On the Complexity of Learning Decision Trees

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

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 variety of potential trees to be explored. Other factors, *e.g.,* 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 produce shallower and more balanced trees. 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 tests are biased towards unbalanced splits, and neither is admissible 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 trees resulting from other heuristics and more complex post-processing.

Introduction

This paper studies the complexity of Top-Down Induction of Decision Trees — the TDIDT family of algorithms typified by ID3 (Quinlan 1986) and C4.5 (Quinlan 1993). The input for these algorithms is a set of 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 candidates, and then recursively split each subset until a subset consists only of one class or until the candidates are exhausted. Early algorithms included other stopping criteria, stopping when the improvement achieved by the best candidate was judged to be insignificant. Later algorithms dropped these stopping criteria and added post-processing procedures to prune a tree or to replace subtrees.

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

Figure 1: TDIDT Tree-Building

```
BuildTree(A, V, data)
    if (AllSameClass(data)) then
        MakeLeaf(ClassOf(data))
    else
        A ← |A|
        N ← |data|
        if ((N > 0) and (A > 0)) then
          Initialize(b, best)
          for a = 1 ... A
              sub ← Partition(data,a,V_a)
              E ← Heuristic(sub)
              if (E < best) then
                  best ← E
                  b ← a
          if (not Significance(best)) then
              MakeLeaf(LargestClass(data))
          else
              sub ← Partition(data,b,V_b)
              V ← |V_b|
              for v = 1 ... V
                  if (A−{b} = ∅) then
                      MakeLeaf(LargestClass(sub_v))
                  else
                      BuildTree(A−{b},V−V_b,sub_v)
```

Analysis of TDIDT Tree-Building

Figure 1 summarizes the tree-building phase, where the set of candidates has been defined off-line and is summarized by the input parameters **A** (a vector of partition labels) and **V** (a matrix of values for splitting on each attribute). Some algorithms re-evaluate the **V** matrix on every call to this procedure. Other major differences between algorithms lie in the Heuristic function, in the way numeric attributes are handled by the

Partition function, and in whether stopping based on a Significance function is performed.

Typically, the run times of the Heuristic and Partition functions are linear ($\theta(N)$), compiling a contingency matrix and computing a function such as information gain. Then, the run time T_B of BuildTree is

$$T_B(\mathbf{A},\mathbf{V},\mathbf{data}) = \theta(AN) + \sum_v T_B(\mathbf{A}-\{b\}, \mathbf{V}-\mathbf{V}_b, \mathbf{sub}_v)$$

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 $\mathbf{sub}_v$ 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. If we simply create an equivalent set of $V_i - 1$ binary attributes for each candidate (where V_i is the i^{th} candidate's arity), the time is $O(d^2N)$, where d is the dimensionality of the data, $d = \sum(V_i - 1)$. For real-valued attributes, V_i is $O(N)$ for each attribute, and these methods could cause the behavior to be $O(A^2N^3)$.

Analysis of real-world data sets is more complicated because the splitting process for any path may terminate before all attributes have been utilized, either because all items reaching the node have the same class or because of some stopping criterion. Thus, the tree height may be less than the number of candidates A, and the leaves may lie at different depths. Then, the time complexity is related to the average height (h) of the tree (weighted by the number of items reaching each leaf).

Looking at Figure 1 in detail, and assuming that all candidates have the same arity V, the time complexity can be modeled as

$$\begin{aligned} T_B(A,V,N) = {} & K_0 + K_1N + K_3V + \textstyle\sum_v T_B(A-1,V,m_v) \\ & + A\,[K_2 + (\mathcal{E}_0 + \mathcal{E}_1N + \mathcal{E}_2C + \mathcal{E}_3V + \mathcal{E}_4CV)] \end{aligned}$$

where C is the number of classes, K_0, K_2, & K_3 small overhead constants, K_1 the incremental cost of the Partition function, and $\mathcal{E}_i$ the coefficients of the Heuristic function ($\mathcal{E}_0$ is a small overhead term, and $\mathcal{E}_1N$ typically dominates the other terms). Assuming that all leaves lie at the same depth $(A-1)$, this leads to

$$\begin{aligned} T_B \approx {} & (K_0 + K_3V)\ f_1(A,V) \\ & + (K_2 + \mathcal{E}_0 + \mathcal{E}_3V + \mathcal{E}_4CV)\ f_2(A,V) \\ & + (\mathcal{E}_1N + \mathcal{E}_2C)\ A(A+1)/2\ +\ K_1AN \end{aligned}$$

where $f_1(A,V) = (V^A - 1)/(V-1)$ and $f_2(A,V) = (\,f_1(A+1,V) - (A+1)\,)\,/(V-1)$.

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 well-fit by the following pro-rated model:

$$\begin{aligned} T_B \approx {} & [\,(\mathcal{E}_1N + \mathcal{E}_2C)A(A+1)/2 \\ & + K_1AN + K_0 f_1(\mathcal{V},h)\,]\ h/(A-1) \\ & + (\mathcal{E}_1N + \mathcal{E}_2C)A + K_0 + K_1N \end{aligned}$$

Figure 2: Pessimistic Pruning

```
PostProc(dtree, data)
  AsIs, Pruned, Surgery, and Q
      are defined as in Eval(dtree, data)
  if (AsIs ≤ min{Pruned,Surgery})
      return AsIs
  else
      if (Pruned ≤ Surgery) then
          dtree ← MakeLeaf(data)
          return Pruned
      else     /* surgery performed here */
          dtree ← Q
          Surgery ← PostProc(Q, data)
          return Surgery
```

```
Eval(dtree, data)
  N ←|data|
  Q ← nil
  if dtree is a leaf then
      P ← PredictedClass(dtree)
      E ← (N − |P|)
      Pruned = Surgery = AsIs ← f(E, N)
      return AsIs
  else
      F ← SplitInfo(Root(dtree))
      Ldata, Rdata ← Partition(data, F)
      L ←|Ldata| /N
      R ←|Rdata| /N
      AsIs ←    L × Eval(Ltree, Ldata)
            +   R × Eval(Rtree, Rdata)
      P ← LargestClass(data)
      E ← (N − |P|)
      Pruned ← f(E, N)
      if (L > R) then Q ← Ltree
      else Q ← Rtree
      Surgery ← Eval(Q, data)
      return min(AsIs, Pruned, Surgery)
```

where $\mathcal{V} = d/A$ is the average branching factor.

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 $K_0\ (\mathcal{V}^h - 1)/(\mathcal{V}-1) \ll \mathcal{E}_1NA^2$, so that $T_B = \theta(A^2N)$ in most cases. Applications do exist, however, where the exponential growth of this term cannot be ignored. For our longest run-time (about 1.5 hrs), $A = 100$ and a binary tree with average depth $h = 30$ was built — the factor of 2^{30} for the K_0 term is significant here. In the worst case $h = A-1$ and $T_B = O(\mathcal{V}^A)$.

Analysis of Post-Processing

Figure 2 summarizes a typical post-processing routine, C4.5's (Quinlan 1993) pessimistic pruning, for binary trees. Examination of Figure 2 reveals that the dominant factors are the height of the 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 post-processing and evaluation as $T_P(H,N)$ and $T_E(H,N)$, respec-

tively. In all cases, $0 \leq H < N$ and the recursion ends with $T_P(0,N) = T_E(0,N) = \theta(N)$. When tree surgery is not actually performed, but merely evaluated, $T_P(H,N) = T_E(H,N)$.

When the surgery is performed, then $T_P(H,N) = T_E(H,N) + T_P(q,N)$ where q is the height of the child covering the most items (the larger child). In the worst case q is $H-1$, and so

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

If m is the size of the larger child and r the height of the smaller child, then either $q = H-1$ or $r = H-1$, and

$$T_E(H,N) = \theta(N) + T_E(q,m) + T_E(r,N-m) + T_E(q,N) \quad (1)$$

We prove that $T_E(H,N)$ and $T_P(H,N)$ are $\theta(N)$ in the best case and that their worst case complexity is $\theta(N\ 2^H)$. Thus, both $T_E()$ and $T_P()$ have tight bounds of $\Omega(N)$ and $O(N\ 2^H)$.

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 contain instances from more than one class is given by $P = 1-(1-I)^n$, and is an increasing function of the subset size n. For such an impure branch, additional splits will be needed to separate the classes. Thus, there is a tendency for larger subsets to have deeper subtrees.

If H_L and H_R are respectively the left and right subtree heights, and n the left subset size, 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 $\text{Prob}(H_L = H-1) = p(x)$, where $x = n/N$, then the approximate expected value, $t(H,N)$, of $T_E(H,N)$ is

$$t(H,N) \ \geq \ \theta(N) \ + \ 0.5\ t(H-1,N) + p(x)\ t(H-1,Nx) + (1-p(x))\ t(H-1,N(1-x)) \quad (2)$$

If we assume that $p(x) = x$, and that x is constant throughout the tree, we can solve Equation 2 by induction on H, obtaining $t(H,N) \geq \theta\left(N\ (1+z)^H\right)$, where $z = 2(x-0.5)^2$.

Obtaining a solution is more complex when the weight balance x is not the same for every split, but the solution has a similar 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 and increases with the variance of x. This expectation is borne out by simulation results. $t(H,N)$ is also exponential in H for other forms for $p(x)$, as shown by simulation of a sigmoid $p(x)$ function, which is more plausible than the linear form because of certain boundary constraints for both very large and very small x.

The recurrence (see Equation 2) on which our $\theta\left(N\ (1+z)^H\right)$ result is 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. The contribution of the omitted subtrees increases as their height or weight increases, and we expect that the incremental run times would be more nearly correlated with a weighted average depth than with the maximum depth of the tree. This expectation is confirmed by simulation outcomes.

Based on these results, we expect run times to be proportional to $N\ (1+\xi)^{h^*}$, where h^* is an average height (not necessarily the weight average, h). A very good fit to empirical data spanning 5 decades of run time is obtained using $h^* = \sqrt{h\ h'}$; where h and h' are, respectively, the weight-average heights before and after post-processing.

Discussion

Both tree-building and post-processing have components that are exponential in tree height, which is bound above by the candidate set's dimensionality. While the dominant goal is to maximize accuracy, we must remember that TDIDT inherently compromises by using greedy, heuristic search and limiting the candidates. A premium is placed on methods which minimize dimensionality without sacrificing accuracy. In particular, approaches which replace a V-ary split with binary splits should be avoided, as this leads to deeper trees and to re-defining the candidates at every node, a considerable computational expense.

We have confirmed these observations experimentally using 16 data sets from various application domains. Ten of these data sets involved continuous attributes which were converted to nominal attributes in two ways: using arbitrary cut-points at approximately the quartile values, and choosing cut-points at local minima of smoothed histograms (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 attribute for each attribute-value pair. (See (Martin 1995) for details of these experiments).

Multi-way trees consistently have more leaves, are shallower, and are learned more quickly than their binary counterparts. On the few occasions when there is a notable difference in accuracy, the binary trees appear to be more accurate. We show that remapping 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 sometimes improves accuracy. However, a large penalty is paid in terms of increased run time and tree complexity.

In the quartiles method, the most unbalanced split is approximately 75/25. The 'natural' cut-points, by contrast, tend to produce very unbalanced splits. The differences in accuracy 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 quickly learned).

Post-processing does not improve accuracy, and only occasionally does it significantly reduce the number of leaves or the average height. 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 very unbalanced.

The effects of using different heuristics are illustrated using three functions (information gain, orthogonality, and the hypergeometric). There are no significant differences in accuracy between heuristics, and the inferred trees all have about the same number of leaves. The tree height and learning time, however, do vary significantly and systematically. For the worst case data, learning times differ by a factor of 12. The hypergeometric is better justified on statistical grounds than the other heuristic functions, and it also gives better results empirically.

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). The effect of surgery can be achieved by simple pruning or stopping performed on another tree in which the order of the splits is different. This point is very important, since it is the tree surgery which drives the exponential growth of run time *vs.* tree depth — 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. Since the choice of heuristic is largely a matter of complexity, not of accuracy, 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. In particular, if the split selection heuristic (*e.g.,* information gain) and the stopping criterion (*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. The obvious solution to this dilemma is to use the same evaluation function for ranking splits as for deciding whether to prune/stop. We show that neither information gain nor χ^2 is suitable for this dual purpose.

χ^2 and gain approximate the logarithm of the hypergeometric function, the exact likelihood that the observed degree of association between subset membership and class is coincidental. We compare the results of stopping using the hypergeometric to unstopped and post-processed trees. Stopping in this way is not detrimental to accuracy. 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).

Conclusions

1. Insofar as accuracy is concerned, the important design decisions are those which expand the candidate set. Other factors (*e.g.,* the heuristic and stopping/pruning) generally have little impact on accuracy.
2. For single-attribute, multi-way splits on A discrete variables, the time to build a tree for N items is $O(A^2N)$. If all V-ary splits are binarized, this becomes $O(d^2N)$ where d is the dimensionality. For continuous attributes, the tree building time may be $\theta(A^2N^3)$. Thus, there is potentially a large payoff for pre-processing to reduce dimensionality.
3. Both tree building and post-processing have components which increase exponentially in tree height. This puts a great premium on design decisions which tend to produce shallower trees.
4. Tree surgery is equivalent to simply pruning an alternative tree in which the root split of the current tree is made last, rather than first. The need for such surgery arises from using biased heuristics and different criteria for selection than for pruning.
5. χ^2 and information gain are biased and inadmissable, and should not be used. The hypergeometric is admissible, and it allows efficient pruning or stopping. It builds simpler and shallower trees which are no less accurate than those built using other heuristics. Learning time can be reduced as much as an order of magnitude by using the hypergeometric.

References

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

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

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