International Journal on Artificial Intelligence Tools
Vol. 18, No. 4 (2009) 613–620
© World Scientific Publishing Company

REDUCING DECISION TREE ENSEMBLE SIZE USING PARALLEL DECISION DAGS

ADAM H. PETERSON
Computer Science Department, Brigham Young University
Provo Utah 84602, United States
adam@axon.cs.byu.edu

TONY R. MARTINEZ
Computer Science Department, Brigham Young University
Provo Utah 84602, United States
martinez@cs.byu.edu

Received 1 December 2007
Accepted 4 December 2008

This research presents a new learning model, the Parallel Decision DAG (PDDAG), and shows how to use it to represent an ensemble of decision trees while using significantly less storage. Ensembles such as Bagging and Boosting have a high probability of encoding redundant data structures, and PDDAGs provide a way to remove this redundancy in decision tree based ensembles. When trained by encoding an ensemble, the new model behaves similar to the original ensemble, and can be made to perform identically to it. The reduced storage requirements allow an ensemble approach to be used in cases where storage requirements would normally be exceeded, and the smaller model can potentially execute faster by reducing redundant computation.

Keywords: Decision tree; graph; ensemble; bagging; boosting.

1. Introduction

A pioneering work in machine learning is the development of the ID3 decision tree learning algorithm by Quinlan.[1] This algorithm expanded on other work in decision trees, such as CART by Breiman *et al.*[2] Successor tree based algorithms such as C4.5[3] and random forests[4] have expanded the success of decision trees.

Although decision trees are powerful learners, it has been observed that they tend to have high variance. It is not uncommon for tree-based algorithms to be used in conjunction with combination techniques such as Bagging[5] or random forests.

Ensembles such as Bagging contain many individual models, consuming space resources, and each individual model must be evaluated during execution, impacting speed. This research presents the Parallel Decision DAG (PDDAG), which generalizes the concept of a decision tree to a rooted directed acyclic graph structure and can have multiple paths from the root node to each subtree. PDDAGs also have *parallel nodes*, which allow more than one decision path to be queried in finding the classification. This research shows how this extension allows the strengths of ensembles to be leveraged without the same overhead in storage.

The concept of using a graph structure instead of a decision tree has previously been explored by Oliver.[6] When inducing subtrees somewhat deep in the decision tree, the data can tend to become scarce as most of the training set is sectioned off in higher levels. This issue can impact large portions of the overall tree, especially at or near the leaf nodes. Oliver motivates the use of a directed acyclic graph to avoid data scarcity, using more data to induce subgraphs. Parallel Decision DAGs employ a *parallel node* not found in Oliver's model. This allows PDDAGs to evaluate more than one subgraph and provides a natural representation of Bagged hypotheses.

Work in pruning to reduce decision tree ensemble size has been performed by Margineantu and Dietterich.[7] In this work, the model was able to significantly reduce the size of the ensemble, while still maintaining about 80% of the improvement of Bagging. Domingos[8] gives a method for reducing the size of an ensemble of *C4.5 Rules* models by generating a new set of input vectors and classifying the data with the ensemble, and then using this data set to construct a new rule model. Zeng[9] continues this idea by approximating an ensemble with a neural network. The accuracy of the approximating neural network is only marginally lower than the Bagged ensemble. A method of reducing model size is also given by Menke *et al.*,[10] where any arbitrarily large model can be reduced in size by constructing a smaller neural network model trained to mimic the larger model. Such "oracle trained" models are significantly smaller, with only a marginal loss of accuracy.

The above size reduction methods each present an approach that significantly reduces model size while retaining varying levels of the accuracy of the original model. This research gives a method for producing a PDDAG model that is perfectly faithful to the original Bagged model's behavior while still giving significant size reduction. Alternatively, section 4 will show that PDDAG representations can reduce the size of an ensemble even further with essentially the same performance.

2. Parallel Decision DAGs

The core idea presented here is the Parallel Decision Directed Acyclic Graph (PDDAG). PDDAGs are a generalized concept of decision trees with added support for ensemble-type behavior. A decision tree takes as input a vector of attributes, which may be nominally-valued or real-valued. During classification, each tree node examines one attribute and uses its value to delegate the decision to one of its subtrees. A tree may be composed of three types of nodes:

(i) Nominal nodes, which branch on the value of a nominally-valued input and have a subtree for every possible value.
(ii) Real-valued nodes, which branch on a real-valued input relative to a threshold value and have two subtrees.
(iii) Leaf nodes, which simply return a classification decision.

During classification, the tree is traversed from the root node down to a single leaf node by examining a feature attribute at each branching node. The leaf node arrived at determines the pattern's classification. Decision trees may be induced using algorithms such as ID3 and C4.5.[3] Here, ID3 is used. Because ID3 tends to have high variance, Bagging ensembles are effective for improving accuracy.

Bagging is performed by taking the training set and resampling (with replacement) to form a new training set, from which a single model is induced. This is repeated until several individual models are obtained, which are then executed separately during classification, with each model casting a vote for the classification.

When constructing multiple trees for Bagging, some trees may end up performing redundant computations. To leverage these redundant structures, the trees are first composed into a single model, using a new type of tree node: the parallel node. A parallel node has one or more subtrees as children. During execution, all subtrees are evaluated and these decisions are combined using a plurality vote.

3. Packing Algorithms

By using a parallel node as the root of a tree, individual trees can be combined into a single model with the behavior of a decision tree ensemble. Nodes which perform the same function can be collapsed into a single node while preserving the overall behavior. This is done by traversing the tree depth-first and post-order, storing unique subtrees and removing redundant nodes. Two nodes are equivalent if:

- both are leaves which return the same classification, or
- both are nominal nodes which split on the same attribute and branch to matching subtrees, or
- both are real-valued nodes which split on the same attribute, have the same threshold, and branch to matching left and right subtrees.

This packing method (see algorithm 1) will be referred to as **basic** packing. Its worst case time complexity is $O(n \log n)$, in the number of tree nodes.

With redundant subtrees merged, it may be possible to improve the execution time of the tree. In a packed model, the same subtree may be queried multiple times for a single pattern. Tree nodes may have more than one parent, and parallel nodes cause more than one path through the model to be evaluated. By storing responses for such subtrees as they are evaluated, they need only be evaluated once per pattern. In a Bagged model, such redundant trees would have to be re-evaluated within each component model in which they occur.

The ability to collapse redundant subtrees is somewhat hampered in trees with real-valued splits because of the requirement that the real-valued threshold match exactly. Although such matches do occur, various conditions such as round-off error, inexact representations, and varying orders of operations can result in real-valued thresholds which are very close to each other. Although a threshold may not be an exact match, the effective difference may have a negligible impact on accuracy.

Algorithm 1 Constructing a PDDAG from an ensemble model.

```
    function MAKE_PDDAG($D$)                     ▷ Construct a PDDAG from data $D$
        $N$:=$\emptyset$, $M$:=$\varnothing$    ▷ Initialize the final model $M$ and the set of seen nodes $N$
        for all $B_i \in$ bag($D$) do                              ▷ For each Bagged tree
            $M_i$:=DFS_traverse($B_i, N$)  ▷ Collapse the tree and add it to the model
5:      return $M$      ▷ Return the model, a list of root nodes with shared children

    function DFS_TRAVERSE($B_i, N$)                   ▷ Traverse tree $B_i$ depth-first
        $R$:=root($B_i$)                                  ▷ Find the root node of $B_i$
        for all $C_i \in$ children($R$) do               ▷ Replace each child tree ...
10:         $C_i$:=DFS_traverse($C_i, N$)  ▷ with a shared equivalent node, if available
        if type($R$)='leaf' then                   ▷ Find an equivalent leaf node in $N$
            $n$:=find($n \in N$ where type($n$)='leaf' and class($n$)=class($R$) )
        else if type($R$)='nom-spl' then  ▷ ... Or an equivalent nominal-split node
            $n$:=find($n \in N$ where type($n$)='nom-spl' and feature($n$)=feature($R$)
    and children($n$)=children($R$) )
15:     else if type($R$)='rl-spl' then  ▷ ... Or an equivalent real-valued split node
            $n$:=find($n \in N$ where type($n$)='rl-spl' and feature($n$)=feature($R$) and
    threshold($n$)=threshold($R$) and children($n$)=children($R$) )
        if $n = \varnothing$ then                                  ▷ If $R$ isn't found in $N$
            $N \leftarrow R$                                            ▷ Add $R$ to $N$
            return $R$                                             ▷ ... and return it
20:     else                                                          ▷ Otherwise
            return $n$                          ▷ ... return the equivalent node from $N$
```

The requirement for an exact threshold match on real-valued nodes may be relaxed to use a heuristic match instead during packing. This research explores two such heuristics:

- **Conservative:** There are no training instances in the training set which fall between the two thresholds.
- **Mapped Class:** Of all the instances processed by either node, only one output class is represented by patterns falling between the thresholds.

When two real-value split nodes are merged using one of these heuristics, the threshold of the new node is a weighted average of the thresholds of the original nodes, with each threshold weighted according to how many parents it has.

4. Results and Analysis

To show the utility of packing Bagged ensembles, experiments were run on fifteen problems in the Irvine Machine Learning Repository.[11] These tasks were selected

Table 1. The average size in nodes for the tree ensemble and the reduction algorithm with perfect fidelity. The accuracy figures show the mean accuracy, and also the standard deviation preceded by a plus/minus sign.

Dataset	Accuracy	Original	Packed	Relative size
balanc	75.1 ± 2.3	26890.6	1806.4	6.7%
bupa	68.3 ± 3.6	5318.8	2173.1	40.9%
iono	91.8 ± 3.6	1769.2	735.0	41.5%
iris	96.1 ± 1.2	824.0	198.5	24.1%
lymph	80.1 ± 4.0	4706.3	419.7	8.9%
musk1	84.7 ± 3.1	3358.4	1542.9	45.9%
newthy	93.6 ± 2.3	1085.0	335.5	30.9%
pima	76.8 ± 0.9	9072.2	3846.4	42.4%
segm	96.7 ± 0.8	7051.6	2775.3	39.4%
sonar	74.3 ± 5.4	1570.8	701.8	44.7%
sthear	80.2 ± 4.4	3918.6	957.2	24.4%
stvehi	73.8 ± 1.8	10304.8	4271.1	41.4%
vowel	85.3 ± 2.3	8999.0	3867.6	43.0%
wine	94.2 ± 2.1	866.2	281.4	32.5%
zoo	88.9 ± 7.7	1691.0	132.0	7.8%

to cover a variety of tasks. The *balanc*, *lymph*, *sthear*, and *stvehi* task have nominal features, while all other tasks are real-valued, and *sthear* has both feature types.

In each set of experiments, a Bagged model containing 100 decision trees is constructed and then packed using one of six packing heuristics. The accuracies and relative sizes of the packed model and original bagged model are compared. 5x2 cross-validation is used, as recommended by Dietterich.[12] That is, ten measurement samplings are obtained for each task by running twofold cross-validation five times.

The first packing method tested is the exact mapping wherein nodes are only collapsed if they are exactly equivalent. This is a purely representational transformation so accuracy numbers are not reported, as they are identical to the original Bagged model. As Table 1 shows, the representation for every learning task has significantly fewer nodes in the packed model than in the original Bagged model. In every case, the node count for the packed model is less than 50% of the size of the original, and in a few cases, the model's node count is reduced to as little as 7% of the original.

A large amount of the size reduction in the Bagged models is a direct consequence of collapsing the leaf nodes. For any decision tree, the leaf nodes will outnumber the internal nodes of the tree. In many decision tree models, there is no need for more leaf nodes than there are classes, so most of these leaf nodes are redundant. By collapsing these leaf nodes alone, significant model size reduction can occur, and this is leveraged to the fullest extent in a PDDAG since all individual models may share the same leaf nodes. The realization that pooling leaf nodes can reduce model size significantly may by itself motivate the use of graph structures for decision tree learning.

The savings in node count are most significant in the three tasks *balanc*, *lymph*, and *zoo*, which have only nominal inputs. In learning tasks where the values given in input each take on one of several values, the likelihood of finding equivalent nodes appears empirically to be relatively high (as evidenced by the relative size drop for *balanc*, *lymph*, and *zoo*), resulting in significant space savings when removing this redundancy. Empirically, it appears that when a problem domain has nominally valued inputs, use of PDDAGs for Bagging is a clear and significant win for model space reduction.

Exact matches can be less likely for real-valued split nodes. Real valued thresholds are chosen by picking the midpoint between two training instances. Because the resampling of Bagging can cause any training instance to be either present or absent, chosen midpoint values can vary between models which are given a different sampling of the training instances. This results in real-valued node equivalence being much less likely than nominally valued node equivalence.

Despite the difficulty in general of meeting real-valued equivalence criteria, some of the tasks with real-valued input attributes (although not exhibiting the extreme size reduction of the three nominally valued tasks) still show significant size reductions. Notable among these are *iris*, *newthy*, and *sthear*, each showing size reductions in excess of 2/3. In general, though, the real-valued splits in many problems cause exact matches in nodes to be less common. The experiments below explore reducing nodes which, although not exactly equivalent, have relatively small differences in function. These results are given in Table 2. (Tasks which have no real-valued inputs are omitted.)

The first of these packing strategies explored for real-valued inputs is conservative packing. Rather than requiring the thresholds of two real-valued split nodes to be identical, two nodes may be considered equal if their thresholds are so close that no training instances fall between them. The middle columns of Table 2 summarize the results for these models, comparing them with the original Bagged model. In these results, it can be observed that the model sizes are generally improved by

Table 2. Accuracy and relative size of packed models, using conservative and mapped-class packing.

	Basic packing		Conservative packing		Mapped class packing	
Dataset	Accuracy	Size	Accuracy	Size	Accuracy	Size
bupa	68.3 ± 3.6	40.9%	68.3 ± 3.6	39.0%	68.3 ± 3.5	36.4%
iono	91.8 ± 3.6	41.5%	91.9 ± 3.6	39.8%	91.9 ± 3.4	35.3%
iris	96.1 ± 1.2	24.1%	96.1 ± 1.2	20.1%	96.3 ± 1.5	17.1%
musk1	84.7 ± 3.1	45.9%	84.7 ± 3.1	44.5%	84.6 ± 3.0	41.1%
newthy	93.6 ± 2.3	30.9%	93.6 ± 2.4	28.3%	93.3 ± 2.1	22.9%
pima	76.8 ± 0.9	42.4%	76.8 ± 0.9	40.7%	76.8 ± 1.0	37.6%
segm	96.7 ± 0.8	39.4%	96.7 ± 0.8	38.4%	96.7 ± 0.8	31.4%
sonar	74.3 ± 5.4	44.7%	74.3 ± 5.4	42.6%	73.9 ± 5.1	38.4%
sthear	80.2 ± 4.4	24.4%	80.2 ± 4.4	23.2%	80.3 ± 3.9	20.7%
stvehi	73.8 ± 1.8	41.4%	73.8 ± 1.8	39.9%	73.8 ± 1.7	37.7%
vowel	85.3 ± 2.3	43.0%	85.3 ± 2.4	41.1%	85.3 ± 2.1	34.9%
wine	94.2 ± 2.1	32.5%	94.2 ± 2.4	29.4%	94.4 ± 2.7	21.7%

between one and two percent of the original model's size, with one case (*iris*) as high as four percent. More interesting, though, is that the accuracy measurements are extremely close between the two sets (and well within a standard deviation). In the only instance (*iono*) where there is a visible difference in the mean, the conservatively packed model's accuracy is slightly higher than the original Bagged model.

The second method of packing explored, mapped class packing, allows patterns to fall between the two threshold values of the candidate nodes as long as those processed by either of the original nodes all belong to the same output class. These results are in the right most columns of Table 2. For the most part, accuracy values track fairly well with their Bagging counterparts, although variation is more pronounced than with conservative packing. Size reductions tend to be between 3% and 8% improvement in size over conservative packing.

Although this research has only shown packing Bagged models, any model that takes the form of a collection of trees with a weighted voting for the top level decision may be packed as a PDDAG model. For this research, experiments were also performed with AdaBoost[13] models and the same datasets. In summary, the relative size improvement was greater for a Boosted ensemble than it was for the corresponding Bagged ensemble for eleven of the fifteen examples. This result is reasonable when one considers the difference in the way Bagged and Boosted ensembles are constructed. In Bagging, a new component model is constructed from a new and independent resampling of the training set. Thus, each component model is relatively independent from the others with respect to the original training set. With Boosting, on the other hand, the training set is resampled for each new model based on the previous performance on the patterns. As the Boosted models struggle to learn some subset of points, the resampled training sets will tend to be correlated in including these patterns, leading to a correlation in the structures of the models. These correlated structures are more likely to include repeated subtrees that can be packed in a PDDAG, leading to greater size reduction in a PDDAG representation.

5. Conclusion and Future Work

Three procedures and variants were explored for removing redundant tree structures from the ensemble. For nominal-input valued tasks and some real-valued tasks, significant space savings were shown without any effect on the accuracy or behavior of the model, using the most conservative packing method. For some real-valued tasks, the space savings using this transformation were less, and much of that savings can be attributed to the removal of redundant leaf nodes.

When packing Bagging models for real-valued trees, All packing methods tend to give accuracy comparable to Bagging, while taking up significantly less space.

This research has shown the use of a parallel rooted graph structure to leverage the principles behind ensembles such as Bagging and Boosting for decision trees. The packing methods in this research are relatively conservative, with one of the methods effecting no change in the original ensemble behavior at all. It is likely that other transformations and packing methods exist which should be more fully explored in future work.

The authors are also exploring methods of training PDDAGs directly without using ensembles as an intermediary. A direct training algorithm would save the initial cost of first constructing an ensemble model. It would also be able to use parallel nodes in other locations of the tree beyond the root node. This can be useful because a parallel root node is the most expensive of parallel nodes, impacting all patterns classified. Additionally, there may be other portions of the tree which may benefit from formulating multiple alternate models.

Parallel nodes in a decision tree also provide a natural way for decision trees to output probabilities over all classes rather than a single classification. The voting mechanism in the decision of a parallel node can be configured to pass a probability over each classification rather than a single decision. The ability to output probabilities can increase the usefulness of decision tree classifiers in utility theory.

References

1. J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
2. Leo Breiman, Jerome H. Freidman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.
3. J. Ross Quinlan. *C4.5: Programs For Machine Learning*. Morgan Kaufmann Publishers, Inc., 1993.
4. Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
5. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
6. J. J. Oliver. Decision graphs - an extension of decision trees. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pages 343–350, 1993. Extended version available as TR 173, Department of Computer Science, Monash University, Clayton, Victoria 3168, AUSTRALIA.
7. Dragos D. Margineantu and Thomas G. Dietterich. Pruning adaptive boosting. In *Proceedings of the 14th International Conference on Machine Learning*, pages 211–218. Morgan Kaufmann, 1997.
8. Pedro Domingos. Knowledge acquisition from examples via multiple models. In *Proceedings of the 14th International Conference on Machine Learning*, pages 98–106. Morgan Kaufmann, 1997.
9. Xinchuan Zeng and Tony R. Martinez. Using a neural network to approximate an ensemble of classifiers. *Neural Processing Letters*, 12:225–237, 2000.
10. Joshua Menke, Adam H. Peterson, Michael E. Rimer, and Tony R. Martinez. Neural network simplification through oracle learning. In *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'02*, pages 2482–2497. IEEE Press, 2002.
11. Christopher J. Merz and Patrick M. Murphy. UCI repository of machine learning databases. Available at http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1996.
12. Thomas G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.
13. Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.