Fast and Light Boosting for Adaptive Mining of Data Streams

Fang Chu and Carlo Zaniolo
fchu, zaniolo@cs.ucla.edu

University of California, Los Angeles, CA 90095, USA

Abstract. Supporting continuous mining queries on data streams requires algorithms that (i) are fast, (ii) make light demands on memory resources, and (iii) are easily to adapt to concept drift. We propose a novel boosting ensemble method that achieves these objectives. The technique is based on a dynamic sample-weight assignment scheme that achieves the accuracy of traditional boosting without requiring multiple passes through the data. The technique assures faster learning and competitive accuracy using simpler base models. The scheme is then extended to handle concept drift via change detection. The change detection approach aims at significant data changes that could cause serious deterioration of the ensemble performance, and replaces the obsolete ensemble with one built from scratch. Experimental results confirm the advantages of our adaptive boosting scheme over previous approaches.

keywords. stream data mining, adaptive boosting ensembles, change detection

1 Introduction

A substantial amount of recent work has focused on continuous mining of data streams [4, 10, 11, 15, 16]. Typical applications include network traffic monitoring, credit card fraud detection and sensor network management systems. Challenges are posed by data ever increasing in amount and in speed, as well as the constantly evolving concepts underlying the data. Two fundamental issues have to be addressed by any continuous mining attempt.

Performance Issue. Constrained by the requirement of on-line response and by limited computation and memory resources, continuous data stream mining should conform to the following criteria: (1) Learning should be done very *fast*, preferably in one pass of the data; (2) Algorithms should make very *light* demands on memory resources, for the storage of either the intermediate results or the final decision models. These *fast and light* requirements exclude high-cost algorithms, such as support vector machines; also decision trees with many nodes should preferably be replaced by those with fewer nodes as base decision models.

Adaptation Issue. For traditional learning tasks, the data is stationary. That is, the underlying concept that maps the features to class labels is unchanging [17]. In the context of data streams, however, the concept may drift due to gradual or sudden changes of the external environment, such as increases of network traffic or failures in sensors. In fact, mining changes is considered to be one of the core issues of data stream mining [5].

In this paper we focus on continuous learning tasks, and propose a novel *Adaptive Boosting Ensemble* method to solve the above problems. In general, ensemble methods combine the predictions of multiple base models, each learned using a learning algorithm called the base learner [2]. In our method, we propose to use very simple base models, such as decision trees with a few nodes, to achieve fast and light learning. Since simple models are often weak predictive models by themselves, we exploit boosting technique to improve the ensemble performance. The traditional boosting is modified to handle data streams, retaining the essential idea of dynamic sample-weight assignment yet eliminating the requirement of multiple passes through the data. This is then extended to handle concept drift via change detection. Change detection aims at significant changes that would cause serious deterioration of the ensemble performance. The awareness of changes makes it possible to build an active learning system that adapts to changes promptly.

Related Work. Ensemble methods are hardly the only approach used for continuous learning. Domingos et al. [4] devised a novel decision tree algorithm, the Hoeffding tree, that performs asymptotically the same as or better than its batched version. This was extended to CVFDT in an attempt to handle concept drift [11]. But, Hoeffding-tree like algorithms need a large training set in order to reach a fair performance, which makes them unsuitable to situations featuring frequent changes. Domeniconi et al. [3] designed an incremental support vector machine algorithm for continuous learning.

There has been work related to boosting ensembles on data streams. Fern et al. [6] proposed online boosting ensembles, and Oza et al. [12] studied both online bagging and online boosting. Frank et al. [7] used a boosting scheme similar to our boosting scheme. But none of these work took concept drift into consideration.

Previous ensemble methods for drifting data streams have primarily relied on bagging-style techniques [15, 16]. Street et al. [15] gave an ensemble algorithm that builds one classifier per data block independently. Adaptability relies solely on retiring old classifiers one at a time. Wang et al. [16] used a similar ensemble building method. But their algorithm tries to adapt to changes by assigning weights to classifiers proportional to their accuracy on the most recent data block. As these two algorithms are the most related, we call them *Bagging* and *Weighted Bagging*, respectively, for later references in our experimental comparison. [1]

This paper is organized as follows. Our adaptive boosting ensemble method is presented in section 2, followed by a change detection technique in section 3. Section 4 contains experimental design and evaluation results, and we conclude in section 5.

2 Adaptive Boosting Ensembles

We use the boosting ensemble method since this learning procedure provides a number of formal guarantees. Freund and Schapire proved a number of positive results about its generalization performance [13]. More importantly, Friedman et al. showed that boosting is particularly effective when the base models are simple [9]. This is most desirable for *fast and light* ensemble learning on steam data.

In its original form, the boosting algorithm assumes a static training set. Earlier classifiers increase the weights of misclassified samples, so that the later classifiers will

[1] The name "*bagging*" derives from their analogy to traditional bagging ensembles [1].

Algorithm 1 Adaptive boosting ensemble algorithm

Ensure: Maintaining a boosting ensemble E_b with classifiers $\{C_1, \cdots, C_m\}$, $m \leq M$.

1: **while** (1) **do**
2: Given a new block $B_j = \{(x_1, y_1), \cdots, (x_n, y_n)\}$, where $y_i \in \{0, 1\}$,
3: Compute ensemble prediction for sample i: $E_b(x_i) = \text{round}(\frac{1}{m}\sum_{k=1}^{m} C_k(x_i))$,
4: *Change Detection: $E_b \Leftarrow \emptyset$ if a change detected!*
5: **if** ($E_b \neq \emptyset$) **then**
6: Compute error rate of E_b on B_j: $e_j = E[1_{E_b(x_i) \neq y_i}]$,
7: Set new sample weight $w_i = (1 - e_j)/e_j$ if $E_b(x_i) \neq y_i$; $w_i = 1$ otherwise
8: **else**
9: set $w_i = 1$, for all i.
10: **end if**
11: Learn a new classifier C_{m+1} from weighted block B_j with weights $\{w_i\}$,
12: Update E_b: add C_{m+1}, retire C_1 if $m = M$.
13: **end while**

focus on them. A typical boosting ensemble usually contains hundreds of classifiers. However, this lengthy learning procedure does not apply to data streams, where we have limited storage but continuous incoming data. Past data can not stay long before making place for new data. In light of this, our boosting algorithm requires only two passes of the data. At the same time, it is designed to retain the essential idea of boosting—the dynamic sample weights modification.

Algorithm 1 is a summary of our boosting process. As data continuously flows in, it is broken into blocks of equal size. A block B_j is scanned twice. The first pass is to assign sample weights, in a way corresponding to AdaBoost.M1 [8]. That is, if the ensemble error rate is e_j, the weight of a misclassified sample x_i is adjusted to be $w_i = (1 - e_j)/e_j$. The weight of a correctly classified sample is left unchanged. The weights are normalized to be a valid distribution. In the second pass, a classifier is constructed from this weighted training block.

The system keeps only the most recent classifiers, up to *M*. We use a traditional scheme to combine the predictions of these base models, that is, by averaging the probability predictions and selecting the class with the highest probability. Algorithm 1 is for binary classification, but can easily be extended to multi-class problems.

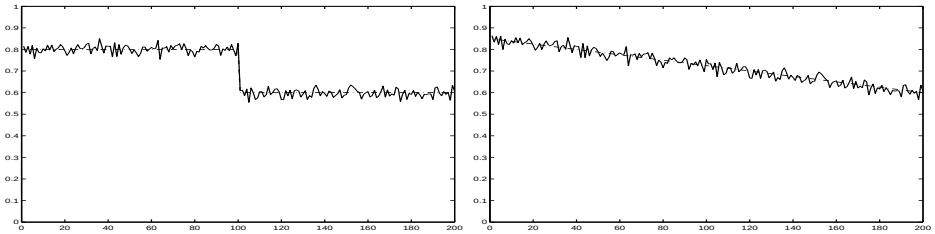
Adaptability Note that there is a step called "*Change Detection*" (line 4) in Algorithm 1. This is a distinguished feature of our boosting ensemble, which guarantees that the ensemble can adapt promptly to changes. Change detection is conducted at every block. The details of how to detect changes are presented in the next section.

Our ensemble scheme achieves adaptability by actively detecting changes and discarding the old ensemble when an alarm of change is raised. No previous learning algorithm has used such a scheme. One argument is that old classifiers can be tuned to the new concept by assigning them different weights. Our hypothesis, which is borne out by experiment, is that obsolete classifiers have bad effects on overall ensemble performance even they are weighed down. Therefore, we propose to learn a new ensemble from scratch when changes occur. Slow learning is not a concern here, as our base

learner is fast and light, and boosting ensures high accuracy. The main challenge is to detect changes with a low false alarm rate.

3 Change Detection

In this section we propose a technique for change detection based on the framework of statistical decision theory. The objective is to detect changes that cause significant deterioration in ensemble performance, while tolerating minor changes due to random noise. Here, we view ensemble performance θ as a random variable. If data is stationary and fairly uniform, the ensemble performance fluctuations are caused only by random noise, hence θ is normally assumed to follow a Gaussian distribution. When data changes, yet most of the obsolete classifiers are kept, the overall ensemble performance will undergo two types of decreases. In case of an abrupt change, the distribution of θ will change from one Gaussian to another. This is shown in Figure 1(a). Another situation is when the underlying concept has constant but small shifts. This will cause the ensemble performance to deteriorate gradually, as shown in Figure 1(b). Our goal is to detect both types of significant changes.



(a) (b)

Fig. 1. Two types of significant changes. Type I: abrupt changes; Type II: gradual changes over a period of time. These are the changes we aim to detect.

Every change detection algorithm is a certain form of *hypothesis test*. To make a decision whether or not a change has occurred is to choose between two competing hypotheses: the *null hypothesis* $\mathcal{H}_0$ or the *alternative hypothesis* $\mathcal{H}_1$, corresponding to a decision of *no-change* or *change*, respectively. Suppose the ensemble has an accuracy θ_j on block j. If the conditional probability density function (*pdf*) of θ under the null hypothesis $p(\theta|\mathcal{H}_0)$ and that under the alternative hypothesis $p(\theta|\mathcal{H}_1)$ are both known, we can make a decision using a *likelihood ratio test*:

$$L(\theta_j) = \frac{p(\theta_j|\mathcal{H}_1)}{p(\theta_j|\mathcal{H}_0)} \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} \tau. \tag{1}$$

The ratio is compared against a threshold τ. $\mathcal{H}_1$ is accepted if $L(\theta_j) \geq \tau$, and rejected otherwise. τ is chosen so as to ensure an upper bound of false alarm rate.

Now consider how to detect a possible type I change. When the null hypothesis $\mathcal{H}_0$ (no change) is true, the conditional *pdf* is assumed to be a Gaussian, given by

$$p(\theta|\mathcal{H}_0) = \frac{1}{\sqrt{2\pi\sigma_0^2}} exp\left\{-\frac{(\theta-\mu_0)^2}{2\sigma_0^2}\right\}, \tag{2}$$

where the mean μ_0 and the variance σ_0^2 can be easily estimated if we just remember a sequence of most recent θ's. But if the alternative hypothesis $\mathcal{H}_1$ is true, it is not possible to estimate $P(\theta|\mathcal{H}_1)$ before sufficient information is collected. This means a long delay before the change could be detected. In order to do it in time fashion, we perform a *significance test* that uses $\mathcal{H}_0$ alone. A significant test is to assess how well the null hypothesis $\mathcal{H}_0$ explains the observed θ. Then the general likelihood ratio test in Equation 1 is reduced to:

$$p(\theta_j|\mathcal{H}_0) \underset{\mathcal{H}_1}{\overset{\mathcal{H}_0}{\gtrless}} \tau. \tag{3}$$

When the likelihood $p(\theta_j|\mathcal{H}_0) \geq \tau$, the null hypothesis is accepted; otherwise it is rejected. Significant tests are effective in capturing large, abrupt changes.

For type II changes, we perform a typical hypothesis test as follows. First, we split the history sequence of θ's into two halves. A Gaussian *pdf* can be estimated from each half, denoted as G_0 and G_1. Then a likelihood ratio test in Equation 1 is conducted.

So far we have described two techniques aiming at two types of changes. They are integrated into a two-stage method as follows. As a first step, a significant test is performed. If no change is detected, then a hypothesis test is performed as a second step. This two-stage detection method is shown to be very effective experimentally.

4 Experimental Evaluation

In this section, we first perform a controlled study on a synthetic data set, then apply the method to a real-life application.

In the synthetic data set, a sample x is a vector of three independent features $< x_i >$, $x_i \in [0, 1], i = 0, 1, 2$. Geometrically, samples are points in a 3-dimension unit cube. The class boundary is a sphere defined as: $B(x) = \sum_{i=0}^{2}(x_i - c_i)^2 - r^2 = 0$, where c is the center of the sphere, r the radius. x is labelled class 1 if $B(x) \leq 0$, class 0 otherwise. This learning task is not easy, because the feature space is continuous and the class boundary is non-linear.

We evaluate our boosting scheme extended with change detection, named as *Adaptive Boosting*, and compare it with *Weighted Bagging* and *Bagging*.

In the following experiments, we use decision trees as our base model, but the boosting technique can, in principle, be used with any other traditional learning model. The standard C4.5 algorithm is modified to generate small decision trees as base models, with the number of terminal nodes ranging from 2 to 32. Full-grown decision trees generated by C4.5 are also used for comparison, marked as *fullsize* in Figure 2-4 and Table 1-2.

4.1 Evaluation of Boosting Scheme

The boosting scheme is first compared against two bagging ensembles on stationary data. Samples are randomly generated in the unit cube. Noise is introduced in the training data by randomly flipping the class labels with a probability of *p*. Each data block has *n* samples and there are 100 blocks in total. The testing data set contains 50k noiseless samples uniformly distributed in the unit cube. An ensemble of *M* classifiers is maintained. It is updated after each block and evaluated on the test data set. Performance is measured using the generalization accuracy averaged over 100 ensembles.

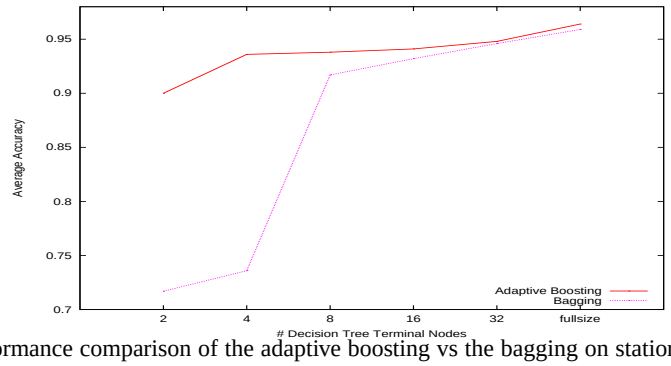


Fig. 2. Performance comparison of the adaptive boosting vs the bagging on stationary data. The weighted bagging is omitted as it performs almost the same as the bagging.

Figure 2 shows the generalization performance when *p=5%*, *n=2*k and *M=30*. Weighted bagging is omitted from the figure because it makes almost the same predictions as bagging, a not surprising result for stationary data. Figure 2 shows that the boosting scheme clearly outperforms bagging. Most importantly, boosting ensembles with very simple trees performs well. In fact, the boosted two-level trees(2 terminal nodes) have a performance comparable to bagging using the full size trees. This supports the theoretical study that boosting improves weak learners.

Higher accuracy of boosted weak learners is also observed for (1) block size n of 500, 1k, 2k and 4k, (2) ensemble size M of 10, 20, 30, 40, 50, and (3) noise level of 5%, 10% and 20%.

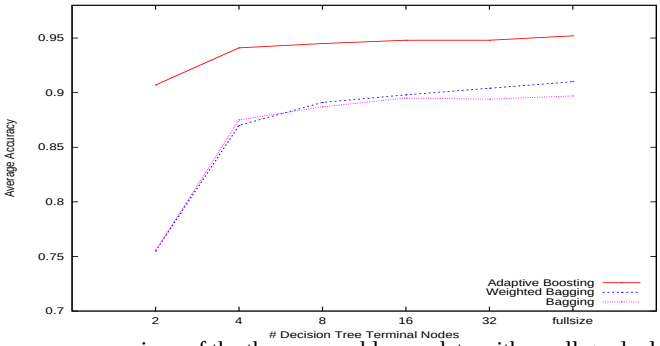


Fig. 3. Performance comparison of the three ensembles on data with small gradual concept shifts.

4.2 Learning with Gradual Shifts

Gradual concept shifts are introduced by moving the center of the class boundary between adjacent blocks. The movement is along each dimension with a step of $\pm\delta$. The value of δ controls the level of shifts from small to moderate, and the sign of δ is randomly assigned. The percentage of positive samples in these blocks ranges from 16% to 25%. Noise level *p* is set to be *5%*, *10%* and *20%* across multiple runs.

The average accuracies are shown in Figure 3 for small shifts ($\delta = 0.01$), and in Figure 4 for moderate shifts ($\delta = 0.03$). Results of other settings are shown in Table 1. These experiments are conducted where the block size is 2k. Similar results are obtained for other block sizes. The results are summarized below:

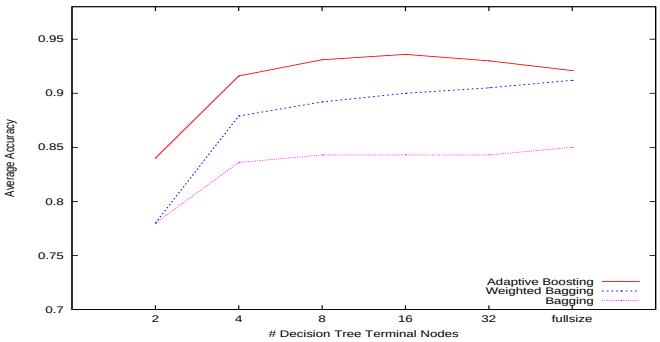


Fig. 4. Performance comparison of the ensembles on data with moderate gradual concept shifts.

	$\delta = .005$				$\delta = .02$			
	2	4	8	fullsize	2	4	8	fullsize
Adaptive Boosting	**89.2%**	**93.2%**	**93.9%**	**94.9**%	**92.2%**	**94.5%**	**95.7%**	**95.8%**
Weighted Bagging	71.8%	84.2%	89.6%	91.8%	83.7%	92.0%	93.2%	94.2%
Bagging	71.8%	84.4%	90.0%	92.5%	83.7%	91.4%	92.4%	90.7%

Table 1. Performance comparison of the ensembles on data with varying levels of concept shifts. Top accuracies shown in bold fonts.

- Adaptive boosting outperforms two bagging methods at all time, demonstrating the benefits of the change detection technique; and
- Boosting is especially effective with simple trees (terminal nodes ≤ 8), achieving a performance compatible with, or even better than, the bagging ensembles with large trees.

4.3 Learning with Abrupt Shifts

We study learning with abrupt shifts with two sets of experiments. Abrupt concept shifts are introduced every 40 blocks; three abrupt shifts occur at block 40, 80 and 120. In one set of experiments, data stays stationary between these blocks. In the other set, small shifts are mixed between adjacent blocks. The concept drift parameters are set to be $\delta_1 = \pm 0.1$ for abrupt shifts , and $\delta_2 = \pm 0.01$ for small shifts.

Figure 5 and Figure 6 show the experiments when base decision trees have no more than 8 terminal nodes. Clearly the bagging ensembles, even with an empirical weighting scheme, are seriously impaired at changing points. Our hypothesis, that obsolete classifiers are detrimental to overall performance even if they are weighed down, are proved experimentally. Adaptive boosting ensemble, on the other hand, is able to respond promptly to abrupt changes by explicit change detection efforts. For base models of different sizes, we show some of the results in Table 2. The accuracy is averaged over 160 blocks for each run.

4.4 Experiments on Real Life Data

In this subsection we further verify our algorithm on a real life data containing 100k credit card transactions. The data has 20 features including the transaction amount, the time of the transaction, etc. The task is to predict fraudulent transactions. Detailed data

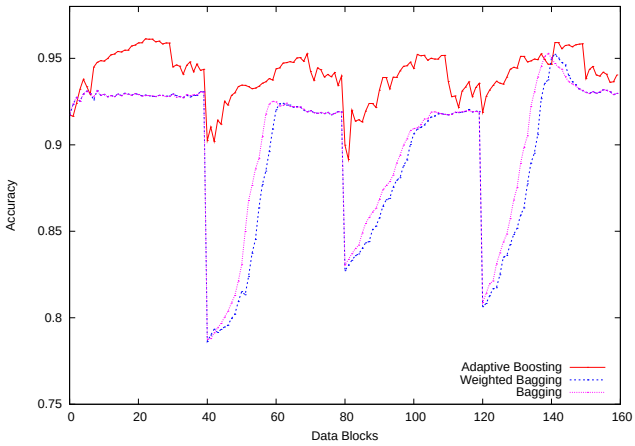


Fig. 5. Performance comparison of the three ensembles on data with abrupt shifts. Base decision trees have no more than 8 terminal nodes.

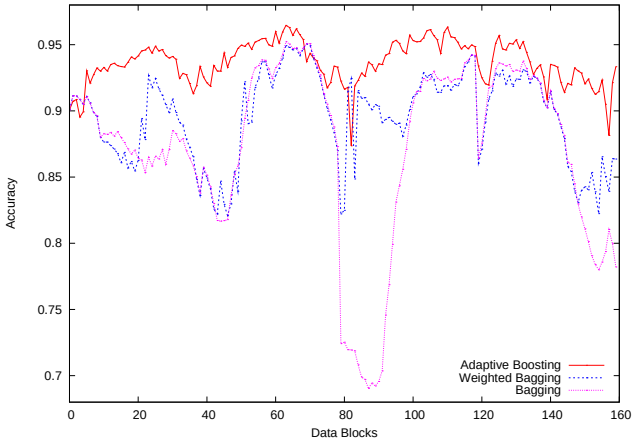


Fig. 6. Performance comparison of the three ensembles on data with both abrupt and small shifts. Base decision trees have no more than 8 terminal nodes.

description is given in [14]. The part of the data we use contains 100k transaction each with a transaction amount between $0 and $21. Concept drift is simulated by sorting transactions by changes by the transaction amount.

We study the ensemble performance using varying block sizes (1k, 2k, 3k and 4k), and different base models (decision trees with terminal nodes no more than 2, 4, 8 and full-size trees). We show one experiment in Figure 7, where the block size is 1k, and the base models have at most 8 terminal nodes. The curve shows three dramatic drops in accuracy for bagging, two for weighted bagging, but only a small one for adaptive boosting. These drops occur when the transaction amount jumps. Overall, the boosting ensemble is much better than the two baggings. This is also true for the other experiments, whose details are omitted here due to space limit.

The boosting scheme is also the fastest. Moreover, the training time is almost not affected by the size of base models. This is due to the fact that the later base models

$\delta_1 = \pm 0.1$	$\delta_2 = 0.00$		$\delta_2 = \pm 0.01$	
	4	fullsize	4	fullsize
Adaptive Boosting	**93.2%**	**95.1%**	**93.1%**	**94.1**%
Weighted Bagging	86.3%	92.5%	86.6%	91.3%
Bagging	86.3%	92.7%	85.0%	88.1%

Table 2. Performance comparison of three ensembles on data with abrupt shifts or mixed shifts. Top accuracies are shown in bold fonts.

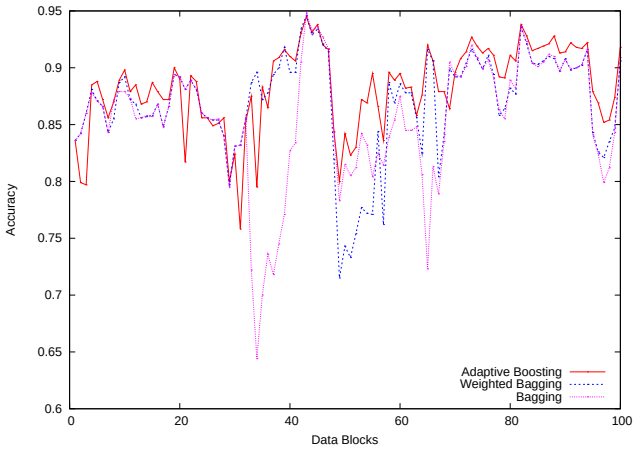


Fig. 7. Performance comparison of the three ensembles on credit card data. Concept shifts are simulated by sorting the transactions by the transaction amount.

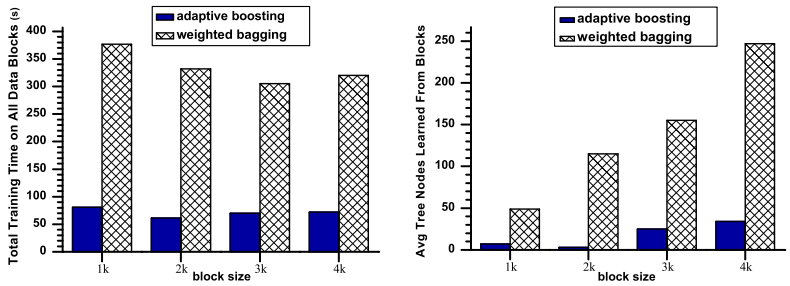
tend to have very simple structures; many of them are just decision stumps (one level decision trees). On the other hand, training time of the bagging methods increases dramatically as the base decision trees grow larger. For example, when the base decision tree is full-grown, the weighted bagging takes 5 times longer to do the training and produces a tree 7 times larger on average. The comparison is conducted on a 2.26MHz Pentium 4 Processor. Details are shown in Figure 8.

To summarize, the real application experiment confirms the advantages of our boosting ensemble methods: it is fast and light, with good adaptability.

5 Summary and Future Work

In this paper, we propose an adaptive boosting ensemble method that is different from previous work in two aspects: (1) We boost very simple base models to build effective ensembles with competitive accuracy; and (2) We propose a change detection technique to actively adapt to changes in the underlying concept. We compare adaptive boosting ensemble methods with two bagging ensemble-based methods through extensive experiments. Results on both synthetic and real-life data set show that our method is much faster, demands less memory, more adaptive and accurate.

The current method can be improved in several aspects. For example, our study of the trend of the underlying concept is limited to the detection of significant changes. If changes can be detected on a finer scale, new classifiers need not be built when changes are trivial, thus training time can be further saved without loss on accuracy. We also plan to study a classifier weighting scheme to improve ensemble accuracy.



(a) (b)

Fig. 8. Comparison of the adaptive boosting and the weighted bagging, in terms of (a) building time, and (b) average decision tree size. In (a), the total amount of data is fixed for different block sizes.

References

1. L. Breiman. Bagging predictors. In *ICML*, 1996.
2. T. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, 2000.
3. C. Domeniconi and D. Gunopulos. Incremental support vector machine construction. In *ICDM*, 2001.
4. P. Domingos and G. Hulten. Mining high-speed data streams. In *ACM SIGKDD*, 2000.
5. Guozhu Dong, Jiawei Han, Laks V.S. Lakshmanan, Jian Pei, Haixun Wang, and Philip S. Yu. Online mining of changes from data streams: Research problems and preliminary results. In *ACM SIGMOD MPDS*, 2003.
6. A. Fern and R. Givan. Online ensemble learning: An empirical study. In *ICML*, 2000.
7. E. Frank, G. Holmes, R. Kirkby, and M. Hall. Racing committees for large datasets. In *Discovery Science*, 2002.
8. Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *ICML*, 1996.
9. J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. In *The Annals of Statistics, 28(2):337–407*, 1998.
10. V. Ganti, J. Gehrke, R. Ramakrishnan, and W. Loh. Mining data streams under block evolution. In *SIGKDD Explorations 3(2):1-10*, 2002.
11. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *ACM SIGKDD*, 2001.
12. N. Oza and S. Russell. Experimental comparisons of online and batch versions of bagging and boosting. In *ACM SIGKDD*, 2001.
13. R. Schapire, Y. Freund, and P. Bartlett. Boosting the margin: A new explanation for the effectiveness of voting methods. In *ICML*, 1997.
14. S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan. Credit card fraud detection using meta-learning: Issues and initial results. In *AAAI-97 Workshop on Fraud Detection and Risk Management*, 1997.
15. W. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *ACM SIGKDD*, 2001.
16. H. Wang, W. Fan, P. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *ACM SIGKDD*, 2003.
17. G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. In *Machine Learning*, 1996.