Decision Tree Induction Based on Efficient Tree Restructuring[1]

Paul E. Utgoff utgoff@cs.umass.edu
Department of Computer Science, University of Massachusetts, Amherst, MA 01003

Neil C. Berkman neil@corvid.com
Corvid Corporation, 779 West Street, Carlisle, MA 01741

Jeffery A. Clouse clouse@ncat.edu
Department of Computer Science, North Carolina A&T State University, Greensboro, NC 27411

Abstract: The ability to restructure a decision tree efficiently enables a variety of approaches to decision tree induction that would otherwise be prohibitively expensive. Two such approaches are described here, one being incremental tree induction (ITI), and the other being non-incremental tree induction using a measure of tree quality instead of test quality (DMTI). These approaches and several variants offer new computational and classifier characteristics that lend themselves to particular applications.

Keywords: decision tree, incremental induction, direct metric, binary test, example incorporation, missing value, tree transposition, installed test, virtual pruning, update cost.

1 Introduction

Decision tree induction offers a highly practical method for generalizing from examples whose class membership is known. The most common approach to inducing a decision tree is to partition the labelled examples recursively until a stopping criterion is met. The partition is defined by selecting a test that has a small set of outcomes, creating a branch for each possible outcome, passing each example down the corresponding branch, and treating each block of the partition as a subproblem, for which a subtree is built recursively. A common stopping criterion for a block of examples is that they all be of the same class.

This non-incremental approach to inducing a decision tree is quite inexpensive because exactly one tree is generated, without constructing or evaluating explicit alternatives. In terms of searching the space of all possible decision trees, the induction process consists of instantiating a specific tree, starting at the root. When one determines that a particular node shall be a decision node with a specified test, or a leaf with a specified class label, one implicitly rejects all other trees that would differ at this node. This greedy tree construction process implements a function that maps a particular set of examples to a particular tree.

There are alternative strategies for searching tree-space, two of which are presented here. First, for incremental decision tree induction, one can map an existing tree and a new training example to a new tree. This is different from the non-incremental approach described above, in which one maps a single batch of examples to a particular tree. Second, for decision tree induction using

[1]The correct citation for this article, (C) 1997 Copyright Kluwer Academic Publishers, is: Utgoff, P. E., Berkman, N. C., and Clouse, J. A. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning, 29*, 5-44.

a measure of tree quality, hereafter called *direct metric tree induction*, one simply maps an existing tree to another. As explained below, each of these methods requires the ability to restructure an existing decision tree efficiently. The next section presents the tree revision mechanism, and the following two sections present the two decision tree induction algorithms that are based upon it.

2 Tree Revision

Both of the decision tree induction algorithms presented here depend on the ability to transform one decision tree into another. For simplicity, the discussion is limited in this section to a tree that is based on a consistent set of labelled examples. For any particular set of consistent examples, there exists a multitude of decision trees that would classify each of those examples correctly. For example, if one were to place a test t_1 at the root, and then build the rest of the tree recursively, a particular tree would result. However, if instead one were to place a different test t_2 at the root, and then build the rest of the tree recursively, a different particular tree would be produced. Each tree would be consistent with the training examples, but would likely represent a different partition of the example space.

An algorithm that restructures a tree will sometimes need to change the test that is installed at a decision node. When this occurs, one would like to be able to produce as efficiently as possible the tree that would result when using the newly installed test. One would like to effect such a change of the installed test by revising the existing tree, instead of building a new tree from the original training examples, assuming that this is computationally more efficient.

2.1 Representation

It is assumed that every possible test at a decision node has exactly two possible outcomes, which means that the decision tree is always binary. There is no loss of representational power in this choice because for every non-binary tree there are one or more binary trees that produce an identical partition of the example space. Symbolic variables with more than two possible values are mapped automatically to an equivalent set of propositional variables. For example, if the value set for the variable *color* were $\{$*red, green, blue*$\}$ then the possible binary tests would be *(color = red)*, *(color = green)*, and *(color = blue)*. For numeric variables, the conversion to a binary test is done as it is by C4.5 (Quinlan, 1993), by finding a cutpoint and incorporating it into a threshold test, for example $(x < cutpoint)$. The outcome of a test is either that the value of the variable in the example satisfies the test, or that it does not.

The adoption of only binary tests brings two principal benefits. The first is that there can be no bias among tests that is due to the tests having a different number of possible outcomes. This is important because many common methods for selecting a test are biased in this manner (White & Liu, 1994). Second, choosing a binary split at a decision node is a conservative approach to partitioning, because a block of examples is divided into at most two smaller blocks. This is beneficial because each block can be further subdivided, if necessary, by selection of a new test that is deemed best for that block. Choosing a test that immediately partitions a set of examples into more than two blocks is more aggressive. By partitioning more conservatively, one keeps a larger number of examples available in each block, which is important if additional partitioning will be done in that block.

Mooney, Shavlik, Towell, and Gove (1989) found that recoding discrete variables as propositional variables improved classification accuracy for the ID3 decision tree induction algorithm. Breiman, Friedman, Olshen, and Stone (1984) employ binary tests in the CART decision tree induction program. Fayyad (1991) observes that for numeric variables, it can be advantageous to search for multiple cutpoints, rather than a single one. While several binary tests can represent the same partition that would be produced by a multiway test, there is no reason to believe that a greedy selection of such binary tests would actually lead to the same partition. One could search for multiple cutpoints, and then pick just one for a binary test, leaving other cutpoints to be found for the subtrees.

2.2 Information Maintained at Each Decision Node

To be able to change the test that is installed at a decision node, one needs to maintain information at that node that provides the basis for evaluating the quality of each possible test. The idea of maintaining such information for symbolic variables was demonstrated by Schlimmer and Fisher (1986). For each test that is based on a specific value of a symbolic variable, e.g. *(color = blue)*, the frequency counts for each outcome-class combination are kept and updated as necessary. For each possible test (based on a specific cutpoint) of a numeric variable, it would be too costly to keep a separate set of frequency counts for each test. Instead, the list of values observed in the examples at that node is maintained in sorted order by value, with each value tagged by the class of the example in which it was observed. For each pair of adjacent values, the midpoint of the two values defines a possible cutpoint. The possible cutpoints and the merit of each one can be computed efficiently during a single pass over the sorted list of tagged values. When average class entropy is the metric for test selection, one needs only to consider those cutpoints that separate two values from different classes (Fayyad & Irani, 1992).

2.3 Incorporating a Training Example

A basic operation in tree revision is to change the set of examples on which the tree is based. Various forms of incremental tree induction are presented in the next section, each of which depends on the ability to add an example to the set of examples on which the tree is based. Adding an example in this way is also known as incorporating the training example into the tree.

When an example is to be incorporated into an empty tree, the tree is replaced by a leaf node that indicates the class of the leaf, and the example is saved at the leaf node. Whenever an example is to be incorporated, the branches of the tree are followed as far as possible according to the values in the example. If the example has the same class as the leaf, the example is simply added to the set of examples saved at the leaf node. If the example has a different class label from the leaf, the algorithm attempts to turn the leaf into a decision node, picking the best test according to the test-selection metric. The examples saved at the node that was just converted from a leaf node to a decision node are then incorporated recursively by sending each one down its proper branch according to the new test. Thus, an example can be added to a node, and it will work its way down the tree to a leaf, possibly sprouting branches at leaves as it moves downward through the tree. The procedure that accomplishes this task is named *add_example_to_tree*, and pseudo-code for it is shown in Table 1.

2.4 Missing Values

One must be able to handle an example for which the value of one or more input variables are missing. For tree construction, this is problematic when a test has been chosen for the decision node that requires knowing the value that is missing in an example. For classification of an unlabelled example, one must somehow use the tree and its leaves to infer the label for the unlabelled example.

A missing value is treated as a special value that does not satisfy the test at a decision node. A

Table 1. Procedure add_example_to_tree()

```
add_example_to_tree(node,example)
  if node is NULL
    then convert node to an empty leaf

  if node is a leaf
    then save example at node
         if node should be converted to a decision node
           then construct information at node
                mark node fresh
                for each example j saved at node
                  if test_is_true(node,j)
                    then add_example_to_tree(node->left,j)
                    else add_example_to_tree(node->right,j)
    else update information at node
         mark node stale
         if test_is_true(node,example)
           then add_example_to_tree(node->left,example)
           else add_example_to_tree(node->right,example)
```

value in an example either satisfies the test, or it does not. For example, if the test is *(color = blue)*, then a value of *blue* satisfies the test, and neither the value *red* nor the value *?* (missing) satisfies the test. Similarly, if the test is $(age < 46)$, then a value of *31* satisfies the test, and neither the value *57* nor the value *?* (missing) satisfies the test. During tree construction and classification alike, an example with a missing value for the test at the decision node will be passed down the false branch. This is similar to treating *?* as a bonafide value, but it is different because no symbolic equality test can test for this value, nor can any numeric inequality test use *?* as its cutpoint.

With binary tests, all the examples with missing values for the test are sent down the false branch. This concentrates examples missing a value for the particular test into the right subtree, and keeps the left subtree free of such examples. If further partitioning is required in the right subtree, the tree induction algorithm is of course free to select a new test, which can be based on any variable.

This approach to handling missing values also simplifies computation of the test selection metric. For the purpose of computing the metric, there is no such thing as a missing value because every example is placed into one of the two blocks of the candidate partition. There are many test selection metrics that one could choose, but Quinlan's gain-ratio metric is adopted here. Given that a missing value for a test is false for that test, there are no missing values for the computation of the metric. Hence Quinlan's (1993) special adjustments for missing values do not apply here.

2.5 Recursive Tree Transposition

When a decision tree induction algorithm elects to change the test that is installed at a decision node, the tree needs to be revised so that it corresponds to the newly installed test. The set of training examples that satisfied the old test is generally different from the set of examples that satisfies the new test. The subtrees need to be revised accordingly, and one would like this process

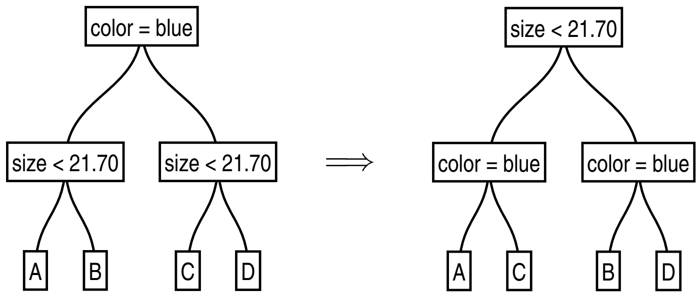

Figure 1. Tree Transposition Operator (left on TRUE, right on FALSE)

to be less expensive than rebuilding the subtrees from the examples on which they are based.

The operation of changing the installed test at a decision node is accomplished through a sequence of tree transpositions that is determined recursively. After the desired test has been installed, it may be that one or more subtrees has a test installed at a decision node that was necessary to accomplish the original tree transposition, rather than one that the decision tree induction algorithm would otherwise choose. To ensure that the best test is installed at each of the decision nodes below, the tree revision process checks that each installed test below is the one desired. If not, the subtree is revised recursively to install the desired test at its root decision node.

Consider one of the base cases, as illustrated in Figure 1. The goal is to transpose the lefthand tree into the righthand tree. This can be accomplished easily by rearranging pointers because the test *(size < 21.70)* that is to be installed at the root is already installed at each of the children of the root. Notice that the subtrees A, B, C, and D are simply reattached, and are not revisited for this case. The sets of examples on which each of A, B, C, and D are based have not changed, so there is no need to change those subtrees. Hence, the test or leaf information maintained at any of these grandchildren nodes has not changed. Similarly, the set of examples corresponding to the root has also not changed, though the designation of the installed test has, which means that the test information maintained at the root also remains unchanged. Only the sets of examples at the two new children of the root, now with test *(color = blue)*, have changed. For example, the left subtree formerly corresponded to the examples used to build subtrees A and B, but now corresponds to the examples used to build subtrees A and C. This raises the problem of how to update the test information at each child of the root.

Fortunately, the problem has an inexpensive solution that does not generally require rebuilding the information from the training examples. One simply recreates the test information by merging the test information of the two grandchildren. For example, for the left subtree, one would define the information as the 'sum' of the information at nodes A and C. For a symbolic variable, one adds the corresponding frequency counts, and for a numeric variable one copies and merges the two sorted tagged lists of values.

There are several other base cases, all dealing with one or more grandchildren being leaves, or one or both children being leaves. If one of the children of the root is a leaf node, instead of a decision node, then transposition is accomplished somewhat differently. For example, consider the case in which the right subtree is a leaf. Then transposition is accomplished by discarding the root node, reattaching the left subtree in its place, discarding the right subtree (the leaf), and reincorporating its examples at the root. For the base case in which both subtrees are leaves, one discards both subtrees, installs the desired test at the root, and then reincorporates the examples

Table 2. Procedure ensure_best_test()

```
ensure_best_test(node)
  if node is a decision node and node is stale
    then find best_test for this node
         if best_test is not already installed
           then transpose_tree(node,best_test)
         if node is a decision node
           then mark node fresh
                ensure_best_test(node->left)
                ensure_best_test(node->right)
```

from both leaves at the root. Reincorporating an example from a discarded leaf does constitute reprocessing the example, but this occurs only at the fringe of the tree.

All of these base cases revise the tree in such a way that the tree has no node that is unnecessarily expanded. Every node that should be a leaf is a leaf. There are a few other base cases that arise during the transposition process when a subtree may not exist because no examples had that outcome for the installed test, but these represent temporary states that are handled in a straightforward manner.

The base cases handle any tree in which a child of the root is either a leaf or is a decision node whose test is the one that is to be installed at the root. When either child is a decision node that has an installed test that is not the one to be installed at the root, then tree transposition is applied recursively to that child so that it has the desired installed test. This always produces one of the base cases, which is then handled directly.

2.6 When to Apply Recursive Transposition

As described above, the recursive tree transposition operator provides the ability to restructure a given tree into another that has the designated test installed at the root. This is quite useful, but it is not enough by itself for producing the best tree because it has only caused one decision node to have the desired test installed. During the recursive transposition, it may be that the subtrees have been transposed as a by-product of bringing the desired test to the root. The installed test of each decision node in the subtrees may be there as the result of transposition rather than as the result of a deliberate choice. Again consider the transposed tree in Figure 1. Each of the two subtrees has test *(color = blue)* installed, but that is only because a transposition was performed from above, not because *(color = blue)* was identified as the best choice and installed intentionally.

To ensure that every decision node has the desired test installed, according to the attribute selection metric, one needs to visit the subtrees recursively. At each decision node that requires that a different test be installed, the algorithm transposes the tree to install the best test at the node. It could become costly to check every decision node of the subtrees after a transposition. Often, a subtree is not touched during a transposition. To this end, a marker is maintained in each decision node that indicates whether the choice of the installed test is stale. Any operation that changes the set of examples on which a subtree is based marks the test as stale. This can happen either when incorporating an example or when transposing a tree. Changing the set of examples on which a subtree is based changes the test information, invalidating the basis on which the installed test was selected.

Table 3. Procedure incremental_update()

```
incremental_update(node,example)
  add_example_to_tree(node,example)
  ensure_best_test(node)
```

Whenever a desired test has been identified and, if necessary, installed, one removes the test's stale mark. To ensure that every decision node has the desired test installed, one proceeds recursively in the following manner: at the root, identify the desired test and install it via recursive transposition; for each subtree, if it is marked stale, then recursively identify its desired test and install it. The procedure that accomplishes this task is named *ensure_best_test*, and pseudo-code for it is shown in Table 2.

3 Incremental Tree Induction

This section presents an incremental tree induction algorithm ITI (incremental tree inducer) that makes extensive use of the tree transformation mechanism described in the previous section. An incremental algorithm makes sense for an application that uses an embedded classifier that is based on a stream of observed examples. Applications currently exist that accumulate examples by day, and rebuild the embedded decision tree by night. Employing an incremental method would allow online tree updating.

The algorithm described here was motivated by several design goals:

1. The average incremental cost of updating the tree should be much lower than the average cost of building a new decision tree from scratch. However, it is not necessary that the sum of the incremental costs be lower because we care only about the cost of being brought up to date at a particular point in time.

2. To the extent possible, the update cost should be independent of the number of training examples on which the tree is based.

3. The tree that is produced by the incremental algorithm should depend only on the set of examples that has been incorporated into the tree, without regard to the sequence in which those examples were presented.

4. The algorithm should not be biased toward selection of a test because the test has a larger set of possible outcomes than that of another test.

Additional well-accepted design goals are that the algorithm should accept examples described by any mix of symbolic and numeric variables (attributes), handle multiple classes, handle inconsistent training examples, handle examples with missing values, and avoid fitting noise in the examples.

3.1 Algorithm ITI

The basic ITI incremental decision tree induction algorithm is based on the tree revision mechanism described in Section 2, and thus can be stated simply. When given a training example that is to be incorporated into the tree, pass it down the proper branches until a leaf is reached. This includes updating the test information kept at each node through which it passes, and marking each such node stale. It also includes the process of incorporating an example at a leaf, which may cause additional growth of the tree below that leaf. After the example has been incorporated, visit each stale node recursively, as described above, ensuring that the desired test is installed at that node. The procedure that accomplishes this task is named *incremental_update*, and pseudo-code for it is shown in Table 3.

As usual, a test is considered best if it has the most favorable value of the attribute-selection metric. For the order of the training examples to remain immaterial, a tie for the best test must be broken deterministically. Recall that a test is constructed from a variable and its observed value set. For ITI, such a tie is resolved in favor of the lexically lower variable name. For tests based on the same variable, a tie is resolved in favor of the lexically lower symbolic value or the numerically lower cutpoint, depending on the type of the variable. This tie-breaking mechanism ensures that there is a unique tree for any given set of training examples.

3.2 Inconsistent Training Examples

Two examples are inconsistent if they are described by the same variable values but have different class labels. When inconsistent examples occur, they will be directed to the same leaf. If one were to split every impure leaf, this would cause an infinite recursion. However, since converting the leaf to a decision node would provide no information, and this is easily detected by the gain-ratio metric, ITI keeps the node as a leaf and simply adds the example to the set of examples retained at the leaf, making an impure leaf. This causes no trouble for classification because the class name that is returned for the unlabelled example is that of the majority class of the examples at that leaf. A tie is broken in favor of the lexically lower class name.

3.3 Virtual Pruning

An important component of decision tree induction is to avoid overfitting the training data, especially when the data are known to contain attribute or classification error (noise). A variety of methods have come into existence, and the question is which of them is best suited to the incremental induction problem. All of the approaches that maintain a separate pruning set are oxymoronic for incremental tree induction. For ITI, a suitable approach is based on the minimum description length principle (Rissanen, 1978; Quinlan & Rivest, 1989).

To decide whether to prune a subtree to a leaf, one considers whether the subtree could be represented more compactly by a leaf with a default class and a list of exceptions, where each exception is an index into the list of examples and an indication of its non-default class label. For any subtree that one would want to be pruned (replaced with a leaf), one marks its root decision node as being pruned, but does not actually discard anything. For incremental induction, one preserves all information so that it is possible to reconsider whether a subtree should or should not be virtually pruned. To unprune it, one simply removes the mark that it is considered to be pruned. For all practical purposes, such as classifying examples with the tree, or inspecting the tree by printing it, a virtually pruned tree behaves and appears as though it had been truly pruned.

The virtual pruning process is accomplished by a post-order traversal of the decision tree that sets a marker in each decision node to indicate whether that decision node is to be considered pruned to a leaf. The previous status of whether the node was marked as pruned is immaterial. The procedure winds its way down to the leaves via the post-order traversal, and sets each subtree as pruned (or not) based on the minimum description length (MDL).

For each leaf, the number of bits needed to encode the leaf is measured as $1 + \log(c) + x(\log(i) + \log(c-1))$, where c is the number of classes observed at the leaf, x is the number of examples at the leaf that are not of the default class, and i is the total number of examples at the leaf. One bit is needed to indicate whether the node is a leaf, $\log(c)$ bits are needed to indicate the default class, and for each of the x exceptions, one needs $\log(i)$ bits to specify the exception, and $\log(c-1)$ bits to specify its non-default class. This total number of bits to encode the leaf is stored in the leaf node.

For each decision node, the number of bits needed to encode the subtree is measured as $1 + \log(t) + l + r$, where t is the number of possible tests at the node, l is the MDL of the left subtree (already set), and r is the MDL of the right subtree (already set). One bit is needed to indicate whether the node is a decision node, $\log(t)$ bits are needed to identify the test, and l and r bits are needed to encode the left and right subtrees respectively. This total number of bits to encode the decision node is stored in the decision node.

To decide whether to mark a decision node as pruned, the MDL for the node is computed for the case in which the node is left as a decision node (not pruned), and for the case in which it would be pruned to a leaf. If the virtual leaf would require fewer bits to encode, then the node is marked as pruned, and the MDL of the virtual leaf is saved at the node. Otherwise, the node is marked as not pruned, and the MDL of the subtree is saved instead.

3.4 Variants of ITI

A variety of training modes are possible, four of which are described here. Upon presentation of a training example, one can decide whether or not to incorporate that training example into the tree. Any policy for making this decision constitutes one element of a training mode. When one does elect to incorporate a new training example into the tree, one can then decide whether or not to ensure immediately afterward that the best test is installed at each decision node. Because the process of adding an example to a tree can be accomplished independently from revising the tree, it is possible to accept several examples at a time, add each one to the tree without revising (except possibly for growing new leaves), and then revise the tree just once by visiting the decision nodes to ensure that each has the best test installed.

In *incremental mode*, each training example that is presented is immediately incorporated into the tree, and the tree is then immediately restructured as necessary so that every decision node has its most desired test installed. This mode always produces the same tree that one would obtain with the batch version that is described below.

For *error-correction mode*, one incorporates a training example only if the existing tree would misclassify it. Otherwise the training example is not incorporated and therefore has no effect on the tree. This mode of training is akin to the error correction procedures of statistical pattern recognition, and it was also suggested by Schlimmer and Fisher (1986). There are two variations of this mode. First, for a stream of training examples, one simply ignores examples for which the tree is currently correct. Second, for a fixed pool of examples that have not been incorporated into the tree, one cycles through the pool repeatedly, removing each incorrectly classified example from the pool and incorporating it into the decision tree, until the tree does not misclassify any example still remaining in the pool. Although examples are examined one at a time, this training regimen departs somewhat from the notion of a linear stream of training examples.

For a pool of examples, ITI will always build a tree and halt. This is true even when the examples are noisy, because ITI continues cycling through the pool until every example remaining

Table 4. Procedure error_correction_train()

```
error_correction_train(node,pool)
  pool_perfect := false
  while not pool_perfect
    pool_perfect := true
    for each example in pool
      if classify(node,strip_label(example)) != label(example)
        then pool_perfect := false
             remove example from pool
             add_example_to_tree(node,example)
             ensure_best_test(node)
```

in the pool is classified correctly. When an example in the pool is misclassified, it is removed from the pool and added to the tree. Thus, even though the tree may not become a perfect classifier on all the training examples that it has incorporated (when there is noise or inconsistency or pruning is turned on), it does continue to select and remove currently misclassified training examples from the pool until no misclassified examples remain in the pool. This occurs either when the current tree classifies all training examples still in the pool correctly or when the pool becomes empty because all examples have been incorporated in the tree. The procedure that accomplishes training from a pool of examples is named *error_correction_train*, and pseudo-code for it is shown in Table 4.

In *lazy mode*, one delays ensuring that the tree is up to date until the tree is needed for some purpose, such as to classify an unlabelled example. Most of the effort in ITI goes to ensuring, after each training example, that the tree has the best test installed at each decision node. However, one could avoid this work by not doing it until the tree is needed. Instead, one can add each example to the tree without revising the tree, beyond the simple operations that occur when incorporating an example. Then, whenever the tree is needed, a single call to the procedure for ensuring that the best test is installed at each node brings the tree to its proper form. For all practical purposes, the tree is being updated in incremental mode, but presumably with less overall computation.

A *batch mode* exists for the case in which one has an initial batch of examples and no current tree, and wants to build a tree as quickly as possible. Of course this mode is not incremental in any sense, but it provides a method of building an initial tree from a set of examples that is more efficient than incremental mode. The algorithm constructs all the data structures that the incremental mode would, but in the traditional top-down manner. This mode is the fastest way to build an ITI tree from scratch, and makes sense for someone who wants to build a tree in the standard one-shot way. The data structures are created as they would be for incremental mode so that subsequent operations that might require revising the tree remain applicable. If one were to give up the ability to do subsequent revisions, one could be even more efficient by not building the data structures that go along with the tree revision capability.

4 Direct Metric Tree Induction

This section presents a direct metric tree induction algorithm DMTI (direct metric tree inducer) that relies on the tree restructuring mechanism described in Section 2. DMTI is not an incremental algorithm, but is instead a greedy top-down tree inducer. The distinguishing aspect

Table 5. Procedure dmti()

```
dmti(node)
  if node is a decision node
    then for each eligible_test at node
           transpose_tree(node,eligible_test)
           note direct metric value for the resulting tree
         transpose_tree(node,best_eligible_test)
         dmti(node->left)
         dmti(node->right)
```

of DMTI is that for each test that it could install at a node, it actually installs it, and then uses a metric that is a function of the resulting tree to assess the desirability of that test. This differs from the usual approach of picking a test based on a heuristic function (an indirect metric) of various frequency counts tallied from the examples.

4.1 Algorithm DMTI

The DMTI algorithm is a variation of the classical top-down approach, because one finds the best test to install at the root, installs it, and then solves the subproblems recursively. There are three important differences. First, one starts the process with a decision tree built by ITI, instead of a set of examples. Second, a test is assessed at a node by installing it, including automatic revision of the subtrees using the indirect metric, and then by evaluating the direct metric on the resulting tree. Thus, for n permissible tests at a node, DMTI evaluates n different trees in order to pick the best test to install at that node. Third, it would typically be quite expensive to consider all possible tests at a node, so the set of permissible tests is limited to the best test for each input variable according to the indirect metric. For a symbolic variable, the best test is the best equality test variable, and for a numeric variable it is the best threshold test. So, for DMTI, the set of permissible tests at a node is limited in size to the number of input variables. Pseudo-code for the DMTI procedure is shown in Table 5.

4.2 Direct Metrics

A direct metric defines whether one test is preferred to another based on a comparison of the trees that would result. This raises the familiar question of how to decide whether one tree should be preferred to another. With no additional information, there can be no universally correct answer to this question, but in practice it is usually possible to make a good choice. For example, if one has a strong prior belief that the target concept can be modelled best by a decision tree that is a simple function of the input variables, then one would prefer a smaller consistent tree to a larger consistent tree. Such a preference accompanies the belief that a designer of a set of training examples will try to choose input variables that are as predictive of the class label as possible. Other prior knowledge can be used to determine a direct metric that will be appropriate for a given tree induction task. Consider four possible direct metrics.

The direct metric *expected-number-of-tests* returns the number of tests that one would expect to evaluate in order to classify an example, assuming that testing examples are drawn according to the same probability distribution as the training examples. The expected number of tests can be computed by counting the total number of tests evaluated while classifying all the training examples, and dividing by the total number of training examples. It is possible to calculate the value of this metric during a single traversal of the tree, without actually classifying any examples. All the examples that have been incorporated into the tree are saved at a leaf of the tree, and all node heights are known during the traversal.

The metric *leaf-count* returns the number of leaf nodes in the tree. This count can be computed during a single traversal of the tree. Though the number of leaves is related to the number of tests, it is not directly related to the expected number of tests. For example, it is possible for a tree t_1 to have a higher leaf count and a lower expected number of tests than another tree t_2.

The *minimum-description-length* metric returns the number of bits needed to encode the tree, using the encoding scheme described in Section 3.3. Whether or not one has pruning turned on, one can apply DMTI to search for a test that produces a tree with the locally smallest attainable MDL.

The metric *expected-classification-cost* is identical to expected-number-of-tests except that each test has a specified evaluation cost, instead of the implied uniform evaluation cost. In some applications, such as diagnosis, some tests are much more expensive than others, and the cost of producing the answer is an important factor (Tan & Schlimmer, 1990).

Finally, the direct metric *expected-misclassification-cost* measures the penalty that one would pay when misclassifying an example, assuming that testing examples are drawn according to the same probability distribution as training examples. Often, tree induction algorithms embody the assumption that all classification errors incur the same cost (Pazzani, Merz, Murphy, Ali, Hume & Brunk, 1994). To be more comprehensive, one can include an explicit cost matrix that specifies the cost of labelling an example with class X when it should have been class Y. It is possible to calculate the value of this metric in a single traversal of the tree.

5 Comparison of Performance Characteristics

When might one want to use ITI or DMTI? To answer such a question, one first needs to know how good a classifier one can expect, and how much it may cost to produce it. To this end, consider a comparison of several algorithms along several criteria: classification accuracy, tree size as measured by number of leaves, classification cost as measured by the number of tests one can expect to evaluate when classifying an example, and the CPU cost to build the tree. Which algorithms are better than which?

5.1 Experimental Design

The algorithms to be compared are three variants of ITI, three variants of DMTI, and two variants of Quinlan's C4.5. All eight algorithms are applied to forty-six classification tasks in a perfectly balanced and perfectly crossed manner. For each task-algorithm combination, the variables mentioned above are measured by a ten-fold stratified cross validation. The purpose of the cross validation is to obtain a reasonably unbiased point estimate, not to obtain ten separate measurements. The same splits of the data were used for all algorithms.

To be able to draw conclusions about which algorithms are significantly different from the others, one must choose a significance test that is designed to handle multiple comparisons. One cannot simply apply a test for each comparison, because the chance of a false difference rises with each test, much like repeatedly rolling a twenty-sided die that has one face marked 'significant'. A variety of multiple comparison procedures have been devised, and Duncan's Multiple Range Test (Walpole, 1974) is used here.

To use the Duncan Multiple Range Test (DMRT), one first isolates the variance that cannot be attributed to any treatment, which is called the error variance. To do this, one proceeds with the initial stages of an analysis of variance. Here, a two-way analysis is used to factor out variance that can be attributed to choice of task or choice of algorithm. Then, instead of proceeding with an F-test, as one would for an analysis of variance, one proceeds with the multiple range test, which involves comparing the means of the algorithms. The term 'algorithm mean' indicates the mean for the algorithm across all the tasks.

The least significant range for the p means to be compared is the product of the least significant studentized range and the square root of the quotient of the error variance and the number of tasks. The least significant studentized range can be determined from a table that is indexed by the error degrees of freedom and the number p of means being compared. If the range of the means (highest - lowest) being compared is greater than the corresponding least significant range, then the means are presumed to be significantly different. The table of least significant studentized ranges was computed by Duncan to compensate for the fact that multiple comparisons are being made.

One initially sorts the algorithm means, and then tests each of the possible ranges of means. If a group of adjacent means is not significantly different, then that is depicted by drawing a line segment under that group. No subgroup is tested when a group is not significantly different. Any two means that are underscored by any common line are not significantly different. Any two means that are not underscored by a common line are significantly different (Steel & Torrie, 1980).

Finally, to reduce experimental error, the eight algorithm variants are run as a group with pruning turned off, and then again as a separate group with pruning turned on. There is no discussion here of the relative merits of pruning versus not pruning.

5.2 Algorithms

For ease of reference, each of the eight algorithms is given a simple name here. Variant I1 is ITI in batch mode, with a leaf being split when there is at least one example of the second-most frequently occurring (second-majority) class. Variant I2 is ITI in batch mode, with a leaf being split only when there are at least two examples of the second-majority class. Batch mode is used here because it runs more quickly than incremental mode, and builds the same tree. The computational characteristics of incremental mode are discussed below in Section 6, and are not of interest here. Variant IE is like I1, but runs in error-correction mode instead of batch mode.

Variant C2 is C4.5 with its default settings. Like the I2 variant, C2 splits a leaf when there are at least two examples of the second-majority cass. The C1 variant uses the -m1 option of C4.5 to make it correspond to the I1 variant of ITI. The C4.5 algorithm uses non-binary tests for discrete variables that have a value set larger than two. One could recode the data in a preprocessing step, so that C4.5 would be forced to produce binary tests, but that is not what C4.5 does, so no recoding has been done here.

The remaining three algorithms are variants of DMTI, each using a different direct metric. The DM variant uses minimum description length, the DE variant uses expected number of tests for classification, and the DL variant uses the number of leaves.

5.3 Tasks

Thirty-nine of the tasks were taken from the UCI (Murphy & Aha, 1994) repository, and the remaining seven were acquired or produced elsewhere. From all the available UCI tasks, one was generally taken from UCI and included here if it was not extraordinarily large and if there was a clearly defined class label. The forty-six tasks differ in many ways, including number and type of attributes, number of classes, noise, missing values, and default accuracy. The goal was to pick as many tasks as was practical.

The heart disease tasks (cleveland, hungarian, switzerland, and va) are each five-class problems, whereas in many reports one sees four of the classes grouped, resulting in a two-class problem. The fayyad and usama-mys tasks come from Usama Fayyad. The horse-dead and horse-sick tasks are different from the UCI horse-cholic task. The mplex-6 and mplex-11 are six and eleven bit versions of the multiplexor problem. The tictactoe data is different from the UCI task of similar name. Here, all positions that can occur during play appear as examples, labelled with 'draw', or the number of ply until a win will be achieved. All positions are represented from the point of view of the player on-move, coded as 'x'.

For any task in which a training set and testing set were given, those sets were merged into a single set of examples. In this way, the stratified cross validation could be applied to all tasks. The tasks for which this was done were audio-no-id, monks-2, soybean, splice, and vowel.

5.4 Accuracy

For average accuracy of the algorithms when pruning is turned off, the DMRT

DM	DE	IE	DL	I1	I2	C1	C2
77.98	77.86	77.73	77.70	76.87	76.64	76.01	75.68

indicates no significant differences. When pruning is turned on, the DMRT

DM	IE	C1	I2	I1	C2	DL	DE
79.48	78.14	77.92	77.53	77.53	77.34	76.89	76.74

indicates that DM is significantly more accurate on average than each of I2, I1, C2, DL, and DE. There are no other significant differences.

Table 6 shows the cross-validated accuracy for each of the task-algorithm combinations, with pruning turned off. The corresponding standard deviations are shown separately in Table 7 because they do not fit into one table. These deviations do not play a role in the analyses, but are included in order to help interpret the point estimates. Similarly, Table 8 shows the cross-validated accuracy for each of the task-algorithm combinations when pruning is turned on, with the standard deviations shown separately in Table 9.

5.5 Leaves

Consider the average tree size, as measured by the number of leaves. It is important to use the number of leaves because this indicates the number of blocks in the partition of the example space. This number is comparable for the different algorithms, whether or not they use a binary test at a decision node. For average tree size of the algorithms when pruning is turned off, the DMRT

C1	C2	I1	IE	DE	DL	DM	I2
371.18	189.47	108.04	101.49	94.33	84.52	84.43	69.09

indicates that C1 produces significantly larger trees on average than each of I1, IE, DE, DL, DM, and I2. When pruning is turned on, the DMRT

C1	C2	I1	I2	IE	DE	DL	DM
155.12	91.18	43.91	42.77	40.84	36.14	34.32	33.90

indicates the same set of significant differences.

Table 10 shows the leaf counts for each task-algorithm combination when pruning is turned off, with the corresponding deviations appearing in Table 11. Inspection of the data shows considerable variability with respect to which algorithms produced the smallest or largest trees. The means for C1 and C2 seem to be pulled up by just a few tasks. For example, C1 and C2 produce large trees for the nettalk task. Although there are only seven attributes, each has a very large value set, and neither the C1 algorithm nor the C2 algorithm are restricted to binary tests. For this task, any test will have a large number of branches. In contrast, the binary tests of ITI/DMTI cause a more conservative two-way split. Table 12 shows the leaf counts for each task-algorithm combination when pruning is turned on, with the associated deviations appearing in Table 13.

5.6 Expected Tests

The expected number of tests provides a measure of classification efficiency. The C1 and C2 variants were omitted because this measure is not readily available. One would expect DE to have the lowest mean since it attempts to minimize this very measure. For the average number of expected tests when pruning is turned off, the DMRT

IE	I1	I2	DL	DM	DE
10.45	8.98	8.45	7.13	6.66	5.54

indicates that I1 and I2 are not significantly different, nor are DL and DM. All other differences are significant. When pruning is turned on, the DMRT

IE	I1	I2	DL	DM	DE
7.76	6.58	6.57	5.40	5.03	4.46

indicates a similar story, with the addition that DM and DE are not significantly different. More generally, the DMTI variants produce more efficient trees than the ITI variants.

Table 14 shows the expected number of tests for each task-algorithm combination when pruning is turned off, and Table 15 shows the corresponding deviations. The means and deviations when pruning is turned on are shown in Tables 16 and 17.

5.7 CPU Cost

Finally, consider the question of how the CPU requirements compare (on a DEC 3000 with 96M main memory). The DMRT

DL	DM	DE	IE	I1	I2	C1	C2
514.12	488.82	394.23	152.17	1.52	1.44	0.59	0.50

indicates that each of DL and DM is significantly more costly than each of IE, I1, I2, C1, and C2. It also indicates that DE is significantly more costly than each of I1, I2, C1, and C2. There are no other significant differences. The DMRT

DL	DM	DE	IE	I1	I2	C1	C2
591.00	479.32	456.64	171.74	1.54	1.46	0.59	0.50

shows the same set of significant differences when pruning is turned on. The ITI and C4.5 variants are not significantly different in CPU cost. The I1 and I2 CPU requirements are within one second of those of C1 and C2 on average.

The DMTI variants are much more expensive, but perhaps worth the cost for some applications. For example, if one wants a decision tree that requires few tests on average for classification, then DE produces a significant improvement, at the cost of extra computation. One might argue that an algorithm such as I1 that builds a classifier in an average of 1.5 seconds is not working hard enough to find a tree that provides efficient classification.

Table 18 shows the values for each task-algorithm combination when pruning is turned off, with the corresponding deviations shown in Table 19. Table 20 shows the costs for each task-algorithm combination when pruning is turned on, with the corresponding deviations shown in Table 21.

6 Incremental Update Cost

Might one want to use the incremental ITI algorithm within a serial learning system or knowledge maintenance system? The primary issue is whether online learning via incremental tree revision is sufficiently time efficient. Under what conditions, if any, would one prefer to build a new tree from scratch?

In a serial task, in which each new training example is received sequentially, the problem to be solved by the algorithm is to obtain the new tree based on the newly augmented set of training examples. In the batch case, one takes the complete set of examples that has accumulated, and builds a new tree from them. In the incremental case, one takes the current decision tree and the new training example, and produces the new decision tree from them. What is the cost of obtaining the tree at time t, where t is the number of training examples that have been observed? For the incremental case, one has the tree at time $t-1$ and the new example observed at time t. The costs accrued prior to time t are not a concern because they have already been incurred and paid. Of interest is the cost of producing the tree at time t.

6.1 Cost Factors

There are five important factors that affect the cost of an incremental update of the existing decision tree. First is the cost of adding the information from the example to the data structures maintained at a decision node. For each symbolic variable, one increments the counter for the value-class combination present in the example. As the number of different observed values (the value set) grows, there is additional overhead in locating the counter in the data structure. For each numeric variable, one inserts the value found in the example (tagged with the class in the example) into a sorted list of values. For symbolic and numeric variables, the cost is proportional to the log of the size of the value set. For symbolic variables, the value set of a variable typically reaches its final size early in the training.

The second cost factor is the number of decision nodes that are updated when an example is incorporated into the tree. The tree is relatively small early in the training, and will tend to do most of its growing early, as the utility of observing new examples diminishes. It is a matter of

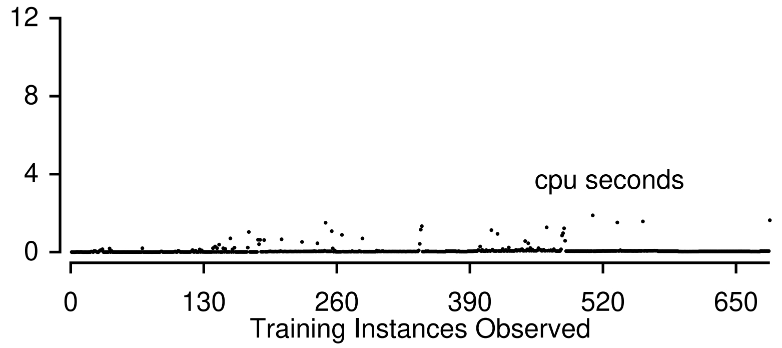

Figure 2. Incremental Update Cost for soybean

seeing a large enough number of training examples to be representative of the underlying example distribution. As enough examples come to have been seen, the size of the tree tends to stabilize.

The third factor is the frequency of changing the installed test at a decision node somewhere in the tree. Early in the training, the example distribution is not well represented, meaning that a new example can by itself cause a noticeable change in the distribution and the conditional probabilities that are computed within the test metric computation. Later in training, one example is unlikely to have much effect. Indeed, most examples do not lead to a change of the installed test. This effect of increased stability occurs earlier at higher decision nodes in the tree because these nodes are based on more examples. As the examples are partitioned, the decision nodes lower in the tree are based on fewer examples, and tend to lag in stability.

The fourth factor is the cost of restructuring a tree or subtree. This cost is attributable mostly to tree transpositions. What affects the cost of tree transposition? There is the matter of how many recursive calls are needed to set up a base case at the node in hand, and how many follow-up recursive calls are needed to ensure that the best test is installed at each of the decision nodes below. As mentioned above, stability of the best test at a node tends to be greater at nodes closer to the root of the tree. From this, one can surmise that transposition activity is generally lowest at the root, and generally highest at the fringe. The closer to the fringe that one transposes, the cheaper it is.

For a base case tree transposition of the kind shown in Figure 1, adjusting the pointers to reattach the subtrees has negligible cost. The bulk of the expense is in recreating the counting information that is maintained in each of the two decision nodes below the root. Although one does not re-examine training examples, it still takes effort to reconstruct the information from the two grandchildren decision nodes. Merging of symbolic variable information involves creating the structures to hold the information, and then adding it from each grandchild. Merging of numeric variable information requires copying and merging the two sorted-tagged lists of the grandchildren.

If one assumes that the size of the value set is independent of the number of examples seen, then the cost of a base case tree transposition is independent of the number of examples. The size of the value set of a symbolic variable is often smaller than the size of the value set for a numeric variable. For a symbolic variable, the size of the value set is likely to be independent of the number of examples, but for a numeric variable, this is less likely. It is quite possible to have a real-valued numeric variable in which the value in each example is unique, meaning that the size of the value set continues to grow as new examples are observed.

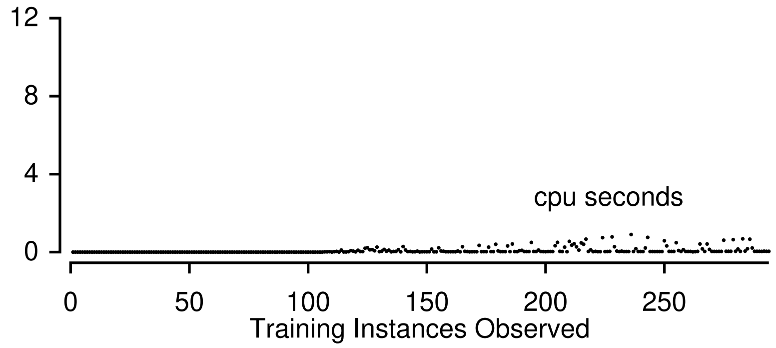

Figure 3. Incremental Update Cost for hungarian

The fifth and final important cost factor is associated with the simpler base cases of tree transposition. As mentioned above, when one of the subtrees is a leaf, transposition is accomplished by taking the non-leaf subtree as the result, and then incorporating the examples from the leftover leaf into the scavenged subtree. This causes an example to be handled further as it becomes incorporated into the surviving subtree. This activity qualifies as a limited re-examination of the example. Transposition is implemented this way in order to keep the tree in a reduced form. There is no decision node in the tree that could have been a leaf.

6.2 Illustrations

Three illustrations of ITI in its incremental mode show the net effect of these cost factors under different circumstances. The soybean, hungarian, and vowel tasks were run on all of the available examples, while measuring the cost of each update of the tree.

The soybean task has all symbolic variables, and in Figure 2 one sees that most updates are inexpensive. The cost of building the tree in batch mode for all the examples is 2.01 seconds. The *hungarian* task has an equal number of symbolic and numeric variables, and as one can see in Figure 3, there are only a few relatively costly revisions. For this task, the cost of building the tree in batch mode for all the examples is 0.43 seconds. The vowel task has all numeric variables, and Figure 4 shows that a large number of expensive updates occurred. Furthermore, one can see that the cost of each revision tends to increase with training. This is a case in which the value sets continue to grow throughout training. In batch mode, the cost of building the tree for all the examples is 5.63 seconds. One can see that several of the updates cost more than this.

On average, the cost of an incremental update will be lower than the cost of rebuilding the tree, despite an occasional expensive update. The figures do not show the cost of building a new tree at each point. It would be useful to be able to predict when an update will be more expensive than rebuilding the tree. In such a case, one could instead opt to rebuild the tree. It would be better still to improve upon the current ITI so that no update ever costs as much as rebuilding the tree.

7 Leave-One-Out Cross Validation

For some learning methods, it is possible to perform a leave-one-out cross validation inexpensively because it is easy to modify the classifier incrementally. For example, one can change an instance-based classifier simply by adding or subtracting an example from the instance base. This makes leave-one-out cross validation inexpensive because for each instance in the base, one removes it from the base, classifies it, and then puts it back into the base. The cross-validated

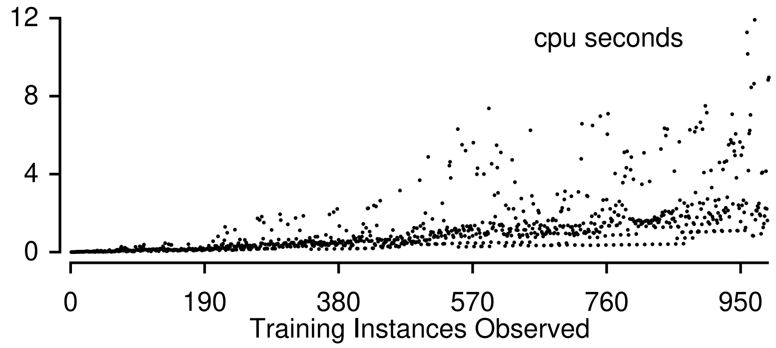

Figure 4. Incremental Update Cost for vowel

accuracy is the percentage of classifications that were correct.

With efficient tree revision, leave-one-out cross validation is practical for decision tree classification (Kohavi, 1995). One first builds a tree from all the examples using ITI in incremental or batch mode. Then for each example, one subtracts it from the tree, classifies it, and adds it to the tree. This requires adding a primitive that subtracts an example from the tree, which is straightforward; it is the inverse of adding an example. When an example is subtracted or added, the algorithm uses the indirect metric to identify the best test at each decision node. The cost of subtracting an example and adding it back is dramatically less, on average, than the cost of building the tree from scratch.

8 Software

The ITI system is available via http://www-ml.cs.umass.edu/iti.html or anonymous ftp to ftp.cs.umass.edu on directory /pub/iti. The distribution includes the C source code for the two algorithms ITI and DMTI, and for several additional small programs for running experiments, plotting performance graphs, and plotting decision trees. The tree plotting program PST generates PostScript code that draws a tree on as many pages as necessary for the specified font and pointsize.

Everything discussed in this article has been implemented except for the ITI lazy mode, the DMTI expected-classification-expense metric, and the DMTI expected-misclassification-cost metric. Several useful operators have been implemented that have not been discussed here, such as save-tree and restore-tree. In addition, a Kolmogorov-Smirnoff distance attribute selection metric can be selected as the indirect metric (Utgoff & Clouse, 1996) instead of the default.

In the implementation, every set of information items kept at a decision node is maintained as an AVL tree, which is an almost-balanced binary search tree (Wirth, 1976). This organization provides $O(\log n)$ insert, delete and lookup. Specifically, the set of variables is maintained at a decision node as an attached AVL-tree of variables. For each variable (node) in this attached AVL-tree, the set of observed values for that variable is maintained as its own attached AVL-tree. Similarly, for each value (node) in that attached AVL-tree, the set of observed classes with frequency counts, is kept as an attached AVL-tree. This tree of trees of trees is independent of the semantics of the decision tree itself, and serves merely as an efficient scheme for tracking the information that must be maintained at the decision node. Due to this organization, neither a large number of variables, nor a large number of values, nor a large number of classes is debilitating computationally. This is all the information that is needed to evaluate all of the possible binary tests that are permitted at a decision node. It does however bring some inefficiency when merging two AVL-trees, for example two sorted lists of tagged values, making the cost $O(n \log n)$ instead of $O(n)$.

9 Related Work

The incremental tree induction algorithm ID5R (Utgoff, 1989) demonstrated the basic process of tree revision. It did not handle numeric variables, multiclass tasks, or missing values, and did not include any prepruning or postpruning method for avoiding overfitting. The first version of ITI (Utgoff, 1994) had an awkward manner for handling numeric variables and for handling missing values, that have been replaced here. Schlimmer and Fisher's (1986) ID4 demonstrated incremental tree induction through test revision and discarding of subtrees.

Crawford (1989) has constructed an incremental version of the CART algorithm (Breiman, Friedman, Olshen & Stone, 1984). When a new example is received, if a new test would be picked at a decision node, a new subtree with the new test is constructed by building the new subtree from scratch from the corresponding subset of the training examples. Crawford notes that this approach is expensive, and proposes an alternative that invokes tree rebuilding less often. Van de Velde (1990) designed IDL, based on ID4 and ID5, with the goal of finding trees smaller than those that result from the standard top-down induction methods. Lovell and Bradley (1996) present the MSC algorithm, which refines a decision tree incrementally, with limited backtracking, making it dependent on the order of the presented training examples.

Fisher (1996) presents a method for optimizing a hierarchical clustering built initially by COBWEB. It implements a hill-climbing search through the space of clusterings, attempting to find an improved clustering according to specified metric, similar in spirit to DMTI. Cockett and Herrera (1990) present an algebraic approach to finding irreducible trees. Kalles and Morris (1996) have devised a scheme to reduce the number of times the test selection metric must be evaluated.

10 Conclusions

This article has presented a set of fundamental tree revision operators, and shown how two decision tree induction algorithms can be built from them. The ITI algorithm performs incremental decision tree induction on symbolic or numeric variables, and handles noise and missing values. The algorithm also includes a virtual pruning mechanism that can operate in conjunction with a tree induction algorithm. ITI is suitable for embedding in an application that receives or creates new examples online, such as knowledge maintenance systems. For tasks with no numeric variables, the cost of tree revision is largely independent of the number of examples that have been incorporated. For tasks with numeric variables that have large value sets, the cost of tree revision can grow noticeably with the number of examples.

The non-incremental DMTI algorithm uses an attribute selection metric that is a function of a tree instead of a function of counting information kept at a node. This makes it possible to choose from among a set of trees based on a direct measure of tree quality. It also lends itself to studies of how well the indirect metrics do at identifying tests that lead to induction of the most preferred trees. DMTI is suitable for producing trees that are intended to minimize a specified objective function. Due to DMTI's greater computational expense and associated greater minimization ability, it is suitable when one is willing to spend extra time to produce a superior tree.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. IRI-9222766, by a grant from the Digital Equipment Corporation, and by a grant to Ross Quinlan from the Australian Research Council. Ross Quinlan made many excellent suggestions during the first author's Summer 1993 stay at the University of Sydney. Doina Precup, David Skalak, Gunnar Blix, and David Jensen provided very helpful comments. Doug Fisher suggested many improvements that have greatly improved the presentation. The idea of implementing an efficient leave-one-out cross validation was suggested independently by each of Ron Kohavi and Mike Pazzani.

Many of the UCI tasks originate from sources outside the machine learning community. The audiology data file originates from Professor Jergen at Baylor College of Medicine. The breast-cancer data file, lymphography data file, and primary-tumor data file come from M. Zwitter and M. Soklic of the University Medical Centre at the Institute of Oncology in Ljubljana. The bupa data file was provided by Richard S. Forsyth of BUPA Medical Research Ltd. The cleveland and va data files were created by Robert Detrano of the Long Beach and Cleveland Clinic Foundation. The hungarian data file was compiled by Andras Janosi at the Hungarian Institute of Cardiology in Budapest. The switzerland data was produced by William Steinbrunn of the University Hospital in Zurich, and Matthias Pfisterer of the University Hospital in Basel. The remaining UCI data files not mentioned here by name were also provided to UCI by generous donors.

A Appendix

The tables here show the point estimates and standard deviations for each of the measurements for each of the algorithm-task combinations discussed in Section 5. The point estimates and deviations are presented as a pair of tables because a single table with all this information would be too large. For each of the variables measured, there is a pair of tables for the case in which pruning is turned off, and another pair of tables for when pruning is turned on. See Section 5.2 for an explanation of the algorithm names.

Table 6. Accuracy (no pruning)

See Table 7 for associated standard deviations

Task	DM	DE	IE	DL	I1	I2	C1	C2	Mean
audio-no-id	81.7	76.1	81.7	83.0	80.4	80.4	83.5	75.7	80.3
balance-scale	76.8	77.8	77.0	75.4	76.5	78.1	76.0	78.3	77.0
bc-wisc	95.6	94.4	94.4	94.9	93.6	93.6	93.3	93.7	94.2
breast-cancer	66.6	72.8	66.2	65.9	64.5	63.8	63.8	66.2	66.2
bupa	64.6	65.7	64.0	65.4	61.1	62.0	61.4	64.6	63.6
chess-551x39	92.1	86.8	94.5	93.4	92.3	92.3	91.6	92.5	91.9
cleveland	49.7	50.3	46.8	49.4	46.8	48.4	46.1	46.8	48.0
crx	80.3	79.6	79.9	79.7	80.1	81.6	80.3	80.9	80.3
fayyad	89.1	84.5	87.3	82.7	88.2	86.4	87.3	86.4	86.5
glass-no-id	68.6	70.0	65.9	67.7	66.4	66.8	66.4	67.7	67.4
hepatitis	80.0	71.3	83.1	80.6	78.1	81.9	76.9	76.9	78.6
horse-dead	54.5	63.6	58.2	62.7	64.5	65.5	64.5	61.8	61.9
horse-sick	95.6	95.6	96.9	96.9	96.9	96.3	95.0	95.6	96.1
hungarian	78.0	75.3	73.7	74.3	75.0	78.0	73.7	75.7	75.5
hypothyroid	98.7	98.2	99.0	98.9	98.6	98.9	99.0	99.1	98.8
ionosphere	89.4	87.8	93.6	89.2	93.9	93.3	91.1	91.4	91.2
iris	95.6	95.0	93.8	95.6	94.4	95.0	93.1	94.4	94.6
landsat	81.1	80.0	80.5	80.5	81.3	82.4	81.6	82.1	81.2
led24	52.9	52.9	55.2	51.9	52.4	61.4	58.1	61.0	55.7
led7	65.7	68.6	67.1	67.1	67.6	65.7	68.6	68.1	67.3
lenses	63.3	73.3	70.0	63.3	70.0	73.3	70.0	73.3	69.6
lung-cancer	50.0	42.5	37.5	35.0	42.5	40.0	47.5	37.5	41.6
lymphography	76.0	80.0	80.0	78.7	76.0	70.7	73.3	77.3	76.5
monks-1	100.0	100.0	100.0	100.0	97.0	91.6	96.1	95.2	97.5
monks-2	99.5	100.0	99.5	100.0	96.8	93.9	42.7	44.5	84.6
monks-3	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
mplex-11	100.0	100.0	100.0	100.0	100.0	99.9	100.0	99.8	100.0
mplex-6	100.0	100.0	90.0	100.0	84.3	65.7	87.1	57.1	85.5
mushroom	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
nettalk	84.0	82.4	83.8	84.9	83.8	83.4	82.7	82.1	83.4
pima	70.0	69.0	71.0	70.6	69.9	70.0	69.7	69.6	70.0
post-op	66.0	62.0	64.0	60.0	57.0	54.0	60.0	63.0	60.8
primary-tumor	34.1	35.9	36.8	34.4	36.5	36.2	37.6	40.9	36.5
promoter	70.9	78.2	80.0	80.9	77.3	78.2	77.3	80.9	78.0
road	77.9	75.5	77.0	78.5	77.4	78.7	79.2	79.2	77.9
soybean	92.9	91.9	93.5	91.7	93.3	93.0	91.0	90.3	92.2
splice	92.4	91.7	92.0	91.3	91.7	92.8	91.6	91.8	91.9
switzerland	36.2	34.6	35.4	36.2	30.0	33.1	25.4	32.3	32.9
tictactoe	76.8	75.1	75.5	76.7	72.0	73.8	66.4	66.3	72.8
usama-mys	76.8	78.4	77.4	76.3	77.4	78.4	83.7	82.6	78.9
va	30.0	34.3	25.2	29.0	20.5	23.3	29.5	28.1	27.5
votes	91.8	92.0	91.8	92.0	92.5	92.7	94.3	95.7	92.9
vowel	81.9	79.4	78.9	81.5	79.6	78.5	81.0	79.2	80.0
waveform	68.1	70.0	65.8	66.1	66.5	65.8	68.7	69.7	67.6
wine	95.6	93.9	96.1	95.0	95.0	95.0	92.8	93.3	94.6
zoo	96.4	95.5	95.5	96.4	96.4	91.8	97.3	92.7	95.2
Mean	78.0	77.9	77.7	77.7	76.9	76.6	76.0	75.7	77.1

Table 7. Standard Deviation for Accuracy (no pruning)

See Table 6 for associated point estimates

Task	DM	DE	IE	DL	I1	I2	C1	C2
audio-no-id	7.5	8.1	9.9	7.6	9.0	7.1	5.1	9.6
balance-scale	4.2	3.5	4.4	4.0	3.7	2.8	3.4	4.1
bc-wisc	1.5	2.1	2.3	1.8	3.1	3.4	2.4	2.8
breast-cancer	7.2	9.3	11.3	5.9	6.0	4.9	6.0	6.9
bupa	6.5	7.5	6.5	7.4	6.9	5.3	4.3	5.3
chess-551x39	3.1	3.9	3.1	2.1	1.8	3.0	2.9	4.0
cleveland	6.8	6.9	10.1	6.9	7.2	6.8	7.8	4.1
crx	5.2	3.1	3.6	4.9	6.2	4.9	4.1	3.5
fayyad	6.8	9.1	8.3	9.5	7.1	7.3	6.0	7.3
glass-no-id	9.0	7.1	12.7	10.1	11.5	10.4	9.1	8.7
hepatitis	9.6	11.3	6.3	7.1	8.9	7.1	6.9	4.9
horse-dead	19.9	10.0	14.2	16.0	12.5	12.1	8.6	9.8
horse-sick	2.9	4.9	3.1	3.1	3.1	4.1	4.7	4.0
hungarian	8.7	4.8	6.2	7.2	4.8	4.3	5.0	6.8
hypothyroid	0.6	0.7	0.5	0.4	0.6	0.7	0.5	0.8
ionosphere	5.2	5.2	3.5	4.2	3.7	3.6	3.0	3.2
iris	4.0	4.7	4.0	4.0	5.2	4.7	7.6	7.6
landsat	3.2	3.3	2.3	3.2	1.7	2.1	3.0	2.2
led24	8.4	8.6	12.1	10.3	13.6	11.0	11.6	9.0
led7	13.9	12.8	13.0	13.9	13.1	12.0	12.1	10.9
lenses	18.0	24.9	18.0	18.0	18.0	20.0	18.0	20.0
lung-cancer	19.4	22.5	16.8	16.6	22.5	16.6	26.1	16.8
lymphography	10.4	7.3	9.9	12.9	12.4	13.7	11.2	12.4
monks-1	0.0	0.0	0.0	0.0	2.0	7.4	3.9	4.5
monks-2	1.4	0.0	0.9	0.0	3.7	6.3	9.8	9.3
monks-3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
mplex-11	0.0	0.0	0.0	0.0	0.0	0.4	0.0	0.5
mplex-6	0.0	0.0	12.9	0.0	14.9	22.3	13.5	20.2
mushroom	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
nettalk	1.7	1.0	1.0	1.7	1.2	1.0	1.8	1.4
pima	4.7	5.2	4.1	2.9	3.7	3.5	3.6	4.3
post-op	9.2	8.7	9.2	10.0	9.0	12.8	15.5	10.0
primary-tumor	6.1	6.1	6.1	5.4	6.3	7.1	6.8	5.3
promoter	13.4	11.6	10.6	16.0	11.7	10.9	10.9	14.3
road	2.3	2.1	1.4	2.9	2.3	2.5	2.2	1.9
soybean	2.9	2.9	2.4	3.2	3.0	2.4	2.2	2.8
splice	1.2	1.4	1.8	0.8	0.9	0.8	2.0	2.1
switzerland	11.4	8.6	13.8	14.2	12.6	9.8	8.5	9.6
tictactoe	1.6	1.7	2.6	2.3	1.7	1.8	2.8	2.0
usama-mys	6.7	7.2	8.2	6.8	9.4	7.6	8.9	7.1
va	10.4	9.0	8.0	10.7	7.7	6.9	11.6	12.7
votes	4.2	3.1	4.1	3.3	4.1	3.2	4.6	3.7
vowel	3.6	3.7	5.0	4.0	4.5	4.2	4.5	4.5
waveform	6.5	8.0	6.9	8.6	11.1	9.3	9.5	10.4
wine	4.2	5.2	5.6	4.6	4.6	5.2	6.1	6.0
zoo	4.5	4.5	4.5	4.5	4.5	6.4	4.2	6.8

Table 8. Accuracy (pruning)

See Table 9 for associated standard deviations

Task	DM	IE	C1	I2	I1	C2	DL	DE	Mean
audio-no-id	79.1	81.7	84.8	79.6	79.6	77.8	75.7	64.3	77.8
balance-scale	78.1	77.6	77.5	78.9	78.9	77.5	78.1	77.5	78.0
bc-wisc	94.3	94.6	94.9	93.7	93.7	94.9	94.7	94.7	94.4
breast-cancer	71.4	69.7	75.5	68.6	68.6	75.5	73.1	72.8	71.9
bupa	67.4	67.1	65.1	66.3	66.3	64.6	62.0	64.6	65.4
chess-551x39	92.0	94.1	92.0	92.0	92.0	91.1	89.6	84.3	90.9
cleveland	50.3	49.4	47.1	51.6	51.6	46.8	51.3	53.9	50.2
crx	83.7	85.3	82.9	82.6	82.6	83.4	85.3	84.4	83.8
fayyad	87.3	87.3	86.4	84.5	84.5	86.4	81.8	80.9	84.9
glass-no-id	71.4	65.0	67.3	66.8	66.8	69.1	64.1	65.0	66.9
hepatitis	85.0	83.1	79.4	80.6	80.6	77.5	81.9	80.0	81.0
horse-dead	62.7	59.1	65.5	60.0	60.0	65.5	63.6	60.9	62.2
horse-sick	94.4	95.6	95.6	95.6	95.6	95.6	93.8	91.9	94.8
hungarian	80.3	80.3	78.0	81.3	81.3	78.3	78.3	79.0	79.6
hypothyroid	99.2	99.2	99.0	99.0	99.0	99.1	99.0	99.0	99.1
ionosphere	90.8	93.3	91.1	92.5	92.5	91.4	91.4	84.7	91.0
iris	95.0	94.4	94.4	94.4	94.4	94.4	92.5	93.8	94.1
landsat	84.7	84.9	84.5	85.8	85.7	85.1	85.0	83.8	84.9
led24	61.9	62.9	61.4	62.9	62.4	62.4	60.5	57.1	61.4
led7	68.6	66.2	70.0	65.7	66.7	69.5	68.6	70.5	68.2
lenses	90.0	83.3	83.3	83.3	83.3	83.3	70.0	90.0	83.3
lung-cancer	60.0	35.0	47.5	32.5	32.5	42.5	22.5	37.5	38.8
lymphography	69.3	76.0	76.7	73.3	73.3	78.0	69.3	69.3	73.2
monks-1	100.0	100.0	96.8	91.6	91.6	96.8	100.0	100.0	97.1
monks-2	93.6	96.1	65.9	92.3	92.3	65.9	89.3	85.2	85.1
monks-3	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
mplex-11	100.0	100.0	99.8	99.7	99.7	99.8	100.0	100.0	99.9
mplex-6	100.0	68.6	71.4	64.3	64.3	57.1	65.7	72.9	70.5
mushroom	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
nettalk	82.9	83.3	80.9	83.0	83.0	80.6	81.8	79.9	81.9
pima	74.4	72.9	71.8	74.5	74.5	71.9	72.9	71.7	73.1
post-op	63.0	61.0	69.0	62.0	61.0	69.0	65.0	66.0	64.5
primary-tumor	38.5	38.5	40.9	38.2	37.6	40.9	42.6	40.0	39.7
promoter	71.8	79.1	78.2	79.1	79.1	77.3	71.8	69.1	75.7
road	78.7	82.2	81.0	81.4	81.4	81.1	79.6	79.9	80.6
soybean	92.0	93.6	92.5	93.3	93.3	92.2	90.1	87.8	91.9
splice	93.6	94.8	93.8	94.3	94.3	93.7	92.6	92.3	93.7
switzerland	35.4	33.1	26.9	33.1	33.1	33.1	46.9	40.8	35.3
tictactoe	78.4	75.9	68.9	74.5	74.7	68.1	73.2	75.4	73.6
usama-mys	84.2	84.2	85.8	84.2	84.2	83.2	81.1	83.7	83.8
va	27.1	24.8	26.7	23.3	23.8	26.7	36.7	30.0	27.4
votes	94.3	95.9	96.4	95.0	95.0	96.6	94.3	93.6	95.1
vowel	77.1	76.2	80.1	76.1	76.1	78.9	71.3	70.6	75.8
waveform	70.0	64.5	70.0	65.5	65.5	70.0	65.5	67.4	67.3
wine	95.6	93.9	93.3	95.0	95.0	93.3	91.7	92.2	93.7
zoo	88.2	90.9	94.5	94.5	94.5	91.8	92.7	91.8	92.4
Mean	79.5	78.1	77.9	77.5	77.5	77.3	76.9	76.7	77.7

Table 9. Standard Deviation for Accuracy (pruning)

See Table 8 for associated point estimates

Task	DM	IE	C1	I2	I1	C2	DL	DE
audio-no-id	3.8	7.0	5.6	8.5	8.5	6.6	8.7	8.0
balance-scale	3.6	3.4	3.2	3.3	3.3	3.2	3.7	3.5
bc-wisc	2.5	3.1	2.3	2.7	2.7	2.3	1.9	1.8
breast-cancer	6.4	4.8	3.9	6.1	6.1	3.9	3.7	4.7
bupa	4.3	4.5	5.7	5.1	5.1	5.6	8.5	4.8
chess-551x39	3.8	3.5	3.9	4.2	4.2	4.8	4.1	5.6
cleveland	8.2	6.1	6.8	5.8	5.8	5.4	7.0	4.3
crx	2.1	3.1	3.8	2.4	2.4	2.4	3.5	3.6
fayyad	8.3	8.3	7.3	8.2	8.2	7.3	11.5	10.3
glass-no-id	4.6	12.2	8.3	10.2	10.2	7.3	9.2	9.1
hepatitis	7.5	5.6	6.9	9.0	9.0	5.7	4.4	6.7
horse-dead	13.1	17.4	8.9	15.3	15.3	7.9	10.0	10.0
horse-sick	5.2	4.9	4.0	4.0	4.0	4.0	2.8	4.9
hungarian	5.7	4.8	6.0	7.9	7.9	4.0	7.8	6.2
hypothyroid	0.4	0.6	0.7	0.6	0.6	0.6	0.5	0.4
ionosphere	5.1	4.5	3.0	4.3	4.3	3.6	4.9	6.2
iris	6.1	6.5	7.6	6.5	6.5	7.6	6.1	6.2
landsat	2.5	3.0	3.3	1.8	1.8	2.6	2.2	2.7
led24	7.4	10.4	8.4	9.9	9.4	9.4	10.0	12.8
led7	12.3	13.9	9.5	12.0	14.0	10.7	13.8	13.4
lenses	21.3	22.4	22.4	22.4	22.4	22.4	31.4	21.3
lung-cancer	25.5	16.6	26.1	16.0	16.0	22.5	17.5	28.0
lymphography	9.0	9.0	11.6	13.7	13.7	11.9	10.8	10.0
monks-1	0.0	0.0	3.7	7.2	7.2	3.7	0.0	0.0
monks-2	2.6	2.9	0.0	6.2	6.2	0.0	4.4	4.0
monks-3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
mplex-11	0.0	0.0	0.4	0.7	0.7	0.4	0.0	0.0
mplex-6	0.0	14.0	20.2	16.0	16.0	19.2	11.4	7.7
mushroom	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0
nettalk	1.4	1.1	2.1	1.0	1.0	1.9	0.7	0.9
pima	3.3	4.4	3.9	5.1	5.1	4.2	3.2	5.0
post-op	7.8	12.2	8.3	16.0	15.1	8.3	6.7	8.0
primary-tumor	5.8	5.8	6.5	7.0	7.1	6.4	7.7	6.5
promoter	11.8	12.2	13.0	11.5	11.5	14.2	17.5	14.8
road	2.4	1.4	2.2	2.0	2.0	2.2	1.4	1.6
soybean	1.6	1.6	2.7	2.2	2.2	2.4	2.0	3.8
splice	1.8	1.6	1.3	1.2	1.2	1.3	1.5	1.4
switzerland	15.5	12.9	10.5	10.3	10.3	7.7	10.6	11.4
tictactoe	2.5	1.8	2.1	1.9	2.1	2.3	1.7	1.4
usama-mys	7.1	5.3	8.2	4.7	4.7	7.4	7.9	8.3
va	7.4	11.2	12.6	7.5	8.2	7.7	10.0	9.0
votes	3.6	3.3	3.1	4.5	4.5	3.3	3.6	4.4
vowel	4.5	3.1	4.4	4.8	4.8	4.5	4.8	4.4
waveform	6.0	12.2	9.7	10.6	10.6	10.7	10.6	5.3
wine	4.2	7.6	6.0	5.2	5.2	6.0	4.5	3.7
zoo	5.8	4.1	4.5	4.5	4.5	6.4	5.5	7.6

Table 10. Leaves (no pruning)

See Table 11 for associated standard deviations

Task	C1	C2	I1	IE	DE	DL	DM	I2	Mean
audio-no-id	74.8	41.7	45.4	46.9	56.2	42.3	41.5	33.9	47.8
balance-scale	138.2	61.7	138.8	143.9	139.2	135.6	134.7	55.4	118.4
bc-wisc	33.8	21.7	39.9	33.4	29.6	27.5	27.5	26.7	30.0
breast-cancer	277.8	143.4	98.5	95.5	77.0	73.4	75.5	69.7	113.8
bupa	77.6	53.4	88.7	88.3	71.1	71.3	67.5	71.7	73.7
chess-551x39	72.2	41.0	71.1	61.1	93.0	56.3	56.4	48.4	62.4
cleveland	95.0	56.8	102.2	102.6	85.6	80.9	79.5	66.7	83.7
crx	141.8	84.0	92.9	95.6	84.6	75.4	72.5	67.5	89.3
fayyad	12.4	7.6	12.4	12.4	12.1	12.2	11.4	8.6	11.1
glass-no-id	42.8	28.4	43.9	41.1	42.3	37.4	35.5	31.3	37.8
hepatitis	22.0	15.2	22.6	20.1	18.8	16.8	16.6	13.9	18.3
horse-dead	20.5	15.1	22.6	21.9	18.6	17.8	16.9	18.3	19.0
horse-sick	6.6	6.2	5.7	5.8	6.4	5.7	5.7	4.6	5.8
hungarian	54.8	34.1	57.9	54.6	46.0	42.7	41.6	40.4	46.5
hypothyroid	36.7	15.9	42.9	38.0	36.8	33.4	34.1	22.9	32.6
ionosphere	20.4	16.2	23.0	21.4	20.7	17.8	18.1	17.8	19.4
iris	9.8	5.3	9.2	9.4	8.8	8.3	8.3	5.7	8.1
landsat	134.2	83.2	141.1	136.5	127.4	114.5	112.8	92.4	117.8
led24	63.7	35.6	63.7	62.5	63.6	57.7	57.4	33.3	54.7
led7	33.1	18.1	46.0	43.6	44.7	44.4	44.4	26.3	37.6
lenses	7.9	4.2	6.6	7.3	6.3	6.6	6.5	3.9	6.2
lung-cancer	10.5	7.4	11.0	11.0	8.5	8.1	8.1	7.7	9.0
lymphography	63.3	32.3	26.2	30.3	23.1	20.5	20.8	20.1	29.6
monks-1	84.3	71.7	40.2	8.4	9.1	9.8	9.8	36.6	33.7
monks-2	262.5	92.4	51.0	46.5	41.0	41.0	41.0	46.4	77.7
monks-3	14.0	14.0	5.0	5.1	5.0	5.0	5.0	5.0	7.3
mplex-11	91.8	91.3	92.2	60.2	16.0	16.0	16.0	92.0	59.4
mplex-6	21.6	15.1	20.5	16.8	8.0	8.0	8.0	16.1	14.3
mushroom	25.6	25.6	12.9	12.3	9.6	10.0	9.0	12.9	14.7
nettalk	10528.4	5065.8	840.3	823.8	923.5	742.1	754.2	560.7	2529.9
pima	134.2	93.8	153.4	151.2	122.1	113.6	115.4	124.1	126.0
post-op	48.8	21.2	37.7	35.6	33.9	31.3	30.2	22.4	32.6
primary-tumor	163.5	66.7	174.4	171.2	169.6	157.0	157.2	84.0	142.9
promoter	30.7	24.1	12.3	11.9	10.6	9.7	9.8	8.7	14.7
road	302.2	185.6	327.9	311.8	312.9	265.2	269.2	216.2	273.9
soybean	172.0	118.3	67.3	64.5	68.8	60.7	60.3	54.5	83.3
splice	1002.7	721.3	142.1	136.6	150.1	120.3	118.8	98.8	311.3
switzerland	52.7	32.5	53.6	52.5	48.5	45.2	44.9	35.3	45.6
tictactoe	2369.6	1024.2	1348.1	1220.0	977.0	962.3	957.3	705.8	1195.5
usama-mys	18.7	12.8	28.1	25.7	22.7	20.3	21.0	17.6	20.9
va	80.2	50.6	103.0	101.0	84.3	80.1	80.7	62.2	80.3
votes	24.6	13.9	28.1	24.8	23.6	22.0	21.9	17.5	22.1
vowel	136.2	102.7	154.7	143.9	135.7	116.9	117.7	122.4	128.8
waveform	37.6	27.9	46.7	44.9	31.1	29.3	27.8	37.9	35.4
wine	8.6	5.6	7.8	6.7	6.3	5.8	5.8	5.7	6.5
zoo	14.0	10.2	10.1	9.8	9.6	9.6	9.6	8.3	10.2
Mean	371.2	189.5	108.0	101.5	94.3	84.5	84.4	69.1	137.8

Table 11. Standard Deviation for Leaves (no pruning)

See Table 10 for associated point estimates

Task	C1	C2	I1	IE	DE	DL	DM	I2
audio-no-id	7.4	7.7	2.6	2.3	1.6	1.6	1.6	2.5
balance-scale	3.9	3.8	3.7	2.3	2.1	2.9	3.7	2.8
bc-wisc	2.3	3.3	2.7	2.4	1.4	1.2	1.9	2.2
breast-cancer	26.1	16.0	5.6	5.9	3.1	3.4	3.8	3.7
bupa	3.6	3.7	6.5	8.7	4.0	3.0	2.8	6.5
chess-551x39	6.3	1.9	5.9	3.7	10.0	2.9	3.0	1.6
cleveland	4.6	2.4	4.4	3.7	3.5	2.1	1.7	4.4
crx	18.0	14.3	4.9	5.4	2.7	3.1	3.1	4.9
fayyad	1.5	0.7	1.6	1.6	1.3	1.6	1.1	1.2
glass-no-id	2.4	2.4	2.9	4.6	1.6	2.6	1.9	2.0
hepatitis	2.8	1.5	2.7	3.3	1.5	1.5	1.6	1.5
horse-dead	1.3	0.9	1.6	2.8	1.4	1.4	1.2	1.7
horse-sick	0.8	1.3	0.5	0.4	0.5	0.5	0.5	0.7
hungarian	3.4	3.9	2.0	3.1	4.1	2.1	1.6	1.4
hypothyroid	3.2	3.1	4.2	4.9	3.1	3.3	2.9	2.8
ionosphere	2.1	1.9	1.5	2.2	1.7	1.3	1.3	1.2
iris	1.3	0.8	1.1	1.4	1.1	0.8	0.8	0.8
landsat	4.0	5.4	5.0	6.0	6.6	4.2	3.5	4.1
led24	2.3	1.9	2.3	2.9	3.9	3.3	3.4	2.0
led7	2.2	1.1	1.8	2.0	2.0	1.8	1.8	1.0
lenses	1.2	0.9	1.1	1.4	0.8	1.1	1.0	0.3
lung-cancer	1.2	1.4	1.8	1.8	0.5	0.5	0.3	2.2
lymphography	6.3	7.3	2.5	3.6	1.9	1.6	1.4	3.1
monks-1	29.5	21.7	16.6	1.2	0.8	1.9	1.9	13.8
monks-2	9.2	4.3	9.7	6.4	0.0	0.0	0.0	5.3
monks-3	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0
mplex-11	22.4	22.1	20.4	14.6	0.0	0.0	0.0	20.4
mplex-6	2.4	1.0	2.0	3.1	0.0	0.0	0.0	0.9
mushroom	2.0	2.0	0.3	0.6	0.5	0.0	0.0	0.3
nettalk	110.6	118.5	11.7	12.6	19.7	12.1	8.5	7.7
pima	6.0	4.8	7.7	12.3	3.7	5.4	6.0	10.3
post-op	4.2	4.2	4.1	3.4	2.3	0.6	0.7	1.9
primary-tumor	4.8	4.1	3.3	4.1	4.2	2.6	3.5	4.3
promoter	3.7	1.9	1.8	1.6	0.9	0.8	1.0	1.3
road	9.1	5.8	9.5	6.6	13.6	5.1	6.3	7.5
soybean	11.8	6.2	3.2	2.2	4.2	2.6	1.8	2.6
splice	34.2	37.5	3.5	3.2	7.6	3.3	3.5	3.9
switzerland	3.1	2.5	2.8	3.8	3.7	2.0	2.5	2.2
tictactoe	20.0	23.8	18.1	33.7	18.1	27.6	29.0	10.1
usama-mys	1.8	1.9	1.9	2.2	2.1	1.2	1.6	1.3
va	5.5	3.9	6.8	5.1	3.6	2.4	2.5	4.7
votes	2.8	3.0	3.4	2.8	1.6	2.0	2.4	2.2
vowel	3.5	4.0	6.1	4.3	5.6	3.9	3.5	5.7
waveform	3.0	3.3	8.8	6.8	2.0	1.8	1.7	7.3
wine	1.6	0.9	1.1	0.8	0.8	0.4	0.4	1.0
zoo	1.0	2.8	0.8	0.6	0.5	0.5	0.5	0.8

Table 12. Leaves (pruning)

See Table 13 for associated standard deviations

Task	C1	C2	I1	I2	IE	DE	DL	DM	Mean
audio-no-id	47.4	30.2	26.8	26.8	29.1	26.0	22.6	20.0	28.6
balance-scale	52.3	40.0	48.7	47.0	49.4	43.8	41.9	41.0	45.5
bc-wisc	12.6	10.0	12.1	12.1	9.2	9.5	7.8	8.1	10.2
breast-cancer	9.0	8.0	21.4	21.4	26.1	11.7	12.4	15.2	15.6
bupa	48.1	37.3	30.1	30.1	29.1	17.3	17.5	23.1	29.1
chess-551x39	34.2	24.3	27.7	27.7	36.3	32.2	33.3	20.6	29.5
cleveland	71.6	45.7	42.6	42.5	38.7	33.1	33.7	29.4	42.2
crx	45.5	34.2	21.5	21.5	12.3	33.1	34.2	17.8	27.5
fayyad	7.2	7.1	6.9	6.9	7.7	6.0	6.0	7.0	6.9
glass-no-id	35.2	25.2	23.5	23.5	24.6	20.1	22.0	16.8	23.9
hepatitis	10.6	8.8	6.7	6.7	4.5	4.0	4.7	6.0	6.5
horse-dead	16.8	11.3	6.5	6.5	5.8	4.1	4.2	3.7	7.4
horse-sick	4.6	4.6	3.9	3.9	3.5	3.3	3.6	3.1	3.8
hungarian	15.6	13.8	10.0	10.0	8.6	7.3	7.5	8.7	10.2
hypothyroid	7.4	6.0	9.3	9.3	6.3	5.2	5.0	4.5	6.6
ionosphere	16.4	12.9	7.5	7.5	5.9	10.0	6.0	7.4	9.2
iris	5.5	5.1	3.7	3.7	3.7	3.0	3.6	3.1	3.9
landsat	62.6	42.7	34.5	33.8	37.5	33.3	32.9	30.3	38.5
led24	41.3	29.0	25.9	24.8	25.8	22.5	23.2	22.9	26.9
led7	22.1	17.5	29.6	22.7	29.9	26.9	26.3	26.6	25.2
lenses	3.9	3.6	3.2	3.2	3.1	3.0	3.0	3.0	3.2
lung-cancer	9.0	6.4	4.0	4.0	4.5	3.4	3.3	3.5	4.8
lymphography	28.2	18.2	10.9	10.9	12.0	9.7	9.2	7.4	13.3
monks-1	29.4	29.4	32.5	32.5	8.0	9.1	9.8	9.8	20.1
monks-2	1.0	1.0	40.9	40.9	40.1	34.5	35.7	34.8	28.6
monks-3	14.0	14.0	5.0	5.0	5.0	5.0	5.0	5.0	7.2
mplex-11	88.2	88.2	91.6	91.6	54.8	16.0	16.0	16.0	57.8
mplex-6	12.9	12.3	7.6	7.6	9.0	6.0	6.0	8.0	8.7
mushroom	25.6	25.6	12.9	12.9	12.2	9.6	10.0	9.0	14.7
nettalk	4610.8	2270.8	418.6	411.7	403.5	411.9	333.0	334.7	1149.4
pima	80.9	63.6	24.0	24.0	29.0	40.2	44.6	31.8	42.3
post-op	3.7	1.8	9.5	9.2	8.6	8.0	8.2	7.0	7.0
primary-tumor	95.3	43.5	78.2	62.0	79.6	74.3	67.0	67.6	70.9
promoter	17.5	16.3	6.3	6.3	5.5	4.2	3.9	4.4	8.1
road	175.2	119.3	95.0	94.6	78.6	85.9	83.6	81.3	101.7
soybean	77.0	62.5	34.1	34.0	27.9	39.8	35.4	32.8	42.9
splice	397.2	298.5	51.6	51.6	47.1	49.2	46.1	43.6	123.1
switzerland	40.5	28.0	26.3	26.1	24.6	18.6	17.4	16.4	24.7
tictactoe	611.4	484.4	485.6	468.3	450.6	328.4	338.1	383.1	443.7
usama-mys	12.2	10.7	4.9	4.9	7.6	7.3	12.0	5.7	8.2
va	57.8	36.9	46.5	46.2	46.4	32.9	32.4	33.2	41.5
votes	6.2	5.9	6.1	6.1	5.7	4.5	5.5	6.4	5.8
vowel	122.2	97.9	97.9	97.8	92.8	83.4	78.4	75.8	93.3
waveform	31.9	26.0	15.2	15.2	16.3	11.5	13.4	13.4	17.9
wine	5.8	5.5	5.4	5.4	5.5	5.0	4.7	4.1	5.2
zoo	11.9	10.1	7.0	7.0	6.8	8.7	8.5	6.5	8.3
Mean	155.1	91.2	43.9	42.8	40.8	36.1	34.3	33.9	59.8

Table 13. Standard Deviation for Leaves (pruning)

See Table 12 for associated point estimates

Task	C1	C2	I1	I2	IE	DE	DL	DM
audio-no-id	4.3	3.0	1.5	1.5	1.1	4.4	1.8	1.5
balance-scale	6.5	5.7	2.9	2.9	3.2	3.4	2.0	2.7
bc-wisc	2.6	2.0	1.3	1.3	1.9	3.5	2.7	1.6
breast-cancer	10.7	8.0	4.8	4.8	5.0	3.7	4.0	2.4
bupa	5.2	6.2	6.5	6.5	5.9	7.7	5.6	4.3
chess-551x39	7.8	3.2	4.7	4.7	4.1	8.4	6.4	1.6
cleveland	7.6	5.8	4.4	4.3	6.0	4.4	5.9	2.3
crx	13.3	7.5	8.2	8.2	3.6	6.2	8.9	2.7
fayyad	0.6	0.3	1.0	1.0	0.6	0.9	1.0	0.8
glass-no-id	3.1	2.9	3.0	3.0	2.4	4.9	5.7	1.7
hepatitis	3.8	2.3	0.8	0.8	2.3	1.1	1.4	0.8
horse-dead	2.5	3.2	2.2	2.2	1.4	1.9	1.9	1.0
horse-sick	0.8	0.8	0.5	0.5	0.5	0.5	1.2	0.3
hungarian	6.9	5.0	1.8	1.8	1.2	2.0	1.7	1.6
hypothyroid	1.9	1.0	1.3	1.3	1.9	0.6	0.4	0.5
ionosphere	2.4	2.3	0.8	0.8	0.3	2.3	0.4	0.5
iris	0.8	0.9	0.5	0.5	0.5	0.0	0.9	0.3
landsat	9.4	7.4	2.8	2.6	2.6	3.4	5.5	2.7
led24	3.2	1.3	1.8	1.5	1.4	1.4	1.9	1.1
led7	2.1	1.7	2.4	1.4	2.8	2.5	2.2	2.4
lenses	0.3	0.5	0.4	0.4	0.3	0.0	0.0	0.0
lung-cancer	1.8	1.5	1.2	1.2	1.6	1.7	1.9	0.5
lymphography	7.2	4.5	1.9	1.9	1.9	2.7	2.2	1.0
monks-1	5.0	5.0	10.9	10.9	0.0	0.8	1.9	1.9
monks-2	0.0	0.0	3.3	3.3	3.1	1.3	2.1	2.0
monks-3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
mplex-11	19.1	19.1	20.0	20.0	1.8	0.0	0.0	0.0
mplex-6	3.1	2.4	1.5	1.5	1.3	0.0	0.0	0.0
mushroom	2.0	2.0	0.3	0.3	1.6	0.5	0.0	0.0
nettalk	182.4	91.6	7.9	7.9	8.8	29.7	17.0	8.9
pima	7.5	3.1	7.2	7.2	10.4	14.4	8.2	4.9
post-op	2.1	1.6	2.3	2.4	3.1	1.4	1.2	1.2
primary-tumor	7.4	4.4	3.2	4.6	3.8	8.2	4.4	3.4
promoter	3.1	2.8	1.0	1.0	0.7	0.9	0.5	1.0
road	4.6	7.5	5.4	5.7	3.0	12.4	13.8	5.7
soybean	7.8	5.8	2.8	2.9	3.0	5.2	2.8	1.8
splice	32.1	26.1	2.2	2.2	4.5	6.2	4.0	3.0
switzerland	4.5	2.9	3.4	3.3	3.6	4.3	4.0	1.6
tictactoe	33.9	34.2	11.2	10.1	19.2	11.2	11.0	6.9
usama-mys	2.1	1.7	1.8	1.8	1.8	3.3	4.7	0.8
va	7.3	6.0	2.9	3.1	3.0	3.8	3.9	3.1
votes	0.4	0.3	0.8	0.8	0.8	0.7	0.8	1.4
vowel	3.0	3.8	4.9	5.0	5.1	7.0	6.1	3.3
waveform	2.5	3.0	2.2	2.2	2.1	2.4	5.2	1.8
wine	1.2	0.7	0.5	0.5	0.5	0.9	0.5	0.3
zoo	2.3	2.7	0.0	0.0	0.4	0.6	0.8	0.5

Table 14. Expected-Tests (no pruning)

See Table 15 for associated standard deviations

Task	IE	I1	I2	DL	DM	DE	Mean
audio-no-id	12.2	11.1	10.3	10.7	8.6	6.0	9.8
balance-scale	9.0	7.0	5.9	7.0	6.9	6.5	7.0
bc-wisc	8.2	7.4	7.1	4.5	4.3	3.8	5.9
breast-cancer	15.9	12.9	12.4	8.5	8.9	6.7	10.9
bupa	25.3	21.3	20.9	15.7	12.3	9.6	17.5
chess-551x39	19.4	9.4	8.8	14.8	14.1	7.1	12.3
cleveland	13.6	12.6	11.9	8.3	7.6	6.4	10.1
crx	18.6	11.1	10.6	9.4	7.7	6.3	10.6
fayyad	5.9	4.7	4.3	5.3	4.5	4.4	4.9
glass-no-id	9.6	9.2	8.5	8.1	6.5	6.0	8.0
hepatitis	6.4	6.6	5.7	5.2	4.8	3.9	5.4
horse-dead	12.2	11.2	10.7	6.9	5.8	4.9	8.6
horse-sick	2.8	2.6	2.1	2.7	2.2	2.2	2.4
hungarian	10.5	9.8	9.2	7.3	6.6	5.8	8.2
hypothyroid	6.5	6.9	6.0	4.1	3.6	2.9	5.0
ionosphere	11.0	10.7	10.4	6.0	5.7	4.4	8.0
iris	4.0	2.9	2.3	2.7	2.7	2.6	2.9
landsat	15.5	13.0	12.5	10.5	10.0	7.6	11.5
led24	6.4	6.2	5.3	6.1	5.9	5.8	5.9
led7	5.9	5.8	5.0	5.9	5.8	5.6	5.7
lenses	2.9	2.2	1.7	2.4	2.2	2.1	2.2
lung-cancer	5.6	4.8	4.1	3.5	3.3	3.3	4.1
lymphography	11.1	7.8	7.4	5.0	4.9	4.4	6.8
monks-1	4.0	4.6	4.6	3.5	3.5	3.4	3.9
monks-2	5.7	5.5	5.5	5.5	5.5	5.3	5.5
monks-3	2.6	2.1	2.1	2.1	2.1	2.1	2.2
mplex-11	5.6	6.1	6.1	4.0	4.0	4.0	5.0
mplex-6	3.9	4.1	3.9	3.0	3.0	3.0	3.5
mushroom	7.3	5.0	5.0	3.7	3.8	2.8	4.6
nettalk	33.6	26.7	26.0	22.1	22.2	19.2	25.0
pima	23.5	18.1	17.6	12.1	11.8	8.2	15.2
post-op	8.2	8.1	7.2	6.2	5.8	5.2	6.8
primary-tumor	10.6	10.2	8.8	9.6	9.1	7.7	9.4
promoter	4.3	3.9	3.5	3.6	3.5	3.1	3.7
road	22.9	18.3	17.6	15.0	14.8	11.2	16.6
soybean	11.0	8.4	8.2	7.3	6.9	5.8	7.9
splice	8.2	7.3	6.9	6.8	6.8	6.5	7.1
switzerland	12.8	12.8	12.1	8.7	8.3	5.9	10.1
tictactoe	11.3	10.5	9.9	10.1	10.0	9.5	10.2
usama-mys	9.3	9.4	8.6	6.5	5.8	4.7	7.4
va	18.9	17.8	17.0	10.3	9.6	6.8	13.4
votes	4.7	3.5	3.1	3.8	3.6	3.2	3.7
vowel	14.1	13.8	13.5	9.8	9.2	7.9	11.4
waveform	12.8	13.3	12.7	7.5	6.4	5.5	9.7
wine	2.8	3.1	2.7	2.7	2.7	2.6	2.8
zoo	4.1	2.9	2.8	3.5	3.1	2.7	3.2
Mean	10.4	9.0	8.4	7.1	6.7	5.5	7.9

Table 15. Standard Deviation for Expected-Tests (no pruning)

See Table 14 for associated point estimates

Task	IE	I1	I2	DL	DM	DE
audio-no-id	0.9	0.9	1.0	0.7	0.9	0.1
balance-scale	0.2	0.1	0.1	0.2	0.1	0.1
bc-wisc	1.1	0.4	0.5	0.4	0.5	0.0
breast-cancer	0.7	0.6	0.6	1.0	1.7	0.3
bupa	5.0	2.3	2.3	2.2	1.8	1.3
chess-551x39	1.5	0.4	0.5	1.7	1.9	0.3
cleveland	1.0	1.2	1.1	0.9	0.9	0.2
crx	2.8	1.0	1.0	0.7	0.9	0.1
fayyad	0.5	0.4	0.4	0.9	0.2	0.3
glass-no-id	1.3	0.7	0.6	1.3	1.0	0.3
hepatitis	0.7	0.6	0.6	0.4	0.3	0.2
horse-dead	1.7	0.5	0.6	1.6	1.0	0.5
horse-sick	0.2	0.1	0.3	0.2	0.1	0.1
hungarian	1.3	0.3	0.3	0.8	0.6	0.2
hypothyroid	0.4	0.7	0.7	0.9	0.7	0.1
ionosphere	1.0	0.5	0.5	0.6	0.8	0.3
iris	0.6	0.2	0.2	0.2	0.2	0.2
landsat	0.9	1.0	0.9	1.0	1.0	0.5
led24	0.1	0.1	0.1	0.2	0.2	0.2
led7	0.1	0.1	0.1	0.2	0.2	0.1
lenses	0.3	0.2	0.1	0.4	0.2	0.2
lung-cancer	1.3	1.3	1.5	0.2	0.2	0.1
lymphography	1.7	1.1	1.1	0.3	0.3	0.2
monks-1	0.4	0.6	0.5	0.2	0.2	0.1
monks-2	0.2	0.1	0.1	0.0	0.0	0.0
monks-3	0.2	0.0	0.0	0.0	0.0	0.0
mplex-11	0.4	0.3	0.3	0.0	0.0	0.0
mplex-6	0.3	0.1	0.1	0.0	0.0	0.0
mushroom	0.4	0.1	0.1	0.0	0.0	0.1
nettalk	0.8	0.2	0.2	0.6	0.6	0.2
pima	3.5	3.0	3.0	1.9	2.3	0.4
post-op	0.7	0.4	0.4	0.6	0.4	0.2
primary-tumor	0.3	0.3	0.2	0.6	0.5	0.1
promoter	0.5	0.6	0.5	0.3	0.4	0.2
road	1.2	1.3	1.3	0.9	0.7	0.4
soybean	0.6	0.1	0.1	0.3	0.4	0.2
splice	0.2	0.1	0.1	0.2	0.3	0.2
switzerland	2.0	2.1	2.0	1.5	1.5	0.1
tictactoe	0.1	0.0	0.0	0.1	0.1	0.1
usama-mys	0.8	0.9	0.8	1.0	1.1	0.6
va	1.5	2.8	2.8	2.0	1.9	0.1
votes	0.3	0.4	0.3	0.3	0.2	0.3
vowel	0.5	0.6	0.7	0.9	0.8	0.4
waveform	2.0	2.9	2.8	0.9	0.8	0.5
wine	0.3	0.4	0.4	0.2	0.1	0.1
zoo	0.2	0.0	0.1	0.1	0.2	0.0

Table 16. Expected-Tests (pruning)

See Table 17 for associated standard deviations

Task	IE	I1	I2	DL	DM	DE	Mean
audio-no-id	11.3	9.7	9.7	7.9	7.5	5.2	8.5
balance-scale	7.2	5.7	5.7	5.8	5.7	5.4	5.9
bc-wisc	4.5	5.2	5.2	3.3	3.2	3.5	4.2
breast-cancer	11.5	8.0	8.0	3.8	4.5	3.5	6.5
bupa	18.7	15.5	15.5	9.7	9.7	7.1	12.7
chess-551x39	17.9	7.4	7.4	10.2	7.2	5.9	9.3
cleveland	10.8	10.3	10.3	7.2	5.8	5.4	8.3
crx	7.8	6.0	6.0	7.4	4.5	6.3	6.3
fayyad	4.9	3.8	3.8	3.7	3.9	3.5	3.9
glass-no-id	9.6	7.6	7.6	7.3	5.5	5.4	7.2
hepatitis	2.9	3.5	3.5	2.3	3.2	1.8	2.9
horse-dead	4.2	4.8	4.8	2.4	1.9	2.0	3.3
horse-sick	2.0	1.9	1.9	1.9	1.4	1.6	1.8
hungarian	6.2	4.6	4.6	3.3	3.7	2.7	4.2
hypothyroid	3.4	3.1	3.1	1.4	1.4	1.3	2.3
ionosphere	3.8	5.0	5.0	4.3	4.5	2.9	4.3
iris	2.4	1.9	1.9	1.9	1.7	1.7	1.9
landsat	11.3	9.1	9.1	7.3	6.7	5.6	8.2
led24	5.1	5.0	4.9	4.8	4.7	4.5	4.8
led7	5.3	5.0	4.7	5.0	5.0	4.8	5.0
lenses	1.6	1.5	1.5	1.5	1.5	1.5	1.5
lung-cancer	2.7	2.3	2.3	1.7	2.1	1.7	2.1
lymphography	8.3	6.0	6.0	3.8	3.4	3.2	5.1
monks-1	4.0	4.5	4.5	3.5	3.5	3.4	3.9
monks-2	5.5	5.4	5.4	5.3	5.3	5.1	5.3
monks-3	2.7	2.1	2.1	2.1	2.1	2.1	2.2
mplex-11	5.6	6.1	6.1	4.0	4.0	4.0	5.0
mplex-6	3.1	2.8	2.8	2.5	3.0	2.5	2.8
mushroom	7.6	5.0	5.0	3.7	3.8	2.8	4.7
nettalk	31.2	24.9	24.8	20.0	20.2	17.8	23.2
pima	14.6	8.6	8.6	10.5	8.6	8.3	9.9
post-op	4.5	4.9	4.9	3.4	3.1	3.3	4.0
primary-tumor	9.1	8.4	8.2	8.1	7.9	6.7	8.1
promoter	3.1	3.0	3.0	2.1	2.1	2.1	2.6
road	13.7	13.9	13.9	11.5	10.4	9.2	12.1
soybean	8.3	7.7	7.7	6.4	6.4	5.5	7.0
splice	7.8	5.5	5.5	5.2	5.3	4.9	5.7
switzerland	10.8	11.3	11.3	5.6	5.9	4.7	8.3
tictactoe	9.9	9.4	9.3	9.0	9.2	8.5	9.2
usama-mys	5.6	3.2	3.2	6.8	3.1	4.0	4.3
va	18.0	16.4	16.4	7.8	7.9	5.6	12.0
votes	2.8	2.1	2.1	2.3	2.2	1.7	2.2
vowel	13.1	12.8	12.8	9.9	8.9	7.4	10.8
waveform	6.3	6.5	6.5	5.1	4.8	3.9	5.5
wine	2.8	2.6	2.6	2.3	2.1	2.3	2.4
zoo	3.4	2.7	2.7	3.4	2.7	2.6	2.9
Mean	7.8	6.6	6.6	5.4	5.0	4.5	6.0

Table 17. Standard Deviation for Expected-Tests (pruning)

See Table 16 for associated point estimates

Task	IE	I1	I2	DL	DM	DE
audio-no-id	0.7	0.9	0.9	1.0	0.7	0.5
balance-scale	0.3	0.1	0.1	0.2	0.1	0.1
bc-wisc	1.1	0.5	0.5	0.6	0.3	0.7
breast-cancer	2.4	1.6	1.6	0.7	0.9	0.8
bupa	3.3	2.3	2.3	2.3	2.2	2.4
chess-551x39	1.4	1.0	1.0	0.7	0.5	1.0
cleveland	1.7	1.3	1.3	1.4	0.7	0.6
crx	2.3	1.2	1.2	1.9	0.5	1.8
fayyad	0.4	0.5	0.5	0.6	0.2	0.3
glass-no-id	1.6	0.8	0.8	2.0	0.9	0.8
hepatitis	1.7	0.5	0.5	0.7	0.4	0.4
horse-dead	1.0	1.6	1.6	0.9	0.5	0.9
horse-sick	0.4	0.3	0.3	0.7	0.1	0.2
hungarian	1.1	0.6	0.6	0.7	0.7	0.5
hypothyroid	1.2	0.6	0.6	0.1	0.0	0.0
ionosphere	0.2	0.6	0.6	0.3	0.8	0.3
iris	0.3	0.2	0.2	0.3	0.1	0.0
landsat	0.8	0.7	0.7	0.9	0.4	0.3
led24	0.1	0.1	0.1	0.1	0.1	0.1
led7	0.2	0.1	0.1	0.1	0.2	0.2
lenses	0.1	0.1	0.1	0.1	0.0	0.0
lung-cancer	1.2	0.8	0.8	1.0	0.3	0.8
lymphography	1.1	0.8	0.8	0.8	0.6	0.5
monks-1	0.1	0.5	0.5	0.2	0.2	0.1
monks-2	0.1	0.1	0.1	0.1	0.1	0.1
monks-3	0.1	0.0	0.0	0.0	0.0	0.0
mplex-11	0.1	0.3	0.3	0.0	0.0	0.0
mplex-6	0.2	0.3	0.3	0.0	0.0	0.0
mushroom	1.0	0.1	0.1	0.0	0.0	0.1
nettalk	0.7	0.3	0.3	0.3	0.7	0.3
pima	2.6	1.8	1.8	2.2	1.5	2.3
post-op	1.7	1.2	1.2	0.3	0.3	0.5
primary-tumor	0.5	0.3	0.3	0.3	0.4	0.1
promoter	0.6	0.7	0.7	0.2	0.4	0.2
road	1.1	1.5	1.5	2.0	0.8	1.3
soybean	0.9	0.2	0.2	0.6	0.4	0.1
splice	0.4	0.1	0.1	0.1	0.6	0.2
switzerland	1.8	2.4	2.4	1.2	0.9	0.7
tictactoe	0.1	0.1	0.1	0.1	0.1	0.1
usama-mys	1.2	1.0	1.0	2.3	0.6	1.8
va	2.7	2.8	2.8	1.0	2.0	0.4
votes	0.4	0.2	0.2	0.4	0.6	0.3
vowel	0.7	0.6	0.6	0.8	0.8	0.4
waveform	0.6	1.1	1.1	1.4	0.5	0.6
wine	0.5	0.3	0.3	0.2	0.1	0.3
zoo	0.1	0.0	0.0	0.1	0.1	0.0

Table 18. CPU (no pruning)

See Table 19 for associated standard deviations

Task	DL	DM	DE	IE	I1	I2	C1	C2	Mean
audio-no-id	1338.5	1084.5	552.9	20.9	1.7	1.6	0.3	0.2	375.1
balance-scale	2.9	2.8	2.5	5.3	0.1	0.1	0.2	0.2	1.8
bc-wisc	14.0	13.3	11.9	7.7	0.3	0.2	0.1	0.1	6.0
breast-cancer	36.2	40.8	23.3	22.7	0.2	0.2	0.1	0.0	15.5
bupa	124.9	92.3	74.4	152.8	0.5	0.5	0.3	0.2	55.8
chess-551x39	1619.4	1463.4	777.1	61.5	1.4	1.3	0.3	0.2	490.6
cleveland	94.4	88.4	70.4	73.0	0.6	0.6	0.4	0.3	41.0
crx	315.1	244.1	166.5	199.4	1.2	1.1	0.4	0.4	116.0
fayyad	0.3	0.2	0.2	0.1	0.0	0.0	0.0	0.0	0.1
glass-no-id	37.9	30.7	23.1	20.9	0.4	0.3	0.3	0.3	14.2
hepatitis	25.9	24.1	19.2	5.2	0.2	0.1	0.1	0.1	9.4
horse-dead	57.2	48.0	38.2	9.1	0.4	0.3	0.2	0.2	19.2
horse-sick	6.1	4.8	5.0	0.7	0.1	0.1	0.1	0.1	2.1
hungarian	35.8	34.0	26.5	32.3	0.3	0.3	0.2	0.2	16.2
hypothyroid	433.4	374.0	303.7	24.1	4.1	3.6	1.3	1.1	143.2
ionosphere	976.9	1033.3	657.0	18.6	3.1	3.1	1.4	1.4	336.8
iris	0.2	0.2	0.2	0.1	0.0	0.0	0.0	0.0	0.1
landsat	83.0	79.2	53.5	89.2	0.8	0.8	0.7	0.6	38.5
led24	62.2	61.6	60.9	4.5	0.3	0.3	0.1	0.1	23.7
led7	3.0	2.9	2.8	0.6	0.1	0.1	0.0	0.0	1.2
lenses	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
lung-cancer	21.3	20.0	19.5	1.2	0.1	0.1	0.0	0.0	7.8
lymphography	17.3	17.7	15.2	4.5	0.1	0.1	0.0	0.0	6.9
monks-1	1.4	1.4	1.4	0.1	0.1	0.1	0.0	0.0	0.6
monks-2	1.7	1.7	1.5	2.5	0.1	0.1	0.1	0.0	1.0
monks-3	0.4	0.4	0.4	0.0	0.0	0.0	0.0	0.0	0.2
mplex-11	23.9	23.4	22.9	6.7	0.8	0.8	0.2	0.2	9.9
mplex-6	0.2	0.2	0.2	0.1	0.0	0.0	0.0	0.0	0.1
mushroom	888.4	793.0	690.9	1.9	5.7	5.7	0.4	0.4	298.3
nettalk	3925.3	3885.5	2867.9	1083.0	12.0	11.7	5.0	3.0	1474.2
pima	395.9	378.2	192.7	1121.3	1.6	1.6	0.9	0.8	261.6
post-op	2.9	2.7	2.3	0.9	0.0	0.0	0.0	0.0	1.1
primary-tumor	127.2	122.2	96.5	25.0	0.7	0.5	0.3	0.2	46.6
promoter	113.3	113.0	93.0	7.6	0.2	0.2	0.0	0.0	40.9
road	969.4	941.9	642.6	1709.5	4.4	4.2	3.8	3.5	534.9
soybean	504.7	453.6	337.2	14.8	1.6	1.6	0.4	0.4	164.3
splice	7747.0	7746.4	7605.4	469.2	12.3	11.5	1.6	1.5	2949.4
switzerland	35.3	35.6	18.2	14.0	0.2	0.2	0.2	0.1	13.0
tictactoe	488.8	488.3	478.1	496.7	3.1	2.7	1.2	0.8	245.0
usama-mys	121.9	99.0	70.2	13.6	0.7	0.7	0.3	0.3	38.3
va	109.5	103.3	53.4	62.0	0.6	0.5	0.3	0.2	41.2
votes	14.8	13.8	11.9	1.5	0.2	0.1	0.1	0.1	5.3
vowel	734.2	675.4	512.3	831.4	4.9	4.8	3.4	3.3	346.2
waveform	2127.6	1836.7	1522.9	382.2	4.2	4.0	1.9	1.8	735.2
wine	7.5	7.3	6.9	1.0	0.1	0.1	0.1	0.1	2.9
zoo	2.2	2.1	2.3	0.1	0.0	0.0	0.0	0.0	0.8
Mean	514.1	488.8	394.2	152.2	1.5	1.4	0.6	0.5	194.2

Table 19. Standard Deviation for CPU (no pruning)

See Table 18 for associated point estimates

Task	DL	DM	DE	IE	I1	I2	C1	C2
audio-no-id	321.1	278.8	81.9	5.4	0.1	0.1	0.0	0.0
balance-scale	0.3	0.3	0.1	0.6	0.0	0.0	0.0	0.0
bc-wisc	1.6	1.8	0.7	1.7	0.0	0.0	0.0	0.0
breast-cancer	10.8	14.7	2.8	2.9	0.0	0.0	0.0	0.0
bupa	28.7	21.0	17.5	35.2	0.1	0.1	0.0	0.0
chess-551x39	200.7	262.2	128.5	7.2	0.1	0.1	0.1	0.1
cleveland	16.4	13.7	4.2	12.8	0.1	0.1	0.0	0.0
crx	54.3	60.7	9.2	32.8	0.1	0.1	0.0	0.0
fayyad	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
glass-no-id	9.2	7.6	2.3	3.7	0.0	0.0	0.0	0.0
hepatitis	2.2	2.5	1.5	1.3	0.0	0.0	0.0	0.0
horse-dead	19.8	12.7	7.3	2.0	0.0	0.0	0.0	0.0
horse-sick	0.8	0.5	0.5	0.2	0.0	0.0	0.0	0.0
hungarian	3.0	3.9	1.0	6.0	0.0	0.0	0.0	0.0
hypothyroid	96.8	65.0	13.8	5.6	0.3	0.4	0.2	0.1
ionosphere	192.7	276.2	88.2	5.2	0.1	0.1	0.1	0.1
iris	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
landsat	8.5	7.5	4.4	16.6	0.0	0.0	0.0	0.0
led24	2.3	2.5	3.2	1.1	0.0	0.0	0.0	0.0
led7	0.2	0.2	0.2	0.1	0.0	0.0	0.0	0.0
lenses	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
lung-cancer	2.4	1.9	1.8	0.6	0.0	0.0	0.0	0.0
lymphography	2.1	2.2	1.5	1.1	0.0	0.0	0.0	0.0
monks-1	0.2	0.2	0.1	0.0	0.0	0.0	0.0	0.0
monks-2	0.1	0.1	0.1	0.7	0.0	0.0	0.0	0.0
monks-3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
mplex-11	1.2	1.2	1.2	3.3	0.0	0.0	0.0	0.0
mplex-6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
mushroom	12.4	37.5	22.1	0.8	0.2	0.2	0.0	0.0
nettalk	188.2	175.1	183.5	77.1	0.1	0.1	0.0	0.0
pima	146.0	173.8	16.5	128.9	0.3	0.3	0.0	0.0
post-op	0.4	0.4	0.1	0.1	0.0	0.0	0.0	0.0
primary-tumor	15.7	12.3	5.9	4.9	0.0	0.0	0.0	0.0
promoter	21.9	23.0	9.5	2.2	0.0	0.0	0.0	0.0
road	135.9	70.5	37.8	313.3	0.3	0.3	0.1	0.1
soybean	49.8	43.6	16.0	1.8	0.0	0.0	0.0	0.0
splice	352.8	613.0	498.0	108.6	0.2	0.3	0.0	0.0
switzerland	12.1	11.7	2.1	3.3	0.0	0.0	0.0	0.0
tictactoe	12.2	12.3	8.2	63.0	0.0	0.0	0.0	0.0
usama-mys	28.3	23.7	13.6	3.0	0.1	0.1	0.0	0.0
va	43.1	41.6	3.9	8.8	0.1	0.1	0.0	0.0
votes	2.0	1.4	1.7	0.5	0.0	0.0	0.0	0.0
vowel	114.2	109.7	67.6	108.5	0.2	0.2	0.1	0.1
waveform	480.8	396.5	358.1	87.9	0.9	0.8	0.1	0.1
wine	0.6	0.7	0.9	0.4	0.0	0.0	0.0	0.0
zoo	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0

Table 20. CPU (pruning)

See Table 21 for associated standard deviations

Task	DL	DM	DE	IE	I1	I2	C1	C2	Mean
audio-no-id	1178.5	1159.5	575.6	22.5	1.7	1.6	0.3	0.2	367.5
balance-scale	3.1	2.9	2.6	8.3	0.1	0.1	0.2	0.2	2.2
bc-wisc	15.7	13.0	15.7	12.0	0.3	0.3	0.1	0.1	7.1
breast-cancer	50.4	29.9	45.3	25.0	0.2	0.2	0.1	0.0	18.9
bupa	166.1	94.0	133.0	160.3	0.6	0.5	0.3	0.2	69.4
chess-551x39	1154.4	919.8	847.4	89.8	1.4	1.3	0.3	0.2	376.8
cleveland	116.6	90.5	81.6	87.8	0.6	0.6	0.4	0.3	47.3
crx	505.4	228.8	371.7	298.2	1.2	1.2	0.4	0.4	175.9
fayyad	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.1
glass-no-id	45.3	30.8	27.9	22.0	0.4	0.4	0.3	0.3	15.9
hepatitis	33.9	25.9	30.0	5.5	0.2	0.1	0.1	0.1	12.0
horse-dead	99.4	45.5	68.6	13.0	0.4	0.3	0.2	0.2	28.4
horse-sick	7.0	4.9	5.9	1.4	0.1	0.1	0.1	0.1	2.4
hungarian	50.1	33.3	46.1	39.3	0.3	0.3	0.2	0.2	21.2
hypothyroid	404.9	365.1	387.3	29.3	4.1	3.6	1.3	1.1	149.6
ionosphere	2281.0	1366.9	893.6	20.5	3.3	3.2	1.4	1.4	571.4
iris	0.3	0.3	0.3	0.1	0.0	0.0	0.0	0.0	0.1
landsat	88.2	73.5	70.7	116.8	0.9	0.8	0.7	0.6	44.0
led24	66.6	61.1	63.7	4.9	0.3	0.3	0.1	0.1	24.6
led7	2.9	2.9	2.8	0.7	0.1	0.1	0.0	0.0	1.2
lenses	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
lung-cancer	45.2	21.9	37.9	1.2	0.1	0.1	0.0	0.0	13.3
lymphography	23.1	19.2	18.4	5.6	0.1	0.1	0.0	0.0	8.3
monks-1	1.4	1.4	1.4	0.2	0.1	0.1	0.0	0.0	0.6
monks-2	1.9	1.8	1.7	4.7	0.1	0.1	0.1	0.0	1.3
monks-3	0.4	0.4	0.4	0.0	0.0	0.0	0.0	0.0	0.2
mplex-11	24.0	23.5	23.0	12.2	0.8	0.8	0.2	0.2	10.6
mplex-6	0.3	0.2	0.3	0.2	0.0	0.0	0.0	0.0	0.1
mushroom	888.2	792.1	690.9	5.5	5.7	5.7	0.4	0.4	298.6
nettalk	3853.6	3815.9	2870.9	1380.1	12.1	11.7	5.0	3.0	1494.0
pima	897.5	390.9	682.3	1422.6	1.6	1.6	0.9	0.8	424.8
post-op	3.4	2.8	3.3	0.8	0.0	0.0	0.0	0.0	1.3
primary-tumor	125.3	122.8	100.9	26.0	0.7	0.5	0.3	0.2	47.1
promoter	134.7	100.9	125.7	9.7	0.2	0.2	0.0	0.0	46.4
road	1178.5	793.4	840.0	1355.8	4.5	4.3	3.8	3.5	523.0
soybean	430.7	454.8	365.3	16.4	1.6	1.6	0.4	0.4	158.9
splice	8281.7	7744.9	8184.6	466.4	12.4	11.6	1.6	1.5	3088.1
switzerland	27.9	29.8	20.7	14.0	0.2	0.2	0.2	0.1	11.6
tictactoe	481.1	485.5	487.6	724.7	3.1	2.7	1.2	0.8	273.3
usama-mys	193.0	86.4	129.8	14.0	0.8	0.7	0.3	0.3	53.2
va	96.1	99.9	59.4	64.0	0.6	0.5	0.3	0.2	40.1
votes	16.3	13.8	13.7	1.4	0.2	0.1	0.1	0.1	5.7
vowel	929.4	741.1	541.2	1019.9	5.0	5.0	3.4	3.3	406.0
waveform	3272.0	1747.6	2126.7	395.5	4.3	4.2	1.9	1.8	944.2
wine	8.1	7.0	7.3	1.5	0.1	0.1	0.1	0.1	3.0
zoo	2.2	2.1	2.3	0.1	0.0	0.0	0.0	0.0	0.9
Mean	591.0	479.3	456.6	171.7	1.5	1.5	0.6	0.5	212.8

Table 21. Standard Deviation for CPU (pruning)

See Table 20 for associated point estimates

Task	DL	DM	DE	IE	I1	I2	C1	C2
audio-no-id	240.4	201.5	89.8	4.6	0.1	0.1	0.0	0.0
balance-scale	0.3	0.3	0.1	1.0	0.0	0.0	0.0	0.0
bc-wisc	2.2	0.7	2.4	1.8	0.0	0.0	0.0	0.0
breast-cancer	7.0	6.7	6.0	2.7	0.0	0.0	0.0	0.0
bupa	41.2	24.1	27.3	27.8	0.1	0.1	0.0	0.0
chess-551x39	141.9	107.0	143.6	18.5	0.1	0.1	0.1	0.1
cleveland	25.9	12.3	6.2	9.0	0.1	0.1	0.0	0.0
crx	175.4	56.2	215.9	49.5	0.1	0.1	0.0	0.0
fayyad	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
glass-no-id	16.5	8.0	4.4	3.6	0.0	0.0	0.0	0.0
hepatitis	6.3	3.8	5.7	1.3	0.0	0.0	0.0	0.0
horse-dead	45.3	7.6	21.6	1.3	0.0	0.0	0.0	0.0
horse-sick	2.7	0.5	1.5	0.4	0.0	0.0	0.0	0.0
hungarian	9.4	5.1	7.9	7.5	0.0	0.0	0.0	0.0
hypothyroid	31.5	37.7	33.8	9.5	0.4	0.4	0.2	0.1
ionosphere	223.3	276.5	152.8	3.5	0.1	0.1	0.1	0.1
iris	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
landsat	8.2	6.9	4.8	17.1	0.0	0.0	0.0	0.0
led24	2.0	2.5	2.1	0.8	0.0	0.0	0.0	0.0
led7	0.2	0.2	0.2	0.1	0.0	0.0	0.0	0.0
lenses	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
lung-cancer	19.0	2.3	18.1	0.4	0.0	0.0	0.0	0.0
lymphography	4.6	2.8	1.9	0.6	0.0	0.0	0.0	0.0
monks-1	0.2	0.2	0.1	0.1	0.0	0.0	0.0	0.0
monks-2	0.1	0.1	0.2	0.7	0.0	0.0	0.0	0.0
monks-3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
mplex-11	1.2	1.2	1.2	3.7	0.0	0.0	0.0	0.0
mplex-6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
mushroom	12.4	37.7	21.9	0.9	0.2	0.2	0.0	0.0
nettalk	166.6	186.0	232.3	66.7	0.1	0.1	0.0	0.0
pima	329.0	130.0	314.2	143.6	0.3	0.3	0.0	0.0
post-op	0.4	0.3	0.4	0.1	0.0	0.0	0.0	0.0
primary-tumor	9.3	10.9	7.0	2.5	0.0	0.0	0.0	0.0
promoter	10.7	12.8	10.0	1.9	0.0	0.0	0.0	0.0
road	211.5	73.6	110.9	190.3	0.3	0.3	0.1	0.1
soybean	69.6	53.5	13.4	1.8	0.0	0.0	0.0	0.0
splice	313.1	675.5	386.5	54.8	0.2	0.3	0.0	0.0
switzerland	7.9	6.9	1.9	2.2	0.0	0.0	0.0	0.0
tictactoe	9.7	8.9	9.0	77.1	0.0	0.0	0.0	0.0
usama-mys	39.9	13.8	31.0	2.8	0.1	0.1	0.0	0.0
va	18.5	43.6	5.3	8.3	0.1	0.1	0.0	0.0
votes	1.8	1.8	2.0	0.4	0.0	0.0	0.0	0.0
vowel	159.0	133.4	67.2	140.3	0.2	0.2	0.1	0.1
waveform	722.0	248.6	581.4	57.4	0.9	0.9	0.1	0.1
wine	0.9	0.6	1.2	0.4	0.0	0.0	0.0	0.0
zoo	0.1	0.2	0.2	0.0	0.0	0.0	0.0	0.0

References

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

Cockett, J. R. B., & Herrera, J. A. (1990). Decision tree reduction. *Journal of the ACM, 37*, 815-842.

Crawford, S. L. (1989). Extensions to the CART algorithm. *International Journal of Man-Machine Studies, 31*, 197-217.

Fayyad, U. M. (1991). *On the induction of decision trees for multiple concept learning*. Doctoral dissertation, Computer Science and Engineering, University of Michigan.

Fayyad, U. M., & Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning, 8*, 87-102.

Fisher, D. (1996). Iterative optimization and simplification of hierarchical clusterings. *Journal of Artificial Intelligence Research, 4*, 147-178.

Kalles, D., & Morris, T. (1996). Efficient incremental induction of decision trees. *Machine Learning, 24*, 231-242.

Kohavi, R. (1995). The power of decision tables. *Proceedings of the European Conference on Machine Learning*.

Lovell, B. C., & Bradley, A. P. (1996). The multiscale classifer. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 18*, 124-137.

Mooney, R., Shavlik, J., Towell, G., & Gove, A. (1989). An experimental comparison of symbolic and connectionist learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 775-780). Detroit, Michigan: Morgan Kaufmann.

Murphy, P. M., & Aha, D. W. (1994). *UCI repository of machine learning databases*, Irvine, CA: University of California, Department of Information and Computer Science.

Pazzani, M., Merz, C., Murphy, P., Ali, K., Hume, T., & Brunk, C. (1994). Reducing misclassification costs. *Machine Learning: Proceedings of the Eleventh International Conference* (pp. 217-225). New Brunswick, NJ: Morgan Kaufmann.

Quinlan, J. R., & Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation, 80*, 227-248.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica, 14*, 465-471.

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

Steel, R. G. D., & Torrie, J. H. (1980). *Principles and procedures of statistics, 2nd edition*. New York, NY: McGraw-Hill.

Tan, M., & Schlimmer, J. C. (1990). Two case studies in cost-sensitive concept acquisition. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 854-860). Boston, MA: Morgan Kaufmann.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning, 4*, 161-186.

Utgoff, P. E. (1994). An improved algorithm for incremental induction of decision trees. *Machine Learning: Proceedings of the Eleventh International Conference* (pp. 318-325). New Brunswick, NJ: Morgan Kaufmann.

Utgoff, P. E., & Clouse, J. A. (1996). *A Kolmogorov-Smirnoff metric for decision tree induction*, (Technical Report 96-3), Amherst, MA: University of Massachusetts, Department of Computer Science.

Van de Velde, W. (1990). Incremental induction of topologically minimal trees. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 66-74). Austin, TX: Morgan Kaufmann.

Walpole, R. E. (1974). *Introduction to statistics*. New York: Macmillan.

White, A. P., & Liu, W. Z. (1994). Bias in information-based measures in decision tree induction. *Machine Learning, 15*, 321-329.

Wirth, N. (1976). *Algorithms + data structures = programs*. Englewood Cliffs, NJ: Prentice-Hall.