From Block-based Ensembles to Online Learners In Changing Data Streams: If- and How-To

Dariusz Brzezinski and Jerzy Stefanowski

Institute of Computing Science, Poznan University of Technology,
ul. Piotrowo 2, 60–965 Poznan, Poland
{dariusz.brzezinski,jerzy.stefanowski}@cs.put.poznan.pl

Abstract. Ensemble classifiers have become an established research line in the field of mining time-changing data streams. However, in environments where class labels are available after each example, ensembles which process instances in blocks do not react to sudden changes sufficiently quickly. On the other hand, existing online ensemble algorithms, which process streams incrementally, do not take advantage of periodical weighting mechanisms known from block-based ensembles, which offer accurate reactions to gradual and recurring changes. In this paper, we analyze if and how the characteristics of block and incremental processing can be combined to produce accurate ensemble classifiers. We propose and experimentally evaluate three strategies to transforming a block ensemble into an online learner: the use of a sliding window, an additional incrementally trained ensemble member, and a drift detector. The obtained results verify, which of these approaches is most effective and what characteristics of block processing are most beneficial in online environments.

Keywords: concept drift, data streams, online classifier ensemble

1 Introduction

Progress in information technology as well as the continuous development of sensor and telecommunication systems have paved the way to processing massive datasets in the form of data streams. For the last two decades, several data mining techniques have been proposed to extract interesting knowledge from large data volumes, yet data streams pose a number of unique challenges, which are not easily solved by traditional machine learning methods. The processing of streaming data implies new requirements concerning limited amount of memory, small processing time, and one scan of incoming examples [1, 2]. Moreover, data streams are often subject to concept drifts, i.e., changes in distributions and definitions of target classes over time. Depending on the rate of these changes, concept drifts are usually divided into sudden or gradual ones, both of which require different reaction mechanisms. All these challenges make data streams a complex source which calls for separate data mining techniques.

Out of several machine learning algorithms proposed to tackle time-changing data streams, ensemble methods play an important role in reacting to many types

of concept drift. Essentially, a classifier ensemble is a set of component classifiers whose predictions are combined into a single, more accurate, prediction. In the field of data stream mining, ensemble methods are divided into block-based and online approaches. Block-based approaches are designed to work in environments were examples arrive in portions, called blocks or chunks. Most block ensembles periodically evaluate their components and substitute the weakest ensemble member with a new (candidate) classifier after each chunk of examples. Such an approach ensures accurate reactions to gradual concept drifts. Furthermore, when training their components block-based methods often take advantage of batch algorithms known from static classification.

In contrast to chunk-based approaches, online ensembles are designed to accurately learn in environments were labels are available after each example. With class labels arriving online, algorithms should adapt to changes as quickly as possible. Many researchers tackle this problem by designing new online ensemble methods, ignoring weighting mechanisms known from block-based algorithms. We argue that these weighting mechanisms as well as periodical component evaluations could be still of much value in online ensembles. Our previous work concerning data stream ensembles suggests that by modifying block-based ensembles towards incremental classifiers one can improve classification accuracy on gradual and sudden drifts [3, 4]. This leads us to our research question: Is it profitable to retain periodic evaluations and weighting mechanisms known from block-based algorithms while constructing on-line ensembles for time-changing data streams?

In this paper, we propose to review existing block-based ensemble methods and seek ways of adapting them to suit online environments. Our contributions are as follows:

- In Section 3, we put forward three general strategies to transforming block-based ensembles into online learners. More precisely, we investigate: 1) the use of a windowing technique which updates component weights after each example, 2) the extension of the ensemble by an incremental classifier which is trained between component reweighting, and 3) the use of an online drift detector which allows to shorten drift reaction times.
- In Section 4, we experimentally compare these modifications with popular online ensembles and verify whether block-based algorithms can be successfully transformed into incremental learners.
- In Section 5, we discuss the most important issues in transforming block-based ensembles and draw lines of further research.

2 Related Work

Categorizations of methods for handling concept drift divide stream classifiers into several categories, such as windowing techniques, adaptive algorithms (e.g. decision trees), drift detectors, and active ensembles [5, 2, 6]. In our study, we focus on ensemble methods, which can be further divided into two general groups: *online ensembles*, which learn incrementally after processing single examples, and

block-based ensembles, which process blocks of data. In the following paragraphs, we review existing methods falling into both groups.

Referring to online ensembles, one of the first proposed algorithms was Online Bagging [7], a generalization of batch bagging known from static environments. In Online Bagging, Oza and Russell propose to use incremental learners as component classifiers that combine their decision using a simple majority vote. Sampling, crucial to batch bagging, is performed incrementally by presenting each example to a component k times, where k is defined by the Poisson distribution. More recently, Bifet et al. introduced a modification of Oza's and Russell's algorithm, called Leveraging Bagging [8], which aims at combining the simplicity of Online Bagging with adding more randomization to base classifiers.

Another online ensemble was presented in an algorithm called Dynamic Weighted Majority (DWM) [9]. In DWM a set of incremental classifiers is weighted according to their accuracy after each incoming example. With each mistake made by one of DWM's component classifiers, its weight is decreased by a user specified-factor. Furthermore, after a period of predictions the entire ensemble is evaluated and, if needed, a new classifier is added to the ensemble. Finally, a hybrid online approach was proposed by Nishida with the Adaptive Classifier Ensemble (ACE) [10]. This solution aims at reacting to sudden drifts by tracking the error-rate of a single incremental classifier with each incoming example, similarly to the Drift Detection Method (DDM) proposed by Gama et al. [11]. In contrast to DDM, drift detection in ACE is used to control the validity of an ensemble classifier, which is slowly reconstructed with large chunks of examples.

Out of the presented online ensembles, Online and Leveraging Bagging do not perform periodical component pruning nor reweighting, possibly causing high computational costs and poor reactions to gradual and recurring changes. On the other hand, the DWM algorithm periodically reweights ensemble members but component substitution is conditional and, therefore, for gradually changing streams the forgetting mechanism may not trigger. Finally, ACE does not prune component classifiers and uses a drift detector, both of which possibly lead to poor reactions to gradual changes.

The discussed alternative for online ensembles involves re-evaluating components with fixed-size blocks of incoming instances and replacing the worst component with a classifier trained on the most recent examples. The first of such block-based ensembles was the Streaming Ensemble Algorithm (SEA) [1], which used a heuristic replacement strategy based on accuracy and diversity. Using these two factors, after each block of examples SEA reevaluates a set of decision trees and substitutes the weakest classifier with a new decision tree trained on examples from the most recent chunk. Following a similar scheme, Wang et al. put forward an algorithm called Accuracy Weighted Ensemble (AWE) [12], which also trains a new classifier on each incoming data block by a typical static learning algorithm such as C4.5, RIPPER, or Naive Bayes. Similarly, after a new classifier is trained, all previously learned component classifiers, already present in the ensemble, are evaluated on the most recent chunk. However, in AWE evaluations are done with a special version of the mean square error (MSE),

which estimates the error-rate of the component classifiers (with probability distributions of their class predictions). Such an evaluation method is justified as Wang et al. stated and proved that if component classifiers are weighted by their expected accuracy on the test data, the ensemble achieves greater or equal classification accuracy compared to a single classifier [12]. It is important to notice that the performance of SEA, AWE, and other block-based ensembles largely depends on the size of the processed data chunks. Bigger chunks can lead to more accurate classifiers, but can contain more than one concept drift. On the other hand, smaller chunks are better at separating changes, but usually lead to creating poorer classifiers.

More recently proposed block-based ensemble methods include: Learn^{++}NSE [13] which uses a sophisticated accuracy-based weighting mechanism, the Batch Weighted Ensemble (BWE) [14] which contains a special drift detector analogously to ACE but processes streams in blocks, and the Accuracy Updated Ensemble (AUE) [3] which incrementally trains its component classifiers after every processed block of examples. Although AUE is not the only block ensemble that uses incremental learners as component classifiers, it is unique due to the fact that it updates component classifiers already present in the ensemble. Results obtained by AUE [3, 4] suggest that by incremental learning of periodically weighted ensemble members one could preserve good reactions to gradual changes, while reducing the chunk size problem and, therefore, improving accuracy on abruptly changing streams. In the following section, we propose and analyze three general strategies to transforming block-based ensembles into online learners and verify whether such approaches can produce algorithms that achieve higher accuracy.

3 Strategies to Transforming Block-based Ensembles into Online Learners

Before discussing approaches to converting block ensembles into online ensembles, let us define a generic chunk-based training scheme, which will help describe the proposed strategies.

Algorithm 1 Generic block ensemble training scheme

Input: $\mathcal{S}$: data stream of examples partitioned into blocks of size d, k: number of ensemble members, $Q()$: classifier quality measure
Output: $\mathcal{E}$: ensemble of k weighted classifiers

1: **for all** data blocks $B_i \in \mathcal{S}$ **do**
2: build and weight candidate classifier C_c using B_i and $Q()$;
3: weight all classifiers C_j in ensemble $\mathcal{E}$ using B_i and $Q()$;
4: **if** $|\mathcal{E}| < k$ **then** $\mathcal{E} \leftarrow \mathcal{E} \cup \{C_c\}$;
5: **else if** $\exists j : Q(C_c) > Q(C_j)$ **then** replace weakest ensemble member with C_c;

Let $\mathcal{S}$ be a data stream partitioned into evenly sized blocks $B_1, B_2, \ldots, B_n$, each containing d examples. For every incoming block B_i, the weights of compo-

nent classifiers $C_j \in \mathcal{E}$ are calculated by a classifier quality measure $Q()$, often called a weighting function. The function behind $Q()$ depends on the algorithm being analyzed; e.g., for AWE $Q(C_j) = MSE_r - MSE_i$ [12], while for AUE $Q(C_j) = \frac{1}{(MSE_i + \epsilon)}$ [3]. In addition to component reweighting, a candidate classifier is built from block B_i and added to the ensemble if the ensemble's size is not exceeded. If the ensemble is full but the candidate's quality measure is higher than any member's weight, the candidate classifier substitutes the weakest ensemble member.

The described training scheme, presented in Algorithm 1, can be used to generalize most popular block-based stream ensemble classifiers, such as SEA, AWE, AUE, Learn^{++}.NSE, or BWE. The following subsections present three strategies to modifying this generic algorithm to suit incremental environments.

3.1 Windowing

The first strategy converts a data block into a sliding window. Instead of evaluating component classifiers every d examples, ensemble members are weighted after each example using the last d training instances. This way component weights are incrementally updated and can follow changes in data faster. Because the creation of the candidate classifier is a costly process, especially in block-based ensembles which use batch component classifiers, we propose to add new classifiers to the ensemble every d examples, just as in the original block processing scheme. The described strategy is presented in Algorithm 2.

Algorithm 2 Windowing strategy

Input: $\mathcal{S}$: data stream of examples, k: number of ensemble members, W: window of examples, d: size of window, $Q()$: classifier quality measure, i: example number
Output: $\mathcal{E}$: ensemble of k weighted classifiers

1: **for all** examples $x_i \in \mathcal{S}$ **do**
2: **if** $|W| < d$ **then** $W \leftarrow W \cup \{x_i\}$;
3: **else** replace oldest example in W with x_i;
4: weight all classifiers C_j in ensemble $\mathcal{E}$ using W and $Q()$;
5: **if** $i > 0$ **and** i **mod** $d = 0$ **then**
6: build and weight candidate classifier C_c using W and $Q()$;
7: **if** $|\mathcal{E}| < k$ **then** $\mathcal{E} \leftarrow \mathcal{E} \cup \{C_c\}$;
8: **else if** $\exists j : Q(C_c) > Q(C_j)$ **then** replace weakest ensemble member with C_c;

3.2 Additional Incremental Learner

The second strategy involves using an incremental classifier. The ensemble works exactly like in the original algorithm but an additional online learner, which is trained with each incoming example, is taken into account during component voting. Such a strategy ensures that the most recent data is included in the prediction. Two factors are crucial for the incremental classifier to have an effect on the ensemble's performance: its weight and its accuracy. We propose to use

the maximum of the weights of remaining ensemble members as the candidate's weight. Using such a value ensures that this strategy remains independent of the algorithm being modified and that the incremental learner will have substantial voting power. As for accuracy, to ensure accurate predictions in a time changing environment a classifier should be trained only on the most recent data. On the other hand, using too few examples will make the classifier inaccurate. That is why we propose to initialize the incremental learner with the last full buffer of examples and incrementally train for the next d examples, after which the incremental learner is reinitialized. This strategy is presented in Algorithm 3.

Algorithm 3 Additional incremental learner strategy

Input: $\mathcal{S}$: data stream of examples, C_o online learner, k: number of ensemble members, B: example buffer of size d, $Q()$: classifier quality measure, i: example number
Output: $\mathcal{E}$: ensemble of k weighted classifiers and 1 incremental classifier

for all examples $x_i \in \mathcal{S}$ **do**
  incrementally train C_o with x_i
  $B \leftarrow B \cup \{x_i\}$
  if $i > 0$ **and** i **mod** $d = 0$ **then**
    build and weight candidate classifier C_c using B and $Q()$;
    weight all classifiers C_j in ensemble $\mathcal{E}$ using B and $Q()$;
    if $|\mathcal{E}| < k$ **then** $\mathcal{E} \leftarrow \mathcal{E} \cup \{C_c\}$;
    else if $\exists j : Q(C_c) > Q(C_j)$ **then** replace weakest ensemble member with C_c;
    reinitialize C_o with B;
    $B \leftarrow \emptyset$;

3.3 Drift Detector

The last strategy, presented in Algorithm 4, uses a drift detector attached to an online learner which triggers component reweighting.

Algorithm 4 Drift detector strategy

Input: $\mathcal{S}$: data stream of examples, D: drift detector, k: number of ensemble members, B: example buffer of size d, $Q()$: classifier quality measure, i: example number
Output: $\mathcal{E}$: ensemble of k weighted classifiers and 1 classifier with a drift detector

for all examples $x_i \in \mathcal{S}$ **do**
  incrementally train D with x_i
  $B \leftarrow B \cup \{x_i\}$
  if $|B| = d$ **or** drift detected **then**
    build and weight candidate classifier C_c using B and $Q()$;
    weight all classifiers C_j in ensemble $\mathcal{E}$ using B and $Q()$;
    if $|\mathcal{E}| < k$ **then** $\mathcal{E} \leftarrow \mathcal{E} \cup \{C_c\}$;
    else if $\exists j : Q(C_c) > Q(C_j)$ **then** replace weakest ensemble member with C_c;
    reinitialize D;
    $B \leftarrow \emptyset$;

In periods of stability, when no drifts occur, the algorithm works similarly to the second strategy. If a drift occurs, a candidate classifier is built on a smaller portion of the most recent examples, weighted, and added to the ensemble according to $Q()$. Existing ensemble members are also reweighted after each drift. This approach aims at faster, online, reactions to sudden changes.

4 Experiments

In our experiments we evaluate 4 versions (the original algorithm and the three proposed modifications) of 2 block-based ensembles: the Accuracy Weighted Ensemble (AWE) and the Accuracy Updated Ensemble (AUE). Furthermore, AWE and AUE, along with their modifications, are compared with 3 online ensembles: Online Bagging (Bag), the Dynamic Weighted Majority (DWM), and the Adaptive Classifier Ensemble (ACE). We chose AWE and AUE as representatives of block-based ensembles because periodical component weighting is very important to the performance of these algorithms. Moreover, AWE uses batch component classifiers while AUE has incremental components. Online bagging was chosen as a strong representative of online ensembles, DWM was selected because it periodically evaluates an ensemble, and ACE represents a processing scheme similar to the third proposed modification.

All of the tested algorithms were implemented in Java as part of the MOA framework [15]. We implemented the AWE and AUE algorithms and all their modifications, DWM was implemented and published by Paulo Gonçalves, the code of the Adaptive Classifier Ensemble was provided courtesy of Dr. Nishida and adapted to MOA, while all the remaining classifiers were already a part of MOA. The experiments were performed on a machine equipped with an Intel Pentium Core 2 Duo P9300 @ 2.26 GHz processor and 3.00 GB of RAM. All the tested ensembles used $k = 10$ component classifiers; for AWE and ACE those classifiers were J48 trees with default WEKA parameters, while AUE, Bag, and DWM used Hoeffding trees with adaptive Naive Bayes leaf predictions with a grace period $n_{min} = 100$, split confidence $\delta = 0.01$, and tie-threshold $\tau = 0.05$ [16]. ACE was used with its proprietary drift detector combined with a Naive Bayes classifier [10] while the AWE and AUE modifications which used drift detectors utilized DDM with a Hoeffding tree. The data chunk size used for block ensembles was equal $d = 500$ for all the datasets as this size was considered the minimal suitable for block ensembles [1, 12].

In accordance with the main characteristics of data stream processing [4], we evaluate the performance of the analyzed algorithms with respect to time efficiency, memory usage, and accuracy. All evaluation measures were periodically calculated using the prequential evaluation method [17] with a window of $d = 500$ examples and a fading factor $\alpha = 0.01$.

4.1 Datasets

There is a shortage of publicly available datasets for evaluating data stream classification methods. Most of the common benchmarks for machine learning

algorithms, e.g., gathered in the UCI repository [18], contain too few examples and usually do not contain any type of concept drift. For this reason, data stream classifiers are tested mostly on synthetic datasets, in which concept drift can be introduced. Below, we briefly describe 6 artificial and 2 real datasets that were used in our experiments. All the synthetic datasets were generated using the MOA framework, while real datasets are publicly available.

Hyperplane (`Hyp`) is a popular dataset generator used in many stream classification experiments [12]. We use this generator to create a dataset containing 1,000,000 instances, which is described by 10 features, contains 5 gradual drifts with a magnitude of change $w_i = 0.001$, and has 5% of class noise added to the concepts. The Radial Basis Function generator (`RBF`) creates a user specified number of centroids and assigns each incoming example to one centroid with the probability given by that center's weight. We use this generator to create two datasets: the $\texttt{RBF}_B$ dataset, which contains 4 decision classes and 4 very short sudden drifts (2 blips), which should be ignored by the tested classifier, and the $\texttt{RBF}_G$, which is designed to contain 4 gradual recurring drifts with each concept containing 4 decision classes.

To generate the three remaining artificial datasets we used the SEA generator (`SEA`), Random Tree Generator (`Tree`), and LED generator (`LED`). The `SEA` datasets contains 1,000,000 examples with sudden drifts occurring every 250,000 examples. The `TreeGen` dataset contains only 100,000 instances but is the fastest drifting dataset with 15 sudden recurring drifts, each occurring every 3000 examples. Finally, the `LED` dataset contains 250,000 examples and 20% of noise, but has no drift.

The two real datasets used are the Electricity (`Elec`) and Covertype (`Cov`) datasets. `Elec` is one of the most widely used real datasets in data stream classification and consists of energy prices from the electricity market in the Australian state of New South Wales [19]. The dataset contains 45,312 instances each described by 7 features. Decision class values "up" and "down" indicate the change of the price. The `Cov` dataset contains cover type information about four wilderness areas located in the Roosevelt National Forest of northern Colorado. The dataset consists of 581,012 examples, which are defined by 53 cartographic variables that describe one of 7 possible forest cover types. We used normalized versions of these two datasets available from the MOA website.

The described synthetic datasets were chosen to evaluate the analyzed algorithms in different drift scenarios, such as sudden, gradual, and recurring drifts, blips, and static environments. As for the real datasets, we cannot unequivocally state when drifts occur or if there is any drift; they serve to compare the algorithms in a simple real life scenario rather than a concrete drift situation.

4.2 Results

Tables 1–3 present average prequential accuracy, processing time, and memory usage of the analyzed methods. Algorithms modified using the windowing, incremental candidate, and drift detector strategies are denoted with subscripts: $_W$, $_C$, and $_D$ respectively.

Table 1. Average prequential accuracy [%]

	AWE	AWE_W	AWE_C	AWE_D	AUE	AUE_W	AUE_C	AUE_D	DWM	Bag	ACE
LED	37.76	46.56	43.36	43.10	51.02	51.52	50.73	51.56	42.24	51.50	40.14
Hyp	75.48	84.03	77.90	62.27	88.34	88.77	88.33	88.79	78.07	88.35	79.67
SEA	88.12	88.22	88.17	84.94	89.15	89.33	88.98	89.32	84.14	88.94	85.86
RBF_B	94.80	94.69	95.04	94.52	95.69	96.95	95.19	96.55	87.75	97.85	84.72
RBF_G	93.97	93.72	94.23	94.11	95.74	96.95	95.16	96.69	85.36	97.52	84.13
Tree	65.67	45.71	65.86	57.40	44.04	44.43	44.85	42.06	45.33	49.82	43.82
Cov	87.88	84.50	88.93	49.07	82.79	87.95	84.75	70.30	90.68	88.80	69.54
Elec	75.62	76.81	79.60	58.88	73.12	78.27	80.57	77.15	85.86	87.83	77.32

Table 2. Average time required to process $d = 500$ examples [s]

	AWE	AWE_W	AWE_C	AWE_D	AUE	AUE_W	AUE_C	AUE_D	DWM	Bag	ACE
LED	0.30	8.66	0.32	0.29	0.24	30.21	0.26	0.33	0.08	0.24	0.08
Hyp	0.22	2.60	0.19	0.39	0.38	5.20	0.38	0.65	0.10	1.08	0.25
SEA	0.12	0.77	0.13	0.16	0.37	1.58	0.39	0.51	0.04	1.06	0.05
RBF_B	0.19	2.13	0.23	0.22	1.01	15.86	1.02	1.17	0.18	1.69	0.62
RBF_G	0.18	2.09	0.22	0.22	1.00	16.28	0.93	1.17	0.17	1.54	0.62
Tree	0.19	2.12	0.22	0.18	0.21	9.20	0.20	0.25	0.08	0.25	0.26
Cov	0.21	2.68	0.24	0.17	0.54	22.39	0.57	0.56	0.31	0.43	0.22
Elec	0.12	0.70	0.14	0.10	0.09	2.30	0.10	0.12	0.04	0.13	0.05

Table 3. Average ensemble memory usage [MB]

	AWE	AWE_W	AWE_C	AWE_D	AUE	AUE_W	AUE_C	AUE_D	DWM	Bag	ACE
LED	3.53	3.55	3.76	3.40	0.23	0.36	0.26	0.93	0.03	1.28	0.25
Hyp	1.24	1.25	1.28	3.69	1.31	1.48	1.33	3.85	0.14	5.91	0.12
SEA	0.71	0.73	0.74	1.06	1.76	1.90	1.77	2.33	0.07	6.83	0.10
RBF_B	1.58	1.61	1.63	2.09	4.03	4.28	4.06	5.19	0.32	12.14	0.16
RBF_G	1.62	1.64	1.66	2.40	6.89	8.00	6.92	9.25	0.30	12.19	0.17
Tree	1.86	1.81	1.97	1.67	0.54	1.19	0.56	1.10	0.06	1.13	0.21
Cov	3.12	3.13	3.16	2.14	1.05	1.70	1.10	0.92	0.45	1.13	0.17
Elec	0.91	0.92	0.93	0.54	0.24	0.85	0.26	0.30	0.10	0.46	0.09

Comparing the performance of AWE and its first modification, AWE_W, we can see that apart from the Tree and Cov datasets the windowing technique seems to improve classification accuracy. The improvement, however, comes at the cost of much higher processing time, which is a direct result of testing the classifier with a window of examples to recalculate component weights after each processed instance. The second modification, AWE_C, increases accuracy on all the datasets and does not require so much additional processing time. Finally, the classification accuracy of the AWE_D modification seems to show that a simple addition of a drift detector is not sufficient to improve reactions on sudden

drifts while not deteriorating the ensemble's ability to react to gradual changes. All the modifications have similar memory requirements to AWE, with AWE_D showing higher variance depending on the number of detected drifts. The differences between accuracies of the analyzed algorithms were verified to be statistically significant by performing the Friedman test at $p = 0.050$. Furthermore, by performing a series of Wilcoxon tests it was confirmed that AWE_C increases ($p_C = 0.006$) while AWE_D deteriorates ($p_D = 0.034$) the accuracy of AWE.

Looking at the results of AUE and its modifications we can see trends slightly different than those observed in AWE. The AUE_W modification improves classification accuracy much more than AWE_W but at higher processing costs. As AUE_W updates existing component classifiers it can grow larger component Hoeffding trees, which require more time to test on a window of examples. Thus, the windowing technique is much more time consuming when used to modify AUE than it was on AWE. The additional incremental classifier, present in AUE_C, allows to improve AUE's accuracy on fast changing datasets such as `Tree`, `Cov`, and `Elec`, but does not seem to be so useful on slower changing data. This is probably the effect of using a static (maximum) weight for the incremental candidate; in AWE which uses a linear weighting function it had a different effect than in AUE which uses a non-linear quality measure. Nevertheless, the use of an additional incremental component gives comparable or better accuracy than the original AUE at very small time and memory costs. Finally, the use of a drift detector with AUE proves more rewarding that its addition to AWE. Since, in contrast to AWE, AUE's components can be incrementally updated after a drift is detected, AUE_D could manage to build strong component classifiers while AWE is left with weak learners after each drift. This seems to show that when combined with periodical incremental component updates a drift detector can enhance sudden drift reactions without degrading performance on gradual changes. As Table 1 shows, accuracies of AUE and its modifications are generally higher than AWE's which could also be caused by incremental updating of component classifiers. Concerning classification accuracies of the modifications of AUE, the null hypothesis of the Friedman test can be rejected at $p = 0.050$, while the Wilcoxon test shows that AWE_W significantly increases ($p_W = 0.006$) the accuracy of AUE.

By comparing the discussed results with the performance of the tested online ensembles, we can see that the proposed modifications were more accurate than DWM and ACE on almost all datasets and gave results comparable to Online Bagging. However, Online Bagging is the most expensive of all the compared algorithms in terms of memory and, if we exclude the windowing modification, time. Therefore, the proposed generic modifications could be considered as valid strategies for improving the performance of block ensembles in incremental environments. For AWE the additional incremental component seems to be the best strategy while AUE benefited from using periodical weighting. In both cases, the incremental updating of existing component classifiers seems to be an important performance factor. For this reason, we decided to implement and test modifications of AUE which concentrated even more on updating components.

Table 4. Average performance of online updated versions of AUE. Prequential accuracy presented in percentage [%], time required to process $d = 500$ examples in seconds [s], classifier memory usage in megabytes [MB].

	AUE_W^O			AUE_C^O			AUE_D^O		
	Acc.	Time	Mem.	Acc.	Time	Mem.	Acc.	Time	Mem.
LED	51.54	19.67	0.34	50.17	0.24	0.25	51.43	0.30	0.93
Hyp	88.97	4.80	1.72	88.34	0.25	0.95	89.11	0.63	4.38
SEA	89.35	1.48	2.34	88.53	0.22	1.10	89.25	0.28	1.35
RBF_B	97.40	15.74	4.53	95.17	1.18	4.92	97.11	1.48	6.49
RBF_G	97.09	15.77	7.90	95.18	0.96	6.69	96.89	1.30	9.93
Tree	46.79	9.17	1.32	45.06	0.19	0.51	41.75	0.18	0.53
Cov	89.90	21.85	1.96	85.53	0.63	1.21	83.51	0.51	0.60
Elec	87.37	2.44	1.03	84.72	0.21	0.67	86.00	0.16	0.42

Since AUE already uses incremental component classifiers, we decided to implement additional modifications of AUE in which component classifiers are updated after each example, not after drift detection or d processed instance. Such a modification is not limited to AUE and could be applied to any block ensemble which uses incremental components, such as Learn^{++}.NSE. Table 4 presents the average prequential accuracy, processing time, and memory usage of these online updated versions of AUE. We can see that online updating can further increase classification accuracy, especially on dynamic datasets like Cov and Elec. The best performing online modification is AUE_W^O, which according to the Wilcoxon test achieves accuracy higher than AUE_W with $p = 0.006$. Furthermore, online updating does not pose additional time and memory requirements compared to chunk versions of AUE.

5 Conclusions

We analyzed the problem of integrating weighting mechanisms and periodical component evaluations, known from block ensembles, into online classifiers. To verify the validity of such an approach, we proposed and evaluated three strategies to transforming block-based classifiers into online learners: a windowing technique, the use of an additional incremental learner, and the use of a drift detector. Experimental results demonstrate that all three strategies can be beneficial to the performance of a block ensemble and, therefore, it *is* profitable to retain periodic evaluations and weighting mechanisms while constructing on-line ensembles. Concerning the question *how* to transform a block-based ensemble to suit online environments, differences in the performance of the applied strategies seem to suggest that incremental classifier updates are more advantageous than continuous component reweighting. As future research, we plan to investigate possible combinations of the proposed strategies and tailor them to specific algorithms.

Acknowledgments. The authors are grateful to Dr. Nishida for sharing his implementation of ACE and to Bifet et al. for making the MOA framework available as open source. This work was partly supported by the Polish National Science Center under Grant No. DEC-2011/03/N/ST6/00360.

References

1. Street, W.N., Kim, Y.: A streaming ensemble algorithm (SEA) for large-scale classification. In: Proc. 7th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min., New York, NY, USA, ACM Press (2001) 377–382
2. Kuncheva, L.I.: Classifier ensembles for changing environments. In: Proc. 5th MCS Int. Workshop on Mult. Class. Syst. Volume 3077 of LNCS., Springer (2004) 1–15
3. Brzezinski, D., Stefanowski, J.: Accuracy updated ensemble for data streams with concept drift. In: Proc. 6th HAIS Int. Conf. Hyb. Art. Intell. Syst., Part II. Volume 6679 of LNCS., Springer (2011) 155–163
4. Brzezinski, D.: Mining data streams with concept drift. Master's thesis, Poznan University of Technology, Poznan, Poland (2010)
5. Gama, J.: Knowledge Discovery from Data Streams. Chapman and Hall (2010)
6. Zliobaite, I.: Adaptive training set formation. PhD thesis, Vilnius University (2010)
7. Oza, N.C., Russell, S.J.: Experimental comparisons of online and batch versions of bagging and boosting. In: Proc. 7th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min., New York, NY, USA, ACM Press (2001) 359–364
8. Bifet, A., Holmes, G., Pfahringer, B.: Leveraging bagging for evolving data streams. In: ECML/PKDD (1). (2010) 135–150
9. Kolter, J.Z., Maloof, M.A.: Dynamic weighted majority: An ensemble method for drifting concepts. J. Mach. Learn. Res. **8** (2007) 2755–2790
10. Nishida, K., Yamauchi, K., Omori, T.: ACE: Adaptive classifiers-ensemble system for concept-drifting environments. In: Proc. 6th Int. Workshop Multiple Classifier Systems. Volume 3541 of LNCS., Springer (2005) 176–185
11. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Proc. 17th SBIA Brazilian Symp. Art. Intel. (2004) 286–295
12. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proc. 9th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min., USA, ACM Press (2003) 226–235
13. Elwell, R., Polikar, R.: Incremental learning of concept drift in nonstationary environments. IEEE Trans. Neural Netw. **22**(10) (Oct. 2011) 1517–1531
14. Deckert, M.: Batch weighted ensemble for mining data streams with concept drift. In: ISMIS. Volume 6804 of LNCS., Springer (2011) 290–299
15. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive Online Analysis. J. Mach. Learn. Res. **11** (2010) 1601–1604
16. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proc. 6th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min., USA, ACM Press (2000) 71–80
17. Gama, J., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. In: Proc. 15th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min. (2009) 329–338
18. Frank, A., Asuncion, A.: UCI machine learning repository. `http://archive.ics.uci.edu/ml` (2010)
19. Harries, M.: SPLICE-2 comparative evaluation: Electricity pricing. Technical report, The University of South Wales (1999)