Experimental Comparisons of Online and Batch Versions of Bagging and Boosting

Nikunj C. Oza
oza@cs.berkeley.edu

Stuart Russell
russell@cs.berkeley.edu

Computer Science Division
University of California
Berkeley, CA 94720-1776

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, i.e., they require multiple passes through the training data. In previous work, we presented online bagging and boosting algorithms that only require one pass through the training data and presented experimental results on some relatively small datasets. Through additional experiments on a variety of larger synthetic and real datasets, this paper demonstrates that our online versions perform comparably to their batch counterparts in terms of classification accuracy. We also demonstrate the substantial reduction in running time we obtain with our online algorithms because they require fewer passes through the training data.

1. INTRODUCTION

Traditional supervised learning algorithms generate a single model such as a decision tree or neural network and use it to classify examples.[1] *Ensemble* learning algorithms combine the predictions of multiple *base models*, each of which is learned using a traditional algorithm. *Bagging* [3] and *Boosting* [4] are well-known ensemble learning algorithms that have been shown to be very effective in improving generalization performance compared to the individual base models. Theoretical analysis of boosting's performance supports these results [4].

In previous work [7], we developed *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. In Sections 2 and 3, we describe our online bagging and online boosting algorithms, respectively. Specifically, we describe how we mirror the methods that the batch bagging and boosting algorithms use to generate distinct base models, which are known to help ensemble performance.

In our previous work, we also discussed our theoretical results and some empirical comparisons of the classification accuracies of our online algorithms and the corresponding batch algorithms on some relatively small datasets. In Section 4, we review the previous experiments and further explore the behavior of our online algorithms through experiments with larger datasets—both synthetic and real. Consistent with previous work, we run our online bagging and boosting algorithms with *lossless* online algorithms for decision trees and Naive Bayes classifiers—for a given training set, a lossless online learning algorithm returns a hypothesis identical to that returned by the corresponding batch algorithm. Overall, our online bagging and boosting algorithms perform comparably to their batch counterparts in terms of classification accuracy. We also compare their running times. If the online base model learning algorithm is not significantly slower than the corresponding batch algorithm, then the bagging and online bagging algorithms do not have a large difference in their running time in our tests. On the other hand, our online boosting algorithm runs significantly faster than batch boosting. For example, on our largest dataset, batch boosting ran four times longer than online boosting to achieve comparable classification accuracy.

Sometimes our online boosting algorithm significantly underperforms batch boosting for a small number of training examples. Even when the training set is large, our online algorithm may underperform initially before finally catching up to the batch algorithm. This characteristic is common with online algorithms because they do not have the luxury of viewing the training set as a whole the way batch algorithms do. We experiment with "priming" online boosting by running it in batch mode for some initial subset of the training set and running in online mode for the remainder of the training set. In most of the experiments that we discuss in this paper, priming leads to improved classification performance.

We also compare the base models' error rates on the train-

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

KDD '01 San Francisco, California USA

OnlineBagging($\mathbf{h}, d$)
For each base model h_m, ($m \in \{1, 2, \ldots, M\}$) in $\mathbf{h}$,
Set k according to $Poisson(1)$.
Do k times
$h_m = L_o(h_m, d)$

Figure 1: Online Bagging Algorithm: h is the set of M base models learned so far, d is the latest training example to arrive, and L_o is the online base model learning algorithm.

ing set under batch and online boosting. This is an important comparison because these errors are used to calculate the weights of the training examples supplied to the different base models. They are also used to assign weights to the base models for use when classifying a new example. The closer these errors are, the more similar the training example weights and base model weights are in the two algorithms, leading to more similar classification performances. The base model error rates exhibit similar trends in batch and online boosting, which partly explains the similar classification accuracies we have obtained so far.

Our experiments with synthetic datasets are meant to compare batch and online boosting with base models having small, medium, and large errors. Our three synthetic datasets are of varying difficulty for learning Naive Bayes classifiers. We show that boosting behaves differently on these datasets and that online boosting mirrors these behaviors.

2. ONLINE BAGGING

Given a training dataset of size N, standard batch bagging creates M base models, each trained on a bootstrap sample of size N created by drawing random samples with replacement from the original training set. Each base model's training set contains K copies of each of the original training examples 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: $P(K = k) \sim \frac{exp(-1)}{k!}$. As discussed in [7], 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 (see figure 1). 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. In past work [7], we proved that if the same original training set is supplied to the two bagging algorithms, then the distributions 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. We further proved, for some very simple learning algorithms (K-Nearest Neighbor and contingency-table learning), that the convergence of the distributions over the training sets leads to convergence of the classification performance of online bagging to that of batch bagging. We are working on tightly characterizing the learning algorithms for which we obtain this type of convergence.

Initial conditions: $\lambda_m^{sc} = 0, \lambda_m^{sw} = 0$.
OnlineBoosting($\mathbf{h}, L_o, d$)
Set the example's "weight" $\lambda_d = 1$.
For each base model h_m, ($m \in \{1, 2, \ldots, M\}$) in $\mathbf{h}$,
Set k according to $Poisson(\lambda_d)$.
Do k times
$h_m = L_o(h_m, d)$
If $h_m(d)$ is the correct label,
then
$\lambda_m^{sc} \longleftarrow \lambda_m^{sc} + \lambda_d$
$\epsilon_m \longleftarrow \frac{\lambda_m^{sw}}{\lambda_m^{sc}+\lambda_m^{sw}}$
$\lambda_d \longleftarrow \lambda_d \left(\frac{1}{2(1-\epsilon_m)}\right)$
else
$\lambda_m^{sw} \longleftarrow \lambda_m^{sw} + \lambda_d$
$\epsilon_m \longleftarrow \frac{\lambda_m^{sw}}{\lambda_m^{sc}+\lambda_m^{sw}}$
$\lambda_d \longleftarrow \lambda_d \left(\frac{1}{2\epsilon_m}\right)$

To classify new examples:
Return $h(x) = argmax_{c \in C} \sum_{m:h_m(x)=y} log \frac{1-\epsilon_m}{\epsilon_m}$.

Figure 2: Online Boosting Algorithm: h is the set of M base models learned so far, d is the latest training example to arrive, and L_o is the online base model learning algorithm.

3. ONLINE BOOSTING

Our online boosting algorithm is designed to correspond to the batch boosting algorithm, AdaBoost.M1 [4]. AdaBoost 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 when generating model h_m and the correctly classified examples are given the remaining half of the weight.

Our online boosting algorithm (Figure 2) 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. 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.

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.[2] In online boosting, h_1 is generated from only the first ten examples before being

[2]Recall that we test base model h_m on the training examples in order to adjust their weights before using them to generate base model h_{m+1}.

Table 1: The datasets used in our experiments. For the Soybean and Census Income datasets, we have given the sizes of the supplied training and test sets. For the remaining datasets, we have given the sizes of the training and test sets in our five-fold cross-validation runs.

Data Set	Training Set	Test Set	Inputs	Classes
Promoters	84	22	57	2
Balance	500	125	4	3
Soybean-Large	307	376	35	19
Breast Cancer [5]	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
Connect4	54045	13512	42	3
Synthetic-1	80000	20000	20	2
Synthetic-2	80000	20000	20	2
Synthetic-3	80000	20000	20	2
Census Income	199523	99762	39	2
Forest Covertype	464809	116203	54	7

tested on the tenth example. Clearly, at the moment when the tenth training example is being tested, 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 training example. This may, in turn, lead to different weights for the tenth example when generating h_3 in each algorithm, and so on. Intuitively, we want online boosting to get a good mix of training examples so that the base models and their normalized errors in online boosting quickly converge to what they are in AdaBoost. The more rapidly this convergence occurs, the more similar the training examples' weight adjustments will be and the more similar their performances will be. In the next section, we demonstrate, for some of our larger datasets, that this appears to happen.

4. EXPERIMENTAL RESULTS

In this section, we discuss some experiments that demonstrate the performance of our online algorithms relative to their batch counterparts. For decision trees, we have reimplemented the lossless ITI online algorithm [8] in C++; batch and online Naive Bayes algorithms are essentially identical. We ran these experiments on Dell 6350 computers having 600MHz Pentium III processors and 2GB of memory.

4.1 The Data

We tested our algorithms on several UCI datasets [2], two datasets (Census Income and Forest Covertype) from the UCI KDD archive [1], and three synthetic datasets. We give their sizes and numbers of attributes and classes in Table 1.

All three of our synthetic datasets have two classes. The prior probability of each class is 0.5, and every attribute except the last one is conditionally dependent upon the class and the next attribute. We set up the attributes this way because the Naive Bayes model only represents the probabilities of each attribute given the class, and we wanted data that is not realizable by a single Naive Bayes classifier so that boosting is more likely to yield improvement. The probabilities of each attribute except the last one (A_a for $a \in \{1, 2, \ldots, 19\}$) are as shown in Table 2.

Table 2: $P(A_a = 0|A_{a+1}, C)$ for $a \in \{1, 2, \ldots, 19\}$ in Synthetic Datasets

$P(A_a = 0)$	$A_{a+1} = 0$	$A_{a+1} = 1$
$C = 0$	0.8	0.2
$C = 1$	0.9	0.1

The only difference between the three synthetic datasets is $P(A_{20}|C)$. In Synthetic-1, $P(A_{20} = 0|C = 0) = 0.495$ and $P(A_{20} = 0|C = 1) = 0.505$. In Synthetic-2, these probabilities are 0.1 and 0.8, while in Synthetic-3, these are 0.01 and 0.975, respectively.

4.2 General Results

Figures 3 and 4 are scatterplots comparing the errors of the batch and online versions of bagging and boosting. The full paper [6] contains a table with all the results. Each point in the figures represents one dataset. To reduce clutter, we do not show error bars in our figures, however we performed significance tests (t-test, $\alpha = 0.05$) and discuss the results later in this paper. The batch algorithm accuracies are averages over ten runs of five-fold cross-validation for a total of 50 runs, except for the Soybean and Census Income datasets where we performed ten runs with the supplied training and test set. 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) for a total of 250 runs (50 runs on the Soybean and Census Income datasets). We tested bagging and boosting with decision trees only on some of the smaller datasets (Promoters, Balance, Breast Cancer, Car Evaluation) because the lossless decision tree algorithm is too expensive with larger datasets in online mode. Bagging and online bagging perform comparably in all our tests. Boosting and online boosting perform comparably on all except the very small Promoters dataset.

The largest dataset for which we ran the bagging and boosting algorithms with decision trees was the Car Evaluation dataset from the UCI Repository. Figure 5 shows the learning curve. Batch and online bagging with decision trees perform almost 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.

Figure 6 shows the learning curves for the Census Income dataset. Batch and online boosting perform comparably to each other and significantly outperform a single model for all numbers of examples. On the other hand, bagging and online bagging do not improve significantly upon a single Naive Bayes classifier. Bagging does not improve upon Naive Bayes on any of the datasets, which we expected because of the *stability* of Naive Bayes [3], i.e., small changes in the dataset do not significantly change each Naive Bayes classifier, so that almost all of the base models tend to vote the same way for a given example. Online bagging always performs comparably to batch bagging in our experiments; therefore, online bagging also does not improve upon Naive Bayes.

4.3 Priming the Online Boosting Algorithm

Figure 7 gives a scatterplot similar to Figure 4 except that the online boosting algorithm trains in batch mode with some initial portion of the training set and online mode with the remainder. In primed mode, batch training was done with the *lesser* of the first 20% of the dataset or the first 10000 training examples. Overall, primed online boosting improves upon the unprimed version. Only in case of the Promoters dataset with Naive Bayes classifiers did priming yield significant improvement over unprimed online boosting. Nevertheless, we did achieve some improvement through priming in all cases except Promoters and Breast Cancer with decision trees, and Soybean, Car Evaluation, and Forest Covertype with Naive Bayes.

As we discussed earlier, in the Car Evaluation dataset's learning curves (Figure 5), online boosting significantly underperforms batch boosting for all but the maximum number of examples. Figure 8 displays the original batch boosting and online boosting learning curves along with primed online boosting with the first 200 training examples learned in batch mode. Primed online boosting with decision trees performs comparably to batch boosting for all numbers of examples, i.e., its performance gets close to batch boosting's performance much quicker.

4.4 Base Model Errors

Figures 9 and 10 show the average errors on the training sets of the consecutive base models in batch and online boosting with Naive Bayes for the second synthetic dataset and Census Income dataset, respectively (see the full paper [6] for more such graphs). As mentioned earlier, the closer these errors are in batch and online boosting, the closer the behavior of these two algorithms. We depict the average errors for the maximum number of base models generated by the batch boosting algorithm. For example, on the Census Income dataset, no run of batch boosting ever generated more than 22 base models. This happens because if the next base model that is generated has error greater than 0.5, then the algorithm stops. Our online boosting algorithm always generates the full set of 100 base models because, during training, we do not know how the base model errors will fluctuate; however, to classify a new example, we only use the first L base models such that model $L+1$ has error greater than 0.5.

The base model errors of online and batch boosting are quite similar for Synthetic-2: the first base model performs quite well in both batch and online boosting. Both algorithms then follow the pattern of having subsequent base models perform worse, which is typical because subsequent base models are presented with previously misclassified examples having higher weight, which makes their learning problems more difficult. In the Census Income dataset, the performances of the base models also follow this general trend, although more loosely.

4.5 Running Times

Figures 11 and 12 contain the average running times of Naive Bayes and the ensemble algorithms with Naive Bayes base models for the Census Income dataset and Forest Covertype dataset, respectively. Both the online and batch ensemble algorithms use a learning algorithm for the Naive Bayes base models that requires one pass through the training set. As the number of training examples increases, we expect the rate of growth of the running time to be less for our online ensemble algorithms than for the batch algorithms. Our online algorithms require only one pass through the training set whereas batch bagging requires one pass per base model (to generate its training set and perform the training) and batch boosting requires two passes per base model (once to generate the Naive Bayes classifier and once to test the newly-generated classifier on the training examples to update their weights). However, for small numbers of training examples, the running time may be greater for online learning because the greater number of passes required through the data structures that represent the base models may outweigh the greater number of passes required through the training set. Also, in case of base models for which online learning takes much more time than batch learning, the total execution time for the online ensemble algorithm would be much greater than for the batch algorithm. Additionally, our online boosting algorithm always generates and updates 100 base models, whereas boosting often generates fewer base models as discussed above.

The running time for online boosting is substantially less than for batch boosting on both Census Income (20 minutes vs. 7.1 hours on the entire dataset) and Forest Covertype (4.3 hours vs. 18.8 hours). Relative to the boosting algorithms, the running times of the bagging algorithms are negligible.

5. CONCLUSIONS

This paper discusses online versions of the popular bagging and boosting algorithms. We have demonstrated that they mostly perform comparably to their batch counterparts in terms of classification accuracy. We experimented with priming our algorithm by running an initial subset of the training set in batch mode and then processing the remaining examples online and achieved improvement by doing so. We also demonstrated the comparable performance of online boosting and batch boosting in more detail by examining the errors of the base models on the training set, which directly affect the weights given to the training examples in the different stages of boosting. We have also shown that, if the online base model learning algorithm has a running time comparable to the corresponding batch algorithm, then the running time of online boosting can be much lower than batch boosting, demonstrating the significant savings obtained by processing the training set just once.

In addition to continuing empirical work with large datasets and different base model learning algorithms, we are working on several theoretical tasks including tightly characterizing the class of learning algorithms for which convergence between online and batch bagging can be proved and developing an analytical framework for online boosting. We are also investigating the case of lossy online base model

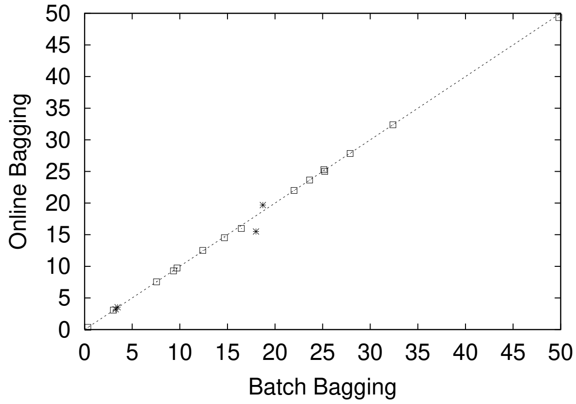

Figure 3: Test Error Rates: Batch Bagging vs. Online Bagging. A star indicates that the two algorithms used decision tree base models while a square indicates Naive Bayes base models.

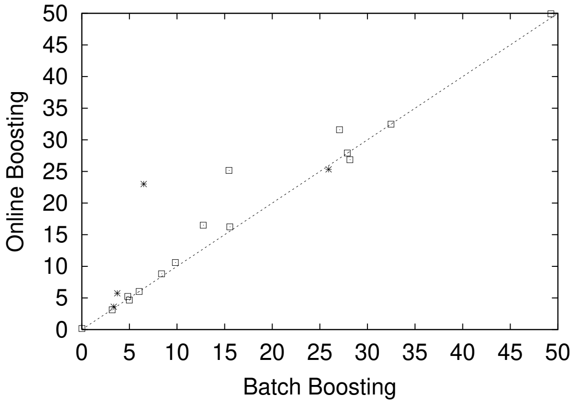

Figure 4: Test Error Rates: Batch Boosting vs. Online Boosting, A star indicates that the two algorithms used decision tree base models while a square indicates Naive Bayes base models.

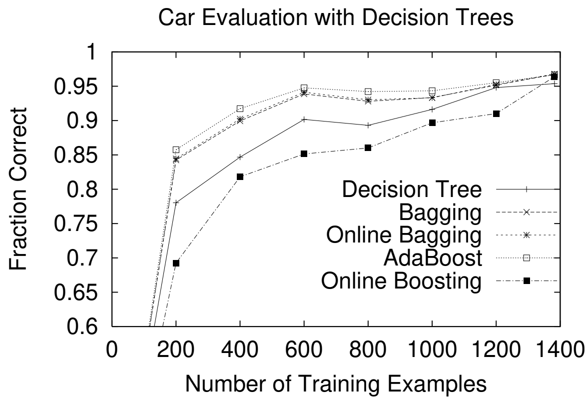

Figure 5: Learning curves for Car-Evaluation dataset.

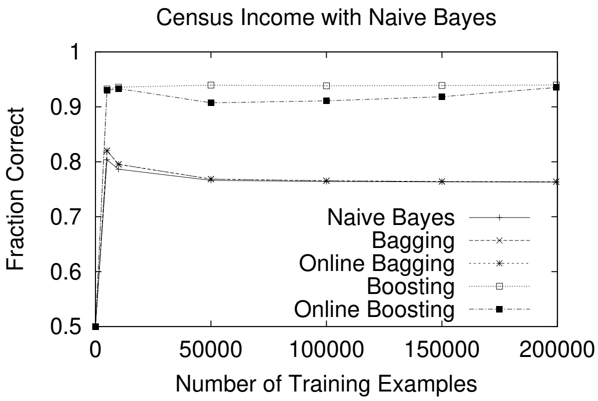

Figure 6: Learning curves for Census Income dataset.

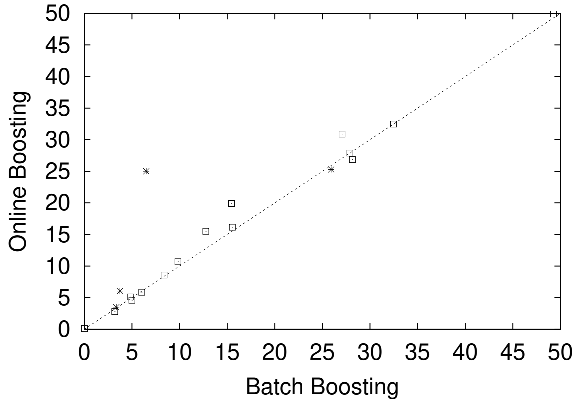

Figure 7: Test Error Rates: Batch Boosting vs. Primed Online Boosting.

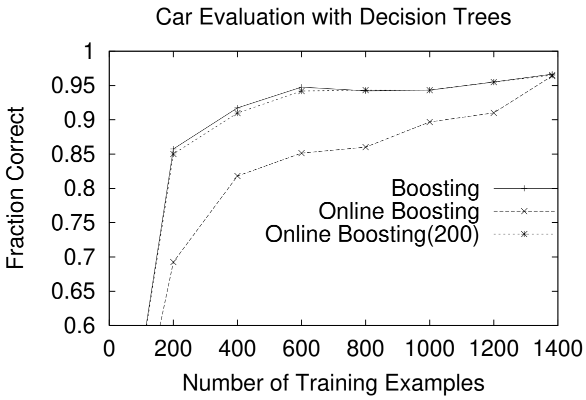

Figure 8: Learning curves for Car Evaluation dataset. Online Boosting(200) is primed online boosting with the first 200 examples learned in batch mode—it performs comparably to batch boosting.

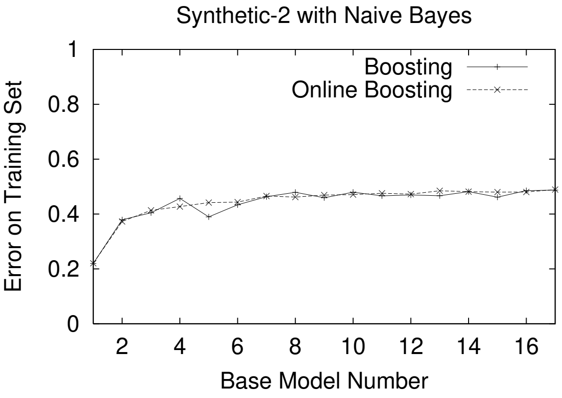

Figure 9: Base Model Errors for Synthetic-2 Dataset.

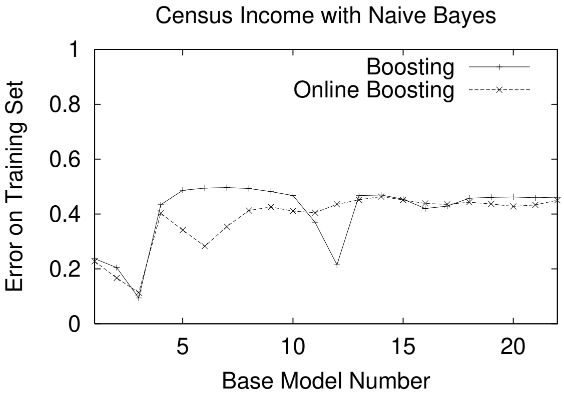

Figure 10: Base Model Errors for Census Income Dataset.

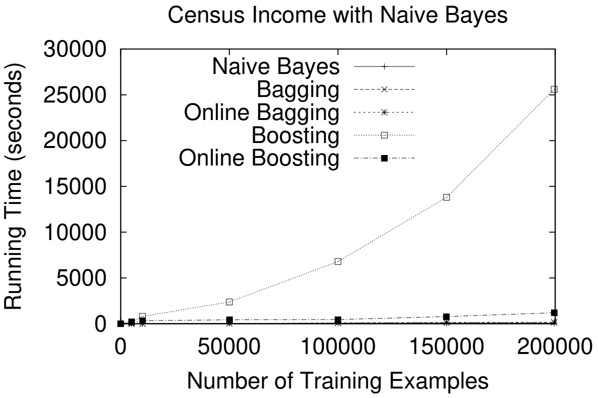

Figure 11: Running Times for Census Income Dataset.

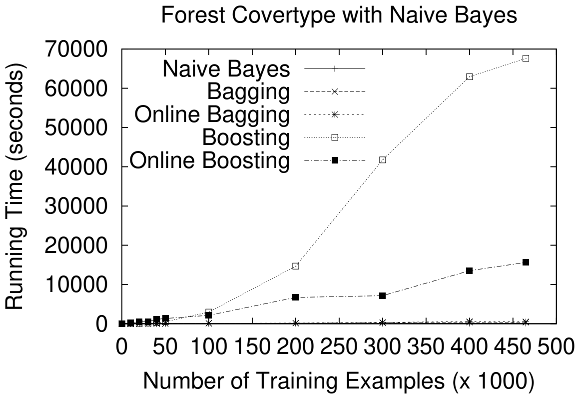

Figure 12: Running Times for Forest Covertype Dataset.

learning and its effect on ensemble performance.

6. ACKNOWLEDGEMENTS

The Wisconsin Breast Cancer dataset was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. The Forest Covertype is Copyrighted 1998 by Jock A. Blackard and Colorado State University.

7. REFERENCES

[1] S.D. Bay. The UCI KDD archive, 1999. (URL: http://kdd.ics.uci.edu).

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

[4] 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.

[5] O. L. Mangasarian, R. Setiono, and W. H. Wolberg. Pattern recognition via linear programming: Theory and application to medical diagnosis. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 22–30. SIAM Publications, 1990.

[6] Nikunj C. Oza and Stuart Russell. Experimental comparisons of online and batch versions of bagging and boosting. Technical report, Electrical Engineering and Computer Science Department, University of California, Berkeley, CA. In preparation.

[7] Nikunj C. Oza and Stuart Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics 2001*, pages 105–112. Morgan Kaufmann, 2001.

[8] 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.