the symbol "low" has the highest F value and is chosen for branching out. However, this branch does not cover all the examples corresponding to class "good." Another symbol, "full," which has the next highest F value is selected. The two selected symbols cover all the examples of the class. Therefore, the decision tree of class "good" has two branches at the root node. At node level 1, the branch with symbol "low" has examples of class "good" only. Therefore, no more branching is required. However, at the same node level 1, the branch with symbol "full" has examples corresponding to more than one class. The F values of all symbols are computed at this node for further branching. Symbol "propane" has the highest F value and is selected for branching, but it does not cover all the examples at this node. So, another symbol, "diesel," which has the next highest F value is selected for branching. At node level 2, both the branches have examples of class "good" only, therefore branching is stopped. This procedure is repeated for the generation of decision trees for classes "excellent" and "poor." The set of rules derived from the trees are listed in Table II. For comparison, Table III shows the rules obtained by RULES-3, which is version 3 of RULES [12].

B. IRIS Data Classification

The IRIS data set [7] is a collection of continuous-valued data commonly used in bench marking pattern classification algorithms. Each example in the set is described in terms of four numerical attributes:

TABLE VI
RULE SET OBTAINED BY RULES-3 (IRIS DATA)

Algorithm	Classification Accuracy (%) Using IRIS data
RULES 2	86
RULES 3	90
ID3	96.7
AQ	96
Information Theory	98.6
SAFARI	100

Sepal_length, Sepal_width, Petal_length, Petal_width and can be classified into one of three categories: Iris_Setosa, Iris_Versicolor, or Iris_Virginica. The total number of examples is 150. In the application, 100 examples were randomly picked for extracting rules and 50 for testing the extracted rules.

The symbols obtained by quantizing the attributes are shown in Table IV. The sets of rules derived from the trees are listed in Table V. The classification results are shown in Table VI together with those produced by RULES-2 (version 2 of RULES) and RULES-3.

V. CONCLUSION

A new algorithm for obtaining decision trees automatically from training examples has been presented. The algorithm has a number of powerful features. It has the inherent ability to generalize, which permits it to classify unseen examples accurately. The algorithm produces the same rules irrespective of the order of presentation of training examples. The algorithm automatically finds the quantization levels for continuous-valued attributes. This is done by minimizing the disorder of symbols representing an attribute, which enables the algorithm to generate efficient rules. The algorithm provides a facility for reducing the number of rules for situations requiring fast inferencing. When the number of rules is reduced, the rules will be more general but also less accurate. The algorithm produces a decision tree corresponding to each class, which speeds up inferencing. The algorithm can obtain partial matches, a very helpful feature for handling noisy data. Finally, the algorithm is free from irrelevant conditions, missing branches and bias problems. The strengths of the algorithm have been demonstrated in bench mark examples covered in the paper. For instance, applying the algorithm to the IRIS data has produced 100% classification results, which to the best knowledge of the authors has not been achieved by any other classifier.

REFERENCES

[1] M. S. Aksoy, "New algorithms for machine learning," Ph.D. dissertation, Cardiff Univ Wales, Sch. Eng., Cardiff, U.K., 1993.

[2] E. Brodley and P. E. Utgoff, "Multivariate decision trees," *Mach. Learn.*, vol. 19, pp. 45–77, 1995.

[3] P. K. Chan, "A critical review of CN2: A polythetic classifier system," Vanderbilt Univ., Dept. Comput. Sci., Nashville, TN, Tech. Rep. CS-88-09, 1988.

[4] J. Cheng *et al.*, "Improved decision trees: A generalized version of ID3," in *Proc. Fifth Int. Conf. Machine Learning*, Ann Arbor, MI, 1988, pp. 100–120.

[5] K. J. Cios and N. Lin, "A machine learning method for generation of a neural network architecture: A continuous ID3 algorithm," *IEEE Trans. Neural Networks*, vol. 3, pp. 280–290, Apr. 1992.

[6] U. M. Fayyad and K. B. Irani, "On the handling of continuous-valued attributes in decision tree generation," *Mach. Learn.*, vol. 8, pp. 87–102, 1992.

[7] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 7, pp. 179–188, 1936.

[8] E. B. Hunt, *Concept Learning: An Information Processing Problem*. New York: Wiley, 1962.

[9] A. M. Kabakcioglu, "Induction of production rules for classification," in *Proc. Fifth Florida Artificial Intell. Research Symp.*, Flairs, FL, 1992, pp. 206–211.

[10] C. Lee, "Generating classification rules from databases," in *Proc. Ninth Int. Conf. Applicat. Artificial Intell. Eng.*, 1994, pp. 205–212.

[11] R. S. Michalski and J. B. Larson, "Selection of most representative training examples and incremental generation of VL1 hypotheses: The underlying methodology and the descriptions of programs ESEL and AQ11," Dept. Comput. Sci., Univ. Illinois, Urbana, Rep. 867, 1978.

[12] D. T. Pham and M. S. Aksoy, "A new algorithm for inductive learning," *J. Syst. Eng.*, vol. 5, no. 2, pp. 115–122, 1995.

[13] ——, "An algorithm for automatic rule induction," *Artif. Intell. Eng.*, vol. 8, pp. 277–282, 1993.

[14] J. R. Quinlan, "Discovering rules by induction from large collections of examples," in *Expert Systems in the Micro-Electronic Age*, D. Michie, Ed. Edinburgh, U.K.: Edinburgh Univ. Press, 1979, pp. 168–201.

[15] ——, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, 1986.

[16] P. Smyth and R. M. Goodman, "Rule induction using information theory," in *Knowledge Discovery in Databases*, G. P. Shapiro and W. J. Frawley, Eds: AAAI, 1991, pp. 159–176.

[17] B. L. Whitehall, S. C. Y. Lu, and R. E. Stepp, "CAQ: A machine learning tool for engineering," *Artif. Intell. Eng.*, vol. 5, no. 4, pp. 189–198, 1990.

[18] R. Yasdi, "Learning classification rules from database in the context of knowledge acquisition and representation," *IEEE Trans. Knowl. Data Eng.*, vol. 3, pp. 293–306, Mar. 1991.

A Decomposition of Fuzzy Relations

Witold Pedrycz, Kaoru Hirota, and Salvatore Sessa

***Abstract*—This study is concerned with a decomposition of fuzzy relations, that is their representation with the aid of a certain number of fuzzy sets. We say that some fuzzy sets decompose an original fuzzy relation if the sum of their Cartesian products approximate the given fuzzy relation. The theoretical underpinnings of the problem are presented along with some linkages with Boolean matrices (such as a Schein rank). Subsequently, we reformulate the decomposition of fuzzy relations as a problem of numeric optimizing and propose a detailed learning scheme leading to a collection of decomposing fuzzy sets. The role of the decomposition in a general class of data compression problems (including those of image compression and rule-based system condensation) is formulated and discussed in detail.**

***Index Terms*—Data compression, decomposition, fuzzy relations, max-t composition, Schein rank of Boolean matrices.**

I. INTRODUCTION

The language of fuzzy relations is commonly encountered in fuzzy sets. Most of essential findings can be formulated and organized in a

Manuscript received September 23, 2000; revised April 24, 2001. This work was supported by the Natural Sciences and Engineering Research Council (NSERC) and the Alberta Software Engineering Research Consortium (ASERC). This paper was recommended by Associate Editor T. Sudkamp.

W. Pedrycz is with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, T6G 2G7 Canada, and also with the Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland (e-mail: pedrycz@ee.ualberta.ca).

K. Hirota is with the Department of Computational Intelligence and Systems Science, Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology, Yokohama, Japan.

S. Sessa is with the Institute of Mathematics, University of Napoli, Napoli, Italy.

Publisher Item Identifier S 1083-4419(01)05977-5.