The Time Complexity of Decision Tree Induction

J. Kent Martin and D. S. Hirschberg
(jmartin@ics.uci.edu) (dan@ics.uci.edu)
Department of Information and Computer Science
University of California, Irvine, CA, 92717

Technical Report 95-27
August 25, 1995

Abstract

Various factors affecting decision tree learning time are explored. The factors which consistently affect accuracy are those which directly or indirectly (as in the handling of continuous attributes) allow a greater number and variety of potential trees to be explored. Other factors, such as pruning and choice of heuristics, generally have little effect on accuracy, but significantly affect learning time. We prove that the time complexity of induction and post-processing is exponential in tree height in the worst case and, under fairly general conditions, in the average case. This puts a premium on designs which tend to produce shallower trees (*e.g.,* multi-way rather than binary splits, and heuristics which prefer more balanced splits). Simple pruning is linear in tree height, contrasted to the exponential growth of more complex operations. The key factor influencing whether simple pruning will suffice is that the split selection and pruning heuristics should be the same and unbiased. The information gain and χ^2 heuristics are biased towards unbalanced splits, and neither is an admissible test for pruning. Empirical results show that the hypergeometric function can be used for both split selection and pruning, and that the resulting trees are simpler, more quickly learned, and no less accurate than the trees resulting from other heuristics and more complex post-processing.

1 Introduction

This paper studies the complexity of Top-Down Induction of Decision Trees — the TDIDT [25] family of algorithms typified by Quinlan's ID3 [25] and C4.5 [26]. The input for these algorithms is a set of data items, each described by a class label and its values for a set of attributes, and a set of candidate partitions of the data. These algorithms make a greedy, heuristic choice of one of the set of candidate partitions, and then recursively split each subset of that partition until a subset consists only of one class or until the set of candidate partitions is exhausted.

Early algorithms included other stopping criteria, stopping when the improvement achieved by the best candidate partition was judged to be insignificant. Later algorithms dropped these stopping criteria and added post-processing procedures to prune a tree or to replace subtrees (typically replacing a decision node by its largest child and merging instances from the other children).

The number of possible partitions is intractable, even for very small data sets (*e.g.,* $> 100,000$ ways to partition a set of only 10 items). Thus, the set of candidate partitions is necessarily limited, typically to exploring only splits on the value of a single attribute. There has recently been considerable interest in various ways of expanding the set of candidates so as to permit a bounded look-ahead and to explore splits on functions of several attributes [9, 11, 22, 30].

If each leaf of a decision tree is labeled with a predicted class (*e.g.,* the largest class in the leaf), then the accuracy of a tree is the fraction of instances for which the class is correctly predicted. To obtain an unbiased estimate of predictive accuracy, different sets of instances are used to build a decision tree (the training set) and to assess its accuracy (the test set). For smaller data sets (less than several thousand instances), cross-validation or bootstrapping techniques are used (the accuracy of many trees is averaged by iteratively splitting the data into a training set and a test set, then building and evaluating a tree — see [18] for a review of these techniques).

Usually, the dominant goal of decision tree learning is to maximize predictive accuracy, with a secondary goal of minimizing complexity among competing trees of equivalent accuracy. Efforts to devise a combined measure of accuracy and complexity, such as Minimum Description Length [21, 27], encounter many practical difficulties (see [16] for a review of these techniques and issues). Complexity has many facets. Here, we distinguish three measures: (1) number of leaves, (2) average depth, weighted by the fraction of instances reaching each leaf, and (3) run time for training, post-processing, and cross-validation.

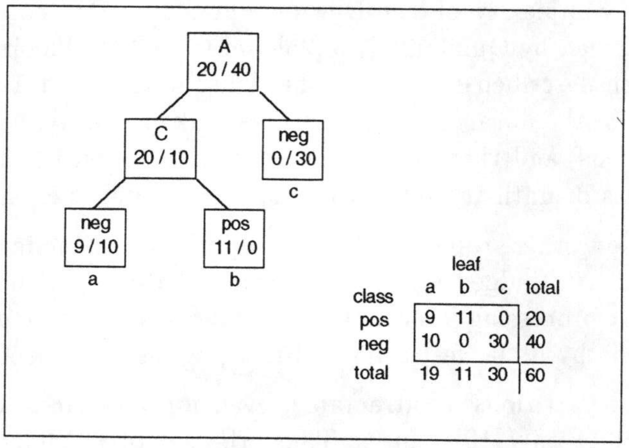
In Figure 1 we illustrate two alternative representations of a partitioning, a decision tree and a contingency table (a frequency matrix, where the rows represent the classes and the columns represent the leaves of a tree or the subsets of a partition). The tree better represents the hierarchy of the splits, whereas the contingency table is more compact and better depicts the frequencies. The following notation is used here for the frequency data:

C	is the number of categories
V	is the number of subsets in the split
m_v	is the no. of instances in subset v
f_{cv}	is the no. of those which are in class c
N	is the total no. in the sample
n_c	is the total no. in class c

	sub-1	$\cdots$	sub-V	Total
cat-1	f_{11}	$\cdots$	f_{1V}	n_1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
cat-C	f_{C1}	$\cdots$	f_{CV}	n_C
Total	m_1	$\cdots$	m_V	N

We analyze TDIDT algorithms, not only in the usual terms of best and worst case data, but also in terms of the choices available in designing an algorithm and, particularly, in those elements of the

Figure 1: Partitioning Representations



choices which generalize over many input sets. At the highest level, there are three such choices: (1) how the set of candidate partitions is chosen (including handling continuous variables, look-ahead, *etc.*, (2) what heuristic function is used in choosing among candidate partitions, and (3) whether to stop splitting based on significance or to post-process.

2 Analysis of TDIDT Tree-Building

Figure 2 summarizes the TDIDT tree-building phase, where the set of candidate partitions has been defined off-line and is summarized by the input parameters **A** (a vector of partition labels or attribute names) and **V** (a matrix of values for splitting on each attribute).

Some algorithms re-evaluate the **V** matrix on every call to this procedure — this generally involves re-sorting the data, adding $O(A \times Nlg(N))$ time to each recursive call, where A is the size of the candidate set and N is the size of the data set. When such an on-line re-evaluation is used, it may dominate the algorithm. Only off-line definition of the candidate set is studied here.

Other major differences between various algorithms lie in the Heuristic function (line 11), in the way numeric attributes are handled by the Partition function (lines 10 and 19), and in whether stopping based on a Significance function (lines 16 and 17) is performed.

Typically, the run times of the Heuristic and Partition functions, T_{heur} and T_{part} respectively, are linear (*i.e.*, $T_{heur} = \theta(N)$ and $T_{part} = \theta(N)$), effectively compiling the contingency matrix and computing some function such as information gain. Then, the run time T_B of the BuildTree function is

$$T_B(\mathbf{A},\ \mathbf{V},\ \mathbf{data}) = \theta(AN) + \sum_{v=1}^{V_b} T_B(\mathbf{A} - \{b\},\ \mathbf{V} - \mathbf{V}_b,\ \mathbf{subdata}_v)$$

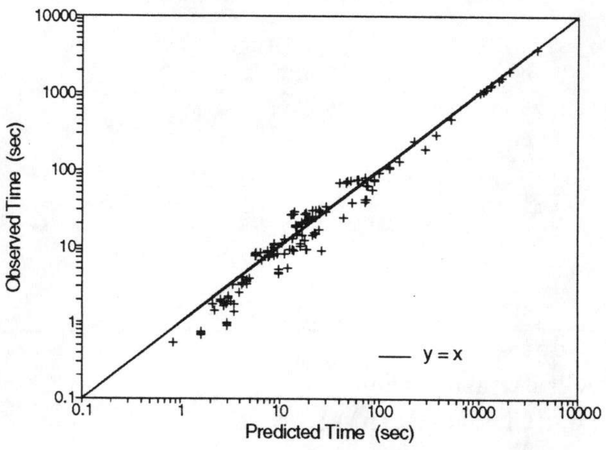
which leads to $T_B = O(A^2N)$ for a complete tree of height A. (Here, b is the 'best' split, $\mathbf{V}_b$ its splitting criteria, $V_b = |\mathbf{V}_b|$ its arity, and $\mathbf{subdata}_v$ its v^{th} subset.)

Some algorithms build only binary trees, and most allow only binary splits on real-valued attributes. In these cases, the effect is the same as that of increasing the number of candidates. Converting the

Figure 2: TDIDT Tree-Building

```
BuildTree($\mathbf{A}$, $\mathbf{V}$, $\mathbf{data}$)                                              (1)
{   if (AllSameClass($\mathbf{data}$)) then                                          (2)
        MakeLeaf(ClassOf($\mathbf{data}$)) ;                                         (3)
    else                                                                 (4)
    {   $A \leftarrow |\mathbf{A}|$ ;                                                    (5)
        $N \leftarrow |\mathbf{data}|$ ;                                                 (6)
        if (($N > 0$) and ($A > 0$)) then                                      (7)
        {   Initialize($b$, best) ;                                          (8)
            for $a = 1 \ldots A$                                                  (9)
            {   $\mathbf{subdata} \leftarrow$ Partition($\mathbf{data}$, $a$, $\mathbf{V}_a$) ;                  (10)
                $\mathcal{E} \leftarrow$ Heuristic($\mathbf{subdata}$) ;                              (11)
                if ($\mathcal{E} <$ best) then                                       (12)
                {   best $\leftarrow \mathcal{E}$ ;                                        (13)
                    $b \leftarrow a$ ;                                               (14)
            }   }                                                        (15)
            if (not Significance(best)) then                             (16)
                MakeLeaf(LargestClass($\mathbf{data}$)) ;                            (17)
            else                                                         (18)
            {   $\mathbf{subdata} \leftarrow$ Partition($\mathbf{data}$, $b$, $\mathbf{V}_b$) ;                  (19)
                $V \leftarrow |\mathbf{V}_b|$ ;                                              (20)
                for $v = 1 \ldots V$                                              (21)
                {   if ($\mathbf{A} - \{b\} = \emptyset$) then                                  (22)
                        MakeLeaf(LargestClass($\mathbf{subdata}_v$)) ;               (23)
                    else                                                 (24)
                        BuildTree($\mathbf{A} - \{b\}$, $\mathbf{V} - \mathbf{V}_b$, $\mathbf{subdata}_v$) ;             (25)
}   }   }   }   }                                                    (26)
```

Figure 3: Empirical Run-Time Data



data so that every attribute becomes binary increases the number of attributes and, if we simply create a logically equivalent set of $V_i - 1$ binary attributes for each candidate (where V_i is the arity of the i^{th} candidate), the time is $O(d^2N)$, where d is the dimensionality of the data, $d = \sum(V_i - 1)$. For real-valued attributes, the number of potential split-values V_i is $O(N)$ for each attribute and $d = O(AN)$, so we can see that the method of handling real-valued attributes could cause the behavior of TDIDT algorithms to vary from $O(A^2N)$ to $O(A^2N^3)$. Thus, as the set of candidates is expanded to allow look-ahead and functions of several attributes, it will become increasingly important to control the dimensionality of the candidates and, especially, to deal with real-valued attributes more efficiently.

Analysis of real-world data sets is more complicated because the node-splitting process for any given path from the root may terminate before all attributes have been utilized, either because all instances reaching the node have the same class (no further information will be gained by continuing to split) or because of some stopping criterion when no further *significant* information will be gained by continuing to split. Thus, the height of the tree may be less than the number of candidates A, and the leaves may lie at different depths. In these circumstances, the time complexity of the algorithm is related to the average height (h) of the tree (weighted by the number of instances reaching each leaf).

Looking at Figure 2 in detail, and assuming that all candidates have the same arity V, the time complexity can be modeled as

$$\begin{aligned} T_B(A,V,N) &= K_0 + A\left[K_2 + (\mathcal{E}_0 + \mathcal{E}_1 N + \mathcal{E}_2 C + \mathcal{E}_3 V + \mathcal{E}_4 CV)\right] \\ &+ K_1 N + K_3 V + \sum_{v=1}^{V} T_B(A-1, V, m_v) \end{aligned}$$

Where C is the number of classes, K_0, K_2, and K_3 are small overhead constants, K_1 is the incremental cost of the Partition function, and the $\mathcal{E}_i$ are the coefficients of the Heuristic function ($\mathcal{E}_0$ is a small overhead term, and $\mathcal{E}_1 N$ typically dominates the other terms). Assuming that all leaves lie at the same depth ($A-1$), this leads to the following

$$\begin{aligned} T_B(A,V,N) &\approx (K_0 + K_3 V)\, f_1(A,V) + (K_2 + \mathcal{E}_0 + \mathcal{E}_3 V + \mathcal{E}_4 CV)\, f_2(A,V) \\ &+ (\mathcal{E}_1 N + \mathcal{E}_2 C)\,\frac{A(A+1)}{2} + K_1 AN \end{aligned}$$

where $$f_1(A,V) = \frac{V^A - 1}{V-1} \quad \text{and} \quad f_2(A,V) = \left(\frac{V^{A+1}-1}{V-1} - 1 - A\right) \Big/ (V-1)$$

Empirical data from 16 different populations representing a wide range of sample sizes, number of attributes, arity of attributes, and a mixture of discrete and continuous attributes are shown in Figure 3. The data include two different protocols for converting continuous attributes to discrete form, 9 different heuristic functions, both binary and multi-way splits, and stopping the splitting process based on a statistical criterion [17]. (In these studies, N ranged from 45 to 869, with an average data set size of 250; A ranged from 4 to 100, with an average of 15 attributes; and V ranged from 2 to 9). For $A > 1$, these data are well-fit by the following pro-rated model:

$$\begin{aligned} T_B \approx & \left((\mathcal{E}_1 N + \mathcal{E}_2 C) \frac{A(A+1)}{2} + K_1 A N + K_0 \frac{\mathcal{V}^h - 1}{\mathcal{V} - 1} \right) \frac{h}{A-1} \\ & + (\mathcal{E}_1 N + \mathcal{E}_2 C) A \; + \; K_0 \; + \; K_1 N \end{aligned}$$

where $\mathcal{V} = d/A$ is the average branching factor. The approximate values of the parameters are[1]:

$$\begin{aligned} \mathcal{E}_1 &\approx 1.34 \pm 0.12 \text{ msec} & \mathcal{E}_2 &\approx 47.6 \pm 0.8 \text{ msec} \\ K_0 &\approx (3.10 \pm 0.18) \times 10^{-3} \text{ msec} & K_1 &\approx 3.7 \pm 1.6 \text{ msec} \end{aligned}$$

For trees of modest height and relatively large samples, the cumulative overhead (K_0) term is insignificant — typically, that is, $h \approx 0.3A$ and $N \approx 25A$, and $K_0\ (\mathcal{V}^h - 1)/(\mathcal{V}-1) \ll \mathcal{E}_1 N A^2$, so that $T_B = \theta(A^2 N)$ in most cases. (This term arises because every node created involves a small amount of work — $f_1(\mathcal{V}, h)$ is related to the number of nodes.)

Applications do exist, however, where the exponential growth of this term cannot be ignored. For the longest run-time in Figure 3 (about 1.5 hrs), $N = 869$, $A = 100$, and a binary tree with an average depth of $h = 30$ was built using one of the 9 heuristics — the factor of 2^{30} for the K_0 term is significant here. In the worst case $h = A - 1$ and $T_B = O(\mathcal{V}^A)$ for applications where there are many attributes (large A).

3 Analysis of Post-Processing

Figure 4 summarizes a typical post-processing routine, Quinlan's C4.5 [26] pessimistic pruning method, for binary trees. Examination of Figure 4 reveals that the dominant factors are the height of the input tree, H, the data set size, N, the height and weight balance of the tree, and whether a decision node is replaced by a child rather than simply pruned or left unmodified.

We denote the time complexity of the post-processing and evaluation functions as $T_P(H,N)$ and $T_E(H,N)$, respectively. In all cases, $0 \leq H < N$ and the recursion terminates with

$$T_P(0,N) \; = \; T_E(0,N) \; = \; \theta(N)$$

When the tree surgery operation is not actually performed, but merely evaluated,

$$T_P(H,N) \; = \; T_E(H,N)$$

[1]These values are, of course, specific to the particular implementation (Common Lisp on a Sparc 4), but do indicate the relative costs.

Figure 4: Pessimistic Pruning

```
PostProc(dtree, data)
{   AsIs, Pruned, Surgery, and Q
        are defined as in Eval(dtree, data) ;
    if (AsIs $\leq$ min(Pruned, Surgery) then
        return AsIs ;
    else
    {   if (Pruned $\leq$ Surgery) then
        {   dtree $\leftarrow$ MakeLeaf(LargestClass(data)) ;
            return Pruned ;
        }
        else   /* surgery performed here */
        {   dtree $\leftarrow$ Q ;
            Surgery $\leftarrow$ PostProc(Q, data) ;
            return Surgery ;
}   }   }

Eval(dtree, data)
{   $N \leftarrow |$data$|$ ;
    Q $\leftarrow$ nil ;
    if dtree is a leaf then
    {   P $\leftarrow$ PredictedClass(dtree) ;
        $E \leftarrow (N - |$P$|)$ ;
        Pruned = Surgery = AsIs $\leftarrow f(E, N)$ ;
        return AsIs ;
    }
    else
    {   F $\leftarrow$ SplitInfo(Root(dtree)) ;
        Ldata, Rdata $\leftarrow$ Partition(data, F) ;
        $L \leftarrow |$Ldata$| / N$ ;
        $R \leftarrow |$Rdata$| / N$ ;
        AsIs $\leftarrow$   $L \times$ Eval(Ltree, Ldata)  +
                   $R \times$ Eval(Rtree, Rdata) ;
        P $\leftarrow$ LargestClass(data) ;
        $E \leftarrow (N - |$P$|)$ ;
        Pruned $\leftarrow f(E, N)$ ;
        if $(L > R)$ then Q $\leftarrow$ Ltree ;
        else   Q $\leftarrow$ Rtree ;
        Surgery $\leftarrow$ Eval(Q, data) ;
        return min(AsIs, Pruned, Surgery) ;
}   }
```

When the surgery is performed (lines 12 and 13),

$$T_P(H,N) \;=\; T_E(H,N) \;+\; T_P(q,N)$$

where q is the height of the child covering the majority of the instances (the larger child). In the worst case q is $H-1$, and so

$$T_P(H,N) \;\le\; T_E(H,N) \;+\; T_P(H-1,N) \;\le\; \sum_{i=0}^{H} T_E(i,N)$$

If m is the size of the larger child ($\lceil N/2 \rceil \le m \le N-1$), and r the height of the smaller child, then either $q=H-1$ or $r=H-1$, and

$$T_E(H,N) \;=\; \theta(N) \;+\; T_E(q,m) \;+\; T_E(r,N-m) \;+\; T_E(q,N) \tag{1}$$

Consider the case when $q=0$, and thus $r=H-1$. Since $N/2 \le m < N$ and $T_E(0,m)=\theta(m)$,

$$T_E(H,N) \;=\; \theta(N) \;+\; T_E(H-1,N-m) \;\le\; \theta(N) \;+\; T_E(H-1,N/2) \;=\theta(N)$$

The Eval function must consider all data, and so $T_E(H,N)=\Omega(N)$. From the above case ($q=0$), $\theta(N)$ is achievable and so the best case of $T_E(H,N)$ is $\theta(N)$.

Now, consider the case when $q=H-1$, $r=H-1$, and $m=N/2$. Then

$$T_E(H,N) \;=\; \theta(N) \;+\; 2\,T_E(H-1,N/2) \;+\; T_E(H-1,N)$$

In this case, $T_E(H,N)=\Omega(N\,2^H)$, as we now prove by induction on H. This is clearly true for the base case when $H=0$. Assume that $T_E(H,N)\ge cN2^H$ for some constant c and for all $H \le h-1$, and consider the case $H=h$.

$$\begin{aligned} T_E(h,N) \;&\ge\; 2\,T_E(h-1,N/2) \;+\; T_E(h-1,N) \\ &\ge\; 2\,c(N/2)\,2^{h-1} \;+\; c\,N\,2^{h-1} \;=\; cN2^h \end{aligned} \tag{2}$$

The worst case of $T_E(H,N)$ is thus $\Omega(N\,2^H)$.

Assuming that the $\theta(N)$ term is bounded by cN, for some constant c, the worst case of the Eval function $T_E(H,N)$ (Equation 1) is bounded from above by $\tau(H,N)$, defined recursively by

$$\begin{aligned} \tau(H,N) \;&=\; cN \;+\; \tau(H-1,m) \;+\; \tau(H-1,N-m) \;+\; \tau(H-1,N) \\ \tau(0,N) \;&=\; cN \end{aligned}$$

We prove, by induction on H, that $\tau(H,N)=c(2^{H+1}-1)N$. The base case, when $H=0$, is trivially true. Assume that $\tau(H,N)=c(2^{H+1}-1)N$ for all $H \le h-1$ and all N, and consider the case $H=h$.

$$\begin{aligned} \tau(h,N) \;&=\; cN \;+\; \tau(h-1,m) \;+\; \tau(h-1,N-m) \;+\; \tau(h-1,N) \\ &=\; cN \;+\; c(2^h-1)m \;+\; c(2^h-1)(N-m) \;+\; c(2^h-1)N \\ &=\; cN \;+\; c(2^h-1)(2N) \;=\; c(2^{h+1}-1)N \end{aligned}$$

From the valuation of $\tau(H,N)$ and the preceeding example ($q=r=H-1$ and $m=N/2$), we see that the worst case of $T_E(H,N)$ is $\theta(N\,2^H)$.

Recalling that for post-processing the best behavior (when no surgery is actually performed) is $T_P(H,N)=T_E(H,N)$, and the worst is $T_P(H,N)=\sum_{i=0}^{H} T_E(i,N)$, and substituting the best and worst cases of the Eval function T_E we see that, for binary trees,

$$T_P(H,N) = \begin{cases} \Omega(N) \\ O(N\ 2^H) \end{cases}$$

To infer typical behavior, we note that real world data are usually noisy, and real attributes are seldom perfect predictors of class. For these reasons, there is usually some finite impurity rate I for each branch of a split. For n instances from a population with rate I, the likelihood P that the branch will be impure (*i.e.,* will contain instances from more than one class) is given by the binomial distribution, $P = 1-(1-I)^n$, and is an increasing function of the subset size n. For such an impure branch, additional splits on other attributes (*i.e.,* a deeper subtree) will be needed to fully separate the classes. Thus, there is a tendency for larger subsets to have deeper subtrees.

If H_L and H_R are respectively the heights of the left and right subtrees, and n the size of the left subset, then

$$T_E(H,N) \ = \ \theta(N) \ + \ T_E(H_L,n) \ + \ T_E(H_R,N-n) + \begin{cases} T_E(H_L,N) & \text{if } n \geq N/2 \\ T_E(H_R,N) & \text{if } n < N/2 \end{cases}$$

Now, either $H_L = H-1$ or $H_R = H-1$, and the likelihood that $H_L = H-1$ increases as n increases. If we express this increasing likelihood as $p = \text{Prob}(H_L = H-1) = p(x)$, where $x \equiv n/N$, then the approximate expected value, $t(H,N)$, of $T_E(H,N)$ is (see Appendix A for a derivation)

$$t(H,N) \ \geq \ \theta(N) \ + \ p\,(H-1,Nx) \ + \ (1-p)\, t(H-1,N(1-x)) \ + \ 0.5\, t(H-1,N) \qquad (3)$$

If we assume that $p(x) = x$,

$$t(H,N) \ \geq \ \theta(N) \ + \ x\, t(H-1,Nx) \ + \ (1-x)\, t(H-1,N(1-x)) \ + \ 0.5\, t(H-1,N) \qquad (4)$$

and, if we further assume that x is constant throughout, we can solve Equation 4 by induction on H (see Appendix A for details), obtaining

$$\begin{aligned} t(H,N) \ &\geq \ \theta(N) \sum_{i=0}^{H} \left(x^2 + (1-x)^2 + \frac{1}{2}\right)^i \ \geq \ \theta(N) \sum_{i=0}^{H} (1+z)^i \\ &\geq \ \theta\left(N\ (1+z)^H\right) \qquad \text{where } z = 2(x-0.5)^2 \end{aligned} \qquad (5)$$

Obtaining a solution to Equation 4 is much more complex when the weight balance x is not the same for every split, but the solution has a similar general form, *i.e.,* $t(H,N) \approx \theta(N) \sum_i \prod_j (1+z_{ij})$ and we should expect a similar result, namely $t(H,N) \approx \theta(N\ (1+\xi)^H)$, where ξ is a geometric mean of the various z_{ij} terms. This expectation is borne out by the simulation results shown in Figure 5, where each data point is the average of 100 repetitions of Equation 4 when x is random and uniformly distributed on $(0,1)$ (and assuming that $p(x) = x$ and $\theta(N) = N$).

When $x = 0.5$ in Equation 5, $z = 0$ and we have simply a linear lower bound on $t(H,N)$. We note, however, that this result is obtained only when $x = 0.5$ and x is constant throughout the tree. When x varies from node to node in the tree, the expected behavior is exponential in H, as shown in Figure 5. This result obtains because ξ is a mean of z_{ij} terms, where $z_{ij} = 2(x_{ij} - 0.5)^2$ — if the mean x is 0.5, then the arithmetic mean of z_{ij} is simply twice the variance of x, and ξ is monotone to the variance (they are not strictly proportional, because ξ is a geometric, rather than an arithmetic mean). Additional simulations using different distributions for x confirm this relationship between ξ and the variance of x:

Figure 5: Simulated Incremental Run Times for $p(x) = x$

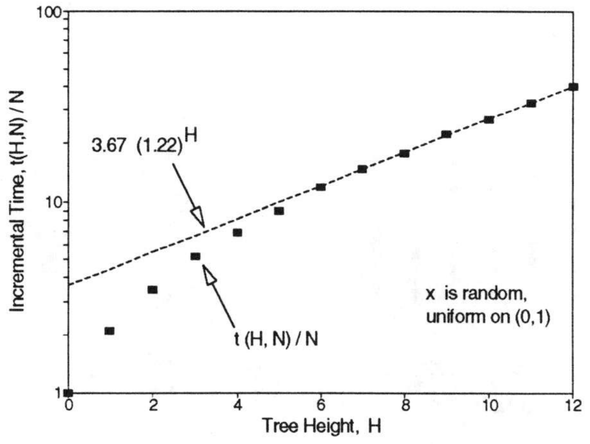
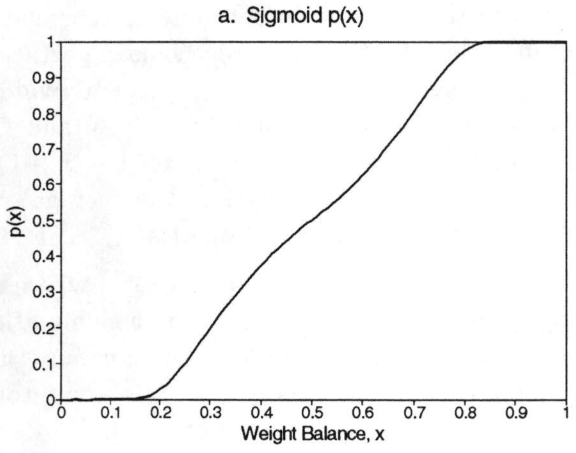
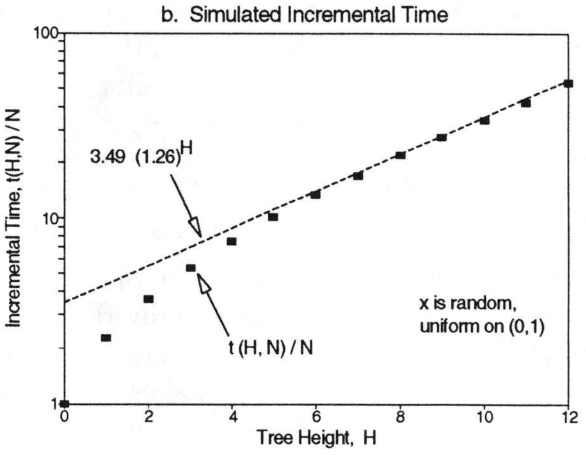


Figure 6: Effects of a Sigmoid $p(x)$





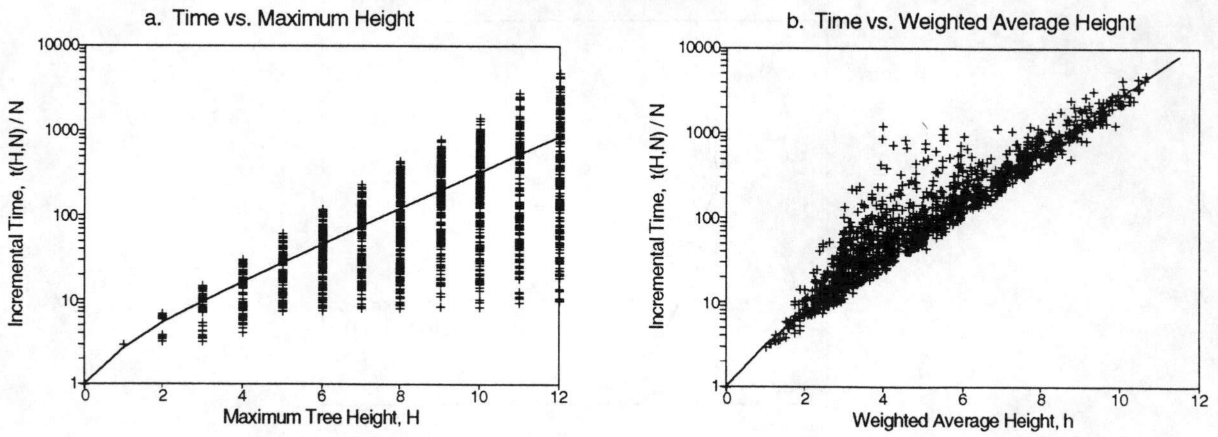
distribution	variance	ξ
triangular	0.042	0.166
uniform	0.083	0.220
inverted triangular	0.229	0.280

$t(H, N)$ is also exponential in H for other forms for $p(x)$ in Equation 3, as shown by the simulation results in Figure 6 for a sigmoid $p(x)$ function and uniformly distributed x. A sigmoid $p(x)$ is more plausible than the linear form, because of certain boundary constraints for both very large and very small x (see Appendix B for a discussion of $p(x)$ and a rationale for the sigmoid).

The recurrence (Equation 3) on which Equation 5 and Figures 5 and 6 are based gives only a lower bound on the expected behavior, obtained by omitting the expected time to evaluate the shallower subtree at each internal node. (See Appendix A for the derivation of Equation 3). The relative contribution of the omitted subtrees increases as their height or weight increases and also as the frequency of the shallower subtree being heavier than the deeper one increases.

When this frequency is low, the contribution of the shallower parts of the tree is small and the incremental run time tends to be governed by the depth of the deepest leaf, H, as in our lower

Figure 7: Correlation of Time and Height



bound analysis. When the frequency is high, the height and weight of the shallower parts of the tree play a larger role, and we expect that the incremental run times would be more nearly correlated with a weighted average leaf depth than with the maximum leaf depth of the tree. This expectation is confirmed by the simulation results shown in Figure 7, where the data points reflect individual simulation outcomes (not averages) for a relatively high frequency, and where $\theta(N) = N$ and the approximate height ratio, f, of the shallower subtree (of height r, where $r = \text{round}(f \times (H-1))$) and the weight balance, x, of the tree were assumed to be constant throughout — the average tree height plotted in Figure 7b is weighted by the fraction of instances reaching each leaf.

Based on these results, we expect run times to be approximately $t(H,N) \propto N\ (1+\xi)^{h^*}$, where h^* is an average height (not necessarily the weight average height h). We note that in the simulation results depicted in Figures 5, 6, and 7, this describes the asymptotic (large h^*) behavior; but that the incremental time for very shallow trees is faster and increases more rapidly than this asymptotic behavior. This is because we have used θ-notation, as in Equation 5, wherein (for $z > 0$)

$$\sum_{i=0}^{H} (1+z)^i = \frac{(1+z)^{H+1} - 1}{z} = \theta((1+z)^H) \tag{6}$$

Equation 6 can also be written as

$$\sum_{i=0}^{H} (1+z)^i = (1+z)^H\ f_1(H,z) \tag{7}$$

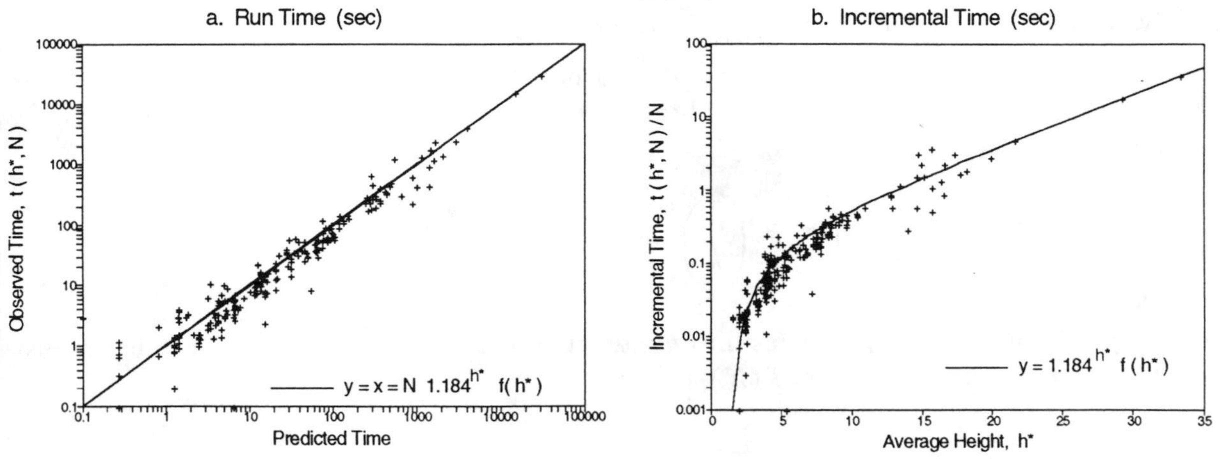
$$\text{where} \qquad f_1(H,z) = 1 + \frac{1}{z}\left(1 - (1+z)^{-H}\right), \qquad \text{and } 1 \le f_1(H,z) \le \frac{1+z}{z}$$

The function $f_1(H,z)$ is monotone, increasing in H with diminishing slope.

For the post-processing function, $T_E(H,N) \le T_P(H,N) \le \sum_{i=0}^{H} T_E(i,N)$ and, when $T_E(H,N) = \theta(N)\ (1+z)^H\ f_1(H,z)$, the upper bound can be re-written as

$$\begin{aligned} T_P(H,N) &\le \theta(N) \sum_{i=0}^{H} \frac{(1+z)^{i+1} - 1}{z} \le \theta(N) \left(\frac{(1+z)^{H+2} - 1}{z^2} - \frac{(H+2)}{z} \right) \\ &\le \theta(N)\ (1+z)^H\ f_2(H,z) \end{aligned}$$

Figure 8: Post-processing Time



$$
\begin{aligned}
\text{where} \qquad f_2(H,z) &= (f_1(H+1) - (H+2)) \;/\; z \\
&= f_1(H,z) + \frac{1+z}{z^2}\left(1-(1+z)^{-H}\right) - \frac{H}{z}(1+z)^{-H} \\
\text{and} \qquad 1 \le f_2(H,z) &\le \left(\frac{1+z}{z}\right)^2
\end{aligned}
$$

The function $f_2(H,z)$ is also monotone, increasing in H with diminishing slope. The shape of the f_1 and f_2 functions is very similar, though f_2 has a higher asymptotic limit than f_1 and converges more slowly. Both f_1 and f_2 are instances of the general form

$$
\begin{aligned}
&f(H) = k_0 - (k_1 + k_2\, H)\,(1+z)^{-H} \\
&\text{in } f_1: \quad k_0 = \left(\frac{1+z}{z}\right) \qquad k_1 = \frac{1}{z} \qquad k_2 = 0 \\
&\text{in } f_2: \quad k_0 = \left(\frac{1+z}{z}\right)^2 \qquad k_1 = \frac{1}{z}\left(2+\frac{1}{z}\right) \qquad k_2 = \frac{1}{z}
\end{aligned}
$$

From these results, by substituting the average height h^* for H and ξ for z, we expect the run times for empirical data which includes both very deep and very shallow trees to be described by

$$
\begin{aligned}
t_p(H,N) &\approx N\,(1+\xi)^{h^*}\, f(h^*) \qquad (8) \\
\text{where} \qquad f(h^*) &= k_0 - (k_1 + k_2 h^*)\,(1+\xi)^{-h^*}
\end{aligned}
$$

As shown in Figure 8, a very good fit to empirical data[2] spanning 5 decades of run time is obtained using Equation 8 with $h^* = \sqrt{h\,h'}$, where h and h' are, respectively, the weighted average heights of the tree before and after post-processing, and

$$k_0 = 0.128 \pm 0.017 \text{ sec}, \qquad k_1 = 0.174 \pm 0.054 \text{ sec}, \qquad k_2 = 0, \qquad \xi = 0.184 \pm 0.005$$

4 Discussion

Both tree-building and post-processing time have components which are exponential in the height of the inferred decision tree. This fact should be kept in mind in making design decisions within

[2]234 observations: 16 data sets, 9 heuristic functions, and 2 methods for handling continuous attributes (in 10 of the data sets).

the TDIDT framework. While the dominant goal is to maximize predictive accuracy, we must remember that TDIDT inherently compromises in order to obtain practical algorithms by using greedy, heuristic search and by limiting the candidates.

The leaves of a decision tree are equivalent to a partitioning of the data set, and the number of ways of partitioning the set (*i.e.*, the number of possible distinct decision trees) is intractable. More concretely, there are

$$S(N,m) = \frac{1}{m!} \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} k^N$$

ways of partitioning a set of N items into m non-empty subsets[3]; and the total number of distinct possible partitionings of N items, $\mathcal{P}(N)$, is

$$\mathcal{P}(N) \equiv \sum_{m=1}^{N} S(N,m)\,; \qquad \lfloor N/2 \rfloor! \leq \mathcal{P}(N) \leq N!$$

Obviously, practical algorithms can search only a small fraction of such a factorially large space.

We refer to the subset of possible splits considered at any internal node of a decision tree as the candidate set. A is the size of this set, and $A-1$ is an upper limit on the height of the subtree. Both tree-building time and post-processing time potentially increase exponentially with A. The obvious implication, in terms of designing practical TDIDT algorithms, is that a premium is placed on methods for defining the candidate set which minimize the dimensionality of the candidate set without sacrificing accuracy. In particular, (a) sound techniques from multivariate statistical analysis, such as factor analysis, discriminant analysis, and cluster analysis should be pursued (see Kachigan [14] for a review of these, and also see [5, 13, 15, 30] for some current machine learning approaches) and (b) approaches which replace a multi-way partitioning with logically equivalent binary splits should be avoided, as this unnecessarily increases the height of the tree and the learning time.

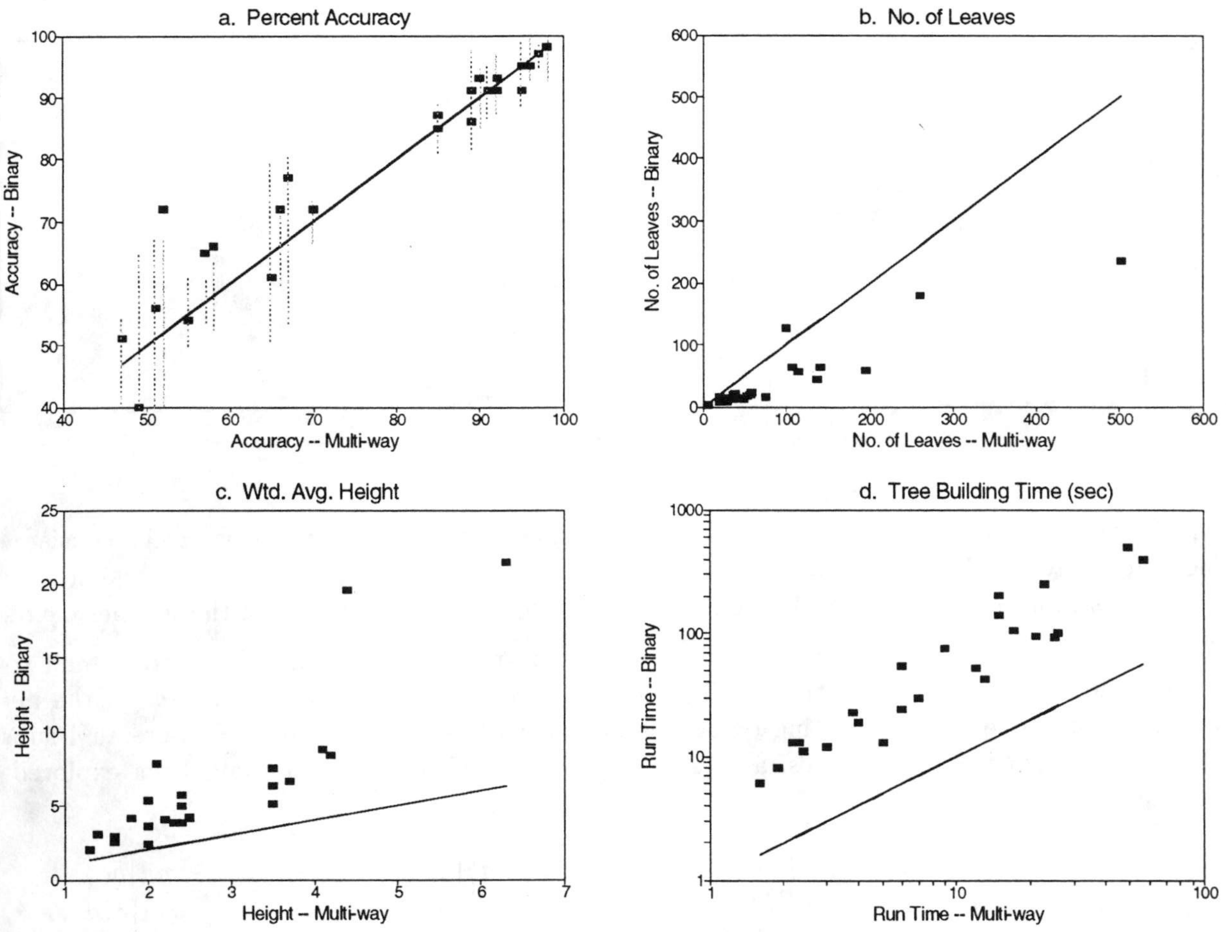
With regard to the second item above, we note that most current methods for handling continuous attributes (see [11] for an exception) treat all continuous attributes as a series of binary splits. The handling of continuous attributes is inherently a problem of optimal clustering analysis, and one of the most significant unresolved problems in this area is deciding how many subgroups, or clusters, there should be [10]. The approach of making only binary splits avoids coming to grips with this problem directly but, since it leads to deeper trees and to re-visiting the candidate set definition at every decision node, it does so at considerable computational expense.

We have confirmed these observations experimentally using 16 data sets from various application domains (see [17] for complete details of the experiments). Ten of these data sets involved continuous attributes which were converted to nominal attributes in two ways: (1) using arbitrary cut-points at approximately the quartile values for the attribute (approximate because the cuts do not separate items having the same value for the attribute) and (2) by choosing cut-points at the local minima of smoothed histograms of the attribute distribution (the 'natural' cut-points),

Of the 26 resulting data sets, 2 involved only binary attributes, and 24 had attributes of different arities. These 24 data sets were converted to all binary splits by simply creating a new binary

[3] $S(N,m)$ is a Stirling number of the second kind (see [1, pp. 822-827]). Also, note that Hyafil and Rivest [12] have formally shown that learning optimal decision trees is NP-complete.

Figure 9: Multi-way *vs.* Binary Splits



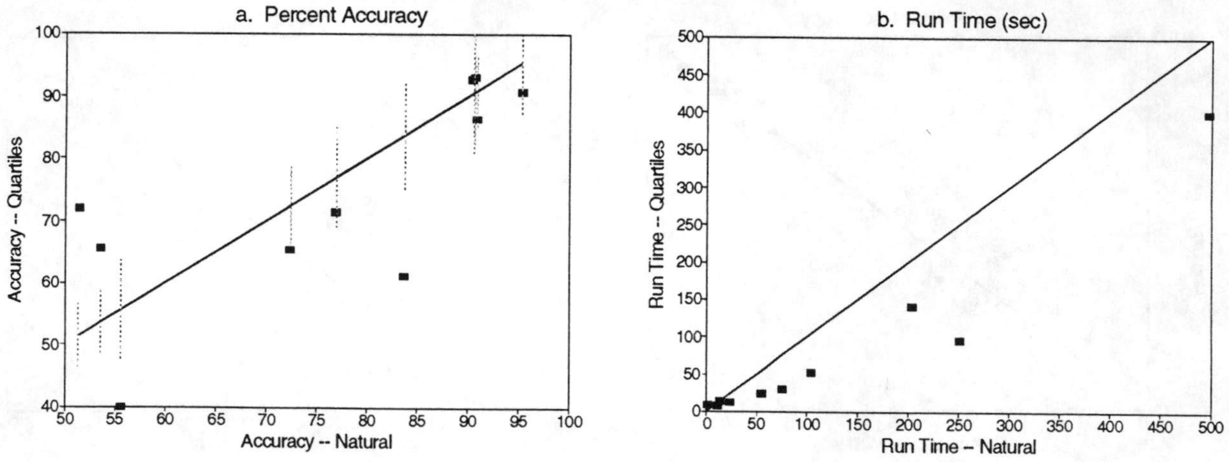
attribute for each attribute-value pair. Other methods of merging attribute values to create new attributes of lower arity, *e.g.*, mapping attributes into binary forms, are explored in C4.5 [26], CART [3], and ASSISTANT [7].

Figure 9 illustrates the differences between the multi-way and binary results, using information gain to choose among candidate splits.

> *Note* — In Figure 9 and in other, similar figures throughout this section, each data point represents the accuracy (or run time, *etc.*) obtained for one set of data under two different versions of the TDIDT algorithm, one version plotted on the vertical axis of the graph and the other version on the horizontal axis. Each figure also shows a line, $y = x$, such that for points lying along the line there is no preference for one version over another — for points lying below and to the right of this line, the x-axis version has a larger value of the measure in question; for points above and to the left, the y-axis version has a larger value. The accuracy plots also show error bars, indicating the approximate confidence interval for the x-axis version.

The multi-way trees consistently have more leaves, are shallower, and are learned more quickly than their binary counterparts. The differences in accuracy between the two is generally negligible but,

Figure 10: Continuous Attribute Handling



on the few occasions when there is a notable difference, the binary trees appear to be consistently more accurate. Similar results were obtained using a different heuristic, except that the occasional differences in accuracy involved different data sets (see [17] for full details of these experiments).

The following example illustrates how the differences arise, and why the binary trees tend to be more accurate. Consider an attribute X which has 4 possible values, $X = a, b, c, d$. In the multi-way tree, only one way of splitting on X has been considered for each leaf (*i.e.,* create 4 subsets $(a\,|\,b\,|\,c\,|\,d)$). For the binary trees, a total of 14 possible splits on X may have been explored for each leaf:

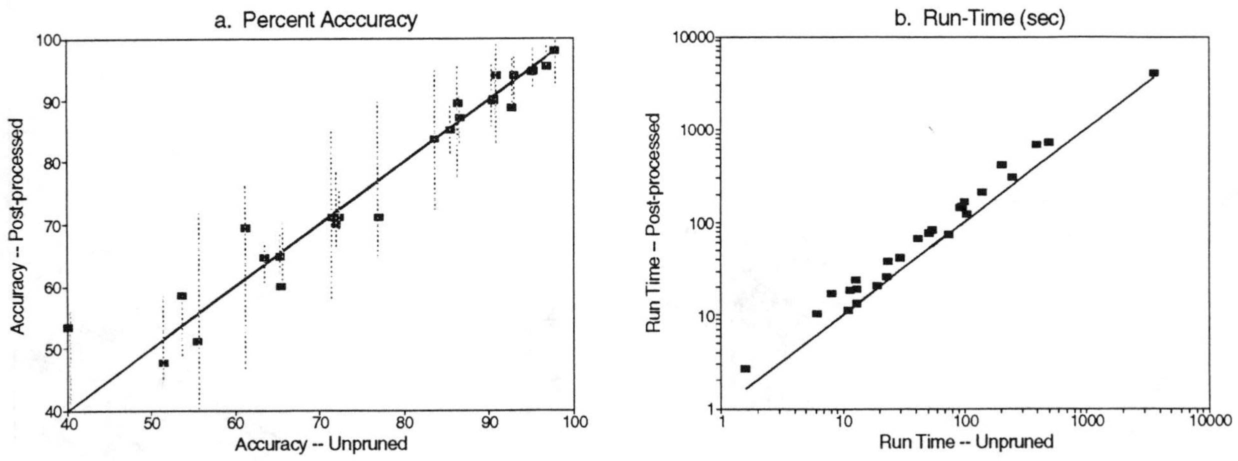
$a\,\|\,bcd$	$b\,\|\,acd$	$c\,\|\,abd$	$d\,\|\,abc$	$ab\,\|\,cd$	$ac\,\|\,bd$	$ad\,\|\,bc$
$ab\,\|\,c\,\|\,d$	$ac\,\|\,b\,\|\,d$	$ad\,\|\,b\,\|\,c$	$a\,\|\,b\,\|\,cd$	$a\,\|\,c\,\|\,bd$	$a\,\|\,d\,\|\,bc$	$a\,\|\,b\,\|\,c\,\|\,d$

From this example we see that remapping the multi-valued attributes into a new set of binary attributes indirectly expands the candidate set and, by heuristically choosing among a larger set of candidates, it is sometimes possible to improve accuracy. However, a large penalty is paid in terms of increased run time and tree complexity.

By considering a number of partitioning alternatives rather than just either creating one subset for each attribute value, or not, we improve our chances of finding a more accurate tree. What is clear from the examples above is that if we do not do this in an informed way (*i.e.,* if we just blindly force every decision into a binary form) we may confront a combinatorial explosion in complexity. This problem will become more acute as the candidate set is expanded to consider compound features (functions of several attributes), and as the application areas are expanded beyond current, relatively simple, problems into domains where there may be scores of attributes and thousands of instances.

Figure 10 illustrates the differences between the two ways of handling continuous attributes for binary trees, using information gain to choose among candidate splits (similar results are obtained using other heuristics). The quartiles method divides the data into 4 approximately equal subsets, so that when converted to binary form the most unbalanced split is approximately 75/25. The 'natural' cut-points method, by contrast, tends to produce very unbalanced splits. The differences in accuracy between the two methods may be large, and either method may be the more accurate one. The quartiles trees consistently have more leaves, but are shallower (more efficient and more

Figure 11: Post-Processing



quickly learned). The degree of imbalance in the splits is an important characteristic for selecting among alternative ways of handling continuous variables. Some alternative methods for handling continuous attributes are given in [11] and [26].

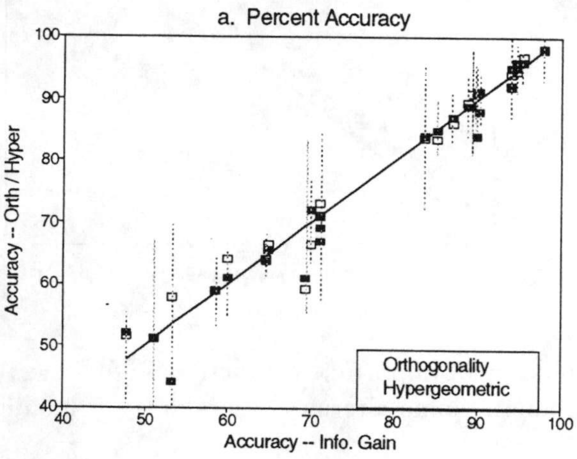
Figure 11 shows the effects of post-processing for binary trees built using the information gain heuristic. Post-processing does not improve the accuracy of the trees, and only occasionally does it significantly reduce the number of leaves or the average height of the trees. It does, however, consistently increase the learning time by as much as a factor of 2. In the few cases where post-processing does significantly modify the trees, the unpruned trees are degenerate, in the sense that a significant number of the candidate splits are very unbalanced, and the split selection heuristic function tends to choose these in preference to more balanced splits. Mingers [20] has previously noted and remarked on the tendency of information gain and similar heuristics to favor very unbalanced splits.

A major source of these unbalanced splits is the handling of continuous attributes — particularly the natural cut-points method, because it tends to create separate subsets for the 1 or 2 outliers which are frequently found. Methods based on information gain, gain ratio, *etc.,* share this problem, because of their bias towards unbalanced splits. The problem is not restricted to continuous attributes, however, as many nominal attributes have one dominant value (which might be labeled 'normal') and one or more infrequent, abnormal values. A particular application domain for which this problem is acute is the word-sense disambiguation problem as posed by Church, *etal* [8], who use a Mutual Information (information gain) heuristic to select which co-occurring words are useful features for deciding among alternative senses of a verb. In this application, the bias towards unbalanced splits is expressed as a preference for selecting a word which occurs only once (a proper noun, for instance) over a more common and more pertinent word.

Figure 12 shows the effects of using different split selection heuristics for post-processed binary trees. The three functions shown are the most typical (information gain) and the two most different (orthogonality and the hypergeometric), from a study of 9 different functions [17]. There are no significant differences in accuracy between these metrics, and the inferred trees all have about the same number of leaves. The height of the trees and the learning time, however, do vary significantly and systematically. For the worst case (word sense) data, the learning times differ by a factor of 12 (10 hours for orthogonality *vs.* 50 minutes for the hypergeometric).

Figure 13 shows the differences in the ranking of various splits by the heuristics, for a data set

Figure 12: Heuristic Functions



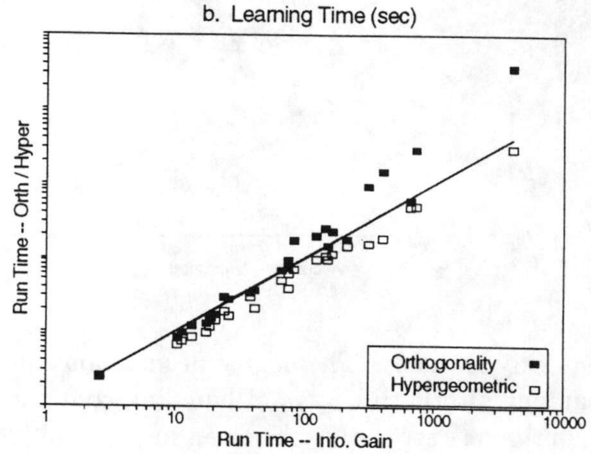
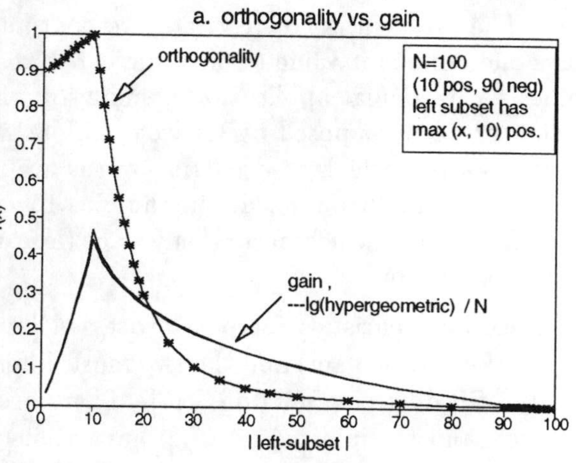
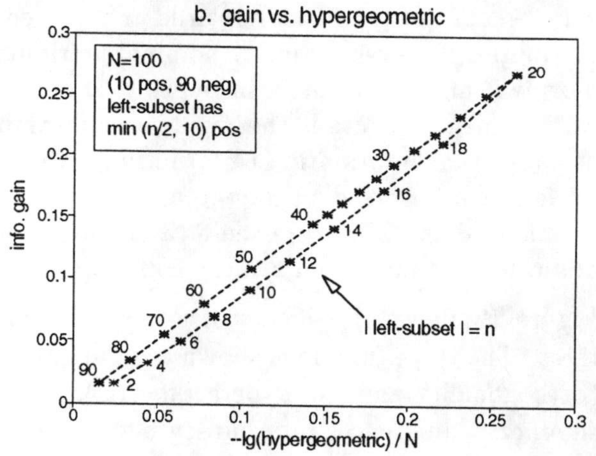


Figure 13: Heuristic Function Values





containing 100 items only 10 of which are positive. Figure 13a shows the most extreme splits, in that the left subset always contains as many positive items as possible as the size of the left subset is varied. Notice that orthogonality has a very high value (0.9, very near the maximum possible value of 1) for a 1/99 split in which the 1 item in the left subset happens to be positive. This apparent association between left subset membership and a positive classification would occur by chance in 10% of all 1/99 splits. Any inference that left subset membership is predictive of a positive classification stands on very thin evidence. By contrast, consider a 10/90 split in which 9 of the 10 left subset items are positive — there is virtually no chance that this would occur randomly ($< 10^{-10}$), and yet orthogonality values this split less highly than the 1/99 split (0.88 *vs.* 0.90).

As indicated in Figure 13a, information gain and the hypergeometric are closely related, and it can be shown [17] that gain $\approx -\log_2(\text{hypergeometric})/N$. As shown in Figure 13b, this relationship is only approximate, and the two functions may rank candidate splits differently when the splits and classes are unbalanced (for instance, information gain ranks a 40/60 split with all 10 positive items in the left subset ahead of a 14/86 split with 7 positive items in the left subset, while the hypergeometric reverses the order). The hypergeometric is better justified on statistical grounds than the other heuristic functions [17], and it also gives better results empirically (see Figure 12).

The final area to be explored is the question of whether to prune the decision trees, and whether to simply stop splitting or to post-process (and whether to perform tree surgery beyond mere pruning). From the point of view of minimizing learning time, there can be no question that post-processing is very expensive, and that stopping is a more effective approach. There have been a number of studies in the area of stopping, pruning, *etc.* [4, 6, 19, 20, 23, 24, 29]. Some of the noteworthy findings are that: (1) in general, it seems better to post-process using an independent data set than to stop, (2) cross-validation seems to work better than point estimates such as the χ^2 test, (3) the decision to prune is a form of bias — whether pruning will improve or degrade accuracy depends on how appropriate the bias is to the problem at hand, and (4) pruning may have a negative effect on accuracy when the training data are sparse.

Figure 14 illustrates the three operations (pruning, stopping, and tree surgery) that we have considered here. Note that stopping and pruning are basically the same operation, differing in that the decision whether to stop is based on only local information, whereas the decision whether to prune is based on information from subsequent splits (look-ahead). Note also that the effect of the surgical operation depicted in Figure 14c can be achieved by a simple pruning or stopping operation performed on a different tree in which the order of the splits on attributes B and C is reversed (which might be achieved by using a different split selection heuristic). This point is very important, since it is the tree surgery operations which drive the exponential growth of run time *vs.* tree depth in post-processing — a simple post-pruning algorithm that does not consider the surgical option would have $\theta(HN)$ time complexity.

The surgical option can be viewed as an expensive *ex post* attempt to correct for a bad choice made during tree building, which is necessary because the selection heuristic does not order the splits correctly from the point of view of efficient pruning. We saw earlier in Figure 12 that the choice of the selection heuristic is largely a matter of complexity, not of accuracy. Thus, in choosing among alternative selection heuristics, we should prefer heuristics which tend to build balanced, shallow trees in which the splits are ordered so as to allow efficient pruning.

The preceding observations concerning the ordering of splits are doubly important in the context of stopping, since stopping is simply a pruning decision made with less information — if the splits are not ordered so as to allow efficient pruning, they cannot be be ordered correctly from the point of view of stopping. If our options are simply to accept the split put forward by the heuristic

Figure 14: Pruning, Stopping, Tree Surgery

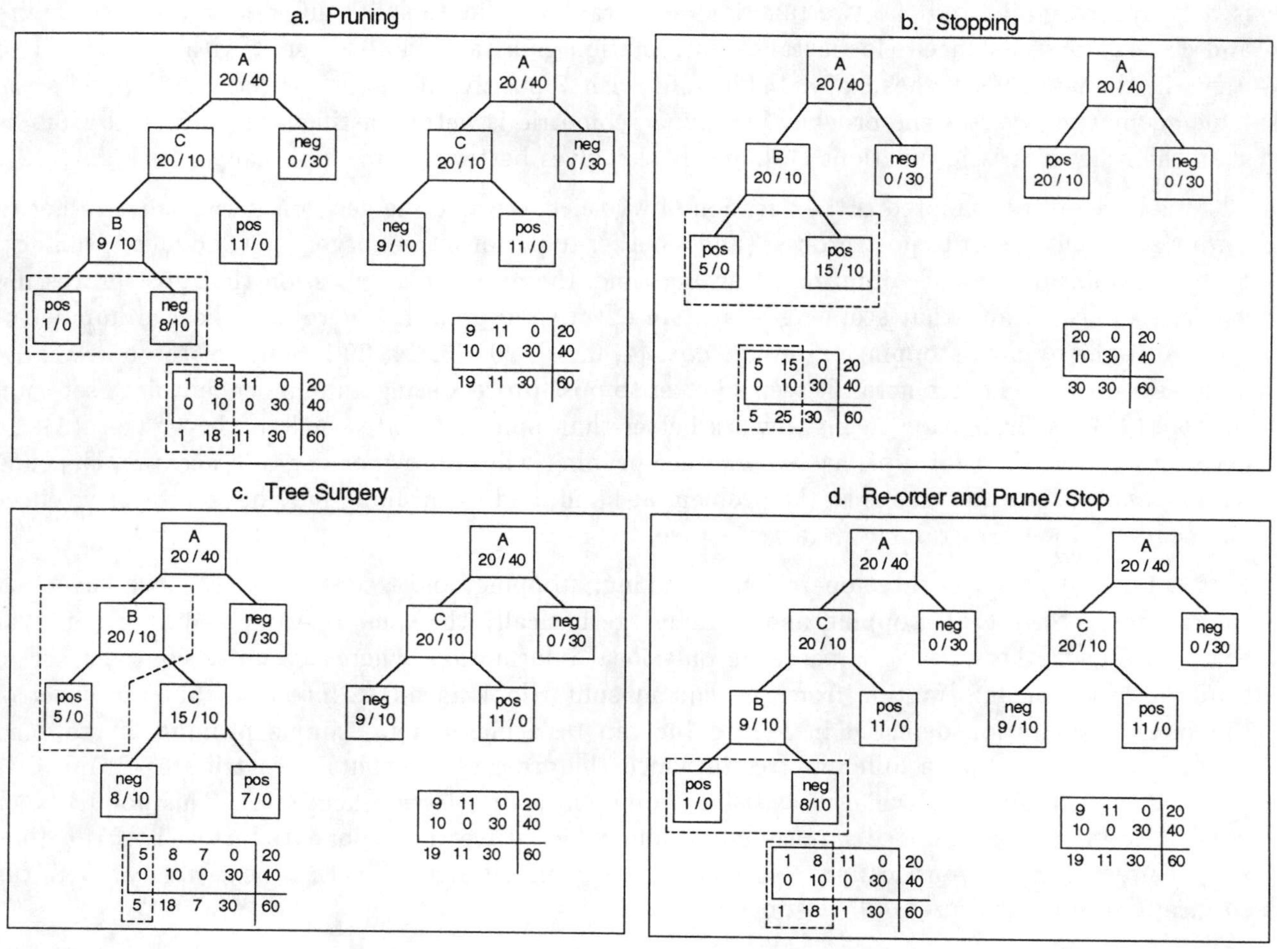
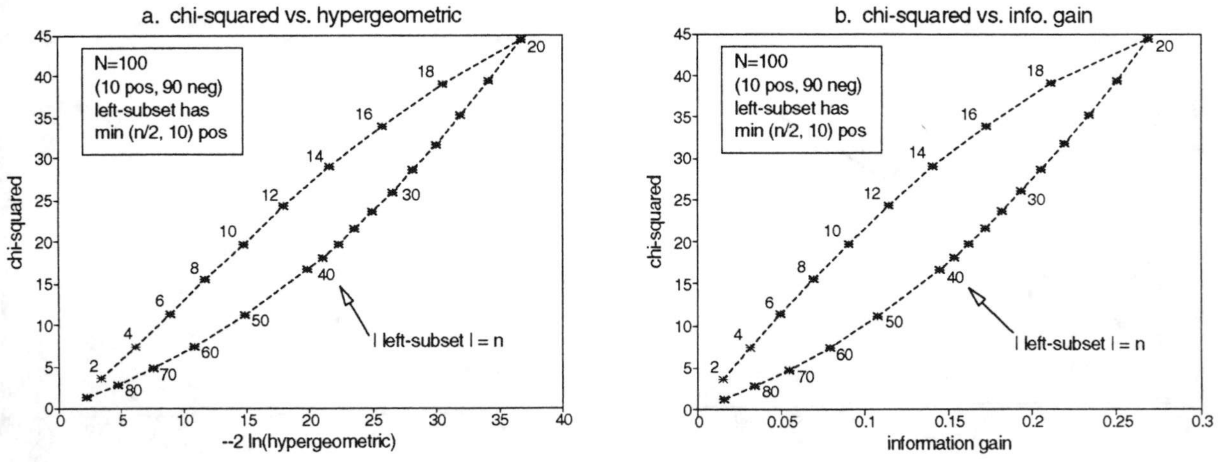


Figure 15: Chi-squared Statistic



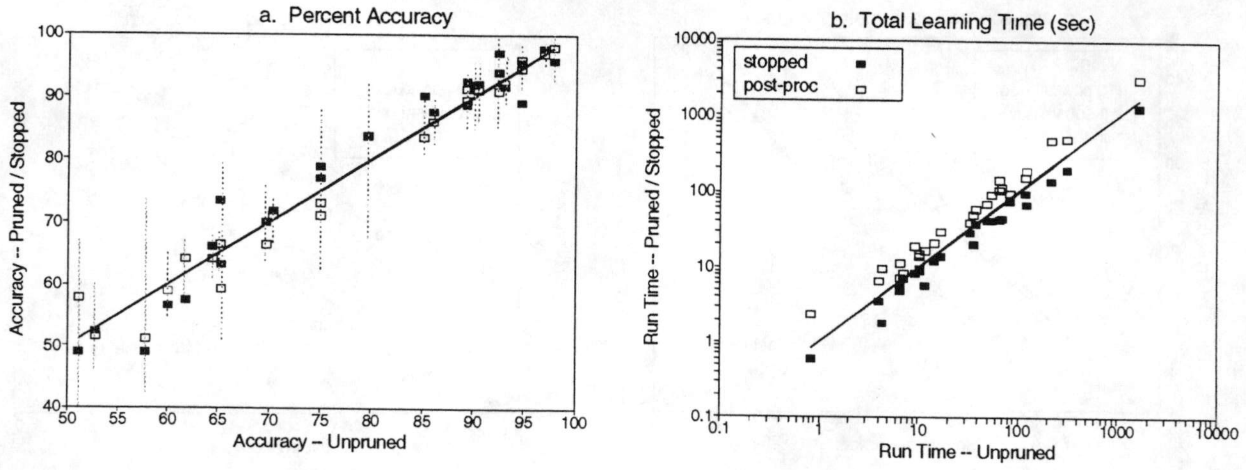
and recursively split its subsets, or to stop, these stopping decisions cannot be made correctly if the heuristic advances the wrong candidate. In particular, if the split selection heuristic (*e.g.*, information gain) and the stopping criterion function (*e.g.*, χ^2) differ significantly as to the relative merit of a candidate (and these two do), then our stopping strategy (*e.g.*, choose a candidate based on information gain and accept or reject it based on χ^2) will almost certainly lead to poor results.

We saw in Figure 13 that information gain and the hypergeometric function are closely related (there is a strong log-linear correlation), but may differ in the relative order of a balanced *vs.* an unbalanced split. In Figure 15a, we show a similar log-linear relationship between the χ^2 statistic and the hypergeometric, and Figure 15b shows a direct comparison of gain and χ^2. In the series of splits depicted in Figures 13 and 15, two splits which have the same value for the hypergeometric function have different values for the gain function, with gain preferring a split having more than 20 items in the left subset over a split having fewer than 20 items in the left subset. The splits also have different values for the χ^2 function, but the relative preference for fewer or less than 20 items in the left subset is reversed. Clearly, information gain does not rank candidate splits in the best order from the point of view of efficient pruning using χ^2.

The obvious solution to this dilemma is to use the same evaluation function for ranking splits as for deciding whether to prune. Neither the information gain function nor the χ^2 function is suitable for this dual purpose, for the following reasons: (1) the χ^2 significance test for contingency tables is only an approximate test — the approximation is not good and comparisons of two χ^2 values are biased whenever any of the expected values (e_{cv}) for the cells in the contingency tables is small, where $e_{cv} \equiv n_c m_v / N$, (2) Cochran's criteria for the admissibility of χ^2 [28, p. 575] are that *all* $e_{cv} \geq 1$ and at least 80% of the e_{cv} are ≥ 5 — the tree building phase of TDIDT leads to increasingly smaller subsets and, regardless of the initial data set, Cochran's criteria cannot be satisfied after at most $\log_2(N_0/5)$ splits have been made (where N_0 is the initial data set size), and (3) $G^2 = 2N \ln(2) \times$ gain is Wilks' log-likelihood χ^2 statistic [2, pp. 48-49] — gain has approximately a χ^2 probability distribution when scaled appropriately, and it shares the bias and admissibility problems of χ^2 when Cochran's criteria are not met.

The quantity that χ^2 and gain approximate is the logarithm of the hypergeometric function, which expresses the exact probability of obtaining the observed values of f_{cv} given the values of n_c and m_v and assuming that the class of an item is independent of its subset membership. That is, the hypergeometric expresses the likelihood that the observed degree of association between subset

Figure 16: Stopping *vs.* Post-processing



membership and class is simply coincidental. The statistical significance test for 2×2 contingency tables based on the hypergeometric distribution is known as Fisher's Exact Test [2].

Figure 16 compares the results of stopping using the hypergeometric probability function (stop when the function is > 0.05) to unstopped trees and post-processed trees for binary trees built using the hypergeometric to select splits. Stopping in this way is not detrimental to accuracy (in the only case where there was a statistically significant difference, stopping was beneficial). Post-processing accomplished little or nothing for these trees, and consistently doubled the run time. The stopped trees are markedly simpler than the unstopped trees, and are learned in about half the time (one-fourth the time for post-processing).

5 Conclusions

1. Insofar as potentially improving predictive accuracy of TDIDT trees is concerned, the important design decisions are those which expand the candidate set, either directly (as in allowing a bounded look-ahead) or indirectly (as in making only binary splits on continuous attributes). That is, the more candidate splits are explored at each decision node, the more likely that the sequence of greedy split selections will yield a more accurate tree. Other factors, such as the choice of a split selection heuristic and stopping or pruning, generally have little or no impact on accuracy and, in the few cases where accuracy is significantly affected, the effect is sometimes beneficial, sometimes harmful.

2. In expanding the candidate set we are faced with a combinatorial explosion of potential candidates, as there are factorially many ways to split a data set. It is for this reason that TDIDT algorithms are restricted to considering only splits on the value of a single attribute (or a simple combination of only a few attributes, such as a bounded look-ahead).

   For single-attribute, multi-way splits on A discrete variables, the time to build a tree for a data set of size N is $O(A^2N)$. If all V-ary splits for $V > 2$ are converted to an equivalent series of $V-1$ binary splits, this becomes $O(d^2N)$ where d is the dimensionality of the data set, $d = \sum(V_i - 1)$. For continuous attributes, each $V_i = O(N)$, and the tree building time becomes $O(A^2N^3)$. Thus, even for single-attribute splits, there is potentially a large payoff for some form of pre-processing to reduce the dimensionality of the problem, especially as regards the handling of continuous attributes and the binarization of multi-way splits. This will become even more important as the candidate sets are expanded to consider splitting on functions of several variables and as data sets of greater size and dimensionality are taken on.

3. Both tree building and post-processing have components which increase exponentially as the height of the learned tree increases. This puts a great premium on design decisions which tend to produce shallower, more balanced trees, such as: (a) using multi-way splits rather than forcing only binary splits, (b) methods for handling continuous attributes (and for possibly merging subsets of discrete attributes) which tend to favor more balanced splits, and (c) selecting heuristic functions for choosing among candidate splits which tend to favor more balanced splits, rather than those which are biased towards very unbalanced splits.

4. Simple algorithms for pruning are $O(Nh^*)$, as opposed to $O(N(1+\xi)^{h^*})$ for tree surgery. Tree surgery is equivalent to simply pruning or stopping an alternative tree in which the split which appears at the root of the current tree is made last, rather than first. Thus, the need for such surgical operations arises from using different criteria for split selection and for pruning or stopping (*e.g.*, information gain for selection and χ^2 for stopping).

   The χ^2 test is biased, and should not be used, when Cochran's admissibility criteria are not satisfied. The divide-and-conquer TDIDT process almost inevitably leads to a violation of Cochran's criteria, and χ^2 is not suitable for use in TDIDT. Information gain is equivalent to Wilks' log-likelihood chi-squared function and, like the more familiar Pearson's χ^2 test, information gain is not admissible when Cochran's criteria are not satisfied.

5. The hypergeometric distribution function found in Fisher's exact test is admissible even when χ^2 and information gain are not, and the hypergeometric allows efficient pruning or stopping. It also tends to build simpler, more balanced, and shallower trees which are no less accurate than those built using other heuristics. Learning time can be reduced by as much as an order of magnitude by using the hypergeometric.

A Appendix — Derivation of Expected Post-Processing Time

$$T_E(H,N) = \theta(N) + T_E(H_L,Nx) + T_E(H_R,N(1-x)) + \begin{cases} T_E(H_L,N) & \text{if } x \geq 1/2 \\ T_E(H_R,N) & \text{if } x < 1/2 \end{cases}$$

With probability $p(x)$: $H_L = H-1$, $H_R = r \leq H-1$,

$$T_E(H,N) = \theta(N) + T_E(H-1,Nx) + T_E(r,N(1-x)) + \begin{cases} T_E(H-1,N) & \text{if } x \geq 1/2 \\ T_E(r,N) & \text{if } x < 1/2 \end{cases}$$

where $p(x)$ is a symmetric, non-decreasing function of x (*i.e.*, $p(1-x) = 1-p(x)$ and if $x \geq 0.5$ then $p(x) \geq 0.5$).

Using the principle that $\text{mean}(X+Y) = \text{mean}(X) + \text{mean}(Y)$,

$$t(H,N) = \theta(N) + t(H-1,Nx) + t(r,N(1-x)) + \begin{cases} t(H-1,N) & \text{if } x \geq 1/2 \\ t(r,N) & \text{if } x < 1/2 \end{cases} \tag{9}$$

Similarly, with probability $1-p(x)$: $H_L = l < H-1$, $H_R = H-1$, and

$$t(H,N) = \theta(N) + t(l,Nx) + t(H-1,N(1-x)) + \begin{cases} t(l,N) & \text{if } x \geq 1/2 \\ t(H-1,N) & \text{if } x < 1/2 \end{cases} \tag{10}$$

Since the events described by Equations 9 and 10 are mutually exclusive with probabilities summing to 1, we multiply them by their respective probabilities and sum, giving

$$\begin{aligned} t(H,N) &= \theta(N) + p\, t(H-1,Nx) + (1-p)\, t(H-1,N(1-x)) \\ &\quad + p\, t(r,N(1-x)) + (1-p)\, t(l,Nx) \\ &\quad + \begin{cases} p\, t(H-1,N) + (1-p)\, t(l,N) & \text{if } x \geq 1/2 \\ (1-p)\, t(H-1,N) + p\, t(r,N) & \text{if } x < 1/2 \end{cases} \\ \text{where} &\quad 0 \leq r \leq H-1 \quad \text{and} \quad 0 \leq l < H-1 \end{aligned} \tag{11}$$

and, if we neglect the $t(l,i)$ and $t(r,i)$ terms,

$$t(H,N) > \theta(N) + p\, t(H-1,Nx) + (1-p)\, t(H-1,N(1-x)) + 0.5\, t(H-1,N)$$

If we assume that $\theta(N)=N$ and $p(x)=x$, then

$$t(H,N) > N + x\, t(H-1,Nx) + (1-x)\, t(H-1,N(1-x)) + 0.5\, t(H-1,N)$$

If we let $y=(1-x)$ and further assume that x is constant throughout the tree then,

$$t(H,N) \geq N \sum_{i=0}^{H} (x^2+y^2+0.5)^i = N \sum_{i=0}^{H} \left(1+2(x-0.5)^2\right)^i$$

which we prove by induction on H. The base case, $H=0$, is trivially true. Assume that $t(H,N) \geq N \sum_{i=0}^{H}(x^2+y^2+0.5)^i$ is true for $H \leq h-1$, and consider $H=h$.

$$\begin{aligned} t(h,N) &> N + x\, t(h-1,Nx) + y\, t(h-1,Ny) + 0.5\, t(h-1,N) \\ &= N + x\, Nx \sum_{i=0}^{h-1} (x^2+y^2+0.5)^i \\ &\quad + y\, Ny \sum_{i=0}^{h-1} (x^2+y^2+0.5)^i \\ &\quad + 0.5\, N \sum_{i=0}^{h-1} (x^2+y^2+0.5)^i \\ &= N + N\,(x^2+y^2+0.5) \sum_{i=0}^{h-1} (x^2+y^2+0.5)^i \\ &= N + N \sum_{i=1}^{h} (x^2+y^2+0.5)^i = N \sum_{i=0}^{h} (x^2+y^2+0.5)^i \end{aligned}$$

B Rationale for and Effects of a Sigmoid $p(x)$

The function $p(x)$ expresses the likelihood that the left subtree, which covers fraction x of the instances has height $H_L=H-1$ (*i.e.*, is at least as deep as the right subtree).

If subtrees with empty leaves are not allowed, then $p(x)$ must satisfy the boundary conditions that a subtree's weight is greater than its height, *i.e.*,

$$\begin{aligned} p(x) &= \begin{cases} 0, & \text{if } x \leq (H-1)/N \\ 1, & \text{if } x \geq 1-(H-1)/N \end{cases} \\ \frac{dp(x)}{dx} &= 0, \quad \text{if} \quad x \leq (H-1)/N \quad \text{or} \quad x \geq 1-(H-1)/N \end{aligned}$$

$p(x)$ also expresses the general tendency (argued in the body of this report) that heavier subtrees tend to be deeper. Based both on this argument and on the boundary conditions, $p(x)$ is non-decreasing. Since the argument applies equally to the right subtree, and since either $H_L = H-1$ or $H_R = H-1$, then $p(x)$ must also be symmetric, *i.e.*, $p(1-x) = 1-p(x)$.

Because of the boundary conditions, a sigmoid form for $p(x)$ is more plausible than the simpler linear form $p(x) = x$ (though other forms, such as a step function or piece-wise linear function, might also be proposed). For this sigmoid family, the strength of the tendency for heavier subtrees to be deeper is largely expressed by the slope of the central portion of the $p(x)$ curve — when the tendency is absolute (*i.e.*, the heavier subtree is always $H-1$) then $p(x)$ is a step function ($p(x)=0$ if $x < 0.5$, else $p(x)=1$); when there is no such tendency then $p(x)=0.5$ (a coin toss).

Many sigmoid functions might be proposed — we have chosen to apply a symmetric transformation to the variable x and to use the cumulative normal distribution of the transformed variable because it is sigmoid, symmetric, non-decreasing, and expresses a probability.

$$\text{let} \quad w = \frac{H-1}{N} \qquad \text{and} \quad x' = \begin{cases} -\infty & \text{if } x \leq w \\ +\infty & \text{if } x \geq (1-w) \\ \frac{1}{\sigma} \times \dfrac{(x-0.5)}{(0.5-w)-|x-0.5|} & \text{else} \end{cases}$$

$$\text{then} \quad p(x) = \begin{cases} 0 & \text{if } x \leq w \\ 1 & \text{if } x \geq (1-w) \\ F(x') = \displaystyle\int_{-\infty}^{x'} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2}\, dt & \text{else} \end{cases}$$

$$\text{and} \quad F(x') \approx \frac{1}{1+e^{-Z}} \qquad \text{where } Z = \frac{x'}{\sqrt{2\pi}} \left(4 + 8.3452 \sin^2(0.1486\, x')\right)$$

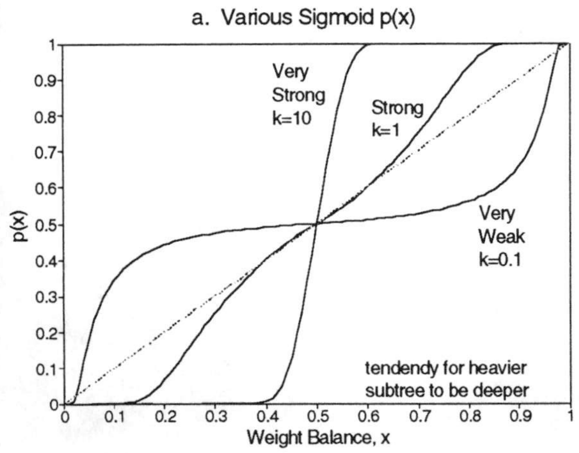
Our transformed variable x' can be interpreted as how far away we are from a 50/50 weight-balanced tree ($x=0.5$), out towards the boundary conditions defined by w, where σ is the unit of measurement. The central slope of $p(x)$ is proportional to $k=\sigma^{-1}$ — a small value of k gives a very flat $p(x)$ (*i.e.*, only a weak tendency for heavier trees to be deeper as $k \to 0$), and a large value of k gives more nearly a step function, (*i.e.*, a very strong tendency as $k \to \infty$), as shown in Figure 17a.

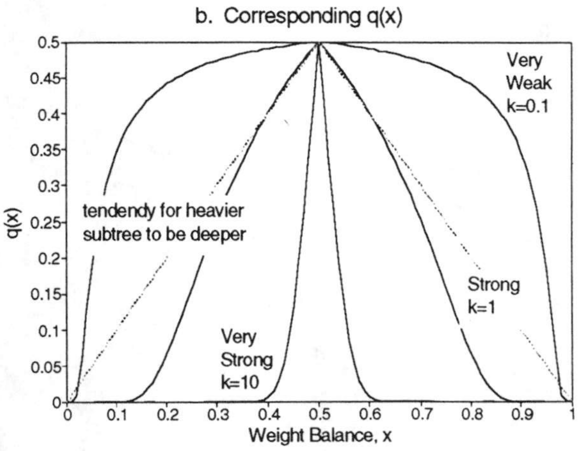
An important characteristic of the data set/tree combination, in addition to the height and weight balance of the splits, is the frequency with which the surgical operation is evaluated on the shallower, rather than on the deeper subtree — when this frequency is low, the run time is largely governed by the maximum depth of the tree H; when this frequency is high, the run time is largely governed by the weighted average depth h. This frequency is given by

$$Q(x) = \begin{cases} 1-p(x), & \text{if } x \geq 0.5 \\ p(x), & \text{if } x < 0.5 \end{cases} \qquad \textit{i.e.,} \quad Q(x) = 0.5 - |p(x)-0.5|$$

and it also is largely governed by k, as shown in Figure 17b.

Figure 17: Contribution of Shallower Subtree





References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions.* Dover Publications, Inc., New York, 1972.

[2] A. Agresti. *Categorical Data Analysis.* John Wiley & Sons, New York, 1990.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, CA, 1984.

[4] W. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.

[5] R. Caruana and D. Freitag. Greedy attribute selection. In *Machine Learning: Proceedings of the 11th International Conference (ML-94)*, pages 28–36, San Francisco, 1994. Morgan Kaufmann.

[6] B. Cestnik and I. Bratko. On estimating probabilities in tree pruning. In *Proceedings of the European Working Session on Learning (EWSL-91)*, pages 138–150, Berlin, 1991. Springer-Verlag.

[7] B. Cestnik, I. Kononenko, and I. Bratko. ASSISTANT 86: A knowledge-elicitation tool for sophisticated users. In *Progress in Machine Learning — Proceedings of EWSL 87: 2nd European Working Session on Learning*, pages 31–45, Wilmslow, 1987. Sigma Press.

[8] K. Church, W. Gale, P. Hanks, and D. Hindle. Using statistics in lexical acquisition. In U. Zernik, editor, *Lexical Acquisition: Exploiting On-Line Resources to Build A Lexicon*, pages 115–164. Lawrence Erlbaum & Assoc., Hillsdale, NY, 1991.

[9] J. F. Elder. Heuristic search for model structure. In *Proceedings Fifth International Workshop on Artificial Intelligence and Statistics*, pages 199–210, Fort Lauderdale, FL, 1995.

[10] B. Everitt. Unresolved problems in cluster analysis. *Biometrics*, 35:169–181, 1979.

[11] U. M. Fayyad and K. B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87–102, 1992.

[12] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-Complete. *Information Processing Letters*, 5:15–17, 1976.

[13] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning: Proceedings of the 11th International Conference (ML-94)*, pages 121–129, San Francisco, 1994. Morgan Kaufmann.

[14] S. K. Kachigan. *Multivariate Statistical Analysis: A Conceptual Introduction.* Radius Press, New York, 2nd edition, 1991.

[15] K. Kira and L. A. Rendell. A practical approach to feature selection. In *Machine Learning: Proceedings of the 9th International Conference (ML-92)*, pages 249–256, San Mateo, CA, 1992. Morgan Kaufmann.

[16] J. K. Martin. Evaluating and comparing classifiers: Complexity measures. In *Proceedings Fifth International Workshop on Artificial Intelligence and Statistics*, pages 372–378, Fort Lauderdale, FL, 1995.

[17] J. K. Martin. An exact probability metric for decision tree splitting and stopping. Technical Report 95-16, University of California, Irvine, Irvine, CA, 1995.

[18] J. K. Martin and D. S. Hirschberg. Small sample statistics for classification error rates. Technical Report 95-1, University of California, Irvine, Irvine, CA, 1995.

[19] J. Mingers. An empirical comparison of pruning measures for decision tree induction. *Machine Learning*, 4:227–243, 1989.

[20] J. Mingers. An empirical comparison of selection measures for decision tree induction. *Machine Learning*, 3:319–342, 1989.

[21] S. Muggleton, A. Srinivasan, and M. Bain. Compression, significance and accuracy. In *Proceedings of the 9th International Workshop on Machine Learning (ML-92)*, pages 338–347, San Mateo, CA, 1992. Morgan Kaufmann.

[22] P. M. Murphy and M. J. Pazzani. ID2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In L. A. Bernbaum and G. C. Collins, editors, *Machine Learning: Proceedings of the 8th International Workshop (ML91)*, pages 183–187, San Mateo, CA, 1991. Morgan Kaufmann.

[23] T. Niblett. Constructing decision trees in noisy domains. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning: Proceedings of the European Working Session on Learning (EWSL-87)*. Sigma Press, Wilmslow, 1987.

[24] T. Niblett and I. Bratko. Learning decision rules in noisy domains. In *Proceedings of Expert Systems 86*, Cambridge, 1986. Cambridge U. Press.

[25] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[26] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[27] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[28] S. Rasmussen. *An Introduction to Statistics with Data Analysis.* Brooks/Cole Publishing Co., Pacific Grove, CA, 1992.

[29] C. Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.

[30] S. M. Weiss and N. Indurkhya. Reduced complexity rule induction. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 678–684, San Mateo, CA, 1991. Morgan Kaufmann.