Reacting to Different Types of Concept Drift: The Accuracy Updated Ensemble Algorithm

Dariusz Brzezinski and Jerzy Stefanowski

***Abstract*— Data stream mining has been receiving increased attention due to its presence in a wide range of applications, such as sensor networks, banking, and telecommunication. One of the most important challenges in learning from data streams is reacting to concept drift, i.e., unforeseen changes of the stream's underlying data distribution. Several classification algorithms that cope with concept drift have been put forward, however, most of them specialize in one type of change. In this paper, we propose a new data stream classifier, called the Accuracy Updated Ensemble (AUE2), which aims at reacting equally well to different types of drift. AUE2 combines accuracy-based weighting mechanisms known from block-based ensembles with the incremental nature of Hoeffding Trees. The proposed algorithm is experimentally compared with 11 state-of-the-art stream methods, including single classifiers, block-based and online ensembles, and hybrid approaches in different drift scenarios. Out of all the compared algorithms, AUE2 provided best average classification accuracy while proving to be less memory consuming than other ensemble approaches. Experimental results show that AUE2 can be considered suitable for scenarios, involving many types of drift as well as static environments.**

***Index Terms*— Concept drift, data stream mining, ensemble classifier, nonstationary environments.**

I. Introduction

LEARNING classifiers from data is one of the main tasks in machine learning, data mining, and pattern recognition. Most of the previous and current research in this field is devoted to static environments, where a complete dataset is presented to the learning algorithm. These data are usually electronically stored and, if needed, can be accessed by algorithms several times. Moreover, the target concepts which should be learned are fixed. Over the years, many solutions to this static classification task have been developed and several quite accurate classifiers are now available.

However, in some of the newest applications, learning algorithms work in dynamic environments, where data are continuously generated. Sensor networks, monitoring, traffic management, telecommunication, or web log analysis are examples of such applications [1]. In these dynamic environments, incoming data form a data stream characterized by huge volumes of instances and rapid arrival-rate, which often requires quick, real-time response. Compared to static environments, the processing of data streams implies new requirements for algorithms, such as constraints on memory usage, restricted learning, and testing time, and one scan of incoming instances [2]–[4]. Furthermore, due to the nonstationary nature of data streams, the target concepts tend to change over time in an event called concept drift. Concept drift occurs when the concept about which data are being collected shifts from time to time after a minimum stability period [1]. Such changes are reflected in incoming instances and deteriorate the accuracy of classifiers learned from past training tuples. Examples of real life concept drifts include spam categorization, weather predictions, monitoring systems, financial fraud detection, and evolving customer preferences [5].

Changes of target concepts are categorized into sudden, gradual, or recurring drifts. A good classifier should be able to learn incrementally and adapt to such changes. Standard static classifiers are not capable of fulfilling these conditions, although the issue of incremental learning has already been studied. For example, neural networks or Bayesian classifiers can naturally incorporate incoming examples, while other approaches, such as decision trees, have been adapted to work online (see VFDT [6]). However, simple incremental learning is not sufficient for dealing with concept drifts as forgetting outdated data and quick adaptation to most recent states are a necessity in nonstationary environments [1].

In recent years, several approaches that cope with concept drift have been proposed, including mainly sliding window approaches, new online algorithms, special detection techniques, and adaptive ensembles [1], [7]. Ensembles are popular approaches for improving classification accuracy in static learning problems [8], however, they need to be generalized for changing environments. Such a generalization could concern either modifying the structure of the ensemble (weaker components are replaced by base classifiers trained on the most recent data), updating the aggregation technique (e.g., updating weights in the voting formula), or introducing direct online learning from single incoming examples (e.g., Online Bagging [9]).

In this paper, we focus on the topic of adaptive ensembles that generate component classifiers sequentially from fixed-size blocks of training examples called data chunks. In such ensembles, when a new block arrives, existing component classifiers are evaluated and their combination weights are updated. A new classifier learned from the recent block is added to the ensemble and the weakest classifiers are removed according to the result of the evaluation. Moreover, standard, static learning algorithms, such as C4.5, are applied to generate

Manuscript received May 10, 2012; revised December 18, 2012; accepted February 27, 2013. Date of publication April 4, 2013; date of current version December 13, 2013. This work was supported by the Polish National Science Center under Grant DEC-2011/03/N/ST6/00360.

The authors are with the Institute of Computing Science, Poznan University of Technology, Poznan 60-965, Poland (e-mail: dbrzezinski@cs.put.poznan.pl; jstefanowski@cs.put.poznan.pl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2013.2251352

2162-237X © 2013 IEEE

classifiers from a given block. The SEA algorithm [3] was the first of such adaptive ensembles and was soon followed by the Accuracy Weighted Ensemble [10], which is currently the most representative method of this type.

However, depending on the occurrence of concept drifts within the fixed-size data chunk, the mentioned block-based ensembles may not react sufficiently to changes. In particular, for sudden drifts, they may react too slowly as classifiers generated from outdated blocks still remain valid components even though they have inaccurate weights. This situation is connected with the problem of proper tuning of the data block size. Using small size chunks can partly help in reacting to sudden changes, but doing so will damage the performance of the ensemble in periods of stability and increase computational costs. An unsatisfactory reaction of block-based ensembles to other types of drifts has already been noticed in [11], [12]. On the other hand, online incremental ensembles, such as online or Leveraging Bagging, react faster to sudden drifts but do not take advantage of periodical weighting mechanisms, which could offer sufficient reactions to gradual changes. Furthermore, incremental ensembles are also often characterized by higher computational costs than block-based methods. Such observations suggest that it could be profitable to combine characteristic features from both groups of approaches in order to sufficiently adapt to both gradual and sudden changes.

Following these critical motivations, we have decided to propose a new hybrid algorithm, called Accuracy Updated Ensemble, which should react to different types of concept drift much better than related adaptive ensembles. Our goal is to retain the simple schema of learning component classifiers and weighting their predictions, characteristic for block-based algorithms, while adding elements known from online methods. Our main novel contribution is the introduction of incremental updating of component classifiers, which should improve the ensemble's reactions to different types of concept drift as well as reduce the impact of the chunk size. Online updates allow all ensemble members to adapt to the most recent concept simultaneously and, therefore, change the basic idea behind existing block-based algorithms. We have already considered a preliminary version of the Accuracy Updated Ensemble (AUE1) [12]. However, now we significantly extend its concept and that is why in this paper, we shall refer to the proposed algorithm as AUE2. Compared to AUE1, we put forward a new weighting and updating mechanism as well as modify many other construction details to reduce computational costs and improve classification accuracy. As a result, the analysis of different possible weighting and updating schemes has led to interesting findings concerning incremental training of adaptive ensembles.

We experimentally evaluate the proposed AUE2 algorithm with respect to classification accuracy, processing time, and memory costs. Furthermore, we extend our experiments to a wider comparative study, including 11 additional state-of-the-art methods, all implemented and tested through the MOA framework [13]. To capture the differences in performance of the algorithms, we collect several real world and synthetic datasets representing gradual, sudden, and recurrent concept drifts occurring at different speeds. To the best of our knowledge, no previous research has included such a comprehensive experimental study comparing so many types of online and block-based classifiers in such diverse scenarios.

The remainder of this paper is organized as follows. Section II presents related work. In Section III, we explain the basic intuition behind our approach and present its details. The algorithm is later analyzed and experimentally evaluated on real and synthetic datasets in Section IV. Finally, in Section V, we draw conclusions and discuss lines of future research.

II. Related Work

A. Basic Concepts and Notation

In static classification problems, a set of learning examples contains pairs $\{\mathbf{x}, y\}$, where $\mathbf{x}$ is a vector of attribute values and y is a class label ($y \in \{K_1, \ldots, K_l\}$). Classes K_j are known a priori. The learning algorithm constructs a classifier, which outputs a class prediction for a given example.

In incremental learning, examples arrive continuously in the form of a data stream $\mathcal{S}$. A learning algorithm is presented with a sequence of labeled examples $s_t = \{\mathbf{x}_t, y_t\}$ for $t = 1, 2, \ldots, T$. At each time step t, a learner can analyze historical labeled training examples $(s_1, s_2, \ldots, s_t)$ and an incoming instance s_{t+1}, which is treated as a testing example. The classifier predicts its class label $\hat{y}_{t+1}$. It is assumed that after some time, the true class label y_{t+1} is provided.[1] Having both y_{t+1} and $\hat{y}_{t+1}$, the learning algorithm can update its hypothesis about a classifier if necessary. Then, example s_{t+1} with its class y_{t+1} becomes a part of the training data and the process is repeated when the next instance is observed.

Generally, data streams can be processed either incrementally by single examples s_t (as described above) or they are divided into equally sized blocks (data chunks) $B_1, B_2, \ldots, B_n$ and the evaluation or updating of classifiers is performed after processing all examples from a block. That second perspective will be further discussed in our paper.

Each training example is generated by a source S_j with a stationary distribution P_j. If all the data in the stream are generated by the same distribution, we say that the concepts represented in incoming data are stable, otherwise, concept drift occurs [1]. A sudden (abrupt) drift occurs when at a moment in time t the source distribution in S_t is suddenly replaced by a different distribution in S_{t+1}. Abrupt drifts directly deteriorate the classification abilities of a classifier, as a once generated classifier has been trained on a different class distribution. Gradual drifts are not so radical and they are connected with a slower rate of changes. The first type of gradual drift refers to a transition phase where examples from two different distributions P_j and P_{j+1} are mixed. As time goes on, the probability of observing examples from P_j decreases, while that of examples from P_{j+1} increases. Another type of gradual drift, which we will refer to as incremental, includes more than two sources, however the difference between them is small and the change is noticed in a longer period of time [5], [16].

[1] In this paper, we do not consider learning in a semi-supervised framework where labels are not available for all incoming examples; see, e.g., [14], [15].

Yet another type of drift concerns recurrent concepts, i.e., previously active concepts that may reappear after some time. Moreover, some authors distinguish blips which represent "rare events" (outliers) in a stable distribution. This is not an exhaustive discussion of drift types—the reader is referred to [1], [5], [7], [17], and [18] for more information on class label swaps or changes in underlying data distributions.

Discussing the nature of concept drifts, several researchers refer to a probabilistic view on changes either in attribute value incoming distribution or class conditional probabilities $p(y|\mathbf{x})$ [7], [17]. In this paper, we consider the second case. Finally, a key issue while handling concept drift is not responding to minor fluctuations, which can be perceived as noise. A good algorithm should be capable to combine robustness to noise with sensitiveness to drifts.

B. Classifiers for Data Streams With Concept Drift

Different categorizations of methods for handling concept drift in data streams have been proposed [1], [5], [7], [17]. For the purposes of this paper, we discuss three categories most related to our research: windowing techniques, drift detectors, and ensemble methods. Windowing techniques provide a simple forgetting mechanism by selecting examples introduced to the learning algorithm, thus eliminating those examples that come from an old concept distribution. A different idea stands behind trigger approaches, which are based on drift detectors that react to concept changes and alarm when the classifier should be rebuilt or updated. Last, classifier ensembles, due to their modularity, provide a natural way of adapting to change by modifying ensemble components or their aggregation. Below, we discuss the most related algorithms falling into all three categories.

The most common strategy within the category of windowing techniques involves using sliding windows, which limit the number of classifier training examples to the most recent ones. Sliding windows combined with traditional batch algorithms, known from static environments, can produce stream classifiers. Unfortunately, when using windows of fixed size the user is caught in a tradeoff. A classifier built on a small window of examples will react quickly to changes but may lose on accuracy in periods of stability. On the other hand, a classifier built on a large window of examples will fail to adapt to rapidly changing concepts. For this reason, more dynamic ways of modeling the forgetting process, such as heuristic adjusting of the window size [19] or decay functions [20], have been proposed. A windowing technique was also used to adapt one of the most popular data stream classifiers, called the Very Fast Decision Tree, to evolving environments.

Very Fast Decision Tree (VFDT or Hoeffding Tree) is an incremental classifier for massive data streams proposed by Domingos and Hulten [6]. The algorithm induces a decision tree from a data stream incrementally by modifying the tree, without the need for storing examples after they have been used to update the tree. It works similarly to the classic tree induction algorithm and differs mainly in the selection of the split attribute. Instead of selecting the best attribute (in terms of a given split evaluation function) after viewing all the examples, it uses the Hoeffding bound to calculate the number of examples necessary to select the right split-node with a user-specified probability. The originally proposed VFDT algorithm was designed for static data streams and provided no forgetting mechanism. Hulten *et al.* [21] addressed this issue by introducing a new algorithm called CVFDT, which used a fixed-size window to determine which nodes are aging and may need updating.

In the category of drift detectors, first approaches used statistical tests to verify if the class distribution remains constant over time and rebuilt the base learner otherwise [22]. The most popular algorithm, called Drift Detection Method (DDM) [23], relies on the fact that in each iteration an online classifier predicts the class of an example. The prediction errors are modeled according to a binomial distribution. With such a model, one can track the error rate and verify whether it falls into the expected bounds or a warning or alarm level [23]. If the alarm level is reached, the classifier is dropped and a new one is created but only from examples stored in the separate "warning" window. DDM works best on data streams with sudden drift as gradually changing concepts can pass without triggering the alarm level. Baena–García *et al.* [24] proposed a modification of DDM, called EDDM, which works better than DDM for slow gradual drift but is more sensitive to noise.

Aside from drift detectors and windowing techniques, many researchers propose to use classifier ensembles to mine evolving data streams [2], [7]. Two general types of ensembles dedicated for evolving data are considered: online ensembles, which learn incrementally after processing single examples and ensembles based on processing blocks of data. Within the first group of approaches, generalizations of static solutions are often considered, e.g., Oza introduced an online version of bagging [25]. In Online Bagging, component classifiers are incremental learners that combine their decisions using a simple majority vote. The 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. Recently, Bifet *et al.* [26] introduced a modification of Oza's algorithm called Leveraging Bagging, which aims to add more randomization to the input and output of the base classifiers. The analysis of levels of diversity of ensembles was also considered in the DDD algorithm, a meta-system combining four diversified ensembles [27].

Another incremental ensemble was presented in an algorithm called Dynamic Weighted Majority (DWM) [28]. In DWM, a set of incremental classifiers is weighted according to their accuracy after each incoming example. With each mistake made by one of the DWM's component classifiers, its weight is decreased by a user-specified factor. Although weighting based on component predictions is the most popular aggregation technique, it is worth mentioning that special Bayesian combinations of local expert classifiers have also been studied [29]. A different approach was proposed by Kirkby in an algorithm called the Hoeffding Option Tree (HOT) [30]. This generalization of the Hoeffding Tree includes option nodes where instead of selecting only the best split-test attribute all promising attributes are kept. Later, for each of those attributes, a decision subtree is constructed. Making a

final decision with an option tree involves weighted combining of the predictions of all applicable subtrees.

An alternative approach to learning ensembles involves re-evaluating ensemble components with fixed-size blocks of incoming examples, called data chunks, and replacing the worst component with a classifier trained on the most recent examples. The most representative ensemble following this scheme was proposed in [10]. In their algorithm, called Accuracy Weighted Ensemble (AWE), the authors propose to train a new classifier on each incoming data chunk by a typical static learning algorithm, such as C4.5, RIPPER, or Naive Bayes. After the new classifier is trained, all previously learned component classifiers, already existing in the ensemble, are evaluated on the most recent chunk. These evaluations are done with a special version of the mean square error (see Section III for the precise formula), which allows the algorithm to select the k best classifiers to create a new ensemble. Usually the newest classifier replaces the worst performing from the existing ones. Moreover, the structure of the ensemble is pruned if errors of component classifiers are worse than the error of a random classifier. Similar chunk-based approaches include the Streaming Ensemble Algorithm (SEA) [3] and more recently Learn^{++}.NSE [31]. It is important to notice that the similarity of distributions in data chunks depends largely on the size of the chunks. Thus, bigger chunks will build more accurate classifiers but may contain more than one change. On the other hand, smaller chunks are better at separating changes but usually lead to poorer classifiers. In particular, ensembles built on large data chunks may react too slowly to sudden drifts occurring inside the chunks [12].

To overcome AWE's slow drift reactions, Nishida proposed a hybrid approach in which a data chunk ensemble is aided by a drift detector [11]. This solution, implemented in an algorithm called Adaptive Classifier Ensemble (ACE), aims at reacting to sudden drifts by tracking the classifier's error rate with each incoming example, while slowly reconstructing a classifier ensemble with large chunks of examples. This proposal could be also seen as a hybridization of the data chunk scheme with an incremental element. With a similar intention of tackling sudden and gradual drifts, Brzezinski and Stefanowski put forward the Accuracy Updated Ensemble, an algorithm that combines the principles of chunk-based ensembles with incremental base components [12]. The algorithm presented in this paper builds on that last solution and offers a mechanism capable of achieving accurate predictions in the presence of different types of drift at relatively low computational costs. In the following section, we discuss the details of the proposed algorithm and highlight its characteristic features.

III. Accuracy Updated Ensemble

Most data stream classification algorithms tend to specialize in one type of drift. Some classifiers are more accurate on datasets with sudden drifts while others perform better in the presence of gradual changes. The aim of our research is to put forward a data stream classifier that will react equally well to different types of drift. To achieve this goal, we propose to combine accuracy-based weighting mechanisms known from block-based ensembles with the incremental nature of Hoeffding Trees, in an algorithm called the Accuracy Updated Ensemble (AUE2).

The Accuracy Updated Ensemble maintains a weighted pool of component classifiers and predicts the class of incoming examples by aggregating the predictions of components using a weighted voting rule. After each data chunk of examples, a new classifier is created, which substitutes the weakest performing ensemble member. The performance of each component classifier is evaluated by estimating its expected prediction error on the examples from the most recent data chunk. After substituting the poorest performing component, the remaining ensemble members are updated, i.e., incrementally trained, and their weights are adjusted according to their accuracy. We propose to use Hoeffding Trees as component classifiers, but the presented algorithm can be considered as a general method, and in principle, one could use other online learning algorithms as base learners.

Let $\mathcal{S}$ be a data stream partitioned into evenly sized chunks $B_1, B_2, \ldots, B_n$ each containing d examples. For every incoming chunk B_i, the weights w_{ij} of component classifiers $C_j \in \mathcal{E}$ $(j = 1, 2, \ldots, k)$ are calculated by estimating the error rate on data chunk B_i as shown in (1)–(3)

$$\mathrm{MSE}_{ij} = \frac{1}{|B_i|} \sum_{\{\mathbf{x}, y\} \in B_i} \left(1 - f_y^j(\mathbf{x})\right)^2 \tag{1}$$

$$\mathrm{MSE}_r = \sum_y p(y)(1 - p(y))^2 \tag{2}$$

$$w_{ij} = \frac{1}{\mathrm{MSE}_r + \mathrm{MSE}_{ij} + \epsilon}. \tag{3}$$

Function $f_y^j(\mathbf{x})$ denotes the probability given by classifier C_j that $\mathbf{x}$ is an instance of class y. Following inspirations from the AWE algorithm [10], instead of single class predictions, probabilities of all classes are considered. The value of MSE_{ij} estimates the prediction error of classifier C_j on chunk B_i, while MSE_r is the mean square error of a randomly predicting classifier and is used as a reference point to the current class distribution. Additionally a very small positive value ϵ is added to the equation to avoid division by zero problems. The weighting formula presented in (3) aims at combining information about the classifier's accuracy and the current class distribution. Furthermore, by using a nonlinear function, compared to a linear one used in AWE, we highly differentiate component classifiers. The final version of the component weighting function (as well as the candidate weighting function discussed further in this section) was chosen after performing a comparative study of several alternative weighting approaches discussed in Section IV-C.

Apart from assigning new weights to ensemble members, with each data chunk B_i a candidate classifier, denoted as C', is created from examples within the most recent chunk. As C' is trained on the most recent data, it is treated as a "perfect" classifier and assigned a weight according to (4)

$$w_{C'} = \frac{1}{\mathrm{MSE}_r + \epsilon}. \tag{4}$$

Algorithm 1 Accuracy Updated Ensemble (AUE2)

Require: $\mathcal{S}$: data stream of examples partitioned into chunks, k: number of ensemble members, m: memory limit
Ensure: $\mathcal{E}$: ensemble of k weighted incremental classifiers

```
1: E ← ∅;
2: for all data chunks B_i ∈ S do
3:   C' ← new component classifier built on B_i;
4:   w_C' ← 1/(MSE_r + ε) (4);
5:   for all classifiers C_j ∈ E do
6:     apply C_j on B_i to derive MSE_ij;
7:     compute weight w_ij based on (3);
8:   end for
9:   if |E| < k then
10:    E ← E ∪ {C'};
11:  else
12:    substitute least accurate classifier in E with C';
13:  end if
14:  for all classifiers C_j ∈ E \ {C'} do
15:    incrementally train classifier C_j with B_i;
16:  end for
17:  if memory_usage(E) > m then
18:    prune (decrease size of) component classifiers;
19:  end if
20: end for
```

Compared to the function used to weight existing ensemble members, the weight of the candidate classifier $w_{C'}$ does not take into account the prediction error of C' on B_i. Such an approach is based on the assumption that the most recent chunk provides the best representation of the current and near-future data distribution. Since C' is trained on the most recent data, it should be treated as the best possible classifier.

If the ensemble contains less than k components, the candidate classifier is simply added to the ensemble. Otherwise, out of the k existing ensemble members $C_j \in \mathcal{E}$, the poorest performing classifier, i.e., the component with the lowest weight, is substituted with the candidate classifier C'. After the substitution, remaining ensemble members are incrementally trained by presenting examples from the most recent data chunk B_i. Our experiments (discussed in more detail in Section IV-C) have shown that for datasets which do not contain any drift, the incremental training of component classifiers in AUE2 can cause nonconstant memory usage. For this reason, after each data chunk, the size of the ensemble is compared with a user-specified memory limit. If the memory limit is exceeded, then the least active leaves of component Hoeffding Trees are pruned to match the memory restriction. After pruning, the ensemble is ready to classify examples from the next incoming data chunk. The pseudocode of AUE2 is presented in Algorithm 1.

In contrast to earlier proposed block-based ensembles, such as AWE or SEA, the AUE2 algorithm is not designed to use batch static learners but, instead, incrementally updates component classifiers. In our opinion, this should lead to better classification accuracy in the presence of slow gradual drifts and periods of stability. Additionally, since the components can be retrained, the algorithm should be less dependent on the chunk size and can use smaller chunks without deteriorating its accuracy [12], [32]. The first version of the proposed algorithm (AUE1) inherited several mechanisms from its predecessor AWE, such as candidate cross-validation and a classifier buffer, but also improved upon many elements, e.g., AUE1 conditionally updates component classifiers. However, our experiments, discussed in Section IV-C, have shown that there is significant room for improvement in terms of memory usage and classification accuracy. That is why, compared to AUE1, AUE2 introduces a new weighting function, does not require cross-validation of the candidate classifier, does not keep a classifier buffer, prunes its base learners, and always updates its components.

The Accuracy Updated Ensemble also differs from other data stream ensemble approaches. Ensemble members of AUE2 are weighted and can be removed, unlike in Online Bagging. Compared to VFDT-based ensembles, such as ASHT and HOT, we do not limit base classifier size and do not use any windows. Compared to Learn^{++}.NSE, the proposed algorithm incrementally trains existing component classifiers, retains only k of all the created components, and uses a different weighting function which ensures that components will have nonzero weights. In contrast to DWM, AUE2 processes the stream in chunks, weights components according to their prediction error, treats the candidate classifier as a perfect learner, and its weighting function does not require any user-specified parameters.

In a way, AUE2 can be considered as a hybrid approach—it can react to sudden drifts and it can gradually evolve with slow changing concepts. The rapid adaptation after sudden drifts is achieved by weighting classifiers according to their prediction error and giving the highest possible weight to the newest classifier. On the other hand, because components are updated after every chunk, they can react to gradual drifts. Additionally, the modular structure of AUE2 should protect the classifier from drastic accuracy losses in the presence of random blips, as a single "outlier" component can be overvoted when the target concept stabilizes. The performance of AUE2 in scenarios, involving different types of drifts, as well as no drift, will be examined in the following section.

IV. Experimental Evaluation

The proposed algorithm is evaluated in several experiments to simulate scenarios, involving different types of changes. In the following sections, we describe all of the used datasets, discuss experimental setup, and analyze experiment results.

A. Datasets

Most of the common benchmarks for machine learning algorithms, e.g., gathered in the UCI repository [33], contain too few examples to be concerned suitable for evaluating data stream classification methods, especially in terms of algorithm efficiency. Furthermore, datasets used to test algorithms designed for static environments usually do not contain any type of concept drift. In terms of real-world data, there is still a shortage of suitable and publicly available benchmark

datasets. Some researchers have used private data that cannot be reproduced by others [6], [10], [34], [35]. For this reason, data stream classification algorithms are tested mostly on synthetic datasets in which concept drift is introduced. Following this common approach, the proposed algorithm is compared with other classifiers on 11 synthetic and four real datasets. Artificial datasets were generated using the MOA framework and the real datasets are publicly available. A brief description of each dataset is provided below.[2]

Hyp: Hyperplane is a popular dataset generator utilized in many stream classification experiments [10], [35], [36]. It is mainly used to generate streams with incremental concept drift by slightly rotating the decision boundary with each consecutive example. We set the hyperplane generator to create two datasets, each containing 1 000 000 instances described by ten features. The first dataset (Hyp_S) contains incremental drift with the modification weight w_i changing by 0.001 with each example. The second dataset (Hyp_F) is similar to the first one but the change is more rapid with the weight changing by 0.1 with each example. Additionally, both datasets contain 5% of noise added to the concepts to randomly differentiate the instances. In this paper, by noise we shall refer to class noise, i.e., errors artificially introduced to class labels.

RBF: The radial basis function generator creates a user-specified number of drifting centroids, each defined by a class label, position, weight, and standard deviation. We use this generator to create three datasets, 1 000 000 examples each. The RBF_{ND} dataset has two decision classes and no drift. The RBF_B dataset contains four decision classes and four very short, sudden drifts (two blips), which should be ignored by the tested classifier. The last dataset from this group, RBF_{GR}, is designed to contain four gradual recurring drifts with each concept containing four decision classes.

SEA: The SEA generator [3] is used to create two datasets with sudden concept drifts. Each concept is defined by a sum of two functions, both dependent on a single attribute, which outputs a point belonging to one of the four possible decision classes. For our tests, we generate 1 000 000 instances with drifts occurring every 250 000 examples (SEA_S) and 2 000 000 instances with drifts occurring every 200 000 examples (SEA_F), with 10% noise introduced.

Tree: We use the random tree generator to create two drifting datasets, each described by five nominal and five numerical attributes. The Tree_S dataset contains four sudden recurring drifts evenly distributed over 1 000 000 examples. The Tree_F dataset contains only 100 000 instances but is the fastest changing dataset with 15 sudden drifts. In both cases, drift is introduced by abruptly changing the concept (randomly generated tree) after a given number of examples.

LED: LED [37] is a popular artificial dataset, which consists of a stream of 24 binary attributes that define the digit displayed on a seven-segment LED display. We use this generator to acquire two datasets. The first dataset, called LED_M, contains 1 000 000 instances with two gradually drifting concepts suddenly switching after 500 000 examples. Such a mixed type of drift is particularly difficult to learn. The second dataset (LED_{ND}) contains no drift but instead it is the largest and noisiest dataset with 10 000 000 examples and 20% of noise.

[2]Scripts available at: http://www.cs.put.poznan.pl/dbrzezinski/software.php.

TABLE I
CHARACTERISTIC OF DATASETS

Dataset	No. Inst	No. Attrs	No. Cls	Noise	No. Drifts	Drift Type
Hyp_S	1 M	10	2	5%	1	incremental
Hyp_F	1 M	10	2	5%	1	incremental
RBF_B	1 M	20	4	0%	2	blips
RBF_{GR}	1 M	20	4	0%	4	gradual
RBF_{ND}	1 M	20	2	0%	0	none
SEA_S	1 M	3	4	10%	3	sudden
SEA_F	2 M	3	4	10%	9	sudden
Tree_S	1 M	10	4	0%	4	s. recurring
Tree_F	100k	10	6	0%	15	s. recurring
LED_M	1 M	24	10	10%	3	mixed
LED_{ND}	10 M	24	10	20%	0	none
Elec	45 k	7	2	-	-	unknown
Cov	581 k	53	7	-	-	unknown
Poker	1 M	10	10	-	-	unknown
Airlines	539 k	7	2	-	-	unknown

Elec, Cov, Poker, and Airlines: The first of four utilized real datasets, called Electricity (Elec) [38], is one of the most widely used in data stream classification. It consists of energy prices from the electricity market, which were affected by market demand, supply, season, weather, and time of day. Elec contains 45 312 instances each described by seven features. The second real dataset, Covertype (Cov), contains cover type information about four wilderness areas. Examples are defined by 53 cartographic variables that describe one of seven possible forest cover types. The whole dataset consists of 581 012 instances and has been used in several papers on data stream classification [25], [39]. The third real benchmark data are the Poker dataset [39], which consists of 1 000 000 examples describing the suits and ranks of a hand of five playing cards. This gives a total of ten predictive attributes per instance (5 cards $\times$ 2 attributes—suit and rank) with an additional class attribute that describes 1 of 10 poker hands. Finally, Airlines is a real dataset containing 539 383 examples described by seven attributes. Airlines encapsulates the task of predicting whether a given flight will be delayed, given the information of the scheduled departure.

The described synthetic datasets were chosen to evaluate all of the analyzed algorithms in different scenarios. As for the real datasets, we share the common assumption that we cannot unequivocally state when drifts occur or if there is any drift. The real datasets serve to compare the algorithms in a real-life scenario rather than a concrete drift situation. Table I summarizes the characteristics of each dataset.

B. Experimental Setup

All of the tested algorithms were implemented in Java as part of the MOA framework [13]. In particular, AUE2 was implemented for this paper, the source codes of the ACE and Learn++.NSE were provided courtesy of Dr. Nishida and Dr. Gonçalves, respectively, while all the remaining classifiers were already a part of MOA. The experiments were conducted

on a machine equipped with two 12-core AMD Opteron 6172, 2.1-GHz processors and 64 GB of RAM. To make the comparison more meaningful, we set the same parameter values for all the algorithms. For ensemble methods, we set the number of component classifiers to ten: AUE2, AUE1, AWE, DWM, ACE, Online Bagging, Leveraging Bagging have ten Hoeffding Trees, HOT has ten options. We decided to use ten component classifiers as, according to our preliminary study, using more classifiers linearly increased processing time and memory, but did not notably improve classification accuracy of the analyzed ensemble methods. The data block size used for chunk ensembles was equal $d = 500$ for all the datasets. Although AUE2 can remain accurate using smaller chunks, we chose to use blocks containing 500 examples as this size was considered the minimal suitable for block-based ensembles, such as AWE [3], [10], and lower values would drastically decrease AWE's accuracy. We set the static window size of Win to $10 \times d$ to make the number of examples seen by the windowed classifier similar to that seen by ensemble methods. The parameters of the Hoeffding Tree used with the static window were the same as those of the option tree and the component classifiers (also Hoeffding Trees) of all the ensemble methods. More precisely, we used Hoeffding Trees enhanced with adaptive Naive Bayes leaf predictions with a grace period $n_{\min} = 100$, split confidence $\delta = 0.01$, and tie-threshold $\tau = 0.05$ [6]. Due to the fact that the only available implementation of ACE could not be fully adjusted to use classifiers from the MOA framework, we used ACE (as originally proposed by Nishida [11]) with 10 C4.5 trees as batch learners and Naive Bayes as an online learner. As suggested in [31], Learn++.NSE does not use any pruning mechanism and has the sigmoid slope $a = 0.5$ and the sigmoid crossing point $b = 10$.

According to the main characteristics of data streams [2], [3], [13], we evaluate the performance of algorithms with respect to time efficiency, memory usage, and classification accuracy. All the performance measures were calculated using the data chunk evaluation method, which works similarly to the test-then-train paradigm with the difference that it uses data chunks instead of single examples [32]. This method reads incoming examples without processing them, until they form a data chunk of size d. Each new data chunk is first used to evaluate the existing classifier, then it updates the classifier, and finally it is disposed to preserve memory. Such an approach allows us to measure average chunk training and testing times and is less pessimistic than the test-then-train method. It is suitable for static and evolving streams and provides a natural method of reducing result storage requirements.

C. Analysis of the Components of the Proposed Algorithm

While constructing the AUE2 algorithm, we decided to use the experience gathered from our previous comparative study between AUE1 and AWE [12] and verify the properties of AUE1 in search of improvements. One of the first analyzed properties was the use of the classifier buffer. With each chunk, after evaluating its components AUE1 uses k best classifiers to form an ensemble. To reduce memory usage, only n of all the constructed component classifiers are stored until the next chunk is processed. During our previous experiments with AUE1 [12], we used a buffer of $n = 30$ classifiers out of which $k = 15$ were selected to form an ensemble. The assumption behind such an approach was that a buffer of additional, out-of-ensemble, classifiers could prove profitable in the presence of recurring drifts. In the design phase of AUE2, we decided to verify this assumption by analyzing the pros and cons of maintaining a buffer. Table II presents the results of comparing AUE2 with a buffer ($k = 10$ and $n = 30$) and AUE2 without one ($k = n = 10$).

TABLE II
COMPARISON OF AUE2 WITH AND WITHOUT A BUFFER IN TERMS OF AVERAGE ACCURACY [%], AVERAGE MEMORY USAGE [MB], AVERAGE CHUNK TRAINING TIME AND AVERAGE CHUNK TESTING TIME [S]

	AUE2 With a Buffer				AUE2 Without a Buffer			
	Acc.	Mem.	Train.	Test.	Acc.	Mem.	Train.	Test.
Hyp_S	88.59	1.86	0.23	**0.02**	**88.64**	**0.58**	**0.07**	**0.02**
RBF_B	**94.07**	2.73	0.66	**0.06**	94.06	**2.15**	**0.19**	**0.06**
RBF_{GR}	**93.37**	4.30	0.82	**0.06**	93.30	**3.91**	**0.19**	**0.06**
RBF_{ND}	**92.42**	121.51	1.33	**0.02**	92.41	**11.91**	**0.07**	**0.02**
SEA_S	89.00	1.46	0.16	**0.01**	**89.02**	**0.88**	**0.03**	**0.01**
Elec	**70.86**	0.39	0.05	**0.01**	70.76	**0.09**	**0.03**	**0.01**
Cov	**81.24**	1.56	0.78	0.12	81.19	**0.78**	**0.30**	**0.11**
Poker	**60.57**	0.29	0.13	0.03	59.86	**0.09**	**0.06**	**0.02**

As Table II shows, in terms of accuracy, AUE2 with a buffer seems to perform slightly better than AUE2 without one. Nevertheless, the gain in accuracy is quite minor or even neglectable compared to the training time and memory cost. In the analyzed scenarios, using a buffer of 20 additional classifiers requires on an average over five times more training time and twice as much memory as not using any buffer. For this reason, the buffer was excluded from AUE2.

These results led to an additional conclusion. Although AUE2 does not require any pruning to restrict memory usage on datasets with drift [12], [32], by testing the algorithm on a dataset without any drift (RBF_{ND}), we noticed that it requires such a mechanism in static environments. That is why, in contrast to AUE1, AUE2 comes with a pruning mechanism that removes the least used leaves of each component Hoeffding Tree to fit a user-specified memory limit.

Another costly property of AUE1 was the weighting of each newly created component classifier. AUE1, as well as AWE, uses expensive ten-fold cross-validation (10 cv) to weight the candidate classifier on the most recent data chunk [10], [12]. We analyzed the impact of using other weighting schemes starting with other cross-validations, such as four-fold (4 cv) and two-fold (2 cv) cross-validation. We also considered the candidate's weight as a function of the remaining classifiers' weights. We investigated the performance of the candidate classifier with a weight equal to the maximum (Max), average (Mean), and minimum (Min) weight of the remaining classifiers, half of the sum of remaining classifier weights (Half), and half of the sum of remaining classifier weights minus a very small value ϵ (Half_ϵ). Additionally, we experimented

TABLE III
AVERAGE ACCURACIES OF AUE2 WITH DIFFERENT CANDIDATE WEIGHTING FUNCTIONS [%]

	10 cv	4 cv	2 cv	Max	Mean	Min	Half	Half$_\epsilon$	w_{CL}	w_{CN}
Hyp$_S$	88.64	**88.70**	88.44	88.36	88.30	84.99	88.58	88.49	88.52	88.43
RBF$_B$	94.06	94.64	94.81	94.82	94.84	**95.87**	92.61	93.09	94.78	94.77
RBF$_{\text{GR}}$	93.30	93.98	94.10	94.21	94.23	74.73	63.38	63.64	94.15	**94.43**
RBF$_{\text{ND}}$	92.41	93.08	92.58	93.22	93.27	**93.40**	91.33	91.65	93.12	93.33
SEA$_S$	89.02	89.20	**89.21**	89.20	89.20	87.65	89.03	89.02	89.21	89.19
Elec	70.76	71.16	71.83	62.66	61.88	43.99	51.10	49.69	69.35	**77.32**
Cov	81.19	84.03	84.79	84.70	84.72	75.03	81.10	81.50	84.46	**85.20**
Poker	59.86	60.39	60.77	60.20	60.54	46.55	46.53	46.52	59.67	**66.23**

not only with the candidate weight, but also with the overall weight definition itself. We analyzed linear and nonlinear functions, such as $w_L = \max(\text{MSE}_r - \text{MSE}_i, 0) + \epsilon$ and $w_N = 1/(\text{MSE}_r + \text{MSE}_i + \epsilon)$. By using MSE_i and MSE_r, we associate the component classifiers weight with its accuracy and the current class distribution. The ϵ in these functions is a very small value used to ensure that the ensemble will always be able to give a nonzero prediction. In reference to functions w_L and w_N, we decided to treat the candidate classifier as a "perfect classifier", i.e., one for which $\text{MSE}_i = 0$. Such an approach is based on the implicit assumption that the most recent data chunk provides the best representation of the near-future data distribution. The resulting candidate weight functions for these methods are $w_{CL} = \text{MSE}_r + \epsilon$ and $w_{CN} = 1/(\text{MSE}_r + \epsilon)$. It is worth noticing that the calculation of w_{CL} and w_{CN} does not require any cross-validation nor the analysis of remaining classifier weights and can be performed in constant time.

As Table III shows, treating the candidate classifier as a "perfect" classifier substantially increases accuracy, especially when combined with a nonlinear weighting function. The most interesting results are achieved by w_{CN}, which proves best on most datasets and close to best on the remaining ones. The difference is especially visible on real datasets (Elec, Cov, Poker) where w_{CN} improves accuracy by a few percent compared to other solutions. It is worth noticing that, compared to using a linear function, by using a nonlinear weighting function more voting power is given to the candidate classifier. This is especially important in the presence of concept drift when the candidate is the only component of the ensemble with information about the incoming new concept. Giving so much power to the candidate can prove inconvenient in the presence of sudden noise when the incoming concept should be treated as an outlier or when no drift occurs and the more experienced components should be more important. The obtained results seem to support that hypothesis as for data with no drift (RBF$_{\text{ND}}$ and RBF$_B$) best results are achieved by the weighting mechanism that gives the most voting power to older components, i.e., the Min approach. Being the most accurate in different scenarios and much more computationally effective than cross-validation, we chose the w_{CN} function as the candidate classifier weighting mechanism for AUE2.

In an attempt to further decrease memory usage and possibly improve classification accuracy, we proposed and analyzed two alternative component updating mechanisms. The first mechanism selects only the $b < k$ best weighted components for updating. We experimentally evaluated the effect of updating $b \in [4; 9]$ highest weighted components of an ensemble of ten classifiers. In the second mechanism, we proposed to stop updating a component classifier if the difference between the mean square error of that component obtained on the most recent data chunk ($\text{MSE}_{i,t}$) and the error obtained on the previous chunk ($\text{MSE}_{i,t-1}$) is greater than 0 and less than a user-defined threshold θ, i.e., component C_i is not updated if $0 < \text{MSE}_{i,t} - \text{MSE}_{i,t-1} < \theta$. We experimentally tested this approach for $\theta \in [0.005; 0.05]$.

The obtained results, omitted due to space limitations, have shown that refraining from updating component classifiers is not the best strategy in a stream with drifts. Not only does updating all components give best average accuracy, but also the less refraining was performed the better the results were. On the other hand, substantial savings in terms of memory can be achieved by not updating all of the component classifiers. Refraining from updating when MSE_i settles at $\theta = 0.5\%$ of what it was on the previous chunk requires, on an average, 14% less memory than always updating all components. Nevertheless, the proposed refraining techniques allowed to reduce memory requirements but did not increase accuracy. Such an outcome may suggest that the incremental creation of strong classifiers as ensemble members is of more value to the prediction of the ensemble. These results may, therefore, be considered concordant with the standpoint presented in [31], suggesting that drifting environments provide natural diversity and the premise of weaklearnability does not apply to them. As the main aim of the proposed algorithm is to react accurately to different types of drift, we decided not to use any refraining technique in AUE2.

D. Comparative Study of Classifiers

After establishing the properties of AUE2, a set of experiments was conducted to compare the newly proposed algorithm against 11 classifiers: the Hoeffding Option Tree (HOT), ACE, the previous version of the Accuracy Updated Ensemble (AUE1), the Accuracy Weighted Ensemble (AWE), Leveraging Bagging (Lev), Online Bagging (Oza), Dynamic Weighted Majority (DWM), Learn++.NSE (NSE), Drift Detection Method with a Hoeffding Tree (DDM), a single Hoeffding Tree with a static window (Win), and the Naive Bayes algorithm (NB). We chose AWE and AUE1 as those are

TABLE IV
AVERAGE CLASSIFICATION ACCURACIES IN PERCENTAGE [%]

	ACE	AUE1	AWE	AUE2	HOT	DDM	Win	Lev	NB	Oza	DWM	NSE
Hyp_S	80.65	88.59	**90.43**	88.43	83.23	87.92	87.56	85.36	81.00	89.89	71.20	86.83
Hyp_F	84.56	88.58	89.21	**89.46**	83.32	86.86	86.92	87.21	78.05	89.32	76.69	85.39
RBF_B	87.34	94.07	78.82	94.77	93.79	88.30	73.07	**95.28**	66.97	93.08	78.11	73.02
RBF_{GR}	87.54	93.37	79.74	94.43	93.24	87.99	74.67	94.74	62.01	92.56	77.80	74.49
RBF_{ND}	84.74	92.42	72.63	**93.33**	91.20	87.62	71.12	92.24	72.00	91.37	76.06	71.07
SEA_S	86.39	89.00	87.73	**89.19**	87.07	88.37	86.85	87.09	86.18	88.80	78.30	86.23
SEA_F	86.22	88.36	86.40	**88.72**	86.25	87.80	85.55	86.68	84.98	88.37	79.33	85.07
Tree_S	65.77	84.35	63.74	**84.94**	69.68	80.58	50.15	81.69	47.88	81.67	51.19	49.37
Tree_F	45.97	**52.87**	45.35	45.32	40.34	42.74	41.54	33.42	35.02	43.40	29.30	33.90
LED_M	64.70	67.29	67.11	67.58	66.92	67.17	65.52	66.74	67.15	**67.62**	44.43	62.86
LED_{ND}	46.33	50.68	51.27	51.26	51.17	51.05	47.07	50.64	**51.27**	51.23	26.86	47.16
Elec	75.83	70.86	69.33	77.32	**78.21**	64.45	70.35	76.08	73.08	77.34	72.43	73.34
Cov	67.05	81.24	79.34	85.20	**86.48**	58.11	77.19	81.04	66.02	80.40	80.84	77.16
Poker	67.38	60.57	59.99	66.10	74.77	60.23	58.26	**82.62**	58.09	61.13	74.49	59.56
Airlines	66.75	63.92	63.31	**67.37**	66.18	65.79	64.93	63.10	66.84	66.39	61.00	63.83

TABLE V
AVERAGE CHUNK TRAINING TIME IN CENTISECONDS [CS]

	ACE	AUE1	AWE	AUE2	HOT	DDM	Win	Lev	NB	Oza	DWM	NSE
Hyp_S	26.83	14.82	12.15	4.41	0.69	0.33	0.17	6.04	**0.03**	3.94	7.26	116.20
Hyp_F	25.78	14.13	12.11	4.57	2.39	0.38	0.20	5.62	**0.03**	3.97	7.77	173.73
RBF_B	72.27	47.93	42.46	13.88	2.72	0.87	0.29	13.34	**0.05**	9.63	14.22	628.33
RBF_{GR}	72.72	54.51	42.63	14.08	3.45	0.88	0.29	13.83	**0.04**	9.96	14.57	679.69
RBF_{ND}	19.94	44.58	11.80	4.67	1.40	0.27	0.17	6.69	**0.03**	4.74	8.35	186.12
SEA_S	4.95	7.64	4.36	1.63	0.37	0.15	0.13	2.65	**0.01**	2.59	2.58	66.53
SEA_F	5.06	5.49	4.24	1.60	0.28	0.13	0.13	2.58	**0.01**	2.31	2.76	64.74
Tree_S	20.63	26.99	15.07	5.24	0.78	0.37	0.19	7.00	**0.02**	4.79	7.41	196.91
Tree_F	27.98	20.55	18.20	7.90	1.73	0.73	0.51	8.81	**0.02**	5.36	9.19	32.85
LED_M	7.75	31.47	25.48	8.81	4.44	0.62	0.26	9.49	**0.03**	7.98	8.21	402.41
LED_{ND}	7.73	27.72	25.30	8.99	10.11	1.28	0.22	10.14	**0.03**	11.60	7.98	932.29
Elec	4.47	5.00	5.57	3.26	1.89	1.00	1.00	3.75	**0.07**	2.70	5.90	6.74
Cov	23.35	40.87	41.29	14.83	6.64	1.23	0.38	10.21	**0.09**	9.66	18.24	425.72
Poker	2.78	9.57	6.56	4.20	1.91	0.39	0.18	3.07	**0.02**	2.83	7.69	108.00
Airlines	4.37	10.45	14.15	6.79	1.62	0.58	0.76	7.01	**0.02**	4.78	32.23	69.05

the classifiers we tried to improve upon. HOT and ACE were selected as they can be considered as hybrid ensemble algorithms combining elements of incremental learning. Oza, Lev, NSE, and DWM were chosen as strong representatives of online ensembles. The DDM algorithm and the windowed Hoeffding Tree were chosen as representatives of single classifiers. Additionally, the Naive Bayes algorithm is added to the comparison as a reference for using an algorithm without any drift reaction mechanism. All the studied algorithms were evaluated in terms of classification accuracy, memory usage, chunk training time, and testing time. Average values of the analyzed performance measures are given in Tables IV–VII.

Apart from analyzing the average performance of the algorithms, we generated four graphical plots for each dataset depicting the algorithms' functioning in terms of training time, testing time, memory usage, and classification accuracy. By presenting the performance measure calculated after each data chunk on the y-axis and the number of processed training examples on the x-axis, one can examine the dynamics of a given classifier, in particular, its reactions to concept drift. Such graphical plots are the most common way of displaying results in data stream mining papers [26], [31]. Due to space limitations we will only analyze the most interesting plots, which highlight characteristic features of the studied algorithms.

Fig. 1 reports accuracies of the analyzed algorithms on the RBF_{GR} dataset, which contains gradual recurring drifts. Looking at the plot one can see drops in accuracy around examples number 125, 250, 375, and 500 k. The most severely malfunctioning algorithm in the presence of gradual recurring drifts is NB, followed by Win, NSE, DWM, and AWE. The subsequent drops in accuracy of the Naive Bayes algorithm suggest that classifiers without any drift reaction mechanism fail to successfully learn from data with gradual recurrent drifts. On the other hand, Win, NSE, DWM, and AWE appear to react too slowly, possibly due to the strong time similarity of data used for prediction. Additionally, DDM and ACE both use drift detectors, which are designed to work best with sudden changes and for this reason the performance of these algorithms may not be as good as the performance of ensemble approaches. The two most accurate algorithms on this dataset are AUE2 and Lev. Both of these algorithms require similar

TABLE VI
AVERAGE CHUNK TESTING TIME IN CENTISECONDS [CS]

	ACE	AUE1	AWE	AUE2	HOT	DDM	Win	Lev	NB	Oza	DWM	NSE
Hyp_S	0.60	1.82	1.72	1.75	0.33	0.19	0.18	2.29	**0.18**	1.97	0.80	7.48
Hyp_F	0.59	1.82	1.74	1.73	1.04	0.20	0.21	1.99	**0.18**	1.99	0.78	7.40
RBF_B	1.10	6.15	6.58	6.06	1.35	0.66	0.68	6.45	**0.66**	6.78	2.51	31.80
RBF_{GR}	1.11	6.44	6.53	6.29	1.69	0.70	0.71	6.81	**0.65**	7.11	2.68	20.10
RBF_{ND}	0.58	2.47	1.67	2.23	0.60	0.20	0.20	3.22	**0.19**	2.57	0.79	3.44
SEA_S	0.47	0.76	0.61	0.67	0.09	0.07	0.08	0.73	**0.07**	0.82	0.28	3.70
SEA_F	0.47	0.66	0.59	0.65	0.09	**0.07**	0.08	0.71	0.08	0.73	0.29	2.87
Tree_S	0.82	2.54	2.32	2.52	0.36	**0.22**	0.22	3.32	0.23	2.96	0.73	1.88
Tree_F	0.97	2.93	2.46	3.31	**0.27**	0.36	0.39	3.70	0.48	3.43	0.95	1.25
LED_M	2.10	4.92	4.05	3.83	**0.29**	0.48	0.41	4.58	0.39	5.62	2.02	4.49
LED_{ND}	2.05	4.15	4.01	3.90	**0.27**	0.97	0.42	4.95	0.40	9.28	2.03	4.08
Elec	0.62	0.85	0.40	1.18	0.73	**0.21**	0.27	1.29	0.35	1.30	0.46	2.05
CovType	0.84	6.17	6.34	6.74	4.33	**0.55**	0.71	5.22	1.26	7.45	2.29	15.36
Poker	0.57	1.79	0.37	1.92	1.31	**0.22**	0.20	1.09	0.47	1.68	0.48	2.69
Airlines	0.30	0.44	0.22	2.22	0.36	0.23	**0.21**	1.78	0.19	2.04	0.35	1.78

TABLE VII
AVERAGE CLASSIFIER MEMORY USAGE IN MEGABYTES [MB]

	ACE	AUE1	AWE	AUE2	HOT	DDM	Win	Lev	NB	Oza	DWM	NSE
Hyp_S	0.14	1.97	0.28	0.63	2.94	0.24	**0.00**	4.30	0.01	0.71	0.15	16.85
Hyp_F	0.13	1.23	0.31	0.57	9.57	0.55	**0.00**	1.70	0.01	0.87	0.20	36.98
RBF_B	0.18	2.99	0.45	2.40	5.38	0.33	**0.01**	5.32	0.01	1.16	0.35	36.99
RBF_{GR}	0.19	4.67	0.43	4.65	5.94	0.30	**0.01**	6.84	0.01	1.94	0.33	36.99
RBF_{ND}	0.14	13.07	0.25	12.74	5.88	0.59	**0.00**	38.49	0.01	5.83	0.22	36.97
SEA_S	0.10	1.56	0.20	0.92	0.71	0.15	**0.00**	0.80	0.00	1.12	0.07	36.97
SEA_F	0.10	1.02	0.20	0.57	0.71	0.08	**0.00**	0.52	0.00	0.65	0.08	36.97
Tree_S	0.22	5.22	0.49	4.95	4.34	0.59	**0.00**	17.28	0.01	5.75	0.15	36.98
Tree_F	0.22	1.68	0.35	0.88	0.52	0.12	**0.01**	0.55	0.01	0.28	0.07	0.41
LED_M	0.27	0.62	0.61	0.22	2.06	0.17	**0.01**	0.62	0.03	1.50	0.04	36.99
LED_{ND}	0.27	0.62	0.61	0.22	15.74	4.73	**0.01**	0.29	0.03	6.16	0.03	180.68
Elec	0.10	0.39	0.27	0.46	0.75	0.03	**0.00**	0.34	0.01	0.14	0.11	0.10
Cov	0.20	1.57	0.68	0.85	17.17	0.08	**0.02**	0.82	0.05	0.32	0.48	12.59
Poker	0.14	0.33	0.27	0.20	8.05	0.14	**0.00**	1.23	0.01	0.12	0.31	25.47
Airlines	0.11	2.35	5.71	62.34	65.65	13.47	**0.05**	38.95	0.06	30.80	1.14	11.10

training and testing time but Lev requires almost twice as much memory as AUE2.

Figs. 2 and 3 show classification accuracy and memory usage on the Tree_S dataset, which was designed to test the algorithms' reaction to recurring sudden drifts. The drifts occurring every 200 k examples are clearly visible both on the accuracy and memory plot. In the presence of sudden recurring drifts, AUE1 and AUE2 seem to perform best, with only the first drift having a major impact on their accuracy. Compared to recurring gradual drifts, the remaining algorithms are further behind in terms of accuracy. This is especially apparent with the HOT algorithm, which appears to lose accuracy with every consecutive drift. Looking at the memory plot in Fig. 3, we can see that AUE1 and AUE2 abruptly reduce their memory usage when a drift occurs. The drop in accuracy of the previously learned components is reflected in their mean square error, which forces one of the previously learned base classifiers to be disposed. Algorithms that seem not to have pruned their base classifiers after a sudden drift, such as HOT or Lev, lose accuracy. Similar behavior was observed in figures for the SEA_S and SEA_F datasets, which represent scenarios with sudden concept drifts.

It is worth noting that NSE requires much more time and memory than the remaining algorithms. This is only due to the fact that, following [31], no pruning was used to limit the number of NSE's component classifiers. On small datasets, like Elec, we can see that when only few components are created NSE uses less memory than other ensemble methods.

Although on the Tree_S dataset, AUE2 performed slightly better than AUE1, on the Tree_F dataset AUE1 is clearly the winning algorithm. The characteristic feature of the Tree_F dataset is the speed of recurring changes. The classifier buffer which was removed from the AUE2 algorithm is the attribute that most probably helped AUE1 outclass other data stream learners on this dataset.

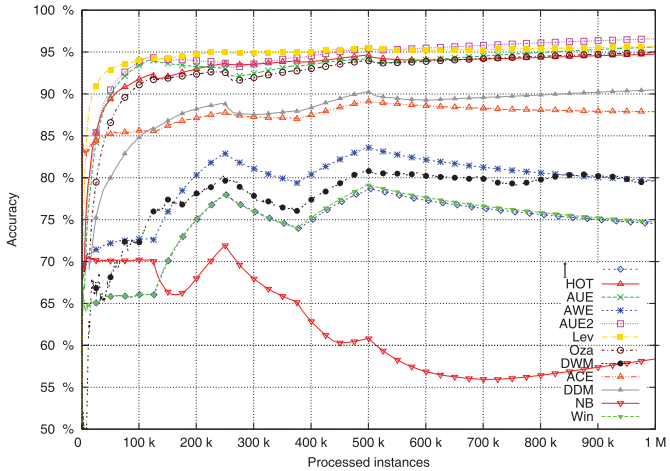

Fig. 1. Classification accuracy on the $\mathrm{RBF}_{\mathrm{GR}}$ dataset.

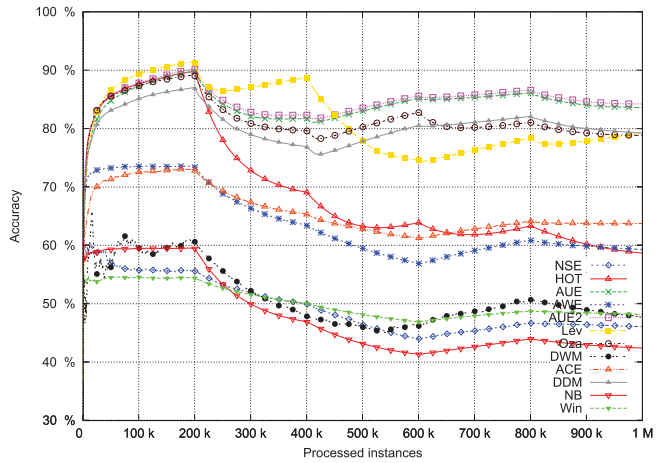

Fig. 2. Classification accuracy on the Tree_S dataset.

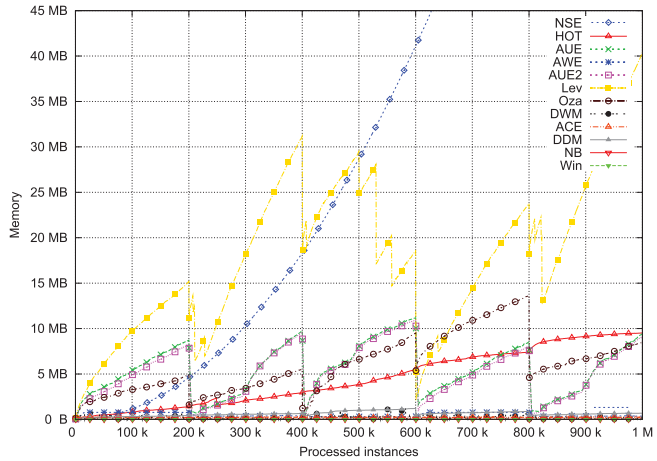

Fig. 3. Memory usage on the Tree_S dataset.

A different experiment used the RBF_B dataset, which incorporates very short, sudden concept changes (blips). Blips should be treated as outliers and should not have any long-term impact on the classifier's functioning. As Fig. 4 shows, apart from NB, Win, NSE, DWM, and AWE all the

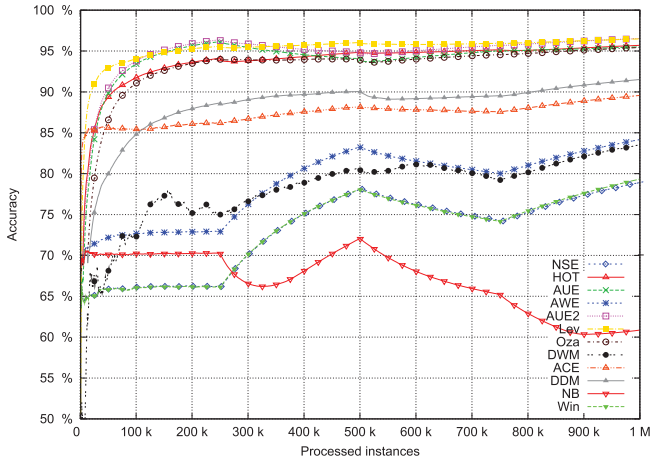

Fig. 4. Classification accuracy on the RBF_B dataset.

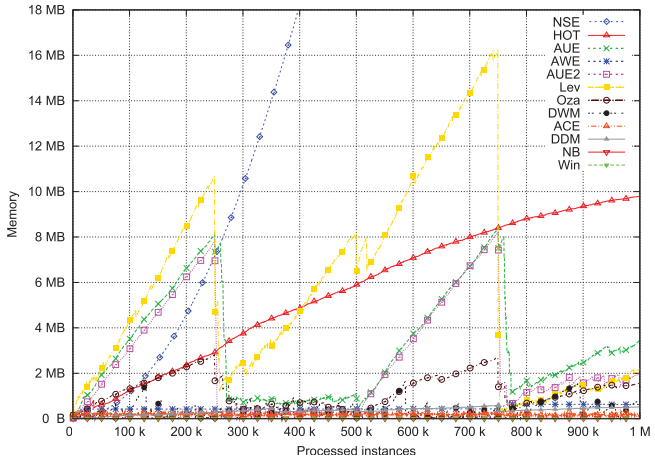

Fig. 5. Memory usage on the RBF_B dataset.

classifiers maintain stable accuracy throughout the entire dataset. Analyzing the memory plot in Fig. 5, one can see that AUE1 and AUE2 react to blips just like they reacted to sudden changes. The capability of sustaining accuracy by these two algorithms is possible due to the fact that only one ensemble component is removed per chunk. Even when a single component is removed in the occurrence of an outlier concept, AUE1 and AUE2 still perform well after the blip. It is also worth noticing that the warning/alarm level mechanism used in DDM and ACE worked well and allowed these algorithms to stay accurate even though their classification error must have raised.

For datasets with incremental drifts, i.e., Hyp_S and Hyp_F, the best performing algorithms are AWE, Oza, and AUE2. AWE seems to perform particularly well on the Hyp_S dataset. It is also worth noting that the Win classifier, usually performing rather poorly in terms of accuracy, reacts quite well to slow changes. The algorithms that perform the worst are NB, ACE, DWM, and HOT. The Naive Bayes classifier has no drift reaction mechanism; the drift detector in ACE is not triggered, thereby causing poor reaction to gradual changes, while HOT and DWM appear to not be pruning outdated data, with HOT additionally using too much memory.

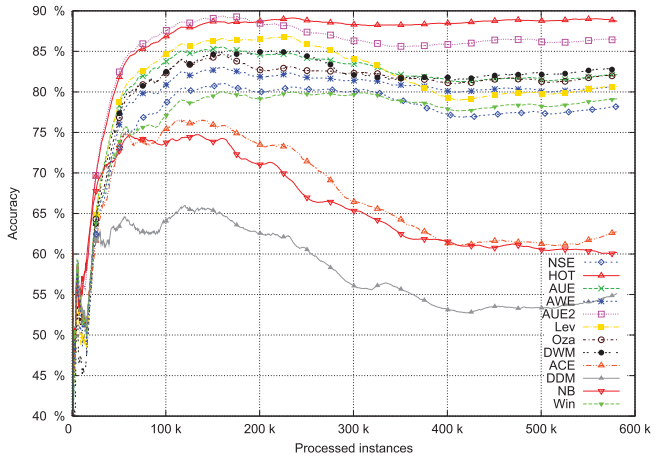

Fig. 6. Classification accuracy on the Cov dataset.

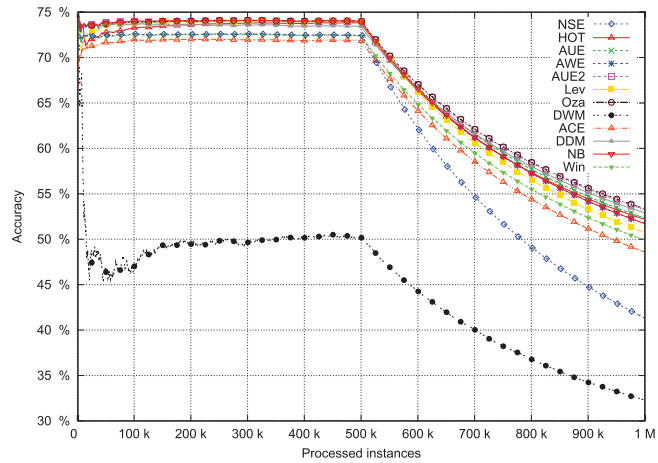

Fig. 7. Classification accuracy on the LED_M dataset.

When no drift is present, AUE2 and AUE1 are the most accurate classifiers. On the RBF_{ND} dataset, AUE2 has the highest accuracy followed by AUE1 and Lev, while on LED_{ND} AUE1, AUE2, and NB achieve almost identical results.

On real datasets (Elec, Cov, Poker, and Airlines), HOT is the best performing learning algorithm followed by AUE2. Additionally, on the Poker dataset, Lev clearly outperforms all the other classifiers. It is worth mentioning that the accuracy of HOT comes at the price of high memory costs. It seems that for the analyzed real-world datasets the pruning mechanism, present in most adaptive ensembles, is not as important as the constant training of base classifiers, characteristic for HOT. The accuracy plot for the Cov dataset is presented in Fig. 6. By looking at the performance of NB, DDM, and ACE, one can see that the analyzed dataset probably contains changes. The accuracy plots for Elec, Poker, and Airlines also contain fluctuations, which were not present in the accuracy plots of artificial datasets without drift, i.e., RBF_{ND} and LED_{ND}.

Finally, let us analyze the accuracy plot for the LED_M dataset presented in Fig. 7. In this dataset, we incorporated a complex change by joining two gradually drifting streams. After 500 k examples, the target concept is suddenly switched but the gradual changes in the new concept prove to be very difficult to classify. Although Oza and AUE2 achieve best average accuracies, all algorithms seem to fail in reacting to the change. This shows that complex combinations of drifts can be an interesting topic for further research.

E. Statistical Analysis of Results

To extend the analysis provided in Section IV-D, we carried out statistical tests for comparing multiple classifiers over multiple datasets [40]. We used the nonparametric Friedman test, where average results of compared algorithms are ranked on each dataset (the lower the rank the better). The null-hypothesis for this test is that there is no difference between the performances of all the tested algorithms. Moreover, in case of rejecting this null-hypothesis, we use the Bonferroni–Dunn post-hoc test [40] to verify whether the performance of AUE2 is statistically different from the remaining algorithms.

The average ranks of the analyzed algorithms are presented in Table VIII, providing a comparison in terms of accuracy, training and testing time, as well as memory usage. First, we perform the Friedman test to verify the statistical significance of the differences between accuracies of the algorithms. As the test statistic $F_F = 12.902$ and the critical value for $\alpha = 0.05$ is 1.851, the null hypothesis is rejected. Considering accuracies, AUE2 provides the best average achieving usually first or second rank, regardless of the existence or type of drift. To verify whether AUE2 performs better than the remaining algorithms, we compute the critical difference (CD) chosen by the Bonferroni–Dunn test [40]. When the difference between corresponding average ranks of two classifiers is greater or equal to CD, one can state that they are significantly different.

As $CD = 3.736$, AUE2 performs significantly better than NSE, DDM, DWM, AWE, ACE, Win, and NB. As for the difference between AUE2 and the remaining algorithms, the experimental data are not sufficient to reach such a conclusion. Motivated by the fact that AUE2 has an accuracy rank much higher than AUE1, HOT, Lev, and Oza, we have decided to additionally perform the Wilcoxon signed rank test to get a better insight into the comparison of pairs of classifiers [40]. In contrast to the Friedmann test, in the Wilcoxon signed rank test, the values of differences in performance of a pair of classifiers are taken into account. The p-values resulting from this test are: $p_{\text{AUE1}} = 0.006$, $p_{\text{HOT}} = 0.020$, $p_{\text{Lev}} = 0.009$, and $p_{\text{Oza}} = 0.003$ for AUE1, HOT, Lev, and Oza, respectively. All these p-values support our observation that AUE2 is better in terms of accuracy than any of the compared algorithms.

We perform a similar analysis concerning average classifier training time, also presented in Table VIII. Computing the test value, we obtain $F_F = 133.834$. The null hypothesis can be rejected and by comparing average algorithm ranks with CD and performing additional Wilcoxon signed rank tests, we can state that AUE2 is trained slower than Win, NB, but significantly faster than NSE, AUE1, Lev, ACE, AWE, and DWM ($p_{\text{Lev}} = 0.023$, $p_{\text{ACE}} = 0.004$, $p_{\text{AWE}} = 0.001$, and $p_{\text{DWM}} = 0.002$). Analogously, comparing average testing time we also reject the null hypothesis ($F_F = 74.549$) and state that AUE2 classifies slower than DDM, HOT, Win, and NB, but faster than Oza and NSE ($p_{\text{Oza}} = 0.003$, $p_{\text{NSE}} = 0.004$). Such

TABLE VIII
AVERAGE ALGORITHM RANKS USED IN THE FRIEDMAN TESTS

	ACE	AUE1	AWE	AUE2	HOT	DDM	Win	Lev	NB	Oza	DWM	NSE
Acc.	7.33	4.00	6.40	**2.20**	5.40	6.47	8.80	5.27	9.07	3.67	9.80	9.60
Train.	8.73	10.33	9.40	6.33	4.20	2.80	2.13	7.13	**1.00**	5.73	8.13	12.00
Test.	5.13	9.07	7.07	8.53	4.53	2.27	2.33	9.93	**2.13**	10.80	5.33	10.80
Mem.	4.00	9.33	6.27	8.07	10.13	4.93	**1.00**	9.40	2.00	7.73	4.33	10.80

an outcome is not surprising as Win, NB, HOT, and DDM are single classifiers, while the rest of the analyzed algorithms are ensembles, each with ten base learners.

Finally, we compare average memory usage of each algorithm. The test value being $F_F = 60.307$, we reject the null hypothesis. By comparing average ranks, we can state that AUE2 uses more memory than Win, NB, and ACE but is more memory efficient than NSE, AUE1, HOT, Lev ($p_{\text{NSE}} = 0.007$, $p_{\text{AUE1}} = 0.007$, $p_{\text{HOT}} = 0.015$, and $p_{\text{Lev}} = 0.050$).

V. CONCLUSION

In this paper, we presented and evaluated a block-based stream ensemble classifier, called AUE2, designed to react to different types of concept drift. The main novelty of the proposed algorithm is the combination of an AWE inspired ensemble weighting mechanism with incremental training of component classifiers. This hybrid approach allows AUE2 react to many different types of concept changes, such as sudden, gradual, recurring, short-term, and mixed drifts and makes AUE2 less dependent on data block size. Additional contributions of AUE2 include the proposal of a new component weighting function and a cost-effective candidate weight. By treating the candidate classifier as a "perfect" classifier, AUE2 ensures that the current concept is strongly reflected in the ensemble's prediction. The proposed algorithm was also optimized for memory usage by restricting ensemble size and incorporating a simple inner-component pruning mechanism.

We also investigated different strategies concerning component classifier updates. Our experiments have shown that, in terms of accuracy, all component classifiers in AUE2 should be updated after each incoming data block. Such an approach promotes the incremental creation of strong classifiers as ensemble members and provides more accurate predictions of the ensemble. From this point of view, our results coincide with those presented in [31], therefore, suggesting that drifting environments provide natural diversity and the premise of weaklearnability does not apply to them.

We also carried out an experimental study comparing AUE2 with 11 additional state-of-the-art data stream methods, including single classifiers, ensembles, and hybrid approaches in different scenarios. The obtained results confirmed that classifiers without any drift reaction mechanism fail to successfully learn from data with sudden, gradual, or recurrent drifts. They also seem to confirm that ensemble approaches that use batch classifiers, such as AWE, may suffer accuracy drops after sudden concept drifts [11], while drift detectors are less accurate on gradually drifting streams [24]. Novel findings include the reaction of algorithms to short random abrupt changes. The obtained results showed that block-based ensemble methods are more robust to random blips than single classifiers, as previously trained components allow them to recover from premature reactions. Furthermore, experiments on datasets with fast recurring drifts showcased that the speed of changes is crucial to the decision whether a buffer of previously constructed component classifiers is useful or not. If recurrent changes are very frequent, a buffer can improve accuracy but in other cases it only increases memory requirements and algorithm processing time.

Above all, the experimental study demonstrated that AUE2 can offer high classification accuracy in environments with different types of drift as well as in static environments. AUE2 provided best average classification accuracy out of all the tested algorithms, while proving less memory consuming than other ensemble approaches, such as Leveraging Bagging or Hoeffding Option Trees. As future work, we plan to investigate the possibility of adapting the proposed algorithm to work in a truly incremental fashion in partially labeled streams.

ACKNOWLEDGMENT

The authors would like to thank K. Nishida and P. Gonçalves for sharing their implementations of the ACE and Learn++.NSE algorithms.

REFERENCES

[1] J. Gama, *Knowledge Discovery from Data Streams*, 1st ed. London, U.K.: Chapman & Hall, 2010.
[2] L. I. Kuncheva, "Classifier ensembles for detecting concept change in streaming data: Overview and perspectives," in *Proc. 2nd Workshop SUEMA*, 2008, pp. 5–10.
[3] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 377–382.
[4] Y. Cao, H. He, and H. Man, "SOMKE: Kernel density estimation over data streams by sequences of self-organizing maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 8, pp. 1254–1268, Aug. 2012.
[5] I. Zliobaite, "Adaptive training set formation," Ph.D. dissertation, Dept. Comput. Sci., Vilnius Univ., Vilnius, Lithuania, 2010.
[6] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2000, pp. 71–80.
[7] L. I. Kuncheva, "Classifier ensembles for changing environments," in *Proc. 5th Int. Workshop Multiple Classifier Syst.*, 2004, pp. 1–15.
[8] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. New York, USA: Wiley, 2004.
[9] N. C. Oza, "Online ensemble learning," Ph.D. dissertation, Computer Science Division, Univ. California, Berkeley, CA, USA, Sep. 2001.
[10] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 226–235.
[11] K. Nishida, K. Yamauchi, and T. Omori, "ACE: Adaptive classifiers-ensemble system for concept-drifting environments," in *Proc. 6th Int. Workshop Multiple Classifier Syst.*, 2005, pp. 176–185.

[12] D. Brzezinski and J. Stefanowski, "Accuracy updated ensemble for data streams with concept drift," in *Proc. 6th HAIS Int. Conf. Hybrid Artif. Intell. Syst., II*, 2011, pp. 155–163.
[13] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive online analysis," *J. Mach. Learn. Res.*, vol. 11, no. 5, pp. 1601–1604, May 2010.
[14] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "A practical approach to classify evolving data streams: Training with limited amount of labeled data," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 929–934.
[15] M. Kmieciak and J. Stefanowski, "Handling sudden concept drift in Enron message data streams," *Control Cybern.*, vol. 40, no. 3, pp. 667–695, Mar. 2011.
[16] L. L. Minku, A. P. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 5, pp. 730–742, May 2010.
[17] A. Tsymbal, "The problem of concept drift: Definitions and related works," Dept. Comput. Sci., Trinity College Dublin, Dublin, Ireland, Tech. Rep. TCD-CS-2004-15, 2004.
[18] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Mach. Learn.*, vol. 23, no. 1, pp. 69–101, Apr. 1996.
[19] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proc. 7th SIAM Int. Conf. Data Mining*, 2007, pp. 443–448.
[20] E. Cohen and M. J. Strauss, "Maintaining time-decaying stream aggregates," *J. Algorithms*, vol. 59, no. 1, pp. 19–36, Apr. 2006.
[21] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 97–106.
[22] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, nos. 1–2, pp. 100–115, Jun. 1954.
[23] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Proc. 17th SBIA Brazilian Symp. Artif. Intell.*, 2004, pp. 286–295.
[24] M. Baena-García, J. D. Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, "Early drift detection method," in *Proc. 4th Int. Workshop Knowl. Discovery Data Streams*, 2006, pp. 1–10.
[25] N. C. Oza and S. J. Russell, "Experimental comparisons of online and batch versions of bagging and boosting," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 359–364.
[26] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Proc. Eur. Conf. Mach. Learn./PKDD, I*, 2010, pp. 135–150.
[27] L. L. Minku and X. Yao, "DDD: A new ensemble approach for dealing with concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 4, pp. 619–633, Apr. 2012.
[28] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *J. Mach. Learn. Res.*, vol. 8, pp. 2755–2790, Dec. 2007.
[29] A. Kehagias and V. Petridis, "Predictive modular neural networks for time series classification," *Neural Netw.*, vol. 10, no. 1, pp. 31–49, Jan. 1997.
[30] R. Kirkby, "Improving Hoeffding trees," Ph.D. dissertation, Dept. Comput. Sci., Univ. Waikato, Hamilton, New Zealand, 2007.
[31] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517–1531, Oct. 2011.
[32] D. Brzezinski, "Mining data streams with concept drift," M.S. thesis, Inst. Comput. Sci., Poznan Univ. Technology, Poznan, Poland, 2010.
[33] A. Frank and A. Asuncion. (2010). *UCI Machine Learning Repository* [Online]. Available: http://archive.ics.uci.edu/ml
[34] W. Fan, "Systematic data selection to mine concept-drifting data streams," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2004, pp. 128–137.
[35] W. Fan, Y. A. Huang, H. Wang, and P. S. Yu, "Active mining of data streams," in *Proc. 4th SIAM Int. Conf. Data Mining*, Apr. 2004, pp. 457–461.
[36] I. Zliobaite, "Combining time and space similarity for small size learning under concept drift," in *Proc. 18th ISMIS Int. Symp.*, 2009, pp. 412–421.
[37] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees* (Wadsworth Statistics/Probability). Belmont, CA, USA: Wadsworth, 1984.
[38] M. Harries, "SPLICE-2 comparative evaluation: Electricity pricing," School of Computer Science and Engineering, Univ. New South Wales, New South Wales, Australia, Tech. Rep. 9905, 1999.
[39] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New ensemble methods for evolving data streams," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2009, pp. 139–148.
[40] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, no. 1, pp. 1–30, Jan. 2006.

Dariusz Brzezinski received the B.Sc. and M.Sc. degrees in computer science from the Poznan University of Technology, Poznan, Poland, in 2009 and 2010, respectively, where he is currently pursuing the Ph.D. degree.

His current research interests include data stream mining, concept drift, online classification algorithms, and XML document clustering.

Jerzy Stefanowski received the Ph.D. and Habilitation degrees in computer science from the Institute of Computing Science, Poznan University of Technology, Poznan, Poland.

He is an Associate Professor with the Institute of Computing Science, Poznan University of Technology. His current research interests include machine learning, data mining and intelligent decision support, in particular rule induction, multiple classifiers, class imbalance, data preprocessing, and handling uncertainty in data.