Additive Groves of Regression Trees

Daria Sorokina, Rich Caruana, and Mirek Riedewald

Department of Computer Science, Cornell University, Ithaca, NY, USA
{daria,caruana,mirek}@cs.cornell.edu

Abstract. We present a new regression algorithm called Groves of trees and show empirically that it is superior in performance to a number of other established regression methods. A Grove is an additive model containing a small number of large trees. Trees added to the Grove are trained on the residual error of other trees already in the Grove. We begin the training process with a single small tree and gradually increase both the number of trees in the Grove and their size. This procedure ensures that the resulting model captures the additive structure of the response. A single Grove may still overfit to the training set, so we further decrease the variance of the final predictions with bagging. We show that in addition to exhibiting superior performance on a suite of regression test problems, bagged Groves of trees are very resistant to overfitting.

1 Introduction

We present a new regression algorithm called Grove, an ensemble of additive regression trees. We initialize a Grove with a single small tree. The Grove is then gradually expanded: on every iteration either a new tree is added, or the trees that already are in the Grove are made larger. This process is designed to try to find the simplest model (a Grove with the fewest number of small trees) that captures the underlying additive structure of the target function. As training progesses, this algorithm yields a sequence of Groves of slowly increasing complexity. Eventually, the largest Groves may begin to overfit the training set even as they continue to learn important additive structure. This overfitting is reduced by applying bagging on top of the Grove learning process.

In Section 2 we describe the Grove algorithm step by step, beginning with the classical way of training additive models and incrementally making this process more complicated – and better performing – at each step. In Section 3 we compare bagged Groves with two other regression ensembles: bagged regression trees and stochastic gradient boosting. The results show that bagged Groves outperform these other methods and work especially well on highly non-linear data sets. In Section 4 we show that bagged Groves are resistant to overfitting. We conclude and discuss future work in Section 5.

2 Algorithm

Bagged Groves of Trees, or bagged Groves for short, is an ensemble of regression trees. Specifically, it is a bagged additive model of regression trees where each individual additive model is trained in an adaptive way by gradually increasing both number of trees and their complexity.

Regression Trees. The unit model in a Grove is a regression tree. Algorithms for training regression trees differ in two major aspects: (1) the criterion for choosing the best split in a node and (2) the way in which tree complexity is controlled. We use trees that optimize RMSE (root mean squared error) and we control tree complexity (size) by imposing a limit on the size (number of cases) at an internal node. If the fraction of the data points that reach a node is less than a specified threshold α, then the node is declared a leaf and is not split further. Hence the smaller α, $0 \leq \alpha \leq 1$, the larger the tree. (See Figure 7.)

Note that because we will later bag the tree models, the specific choice of regression tree is not particularly important. The main requirement is that the complexity of the tree should be controllable.

2.1 Additive Models — Classical Algorithm

A Grove of trees is an additive model where each additive term is represented by a regression tree. The prediction of a Grove is computed as the sum of the predictions of these trees: $F(\mathbf{x}) = T_1(\mathbf{x}) + T_2(\mathbf{x}) + \cdots + T_N(\mathbf{x})$. Here each $T_i(\mathbf{x})$, $1 \leq i \leq N$, is the prediction made by the i-th tree in the Grove. The Grove model has two main parameters: N, the number of trees in the Grove, and α, which controls the size of each individual tree. We use the same value of α for all trees in a Grove.

In statistics, the basic mechanism for training an additive model with a fixed number of components is the *backfitting algorithm* [1]. We will refer to this as the Classical algorithm for training a Grove of regression trees (Algorithm 1).

The algorithm cycles through the trees until the trees converge. The first tree in the Grove is trained on the original data set, a set of training points $\{(\mathbf{x}, y)\}$. Let $\hat{T}_1$ denote the function encoded by this tree. Then we train the second tree, which encodes $\hat{T}_2$, on the residuals, i.e., on the set $\{(\mathbf{x}, y - \hat{T}_1(\mathbf{x}))\}$. The third tree then is trained on the residuals of the first two, i.e., on $\{(\mathbf{x}, y - \hat{T}_1(\mathbf{x}) - \hat{T}_2(\mathbf{x}))\}$, and so on.

After we have trained N trees this way, we *discard* the first tree and retrain it on the residuals of the other $N - 1$ trees, i.e. on the set $\{(\mathbf{x}, y - \hat{T}_2(\mathbf{x}) - \hat{T}_3(\mathbf{x}) - \cdots - \hat{T}_N(\mathbf{x}))\}$. Then we similarly discard and retrain the second tree, and so on. We keep cycling through the trees in this way until there is no significant improvement in the RMSE on the training set.

Bagging. As with single decision trees, a single Grove tends to overfit to the training set when the trees are large. Such models show a large variance with respect to specific subsamples of the training data and benefit significantly from bagging, a well-known procedure for improving model performance by reducing variance [2]. On each iteration of bagging, we draw a bootstrap sample (bag) from the training set, and train the full model (in our case a Grove of additive trees) from that sample. After repeating this procedure a number of times, we

Algorithm 1 Classical additive model training

```
function Classical($\alpha$,$N$,{$\mathbf{x}$,y})
  for $i = 1$ to $N$ do
    $\text{Tree}_i^{(\alpha,N)} = 0$
  Converge($\alpha$,$N$,{$\mathbf{x}$,y}, $\text{Tree}_1^{(\alpha,N)}, \ldots, \text{Tree}_N^{(\alpha,N)}$)

function Converge($\alpha$,$N$,{$\mathbf{x}$,y},$\text{Tree}_1^{(\alpha,N)}, \ldots, \text{Tree}_N^{(\alpha,N)}$)
  repeat
    for $i = 1$ to $N$ do
      $\text{newTrainSet} = \{\mathbf{x}, y - \sum_{k \neq i} \text{Tree}_k^{(\alpha,N)}(\mathbf{x})\}$
      $\text{Tree}_i^{(\alpha,N)} = \text{TrainTree}(\alpha, \text{newTrainSet})$
  until (change from the last iteration is small)
```

end up with an ensemble of models. The final prediction of the ensemble on each test data point is an average of the predictions of all models.

Example. In this section we illustrate the effects of different methods of training bagged Groves on synthetic data. The synthetic data set was generated by a function of 10 variables that was previously used by Hooker [3].

$$F(x) = \pi^{x_1 x_2}\sqrt{2x_3} - sin^{-1}(x_4) + log(x_3 + x_5) - \frac{x_9}{x_{10}}\sqrt{\frac{x_7}{x_8}} - x_2 x_7 \qquad (1)$$

Variables x_1, x_2, x_3, x_6, x_7, x_9 are uniformly distributed between 0.0 and 1.0 and variables x_4, x_5, x_8 and x_{10} are uniformly distributed between 0.6 and 1.0.[1]

Figure 1 shows a contour plot of how model performance depends on both α, the size of tree, and N, the number of trees in a Grove, for 100 bagged Groves trained with the classical method on 1000 training points from the above data set. The performance is measured as RMSE on an independent test set consisting of 25,000 points. Notice that lower RMSE implies better performance. The bottom-most horizontal line for $N = 1$ corresponds to bagging single trees. The plot clearly indicates that by introducing additive model structure, with $N > 1$, performance improves significantly. We can also see that the best performance is achieved by Groves containing 5-10 relatively small trees (large α), while for larger trees performance deteriorates.

2.2 Layered Training

When individual trees in a Grove are large and complex, the Classical additive model training algorithm (Section 2.1) can overfit even if bagging is applied. Consider the extreme case $\alpha = 0$, i.e., a Grove of full trees. The first tree will perfectly model the training data, leaving residuals with value 0 for the other

[1] Ranges are selected to avoid extremely large or small function values.

Algorithm 2 Layered training

```
function Layered($\alpha$,$N$,train)
  $\alpha_0 = 0.5, \alpha_1 = 0.2, \alpha_2 = 0.1, \ldots, \alpha_{\max} = \alpha$
  for $j = 0$ to max do
    if $j = 0$ then
      for $i = 1$ to $N$ do
        $\text{Tree}_i^{(\alpha_0,N)} = 0$
    else
      for $i = 1$ to $N$ do
        $\text{Tree}_i^{(\alpha_j,N)} = \text{Tree}_i^{(\alpha_{j-1},N)}$
    Converge($\alpha_j$,$N$,train,$\text{Tree}_1^{(\alpha_j,N)}, \ldots, \text{Tree}_N^{(\alpha_j,N)}$)
```

trees in the Grove. Hence the intended Grove of several large trees will degenerate to a single tree.

One could address this issue by limiting trees to very small size. However, we still would like to be able to use large trees in a Grove so that we can capture complex and non-linear functions. To prevent the degeneration of the Grove as the trees become larger, we developed a "layered" training approach. In the first round we grow N small trees. Then in later cycles of discarding and re-training the trees in the Grove we gradually increase tree size.

More precisely, no matter what the value of α, we *always* start the training process with small trees, typically using a start value $\alpha_0 = 0.5$. Let α_j denote the value of the size parameter after j iterations of the Layered algorithm (Algorithm 2). After reaching convergence for α_{j-1}, we increase tree complexity by setting α_j to approximately half the value of α_{j-1}. We continue to cycle through the trees, re-training all trees in the Grove in the usual way, but now allow them to reach the size correspondent to the new larger α_j, and as before, we proceed until the Grove converges on this layer. We keep gradually increasing tree size until $\alpha_j \approx \alpha$.

For a training set with 1000 data points and $\alpha = 0$, we use the following sequence of values of α_j: $(0.5, 0.2, 0.1, 0.05, 0.02, 0.01, 0.005, 0.002, 0.001)$. It is worth noting that while training a Grove of large trees, we automatically obtain all Groves with the same N for all smaller tree sizes in the sequence. Figure 2 shows how 100 bagged Groves trained by the layered approach perform on the synthetic data set. Overall performance is much better than for the classical algorithm and bagged Groves of N large trees now perform at least as well as bagged Groves of N smaller trees.

2.3 Dynamic Programming Training

There is no reason to believe that the best (α, N) Grove should always be constructed from a $(\approx 2\alpha, N)$ Grove. In fact, a large number of small trees might overfit the training data and hence limit the benefit of increasing tree size in later iterations. To avoid this problem, we need to give the Grove training algorithm additional flexibility in choosing the right balance between increasing tree size and the number of trees. This is the motivation behind the *Dynamic Programming* Grove training algorithm.

This algorithm can choose to construct a new Grove from an existing one by either adding a new tree (while keeping tree size constant) or by increasing tree size (while keeping the number of trees constant). Considering the parameter grid, the Grove for a grid point (α_j, n) could be constructed either from its left neighbor (α_{j-1}, n) or from its lower neighbor $(\alpha_j, n-1)$. Pseudo-code for this approach is shown in Algorithm 3. We make a choice between the two options for computing each Grove (adding another tree or making the trees larger) in a greedy manner, i.e., the one that results in better performance of the Grove on the validation set. We use the out-of-bag data points [4] as the validation set for choosing which of the two Groves to use at each step.

Figure 3 shows how the Dynamic Programming approach improves bagged Groves over the layered training. Figure 4 shows the choices that are made during the process: it plots the average difference between RMSE of the Grove created from the lower neighbor (increase n) and performance of the Grove created from the left neighbor (decrease α_j). That is, a negative value means that the former is preferred, while a positive value means that the latter is preferred at that grid point. We can see that for this data set increasing the tree size is the preferred direction, except for cases with many small trees.

This dynamic programming version of the algorithm does not explore all possible sequences of steps to build a Grove of trees, because we require that every grove built in the process should contain trees of equal size. We have tested several other possible approaches that don't have this restriction, but they failed to produce any improvements and were noticeably worse from the running time point of view. For these reasons we prefer the dynamic programming version over other, less restricted options.

2.4 Randomized Dynamic Programming Training

Our bagged Grove training algorithms so far performed bagging in the usual way, i.e., create a bag of data, train all Groves for different vallues of (α, N) on that bag, then create the next bag, generate all models on this bag; and so on for 100 different bags. When the Dynamic Programming algorithm generates a Grove using the same bag, i.e., the same train set that was used to generate its left and lower neighbors, complex models might not be very different from their neighbors because those neighbors already might have overfitted and there is not enough training data to learn anything new. We can address this problem by using a different bag of data on every step of the Dynamic Programming algorithm, so that every Grove has some new data to learn from. While performance of a single Grove might become worse, performance of bagged Groves improves due to increased variability in the models. Figure 5 shows the improved performance of this final version of our Grove training approach. The most complex Groves are now performing worse than their left neighbors with smaller trees. This happens because those models need more bagging steps to converge to their best quality. Figure 6 shows the same plot for bagging with 500 iterations where the property

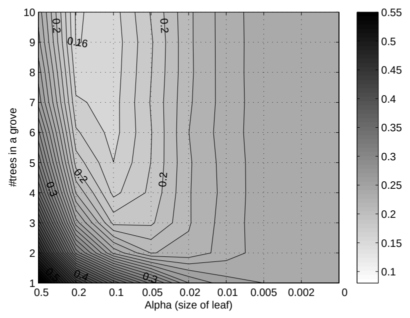

Fig. 1. RMSE of bagged Grove, Classical algorithm

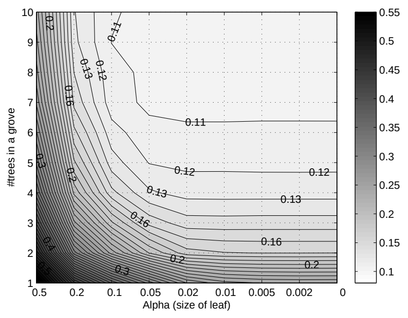

Fig. 2. RMSE of bagged Grove, Layered algorithm

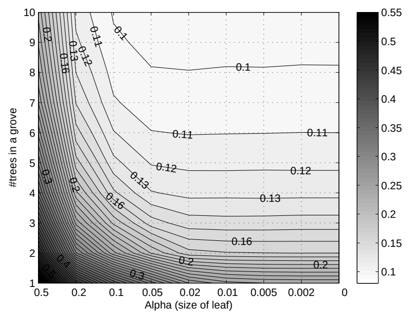

Fig. 3. RMSE of bagged Grove, Dynamic Programming algorithm

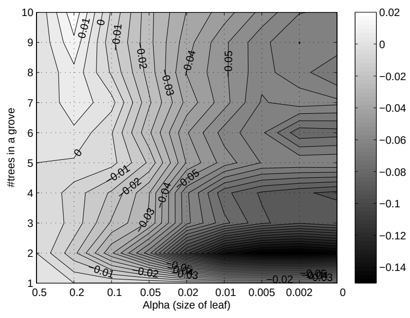

Fig. 4. Difference in performance between "horizontal" and "vertical" steps

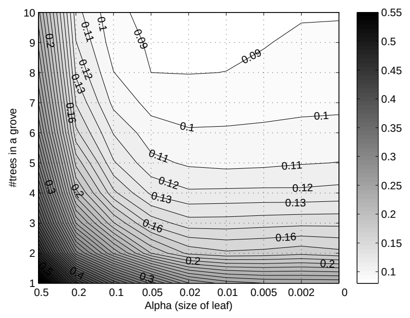

Fig. 5. RMSE of bagged Grove (100 bags), Randomized Dynamic Programming algorithm

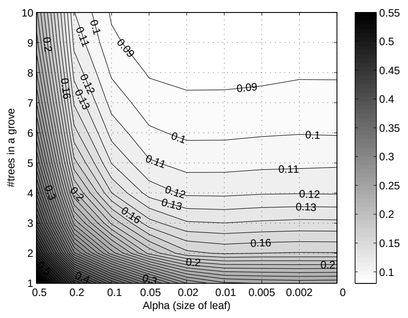

Fig. 6. RMSE of bagged Grove (**500 bags**), Randomized Dynamic Programming algorithm

Algorithm 3 Dynamic Programming Training

```
function DP($\alpha$,$N$,trainSet)
  $\alpha_0 = 0.5, \alpha_1 = 0.2, \alpha_2 = 0.1, \ldots, \alpha_{\max} = \alpha$
  for $j = 0$ to max do
    for $n = 1$ to $N$ do

      for $i = 1$ to $n - 1$ do
        $\text{Tree}_{\text{attempt1},i} = \text{Tree}_i^{(\alpha_j, n-1)}$
      $\text{Tree}_{\text{attempt1},n} = 0$
      Converge($\alpha_j$,$n$,train,$\text{Tree}_{\text{attempt1},1}, \ldots, \text{Tree}_{\text{attempt1},n}$)

      if $j > 0$ then
        for $i = 1$ to $n$ do
          $\text{Tree}_{\text{attempt2},i} = \text{Tree}_i^{(\alpha_{j-1}, n)}$
        Converge($\alpha_j$,$n$,train,$\text{Tree}_{\text{attempt2},1}, \ldots, \text{Tree}_{\text{attempt2},n}$)

      winner = Compare $\sum_i \text{Tree}_{\text{attempt1},i}$ and $\sum_i \text{Tree}_{\text{attempt2},i}$ on validation set
      for $i = 1$ to $n$ do
        $\text{Tree}_i^{(\alpha_j, n)} = \text{Tree}_{\text{winner},i}$
```

“more complex models are at least as good as their less complex counterparts” is restored.

3 Experiments

We evaluated bagged Groves of trees on 2 synthetic and 5 real-world data sets and compared the performance to two other regression tree ensemble methods that are known to perform well: stochastic gradient boosting and bagged regression trees. Bagged Groves consistently outperform both of them. For real data sets we performed 10 fold cross validation: for each run 8 folds were used as a training set, 1 fold as a validation set for choosing the best set of parameters and the last fold was used as the test set for measuring performance. For the two synthetic data sets we generated 30 blocks of data containing 1000 points each and performed 10 runs using different blocks for training, validation and test sets. We report mean and standard deviation of the RMSE on the test set. Table 1 shows the results; for comparability across data sets all numbers are scaled by the standard deviation of the response in the dataset itself.

3.1 Parameter Settings

Groves. We trained 100 bagged Groves using the Randomized Dynamic Programming technique for all combinations of parameters N and α with $1 \leq N \leq 15$ and $\alpha \in \{0.5, 0.2, 0.1, 0.05, 0.02, 0.01, 0.005\}$. Notice that with these settings the resulting ensemble can consist of at most 1500 trees. From these models we selected the one that gave the best results on the validation set. The performance of the selected Grove on the test set is reported.

	California Housing	Elevators	Kinematics	Computer Activity	Stock	Synthetic No Noise	Synthetic Noise
Bagged Groves							
RMSE	0.38	0.309	0.364	0.117	0.097	0.087	0.483
StdDev	0.015	0.028	0.013	0.0093	0.029	0.0065	0.012
Boosting							
RMSE	0.403	0.327	0.457	0.121	0.118	0.148	0.495
StdDev	0.014	0.035	0.012	0.01	0.05	0.0072	0.01
Bagged trees							
RMSE	0.422	0.44	0.533	0.136	0.123	0.276	0.514
StdDev	0.013	0.066	0.016	0.012	0.064	0.0059	0.011

Table 1. Performance of bagged Groves (Randomized Dynamic Programming training) compared to boosting and bagging. RMSE on the test set averaged over 10 runs.

Stochastic Gradient Boosting. The obvious main competitor to bagged Groves is gradient boosting [5] [6], a different ensemble of trees also based on additive models. There are two major differences between boosting and Groves. First, boosting never discards trees, i.e., every generated tree stays in the model. Grove iteratively retrains its trees. Second, all trees in a boosting ensemble are always built to a fixed size, while groves of large trees are trained first using groves of smaller trees. We believe that these differences allow Groves to better capture the natural additive structure of the response function.

The general gradient boosting framework supports optimizing for a variety of loss functions. We selected squared-error loss because this is the loss function that our current version of the Groves algorithm optimizes for. However, like gradient boosting, Groves can be modified to optimize for other loss functions.

Friedman [6] recommends boosting small trees with at most 4–10 leaf nodes for best results. However, we discovered for one of our datasets that using larger trees with gradient boosting did significantly better. This is not surprising since some real datasets contain complex interactions, which cannot be accurately modeled by small trees. For fairness we therefore also include larger boosted trees in the comparison than Friedman suggested. More precisely, we tried all $\alpha \in \{1, 0.5, 0.2, 0.1, 0.05\}$. Figure 7 shows the typical correspondence between α and number of leaf nodes in a tree, which was very similar across the data sets. Preliminary results did not show any improvement for tree size beyond $\alpha = 0.05$.

Stochastic gradient boosting deals with overfitting by means of two techniques: regularization and subsampling. Both techniques depend on user-set parameters. Based on recommendations in the literature and on our own evaluation we used the following values for the final evaluation: 0.1 and 0.05 for the regularization coefficient and 0.4, 0.6, and 0.8 as the fraction of the subsampling set size from the whole training set.

Boosting can also overfit if it is run for too many iterations. We tried up to 1500 iterations to make the maximum number of trees in the ensemble equal for

α	# leaf nodes
1	2 (stump)
0.5	3
0.2	8
0.1	17
0.05	38
0.02	100
0.01	225
0.005	500
0	full tree

Fig. 7. Typical number of leaf nodes for different values of α

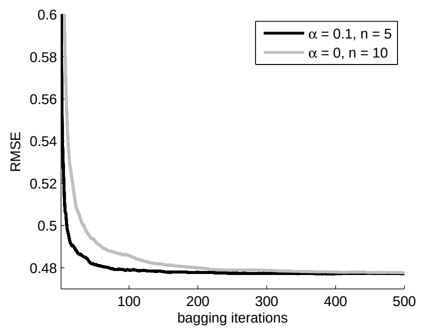

Fig. 8. Performance of bagged Grove for simpler and more complex models

all methods in comparison. The actual number of iterations that performs best was determined based on the validation set, and therefore can be lower than 1500 for the best boosted model.

In summary, to evaluate stochastic gradient boosting, we tried all combinations of the values described above for the 4 parameters: size of trees, number of iterations, regularization coefficient, and subsampling size. As for Groves, we determine the best combination of values for these parameters based on a separate validation set.

Bagging. Bagging single trees is known to provide good performance by significantly decreasing variance of the individual tree models. However, compared with Groves and boosting, which are both based on additive models, bagged trees do not explicitly model the additive structure of the response function. Increasing the number of iterations in bagging does not result in overfitting and bagging of larger trees usually produces better models than bagging smaller trees. Hence we omitted parameter tuning for bagging. Instead we simply report results for a model consisting of 1500 bagged full trees.

3.2 Datasets

Synthetic Data without Noise. This is the same data set that we used as a running example in the earlier sections. The response function is generated by Equation 1. The performance of bagged Groves on this dataset is much better than the performance of other methods.

Synthetic Data with Noise. This is the same synthetic dataset, only this time Gaussian noise is added to the response function. The standard deviation σ of the noise distribution is chosen as 1/2 of the standard deviation of the response in the original data set. As expected, the performance of all methods drops. Bagged Groves still perform clearly better, but the difference is smaller.

We have used 5 regression data sets from the collection of Luís Torgo [7] for the next set of experiments.

Kinematics. The Kinematics family of datasets originates from the Delve repository [8] and describes a simulation of robot arm movement. We used a $kin8nm$ version of the dataset: 8192 cases, 8 continuous attributes, high level of non-linearity, low level of noise. Groves show 20% improvement over gradient boosting on this dataset. It is worth noticing that boosting preferred large trees on this dataset; trees with $\alpha = 0.05$ showed clear advantage over smaller trees. However, there was no further improvement for boosting even larger trees. We attribute these effects to high non-linearity of the data.

Computer Activity. Another dataset from the Delve repository, describes the state of multiuser computer systems. 8192 cases, 22 continuous attributes. The variance of performance for all algorithms is low. Groves show small (3%) improvement compared to boosting.

California Housing. This is a dataset from the StatLib repository [9] and it describes housing prices in California from the 1990 Census: $20,640$ observations, 9 continuous attributes. Groves show 6% improvement compared to boosting.

Stock. This is a relatively small (960 data points) regression dataset from the StatLib repository. It describes daily stock prices for 10 aerospace companies: the task is to predict the first one from the other 9. Prediction quality from all methods is very high, so we can assume that the level of noise is small. This is another case when Groves give significant improvement (18%) over gradient boosting.

Elevators. This data set is obtained from the task of controlling an aircraft [10]. It seems to be noisy, because the variance of performance is high although the data set is rather large: $16,559$ cases with 18 continuous attributes. Here we see a 6% improvement.

3.3 Discussion

Based on the empirical results we conjecture that Bagged Groves outperform the other algorithms most when the datasets are highly non-linear and not very noisy. (Noise can obscure some of the non-linearity in the response function, making the best models that can be learned from the data more linear than they would have been for models trained on the response without noise.) This can be explained as follows. Groves can capture additive structure yet at the same time use large trees. Large trees capture non-linearity and complex interactions well, and this gives Groves an advantage over gradient boosting which relies mostly on additivity. Gradient boosting usually works best with small trees, and fails to make effective use of large trees. At the same time most data sets, even non-linear ones, still have significant additive structure. The ability to detect and model this additivity gives Groves an advantage over bagging, which is effective with large trees, but does not explicitly model additive structure.

Gradient boosting is a state of the art ensemble tree method for regression. Chipman et al [11] recently performed an extensive comparison of several algorithms on 42 data sets. In their experiments gradient boosting showed performance similar to or better than Random Forests and a number of other types of models. Our algorithm shows performance consistently better than gradient boosting and for this reason we do not expect that Random Forests or other methods that are not superior to gradient boosting would outperform our bagged Groves.

In terms of computational cost, bagged Groves and boosting are comparable. In both cases a large number of tree models has to be trained (more for Groves) and there is a variety of parameter combinations that need to be examined (more for boosting).

4 Bagging Iterations and Overfitting Resistance

In our experiments we used a fixed number of bagging iterations and did not consider this a tuning parameter because bagging rarely overfits. In bagging the number of iterations is not as crucial as it is for boosting: if we bag as long as we can afford, we will get the best value that we can achieve. In that sense the experimental results we report are conservative and Bagged Groves could potentially be improved by additional bagging iterations.

We observed a similar trend for parameters α and N as well: more complex models (larger trees, more trees) are at least as good as their less complex counterparts, but only if they are bagged sufficiently many times. Figure 3.1 shows how the performance on the synthetic data set with noise depends on the number of bagging iterations for two bagged Groves. The simpler one is trained with $N = 5$ and $\alpha = 0.1$ and the more complex one is trained with $N = 10$ and $\alpha = 0$. We can see that eventually they converge to the same performance and that the simpler model only does better than the complex model when the number of bagging iterations is small. [2]

We observed similar behavior for the other datasets. This suggests that one way to get good performance with bagged Groves might be to build the most complex Groves (large trees, many trees) that can be afforded and bag them many, many times until performance tops out. In this case we might not need a validation set to select the best parameter settings. However, in practice the most complex models can require many more iterations of bagging than simpler models that achieve almost the same level of performance much faster. Hence the approach that used in our experiments can be more useful in practice: select a computationally acceptable number of bagging iterations (100 seems to work fine, but one could also use 200 or 500 to be more confident) and search for the best N and α for this number of bagging iterations on the validation set.

[2] Note that this is only true because of the layered approach to training Groves which trains Groves of trees of smaller size before moving on to Groves with larger trees. If one initialized a Grove with a single large tree, performance of bagged Groves might still decrease with increasing tree size because the ability of the Grove to learn the additive structure of the problem would be injured.

5 Conclusion

We presented a new regression algorithm, bagged Groves of trees, which is an additive ensemble of regression trees. It combines the benefits of large trees that model complex interactions with benefits of capturing additive structure by means of additive models. Because of this, bagged Groves perform especially well on complex non-linear datasets where the structure of the response function contains both additive structure (which is best modeled by additive trees) and variable interactions (which is best modeled within a tree). We have shown that on such datasets bagged Groves outperform state-of-the-art techniques such as stochastic gradient boosting and bagging. Thanks to bagging, and the layered way in which Groves are trained, bagged Groves resist overfitting—more complex Groves tend to achieve the same or better performance as simpler Groves.

Groves are good at capturing the additive structure of the response function. A future direction of our work is to develop techniques for determining properties inherent in the data using this algorithm. In particular, we believe we can use Groves to learn useful information about statistical interactions between variables in the data set.

Acknowledgements. The authors would like to thank Daniel Fink, Wes Hochachka, Steve Kelling and Art Munson for useful discussions. This work was supported by NSF grants 0427914 and 0612031.

References

1. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer (2001)
2. Breiman, L.: Bagging Predictors. Machine Learning **24** (1996) 123–140
3. Hooker, G.: Discovering ANOVA Structure in Black Box Functions. In: Proc. ACM SIGKDD. (2004)
4. Bylander, T.: Estimating Generalization Error on Two-Class Datasets Using Out-of-Bag Estimates. Machine Learning **48**(1–3) (2002) 287–297
5. Friedman, J.: Greedy Function Approximation: a Gradient Boosting Machine. Annals of Statistics **29** (2001) 1189 – 1232
6. Friedman, J.: Stochastic Gradient Boosting. Computational Statistics and Data Analysis **38** (2002) 367 – 378
7. Torgo, L.: Regression DataSets. http://www.liacc.up.pt/~ltorgo/Regression/DataSets.html
8. Rasmussen, C.E., Neal, R.M., Hinton, G., van Camp, D., Revow, M., Ghahramani, Z., Kustra, R., Tibshirani, R.: Delve. University of Toronto. http://www.cs.toronto.edu/~delve
9. Meyer, M., Vlachos, P.: StatLib. Department of Statistics at Carnegie Mellon University. http://lib.stat.cmu.edu
10. Camacho, R.: Inducing Models of Human Control Skills. In: European Conference on Machine Learning (ECML'98). (1998)
11. Chipman, H., George, E., McCulloch, R.: Bayesian Ensemble Learning. In: Advances in Neural Information Processing Systems 19. (2007) 265–272