To appear in ICML97

Option Decision Trees with Majority Votes

Ron Kohavi
Silicon Graphics, Inc.
2011 N. Shoreline Blvd
Mountain View, CA 94043-1389
ronnyk@sgi.com

Clayton Kunz
Computer Science Dept.
Stanford University
Stanford, CA. 94305
clay@cs.stanford.edu

Abstract

We describe an experimental study of *Option Decision Trees* with majority votes. Option Decision Trees generalize regular decision trees by allowing *option* nodes in addition to decision nodes; such nodes allow for several possible tests to be conducted instead of the commonly used single test. Our goal was to explore when option nodes are most useful and to control the growth of the trees so that additional complexity of little utility is limited. Option Decision Trees can reduce the error of decision trees on real-world problems by combining multiple options, with the motivation similar to that of voting algorithms that learn multiple models and combine the predictions. However, unlike voting algorithms, an Option Decision Tree provides a single structured classifier (one decision tree), which can be interpreted more easily by humans. Our results show that for the tested problems, we can achieve significant reduction in error rates for trees restricted to two levels of option nodes at the top. When very large Option Decision Trees are built, Option Decision Trees outperform Bagging in reducing error, although the trees are much larger and cannot be reasonably interpreted by humans.

1 Introduction

Regular decision trees make a single test at each node and trace a single path corresponding to test outcomes until a leaf is reached and a prediction is made. Option trees were introduced by Buntine (1992*a*) as a generalization of decision trees. Option trees allow *option nodes*, which replace a single decision by a set of decisions. Classification is done in a way similar to regular decision trees, except that a rule is applied to option nodes to combine the predictions of the children nodes.

There are several reasons why we can expect option decision trees to outperform regular decision trees in their prediction accuracy. The important ones are limited lookahead and stability. The common strategy for building decision trees is top-down using one level of lookahead. Specifically, an evaluation criterion scores possible tests at a node based on the children the node would have. Such limited lookahead prefers attributes that score high in isolation and may overlook combinations of attributes. Multi-ply lookahead is computationally expensive and has not proven itself useful (Murthy & Salzberg 1995).

The second reason why option trees might be better than regular decision trees is related to stability (bias-variance) and risk reduction. Ali (1996, p. 106) describes the stability issue:

> ...it may happen that the candidate (*e.g.*, a literal in a rule or a node in a decision tree) with the highest information gain is flanked by other candidates that have "almost as much" information gain. It may be that due to the inclusion or exclusion of a few examples, in the training set, the candidate that is truly the best appears to be second best.

Breiman (1994, 1996) describes why decision trees are especially unstable: a minor change in one split close to the root will change the whole subtree below.

Several methods for combining classifiers have been proposed, including bagging, boosting, stacking, voting, averaging, ensembles (see Ali (1996) for a good review and additional recent work in Stolfo (1996)). Under these multiple-model schemes, multiple classifiers

are generated and then combined into a final predictor. The main disadvantage of these techniques is that the resulting structures are hard to interpret. Breiman (1996) wrote that "What one loses, with the [bagging of] trees, is a simple and interpretable structure. What one gains is increased accuracy."

While option trees have been previously investigated by Buntine (1992*a*, 1992*b*), he proposed other modifications at the same time and Option Decision Trees were not examined in isolation. Buntine (1992*a*) wrote that "It is unknown how much smoothing, option trees and multi-ply lookahead each contribute to the observed gain in prediction accuracy." Oliver & Hand (1995) described a study of several averaging alternatives in decision trees (path sets, fanned sets and extended fanned sets) and selected the subtrees to average based on the path an instance takes to the leaf. Earlier work of Kwok & Carter (1990) averages over multiple trees, which are selected by users or automatically in an attempt to limit the selected models to high-posterior trees given the data.

This paper provides a detailed study of option trees, showing how several parameters affect the tree size and the resulting error rates. Interestingly, our conclusions describing when option nodes are most useful differ from Buntine's. While Buntine wrote that option nodes are most useful for small samples because with large samples one test significantly outranks the others, we found option nodes to be more useful for large nodes near the root of the tree. While Buntine chose to retain tests within a factor of 0.005 of the best test, we found it most useful to choose factors two orders of magnitude larger, especially close to the root. The Option Decision Trees used here and in Buntine's study *are* different (we conduct majority voting while Buntine weighted them), but we believe our conclusions might be relevant for other types of Option Decision Trees as well. Buntine (personal communication) claimed that with Bayesian averaging, root options usually got eliminated during pruning. Therefore, for efficiencies sake, Buntine often eliminated them in the forward part of the search.

As we show later in the paper, the different parameters for growing Option Decision Trees allow trading error rates for comprehensibility (as represented by the number of nodes in the tree), and longer induction time. At one extreme are regular decision trees, which are the smallest but least accurate on average. At the other extreme are Option Decision Trees that have three orders of magnitude more nodes, but are the most accurate. Although in theory the no-free-lunch theorems imply that no algorithm can uniformly outperform others on all data sets, on the practical problems that we have tested, there exists a clear tradeoff.

Compared to Bagging, boosting, and similar voting schemes that offer the human a set of unrelated trees, Option Decision Trees provide the human with a single structure that is easier to interpret, albeit possibly very large. Combined with interactive tools for pruning options and subtrees, Option Decision Trees offer users a choice of a few good splits at nodes where there is uncertainty. This would be particularly helpful near the root, where option nodes are most important as will be shown empirically.

2 Classifying Using Option Decision Trees

Option Decision Trees are similar to regular decision trees, except that they can contain *option* nodes in addition to regular decision nodes and leaf nodes. Figure 1 depicts part of an actual Option Decision Tree that was created for the Tic-Tac-Toe domain with test-set error rates at the nodes. The task is to classify whether the end-game board is a "win for x." The choice of the center square for the root node was significantly better than all others and hence it was chosen. If the center square has an X or an O, there are reasonably good subsequent decisions to be made; however, if the center square is blank, there are several choices that rank close to each other and are hence chosen as options (three corners and the middle-left square). A few things to note:

1. The error rate for option 1, which had the highest split evaluation, and would have been chosen in place of the option node, is higher than the other options.
2. The error rate for the option node is lower than each child alone, an effect due to majority voting.
3. To create an equivalent tree using multiple models (four trees), one would have to replicate the left and right subtrees. Option Decision Trees provide a compact representation for many possible trees.

Classification of an instance I using Option Decision Trees with majority voting is a recursive procedure defined as follows:

1. For a leaf node, return the label predicted by the leaf.
2. For a decision node N, let N_c be the (unique) child of node N that matches the test outcome for in-

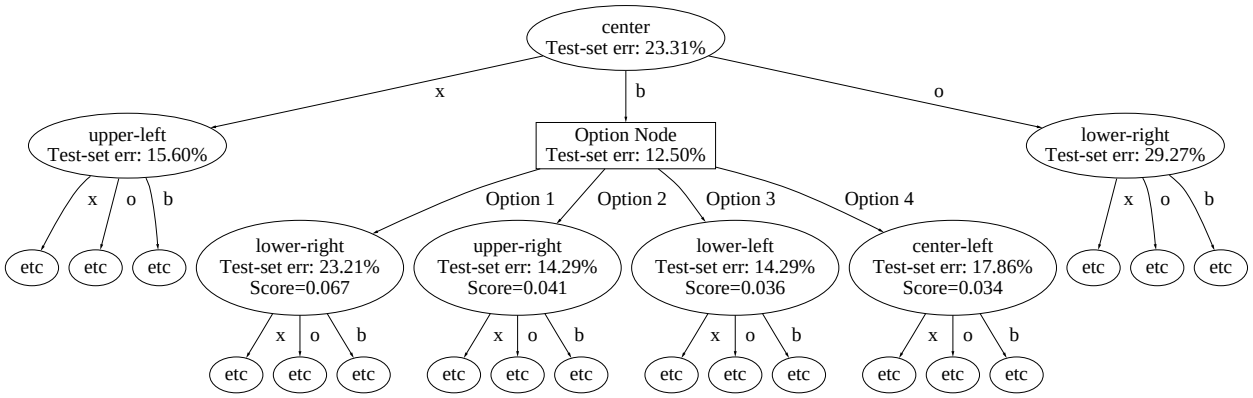

Figure 1: An example of an Option Decision Tree for the Tic-Tac-Toe data set. The root node is testing the center square. When the center square is blank, there are several good choices, and they are all generated under an option node.

stance I at the node. For a test, say $X < 30$, exactly one child node must match. The output of the decision node is the label predicted by node N_c.

3. For an option node N, the predicted label is the majority label of the label predicted by all children of N (ties are broken arbitrarily).

An option node is like an *or* node in *and-or* trees. It represents uncertainty in the decision process and serves two purposes: it informs the human who is looking at the tree of the uncertainty and shows "expert" subtrees that each solve the problem and make a prediction. From a Bayesian perspective, it reduces the risk by "averaging" several choices.

3 Inducing Option Decision Trees

We begin by describing our induction algorithm and then discuss several hypotheses in an attempt to understand when constructing option nodes is useful and when such construction adds little value (or hurts). As we will show, the tree growth has to be carefully controlled to avoid creating unmanageable trees that both exhaust memory and increase induction time.

3.1 The TDDT^{op} Induction Algorithm and Classifier

TDDT^{op} is a Top-Down Decision-Tree (TDDT) induction algorithm implemented in $\mathcal{MLC}$++ (Kohavi, Sommerfield & Dougherty 1996) that creates option nodes. The algorithm is similar to C4.5 (Quinlan 1993) with the evaluation criterion being normalized information-gain (information gain divided by log the number of children). Unknowns were regarded as a separate value. The algorithm grows the decision tree following the standard methodology of choosing the best attribute according to the evaluation criterion. After the tree is grown, a pruning phase replaces subtrees with leaves using the same pruning algorithm that C4.5 uses.

The principal modification to the basic TDDT algorithm is that instead of always selecting a single test, when several tests evaluate close to the best test, the TDDT^{op} algorithm creates an option node. All the data is sent to each child of an option node, which then splits the data according to its predetermined test. Since the votes are not weighted, it never makes sense to create an option node with two children; hence the minimum number of choices for an option node is three (four or more choices are reasonable in multi-class problems). The C4.5 pruning algorithm was modified so that the pessimistic error (used to evaluate whether to prune a node) of an option node was the average of the pessimistic errors of its children.

3.2 The Experimental Methodology

For the initial study in this section, we compare the TDDT algorithm and the TDDT^{op} algorithm. In the final comparison we add C4.5 (Quinlan 1993) and a bagged version of TDDT.

The data sets used in our final experiment are a superset of those used in Breiman (1996) taken from the UCI repository (except that the heart database from UCI we used did not match the CART version Breiman used). Table 1 shows a summary of the characteristics of the files and the evaluation methodology. In the final experiments we use the same methodology that

Data set	Instances	Evaluation train/test	Attributes	Classes
Breast cancer (W)	699	10-CV	9	2
Credit (crx)	690	10-CV	15	2
DNA (nominal)	3,186	2,000/1,186	60	3
Glass	214	10-CV	9	6
Heart (Cleve)	303	10-CV	13	2
Ionosphere	351	10-CV	34	2
LED	3,200	200/3000	7	10
Letter	20,000	15,000/5,000	16	26
Pima Diabetes	768	10-CV	8	2
Satellite image	6,435	4,435/2,000	36	6
Shuttle	58,000	43,500/14,500	9	7
Soybean (large)	683	10-CV	35	19
Tic-Tac-Toe	958	10-CV	9	2
Waveform	5000	300/4700	21	3

Table 1: Characteristics of the data sets and the evaluation method. 10-CV denotes 10-fold cross-validation in our final study and 1/3-holdout in the initial study.

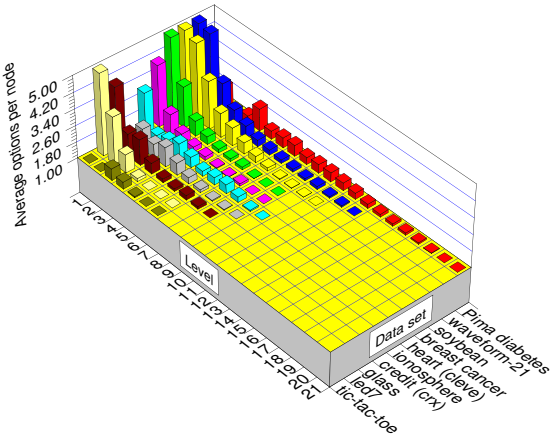

Figure 3: A plot of the average number of options generated per node (see text for definition). The files are sorted to make the display meaningful in 3D.

Breiman used in his study. In our initial study, we use only the smaller files and a single holdout (1/3-holdout when the table says 10-CV); we did this in order to understand how to control tree growth before using large data sets that generate very large trees. All algorithm variants use the same training sets and test sets.

3.3 The Simplest Option Decision Tree Algorithm

We begin with a simple version that creates an option node whenever there are three or more attributes that rate at least a multiplicative factor x, the **option factor**, of the best attribute on the evaluation criterion. We ran TDDT^{op} varying the option factor from 0.1 to 0.4, and imposed an arbitrary limit of five options per node to control the tree size. Using an option factor of 0.5 increased the run-times dramatically so we stopped at that point. Figure 2 shows a graph of the error rates and error ratios relative to the simple TDDT. The average absolute error decreased from 20.92% for TDDT to 19.70%, 19.66%, 18.20%, and 17.82% for the option factor settings of 0.1, 0.2, 0.3, and 0.4, respectively. The average relative error (the average of the relative error with respect to TDDT for each data set) was 0.95, 0.96, 0.90, and 0.87, respectively. Using 0.4 as the option factor produced huge trees; Soybean was the extreme case, which increased from 68 nodes to 203,577 nodes, a factor of almost 3,000.

3.4 Where are Option Nodes Most Useful?

Option nodes can, in principle, be created anywhere in the tree. We investigate where option nodes are created and where they are useful for reducing error.

On the one hand, the decision criteria commonly used to choose the attribute to split on (and the threshold for continuous attributes) are usually myopic and look only at the children nodes of the splits being considered. This would indicate that most criteria are not very reliable near the root of the tree; specifically, minor differences between two attributes may not be good predictors of how good the subtrees will be. Furthermore, researchers have shown (Ali 1996, Perrone 1993) that combining multiple models works best when the structures are not correlated or when they are anti-correlated. By making different splits closer to the root, the subtrees below them should be very different and hence will not correlate as much as would subtrees closer to the leaves.

On the other hand, making option nodes closer to the leaves spreads the risk when there are few instances and the evaluation criteria are unstable. Many of the significant improvements shown in Ali (1996, p. 141), for example, were shown for small data sets. Ali & Pazzani (1995) also wrote that "For learning tasks with few examples, greater classification accuracy can be achieved by learning several concept descriptions...." Learning multiple models seems to be more useful on small data sets.

We hypothesized that creating option nodes closer to the root would be more useful. We tested this hypothesis in three distinct ways:

1. We modified TDDT^{op} to induce option nodes only at the last three levels of the tree (three being an arbitrary cutoff). As in Brodley (1995), such a decision has to be done after a subtree is built and

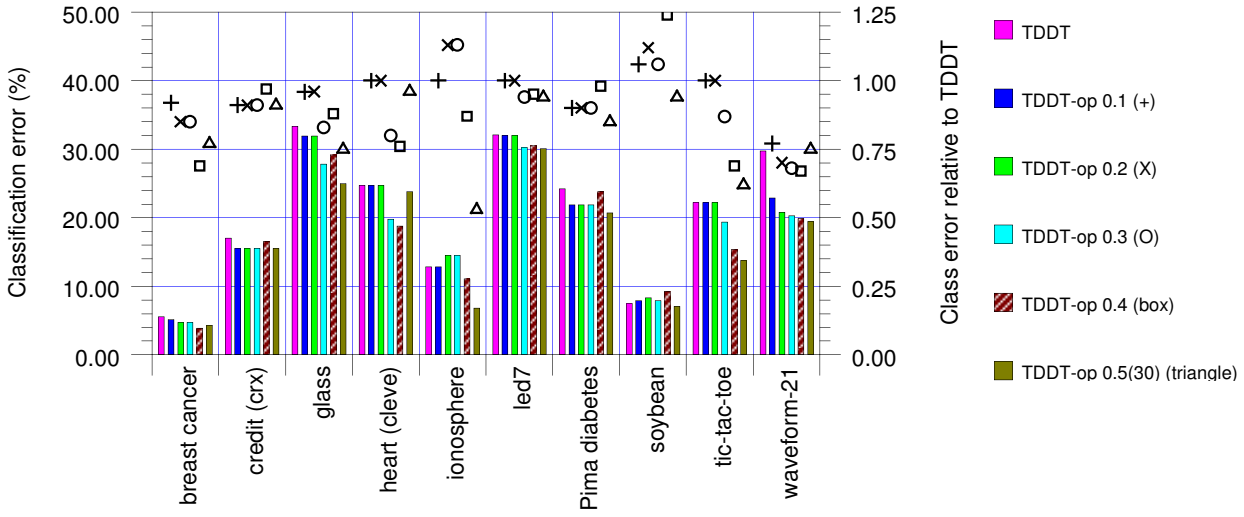

Figure 2: Absolute error shown in bars with the left axis defining the range; relative error rates (with respect to TDDT) shown using symbols with the right axis defining the range. Low bars are better and low relative errors (below 1) show improvement. TDDT^{op}-x denotes running with option factor of x.

pruned because we do not know in advance how deep a tree will be after pruning. The algorithm was modified such that during the pruning phase, each time a node is reached that is three levels above the leaves, we rebuild the subtree using the standard TDDT^{op} algorithm. The average error on the tested data sets was reduced from 20.92% to 20.49%, with the average relative error rate at 0.99. Some trees didn't grow much; Soybean grew by a factor of 43, but the overall error reduction was disappointing.

We modified the TDDT^{op} to induce option nodes only at the top levels. Specifically, we limited option nodes to the first two levels of the tree (option nodes do not count as a level in any experiments). Because the maximum number of options per node was set to five, this implied that the tree would be about 25 times larger than a standard decision tree. We therefore set the option factor to 0.95, effectively ignoring only options with very low evaluations (bottom 5%). The average error was reduced from 20.92% to 15.98%, with the average relative error rate at 0.79. We concluded from this experiment that option nodes at the top seem to be much more useful relative to the added complexity.

2. Figure 3 shows a histogram of the average number of choices per node. If the node is a decision node, it contributes one to the count. If the node is an option node, it contributes its number of options to the count. We then divide the count by the number of nodes that would have been formed if option nodes were not used. From the figure, we can see more options are created at the top few levels of the tree.

3. Define the *weighted error difference* between two nodes as the test-set difference in error on the two nodes multiplied by an importance factor. Figure 4 shows the difference in weighted test-set error between an option node and its first option (the option that would have been chosen if there was no option node) averaged over all nodes and all data sets. The importance factor was the training-set instance count. The motivation for this weighting is that big differences for a node with one training instance are less important than are small differences for nodes that were created using many instances. A negative factor indicates that the option node was (on average) worse than the best option. We can see that creating option nodes for nodes with few instances is generally of little help, and can even be harmful.

Given this support for our hypothesis about the usefulness of option nodes closer to the root, we restricted the creation of option nodes to cases where the number of instances is greater than 30, and reran the algorithm with an option factor of 0.5 (see Figure 2). This not only reduced the total number of nodes, but drastically reduced the running time since less option node activity took place near the bottom of the tree. The running time for Soybean reduced from over six hours to less than one and a half hours (a factor of four). The number of nodes for Pima Diabetes was reduced

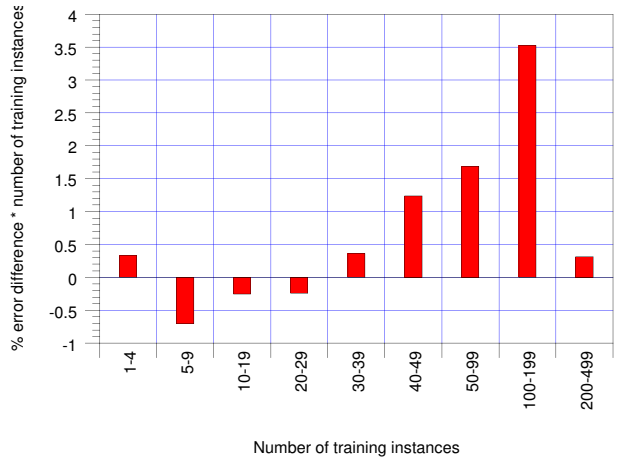

Figure 4: A histogram of the weighted error difference (see text) for nodes trained with different numbers of instances.

from 66,000 to 6807 (a factor of almost ten). The average error was reduced from 20.92% to 16.64% and the average relative error rate was 0.79.

It is interesting to note that our conclusions about option trees with majority votes are different than those of Buntine (1992*a*), who wrote:

> For nodes that have large counts, usually only one test is expanded because all others are insignificant according to the quality measure. With smaller samples... many tests may be almost as good according to the quality measure so many options will be expanded.

In our experiments, nodes with small counts (*i.e.*, leaves) do not have many values close to the best one, while nodes closer to the root do exhibit such behavior.

4 Pushing the Envelope

In the previous section, we arrived at the conclusion that option nodes near the top are more useful. However, the runs were still taking a long time, precluding a larger evaluation with 10-fold cross-validation and tests on bigger files. Limiting the number of levels from the top seemed like a good practical compromise. In addition, because increasing the option factor seemed to help reduce the error, we decided to increase the factor even more and use an exponentially decaying factor. Starting with an option factor of x, each level used a factor of $x \cdot y^{\text{level}}$ where y is the decay factor and the root is at level 0 (option nodes were ignored in the level count). We chose a decay factor of $y = 0.9$ and varied x. Option nodes were restricted to five levels, except for the data sets Letter, Satellite image, and Soybean, which were limited to three levels due to memory limitations.

Figure 5 shows the results for all files. In addition to TDDT, we compared against a bagged version of TDDT using 50 replicates.

The results show that an option factor of 1.0 was most useful (always making a five-way option node at the root). The average error rate decreased from 16.86% (for TDDT) to 12.08%; the average relative error was 0.70! As the option factor was reduced, the average error rate grew compared to the best one. At a factor of 0.8, the error rate was 12.54% and the average relative error was .72. Bagging had an error rate of 12.88% and an average relative error of 0.87 (Shuttle is very influential, though even without it, the average relative errors were 0.70 and 0.77 for an option factor of 1.0 and for Bagging). The main disadvantage of the Option Decision Trees generated is that they were huge, sometimes two or more orders of magnitude larger than those induced by TDDT. For example, Letter (the largest tree) had 276,978 nodes, while the TDDT tree had "only" 2167; in comparison, the 50 Bagged trees together had 97,550 nodes.

Although we are using more nodes as the option factor is increased, the error continued to decrease. Bagging, on the other hand, is limited and does not improve much if more samples are used. Breiman (1996, p. 135) wrote that "More than 25 bootstrap replicates is love's labor lost." When we increased the number of bootstrap samples to 250 to make the tree sizes comparable to ours, the average error went down only 0.18% to 12.70%.

One possible criticism is that our final results are reported on the same data sets we used to study the problem in the first place. The fact that we used a holdout set for accuracy estimation initially, then full cross-validation for those files in the final results (other than the artificial datasets with large test sets), mitigates the effect of overfitting to the test set. Furthermore, for the four files we did not use in our initial study (Letter, Shuttle, Satellite image, and DNA) the average relative error rate for the option factor of 1.0 was 0.62, even better than the overall average.

5 Comprehensible Option Decision Trees

In the final set of experiments, we decided to limit the number of nodes to an amount comparable to Bagging. We set the level limit to two and increased the maximum number of options allowed to seven. We denoted this version by TDDT$^{\text{op-7}^2}$. This variation should in-

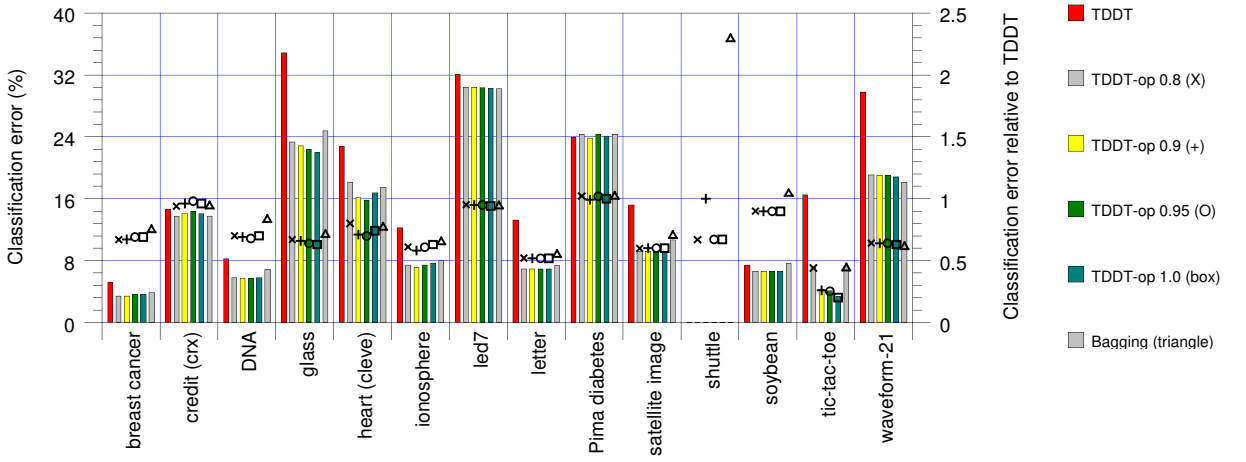

Figure 5: A plot of the absolute error rates shown in bars with the left axis defining the range and relative error rates (relative to TDDT) shown using symbols with the right axis defining the range. TDDT$^{\text{op}}$-x denotes running with option factor of x, Bagging denotes Breiman's bagging with 50 TDDT replicates. Shuttle has error rates in the range 0.01%-0.05% so it does not appear.

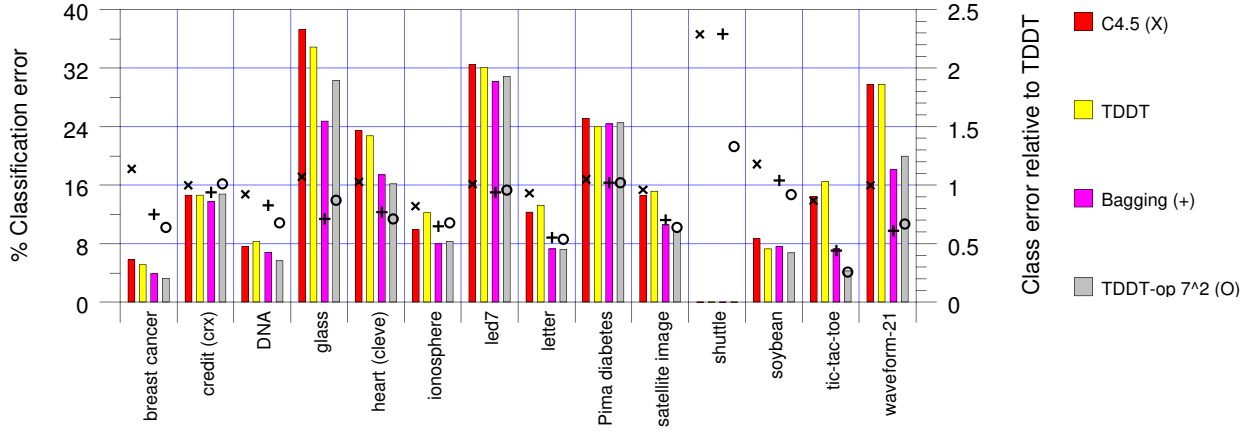

Figure 6: A plot of the absolute error rates and relative error rates (relative to TDDT) for different algorithms: C4.5, TDDT, Bagging, TDDT$^{\text{op}}$ with 7 options at each of the first two levels. The error rates for shuttle are very small: 3, 4, 7, 7 misclassifications for TDDT, TDDT$^{\text{op}}$, C4.5, and Bagging respectively (out of 14,500 test instances); differences are not significant at the 95% confidence level.

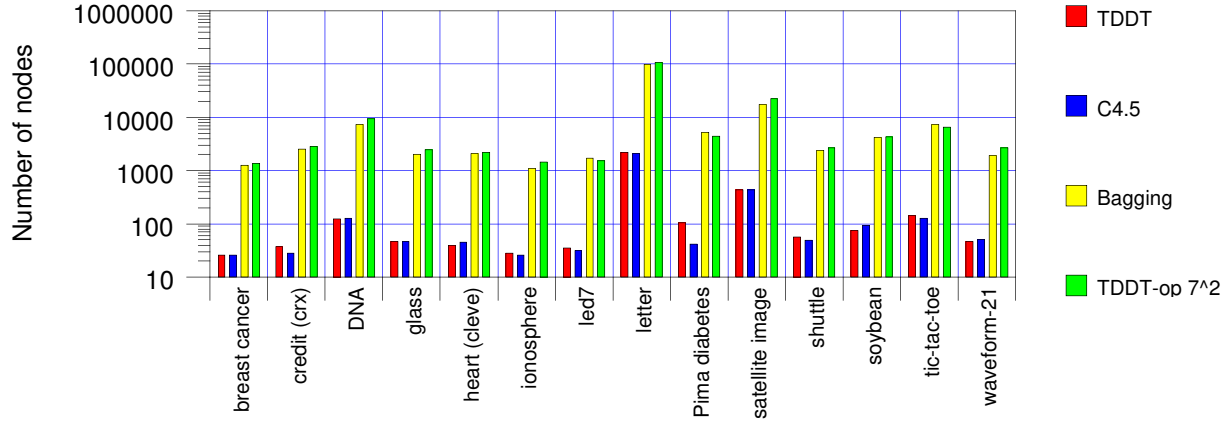

Figure 7: A plot of the tree sizes for different algorithms.

Data set	TDDT error	Bagging error	TDDT$^{\text{op-7}^2}$ error	TDDT-op 1.0 error	prob TDDT$^{\text{op-7}^2}$ better than TDDT	prob TDDT$^{\text{op-7}^2}$ better than Bagging	prob TDDT-op 1.0 better than Bagging
Breast cancer (W)	5.14±0.77	3.86±0.71	3.29±0.77	3.57±0.68	0.9553	0.7069	0.6160
Credit (crx)	14.64±0.63	13.77±0.95	14.78±1.18	14.06±1.01	0.4583	0.2525	0.4172
DNA (nominal)	8.26±0.80	6.83±0.73	5.65±0.67	5.82±0.68	0.9939	0.8827	0.8443
Glass	34.89±3.84	24.76±2.80	30.32±2.22	21.93±1.92	0.8486	0.0599	0.7977
Heart (Cleve)	22.75±2.89	17.46±2.93	16.16±3.17	16.78±3.61	0.9378	0.6184	0.5581
Ionosphere	12.21±2.37	7.99±1.27	8.26±0.90	7.70±1.13	0.9404	0.4311	0.5677
LED	32.07±0.85	30.20±0.84	30.87±0.84	30.30±0.84	0.8415	0.2875	0.4664
Letter	13.24±0.48	7.32±0.37	7.18±0.36	6.92±0.36	1.0000	0.6065	0.7817
Pima Diabetes	23.98±2.02	24.36±1.81	24.49±1.16	24.09±1.08	0.4133	0.4759	0.5510
Satellite image	15.20±0.80	10.65±0.69	9.70±0.66	9.15±0.65	1.0000	0.8398	0.9439
Shuttle	0.02±0.01	0.05±0.02	0.03±0.01	0.01±0.01	0.3521	0.8098	0.9506
Soybean (large)	7.32±1.11	7.62±1.17	6.74±0.90	6.59±0.85	0.6576	0.7245	0.7618
Tic-Tac-Toe	16.50±1.05	7.31±0.80	4.28±0.50	3.34±0.46	1.0000	0.9993	1.0000
Waveform	29.77±0.67	18.11±0.56	19.96±0.58	18.83±0.57	1.0000	0.0111	0.1829

Table 2: Error rates and probabilities that one algorithm is better than another based on a t-test. TDDT$^{\text{op-7}^2}$ is comparable to Bagging; TDDT-op 1.0 is usually better than Bagging.

duce trees that are about 49 times larger than the original TDDT tree (this may be more or less because the subtrees built are different). Figure 6 and Table 2 show the error rates, error ratios, and statistical significance of the results. Figure 7 shows the tree sizes.

The average relative error for the TDDT$^{\text{op-7}^2}$ was 12.98% and for bootstrap it was 12.88%. The average relative error rate was 0.78 for TDDT$^{\text{op-7}^2}$ and 0.874 for Bagging.

The tree sizes for both algorithms are about the same. We believe that the extra structure that TDDT$^{\text{op}}$ provides would be more useful for users, since it avoids replication and focuses the attention on uncertain nodes. Coupled with an interactive tree visualizer, the advantage of Option Decision Trees is that users could easily prune some subtrees whose root split they do not like, an idea proposed by Kwok & Carter (1990). Moreover, to choose a single tree, they would have to make only two choices (one out of seven at the root and one out of seven at the level below). Running times for TDDT$^{\text{op-7}^2}$ on an SGI Challenge ranged from 20 seconds to 38 minutes for Shuttle and an hour and three quarters for Letter.

6 Future Work

We chose to vote the children of option nodes because it is a simple method that users can understand. Weighted voting could be a useful alternative that might reduce the error further without complicating the tree structure excessively. Such weighted combinations were used, for example, by Oliver & Hand (1995) and in Bahl, Brown, de Souza & Mercer (1989). Another simple possibility is to allow subtrees to return "unknown" (abstain from the vote).

Option nodes, as implemented in TDDT$^{\text{op}}$, choose only different attributes while it may be useful to select the same attribute with different thresholds. While Oliver & Hand (1995) have done the same for their tree averaging, it is highly likely that bootstrap samples in Bagging *do* result in different thresholds for different trees, a characteristic we believe is advantageous.

It is possible to run a post-processing step to prune some option nodes and shrink the size of the tree. For example, it may be useful to choose one child from each option node based on its performance on a validation set. This will yield a regular decision tree that was built indirectly through an Option Decision Tree.

7 Conclusions

We showed that option nodes are more useful closer to the root using three different methodologies (modified algorithm, average choices per node, and weighted error histogram). We described several ways of controlling the growth of Option Decision Trees, so that when option nodes are created, they are useful in reducing the error rate. Kwok & Carter (1990) reported that after a few trees were generated, the accuracy slightly degraded because lower probability trees were used. In our experiments, as we allowed for more options (both by increasing the option factor and by increasing the

maximum number of options), the resulting trees produced fewer errors on average. Although we suspect that using too many options that rank lower in our evaluation criterion will hurt performance (because all options are weighted equally), we have not seen this effect.

Option Decision Trees provide users with control over the complexity versus the error for the resulting trees: the bigger the trees, the lower the error (on average), but with diminishing returns. As we pushed the envelope and created huge trees, the error continued to decrease.

As opposed to Bagging, the TDDT$^{\text{op}}$ algorithm is deterministic and does not rely on random bootstrap samples. It also uses all of the data to build the tree, while Bagging constructs multiple trees from samples that contain only a factor of about 0.632 unique instances from the training set. Unlike boosting (Freund & Schapire 1995), it is easy to parallelize Option Decision Tree induction.

Option Decision Trees provide a compact representation of many regular decision trees. The error reductions reported here are significant and large. The best average relative error reduction was 0.7, which is equivalent to an average of 30% reduction for the 14 data sets! For similar complexity to Bagging, the error rates are comparable, although Option Decision Trees provide a more structured classifier.

Acknowledgments

We thank Carla Brodley, Cliff Brunk, and Wray Buntine for their comments. The second author's work at Stanford University is funded by Silicon Graphics Inc. All experiments reported here were done using $\mathcal{MLC}$++.

References

Ali, K. M. (1996), Learning Probabilistic Relational Concept Descriptions, PhD thesis, University of California, Irvine. http://www.ics.uci.edu/~ali.

Ali, K. & Pazzani, M. (1995), 'HYDRA-MM: Learning multiple descriptions to improve classification accuracy', *International Journal on Artificial Intelligence Tools* **4**(1/2), 115–133.

Bahl, L. R., Brown, P. F., de Souza, P. V. & Mercer, R. L. (1989), 'A tree-based statistical language model for natural language speech recognition', *IEEE Transactions on Acoustics, Speech, and Signal Processing* **37**(7), 1001–1008.

Breiman, L. (1994), Heuristics of instability in model selection, Technical Report Statistics Department, University of California at Berkeley.

Breiman, L. (1996), 'Bagging predictors', *Machine Learning* **24**, 123–140.

Brodley, C. E. (1995), Automatic selection of split criterion during tree growing based on node location, *in* A. Prieditis & S. Russell, eds, 'Machine Learning: Proceedings of the Twelfth International Conference', Morgan Kaufmann, pp. 73–80.

Buntine, W. (1992*a*), 'Learning classification trees', *Statistics and Computing* **2**(2), 63–73.

Buntine, W. (1992*b*), A Theory of Learning Classification Rules, PhD thesis, University of Technology, Sydney, School of Computing Science.

Freund, Y. & Schapire, R. E. (1995), A decision-theoretic generalization of on-line learning and an application to boosting, *in* 'Proceedings of the Second European Conference on Computational Learning Theory', Springer-Verlag, pp. 23–37.

Kohavi, R., Sommerfield, D. & Dougherty, J. (1996), Data mining using $\mathcal{MLC}$++: A machine learning library in C++, *in* 'Tools with Artificial Intelligence', IEEE Computer Society Press, pp. 234–245. `http://www.sgi.com/Technology/mlc`.

Kwok, S. W. & Carter, C. (1990), Multiple decision trees, *in* R. D. Schachter, T. S. Levitt, L. N. Kanal & J. F. Lemmer, eds, 'Uncertainty in Artificial Intelligence', Elsevier Science Publishers, pp. 327–335.

Murthy, S. & Salzberg, S. (1995), Lookahead and pathology in decision tree induction, *in* C. S. Mellish, ed., 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann, pp. 1025–1031.

Oliver, J. & Hand, D. (1995), On pruning and averaging decision trees, *in* A. Prieditis & S. Russell, eds, 'Machine Learning: Proceedings of the Twelfth International Conference', Morgan Kaufmann, pp. 430–437.

Perrone, M. (1993), Improving regression estimation: averaging methods for variance reduction with extensions to general convex measure optimization, PhD thesis, Brown University, Physics Dept.

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

Stolfo, S. (1996), Integrating multiple learned models for improving and scaling machine learning algorithms. AAAI Workshop.