Tracking concept change with Incremental Boosting by Minimization of the Evolving Exponential Loss

Mihajlo Grbovic and Slobodan Vucetic

Department of Computer and Information Sciences
Temple University, Philadelphia, USA
{mihajlo.grbovic,slobodan.vucetic}@temple.edu

Abstract. Methods involving ensembles of classifiers, such as bagging and boosting, are popular due to the strong theoretical guarantees for their performance and their superior results. Ensemble methods are typically designed by assuming the training data set is static and completely available at training time. As such, they are not suitable for online and incremental learning. In this paper we propose *IBoost*, an extension of *AdaBoost* for incremental learning via optimization of an exponential cost function which changes over time as the training data changes. The resulting algorithm is flexible and allows a user to customize it based on the computational constraints of the particular application. The new algorithm was evaluated on stream learning in presence of concept change. Experimental results showed that *IBoost* achieves better performance than the original *AdaBoost* trained from scratch each time the data set changes, and that it also outperforms previously proposed *Online Coordinate Boost*, *Online Boost* and its non-stationary modifications, *Fast and Light Boosting*, ADWIN *Online Bagging* and *DWM* algorithms.

Keywords: Ensemble Learning, Incremental Learning, Boosting, Concept Change

1 Introduction

There are many practical applications in which the objective is to learn an accurate model using training data set which changes over time. A naive approach is to retrain the model from scratch each time the data set is modified. Unless the new data set is substantially different from the old data set, retraining can be computationally wasteful. It is therefore of high practical interest to develop algorithms which can perform incremental learning. We define incremental learning as the process of updating the existing model when the training data set is changed. Incremental learning is particularly appealing for Online Learning, Active Learning, Outlier Removal and Learning with Concept Change.

There are many single-model algorithms capable of efficient incremental learning, such as linear regression, Naïve Bayes and kernel perceptrons. However, it is still an open challenge how to develop efficient ensemble algorithms for incremental learning. In this paper we consider boosting, an algorithm that trains a weighted ensemble of simple weak classifiers. Boosting is very popular because of its ease of implementation and very good experimental results. However, it requires sequential training of a large

number of classifiers which can be very costly. Rebuilding a whole ensemble upon slight changes in training data can put an overwhelming burden to the computational resources. As a result, there exists a high interest for modifying boosting for incremental learning applications.

In incremental learning with concept change, a typical approach is to use a sliding window and train a model using examples within the window. Upon each window repositioning the data set changes only slightly and it is reasonable to attempt to update the existing model instead of training a new one. Many ensemble algorithms have been proposed for learning with concept change [1–5]. However, in most cases, the algorithms are based on heuristics and applicable to a limited set of problems. Boosting algorithm proposed in [4] uses each new data batch to train an additional classifier and to recalculate the weights for the existing classifiers. These weights are recalculated instead of updated, thus discarding the influence of the previous examples. In [5] a new data batch is weighted depending on the current ensemble error and used to train a new classifier. Instead of classifier weights, probability outputs are used for making ensemble predictions. *OnlineBoost* [6], which uses a heuristic method for updating the example weights, was modified for evolving concepts in [8] and [7]. The *Online Coordinate Boost* (OCB) algorithm proposed in [9] performs online updates of weights of a fixed set of base classifiers trained offline. The closed form weight update procedure is derived by minimizing the approximation on *AdaBoost*'s loss. Because OCB does not have a mechanism for adding and removing base classifiers and one cannot straightforwardly be derived, the algorithm is not suitable for concept change applications.

In this paper, an extension of the popular *AdaBoost* algorithm for incremental learning is proposed and evaluated on concept change applications. It is based on the treatment of *AdaBoost* as the additive model that iteratively optimizes an exponential cost function [10]. Given this, the task of *IBoost* can be stated as updating of the current boosting ensemble to minimize the modified cost function upon change of the training data. The issue of model update consists of updating the existing classifiers and their weights or adding new classifiers using the updated example weights. We intend to experimentally show that *IBoost*, in which the ensemble update always leads towards minimization of the exponential cost, can significantly outperform heuristically based modifications of *AdaBoost* for incremental learning with concept change which do not consider this.

1.1 Preliminaries

The *AdaBoost* algorithm is formulated in [10] as an ensemble of base classifiers trained in a sequence using weighted data set versions. At each iteration, it increases the weights of examples which were misclassified by the previously trained base classifier. Final classifier is defined as a linear combination of all base classifiers.

While *AdaBoost* has been developed using arguments from the statistical learning theory, it has been shown that it can be interpreted as fitting an additive model through an iterative optimization of an exponential cost function.

For a two-class classification setup, let us assume a data set D is available for training, $D = \{(x_i, y_i), i = 1, ..., N\}$, where x_i is a K-dimensional feature vector and

Algorithm 1 *AdaBoost* algorithm

Input: $D = \{(x_i, y_i), i = 1, ..., N\}$, initial data weights $w_i^0 = 1/N$, # iterations M

FOR $m = 0$ **TO** $M - 1$
(a) Fit a classifier $f_{m+1}(x)$ to training data by minimizing

$$J_{m+1} = \sum_{i=1}^{N} w_i^m I(y_i \neq f_{m+1}(x_i)) \tag{1}$$

(b) Evaluate the quantities:

$$\varepsilon_{m+1} = \sum_{i=1}^{N} w_i^m I(y_i \neq f_{m+1}(x_i)) / \sum_{i=1}^{N} w_i^m \tag{2}$$

(c) and then use these to evaluate

$$\alpha_{m+1} = \ln(\frac{1 - \varepsilon_{m+1}}{\varepsilon_{m+1}}) \tag{3}$$

(d) Update the example weights

$$w_i^{m+1} = w_i^m e^{\alpha_{m+1} I(y_i \neq f_{m+1}(x_i))} \tag{4}$$

END
Make Predictions for new point X using:

$$Y = sign(\sum_{n=1}^{M} \alpha_m f_m(X)) \tag{5}$$

$y_i \in \{+1, -1\}$ is its class label. The exponential cost function is defined as

$$E_m = \sum_{i=1}^{N} e^{-y_i \cdot F_m(x_i)}, \tag{6}$$

where $F_m(x)$ is the current additive model defined as a linear combination of m base classifiers produced so far,

$$F_m(x) = \sum_{j=1}^{m} \alpha_j f_j(x), \tag{7}$$

where base classifier $f_j(x)$ can be any classification model with output values $+1$ or -1 and α_j are constant multipliers called the confidence parameters. The ensemble prediction is made as the sign of the weighted committee, $sign(F_m(x))$.

Given the additive model $F_m(x)$ at iteration $m - 1$ the objective is to find an improved one, $F_{m+1}(x) = F_m(x) + \alpha_{m+1} f_{m+1}(x)$, at iteration m. The cost function can be expressed as

$$E_{m+1} = \sum_{i=1}^{N} e^{-y_i \cdot (F_m(x_i) + \alpha_{m+1} f_{m+1}(x_i))} = \sum_{i=1}^{N} w_i^m e^{-y_i \alpha_{m+1} f_{m+1}(x_i)}, \tag{8}$$

where

$$w_i^m = e^{-y_i F_m(x_i)} \tag{9}$$

are called the *example weights*. By rearranging E_{m+1} we can obtain an expression that leads to the familiar *AdaBoost* algorithm,

$$E_{m+1} = (e^{\alpha_{m+1}} - e^{-\alpha_{m+1}}) \sum_{i=1}^{N} w_i^m I(y_i \neq f_{m+1}(x_i)) + e^{-\alpha_{m+1}} \sum_{i=1}^{N} w_i^m. \quad (10)$$

For fixed α_{m+1}, classifier $f_{m+1}(x)$ can be trained by minimizing (10). Since α_{m+1} is fixed, the second term is constant and the multiplication factor in front of the sum in the first term does not affect the location of minimum, the base classifier can be found as $f_{m+1}(x) = \arg\min_{f(x)} J_{m+1}$, where J_{m+1} is defined as the weighted error function (1). Depending on the actual learning algorithm, the classifier can be trained by directly minimizing the cost function (1) (e.g. Naïve Bayes) or by resampling the training data according to the weight distribution (e.g. decision stumps). Once the training of the new base classifier $f_{m+1}(x)$ is finished, α_{m+1} can be determined by minimizing (10) assuming $f_{m+1}(x)$ is fixed. By setting $\partial E_{m+1}/\partial \alpha_{m+1} = 0$ the closed form solution can be derived as (3), where ε_{m+1} is defined as in (2). After we obtain $f_{m+1}(x)$ and α_{m+1}, before continuing to round $m+1$ of the boosting procedure and training of f_{m+2}, the example weights w_i^m have to be updated. By making use of (9), weights for the next iteration can be calculated as (4), where $I(y_i \neq f_{m+1}(x_i))$ is an indicator function which equals 1 if i-th example is misclassified by f_{m+1} and 0 otherwise. Thus, weight w_i^{m+1} depends on the performance of all previous base classifiers on i-th example. The procedure of training an additive model by stage-wise optimization of the exponential function is executed in iterations, each time adding a new base classifier. The resulting learning algorithm is identical to the familiar *AdaBoost* algorithm summarized in Algorithm 1.

Consequences. There is an important aspect of *AdaBoost* relevant to development of its incremental variant. Due to the iterative nature of the algorithm, $\partial E_{m+1}/\partial \alpha_j = 0$ will only hold for the most recent classifier, $j = m+1$, but not necessarily for the previous ones, $j = 1, ..., m$. Thus, *AdaBoost* is not a global optimizer of the confidence parameters α_j [12]. As an alternative to the iterative optimization, one could attempt to globally optimize all a parameters after addition of a base classifier. In spite of being more time consuming, this would lead to better overall performance.

Weak learners (e.g decision stumps) are typically used as base classifiers because of their inability to overfit the weighted data, which could produce a very large or infinite value of α_{m+1}.

As observed in [10], *AdaBoost* guarantees exponential progress towards minimization of the training error (6) with addition of each new weak classifier, as long as they classify the weighted training examples better than random guessing ($\alpha_{m+1} > 0$). The convergence rate is determined in [13]. Note that α_{m+1} can also be negative if f_{m+1} does worse than 50% on the weighted set. In this case $(m+1)$-th classifier automatically changes polarity because it is expected to make more wrong predictions than the correct ones. Alternatively, f_{m+1} can be removed from the ensemble. A common approach is to terminate the boosting procedure when weak learners with positive confidence parameters can no longer be produced.

In the next section, we introduce *IBoost*, an algorithm which naturally extends *AdaBoost* to incremental learning where the cost function changes as the data set changes.

2 Incremental Boosting (IBoost)

Let us assume an *AdaBoost* committee with m base classifiers $F_m(x)$ has been trained on data set $D_{old} = \{(x_i, y_i), i = 1, ..., N\}$ and that we wish to train a committee upon the data set changed to D_{new} by addition of N_{in} examples, $D_{in} = \{(x_i, y_i), i = 1, ..., N_{in}\}$, and removal of N_{out} examples, $D_{out} \subset D$. The new training data set is $D_{new} = D_{old} - D_{out} + D_{in}$.

One option is to discard $F_m(x)$ and train a new ensemble from scratch. Another option, more appealing from the computational perspective, is to reuse the existing ensemble. If the reuse option is considered, it is very important in the design of incremental *AdaBoost* to observe that the cost function changes upon change of the data set. Specifically, it changes from

$$E_m^{old} = \sum_{i \in D_{old}} e^{-y_i \cdot F_m(x_i)} \tag{11}$$

to

$$E_m^{new} = \sum_{i \in D_{new}} e^{-y_i \cdot F_m(x_i)}. \tag{12}$$

There are several choices regarding reuse of the current ensemble $F_m(x)$:

1. update $\alpha_t, t = 1, ..., m$, to better fit the new data set;
2. update base classifiers in $F_m(x)$;
3. update both at, $\alpha_t, t = 1, ..., m$ and base classifiers in $F_m(x)$;
4. add a new base classifier f_{m+1} and its α_{m+1}.

Second and third alternatives are not considered here because they would require potentially costly updates of base classifiers and would also require updates of example weights and confidence parameters of base classifiers. In the remainder of the paper, it will be assumed that trained base classifiers are fixed. It will be allowed, however, to remove the existing classifiers from an ensemble. The first alternative involves updating confidence parameters $\alpha_j, j = 1, ..., m$, in such way that they now minimize (12). This can be achieved in two ways.

Batch Update updates each α_j using the gradient descent algorithm $\alpha_j^{new} = \alpha_j^{old} - \eta \cdot \partial E_m^{new} / \partial \alpha_j^{old}$, where η is the learning rate. The resulting update rule is

$$\alpha_j^{new} = \alpha_j^{old} + \sum_{i \in D_{new}} y_i f_j(x_i) e^{-y_i \sum_{k=1}^{m} \alpha_k^{old} f_k(x_i)}. \tag{13}$$

One update of the m confidence parameters takes $O(N \cdot m)$ time. If the training set changed only slightly, only a few updates should be sufficient for the convergence. The number of batch updates to be performed should be selected depending on the computational constraints.

Stochastic Update is a faster alternative for updating each α_j. It uses stochastic gradient descent instead of the batch version. The update of α_j only using example $(x_i, y_i) \in D_{new}$ is

$$\alpha_j^{new} = \alpha_j^{old} + y_i f_j(x_i) e^{-y_i \sum_{k=1}^{m} \alpha_k^{old} f_k(x_i)}. \tag{14}$$

At the extreme, we can run the stochastic gradient using only the new examples, $(x_i, y_i) \in D_{in}$. This kind of updating is especially appropriate for an aggressive iterative schedule where data are arriving one example at a time at a very fast rate and it is infeasible to perform batch update.

The fourth alternative (adding a new base classifier) is attractive because it allows training a new base classifier on the new data set in a way that optimally utilizes the existing boosting ensemble. Before training f_{m+1} we have to determine example weights.

Weight Calculation. There are three scenarios when there is a need to calculate or update the example weights.

First, if confidence parameters were unchanged since the last iteration, we can keep the weights of the old examples and only calculate the weights of the new ones using

$$w_i^m = e^{\sum_{t=1}^{m} \alpha_t I(y_i \neq f_t(x_i))}, i \in D_{in}. \tag{15}$$

Second, if confidence parameters were updated, then all example weights have to be calculated using (9).

Third, if any base classifier f_j was removed, the example weights can be updated by applying

$$w_i^m = w_i^{m-1} e^{-\alpha_j I(y_i \neq f_j(x_i))}, \tag{16}$$

which is as fast as (4).

Adding Base Classifiers. After updating the example weights, we can proceed to train a new base classifier in the standard boosting fashion. When deciding how exactly to update the ensemble when the data changes, one should consider a tradeoff between the accuracy and computational effort. The first question is whether to train a new classifier and the second whether to update α values, and if the answer is affirmative, which update mode to use and how many update iterations to run.

In applications where data set is being changed very frequently and by a little it can become infeasible to train a new base classifier after each change. In that case one can intentionally wait until enough incoming examples are misclassified by the current model F_m and only then decide to add a new base classifier f_{m+1}. In the meantime, the computational resources can be used to update α parameters.

Removing Base Classifiers. In order to avoid an unbounded growth in number of base classifiers, we propose a strategy that removes a base classifier each time a predetermined budget is exceeded. Similar strategies, in which the oldest [5] or the base model with the poorest performance on the current data [1, 2, 4] is removed, were proposed.

Additionally, in case of data with concept change, a classifier f_j can become outdated and receive negative α_j, as a result of (13) or (14), because it is trained on older examples that were drawn from a different concept.

Following this discussion, our strategy is to remove classifiers if one of the two scenarios occurs: Memory is full and we want to add f_{m+1}, remove the classifier with the lowest α Remove f_j if α_j becomes negative during a updates

If classifier f_j is removed, it is equivalent to setting $\alpha_i = 0$. To account for this change, a parameters of the remaining classifiers are updated using (13) or (14) and the example weights are recalculated using (9). If the time does not permit any modification

of a parameters, influence of the removed classifier on example weights can be canceled using (16).

Convergence. An appealing feature of *IBoost* is that it retains the *AdaBoost* convergence properties. At any given time and for any given set of m base classifiers $f_1, f_2, ..., f_m$, as long as the confidence parameters $\alpha_1, \alpha_2, ..., \alpha_m$ are positive and minimize E_m^{new}, addition of new base classifier f_{m+1} by minimizing (1) and calculation of α_{m+1} using (3) will lead towards minimization of E_m^{new}. This ensures the convergence of *IBoost*.

2.1 IBoost Flowchart

Figure 1 presents a summary of *IBoost* algorithm. We are taking a slightly wider view and point to all the options a practitioner could select, depending on the particular application and computational constraints.

Initial data and example weights are used to train the first base classifier. After the data set is updated, the user always has a choice of just updating the confidence parameters, training a new classifier, or doing both.

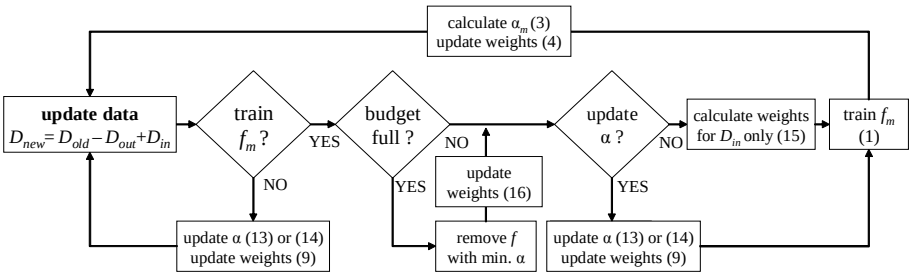

Fig. 1: *IBoost* algorithm flowchart

First, the user decides whether to train a new classifier or not. If the choice is not to, the algorithm just updates the confidence parameters using (13) or (14) and modifies the example weights (9). Otherwise, it proceeds to check if the budget is full and potentially removes the base classifier with minimum a. Next, the user can choose whether to update a parameters. If the choice is to perform the update, a parameters are updated using (13) or (14) which is followed by recalculation of example weights (9). Otherwise, before proceeding to training a new base classifier, the algorithm still has to calculate weights for the new examples D_{in} using (15). Finally, the algorithm proceeds with training f_m by minimizing (1), calculating α_m (3) and updating example weights (4).

IBoost (Fig. 1) was designed to provide large flexibility with respect to budget, training and prediction speed, and stream properties. In this paper, we present an *IBoost* variant for Concept Change. However, using the flowchart we can also easily design variants for Active Learning or Outlier Removal.

2.2 IBoost for Concept Change

Learning under concept change has received a great deal of attention during the last decade, with a number of developed learning strategies (see overview [14]). *IBoost* falls into the category of adaptive ensembles with instance weighting.

When dealing with data streams with concept change it is beneficial to use a sliding window approach. At each step, one example is being added and one is being removed. The common strategy is to remove the oldest one; however, other strategies exist. Selection of window size n presents a tradeoff between achieving maximum accuracy on the current concept and fast recovery from distribution changes.

As we previously discussed, *IBoost* is highly flexible as it can be customized to meet memory and time constrains. For concept change applications we propose the *IBoost* variant summarized in Algorithm 2. In this setup, after each window repositioning the data within the window is used to update the a parameters and potentially train a new base classifier f_{m+1}. The Stochastic version performs b updates of each a using the newest example only, while Batch version performs b iterations of a updates using all the examples in the window. The new classifier is added when the *AddCriterion*: $(k \mod p = 0) \wedge (y_k \neq F_m(x_k))$ is satisfied, where (x_k, y_k) is the new data point, $F_m(x_k)$ is the current ensemble prediction and p is the parameter which controls how often are base models potentially added. This is a common criterion used in ensemble algorithms which perform base model addition and removal [4, 5].

The free parameters (M, n, p and b) are quite intuitive and should be relatively easy to select for a specific application. Larger p values can speed-up the process with slight decrease in performance. As the budget n increases, so does the accuracy at cost of increased cost of prediction, model update and storage requirements. Finally, selection of b is a tradeoff between accuracy, concept change recovery and time.

Algorithm 2 *IBoost* variant for Concept Change applications

Input: Data set $D = \{(x_i, y_i), i = 1, ..., N\}$, window size n, budget M, frequency of model addition p, number of gradient descent updates b

(0) initialize window $D_{new} = \{(x_i, y_i), i = 1, ..., n\}$ and window data weights $w_i^0 = 1/n$
(a) $k = n$, Train f_1 (1) using D_{new}, calculate α_1 (3), update weights w_i^{new} (4), $m = 1$
(b) **Slide the window**: $k = k + 1$, $D_{new} = D_{old} + (x_k, y_k) - (x_{k-n}, y_{k-n})$
(c) **If** $(k \mod p = 0) \wedge (y_k \neq F_m(x_k))$,
  (c.1) If $(m = M)$
    (c.1.1) Remove f_j with minimum α_j, $m = m - 1$
  (c.2) Update $\alpha_j, j = 1, ..., m$ (13) or (14) b times, Recalculate w_i^{new} using (9)
  (c.3) Train f_{m+1} (1), calculate α_{m+1} (3), update weights w_i^{new} (4)
  (c.4) $m = m + 1$
(d) **Else**
  (d.1) Update $\alpha_j, j = 1, ..., m$ (13) or (14) b times, Recalculate w_i^{new} using (9)
(e) **If** any $\alpha_j < 0, j = 1, ..., m$
  (e.1) Remove $f_j, m = m - 1$
  (e.2) Update $\alpha_j, j = 1, ..., m$ (13) or (14) b times, Recalculate w_i^{new} using (9)
(f) **Jump** to (b)

3 Experiments

In this section, *IBoost* performance in four different concept change applications will be evaluated. Three synthetic and one real-world data set, with different drift types (sudden, gradual and rigorous) were used. The data generation and all the experiments were repeated 10 times. The average test set classification accuracy is reported.

3.1 Data Sets

SEA synthetic data [15]. The data consists of three attributes, each one in the range from 0 to 10, and the target variable yi which is set to $+1$ if $x_{i1} + x_{i2} = b$ and -1 otherwise, where $b \in \{7, 8, 9, 9.5\}$. The data stream used has $50,000$ examples. For the first $12,500$ examples, the target concept is with $b = 8$. For the second $12,500$ examples, $b = 9$; the third, $b = 7$; and the fourth, $b = 9.5$. After each window slide, the current ensemble is tested using the current concept $2,500$ test set examples.

Santa Fe time series data (collection A) [16] was used to test *IBoost* performance on a real life gradual concept change problem. The goal is to predict the measurement $g_i \in \mathbb{R}$ based on 9 previous observations The original regression problem with a target value g_i was converted to classification such that $y_i = 1$ if $g_i = b$, where $b \in \{-0.5, 0, 1\}$ and $y_i = -1$ otherwise. The data stream contains $9,990$ examples. For the first $3,330$ examples, $b = -0.5$; for the second $3,330$ examples, $b = 0$; and $b = 1$ for the remaining ones. Testing is done using a holdout data with 825 examples from the current concept. Gradual drifts were simulated by smooth transition of b over $1,000$ examples.

Random RBF synthetic data [3]. This generator can create data which contains a rigorous concept change type. First, a fixed number of centroinds are generated in feature space, each assigned a single class label, weight and standard deviation. The examples are then generated by selecting a center at random, taking weights into account, and displacing them in random direction from the centroid by random displacement length, drown from a Gaussian distribution with centeroids standard deviation. Drift is introduced by moving the centers with constant speed. In order to test *IBoost* on large binary data sets, we generated 10 centers, which are assigned class labels $\{-1, +1\}$ and a drift parameter 0.001, and simulated one million RBF data examples. Evaluation was done using interleaved test-then-train methodology: every example was used for testing the model before it was used for training the model.

LED data. The goal is to predict the digit displayed on a seven segment LED display, where each binary attribute has a 10% chance of being inverted. The original 10-class problem was converted to binary by representing digits $1, 2, 4, 5, 7$ (non-round digits) as $+1$ and digits $3, 6, 8, 9, 0$ (round digits) as -1. Four attributes (out of 7) were selected to have drifts. We simulated one million examples and evaluated the performance using interleaved test-then-train. The data is available in UCI repository.

3.2 Algorithms

IBoost was compared to non-incremental *AdaBoost*, *Online Coordinate Boost*, *OnlineBoost* and its two modifications for concept change (*NSOnlineBoost* and *FLC*), *Fast and Light Boosting*, *DWM* and *AdWin Online Bagging*.

OnlineBoost [6] starts with some initial base models $f_j, j = 1, ..., m$ which are assigned weights $\lambda_j^{sc} = 0$ and $\lambda_j^{sw} = 0$. When a new example (x_i, y_i) arrives it is assigned an initial example weight of $\lambda_d = 1$. Than, *OnlineBoost* uses a Poisson distribution for sampling and updates each f_j model $k = Poisson(\lambda_d)$ times using (x_i, y_i). Next, if $f_j(x_i) = y_i$ the example weight is updated as $\lambda_d = \lambda_d/2(1 - \varepsilon_j)$ and $\lambda_j^{sc} = \lambda_j^{sc} + \lambda_d$; otherwise $\lambda_d = \lambda_d/2\varepsilon_j$ and $\lambda_j^{sw} = \lambda_j^{sw} + \lambda_d$, where $\varepsilon_j = \lambda_j^{sw}/(\lambda_j^{sw} + \lambda_j^{sc})$, before proceeding to updating the next base model f_{j+1}. Confidence parameters α for each base classifier are obtained using (3) and the final predictions are made using (5). Since *OnlineBoost* updates all the base models using each new observation, their performance on the previous examples changes and so should the weighted sums λ_m^{sc} and λ_m^{sw}. Still, the unchanged sums are used to calculate α, and thus the resulting a are not optimized.

NSOnlineBoost [7] In the original *OnlineBoost* algorithm, initial classifiers are incrementally learned using all examples in an online manner. Base classifier addition or removal is not used. This is why poor recovery from concept change is expected. *NSOnlineBoost* modification introduces a sliding window and base classifier addition and removal. The training is conducted in the *OnlineBoost* manner until the update period p_{ns} is reached. Then, the ensemble F_m classification error on the examples in the window is calculated and compared to the ensemble $F_m - f_j$, where m includes all the base models trained using at least $K_{ns} = 100$ points. If removing any f_j improves the ensemble performance on the window, it is removed and a new classifier is added with initial values $\lambda_m^{sc} = 0$, $\lambda_m^{sw} = 0$ and $\varepsilon_m = 0$.

Fast and Light Classifier (*FLC*) [8] is a straightforward extension of *OnlineBoost* that uses an Adaptive Window (*AdWin*) change detection technique [17] to heuristically increase example weights when the change is detected. The base classifiers and their confidence parameters are initialized and updated in the same way as in *OnlineBoost*. When a new example arrives ($\lambda_d = 1$), *AdWin* checks, for every possible split into "large enough" sub-windows, if their average classification rates differ by more than a certain threshold d, set to k window standard deviations. If the change is detected, the new example updates all m base classifiers with weights that are calculated using $\lambda_d = (1 - \varepsilon_j)/\varepsilon_j$, where $\varepsilon_j = \lambda_j^{sw}/(\lambda_j^{sw} + \lambda_j^{sc}), j = 1, ..., m$. The window then drops the older sub-window and continues to grow back to its maximum size with the examples from the new concept. If the change is not detected, the example weights and base classifiers are updated in the same manner as in *OnlineBoost*.

AdWin Bagging [3] is the *OnlineBagging* algorithm proposed in [6] which uses the *AdWin* technique [17] as a change detector and to estimate the error rates for each base model. It starts with initial base models $f_j, j = 1, ..., m$. Then, when a new example (x_i, y_i) arrives, each model f_j is updated $k = Poisson(1)$ times using (x_i, y_i). Final prediction is given by simple majority vote. If the change is detected, the base classifier with the highest error rate ε_j is removed and a new one is added.

Online Coordinate Boost (*OCB*) [9] requires initial base models $f_j, j = 1, ..., m$, trained offline using some initial data. The initial training also provides the starting confidence parameter values $\alpha_j, j = 1, ..., m$, and sums of weights of correctly and incorrectly classified examples for each base classifier, (λ_j^{sc} and λ_j^{sw}, respectively). When a new example (x_i, y_i) arrives, the goal is to find the appropriate updates $\Delta\alpha_j$ for α_j such that the *AdaBoost* loss (6) with the addition of the last example is minimized. Be-

Table 1: Dividing concept change algorithms into overlapping groups

Characteristics	*IBoost*	*Online Boost*	*NSO Boost*	*FLC*	*AdWin Bagg*	*OCB*	*DWM*	*FLB*
Change Detector Used				•	•			•
Online Base Classifier Update		•	•	•	•		•	
Classifier Addition and Removal	•		•		•		•	•
Sliding Window	•		•	•	•			•

cause these updates cannot be found in the closed form, the authors derived closed form updates that minimize the approximate loss instead of the exact one. Such optimization requires keeping and updating the sums of weights ($\lambda^{sc}_{(j,l)}$ and $\lambda^{sw}_{(j,l)}$) which involve two weak hypotheses j and l and introduction of the order parameter o. To avoid numerical errors, the algorithm requires initialization with the data of large enough length n_{ocb} and selection of the proper order parameter.

FLB [5] The algorithm assumes that data are arriving in disjoint blocks of size n_{fb}. Given a new block B_j ensemble example weights are calculated depending on the ensemble error rate ε_j, where the weight of a misclassified example x_i is set to $w_i = (1 - \varepsilon_j)/\varepsilon_j$ and the weight of a correctly classified sample is left unchanged. A new base classifier is trained using the weighted block. The process repeats until a new block arrives. If the number of classifiers reaches the budget M, the oldest one is removed. The base classifier predictions are combined by averaging the probability predictions and selecting the class with the highest probability. There is also a change detection algorithm running in the background, which discards the entire ensemble when a change is detected. It is based on the assumption that the ensemble performance θ on the batch follows Gaussian distribution. The change is detected when the distribution of θ changes from one Gaussian to another, which is detected using a threshold τ.

DWM [1] is a heuristically-based ensemble method for handling concept change. It starts with a single classifier f_1 trained using the initial data and $\alpha_1 = 1$. Then, each time a new example x_k arrives it updates weights a for the existing classifiers: a classifier that incorrectly labels the current example receives a reduction in its weight by multiplicative constant $\beta = 0.5$. After each p_{dwm} examples classifiers whose weights fall under a threshold $\theta_r = 0.01$ are removed and if, in addition, $(y_k \neq F_m(x_k))$ a new classifier f_{m+1} with $\alpha_{m+1} = 1$ is trained using the data in the window. Finally, all classifiers are updated using (x_k, y_k) and their weights a are normalized. The global prediction for the current ensemble $F_m(x_k)$ is always made by the weighted majority (5). When the memory for storing base classifiers is full and a new classifier needs to be stored, the classifier with the lowest α is removed.

In general, the described concept change algorithms can be divided into several groups based on their characteristics (Table 1).

3.3 Results

We performed an in-depth evaluation of *IBoost* and competitor algorithms for different values of window size $n = \{100, 200, 500, 1000, 2000\}$, base classifier budget $M = \{20, 50, 100, 200, 500\}$ and update frequency $p = \{1, 10, 50\}$. We also evaluated the

Table 2: Performance Comparison on SEA dataset

Algorithm		window size $n = 200$					budget size $M = 200$				
		budget size M					window size n				
		20	50	100	200	500	100	200	500	1000	2000
IBoost	test accuracy (%)	94.5	96.4	96.7	**97.1**	97.5	96.9	97.1	97.3	97.5	98
Stochastic	recovery (%)	92.5	93.1	93.3	**93.5**	93.4	93.4	93.5	92.4	90.1	89.6
$b = 5$	time (s)	39	90	183	**372**	751	221	372	396	447	552
IBoost	test accuracy (%)	95.9	97.4	97.8	**97.9**	98	97.2	97.9	98.1	98.3	98.5
Batch	recovery (%)	91.5	92.1	92.9	**92.5**	93.4	92.8	92.5	91.2	88.8	88.4
$b = 5$	time (s)	77	188	401	**898**	2.1K	801	885	1K	1.7K	2.3K
	test accuracy (%)	94.5	95	95	**94.9**	94.9	92.8	94.9	96.7	97	97.5
AdaBoost	recovery (%)	92	92.1	92.2	**91.9**	91.9	91.7	91.9	89.9	88.1	86.3
	time (s)	91	192	432	**913**	2.1K	847	913	1K	1.3K	1.8K
	test accuracy (%)	92.7	93.9	94.3	**94.4**	94.1	91.3	94.4	95.4	95.8	96.8
OCB	recovery (%)	84.3	86.4	89.8	**91.2**	91.2	88.7	91.2	90.1	84.4	93.5
	time (s)	47	120	259	**590**	2K	584	590	567	560	546
	test accuracy (%)	82.6	89.4	92.9	**94.4**	94.9	94.7	94.4	90.5	87.5	83.4
FLB	recovery (%)	82.3	85.3	86.1	**84.7**	84.9	85.2	84.7	83.8	83.5	81.9
	time (s)	73	104	156	**207**	435	183	207	262	390	456

performance of *IBoost* for different values of $b = \{1, 5, 10\}$. Both *IBoost Batch* (13) and *IBoost Stochastic* (14) were considered.

In the first set of experiments *IBoost* was compared to the benchmark *AdaBoost* algorithm, OCB and FLB on the SEA data set. Simple Decision Stumps (single-level Decision Trees) were used as base classifiers. Their number was limited to M. Both *AdaBoost* and *IBoost* start with a single example and when the number of examples reaches n they begin using a window of size n. Each time the window slides and the *AddCriterion* with $p = 1$ is satisfied, *AdaBoost* retrains M classifiers from scratch, while *IBoost* trains a single new classifier (Alg. 2).

OCB was initialized using the first $n_{ocb} = n$ data points and then it updated confidence parameters using the incoming data. Depending on the budget M, the OCB order parameter o was set to the value that resulted in the best performance. Increasing o results in improved performance. However, performance deteriorates if it is increased too much. In FLB, disjoint data batches of size $n_{fb} = n$ were used (equivalent to $p = n_{fb}$). Five base models were trained using each batch while constraining the budget as explained in [5]. Class probability outputs for Decision Stumps were calculated based on the distance from the split and the threshold for the change detector was selected such that the false positive rate is 1%.

Figure 2 compares the algorithms in the $M = 200$, $n = 200$ setting. Performances for different values of M and n are compared in Table 2 based on the test accuracy, concept change recovery (average test accuracy on the first 600 examples after introduction of new concept) and training times.

Both *IBoost* versions achieved much higher classification accuracy than *AdaBoost* and it was faster to train. This can be explained by the fact that *AdaBoost* deletes the

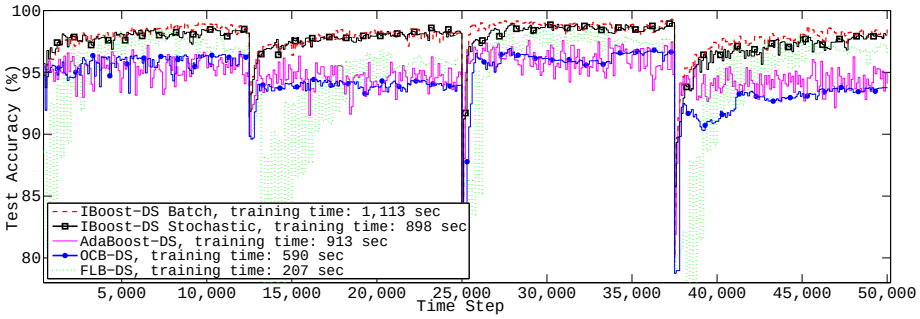

Fig. 2: Predictive Accuracy on SEA Data Set, $M = 200, n = 200$

Table 3: *IBoost* performance for different b and p values on SEA dataset

Algorithm	$M = 200, n = 200$	$p = 1$			$b = 1$		
		$b = 1$	$b = 5$	$b = 10$	$b = 10$	$b = 50$	$b = 100$
IBoost Stochastic	test accuracy (%)	96.7	**97.1**	97.4	96.5	94.7	93.1
	recovery (%)	93.1	**93.5**	93.7	92.8	92.7	92.1
	time (s)	201	**372**	635	104	45	22
IBoost Batch	test accuracy (%)	97.6	**97.9**	98.2	97.1	95.6	93.7
	recovery (%)	92.3	**92.5**	92.9	92.6	91.6	91.4
	time (s)	545	**898**	1.6K	221	133	96

influence of all previously seen examples outside the current window by discarding the whole ensemble and retraining. Better performance of *IBoost* than *OCB* can be explained by the difference in updating confidence parameters and the fact that *OCB* never adds or removes base classifiers. Inferior *FLC* results show that removing the entire ensemble when the change is detected is not the most effective solution.

IBoost Batch was more accurate than *IBoost Stochastic*. However, the training time of *IBoost Batch* was significantly higher. Considering this, *IBoost Stochastic* represents a reasonable tradeoff between performance and time. Fastest recovery for all three concept changes was achieved by *IBoost Stochastic*. This is because the confidence parameters updates of *IBoost Stochastic* are performed using only the most recent example.

Some general conclusions are that the increase in budget M resulted in larger training times and accuracy gain for all algorithms. Also, as the window size n grew, the performance of both *IBoost* and retrained *AdaBoost* improved at cost of increased training time and slower concept change recovery. With a larger window the recovery performance gap between the two approaches increased, while the test accuracy gap reduced as the retrained *AdaBoost* generalization error decreased.

As we discussed in section 3.2, one can select different *IBoost* b and p parameters depending on the stream properties. Table 3 shows how the performance on SEA data changed as they were adjusted. We can conclude that bigger values of b improved the performance at cost of increasing the training time, while bigger values of p degraded the performance (some just slightly, e.g. $p = 10$) coupled with big time savings.

In the second set of experiments, *IBoost Stochastic* $(p = 1, b = 5)$ was compared to the algorithms from Table 1 on both SEA and Santa Fe data sets. Naïve Bayes was

chosen to be the base classifier in these experiments because of its ability to be incrementally improved, which is a prerequisite for some of the competitors (Table 1). All algorithms used a budget of $M = 50$. The algorithms that use a moving window had a window of size $n = 200$. Additional parameters for different algorithms were set as follows: *OCB* was initialized offline with $n_{ocb} = 2K$ data points and used $o = 5$, *FLB* used batches of size $p_{fb} = 200$, *NSOnlineBoost* used $p_{ns} = 10$ and *DWM* $p_{dwm} = 50$.

Results for the SEA data set are presented in Fig. 3. As expected, *OnlineBoost* had poor concept change recovery because it never removes or adds new models. Its two non-stationary versions, NSOnlineBoost and *FLC*, outperformed it. *FLC* was particularly good during the first three concepts where it had almost the same performance as *IBoost*. However, its performance deteriorated after introduction of the fourth concept. The opposite happened for *DWM* which was worse than *IBoost* in all but the fourth concept. We can conclude that *IBoost* outperformed all algorithms, while being very fast (it came in second, after *DWM*).

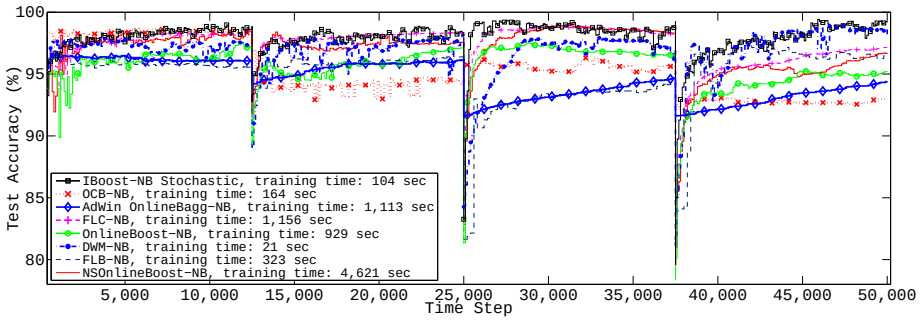

Fig. 3: Predictive Accuracy on SEA Data Set, $M = 50$

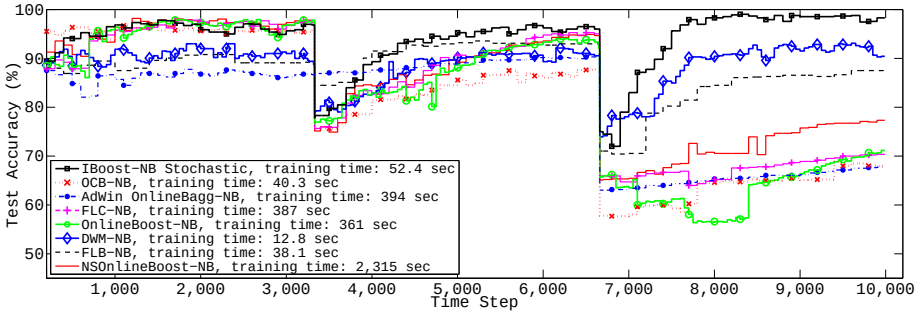

Fig. 4: Predictive Accuracy on Santa Fe Data Set, $M = 50$

Results for the Santa Fe data set are presented in Fig. 4. Similar conclusions as previously can be drawn. In the first concept several algorithms showed almost identical performance. However, when the concept changed *IBoost* was the most accurate. Table 4 summarizes test accuracies on both data sets.

AdWin Online Bagging had an interesting behavior in both SEA and Santa Fe data sets. It did not suffer as large accuracy drop due to concept drift as the other algorithms,

Table 4: SEA and Santa Fe performance summary based on the test accuracy

Data Set	*IBoost Stochastic*	*Online Boost*	*NSO Boost*	*FLC*	*AdWin Bagg*	*OCB*	*DWM*	*FLB*
SEA	**98.0**	95.6	96.9	97.4	94.5	95.2	96.9	94.9
Santa Fe	**94.1**	81.8	85.1	83.4	80.0	80.6	88.8	87.6

and the direction of improvement suggests that it would reach *IBoost* performance if duration of each particular concept were longer.

To study the performance on large problems, we used only *IBoost Stochastic* ($p = 10, b = 1$) version because fast processing of the data was required. Budget was set to $M = 20$ and for algorithms that require window we used $n = 200$. Parameters for remaining algorithms were set as: *OCB* ($n_{ocb} = 2K$ and $o = 5$), *DWM* ($p_{dwm} = 500$).

Figure 5 shows the results for LED data. We can conclude that *AdWin Online Bagging* outperformed *OnlineBoost*, *FLC* and *DWM*, and was very similar to *OCB*. *IBoost Stochastic* was the most accurate algorithm (except in the first $80K$ examples).

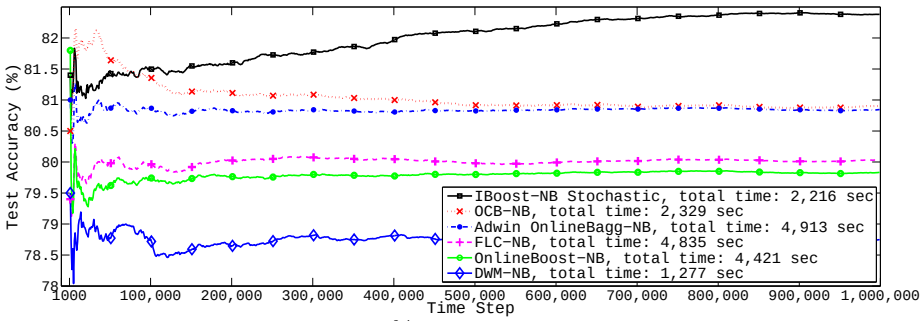

Fig. 5: LED Data Set, 10% noise, 4 drifting attributes, $M = 20$

In Figure 6 we present the results for RBF data. *IBoost Stochastic* was the most accurate model, by a large margin. It was the second fastest, after *DWM*. *AdWin Online Bagging* was reasonably accurate, and it was the second performing overall.

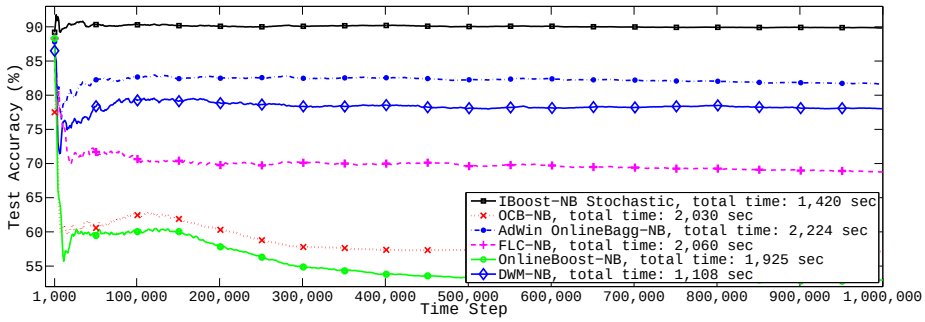

Fig. 6: RBF Data Set, 10 centroids, drift 0.001, $M = 20$

4 Conclusion

In this paper we addressed a very important problem of incremental learning. We proposed an extension of *AdaBoost* to incremental learning. The idea was to reuse and upgrade the existing ensemble when the training data are modified. The new algorithm was evaluated on concept change applications. The results showed that *IBoost* is more efficient, accurate, and resistant than the original *AdaBoost*, mainly because it retains memory about the examples that are removed from the training sliding window. It also performed better than previously proposed *OnlineBoost* and its non-stationary versions, *DWM*, *Online Coordinate Boosting*, *FLB* and *AdWin Online Bagging*. Our future work will include extending *IBoost* to perform multi-class classification, combining it with the powerful *AdWin* change detection technique and experimenting with Hoeffding Trees as base classifiers.

References

[1] Kolter J. Z., Maloof M. A.: Dynamic weighted majority: A new ensemble method for tracking concept drift, In: ICDM, pp. 123–130 (2003)
[2] Scholz M.: Knowledge-Based Sampling for Subgroup Discovery, In: Local Pattern Detection, Springer pp. 171–189 (2005)
[3] Bifet A., Holmes G., Pfahringer B., Kirkby R., Gavald R.: New ensemble methods for evolving data streams, In: ACM SIGKDD (2009)
[4] Wang H., Fan W., Yu P. S., Han J.: Mining concept-drifting data streams using ensemble classifiers, In: Proc. ACM SIGKDD, pp. 226–235 (2003)
[5] Chu F., Zaniolo C.: Fast and light boosting for adaptive mining of data streams, In: Proc. PAKDD, pp. 282–292 (2004)
[6] Oza N., Russell S.: Experimental comparisons of online and batch versions of bagging and boosting, In: ACM SIGKDD, (2001)
[7] Pocock A., Yiapanis P., Singer J., Lujan M., Brown G.: Online Non-Stationary Boosting, In: Intl Workshop on Multiple Classifier Systems (2010)
[8] Attar V., Sinha P., Wankhade K.: A fast and light classifier for data streams, In: Evolving Systems, vol. 1, number 4, pp. 199–207 (2010)
[9] Pelossof R., Jones M., Vovsha I., Rudin C.: Online Coordinate Boosting, In: On-line Learning for Computer Vision Workshop, ICCV (2009)
[10] Friedman J., Hastie T., Tibshirani R.: Additive logistic regression: a statistical view of boosting, In: The Annals of Statistics, vol. 28 pp. 337–407 (2000)
[11] Freund Y., Schapire R. E.: Experiments with a new boosting algorithm, In: Machine Learning: Proceedings of the Thirteenth International Conference, pp. 148–156 (1996)
[12] Schapire R. E., Singer Y.: Improved Boosting Algorithms Using Confidence-rate Predictions, In: Machine Learning Journal, vol. 37 pp. 297–336 (1999)
[13] Schapire R. E.: The convergence rate of adaboost. In: COLT (2010)
[14] Zliobaite I.: Learning under Concept Drift: an Overview, Technical Report, Vilnius University, Faculty of Mathematics and Informatics (2009)
[15] Street W., Kim Y.: A streaming ensemble algorithm (SEA) for large-scale classification, In: ACM SIGKDD, pp. 377–382 (2001)
[16] Weigend A. S., Mangeas M., Srivastava A. N.: Nonlinear gated experts for time series: discovering regimes and avoiding overfitting, In: IJNS, vol. 6, pp. 373–399 (1995)
[17] Bifet A., Gavald R.: Learning from time changing data with adaptive windowing, In: SIAM International Conference on Data Mining, pp. 443–448 (2007)