Machine Learning, 4, 161–186 (1989)

Incremental Induction of Decision Trees

PAUL E. UTGOFF UTGOFF@CS.UMASS.EDU
Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003

Editor: Pat Langley

Abstract. This article presents an incremental algorithm for inducing decision trees equivalent to those formed by Quinlan's nonincremental ID3 algorithm, given the same training instances. The new algorithm, named ID5R, lets one apply the ID3 induction process to learning tasks in which training instances are presented serially. Although the basic tree-building algorithms differ only in how the decision trees are constructed, experiments show that incremental training makes it possible to select training instances more carefully, which can result in smaller decision trees. The ID3 algorithm and its variants are compared in terms of theoretical complexity and empirical behavior.

Keywords. Decision tree, concept learning, incremental learning, learning from examples.

1. Introduction

The ability to learn classifications is fundamental to intelligent behavior. Concepts such as "dangerous," "edible," "ally," and "profitable" are learned routinely and are essential to getting along in the world. During the last two decades, researchers have developed a number of algorithms for automated concept acquisition. These include Meta-Dendral [Buchanan & Mitchell, 1978], which learned the concept of a chemical bond that is likely to break in a mass-spectrometer, and AQ11 [Michalski & Chilausky, 1980], which learned to diagnose soybean disease according to observable symptoms.

One dimension along with concept-learning algorithms can be distinguished is whether they operate in an incremental or nonincremental mode. In the latter case, the algorithm infers a concept once, based on the entire set of available training instances. Nonincremental algorithms include Vere's THOTH [1980], Michalski's INDUCE [1983], and Quinlan's ID3 [1983, 1986]. In the incremental case, the algorithm revises the current concept definition, if necessary, in response to each newly observed training instance. Incremental algorithms include Mitchell's Candidate Elimination Algorithm [1978], Schlimmer and Granger's STAGGER [1986], Fisher's COBWEB [1987], and Gallant's Pocket Algorithm [1988]. A nonincremental algorithm is appropriate for learning tasks in which a single fixed set of training instances is provided; an incremental algorithm is appropriate for learning tasks in which there is a stream of training instances. For serial learning tasks, one would prefer an incremental algorithm, on the assumption that it is more efficient to revise an existing hypothesis than it is to generate a hypothesis each time a new instance is observed. The cost of generating hypotheses from scratch may be too expensive to apply a nonincremental method to a serial-learning task.

This article presents an algorithm for the incremental induction of decision trees. The method infers the same decision tree as Quinlan's [1983, 1986] nonincremental ID3 algorithm, given the same set of training instances. This makes it practical to build decision trees for tasks that require incremental learning. The algorithm also makes it practical to choose training instances more selectively, which can lead to a smaller decision tree.

The following section reviews decision trees and the ID3 algorithm. Section 3 explains the need for an incremental algorithm, reviews Schlimmer and Fisher's ID4 algorithm, and presents a new incremental algorithm called ID5R. Section 4 provides a theoretical analysis of worst-case complexity for four decision-tree algorithms, and Section 5 compares the algorithms in terms of their empirical behavior. The last two sections summarize the main conclusions and discuss directions for further work.

2. Decision Trees

A decision tree is a representation of a decision procedure for determining the class of a given instance. Each node of the tree specifies either a class name or a specific test that partitions the space of instances at the node according to the possible outcomes of the test. Each subset of the partition corresponds to a classification subproblem for that subspace of the instances, which is solved by a subtree. A decision tree can be seen as a divide-and-conquer strategy for object classification. Formally, one can define a decision tree to be either:

1. a leaf node (or answer node) that contains a class name, or
2. a non-leaf node (or decision node) that contains an attribute test with a branch to another decision tree for each possible value of the attribute.

For a comprehensive discussion of decision trees, see Moret [1982] and Breiman, Friedman, Olshen, and Stone [1984].

Quinlan's ID3 program induces decision trees of the above form. Each training instance is described as a list of attribute-value pairs, which constitutes a conjunctive description of that instance. The instance is labeled with the name of the class to which it belongs. To simplify the discussion, it is assumed that an instance belongs to one of two classes: the positive instances, which are examples of the concept to be learned (the target concept), and the negative instances, which are counterexamples of the target concept. The algorithm can be applied to more than two classes in a straightforward manner. In addition, it is assumed that attributes have discrete values.

The complete ID3 algorithm, discussed in Section 5.3, includes a method for selecting a set of training instances from which the decision tree will be built. For now the discussion focuses on the basic problem of constructing a decision tree based on a given set of selected training instances. Table 1 shows just the tree-construction algorithm, here called the basic ID3 tree-construction algorithm, that is embedded within the complete ID3 algorithm. The set of attributes used to describe an instance is denoted by A, and the individual attributes are indicated as a_i, with i between one and the number of attributes, denoted $|A|$. For each attribute a_i, the set of possible values for that attribute is denoted V_i. The individual values

Table 1. The basic ID3 tree construction algorithm.

1. If all the instances are from exactly one class, then the decision tree is an answer node containing that class name.
2. Otherwise,
   (a) Define a_{best} to be an attribute with the lowest E-score.
   (b) For each value $v_{best,i}$ of a_{best}, grow a branch from a_{best} to a decision tree constructed recursively from all those instances with value $v_{best,i}$ of attribute a_{best}.

are indicated by v_{ij}, with j between one and the number of values for attribute a_i, denoted $|V_i|$. The E-score is the result of computing Quinlan's [1986] expected information function E of an attribute at a node. At a given node, let

$$\begin{aligned} p &= \text{number of positive instances} \\ n &= \text{number of negative instances} \\ p_{ij} &= \text{number of positive instances with value } v_{ij} \text{ of attribute } a_i \\ n_{ij} &= \text{number of negative instances with value } v_{ij} \text{ of attribute } a_i \end{aligned}$$

Then

$$E(a_i) = \sum_{j=1}^{|V_i|} \frac{p_{ij} + n_{ij}}{p + n} I(p_{ij}, n_{i,j})$$

with

$$I(x, y) = \begin{cases} 0 & \text{if } x = 0 \\ 0 & \text{if } y = 0 \\ -\dfrac{x}{x+y} \log \dfrac{x}{x+y} - \dfrac{y}{x+y} \log \dfrac{y}{x+y} & \text{otherwise} \end{cases}$$

Quinlan [1986] has pointed out that selecting an attribute with a lowest E-score is equivalent to selecting an attribute with a highest information gain, defined as $I(p, n) - E(a_i)$.

The E function is an information-theoretic metric based on entropy[1] [Shannon, 1948; Lewis, 1962]. This function estimates the amount of ambiguity in classifying instances that would result from placing the attribute as a test at a decision node. The attribute with the lowest E-score is assumed to give a good partition of the instances into subproblems for classification. Although the algorithm is not guaranteed to find a smallest tree, experience has shown that it usually builds a small decision tree that generalizes well to the unobserved instances. Quinlan [1986] has reported an improved attribute selection metric, called the gain-ratio criterion, which compensates for the fact that attributes with more values tend to be preferred to attributes with fewer values. This improved metric is discussed in Section 5.3.

3. Incremental Induction of Decision Trees

ID3 is a useful concept-learning algorithm because it can efficiently construct a decision tree that generalizes well. For nonincremental learning tasks, this algorithm is often a good choice for building a classification rule. However, for incremental learning tasks, it would

be far preferable to accept instances incrementally, without needing to build a new decision tree each time. This section first reviews Schlimmer and Fisher's ID4 algorithm, which was designed to learn decision trees incrementally, and then presents a new incremental tree-construction algorithm, named ID5R, that builds the same decision tree as ID3's basic tree building algorithm from a given set of training instances.

3.1. The ID4 Algorithm

Previously, Schlimmer and Fisher [1986] designed an algorithm named ID4 to address the incremental construction of decision trees. However, as shown below, there are many concepts that are not learnable by ID4, even though they are learnable by ID3. The need for an incremental tree-induction algorithm, and the inadequacy of the ID4 algorithm in this regard, motivated the research reported here. The ID4 algorithm is presented here in detail for two reasons. First, the algorithm needs to be understood so that its inability to learn certain concepts can be explained. Second, the mechanism in the ID4 algorithm for determining when to change the attribute test at a node is included in the new algorithm ID5R, described in the next section.

As shown in Table 2, the ID4 algorithm accepts a training instance and then updates the decision tree, which is kept as a global data structure. Step 1 indicates that all information needed to compute the E-score for a possible test attribute at a node is kept at that node. This information consists of positive and negative counts for each possible value of each possible test attribute at each node in the current decision tree. For ease of discussion, let a *non-test* attribute be an attribute that could be the test attribute at a node, but is not. From the information kept at a node, it is possible to determine which attribute has the lowest E-score. If the current test attribute does not have the lowest E-score, the test attribute is replaced by a non-test attribute with the lowest E-score. Because the counts have been maintained, there is no need to re-examine the training instances.

The complete ID4 algorithm also includes a χ^2 test for independence, as described by Quinlan [1986]. The test helps prevent overfitting the training data, when noise is present, by preventing replacement of an answer node with a decision node. To include the test, step 3 of the algorithm is executed only if there is an attribute that is also not χ^2 independent. Throughout this article, the discussion focuses on the basic algorithms, omitting the χ^2 test.

Table 2. The basic ID4 tree-update procedure.

1. For each possible test attribute at the current node, update the count of positive or negative instances for the value of that attribute in the training instance.
2. If all the instances observed at the current node are positive (negative), then the decision tree at the current node is an answer node containing a "+" ("−") to indicate a positive (negative) instance.
3. Otherwise,
   (a) If the current node is an answer node, then change it to a decision node containing an attribute test with the lowest E-score.
   (b) Otherwise, if the current decision node contains an attribute test that does not have the lowest E-score, then
      i. Change the attribute test to one with the lowest E-score.
      ii. Discard all existing subtrees below the decision node.
   (c) Recursively update the decision tree below the current decision node along the branch of the value of the current test attribute that occurs in the instance description. Grow the branch if necessary.

The ID4 algorithm builds the same tree as the basic ID3 tree-construction algorithm only when there is an attribute at each decision node that is clearly the best choice in terms of its E-score. Whenever the relative ordering of the possible test attributes at a node changes during training, step 3(b)ii discards all subtrees below the node. If the relative ordering does not stabilize with training, then subtrees will be discarded repeatedly, rendering certain concepts unlearnable by the algorithm. Such thrashing can be observed for concepts in which the decision tree must extend three or more levels below the current node and in which there is no stable best test attribute for the node.

For example, consider the concept of even parity for three Boolean variables a, b, and c. Because a three-level tree is needed and because any of the three attributes is equally good at the root, small fluctuations in the E-scores during training will cause the test attribute at the root to change repeatedly. If the eight instances are presented repeatedly in a fixed order (a simple and fair sampling strategy), the tree will never stabilize at the root, rendering the concept unlearnable. If the instances are sampled in a random order, then one must hope for a sequence that permits a consistent tree to be found and never lost. Note that if one includes the χ^2 test, the concept is still unlearnable because all attributes at the root appear to be noisy.

3.2. A New Algorithm: ID5R

This section presents a new incremental algorithm, called ID5R, that is guaranteed to build the same decision tree as ID3 for a given set of training instances. Effectively, this lets one apply Quinlan's nonincremental method to incremental learning tasks, without the expense of building a new decision tree after each new training instance. ID5R is a successor of the ID5 algorithm [Utgoff, 1988] described in Section 4.4. Like ID4, the ID5R algorithm maintains sufficient information to compute the E-score for an attribute at a node, making it possible to change the test attribute at a node to one with the lowest E-score. However, ID5R differs in its method for changing the test attribute at a node. Instead of discarding the subtrees below the old test attribute, ID5R restructures the tree so that the desired test attribute is at the root. The restructuring process, called a *pull-up*, is a tree manipulation that preserves consistency with the observed training instances, and that brings the indicated attribute to the root node of the tree or subtree. The advantage of restructuring the tree is that it lets one recalculate the various positive and negative counts during the tree manipulations, without re-examining the training instances.

Formally, define an ID5R decision tree to be either of:

1. a leaf node (or *answer node*) that contains
   (a) a class name, and
   (b) the set of instance descriptions at the node belonging to the class
2. a non-leaf node (or *decision node*) that contains
   (a) an attribute test, with a branch to another decision tree for each possible value of the attribute, the positive and negative counts for each possible value of the attribute, and
   (b) the set of non-test attributes at the node, each with positive and negative counts for each possible value of the attribute.

This form of decision tree differs from those of both ID3 and ID4 because it retains the training instances in the tree. Accordingly, the tree must be interpreted differently. When using the tree to classify an instance, the tree is traversed until a node is reached for which all training instances are from the same class, at which point the classification is determined and an answer can be returned. Such a node can be either a leaf or non-leaf node. In the latter case, there will be only one class with a non-zero instance count.

This method for interpreting the tree structure allows the intended generalization without actually discarding the information in the training instances, which must be retained so that restructuring remains possible. Call the leaf-node form of tree an *unexpanded* tree and the nonleaf-node form an *expanded* tree. Note that this method for interpreting a tree means that the tree structure underlying a tree's interpretation may not be unique. The actual form of the tree could be the unexpanded form or an expanded form with one or more superfluous attribute tests.

The instances at a node are described only by the attribute-value pairs that have not been determined by tests above the node. This means that the instance descriptions are reduced by one attribute-value pair for each attribute tested above in the tree. One might imagine that the simplest approach would be to maintain the tree structure in fully expanded form. However, this is inefficient in both time and space because at each non-leaf node all the counts for the non-test attributes must be maintained.

Table 3 shows the ID5R algorithm for updating a decision tree. If the tree is in unexpanded form and the instance is from the same class, then the instance is added to the set kept in the node. Otherwise, the tree is expanded one level if necessary (choosing the test attribute arbitrarily because no counts of positives and negatives yet exist at the node), and the attribute-value counts are updated for the test and non-test attributes according to the training instance. If the test attribute no longer has the lowest E-score, the tree is restructured so that the test attribute is one with the lowest E-score. If the tree was restructured then one recursively checks the subtrees that will not be touched again during the tree update, restructuring them as necessary so that every test attribute at a node has the lowest E-score

Table 3. The ID5R tree-update algorithm.

1. If the tree is empty, then define it as the unexpanded form, setting the class name to the class of the instance, and the set of instances to the singleton set containing the instance.
2. Otherwise, if the tree is in unexpanded form and the instance is from the same class, then add the instance to the set of instances kept in the node.
3. Otherwise,
   (a) If the tree is in unexpanded form, then expand it one level, choosing the test attribute for the root arbitrarily.
   (b) For the test attribute and all nontest attributes at the current node, update the count of positive or negative instances for the value of that attribute in the training instance.
   (c) If the current node contains an attribute test that does not have the lowest E-score, then
      i. Restructure the tree so that an attribute with the lowest E-score is at the root.
      ii. Recursively reestablish a best test attribute in each subtree except the one that will be updated in step 3d.
   (d) Recursively update the decision tree below the current decision node along the branch for the value of the test attribute that occurs in the instance description. Grow the branch if necessary.

Table 4. The ID5R pull-up algorithm.

1. If the attribute a_{new} to be pulled up is already at the root, then stop.
2. Otherwise,
   (a) Recursively pull the attribute a_{new} to the root of each immediate subtree. Convert any unexpanded tree to expanded form as necessary, choosing attribute a_{new} as the test attribute.
   (b) Transpose the tree, resulting in a new tree with a_{new} at the root, and the old root attribute a_{old} at the root of each immediate subtree.

at the node. Then the tree update continues recursively at the next lower level, following the branch with the value that appears in the instance description. The ID5R algorithm builds the same tree as the basic ID3 tree construction algorithm, given the same instances and given the same method for breaking ties among equally good attributes.

Table 4 shows the algorithm for restructuring the decision tree so that the desired test attribute is at the root. An *immediate subtree* of a node is a subtree that is rooted at a child of the node. To transpose[2] a tree at the root (level zero), the order of attribute tests from level zero to level one is reversed, resulting in a regrouping of the level-two subtrees. Since the level-zero attribute-value counts are already known from the tree update that led to the pull-up, and the level-two trees are not touched; only the counts for the level-one test and non-test attributes need to be computed. These are obtained directly by adding the counts from the level-two subtrees. Then the tree update procedure continues recursively at the next level.

3.3. An Illustration of the ID5R Algorithm

This section illustrates the ID5R algorithm, showing its behavior on eight serially presented training instances given in Quinlan [1983]. The instances, shown in Table 5, are each described as a conjunction of three attribute-value pairs, using the attributes *height*, *hair* color, and color of *eyes*. In the trees that are drawn below, the positive and negative counts are shown as a bracketed ordered pair, e.g. [positives, negatives].

Table 5. Training instances for the Quinlan [1983] example.

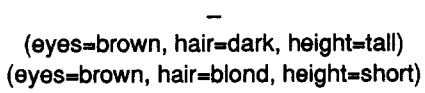
−	short	blond	brown
−	tall	dark	brown
+	tall	blond	blue
−	tall	dark	blue
−	short	dark	blue
+	tall	red	blue
−	tall	blond	brown
+	short	blond	blue

For the first instance (−, *height* = *short*, *hair* = *blond*, *eyes* = *brown*), the tree is updated to be simply the leaf node

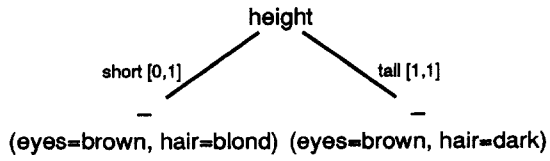
−

(eyes=brown, hair=blond, height=short)

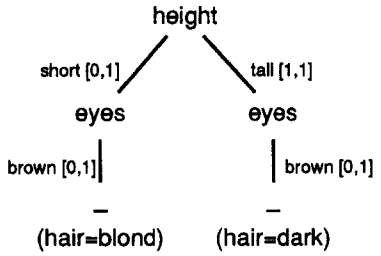
The second instance (−, *height* = *tall*, *hair* = *dark*, *eyes* = *brown*) is also negative, so its is added to the list of instances at the leaf node, giving



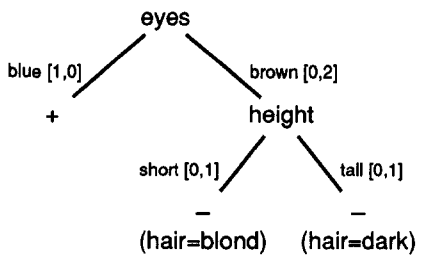
The third instance (+, *height* = *tall*, *hair* = *blond*, *eyes* = *blue*) is positive, so the tree is expanded, arbitrarily choosing attribute *height* as the test attribute. The positive and negative counts are updated at level zero, and the non-test attribute *eyes* is found to have the lowest E-score. At the moment, the tree is



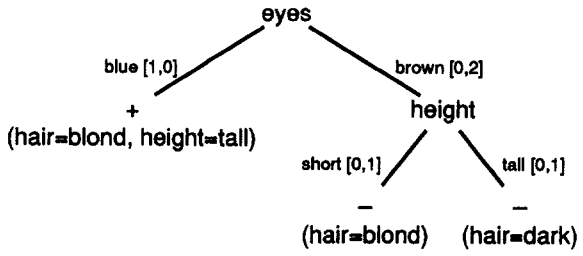
so the attribute *eyes* must be pulled to the root. Note that the tree is in the process of being updated, so level zero is based on three training instances while level one and below are based on two training instances. When the update process is finished, the entire tree will be based on the three training instances. The first step is to restructure the immediate subtrees so that *eyes* is the test attribute for each one. For the current tree, the subtrees are in unexpanded form, so they are expanded, choosing attribute *eyes* as the test attribute. At this point, the tree is



but it is then transposed to

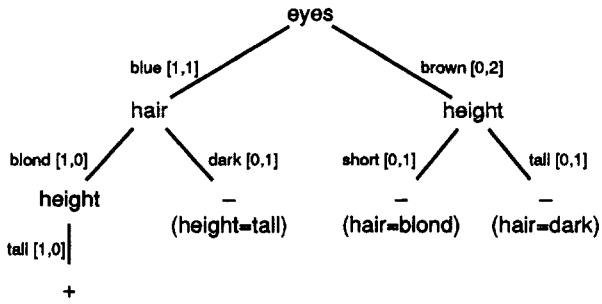


The branch for value *blue* of attribute *eyes* comes about because all the possible level-zero instance counts have already been updated. Thus, when *eyes* becomes the test attribute, its possible branches are already known. Now the algorithm needs to ensure that a best test attribute is at the root of each immediate subtree, except the tree below *blue*, which will be updated when the rest of the training instance is processed. The current test attribute *height* is at the root of the tree below *brown* due to transposition, not due to having the lowest E-score. Note that because the tree has been updated at level zero, but not yet updated at level one, the subtree below *eyes* = *blue* is currently empty. The subtree will be created when level one is updated. It happens that no tree restructuring is needed. The rest of the instance is processed recursively at level one, resulting in

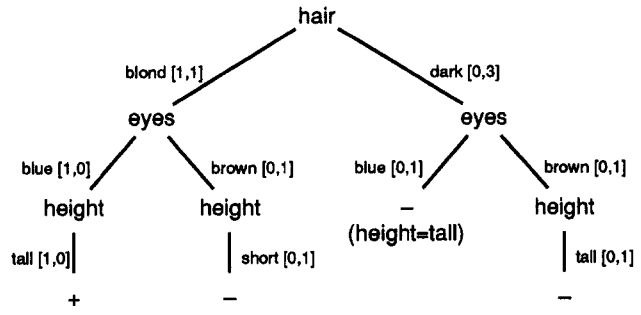


Note that the right subtree, with *height* at its root, could be contracted to unexpanded form because all the instances at the node are from the same class. The ID5R algorithm does not include any contraction step because it is not known when such contractions are worthwhile. An unexpanded tree requires less space and is cheaper to update. However, there is expense in expanding a tree because the instances must be examined so that the positive and negative counts can be computed. Experience with the algorithm has shown that contraction to unexpanded form is generally not worthwhile.

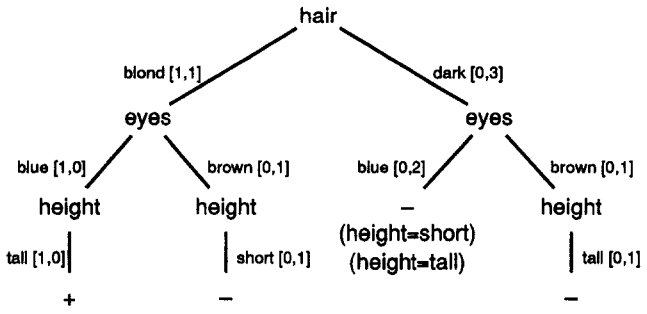
The fourth instance (−, *height* = *tall*, *hair* = *dark*, *eyes* = *blue*) leads to expansion of the left level-one subtree and selection of *hair* as the test attribute, giving



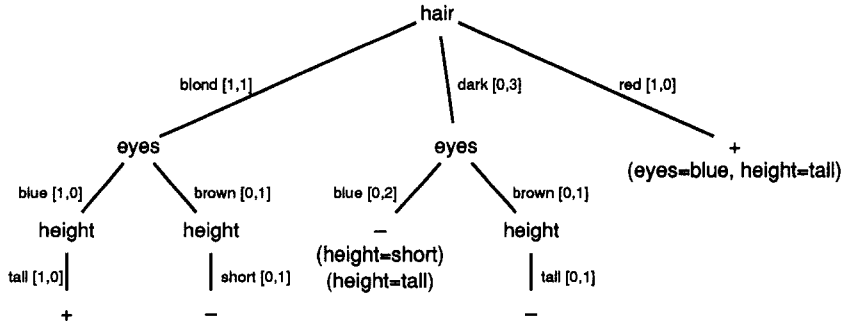
The fifth instance (−, *height* = *short*, *hair* = *dark*, *eyes* = *blue*) causes the test attribute at the root to be changed to *hair*. Immediately following the transposition step, the tree is



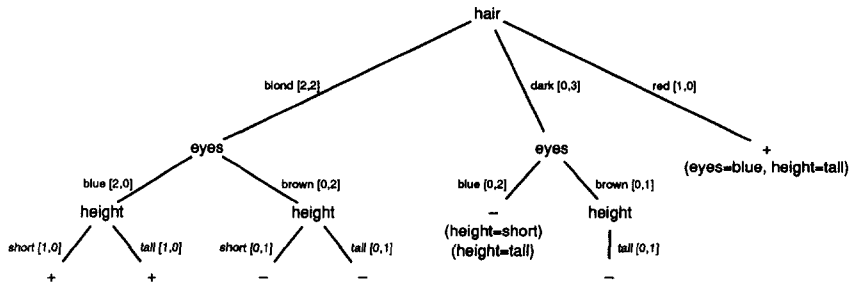
Again it happens that no further tree restructuring is needed. The rest of the instance is processed, and the resulting tree is



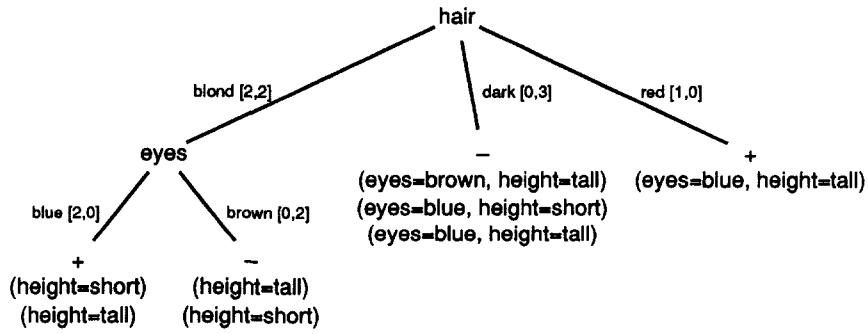
The sixth instance (+, *height* = *tall*, *hair* = *red*, *eyes* = *blue*) does not require changing the test attribute at the root. Because *red* is a new value for *hair*, a new value branch is grown, giving



The seventh and eighth instances update the various counts but do not cause any tree revision. After the eighth instance, the tree is



If the tree were to be contracted by converting to unexpanded form where possible, the result would be



but the algorithm does not perform this step. The tree is shown in this compact form only to facilitate its interpretation. This final tree is equivalent to the one that the basic ID3 tree-construction algorithm would build for these eight training instances.

4. Theoretical Analysis of Four Decision-Tree Algorithms

With alternative methods for constructing a decision tree, it is natural to ask which one is best. This section presents a theoretical worst-case analysis of four algorithms[3] that build a decision tree based on a set of training instances. The first algorithm is the basic tree-construction algorithm of the nonincremental ID3. The second algorithm is a variant of the first suggested by Schlimmer and Fisher and is called ID3′ here. This algorithm accepts the training instances one at a time, each time building a new tree from the accumulated set of training instances using the basic ID3 tree-construction algorithm. This is an obvious brute-force method for inferring a decision tree when the instances are presented serially. The third algorithm is the incremental ID5R. Finally, the fourth algorithm is the incremental ID5, a predecessor of the ID5R algorithm.

The analysis is in terms of worst-case cost to build a tree, given a set of instances, and is similar to that of Schlimmer and Fisher [1986]. Their analysis was measured in terms of the total number of attribute comparisons while building the tree. In the analysis here,

there are two metrics. The first measures the cost of maintaining the positive and negative instance counts in terms of the number of additions performed. Each addition needed to compute such a count is called an *instance-count addition*. The metric is similar to Schlimmer and Fisher's number of attribute comparisons, but also includes the cost of adding counts during the tree transposition process. The second measures the cost of attribute selection in terms of the number of E-score computations. This activity is not very costly for the basic ID3 tree-construction algorithm, but should be considered for the ID5R and ID4 algorithms, which compute the E-score for every non-test attribute at every decision node in the tree.

A worst-case analysis lets one ignore domain characteristics, such as the ability of attributes to discriminate one class from another. In an ideal representation, there is a single attribute that determines class membership, leading to a shallow decision tree; a less-than-ideal representation would require a deeper tree. In general, larger trees are more expensive to construct than smaller trees, so the suitability of the representation affects the size of the decision tree and therefore the expense of building the tree. The analysis also assumes that all possible training instances are observed. Even with these pessimistic worst-case assumptions, it is possible to draw useful conclusions from the analyses. To simplify the notation in the analyses below, let d be the number of attributes (i.e., $|A|$), b be the maximum number of possible values for an attribute, and n be the number of training instances.

4.1. Analysis of the Basic ID3 Algorithm

The basic tree-construction algorithm of ID3 first checks whether all the instances at a node are from the same class. If so, then the tree is just an answer node containing the class name. Otherwise, the algorithm counts the number of positive and negative instances for each value of each attribute. It then computes the E-score for each attribute, selects an attribute with the lowest E-score to be the test attribute at the node, and builds a tree recursively for each subset of the instances with the same value of the selected test attribute. If there is only one possible attribute at a node, then there is no need to compute the E-score.

In choosing the test attribute at the root, the algorithm must count all d attribute values of all n instances. At the root (level zero) there are $d \cdot n$ instance-count additions and d E-score calculations. At level one, there will be b subtrees in the worst case. Because the level-zero test only partitions the instances into a set of subproblems for which subtrees must be built, all n instances will be examined at level one, in the worst case. However, at level one each instance is described by only $d - 1$ attributes, so there will be $(d - 1) \cdot n$ instance-count additions and $(d - 1) \cdot b$ E-score calculations. In general, the worst-case cost of building the tree in terms of instance-count additions is

$$\sum_{i=1}^{d} i \cdot n = \frac{d \cdot (d + 1)}{2}$$

$$= O(n \cdot d^2) .$$

In terms of E-score calculations, the worst-case cost of building the tree is

$$\sum_{i=2}^{d} i \cdot b^{d-i} = \frac{2b^d - b^{d-1} - (d+1)b + d}{(b-1)^2}$$

$$= O(b^d) .$$

This analysis shows that the complexity of the worst-case cost of building a tree with the basic ID3 tree-construction algorithm is linear in the number of instances for instance-count additions and constant for E-score calculations. Quinlan (personal communication) has pointed out that because a tree of n instances will contain at most $n - 1$ decision nodes, and because there are at most d E-score calculations at a node, there will be at most $d \cdot (n - 1)$ E-score calculations. It will usually be the case that $d \cdot (n - 1) < b^d$, so one can often expect to do much better than the worst case. An example problem for which worst-case performance will be observed is to present all 2^d training instances for the d-bit parity concept.

4.2. Analysis of the ID3′ Variant

The ID3′ incremental tree-construction algorithm, suggested by Schlimmer and Fisher for comparison purposes, builds a new tree after each training instance is observed. The algorithm adds each new training instance to the set of observed training instances and then uses the basic ID3 tree-construction algorithm to build a new decision tree. This obvious incremental algorithm provides a basic level of complexity that other incremental algorithms must improve upon to be worth using.

Based on the above analysis of the basic ID3 tree-construction algorithm, the worst-case number of instance-count additions is

$$\sum_{i=1}^{n} i \cdot \frac{d \cdot (d+1)}{2} = \frac{d \cdot (d+1)}{2} \cdot \frac{n \cdot (n+1)}{2}$$

$$= O(n^2 \cdot d^2)$$

and the number of E-score calculations is

$$\sum_{i=1}^{n} \frac{2b^d - b^{d-1} - (d+1)b + d}{(b-1)^2} = n \cdot \left(\frac{2b^d - b^{d-1} - (d+1)b + d}{(b-1)^2} \right)$$

$$= O(n \cdot b^d) .$$

The analysis shows that for instance-count additions, the worst-case cost of building a tree with the ID3′ algorithm is polynomial in the number of training instances n. For the number of E-score calculations, the worst-case cost is linear in n.

4.3. Analysis of the ID5R Algorithm

The ID5R algorithm builds a decision tree incrementally, based on the observed training instances. The tree is revised as needed, giving a tree equivalent to the one that the basic ID3 algorithm would construct, given the same set of training instances. Information is maintained for the possible test attributes at each node so that it remains possible to change the test attribute at the node. This information consists of the positive and negative instance counts for every possible value of every attribute. Accounting for the necessary work in revising the tree makes the analysis more lengthy than for the two cases above. Therefore, the analysis is in two parts, one for the worst-case number of instance-count additions and the other for the worst-case number of E-score calculations.

4.3.1. Worst-Case Number of Instance-Count Additions. There are three components to the expense of updating the decision tree. The first is the cost of updating the instance counts when incorporating the training instance in the tree structure. Second is the cost of restructuring the tree so that the desired test attribute is at the root. Finally, there is the cost of recursively restructuring the subtrees as necessary to ensure that each has a best test attribute at its root. These three components are analyzed individually and then combined.

First, consider the worst-case cost of updating the tree for one training instance without any tree revision. Assuming an expanded tree, the cost of updating the tree at the root (level zero) is d instance-count additions because there are d possible attributes. The update procedure then traverses the corresponding value branch to level one of the tree. At level one there will be $d - 1$ instance-count additions. In general, the worst-case number of instance-count additions to update the tree for one instance is

$$\sum_{i=1}^{d} i = \frac{d \cdot (d + 1)}{2} = O(d^2) .$$

Second, consider the cost of changing the test attribute at the root of the tree. For convenience, let $pa(d)$ be the worst-case cost of changing the test attribute at a node for a tree depth d. For a tree with $d = 1$ attribute, a pull-up is never needed, giving $pa(1) = 0$. For larger values of d, the cost is derived from the definition of the pull-up process, which involves transposing the top two levels of the tree. The level-zero instance counts are already correct due to the regular update procedure. The level-two instance counts are not touched. Only the level-one instance counts for the non-test attributes need to be recomputed due to the regrouping of the level-two subtrees. For a tree with $d = 2$, there will be only one possible attribute at level one, and therefore no non-test attributes at level one, giving $pa(2) = 0$. For a tree with $d = 3$, there will be one non-test attribute at level one. For each non-test attribute there are b possible values with one positive and one negative count for each. The counts are computed by summing the counts for the corresponding level-two trees. Because there are b level-two subtrees, there will be $2b^2$ instance-count additions for each of the non-test attributes. Because there are $d - 2$ non-test attributes, the total is $2b^2(d - 2)$ instance-count additions. In the worst case, the attribute to be pulled up needs to be pulled up recursively from the deepest level of the tree. In general, $pa(d) = b \cdot pa(d - 1) + 2b^2(d - 2)$, which gives

$$pa(d) = \sum_{i=1}^{d} b^{i-2} 2b^2(d - i)$$

$$= \frac{2b^2}{(b-1)^2} (b^{d-1} - b(d-1) + (d-2))$$

$$= O(b^d) .$$

Third, consider the cost of recursively restructuring the subtrees to ensure that each has a best attribute test at its root. The process of pulling an attribute to the root of the tree leaves behind a certain amount of wreckage. The old test attribute is now at the root of each immediate subtree of the root. This is a byproduct of the tree revision, not a result of selection by E-score. For each immediate subtree except the one that will be visited when the training instance is processed at the next level, it is necessary to compute the new E-scores for the possible test attributes, pull a best attribute to the root if necessary, and then repeat the process for its subtrees.

To simplify the analysis, assume that a best test attribute is re-established in all b (instead of $b - 1$) subtrees. For convenience, let $ra(d)$ be the worst-case cost of restructuring the b subtrees at level $d - 1$. For $d = 1$, there are no possible attributes at level one, giving $ra(1) = 0$. For $d = 2$, there is only one possible attribute at the level-one subtrees, meaning that no restructuring is possible, giving $ra(2) = 0$. For $d = 3$, there are b level-one subtrees in which a best attribute must be pulled to the root. In general

$$ra(d) = b \cdot (pa(d-1) + ra(d-1))$$

$$= \sum_{i=3}^{d-1} b^{d-i} \cdot pa(i) .$$

To simplify the analysis, and since the cost of computing $pa(d)$ is $O(b^d)$, assume $pa(d) = b^d$. Then

$$ra(d) = \sum_{i=3}^{d-1} b^{d-i} \cdot b^i$$

$$= (d-3)b^d$$

$$= O(d \cdot b^d) .$$

The worst-case cost of updating the decision tree is the sum of these three components. For one instance, the worst-case number of instance-count additions is

$$\sum_{i=1}^{d} [i + pa(i) + ra(i)] = \sum_{i=1}^{d} i + \sum_{i=1}^{d} pa(i) + \sum_{i=1}^{d} ra(i) .$$

To simplify the analysis, let $pa(d) = b^d$ and $ra(d) = db^d$. Then

$$\sum_{i=1}^{d} [i + pa(i) + ra(i)] = \sum_{i=1}^{d} i + \sum_{i=1}^{d} b^i + \sum_{i=1}^{d} ib^i$$

$$= \frac{d(d+1)}{2} + \frac{db^{d+2} - (n+1)b^{n+1} + b}{(b-1)^2}$$

$$= O(d \cdot b^d) .$$

Finally, for all n instances the worst-case total number of instance-count additions is

$$O(n \cdot d \cdot b^d)$$

The analysis shows that for instance-count additions, the worst-case cost of building a tree with the ID5R algorithm is linear in the number of training instances n.

4.3.2. Worst-Case Number of E-Score Calculations. In terms of the worst-case number of E-score calculations, there are only two components to the expense of updating the decision tree. The first is the cost that is incurred when incorporating the training instance into the the tree structure. The second is the cost of recursively restructuring the subtrees as necessary to ensure that each has a best test attribute at its root. These two components are analyzed individually and then combined.

First, consider the worst-case cost of updating the tree for one instance without any tree revision. Assuming an expanded tree, the cost of updating the tree at the root (level zero) is d E-score calculations because there are d possible attributes. The update procedure then traverses the corresponding value branch to level one of the tree. At level one there will be $d - 1$ instance-count additions. Thus, as in the case for instance-count additions, the worst-case number of E-score calculations to update the tree for one instance is

$$\sum_{i=1}^{d} i = \frac{d \cdot (d+1)}{2} = O(d^2) .$$

Now consider the worst-case number of E-score calculations needed as a result of tree restructuring. During the portion of the pull-up process in which the test attribute is moved to the root, no E-scores are calculated because the tree may be revised again when re-establishing a best test attribute at the root of each of the subtrees. Therefore, one need only compute the worst-case number of E-score calculations during the re-establishment portion of the pull-up process.

For convenience, let $re(d)$ be the number of E-score calculations required while re-establishing a best test attribute for a tree of depth d. For a tree with $d = 1$, there are no possible attributes at level one, giving $re(1) = 0$. For a tree with $d = 2$, there is only one attribute at level one, so there is no need to compute the E-score, giving $re(2) = 0$. For a tree of depth $d = 3$, there are $b - 1$ level-one subtrees with $d - 1$ possible attributes,

giving $re(3) = b(d - 1)$ E-score calculations. To simplify the analysis, assume that a best test attribute is re-established in all b subtrees. Then, in general, there will be $b(d - 1)$ E-score calculations at level one, and $re(d - 1)$ E-score calculations required for the b subtrees, giving $re(d) = b((d - 1) + re(d - 1))$. This reduces to

$$re(d) = \sum_{i=3}^{d} b^{i-2}(d - i + 2)$$

$$= \frac{b}{(b - 1)^2} [2b^{d-1} - b^{d-2} - db + d - 1]$$

$$= O(b^d) .$$

Putting together the worst-case E-score cost during update and the worst-case E-score cost during re-establishment gives an instance update cost of

$$\sum_{i=1}^{d} [i + re(i)] = \sum_{i=1}^{d} i + \sum_{i=1}^{d} re(i) .$$

To simplify the analysis, let $re(d) = b^d$. Then

$$\sum_{i=1}^{d} [i + re(i)] = \sum_{i=1}^{d} i + \sum_{i=1}^{d} b^i$$

$$= \frac{d(d + 1)}{2} + \frac{b^{d+1} - b}{b - 1}$$

$$= O(b^d) .$$

Finally, for all n instances, the worst-case total number of E-score calculations is

$$O(n \cdot b^d) .$$

The analysis shows that for E-score calculations, the worst-case cost of building a tree with the ID5R algorithm is linear in the number of training instances n.

4.4. Analysis of the ID5 Algorithm

The ID5 algorithm [Utgoff, 1988] is equivalent to the ID5R algorithm except that, after restructuring a tree to bring the desired attribute to the root, the subtrees are not restructured recursively. Specifically, step 3(c)ii of the ID5R algorithm shown in Table 3 is omitted. By eliminating this step, one hopes to reduce the expense of tree manipulation. However, an important consequence of omitting this step is that the resulting tree cannot be guaranteed

to be one that the basic ID3 tree-construction algorithm would have built given the same training instances.

The analysis of the ID5 algorithm follows directly from that of ID5R above. For one instance, the worst-case number of instance-count additions is

$$\sum_{i=1}^{d} [i + pa(i)] = \sum_{i=1}^{d} i + \sum_{i=1}^{d} pa(i) .$$

As above, to simplify the analysis, let $pa(d) = b^d$. Then

$$\sum_{i=1}^{d} i + \sum_{i=1}^{d} pa(i) = \sum_{i=1}^{d} i + \sum_{i=1}^{d} b^i$$

$$= \frac{d(d+1)}{2} + \frac{b^{d+1} - b}{b - 1}$$

$$= O(b^d) .$$

For all n instances the worst-case total number of instance-count aditions is

$$O(n \cdot b^d) ,$$

which is a factor of d less expensive than for ID5R.

In terms of E-score calculations, the worst-case number for one instance is simply

$$\sum_{i=1}^{d} i = \frac{d(d+1)}{2}$$

$$= O(d^2) .$$

For all n instances, the complexity in terms of E-score calculations is

$$O(n \cdot d^2) ,$$

which is much simpler than for ID5R and independent of b, the number of possible attribute values.

4.5. Summary

Based on the four analyses, how do the algorithms compare? Table 6 summarizes the complexities, listing the algorithms in order from least to most complex according to the number of instance-count additions. The basic ID3 tree-construction algorithm is least expensive but is not incremental. The ID3′ algorithm, though incremental, is most expensive in terms

Table 6. Summary of worst-case complexities.

	Instance-count additions	E-score calculations
ID3	$O(n \cdot d^2)$	$O(b^d)$
ID5	$O(n \cdot b^d)$	$O(n \cdot d^2)$
ID5R	$O(n \cdot d \cdot b^d)$	$O(n \cdot b^d)$
ID3′	$O(n^2 \cdot d^2)$	$O(n \cdot b^d)$

of instance-count additions. Of the two remaining algorithms, the worst-case cost of building a tree for a given set of instances is less for ID5 than ID5R. Unfortunately, ID5 is not guaranteed to build the same tree as ID3. Thus, if one wants to build a decision tree incrementally, and one wants a tree that ID3 would build, the analysis indicates that the ID5R algorithm is least expensive in the worst case.

5. Empirical Behavior of ID3 Variants

This section reports empirical behavior of the variants of ID3 discussed above. Although the analyses in Section 4 indicate the worst-case complexity of building a decision tree for the algorithms, the worst-case assumptions are quite strong, and one would like to get some sense of expected behavior. In particular, tree restructuring is potentially expensive for ID5R, but it will be worthwhile if one can expect to do significantly better than the worst case. In general, empirical observation can expose aspects of behavior that are not captured in an asymptotic complexity analysis.

The rest of the section describes three experiments. The first two experiments are similar in structure, comparing the basic ID3 tree-construction routine to the incremental algorithms ID3′, ID5R, ID5, and ID4. In addition, each incremental algorithm is run with two different training strategies, giving a total of nine algorithms. The third experiment applies ID5R to Quinlan's classic chess task, comparing two different attribute selection metrics by Quinlan.

For the first two experiments, the ID3′, ID5R, and ID4 algorithms were set so that they would break ties among equally good attributes in an identical manner. In addition, the E-score values were rounded to the nearest .00001 to remove differences in E-score values due solely to different orderings of floating point operations.

5.1. Nine Algorithms on the Six-Bit Multiplexor

In the first experiment, nine variants of the ID3 algorithm were applied to learning a rule that corresponds to a simple multiplexor [Barto, 1985; Wilson, 1987; Quinlan, 1988]. In the six-bit version of the problem, there are two address bits and four data bits. The classification of an instance is equal to the value of the addressed data bit. The problem is difficult because the relevance of each of the four data-bit attributes is a function of the values of the two address-bit attributes.

Four of the algorithms are the incremental ID3′, ID5R, ID5, and ID4. Four more algorithms come about from modifying the training strategy. As suggested by Schlimmer and Fisher [1986], an alternative training strategy is to update the tree only if the existing tree would misclassify the training instance just presented. The algorithms using this strategy

Table 7. Behavior on the six-bit multiplexor task.

Alg.	Insts.	Trained	ICAs	E-scores	CPU	Nodes	Acc.
ID3′	126	126	156,210	6,052	17.1	38.9	100.0
ID5R	126	126	4,319	3,556	4.1	38.9	100.0
ID5	148	148	4,633	2,887	3.2	43.3	100.0
ID4	3,770	3,770	67,125	56,348	61.4	39.6	100.0
$\widehat{\text{ID3}}'$	253	33	9,095	1,143	1.8	33.0	100.0
$\widehat{\text{ID5R}}$	253	33	1,377	1,405	1.5	33.0	100.0
$\widehat{\text{ID5}}$	281	37	1,502	660	1.1	38.0	100.0
$\widehat{\text{ID4}}$	30,000	14,849	196,183	130,515	150.9	5.6	50.0
ID3	64	64	1,176	66	0.2	51.0	100.0

are denoted $\widehat{\text{ID3}}'$, $\widehat{\text{ID5R}}$, $\widehat{\text{ID5}}$, and $\widehat{\text{ID4}}$, respectively, and thus are called the "hat" versions. The ninth algorithm is the basic ID3 tree-construction routine applied nonincrementally to the entire training set of sixty-four instances.

For each of the eight incremental variants, the instances were presented in random order, with replacement, until the tree gave a correct classification for all sixty-four instances in the instance space, or until 30,000 instances had been presented. Each variant was run twenty times, and the results were averaged. The identical sequence of twenty initial random number seeds was used for the twenty runs of the eight incremental algorithms.

Table 7 shows seven measurements for the nine algorithms. For the eight incremental algorithms, the values are averaged over the twenty runs. The first measurement (Insts.) is the number of training instances presented to the algorithm. The third column (Trained) is the number of instances that were actually used to update the tree. For the hat variants, this number is usually less than the number presented, because only instances that would be misclassified by the current tree are used to update the tree. The fourth column shows the average number of instance-count additions, and the fifth column shows the average number of E-score computations. Total cumulative cpu seconds[4] spent in processing presented training instances is shown in column six. The size of the final decision tree is shown in the column labeled "Nodes," as measured by the total number of decision and answer nodes. Finally, the last column shows the percentage of the sixty-four possible instances that were classified correctly by the final decision tree.

The results show that ID5R is much less expensive than ID3′ for this problem, although for $\widehat{\text{ID5R}}$ and $\widehat{\text{ID3}}'$, the difference is less pronounced. The smaller number of instances actually used to update the trees makes $\widehat{\text{ID3}}'$ comparable. However, the greater expense of the ID3′ algorithm is evident for the larger number of instances used by ID5R and ID3′. Note that the hat variants find smaller trees than the plain versions.

The remaining five algorithms are included for comparison purposes, but each are flawed as an incremental version of ID3. The ID5 variant will build a consistent decision tree for a given set of training instances, but the tree is not necessarily the same that ID3 would build. The experiment shows that ID5 needs to examine more training instances than ID3 or ID5R in order to induce a tree that is correct for the entire training set. The ID5 algorithm is less expensive than ID3′ and ID5R, but finds larger trees.

The ID4 algorithm is not guaranteed to find the same tree as ID3, nor is it guaranteed to find a consistent decision tree. The experiment shows that the algorithm requires a large

number of training instances, making its overall expense very high in comparison to the others. The final tree is also large by comparison. The $\widehat{\text{ID4}}$ algorithm never found a consistent decision tree within the cutoff of 30,000 training instances. The average final tree size of only 5.3 nodes gives little hope that the algorithm would find a consistent tree with further training. Even if it did, the expense is already excessive in comparison to the other three hat algorithms. Why does ID4 succeed where $\widehat{\text{ID4}}$ fails? The hat version trains only on the instances that are misclassified by the current tree. When the tree changes, the set of instances that it will misclassify also changes. The new set of instances that the tree will misclassify may lead to yet another change of test attribute at the node. This changing of the training set does not work well for the ID4 algorithm, resulting in thrashing.

Finally, the basic ID3 tree-construction algorithm is not incremental. Although the algorithm was much more efficient than the incremental variants, it also built the largest tree. Of course, if ID5R were to train on the entire set, it would build an identical tree. The issue is why training on a larger set of instances tends to produce a larger decision tree. This phenomenon was not expected and remains unexplained. Apparently, the probability distributions inferred from incorporating only misclassified training instances lead to better attribute selection than distributions inferred from training sets that are larger than necessary, such as the entire training set.

5.2. Nine Algorithms on a Simplified Chess Problem

The second experiment is similar to that of the multiplexor described above, but with two differences. First, the domain is the chess task used as a test problem by Schlimmer and Fisher [1986], which is a simplified version of Quinlan's [1983] chess task. In the simplified version, the problem is to learn a rule that decides whether a chess position with White having King and Rook versus Black having King and Knight is lost in two-ply, with black on move. Each training instance is a chess position described in terms of sixteen three-valued attributes.

The second difference is that the set of test instances is drawn independently from the set of training instances. For training purposes, a single fixed set of 250 training instances was drawn at random. For each of the eight incremental variants, the instances were presented in random order, with replacement, until the tree gave a correct classification for all 250 training instances, or until 30,000 instances had been presented. This training strategy produces a learning task with a terminating condition that does not depend on a fixed number of training instances being presented. This is important because some of the variants require more training instances than the others, and all the variants have differing processing cost.

For testing purposes, a second fixed set of 250 test instances was also drawn at random. Classification accuracy was measured as the proportion of the test instances correctly classified. Each variant was run twenty times, and the results were averaged. For the non-incremental ID3, a single tree was built, based on the 250 training instances. The same sequence of twenty initial random number seeds was used for the twenty runs of the eight incremental algorithms.

Table 8 shows seven measurements for the nine algorithms. The first six are the same as those in Table 7, whereas the seventh reports the accuracy of the final tree on the independent test set of 250 instances. The results of this experiment are similar to those of

Table 8. Behavior on the simplified chess task.

Alg.	Insts.	Trained	ICAs	E-scores	CPU	Nodes	Acc.
ID3′	195	195	737,932	5,357	74.1	10.0	99.6
ID5R	195	195	14,476	10.097	20.9	10.0	99.6
ID5	195	195	13,642	9,538	8.1	10.0	99.6
ID4	236	236	7,940	4,451	6.2	10.0	99.6
$\widehat{\text{ID3}}'$	226	12	1,827	328	0.6	10.0	99.6
$\widehat{\text{ID5R}}$	226	12	2,396	2,686	2.7	10.0	99.6
$\widehat{\text{ID5}}$	236	12	1,544	450	1.1	10.0	99.6
$\widehat{\text{ID4}}$	27,058	1,920	63,219	34,692	59.2	5.3	93.3
ID3	250	250	7,045	46	0.8	10.0	99.6

the first. The ID5R algorithm was less expensive than the ID3′ algorithm for this problem. However, the small number of training instances needed for $\widehat{\text{ID3}}'$ and $\widehat{\text{ID5R}}$ made $\widehat{\text{ID3}}'$ less expensive. For this simple learning problem, $\widehat{\text{ID3}}'$ even beat $\widehat{\text{ID5}}$. The ID4 algorithm had no trouble finding a consistent concept description, and with less expense than each of ID3′, ID5R, and ID5. The hat variants are generally less expensive, but again $\widehat{\text{ID4}}$ had trouble finding a consistent tree. It was cut off at 30,000 instances in eighteen of the twenty runs.

The attributes that are provided for a learning task have a profound effect on the difficulty of that task. The attributes for the multiplexor task describe only the bit values. In contrast, the attributes for the simplified chess task describe more than the board position; they measure relationships among the pieces that are predictive of winning or losing. The result is that the chess rule is easier to learn than the multiplexor rule, even though the space of instance descriptions for the chess task (sixteen three-valued attributes) is larger than that of the multiplexor task (six two-valued attributes).

5.3. Quinlan's Chess Task

In order to observe the ID5R algorithm's behavior on a larger problem, it was applied to Quinlan's [1983] classic chess task, in which there are 551 available training instances, each described by 39 binary attributes. The algorithm was run five times for each of two attribute selection strategies. The first approach selects an attribute with the lowest E-score. The second chooses an attribute with the highest gain ratio [Quinlan, 1986], which compensates for a bias in the E-score metric toward attributes with more possible values. Using the definitions in Section 2, the gain-ratio criterion is defined

$$gain\text{-}ratio(a_i) = \begin{cases} \dfrac{I(p, n) - E(a_i)}{IV(a_i)} & \text{if } I(p, n) - E(a_i) \geq \bar{g} \\ -\infty & \text{otherwise} \end{cases}$$

with

$$IV(a_i) = -\sum_{j=1}^{|V_i|} \frac{p_{ij} + n_{ij}}{p + n} \log \left(\frac{p_{ij} + n_{ij}}{p + n} \right)$$

and

$$\bar{g} = \frac{1}{|A|} \cdot \sum_{i=1}^{|A|} (I(p, n) - E(a_i)) .$$

Table 9 shows the average values for six measures using the two attribute selection metrics. As expected, the gain-ratio criterion is better in terms of finding smaller trees, and it leads to fewer instance-count additions and E-score calculations. However, the gain-ratio criterion requires greater total cpu time because of the additional expense in computing one $I(p, n)$ function and one or more $IV(a_i)$ functions when picking a best attribute.

A surprise is that the average-case tree size is comparable to the previously known smallest tree size of 125 for the gain-ratio critierion [Quinlan, personal communication] and 175 for the lowest E-score criterion [Quinlan, 1986]. Indeed, for the gain-ratio criterion, one of the five trees was only 117 nodes, now the smallest known decision tree for this concept. Given that ID5R builds the same tree as ID3 for a given set of instances, the only difference lies in the selection of instances used to build the tree [Utgoff, 1989]. One can surmise that selecting training instances one at a time, based on inconsistency with the current tree, leads to a smaller tree than selecting instances many at a time, as is done in the complete ID3 learning algorithm.

Figure 1 plots the classification accuracy of the 117 node tree on all 551 instances after each training instance was added to the tree, and Figure 2 plots the size of the decision tree after each training instance was observed. Although tree size is generally increasing, there can be significant differences from one training instance to the next. Consider the change in tree size after the 98th, 99th, and 100th training instance. The tree size changes

Table 9. Behavior of ID5R on Quinlan's [1983] chess task.

Selection	Insts.	Trained	ICAs	E-scores	CPU	Nodes
Gain Ratio	3,340	132.6	308,597	551,237	1,410	125.0
E-score	4,070	183.0	761,534	1,458,722	1,292	170.6

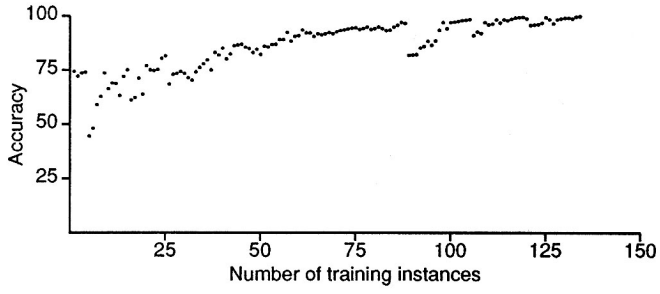


Figure 1. Change in classification accuracy on the chess task for the ID5R algorithm.

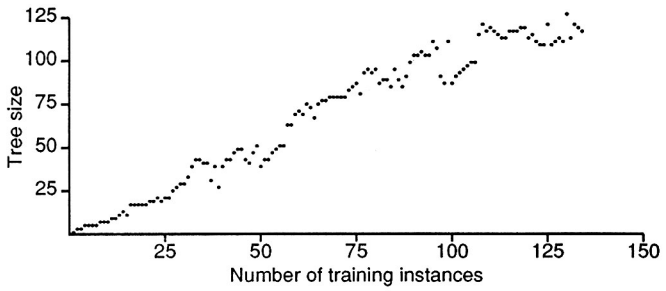


Figure 2. Change in tree size for the chess task for the ID5R algorithm.

from 87 nodes to 111 and back to 87. It is not clear whether this kind of variation is characteristic of the instance selection strategy or is specific to the chess domain. In either case, the large variation does indicate that the quality of the learning, as measured by resulting tree size, can be very sensitive to the selected training instances.

6. Future Work

Several improvements for the ID5R algorithm are possible. For example, a significant expense for the algorithm is computation of E-scores. The dominating cost in computing these scores is the evaluation of the log function in the computation of an $I(p, n)$. The efficiency of the algorithm can be improved by storing the result of each $I(p, n)$ in the tree structure. Because only one positive or negative count for one attribute-value pair is changed when incorporating an instance, only one I computation would be needed in order to compute an E-score.

An issue that was not addressed here is learning from noisy instances. Two extensions to the ID5R algorithm that would let it deal with noise come to mind. First, if a new training instance has been encountered previously with a different classification and the old instance has been saved in the tree, one would remove the older instance, add the newer instance, and then revise the tree as necessary. This modification would let the system handle concept drift. Alternatively, one could retain the instance that results in a smaller tree or that would result in a test attribute with the lower lowest E-score. The second extension would include the χ^2 test for dependence. If splitting on a given attribute cannot be justified then do not allow it to be the test attribute at a node. If no split can be justified, then the decision tree rooted at the node will need to be an unexpanded tree. This will work if the definition of an unexpanded tree allows any set of training instances. This modification is trivial, but it differs from the definition of unexpanded tree given above.

The problem of classifying instances for which certain attribute values are missing has received some attention [Quinlan, 1986], and ID5R could be modified to deal with this issue by restructuring the tree during classification. One would select the best attribute for which the value was available in the instance, restructure the tree if necessary so that the attribute was the test attribute, and then descend to the next level. If all branches corresponding to the available attribute values in the instance were traversed without determining

the class, then the instance counts in the last node could be used (e.g., by majority vote) to decide the classification.

The most important open problem in finding good decision trees is the problem of defining a good set of attributes. For example, the chess task was easier than the multiplexor because the provided chess attributes are better at capturing the regularity in the domain. If an additional attribute were included for the multiplexor that specified the address indicated by the address bits, then the multiplexor task would be significantly easier. The problem of discovering a good set of attributes needs attention and is the principal issue that will be explored as this research continues.

7. Conclusions

The ID5R algorithm builds a decision tree incrementally, by restructuring the tree as necessary so that a best test attribute is at each decision node. Although the algorithm retains training instances within the tree structure, the instances are used only for restructuring purposes; they are not reprocessed. The algorithm builds the same tree as the basic ID3 tree-construction algorithm, given the same training instances, assuming that ties for equally good attributes are broken identically. In empirical tests, the ID5R algorithm is less expensive than the brute-force ID3′ algorithm.

An earlier incremental algorithm, ID4, is incapable of learning certain classes of concepts that ID3 can learn. Furthermore, for concepts that ID4 can acquire, empirical testing shows that the algorithm can be expensive in comparison to ID5R. Moreover, the strategy of training only on instances that the tree would misclassify is detrimental to ID4, reducing the class of concepts that it can learn.

For ID5R, this training strategy generally leads to a smaller final decision tree than the strategy of ID3, which accumulates misclassified instances many at a time. This phenomenon of finding smaller trees was observed both for the multiplexor task and for Quinlan's chess task. With this training strategy, ID5R found the smallest known tree to date for the chess domain. The ID5R algorithm can be viewed as an incremental variant of the ID3 tree building routine. The latter will find the identical tree if the initial window contains just one instance and the window is grown by one instance at a time. Until the creation of the incremental ID5R algorithm, such an approach has been prohibitively expensive.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. IRI-8619107, a General Electric Faculty Fellowship, and by the Office of Naval Research through a University Research Initiative Program, under contract number N00014-86-K-0764. The several discussions with Jeff Schlimmer on ID3 and ID4 were very helpful. In addition, he provided an instance generator for the simplified chess task and made useful comments during the review process. Ross Quinlan made many helpful observations and suggestions, and provided his chess data. Pat Langley provided extensive comments, particularly with regard to clarifying the presentation. I am indebted to Maria Zemankova for pointing out the work of Robin Cockett, and to Robin Cockett for the discussions about revising decision trees. The presentation benefitted from comments by Sharad Saxena, Margie Connell, Jamie Callan, Peter Heitman, Kishore Swaminathan, and Dan Suthers.

Notes

1. See Watanabe [1985] for a historical perspective on applying the notion of physical entropy to information processes.
2. The tree transposition in step 2b is equivalent to an operation described by Cockett [1987].
3. The ID4 algorithm has been omitted because a concept may be unlearnable by ID4, making the worst-case cost of building a correct tree undefined.
4. All algorithms were implemented in Common Lisp and run on a SUN 4/110.

References

Barto, A. G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4, 229–256.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth International Group.

Buchanan, B. G., and Mitchell, T. M. (1978). Model-directed learning of production rules. In D. A. Waterman and F. Hayes-Roth (Eds.), *Pattern-directed inference systems*. New York: Academic Press.

Cockett, J. R. B. (1987). Discrete decision theory: Manipulations. *Theoretical Computer Science*, 54, 215–236.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2, 139–172.

Gallant, S. I. (1988). Connectionist expert systems. *Communications of the ACM*, 31, 152–169.

Lewis, P. M. (1962). The characteristic selection problem in recognition systems. *IRE Transactions on Information Theory*, IT-8, 171–178.

Michalski, R. S., and Chilausky, R. L. (1980). Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *Policy Analysis and Information Systems*, 4, 125–160.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. San Mateo, CA: Morgan Kaufmann.

Mitchell, T. M. (1978). *Version spaces: An approach to concept learning*. Doctoral dissertation, Department of Electrical Engineering, Stanford University, Palo Alto, CA.

Moret, B. M. E. (1982). Decision trees and diagrams. *Computing Surveys*, 14, 593–623.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1, 81–106.

Quinlan, J. R. (1988). An empirical comparison of genetic and decision-tree classifiers. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 135–141). Ann Arbor, MI: Morgan Kaufmann.

Schlimmer, J. C., and Fisher, D. (1986). A case study of incremental concept induction. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 496–501). Philadelphia, PA: Morgan Kaufmann.

Schlimmer, J. C., and Granger, R. H., Jr. (1986). Incremental learning from noisy data. *Machine Learning*, 1, 317–354.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27, 379–423.

Utgoff, P. E. (1988). ID5: An incremental ID3. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 107–120). Ann Arbor, MI: Morgan Kaufmann.

Utgoff, P. E. (1989). Improved training via incremental learning. *Proceedings of the Sixth International Workshop on Machine Learning*. Ithaca, NY: Morgan Kaufmann.

Vere, S. A. (1980). Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence*, 14, 138–164.

Watanabe, S. (1985). *Pattern recognition: Human and mechanical*. New York: Wiley and Sons.

Wilson, S. W. (1987). Classifier systems and the animat problem. *Machine Learning*, 2, 199–228.