Robust Boosting and its Relation to Bagging

Saharon Rosset
IBM T.J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598
srosset@us.ibm.com

ABSTRACT

Several authors have suggested viewing boosting as a gradient descent search for a good fit in function space. At each iteration observations are re-weighted using the gradient of the underlying loss function. We present an approach of weight decay for observation weights which is equivalent to "robustifying" the underlying loss function. At the extreme end of decay this approach converges to Bagging, which can be viewed as boosting with a linear underlying loss function. We illustrate the practical usefulness of weight decay for improving prediction performance and present an equivalence between one form of weight decay and "Huberizing" — a statistical method for making loss functions more robust.

1. INTRODUCTION

Boosting [9, 8] and Bagging [3] are two approaches to combining "weak" models in order to build prediction models that are significantly better. Much has been written about the empirical success of these approaches in creating prediction models in actual modeling tasks [4, 1] . The theoretical discussions of these algorithms [4, 11, 12, 5, 16] have viewed them from various perspectives. The general theoretical and practical consensus, however, is that the weak learners for boosting should be really weak, while the "weak learners" for bagging should actually be strong. In tree terminology, one should use small trees when boosting and big trees for bagging. In intuitive "bias-variance" terms, we can say that bagging is mainly a variance reduction (or stabilization) operation, while boosting, in the way it flexibly combines models, is also a bias reduction operation, i.e., it adds flexibility to the representation beyond that of a single learner. In this paper we present a view of boosting and bagging which allows us to connect them in a natural way, and in fact view bagging as a form of boosting. This view is interesting because it facilitates creation of families of "intermediate" algorithms, which offer a range of "degrees of bias reduction" between boosting and bagging. Our experiments indicate that, while bagging is always significantly inferior to boosting in terms of predictive performance, in some cases intermediate approaches can outperform standard boosting.

Our exposition concentrates on 2-class classification, this being the most common application for both boosting and bagging, but the results mostly carry through to other learning domains where boosting and bagging have been used, such as multi-class classification, regression [10] and density estimation [17].

2. BOOSTING, BAGGING AND A CONNECTION

A view has emerged in the last few years of boosting as a gradient-based search for a good model in a large implicit feature space [16, 10]. More specifically, the components of a 2-class classification boosting algorithm are:

- A data sample $\{\mathbf{x}_i, y_i\}_{i=1}^n$, with $\mathbf{x}_i \in \mathcal{R}^p$ and $y_i \in \{-1, +1\}$
- A loss function $C(y, f)$. The two most commonly used are the exponential loss of AdaBoost [9] and the logistic log-likelihood of LogitBoost [11]. We will assume throughout that $C(y, f)$ is a *monotone decreasing and convex* function of the margin $m = yf$, and so we will sometimes write it as $C(m)$.
- A dictionary $\mathcal{H}$ of "weak learners", where each $h \in \mathcal{H}$ is a classification model: $h : \mathcal{R}^p \to \{-1, +1\}$.
  The dictionary most commonly used in classification boosting is classification trees, i.e., $\mathcal{H}$ is the set of all trees with up to k splits.

Given these components, a boosting algorithm incrementally builds a "good" linear combination of the dictionary functions:

$$F(\mathbf{x}) = \sum_{h \in \mathcal{H}} \beta_h h(\mathbf{x})$$

Where "good" is defined as making the empirical loss small:

$$\sum_i C(y_i F(\mathbf{x}_i))$$

The actual incremental algorithm is an exact or approximate coordinate descent algorithm. At iteration t we have the "current" fit F_t, and we look for the weak learner h_t

KDD'05, August 21–24, 2005, Chicago, Illinois, USA.

which maximizes the first order decrease in the loss, i.e., h_t maximizes

$$-\sum_i \frac{\partial C(y_i, F(\mathbf{x}_i))}{\partial \beta_h}|_{F=F_t}$$

or equivalently and more clearly it maximizes

$$-\sum_i \frac{\partial C(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}|_{F=F_t} \cdot h(\mathbf{x}_i)$$

which in the case of two-class classification is easily shown to be equivalent to minimizing

$$\sum_i w_i I\{y_i \neq h(\mathbf{x}_i)\} \tag{1}$$

Where $w_i = \left|\frac{\partial C(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}|_{F=F_t}\right|$. So we seek to find a weak learner which minimizes *weighted* error rate, with the weights being the gradient of the loss. If we use the exponential loss:

$$C(y, f) = \exp(-yf) \tag{2}$$

then it can be shown (e.g. [13]) that (1) is the exact classification task which AdaBoost [9], the original and most famous boosting algorithm, solves for finding the next weak learner. In their original AdaBoost implementation [8], Freund and Schapire suggested solving (1) on a "new" training data set of size n at each iteration, by sampling from the training dataset "with return" with probabilities proportional to w_i. This facilitates the use of methods for solving non-weighted classification problems for approximately solving (1). We will term this approach the "sampling" boosting algorithm. [10] has argued that sampling actually improves the performance of boosting algorithms, by adding much needed randomness (his approach is to solve the *weighted* version of (1) on a sub-sample, but the basic motivation applies to "sampling boosting" as well). Here is a formal description of a sampling boosting algorithm, given the inputs described above:

ALGORITHM 1. *Generic gradient-based sampling boosting algorithm*

1. *Set* $\beta_0 = \mathbf{0}$ *(dimension of* β *is* $|\mathcal{H}|$*).*
2. *For* $t = 1 : T$*,*
   (a) *Let* $F_i = \beta_{t-1}^{\mathrm{T}}\mathbf{h}(\mathbf{x}_i)$*,* $i = 1, \ldots, n$ *(the current fit, where* $\mathbf{h}(\mathbf{x})$ *is the vector of "weak learner" values at* $\mathbf{x}$*).*
   (b) *Set* $w_i = |\frac{\partial C(y_i, F_i)}{\partial F_i}|$*,* $i = 1, \ldots, n$*.*
   (c) *Draw a sample* $\{\mathbf{x}_i^*, y_i^*\}_{i=1}^n$ *of size* n *by re-sampling with return from* $\{\mathbf{x}_i, y_i\}_{i=1}^n$ *with probabilities proportional to* w_i
   (d) *Identify* $j_t = \arg\min_j \sum_i I\{y_i^* \neq h_j(\mathbf{x}^*{}_i)\}$*.*
   (e) *Set* $\beta_{t,j_t} = \beta_{t-1,j_t} + \epsilon$ *and* $\beta_{t,k} = \beta_{t-1,k}, k \neq j_t$*.*

Comments:

1. Implementation details include the determination of T (or other stopping criterion) and the approach for finding the minimum in step 2(d).
2. We have fixed the step size to ϵ at each iteration (step 2(e)). While AdaBoost uses a line search to determine step size, it can be argued that a fixed (usually "small") ϵ step is theoretically preferable (see [10, 18] for details).
3. An important issue in designing boosting algorithms is the selection of the loss function $C(\cdot, \cdot)$. Extreme loss functions, such as the exponential loss of AdaBoost (2), are not robust against outliers and misspecified data, as they assign overwhelming weight to observations which have the smallest margins. [11] have thus suggested replacing the exponential loss with the logistic log likelihood loss:

   $$C(y, f) = log(1 + exp(-yf)) \tag{3}$$

   but in many practical situations, in particular when the two classes in the training data are separable in $\mathrm{sp}(\mathcal{H})$, this loss can also be non-robust. See Section 4.
4. The algorithm as described here is not affected by positive affine transformations of the loss function, i.e., running Algorithm 1 using a loss function $C(m)$ is exactly the same as using $C^*(m) = aC(m) + b$ as long as $a > 0$.

Bagging for classification [3] is a different "model combining" approach. It searches, in each iteration, for the member of the dictionary $\mathcal{H}$ which best classifies a *bootstrap sample* of the original training data, and then averages the discovered models to get a final "bagged" model. So, in fact, we can say that:

PROPOSITION 1. *Bagging implements Algorithm 1, using a linear loss,* $C(y, f) = -yf$ *(or any positive affine transformation of it)*

PROOF. From the definition of the loss we get:

$$\forall F_t, \ \ \forall i, \ \ |\frac{\partial - yF(\mathbf{x}_i)}{\partial F(\mathbf{x}_i)}|_{F=F_t} \equiv 1$$

So no matter what our "current model" is, all the gradients are always equal, hence the weights will be equal if we apply a "sampling boosting" iteration using the linear loss. In the case that all weights are equal, the boosting sampling procedure described above reduces to bootstrap sampling. Hence the bagging algorithm is a "sampling boosting" algorithm with a linear loss. □

Thus, we can consider Bagging as a boosting algorithm utilizing a very robust (linear) loss. This loss is so robust that it requires no "message passing" between iterations through re-weighting, since the gradient of the loss *does not* depend on the current margins.

2.1 Robustness, convexity and boosting

The gradient of a linear loss does not emphasize low-margin observations over high-margin ("well predicted") ones. In fact, it is the most robust convex loss possible, in the following sense:

PROPOSITION 2. *Any loss which is a differentiable, convex and decreasing function of the margin has the property:*

$$m_1 < m_2 \Rightarrow |C'(m_1)| \geq |C'(m_2)|$$

And a linear loss is the only one which attains equality $\forall m_1, m_2$.

PROOF. Immediate from convexity and monotonicity. □

[16] have used arguments about generalization error bounds to argue that a good loss for boosting would be even more robust than the linear loss, and consequently non-convex. In particular, they argue that both high-margin and low-margin observations should have low weight, leading to a sigmoid-shaped loss function. Non-convex loss functions present a significant computational challenge, which [16] have solved for small dictionary examples. Although the idea of such "outlier tolerant" loss functions is appealing, we limit our discussion to convex loss functions, which facilitate the use of standard fitting methodology, in particular boosting.

Our view of bagging as boosting with linear loss allows us to interpret the similarity — and difference — between the algorithms by looking at the loss functions they are "boosting". The linear loss of bagging implies it is not emphasizing the badly predicted observations, but rather treats all data "equally". Thus it is more robust against outliers and more stable, but less "adaptable" to the data than boosting with an exponential or logistic loss.

3. BOOSTING WITH WEIGHT DECAY

The view of bagging as a boosting algorithm, opens the door to creating boosting-bagging hybrids, by "robustifying" the loss functions used for boosting. These hybrids may combine the advantages of boosting and bagging to give us new and useful algorithms.

There are two ways to go about creating these intermediate algorithms:

- Define a series of loss functions starting from a "boosting loss" – the exponential or logistic – and converging to the linear loss of bagging.
- Implicitly define the intermediate loss functions by decaying the weights w_i given by the boosting algorithm using a boosting loss. The loss implied will be the one whose gradient corresponds to the decayed weights.

The two approaches are obviously equivalent through a differentiation or integration operation. We will adopt the "weight decay" approach, but will discuss the loss function implications of the different decay schemes.

3.1 Weight decay functions

We would like to change the loss $C(\cdot,\cdot)$ to be more robust, by first decaying the (gradient) weights w_i, then considering the implicit effect of the decay on the loss. In general, we assume that we have a decay function $v(p,w)$ which depends on a decay parameter $p \in [0,1]$ and the observation weight $w \geq 0$. We require:

- $v(1,w) = w$, i.e., no decay
- $v(0,w) = 1$, i.e., no weighting, which implicitly assumes a linear loss when considered as a boosting algorithm and thus corresponds to Bagging
- Monotonicity: $w_1 < w_2 \rightarrow v(p,w_1) < v(p,w_2) \ \ \forall p$.
- Continuity in both p and w.

And once we specify a decay parameter p, the problem we solve in each boosting iteration, instead of (1), is to find the dictionary function h to minimize:

$$\sum_i v(p,w_i) I\{y_i \neq h(\mathbf{x}_i)\} \tag{4}$$

which we solve approximately as a non-weighted problem, using the "sampling boosting" approach of Algorithm 1.

3.2 Windsorised weights and Huberized loss

For clarity and brevity, we concentrate in this paper on one decay function — the bounding or "Windsorising" operator:

$$v(p,w) = \begin{cases} \frac{1}{1-p} & \text{if } w > \frac{p}{1-p} \\ \frac{w}{p} & \text{otherwise} \end{cases} \tag{5}$$

Since w is the absolute gradient of the loss, bounding w means that we are "huberizing" the loss function, i.e., continuing it linearly beyond some point (this operation was suggested, in the context of squared error loss for regression, by [14]). The loss function corresponding to the decayed weights is thus:

$$C^{(p)}(m) = \begin{cases} \frac{C(m)}{p} & m > m^*(p) \\ \frac{C(m^*(p))}{p} - \frac{m - m^*(p)}{1-p} & \text{otherwise} \end{cases} \tag{6}$$

where $m^*(p)$ is such that $C'(m^*(p)) = -\frac{p}{1-p}$ (unique because the loss is convex and monotone decreasing).

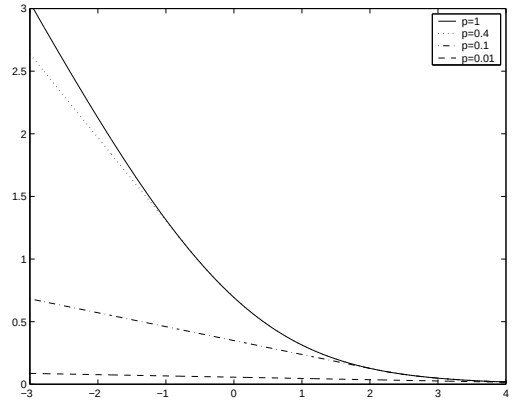

Figure 1: "Huberized" logistic loss, for different values of p (as a function of the margin).

As we have mentioned, for the purpose of the sampling boosting algorithm, only the relative sizes of the weights are important, and thus multiplication of the loss function by a constant of $1/p$ — which we have done to get $C^{(p)}$ from C and achieve the desired properties of $v(p,w)$ — does not

affect the algorithm for fixed p. Figure 1 illustrates the effect of bounding on the logistic log likelihood loss function, for several values of p (for presentation, this plot uses the non-scaled version of C, i.e., (6) multiplied by p).

There are many other interesting decay functions, such as the power transformation:

$$v(p, w) = w^p$$

This transformation is attractive because it does not entail the arbitrary "threshold" determination, but rather decays all weights, with the bigger ones decayed more. However, it is less interpretable in terms of its effect on the underlying loss. For the exponential loss $C(m) = \exp(-m)$, the power transformation does not change the loss, since $[\exp(-m)]^p = \exp(-mp)$. So the effect of this transformation is simply to slow the learning rate (exactly equivalent to decreasing ϵ in Algorithm 1).

4. PROPERTIES OF HUBERIZED BOOSTING LOSS FUNCTIONS

Our suggested "huberizing" transformation in (6) to the original loss function (in our case, the exponential loss (2) or the logistic loss (3)) gives a modified loss which is linear for small margins, then as the margin increases starts behaving like the original loss. We can therefore interpret boosting with the huberized loss function as "bagging for a while", until the margins become big enough and reach the non-huberized region. If we boost long enough with this loss, we will have most of the data margins in the non-huberized region, except for extreme outliers, and the resulting weighting scheme will revert to the "standard" boosting weights, except for extreme outliers.

We can thus consider boosting with the huberized loss to be more robust in the following sense:

> By huberizing the loss function, we are allowing the boosting algorithm to assign relatively small weights to outliers and difficult examples for a number of iterations, until most observations have become "well explained" by attaining large margins

This robustness argument is related to that of [11] and others for boosting with the logistic loss (3) rather than the exponential (2), since the logistic loss is approximately linear for negative margins (as its second derivative vanishes when the margin goes to $-\infty$). However, the logistic loss is very similar to the exponential for non-negative margins, as shown in [18]. Huberizing, on the other hand, allows for a flexible — and specific — definition of a linear region which can be adapted to the modeling task at hand.

It is interesting to note that two desirable properties of boosting loss functions are maintained for their huberized versions:

The boosting property

This property states that if the hypothesis space of weak learners allows "weak learning" — that is, weighted error rate of the weak learners of at most $1/2-\lambda$ for each iteration — then the training margins will increase, and generalization error will go to 0 for large enough training samples (see [20] for more details). Duffy and Helmbold ([7], Theorems 2-4) give sufficient conditions for this property to hold, which apply to the exponential and logistic loss functions. These conditions include strict convexity, and so do not directly apply to the huberized versions. However closer inspection of their conditions exposes that we only need to re-state their Theorem 2, which states that all the (non-normalized) margins are guaranteed to move beyond a fixed point U within $O(n^2/\lambda^2)$ iterations, and never cross it back. We re-prove this result for Huberized loss functions, and take $U = m^*$ to be the "Huberizing" point. Once all the margins have moved beyond m^*, the loss function becomes the logistic or exponential, which are strictly convex, and Duffy and Helmbold's results prove the boosting property[1].

THEOREM 3. *Assume the* weak learnability *property for Algorithm 1, i.e.,*

$$\frac{\sum_i w_i I\{h_t(\mathbf{x}_i) \neq y_i\}}{\sum_i w_i} \leq \frac{1}{2} - \lambda \ , \ \forall t \tag{7}$$

Then for the huberized version (6) of either the exponential loss or the logistic loss, we get that after at most

$$T = \frac{4n^2 C^{(p)}(0)}{C^{(p)}(m^*(p))\lambda^2}$$

iterations we achieve total loss no bigger than $C^{(p)}(m^*(p))$. That is:

$$\sum_i C^{(p)}(y_i, F_t(\mathbf{x}_i)) \leq C^{(p)}(m^*(p)), \ \forall t \geq T$$

and, consequently, all margins are guaranteed to be no bigger than $m^*(p)$ from that iteration on.

PROOF OUTLINE. Combining the definition of w_i with (7) we get at each iteration:

$$-\sum_i \frac{\partial C(y_i, F(\mathbf{x}_i))}{\partial \beta_{h_t}}|_{F=F_t} > 2\lambda \sum_i w_i \tag{8}$$

If our overall loss is bigger than $C^{(p)}(m^*(p))$ then we can show that:

$$\sum_i w_i > \frac{C^{(p)}(m^*(p))}{2} \tag{9}$$

(for the exponential loss this is a direct result of the property:

$$\left|\frac{\partial C^{(p)}(m)}{\partial m}\right| = C^{(p)}(m)$$

and for the logistic loss we also use Proposition 1 of [18]).

[1]Note that the generalization error property is proven for "line search" boosting algorithms, not the "ϵ-boosting" approach we employ in our implementation below. [7]'s results therefore imply that our suggested loss functions would also enjoy this property, if embedded into line-search boosting implementations.

Next, we bound the second derivative of the change in loss:

$$\sum_i \frac{\partial^2 C^{(p)}(y_i, F(\mathbf{x}_i))}{\partial \beta_{h_t}^2}|_{F=F_t} \leq \sum_i \frac{\partial^2 C^{(p)}(m_i)}{\partial m_i^2}|_{F=F_t} \leq 2nC^{(p)}(m^*(p)) \quad (10)$$

because the loss is convex and its second derivative is provably bounded by $2C^{(p)}(m^*(p))$.

Taking these three facts (8,9,10) combined we get that at each boosting iteration if we take a step of size $\frac{\lambda}{2n}$ we gain an improvement in the loss of at least:

$$\frac{C^{(p)}(m^*(p)) \cdot \lambda^2}{4n}$$

as long as the total loss exceeds $C^{(p)}(m^*(p))$. The actual step which a line-search boosting algorithm takes can be bigger than $\frac{\lambda}{2n}$, but the improvement in loss is guaranteed to be at least that attained by taking this fixed step size.

Now, we observe that our initial loss is $nC^{(p)}(0)$ and thus within this number of iterations:

$$T = \frac{4n^2 C^{(p)}(0)}{C^{(p)}(m^*(p))\lambda^2}$$

we are guaranteed to have a non-positive loss if we still have loss bigger than $C^{(p)}(m^*(p))$. This obviously gives a contradiction and we conclude that the total loss after T iterations gives the desired result:

$$\sum_i C^{(p)}(y_i, F_T(\mathbf{x}_i)) \leq C^{(p)}(m^*(p))$$

Since the overall loss always decreases in every iteration in line-search boosting, it can never exceed $C^{(p)}(m^*(p))$ in subsequent iterations, and since the loss function is non-negative decreasing, no margin can be smaller than $m^*(p)$. □

The margin maximizing property

[19] give sufficient conditions for regularized loss functions to be l_p-margin maximizing. This sufficient condition holds for the logistic and exponential loss functions, and also for their huberized versions, since it only depends on the loss function's behavior as the non-normalized margin converges to ∞. [18] explain how this property extends "approximately" to boosting. In essence, it implies that the boosting algorithm is seeking to maximize the l_1 margin of the data examples, and under quite general conditions will succeed in doing so.

5. EXPERIMENTS

We now discuss briefly some experiments to examine the usefulness of weight decay and the situations in which it may be beneficial. We use three datasets: the "Spam" and "Waveform" datasets, available from the UCI repository [2]; and the "Digits" handwritten digits recognition dataset, discussed in [15]. These were chosen since they are reasonably large and represent very different problem domains. Since we have limited the discussion here to 2-class models, we selected only two classes from the multi-class datasets: waveforms 1 and 2 from "Waveform" and the digits "2" and "3" from "Digits" (these were selected to make the problem as challenging as possible). The resulting 2-class data-sets are reasonably large — ranging in size between about 2000 and over 5000. In all three cases we used only 25% of the data for training and 75% for evaluation, as our main goal is not to excel on the learning task, but rather to make it difficult and expose the differences between the models which the different algorithms build.

Our experiments consisted of running Algorithm 1 using various loss functions, all obtained by decaying the observation weights given by the exponential loss function (2). We used "windsorising" decay as in (5) and thus the decayed versions correspond to "huberized" versions of (2). In all our experiments, bagging performed significantly worse than all versions of boosting, which is consistent with observations made by various researchers, that well-implemented boosting algorithms almost invariably dominate bagging (see for example Breiman's own experiments in [4]). It should be clear, however, that this fact does not contradict the view that bagging has some desirable properties, in particular a greater stabilizing effect than boosting.

We present in Figure 2 the results of running Algorithm 1 with $p = 1$ (i.e., using the non-decayed loss function (2)), and with $p = 0.0001$ (which corresponds to "huberizing" the loss, as in (6), at around $m = 9$) on the three data-sets. We use two settings for the "weak learner": 10-node and 100-node trees. The "learning rate" parameter ϵ is fixed at 0.1. The results in Figure 2 represent averages over 20 random train-test splits, with estimated 2-standard deviations confidence bounds.

These plots expose a few interesting observations:

1. Weight decay leads to a slower learning rate. From an intuitive perspective this is to be expected, as a more robust loss corresponds to less aggressive learning, by putting less emphasis on the hardest cases.

2. Weight decay is more useful when the "weak learners" are not weak, but rather strong, like a 100-node tree. This is particularly evident for the "Spam" dataset, where the performance of the non-decayed exponential boosting deteriorates significantly when we move from 10-node to 100-node learners, while that of the decayed version actually improves significantly. This phenomenon is also as we expect, given the more robust "Huberized" loss implied by the decay.

3. There is no consistent winner between the non-decayed and the decayed versions. For the "Spam" and "Waveform" datasets it seems that if we choose the best settings, we would choose the non-decayed loss with small trees. For the "Digits" data, the decayed loss seems to produce consistently better prediction models.

Note that in our examples we did not have big "robustness" issues as they pertain to extreme or highly prevalent outliers in the predictors or the response. Rather, we examine the "bias-variance" tradeoff in employing more robust loss functions in dealing with uncorrupted real-life datasets, and the difficult cases they often contain. It is likely that in extreme

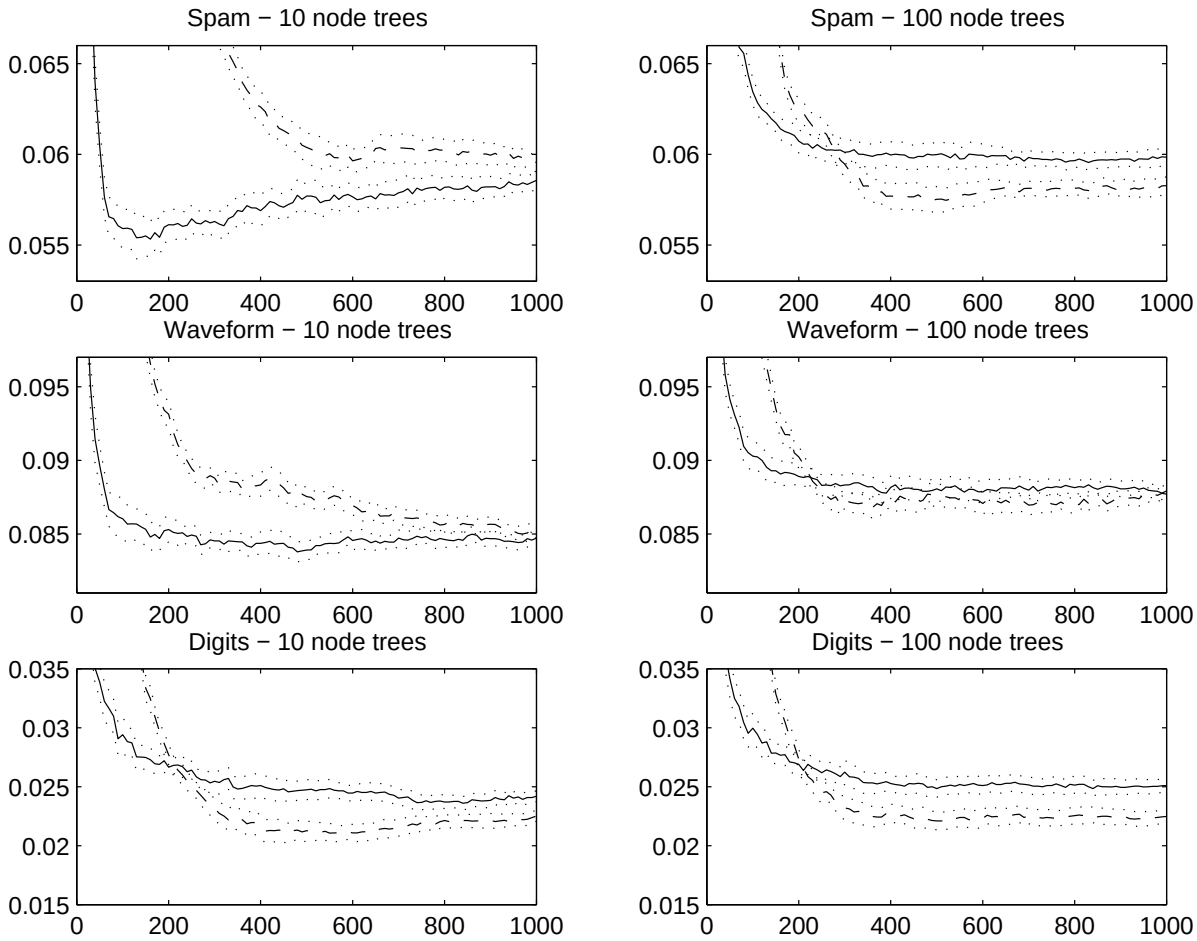

Figure 2: Results of running "sampling boosting" with $p = 1$ (solid) and $p = 0.0001$ (dashed) on the three datasets. with 10-node and 100-node trees as the weak learners. The x-axis is the number of iterations, and the y-axis is the mean test-set error over 20 random training-test assignments. The dotted lines are 2-sd confidence bounds.

situations — of many and/or big outliers — the advantage of using robust loss functions would be more pronounced.

6. DISCUSSION

The gradient descent view of boosting allows us to design boosting algorithms for a variety of problems, and choose the loss functions which we deem most appropriate. [18] show that gradient boosting approximately follows a path of l_1-regularized solutions to the chosen loss function. Thus selection of an appropriate loss is a critical issue in building useful algorithms.

In this paper we have shown that the gradient boosting paradigm covers bagging as well, and used it as rationale for considering new families of loss functions — hybrids of standard boosting loss functions and the bagging linear loss function — as candidates for gradient-based boosting. The results seem promising. There are some natural extensions to this concept, which may be even more promising, in particular the idea that the loss function does not have to remain fixed throughout the boosting iterations. Thus, we can design "dynamic" loss functions which change as the boosting iterations proceed, either as a function of the performance of the model or independently of it. It seems that there are a lot of interesting theoretical and practical questions that should come into consideration when we design such an algorithm, such as:

- Should the loss function become more or less robust as the boosting iterations proceed?
- Should the loss function become more or less robust if there are problematic data points?

In this context, it is interesting to note some previous work on bounding the boosting weights to achieve more robust performance by [6]. The main difference from our approach is that they operate on the re-normalized AdaBoost weights. Their approach thus lacks the gradient descent interpretation in a new loss, since AdaBoost's re-normalization of the weights is equivalent to re-scaling the underlying loss. Thus, [6]'s approach amounts to Huberizing the exponential loss at a different point in each iteration. Their algorithm also lacks a guaranteed boosting property like the one we proved in Section 4. However it would be interesting to compare the empirical merit of their approach to ours and perhaps draw conclusions with regard to using "adaptive" robust loss functions.

7. ACKNOWLEDGMENTS

Jerry Friedman, Trevor Hastie and Ji Zhu contributed to this paper through useful discussions and advice.

8. REFERENCES

[1] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: bagging, boosting and variants. *Machine Learning*, 36(1/2):105–139, 1999.

[2] C. Blake and C. Merz. Repository of machine learning databases. [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science., 1998.

[3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[4] L. Breiman. Arcing classifiers. *Annals of Statistics*, 26(3):801:849, 1998.

[5] P. Buhlmann and B. Yu. Analyzing bagging. *Annals of Statistics*, 2003.

[6] C. Domingo and O. Watanabe. Madaboost: A modification of adaboost. In *13th Annual Conference on Comp. Learning Theory*, 2000.

[7] N. Duffy and D. Helmbold. Potential boosters? In *Advance in Neural Information Processing*, 1999.

[8] Y. Freund and R. Scahpire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, 1996.

[9] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[10] J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5), 2001.

[11] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:337–407, 2000.

[12] J. H. Friedman and P. Hall. On bagging and nonlinear estimation. preprint, 2000.

[13] T. Hastie, T. Tibshirani, and J. Friedman. *Elements of Statistical Learning*. Springer-Verlag, New York, 2001.

[14] P. Huber. Robust estimation of a location parameter. *Annals of Mathematical Statistics*, 35(1):73:101, 1964.

[15] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems 2*, 1990.

[16] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. In *Neural Information Processing Systems*, volume 12, 1999.

[17] S. Rosset and E. Segal. Boosting density estimation. In *Advances in Neural Information Processing Systems 15*, 2003.

[18] S. Rosset, J. Zhu, and T. Hastie. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research*, 5, 2004.

[19] S. Rosset, J. Zhu, and T. Hastie. Margin maximizing loss functions. In *Advances in Neural Information Processing Systems 16*, 2004.

[20] R. E. Schapire, Y. Freund, P. Bartlett, and W. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26:1651–1686, 1998.