ID5: An Incremental ID3

PAUL E. UTGOFF (UTGOFF@CS.UMASS.EDU)

Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003 U.S.A.

Abstract

This paper describes ID5, an incremental algorithm that produces decision trees similar to those built by Quinlan's ID3. The principal benefit of ID5 is that new training instances can be processed by revising the decision tree instead of building a new tree from scratch. ID3, ID4, and ID5 are compared theoretically and empirically.

1. Introduction

This paper describes ID5, an incremental algorithm for building a decision tree similar to that which Quinlan's ID3 (Quinlan, 1983; Quinlan, 1986) would build. A decision tree represents a sequential procedure for deciding the class membership of a given instance. The decision tree concept representation and the ID3 algorithm for inducing a decision tree have performed well on a variety of learning tasks.

Quinlan's algorithm builds a decision tree by choosing a good *test attribute* that partitions the instances into smaller sets for which decision subtrees are constructed recursively. To determine which attribute should be the test attribute for a node, the algorithm applies an information-theoretic measure (Lewis, 1962; Quinlan, 1983), here called $INFO$. The value returned by $INFO$ is an estimate of the average amount of decision making that would need to be done in order to determine the classification of an instance. An attribute with the lowest $INFO$ value is selected as the test attribute. $INFO$ is a function of the number of positive and negative instances seen for each value of the given attribute.

The ability of ID3 to construct decision trees that are efficient classifiers and that generalize well is attractive. For learning problems in which a database of instances is available and is not likely to change, ID3 is a good choice for building a classification rule. However, for problems in which new instances are expected to become available on a regular basis, it would be far preferable to accept instances incrementally, without needing to build a new decision tree from scratch each time.

Schlimmer and Fisher constructed ID4 (Schlimmer & Fisher, 1986), which incrementally builds a decision tree similar to that which ID3 would build. Instead of building a decision tree from a batch of instances, ID4 updates a decision tree based on each individually observed instance. The algorithm for handling a training instance is:

1. For each potential test attribute at a node, update the count of positive and negative instances for that attribute, and the positive and negative count for the observed value of that attribute.

2. If the lowest INFO measure for a non-test attribute is lower than that of the current test

attribute, then discard the subtrees below the current test attribute. Redefine the test attribute to be A_{best}.

3. If no positive instances have been observed at the node, return the decision tree "−".
4. If no negative instances have been observed at the node, return the decision tree "+".
5. Update the decision tree below the value of A_{best} that occurs in the instance description. Return the decision tree with test attribute A_{best}.

The algorithm offers an approach to incremental learning of ID3-type decision trees. A potential drawback of the algorithm is that all or part of a decision tree will be discarded whenever it is determined that the test attribute should be replaced with a better attribute. The effect of such replacements is that training effort is lost. Schlimmer and Fisher indicate that the decision tree should eventually stabilize. More training may be needed, because the benefit of previous training can be lost, but the overall expense of training is considered reduced when compared to ID3.

2. ID5

ID5 builds on the idea in ID4 that one can maintain positive and negative instance counts of every attribute that could be a test attribute for the decision tree or subtree. ID5 differs from ID4 in its method for replacing the test attribute. Instead of discarding the subtrees below the old test attribute, ID5 reshapes the tree by pulling the test attribute up from below. The advantage is that the positive and negative instance counts can be recalculated during the tree manipulations, without reprocessing the instances.

The algorithm for handling an instance is:

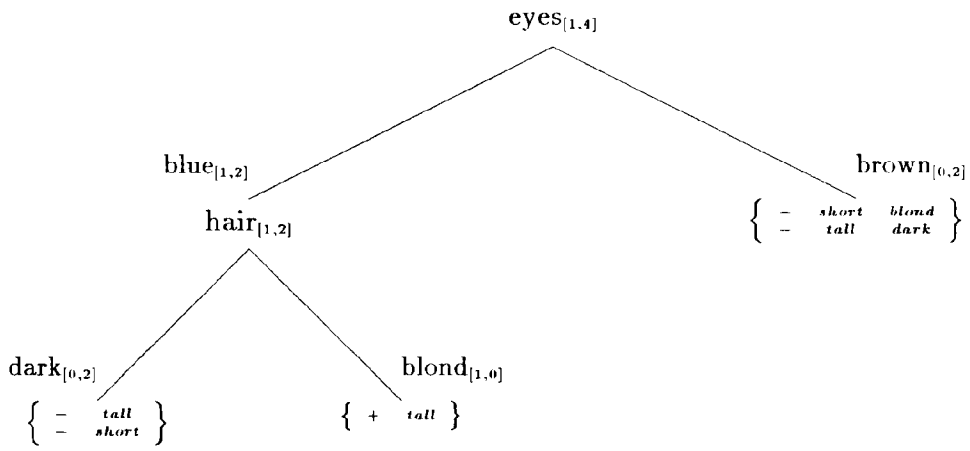
1. For each potential test attribute at a node, update the count of positive and negative instances for that attribute, and the positive and negative count for the observed value of that attribute.
2. If the lowest INFO measure for a non-test attribute is lower than that of the current test attribute, then reshape the tree by pulling the desired test attribute A_{best} up from below.
3. If only positive instances or only negative instances have been observed at the given node, then store the rest of the training instance. Stop.
4. Recursively update the decision tree below the value of A_{best} in the instance description.

Pulling an attribute up to be the new test attribute requires a process that restructures the tree without losing training information. This requires that the portion of each instance description not implicit in the decision tree be stored at each leaf so that a pull-up is possible. The algorithm for pulling up an attribute is:

1. Recursively pull the attribute to the root of each immediate subtree. (Expand any set of instances at a subtree to form a proper decision tree as necessary.)
2. For each value branch of each subtree, construct a new decision tree by pushing the old test attribute below the new test attribute. This results in a set of subtrees, each with the desired test attribute at the root.

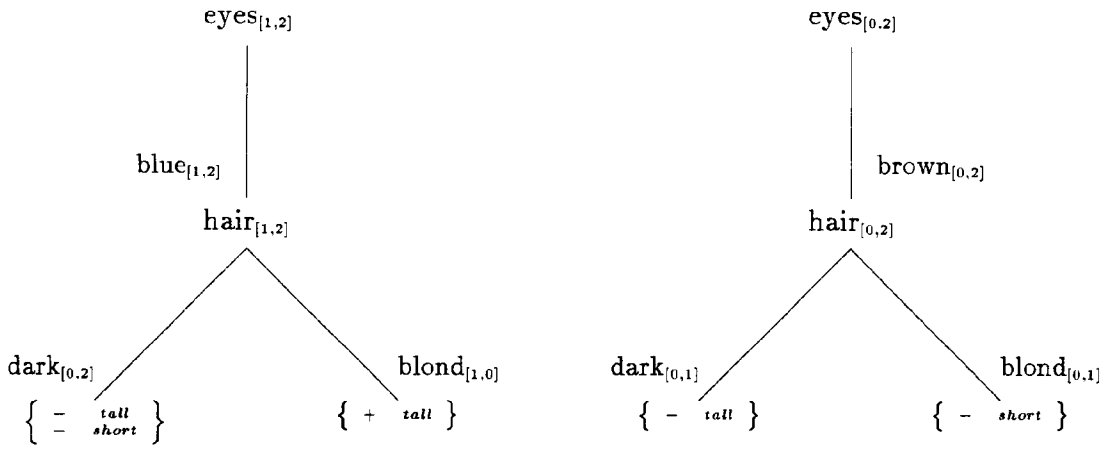
3. Merge the subtrees, so that the desired test attribute is at the root of a single decision tree.

Consider an illustration of the pull-up process. Throughout the discussion, the notation $[p, n]$ indicates the count of positive and negative instances respectively. Assume the current decision tree is:

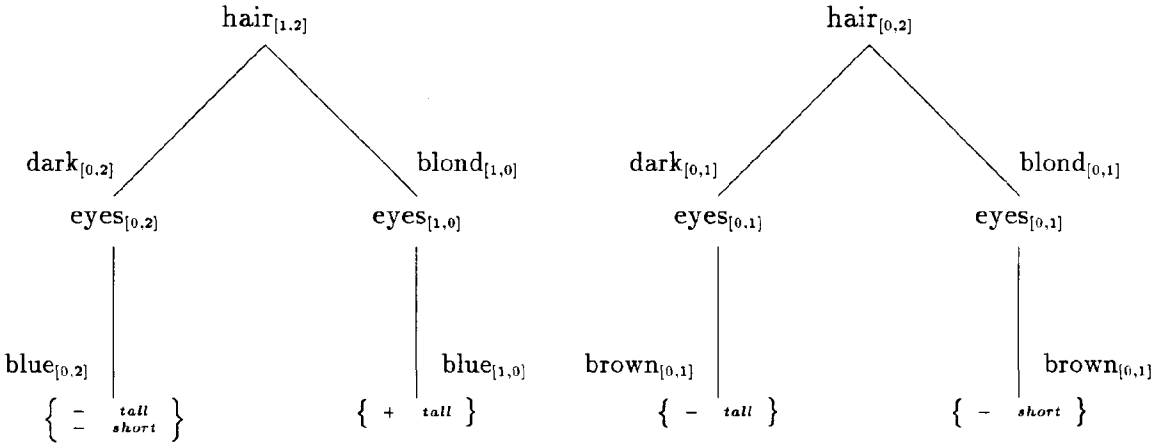


Though the tree is correctly formed, the updated $INFO$ values (not shown) for the attributes indicate that "eyes" no longer has the lowest $INFO$ value and is therefore no longer the best test attribute. Instead, "hair" should be the test attribute, so it needs to be pulled to the root of the tree.

Step 1 of the pull-up process expands the instances below "brown" into a proper tree and then does nothing further, because "hair" is already the test attribute of every subtree of "eyes". The objective of step 1 is to bring the best test attribute, in this case "hair", to the root of every subtree. This is a setup for step 2. Next, step 2 splits the tree into a set of trees, one for each subtree of "eyes", producing:

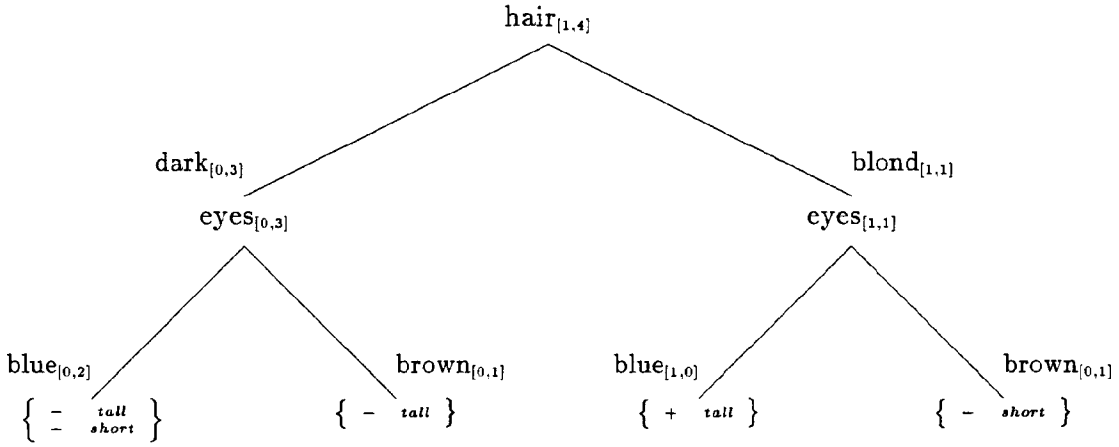


Note that for each tree, since there is a single branch below the root ("eyes"), the positive and negative instance counts for the root can be taken directly from the root of the single subtree ("hair"). Now, for each tree, step 2 proceeds to push the test attribute "eyes" down the branch of its single subtree, producing:



The result is a set of trees with the best test attribute at each root. This is a setup for step 3. Note that for each tree, since the old test attribute ("eyes") was pushed down each value branch for the new test attribute ("hair"), the positive and negative instance counts for the old test attribute can be taken directly from the counts for the value branch below the new test attribute. For example, in the left tree above, "$\text{eyes}_{[0,2]}$" and "$\text{blue}_{[0,2]}$" are defined directly by taking the counts from "$\text{dark}_{[0,2]}$".

Step 3 now merges the set of trees, all of which have "hair" at the root, making:



The merging operation is able to compute the positive and negative instance counts by simple addition of existing counts. For example, the "$dark_{[0,2]}$" branch and the "$dark_{[0,1]}$" branch are merged into a single branch with counts "$dark_{[0,3]}$". The fact that the old test attribute is at the root of every subtree greatly simplifies the merging process. The subtrees for each combination (quadruple) of new test attribute, new test attribute value, old test attribute, and old test attrbute value, are effectively regrouped by the pull-up process, meaning that the merge operation need never examine a subtree below the old test attribute, which is always at level 2 in every tree.

A second point to note is that the subtrees below the new test attribute are not guaranteed to have the best test attribute at the root. This is because the old test attribute, which is now the test attribute of each subtree, was not selected by choosing the best test attribute according to the $INFO$ function. Rather, the test attribute was determined as a byproduct of the pull-up step. To assure that each subtree has the best test attribute at its root, it would be necessary to re-establish the best test attribute for each subtree recursively. This is simple enough to do, but it is not done routinely in ID5 because it is expensive to do on a regular basis. Furthermore, it is not critical, unless one needs an ID3-equivalent tree immediately, because subsequent training that causes a subtree to be visited will bring the best test attribute to the root of that subtree.

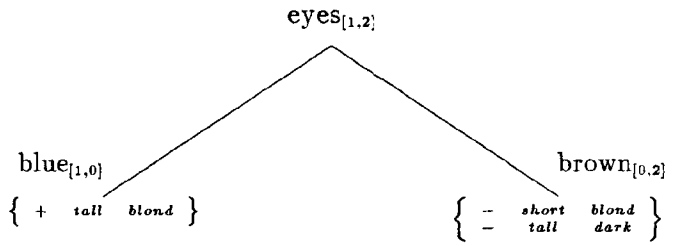
A third point to note is that the subtree below value "dark" can be contracted to a set of instances. This is a simple operation, but there is no need to incur the expense unless space is in demand.

Now observe how ID5 performs on the 8 instance example from (Quinlan, 1983). Each instance is described as a conjunction of three attribute-value pairs, using the attributes: *height*, *hair* color, and color of *eyes*. The instances are:

class	height	hair	eyes
−	short	blond	brown
−	tall	dark	brown
+	tall	blond	blue
−	tall	dark	blue
−	short	dark	blue
+	tall	red	blue
−	tall	blond	brown
+	short	blond	blue

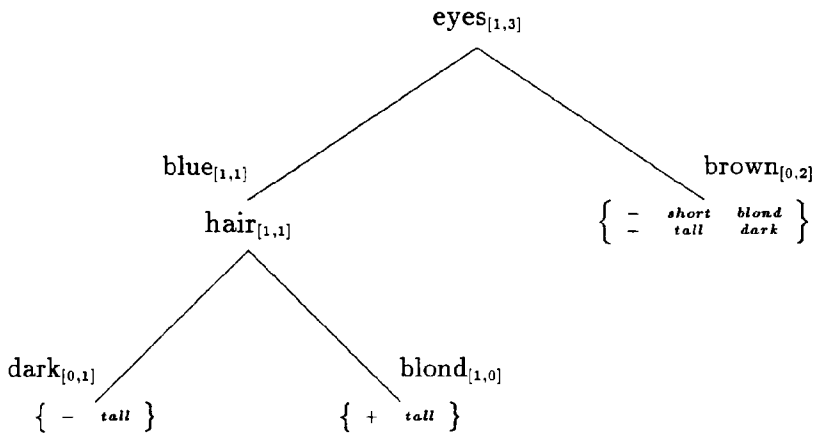
The first two instances are both negative, so they are simply stored at the root. A proper decision tree is not yet formed. ID5 only builds a tree at a node for which an attribute test is needed. When all instances have the same classification, a test attribute is not needed, and the decision is indicated by the instance classification, in this case "−".

The third instance (+,tall,blond,blue) is positive, so a tree is formed:



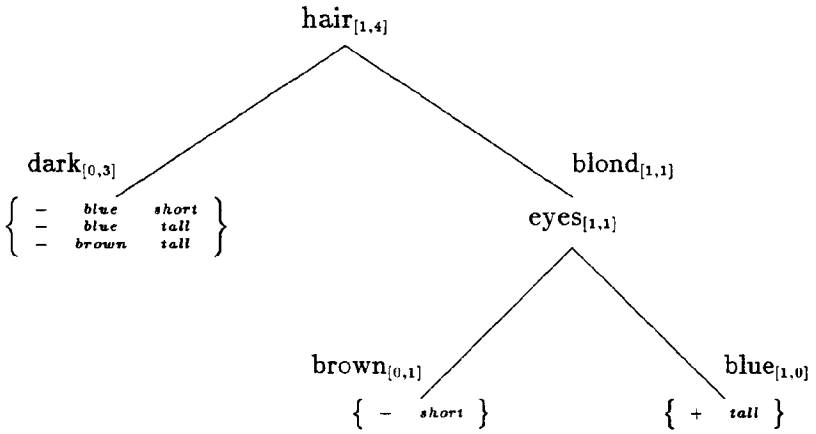
This is the same tree that ID4 would have at this point.

The fourth instance (−,tall,dark,blue) leads to:



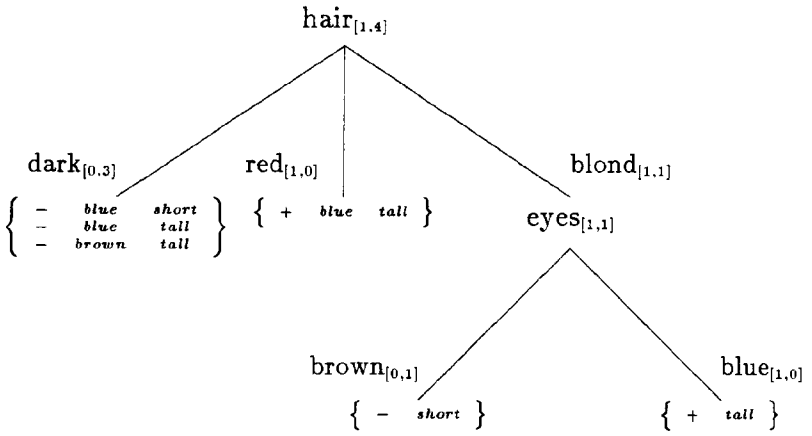
Note that the tree has "hair" as the test attribute below "blue". Because ID5 retained the instances for this subtree, it was able to determine that "hair" is the best test attribute.

The fifth instance (−,short,dark,blue) causes the tree to be restructured, producing:



This restructuring is the case that was used above to illustrate the pull-up process. Here, the instances below "dark" have been contracted. Note that at this point in the training, ID4 would have not have the subtree below "blond".

The sixth instance (+,tall,red,blue) updates the tree to:



At this point, the decision tree happens to classify all eight instances correctly. From the current state of the tree, processing the remaining two instances will update the various positive and negative counts, but will not revise the structure of the tree.

For this example, ID5 built the same decision tree that ID3 built. To guarantee that ID3 and ID5 will always build the same tree, two changes would be required. First, as mentioned above, it would be necessary to re-establish the best test attribute for each subtree below a test attribute that was just pulled up. Second, it would be necessary to insist that ties for equally good test attributes be resolved identically. In practice, it would be a waste of time to make such an imposition, because the ID3 algorithm does not require a tie-breaking algorithm.

3. Analysis

It is important to consider how much training is required to achieve a specific level of performance, and how costly that training is. Schlimmer and Fisher (Schlimmer & Fisher, 1986) provided a theoretical analysis of ID3 and ID4. A similar analysis is given here for ID3, ID4, and ID5.

One measurement of training effort is the total number of instances that must be examined or re-examined. A finer grained measurement is the number of attributes examined or re-examined. Coupled with the number of instances examined is the number of attributes in the instance. These are considerations for all of ID3, ID4, and ID5. Further considerations for ID4, and ID5 are the expected frequency of revising a test attribute and the associated cost of such a revision, both in terms of execution cost and in terms of impact on the need for further training. These are highly dependent on the learning task at hand and are therefore difficult to assess.

3.1 ID3

For ID3, unless all instances are of the same classification, all $|A|$ attributes of all $|I|$ instances are examined. Because the subtrees are derived from a partition of I, as many as $|I|$ are examined on the second level and again on each successive level of the

tree. Less than $|I|$ instances are examined when a subtree terminates with a "+" or a "−". Hence, the total number of instances examined is $O(|I| \cdot |A|)$. For each instance on level one, $|A|$ attributes are examined. For each instance on level two, $|A| - 1$ attributes are examined. Thus, the total number of attribute examinations is $O(|I| \cdot |A|^2)$. If ID3 is used to build a new decision tree after each observed training instance, i.e. a crude incremental algorithm, the total number of attribute examinations is:

$$\sum_{i=1}^{|I|} O(i \cdot |A|^2) = O(|I|^2 \cdot |A|^2)$$

3.2 ID4

For ID4, each instance is examined once at the root, and once at each lower level of the tree. As with ID3, lower levels will exist only where an attribute test is necessary. Still, the number of times the instance is examined is $O(|A|)$. To examine all $|I|$ instances, the total number of instance examinations is $O(|I| \cdot |A|)$. As with ID3, the number of attributes examined for each instance is $|A|$ on level 1, $|A|-1$ on level 2, down to 1 on level $|A|$. Thus, for each instance examined, $O(|A|^2)$ attributes are examined. For all $|I|$ instances, the total number of attribute examinations is $O(|I| \cdot |A|^2)$.

A single pass over the instances does not necessarily cause an ID3 equivalent tree to be constructed. One needs to ask how many total instances, or passes over the given set of instances, are needed to achieve the same result as ID3. Schlimmer and Fischer expected that each pass over the instances would stabilize the next level of the tree. Thus, one could need as many as $|A|$ passes over the $|I|$ instances. The total number of instance examinations would be $O(|I| \cdot |A|^2)$ and the total number of attribute examinations would be:

$$O(|I| \cdot |A|^3)$$

This is usually better than building a new tree with ID3 after each instance, but usually worse than using ID3 to build a tree from all $|I|$ instances at once.

It is important to consider the expense of discarding subtrees when ID4 chooses a new test attribute. How much training effort is lost with such a strategy? Consider revision of the test attribute at the root of the decision tree. All subtrees would be discarded. This could be considered acceptable because the first pass through the instances will result in the same test attribute that ID3 would pick. The same argument would apply at each subtree. However, there is no guarantee that the test attribute will remain the best choice after every instance in I that ID4 will see. If that is not the case, then ID4 will not stabilize because it will repeatedly revise the test attribute. This behavior can be observed when training with ID4 on the following set of instances:

class	a	b	c
+	0	0	0
−	0	0	1
−	0	1	0
+	0	1	1
−	1	0	0
+	1	0	1
+	1	1	0
−	1	1	1

3.3 ID5

For ID5, the number of times an instance is examined is $O(|A|)$, as with ID4. Similarly, to examine all $|I|$ instances, the total number of instance examinations is $O(|I| \cdot |A|)$. As with ID3 and ID4, for each instance examined, $O(|A|)$ attributes are examined. For all $|I|$ instances, the total number of attribute examinations is $O(|I| \cdot |A|^2)$.

One pass over the instances is sufficient to build a tree that correctly classifies all the instances. However, the expected frequency of revising the test attribute at a node, and the cost of such revision must be taken into account.

Regarding the expected frequency of revising the test attribute, it is dependent on the attributes used to describe instances, and the nature of the concept being learned. It should be noted that ID5 and ID4 revise a test attribute in exactly the same cases.

Regarding the cost of revision, this is the cost of the pull-up process. This is difficult to state because there is no indication of where the new test attribute will be found lower in the tree. However, as soon as the new test attribute has been pulled up to the root of each subtree of the old test attribute, the remaining cost of the tree reshaping is proportional to the product of the number of possible values of the old test attribute and the number of possible values of the new test attribute. Intuitively, one would guess that the new test attribute would be found generally near the root of the decision tree, meaning that few recursive pull-ups would be needed, but this has not been established nor has it yet been measured empirically.

4. Experiments

The three algorithms were implemented and performance was compared empirically. For ID3, a crude incremental version was used; after each training instance, build a new decision tree from the set of all instances observed thus far. Two experiments were performed, one on a multiplexor task, the other on a preference predicate task. As in Schlimmer & Fisher, versions in which only necessary training[1] is performed were also tested. Here, these versions are called $\widehat{ID3}$, $\widehat{ID4}$, and $\widehat{ID5}$. Schlimmer and Fisher performed similar experiments for ID3 and ID4 using a learning task from the game of chess. A useful measure they illustrated was cumulative training cost per instance

[1] i.e. only if the current decision tree misclassifies the training instance

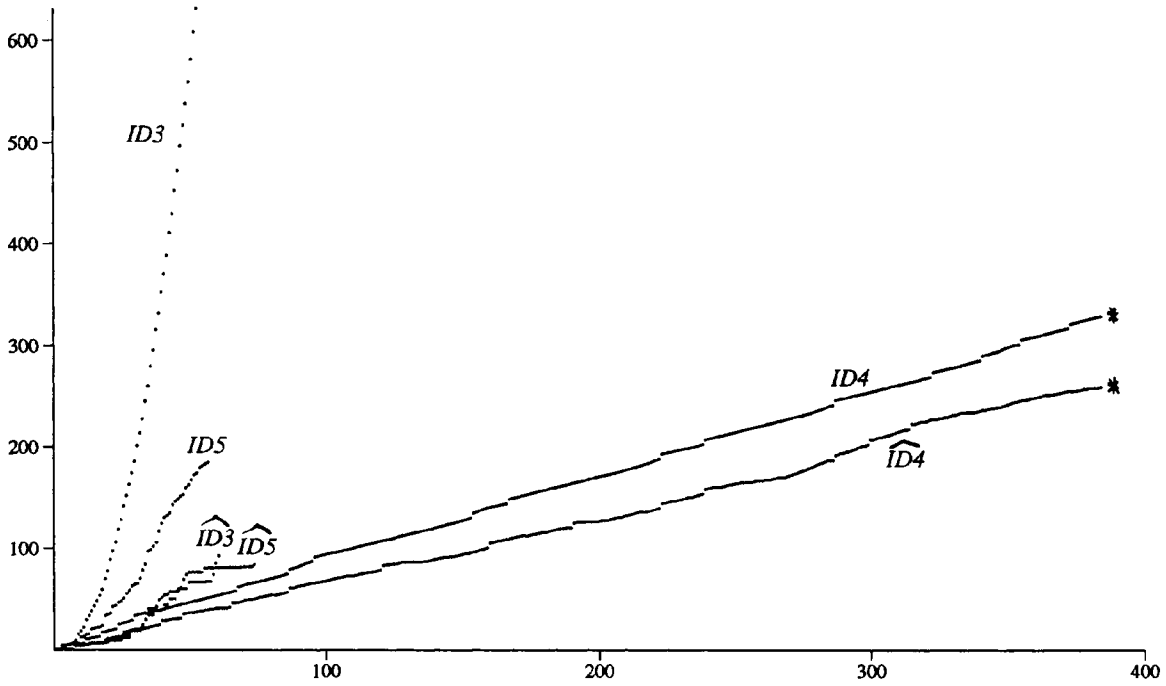


Figure 1: Training cost (y axis) vs. events (x axis)

processed. An additional measure included below is classification performance on the entire training set after each instance is processed.

4.1 Multiplexor

For this experiment, the database I consisted of 64 instances from a commonly studied multiplexor task. Each instance is described by 6 binary-valued attributes. Two of the attributes, the address bits, select one of the four remaining attributes, the data bits. The classification of the instance is equal to the value of the selected attribute (data bit). This task is a difficult one for learning programs because no single attribute accounts for the classification.[2]

Figure 1 shows the cumulative training cost (cpu time) versus the number of training events. The list of 64 instances was treated as circular list. The final number of training events needed to result in 100% correct classification of all 64 instances, the total cumulative training cost, and the proportion of the 64 instances classified correctly when the experiment was stopped were:

Algorithm	Events	Time	Proportion	
$ID3$	53	630.4	100	
$\widehat{ID3}$	61	92.3	100	
$ID4$	384	327.7	63	*
$\widehat{ID4}$	384	258.1	50	*
$ID5$	57	184.3	100	
$\widehat{ID5}$	74	83.5	100	

[2]Indeed, the ID3 algorithm finds a suboptimal decision tree for this learning task.

ID4 and $\widehat{ID4}$ were terminated after $|I| \cdot |A| = 64 \cdot 6 = 384$ training events.

ID5 and $\widehat{ID5}$ required more training instances than ID3 and $\widehat{ID3}$ respectively because ID5 does not re-establish the best test attribute for each subtree below a new test attribute.[3] Instead, the algorithm waits for subsequent instances to cause subtrees to be visited, whereupon the best test will be re-established.[4]

Figures 2 through 7 show the proportion of the 64 instances classified correctly versus the cumulative training cost for each algorithm. In terms of the proportion of instances classified correctly, ID3 and $\widehat{ID3}$ achieve the best performance earliest. This is because ID5 and $\widehat{ID5}$ lag in being ID3 equivalent. Note that ID4 and $\widehat{ID4}$ do not stabilize. This is indicated by a "*" in the figure and in the table above. Figure 4 shows a repeating pattern in classification performance, indicative of the repeating versions of the decision tree that occurred.

4.2 Preference Predicate

For this experiment, the database I contained 100 instances of pairs of feature vectors describing pairs of Othello boards. The 7 features from each pair combine to make 14 integer-valued attributes. An instance is labelled positive when the first board is provably better than the second board and negative otherwise (Utgoff & Saxena, 1987; Utgoff & Heitman, 1988).

The performance of the 6 algorithms was much the same as for the multiplexor example. The final number of training events needed to result in 100% correct classification of all 100 instances, the total cumulative training cost, and the proportion of the 100 instances classified correctly when the experiment was stopped were:

Algorithm	Events	Time	Proportion	
$ID3$	88	6178.4	100	
$\widehat{ID3}$	117	2180.3	100	
$ID4$	1400	6084.8	74	*
$\widehat{ID4}$	1400	1874.3	62	*
$ID5$	98	3769.3	100	
$\widehat{ID5}$	184	1974.4	100	

ID4 and $\widehat{ID4}$ were terminated after $|I| \cdot |A| = 100 \cdot 14 = 1400$ training events.

5. Conclusion

ID5 provides an incremental method for building ID3 type decision trees. An existing routine can be called, when desired, to guarantee that the tree is ID3 equivalent. The cost of using ID5 is less than the cost of using ID3 to build a new decision tree

[3] This is known to be the case because, for the experiment, ties for best test attribute were broken identically for ID3, $\widehat{ID3}$, ID5, and $\widehat{ID5}$.

[4] In the implementation, recursive re-establishment of best test attribute at each subtree following a pull-up is toggled by a switch, which is left turned off. It is cheaper to make the call only when an ID3 equivalent tree must be guaranteed.

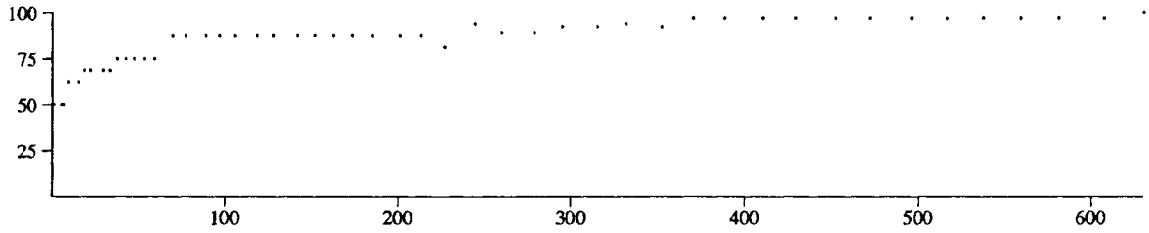


Figure 2: Proportion correct (y axis) vs. training cost (x axis) for ID3

Figure 3: Proportion correct (y axis) vs. training cost (x axis) for ID3

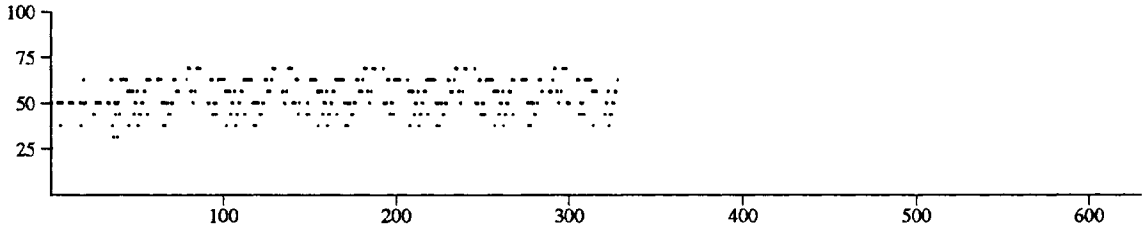


Figure 4: Proportion correct (y axis) vs. training cost (x axis) for ID4

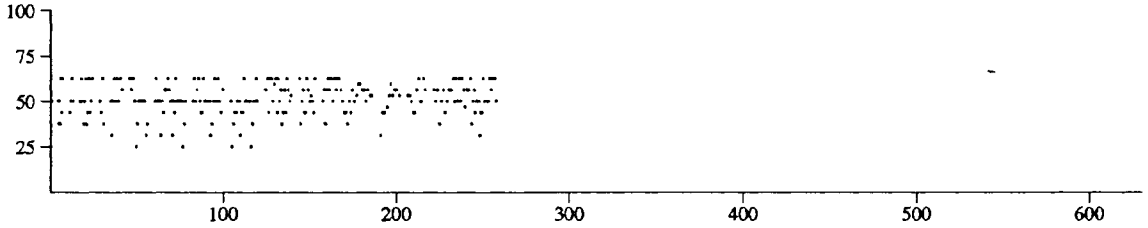


Figure 5: Proportion correct (y axis) vs. training cost (x axis) for ID4

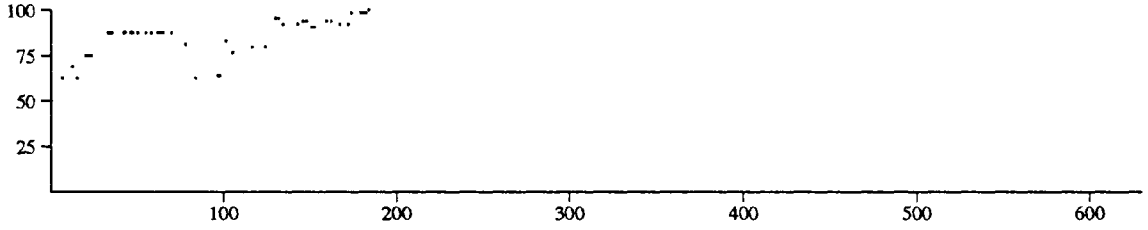


Figure 6: Proportion correct (y axis) vs. training cost (x axis) for ID5

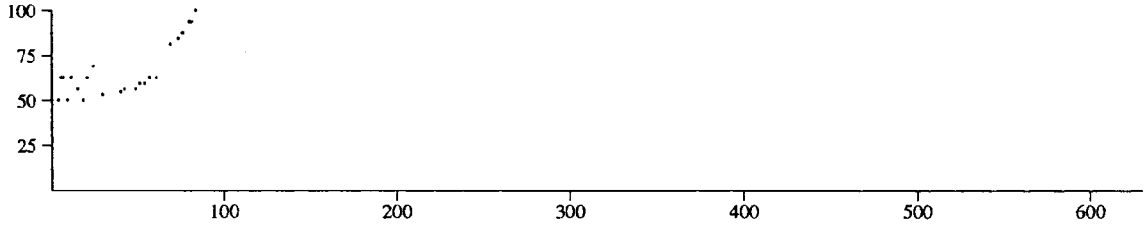


Figure 7: Proportion correct (y axis) vs. training cost (x axis) for ID5

after each training instance, unless tree revisions become too frequent or the attributes being exchanged have a large number of values. ID5 offers an alternative to ID3 and ID4. The analysis gives some guidance on which one to pick for a given learning task.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. IRI-8619107 and by a General Electric Faculty Fellowship. The several discussions with Jeff Schlimmer on ID3 and ID4 were very helpful. The presentation benefitted from helpful comments from Sharad Saxena, Margie Connell, Jamie Callan, Peter Heitman, Kishore Swaminathan, and Dan Suthers.

References

Lewis, P. M. (1962). The characteristic selection problem in recognition systems. *IRE Transactions on Information Theory IT-8(2)*, 171-178.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, Carbonell, & Mitchell (Eds.) *Machine Learning: An artificial intelligence approach*, Morgan Kaufmann, 463-482.

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

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

Utgoff, P. E. and Saxena, S. (1987). Learning a preference predicate. *Proceedings of the Fourth International Workshop on Machine Learning*, Morgan Kaufman, (pp. 115-121).

Utgoff, P. E. and Heitman, P. S. (1988) Learning and generalizing move selection preferences, *Proceedings of the AAAI Symposium on Computer Game Playing*, (pp. 36-40).