Online Bagging and Boosting

Nikunj C. Oza and Stuart Russell
Computer Science Division
University of California
Berkeley, CA 94720-1776
{*oza,russell*}*@cs.berkeley.edu*

Abstract

Bagging and *boosting* are well-known ensemble learning methods. They combine multiple learned base models with the aim of improving generalization performance. To date, they have been used primarily in *batch* mode, and no effective *online* versions have been proposed. We present simple online bagging and boosting algorithms that we claim perform as well as their batch counterparts.

1 Introduction

Traditional supervised learning algorithms classify examples[1] based on a single model such as a decision tree or neural network. *Ensemble* learning algorithms, of which there are many varieties, combine the predictions of multiple *base models*, each of which is learned using a traditional algorithm. *Bagging* [3] and *Boosting* [8] are well-known ensemble learning algorithms that have been shown to be very effective in improving generalization performance compared to individual base models [1]. Theoretical analysis of boosting's performance supports these results [9].

In this paper, we develop *online* versions of these algorithms. Online learning algorithms process each training instance once "on arrival" without the need for storage and reprocessing, and maintain a current hypothesis that reflects all the training instances seen so far. Such algorithms have advantages over typical batch algorithms in situations where data arrive continuously. They are also useful with very large data sets on secondary storage, for which the multiple passes required by most batch algorithms are prohibitively expensive.

[1] In this paper, we only deal with the classification problem.

Batch ensemble algorithms typically use a batch learning algorithm, which we shall call L_b, to generate each base model. The first requirement of an online ensemble algorithm is an online learning algorithm for base models, which we shall call L_o. Online variants of many learning algorithms are available. A *lossless* online algorithm is one whose output hypothesis for a given training set is identical to that of the corresponding batch algorithm. Lossless online algorithms are available for decision trees [14], Naive Bayes models, and nearest-neighbor classifiers, among others. We use lossless online algorithms for decision trees and Naive Bayes models in our experiments.

Producing online versions of bagging and boosting also requires a way to mirror their specific techniques for generating multiple distinct base models. The difficulty is that both algorithms appear to require foreknowledge of the size of the training set, which is unavailable (or meaningless) in the online context. For example, bagging works by resampling the original training set of size N to produce M bootstrap training sets of size N, each of which is used to train a base model. Our online version trains M base models online. It simulates the bootstrap process by sending K copies of each new example to update each base model, where K is a suitable Poisson random variable. This simple trick yields learning behavior similar to that of batch bagging. We describe the online bagging algorithm and give theoretical results in Section 2; empirical results are provided in Section 4.

Boosting is a somewhat more complex process that generates a series of base models $h_1, \ldots, h_M$. Each base model h_m is learned from a weighted training set whose weights are determined by the classification errors of the preceding model h_{m-1}. Specifically, the examples *misclassified* by h_{m-1} are given *more* weight in the training set for h_m, such that the weights of all the misclassified examples constitute half the total weight of the training set. As with bagging, this type of "normalization" appears to require foreknowledge of the complete training set. Again, we use a Poisson sampling process to approximate the reweighting algorithm. The online boosting algorithm is described in detail in Section 3. Empirical results are given in Section 4.

The topic of online bagging and boosting has received very little attention in the literature. In [5], an ensemble of three neural networks was trained using boosting in an online fashion; the method proposed therein often discards substantial amounts of data in the process of drawing the desired distribution of data for its base models. More recently, a "blocked" online boosting algorithm has been proposed [4] that trains several base models using consecutive subsets of training examples of some fixed size; this process also discards a fraction of the data received. Neither of these algorithms is directly comparable to our approach, which focuses on reproducing the advantages of bagging and boosting in an online setting. In [7], an online bagging algorithm is proposed; it attempts to simulate the bootstrap process by sending each new training example to update each base model with some probability that the user fixes in advance. In experiments with various such probabilities, their online bagging algorithm never performed better than a single decision tree. The same paper also proposes an online boosting algorithm that is an online version of Arc-x4 [3], i.e., each example is given weight $1 + m^4$ to update each base model, where m is the number of previous base models that currently misclassify that example. The algorithm was applied to the branch prediction problem from computer architecture. The results suggest that, given limited memory, a boosted ensemble with a greater number of smaller decision trees is generally superior to one with fewer large trees.

Potentially interesting parallels can be drawn between our approach and the Winnow [11] and Weighted Majority [12] algorithms. These algorithms use a fixed set of base models that are trained online and combined using weights that depend on the training set performance of each base model. Their performance can be shown to be almost as good as that of the best component model for any training sequence. On the other hand, ensemble algorithms generally perform *better* than all of their component models. Comparing them to online bagging or boosting, we see that they send identical training sequences to each base model; hence, base model diversity, which is known to aid ensemble performance [13], must be built in *a priori* rather than emerging from the data itself. One can imagine hybrid approaches; it may also be the case that amortized analysis techniques can be applied to our algorithms.

2 Online Bagging

Given a training dataset of size N, standard batch bagging creates M base models,[2] each trained on a bootstrap sample of size N created by drawing random samples with replacement from the original training set. In the following pseudocode, T is the original training set of N examples and M is the number of base models to be learned.:

Bagging(T,M)

- For each $m \in \{1, 2, \ldots, M\}$,
  - $T_m = Sample\_With\_Replacement(T, N)$
  - $h_m = L_b(T_m)$
- Return $\{h_1, h_2, \ldots, h_M\}$

Each base model's training set contains each of the original training examples K times where

$$P(K = k) = \binom{N}{k} \left(\frac{1}{N}\right)^k \left(1 - \frac{1}{N}\right)^{N-k}$$

which is the binomial distribution. As $N \to \infty$, the distribution of K tends to a Poisson(1) distribution: $K \sim \frac{exp(-1)}{k!}$. This suggests that we can perform bagging online as follows: as each training example is presented to our algorithm, for each base model, choose the example $K \sim Poisson(1)$ times and update the base model accordingly. In the pseudocode below, $\mathbf{h}$ is the set of M base models learned so far and d is the latest training example to arrive.

OnlineBagging($\mathbf{h}$, d)

For each base model h_m, ($m \in \{1, 2, \ldots, M\}$) in the ensemble,

- Set k according to $Poisson(1)$.
- Do k times
  $h_m = L_o(h_m, d)$

New instances are classified the same way in online and batch bagging—by unweighted voting of the M base models.

Online bagging is a good approximation to batch bagging to the extent that their base model learning algorithms produce similar hypotheses when trained with similar distributions of training examples. We first prove that if the same original training set is supplied to the two bagging algorithms, then the distributions

[2] The number of base models is normally chosen by trial and error but sometimes a validation set is used [6].

over the training sets supplied to the base models in batch and online bagging converge as the size of that original training set grows to infinity.

Define θ_b^m to be a vector of length N where the ith element represents the number of times that the ith original training example is included in the bootstrap training set of the mth base model under batch bagging. Sampling with replacement in the batch bagging algorithm is done by performing N trials where each trial yields one of the N training examples, all of which have equal probability $\frac{1}{N}$ of being drawn. Therefore, $\theta_b^m \sim Multinomial(N, \frac{1}{N})$, where all the training examples have equal "success probability" $\frac{1}{N}$. Define θ_o^m to be the online bagging version of θ_b^m. We mentioned earlier that, under online bagging, each training example is chosen a number of times according to a $Poisson(1)$ distribution. Since there are N training examples, there are N such trials; therefore, the total number of examples drawn has a $Poisson(N)$ distribution. Because each example has an equal probability of being drawn, we can recast sampling in the online bagging algorithm as performing $N' \sim Poisson(N)$ trials where each trial yields one of the N training examples, all of which have equal probability $\frac{1}{N}$ of being drawn. Therefore, $\theta_o \sim \sum_{t=0}^{N} P(Poisson(N) = t) Multinomial(t, \frac{1}{N})$.

Theorem As $N \to \infty$, $P(\theta_b)$ converges in distribution to $P(\theta_o)$.

Proof The probability generating function [10] for the batch bagging algorithm's sampling distribution, $Multinomial(N, \frac{1}{N})$, is

$$G_{Mult(N,\frac{1}{N})}(x_1, \ldots, x_N) = \left(\frac{1}{N}(x_1 + \ldots + x_N)\right)^N.$$

The generating function for a $Multinomial(1, \frac{1}{N})$ distribution is

$$G_{Mult(1,\frac{1}{N})}(x_1, \ldots, x_N) = \frac{1}{N}(x_1 + \ldots + x_N).$$

The generating function for a $Poisson(N)$ distribution is $G_{Poi(N)}(s) = exp(N(s-1)$. Online bagging's sampling algorithm involves performing N' $Multinomial(1, \frac{1}{N})$ trials; therefore, the generating function for online bagging's sampling distribution is

$$G_{Poi(N)}(G_{Mult(1,\frac{1}{N})}(x_1, \ldots, x_N)) = exp\Big(N\Big(\frac{1}{N}(x_1 + \ldots + x_N) - 1\Big)\Big).$$

Furthermore, it is a standard result [10] that

$$\lim_{N\to\infty} G_{Mult(N,\frac{1}{N})}(x_1, \ldots, x_N) = \lim_{N\to\infty}\Big(1 + \Big(\frac{x_1 + \ldots + x_N - N}{N}\Big)^N\Big) = exp\Big(N\Big(\frac{1}{N}(x_1 + \ldots + x_N) - 1\Big)\Big).$$

The convergence of the generating functions implies the convergence of the probabilities for every possible θ vector; therefore, the two sampling methods converge in distribution.

■

Define $Resample(\theta, T)$ to be a function that takes as input the original training set T and a vector θ which has the same length as T and whose ith element is the number of times that the ith training example from T is included in the bootstrap training set. This function returns the actual bootstrap training set induced by θ and T. We assume that the N examples in T are drawn randomly and independently from a fixed distribution. The sampling distributions of batch and online bagging induce distributions over the base hypotheses $P_{\theta_b} L_b(Resample(\theta_b, T))$ and $P_{\theta_o} L_o(Resample(\theta_o, T))$, respectively. A batch-bagged ensemble consists of M independent and identically distributed (i.i.d.) draws from $P_{\theta_b} L_b(Resample(\theta_b, T))$. An online-bagged ensemble consists of M i.i.d. draws from $P_{\theta_o} L_o(Resample(\theta_o, T))$. We would like to show that $P_{\theta_o} L_o(Resample(\theta_o, T)) \to P_{\theta_b} L_b(Resample(\theta_b, T))$. Clearly, this is not true for all learning algorithms L_b and L_o. Suppose that L_o and L_b return some null hypothesis unless the training set has exactly N examples: L_b is always given N examples, but as $N \to \infty$, the probability that L_o receives N examples tends to 0. Intuitively, we need a learning algorithm that is "well-behaved," in the sense that, as $N \to \infty$, having a few more or few less examples in the bootstrapped training set should not make a significant difference in the learning algorithm's output.

Local learning algorithms such as K-Nearest-Neighbor are clearly well-behaved in this sense. A K-Nearest Neighbor base model returns a classification for a new test example x based on the K nearest neighbors within its bootstrap training set. It can be shown easily that the distribution over the K nearest neighbors for batch bagging converges to that of online bagging as $N \to \infty$.

Simple contingency-table learning is also well-behaved. For every class c, we have $P(C = c|x) = P(x, c)/P(x)$, Since the denominator is the same for all c, we can just consider $P(x, c)$ for the purpose of classification. Define $p_{x,c}$ to be the fraction of examples within T of the form (x, c), i.e., having attribute values x and class c. Batch bagging draws bootstrap training sets according to $\theta_b \sim Multinomial(N, \frac{1}{N})$, which means it performs N i.i.d. trials in which the probability of choosing an example (x, c) is $p_{x,c}$; therefore, $P_{\theta_b}(x, c) = p_{x,c}$. Online bagging draws bootstrap training sets according to $\theta_o \sim \sum_{t=0}^{N} P(Poisson(N) = t) Multinomial(t, \frac{1}{N})$, which involves performing t i.i.d. trials in which the probability of choosing an example (x, c) is $p_{x,c}$; there-

AdaBoost$(\{(x_1,y_1),\ldots,(x_N,y_N)\},L_b,M)$

- Initialize $D_1(n)=1/N$ for all $n\in\{1,2,\ldots,N\}$.
- Do for $m=1,2,\ldots,M$:
  - 1. Call L_b with the distribution D_m.
  - 2. Get back a hypothesis $h_m: X\rightarrow Y$.
  - 3. Calculate the error of h_m : $\epsilon_m = \sum_{n:h_m(x_n)\neq y_n} D_m(n)$. If $\epsilon_m > 1/2$ then set $M=m-1$ and abort this loop.
  - 4. Set $\beta_m = \frac{\epsilon_m}{1-\epsilon_m}$.
  - 5. Update distribution D_m:

$$D_{m+1}(n)=\frac{D_m(n)}{Z_m}\times\begin{cases}\beta_m & \text{if } h_m(x_n)=y_n\\ 1 & \text{otherwise}\end{cases}$$

    where Z_m is a normalization constant chosen so that D_{m+1} is a probability distribution.

- **Output** the final hypothesis: $h_{fin}(x) = argmax_{y\in Y}\sum_{m:h_m(x)=y} log\frac{1}{\beta_m}$.

Figure 1: AdaBoost.M1 algorithm from [8]

fore,

$$P_{\theta_o}(x,c) = \sum_{t=0}^{N} P(Poisson(N)=t)P_{\theta\in Mult(t,\frac{1}{N})}(x,c)=p_{x,c}.$$

Since $P_{\theta_b}(x,c)=P_{\theta_o}(x,c)$ for all examples (x,c), the expected counts in each entry of the contingency tables are the same under online and batch bagging; therefore, the classifications of new examples have the same expectation under online and batch bagging.

We are working on describing a larger set of learning algorithms that are well-behaved.

3 Online Boosting

Our online boosting algorithm is designed to correspond to the batch boosting algorithm, AdaBoost.M1 [8]. We give the pseudocode for AdaBoost in Figure 1, where the inputs are a set of training examples $\{(x_1,y_1),\ldots,(x_N,y_N)\}$, base learning algorithm L_b, and the number of base models M to be generated. As explained earlier, AdaBoost.M1 generates a sequence of base models $h_1,\ldots,h_M$ using weighted training sets such that the training examples misclassified by model h_{m-1} are given half the total weight for model h_m and the correctly classified examples are given the remaining half of the weight.

In our online boosting algorithm pseudocode (Figure 2), $\mathbf{h_M}$ is the set of M base models learned so far, d is the latest training example to arrive, and $OnlineBase$ is the incremental learning algorithm that takes a current hypothesis and training example as input and returns an updated hypothesis. Our online boosting algorithm is similar to our online bagging algorithm except that when a base model misclassifies a training example, the Poisson distribution parameter (λ) associated with that example is increased when presented to the next base model; otherwise it is decreased. For example, in Figure 3, in the upper left corner (point "a" in the diagram) is the first training example. This example updates the first base model but is still misclassified after training, so its weight is increased (the rectangle "b" used to represent it is taller). This example with its higher weight updates the second base model and then correctly classifies it, so its weight decreases (rectangle "c"). Just as in AdaBoost, our algorithm gives the examples misclassified by one stage half the total weight in the next stage; the correctly classified examples are given the remaining half of the weight.[3] We can see this by examining the adjustments to λ_d shown in Figure 2 item 3 as follows. Suppose that λ_m^{sc} is the sum of the λ values for the examples that were classified correctly by the base model at stage m and λ_m^{sw} is the same sum for

OnlineBoosting$(\mathbf{h_M}, OnlineBase, d)$

- Set the example's "weight" $\lambda_d = 1$.
- For each base model h_m, $(m\in\{1,2,\ldots,M\})$ in the ensemble,
  - 1. Set k according to $Poisson(\lambda_d)$.
  - 2. Do k times
    $h_m = OnlineBase(h_m, d)$
  - 3. If $h_m(d)$ is the correct label,
    - then
      - $\lambda_m^{sc}\longleftarrow\lambda_m^{sc}+\lambda_d$
      - $\lambda_d\longleftarrow\lambda_d\left(\frac{N}{2\lambda_m^{sc}}\right)$
    - else
      - $\lambda_m^{sw}\longleftarrow\lambda_m^{sw}+\lambda_d$
      - $\lambda_d\longleftarrow\lambda_d\left(\frac{N}{2\lambda_m^{sw}}\right)$

To classify new examples:

- For each $m\in\{1,2,\ldots,M\}$
  Calculate $\epsilon_m=\frac{\lambda_m^{sw}}{\lambda_m^{sc}+\lambda_m^{sw}}$ and $\beta_m=\frac{\epsilon_m}{1-\epsilon_m}$
- Return $h(x) = argmax_{c\in C}\sum_{m:h_m(x)=y} log\frac{1}{\beta_m}$.

Figure 2: Online Boosting Algorithm

[3] We discuss a caveat to this point at the end of this section.

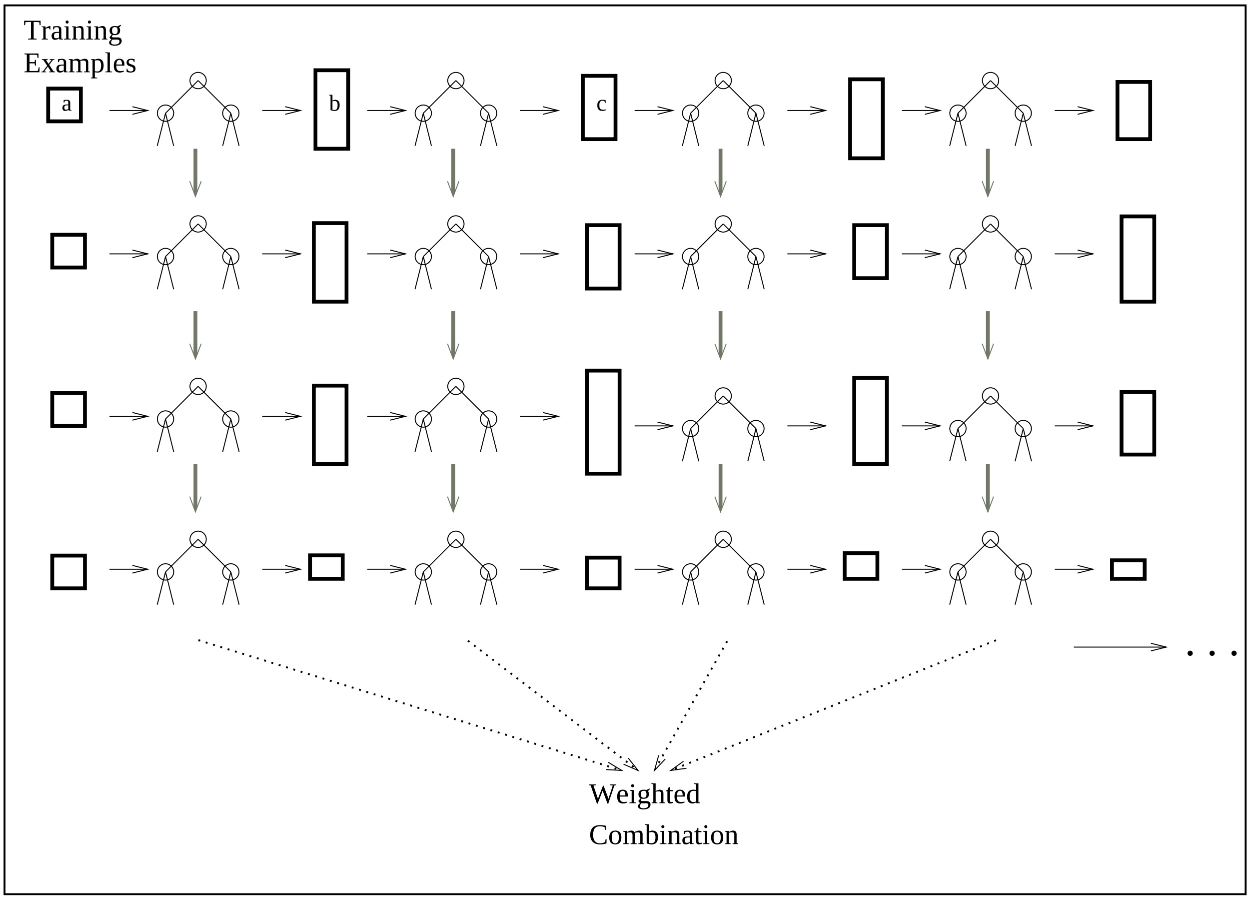

Figure 3: Illustration of online boosting in progress. Each row represents one example being passed in sequence to all the base models for updating; time runs down the diagram. Each base model (depicted as a tree) is generated by updating the base model above it with the next weighted training example. Each rectangle represents a training example—the height of the rectangle represents its weight.

incorrectly classified examples. For the next stage of boosting, we want these two sums to be scaled to the same value, just as in AdaBoost;[4] therefore, we want to find the factors f_m^c and f_m^w that scale λ_m^{sc} and λ_m^{sw} to half the total weight, respectively. The sum of all AdaBoost weights is one; therefore, the sum of all the λs for our online algorithm is N, which is the number of examples seen so far. Therefore, we get:

$$\lambda_m^{sc} f_m^c = \frac{N}{2} \Longrightarrow f_m^c = \frac{N}{2\lambda_m^{sc}}$$
$$\lambda_m^{sw} f_m^w = \frac{N}{2} \Longrightarrow f_m^w = \frac{N}{2\lambda_m^{sw}}.$$

Note that we expect that $\lambda_m^{sc} > N/2$ and $\lambda_m^{sw} < N/2$ and, therefore, that $f_m^c < 1$ and $f_m^w > 1$, which means that the weights of correctly classified examples will decrease, and the weights of incorrectly classified examples will increase, as desired.

One area of concern is that, in AdaBoost, an example's weight is adjusted based on the performance of a base model on the entire training set while in online boosting, the weight adjustment is based on the base model's performance only on the examples seen earlier. To see why this may be an issue, consider running AdaBoost and online boosting on a training set of size 10000. In AdaBoost, the first base model h_1 is generated from all 10000 examples before being tested on, say, the tenth training example. In online boosting, h_1 is generated from only the first ten examples before being tested on the tenth example. Clearly, we may expect the two h_1's to be very different; therefore, h_2 in AdaBoost and h_2 in online boosting may be presented with different weights for the tenth example. This may, in turn, lead to very different weights for

[4] In AdaBoost terminology, the examples' weights would actually be λ_d/N, but since our algorithm works with the λ values, we treat them as weights.

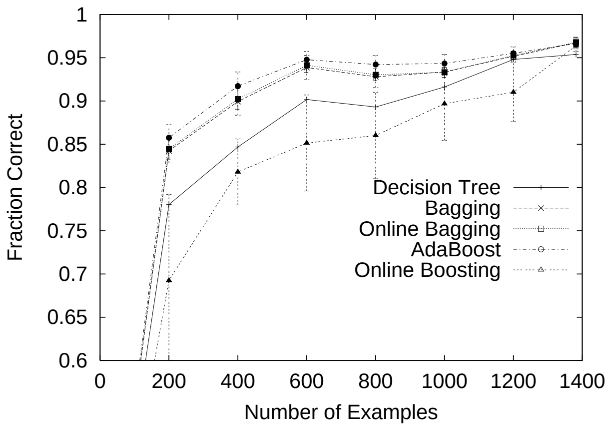

Figure 4: Learning curves for Car-Evaluation dataset

the tenth example when presented to h_3 in each algorithm, and so on. Intuitively, we want online boosting to get a good mix of training examples so that the normalized error of each base model in online boosting quickly converges to what it is in AdaBoost. The more rapidly this convergence occurs, the more similar the weight adjustments will be and the more similar their performances will be.

4 Experimental Results

In this section, we discuss some experiments that demonstrate that our online algorithms perform more like their batch counterparts as the number of training examples increases. We have implemented online bagging and online boosting with decision trees and Naive Bayes classifiers as the base models. For decision trees, we have reimplemented the lossless ITI online algorithm [14]; batch and online Naive Bayes algorithms are essentially identical.

To illustrate the convergence of batch and online learning, we experimented with the Car Evaluation dataset from the UCI Machine Learning Repository [2]. The dataset has 1728 examples, of which we retained 346 (20%) as a test set and used 200, 400, 600, 800, 1000, 1200, and all the remaining 1382 examples as training sets. We ran each algorithm (except decision trees) ten times with each number of training examples to account for the randomness in the ensemble algorithms. The results are shown in Figure 4.

The figure shows batch and online bagging with decision trees performing identically (and always significantly better than a single decision tree). AdaBoost also performs significantly better than a single decision tree for all numbers of examples. Online boosting struggles at first but performs comparably to AdaBoost and significantly better than single decision trees for the maximum number of examples. Note that online boosting's performance steadily becomes closer to that of AdaBoost as the number of examples grows, as one expects from an online algorithm when compared to its batch version.

We tested our algorithms on several UCI datasets [2] with varying sizes and numbers of attributes (see Table 1). The accuracies of our algorithms are given in Table 2 and Table 3 in increasing order of dataset size. Boldface entries represent cases when the ensemble algorithm significantly (t-test, $\alpha = 0.05$) outperformed a single model while italicized entries represent cases when the ensemble algorithm significantly underperformed relative to a single model. The batch algorithm accuracies are averages over ten runs of five-fold cross-validation. We tested our online algorithms with five random orders of each training set generated for the batch algorithms. (Order matters for online boosting, even with a lossless learning algorithm.) We tested bagging and boosting with decision trees only on some of the smaller datasets because the ITI algorithm proved too expensive with larger ones. Even for the very small Promoters dataset, the AdaBoost algorithm ran in around 30 seconds while online boosting needed about 15 hours. This compares to around 1 second for online boosting with Naive Bayes.

With decision trees, online boosting performed significantly worse than AdaBoost on the Promoters dataset, significantly better on Balance, and comparably on the remaining datasets. Bagging and online bagging performed noticeably better than single decision trees on all except the Breast Cancer dataset. With Naive Bayes, bagging and online bagging never performed noticeably better than Naive Bayes, which we expected because of the *stability* of Naive Bayes [3]. Boosting and online boosting performed comparably to each other on all but the relatively small Promoters dataset and their performances relative to a single Naive Bayes classifier consistently improved as the sizes of the datasets grew. On the Balance and Soybean datasets, the boosting algorithms performed significantly worse than Naive Bayes. On the Breast Cancer dataset, AdaBoost performed significantly worse and online boosting performed marginally worse. On the Car Evaluation and Chess datasets, AdaBoost and online boosting performed significantly better than Naive Bayes. On the Nursery dataset, AdaBoost performed significantly better and online boosting performed marginally better.

5 Conclusions

The paper has described online versions of the popular bagging and boosting algorithms and has shown,

Table 1: Sizes of the UCI datasets used in our experiments.

Data Set	Training Set	Test Set	Inputs	Classes
Promoters	86	20	57	2
Balance	500	125	4	3
Soybean-Large	307	376	35	19
WI. Breast Cancer	559	140	9	2
German Credit	800	200	20	2
Car Evaluation	1382	346	6	4
Chess	2556	640	36	2
Mushroom	6499	1625	22	2
Nursery	10368	2592	8	5

Table 2: Results (fraction correct): batch and online algorithms (with Decision Trees) on UCI Datasets

Dataset	Decision Tree	Bagging	Online Bagging	AdaBoost	Online Boosting
Promoters	0.75	0.82	0.845	**0.935**	0.77
Balance	0.792	0.8128	0.8032	0.7408	0.7664
WI Breast Cancer	0.9786	0.9714	0.9714	0.9729	0.9679
Car Evaluation	0.9537	**0.9673**	**0.9679**	**0.9664**	**0.9639**

through experiment, that these online versions typically perform comparably to their batch counterparts. The algorithms have low overhead and are quite suitable for practical applications. Our current empirical work focuses on testing with large, continuously arriving data streams. We have also shown that batch and online bagging are identical for large datasets provided that the base learning algorithm is well-behaved in a certain sense. Theoretical tasks include characterizing more tightly the class of learning algorithms for which convergence between online and offline bagging can be proved and developing an analytical framework for online boosting. We are also investigating the case of lossy online learning and its effect on ensemble performance.

Acknowledgements We would like to thank Leo Breiman, Bin Yu, Michael Jordan, Joe Hellerstein, and Kagan Tumer for useful discussions on this work. Part of this work was done while the first author was at NASA Ames Research Center.

References

[1] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–139, Sep. 1999.

[2] C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases, 1999. (URL: http://www.ics.uci.edu/~mlearn/MLRepository.html).

[3] L. Breiman. Bias, variance and arcing classifiers. Technical Report 460, Department of Statistics, University of California, Berkeley, 1996.

[4] L. Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36:85–103, 1999.

[5] H. Drucker, R. Schapire, and P. Simard. Improving performance in neural networks using a boosting algorithm. In S.J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems-5*, pages 42–49. Morgan Kaufmann, 1993.

[6] Harris Drucker. Boosting using neural networks. In A. J. C. Sharkey, editor, *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, pages 51–77. Springer-Verlag, London, 1999.

[7] Alan Fern and Robert Givan. Online ensemble learning: An empirical study. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 279–286. Morgan Kaufmann, 2000.

[8] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, Bari, Italy, 1996. Morgan Kaufmann.

[9] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[10] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes*. Oxford Science Publications, New York, 1992.

Table 3: Results (fraction correct): batch and online algorithms (with Naive Bayes) on UCI Datasets

Dataset	Naive Bayes	Bagging	Online Bagging	AdaBoost	Online Boosting
Promoters	0.8774	0.8354	0.8401	0.8455	*0.7483*
Balance	0.9072	0.9062	0.9067	*0.8686*	*0.8747*
Soybean-Large	0.7497	0.7487	0.7471	*0.7184*	*0.7315*
WI Breast Cancer	0.9679	0.9698	0.9692	*0.9501*	0.9533
German Credit	0.7410	0.7437	0.7437	0.7318	0.7110
Car Evaluation	0.8569	0.8532	0.8547	**0.9017**	**0.8967**
Chess	0.8757	0.8759	0.8749	**0.9517**	**0.9476**
Mushroom	0.9966	0.9966	0.9966	0.9999	0.9987
Nursery	0.9061	0.9029	0.9027	**0.9163**	0.9118

[11] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[12] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.

[13] Kagan Tumer. *Linear and Order Statistics Combiners for Reliable Pattern Classification*. PhD thesis, The University of Texas, Austin, TX, May 1996.

[14] P.E. Utgoff, N.C. Berkman, and J.A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.