

Predicate Selection for Structural Decision Trees

K.S. Ng and J.W. Lloyd

Computer Sciences Laboratory
Research School of Information Sciences and Engineering
The Australian National University
`{kee,jwl}@discus.rsise.anu.edu.au`

Abstract. We study predicate selection functions (also known as splitting rules) for structural decision trees and propose two improvements to existing schemes. The first is in classification learning, where we reconsider the use of accuracy as a predicate selection function and show that, on practical grounds, it is a better alternative to other commonly used functions. The second is in regression learning, where we consider the standard mean squared error measure and give a predicate pruning result for it.

1 Introduction

In this paper, we study predicate selection functions (also known as splitting rules in the literature) for structural decision trees and suggest two ways to improve existing schemes.

The first is in classification-tree learning, where we reconsider the use of accuracy as a predicate selection function and show that, on practical grounds, it is a better alternative to other commonly used functions in the context of structural trees, its primary advantage being the admission of a simple predicate pruning mechanism. With a small modification, we also show that two recognized problems associated with its use can be resolved easily. All these are discussed in Section 3.

The second, presented in Section 4, is in regression-tree learning. In that section, we consider the standard mean-squared error measure and give an efficiently computable predicate pruning result for it.

To avoid confusion, it's worth stressing that the two pruning mechanisms alluded to above happen *not* in the space of trees, but in the space of predicates we search to split a decision node. This form of pruning, called predicate pruning in this paper, is largely a search efficiency issue; in contrast, tree pruning deals with the more difficult problem of handling overfitting. This point should become clearer in Section 2.

We next give a high-level specification of a family of structural-tree induction systems. Every learner that fits the description can potentially benefit from the proposals of this paper.

2 Induction of Structural Trees

We consider the family of decision-tree learners for *structured data* that uses (variants of) the top-down induction algorithm for learning. A high-level schema of such systems is shown in Figure 1.

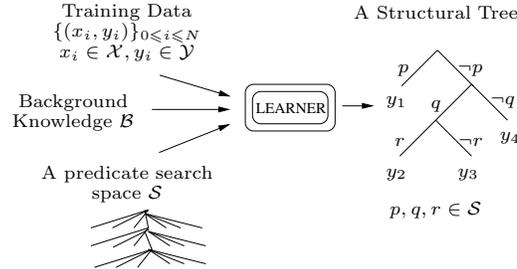


Fig. 1. Structural-tree learning systems

A learning algorithm of this kind takes three inputs:

1. a (finite) set of training examples $z \subseteq \mathcal{X} \times \mathcal{Y}$;
2. some background knowledge \mathcal{B} ; and
3. a collection \mathcal{S} of predicates over \mathcal{X} ,

and produces as output a structural decision tree T with node functions in \mathcal{S} . The terminal nodes of T are labelled with elements from \mathcal{Y} . For classification, \mathcal{Y} is a small finite set; for regression, \mathcal{Y} is (usually a bounded interval of) \mathbb{R} . The input space \mathcal{X} can be any arbitrary set. We assume \mathcal{S} has the following structure:

1. \mathcal{S} is a directed acyclic graph where each vertex is a predicate over \mathcal{X} and there is an edge from a predicate p to a predicate q iff q can be obtained (in some way depending on the setting) from p .
2. Suppose p and q are both predicates in \mathcal{S} . If q is a descendant of p , then $\forall x \in \mathcal{X}, q(x) \implies p(x)$.

We permit the predicate search space to change from node to node in the decision tree.

The top-down induction algorithm makes binary splits at each node in the tree. If \mathcal{E} is the set of examples at the current node, then a predicate $p \in \mathcal{S}$ induces a partition $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$ of \mathcal{E} , where $\mathcal{E}_1 \subseteq \mathcal{E}$ is the set of examples that satisfy p and $\mathcal{E}_2 \subseteq \mathcal{E}$ is the set of examples that do not satisfy p . The quality of the partition \mathcal{P} is determined by a real-valued predicate selection function $f(\mathcal{P})$. Given \mathcal{S} and a set \mathcal{E} of examples, we seek a predicate $p^* \in \mathcal{S}$ such that $f(\mathcal{P}^*)$, where \mathcal{P}^* is the partition of \mathcal{E} induced by p^* , is optimized.

The search space \mathcal{S} can be huge, in which case some form of pruning is needed to explore the space efficiently. It is the form of the predicate selection function, together with the two properties of \mathcal{S} stated earlier, that we will exploit in the design of our predicate pruning mechanisms.

Three learning systems that fit the description given here are Tilde [3], [2], S-CART [9], [10] and Alkemy [4], [11].

The underlying language for Tilde and S-CART is a subset of first-order logic. For both systems, \mathcal{X} are Prolog programs, \mathcal{B} is a Prolog program, and $\mathcal{S} = \{f_c \mid c \in \mathcal{C}\}$ where \mathcal{C} is a set of program clauses. Given $x \in \mathcal{X}$, $f_c(x)$ evaluates to true iff the query $\leftarrow c$ succeeds in the program $x \wedge \mathcal{B}$. In trees induced by both Tilde and S-CART, variables are shared across decision nodes down true branches. For that reason, the predicate search space can be different from node to node.

For both systems, if we only consider the addition of a single literal at each node t in the process of growing a tree, the predicate search space \mathcal{S}_t at t would be a one-level-deep tree that will not actually benefit from the proposals of this paper. In fact, existing predicate selection functions work rather well in this setting. However, if conjunctions of literals are considered, as can be done using the *lookahead* mechanism in Tilde and schemata declarations in S-CART (see, for details, [10, §6.4] and [2, §6.3]), then the pruning results presented here can exploit the richer structure of \mathcal{S}_t to alleviate the usual computational problems associated with the use of such rich search spaces, thus solving an important, thorny problem for both learners.

The underlying language of Alkemy is a typed higher-order logic. In this case, \mathcal{X} are basic terms of a certain type, \mathcal{B} takes the form of transformations (and their definitions), and \mathcal{S} consists of standard predicates over \mathcal{X} defined using a predicate rewrite system. The predicate search space is the same for every node in the tree. See, for more details, [11].

We move on now to the discussion of a new accuracy-based predicate selection function for classification trees.

3 An Accuracy Heuristic for Classification-Tree Learning

The use of accuracy as a heuristic was called into question early on in classification-tree research. Two standard criticisms, stated in [6, §4.1], are as follows.

Criticism 1. *The use of accuracy can result in premature termination of tree growth. This is because tree nodes that are relatively pure, with examples coming from one predominant class, often cannot be split with a strict increase in accuracy.*

Criticism 2. *The accuracy heuristic does not take future growth into account in choosing the current best split. We use the example in [6, §4.1] to illustrate this point. Consider a set $\mathcal{E} = (400, 400)$ of 800 examples. (Here and in the following, (n_1, \dots, n_c) denotes a set of examples with n_i examples in the i th class.) Which*

of the following two is the better partition of \mathcal{E} ?

$$\mathcal{P}_1 = ((300, 100), (100, 300)); \text{ or}$$

$$\mathcal{P}_2 = ((200, 400), (200, 0)).$$

\mathcal{P}_2 is intuitively the more appealing partition of the two, with potential to result in a smaller overall tree. But accuracy can't differentiate between them.

Criticisms 1 and 2 notwithstanding, in the context of structural decision trees, as pointed out in [4], the use of accuracy offers one important advantage that warrants reconsideration: its use admits a particularly simple predicate pruning mechanism. For completeness, we now review this result from [4].

Suppose there are c classes in all. Let \mathcal{E} be a (non-empty) set of examples, N the number of examples in \mathcal{E} , n_i the number of examples in \mathcal{E} in the i th class, and $p_i = n_i/N$, for $i = 1, \dots, c$.

Definition 3. The majority class of \mathcal{E} is defined to be the class to which the greatest number of examples in \mathcal{E} belong. (Ties are broken arbitrarily.)

Definition 4. The accuracy, $A_{\mathcal{E}}$, of a set \mathcal{E} of examples is defined by

$$A_{\mathcal{E}} = p_M,$$

where M is the index of the majority class of \mathcal{E} .

The accuracy is the fraction of examples which are correctly classified on the basis that the majority class gives the classification.

Definition 5. Let $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$ be a partition of a set \mathcal{E} of examples. We define the accuracy, $A_{\mathcal{P}}$, of the partition \mathcal{P} by

$$A_{\mathcal{P}} = \frac{|\mathcal{E}_1|}{|\mathcal{E}|} A_{\mathcal{E}_1} + \frac{|\mathcal{E}_2|}{|\mathcal{E}|} A_{\mathcal{E}_2}.$$

In the predicate search space, if p' is a descendant of p , then p' implies p and the partition $(\mathcal{E}'_1, \mathcal{E}'_2)$ of \mathcal{E} induced by p' has the property that $\mathcal{E}'_1 \subseteq \mathcal{E}_1$, where $(\mathcal{E}_1, \mathcal{E}_2)$ is the partition of \mathcal{E} induced by p . These considerations lead to the following definition.

Definition 6. Let \mathcal{E} be a set of examples and $(\mathcal{E}_1, \mathcal{E}_2)$ a partition of \mathcal{E} . We say a partition $(\mathcal{E}'_1, \mathcal{E}'_2)$ of \mathcal{E} is a refinement of $(\mathcal{E}_1, \mathcal{E}_2)$ if $\mathcal{E}'_1 \subseteq \mathcal{E}_1$.

We now introduce the important measure of classification refinement bound.

Definition 7. Let $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$ be a partition of a set \mathcal{E} of N examples, where n_i is the number of examples in \mathcal{E} in the i th class and $n_{j,i}$ is the number of examples in \mathcal{E}_j in the i th class, for $j = 1, 2$ and $i = 1, \dots, c$. We define the classification refinement bound, $B_{\mathcal{P}}$, of the partition \mathcal{P} by

$$B_{\mathcal{P}} = \frac{1}{N} (\max_i \{n_i + \max_{k \neq i} n_{1,k}\}).$$

The intuitive idea behind the definition of $B_{\mathcal{P}}$ is that the refinement of \mathcal{P} having the greatest accuracy can be obtained by moving all examples in one class from \mathcal{E}_1 across to \mathcal{E}_2 . Here is an example to illustrate the concept of classification refinement bound.

Example 8. Let $\mathcal{E} = (6, 9, 3, 2)$ and suppose $\mathcal{P} = ((2, 1, 0, 2), (4, 8, 3, 0))$. Then $A_{\mathcal{P}} = 10/20$ and $B_{\mathcal{P}} = 11/20$. If $\mathcal{Q} = ((0, 9, 0, 0), (6, 0, 3, 2))$, then $A_{\mathcal{Q}} = B_{\mathcal{Q}} = 15/20$. ◀

The next result shows that $B_{\mathcal{P}}$ is an upper bound for the accuracy of any refinement of a partition \mathcal{P} of a set of examples.

Proposition 9 ([11]). *Let \mathcal{E} be a set of examples and \mathcal{P} a partition of \mathcal{E} . If \mathcal{P}' is a refinement of \mathcal{P} , then $A_{\mathcal{P}'} \leq B_{\mathcal{P}}$. In particular, $A_{\mathcal{P}} \leq B_{\mathcal{P}}$.*

Proposition 9 can be used to prune the search space when searching for a predicate to split a node. During this search, we record the best partition \mathcal{P} found so far and its associated accuracy $A_{\mathcal{P}}$. When investigating a new partition \mathcal{Q} , the quantity $B_{\mathcal{Q}}$ is calculated. According to the proposition, if $B_{\mathcal{Q}} < A_{\mathcal{P}}$, then the partition \mathcal{Q} and all its refinements can be safely pruned. Here is an example to illustrate how this works.

Example 10. Let the set of examples be $(6, 9, 3, 2)$. Suppose the best partition found so far is $\mathcal{P} = ((6, 3, 0, 2), (0, 6, 3, 0))$, which has accuracy $12/20$. Suppose that later on in the search the partition $\mathcal{Q} = ((2, 4, 0, 1), (4, 5, 3, 1))$ is being investigated. Note that $B_{\mathcal{Q}} = 11/20$. Since $B_{\mathcal{Q}} < A_{\mathcal{P}}$, Proposition 9 shows that \mathcal{Q} and its refinements can be pruned.

On the other hand, consider the partition $\mathcal{R} = ((6, 5, 3, 2), (0, 4, 0, 0))$, for which $B_{\mathcal{R}} = 15/20$. Thus \mathcal{R} has refinements, which could be found by the system, whose accuracies exceed that of \mathcal{P} . Thus \mathcal{R} should not be pruned. ◀

The pruning mechanism just described has been shown to work well in many different applications. Table 1 gives an indication of its effectiveness. In it, we list six problems taken from [11, §6.2] and [5]. For each problem, we give (1) the size of the predicate search space $|\mathcal{S}|$ as defined in [11] and [5]; and (2) the number of predicates actually tested in a complete search of the predicate space aided by pruning. The percentage of the predicate space actually searched is also given. As shown, significant reduction in search can be achieved. It is worth pointing out that the effectiveness of the pruning mechanism is not a function of the size of the search space but a function of the structure of the search space and the way the training examples are actually labelled. In general, predicate search spaces that are formed by combining basic conditions in different ways, for example using conjunctions, stand to gain the most from the pruning mechanism. The predicate spaces defined for the last three datasets in Table 1 are of this kind.

It is not clear to the authors whether more commonly used functions like entropy admit similar (efficiently computable) pruning mechanisms. After several failed attempts to find such a result, it seems unlikely to us that there is one. Assuming there is no such result, then in the context of structural decision trees

Table 1. Efficiency of the predicate pruning mechanism

Dataset	$ \mathcal{S} $	Searched
Int Trees	396	120 (30.30%)
Mutagenesis	535	108 (20.19%)
East West	2073	892 (43.03%)
Headline	2850	190 (6.66%)
Protein	9262	581 (6.27%)
Musk-1	1,679,615,641	2,914,727 (0.17%)

we can expect accuracy to be a better predicate selection function compared to other commonly used functions. To back up this claim, we examine two cases under the assumption that there is enough structure in the predicate space for predicate pruning to take effect. The first of these corresponds to the case when an exhaustive search of the predicate space is computationally feasible, and the second when it is not.

In the first case, *with the aid of pruning*, computing the most accurate predicate can be expected to be cheaper than computing the one with, say, the lowest entropy. Assuming accuracy and entropy both yield reasonably accurate decision trees and that their relative performances are not too far apart – we will come back to look at this shortly – then adopting accuracy as the predicate selection function is a good strategy, especially if time efficiency is an issue.

In the second case, we must resort to incomplete searches. Given the same amount of time, we can expect the predicate chosen using, say, entropy in the absence of a pruning mechanism to be, in all likelihood, worse than the predicate chosen using accuracy in the presence of pruning simply because a smaller percentage of the predicate space is actually searched.

To advocate the use of accuracy as a viable predicate selection function, we need to address the two criticisms stated in the beginning of this section. Both criticisms are valid arguments. Criticism 1 has a theoretical basis, as shown in [8]. Criticism 2, being an intuitive argument, is weaker but persuasive nonetheless. Interestingly, both problems can be addressed with an easy solution: one can adopt accuracy as the main predicate selection function and use a concave function like entropy to break ties between equally-accurate predicates. We will call this the Acc^* function.

This scheme addresses Criticism 1 because in the case where no predicate in the search space can achieve a strict increase in accuracy, splits can still be made in accordance with the tie-breaker function, which we know behaves well.

Criticism 2 can also be resolved this way. In the example given, Acc^* will pick \mathcal{P}_2 over \mathcal{P}_1 , as desired.

One final question remains: Is a concave function like entropy, as a predicate selection function, always going to outperform Acc^* ? We investigate this empirically. Two tree-growing algorithms are compared, the first uses Acc^* , and the second, entropy. Twelve datasets were used for this purpose. They consists of the first five datasets in Table 1 and seven other datasets chosen randomly from the

UCI repository. (Musk-1 in Table 1 was excluded because an exhaustive search through the predicate space as required by the entropy-based algorithm is simply not feasible. It can, however, be handled using Acc^* .) The Alkemy learning system was used to perform the experiment. The results are shown in Tables 2 and 3. Table 2 gives the accuracies of the induced classifiers in the absence of tree post-pruning. The effects of post-pruning are shown in Table 3. In both tables, a \checkmark is shown if the entropy-based algorithm is significantly more accurate; a \times is shown if the Acc^* -based algorithm is better. Tree post-pruning is done using the cost-complexity pruning method of CART [6]. The accuracies reported are estimated using 10-fold cross validations.

Table 2. Accuracy vs Entropy (w/o tree post-pruning)

Dataset	Acc^*	Accuracy		Time	
		Ent	$Ent > Acc^*$	Acc^*	Ent
Int Trees	0.400	0.400		0.26	0.82
Mutagenesis	0.820	0.840		186.370	232.63
East West	0.900	0.900		0.68	1.67
Headline	0.950	0.950		0.97	4.86
Protein	0.800	0.800		0.98	20.60
Audiology	0.735	0.765	\checkmark	–	–
Lenses	0.833	0.833		–	–
Mushroom	1.000	1.000		–	–
Votes	0.947	0.942		–	–
Monks-1	0.894	0.886		–	–
Monks-2	0.692	0.692		–	–
Monks-3	0.884	0.876		–	–

Table 3. Accuracy vs Entropy (with tree post-pruning)

Dataset	Acc^*	Accuracy	
		Ent	$Ent > Acc^*$
Int Trees	0.700	0.700	
Mutagenesis	0.820	0.809	
East West	0.800	0.800	
Headline	0.900	0.900	
Protein	0.700	0.700	
Audiology	0.710	0.755	\checkmark
Lenses	0.850	0.850	
Mushroom	0.999	0.999	
Votes	0.959	0.956	
Monks-1	0.920	0.887	\times
Monks-2	0.634	0.638	
Monks-3	0.935	0.935	

It seems safe to conclude that the performance of Acc^* is comparable to most other predicate selection functions for the following reasons. The experiment above certainly suggests that Acc^* is comparable to entropy. We know from experience that entropy and the Gini index have similar behaviour. To top it off, it is shown in [12] and [7] that the Gini index is as good as any other known predicate selection function for the purpose of tree induction. In fact, the general agreement from [12] and [7] is that the exact predicate selection function used doesn't really matter all that much as long as the tree is allowed to grow sufficiently large, in which case tree post-pruning holds the key to the final performance. In that sense, the main problem with (plain) accuracy is that it results in premature termination of tree growth. With Acc^* , this is no longer an issue.

In a separate experiment, we also pitted Acc^* against (plain) accuracy on the twelve datasets. The experiment shows that Acc^* performs at least as well as, and usually better than, accuracy in all except one dataset (Audiology) after tree post-pruning.

Table 2 also records the time (measured in seconds) taken by the two algorithms on the first five datasets on a 1.2 GHz iBook. The results clearly show that non-trivial savings in learning time can be achieved. (The usual tests considered by C4.5 [14] are used for the UCI datasets. Such predicate spaces have no structure that can be exploited by the pruning mechanism, and naturally no reduction in computation time can be obtained – both algorithms need to evaluate every predicate in the search space.) When there is enough structure in the classification problem for predicate pruning to take effect, the only situation where the Acc^* -based algorithm would take longer time than the entropy-based algorithm is when a *significantly* larger tree is grown using the Acc^* -based algorithm. This scenario can probably occur, but it would be very rare.

In summary, we have proposed and shown the viability of a new accuracy-based predicate selection function for top-down induction of classification trees. In the context of structural decision trees, the pruning advantages it offers makes it, on practical grounds, the predicate selection function of choice. The pruning theorem was previously given in [4]; the contribution here is in the design of Acc^* , which overcomes some of the weaknesses of accuracy.

We end with a caveat. The working assumption throughout this section is that an efficiently computable pruning mechanism for some standard predicate selection function cannot be found. If one can be obtained, then the issues investigated here need to be revisited.

4 A Pruning Result for Regression-Tree Learning

We now present a predicate pruning result for structural regression-tree learning. We begin by stating the quadratic loss function commonly used in regression-tree learning. This is followed by the presentation of an efficient predicate pruning algorithm for this predicate selection function. The section ends with a discussion of some implementation issues.

Definition 11. Given a set \mathcal{E} of examples, we define the mean squared error $E_{\mathcal{E}}$ of \mathcal{E} by

$$E_{\mathcal{E}} = \min_c \sum_{(x,y) \in \mathcal{E}} (y - c)^2.$$

The unique minimizing c here is the empirical mean of the regression values in \mathcal{E} , and that is what we use to label the leaf nodes in a regression tree.

Definition 12. Let \mathcal{E} be a set of examples and $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$ a partition of \mathcal{E} . We define the mean squared error, $Q_{\mathcal{P}}$, of \mathcal{P} by

$$Q_{\mathcal{P}} = E_{\mathcal{E}_1} + E_{\mathcal{E}_2}.$$

Given a predicate search space \mathcal{S} and a set \mathcal{E} of examples, the goal is thus to find a predicate in \mathcal{S} that minimizes Q . As pointed in [1, §16.1], this formulation is well-founded since one can show that

$$\mathbb{E}(f(x) - y)^2 = \mathbb{E}(\mathbb{E}(y|x) - f(x))^2 + \mathbb{E}(\mathbb{E}(y|x) - y)^2,$$

which implies that choosing a function f to minimize Q is equivalent to finding the best approximation of the conditional expectation of y given x . See, for more details, [6, §8.3].

Proposition 13. Let \mathcal{E}_1 and \mathcal{E}_2 be sets of examples. If $\mathcal{E}_1 \subseteq \mathcal{E}_2$, then $E_{\mathcal{E}_1} \leq E_{\mathcal{E}_2}$.

Proof. Straightforward. □

To search through the space of predicates efficiently by means of predicate pruning, we need a way to predict the smallest error that can be obtained from the descendants of a predicate. This motivates the next definition.

Definition 14. Let \mathcal{E} be a set of examples and $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$ a partition of \mathcal{E} . We define the regression refinement bound of \mathcal{P} by

$$C_{\mathcal{P}} = \min_{\mathcal{P}'} Q_{\mathcal{P}'},$$

where \mathcal{P}' is a refinement of \mathcal{P} . A refinement \mathcal{P}^* of \mathcal{P} that satisfies $Q_{\mathcal{P}^*} = C_{\mathcal{P}}$ is called a minimizing refinement of \mathcal{P} .

Proposition 15. Let \mathcal{E} be a set of examples and \mathcal{P} a partition of \mathcal{E} . If \mathcal{P}' is a refinement of \mathcal{P} , then $Q_{\mathcal{P}'} \geq C_{\mathcal{P}}$. In particular, $Q_{\mathcal{P}} \geq C_{\mathcal{P}}$.

Proof. By the definition of $C_{\mathcal{P}}$. □

The refinement bound $C_{\mathcal{P}}$ has the obvious definition, but can it be computed efficiently? Clearly, an exhaustive search through all possible refinements of \mathcal{P} is impractical; a more efficient algorithm is needed. We next study this minimization problem.

A lower bound for $C_{\mathcal{P}}$ can be easily obtained.

Proposition 16. *Let \mathcal{E} be a set of examples and $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$ a partition of \mathcal{E} . Then $E_{\mathcal{E}_2} \leq C_{\mathcal{P}}$. In particular, when $\mathcal{E}_1 = \emptyset$, $E_{\mathcal{E}_2} = C_{\mathcal{P}}$.*

Proof. Let $\mathcal{P}^* = (\mathcal{E}_1^*, \mathcal{E}_2^*)$ be a minimizing refinement of \mathcal{P} . By Proposition 13, we have $E_{\mathcal{E}_2} \leq E_{\mathcal{E}_2^*}$ since $\mathcal{E}_2 \subseteq \mathcal{E}_2^*$. From that, we have

$$Q_{\mathcal{P}^*} = E_{\mathcal{E}_1^*} + E_{\mathcal{E}_2^*} \geq E_{\mathcal{E}_2^*} \geq E_{\mathcal{E}_2}.$$

Clearly, when $\mathcal{E}_1 = \emptyset$, $\mathcal{E}_1^* = \emptyset$ and $\mathcal{E}_2^* = \mathcal{E}_2$. □

It turns out that to compute $C_{\mathcal{P}}$ for a partition $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$, we can restrict our attention to just those refinements that can be obtained by cutting a sorted version of \mathcal{E}_1 into two halves, and assigning one or the other to the new \mathcal{E}_2 . Figure 2 gives an algorithm for computing $C_{\mathcal{P}}$. In the algorithm, $\mathcal{E}[i, j]$ denotes the subset of \mathcal{E} formed from taking the i -th to j -th element(s). If $i > j$, we define $\mathcal{E}[i, j]$ to be \emptyset .

```

function RegRefinementBound( $\mathcal{P}$ ) returns  $C_{\mathcal{P}}$ ;
input:  $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$ , a partition;

if  $\mathcal{E}_1 = \emptyset$  return  $E_{\mathcal{E}_2}$ ;
sort( $\mathcal{E}_1$ );
minerr :=  $Q_{\mathcal{P}}$ ;
for each  $i$  from 1 to  $|\mathcal{E}_1|$  do
     $\mathcal{E}_{11}$  :=  $\mathcal{E}_1[1, i]$ ;
     $\mathcal{E}_{12}$  :=  $\mathcal{E}_1[i + 1, |\mathcal{E}_1|]$ ;
     $\mathcal{P}_1$  :=  $(\mathcal{E}_{11}, \mathcal{E}_{12} \cup \mathcal{E}_2)$ ;
     $\mathcal{P}_2$  :=  $(\mathcal{E}_{12}, \mathcal{E}_{11} \cup \mathcal{E}_2)$ ;
    minerr :=  $\min\{\textit{minerr}, Q_{\mathcal{P}_1}, Q_{\mathcal{P}_2}\}$ 
return minerr;

```

Fig. 2. Algorithm for calculating $C_{\mathcal{P}}$

The *sort* function in line 4 in Figure 2 is with respect to the following total order \preceq on the examples. Examples are first ordered increasingly by their regression values. Examples with the same regression values are then ordered according to a lexicographic order on the individuals. We denote by $\max_{\preceq}(\mathcal{E})$ and $\min_{\preceq}(\mathcal{E})$ the largest and smallest examples in \mathcal{E} as ordered by \preceq .

We now show the value returned by the algorithm *RegRefinementBound* on input \mathcal{P} is $C_{\mathcal{P}}$. First, a technical lemma.

Proposition 17. *Let \mathcal{E} be a set of examples and $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$, $\mathcal{E}_1 \neq \emptyset$, a partition of \mathcal{E} . Suppose $\mathcal{P}_{min} = (\mathcal{E}_{11}, \mathcal{E}_{12} \cup \mathcal{E}_2)$ is a minimizing refinement of \mathcal{P} , where $\mathcal{E}_{11} \cup \mathcal{E}_{12} = \mathcal{E}_1$. Then \mathcal{P}_{min} must satisfy the following property:*

1. $\forall (x_1, y_1) \in \mathcal{E}_{11}, \forall (x_2, y_2) \in \mathcal{E}_{12}, y_1 \leq y_2$; or
2. $\forall (x_1, y_1) \in \mathcal{E}_{12}, \forall (x_2, y_2) \in \mathcal{E}_{11}, y_1 \leq y_2$.

Proof. If either \mathcal{E}_{11} or \mathcal{E}_{12} is empty, then the property holds trivially. Now consider the case when both \mathcal{E}_{11} and \mathcal{E}_{12} are non-empty. Suppose the property does not hold. Then there exist (x_1, y_1) in \mathcal{E}_{11} and (x_2, y_2) in \mathcal{E}_{12} such that $y_1 > y_2$, and (x_3, y_3) in \mathcal{E}_{12} and (x_4, y_4) in \mathcal{E}_{11} such that $y_3 > y_4$. We have

$$\max_y(\mathcal{E}_{11}) \geq y_1 > y_2 \geq \min_y(\mathcal{E}_{12}); \text{ and} \quad (1)$$

$$\max_y(\mathcal{E}_{12}) \geq y_3 > y_4 \geq \min_y(\mathcal{E}_{11}). \quad (2)$$

We now show that we can always pick (x', e') from \mathcal{E}_{11} and (x'', e'') from \mathcal{E}_{12} and interchange them to produce another refinement of \mathcal{P} with a lower error, thus contradicting the minimality of \mathcal{P}_{min} .

Let \bar{y}_1 and \bar{y}_2 denote, respectively, the empirical means of the regression values in \mathcal{E}_{11} and $\mathcal{E}_{12} \cup \mathcal{E}_2$. There are two cases to consider, and we state the elements we choose in each case.

1. If $\bar{y}_1 \leq \bar{y}_2$, pick (x', e') to be $\max_{\preceq}(\mathcal{E}_{11})$, and (x'', e'') to be $\min_{\preceq}(\mathcal{E}_{12})$. We have $e' > e''$ from (1).
2. If $\bar{y}_1 > \bar{y}_2$, pick (x', e') to be $\min_{\preceq}(\mathcal{E}_{11})$, and (x'', e'') to be $\max_{\preceq}(\mathcal{E}_{12})$. We have $e'' > e'$ from (2).

Let $\mathcal{E}'_1 = \mathcal{E}_{11} \cup \{e''\} \setminus \{e'\}$ and $\mathcal{E}'_2 = \mathcal{E}_2 \cup \mathcal{E}_{12} \cup \{e'\} \setminus \{e''\}$. Clearly, $\mathcal{P}' = (\mathcal{E}'_1, \mathcal{E}'_2)$ is a refinement of \mathcal{P} . We have

$$\begin{aligned} Q_{\mathcal{P}_{min}} &= \sum_{(x,y) \in \mathcal{E}_{11}} (y - \bar{y}_1)^2 + \sum_{(x,y) \in \mathcal{E}_2 \cup \mathcal{E}_{12}} (y - \bar{y}_2)^2 \\ &= (e' - \bar{y}_1)^2 + \sum_{(x,y) \in \mathcal{E}_{11} \setminus \{e'\}} (y - \bar{y}_1)^2 + (e'' - \bar{y}_2)^2 + \sum_{(x,y) \in \mathcal{E}_2 \cup \mathcal{E}_{12} \setminus \{e''\}} (y - \bar{y}_2)^2 \\ &\geq (e'' - \bar{y}_1)^2 + \sum_{(x,y) \in \mathcal{E}_{11} \setminus \{e'\}} (y - \bar{y}_1)^2 + (e' - \bar{y}_2)^2 + \sum_{(x,y) \in \mathcal{E}_2 \cup \mathcal{E}_{12} \setminus \{e''\}} (y - \bar{y}_2)^2 \\ &> Q_{\mathcal{P}'}. \end{aligned}$$

We can make the third step because

$$(e' - \bar{y}_1)^2 + (e'' - \bar{y}_2)^2 - [(e'' - \bar{y}_1)^2 + (e' - \bar{y}_2)^2] = 2(e'' - e')(\bar{y}_1 - \bar{y}_2) \geq 0$$

in both cases. The fourth step follows because $e' \neq e''$. (Recall Definition 11 and the remark following it.) \square

Proposition 18. *Given a partition \mathcal{P} , $\text{RegRefinementBound}$ correctly finds $C_{\mathcal{P}}$.*

Proof. Let \mathcal{P} be $(\mathcal{E}_1, \mathcal{E}_2)$. There are two cases. If $\mathcal{E}_1 = \emptyset$, the algorithm returns $E_{\mathcal{E}_2}$, which is equal to $C_{\mathcal{P}}$ by Proposition 16. If \mathcal{E}_1 is non-empty, then one of the minimizing refinements must satisfy the property stated in Proposition 17. The algorithm conducts an exhaustive search of all such partitions and must therefore find $C_{\mathcal{P}}$. \square

The *RegRefinementBound* algorithm, when implemented naïvely, has time complexity $O(|\mathcal{E}|^2)$, since there are $O(|\mathcal{E}|)$ iterations, and the computation of $Q_{\mathcal{P}_1}$ and $Q_{\mathcal{P}_2}$ in each iteration takes $O(|\mathcal{E}|)$ time. This can be significantly improved. We now give an implementation of *RegRefinementBound* that runs in time linear in the size of \mathcal{E} .

Proposition 19. *Given a set \mathcal{E} of examples and a partition $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$ of \mathcal{E} , assuming \mathcal{E}_1 is sorted according to \preceq , $C_{\mathcal{P}}$ can be computed in time $O(|\mathcal{E}|)$.*

Proof. Let $\bar{y}, \bar{y}_1, \bar{y}_2$ denote the empirical means of the regression values in $\mathcal{E}, \mathcal{E}_1$ and \mathcal{E}_2 . We can rewrite the error function as follows.

$$\begin{aligned}
Q_{\mathcal{P}} &= \sum_{(x,y) \in \mathcal{E}_1} (y - \bar{y}_1)^2 + \sum_{(x,y) \in \mathcal{E}_2} (y - \bar{y}_2)^2 \\
&= \sum_{(x,y) \in \mathcal{E}_1} [(y - \bar{y}) - (\bar{y}_1 - \bar{y})]^2 + \sum_{(x,y) \in \mathcal{E}_2} [(y - \bar{y}) - (\bar{y}_2 - \bar{y})]^2 \\
&= \sum_{(x,y) \in \mathcal{E}_1} [(y - \bar{y})^2 - (\bar{y}_1 - \bar{y})^2] + \sum_{(x,y) \in \mathcal{E}_2} [(y - \bar{y})^2 - (\bar{y}_2 - \bar{y})^2] \\
&= \sum_{(x,y) \in \mathcal{E}_1} (y - \bar{y})^2 - |\mathcal{E}_1|(\bar{y}_1 - \bar{y})^2 + \sum_{(x,y) \in \mathcal{E}_2} (y - \bar{y})^2 - |\mathcal{E}_2|(\bar{y}_2 - \bar{y})^2 \\
&= \sum_{(x,y) \in \mathcal{E}} (y - \bar{y})^2 - (|\mathcal{E}_1|\bar{y}_1^2 + |\mathcal{E}_1|\bar{y}^2 - |\mathcal{E}_1|2\bar{y}_1\bar{y} + |\mathcal{E}_2|\bar{y}_2^2 + |\mathcal{E}_2|\bar{y}^2 - |\mathcal{E}_2|2\bar{y}_2\bar{y}) \\
&= \sum_{(x,y) \in \mathcal{E}} (y - \bar{y})^2 - (|\mathcal{E}_1 + \mathcal{E}_2|\bar{y}^2 - 2\bar{y}(|\mathcal{E}_1|\bar{y}_1 + |\mathcal{E}_2|\bar{y}_2) + |\mathcal{E}_1|\bar{y}_1^2 + |\mathcal{E}_2|\bar{y}_2^2) \\
&= \sum_{(x,y) \in \mathcal{E}} (y - \bar{y})^2 + \bar{y}|\mathcal{E}|\bar{y} - (|\mathcal{E}_1|\bar{y}_1^2 + |\mathcal{E}_2|\bar{y}_2^2) \\
&= \sum_{(x,y) \in \mathcal{E}} (y - \bar{y})^2 + \frac{1}{|\mathcal{E}|} \left(\sum_{(x,y) \in \mathcal{E}} y \right)^2 - \left(\frac{1}{|\mathcal{E}_1|} \left(\sum_{(x,y) \in \mathcal{E}_1} y \right)^2 + \frac{1}{|\mathcal{E}_2|} \left(\sum_{(x,y) \in \mathcal{E}_2} y \right)^2 \right). \tag{3}
\end{aligned}$$

From that, we can reformulate the optimization problem as

$$\begin{aligned}
C_{\mathcal{P}} &= \min_{\mathcal{P}'} Q_{\mathcal{P}'} \\
&= \sum_{(x,y) \in \mathcal{E}} (y - \bar{y})^2 + \frac{1}{|\mathcal{E}|} \left(\sum_{\mathcal{E}} y \right)^2 - \max_{\mathcal{P}'} \left(\frac{1}{|\mathcal{E}'_1|} \left(\sum_{\mathcal{E}'_1} y \right)^2 + \frac{1}{|\mathcal{E}'_2|} \left(\sum_{\mathcal{E}'_2} y \right)^2 \right) \tag{4}
\end{aligned}$$

where $\mathcal{P}' = (\mathcal{E}'_1, \mathcal{E}'_2)$ is a refinement of \mathcal{P} . Thus we are left with a maximization problem that can be computed in time linear in the size of \mathcal{E} . All that is required are a few preprocessing steps to compute the sum of all the regression values in \mathcal{E}_2 and the prefix sums ($S[i] = \sum_{1 \leq j \leq i} \mathcal{E}_1[j]$) of each element in the sorted \mathcal{E}_1 .

Here, $\mathcal{E}[j]$ denotes the regression value of the j -th element in \mathcal{E} . The details are given in Figure 3. In the algorithm, the function $div(x, y)$ is defined to be x/y if $y \neq 0$, and 0 otherwise. Note that the formula for $Q_{\mathcal{P}_1}$ and $Q_{\mathcal{P}_2}$ has the same general form as the last step in (3).

```

function RegRefinementBound2( $\mathcal{P}$ ) returns  $C_{\mathcal{P}}$ ;
input:  $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$ , a partition,  $\mathcal{E}_1$  sorted;

if  $\mathcal{E}_1 = \emptyset$  return  $E_{\mathcal{E}_2}$ ;
 $K := \sum_{(x,y) \in \mathcal{E}} (y - \bar{y})^2 + \frac{1}{|\mathcal{E}|} \left( \sum_{(x,y) \in \mathcal{E}} y \right)^2$ ;
 $sum2 := \sum_{(x,y) \in \mathcal{E}_2} y$ ;
 $S := prefixSum(\mathcal{E}_1)$ ;
 $minerr := Q_{\mathcal{P}}$ ;
for each  $i$  from 1 to  $|\mathcal{E}_1|$  do

     $Q_{\mathcal{P}_1} := K - \left( \frac{1}{i} (S[i])^2 + div(1, |\mathcal{E}_2| + |\mathcal{E}_1| - i) (sum2 + S[|\mathcal{E}_1|] - S[i])^2 \right)$ ;

     $Q_{\mathcal{P}_2} := K - (div(1, |\mathcal{E}_1| - i) (S[|\mathcal{E}_1|] - S[i])^2 + \frac{1}{|\mathcal{E}_2| + i} (sum2 + S[i])^2)$ ;

     $minerr := \min\{minerr, Q_{\mathcal{P}_1}, Q_{\mathcal{P}_2}\}$ 

return  $minerr$ ;

```

Fig. 3. An implementation of *RegRefinementBound*.

Each of the preprocessing steps can be done in $O(|\mathcal{E}|)$ time. There are $O(|\mathcal{E}|)$ iterations in the **for** loop, and each iteration takes $O(1)$ time. The overall complexity of the algorithm is thus $O(|\mathcal{E}|)$. \square

Can we avoid conducting an exhaustive search of the solution set? Given the form of the minimizing refinement, one might conjecture that given a partition $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$, if $\bar{y}_1 < \bar{y}_2$, where \bar{y}_1 and \bar{y}_2 denote, respectively, the empirical means of the regression values in \mathcal{E}_1 and \mathcal{E}_2 , then the minimizing refinement must be in the set

$$S1 = \{(\mathcal{E}'_1, \mathcal{E}'_2) : \mathcal{E}'_1 = \mathcal{E}_1[1, i], \mathcal{E}'_2 = \mathcal{E}_1[i + 1, |\mathcal{E}_1|] \cup \mathcal{E}_2, 1 \leq i \leq |\mathcal{E}_1|\}$$

where \mathcal{E}_1 here is assumed sorted. Similarly when $\bar{y}_1 > \bar{y}_2$. This is not true, as shown in the next example.

Example 20. Consider the partition

$$\mathcal{P} = (\{(x_1, 0.3), (x_2, 0.5), (x_3, 0.98)\}, \{(x_4, 0.41), (x_5, 0.53), (x_6, 0.77), (x_7, 0.9)\})$$

with $\bar{y}_1 = 0.59$ and $\bar{y}_2 = 0.65$. The best refinement in the set $S1$, with error 0.255, is

$$(\{(x_1, 0.3), (x_2, 0.5)\}, \{(x_3, 0.98), (x_4, 0.41), (x_5, 0.53), (x_6, 0.77), (x_7, 0.9)\}).$$

The actual minimizing refinement is

$$(\{(x_3, 0.98)\}, \{(x_1, 0.3), (x_2, 0.5), (x_4, 0.41), (x_5, 0.53), (x_6, 0.77), (x_7, 0.9)\})$$

with error 0.254. ◀

Another conjecture is that the errors obtained by increasing the index i in both $S1$ and $S2$ (defined similarly to $S1$, see below) traces a quadratic curve with a single (local) minimum. This also turns out to be false, as shown in the next example.

Example 21. Consider the partition

$$\mathcal{P} = (\{(x_1, 0.29), (x_2, 0.36), (x_3, 0.81), (x_4, 0.92)\}, \{(x_5, 0.95)\}).$$

The refinements in the set

$$S2 = \{(\mathcal{E}'_1, \mathcal{E}'_2) : \mathcal{E}'_1 = \mathcal{E}_1[i + 1, |\mathcal{E}_1|], \mathcal{E}'_2 = \mathcal{E}_1[1, i] \cup \mathcal{E}_2, 0 \leq i \leq |\mathcal{E}_1|\}$$

where \mathcal{E}_1 is assumed sorted produces the following sequence of errors with increasing values of i : [0.3001, 0.3938, 0.2689, 0.3202, 0.4009]. ◀

To get an indication of the effectiveness of the pruning mechanism, we conducted a small experiment using the same six datasets given in Table 1. Each of these was converted into a regression problem by appropriate relabellings of the individuals, with individuals coming from the same class relabelled with random numbers chosen from the same subinterval of the real line. A complete search through the predicate space was then performed for each. The results are shown in Table 4. It is clear from the table that significant reduction in computation time can be achieved in practice.

Table 4. Efficiency of the predicate pruning mechanism for regression

Dataset	$ \mathcal{S} $	Searched
Int Trees	396	120 (30.30%)
Mutagenesis	535	124 (23.18%)
East West	2073	633 (30.54%)
Headline	2850	148 (5.19%)
Protein	9262	581 (6.27%)
Musk-1	1,679,615,641	3,814,746 (0.23%)

5 Related Work and Conclusion

The idea of predicate pruning is, of course, not new in ILP. For example, both Progol and Aleph have such mechanisms for some standard rule-evaluation functions; see [13, §7.4.7] and [15, §3.2.3]. This paper extend such techniques to top-down induction of structural trees, in both the classification and regression settings. The pruning theorem for classification has been established in [4] before; the contribution here is in the design of Acc^* . The pruning theorem for regression is new and it is the main contribution of this paper.

Acknowledgements

We would like to thank Evan Greensmith and Mathi Nagarajan for helpful discussions on the subject matter of this paper. Thanks are also due to the anonymous reviewers whose comments helped improve the presentation of the paper.

References

1. Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.
2. Hendrik Blockeel. *Top-Down Induction of First Order Logical Decision Trees*. PhD thesis, Departement Computerwetenschappen, Katholieke Universiteit Leuven, 1998.
3. Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.
4. Antony F. Bowers, Christophe Giraud-Carrier, and John W. Lloyd. Classification of individuals with complex structure. In P. Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, pages 81–88. Morgan Kaufmann, 2000.
5. Antony F. Bowers, Christophe Giraud-Carrier, and John W. Lloyd. A knowledge representation framework for inductive learning. Available at <http://rsise.anu.edu.au/~jw1/>, 2001.
6. Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.
7. Wray Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.
8. Michael Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 459–468. ACM Press, 1996.
9. Stefan Kramer. Structural regression trees. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 812–819. AAAI Press, 1996.
10. Stefan Kramer and Gerhard Widmer. Inducing classification and regression trees in first order logic. In Sašo Džeroski and Nada Lavrač, editors, *Relational Data Mining*, chapter 6. Springer, 2001.
11. John W. Lloyd. *Logic for Learning: Learning Comprehensible Theories from Structured Data*. Cognitive Technologies. Springer, 2003.
12. John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3:319–342, 1989.
13. Stephen Muggleton and John Firth. Relational rule learning with CPROGOL4.4: A tutorial introduction. In Sašo Džeroski and Nada Lavrač, editors, *Relational Data Mining*, chapter 7. Springer, 2001.
14. John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
15. Ashwin Srinivasan. The Aleph Manual. Technical report, Computing Laboratory, Oxford University, 2000.