MDL-based Decision Tree Pruning

Manish Mehta **Jorma Rissanen** **Rakesh Agrawal**

IBM Almaden Research Center
650, Harry Road, K55/801
San Jose, CA 95120-6099
{mmehta, rissanen, agrawal}@almaden.ibm.com

Abstract

This paper explores the application of the Minimum Description Length principle for pruning decision trees. We present a new algorithm that intuitively captures the primary goal of reducing the misclassification error. An experimental comparison is presented with three other pruning algorithms. The results show that the MDL pruning algorithm achieves good accuracy, small trees, and fast execution times.

Introduction

Construction or "induction" of decision trees from examples has been the subject of extensive research in the past [Breiman et. al. 84, Quinlan 86]. It is typically performed in two steps. First, training data is used to grow a decision tree. Then in the second step, called *pruning*, the tree is reduced to prevent "overfitting".

There are two broad classes of pruning algorithms. The first class includes algorithms like cost-complexity pruning [Breiman et. al., 84], that use a separate set of samples for pruning, distinct from the set used to grow the tree. In many cases, in particular, when the number of training instances is small, it is more desirable to use all the samples for both the tree building *and* its pruning. In the absence of separate pruning data, cross-validation is used by these algorithms. In addition to the ad hoc nature of cross-validation, this approach also suffers from the drawback that multiple candidate trees need to be generated, which can be computationally expensive.

The second class of decision-tree pruning algorithms, which includes pessimistic pruning [Quinlan 87], uses all of the training samples for tree generation and pruning. Although these algorithms are computationally inexpensive and do not require separate pruning data, experiments have shown that they typically lead to trees that are "too" large and sometimes higher error rates [Mingers 89].

This paper presents a novel decision-tree pruning algorithm based on the Minimum Description Length (MDL) principle. For earlier but different applications of the same principle, see [Rissanen and Wax 88], [Quinlan and Rivest 89], [Rissanen 89] and [Wallace and Patrick 93]. Our experiments show that the proposed algorithm leads to accurate trees for a wide range of datasets. This algorithm does not employ cross-validation or a separate data set for pruning. Therefore, the tree generation algorithm needs to produce only a single tree and the computational expense is reduced. Moreover, when compared to other algorithms such as pessimistic pruning that do not use cross-validation or a separate data set for pruning, the MDL-based pruning algorithm produces trees that are significantly smaller in size.

The rest of the paper is organized as follows. We first give a formal description of the problem. The MDL criteria used for pruning is presented in the next section followed by a description of the complete pruning algorithm. We next discuss the performance results and finally present our conclusions and suggestions for future work.

Problem Statement

The data for designing a decision tree, also called the "training" sample, consist of n pairs $(c_t, \underline{x}(t))$ for $t = 1, 2, \ldots, n$, where c_t are values of the *class* variable c belonging to the set $0, 1, \ldots, m-1$, and $\underline{x}(t) = x_1(t), \ldots, x_k(t)$ are values of k *feature* variables $\underline{x}_i$, also called *attributes*, and written collectively as $\underline{x}$. Some of the feature variables, called 'categorical', range over finite sets, say x_i over the set $\{1, 2, \ldots, r(i)\}$ while others range over the real line or some of its subset. The intent with the decision tree is to subject a future data item to tests, suitably specified by the features, and depending on the result make a prediction of the class value. Frequently, such tests are of the type: $x_i(t) \leq a_i$ for a real-valued feature, where a_i is a real number, truncated to finite precision, and $x_i(t) \in A_i$, for a categorical feature, where $A_i = \{a_{i_1}, \ldots, a_{i_{n(i)}}\}$, is a finite set. The numbers a_i, or the sets A_i, serve as 'thresholds', which is what we call them.

The MDL criterion seeks a model within a class which permits the shortest encoding of the class sequence c^n in the training sample, given the features. This means that we must select a class of models, which in case of decision trees can be quite an involved process. We define our model class as the set of all subtrees of the decision tree $\mathcal{T}$ induced from the training data. The tree $\mathcal{T}$ has at each of its nodes a feature taken from the sequence of $x^M = x_1, \ldots, x_M$ together with the number of outcomes and ranges of the associated thresholds. In a significant departure from previous applications of MDL to decision-tree pruning ([Quinlan and Rivest 89], [Wallace and Patrick 93]), we assume that the list of features and the ranges for the thresholds are provided as a preamble to the decoder. For the real-valued features the range is the real line, while for the categorical ones the range is specified as a list of subsets $\{A_{i1}, \ldots, A_{i,r(i)}\}$, of the possible values of the feature in question. Therefore, the only parameters for the models are the actual values of the thresholds used at each node and the class probabilities (under the independence assumption). The next section explains the coding mechanism in greater detail.

Prediction Error

In earlier applications of the MDL principle to the tree design the error criterion used has been the code length with which the string can be encoded [Rissanen 87, Quinlan and Rivest 89, Rissanen 89]. This length is either one resulting from predictive coding, or it includes the code length required to encode the various symbol occurrence counts. Such a code length may be written as (all logarithms are natural logarithms)

$$\ln \frac{t!}{t_0! \ldots t_{m-1}!} + \ln \binom{t+m-1}{m-1}, \qquad (1)$$

where n_i denotes the number of times symbol i occurs in c^t (see [Davison 73] for the binary case and [Rissanen 87] for the general case with coding-theoretic interpretations). The second term represents the code length needed to encode the symbol occurrence counts, and it may be viewed as the *model cost* of the model class involved. The code length (equation 1) has the defect that the model cost becomes of the same order of magnitude as the first term, the code length for the data, when some of the counts are either close to zero or close to t. A better formula is the following [Krichevsky and Trofimov 83],

$$L(c^t) = \sum_i t_i \ln \frac{t}{t_i} + \frac{m-1}{2} \ln \frac{t}{2} + \ln \frac{\pi^{m/2}}{,\,(m/2)} \qquad (2)$$

where t_i denote the number of times symbol i occurs in c^t. This code length, called the stochastic complexity, has distinguished optimality properties [Rissanen 94].

Although the design or just the pruning of decision trees can be based on the code length (equation 2) as the criterion, we prefer here another which better captures the intuitive goal, and use the code length (equation 2) for the tie breaks, only. Let $\delta(c_{t+1}, \hat{c}_{t+1})$ be an $m \times m$ matrix of positive elements, except for the diagonals which are zero, where $\hat{c}_{t+1}$ is the prediction of c_{t+1} as a function of the past string c^t. With such a matrix we can select freely the relative penalties incurred when the prediction $\hat{c}_{t+1}$ differs from c_{t+1}. How should we define the predictor? First, predict the very first element c_1 as the smallest symbol j for which $\sum_i \delta(i,j)$ is minimized. Then predict $\hat{c}_{t+1}$ as the symbol for which the sum is minimized:

$$\sum_i t_i \delta(i, \hat{c}_{t+1}) = \min_j \sum_i t_i \delta(i,j),$$

where t_i denotes the number of times symbol i occurs in c^t. In case several symbols achieve the same minimum, we break the tie by taking $\hat{c}_{t+1}$ as that symbol from among the minimizing ones for which (equation 2) is minimized. The so obtained accumulated 'honest' prediction errors, then, for the string c^t when the occurrence counts are not known until the entire string has been processed, are given by

$$S_0(c^t) = \sum_{i=1}^{t} \delta(c_i, \hat{c}_i). \qquad (3)$$

The criterion (equation 3) includes the 'model cost' due to our not knowing the occurrence counts in an implicit way. To see this, suppose the class string were a sample from an m-valued independent stationary random process and we knew the symbol probabilities p_i. Then the optimal predictor would result if we always predict the symbol j that minimizes the sum $\sum_i p_i \delta(i,j)$, say the symbol $\hat{j}$. The mean per symbol prediction error would then be

$$E = \sum_{i=0}^{m-1} p_i \delta(i, \hat{j}) \qquad (4)$$

For strings generated by such a process, the mean per symbol prediction error of (equation 3) is higher, $E + O(1/t)$, the excess being attributed to the fact that for the predictor in (equation 3) we must estimate the symbol, namely $\hat{j}$, with the smallest mean error (equation 4), and this adds to the cost (equation 3) until the estimation is done error free. Accordingly, we may interpret the excess prediction error as a 'model cost'. This cost is less than the one in (equation 2), $O((\ln n)/n)$ reflecting the fact that for optimal prediction only a part of the complete model needs to be estimated, namely, the symbol that has the smallest sum (equation 4), rather than the symbol probabilities themselves.

When predicting the symbols at the nodes of a decision tree there will be other components to be added to the model cost than just the symbol occurrence counts. Although in principle their effect could be included in the same 'honest' predictive error criterion, it would be impractical requiring a large amount of computations. A much simpler way is to calculate the various additional model costs separately, non-predictively, and add them to the prediction errors. This creates a difficulty, because such model costs, which will be of the nature of code lengths, cannot logically be added to anything but code lengths. We resolve the difficulty by defining a bona fide code length which is equivalent with (equation 3). Indeed, define a conditional probability measure as follows

$$P(c_{t+1}|c^t) = K(c^t)e^{-\delta(c_{t+1},\hat{c}_{t+1})} \qquad (5)$$

where the normalizing constant is seen to be

$$K(c^t) = 1/\sum_i e^{\delta(i,\widehat{t+1})} \geq K = \min_j 1/\sum_i e^{\delta(i,j)}.$$

Therefore, regardless of what the predicted symbol is encoding of the data sequence c^t can be done with the ideal code length

$$L_S(c^t) = S_0(c^t) - t \ln K. \qquad (6)$$

This differs from (equation 3) only by a term proportional to the length of the string, which will be seen to be irrelevant to the pruning of the tree. The word "ideal"' refers to the convenient practice of ignoring the integer length requirement for a code length.

It will now be easy to consider the missing code lengths to be added to the model cost. First of these is due to the code length needed to describe the threshold needed at each node in which a test is made. Consider first the case of a real-valued test ξ on the feature x, with thresholds $a_1, \ldots, a_{r-1}$. Truncating each to the precision e^{-q} the thresholds can be encoded with about $q(r-1)$ nats (natural logarithm bits). Ideally, the optimal precision should be determined independently for each test involving a real-valued attribute. However, for practical considerations, we use a constant precision value throughout the decision tree (see the section on Performance Evaluation). For a categorical test, whose single threshold ranges over a list $A_1, \ldots, A_p$ of subsets, as specified in the tree $\mathcal{T}$, the code length needed to specify the threshold is then $\ln p$. Write the required cost in each case (real-valued/categorical) as $L(thr)$. When a class sequence c^t is predicted after the test, the $r-1$ thresholds of the test ($r = 2$ for a categorical feature) partition the string c^t into r substrings, $c(j)$, for $j = 1, \ldots, r$, each of which is predicted with the rule as in (equation 5). The resulting error criterion is then given by

$$S_1(c^t) = \sum_{j=1}^{r} S_0(c(j)) + L(thr). \qquad (7)$$

Notice that if we had instead used the code length criterion (equation 6) in both cases the same constant $-t \ln K$ would have been added to the prediction error criteria, and the comparison of the two ways of doing the prediction and coding would have been the same.

There is one more component to the model cost to be considered, namely, the code length needed to describe the structure of the final subtree. The optimal pruning algorithm will have to distribute this cost among the nodes, which process is easiest to describe together with the algorithm in the next section.

Optimal Pruning Algorithm

In order to describe the optimal pruning algorithm we need a few preliminaries. First, let $P_T(1)$ denote the ratio of the number of internal nodes to the number of all nodes in the tree $\mathcal{T}$, and put $P_T(0) = 1 - P_T(1)$. In the special case where all the outcomes of the tests; i.e., the arities of the nodes in the tree, are equal, say r, the inverse of this ratio is given by $r(1+1/(M-1))$. Next, let s denote any node in the tree and write $c(s)$ for the class sequence that 'falls off' the node s; i.e., a subsequence of c^n whose test values coincide with the path from the root to the node s. Write $L(thr_s)$ for the code length needed to describe the threshold of the test at this node, in the notations of the preceding section either $\ln p$ or $(r-1)q$, depending on the type of the feature. Finally, in order to break the ties in the predictor we also need to collect the occurrence counts $n_i(s)$ of the symbols i in the string $c(s)$ and compute their sum $n(s) = \sum_i n_i(s)$.

The pruning algorithm consists of the steps:

- 1. Initialization. At the leaves s of $\mathcal{T}$ gather the counts $n_i(s)\, i = 0, 1, \ldots, m-1$, and compute
  $S(s) = -\ln P_T(0) + S_0(c(s))$
- 2. Recursively in bottom-up order, put
  $n_i(s) = \sum_j n_i(sj)$, $i = 0, 1, \ldots, m-1$, the sum over all children sj of s, and set

  $$S(s) = \min \begin{cases} -\ln P_T(0) + S_0(c(s)), \\ -\ln P_T(1) + L(thr_s) + \sum_j S(sj) \end{cases}$$

  If the first element is smaller than or equal to the second, delete all children.
- 3. Continue until the root λ is reached.

The value $S(\lambda)$ of the criterion for the classes in the training sample, obtained with the subtree $\mathcal{T}^*$, is the smallest obtainable with the subtrees of $\mathcal{T}$. This is seen to be true by the dynamic programming argument, based on the fact that every subtree of the optimal subtree is optimal. The algorithm generalizes another described in [Nohre 94] in a special case. In particular, the code length for the structure of the optimal subtree $\mathcal{T}^*$, defined by the increments $\ln P_T(0)$ and $\ln P_T(1)$, is given by

$$L(\mathcal{T}^*) = -n^*_{int} \ln P_T(1) - n^*_{leaf} \ln P_T(0), \qquad (8)$$

where n^*_{int} and n^*_{leaf} denote the number of internal nodes and leaves in T^*, respectively. We may regard this as an optimal code length of the structure, for (equation 8) defines a probability for each subtree $\mathcal{V}$ of $\mathcal{T}$ by,

$$P_T(\mathcal{V}) = \frac{e^{-L(\mathcal{V})}}{\sum_{\mathcal{S}} e^{-L(\mathcal{S})}}.$$

where the summation is over all possible subtrees of T. Clearly, no code exists which would have a shorter codeword for every subtree than $\ln P_T(\mathcal{V})$, and we may regard (equation) to define an optimal code for the subtrees. The code length (equation 8) differs from that of the optimal codeword only by a constant, which, however, is the same for all the subtrees.

Performance Evaluation

Experimental Setup

We present results on seven datasets used previously in the STATLOG project (see [Michie et. al. 94] for a full description). The datasets are available via anonymous FTP from ftp.strath.ac.uk in directory Stams/statlog. The experimental methodology for each dataset is the same as in the STATLOG project and is summarized in Table 1.

Dataset	**Domain**
Australian	10-fold Cross-Validation
DNA	Separate Training & Test Set
Diabetes	12-fold Cross-Validation
Letter	Separate Training & Test Set
Segment	10-fold Cross-Validation
Satimage	Separate Training & Test Set
Vehicle	9-fold Cross-Validation

Table 1: Experimental Setup

The initial decision trees were generated using the CART algorithm available in the IND software package [Buntine and Carauan 92]. We experimented with both the "twoing" and the "gini" index [Breiman et. al. 84] for growing the initial tree. The results were similar for both the cases and only the results with the twoing index are presented in this paper. The IND package also provides implementations of three pruning algorithms: the cost-complexity algorithm, the "pessimistic" pruning algorithm, and a modified version of the pessimistic algorithm used in C4.5. In order to avoid making an ad-hoc choice of the size of the data set to be set aside for pruning, 10-fold cross-validation was used for the cost-complexity pruning algorithm. We implemented the MDL-pruning algorithm and ran it on the tree generated by IND. All of the experiments were done on an IBM RS/6000 250 workstation running the AIX 3.2.5 operating system.

Performance Results

Choosing the Precision for MDL Pruning Recall that our MDL pruning algorithm uses a constant precision for all the tests in the decision tree based on continuous attributes. The first experiment determines the precision value that should be used with the MDL pruning algorithm. Table 2 shows the error rates achieved with the MDL pruning algorithm for precision values of 1, 2, 4, and 6.

Dataset	MDL Precision			
	1	**2**	**4**	**6**
Australian	15.3	15.5	15.9	15.8
Diabetes	24.1	25.3	23.2	24.1
DNA	8.1	8.1	8.1	8.1
Letter	15.8	17.4	20.0	22.3
Satimage	14.6	14.9	15.7	16.0
Segment	5.5	5.9	6.5	7.0
Vehicle	29.3	30.7	32.4	33.7

Table 2: Effect of Precision Value on MDL Pruning

The results show that the error rates increase as the precision value increases from 1 to 6. This is because a higher precision value implies that the tree is pruned more aggressively. Therefore, at higher precision values, the tree gets "over-pruned" leading to an increase in the error rates. The best performance is achieved using a precision value of 1. Similar, results were also obtained for a large number of other datasets examined by the authors. Therefore, our MDL pruning algorithm uses a precision value of 1 for all datasets.

Comparing Pruning Algorithms Three criteria are used to compare the pruning algorithms: the *error rate* obtained on the test set, the *size* of the pruned tree, and the *execution time* of the algorithm. Since cost-complexity pruning in IND is executed as part of the tree generation phase, all timing measurements include the time taken to generate *and* prune the decision tree.

Table 3 shows the error rates achieved by each of the algorithms. The results show that all the pruning algorithms lead to error rates which are not too different from each other. C4.5 and MDL pruning perform robustly for all the datasets; for each dataset, the error rates achieved by these algorithms are less than 1% greater than the best performing algorithm.

Next, we compare the sizes (in terms of number of nodes) of the pruned trees produced by each of the pruning algorithms. A smaller decision tree is desirable since it provides more compact class descriptions, unless the smaller tree size leads to a loss in accuracy. Table 4 shows the sizes for each of the datasets. The results show that the MDL pruning algorithm achieves trees that are significantly smaller than the trees generated by the Pessimistic and C4.5 pruning algorithms. However, the smallest trees are generated by the cost-complexity algorithm which prunes trees most aggressively. These results also show that the effect of the tree size on the error rate is domain-dependent. For example, while the large tree produced by the pessimistic algorithm produces the best error rate for the Letter dataset, it leads to the worst performance for the Diabetes and DNA datasets.

The final criterion for comparing the pruning algorithms is the execution times of the algorithms. The

Dataset	Cost-Complexity	C4.5	Pessimistic	MDL
Australian	14.9	15.5	15.8	15.3
Diabetes	25.4	24.7	27.0	24.1
DNA	8.6	8.5	9.1	8.1
Letter	15.7	15.7	15.1	15.8
Satimage	15.3	14.8	15.1	14.6
Segment	5.2	5.3	5.2	5.5
Vehicle	30.3	29.2	29.1	29.3

Table 3: Testing Error Rates

Dataset	Cost-Complexity	C4.5	Pessimistic	MDL
Australian	5.2	38.2	54.0	23.6
Diabetes	11.5	65.5	100.8	34.8
DNA	35.0	65.0	93.0	51.0
Letter	1199.5	1266.3	1459.0	1174.8
Satimage	90.0	244.8	319.0	167.0
Segment	52.0	71.0	77.0	56.2
Vehicle	50.1	101.2	124.5	72.1

Table 4: Pruned-Tree Size

Dataset	Cost-Complexity	C4.5	Pessimistic	MDL
Australian	3.2	0.1	0.1	0.1
Diabetes	4.67	0.1	0.2	0.2
DNA	38.82	4.0	4.0	4.0
Letter	243.9	24.9	24.7	24.6
Satimage	193.0	19.9	20.2	19.9
Segment	41.0	3.8	3.5	3.7
Vehicle	11.9	0.9	1.0	0.9

Table 5: Execution Times

results are collected in Table 5 and show that the cost-complexity algorithm, which uses cross-validation for pruning and grows multiple tree, has the largest execution time. The other three algorithms grow a single decision tree, and therefore are nearly an order of magnitude faster in comparison.

Results Summary The experimental results show that the cost-complexity pruning algorithm achieves good accuracy and small trees. However, the algorithm is nearly an order of magnitude slower than the other pruning algorithms. The Pessimistic and C4.5 pruning algorithms are accurate and have fast execution times, but lead to large decision trees. The MDL pruning algorithm, on the other hand, does not suffer from any of these drawbacks. MDL pruning produces error rates that are comparable or better than those achieved with the other pruning algorithms. It leads to decision trees that are significantly smaller than the ones achieved by C4.5 and the pessimistic algorithms. At the same time, the MDL pruning algorithm executes nearly an order of magnitude faster than the cost-complexity algorithm.

Conclusions and Future Work

We presented a novel algorithm that uses the Minimum Description Length (MDL) principle for pruning decision trees. Instead of minimizing the length of the class sequence in the training sample together with the length of the decision tree, as in previous applications of MDL to decision tree design, a new length criterion was introduced in this paper. This criterion captures well the intuitive goal of reducing the rate of misclassifications. Experimental comparison with other pruning algorithms showed that the proposed algorithm provides high accuracy, small decision trees, and fast execution times.

The MDL algorithm presented in this paper can be extended in several ways. Currently, the algorithm does not permit the removal of a subset of the children of a node. We are in the process of evaluating an extension of the algorithm that allows such "partial" pruning of children at the nodes. This will allow the MDL principle to obtain even smaller decision trees without losing accuracy. Another possible direction for future work is to experiment with different classes of models. This will allow the algorithm to be applied to cases where the model costs are much higher than those in the trees generated by algorithms like CART and C4.5 (e.g. [Bennett 93]).

References

Bennett, K.,"Machine Learning via Mathematical Programming", *PhD Thesis*, University of Wisconsin-Madison, 1993.

Breiman et. al., *Classification and Regression Trees*, Wadsworth International Group, Belmont, CA, 1984.

Buntine, W., and Carauan, R., "Introduction to IND Version 2.1", *User's Manual*, NASA Ames Research Center, 1992.

Catlett, J., "Megainduction: a Test Flight", *Proc. Eighth Intl. Workshop on Machine Learning*, 1991.

Davison, L. D., "Universal Noiseless Coding", *IEEE Trans. on Inform. Theory*, Vol IT-19, Nov, 1973.

Krichevsky, R.E. and Trofimov, V.K. (1983), "The Performance of Universal Coding", *IEEE Trans. on Information Theory*, Vol. IT-27, No. 2.

Michie et. al., *Machine Learning, Neural And Statistical Classification*, Ellis Horwood, 1994.

Mingers, J., "An Empirical Comparison of Pruning Methods for Decision Tree Induction", *Machine Learning*, 4, 1989, pp. 227-243.

Nohre, R., "Some Topics in Descriptive Complexity", *PhD Thesis*, Linkoping University, Linkoping, Sweden

Quinlan, J.R., "Induction of Decision Trees", *Machine Learning*, 1(1), 1986.

Quinlan, J.R., "Simplifying Decision Trees", *Intl. Jrnl. of Man-Machine Studies*, 27, 1987, pp. 221-234.

Quinlan, J.R. and Rivest, R.L. (1989), 'Inferring Decision Trees Using Minimum Description Length Principle', *Information and Computation*, 80.

Rissanen, J. and Wax, M. (1988), "Algorithm for Constructing Tree Structured Classifiers", *US Patent No. 4,719,571.*

Rissanen, J. (1989), *Stochastic Complexity in Statistical Inquiry*, World Scientific Publ. Co., Suite 1B, 1060 Main Street, River Edge, New Jersey, (175 pages)

Rissanen, J. (1994), "Fisher Information and Stochastic Complexity", *submitted to IEEE Trans. Information Theory.*

Wallace, C. S. and Patrick, J. D. (1993), "Coding Decision Trees", *Machine Learning*, 11, 1993.