f (unknow target function) : $f: X \to y$ (ideal)

(+1(good); -1(bad); 0 ignored)

A takes D and H to get g

training eg.

D (Data) : $(x_1, y_1) \sim (x_n, y_n)$

A (algorithm) → g ≈ f (final hypothesis)

hypothesis Set H (a infinite size)

linear formula $h \in H$: (lines or hyperplanes in R^d) weighted 'score' (linear classifier)

$h(x) = \text{sign}\left(\left(\sum_{n=1}^{d} w_i x_i\right) - \text{threshold}\right)$ (PS: Statistical learning: LDA & QDA Naive Bayes)

(perceptron hypothesis)

{positive: +1

negative: -1

Simplify to

$= \text{sign}\left(\left(\sum_{n=1}^{d} w_i x_i\right) + (-\text{threshold}) \cdot (+1)\right)$ (w_0, x_0)

$= \text{sign}\left(\sum_{i=0}^{d} w_i x_i\right)$

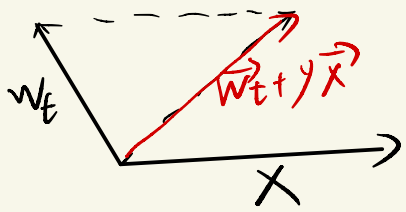
$= \text{sign}(w^T x)$

Select g from H

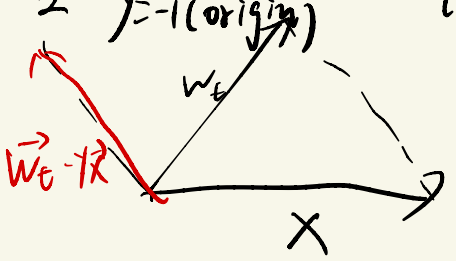
iterate from g_0 by its weight vector $\vec{w}_0$ ($\vec{0}$)

if $\text{sign}(\vec{w}_t^T \vec{x}_{n(t)}) \neq y_{n(t)}$ ($(\vec{x}_{n(t)}, y_{n(t)})$ is one point that not be classified correctly)

1° $y=1$ (origin)



2° $y=-1$ (origin)



Correct : $\vec{w}_{t+1} \leftarrow \vec{w}_t + y_{n(t)} \vec{x}_{n(t)}$

until no more mistakes $\text{sign}(\vec{w}_t^T \vec{x}_{n(t)}) = y_{n(t)}$ for all points.
(eg.: cyclic PCA)

return last $\vec{w}$ ($\vec{w}_{PLA}$) as g.

A fault confessed is half redressed. 知錯能改 善莫大焉

$y_n \vec{w}_{t+1}^T \vec{x}_n > y_n \vec{w}_t^T \vec{x}_n$

$\Rightarrow$ PLA could iteratively make $h(x)$ higher (score minus threshold)

(1) If PLA halts, D is linear separable.

⇕

exists perfect w_f such that $y_n = \text{sign}(w_f^T x_n)$

$\Leftrightarrow$ $y_{n(t)} \vec{W}_f^T \vec{X}_{n(t)} > \min_n y_n w_f^T x_n > 0$

+/− distance

even min distance is > 0, which means the line is perfectly seperate two kinds of things.

Suppose $(X_{n(t)}, y_{n(t)})$ is a point during the iterative process when PLA.

("dot product"'s value ↑ the near relationship)

$$\vec{W}_f^T \vec{W}_{t+1} = \vec{W}_f^T (\vec{W}_t + y_{n(t)} \vec{X}_{n(t)})$$

$$\geq \vec{W}_f^T \vec{W}_t + \min_n y_n \vec{W}_f^T \vec{X}_n$$

$$\geq W_f^T \vec{W}_t + 0$$

(but the scale of scalar is not considered now)

(2) $\text{sign}(\vec{w}_t^T \vec{x}_{n(t)}) \neq \vec{y}_{n(t)} \Leftrightarrow y_{n(t)} \vec{w}_t^T \vec{x}_{n(t)} \leq 0$.

$$\|w_{t+1}\|^2 = \|\vec{w}_t + y_{n(t)} \vec{x}_{n(t)}\|^2$$

$$= \|\vec{w}_t\|^2 + 2 y_{n(t)} \vec{w}_t^T x_{n(t)} + \|y_{n(t)} \vec{x}_{n(t)}\|^2$$

$$\leq \|\vec{w}_t\|^2 + 0 + \|y_{n(t)} x_{n(t)}\|^2$$

$\Rightarrow$ mistakes limits $\|w_t\|^2$ growth. rely on $\|y_{n(t)} x_{n(t)}\|^2 \leq \max_n \|y_n x_n\|^2$

So (1) + (2) $\Rightarrow$ $\frac{\vec{w}_f^T \vec{w}_T}{\|\vec{w}_f\| \|\vec{w}_T\|} \geq \sqrt{T} \cdot \text{constant} \cdot \cos\theta$

(T is the number of mistake corrections)

So it could not $= 1$

So the PLA will halt.

See qmd: $\gamma = \text{Constant}$

$$\min_n \frac{y_n (\vec{w}_T^T \cdot x_n)}{\|\vec{w}_T\| \|x_n\|}$$

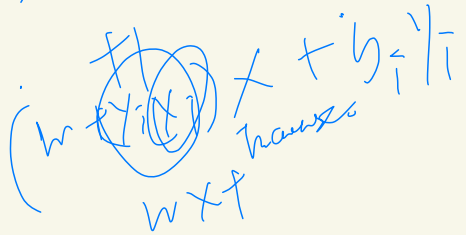
(margin)

Linear SVM

$h(\vec{x}) = 5 \cdot \text{sign}(s)$

$w x + b$. 0. →

$w + y_i x_i$ $(0,0) + -1$ (



+ ,

→ sum

(only linear) / converges to a solution

PLA

↓ not converge. $h(x) = \text{sign}(w \cdot x + b)$

PLA hypo.

$y_i = \text{Sign}(w \cdot x_i + b)$

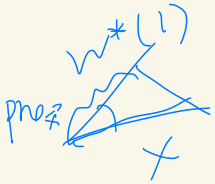
$w = [0, 0, \ldots, 0]$, $b = 0$

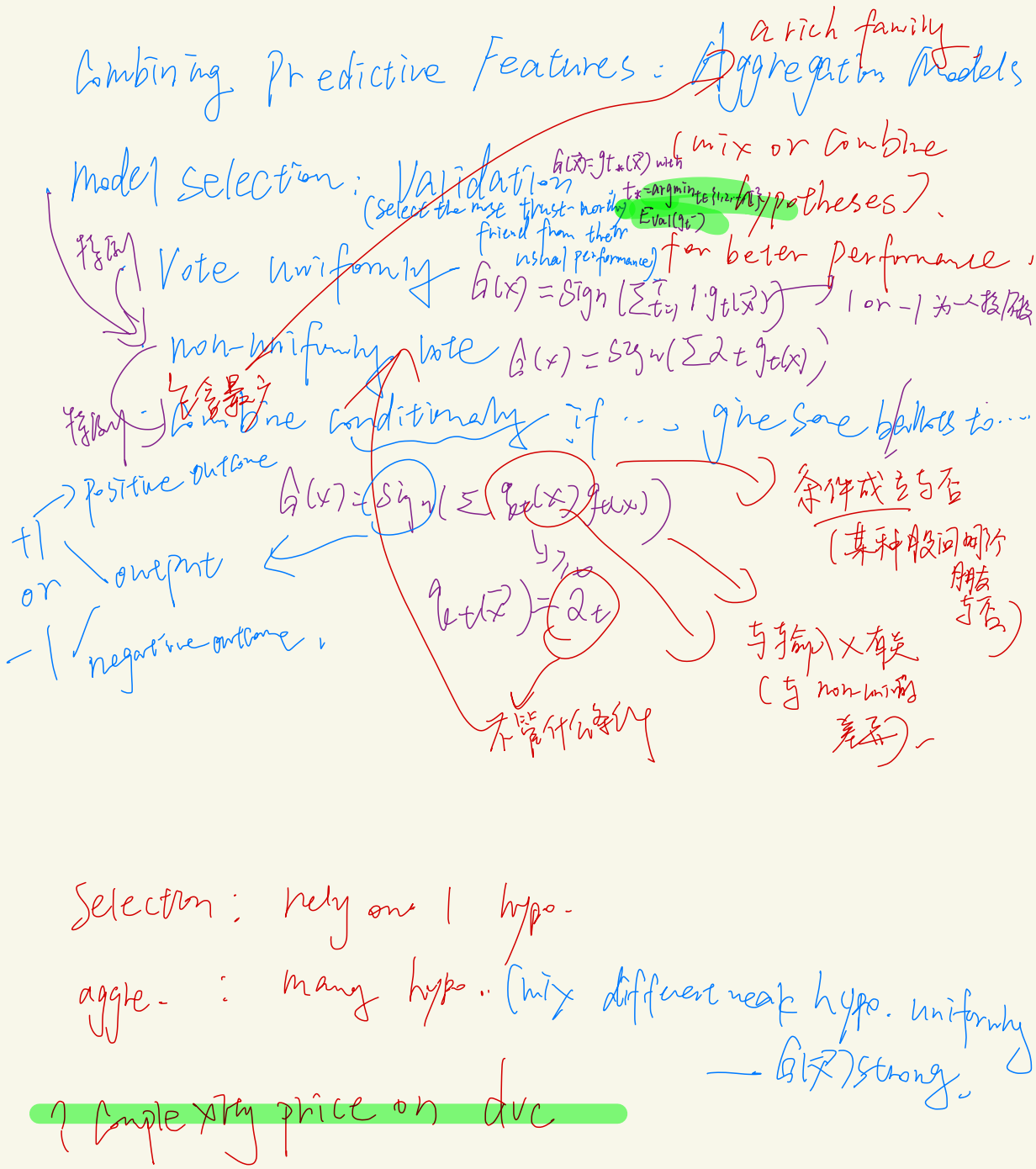
↓ iteratively adjust to separate.

+ , − eqs.

update rule: $w \leftarrow w + y_i x_i$ make x_i near to ~~o~~ reality

$b \leftarrow b + y_i$ ↑ or ↓





Selection: rely on 1 hypo.

aggre.: many hypo. (mix different weak hypo. uniformly → 自己strong.

? Complexity price on dvc

Machine Learning, 24, 123–140 (1996)
© 1996 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Bagging Predictors

LEO BREIMAN leo@stat.berkeley.edu
Statistics Department, University of California, Berkeley, CA 94720

Editor: Ross Quinlan

Abstract. Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class. The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets. Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy. The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy.

Keywords: Aggregation, Bootstrap, Averaging, Combining

1. Introduction

A learning set of $\mathcal{L}$ consists of data $\{(y_n, \boldsymbol{x}_n), n = 1, \ldots, N\}$ where the y's are either class labels or a numerical response. Assume we have a procedure for using this learning set to form a predictor $\varphi(\boldsymbol{x}, \mathcal{L})$ — if the input is $\boldsymbol{x}$ we predict y by $\varphi(\boldsymbol{x}, \mathcal{L})$. Now, suppose we are given a sequence of learnings sets $\{\mathcal{L}_k\}$ each consisting of N independent observations from the same underlying distribution as $\mathcal{L}$. Our mission is to use the $\{\mathcal{L}_k\}$ to get a better predictor than the single learning set predictor $\varphi(\boldsymbol{x}, \mathcal{L})$. The restriction is that all we are allowed to work with is the sequence of predictors $\{\varphi(\boldsymbol{x}, \mathcal{L}_k)\}$.

If y is numerical, an obvious procedure is to replace $\varphi(\boldsymbol{x}, \mathcal{L})$ by the average of $\varphi(\boldsymbol{x}, \mathcal{L}_k)$ over k, i.e. by $\varphi_A(\boldsymbol{x}) = E_{\mathcal{L}}\varphi(\boldsymbol{x}, \mathcal{L})$ where $E_{\mathcal{L}}$ denotes the expectation over $\mathcal{L}$, and the subscript A in φ_A denotes aggregation. If $\varphi(\boldsymbol{x}, \mathcal{L})$ predicts a class $j \in \{1, \ldots, J\}$, then one method of aggregating the $\varphi(\boldsymbol{x}, \mathcal{L}_k)$ is by voting. Let $N_j = nr\{k; \varphi(\boldsymbol{x}, \mathcal{L}_k) = j\}$ and take $\varphi_A(\boldsymbol{x}) = \text{argmax}_j N_j$, that is, the j for which N_j is maximum.

Usually, though, we have a single learning set $\mathcal{L}$ without the luxury of replicates of $\mathcal{L}$. Still, an imitation of the process leading to φ_A can be done. Take repeated bootstrap samples $\{\mathcal{L}^{(B)}\}$ from $\mathcal{L}$, and form $\{\varphi(\boldsymbol{x}, \mathcal{L}^{(B)})\}$. If y is numerical, take φ_B as

$$\varphi_B(\boldsymbol{x}) = av_B\varphi(\boldsymbol{x}, \mathcal{L}^{(B)}).$$

If y is a class label, let the $\{\varphi(\boldsymbol{x}, \mathcal{L}^{(B)})\}$ vote to form $\varphi_B(\boldsymbol{x})$. We call this procedure "**b**ootstrap **agg**rega**t**ing" and use the acronym **bagging**.

The $\{\mathcal{L}^{(B)}\}$ form replicate data sets, each consisting of N cases, drawn at random, *but with replacement*, from $\mathcal{L}$. Each $(y_n, \boldsymbol{x}_n)$ may appear repeated times or not at all in any particular $\mathcal{L}^{(B)}$. The $\{\mathcal{L}^{(B)}\}$ are replicate data sets drawn from the bootstrap distribution approximating the distribution underlying $\mathcal{L}$. For background on bootstrapping, see Efron

and Tibshirani [1993]. A critical factor in whether bagging will improve accuracy is the stability of the procedure for constructing φ. If changes in $\mathcal{L}$, i.e. a replicate $\mathcal{L}$, produces small changes in φ, then φ_B will be close to φ. Improvement will occur for unstable procedures where a small change in $\mathcal{L}$ can result in large changes in φ. Instability was studied in Breiman [1994] where it was pointed out that neural nets, classification and regression trees, and subset selection in linear regression were unstable, while k-nearest neighbor methods were stable.

For unstable procedures bagging works well. In Section 2 we bag classification trees on a variety of data sets. The reduction in test set misclassification rates ranges from 6% to 77%. In Section 3 regression trees are bagged with reduction in test set mean squared error on data sets ranging from 21% to 46%. Section 4 goes over some theoretical justification for bagging and attempts to understand when it will or will not work well. This is illustrated by the results of Section 5 on subset selection in linear regression using simulated data. Section 6 gives concluding remarks. These discuss how many bootstrap replications are useful, bagging nearest neighbor classifiers and bagging class probability estimates. The Appendix gives brief descriptions of the data sets.

The evidence, both experimental and theoretical, is that bagging can push a good but unstable procedure a significant step towards optimality. On the other hand, it can slightly degrade the performance of stable procedures. There has been recent work in the literature with some of the flavor of bagging. In particular, there has been some work on averaging and voting over multiple trees. Buntine [1991] gave a Bayesian approach, Kwok and Carter [1990] used voting over multiple trees generated by using alternative splits, and Heath et al. [1993] used voting over multiple trees generated by alternative oblique splits. Dietterich and Bakiri [1991] showed that a method for coding many class problems into a large number of two class problems increases accuracy. There is some commonality of this idea with bagging.

2. Bagging Classification Trees

2.1. *Results for Moderate Sized Data Sets*

Bagging was applied to classification trees using the following data sets

waveform (simulated)
heart
breast cancer (Wisconsin)
ionosphere
diabetes
glass
soybean

All of these except the heart data are in the UCI repository (ftp to ics.uci.edu/pub/machine-learning-databases). Table 1 gives a numerical summary of the data sets and brief descriptions are contained in the Appendix:

Table 1. Data Set Summary

Data Set	# Samples	# Variables	# Classes
waveform	300	21	3
heart	1395	16	2
breast cancer	699	9	2
ionosphere	351	34	2
diabetes	768	8	2
glass	214	9	6
soybean	683	35	19

In all runs the following procedure was used:

i) The data set is randomly divided into a test set $\mathcal{T}$ and a learning set $\mathcal{L}$. In the real data sets $\mathcal{T}$ is 10% of the data. In the simulated waveform data, 1800 samples are generated. $\mathcal{L}$ consists of 300 of these, and $\mathcal{T}$ the remainder.

ii) A classification tree is constructed from $\mathcal{L}$ using 10-fold cross-validation. Running the test set $\mathcal{T}$ down this tree gives the misclassification rate $e_S(\mathcal{L}, \mathcal{T})$.

iii) A bootstrap sample $\mathcal{L}_B$ is selected from $\mathcal{L}$, and a tree grown using $\mathcal{L}_B$. The original learning set $\mathcal{L}$ is used as test set to select the best pruned subtree (see Section 4.3). This is repeated 50 times giving tree classifiers $\phi_1(\boldsymbol{x}), \ldots, \phi_{50}(\boldsymbol{x})$.

iv) If $(j_n, \boldsymbol{x}_n) \in \mathcal{T}$, then the estimated class of $\boldsymbol{x}_n$ is that class having the plurality in $\phi_1(\boldsymbol{x}_n), \ldots, \phi_{50}(\boldsymbol{x}_n)$. If there is a tie, the estimated class is the one with the lowest class label. The proportion of times the estimated class differs from the true class is the bagging misclassification rate $e_B(\mathcal{L}, \mathcal{T})$.

v) The random division of the data into $\mathcal{L}$ and $\mathcal{T}$ is repeated 100 times and the reported $\bar{e}_S, \bar{e}_B$ are the averages over the 100 iterations. For the waveform data, 1800 new cases are generated at each iteration. Standard errors of $\bar{e}_S$ and $\bar{e}_B$ over the 100 iterations are also computed.

Table 2 gives the values of $\bar{e}_S$, $\bar{e}_B$, and Table 3 their estimated standard errors.

Table 2. Misclassification Rates (%)

Data Set	$\bar{e}_S$	$\bar{e}_B$	Decrease
waveform	29.1	19.3	34%
heart	4.9	2.8	43%
breast cancer	5.9	3.7	37%
ionosphere	11.2	7.9	29%
diabetes	25.3	23.9	6%
glass	30.4	23.6	22%
soybean	8.6	6.8	21%

Table 3. Standard Errors of Misclassification

Data Set	$SE(\bar{e}_S)$	$SE(\bar{e}_B)$
waveform	.2	.1
heart	.2	.1
breast cancer	.3	.2
ionosphere	.5	.4
diabetes	.4	.4
glass	1.1	.9
soybean	.4	.3

For the waveform data, its known that the lowest possible error rate is 14%. Bagging reduces the excess error by about two-thirds. We conjecture that the small decrease in the diabetes data set is because bagging is pushing close to the minimal attainable error rate. For instance, in the comparison by Michie et al. [1994] of 22 classifiers on this data set, the smallest error rate achieved (estimated by 12-fold cross-validation) was 22.3%.

2.2. *Statlog Comparisons for Larger Data Sets*

The Statlog Project [Michie et al., 1994] compared 22 classification methods over a wide variety of data sets. For most of these data sets, error rates were estimated using a single cross-validation. Without knowing the random subdivisions used in these cross-validations, the variability in the resulting error estimates makes comparisons chancey.

However, there were larger data sets used in the project which were divided into training and test sets. Four are publically available, and we used these as a basis for comparison. They can be accessed by ftp to ftp.strath.ac.uk and are described both in the Michie et al. [1994] book and in the data repository. Their numerical characteristics are given in Table 4 with brief descriptions in the Appendix.

Table 4. Statlog Data Set Summary

Data Set	#Training	#Variables	#Classes	#Test Set
letters	15,000	16	26	5000
satellite	4,435	36	6	2000
shuttle	43,500	9	7	14,500
DNA	2,000	60	3	1186

In each data set, a random 10% of the training set was set aside and a tree grown on the other 90%. The set aside 10% was then used to select the best pruned subtree. In bagging, 50 bootstrap replicates of the training set were generated and a large tree grown on each one. The original training set is used to select the best pruned subtree (see Section 4.3). The test set errors are listed in Table 5.

Table 5. Test Set Misclassification Rates (%)

Data Set	e_S	e_B	Decrease
letters	12.6	6.4	49%
satellite	14.8	10.3	30%
shuttle	.062	.014	77%
DNA	6.2	5.0	19%

Compared to the 22 classifiers in the Statlog Project, bagged trees ranked 2nd in accuracy on the DNA data set, 1st on the shuttle, 2nd on the satellite and 1st on letters. Following the Statlog method of ordering classifiers by their average rank, bagged trees was the top classifier on these four data sets with an average rank of 1.8. The next highest of the 22 has an average rank of 6.3. Average ranks for well-known classifiers are given in Table 6.

Table 6. Average Ranks of Classifiers

Algorithm	Average Rank
Radial Basis Functions	8.0
K-NN	8.5
$C4.5$	9.3
Quad. Discriminant	10.8
Neural Net	12.3

Some of the misclassification rates for the CART algorithm in Table 5 differ from those listed in the Statlog results. Possible sources for these differences are:

i) Different strategies may have been used to grow and prune the tree. I used the 90%-10% method specified above. Its not clear what was done in the Statlog project.

ii) An important setting is the minimum node size. This setting is not specified in the Statlog project. We used a minimum node size of one throughout.

iii) In the DNA data, different preprocessing of the input variable was used (see Appendix).

3. Bagging Regression Trees

Bagging trees was used on five data sets with numerical responses.

Boston Housing	
Ozone	
Friedman #1	(simulated)
Friedman #2	(simulated)
Friedman #3	(simulated)

Table 7. Summary of Data Sets

Data Set	#Cases	# Variables	# Test Set
Boston Housing	506	12	51
Ozone	330	8	33
Friedman #1	200	10	1000
Friedman #2	200	4	1000
Friedman #3	200	4	1000

A summary of these data sets is given in Table 7.

Brief descriptions of the data are in the Appendix. A procedure similar to that used in classification was followed:

i) Each real data set is randomly divided into a test set of consisting of 10% of the data and a learning set $\mathcal{L}$ consisting of the other 90%. In the 3 simulated data sets, 1200 cases are generated, 200 used as learning and 1000 as test.

ii) A regression tree is constructed from $\mathcal{L}$ using 10-fold cross-validation. Running the test set down this tree gives the squared error $e_S(\mathcal{L}, \mathcal{T})$.

iii) A bootstrap sample $\mathcal{L}_B$ is selected from $\mathcal{L}$ and a tree grown using $\mathcal{L}_B$ and $\mathcal{L}$ used to select the pruned subtree. This is repeated 25 times giving tree predictors $\phi_1(\boldsymbol{x}), \ldots, \phi_{25}(\boldsymbol{x})$.

iv) For $(y_n, \boldsymbol{x}_n) \in \mathcal{T}$, the bagged predictor is $\hat{y}_n = av_k \phi_k(\boldsymbol{x}_n)$, and the squared error $e_B(\mathcal{L}, \mathcal{T})$ is $av_n(y_n - \hat{y}_n)^2$

v) The random division of the data into $\mathcal{L}$ and $\mathcal{T}$ is repeated 100 times and the errors averaged to give $\bar{e}_S$, $\bar{e}_B$. For the simulated data, the 1200 cases are newly generated for each repetition.

Table 8 lists the values of $\bar{e}_S$, $\bar{e}_B$, and Table 9 gives their estimated standard errors.

Table 8. Mean Squared Test Set Error

Data Set	$\bar{e}_S$	$\bar{e}_B$	Decrease
Boston Housing	20.0	11.6	42%
Ozone	23.9	18.8	21%
Friedman #1	11.4	6.1	46%
Friedman #2	31,100	22,100	29%
Friedman #3	.0403	.0242	40%

Table 9. Standard Errors

Data Set	$SE(\bar{e}_S)$	$SE(\bar{e}_B)$
Boston Housing	1.0	.6
Ozone	.8	.6
Friedman #1	.10	.06
Friedman #2	300	100
Friedman #3	.0005	.0003

4. Why Bagging Works

4.1. Numeric Prediction

Let each $(y, \boldsymbol{x})$ case in $\mathcal{L}$ be independently drawn from the probability distribution P. Suppose y is numerical and $\phi(\boldsymbol{x}, \mathcal{L})$ the predictor. Then the aggregated predictor is the average over $\mathcal{L}$ of $\phi(\boldsymbol{x}, \mathcal{L})$, i.e.

$$\phi_A(\boldsymbol{x}) = E_{\mathcal{L}}\phi(\boldsymbol{x}, \mathcal{L}).$$

Take $\boldsymbol{x}$ to be a fixed input value and y an output value. Then

$$E_{\mathcal{L}}(y - \phi(\boldsymbol{x}, \mathcal{L}))^2 = y^2 - 2yE_{\mathcal{L}}\phi(\boldsymbol{x}, \mathcal{L}) + E_{\mathcal{L}}\phi^2(\boldsymbol{x}, \mathcal{L}). \tag{4.1}$$

Using $E_{\mathcal{L}}\phi(x, \mathcal{L}) = \phi_A(\boldsymbol{x})$ and applying the inequality $EZ^2 \geq (EZ)^2$ to the third term in (4.1) gives

$$E_{\mathcal{L}}(y - \phi(\boldsymbol{x}, \mathcal{L}))^2 \geq (y - \phi_A(\boldsymbol{x}))^2. \tag{4.2}$$

Integrating both sides of (4.2) over the joint $y, \boldsymbol{x}$ output-input distribution, we get that the mean-squared error of $\phi_A(\boldsymbol{x})$ is lower than the mean-squared error averaged over $\mathcal{L}$ of $\phi(\boldsymbol{x}, \mathcal{L})$.

How much lower depends on how unequal the two sides of

$$[E_{\mathcal{L}}\varphi(\boldsymbol{x}, \mathcal{L})]^2 \leq E_{\mathcal{L}}\varphi^2(\boldsymbol{x}, \mathcal{L})$$

are. The effect of instability is clear. If $\varphi(\boldsymbol{x}, \mathcal{L})$ does not change too much with replicate $\mathcal{L}$ the two sides will be nearly equal, and aggregation will not help. The more highly variable the $\varphi(\boldsymbol{x}, \mathcal{L})$ are, the more improvement aggregation may produce. But φ_A always improves on φ.

Now ϕ_A depends not only on $\boldsymbol{x}$ but also the underlying probability distribution P from which $\mathcal{L}$ is drawn, i.e. $\phi_A = \phi_A(\boldsymbol{x}, P)$. But the bagged estimate is not $\varphi_A(\boldsymbol{x}, P)$, but rather

$$\varphi_B(\boldsymbol{x}) = \varphi_A(\boldsymbol{x}, P_{\mathcal{L}}),$$

where $P_{\mathcal{L}}$ is the distribution that concentrates mass $1/N$ at each point $(y_n, \boldsymbol{x}_n) \in \mathcal{L}$, ($P_{\mathcal{L}}$ is called the bootstrap approximation to P). Then φ_B is caught in two currents: on the one hand, if the procedure is unstable, it can give improvement through aggregation. On the

other side, if the procedure is stable, then $\varphi_B = \varphi_A(\boldsymbol{x}, P_{\mathcal{L}})$ will not be as accurate for data drawn from P as $\varphi_A(\boldsymbol{x}, P) \simeq \varphi(\boldsymbol{x}, \mathcal{L})$.

There is a cross-over point between instability and stability at which φ_B stops improving on $\varphi(\boldsymbol{x}, \mathcal{L})$ and does worse. This has a vivid illustration in the linear regression subset selection example in the next section. There is another obvious limitation of bagging. For some data sets, it may happen that $\varphi(\boldsymbol{x}, \mathcal{L})$ is close to the limits of accuracy attainable on that data. Then no amount of bagging will do much improving. This is also illustrated in the next section.

4.2. *Classification*

In classification, a predictor $\varphi(\boldsymbol{x}, \mathcal{L})$ predicts a class label $j \in \{1, \ldots, J\}$. Denote

$$Q(j|\boldsymbol{x}) = P(\phi(\boldsymbol{x}, \mathcal{L}) = j).$$

The interpretation of $Q(j|\boldsymbol{x})$ is this: over many independent replicates of the learning set $\mathcal{L}$, ϕ predicts class label j at input $\boldsymbol{x}$ with relative frequency $Q(j|\boldsymbol{x})$. Let $P(j|\boldsymbol{x})$ be the probability that input $\boldsymbol{x}$ generates class j. Then the probability that the predictor classifies the generated state at $\boldsymbol{x}$ correctly is

$$\sum_j Q(j|\boldsymbol{x})P(j|\boldsymbol{x}). \tag{4.3}$$

The overall probability of correct classification is

$$r = \int [\sum_j Q(j|\boldsymbol{x})P(j|\boldsymbol{x})]P_X(d\boldsymbol{x})$$

where $P_X(d\boldsymbol{x})$ is the $\boldsymbol{x}$ probability distribution.

Looking at (4.3) note that for any $Q(j|\boldsymbol{x})$,

$$\sum_j Q(j|\boldsymbol{x})P(j|\boldsymbol{x}) \leq \max_j P(j|\boldsymbol{x})$$

with equality only if

$$Q(j|\boldsymbol{x}) = \begin{cases} 1 & \text{if } P(j|\boldsymbol{x}) = \max_i P(i|\boldsymbol{x}) \\ 0 & \text{else} \end{cases}.$$

The predictor $\phi^*(\boldsymbol{x}) = \text{argmax}_j P(j|\boldsymbol{x})$ (known as the Bayes predictor) leads to the above expression for $Q(j|\boldsymbol{x})$ and gives the highest attainable correct classification rate:

$$r^* = \int \max_j P(j|\boldsymbol{x})P_X(\boldsymbol{x}).$$

Call ϕ order-correct at the input $\boldsymbol{x}$ if

$$\text{argmax}_j Q(j|\boldsymbol{x}) = \text{argmax}_j P(j|\boldsymbol{x}).$$

This means that if input $\boldsymbol{x}$ results in class j more often than any other class, then ϕ also predicts class j at $\boldsymbol{x}$ more often than any other class. An order-correct predictor is not necessarily an accurate predictor. For instance, in a two class problem, suppose $P(1|\boldsymbol{x}) = .9$, $P(2|\boldsymbol{x}) = .1$ and $Q(1|\boldsymbol{x}) = .6$, $Q(2|\boldsymbol{x}) = .4$. Then the probability of correct classification by ϕ at $\boldsymbol{x}$ is .58, but the Bayes predictor gets correct classification with probability .90.

The aggregated predictor is: $\phi_A(\boldsymbol{x}) = \text{argmax}_j Q(j|\boldsymbol{x})$. For the aggregated predictor the probability of correct classification at $\boldsymbol{x}$ is

$$\sum_j I(\text{argmax}_i Q(i|\boldsymbol{x}) = j) P(j|\boldsymbol{x}) \tag{4.4}$$

where $I(\cdot)$ is the indicator function. If ϕ is order-correct at $\boldsymbol{x}$, then (4.4) equals $\max_j P(j|\boldsymbol{x})$. Letting C be the set of all inputs $\boldsymbol{x}$ at which ϕ is order-correct, we get for the correct classification probability of ϕ_A the expression

$$r_A = \int_{\boldsymbol{x}\in C} \max_j P(j|\boldsymbol{x}) P_X(d\boldsymbol{x}) + \int_{\boldsymbol{x}\in C'} [\sum_j I(\phi_A(\boldsymbol{x}) = j) P(j|\boldsymbol{x})] P_X(\boldsymbol{x}).$$

Even if ϕ is order-correct at $\boldsymbol{x}$ its correct classification rate can be far from optimal. But ϕ_A is optimal. If a predictor is good in the sense that it is order-correct for most inputs $\boldsymbol{x}$, then aggregation can transform it into a nearly optimal predictor. On the other hand, unlike the numerical prediction situation, poor predictors can be transformed into worse ones. The same behavior regarding stability holds. Bagging unstable classifiers usually improves them. Bagging stable classifiers is not a good idea.

4.3. Using the Learning Set as a Test Set

In bagging trees, the training set $\mathcal{L}_B$ is generated by sampling from the distribution $P_{\mathcal{L}}$. Using $\mathcal{L}_B$ a large tree T is constructed. The standard CART methodology finds the sequence of minimum cost pruned subtrees of T. The "best" pruned subtree in this sequence is selected using either cross-validation or a test set.

The idea of a test set is that it is formed by independently sampling from the same underlying distribution that gave rise to the learning set. In the present context, a test set is formed by independently sampling from $P_{\mathcal{L}}$, i.e. we can get a test set by sampling *with replacement*, from the original learning set $\mathcal{L}$.

Consider sampling, with replacement, a large number of times N' from $\mathcal{L}$. If $k(n)$ is the number of times that $(y_n, \boldsymbol{x}_n)$ is selected, then the intuitive content of my argument is that $k(n)/k(n') \simeq 1$, i.e. each case $(y_n, \boldsymbol{x}_n)$ will be selected about the same number of times (ratio-wise) as any other case. Thus, using a very large test set sampled from $P_{\mathcal{L}}$ is equivalent to just using $\mathcal{L}$ as a test set.

A somewhat more convincing argument is this: if there are N cases $(y_n, \boldsymbol{x}_n)$ then the number of times any particular $(y_n, \boldsymbol{x}_n)$ is selected has a binomial distribution with $p = 1/N$, and N' trials. The expected number of times $(y_n, \boldsymbol{x}_n)$ is selected is $N'p = N'/N$. The standard deviation of the number of times $(y_k, \boldsymbol{x}_n)$ is selected is $\sqrt{N'pq} \simeq \sqrt{N'/N}$.

Thus, for all n,

$$k(n)/(N'/N) \simeq 1 + o(1).$$

The fact that $\mathcal{L}$ can be used as a test set for predictors grown on a bootstrap sample $\mathcal{L}_B$ is more generally useful than just in a tree predictor context. For instance, in neural nets early stopping depends on the use of a test set. Thus, in bagging neural nets, the optimal point of early stopping can be estimated using the original learning set as a test set.

5. A Linear Regression Illustration

5.1. *Forward Variable Selection*

Subset selection in linear regression gives an illustration of the points made in the previous section. With data of the form $\mathcal{L} = \{(y_n, \boldsymbol{x}_n), n = 1, \ldots, N\}$ where $\boldsymbol{x} = (x_1, \ldots, x_M)$ consists of M predictor variables, a popular prediction method consists of forming predictors $\varphi_1(\boldsymbol{x}), \ldots, \varphi_M(\boldsymbol{x})$ where φ_m is linear in $\boldsymbol{x}$ and depends on only m of the M x-variables. Then one of the $\{\varphi_m\}$ is chosen as the designated predictor. For more background, see Breiman and Spector [1992].

A common method for constructing the $\{\varphi_m\}$, and one that is used in our simulation, is forward variable entry. If the variables used in φ_k are $x_{m_1}, \ldots, x_{m_k}$, then for each $m \notin \{m_1, \ldots, m_k\}$ form the linear regression of y on $(x_{m_1}, \ldots, x_{m_k}, x_m)$, compute the residual sum-of-squares RSS(m) and take $x_{m_{k+1}}$ such that m_{k+1} minimizes RSS(m) and $\varphi_{k+1}(\boldsymbol{x})$ the linear regression based on $(x_{m_1}, \ldots, x_{m_{k+1}})$.

There are other forms of variable selection e.g. best subsets, backwards variable selection, and variants thereof. What is clear about all of them is that they are unstable procedures (see Breiman [1994]). The variables are competing for inclusion in the $\{\varphi_m\}$ and small changes in the data can cause large changes in the $\{\varphi_m\}$.

5.2. *Simulation Structure*

The simulated data used in this section are drawn from the model.

$$y = \sum_m \beta_m x_m + \epsilon$$

where ϵ is $N(0, 1)$ (normally distributed with mean zero and variance one). The number of variables $M = 30$ and the sample size is 60. The $\{x_m\}$ are drawn from a mean-zero joint normal distribution with $EX_iX_j = \rho^{|i-j|}$ and at each iteration, ρ is selected from a uniform distribution on $[0, 1]$.

It is known that subset selection is nearly optimal if there are only a few large non-zero β_m, and that its performance is poor if there are many small but non-zero β_m. To bridge the spectrum, three sets of coefficients are used. Each set of coefficients consists of three clusters; one is centered at $m = 5$, one at $m = 15$ and the other at $m = 25$. Each cluster is of the form

$$\beta_m = c[(h - |m - k|)^+]^2, \quad m = 1, \ldots, 30$$

where k is the cluster center, and $h = 1, 3, 5$ for the first, second and third set of coefficients respectively. Thus, for $h = 1$, there are only three non-zero $\{\beta_m\}$. For $h = 3$ there are 15 non-zero $\{\beta_m\}$, and for $h = 5$, there are 27 non-zero $\{\beta_m\}$, all relatively small. The normalizing constant c is taken so that the R^2 for the data is $\simeq .75$ where R is the correlation between the output variable y and the full least squares predictor.

For each set of coefficients, the following procedure was replicated 250 times:

i) Data $\mathcal{L} = \{(y_n, \boldsymbol{x}_n), n = 1, \ldots, 60\}$ was drawn from the model

$$y = \sum \beta_m x_m + \epsilon$$

where the $\{x_m\}$ were drawn from the joint normal distribution described above.

ii) Forward entry of variables was done using $\mathcal{L}$ to get the predictors $\varphi_1(\boldsymbol{x}), \ldots, \varphi_M(\boldsymbol{x})$. The mean-squared prediction error of each of these was computed giving $e_1, \ldots, e_M$.

iii) Fifty bootstrap replicates $\{\mathcal{L}^{(B)}\}$ of $\mathcal{L}$ were generated. For each of these, forward stepwise regression was applied to construct predictors $\{\varphi_1(\boldsymbol{x}, \mathcal{L}^{(B)}), \ldots, \varphi_M(\boldsymbol{x}, \mathcal{L}^{(B)})\}$. These were averaged over the $\mathcal{L}^{(B)}$ to give the bagged sequence $\varphi_1^{(B)}(\boldsymbol{x}), \ldots, \varphi_M^{(B)}(x)$. The prediction errors $e_1^{(B)}, \ldots, e_M^{(B)}$ for this sequence were computed.

These computed mean-squared-errors were averaged over the 250 repetitions to give two sequences $\{\bar{e}_m^{(S)}\}$, $\{\bar{e}_m^{(B)}\}$. For each set of coefficients, these two sequences are plotted vs. m in Figure 1a,b,c.

5.3. *Discussion of Simulation Results*

Looking at Figures 1a,b,c, an obvious result is that the most accurate bagged predictor is at least as good as the most accurate subset predictor. When $h = 1$ and subset selection is nearly optimal, there is no improvement. For $h = 3$ and 5 there is substantial improvement. This illustrates the obvious: bagging can improve only if the unbagged is not optimal.

The second point is less obvious. Note that in all three graphs there is a point past which the bagged predictors have larger prediction error than the unbagged. The explanation is this: linear regression using *all* variables is a fairly stable procedure. The stability decreases as the number of variables used in the predictor decreases. As noted in Section 4, for a stable procedure $\varphi_B = \varphi_A(\boldsymbol{x}, P_{\mathcal{L}})$ is not as accurate as $\varphi \simeq \varphi(\boldsymbol{x}, P)$. The higher values of $\bar{\varphi}_m^{(B)}$ for large m reflect this fact. As m decreases, the instability increases and there is a cross-over point at which $\varphi_m^{(B)}$ becomes more accurate than φ_m.

6. Concluding Remarks

6.1. *Bagging Class Probability Estimates*

Some classification methods estimate probabilities $\hat{p}(j|\boldsymbol{x})$ that an object with prediction vector $\boldsymbol{x}$ belongs to class j. Then the class corresponding to $\boldsymbol{x}$ is estimated as $\arg\max_j \hat{p}(j|\boldsymbol{x})$.

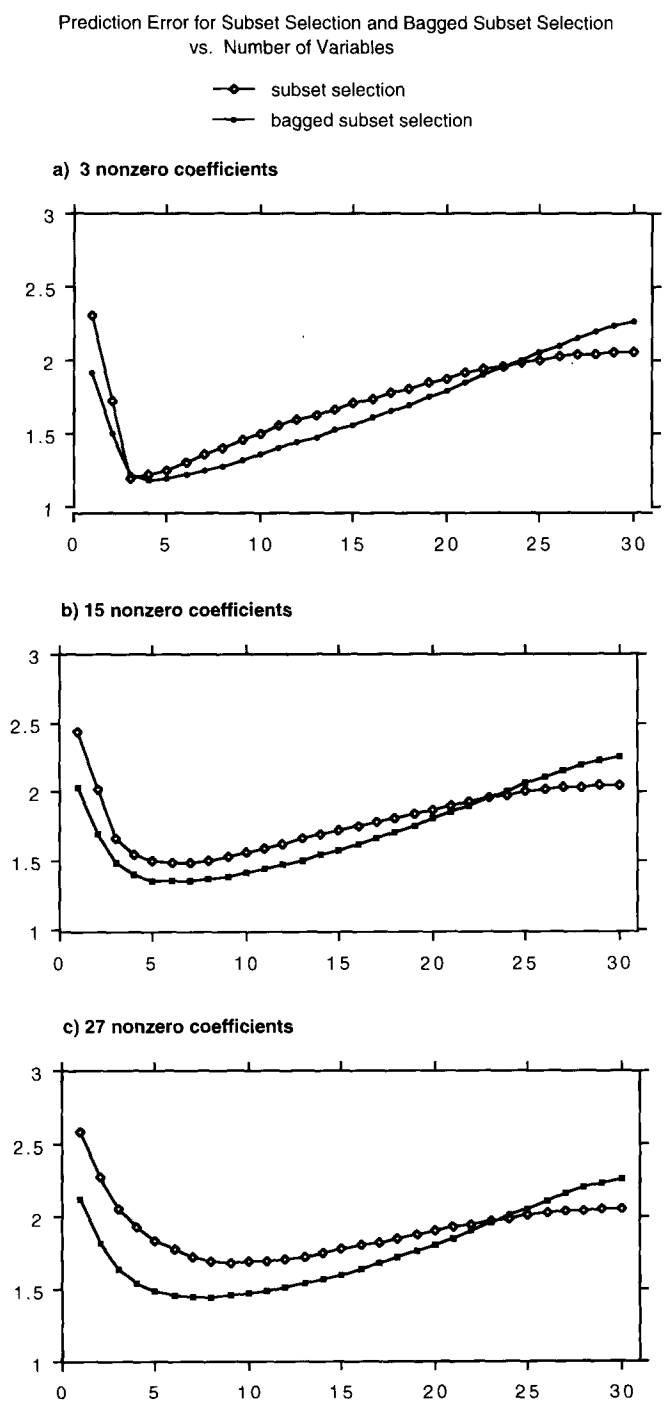


Figure 1. Prediction error for subset selection and bagged subset selection.

For such methods, a natural competitor to bagging by voting is to average the $\hat{p}(j|\boldsymbol{x})$ over all bootstrap replications, getting $\hat{p}_B(j|\boldsymbol{x})$, and then use the estimated class $\arg\max_j \hat{p}_B(j|\boldsymbol{x})$. This estimate was computed in every classification example we worked on. The resulting misclassification rate was always virtually identical to the voting misclassification rate.

In some applications, estimates of class probabilities are required instead of, or along with, the classifications. The evidence so far indicates that bagged estimates are likely to be more accurate than the single estimates. To verify this, it would be necessary to compare both estimates with the true values $p^*(j|\boldsymbol{x})$ over the $\boldsymbol{x}$ in the test set. For real data the true values are unknown. But they can be computed for the simulated waveform data, where they reduce to computing an expression involving error functions.

Using the waveform data, we did a simulation similar to that in Section 2 with learning and test sets both of size 300, and 25 bootstrap replications. In each iteration, we computed the average over the test set and classes of $|\hat{p}(j|\boldsymbol{x}) - p^*(j|\boldsymbol{x})|$ and $|\hat{p}_B(j|\boldsymbol{x}) - p^*(j|x)|$. This was repeated 50 times and the results averaged. The single tree estimates had an error of .189. The error of the bagged estimates was .124, a decrease of 34%.

6.2. *How Many Bootstrap Replicates Are Enough?*

In our experiments, 50 bootstrap replicates was used for classification and 25 for regression. This does not mean that 50 or 25 were necessary or sufficient, but simply that they seemed reasonable. My sense of it is that fewer are required when y is numerical and more are required with an increasing number of classes.

The answer is not too important when procedures like CART are used, because running times, even for a large number of bootstraps, are very nominal. But neural nets progress much more slowly and replications may require many days of computing. Still, bagging is almost a dream procedure for parallel computing. The construction of a predictor on each $\mathcal{L}^{(B)}$ proceeds with no communication necessary from the other CPU's.

To give some ideas of what the results are as connected with the number of bootstrap replicates we ran the waveform data using 10, 25, 50 and 100 replicates using the same simulation scheme as in Section 2. The results appear in Table 10.

Table 10. Bagged Misclassification Rates (%)

No. Bootstrap Replicates	Misclassification Rate
10	21.8
25	19.4
50	19.3
100	19.3

The unbagged rate is 29.1, so its clear that we are getting most of the improvement using only 10 bootstrap replicates. More than 25 bootstrap replicates is love's labor lost.

6.3. *How Big Should the Bootstrap Learning Set Be?*

In all of our runs we used bootstrap replicates $\mathcal{L}^{(B)}$ of the same size as the initial learning set $\mathcal{L}$. While a bootstrap replicate may have $2, 3, \ldots$ duplicates of a given instance, it also leaves out about .37 of the instances. A reader of the technical report on which this paper is based remarked that this was an appreciable loss of data, and that accuracy might improve if a larger bootstrap set was used. We experimented with bootstrap learning sets twice the size of $\mathcal{L}$. These left out about $e^{-2} = .14$ of the instances. There was no improvement in accuracy.

6.4. *Bagging Nearest Neighbor Classifiers*

Nearest neighbor classifiers were run on all the data sets described in Section 2 except for the soybean data whose variables were nominal. The same random division into learning and test sets was used with 100 bootstrap replicates, and 100 iterations in each run. A Euclidean metric was used with each coordinate standardized by dividing by its standard deviation over the learning set. See Table 11 for the results.

Table 11. Misclassification Rates for Nearest Neighbor

Data Set	$\bar{e}_S$	$\bar{e}_B$
waveform	26.1	26.1
heart	5.1	5.1
breast cancer	4.4	4.4
ionosphere	36.5	36.5
diabetes	29.3	29.3
glass	30.1	30.1

Nearest neighbor is more accurate than single trees in 3 of the 6 data sets, but bagged trees are more accurate in all of the 6 data sets.

Cycles did not have to be expended to find that bagging nearest neighbors does not change things. Some simple computations show why. Given N possible outcomes of a trial (the N cases $(y_n, \boldsymbol{x}_n)$ in the learning set) and N trials, the probability that the nth outcome is selected $0, 1, 2, \ldots$ times is approximately Poisson distributed with $\lambda = 1$ for large N. The probability that the nth outcome will occur at least once is $1 - (1/e) \simeq .632$.

If there are N_B bootstrap repetitions in a 2-class problem, then a test case may change classification only if its nearest neighbor in the learning set is not in the bootstrap sample in at least half of the N_B replications. This probability is given by the probability that the number of heads in N_B tosses of a coin with probability .632 of heads is less than $.5N_B$. As N_B gets larger, this probability gets very small. Analogous results hold for J-class problems.

The stability of nearest neighbor classification methods with respect to perturbations of the data distinguishes them from competitors such as trees and neural nets.

6.5. Conclusions

Bagging goes a ways toward making a silk purse out of a sow's ear, especially if the sow's ear is twitchy. It is a relatively easy way to improve an existing method, since all that needs adding is a loop in front that selects the bootstrap sample and sends it to the procedure and a back end that does the aggregation. What one loses, with the trees, is a simple and interpretable structure. What one gains is increased accuracy.

Appendix

Descriptions of Data Sets

A. Classification Data Sets

Waveform This is simulated 21 variable data with 300 cases and 3 classes each having probability 1/3. It is described in Breiman et al [1984] (a C subroutine for generating the data is in the UCI repository subdirectory/waveform).

Heart This is data from the study referred to in the opening paragraphs of the CART book (Breiman et. al. [1984]). To quote:

> At the University of California, San Diego Medical Center, when a heart attack patient is admitted, 19 variables are measured during the first 24 hours. These include blood pressure, age, and 17 other ordered and binary variables summarizing the medical symptoms considered as important indicators of the patient's condition.
>
> The goal of a recent medical study (see Chapter 6) was the development of a method to identify high risk patients (those who will not survive at least 30 days) on the basis of the initial 24-hour data.

The data base has also been studied in Olshen et al [1985]. It was gathered on a project (SCOR) headed by John Ross Jr. Elizabeth Gilpin and Richard Olshen were instrumental in my obtaining the data. The data used had 18 variables. Two variables with high proportions of had missing data were deleted, together with a few other cases that missing values. This left 779 complete cases — 77 deaths and 702 survivors. To equalize class sizes, each case of death was replicated 9 times giving 693 deaths for a total of 1395 cases.

Breast Cancer This is data given to the UCI repository by Willian H. Wolberg, University of Wisconsin Hospitals, Madison (see Wolberg and Mangasarian [1990]). It is two class data with 699 cases (458 benign and 241 malignant). It has 9 variables consisting of cellular characteristics.

Ionosphere This is radar data gathered by the Space Physics Group at Johns Hopkins University (see Sigillito et. al. [1989]). There are 351 cases with 34 variables, consisting of 2 attributes for each at 17 pulse numbers. There are two classes: good = some type of structure in the ionosphere (226); bad = no structure (125).

Diabetes This is a data base gathered among the Pima Indians by the National Institute of Diabetes and Digestive and Kidney Diseases. (See Smith et. al. [1988]). The data base consists of 768 cases, 8 variables and two classes. The variables are medical measurements on the patient plus age and pregnancy information. The classes are: tested positive for diabetes (268) or negative (500).

Glass This data base was created in the Central Research Establishment, Home Office Forensic Science Service Aldermaston, Reading, Berkshire. Each case consists of 9 chemical measurements on one of 6 types of glass. There are 214 cases.

Soybean The soybean data set consists of 683 cases, 35 variables and 19 classes. The classes are various types of soybean diseases. The variables are observations on the plants together with some climatic variables. All are nominal. Some missing values were filled in by their modal values.

Letters This data set was constructed by David J. Slate, Odesta Corporation. Binary pixel displays of the 26 capital English letters were created using 20 different fonts and then randomly distorted to create 20,000 images. Sixteen features, consisting of statistical moments and edge counts, were extracted from each image.

Satellite This is data extracted from Landsat images. For the 3×3 pixel area examined, intensity readings are given in 4 spectral bands for each pixel. The middle pixel is classified as one of 6 different soil types. The multiple authorship of the data set is explained in the documentation in the repository.

Shuttle This data set involves shuttle controls concerning the position of radiators within the Space Shuttle. The data originated from NASA and was provided to the archives by J. Catlett.

DNA The classes in this data set are types of boundaries in a spliced DNA sequence. The input variables consists of a window of 60 nucleotides each having one of 4 categorical values (A, G, C, T). The problem is to classify the middle point of the window as one of two types of boundaries or neither. The data is part of the Genbank and was donated by G. Towell, M. Noordewier and J. Shavlik.

Because some classifiers in the Statlog project could accept only numeric inputs, each of the 60 categoricals was coded into 3 binary variables, resulting in 180 input variables. For reasons not explained, some of the tree algorithms run on the DNA data were given the 60 categoricals as input while the CART algorithm was given the 180 binary inputs. In my runs the 60 categorical inputs were used.

B. Regression Data Sets

Boston Housing This data became well-known through its use in the book by Belsley, Kuh, and Welsch [1980]. It has 506 cases corresponding to census tracts in the greater Boston area. The y-variable is median housing price in the tract. There are 12 predictor

variables, mainly socio-economic. The data has since been used in many studies. (UCI repository/housing).

Ozone The ozone data consists of 366 readings of maximum daily ozone at a hot spot in the Los Angeles basin and 9 predictor variables — all meteorlogical, i.e. temperature, humidity, etc. It is described in Breiman and Friedman [1985] and has also been used in many subsequent studies. Eliminating one variable with many missing values and a few other cases leaves a data set with 330 complete cases and 8 variables.

Friedman #1 All three Friedman data sets are simulated data that appear in the MARS paper (Friedman [1991]). In the first data set, there are ten independent predictor variables $x_1, \ldots, x_{10}$ each of which is uniformly distributed over $[0, 1]$. The response is given by

$$y = 10 \sin(\pi x_1 x_2) + 20(x_3 - .5)^2 + 10x_4 + 5x_5 + \epsilon$$

where ϵ is $N(0, 1)$. Friedman gives results for this model for sample sizes 50, 100, 200. We use sample size 200.

Friedman #2, #3 These two examples are taken to simulate the impedance and phase shift in an alternating current circuit. They are 4 variable data with

#2 $$y = (x_1^2 + (x_2 x_3 - (1/x_2 x_4))^2)^{1/2} + \epsilon_2$$

#3 $$y = \tan^{-1}\left(\frac{x_2 x_3 - (1/x_2 x_4)}{x_1}\right) + \epsilon_3$$

where x_1, x_2, x_3, x_4 are uniformly distributed over the ranges

$$\begin{aligned} 0 &\leq x_1 \leq 100 \\ 20 &\leq (x_2/2\pi) \leq 280 \\ 0 &\leq x_3 \leq 1 \\ 1 &\leq x_4 \leq 11 \end{aligned}$$

The noise ϵ_2, ϵ_3 are distributed as $N(0, \sigma_2^2)$, $N(0, \sigma_3^2)$ with σ_2, σ_3 selected to give 3:1 signal/noise ratios.

References

Belsley, D., Kuh, E., & Welsch, R. (1980). "Regression Diagnostics", John Wiley and Sons.

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

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). "Classification and Regression Trees", Wadsworth.

Breiman, L. & Friedman, J. (1985). Estimating optimal transformations in multiple regression and correlation (with discussion), *Journal of the American Statistical Association*, **80**, 580-619 .

Breiman, L. & Spector, P (1992). Submodel Selection and Evaluation in Regression – the X-Random Case, *International Statistical Review*, **3**, 291-319

Buntine, W. (1991). "Learning classification trees", Artificial Intelligence Frontiers in Statistics, ed D.J. Hand, Chapman and Hall, London, 182-201.

Dietterich, T.G. & Bakiri, G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs, Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), Anaheim, CA: AAAI Press.

Efron, B., & Tibshirani, R. (1993). "An Introduction to the Bootstrap". Chapman and Hall.

Friedman, J. (1991). Multivariate adaptive regression splines (with discussion), *Annals of Statistics*, **19**, 1-141.

Heath, D., Kasif, S., & Salzberg, S. (1993). k-dt: a multi-tree learning method. Proceedings of the Second International Workshop on Multistrategy Learning, 1002-1007, Chambery, France, Morgan Kaufman.

Kwok, S., & Carter, C. (1990). Multiple decision trees, Uncertainty in Artificial Intelligence 4, ed. Shachter, R., Levitt, T., Kanal, L., and Lemmer, J., North-Holland, 327-335.

Michie, D., Spiegelhalter, D.J. & Taylor, C.C. (1994). Machine Learning, Neural and Statistical Classification. Ellis Horwood Limited.

Olshen, R., Gilpin, A., Henning, H., LeWinter, M., Collins, D., & Ross, J. (1985). Twelve-month prognosis following myocardial infarction: Classification trees, logistic regression, and stepwise linear discrimination, Proceedings of the Berkeley conference in honor of Jerzy Neyman and Jack Kiefer, L. Le Cam;R. Olshen, (Ed), Wadsworth, 245-267.

Smith, J., Everhart, J., Dickson, W., Knowler, W., & Johannes, R. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In Proceedings of the Symposium on Computer Applications and Medical Care 261–265. IEEE Computer Society Press.

Sigillito, V. G., Wing, S. P., Hutton, L. V., & Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, **10**, 262-266.

Wolberg, W. & Mangasarian, O (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology, Proceedings of the National Academy of Sciences, U.S.A., Volume 87, December 1990, pp 9193-9196.

Received September 19, 1994
Accepted January 2, 1995
Final Manuscript November 21, 1995

Adaptive Boosting.

Experiments with a New Boosting Algorithm

DRAFT — PLEASE DO NOT DISTRIBUTE

Yoav Freund Robert E. Schapire

AT&T Research
600 Mountain Avenue
Rooms {2B-428, 2A-424}
Murray Hill, NJ 07974-0636

{yoav, schapire}@research.att.com
http://www.research.att.com/orgs/ssr/people/{yoav,schapire}/

January 22, 1996

Abstract

In an earlier paper [9], we introduced a new "boosting" algorithm called **AdaBoost** which, theoretically, can be used to significantly reduce the error of any learning algorithm that consistently generates classifiers whose performance is a little better than random guessing. We also introduced the related notion of a "pseudo-loss" which is a method for forcing a learning algorithm of multi-label concepts to concentrate on the labels that are hardest to discriminate. In this paper, we describe experiments we carried out to assess how well **AdaBoost** with and without pseudo-loss, performs on real learning problems.

We performed two sets of experiments. The first set compared boosting to Breiman's [1] "bagging" method when used to aggregate various classifiers (including decision trees and single attribute-value tests). We compared the performance of the two methods on a collection of machine-learning benchmarks. In the second set of experiments, we studied in more detail the performance of boosting using a nearest-neighbor classifier on an OCR problem.

1 Introduction

"Boosting" is a general method for improving the performance of any learning algorithm. In theory, boosting can be used to significantly reduce the error of any "weak" learning algorithm that consistently generates classifiers which need only be a little bit better than random guessing. Despite the potential benefits of boosting promised by the theoretical results, the true practical value of boosting can only be assessed by testing the method on "real" learning problems. In this paper, we present such an experimental assessment of a new boosting algorithm called **AdaBoost**.

Boosting works by repeatedly running a given weak[1] learning algorithm on various distributions over the training data, and then combining the classifiers produced by the weak learner into a single composite classifier. The first provably effective boosting algorithms were presented by Schapire [19] and Freund [8]. More recently, we described and analyzed **AdaBoost**, and we argued that this new boosting algorithm has certain properties which make it more practical and easier to implement than its predecessors [9]. This algorithm, which we used in all our experiments, is described in detail in Section 2.

This paper describes two distinct sets of experiments. In the first set of experiments, described in Section 3, we compared boosting to "bagging," a method described by Breiman [1] which works in the same general fashion (i.e., by repeatedly rerunning a given weak learning algorithm, and combining the computed classifiers), but which constructs each distribution in a simpler manner. (Details given below.) We compared boosting with bagging because both methods work by combining many classifiers. This comparison allows us to separate out the effect of modifying the distribution on each round (which is done differently by each algorithm) from the effect of voting multiple classifiers (which is done the same by each).

In our experiments, we compared boosting to bagging using a number of different weak learning algorithms of varying levels of sophistication. These include: (1) an algorithm that searches for very simple prediction rules which test on a single attribute (similar to Holte's very simple classification rules [13]); (2) an algorithm that searches for a single good decision rule that tests on a conjunction of attribute tests (similar in flavor to the rule-formation part of Cohen's RIPPER algorithm [2] and Fürnkranz and Widmer's IREP algorithm [10]); and (3) Quinlan's **C4.5** decision-tree algorithm [17]. We tested these algorithms on a collection of 27 benchmark learning problems taken from the UCI repository.

The main conclusion of our experiments is that boosting performs significantly and uniformly better than bagging when the weak learning algorithm generates fairly simple classifiers (algorithms (1) and (2) above). When combined with **C4.5**, boosting still seems to outperform bagging slightly, but the results are less compelling.

We also found that boosting can be used with very simple rules (algorithm (1)) to construct classifiers that are quite good relative, say, to **C4.5**. Kearns and Mansour [15] argue that **C4.5** can itself be viewed as a kind of boosting algorithm, so a comparison of **AdaBoost** and **C4.5** can be seen as a comparison of two competing boosting algorithms. See Dietterich, Kearns and Mansour's paper [3] for more detail on this point.

In the second set of experiments, we test the performance of boosting on a nearest neighbor classifier for handwritten digit recognition. In this case the weak learning algorithm is very simple,

[1] We use the term "weak" learning algorithm, even though, in practice, boosting might be combined with a quite strong learning algorithm such as **C4.5**.

and this lets us gain some insight to the interaction between the boosting algorithm and the nearest neighbor classifier. We show that the boosting algorithm is an effective way for finding a small subset of prototypes that performs almost as well as the complete set. We also show that it compares favorably to the standard method of Condensed Nearest Neighbor [12] in terms of its test error.

There seem to be two separate reasons for the improvement in performance that is achieved by boosting. The first and better understood effect of boosting is that it generates a hypothesis whose error on the training set is small by combining many hypotheses whose error may be large (but still better than random guessing). It seems that boosting may be helpful on learning problems having either of the following two properties. The first property, which holds for many real-world problems, is that the observed examples tend to have varying degrees of hardness. For such problems, the boosting algorithm tends to generate distributions that concentrate on the harder examples, thus challanging the weak learning algorithm to perform well on these harder parts of the sample space.

The second property is that the learning algorithm be sensitive to changes in the training examples so that significantly different hypotheses are generated for different training sets. In this sense, boosting is similar to Breiman's bagging [1] which performs best when the weak learner exhibits such "unstable" behavior. However, unlike bagging, boosting tries actively to force the weak learning algorithm to change its hypotheses by constructing a "hard" distribution over the examples based on the performance of previously generated hypotheses.

The second effect of boosting has to do with variance reduction. Intuitively, taking a weighted majority over many hypotheses, all of which were trained on different samples taken out of the same training set, has the effect of reducing the random variability of the combined hypothesis. Thus, like bagging, boosting may have the effect of producing a combined hypothesis whose variance is significantly lower than those produced by the weak learner. However, unlike bagging, boosting may also reduce the bias of the learning algorithm, as discussed above. (See Kong and Dietterich [16] for further discussion of the bias and variance reducing effects of voting multiple hypotheses.) In our first set of experiments, we compare boosting and bagging, and try to use that comparison to separate between the bias and variance reducing effects of boosting.

Previous work. Drucker, Schapire and Simard [7, 6] performed the first experiments using a boosting algorithm. They used Schapire's [19] original boosting algorithm combined with a neural net for an OCR problem. Follow-up comparisons to other ensemble methods were done by Drucker et al. [5]. More recently, Drucker and Cortes [4] used **AdaBoost** with a decision-tree algorithm for an OCR task. Jackson and Craven [14] used **AdaBoost** to learn classifiers represented by sparse perceptons, and tested the algorithm on a set of benchmarks. Finally, Quinlan [18] recently conducted an independent comparison of boosting and bagging combined with **C4.5** on a collection of UCI benchmarks.

2 The boosting algorithm

In this section, we describe our boosting algorithm, called **AdaBoost**. See our earlier paper [9] for more details about the algorithm and its theoretical properties.

We describe two versions of the algorithm which we denote **AdaBoost.M1** and **AdaBoost.M2**. The two versions are equivalent for binary classification problems and differ only in their handling

Algorithm AdaBoost.M1

Input: sequence of m examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ with labels $y_i \in Y = \{1, \ldots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i.

Do for $t = 1, 2, \ldots, T$

1. Call **WeakLearn**, providing it with the distribution D_t.
2. Get back a hypothesis $h_t : X \rightarrow Y$.
3. Calculate the error of h_t: $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$. If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update distribution D_t: $D_{t+1}(i) = \dfrac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$
   where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log \dfrac{1}{\beta_t}$.

Figure 1: The algorithm **AdaBoost.M1**.

of problems with more than two classes.

2.1 AdaBoost.M1

We begin with the simpler version, **AdaBoost.M1**. The boosting algorithm takes as input a training set of m examples $S = \langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ where x_i is an instance drawn from some space X and represented in some manner (typically, a vector of attribute values), and $y_i \in Y$ is the class label associated with x_i. In this paper, we always assume that the set of possible labels Y is of finite cardinality k.

In addition, the boosting algorithm has access to another unspecified learning algorithm, called the weak learning algorithm, which is denoted generically as **WeakLearn**. The boosting algorithm calls **WeakLearn** repeatedly in a series of rounds. On round t, the booster provides **WeakLearn** with a distribution D_t over the training set S. In response, **WeakLearn** computes a classifier or *hypothesis* $h_t : X \rightarrow Y$ which should misclassify a non trivial fraction of the training examples, relative to D_t. That is, the weak learner's goal is to find a hypothesis h_t which minimizes the (training) error $\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$. Note that this error is measured with respect to the distribution D_t that was provided to the weak learner. This process continues for T rounds, and, at last, the booster combines the weak hypotheses $h_1, \ldots, h_T$ into a single final hypothesis h_{fin}.

Still unspecified are (1) the manner in which D_t is computed on each round, and (2) how h_{fin} is computed. Different boosting schemes answer these two questions in different ways. **AdaBoost.M1** uses the simple rule shown in Figure 1. The initial distribution D_1 is uniform over S so $D_1(i) = 1/m$ for all i. To compute distribution D_{t+1} from D_t and the last weak hypothesis h_t, we multiply the weight of example i by some number $\beta_t \in [0, 1)$ if h_t classifies x_i correctly, and otherwise the weight is left unchanged. The weights are then renormalized by dividing by the normalization constant Z_t. Effectively, "easy" examples that are correctly classified by many

Algorithm AdaBoost.M2

Input: sequence of m examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ with labels $y_i \in Y = \{1, \ldots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Let $B = \{(i, y) : i \in \{1, \ldots, m\}, y \neq y_i\}$
Initialize $D_1(i, y) = 1/|B|$ for $(i, y) \in B$.
Do for $t = 1, 2, \ldots, T$

1. Call **WeakLearn**, providing it with mislabel distribution D_t.
2. Get back a hypothesis $h_t : X \times Y \rightarrow [0, 1]$.
3. Calculate the pseudo-loss of h_t: $\epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y))$.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update D_t: $D_{t+1}(i, y) = \frac{D_t(i, y)}{Z_t} \cdot \beta_t^{(1/2)(1 + h_t(x_i, y_i) - h_t(x_i, y))}$
   where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t} \right) h_t(x, y)$.

Figure 2: The algorithm **AdaBoost.M2**.

of the previous weak hypotheses get lower weight, and "hard" examples which tend often to be misclassified get higher weight. Thus, **AdaBoost** focuses the most weight on the examples which seem to be hardest for **WeakLearn**.

The number β_t is computed as shown in the figure as a function of ϵ_t. The final hypothesis h_{fin} is a weighted vote (i.e. a weighted linear threshold) of the weak hypotheses. That is, for a given instance x, h_{fin} outputs the label y that maximizes the sum of the weights of the weak hypotheses predicting that label. The weight of hypothesis h_t is defined to be $\ln(1/\beta_t)$ so that greater weight is given to hypotheses with lower error.

The important theoretical property about **AdaBoost.M1** is stated in the following theorem. This theorem shows that if the weak hypotheses consistently have error only slightly better than $1/2$, then the error of the final hypothesis h_{fin} drops to zero exponentially fast. For binary classification problems, this means that the weak hypotheses need be only slightly better than random.

Theorem 1 ([9]) *Suppose the weak learning algorithm* **WeakLearn***, when called by* **AdaBoost.M1***, generates hypotheses with errors $\epsilon_1, \ldots, \epsilon_T$, where ϵ_t is as defined in Figure 1. Assume each $\epsilon_t \leq 1/2$, and let $\gamma_t = 1/2 - \epsilon_t$. Then the following upper bound holds on the error of the final hypothesis h_{fin}:*

$$\frac{1}{m} |\{i : h_{fin}(x_i) \neq y_i\}| \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq \exp\left(-2 \sum_{t=1}^{T} \gamma_t^2 \right).$$

The main disadvantage of **AdaBoost.M1** is that it is unable to handle weak hypotheses with error greater than $1/2$. The expected error of a hypothesis which randomly guesses the label is $1 - 1/k$, where k is the number of possible labels. Thus **AdaBoost.M1** requirement for $k = 2$ is that the prediction is just slightly better than random guessing. However, when $k > 2$, the requirement of **AdaBoost.M1** is much stronger than that, and might be hard to meet.

2.2 AdaBoost.M2

The second version of **AdaBoost** attempts to overcome this difficulty by extending the communication between the boosting algorithm and the weak learner. First, we allow the weak learner to generate more expressive hypotheses whose output is a vector in $[0,1]^k$, rather than a single label in Y. Intuitively, the yth component of this vector represents a "degree of belief" that the correct label is y. The components with values close to 1 or 0 correspond to those labels considered to be plausible or implausible, respectively.

While we give the weak learning algorithm more expressive power, we also place a more complex requirement on the performance of the weak hypotheses. Rather than using the usual prediction error, we ask that the weak hypotheses do well with respect to a more sophisticated error measure that we call the pseudo-loss. Unlike ordinary error which is computed with respect to a distribution over examples, pseudo-loss is computed with respect to a distribution over the set of all pairs of examples and incorrect labels. By manipulating this distribution, the boosting algorithm can focus the weak learner not only on hard-to-classify examples, but more specifically, on the incorrect labels that are hardest to discriminate. We will see that the boosting algorithm **AdaBoost.M2**, which is based on these ideas, achieves boosting if each weak hypothesis has pseudo-loss slightly better than random guessing.

More formally, a *mislabel* is a pair (i, y) where i is the index of a training example and y is an *incorrect* label associated with example i. Let B be the set of all mislabels:

$$B = \{(i, y) : i \in \{1, \ldots, m\}, y \neq y_i\}.$$

A *mislabel distribution* is a distribution defined over the set B of all mislabels.

On each round t of boosting, **AdaBoost.M2** (Figure 2) supplies the weak learner with a mislabel distribution D_t. In response, the weak learner computes a hypothesis h_t of the form $h_t : X \times Y \to [0, 1]$.

Intuitively, we interpret each mislabel (i, y) as representing a binary question of the form: "Do you predict that the label associated with example x_i is y_i (the correct label) or y (one of the incorrect labels)?" With this interpretation, the weight $D_t(i, y)$ assigned to this mislabel represents the importance of distinguishing incorrect label y on example x_i.

A weak hypothesis h_t is then interpreted in the following manner. If $h_t(x_i, y_i) = 1$ and $h_t(x_i, y) = 0$, then h_t has (correctly) predicted that x_i's label is y_i, not y (since h_t deems y_i to be "plausible" and y "implausible"). Similarly, if $h_t(x_i, y_i) = 0$ and $h_t(x_i, y) = 1$, then h_t has (incorrectly) made the opposite prediction. If $h_t(x_i, y_i) = h_t(x_i, y)$, then h_t's prediction is taken to be a random guess. (Values for h_t in $(0, 1)$ are interpreted probabilistically.)

This interpretation leads us to define the *pseudo-loss* of hypothesis h_t with respect to mislabel distribution D_t by the formula

$$\epsilon_t = \tfrac{1}{2} \sum_{(i,y) \in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y)).$$

Space limitations prevent us from giving a complete derivation of this formula which is explained in detail in our earlier paper [9]. It should be clear, however, that the pseudo-loss is minimized when correct labels y_i are given values near 1 and incorrect labels $y \neq y_i$ values near 0. Further, note that pseudo-loss $1/2$ is trivially achieved by any constant-valued hypothesis h_t, and moreover

that a hypothesis h_t with pseudo-loss greater than $1/2$ can be replaced by the hypothesis $1 - h_t$ whose pseudo-loss is less than $1/2$.

The weak learner's goal is to find a weak hypothesis h_t with small pseudo-loss. Thus, standard "off-the-shelf" learning algorithms may need some modification to be used in this manner, although this modification is often straightforward. After receiving h_t, the mislabel distribution is updated using a rule similar to the one used in **AdaBoost.M1**. The final hypothesis h_{fin} outputs, for a given instance x, the label y that maximizes a weighted average of the weak hypothesis values $h_t(x, y)$.

The following theorem gives a bound on the training error of the final hypothesis. Note that this theorem requires only that the weak hypotheses have pseudo-loss less than $1/2$, i.e., only slightly better than a trivial (constant-valued) hypothesis, regardless of the number of classes. Also, although the weak hypotheses h_t are evaluated with respect to the pseudo-loss, we of course evaluate the final hypothesis h_{fin} using the ordinary error measure.

Theorem 2 ([9]) *Suppose the weak learning algorithm* **WeakLearn***, when called by* **AdaBoost.M2** *generates hypotheses with pseudo-losses $\epsilon_1, \ldots, \epsilon_T$, where ϵ_t is as defined in Figure 2. Let $\gamma_t = 1/2 - \epsilon_t$. Then the following upper bound holds on the error of the final hypothesis h_{fin}:*

$$\frac{1}{m}|\{i : h_{fin}(x_i) \neq y_i\}| \leq (k-1) \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq (k-1) \exp\left(-2 \sum_{t=1}^{T} \gamma_t^2\right)$$

where k is the number of classes.

3 Boosting and bagging

In this section, we describe our experiments comparing boosting and bagging on the UCI benchmarks.

We first mention briefly a small implementation issue: Many learning algorithms can be modified to handle examples that are weighted by a distribution such as the one created by the boosting algorithm. When this is possible, the booster's distribution D_t is supplied directly to the weak learning algorithm, a method we call boosting by *reweighting*. However, some learning algorithms require an unweighted set of examples. For such a weak learning algorithm, we instead choose a set of examples from S independently at random according to the distribution D_t with replacement. The number of examples to be chosen on each round is a matter of discretion; in our experiments, we chose m examples on each round, where m is the size of the original training set S. We refer to this method as boosting by *resampling*.

Boosting by resampling is also possible when using the pseudo-loss. In this case, a set of mislabels are chosen from the set B of all mislabels with replacement according to the given distribution D_t. Such a procedure is consistent with the interpretation of mislabels discussed in Section 2.2. In our experiments, we chose a sample of size $|B| = m(k-1)$ on each round when using the resampling method.

3.1 The weak learning algorithms

As mentioned in the introduction, we used three weak learning algorithms in these experiments. In all cases, the examples are described by a vector of values which corresponds to a fixed set of

features or attributes. These values may be discrete or continuous. Some of the examples may have missing values. All three of the weak learners build hypotheses which classify examples by repeatedly testing the values of chosen attributes.

The first and simplest weak learner, which we call **FindAttrTest**, searches for the single attribute-value test with minimum error (or pseudo-loss) on the training set. More precisely, **FindAttrTest** computes a classifier which is defined by an attribute a, a value v and three predictions p_0, p_1 and $p_?$. This classifier classifies a new example x as follows: if the value of attribute a is missing on x, then predict $p_?$; if attribute a is discrete and its value on example x is equal to v, or if attribute a is continuous and its value on x is at most v, then predict p_0; otherwise predict p_1. If using ordinary error (**AdaBoost.M1**), these "predictions" $p_0, p_1, p_?$ would be simple classifications; for pseudo-loss, the "predictions" would be vectors in $[0, 1]^k$ (where k is the number of classes).

The algorithm **FindAttrTest** searches exhaustively for the classifier of the form given above with minimum error or pseudo-loss with respect to the distribution provied by the booster. In other words, all possible values of a, v, p_0, p_1 and $p_?$ are considered. This search can be carried out in time linear in the size of the training set (per round of boosting).

For this algorithm, we used boosting with reweighting.

The second weak learner does a somewhat more sophisticated search for a decision rule that tests on a conjunction of attribute-value tests. We sketch the main ideas of this algorithm, which we call **FindDecRule**, but omit some of the finer details for lack of space. These details will be provided in the full paper.

First, the algorithm requires an unweighted training set, so we use the resampling version of boosting. The given training set is randomly divided into a growing set using 70% of the data, and a pruning set with the remaining 30%. In the first phase, the growing set is used to grow a list of attribute-value tests. Each test compares an attribute a to a value v, similar to the tests used by **FindAttrTest**. We use an entropy-based potential function to guide the growth of the list of tests. The list is initially empty, and one test is added at a time, each time choosing the test that will cause the greatest drop in potential. After the test is chosen, only one branch is expanded, namely, the branch with the highest remaining potential. The list continues to be grown in this fashion until no test remains which will further reduce the potential.

In the second phase, the list is pruned by selecting the prefix of the list with minimum error (or pseudo-loss) on the pruning set.

The third weak learner is Quinlan's **C4.5** decision-tree algorithm [17]. We used all the default options with pruning turned on. Since **C4.5** expects an unweighted training sample, we used resampling. Also, we did not attempt to use **AdaBoost.M2** since **C4.5** is designed to minimize error, not pseudo-loss.

3.2 Bagging

We compared boosting to Breiman's [1] "bootstrap aggregating" or "bagging" method for training and combining multiple copies of a learning algorithm. Briefly, the method works by training each copy of the algorithm on a bootstrap sample, i.e., a sample of size m chosen uniformly at random with replacement from the original training set S (of size m). The multiple hypotheses that are computed are then combined using simple voting; that is, the final composite hypothesis classifies an example x to the class most often assigned by the underlying "weak" hypotheses. See

his paper for more details. The method can be quite effective, especially, according to Breiman, for "unstable" learning algorithms for which a small change in the data effects a large change in the computed hypothesis.

In order to compare **AdaBoost.M2**, which uses pseudo-loss, to bagging, we also extended bagging in a natural way for use with a weak learning algorithm that minimizes pseudo-loss rather than ordinary error. As described in Section 2.2, such a weak learning algorithm expects to be provided with a distribution over the set B of all mislabels. On each round of bagging, we construct this distribution using the bootstrap method; that is, we select $|B|$ mislabels from B (chosen uniformly at random with replacement), and assign each mislabel weight $1/|B|$ times the number of times it was chosen. The hypotheses h_t computed in this manner are then combined using voting in a natural manner; namely, given x, the combined hypothesis outputs the label y which maximizes $\sum_t h_t(x, y)$.

For either error or pseudo-loss, the differences between bagging and boosting can be summarized as follows: (1) bagging always uses resampling rather than reweighting; (2) bagging does not modify the distribution over examples or mislabels, but instead always uses the uniform distribution; and (3) in forming the final hypothesis, bagging gives equal weight to each of the weak hypotheses.

3.3 The experiments

We conducted our experiments on a collection of machine learning datasets available from the repository at University of California at Irvine.[2] A summary of some of the properties of these datasets is given in Table 3 in the appendix. Some datasets are provided with a test set. For these, we reran each algorithm 20 times (since some of the algorithms are randomized), and averaged the results. For datasets with no provided test set, we used 10-fold cross validation, and averaged the results over 10 runs (for a total of 100 runs of each algorithm on each dataset).

In all our experiments, we set the number of rounds of boosting or bagging to be $T = 100$.

3.4 Results and discussion

The results of our experiments are shown in Table 1. The figures indicate test error rate averaged over multiple runs of each algorithm. Columns indicate which weak learning algorithm was used, and whether pseudo-loss (**AdaBoost.M2**) or error (**AdaBoost.M1**) was used. Columns labeled "–" indicate that the weak learning algorithm was used by itself (with no boosting or bagging). Columns using boosting or bagging are marked "boost" and "bag," respectively.

One of our goals in carrying out these experiments was to determine if boosting using pseudo-loss (rather than error) is worthwhile. These experiments indicate that pseudo-loss is definitely worth the effort. Using pseudo-loss did dramatically better than error on every non-binary problem (except it did slightly worse on "iris" with three classes). Because **AdaBoost.M2** did so much better than **AdaBoost.M1**, we will only discuss **AdaBoost.M2** in the remaining discussion.

Using pseudo-loss with bagging gave mixed results in comparison to ordinary error. Overall, pseudo-loss gave better results, but occassionally, using pseudo-loss hurt considerably.

For the simpler weak learning algorithms (**FindAttrTest** and **FindDecRule**), boosting did significantly and uniformly better than bagging. The boosting error rate was worse than the bagging

[2] URL "http://www.ics.uci.edu/~mlearn/MLRepository.html"

	FindAttrTest					**FindDecRule**					**C4.5**		
		error		pseudo-loss			error		pseudo-loss			error	
name	–	boost	bag	boost	bag	–	boost	bag	boost	bag	–	boost	bag
soybean-small	57.6	56.4	48.7	0.2	20.5	51.8	56.0	45.7	0.4	2.9	2.2	3.4	2.2
labor	25.1	8.8	19.1	9.0	18.9	24.0	7.3	14.6	7.2	15.7	15.8	13.1	11.3
promoters	29.7	8.9	16.6	9.1	17.2	25.9	8.3	13.7	8.5	14.2	22.0	5.0	12.7
iris	35.2	4.7	28.4	4.8	7.1	38.3	4.3	18.8	4.8	5.5	5.9	5.0	5.0
hepatitis	19.7	18.6	16.8	18.3	17.4	21.6	18.0	20.1	18.4	20.6	21.2	16.3	17.5
sonar	25.9	16.5	25.9	16.8	25.9	31.4	16.2	26.1	15.4	25.7	28.9	19.0	24.3
glass	51.5	51.1	50.9	29.4	54.2	49.7	48.5	47.2	25.0	52.0	31.7	22.7	25.7
audiology.stand	57.7	57.7	57.7	26.9	77.2	57.7	57.7	57.7	18.5	63.5	15.4	15.4	10.2
cleve	27.8	18.8	22.4	18.8	21.9	27.4	19.7	20.3	20.6	19.9	26.6	21.7	20.9
soybean-large	64.8	64.5	59.0	9.8	74.2	73.6	73.6	73.6	7.2	66.0	13.3	6.8	12.2
ionosphere	17.8	8.5	17.3	8.5	17.2	10.3	6.6	9.3	6.4	9.4	8.9	5.8	6.2
house-votes-84	4.4	3.7	4.4	3.7	4.4	5.0	4.4	4.4	4.3	4.5	3.5	5.1	3.6
votes1	12.7	8.9	12.7	8.9	12.7	13.2	9.4	11.2	9.4	10.7	10.3	10.4	9.2
crx	14.5	14.4	14.5	14.4	14.5	14.5	13.5	14.5	13.6	14.5	15.8	13.8	13.6
breast-cancer-w	8.4	4.4	6.7	4.4	6.6	8.1	4.1	5.3	4.0	5.2	5.0	3.3	3.2
pima-indians-di	26.1	24.4	26.1	24.5	26.0	27.8	25.3	26.4	25.4	26.6	28.4	25.7	24.4
vehicle	64.3	64.4	57.6	26.1	56.1	61.3	61.2	61.0	25.0	54.3	29.9	22.6	26.1
vowel	81.8	81.8	76.8	18.2	74.7	82.0	72.7	71.6	6.5	63.2	2.2	0.0	0.0
german	30.0	24.9	30.4	24.9	30.3	30.0	25.4	29.6	25.6	29.7	29.4	25.0	24.6
segmentation	75.8	75.8	54.5	4.2	72.5	73.7	53.3	54.3	2.4	58.0	3.6	1.4	2.7
hypothyroid	2.2	1.0	2.2	1.0	2.2	0.8	1.0	0.7	1.0	0.7	0.8	1.0	0.8
sick-euthyroid	5.6	3.0	5.6	3.0	5.6	2.4	2.4	2.2	2.4	2.1	2.2	2.1	2.1
splice	37.0	9.2	35.6	4.4	33.4	29.5	8.0	29.5	4.0	29.5	5.8	4.9	5.2
kr-vs-kp	32.8	4.4	30.7	4.4	31.3	24.6	0.7	20.8	0.6	21.7	0.5	0.3	0.6
satimage	58.3	58.3	58.3	14.9	41.6	57.6	56.5	56.7	13.1	30.0	14.8	8.9	10.6
agaricus-lepiot	11.3	0.0	11.3	0.0	11.3	8.2	0.0	8.2	0.0	8.2	0.0	0.0	0.0
letter-recognit	92.9	92.9	91.9	34.1	93.7	92.3	91.8	91.8	30.4	93.7	13.8	3.3	6.8

Table 1: Test error rates of various algorithms on benchmark problems.

error rate (using either pseudo-loss or error) on a very small number of benchmark pr oblems, and on these, the difference in performance was quite small. On average, for **FindAttrTest**, boosting improved the error rate over using **FindAttrTest** alone by 55.1%, compared to bagging which gave an improvement of only 10.6% using pseudo-loss or 8.4% using error. For **FindDecRule**, boosting improved the error rate by 53.1%, bagging by only 19.0% using pseudo-loss, 13.1% using error.

When using **C4.5** as the weak learning algorithm, boosting and bagging seem more evenly matched, although boosting still seems to have a slight advantage. On average, boosting improved the error rate by 23.6%, bagging by 20.7%. Boosting beat bagging by more than 2% on 6 of the benchmarks; while bagging beat boosting by this amount on only 1 benchmark. For the remaining 20 benchmarks, the difference in performance was less than 2%.

Using boosting with **FindAttrTest** does quite well as a learning algorithm in its own right, in comparison, say, to **C4.5**. This algorithm beat **C4.5** on 10 of the benchmarks (by at least 2%), tied on 13, and lost on 4. As mentioned above, its average performance relative to using **FindAttrTest** by itself was 55.1%. In comparison, **C4.5**'s improvement in performance over **FindAttrTest** was 49.9%.

Using boosting with **FindDecRule** did somewhat better. The win-tie-lose numbers for this

algorithm (compared to **C4.5**) were 12-12-3, and its average improvement over **FindAttrTest** was 58.2%.

4 Boosting a nearest-neighbor classifier

In this section we study the performance of a learning algorithm which combines **AdaBoost** and a variant of the nearest-neighbor classifier. We test the combined algorithm on the problem of reconizing handwritten digits. Our goal is not to improve on the accuracy of the nearest neighbor classifier, but rather, to speed it up. Speedup is achieved by reducing the number of prototypes in the hypothesis and the number of required distance calculations without increasing the error rate. It is a similar approach to that of nearest-neighbor editing [11, 12] in which one tries to find the minimal set of prototypes that is sufficient to label all the training set correctly.

The dataset comes from the US Postal Service (USPS) and consists of 9709 training examples and 2007 test examples. The training and test examples are evidently drawn from rather different distributions as there is a very significant improvement in the performance if the partition of the data into training and testing is done at random (rather than using the given partition). We report results both on the original partitioning and on a training set and a test set of the same sizes that were generated by randomly partitioning the union of the original training and test sets.

Each image is represented by a 16×16-matrix of 8-bit pixels. The metric that we use is the standard Euclidean distance between the images (viewed as vectors in $\mathbb{R}^{256}$). This is a very naive metric, but it gives reasonably good performance. A nearest-neighbor classifier which uses all the training examples as prototypes achieves a test error of 5.7% (2.3% on randomly partitioned data). Using tangent distance [20] is in our future plans.

Our weak learning algorithm is simply to use a random set of examples as prototypes, chosen according to the distribution provided by the boosting algorithm. A standard nearest-neighbor classifier would predict the label of a test example according to the identity of its closest prototype. However, we found that significant improvement was achieved by using a variant of this algorithm that is optimized for use with **AdaBoost.M2**. To do this we consider all of the training set examples that are closest to each selected prototype, and choose the fixed prediction that minimizes the pseudo-loss for this (weighted) set of examples. This prediction choice is described in detail in Example 5 in [9]. In addition, we make the following modifications to the basic scheme:

- When using **AdaBoost.M2**, it is possible that hypothesis h_t has pseudo-loss less than $1/2$ on distribution D_{t+1}. When this happens, we reuse in hypothesis h_{t+1} the same set of prototypes that were used for h_t. Without increasing the size of the final hypothesis, this method reduces the theoretical error bound, and, as we observe in experiments, also the actual error on the training set.

- Instead of just selecting the prototypes at random, the algorithm repeatedly selects ten random prototypes and adds the one which causes the largest reduction in the pseudo-loss.

We ran 30 iterations of the boosting algorithm, and the number of prototypes we used were 10 for the first weak hypothesis, 20 for the second, 40 for the third, 80 for the next five, and 100 for the remaining weak hypotheses. These sizes were chosen so that the errors of all of the weak hypotheses are approximately equal.

		random partition					given partition				
	# Proto-	**AdaBoost**			**Strawman**		**AdaBoost**			**Strawman**	
round	types	theory	train	test	train	test	theory	train	test	train	test
1	10	524.6	42.8	43.5	44.8	43.5	536.3	49.1	47.6	39.2	41.9
5	230	86.4	5.8	8.7	5.5	7.8	83.0	5.4	13.7	4.3	9.6
10	670	16.0	0.2	5.5	2.5	5.3	10.9	0.1	9.1	1.4	7.7
13	970	4.5	0.0	4.6	1.7	4.7	3.3	0.0	7.8	0.9	7.3
15	1170	2.4	0.0	3.9	1.4	4.7	1.5	0.0	7.8	0.7	7.4
20	1670	0.4	0.0	3.4	1.4	4.9	0.2	0.0	7.2	0.5	6.7
25	2170	0.1	0.0	3.3	1.2	4.2	0.0	0.0	6.8	0.3	7.1
30	2670	0.0	0.0	3.1	1.1	3.9	0.0	0.0	6.6	0.2	6.5

Table 2: Errors on randomly selected training and test sets, in percent. For columns labeled "random partition," a random partition of the union of the training and test sets was used; "given partition" means the provided partition into training and test sets was used. Columns labeled "theory" give theoretical upper bound on training error (see Theorem 2).

We compared the performance of our algorithm to a strawman algorithm which uses a single set of prototypes. Similar to our algorithm, the prototype set is generated incrementally, comparing ten prototype candidates at each step. We compared the performance of the boosting algorithm to that of the strawman hypothesis that uses the same number of prototypes. We also compared our performance to that of the condensed nearest neighbor rule [12] (CNN), a greedy method for finding a small set of prototypes which correctly classify the entire training set.

4.1 Results and discussion

The results of our experiments are summarized in Table 2 and Figure 4.

In the left part of Table 2 we have the results of an experiment in which the test set was selected randomly from the union of the test and training data of the USPS dataset. We see that the boosting algorithm outperforms the strawman algorithm; the difference is dramatic on the training set but is also significant on the test set. Comparing these results to those of CNN, we find that the boosting algorithm reached zero error on the training set after 970 prototypes, which is about the same as CNN which reduced the set of prototypes to 964. However, the *test* error of this CNN was 5.7%, while the test error achieved by the 970 prototypes found by boosting was 4.5% and was further reduced to 3.1% when 2670 prototypes were used. Better editing algorithms might reduce the set of prototypes further, but it seems unlikely that this will reduce the test error. The error achieved by using the full training set (9709 prototypes) was 2.3%.

These results are also described by the graphs in Figure 4. The uppermost jagged line is a concatanation of the errors of the weak hypotheses with respect to the corresponding weights on the training set. Each peak followed by a valley corresponds to the beginning and end errors of a weak hypothesis. As we see the weighted error always started around 50% on the beginning of a boosting iteration and reached $20\% - 30\%$ by its end. The heaviest line describes the upper bound on the training error that is guaranteed by Theorem 2, and the two bottom lines describe the training and test error of the final combined hypothesis.

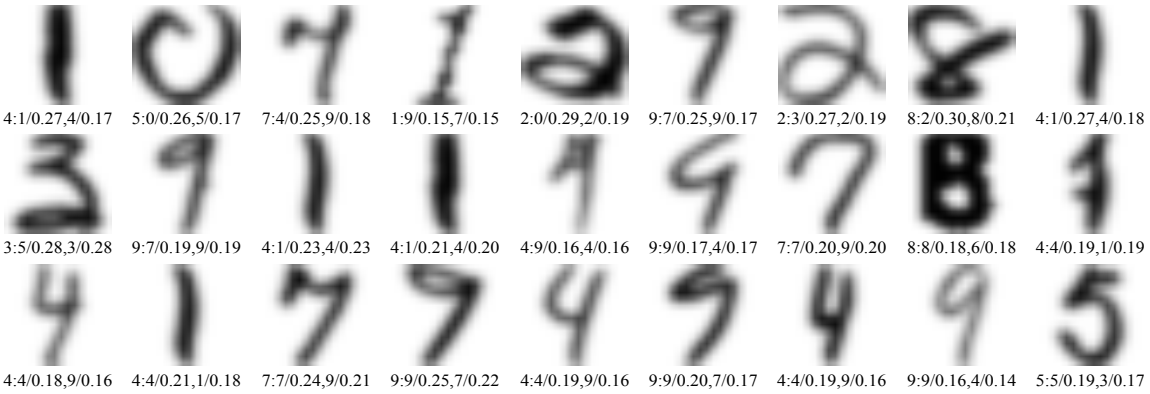


Figure 3: A sample of the examples that have the largest weight after 3 out of the 30 boosting iterations. The first line is after iteration 4, the second after iteration 12 and the third after iteration 25. Underneath each image we have a line of the form d:ℓ_1/w_1,ℓ_2/w_2, where d is the label of the example, ℓ_1 and ℓ_2 are the labels that get the highest and second highest vote from the combined hypothesis at that point in the run of the algorithm, and w_1, w_2 are the corresponding votes.

It is interesting that the performance of the boosting algorithm on the test set improved significantly after the error on the training set has already become zero. This is surprising because an "Occam's razor" argument would predict that increasing the complexity of the hypothesis after the error has been reduced to zero is likely to degrade the performance on the test set.

The right hand side of Table 2 summarizes the results of running our algorithm on the training and test set as they were defined. Here the performance of boosting was similar to that of the strawman algorithm. However, it was still significantly better than that of CNN, which achieved a test error of 7.8% using 835 prototypes. It seems that the difference between the distribution of the test set and the training set removed the advantage that boosting had over the strawman algorithm.

We observed that when calculating the prediction of the combined hypothesis as we add the weighted vote of one weak hypothesis at a time, we can sometimes conclude what the final vote will be before calculating all of the hypotheses. This is possible when the difference between the current largest vote and the current second largest vote is larger than the total weight of the remaining hypotheses. In our experiments we found that the average number of weak hypotheses that had to be considered is 24.0 for the randomly chosen training set and 23.6 for the original training set. We can thus, on average, reduce the number of distance claculations that are required for evaluating the hypothesis from 2670 to 2070 without changing the predictions.

It is instructive to observe the examples that are given large weights by the boosting algorithm. A sample of these is given in Figure 3. There seem to be two types of "hard" examples. First are examples which are very atypical or wrongly labeled (such as example 2 on the first line and examples 3, 4 and 9 on the second line). The second type, which tends to dominate on later iterations, consists of examples that are very similar to each other but have different labels (such as examples 3 versus 4 and 1 versus 8 on the third line). Although the algorithm at this point was correct on all training examples, it is clear from the votes it assigned to different labels for these example pairs that it was still trying to improve the discrimination between these hard to discriminate pairs. This agrees with our intuition that the pseudo-loss is a mechanism that causes

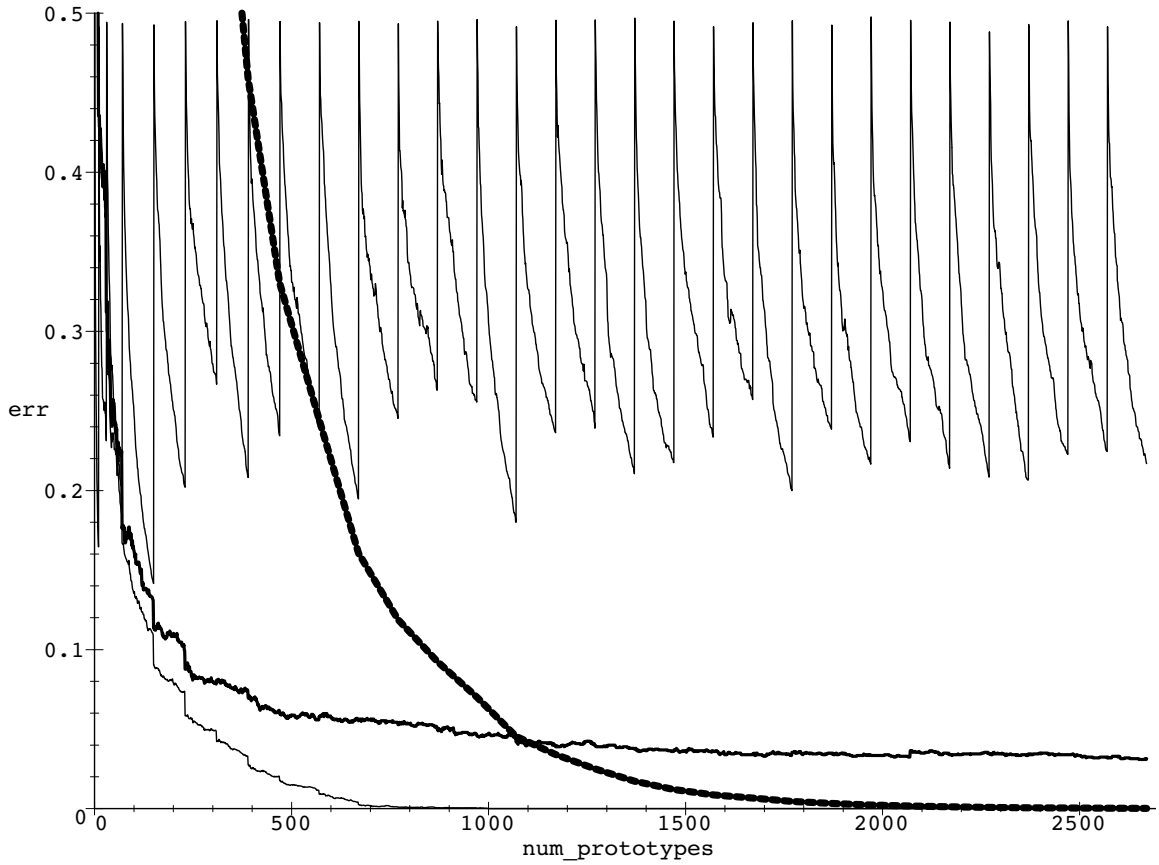


Figure 4: Graphs of the performance of the boosting algorithm on the randomly partitioned USPS dataset. The horizontal axis indicates the total number of prototypes that were added to the combined hypothesis, and the vertical axis indicates error. The topmost jagged line indicates the error of the weak hypothesis that is trained at this point on the weighted training set. The bold curve is the bound on the training error that is calculated based on the performance of the weak learner. The lowest thin curve is the performance of the combined hypothesis on the training set. The medium-bold curve is the performance of the combined hypothesis on the test set.

the boosting algorithm to concentrate on the hard to discriminate labels of hard examples.

Acknowledgements

Thanks to Jason Catlett and William Cohen for extensive advice on the design of our experiments. Thanks to Ross Quinlan for first suggesting a comparison of boosting and bagging. Thanks also to Leo Breiman, Corinna Cortes, Harris Drucker, Jeff Jackson, Michael Kearns, Ofer Matan, Partha Niyogi, Warren Smith, and David Wolpert for helpful comments, suggestions and criticisms.

References

[1] Leo Breiman. Bagging predictors. Technical Report 421, Department of Statistics, University of California at Berkeley, 1994.

[2] William Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[3] Tom Dietterich, Michael Kearns, and Yishay Mansour. Applying the weak learning framework to understand and improve C4.5. Unpublished manuscript, 1996.

[4] Harris Drucker and Corinna Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems 8*, 1996.

[5] Harris Drucker, Corinna Cortes, L. D. Jackel, Yann LeCun, and Vladimir Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301, 1994.

[6] Harris Drucker, Robert Schapire, and Patrice Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):705–719, 1993.

[7] Harris Drucker, Robert Schapire, and Patrice Simard. Improving performance in neural networks using a boosting algorithm. In *Advances in Neural Information Processing Systems 5*, pages 42–49, 1993.

[8] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[9] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Unpublished manuscript available electronically (on our web pages, or by email request). An extended abstract appeared in *Computational Learning Theory: Second European Conference, EuroCOLT '95*, pages 23–37, Springer-Verlag, 1995.

[10] Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 70–77, 1994.

[11] Geoffrey W. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, pages 431–433, 1972.

[12] Peter E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, IT-14:515–516, May 1968.

[13] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–91, 1993.

[14] Jeffrey C. Jackson and Mark W. Craven. Learning sparse perceptrons. In *Advances in Neural Information Processing Systems 8*, 1996.

[15] Michael Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, 1996.

[16] Eun Bae Kong and Thomas G. Dietterich. Error-correcting output coding corrects bias and variance. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 313–321, 1995.

[17] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[18] J. Ross Quinlan. Bagging, boosting, and C4.5. unpublished manuscript, 1996.

[19] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[20] Patrice Simard, Yann Le Cun, and John Denker. Efficient pattern recognition using a new transformation distance. In *Advances in Neural Information Processing Systems*, volume 5, pages 50–58, 1993.

A The Irvine datasets

	# examples		# classes	# attributes		missing
name	train	test		discrete	cont.	values
soybean-small	47	-	4	35	-	-
labor	57	-	2	8	8	√
promoters	106	-	2	57	-	-
iris	150	-	3	-	4	-
hepatitis	155	-	2	13	6	√
sonar	208	-	2	-	60	-
glass	214	-	7	-	9	-
audiology.stand	226	-	24	69	-	√
cleve	303	-	2	7	6	√
soybean-large	307	376	19	35	-	√
ionosphere	351	-	2	-	34	-
house-votes-84	435	-	2	16	-	√
votes1	435	-	2	15	-	√
crx	690	-	2	9	6	√
breast-cancer-w	699	-	2	-	9	√
pima-indians-di	768	-	2	-	8	-
vehicle	846	-	4	-	18	-
vowel	528	462	11	-	10	-
german	1000	-	2	13	7	-
segmentation	2310	-	7	-	19	-
hypothyroid	3163	-	2	18	7	√
sick-euthyroid	3163	-	2	18	7	√
splice	3190	-	3	60	-	-
kr-vs-kp	3196	-	2	36	-	-
satimage	4435	2000	6	-	36	-
agaricus-lepiot	8124	-	2	22	-	-
letter-recognit	16000	4000	26	-	16	-

Table 3: The benchmark machine learning problems used in the experiments.

Machine Learning, 45, 5–32, 2001
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

Random Forests

LEO BREIMAN
Statistics Department, University of California, Berkeley, CA 94720

Editor: Robert E. Schapire

Abstract. Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges a.s. to a limit as the number of trees in the forest becomes large. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. Using a random selection of features to split each node yields error rates that compare favorably to Adaboost (Y. Freund & R. Schapire, *Machine Learning*: *Proceedings of the Thirteenth International conference*, ∗ ∗ ∗, 148–156), but are more robust with respect to noise. Internal estimates monitor error, strength, and correlation and these are used to show the response to increasing the number of features used in the splitting. Internal estimates are also used to measure variable importance. These ideas are also applicable to regression.

Keywords: classification, regression, ensemble

1. Random forests

1.1. Introduction

Significant improvements in classification accuracy have resulted from growing an ensemble of trees and letting them vote for the most popular class. In order to grow these ensembles, often random vectors are generated that govern the growth of each tree in the ensemble. An early example is bagging (Breiman, 1996), where to grow each tree a random selection (without replacement) is made from the examples in the training set.

Another example is random split selection (Dietterich, 1998) where at each node the split is selected at random from among the K best splits. Breiman (1999) generates new training sets by randomizing the outputs in the original training set. Another approach is to select the training set from a random set of weights on the examples in the training set. Ho (1998) has written a number of papers on "the random subspace" method which does a random selection of a subset of features to use to grow each tree.

In an important paper on written character recognition, Amit and Geman (1997) define a large number of geometric features and search over a random selection of these for the best split at each node. This latter paper has been influential in my thinking.

The common element in all of these procedures is that for the kth tree, a random vector Θ_k is generated, independent of the past random vectors $\Theta_{1,\ldots,}\Theta_{k-1}$ but with the same distribution; and a tree is grown using the training set and Θ_k, resulting in a classifier $h(\mathbf{x}, \Theta_k)$ where $\mathbf{x}$ is an input vector. For instance, in bagging the random vector Θ is

generated as the counts in N boxes resulting from N darts thrown at random at the boxes, where N is number of examples in the training set. In random split selection Θ consists of a number of independent random integers between 1 and K. The nature and dimensionality of Θ depends on its use in tree construction.

After a large number of trees is generated, they vote for the most popular class. We call these procedures **random forests**.

Definition 1.1. A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathbf{x}, \Theta_k),\ k = 1, \ldots\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input $\mathbf{x}$.

1.2. *Outline of paper*

Section 2 gives some theoretical background for random forests. Use of the Strong Law of Large Numbers shows that they always converge so that overfitting is not a problem. We give a simplified and extended version of the Amit and Geman (1997) analysis to show that the accuracy of a random forest depends on the strength of the individual tree classifiers and a measure of the dependence between them (see Section 2 for definitions).

Section 3 introduces forests using the random selection of features at each node to determine the split. An important question is how many features to select at each node. For guidance, internal estimates of the generalization error, classifier strength and dependence are computed. These are called out-of-bag estimates and are reviewed in Section 4. Section 5 and 6 give empirical results for two different forms of random features. The first uses random selection from the original inputs; the second uses random linear combinations of inputs. The results compare favorably to Adaboost.

The results turn out to be insensitive to the number of features selected to split each node. Usually, selecting one or two features gives near optimum results. To explore this and relate it to strength and correlation, an empirical study is carried out in Section 7.

Adaboost has no random elements and grows an ensemble of trees by successive reweightings of the training set where the current weights depend on the past history of the ensemble formation. But just as a deterministic random number generator can give a good imitation of randomness, my belief is that in its later stages Adaboost is emulating a random forest. Evidence for this conjecture is given in Section 8.

Important recent problems, i.e., medical diagnosis and document retrieval, often have the property that there are many input variables, often in the hundreds or thousands, with each one containing only a small amount of information. A single tree classifier will then have accuracy only slightly better than a random choice of class. But combining trees grown using random features can produce improved accuracy. In Section 9 we experiment on a simulated data set with 1,000 input variables, 1,000 examples in the training set and a 4,000 example test set. Accuracy comparable to the Bayes rate is achieved.

In many applications, understanding of the mechanism of the random forest "black box" is needed. Section 10 makes a start on this by computing internal estimates of variable importance and binding these together by reuse runs.

Section 11 looks at random forests for regression. A bound for the mean squared generalization error is derived that shows that the decrease in error from the individual trees in the forest depends on the correlation between residuals and the mean squared error of the individual trees. Empirical results for regression are in Section 12. Concluding remarks are given in Section 13.

2. Characterizing the accuracy of random forests

2.1. Random forests converge

Given an ensemble of classifiers $h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_K(\mathbf{x})$, and with the training set drawn at random from the distribution of the random vector Y, $\mathbf{X}$, define the margin function as

$$mg(\mathbf{X}, Y) = av_k I(h_k(\mathbf{X}) = Y) - \max_{j \neq Y} av_k I(h_k(\mathbf{X}) = j).$$

where $I(\cdot)$ is the indicator function. The margin measures the extent to which the average number of votes at $\mathbf{X}$, Y for the right class exceeds the average vote for any other class. The larger the margin, the more confidence in the classification. The generalization error is given by

$$PE^* = P_{\mathbf{X},Y}(mg(\mathbf{X}, Y) < 0)$$

where the subscripts $\mathbf{X}$, Y indicate that the probability is over the $\mathbf{X}$, Y space.

In random forests, $h_k(\mathbf{X}) = h(\mathbf{X}, \Theta_k)$. For a large number of trees, it follows from the Strong Law of Large Numbers and the tree structure that:

Theorem 1.2. *As the number of trees increases, for almost surely all sequences* $\Theta_{1,\ldots}$ PE^* *converges to*

$$P_{\mathbf{X},Y}(P_\Theta(h(\mathbf{X}, \Theta) = Y) - \max_{j \neq Y} P_\Theta(h(\mathbf{X}, \Theta) = j) < 0). \quad (1)$$

Proof: see Appendix I. □

This result explains why random forests do not overfit as more trees are added, but produce a limiting value of the generalization error.

2.2. Strength and correlation

For random forests, an upper bound can be derived for the generalization error in terms of two parameters that are measures of how accurate the individual classifiers are and of the dependence between them. The interplay between these two gives the foundation for understanding the workings of random forests. We build on the analysis in Amit and Geman (1997).

Definition 2.1. The margin function for a random forest is

$$mr(\mathbf{X}, Y) = P_\Theta(h(\mathbf{X}, \Theta) = Y) - \max_{j \neq Y} P_\Theta(h(\mathbf{X}, \Theta) = j) \tag{2}$$

and the strength of the set of classifiers $\{h(\mathbf{x}, \Theta)\}$ is

$$s = E_{\mathbf{X},Y} mr(\mathbf{X}, Y). \tag{3}$$

Assuming $s \geq 0$, Chebychev's inequality gives

$$PE^* \leq \text{var}(mr)/s^2 \tag{4}$$

A more revealing expression for the variance of mr is derived in the following: Let

$$\hat{j}(\mathbf{X}, Y) = \arg\max_{j \neq Y} P_\Theta(h(\mathbf{X}, \Theta) = j)$$

so

$$\begin{aligned} mr(\mathbf{X}, Y) &= P_\Theta(h(\mathbf{X}, \Theta) = Y) - P_\Theta(h(\mathbf{X}, \Theta) = \hat{j}(\mathbf{X}, Y)) \\ &= E_\Theta[I(h(\mathbf{X}, \Theta) = Y) - I(h(\mathbf{X}, \Theta) = \hat{j}(\mathbf{X}, Y)]. \end{aligned}$$

Definition 2.2. The raw margin function is

$$rmg(\Theta, \mathbf{X}, Y) = I(h(\mathbf{X}, \Theta) = Y) - I(h(\mathbf{X}, \Theta) = \hat{j}(\mathbf{X}, Y)).$$

Thus, $mr(\mathbf{X}, Y)$ is the expectation of $rmg(\Theta, \mathbf{X}, Y)$ with respect to Θ. For any function f the identity

$$[E_\Theta f(\Theta)]^2 = E_{\Theta,\Theta'} f(\Theta) f(\Theta')$$

holds where Θ, Θ' are independent with the same distribution, implying that

$$mr(\mathbf{X}, Y)^2 = E_{\Theta,\Theta'} rmg(\Theta, \mathbf{X}, Y) rmg(\Theta', \mathbf{X}, Y) \tag{5}$$

Using (5) gives

$$\begin{aligned} \text{var}(mr) &= E_{\Theta,\Theta'}(\text{cov}_{\mathbf{X},Y} rmg(\Theta, \mathbf{X}, Y) rmg(\Theta', \mathbf{X}, Y)) \\ &= E_{\Theta,\Theta'}(\rho(\Theta, \Theta') sd(\Theta) sd(\Theta')) \end{aligned} \tag{6}$$

where $\rho(\Theta, \Theta')$ is the correlation between $rmg(\Theta, \mathbf{X}, Y)$ and $rmg(\Theta', \mathbf{X}, Y)$ holding Θ, Θ' fixed and $sd(\Theta)$ is the standard deviation of $rmg(\Theta, \mathbf{X}, Y)$ holding Θ fixed. Then,

$$\begin{aligned} \text{var}(mr) &= \bar{\rho}(E_\Theta sd(\Theta))^2 \\ &\leq \bar{\rho} E_\Theta \text{var}(\Theta) \end{aligned} \tag{7}$$

where $\bar{\rho}$ is the mean value of the correlation; that is,

$$\bar{\rho} = E_{\Theta,\Theta'}(\rho(\Theta, \Theta')sd(\Theta)sd(\Theta'))/E_{\Theta,\Theta'}(sd(\Theta)sd(\Theta'))$$

Write

$$\begin{aligned} E_{\Theta}\text{var}(\Theta) &\leq E_{\Theta}(E_{\mathbf{X},Y}rmg(\Theta, \mathbf{X}, Y))^2 - s^2 \\ &\leq 1 - s^2. \end{aligned} \tag{8}$$

Putting (4), (7), and (8) together yields:

Theorem 2.3. *An upper bound for the generalization error is given by*

$$PE^* \leq \bar{\rho}(1 - s^2)/s^2.$$

Although the bound is likely to be loose, it fulfills the same suggestive function for random forests as VC-type bounds do for other types of classifiers. It shows that the two ingredients involved in the generalization error for random forests are the strength of the individual classifiers in the forest, and the correlation between them in terms of the raw margin functions. The c/s2 ratio is the correlation divided by the square of the strength. In understanding the functioning of random forests, this ratio will be a helpful guide—the smaller it is, the better.

Definition 2.4. The c/s2 ratio for a random forest is defined as

$$c/s2 = \bar{\rho}/s^2.$$

There are simplifications in the two class situation. The margin function is

$$mr(\mathbf{X}, Y) = 2P_{\Theta}(h(\mathbf{X}, \Theta) = Y) - 1$$

The requirement that the strength is positive (see (4)) becomes similar to the familiar weak learning condition $E_{\mathbf{X},Y}P_{\Theta}(h(\mathbf{X}, \Theta) = Y) > .5$. The raw margin function is $2I(h(\mathbf{X}, \Theta) = Y) - 1$ and the correlation $\bar{\rho}$ is between $I(h(\mathbf{X}, \Theta) = Y)$ and $I(h(\mathbf{X}, \Theta') = Y)$. In particular, if the values for Y are taken to be $+1$ and -1, then

$$\bar{\rho} = E_{\Theta,\Theta'}[\rho(h(\cdot, \Theta), h(\cdot, \Theta')]$$

so that $\bar{\rho}$ is the correlation between two different members of the forest averaged over the Θ, Θ' distribution.

For more than two classes, the measure of strength defined in (3) depends on the forest as well as the individual trees since it is the forest that determines $\hat{j}(\mathbf{X}, Y)$. Another approach

is possible. Write

$$PE^* = P_{\mathbf{X},Y}(P_\Theta(h(\mathbf{X},\Theta) = Y) - \max_{j \neq Y} P_\Theta(h(\mathbf{X},\Theta) = j) < 0)$$
$$\leq \sum_j P_{\mathbf{X},Y}(P_\Theta(h(\mathbf{X},\Theta) = Y) - P_\Theta(h(\mathbf{X},\Theta) = j) < 0).$$

Define

$$s_j = E_{\mathbf{X},Y}(P_\Theta(h(\mathbf{X},\Theta) = Y) - P_\Theta(h(\mathbf{X},\Theta) = j))$$

to be the strength of the set of classifiers $\{h(\mathbf{x},\Theta)\}$ relative to class j. Note that this definition of strength does not depend on the forest. Using Chebyshev's inequality, assuming all $s_{j>0}$ leads to

$$PE^* \leq \sum_j \text{var}(P_\Theta(h(\mathbf{X},\Theta) = Y) - P_\Theta(h(\mathbf{X},\Theta) = j))s_j^2 \tag{9}$$

and using identities similar to those used in deriving (7), the variances in (9) can be expressed in terms of average correlations. I did not use estimates of the quantities in (9) in our empirical study but think they would be interesting in a multiple class problem.

3. Using random features

Some random forests reported in the literature have consistently lower generalization error than others. For instance, random split selection (Dieterrich, 1998) does better than bagging. Breiman's introduction of random noise into the outputs (Breiman, 1998c) also does better. But none of these these three forests do as well as Adaboost (Freund & Schapire, 1996) or other algorithms that work by adaptive reweighting (arcing) of the training set (see Breiman, 1998b; Dieterrich, 1998; Bauer & Kohavi, 1999).

To improve accuracy, the randomness injected has to minimize the correlation $\bar{\rho}$ while maintaining strength. The forests studied here consist of using randomly selected inputs or combinations of inputs at each node to grow each tree. The resulting forests give accuracy that compare favorably with Adaboost. This class of procedures has desirable characteristics:

i Its accuracy is as good as Adaboost and sometimes better.
ii It's relatively robust to outliers and noise.
iii It's faster than bagging or boosting.
iv It gives useful internal estimates of error, strength, correlation and variable importance.
v It's simple and easily parallelized.

Amit and Geman (1997) grew shallow trees for handwritten character recognition using random selection from a large number of geometrically defined features to define the split at each node. Although my implementation is different and not problem specific, it was their work that provided the start for my ideas.

3.1. Using out-of-bag estimates to monitor error, strength, and correlation

In my experiments with random forests, bagging is used in tandem with random feature selection. Each new training set is drawn, with replacement, from the original training set. Then a tree is grown on the new training set using random feature selection. The trees grown are not pruned.

There are two reasons for using bagging. The first is that the use of bagging seems to enhance accuracy when random features are used. The second is that bagging can be used to give ongoing estimates of the generalization error (PE*) of the combined ensemble of trees, as well as estimates for the strength and correlation. These estimates are done out-of-bag, which is explained as follows.

Assume a method for constructing a classifier from any training set. Given a specific training set T, form boostrap training sets T_k, construct classifiers $h(\mathbf{x},\ T_k)$ and let these vote to form the bagged predictor. For each $y, \mathbf{x}$ in the training set, aggregate the votes only over those classifiers for which T_k does not containing $y, \mathbf{x}$. Call this the out-of-bag classifier. Then the out-of-bag estimate for the generalization error is the error rate of the out-of-bag classifier on the training set.

Tibshirani (1996) and Wolpert and Macready (1996), proposed using out-of-bag estimates as an ingredient in estimates of generalization error. Wolpert and Macready worked on regression type problems and proposed a number of methods for estimating the generalization error of bagged predictors. Tibshirani used out-of-bag estimates of variance to estimate generalization error for arbitrary classifiers. The study of error estimates for bagged classifiers in Breiman (1996b), gives empirical evidence to show that the out-of-bag estimate is as accurate as using a test set of the same size as the training set. Therefore, using the out-of-bag error estimate removes the need for a set aside test set.

In each bootstrap training set, about one-third of the instances are left out. Therefore, the out-of-bag estimates are based on combining only about one-third as many classifiers as in the ongoing main combination. Since the error rate decreases as the number of combinations increases, the out-of-bag estimates will tend to overestimate the current error rate. To get unbiased out-of-bag estimates, it is necessary to run past the point where the test set error converges. But unlike cross-validation, where bias is present but its extent unknown, the out-of-bag estimates are unbiased.

Strength and correlation can also be estimated using out-of-bag methods. This gives internal estimates that are helpful in understanding classification accuracy and how to improve it. The details are given in Appendix II. Another application is to the measures of variable importance (see Section 10).

4. Random forests using random input selection

The simplest random forest with random features is formed by selecting at random, at each node, a small group of input variables to split on. Grow the tree using CART methodology to maximum size and do not prune. Denote this procedure by Forest-RI. The size F of the group is fixed. Two values of F were tried. The first used only one randomly selected

Table 1. Data set summary.

Data set	Train size	Test size	Inputs	Classes
Glass	214	–	9	6
Breast cancer	699	–	9	2
Diabetes	768	–	8	2
Sonar	208	–	60	2
Vowel	990	–	10	11
Ionosphere	351	–	34	2
Vehicle	846	–	18	4
Soybean	685	–	35	19
German credit	1000	–	24	2
Image	2310	–	19	7
Ecoli	336	–	7	8
Votes	435	–	16	2
Liver	345	–	6	2
Letters	15000	5000	16	26
Sat-images	4435	2000	36	6
Zip-code	7291	2007	256	10
Waveform	300	3000	21	3
Twonorm	300	3000	20	2
Threenorm	300	3000	20	2
Ringnorm	300	3000	20	2

variable, i.e., $F = 1$. The second took F to be the first integer less than $\log_2 M + 1$, where M is the number of inputs.

My experiments use 13 smaller sized data sets from the UCI repository, 3 larger sets separated into training and test sets and 4 synthetic data sets. The first 10 sets were selected because I had used them in past research. Table 1 gives a brief summary.

On each of the 13 smaller sized data sets, the following procedure was used: a random 10% of the data was set aside. On the remaining data, random forest was run twice, growing and combining 100 trees—once with $F = 1$, and the second time with $F = \text{int}(\log_2 M + 1)$. The set aside 10% was then put down each forest to get a test set error for both. The test set error selected corresponded to the lower value of the out-of-bag estimate in the two runs. This was repeated 100 times and the test set errors averaged. The same procedure was followed for the Adaboost runs which are based on combining 50 trees.

The use for 100 trees in random forests and 50 for Adaboost comes from two sources. The out-of-bag estimates are based on only about a third as many trees as are in the forest. To get reliable estimates I opted for 100 trees. The second consideration is that growing random forests is many times faster than growing the trees based on all inputs needed in Adaboost. Growing the 100 trees in random forests was considerably quicker than the 50 trees for Adaboost.

Table 2. Test set errors (%).

Data set	Adaboost	Selection	Forest-RI single input	One tree
Glass	22.0	20.6	21.2	36.9
Breast cancer	3.2	2.9	2.7	6.3
Diabetes	26.6	24.2	24.3	33.1
Sonar	15.6	15.9	18.0	31.7
Vowel	4.1	3.4	3.3	30.4
Ionosphere	6.4	7.1	7.5	12.7
Vehicle	23.2	25.8	26.4	33.1
German credit	23.5	24.4	26.2	33.3
Image	1.6	2.1	2.7	6.4
Ecoli	14.8	12.8	13.0	24.5
Votes	4.8	4.1	4.6	7.4
Liver	30.7	25.1	24.7	40.6
Letters	3.4	3.5	4.7	19.8
Sat-images	8.8	8.6	10.5	17.2
Zip-code	6.2	6.3	7.8	20.6
Waveform	17.8	17.2	17.3	34.0
Twonorm	4.9	3.9	3.9	24.7
Threenorm	18.8	17.5	17.5	38.4
Ringnorm	6.9	4.9	4.9	25.7

In the runs on the larger data sets, the random forest results for the first two data sets were based on combining 100 trees; the zip-code procedure combined 200. For Adaboost, 50 trees were combined for the first three data sets and 100 for zip-code. The synthetic data was described in Breiman (1996) and also used in Schapire et al. (1997). There were 50 runs. In each run, a new training set of size 300 and test set of size 3000 were generated. In random forests 100 trees were combined in each run—50 in Adaboost. The results of these runs are given in Table 2.

The second column are the results selected from the two group sizes by means of lowest out-of-bag error. The third column is the test set error using just one random feature to grow the trees. The fourth column contains the out-of-bag estimates of the generalization error of the individual trees in the forest computed for the best setting (single or selection). This estimate is computed by using the left-out instances as a test set in each tree grown and averaging the result over all trees in the forest.

The error rates using random input selection compare favorably with Adaboost. The comparison might be even more favorable if the search is over more values of F instead of the preset two. But the procedure is not overly sensitive to the value of F. The average absolute difference between the error rate using $F = 1$ and the higher value of F is less than 1%. The difference is most pronounced on the three large data sets.

The single variable test set results were included because in some of the data sets, using a single random input variable did better than using several. In the others, results were only slightly better than use of a single variable. It was surprising that using a single randomly chosen input variable to split on at each node could produce good accuracy.

Random input selection can be much faster than either Adaboost or Bagging. A simple analysis shows that the ratio of RI compute time to the compute time of unpruned tree construction using all variables is $F^* \log_2(N)/M$ where F is the number of variables used in Forest-RI, N is the number of instances, and M the number of input variables. For zip-code data, using $F = 1$, the ratio is .025, implying that Forest-RI is 40 times faster. An empirical check confirmed this difference. A Forest-RI run ($F = 1$) on the zip-code data takes 4.0 minutes on a 250 Mhz Macintosh to generate 100 trees compared to almost three hours for Adaboost. For data sets with many variables, the compute time difference may be significant.

5. Random forests using linear combinations of inputs

If there are only a few inputs, say M, taking F an appreciable fraction of M might lead an increase in strength but higher correlation. Another approach consists of defining more features by taking random linear combinations of a number of the input variables. That is, a feature is generated by specifying L, the number of variables to be combined. At a given node, L variables are randomly selected and added together with coefficients that are uniform random numbers on $[-1, 1]$. F linear combinations are generated, and then a search is made over these for the best split. This procedure is called Forest-RC.

We use $L = 3$ and $F = 2$, 8 with the choice for F being decided on by the out-of-bag estimate. We selected $L = 3$ because with $O(M^3)$ different combinations of the input variables, larger values of F should not cause much of a rise in correlation while increasing strength. If the input variables in a data set are incommensurable, they are normalized by subtracting means and dividing by standard deviations, where the means and standard deviations are determined from the training set. The test set results are given in Table 3 where the third column contains the results for $F = 2$. The fourth column contains the results for individual trees computed as for Table 2.

Except for the three larger data sets, use of $F = 8$ is superflous; $F = 2$ achieves close to the minimum. On the larger data sets, $F = 8$ gives better results. Forest-RC does exceptionally well on the synthetic data sets. Overall, it compares more favorably to Adaboost than Forest-RI.

In Tables 2 and 3 there are some entries for which the selected entry is less than the one input value or with Forest-RC, less than the two-feature value. The reason this happens is that when the error rates corresponding to the two values of F are close together, then the out-of-bag estimates will select a value of F almost at random.

A small investigation was carried out to see if performance on the three larger data sets could be improved. Based on the runs with the satellite data in Section 6, we conjectured that the strength keeps rising in the larger data sets while the correlation reaches an asymptote more quickly. Therefore we did some runs with $F = 100$ on the larger data sets using 100, 100 and 200 trees in the three forests respectively. On the satellite data, the error dropped to

Table 3. Test set errors (%).

		Forest-RC		
Data set	Adaboost	Selection	Two features	One tree
Glass	22.0	24.4	23.5	42.4
Breast cancer	3.2	3.1	2.9	5.8
Diabetes	26.6	23.0	23.1	32.1
Sonar	15.6	13.6	13.8	31.7
Vowel	4.1	3.3	3.3	30.4
Ionosphere	6.4	5.5	5.7	14.2
Vehicle	23.2	23.1	22.8	39.1
German credit	23.5	22.8	23.8	32.6
Image	1.6	1.6	1.8	6.0
Ecoli	14.8	12.9	12.4	25.3
Votes	4.8	4.1	4.0	8.6
Liver	30.7	27.3	27.2	40.3
Letters	3.4	3.4	4.1	23.8
Sat-images	8.8	9.1	10.2	17.3
Zip-code	6.2	6.2	7.2	22.7
Waveform	17.8	16.0	16.1	33.2
Twonorm	4.9	3.8	3.9	20.9
Threenorm	18.8	16.8	16.9	34.8
Ringnorm	6.9	4.8	4.6	24.6

8.5%, on the letters data to 3.0%, but the zip-code test set error did not decrease. Acting on an informed hunch, I tried Forest-RI with $F = 25$. The zip-code test set error dropped to 5.8%. These are the lowest test set errors so far achieved on these three data sets by tree ensembles.

5.1. Categorical variables

Some or all of the input variables may be categoricals and since we want to define additive combinations of variables, we need to define how categoricals will be treated so they can be combined with numerical variables. My approach is that each time a categorical variable is selected to split on at a node, to select a random subset of the categories of the variable, and define a substitute variable that is one when the categorical value of the variable is in the subset and zero outside.

Since a categorical variable with I values can be coded into $I - 1$ dummy $0-1$ variables, we make the variable $I - 1$ times as probable as a numeric variable to be selected in node splitting. When many of the variables are categorical, using a low value of F results in low correlation, but also low strength. F must be increased to about two-three times $\text{int}(\log_2 M + 1)$ to get enough strength to provide good test set accuracy.

For instance, on the DNA data set having 60 four-valued categorical values, 2,000 examples in the training set and 1,186 in the test set, using Forest-RI with $F = 20$ gave a test set error rate of 3.6% (4.2% for Adaboost). The soybean data has 685 examples, 35 variables, 19 classes, and 15 categorical variables. Using Forest-RI with $F = 12$ gives a test set error of 5.3% (5.8% for Adaboost). Using Forest-RC with combinations of 3 and $F = 8$ gives an error of 5.5%.

One advantage of this approach is that it gets around the difficulty of what to do with categoricals that have many values. In the two-class problem, this can be avoided by using the device proposed in Breiman et al. (1985) which reduces the search for the best categorical split to an $O(I)$ computation. For more classes, the search for the best categorical split is an $O(2^{I-1})$ computation. In the random forest implementation, the computation for any categorical variable involves only the selection of a random subset of the categories.

6. Empirical results on strength and correlation

The purpose of this section is to look at the effect of strength and correlation on the generalization error. Another aspect that we wanted to get more understanding of was the lack of sensitivity in the generalization error to the group size F. To conduct an empirical study of the effects of strength and correlation in a variety of data sets, out-of-bag estimates of the strength and correlation, as described in Section 3.1, were used.

We begin by running Forest-RI on the sonar data (60 inputs, 208 examples) using from 1 to 50 inputs. In each iteration, 10% of the data was split off as a test set. Then F, the number of random inputs selected at each node, was varied from 1 to 50. For each value of F, 100 trees were grown to form a random forest and the terminal values of test set error, strength, correlation, etc. recorded. Eighty iterations were done, each time removing a random 10% of the data for use as a test set, and all results averaged over the 80 repetitions. Altogether, 400,000 trees were grown in this experiment.

The top graph of figure 1, plots the values of strength and correlation vs. F. The result is fascinating. Past about $F = 4$ the strength remains constant; adding more inputs does not help. But the correlation continues to increase. The second graph plots the test set errors and the out-of-bag estimates of the generalization error against F. The out-of-bag estimates are more stable. Both show the same behavior—a small drop from $F = 1$ out to F about 4–8, and then a general, gradual increase. This increase in error tallies with the beginning of the constancy region for the strength.

Figure 2 has plots for similar runs on the breast data set where features consisting of random combinations of three inputs are used. The number of these features was varied from 1 to 25. Again, the correlation shows a slow rise, while the strength stays virtually constant, so that the minimum error is at $F = 1$. The surprise in these two figures is the relative constancy of the strength. Since the correlations are slowly but steadily increasing, the lowest error occurs when only a few inputs or features are used.

Since the larger data sets seemed to have a different behavior than the smaller, we ran a similar experiment on the satellite data set. The number of features, each consisting of a random sum of two inputs, was varied from 1 to 25, and for each, 100 classifiers were combined. The results are shown in figure 3. The results differ from those on the smaller

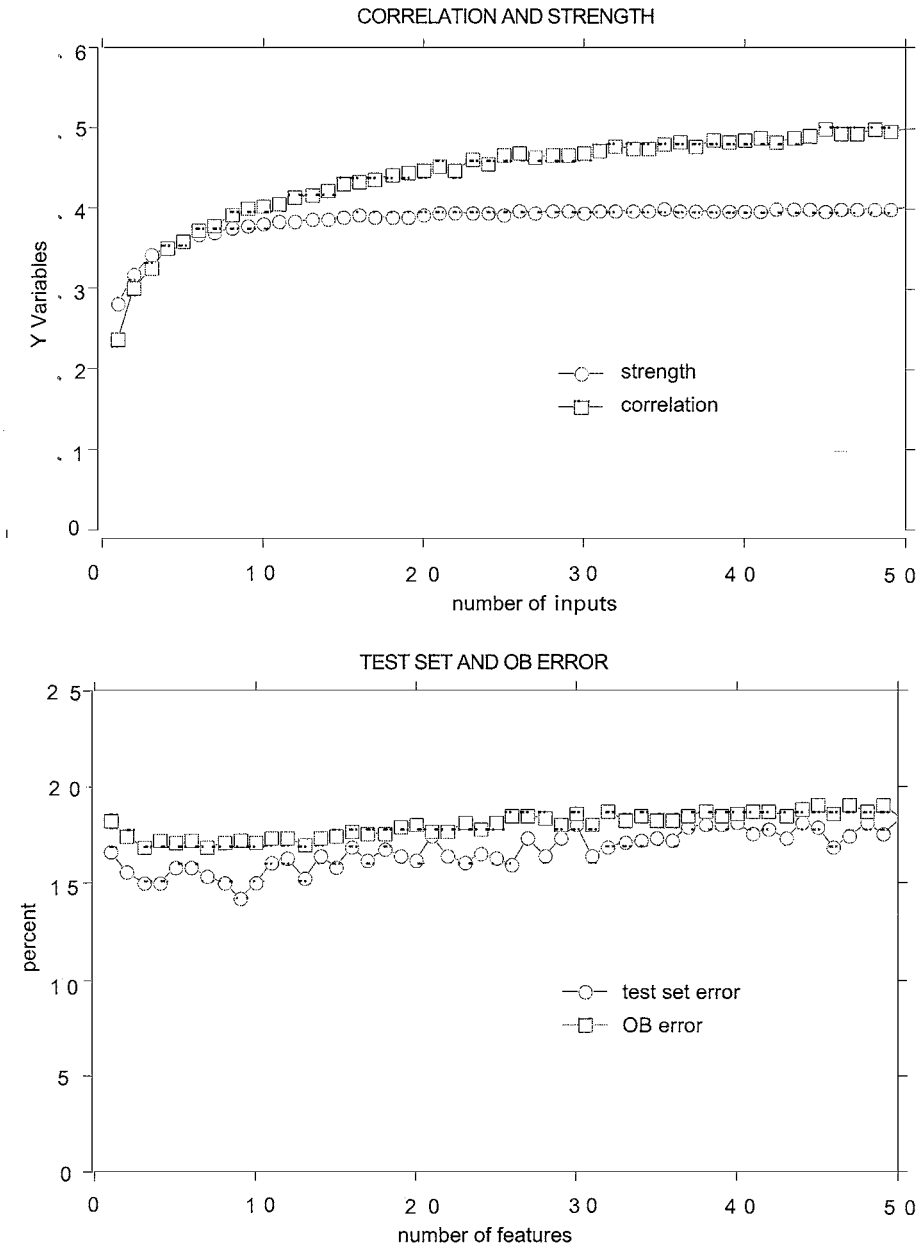


Figure 1. Effect of number of inputs on sonar data.

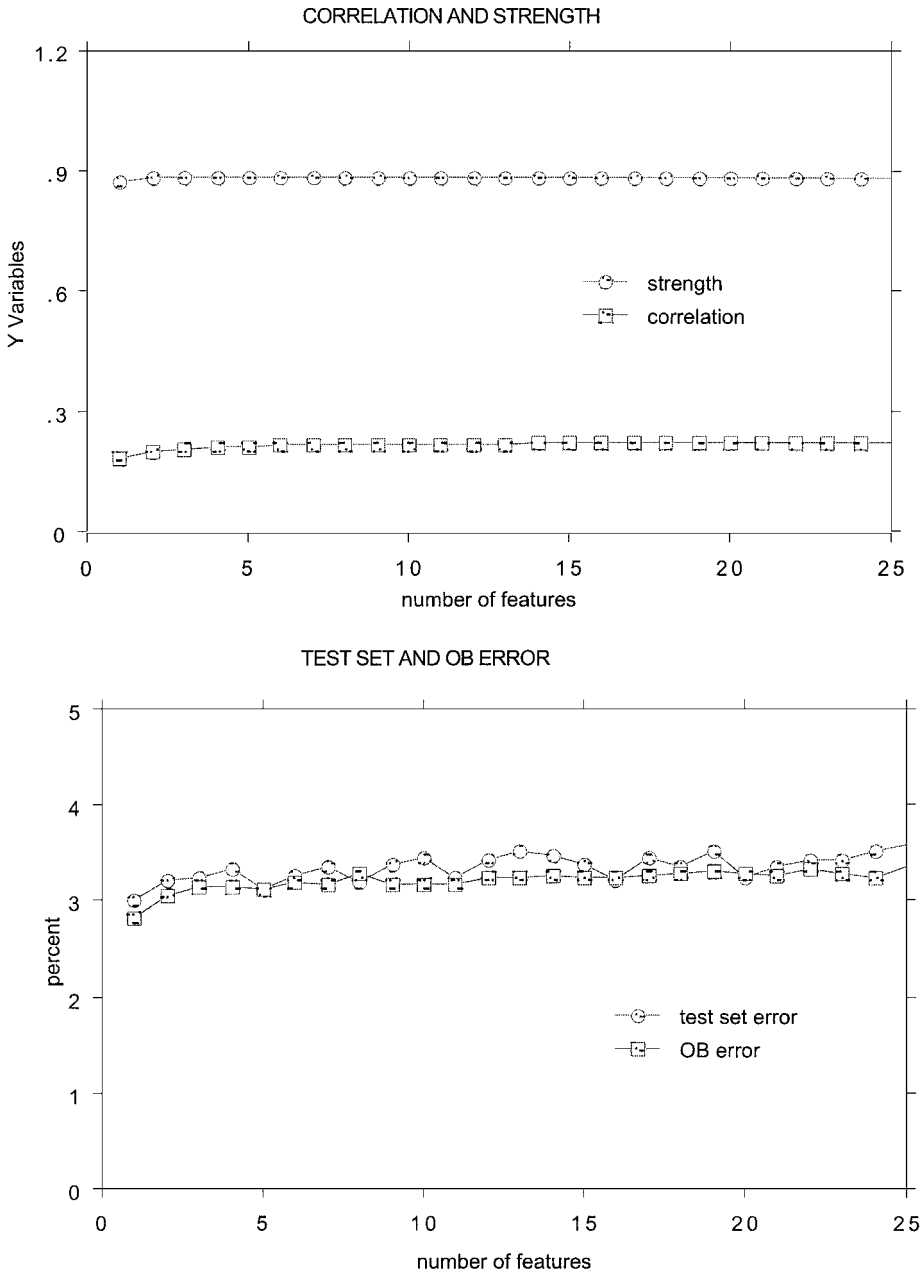


Figure 2. Effect of the number of features on the breast data set.

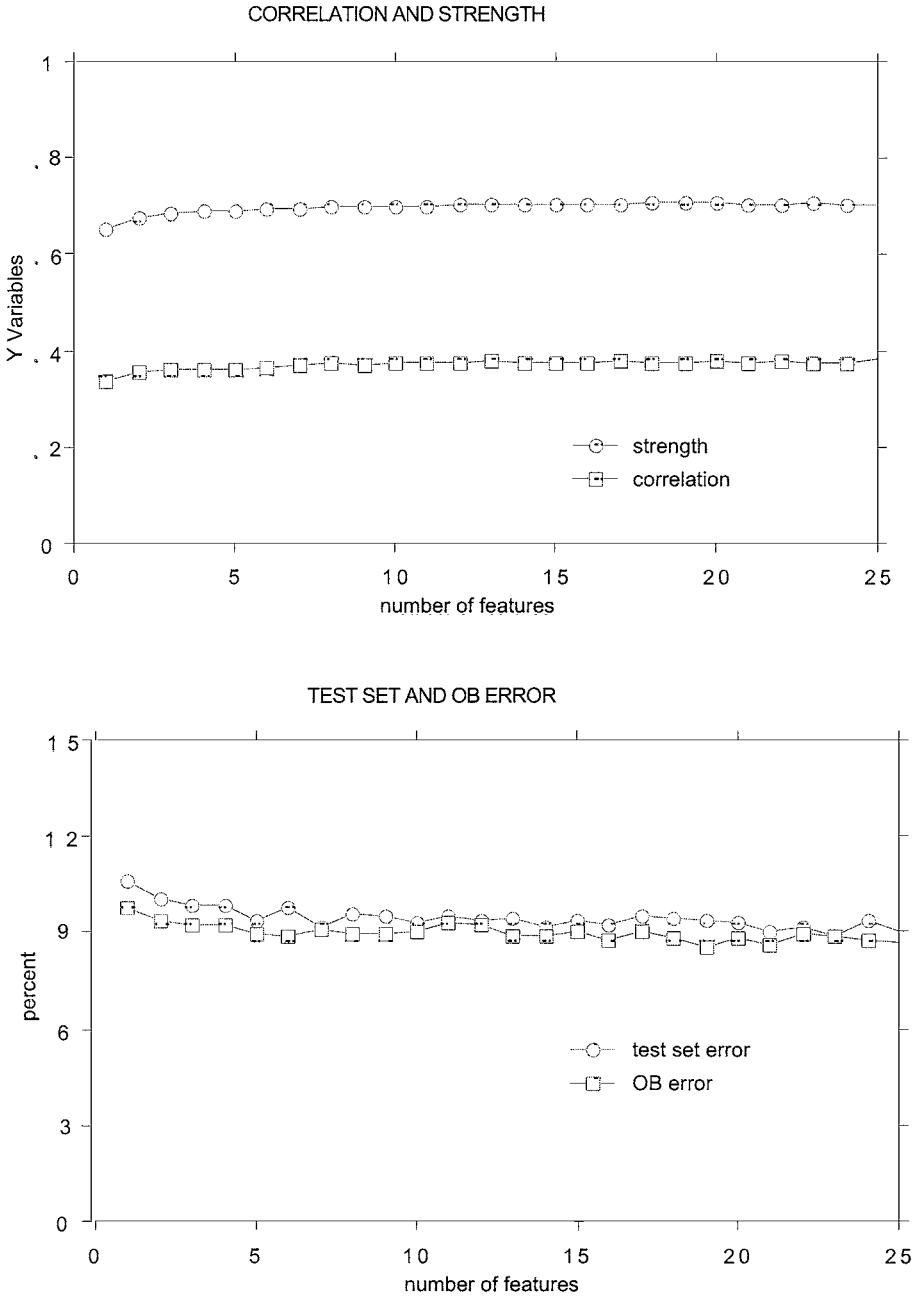


Figure 3. Effect of number of features on satellite data.

data sets. Both the correlation and strength show a small but steady increase. The error rates show a slight decrease. We conjecture that with larger and more complex data sets, the strength continues to increase longer before it plateaus out.

Our results indicate that better (lower generalization error) random forests have lower correlation between classifiers and higher strength. The randomness used in tree construction has to aim for low correlation $\bar{\rho}$ while maintaining reasonable strength. This conclusion has been hinted at in previous work. Dietterich (1998) has measures of dispersion of an ensemble and notes that more accurate ensembles have larger dispersion. Freund (personal communication) believes that one reason why Adaboost works so well is that at each step it tries to decouple the next classifier from the current one. Amit et al. (1999) give an analysis to show that the Adaboost algorithm is aimed at keeping the covariance between classifiers small.

7. Conjecture: Adaboost is a random forest

Various classifiers can be modified to use both a training set and a set of weights on the training set. Consider the following random forest: a large collection of K different sets of non-negative sum-one weights on the training set is defined. Denote these weights by $\mathbf{w}(1), \mathbf{w}(2), \ldots, \mathbf{w}(K)$. Corresponding to these weights are probabilities $p(1), p(2), \ldots, p(K)$ whose sum is one. Draw from the integers $1, \ldots, K$ according to these probabilities. The outcome is Θ. If $\Theta = k$ grow the classifier $h(\mathbf{x}, \Theta)$ using the training set with weights $\mathbf{w}(k)$.

In its original version, Adaboost (Freund & Schapire, 1996) is a deterministic algorithm that selects the weights on the training set for input to the next classifier based on the misclassifications in the previous classifiers. In our experiment, random forests were produced as follows: Adaboost was run 75 times on a data set producing sets of non-negative sum-one weights $\mathbf{w}(1), \mathbf{w}(2), \ldots, \mathbf{w}(50)$ (the first 25 were discarded). The probability for the kth set of weights is set proportional to $Q(\mathbf{w}_k) = \log[(1 - error(k))/error(k)]$ where $error(k)$ is the $\mathbf{w}(k)$ weighted training set error of the kth classifier. Then the forest is run 250 times.

This was repeated 100 times on a few data sets, each time leaving out 10% as a test set and then averaging the test set errors. On each data set, the Adaboost error rate was very close to the random forest error rate. A typical result is on the Wisconsin Breast Cancer data where Adaboost produced an average of 2.91% error and the random forest produced 2.94%.

In the Adaboost algorithm, $\mathbf{w}(k+1) = \phi(\mathbf{w}(k))$ where ϕ is a function determined by the base classifier. Denote the kth classifier by $h(\mathbf{x}, \mathbf{w}_k)$. The vote of the kth classifier is weighted by $Q(w_k)$ so the normalized vote for class j at $\mathbf{x}$ equals

$$\sum_k I(h(\mathbf{x}, \mathbf{w}_k) = j) Q(\mathbf{w}_k) \Big/ \sum_k Q(\mathbf{w}_k). \tag{10}$$

For any function f defined on the weight space, define the operator $\mathbf{T} f(\mathbf{w}) = f(\phi(\mathbf{w}))$. We conjecture that $\mathbf{T}$ is ergodic with invariant measure $\pi(d\mathbf{w})$. Then (10) will converge to $E_{Q\pi}[I(h(\mathbf{x}, \mathbf{w}) = j)]$ where the distribution $Q\pi(d\mathbf{w}) = Q(\mathbf{w})\pi(d\mathbf{w}) / \int Q(\mathbf{v})\pi(d\mathbf{v})$. If this conjecture is true, then Adaboost is equivalent to a random forest where the weights on the training set are selected at random from the distribution $Q\pi$.

Its truth would also explain why Adaboost does not overfit as more trees are added to the ensemble—an experimental fact that has been puzzling. There is some experimental evidence that Adaboost may overfit if run thousands of times (Grove & Schuurmans, 1998), but these runs were done using a very simple base classifier and may not carry over to the use of trees as the base classifiers. My experience running Adaboost out to 1,0000 trees on a number of data sets is that the test set error converges to an asymptotic value.

The truth of this conjecture does not solve the problem of how Adaboost selects the favorable distributions on the weight space that it does. Note that the distribution of the weights will depend on the training set. In the usual random forest, the distribution of the random vectors does not depend on the training set.

8. The effects of output noise

Dietterich (1998) showed that when a fraction of the output labels in the training set are randomly altered, the accuracy of Adaboost degenerates, while bagging and random split selection are more immune to the noise. Since some noise in the outputs is often present, robustness with respect to noise is a desirable property. Following Dietterich (1998) the following experiment was done which changed about one in twenty class labels (injecting 5% noise).

For each data set in the experiment, 10% at random is split off as a test set. Two runs are made on the remaining training set. The first run is on the training set as is. The second run is on a noisy version of the training set. The noisy version is gotten by changing, at random, 5% of the class labels into an alternate class label chosen uniformly from the other labels.

This is repeated 50 times using Adaboost (deterministic version), Forest-RI and Forest-RC. The test set results are averaged over the 50 repetitions and the percent increase due to the noise computed. In both random forests, we used the number of features giving the lowest test set error in the Section 5 and 6 experiments. Because of the lengths of the runs, only the 9 smallest data sets are used. Table 4 gives the increases in error rates due to the noise.

Table 4. Increases in error rates due to noise (%).

Data set	Adaboost	Forest-RI	Forest-RC
Glass	1.6	.4	−.4
Breast cancer	43.2	1.8	11.1
Diabetes	6.8	1.7	2.8
Sonar	15.1	−6.6	4.2
Ionosphere	27.7	3.8	5.7
Soybean	26.9	3.2	8.5
Ecoli	7.5	7.9	7.8
Votes	48.9	6.3	4.6
Liver	10.3	−.2	4.8

Adaboost deteriorates markedly with 5% noise, while the random forest procedures generally show small changes. The effect on Adaboost is curiously data set dependent, with the two multiclass data sets, glass and ecoli, along with diabetes, least effected by the noise. The Adaboost algorithm iteratively increases the weights on the instances most recently misclassified. Instances having incorrect class labels will persist in being misclassified. Then, Adaboost will concentrate increasing weight on these noisy instances and become warped. The random forest procedures do not concentrate weight on any subset of the instances and the noise effect is smaller.

9. Data with many weak inputs

Data sets with many weak inputs are becoming more common, i.e. in medical diagnosis, document retrieval, etc. The common characteristics is no single input or small group of inputs can distinguish between the classes. This type of data is difficult for the usual classifiers—neural nets and trees.

To see if there is a possibility that Forest-RI methods can work, the following 10 class, 1,000 binary input data, was generated: (rnd is a uniform random number, selected anew each time it appears)

```
do j=1,10
do k=1,1000
p(j,k)=.2*rnd+.01
end do
end do

do j=1,10
do i=1, nint(400*rnd)    !nint=nearest integer
k=nint(1000*rnd)
p(j,k)=p(j,k)+.4*rnd
end do
end do

do n=1,N
j=nint(10*rnd)
do m=1,1000
if (rnd<p(j,m) )then
x(m,n)=1
else
x(m,n)=0
end if
y(n)=j       ! y(n) is the class label of the nth example
end do
end do
```

This code generates a set of probabilities {p(j, m)} where j is the class label and m is the input number. Then the inputs for a class j example are a string of M binary variables with the mth variable having probability p(j, m) of being one.

For the training set, N = 1,000. A 4,000 example test set was also constructed using the same {p (j, k)}. Examination of the code shows that each class has higher underlying probability at certain locations. But the total over all classes of these locations is about 2,000, so there is significant overlap. Assuming one knows all of the {p (j, k)}, the Bayes error rate for the particular {p (j, m)} computed in our run is 1.0%.

Since the inputs are independent of each other, the Naive Bayes classifier, which estimates the {p (j,k)} from the training data is supposedly optimal and has an error rate of 6.2%. This is not an endorsement of Naive Bayes, since it would be easy to create a dependence between the inputs which would increase the Naive Bayes error rate. I stayed with this example because the Bayes error rate and the Naive Bayes error rate are easy to compute.

I started with a run of Forest-RI with $F = 1$. It converged very slowly and by 2,500 iterations, when it was stopped, it had still not converged. The test set error was 10.7%. The strength was .069 and the correlation .012 with a c/s2 ratio of 2.5. Even though the strength was low, the almost zero correlation meant that we were adding small increments of accuracy as the iterations proceeded.

Clearly, what was desired was an increase in strength while keeping the correlation low. Forest-RI was run again using $F = \text{int}(\log_2 M + 1) = 10$. The results were encouraging. It converged after 2,000 iterations. The test set error was 3.0%. The strength was .22, the correlation .045 and c/s2 = .91. Going with the trend, Forest-RI was run with $F = 25$ and stopped after 2,000 iterations. The test set error was 2.8%. Strength was .28, correlation .065 and c/s2 = .83.

It's interesting that Forest-RI could produce error rates not far above the Bayes error rate. The individual classifiers are weak. For $F = 1$, the average tree error rate is 80%; for $F = 10$, it is 65%; and for $F = 25$, it is 60%. Forests seem to have the ability to work with very weak classifiers as long as their correlation is low. A comparison using Adaboost was tried, but I can't get Adaboost to run on this data because the base classifiers are too weak.

10. Exploring the random forest mechanism

A forest of trees is impenetrable as far as simple interpretations of its mechanism go. In some applications, analysis of medical experiments for example, it is critical to understand the interaction of variables that is providing the predictive accuracy. A start on this problem is made by using internal out-of-bag estimates, and verification by reruns using only selected variables.

Suppose there are M input variables. After each tree is constructed, the values of the mth variable in the out-of-bag examples are randomly permuted and the out-of-bag data is run down the corresponding tree. The classification given for each $\mathbf{x}_n$ that is out of bag is saved. This is repeated for $m = 1, 2, \ldots, M$. At the end of the run, the plurality of out-of-bag class votes for $\mathbf{x}_n$ with the mth variable noised up is compared with the true class label of $\mathbf{x}_n$ to give a misclassification rate.

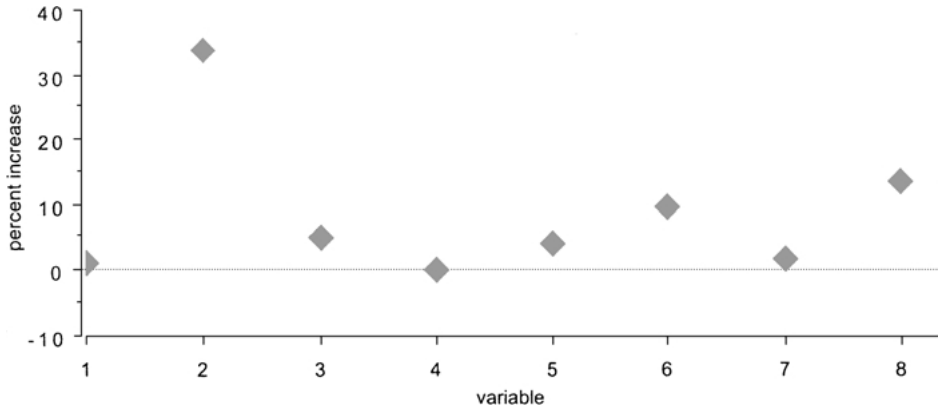


Figure 4. Measure of variable importance—diabetes data.

The output is the percent increase in misclassification rate as compared to the out-of-bag rate (with all variables intact). We get these estimates by using a single run of a forest with 1,000 trees and no test set. The procedure is illustrated by examples.

In the diabetes data set, using only single variables with $F = 1$, the rise in error due to the noising of variables is given in figure 4

The second variable appears by far the most important followed by variable 8 and variable 6. Running the random forest in 100 repetitions using only variable 2 and leaving out 10% each time to measure test set error gave an error of 29.7%, compared with 23.1% using all variables. But when variable 8 is added, the error falls only to 29.4%. When variable 6 is added to variable 2, the error falls to 26.4%.

The reason that variable 6 seems important, yet is no help once variable 2 is entered is a characteristic of how dependent variables affect prediction error in random forests. Say there are two variables x_1 and x_2 which are identical and carry significant predictive information. Because each gets picked with about the same frequency in a random forest, noising each separately will result in the same increase in error rate. But once x_1 is entered as a predictive variable, using x_2 in addition will not produce any decrease in error rate. In the diabetes data set, the 8th variable carries some of the same information as the second. So it does not add predictive accuracy when combined with the second.

The relative magnitudes of rises in error rates are fairly stable with respect to the input features used. The experiment above was repeated using combinations of three inputs with $F = 2$. The results are in figure 5.

Another interesting example is the voting data. This has 435 examples corresponding to 435 Congressmen and 16 variables reflecting their yes-no votes on 16 issues. The class variable is Republican or Democrat. To see which issues were most important, we again ran the noising variables program generating 1,000 trees. The lowest error rate on the original data was gotten using single inputs with $F = 5$, so these parameters were used in the run. The results in figure 6.

Variable 4 stands out—the error triples if variable 4 is noised. We reran this data set using only variable 4. The test set error is 4.3%, about the same as if all variables were used. The

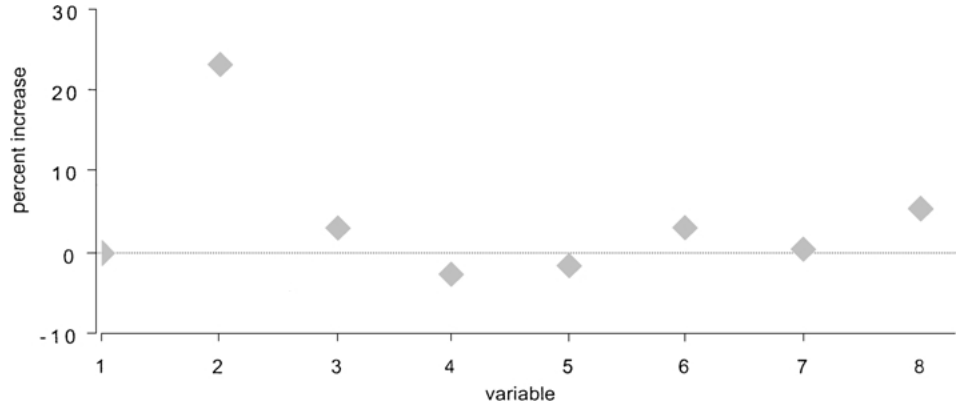


Figure 5. Measure of variable importance-diabetes data.

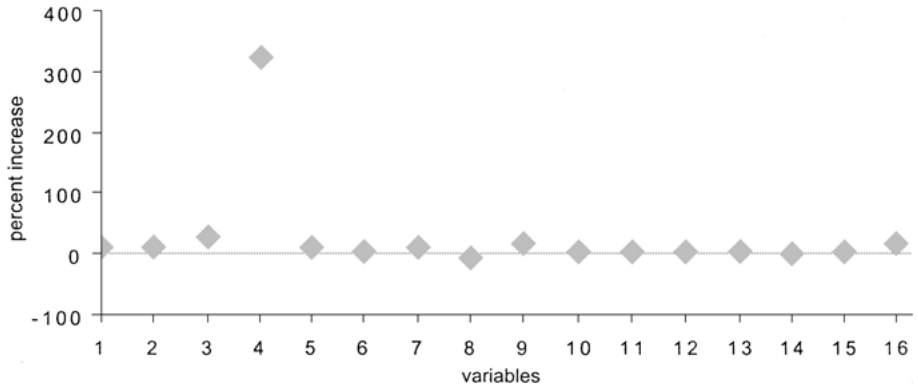


Figure 6. Measure of variable importance–votes data.

vote on 4 separates Republicans from Democrats almost as well as the vote on 4 combined with the votes on all other 15 issues.

The approach given in this section is only a beginning. More research will be necessary to understand how to give a more complete picture.

11. Random forests for regression

Random forests for regression are formed by growing trees depending on a random vector Θ such that the tree predictor $h(\mathbf{x}, \Theta)$ takes on numerical values as opposed to class labels. The output values are numerical and we assume that the training set is independently drawn from the distribution of the random vector $Y, \mathbf{X}$. The mean-squared generalization error for any numerical predictor $h(\mathbf{x})$ is

$$E_{\mathbf{X},Y}(Y - h(\mathbf{X}))^2 \tag{11}$$

The random forest predictor is formed by taking the average over k of the trees $\{h(\mathbf{x}, \Theta_k)\}$. Similarly to the classification case, the following holds:

Theorem 11.1. *As the number of trees in the forest goes to infinity, almost surrely,*

$$E_{\mathbf{X},Y}(Y - av_k h(\mathbf{X}, \Theta_k))^2 \to E_{\mathbf{X},Y}(Y - E_\Theta h(\mathbf{X}, \Theta))^2. \tag{12}$$

Proof: see Appendix I. □

Denote the right hand side of (12) as PE*(forest)—the generalization error of the forest. Define the average generalization error of a tree as:

$$\text{PE}^*(\text{tree}) = E_\Theta\, E_{\mathbf{X},Y}(Y - h(\mathbf{X}, \Theta))^2$$

Theorem 11.2. *Assume that for all* Θ, $EY = E_{\mathbf{X}} h(\mathbf{X}, \Theta)$. *Then*

$$PE^*(forest) \le \bar{\rho} PE^*(tree)$$

where $\bar{\rho}$ *is the weighted correlation between the residuals* $Y - h(\mathbf{X}, \Theta)$ *and* $Y - h(X, \Theta')$ *where* Θ, Θ' *are independent.*

Proof:

$$\begin{aligned} PE^*(forest) &= E_{\mathbf{X},Y}[E_\Theta(Y - h(\mathbf{X}, \Theta)]^2 \\ &= E_\Theta E_{\Theta'} E_{\mathbf{X},Y}(Y - h(\mathbf{X}, \Theta))(Y - h(\mathbf{X}, \Theta')) \end{aligned} \tag{13}$$

The term on the right in (13) is a covariance and can be written as:

$$E_\Theta E_{\Theta'}(\rho(\Theta, \Theta') sd(\Theta) sd(\Theta'))$$

where $sd(\Theta) = \sqrt{E_{\mathbf{X},Y}(Y - h(\mathbf{X}, \Theta))^2}$. Define the weighted correlation as:

$$\bar{\rho} = E_\Theta E_{\Theta'}(\rho(\Theta, \Theta') sd(\Theta) sd(\Theta'))/(E_\Theta sd(\Theta))^2 \tag{14}$$

Then

$$PE^*(forest) = \bar{\rho}(E_\Theta sd(\Theta))^2 \le \bar{\rho} PE^*(tree). \qquad \square$$

Theorem (11.2) pinpoints the requirements for accurate regression forests—low correlation between residuals and low error trees. The random forest decreases the average error of the trees employed by the factor $\bar{\rho}$. The randomization employed needs to aim at low correlation.

12. Empirical results in regression

In regression forests we use random feature selection on top of bagging. Therefore, we can use the monitoring provided by out-of-bag estimation to give estimates of PE*(forest), PE*(tree) and $\bar{\rho}$. These are derived similarly to the estimates in classification. Throughout, features formed by a random linear sum of two inputs are used. We comment later on how many of these features to use to determine the split at each node. The more features used, the lower PE*(tree) but the higher $\bar{\rho}$. In our empirical study the following data sets are used, see Table 5.

Of these data sets, the Boston Housing, Abalone and Servo are available at the UCI repository. The Robot Arm data was provided by Michael Jordan. The last three data sets are synthetic. They originated in Friedman (1991) and are also described in Breiman (1998b). These are the same data sets used to compare adaptive bagging to bagging (see Breiman, 1999), except that one synthetic data set (Peak20), which was found anomalous both by other researchers and myself, is eliminated.

The first three data sets listed are moderate in size and test set error was estimated by leaving out a random 10% of the instances, running on the remaining 90% and using the left-out 10% as a test set. This was repeated 100 times and the test set errors averaged. The abalone data set is larger with 4,177 instances and 8 input variables. It originally came with 25% of the instances set aside as a test set. We ran this data set leaving out a randomly selected 25% of the instances to use as a test set, repeated this 10 times and averaged.

Table 6 gives the test set mean-squared error for bagging, adaptive bagging and the random forest. These were all run using 25 features, each a random linear combination of two randomly selected inputs, to split each node, each feature a random combination of two inputs. All runs with all data sets, combined 100 trees. In all data sets, the rule "don't split if the node size is <5" was enforced.

An interesting difference between regression and classification is that the correlation increases quite slowly as the number of features used increases. The major effect is the decrease in PE*(tree). Therefore, a relatively large number of features are required to reduce PE*(tree) and get near optimal testset error.

Table 5. Data set summary.

Data set	Nr. inputs	#Training	#Test
Boston Housing	12	506	10%
Ozone	8	330	10%
Servo	4	167	10%
Abalone	8	4177	25%
Robot Arm	12	15,000	5000
Friedman#1	10	200	2000
Friedman#2	4	200	2000
Friedman#3	4	200	2000

Table 6. Mean-squared test set error.

Data set	Bagging	Adapt. bag	Forest
Boston Housing	11.4	9.7	10.2
Ozone	17.8	17.8	16.3
Servo $\times 10 - 2$	24.5	25.1	24.6
Abalone	4.9	4.9	4.6
Robot Arm $\times 10 - 2$	4.7	2.8	4.2
Friedman #1	6.3	4.1	5.7
Friedman #2 $\times 10 + 3$	21.5	21.5	19.6
Friedman #3 $\times 10 - 3$	24.8	24.8	21.6

The results shown in Table 6 are mixed. Random forest-random features is always better than bagging. In data sets for which adaptive bagging gives sharp decreases in error, the decreases produced by forests are not as pronounced. In data sets in which adaptive bagging gives no improvements over bagging, forests produce improvements.

For the same number of inputs combined, over a wide range, the error does not change much with the number of features. If the number used is too small, PE*(tree) becomes too large and the error goes up. If the number used is too large, the correlation goes up and the error again increases. The in-between range is usually large. In this range, as the number of features goes up, the correlation increases, but PE*(tree) compensates by decreasing.

Table 7 gives the test set errors, the out-of-bag error estimates, and the OB estimates for PE*(tree) and the correlation.

As expected, the OB Error estimates are consistently high. It is low in the robot arm data, but I believe that this is an artifact caused by separate training and test sets, where the test set may have a slightly higher error rate than the training set.

As an experiment, I turned off the bagging and replaced it by randomizing outputs (Breiman, 1998b). In this procedure, mean-zero Gaussian noise is added to each of the outputs. The standard deviation of the noise is set equal to the standard deviation of the

Table 7. Error and OB estimates.

Data Set	Test error	OB error	PE*(tree)	Cor.
Boston Housing	10.2	11.6	26.3	.45
Ozone	16.3	17.6	32.5	.55
Servo $\times 10 - 2$	24.6	27.9	56.4	.56
Abalone	4.6	4.6	8.3	.56
Robot Arm $\times 10 - 2$	4.2	3.7	9.1	.41
Friedman #1	5.7	6.3	15.3	.41
Friedman #2 $\times 10 + 3$	19.6	20.4	40.7	.51
Friedman #3 $\times 10 - 3$	21.6	22.9	48.3	.49

Table 8. Mean-squared test set error.

Data set	With bagging	With Noise
Boston Housing	10.2	9.1
Ozone	17.8	16.3
Servo $\times 10 - 2$	24.6	23.2
Abalone	4.6	4.7
Robot Arm $\times 10 - 2$	4.2	3.9
Friedman #1	5.7	5.1
Friedman #2 $\times$ 10 + 3	19.6	20.4
Friedman #3 $\times$ 10 − 3	21.6	19.8

outputs. Similar to the bagging experiments, tree construction was done using 25 features, each a random linear combination of two randomly selected inputs, to split each node. The results are given in Table 8.

The error rates on the first two data sets are the lowest to date. Overall, adding output noise works with random feature selection better than bagging. This is illustrative of the flexibility of the random forest setting—various combinations of randomness can be added to see what works the best.

13. Remarks and conclusions

Random forests are an effective tool in prediction. Because of the Law of Large Numbers they do not overfit. Injecting the right kind of randomness makes them accurate classifiers and regressors. Furthermore, the framework in terms of strength of the individual predictors and their correlations gives insight into the ability of the random forest to predict. Using out-of-bag estimation makes concrete the otherwise theoretical values of strength and correlation.

For a while, the conventional thinking was that forests could not compete with arcing type algorithms in terms of accuracy. Our results dispel this belief, but lead to interesting questions. Boosting and arcing algorithms have the ability to reduce bias as well as variance (Schapire et al., 1998). The adaptive bagging algorithm in regression (Breiman, 1999) was designed to reduce bias and operates effectively in classification as well as in regression. But, like arcing, it also changes the training set as it progresses.

Forests give results competitive with boosting and adaptive bagging, yet do not progressively change the training set. Their accuracy indicates that they act to reduce bias. The mechanism for this is not obvious. Random forests may also be viewed as a Bayesian procedure. Although I doubt that this is a fruitful line of exploration, if it could explain the bias reduction, I might become more of a Bayesian.

Random inputs and random features produce good results in classification—less so in regression. The only types of randomness used in this study is bagging and random features. It may well be that other types of injected randomness give better results. For instance, one of the referees has suggested use of random Boolean combinations of features.

An almost obvious question is whether gains in accuracy can be gotten by combining random features with boosting. For the larger data sets, it seems that significantly lower error rates are possible. On some runs, we got errors as low as 5.1% on the zip-code data, 2.2% on the letters data and 7.9% on the satellite data. The improvement was less on the smaller data sets. More work is needed on this; but it does suggest that different injections of randomness can produce better results.

A recent paper (Breiman, 2000) shows that in distribution space for two class problems, random forests are equivalent to a kernel acting on the true margin. Arguments are given that randomness (low correlation) enforces the symmetry of the kernel while strength enhances a desirable skewness at abrupt curved boundaries. Hopefully, this sheds light on the dual role of correlation and strength. The theoretical framework given by Kleinberg (2000) for Stochastic Discrimination may also help understanding.

Appendix I: Almost sure convergence

Proof of theorem 1.2: It suffices to show that there is a set of probability zero C on the sequence space $\Theta_1, \Theta_2, \ldots$ such that outside of C, for all $\mathbf{x}$,

$$\frac{1}{N}\sum_{n=1}^{N} I\left(h(\Theta_n, \mathbf{x}) = j\right) \to P_\Theta(h(\Theta, \mathbf{x}) = j).$$

For a fixed training set and fixed Θ, the set of all $\mathbf{x}$ such that $h(\Theta, \mathbf{x}) = j$ is a union of hyper-rectangles. For all $h(\Theta, \mathbf{x})$ there is only a finite number K of such unions of hyper-rectangles, denoted by $S_1, \ldots, S_K$. Define $\varphi(\Theta) = k$ if $\{\mathbf{x} : h(\Theta, \mathbf{x}) = j\} = S_k$. Let N_k be the number of times that $\varphi(\Theta_n) = k$ in the first N trials. Then

$$\frac{1}{N}\sum_{n=1}^{N} I\left(h(\Theta_n, \mathbf{x}) = j\right) = \frac{1}{N}\sum_{k} N_k I\left(\mathbf{x} \in S_k\right)$$

By the Law of Large Numbers,

$$N_k = \frac{1}{N}\sum_{n=1}^{N} I\left(\varphi(\Theta_n) = k\right)$$

converges a.s. to $P_\Theta(\varphi(\Theta) = k)$. Taking unions of all the sets on which convergence does not occur for some value of k gives a set C of zero probability such that outside of C,

$$\frac{1}{N}\sum_{n=1}^{N} I\left(h(\Theta_n, \mathbf{x}) = j\right) \to \sum_{k} P_\Theta(\varphi(\Theta) = k) I\left(\mathbf{x} \in S_k\right).$$

The right hand side is $P_\Theta(h(\Theta, \mathbf{x}) = j$. □

Proof of theorem 9.1: There are a finite set of hyper-rectangles $R_1, \ldots, R_K$, such that if $\bar{y}_k$ is the average of the training sets y-values for all training input vectors in R_k then $h(\Theta, \mathbf{x})$ has one of the values $I(\mathbf{x} \in S_k)\bar{y}_k$. The rest of the proof parallels that of Theorem 1.2. □

Appendix II: Out-of bag estimates for strength and correlation

At the end of a combination run, let

$$Q(\mathbf{x}, j) = \sum_k I(h(\mathbf{x}, \Theta_k) = j; (y, \mathbf{x}) \notin T_{k,B}) \Big/ \sum_k I((y, \mathbf{x}) \notin T_{k,B}).$$

Thus, $Q(\mathbf{x}, j)$ is the out-of-bag proportion of votes cast at $\mathbf{x}$ for class j, and is an estimate for $P_\Theta(h(\mathbf{x}, \Theta) = j)$. From Definition 2.1 the strength is the expectation of

$$P_\Theta(h(\mathbf{x}, \Theta) = y) - \max_{j \neq \mathbf{y}} P_\Theta(h(\mathbf{x}, \Theta) = j)$$

Substituting $Q(\mathbf{x}, j)$, $Q(\mathbf{x}, y)$ for $P_\Theta(h(\mathbf{x}, \Theta) = j)$, $P_\Theta(h(\mathbf{x}, \Theta) = y)$ in this latter expression and taking the average over the training set gives the strength estimate.

From Eq. (7),

$$\bar{\rho} = \text{var}(mr)/(E_\Theta sd(\Theta))^2.$$

The variance of mr is

$$E_{\mathbf{X},Y}[P_\Theta(h(\mathbf{x}, \Theta) = y) - \max_{j \neq \mathbf{y}} P_\Theta(h(\mathbf{x}, \Theta) = j)]^2 - s^2 \tag{A1}$$

where s is the strength. Replacing the first term in (A1) by the average over the training set of

$$(Q(\mathbf{x}, y) - \max_{j \neq \mathbf{y}} Q(\mathbf{x}, j))^2$$

and s by the out-of-bag estimate of s gives the estimate of var(mr). The standard deviation is given by

$$sd(\Theta) = [p_1 + p_2 + (p_1 - p_2)^2]^{1/2} \tag{A2}$$

where

$$p_1 = E_{\mathbf{X},Y}(h(\mathbf{X}, \Theta) = Y)$$
$$p_2 = E_{\mathbf{X},Y}(h(\mathbf{X}, \Theta) = \hat{j}(\mathbf{X}, Y))$$

After the kth classifier is constructed, $Q(\mathbf{x}, j)$ is computed, and used to compute $\hat{j}(\mathbf{x}, y)$ for every example in the training set. Then, let p_1 be the average over all $(y, \mathbf{x})$ in the training set but not in the kth bagged training set of $I(h(\mathbf{x}, \Theta_k) = y)$. Then p_2 is the similar average of $I(h(\mathbf{x}, \Theta_k) = \hat{j}(\mathbf{x}, y))$. Substitute these estimates into (A2) to get an estimate of $sd(\Theta_k)$. Average the $sd(\Theta_k)$ over all k to get the final estimate of $sd(\Theta)$.

References

Amit, Y. & Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural Computation, 9*, 1545–1588.

Amit, Y., Blanchard, G., & Wilder, K. (1999). Multiple randomized classifiers: MRCL Technical Report, Department of Statistics, University of Chicago.

Bauer, E. & Kohavi, R. (1999). An empirical comparison of voting classification algorithms. *Machine Learning, 36*(1/2), 105–139.

Breiman, L. (1996a). Bagging predictors. *Machine Learning 26*(2), 123–140.

Breiman, L. (1996b). Out-of-bag estimation, ftp.stat.berkeley.edu/pub/users/breiman/OOBestimation.ps

Breiman, L. (1998a). Arcing classifiers (discussion paper). *Annals of Statistics, 26*, 801–824.

Breiman. L. (1998b). Randomizing outputs to increase prediction accuracy. Technical Report 518, May 1, 1998, Statistics Department, UCB (in press, Machine Learning).

Breiman, L. 1999. Using adaptive bagging to debias regressions. Technical Report 547, Statistics Dept. UCB.

Breiman, L. 2000. Some infinity theory for predictor ensembles. Technical Report 579, Statistics Dept. UCB.

Dietterich, T. (1998). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization, *Machine Learning*, 1–22.

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

Grove, A. & Schuurmans, D. (1998). Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Trans. on Pattern Analysis and Machine Intelligence, 20*(8), 832–844.

Kleinberg, E. (2000). On the algorithmic implementation of stochastic discrimination. *IEEE Trans. on Pattern Analysis and Machine Intelligence, 22*(5), 473–490.

Schapire, R., Freund, Y., Bartlett, P., & Lee, W. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics, 26*(5), 1651–1686.

Tibshirani, R. (1996). Bias, variance, and prediction error for classification rules. Technical Report, Statistics Department, University of Toronto.

Wolpert, D. H. & Macready, W. G. (1997). An efficient method to estimate Bagging's generalization error (in press, Machine Learning).

Received November 30, 1999
Revised April 11, 2001
Accepted April 11, 2001
Final manuscript April 11, 2001