Stochastic Gradient Boosting

Jerome H. Friedman*

March 26, 1999

Abstract

Gradient boosting constructs additive regression models by sequentially fitting a simple parameterized function (base learner) to current "pseudo"–residuals by least–squares at each iteration. The pseudo–residuals are the gradient of the loss functional being minimized, with respect to the model values at each training data point, evaluated at the current step. It is shown that both the approximation accuracy and execution speed of gradient boosting can be substantially improved by incorporating randomization into the procedure. Specifically, at each iteration a subsample of the training data is drawn at random (without replacement) from the full training data set. This randomly selected subsample is then used in place of the full sample to fit the base learner and compute the model update for the current iteration. This randomized approach also increases robustness against overcapacity of the base learner.

1 Gradient Boosting

In the function estimation problem one has a system consisting of a random "output" or "response" variable y and a set of random "input" or "explanatory" variables $\mathbf{x} = \{x_1, \cdots, x_n\}$. Given a "training" sample $\{y_i, \mathbf{x}_i\}_1^N$ of known $(y, \mathbf{x})$–values, the goal is to find a function $F^*(\mathbf{x})$ that maps $\mathbf{x}$ to y, such that over the joint distribution of all $(y, \mathbf{x})$–values, the expected value of some specified loss function $\Psi(y, F(\mathbf{x}))$ is minimized

$$F^*(\mathbf{x}) = \arg\min_{F(\mathbf{x})} E_{y,\mathbf{x}} \Psi(y, F(\mathbf{x})). \tag{1}$$

Boosting approximates $F^*(\mathbf{x})$ by an "additive" expansion of the form

$$F(\mathbf{x}) = \sum_{m=0}^{M} \beta_m h(\mathbf{x}; \mathbf{a}_m), \tag{2}$$

where the functions $h(\mathbf{x}; \mathbf{a})$ ("base learner") are usually chosen to be simple functions of $\mathbf{x}$ with parameters $\mathbf{a} = \{a_1, a_2, \cdots\}$. The expansion coefficients $\{\beta_m\}_0^M$ and the parameters $\{\mathbf{a}_m\}_0^M$ are jointly fit to the training data in a forward "stage–wise" manner. One starts with an initial guess $F_0(\mathbf{x})$, and then for $m = 1, 2, \cdots, M$

$$(\beta_m, \mathbf{a}_m) = \arg\min_{\beta, \mathbf{a}} \sum_{i=1}^{N} \Psi\left(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a})\right) \tag{3}$$

and

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}; \mathbf{a}_m). \tag{4}$$

Gradient boosting (Friedman 1999) approximately solves (3) for arbitrary (differentiable) loss functions $\Psi(y, F(\mathbf{x}))$ with a two step procedure. First, the function $h(\mathbf{x}; \mathbf{a})$ is fit by least–squares

$$\mathbf{a}_m = \arg\min_{\mathbf{a},\rho} \sum_{i=1}^{N} [\tilde{y}_{im} - \rho h(\mathbf{x}_i; \mathbf{a})]^2 \tag{5}$$

to the current "pseudo"–residuals

$$\tilde{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}. \tag{6}$$

Then, given $h(\mathbf{x}; \mathbf{a}_m)$, the optimal value of the coefficient β_m is determined

$$\beta_m = \arg\min_{\beta} \sum_{i=1}^{N} \Psi\left(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a}_m)\right). \tag{7}$$

This strategy replaces a potentially difficult function optimization problem (3) by one based on least–squares (5), followed by a single parameter optimization (7) based on the general loss criterion Ψ.

Gradient tree boosting specializes this approach to the case where the base learner $h(\mathbf{x}; \mathbf{a})$ is an L–terminal node regression tree. At each iteration m, a regression tree partitions the $\mathbf{x}$–space into L–disjoint regions $\{R_{lm}\}_{l=1}^{L}$ and predicts a separate constant value in each one

$$h(\mathbf{x}; \{R_{lm}\}_1^L) = \sum_{l=1}^{L} \bar{y}_{lm}\, 1(\mathbf{x} \in R_{lm}). \tag{8}$$

Here $\bar{y}_{lm} = mean_{\mathbf{x}_i \in R_{lm}}(\tilde{y}_{im})$ is the mean of (6) in each region R_{lm}. The parameters of this base learner are the splitting variables and corresponding split points defining the tree, which in turn define the corresponding regions $\{R_{lm}\}_1^L$ of the partition at the mth iteration. These are induced in a top–down "best–first" manner using a least–squares splitting criterion (Friedman, Hastie, and Tibshirani 1998). With regression trees, (7) can be solved separately within each region R_{lm} defined by the corresponding terminal node l of the mth tree. Because the tree (8) predicts a constant value $\bar{y}_{lm}$ within each region R_{lm}, the solution to (7) reduces to a simple "location" estimate based on the criterion Ψ

$$\gamma_{lm} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{lm}} \Psi\left(y_i, F_{m-1}(\mathbf{x}_i) + \gamma\right).$$

The current approximation $F_{m-1}(\mathbf{x})$ is then separately updated in each corresponding region

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot \gamma_{lm} 1(\mathbf{x} \in R_{lm}).$$

The "shrinkage" parameter $0 < \nu \leq 1$ controls the learning rate of the procedure. Empirically (Friedman 1999), it was found that small values ($\nu \leq 0.1$) lead to much better generalization error.

This leads to the following algorithm for generalized boosting of decision trees:

	Algorithm 1: Gradient_TreeBoost
1	$F_0(\mathbf{x}) = \arg\min_{\gamma} \sum_{i=1}^{N} \Psi(y_i, \gamma)$
2	For $m = 1$ to M do:
3	$\tilde{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$, $i = 1, N$
4	$\{R_{lm}\}_1^L = L - \text{terminal node } tree(\{\tilde{y}_{im}, \mathbf{x}_i\}_1^N)$
5	$\gamma_{lm} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{lm}} \Psi\left(y_i, F_{m-1}(\mathbf{x}_i) + \gamma\right)$
6	$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot \gamma_{lm} 1(\mathbf{x} \in R_{lm})$
7	endFor

Friedman 1999 presented specific algorithms based on this template for several loss criteria including least–squares: $\Psi(y,F) = (y-F)^2$, least–absolute–deviation: $\Psi(y,F) = |y-F|$, Huber–M: $\Psi(y,F) = (y-F)^2\ 1(\ |y-F| \leq \delta\ ) + 2\delta(|y-F| - \delta/2)\, 1(\ |y-F| > \delta\ )$, and for classification, K–class multinomial negative log–likelihood .

2 Stochastic gradient boosting

With his "bagging" procedure, Breiman (1996) introduced the notion that injecting randomness into function estimation procedures could improve their performance. Early implementations of AdaBoost (Freund and Schapire 1996) also employed random sampling, but this was considered an approximation to deterministic weighting when the implementation of the base learner did not support observation weights, rather than as an essential ingredient. Recently, Breiman 1999 proposed a hybrid bagging–boosting procedure ("adaptive bagging") intended for least–squares fitting of additive expansions (2). It replaces the base learner in regular boosting procedures with the corresponding bagged base learner, and substitutes "out–of–bag" residuals for the ordinary residuals at each boosting step.

Motivated by Breiman 1999, a minor modification was made to gradient boosting (Algorithm 1) to incorporate randomness as an integral part of the procedure. Specifically, at each iteration a subsample of the training data is drawn at random (without replacement) from the full training data set. This randomly selected subsample is then used, instead of the full sample, to fit the base learner (line 4) and compute the model update for the current iteration (line 5).

Let $\{y_i, \mathbf{x}_i\}_1^N$ be the entire training data sample and $\{\pi(i)\}_1^N$ be a random permutation of the integers $\{1, \cdots, N\}$. Then a random subsample of size $\tilde{N} < N$ is given by $\{y_{\pi(i)}, \mathbf{x}_{\pi(i)}\}_1^{\tilde{N}}$. The stochastic gradient boosting algorithm is then

	Algorithm 2: Stochastic Gradient_TreeBoost
1	$F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} \Psi(y_i, \gamma)$
2	For $m = 1$ to M do:
3	$\{\pi(i)\}_1^N = \text{rand\_perm}\ \{i\}_1^N$
4	$\tilde{y}_{\pi(i)m} = -\left[\frac{\partial \Psi(y_{\pi(i)}, F(\mathbf{x}_{\pi(i)}))}{\partial F(\mathbf{x}_{\pi(i)})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})},\ i = 1, \tilde{N}$
5	$\{R_{lm}\}_1^L = L - \text{terminal node}\ tree(\{\tilde{y}_{\pi(i)m}, \mathbf{x}_{\pi(i)}\}_1^{\tilde{N}})$
6	$\gamma_{lm} = \arg\min_\gamma \sum_{\mathbf{x}_{\pi(i)} \in R_{lm}} \Psi\left(y_{\pi(i)}, F_{m-1}(\mathbf{x}_{\pi(i)}) + \gamma\right)$
7	$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot \gamma_{lm} 1(\mathbf{x} \in R_{lm})$
8	endFor

Using $\tilde{N} = N$ introduces no randomness and causes Algorithm 2 to return the same result as Algorithm 1. The smaller the fraction $f = \tilde{N}/N$, the more the random samples used in successive iterations will differ, thereby introducing more overall randomness into the procedure. Using the value $f = 1/2$ is roughly equivalent to drawing bootstrap samples at each iteration. Using $\tilde{N} = f \cdot N$ also reduces computation by a factor of f. However, making the value of f smaller reduces the amount of data available to train the base learner at each iteration. This will cause the variance associated with the individual base learner estimates to increase.

3 Simulation studies

The effect of randomization on Gradient Tree_Boost procedures will likely depend on the particular problem at hand. Important characteristics of problems that affect performance include training sample size N, true underlying "target" function $F^*(\mathbf{x})$ (1), and the distribution of the departures, ε, of $y \mid \mathbf{x}$ from $F^*(\mathbf{x})$. In order to gauge the value of any estimation method it is necessary to accurately evaluate its performance over many different situations. This is

most conveniently accomplished through Monte Carlo simulation where data can be generated according to a wide variety of prescriptions, and resulting performance accurately calculated.

One of the most important characteristics of any problem affecting performance is the true underlying target function $F^*(\mathbf{x})$ (1). Since the nature of the target function can vary greatly over different problems, and is seldom known, we evaluate the relative merits of randomized gradient tree gradient boosting on a variety of different targets randomly drawn from a broad "realistic" class of functions. The procedure used here to generate the random functions is described in Friedman 1999. The simulation studies below are based on the same 100 randomly generated target functions used in Friedman 1999.

Performance is based on the average–absolute–error of the derived estimate $\hat{F}(\mathbf{x})$ in approximating each target $F^*(\mathbf{x})$

$$A(\hat{F}) = E_{\mathbf{x}}|F^*(\mathbf{x}) - \hat{F}(\mathbf{x})| \tag{9}$$

as estimated from a large independent test data set. Performance comparisons among several different estimates $\{\hat{F}_k(\mathbf{x})\}_1^K$ are based on the absolute error (9) of each one relative to the best performer

$$R(\hat{F}_k) = A(\hat{F}_k) \Big/ \min\{A(\hat{F}_l)\}_1^K. \tag{10}$$

Thus, for each of the 100 target functions, the best method $k^* = \arg\min_k \{A(\hat{F}_k)\}_1^K$ receives the value $R(\hat{F}_{k^*}) = 1.0$, and the others receive a larger value $\{R(\hat{F}_k) > 1.0\}_{k \neq k^*}$. If a particular method was best (smallest error) for every target, its distribution of (10) over all 100 target functions would be a point mass at the value 1.0.

3.1 Regression

In this section, the effect of randomization on the (Huber) M_TreeBoost procedure is investigated. Among the regression procedures derived in Friedman 1999, M_TreeBoost had the best overall performance and was considered the method of choice. Its break–down parameter was set to the default value $\alpha = 0.9$. For small data sets ($N = 500$) the shrinkage parameter ν (Algorithm 2) was set to $\nu = 0.005$. For the larger ones ($N = 5000$) it was set to $\nu = 0.05$. Best–first regressions trees with six terminal nodes were used as the base learner.

Here we compare various levels of randomization in terms of performance over the 100 target functions for two different error distributions. One hundred data sets $\{y_i, \mathbf{x}_i\}_1^N$ were generated according to

$$y_i = F^*(\mathbf{x}_i) + \varepsilon_i \tag{11}$$

where $F^*(\mathbf{x})$ represents each of the 100 randomly generated target functions. For the first study, the errors ε_i were generated from a Gaussian distribution with zero mean, and variance adjusted so that

$$E|\varepsilon| = E_{\mathbf{x}}|F^*(\mathbf{x}) - median_{\mathbf{x}} F^*(\mathbf{x})| \tag{12}$$

giving a 1/1 signal–to–noise ratio. For the second study the errors were generated from a "slash" distribution, $\varepsilon_i = s \cdot (u/v)$, where $u \backsim N(0,1)$ and $v \backsim U[0,1]$. The scale factor s is adjusted to give a 1/1 signal–to–noise ratio (12). The slash distribution has very thick tails and is often used as an extreme to test robustness.

3.1.1 Gaussian errors

Figure 1 compares the performance of M_TreeBoost for different degrees of randomization, for small training data sets ($N = 500$). The degree of randomness is controlled by the fraction $f = \tilde{N}/N$ of randomly drawn observations used to train the regression tree at each iteration.

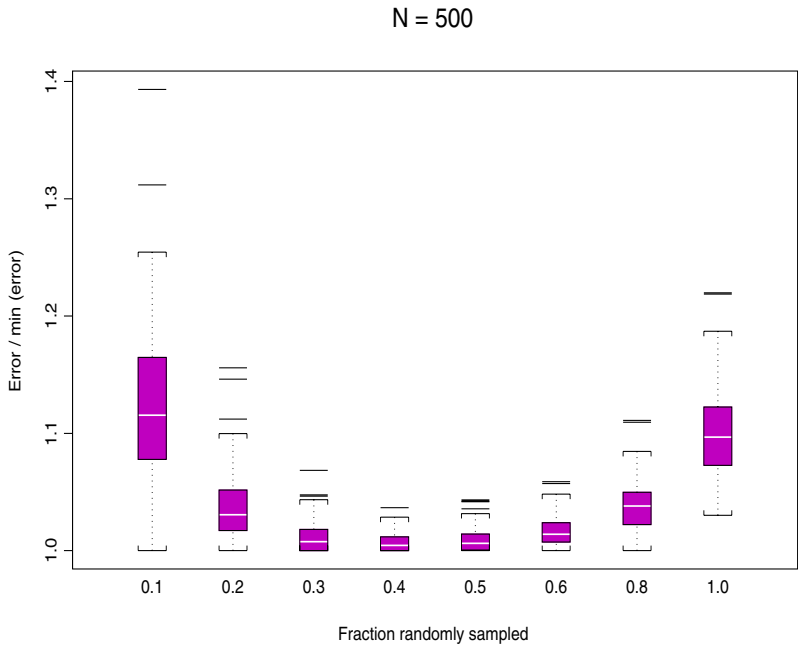

Figure 1: Distributions of absolute approximation error, relative to the best, for training on different fractions of randomly selected observations from the full training sample at each iteration, for small data sets and Gaussian errors.

Shown are the distributions of $\{R(\hat{F}_f)\}_1^8$ (10) over the 100 targets, for eight values of f. The largest value $f = 1.0$ corresponds to deterministic boosting (Algorithm 1), while successively smaller values introduce increasing degrees of randomness. The eight distributions are each summarized by boxplots. The shaded area of each boxplot shows the interquartile range of the distribution with the enclosed white bar being the median. The outer hinges represent the points closest to (plus/minus)1.5 interquartile range units from the (upper/lower) quartiles. The isolated bars represent individual points outside this range (outliers).

One sees from Fig. 1 that randomization improves performance substantially. For all values of $f < 1.0$, except the most extreme one ($f = 0.1$), the distribution of absolute error relative to the best (10) is much closer to the minimum value of 1.0 than is the corresponding distribution for deterministic algorithm ($f = 1.0$). Averaged over these 100 target functions, the best value of the sampling fraction as is approximately 40% ($f = 0.4$) where there is a typical improvement of 11% in absolute error (9) over no sampling. (This represents a 22% improvement on the squared–error scale.) However, sampling only 30% or even 20% of the data at each iteration gives considerable improvement over no sampling at all, with a corresponding computational speed–up by factors of 3 and 5 respectively. For sampling fractions close to optimal ($f = 0.4$), the dispersion of the distributions is also smaller. This means that for nearly all of the targets they produced the best, or close to the best , average absolute errors. Sampling fractions farther from the optimal value have more dispersion in their distributions. On some targets they produced best or close to best results, while on others they did very badly. This illustrates that relative performance can depend strongly on the particular target encountered. For example, there was one target function (out of the 100) for which $f = 0.1$ produced the lowest error. The distribution for $f = 1.0$ indicates that using a sampling fraction of around 50% produces improvements over no sampling in the range of about 4% to 24% in absolute error, with a median of 11%.

Figure 2 shows a similar comparison over the same 100 target functions and error distribution,

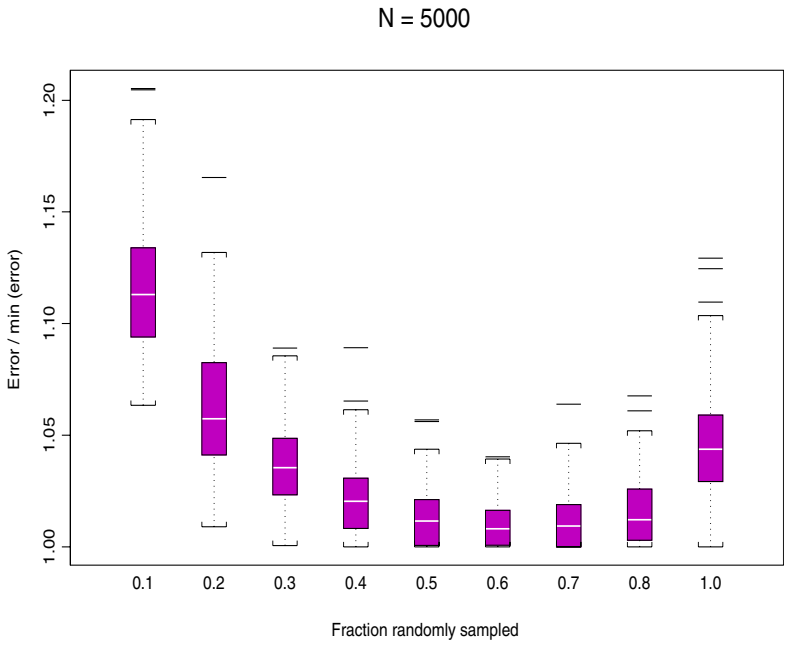

Figure 2: Distributions of absolute approximation error, relative to the best, for training on different fractions of randomly selected observations from the full training sample at each iteration, for moderate sized data sets and Gaussian errors.

but for moderate sized training data sets ($N = 5000$). Here one sees a similar but less dramatic pattern. Note that the vertical scale of Fig. 2 is half that of Fig. 1. The optimal sampling fraction is closer to 60% and typical improvements over the deterministic algorithm are from 1% to 11% in absolute error with a median of 5%. Although the increase in accuracy associated with random sampling is less dramatic with larger data sets, the speed increase is more meaningful.

Figure 3 compares the performance on small data sets ($N = 500$) of M_TreeBoost using different sized regression trees as the base learner. The left panel shows the distribution over the 100 target functions of absolute error relative to the best (10) for $L \in \{3, 6, 11, 21, 41\}$ using the deterministic algorithm. The right panel show the corresponding distributions for 50% sampling ($f = 0.5$). In both cases the optimal tree size as averaged over the 100 targets is $L = 6$. Increasing the capacity of the base learner by using larger trees degrades performance through "over–fitting". However, one sees that applying randomization ($f = 0.5$) moderates the effect of over–fitting. The increase in median of the relative error (10) distribution in going from $L = 6$ to $L = 41$ for 50% sampling is only one–third that for the deterministic algorithm.

Figure 4 shows the same effect from a different perspective. Here the distribution of the ratio of the error of the deterministic algorithm to that using 50% sampling ($A(\hat{F}_{1.0})/A(\hat{F}_{0.5})$) (9) is shown as a function of tree size. For three and six terminal node trees, deterministic M_TreeBoost typically is seen to be 8% worse in absolute error than using 50% random sampling. For 41–node trees that ratio is typically close to 20%. Thus, random subsampling is more effective in reducing error with larger trees, and can thereby mitigate, if not eliminate, the effect of over–capacity of the base learner.

3.1.2 Slash errors

Gaussian error distribution is a well behaved ideal that is seldom realized in practice. To check the extent to which the effect of randomization depends upon the error distribution we applied

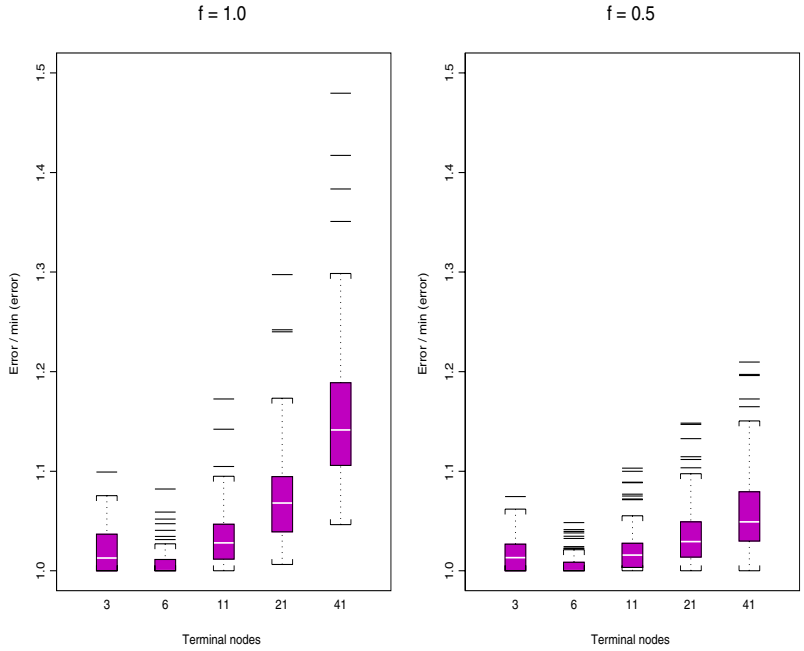

Figure 3: Distributions of absolute approximation error, relative to the best, for different sized regression trees as base learner, for small ($N = 500$) training samples. The left panel is for training on the full sample. The right panel is with 50% random subsampling at each iteration.

M_TreeBoost to the same 100 target functions with slash distributed errors ε_i (11). The slash distribution represents an opposite extreme to the Gaussian with very thick tails and many outliers. Figure 5 shows performance comparisons analogous to Fig. 1 and Fig. 2, but with slash rather than Gaussian errors. The distributions of relative error (10) for both sample sizes resemble those for the large ($N = 5000$) sample with Gaussian errors (Fig. 2). In particular, for the small sample size, the performance gain through random sampling is roughly half that achieved with Gaussian errors. For the larger sample, improvement is relatively insensitive to the sampling fraction f in the range $0.5 \leq f \leq 0.8$.

3.2 Classification

Here we consider a Bernoulli distributed output variable $y \in \{-1, 1\}$ with $\Pr(y = 1 \,|\, \mathbf{x}) = 1/(1 + \exp(-2F(\mathbf{x})))$. An appropriate loss criterion is the "deviance" (twice binomial negative log–likelihood) $\Psi(y, \hat{F}) = 2\log(1 + \exp(-2y\hat{F}))$. It serves as a continuous surrogate for misclassification error $1(y\hat{F} < 0)$. Data was generated using the same 100 target functions as above. For each target $F^*(\mathbf{x})$, the median $\tilde{F} = median_{\mathbf{x}} F^*(\mathbf{x})$ was computed. The data set for each trial was taken to be $\{y_i = sign(F^*(\mathbf{x}_i) - \tilde{F}),\, \mathbf{x}_i\}_1^N$. Thus, there are equal numbers in the two classes and the Bayes error rate is zero. However, the decision boundaries induced by many of the $F^*(\mathbf{x})$ are fairly complex.

Figure 6 shows the distribution of the error rate $e(\hat{F}) = E_{y,\mathbf{x}} 1(y\hat{F}(\mathbf{x}) < 0)$, relative to the best $e(\hat{F}_k) \Big/ \min\{e(\hat{F}_l)\}_1^K$, for several values of the sampling fraction f. The left panel shows the distributions for the small samples, while the right panel shows them for the larger ones. One sees behavior similar, but not identical, to the regression case. Randomization is beneficial on average, but not as universally so as with regression. The distributions corresponding to f–values close to optimal ($0.5 \leq f \leq 0.8$) have higher medians and larger spreads, especially for

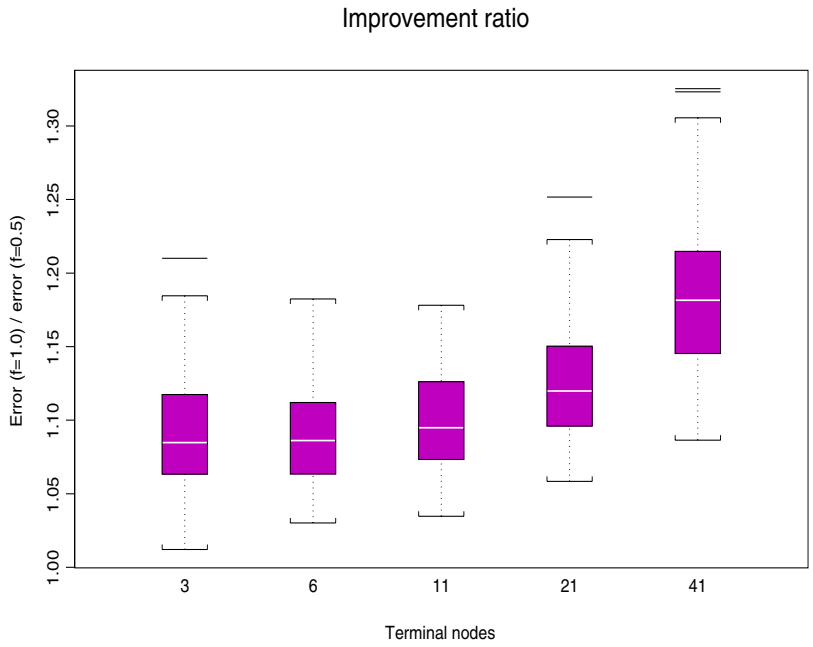

Figure 4: Distributions of the ratio of the error of the deterministic algorithm to that using 50% sampling with different sized regression trees as base learner, for small ($N = 500$) training samples.

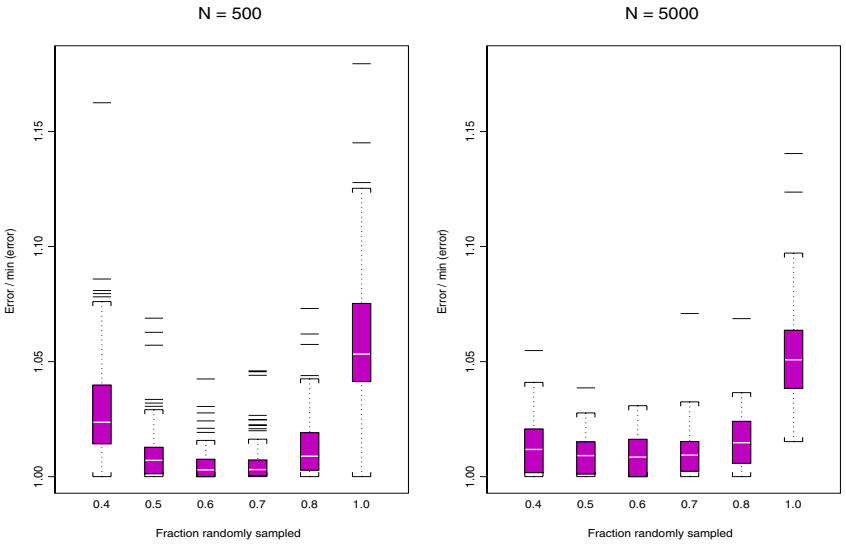

Figure 5: Distributions of absolute approximation error, relative to the best, for training on different fractions of randomly selected observations from the full training sample at each iteration, with slash errors. The left panel is for small data sets and the right one is for moderately sized data sets.

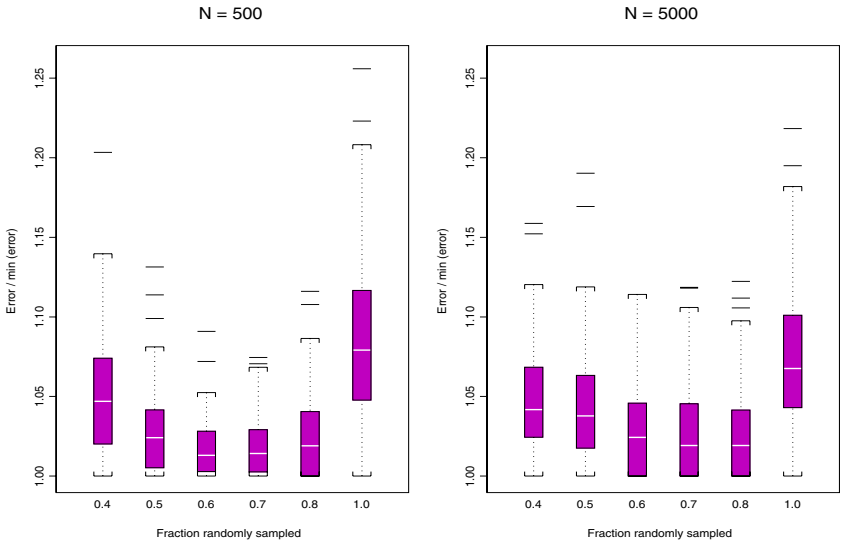

Figure 6: Distributions of error rate, relative to the best, for training on different fractions of randomly selected observations from the full training sample at each iteration, for a binary valued output variable. The left panel is for small data sets and the right one is for moderately sized data sets.

the larger sample. This indicates there is less assurance that randomization will improve error rate in individual situations. In fact, no sampling ($f = 1.0$) had smaller error than any of the sampling fractions $f < 1.0$ on three of the 100 targets for $N = 500$, and on five of them for $N = 5000$.

4 Discussion

The results of the previous section indicate that the accuracy of gradient boosting can be substantially improved by introducing randomization through the simple expedient of training the base learner on different randomly selected data subsets at each iteration. The degree of improvement is seen to depend on the particular problem at hand in terms of the training sample size N, the true underlying target function $F^*(\mathbf{x})$ (1), the distribution of $y \,|\, \mathbf{x}$ (Gaussian, slash, Bernoulli), and the capacity of the base learner. The reason why this randomization produces improvement is not clear. The fact that it is most effective for small samples, and with high capacity base learners, suggest that variance reduction is an important ingredient. Using smaller subsamples causes the variance of the individual base learner estimates at each iteration to *increase*. However, there is less correlation between the these estimates at different iterations. This tends to reduce the variance of the combined model (2), which in effect averages the base learner estimates. Apparently the latter averaging effect dominates the former one even for surprisingly small subsamples. This phenomenon is well known in bagging, where bootstrap sampling produces random subsamples with effective size roughly half that of the full training sample. Stochastic gradient boosting can be viewed in this sense as an boosting–bagging hybrid. Adaptive bagging (Breiman 1999) represents an alternative hybrid approach.

The results obtained here suggest that the original stochastic versions of AdaBoost may have merit beyond that of implementation convenience. With deterministic AdaBoost, weights assigned to each observation are recomputed at successive iterations so as to emphasize observations that are currently difficult to correctly predict. Rather than weighting, stochastic AdaBoost

draws random unweighted samples (with replacement) from the full training sample, with the probability of an observation being selected being proportional to its currently computed weight. This injects a random component into the procedure that tends to reduce the correlation between solutions at successive iterations. The usual prescription is to make the randomly drawn sample the same size as the original training data set. The results for stochastic gradient boosting suggest that stochastic AdaBoost's accuracy might also be improved by increasing randomness through drawing smaller samples at each iteration. This produces a computational savings as well.

5 Acknowledgments

Helpful discussions with Leo Breiman are gratefully acknowledged. This work was partially supported by CSIRO Mathematical and Information Sciences, Australia, the Department of Energy under contract DE-AC03-76SF00515, and by grant DMS9764431 of the National Science Foundation.

References

[1] Breiman, L. (1996). Bagging predictors. *Machine Learning* **26**, 123–140.

[2] Breiman, L. (1999). Using adaptive bagging to debias regressions. Technical report, Dept. of Statistics, University of California, Berkeley.

[3] Freund, Y and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, 148–156.

[4] Friedman, J. H. (1999). Greedy Function Approximation: a Gradient Boosting Machine. Technical report, Dept. of Statistics, Stanford University

[5] Friedman J. H., Hastie, T, and Tibshirani, R. (1998). Additive logistic regression: a statistical view of boosting. Technical Report, Dept. of Statistics, Stanford University.