Appears in *Proceedings, Second International Conference on Knowledge Discovery and Data Mining,* Portland, OR: AAAI Press (1996). © 1996 AAAI.

Growing Simpler Decision Trees to Facilitate Knowledge Discovery

Kevin J. Cherkauer **Jude W. Shavlik**

Department of Computer Sciences
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706, USA
cherkauer@cs.wisc.edu, shavlik@cs.wisc.edu

Abstract

When using machine learning techniques for knowledge discovery, output that is comprehensible to a human is as important as predictive accuracy. We introduce a new algorithm, SET-GEN, that improves the comprehensibility of decision trees grown by standard C4.5 without reducing accuracy. It does this by using genetic search to select the set of input features C4.5 is allowed to use to build its tree. We test SET-GEN on a wide variety of real-world datasets and show that SET-GEN trees are significantly smaller and reference significantly fewer features than trees grown by C4.5 without using SET-GEN. Statistical significance tests show that the accuracies of SET-GEN's trees are either not distinguishable from or are more accurate than those of the original C4.5 trees on all ten datasets tested.

Introduction

One approach to knowledge discovery in databases (DBs) is to apply inductive learning algorithms to derive models of interesting aspects of the data. The predictive accuracy of such a model is obviously important. However, human comprehensibility of the learned model is equally vital so that we can add the knowledge it captures to our understanding of the domain or validate the model for critical applications.

To address the issue of human comprehensibility, we introduce SET-GEN, a new algorithmic approach to knowledge discovery that improves the comprehensibility of decision trees grown by a state-of-the-art tree induction algorithm, C4.5 (Quinlan 1993), without reducing tree accuracy. SET-GEN takes a DB of labeled examples (vectors of feature-value pairs) and selects a subset of the available features for training C4.5. Its goal is to choose a set of features that results in

- Predictive accuracy at least as good as that of running C4.5 without SET-GEN
- Significantly smaller decision trees
- Significantly fewer unique input features referenced

Reducing tree size makes it easier to understand the relationships contained in the tree, and referencing fewer features focuses attention on the most important information. We demonstrate SET-GEN on a wide variety of real-world prediction problems and show empirically that it meets our stated goals.

The SET-Gen Algorithm

SET-GEN performs feature-subset selection for decision-tree induction. Table 1 gives pseudocode for the algorithm. SET-GEN applies a genetic algorithm (GA; Goldberg 1989) with a wrapper-style evaluation function (John, Kohavi, & Pfleger 1994) to search many candidate feature subsets. It uses ten-fold cross validation on the training examples to estimate the quality, or *fitness,* of each candidate. That is, the training data is partitioned into ten equal-sized sets, each of which serves as an unseen *validation set* used to estimate the accuracy of a C4.5 decision tree trained on the remaining nine sets using just the candidate features. Fitness is a function of the number of candidate features, the average size of the ten trees, and the average tree accuracy on the validation sets.

SET-GEN maintains a population of the best feature subsets it has found. New subsets are created by applying genetic operators to population members. If a new subset is more fit than the worst member of the population, it replaces that member; otherwise the new subset is discarded. After completing the desired number of subset evaluations, SET-GEN uses the entire training set to grow a single C4.5 tree using only the features in the best subset it has found. It outputs this final tree and the corresponding feature subset.

SET-Gen's Genome

SET-GEN represents a feature subset as a fixed-length vector called a *genome.* Each genome entry may either contain a feature or be empty. The genome in Figure 1 represents a subset comprised of features f_2, f_7, and f_{15}. A feature may occur multiple times and in any position, making SET-GEN's genome somewhat unusual among GAs. An indicator bit vector with one entry per input feature would be more traditional. Our justification for SET-GEN's genome style is twofold. First, the fact that features can appear multiple times potentially slows the loss of diversity that tends to occur during genetic search (Forrest & Mitchell 1993) and allows better features to proliferate. Second, unlike the bit-vector genome, SET-GEN's genome length does not

Table 1: SET-GEN pseudocode.

```
Algorithm SET-Gen
    Input labeled training examples, program parameters
    Choose pruning level via 10-fold cross validation
        on training data (builds 10 decision trees)
    While perform more evaluations?
        If pop. is not full, Child = fill genome randomly
        Else
            Op = choose genetic operator randomly
            Parents = choose parent(s) randomly from
                population proportional to fitness
            Child = Apply(Op, Parents)
        End If
        Evaluate Child fitness via 10-fold cross validation
            on training data (builds 10 decision trees)
        If population is not full, add Child to population
        Else
            Worst = population member with worst fitness
            If Fitness(Child) > Fitness(Worst)
                replace Worst with Child in population
            Else discard Child
        End If
    End While
    FinalFeatures = features present in best pop. member
    FinalTree = grow decision tree from all training data
        using only features in FinalFeatures
    Output FinalTree, FinalFeatures
End Algorithm SET-Gen
```

		f_7	f_7	f_2		f_7		f_{15}			f_2	

Figure 1: An example SET-GEN genome, representing the feature subset $\{f_2, f_7, f_{15}\}$.

depend on the number of input features. By default, SET-GEN's genome size is the same as the number of available features, but if desired one can choose a larger or smaller genome. Smaller genomes bias SET-GEN toward smaller feature subsets and simpler trees. Because trees can be grown faster when there are fewer features to test as splits, this also reduces evaluation time, making the algorithm tractable for larger DBs.

SET-Gen's Genetic Operators

SET-GEN's genetic operators are *Crossover, Mutate,* and *Delete Feature.* Each uses one or two *parent* feature subsets to create a new *child* subset for evaluation.

The *Crossover* operator is a variant of a uniform crossover, and produces a single child from a primary and a secondary parent. First, the genome of one parent is rotated a random distance. Then each entry of the child is filled by copying the corresponding entry of the primary parent with probability $1 - P_c$ and that of the secondary parent with probability P_c (the *crossover rate*). In our experiments, we set P_c to 0.10, so a typical child receives approximately 90% of its genome from the primary parent.

We chose a uniform crossover with a low crossover rate instead of a one-point crossover because we felt that small "tweaks" would more likely improve a current solution than the larger jumps one-point crossover tends to make. This is only an intuition; we have not yet compared the performance of this crossover to a one-point crossover. SET-GEN's low crossover rate hopefully results in low disruption across generations of high-order schemata involving many features (cf. Goldberg 1989). However, a one-point crossover might take better advantage of lower-level "building blocks" (Goldberg 1989) as individual features could assemble themselves in spatially adjacent fashion to increase their chances of being exchanged as a unit.

SET-GEN's *Mutate* operator uses one parent. Each entry of the child is copied from the parent with probability $1 - P_m$. With probability P_m (the *mutation rate*), it is filled randomly thus: 50% of the time, fill with a feature chosen equiprobably from among all input features; the other 50% of the time, leave the entry empty. P_m is 0.10 for the experiments.

Delete Feature uses one parent to produce a child that is identical except that all occurrences of one (equiprobably chosen) feature in the parent are removed from the child. This operator directly biases SET-GEN toward smaller feature subsets, and thus toward simpler, more comprehensible decision trees.

The initial population members are created by *Mutate* with a temporary mutation rate of 1.00. From then on, each new feature subset is produced by applying one of the genetic operators, chosen equiprobably, to parent(s) picked randomly from the current population proportional to their fitness (Goldberg 1989).

SET-Gen's Fitness Function

The core of SET-GEN is its fitness function, which evaluates feature subsets in terms of the accuracy and simplicity of their resulting trees[1]:

$$Fitness = \frac{3}{4}A + \frac{1}{4}\left(1 - \frac{S+F}{2}\right)$$

where A is the average validation-set accuracy of the trees SET-GEN builds on the training data; S is the average size of these trees, normalized by dividing by the average number of training examples they were built from; and F is the number of features in the subset being evaluated, normalized by the total number of available features. We define F as the number of features present, instead of the average number of features the trees reference, to create a fitness distinction between representations containing the same referenced features but different numbers of extra, unreferenced ones. Without this, there would be no selective pressure to eliminate unused features from the representations. We could instead simply delete all unused features immediately, but this would dramatically reduce the internal diversity of individuals early in the search, before it is apparent whether the unused features would prove valuable under different subset recombinations.

SET-GEN's fitness function is a linear combination of an accuracy term, A, and a simplicity term,

[1]The fitness variables are motivated by our previous work on representation Sufficiency, Economy, and Transparency ("SET"; Cherkauer & Shavlik 1996).

Table 2: Summary of datasets used: number of examples, classes, and features (discrete, continuous).

Dataset	Exs	Cls	Fts (Ds, Cn)
Auto Imports[a]	205	6	25 (10, 15)
Credit Approval[a]	690	2	15 (9, 6)
Heart Disease[a]	303	2	13 (8, 5)
Hepatitis[a]	155	2	19 (13, 6)
Lung Cancer[a]	32	3	56 (56, 0)
Lymphography[a]	148	4	18 (15, 3)
Magellan-SAR	611	2	137 (0, 137)
Promoters	468	2	57 (57, 0)
Ribosome Binding	1,877	2	49 (49, 0)
Splice Junctions[a]	3,190	3	60 (60, 0)

[a]Available publicly (Murphy & Aha 1994).

$(1 - \frac{S+F}{2})$. We weight the accuracy term more heavily to encourage SET-GEN to maintain the original accuracy level. All coefficients in the fitness function were chosen prior to running any experiments. Note that A and F (normalized) vary in the range [0, 1]. The normalization used for S attempts to put it on an approximately equivalent [0, 1] scale so that the weights in the fitness function are meaningful to a human. It is simply a heuristic that uses the number of training examples as a rough upper bound for expected tree size. The result is that the simplicity term ranges roughly over [0, 1] to match the range of the accuracy term, and S and F (tree size and number of features) are of about equal importance in the simplicity term.

Empirical Evaluation

We test SET-GEN on ten real-world problems from business, medicine, biology, and vision. These problems vary widely in the number and types of available input features and the number of examples in the DB. The datasets are summarized in Table 2. The Magellan-SAR data consists of features derived from small patches of radar images of the planet Venus, and the task is to determine if a patch contains a volcano (Burl *et al.* 1994). Promoter, Ribosome Binding, and Splice Junction are all problems of detecting different types of biologically significant sites on strands of DNA. Most of the DBs are publicly available through Murphy and Aha (1994). We chose these problems because of their diversity and interest to scientists in their respective fields. We did not preselect these datasets to favor SET-GEN in any way; these are all ten of the datasets we have tested it on to date.

Experimental Methodology

We evaluate SET-GEN and C4.5 by ten-fold cross validation on each problem's entire DB of examples and report average results over the ten folds, or *trials*. Accuracies are measured on the ten unseen test sets of the cross validation. (SET-GEN itself uses cross validation internally on the training examples to evaluate feature subsets, but this occurs inside the SET-GEN "black box" and has no bearing on the external cross validation used to assess algorithm performance.)

Table 3: SET-GEN parameter settings used (defaults).

Parameter	Value
Population size	100
Feature subsets evaluated	5,000
Genome size	# input feats
Probability a child is produced by *Crossover, Mutate, Del. Feature*	1/3, 1/3, 1/3
Crossover rate, Mutation rate	0.10, 0.10

The amount of tree pruning is a crucial parameter because it greatly affects accuracy, tree size, and number of features referenced. For each trial, both SET-GEN and C4.5 chose the pruning level by doing an initial, internal ten-fold cross-validation of standard C4.5 using only the training examples. The pruning level was chosen from among ten equally spaced confidence levels: 5%, 15%, 25%, ..., 95% (Quinlan 1993), and the one yielding the most accurate trees on the validation sets was then used to train on the entire training set for the remainder of the trial. (Thus, choosing the pruning level is part of SET-GEN and C4.5 training. This process is identical for the two algorithms.)

We fixed all other SET-GEN and C4.5 parameters at their default values. SET-GEN's parameter defaults are summarized in Table 3, and were chosen before running it on any of the datasets used in the experiments. C4.5's parameters are described in Quinlan (1993).

Experimental Results

We compare the average test-set accuracy, tree size, and number of features referenced for the ten pruned trees of C4.5 versus SET-GEN using two-tailed, matched-pair t-tests to check for statistically significant differences at the 0.05 significance level. The unpruned trees give qualitatively similar results, but tend to be larger and thus of less interest from a comprehensibility standpoint, so we do not include those results.

Figure 2 shows the average percent error on the ten unseen test sets of the final pruned trees for each problem. The C4.5 and SET-GEN error rates only differ statistically significantly on the Ribosome Binding problem, where SET-GEN has a lower error rate. SET-GEN thus meets our goal of retaining the accuracy level of standard C4.5.

Figure 3 shows the average number of (internal plus leaf) nodes in the final pruned trees. The size differences between C4.5 and SET-GEN are statistically significant for all datasets except Lung Cancer,[2] and in all ten cases the SET-GEN trees are smaller, frequently by a factor of two or more. Hence, SET-GEN meets our goal of reducing tree size.

Figure 4 shows the average number of unique features referenced by the final pruned trees. The C4.5

[2]Lung Cancer has only 32 available examples (Table 2), so variance is quite high.

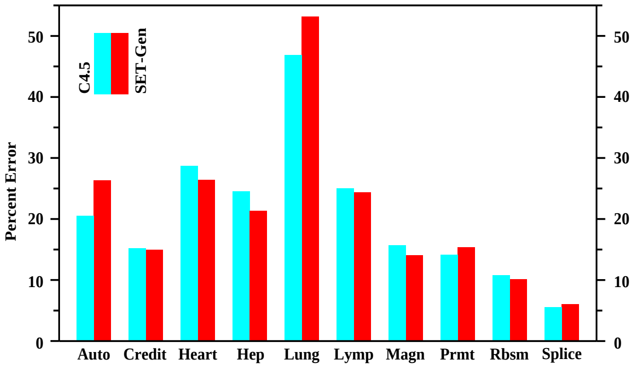

Figure 2: Average test-set error rates of final pruned trees.

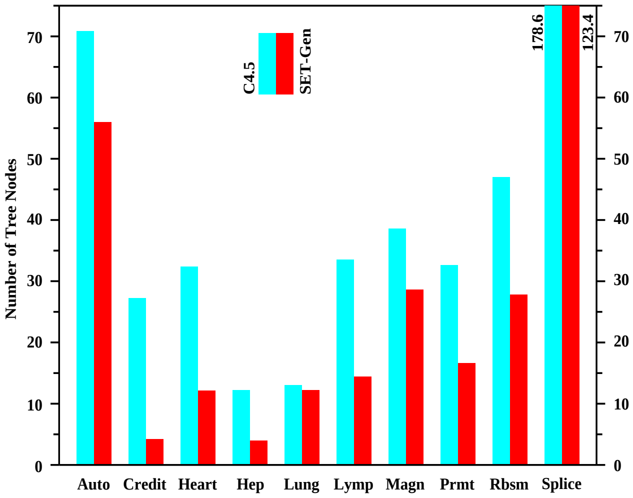

Figure 3: Average number of nodes in final pruned trees.

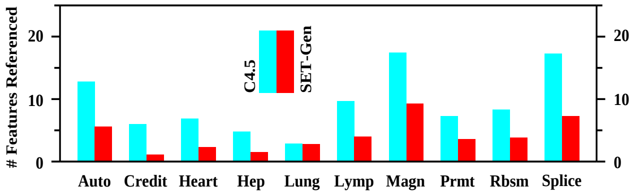

Figure 4: Average number of unique features referenced by final pruned trees.

versus SET-Gen differences are again statistically significant for all datasets except Lung Cancer, and in all ten cases the SET-Gen trees reference fewer features. SET-Gen's advantage over C4.5 here is almost always at least two to one. It thus meets our goal of reducing the number of features referenced.

In summary, these experiments show that SET-Gen simultaneously fulfills all three criteria we set for improving human comprehensibility without accuracy loss on a wide variety of real-world learning problems.

Related Work

The most closely related prior work is by Skalak (1994) and John, Kohavi, and Pfleger (1994). Skalak uses stochastic hill climbing to reduce nearest-neighbor model size, thus lowering computational cost, while retaining accuracy. However, his main thrust is selecting prototypical subsets of examples, rather than features. John *et al.* apply greedy feature selection to reduce C4.5 tree size without losing accuracy. In contrast to these systems, SET-Gen's genetic search is not greedy and thus can escape local optima. SET-Gen also focuses more strongly on human comprehensibility than John *et al.* by specifically seeking model simplicity.

Conclusions

Our goal is to induce more comprehensible decision trees to facilitate knowledge discovery, without reducing predictive accuracy. To achieve this, we introduced the SET-Gen feature-selection system and tested it on a wide variety of real-world problems, demonstrating empirically that it meets our goal. SET-Gen dramatically reduced the complexity of induced trees compared to C4.5, both in size and number of features referenced. Moreover, it did so without significantly reducing tree accuracy for any dataset, and in one case it even improved significantly on C4.5's accuracy. We hope SET-Gen will aid experts in better understanding these and other important problems. Our future work will compare SET-Gen to other feature selectors on the comprehensibility dimension and evaluate its current representational assumptions.

Acknowledgements

Thanks to U. Fayyad and P. Smyth for their aid in creating the Magellan-SAR dataset, R. Detrano (Heart Disease data), and M. Zwitter and M. Soklic (Lymphography data). This work was supported in part by a NASA GSRP fellowship held by KJC and ONR grant N00014-93-1-0998.

References

Burl, M.; Fayyad, U.; Perona, P.; Smyth, P.; and Burl, M. 1994. Automating the hunt for volcanoes on Venus. In *IEEE Comp Soc Conf Comp Vision & Pat Rec: Proc.* IEEE Computer Society Press.

Cherkauer, K., and Shavlik, J. 1996. Rapid quality estimation of neural network input representations. In *Advances in Neural Info Proc Sys 8*. MIT Press.

Forrest, S., and Mitchell, M. 1993. What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. *Machine Learning* 13:285–319.

Goldberg, D. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

John, G.; Kohavi, R.; and Pfleger, K. 1994. Irrelevant features and the subset selection problem. In *Mach Learn: Proc 11th Intl Conf*, 121–129. Morgan Kaufmann.

Murphy, P., and Aha, D. 1994. Univ. California Irvine repository of machine learning databases. At *http://www.ics.uci.edu/~mlearn/MLRepository.html.*

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

Skalak, D. 1994. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Mach Learn: Proc 11th Intl Conf*, 293–301. Morgan Kaufmann.