Induction By Attribute Elimination

Xindong Wu, *Senior Member*, *IEEE*, and
David Urpani

Abstract—In most data-mining applications where induction is used as the primary tool for knowledge extraction from real-world databases, it is difficult to precisely identify a complete set of relevant attributes. This paper introduces a new rule induction algorithm called Rule Induction Two In One (RITIO), which eliminates attributes in the order of decreasing irrelevancy. Like ID3-like decision tree construction algorithms, RITIO makes use of the entropy measure as a means of constraining the hypothesis search space; but, unlike ID3-like algorithms, the hypotheses language is the rule structure and RITIO generates rules without constructing decision trees. The final concept description produced by RITIO is shown to be largely based on only the most relevant attributes. Experimental results confirm that, even on noisy, industrial databases, RITIO achieves high levels of predictive accuracy.

Index Terms—Data mining, rule induction, attribute selection, information entropy.

1 Introduction

Real-world data is dirty. Data cleaning, including the removal of contradictory and redundant data items and the elimination of irrelevant attributes, has been an important topic in data-mining research and development. In most data-mining applications where induction is used as the primary tool for knowledge extraction from real-world databases, attribute elimination is an essential step to improve the efficiency and accuracy of data-mining results. The kind of real-world databases we refer to here are medical databases, where a lot of attributes like age, sex, blood pressure, and temperature are always used to record each patient's medical history and symptoms. Existing data-mining algorithms, such as C4.5 [12] and HCV [15], start with all attributes in a database, and choose useful attributes for concept descriptions. The Rule Induction Two In One (RITIO) algorithm designed in this paper takes a different approach; it eliminates attributes in a database in order of decreasing irrelevancy from the very beginning of its induction. A clear advantage of this approach is that, since the most irrelevant attributes are eliminated from the beginning, they will not mislead the induction process in terms of both efficiency and accuracy.

One major form of representation used in inductive learning is the decision tree in a process known as the top-down induction of decision trees (TDIDT). Example algorithms include C4.5 [12] and its predecessor ID3 [11]. The other major representation form is the "*if... then*" rule structure. Examples include the AQ series [6], CN2 [2], and HCV [15]. Decision tree structures as induced by ID3-like algorithms are known to cause fragmentation of the database whenever a high-arity attribute is tested at a node [10]. This diminishes the understandability of the induced concept hypotheses. Furthermore, ID3-like decision tree structures have a tendency to repeat subtrees when expressing disjunctive concepts of the form `(attribute 1 = A)` or `(attribute 2 = B)`. This is

- *X. Wu is with the Department of Mathematical and Computer Sciences, Colorado School of Mines, 1500 Illinois St., Golden, CO 80401. E-mail: xindong@computer.org.*
- *D. Urpani is with the School of Information Systems, Swinburne University of Technology, Hawthorn 3122, Melbourne, Australia.*

Manuscript received 26 Jan. 1998; revised 29 June 1999.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 106231.

referred to as the replication problem. As a consequence of these two problems, decision trees tend to grow very large in most realistic problem domains.

Rule-like structures are a more convenient form to express knowledge. Rules are similar to the way human experts express their expertise and human users are comfortable with this way of expressing newly extracted knowledge [4]. This is an important consideration during expert validation of a large knowledge base (KB) which might well be equivalent to tens of thousands of decision trees during debugging of the KB and in engendering user acceptance of a KB system. Therefore, with TDIDT algorithms, there is often the added overhead of having to decompile the decision tree into a set of rules. Algorithms directly inducing a set of rules are, therefore, at a distinct advantage as they circumvent the creation of a decision tree and go directly to the creation of the rule set. A direct rule inducer also often produces a more compact concept description than a decision tree inducer.

Hypothesis generation in induction involves searching through a vast (possibly infinite) space of concept descriptions. Practical systems constrain the search space through the use of bias [14]. Bias forces the search to prefer certain hypothesis spaces over others. One such bias that has not been given much attention is to minimize the number of features in the concept description. We propose a rule induction algorithm, RITIO, which prefers those hypothesis spaces that contain fewer attributes by removing irrelevant attributes. We show that this form of attribute-based induction can very efficiently provide syntactically simple concept descriptions with high generalizing powers, even in noisy environments.

This paper presents the RITIO algorithm and provides an empirical evaluation of RITIO with C4.5 (which is commonly recognized as a state-of-the-art method for inducing decision trees [3]), C4.5rules (which transforms C4.5 decision trees into decision rules and manipulates these rules), and HCV [15] (which performs rule induction without generating decision trees). RITIO, like ID3-like algorithms, makes use of the entropy measure, albeit in a different way, as a means of constraining the hypothesis search space but, unlike ID3-like algorithms, the hypotheses language is the rule structure. ID3-like algorithms, including ID3 and C4.5, need a decompiler (such as C4.5rules) to transform decision trees into rules, whereas RITIO carries out rule induction without decision tree construction. In Section 2, we describe the RITIO algorithm and briefly outline C4.5, C4.5rules, and HCV for comparison purposes. In Section 3, we present and compare the results of our empirical investigation of the RITIO algorithm.

2 RITIO and Related Algorithms

In this section, we first describe the RITIO algorithm with an example run and then outline the C4.5, C4.5rules, and HCV programs used for comparative purposes.

C4.5 is a recent successor of ID3, one of the most widely used decision tree inducers, and has been augmented by a decompiler, C4.5rules, to convert a decision tree into a rule set. HCV induces rules in the form of variable-valued logic using an extension matrix approach [17].

2.1 RITIO: A Rule Induction Algorithm

2.1.1 The Algorithm

RITIO carries out a data-driven, specific-to-general search for a consistent set of rules that describe the different classes in the data.

```
Procedure RITIO(E, N; RS)
   /* E is the training example set and RS is the induced rule set */
   /* Let N be the number of attributes used in E */
   RM ← E; L ← 1;
   Partition(RM, L, N; RS);
   Remove duplicate rules in RS;
   Return(RS).

Function Partition(RM, L, N; RS)
   if L < N then
      { Evaluate(RM, Least-Relevant);
        /* Least-Relevant: the least-relevant attribute */
        Remove_Group ← { }; Retain_Group ← { };
        /* Remove_Group and Retain_Group are the 2 partitions
           of RM */
        for each rule R in RM do
          Consistency(RM, Least-Relevant, R; Remove_Attribute);
          /* The Remove_Attribute flag is either TRUE or FALSE */
          if Remove_Attribute = TRUE then
            { R ← R - Least-Relevant;
              /* Remove attribute Least-Relevant from R */
              Remove_Group ← Remove_Group + R }
          else Retain_Group ← Retain_Group + R
        endif;
      endfor;
      L ← L+1;
      if L = N then RS ← Remove_Group + Retain_Group
        else
        { Partition(Remove_Group, L; RS1);
          Partition(Retain_Group, L; RS2);
          RS ← RS1 + RS2 }
      endif
    };
   Return(RS).

   Function Evaluate(RM; Least-Relevant)
      /* This function receives current RM and returns
         the least-relevant attribute */
      Candidate-Attrib ← the set of all attributes in RM;
      for each attribute in Candidate-Attrib
         Calculate its entropy value using data in E;
      endfor;
      Designate the attribute with maximum entropy
         as least-Relevant;
      Return(Least-Relevant).

   Function Consistency(RM, Least-Relevant,
         R; Remove_Attribute)
      /* This function takes the least-relevant attribute and the
         current rule R and returns a Remove_Attribute flag */
      for each Rule in RM do
         for each attribute A in RM do
            if (A <> Least_Relevant)
               & (the value of A in Rule = the value of A in R)
               & (the class of Rule < > the class of R)
               then Remove_Attribute ← FALSE
                  /* There are 2 rules in RM with the same
               attribute value but different classes */
          else Remove-Attribute ← TRUE
      endif
   endfor
endfor
Return(Remove-Attribute).
```

Initially, the rule set is a copy of the training set representing a set of maximally specific generalizations[1] referred to as the rule matrix (RM). In the initial RM, which is the rule matrix at level $L = 0$, a rule exists for each training instance. The level L in Function `Partition` refers to a particular stage in the induction process, with higher levels denoting increasing rule generalization. There are a maximum of $N - 1$ levels, where N is the number of attributes in the database.

RITIO examines each attribute in the training set at each level L), and selects the least relevant in Function `Evaluate`. The heuristic used in `Evaluate` to identify relevancy is the information theoretic function designed by Shannon and Weaver [13] and popularized in Quinlan's ID3 [11]. In contrast to ID3, the heuristic used in RITIO selects the attribute providing the lowest information gain. The entropy of an attribute $A, E(A)$ is the information required to classify an instance based on the information in that attribute. It is defined as follows:

$$E(A) = \sum_{j=1}^{V} (RelFreq_j \times Inf_j),$$

where V is the total number of distinct values in attribute A,

$$RelFreq_j = P_j/T,$$

with P_j being the number of occurrences of value j in attribute A and T the number of training instances, and

$$Inf_j = -\sum_{k=0}^{C} [P_{j_k}/P_j \times log_2(P_{j_k}/P_j)],$$

with P_{j_k} being the number of occurrences of value j in attribute A belonging to class k and C the number of classes in the training set.

$E(A)$ is calculated for each attribute in the database. In ID3, the attribute with the minimum entropy is selected at a decision tree node to split the tree. RITIO chooses the attribute with the maximum entropy as the candidate for elimination from the RM. This guarantees that the least relevant attribute (according to information theory) is eliminated. The induction process will make a total of $N - 1$ entropy calculations.

On identifying the first least relevant attribute, RITIO checks against each training instance to see whether removal of that attribute results in an inconsistency. An inconsistency here is defined as the occurrence of the same example with different classifications after the removal of the least relevant attribute. Function `Consistency` is designed to check for consistency. An attribute whose removal from a rule causes no inconsistency anywhere in the training set is termed a 'removed' attribute.

After removal of the least relevant attribute from all instances where such a removal does not cause an inconsistency, a new, more general RM is generated. The RM has now been partitioned into two distinct groups: one that still retains the full, initial dimensionality, N, called the 'retain' group (`Remove_Group` in Function `Partition`), and the other with a reduced dimensionality, $N - 1$, called the 'remove' group (`Retain_Group` in `Partition`). The RM is now at level 1.

In succeeding rounds of entropy calculations, all previous 'least relevant' attributes are not considered. In this case, the least relevant attribute from the remaining, $N - 1$ attributes across the two existing partitions is chosen as the next candidate for elimination.

1. A maximally specific generalization at this stage is the description of an example in the training set. For example, if we have an example of the `T` class with `(attribute 1 = A)` and `(attribute 2 = B)`, then we have a maximally specific generalization `If (attribute 1 = A) and (attribute 2 = B) then class = T`.

TABLE 1
The Golf Data Set [12]

Order	*Outlook*	*Temperature*	*Humidity*	*Windy*	**Class**
1	sunny	hot	high	false	*Don't Play*
2	sunny	hot	high	true	*Don't Play*
3	overcast	hot	high	false	*Play*
4	rain	mild	high	false	*Play*
5	rain	cool	normal	false	*Play*
6	rain	cool	normal	true	*Don't Play*
7	overcast	cool	normal	true	*Play*
8	sunny	mild	high	false	*Don't Play*
9	sunny	cool	normal	false	*Play*
10	rain	mild	normal	false	*Play*
11	sunny	mild	normal	true	*Play*
12	overcast	mild	high	true	*Play*
13	overcast	hot	normal	false	*Play*
14	rain	mild	high	true	*Don't Play*

Once this attribute has been chosen, the next round in the RM generalization process commences. While checking for consistency, the following rules henceforth apply when identifying members of the training set to be used in the checking process:

1. If checking a rule belonging to the 'retain' group, all training instances are used. In some cases, this also means checking against previously eliminated attributes.
2. If checking a rule belonging to the 'remove' group, only training instances belonging to that group are used. As before, the current least relevant attribute is dropped from those rules which do not cause any inconsistency.

The RM at level $L = 2$ has now been partitioned into four groups (i.e., retain & retain, remove & retain, retain & remove, and remove & remove groups). The process repeats itself iteratively $N - 1$ times with new entropy calculations, consistency checks, and further partitioning of the RM. At any point in the induction process, the RM will contain a maximum of 2^{L-1} partitions up to a final maximum of 2^{N-1} different partitions.

The final RM contains a set of maximally generalized rules guaranteed for consistency. The generalizing process results in a reduction in the number of rules from the original training set size. This manifests itself by repeating rules which are eliminated in the rule extraction process. Another effect of the generalizing process is a reduction in the average dimensionality of the RM as attributes are progressively eliminated. The rules are finally presented as an unordered list in conjunctive normal form.

2.1.2 An Example Run of RITIO

For the data set given in Table 1, there are four attributes; *Outlook*, *Temperature*, *Humidity*, and *Windy*. RITIO will carry out three (the number of attributes minus 1) partitioning loops on the data set.

At Level = 0, the rule matrix (RM) contains all the 14 examples in Table 1.

Using these 14 examples, the Level 1 entropy for each attribute is given below using the formulas in Section 2.1.1.

$$E(Outlook) = 0.694$$

$$E(Temperature) = 0.911$$

$$E(Humidity) = 0.788$$

$$E(Windy) = 0.892$$

TABLE 2
Reduced Golf Data Set at Level 1

Order	*Outlook*	*Humidity*	*Windy*	**Class**
1	sunny	high	false	*Don't Play*
2	sunny	high	true	*Don't Play*
3	overcast	high	false	*Play*
4	rain	high	false	*Play*
5	rain	normal	false	*Play*
6	rain	normal	true	*Don't Play*
7	overcast	normal	true	*Play*
8	sunny	high	false	*Don't Play*
9	sunny	normal	false	*Play*
10	rain	normal	false	*Play*
11	sunny	normal	true	*Play*
12	overcast	high	true	*Play*
13	overcast	normal	false	*Play*
14	rain	high	true	*Don't Play*

From these entropy values, *Temperature* has the highest entropy and, therefore, is the least relevant attribute. RITIO now tries to eliminate this attribute from the data set by checking for consistency.

When attribute *Temperature* is deleted from Table 1, the data set becomes Table 2.

Since the deletion does not cause any inconsistency (with the same example description belonging to different classes), Table 2 becomes our reduced data set and each example gets a more general description (because the condition on *Temperature* has been dropped).

At Level = 2, the entropy values for attributes *Outlook*, *Humidity*, and *Windy* remain the same as at Level 1 and, therefore, *Windy* becomes the current, most relevant attribute. When we try to delete *Windy* from Table 2, the data set becomes Table 3.

This deletion causes inconsistency because Examples 4 and 14 are now contradictory with each other, as are Examples 5, 6, and 10. Therefore, Table 2 has to be partitioned into two groups (Table 4 and Table 5); the 'remove' group (Table 4) with *Windy* deleted and the 'retain' group with *Windy* retained.

With the five examples in Partition 2, the Level 3 entropy for each attribute is as follows (attribute *Windy* is no longer considered at this level).

$$E(Outlook) = 0.971$$

$$E(Humidity) = 0.951$$

Attribute *Outlook* is chosen as the current least relevant attribute. Deleting *Outlook* from Table 5 results in two further partitions (Table 6 and Table 7); the 'remove' group (Table 6) with *Outlook* deleted and the 'retain' group (Table 7) with *Outlook* retained.

The rule set from (Table 6) is: If *Humidity* = normal and *Windy* = false, then **Class** = *Play*. The rule set from Table 7 is: If *Outlook* = rain and *Humidity* = high and *Windy* = false, then **Class** = *Play*; if *Outlook* = rain and *Humidity* = normal and *Windy* = true, then **Class** = *Don't Play*; and if *Outlook* = rain and *Humidity* = high and *Windy* = true, then Class = *Don't Play*.

Iterating the same procedure on Partition 1, we get the final rule set in Table 8.

TABLE 3
Deleting *Windy* from Table 2 at Level 2

Order	*Outlook*	*Humidity*	**Class**
1	sunny	high	*Don't Play*
2	sunny	high	*Don't Play*
3	overcast	high	*Play*
4	rain	high	*Play*
5	rain	normal	*Play*
6	rain	normal	*Don't Play*
7	overcast	normal	*Play*
8	sunny	high	*Don't Play*
9	sunny	normal	*Play*
10	rain	normal	*Play*
11	sunny	normal	*Play*
12	overcast	high	*Play*
13	overcast	normal	*Play*
14	rain	high	*Don't Play*

2.1.3 Noise Handling

Real-world databases are often 1) noisy (with wrong attribute values and/or misclassifications), 2) contradictory (the same data with different class assignments), 3) incomplete (not enough data points) and 4) redundant (attributes or examples do not provide any new information). To be of any practical use, RITIO needs to be able to generalize in the presence of noisy data. In this paper, noise is used in the general sense to imply any and all of the above four types. RITIO handles noise by a series of processes distributed throughout the induction process.

Data Preprocessing. During the data preparation or preprocessing stage, RITIO handles the problems of contradictory and redundant instances and the case of missing attribute values.

TABLE 4
Partition 1 (with *Windy* Deleted) at Level 2

Order	*Outlook*	*Humidity*	**Class**
1	sunny	high	*Don't Play*
2	sunny	high	*Don't Play*
3	overcast	high	*Play*
7	overcast	normal	*Play*
8	sunny	high	*Don't Play*
9	sunny	normal	*Play*
11	sunny	normal	*Play*
12	overcast	high	*Play*
13	overcast	normal	*Play*

TABLE 5
Partition 2 (with *Windy* Retained) at Level 2

Order	*Outlook*	*Humidity*	*Windy*	**Class**
4	rain	high	false	*Play*
5	rain	normal	false	*Play*
6	rain	normal	true	*Don't Play*
10	rain	normal	false	*Play*
14	rain	high	true	*Don't Play*

TABLE 6
Partition 21 (with *Outlook* Deleted) at Level 3

Order	*Humidity*	*Windy*	**Class**
5	normal	false	*Play*
10	normal	false	*Play*

TABLE 7
Partition 22 (with *Outlook* Retained) at Level 3

Order	*Outlook*	*Humidity*	*Windy*	**Class**
4	rain	high	false	*Play*
6	rain	normal	true	*Don't Play*
14	rain	high	true	*Don't Play*

Contradictory instances are allowed. During the induction stage, a contradictory rule arising out of contradictory data is created and flagged as being contradictory. This rule is further processed in the postprocessing stage and may be eliminated or retained (depending on user specified criteria described further on). Redundant instances are retained in the database as they do not affect the final, induced rules. There is an increase in the overheads of time and storage requirements of the algorithm. However, in a real-world environment, assuming random sampling of the training instances, redundancy is useful as it indicates through rule coverage which phenomena are occurring more frequently than others.

The default method used in RITIO for handling missing attribute values is to treat each unknown value as a new attribute value. Another method which can be chosen by the user is to replace the unknown value by the most frequently occurring value, either in the database or in members of the same class.

In RITIO, continuous attributes are discretized into a number of intervals. The resulting intervals can then be used in the same way as nominal attributes during induction and deduction. The methods used for the discretization process are those described in [17] and implemented in the HCV (Version 2.0) software [15]. These methods include, among others, making use of Bayesian classifiers, the information theoretic heuristic or *k*-nearest neighbors. All empirical evaluations of RITIO presented in this paper on those databases that required discretization have used the information theoretic function with class probabilities estimated by the Laplacian Law of Succession [9]. Discretization can also be viewed as contributing to noise handling. A small (with respect to the discretized interval chosen) variation in an attribute value due to noise is smoothed out on assignment of the value to a discretized interval.

During Induction. Induction time noise handling in RITIO is carried out using the information theoretic function to choose the least relevant attribute at each stage of the induction process. This means that the least relevant attribute is chosen before a more relevant attribute as a candidate for elimination from the rule set. The order of attribute selection is an important consideration if the final induced rules are not to be unduly influenced by potentially noisy attributes.

Relevancy refers to the ability of an attribute to discriminate between the different classes in the training set. If an attribute's lack of relevancy is due to corrupt data, then choosing an attribute based on relevancy is, in fact, a noise handling operation. The later in the induction process an attribute is chosen, the bigger the probability that it will be used in the final induced rule set. Therefore, this ordering in the choice of attributes as candidates for elimination ensures that the noisier attributes are less likely to end up in the rule set. This should lead to more accurate rules.

Postinduction Processing. Some of the induced rules may cover only a small number of training instances. It has been shown by other research [1] that such rules are responsible for an unproportional part of the misclassifications of a rule set. Such rules, known as small disjuncts, may be solely a product of noise and not of a real underlying regularity in the training set. A pruning procedure may be used to eliminate from the rule set those rules that are only weakly supported by the empirical evidence in the data.

Two pruning criteria are used in RITIO. The first criterion eliminates those rules which obtain scores below a minimum number of correct classifications expressed as a proportion of the number of instances that match the rule antecedent. This pruning criterion is really the generalizing ability of the rule. The second criterion eliminates rules based on their coverage. The coverage is worked out either with respect to instances of the same class as the rule or else over all training instances. The cut-off values in both criteria are user-defined.

During Deduction. When using the rule set on unseen data, an instance may have no rule that matches it (referred to as *no match* hereafter) or more than one rule with contradictory conclusions (referred to as *multiple match* hereafter).

In no-match and multiple-match cases, RITIO will defer classifying the instances until all other instances have been classified for which unique classifications can be provided by the induction rules. This allows it to build a rule hit score table where 1 point is credited to a rule for each correct classification. After this, RITIO will consider the unclassified set. In the case of no match, it will iteratively cycle through an ever increasing number of allowable mismatches between rules and instances until one or more matches are found. The maximum allowable number of mismatches is a user-set parameter. If a rule or rules are found that match the instance, then the situation is identical to the one where there is at least one rule match. In this case, RITIO will choose the rule with the highest hit score for the classification of the instance in question. In the case of a tie, the first occurring rule is chosen. If, alternatively, the maximum number of mismatches is reached and still no match has been found, the highest scoring rule is used as a default.

2.2 C4.5 (A Decision Tree Inducer) and C4.5 Rules (A Rule Decompiler)

The heart of the popular and robust C4.5 program is a decision tree inducer. It performs a depth-first, general-to-specific search for hypotheses by recursively partitioning the data set at each node of a decision tree.

TABLE 8
RITIO Induced Rule Set from Table 1

Rule No.	*Rule*
1	If *Outlook* = sunny and *Humidity* = high then **Class** = *Don't Play*
2	If *Outlook* = overcast and *Humidity* = high then **Class** = *Play*
3	If *Humidity* = normal then **Class** = *Play*
4	If *Humidity* = normal and *Windy* = false then **Class** = *Play*
5	If *Outlook* = rain and *Humidity* = high and *Windy* = false then **Class** = *Play*
6	If *Outlook* = rain and *Humidity* = normal and *Windy* = true then **Class** = *Don't Play*
7	If *Outlook* = rain and *Humidity* = high and *Windy* = true then **Class** = *Don't Play*

C4.5 attempts to build a simple tree by using a measure of the information gain ratio of each feature and branching on the attribute which returns the maximum information gain ratio. At each point during the search, a chosen attribute is considered to have the highest discriminating ability between the different concepts whose descriptions are being generated. This bias constrains the search space by generating partial hypotheses using a subset of the dimensionality of the problem space. This is a depth-first search in which no alternative strategies are maintained and in which no backtracking is allowed. The final decision tree built, therefore, though simple, is not guaranteed to be the simplest possible tree.

C4.5 uses a pruning mechanism wherein the tree construction process is stopped if an attribute is deemed to be irrelevant and should not be branched upon. A χ^2- test for statistical dependency between the attribute and the class label is carried out to test for this irrelevancy. The induced decision tree can be converted to a set of rules with some pruning and the generation of a default rule.

In the case of missing attribute values, the unknown values are assumed to be distributed in proportion to the relative frequency of these values in the training set. Replacement of unknown values is, therefore, carried out according to this assumption. When classifying test data with missing attribute values, C4.5 tests all branches of this attribute and works out the probability that each value is the correct choice. This probability is summed over all classes. This method performs well under conditions of increasing incidence of unknown values on the databases used by Quinlan [12].

The C4.5rules program contains three basic steps to decompile decision trees to production rules:

1. Traverse a decision tree to obtain a number of conjunctive rules. Each path from the root to a leaf in the tree corresponds to a conjunctive rule with the leaf as its conclusion.
2. Manipulate each condition in each conjunctive rule to see if it can be dropped or can be merged into a similar condition in another rule without more misclassification than expected on the original training examples.
3. If some conjunctive rules are the same after Step 2, then keep only one of them.

Transformation of decision trees to production rules provides a way of combining different trees into the same rule base for more complicated domains. The final decision rules produced are expected to be simpler than the original decision trees, but (as shown in Section 3) not necessarily more accurate when classifying new examples in noisy environments. However, manipulating conditions from the decision-tree-traversal rules in Step 2 is like a new induction algorithm that can work on the original example sets, but in a way totally different from the ID3-like algorithms. Therefore, the time complexity for the transformation is expensive.

2.3 HCV (Version 2.0)—A Rule Induction Program

HCV avoids constructing a decision tree by using an extension matrix approach [5] to generate a set of conjunctive rules to cover each class in the training set relative to all other classes. The extension matrix approach was originally proposed by Hong [5] and then extended in HCV by Wu [15]. In a revised version, HCV (Version 2.0) has been extended to be able to deal with both noisy data and continuous attributes [17]. HCV (Version 2.0) is the version used in this study.

The HCV algorithm considers one class of instances in turn (termed the positive examples, PE) against all the other classes (termed the negative examples, NE). The matrix of NE is termed the negative example matrix (NEM). An extension matrix (EM) is constructed by taking a positive example and comparing it against each member in the NEM. Each attribute value in the NEM that is equal to the corresponding attribute value of the positive example is replaced by a flag denoting a dead element that is unable to distinguish between the positive example and the NE. Repeating this exercise for all positive examples results in a set of extension matrices. Superimposing a group of extension matrices results in the formation of a disjunction matrix (EMD). This superpositioning is governed by two rules: A dead element flag results where at least one EM has a dead element and the original NEM attribute value is retained where no EM has a dead element.

A path is defined as a set of attribute values in the NEM such that there is only one value per row and none of the attribute values are dead elements. An intersecting group of positive examples occurs when a disjunction matrix contains at least one valid path. Such a path corresponds to a conjunctive cover for all the positive examples in the EMD but none of the negative ones. Since the tasks of finding optimal partitions and extracting optimal conjunctive rules from the disjunction matrices are both NP-hard, HCV provides a set of heuristics to carry out these tasks.

HCV has been shown to induce compact rules in low order polynomial time. The rules are expressed in the form of variable-valued logic [7].[2] On a battery of databases from both artificial and real-world domains, HCV has been found to be highly accurate relative to results from ID3-like algorithms [17].

3 Experimental Evaluation

In this section, we present the results of an empirical investigation into the performance of RITIO and compare the results obtained by RITIO with those using C4.5, C4.5rules, and HCV (Version 2.0).

3.1 Test Conditions

Throughout the experiments, the same default conditions were used for all the databases. Obviously, fine tuning different parameters in RITIO would have achieved higher accuracy rates. This, however, would have been at the expense of a loss in generality and applicability of the conclusions. The default conditions used in RITIO were as follows:

1. The induced rule set was pruned by eliminating those rules that had the same class coverage of less than 5.0 percent.
2. The maximum number of mismatches allowed during deduction is the number of attributes minus one.

Similarly, default conditions were adopted for the three other programs C4.5, C4.5rules, and HCV (Version 2.0) as recommended by the respective authors.

All databases used were divided randomly into training and testing partitions by a 70/30 split of the instances. This was carried out 10 times to obtain 10-fold cross-validation trials.

3.2 The Data

The data used in our experiments (see Table 9) can be divided into three groups. The first group is made up of data with 100 percent nominal attributes. The second group contains data of mixed

2. The variable-valued logic originally developed by Michalski [7] is a calculus for representing decision problems where decision variables can take on some range of values. Its principal syntactic entity is a *selector* with the general form, $[X\#R]$, designated as equation (1), where X is a variable or attribute, $\#$ is a relational operator (such as $=, \neq, <, >, \leq$, and $\geq$), and R, called a *reference*, is a list of one or more values that X could take on. A well-formed rule in the logic is similar to a production rule, but with selectors as the basic components of both its lefthand and righthand sides.

TABLE 9
Database Characteristics

Database	# of Instances	Attributes	Classes	Majority Class (%)	Continuous Attributes (%)	Avg # of Values per Attributes	Unknown Values (%)
Hayes-Roth	160	4	3	40.60	0.00	4.00	0.00
Monk1	556	6	2	50.00	0.00	2.80	0.00
Monk2	601	6	2	65.70	0.00	2.80	0.00
Monk3	554	6	2	52.00	0.00	2.80	0.00
Tic-tac-toe	958	9	2	65.30	0.00	3.00	0.00
Soybean	683	35	19	13.50	0.00	2.80	12.00
Vote	435	16	2	61.40	0.00	3.00	5.00
Aus-Credit	690	15	2	56.00	40.00	4.56	0.65
Lab Neg	56	16	2	65.00	50.00	2.62	35.75
Wine	178	13	3	40.00	100.00	n/a	0.00
WTP	523	38	13	52.18	100.00	n/a	2.95
Pot Noise	542	22	2	50.00	100.00	n/a	0.00
UFT	407	20	2	50.00	100.00	n/a	0.00
Temperature	321	22	3	33.33	100.00	n/a	0.00
Pot Difference	195	22	2	50.00	100.00	n/a	0.00
Prediction 1	721	56	2	50.00	100.00	n/a	0.00
Prediction 2	721	68	2	50.00	100.00	n/a	0.00
Prediction 3	616	68	2	50.00	100.00	n/a	0.00

nominal and continuous attributes. These two groups of data were obtained from the University of California at Irvine machine learning database repository [8]. The majority of these databases are noisy. For example, the urban waste water treatment plant data (WTP data) was obtained from daily measures of instrumentation sensors. The classes are different operational states of the plant. This is an ill-structured domain: The class distribution is severely skewed, with some classes having only one or two instances to represent them, and the data is in a time-series.

The third group of data originates from an aluminium smelter and was obtained by process sensors monitoring the aluminium production process. The first three databases in this group, Noise, UFT, and Temperature, predict different abnormal process conditions based on sensor data. The next database, Pot Difference, is used to see whether different production cells differ from each other and by which particular combination of process variables they can be distinguished. The last three databases (Predictions 1, 2, and 3) augment the original sensor data through the use of derived variables (such as rate of change and differences) to try and predict one, two, and three days ahead whether a production cell will exhibit abnormal operating conditions. All these databases are very noisy and not guaranteed to be complete in terms of the given attributes.

3.3 Rule Accuracy

Table 10 shows the accuracy results obtained by the four programs, HCV (Version 2.0), C4.5, RITIO, and C4.5rules. The best result for each problem is highlighted with **boldface** font in the table. Results for C4.5 are the pruned ones. The RITIO results are the average of ten fold cross-validated results on unseen test cases. Also included for RITIO is the 95 percent confidence interval estimate of the mean of the accuracy results. This estimate shows the variance associated with the results and is a good indication of the stability of the algorithm over different databases. For five databases out of the first seven in Table 10, RITIO obtained the best results.

Out of the four databases with continuous data in the second group, RITIO obtained the best results on two of them. RITIO produced a particularly good result on the water treatment plant database, WTP, which is a notoriously difficult real-world database, significantly exceeding the next best. With the industrial databases in the third group, RITIO again obtained the best accuracy results for five out of the seven databases. In several cases, such as the 'Temperature' and 'Prediction 3' databases, RITIO obtained a very significant improvement over the next best result.

From Table 10, RITIO performs significantly better on certain databases (especially the industrial databases from the third group) than others. The reason is that these real-world databases each contain a significant number of "dirty" attributes, as well as contradictory and redundant data items, and attribute elimination in RITIO has managed to take out most of these irrelevant attributes from the beginning of its induction.

4 Conclusions

As real-world databases are normally large and noisy, the problem of focusing on relevant information has become increasingly important in data mining. We have presented, in this paper, a new induction algorithm, RITIO, which uses the information theoretic function in a novel way to induce rules by attribute elimination. It is similar to HCV in its approach of using matrices, but has stronger noise handling capabilities in that it eliminates attributes from the emerging rule set starting with the least relevant attribute. This is in direct contrast to the decision tree inducer in C4.5 which uses the most relevant attribute first to branch on. RITIO has been shown in the experiments carried out on a wide variety of 18 databases to produce concept descriptions of consistently high accuracy which perform better in most cases than C4.5, C4.5rules, or HCV (Version 2.0). The concept description size has been found to be slightly larger in many cases than the sizes of the outputs of the other three programs in terms of the numbers of conjunctive rules and conjunctions in these rules.

TABLE 10
Accuracy Results (in Percentages)

Database	HCV	C4.5	RITIO	95% Est	C4.5rules
Hayes-Roth	85.70	85.6	**87.92**	4.14	71.4
Monk1	**100.00**	83.3	97.37	5.96	**100.0**
Monk2	85.20	69.7	**94.97**	3.39	65.3
Monk3	98.10	97.2	**99.45**	0.61	96.3
Tic-tac-toe	88.00	94.3	98.32	1.11	**100.0**
Soybean	80.20	82.4	**96.80**	0.68	80.6
Vote	97.80	97.0	**98.97**	1.10	93.6
Aus-Credit	82.50	91.0	**93.22**	2.38	90.0
Lab Neg	76.50	**88.2**	77.16	5.37	**88.2**
Wine	90.40	**98.1**	92.37	2.16	**98.1**
WTP	58.62	60.9	**94.77**	0.83	59.2
Pot Noise	94.48	**97.8**	96.00	1.47	97.2
UFT	70.37	69.6	**89.11**	4.46	72.1
Temperature	48.57	61.9	**82.71**	4.60	67.6
Pot Difference	89.23	**96.9**	93.38	3.40	**96.90**
Prediction 1	74.90	68.6	**89.58**	5.58	75.3
Prediction 2	67.78	57.9	**83.71**	2.13	57.5
Prediction 3	60.50	53.8	**86.63**	2.35	54.6

Where this has occurred, however, in the majority of cases, there was a marked increase in accuracy. It is also difficult to compare representation schemes on the basis of the concept description size of different hypothesis.

Future work will involve developing more compact hypothesis representation schemes such as through the use of multiple attribute values and nonmembership. We are also looking at different evaluation functions to employ besides the information theoretic function when selecting attributes for elimination. The consistency check procedure where the cut-off point for excluding or including an attribute will also be modified to take a 'softer' fuzzy approach [16]. This ability to tolerate different levels of inconsistency should add to the already good noise tolerance of the RITIO algorithm.

ACKNOWLEDGMENTS

The authors are grateful to the two anonymous referees for their supportive and constructive comments on the first version of the paper.

REFERENCES

[1] K.M. Ali and M.J. Pazzani, "Reducing the Small Disjuncts Problem by Learning Probabilistic Concept Descriptions," *Computational Learning Theory and Natural Learning Systems,* T. Petsche et al., eds., vol. 3, 1992.
[2] P.E. Clark and R. Boswell, "Rule Induction with CN2: Some Recent Improvements," *Proc. Fifth European Working Session on Learning,* pp. 151–163. Porto, Portugal: Springer-Verlag, 1991.
[3] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and Unsupervised Discretization of Continuous Features," *Proc. 12th Int'l Conf. Machine Learning,* pp. 194–202, 1995.
[4] M. Gams, M. Drobnic, and M. Petkovsek, "Learning from Examples—A Uniform View," *Int'l J. Man-Machine Studies,* vol. 34, pp. 49–68, 1991.
[5] J. Hong, "AE1: An Extension Matrix Approximate Method for the General Covering Problem," *Int'l J. Computer and Information Sciences,* vol. 14, no. 6, pp. 421–437, 1985.
[6] R.S. Michalski, I. Mozetic, J. Hong, and N. Lavrac, "The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains," *Proc. Fifth Nat'l Conf. Artificial Intelligence,* pp. 1,041–1,045, 1986.
[7] R.S. Michalski, "Variable-Valued Logic and Its Applications to Pattern Recognition and Machine Learning," *Computer Science and Multiple-Valued Logic Theory and Applications,* D.C. Rine, ed., pp. 506–534. Amsterdam: North-Holland, 1975.
[8] P.M. Murphy and D.W. Aha, "UCI Repository of Machine Learning Databases, Machine-Readable Data Repository," Dept. of Information and Computer Science, Univ. of California, Irvine, Calif., 1995.
[9] T. Niblett and I. Bratko, "Learning Decision Rules in Noisy Domains," *Research and Development in Expert Systems,* M.A. Bramer, ed., vol. 3, pp. 25–34, Cambridge Univ. Press, 1987.
[10] G. Pagllo and D. Haussler, "Boolean Feature Discovery in Empirical Learning," *Machine Learning,* pp. 71-99, vol. 5, 1990.
[11] J.R. Quinlan, "Induction of Decision Trees," *Machine Learning,* vol. 1, 1986.
[12] J.R. Quinlan, *C4. 5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
[13] C.E. Shannon and W. Weaver, *Math. Theory of Comm.,* Univ. of Illinois Press, Urbana, Ill., 1949.
[14] P.E. Utgoff, "Shift of Bias for Inductive Concept Learning," *Machine Learning: An AI Approach,* pp. 107–148, vol. 2, chapter 5, Morgan Kaufmann, 1986.
[15] X. Wu, *Knowledge Acquisition from Databases.* Ablex, 1995.
[16] X. Wu and P. Måhlén, "Fuzzy Interpretation of Induction Results," *Proc. 1995 Int'l Conf. Knowledge Discovery and Data Mining (KDD-95),* pp. 325–330, Montreal, Aug. 1995.
[17] X. Wu, J. Krisár, and P. Måhlén, "Noise Handling with Extension Matrices," *Int'l J. Artificial Intelligence Tools,* vol. 5, no. 1, pp. 81–97, 1996.