A Boosting-like Online Learning Ensemble

Conference Paper · July 2016

DOI: 10.1109/IJCNN.2016.7727427

3 authors:

Roberto Souto Maior de Barros
Federal University of Pernambuco
100 PUBLICATIONS **315** CITATIONS
SEE PROFILE

Silas Garrido Teixeira de Carvalho Santos
Federal University of Pernambuco
18 PUBLICATIONS **83** CITATIONS
SEE PROFILE

Paulo Mauricio Gonçalves Jr.
Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco (IFPE)
24 PUBLICATIONS **158** CITATIONS
SEE PROFILE

A Boosting-like Online Learning Ensemble

Roberto Souto Maior de Barros
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
Email: roberto@cin.ufpe.br

Silas Garrido T. de Carvalho Santos
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
Email: sgtcs@cin.ufpe.br

Paulo Mauricio Gonçalves Júnior
Instituto Federal de Educação,
Ciência e Tecnologia de Pernambuco
Recife, Brazil
Email: paulogoncalves@recife.ifpe.edu.br

***Abstract*—Changes in the data distribution (concept drift) makes online learning a challenge that is progressively attracting more attention. This paper proposes *Boosting-like Online Learning Ensemble (BOLE)* based on heuristic modifications to *Adaptable Diversity-based Online Boosting (ADOB)*, which is a modified version of Oza and Russell's *Online Boosting*. More precisely, we empirically investigate the effects of (a) weakening the requirements to allow the experts to vote and (b) changing the concept drift detection method internally used, aiming to improve the ensemble accuracy. BOLE was tested against the original and other modified versions of both boosting methods as well as three renowned ensembles using well-known artificial and real-world datasets and *statistically* surpassed the accuracies of both boosting methods as well as those of the three ensembles. The accuracy improved in most tested situations but this is more evident in the datasets with more concept drifts, where the accuracy gains were very high. Finally, we believe the proposed heuristics are generalizable to at least some of the other existing online boosting algorithms.**

I. Introduction

Data streams are environments where information occurs in the form of a flow of data arriving continuously, in large quantities, and quickly. Applications that deal with data streams are usually required to process this information online, i.e., as they arrive, because it is often impractical or even impossible to store all the data. In other words, restrictions on usage of memory and time usually apply and multiple reading of the same instance of data is normally not allowed.

In addition, as these data are frequently collected over long periods of time, usually the data distribution will not be stationary, which is commonly known as concept drift [1], and concepts may recur [2]. Machine learning from data streams in the presence of concept drift is the scenario we consider online learning. Examples of online learning applications include monitoring the movement data from sensors, TCP/IP traffic, the purchase history of customers, or water temperatures.

Concept drift is often classified based on the speed of the change: it is abrupt, when the transition from an old to a new concept occurs suddenly or quickly, and gradual, when such a transition happens over a number of instances.

Boosting [3], [4] is a well-known and general method for improving the accuracy of other algorithms ("weak" learners). The idea is to train several classifiers using different distributions over the training data and combine them in an ensemble. Notice that many boosting algorithms come with some theoretical guarantees about their results.

Boosting implementations based on AdaBoost.M1 [4] only permit a classifier to vote if its error is below 50%, the value associated to random guessing. However, when the problem is not binary, this 50% requirement is often too strong, as stated by Freund and Schapire [4].

Moreover, AdaBoost.M1 also stops processing a given instance of data as soon as it finds a classifier with error greater than 50%. We suspect that discarding an instance because one of the classifiers presents low accuracy is a damaging strategy for online methods, as they only access each data once.

Originated from AdaBoost.M1, Oza and Russell's Online Boosting (OzaBoost) [5] proposed to use a Poisson distribution to simulate its behavior in online environments.

Inspired on OzaBoost, Adaptable Diversity-based Online Boosting (ADOB) [6] proposed to change the way diversity is distributed during training to speed up the experts recovery after concept drifts.

Based on the aforementioned observations, we decided to modify ADOB (and in fact also OzaBoost) to try a few different strategies aimed at improving the precision of the corresponding ensembles. So, in this work, we empirically studied three different heuristic configuration strategies by testing some plausible combinations in both algorithms using the Massive Online Analysis (MOA) framework, release 2012.08.

More specifically, we decided to investigate two possibilities for weakening the traditional boosting voting strategy and force more experts to vote. In the first scenario, we keep the below 50% error requirement to vote but accept the votes of all the classifiers in the ensemble that meet this 50% condition. In the second (more permissive and daring) scenario, we adopt a higher error limit to accept the classifiers' votes, and a slightly different strategy to aggregate them.

However, because we are deliberately disregarding some of the theoretical assumptions of boosting, and thus the associated theoretical guarantees, we avoid calling the resulting ensemble an online boosting variation. Instead, we named it Boosting-like Online Learning Ensemble (BOLE), which is merely inspired on online boosting. We refer to the version based on OzaBoost as (OzaBole).

The third adaptation that was examined is the substitution of the drift detector internally used in both methods, which is Adaptive Windowing (Adwin) [7]. Considering the results of a recent comparison of concept drift detectors [8] concluded

that the Drift Detection Method (DDM) [9] was the best method *overall*, we decided that BOLE would use DDM.

This paper is structured as follows: Section II surveys related work; Section III details the proposed modifications and describes BOLE; Section IV briefly introduces the datasets used in the experiments; Section V describes the experiments configuration; Section VI analyses the results, including the corresponding statistical evaluation; and, finally, Section VII presents our conclusions.

II. Related Work

Bagging and Boosting [4] use a set of classifiers trained on the original data by aggregating the responses of each classifier to deliver better predictions but using different strategies. More specifically, Boosting uses different distributions over the training data and varies the amount of diversity given to each classifier depending on their previous predictions.

Oza and Russell's Online Bagging and Boosting [5] both use a Poisson distribution to simulate the behavior of their corresponding offline algorithms in online environments. Adwin Online Bagging (ADWIN OZABAG) [10] is basically Oza and Russell's Online Bagging (OZABAG) using ADWIN to detect concept drifts. Similarly, Adwin Online Boosting (ADWIN OZABOOST), available in the MOA framework, is OZABOOST equipped with ADWIN as its drift detector.

Leveraging Bagging (LEVBAG) [11], a modified version of ADWIN OZABAG, uses higher diversity in the training and in the predictions and usually delivers improved accuracies.

Dynamic Weighted Majority (DWM) [12] is a weighted ensemble that extends the Weighted Majority Algorithm (WMA) [13] to identify concept drifts. DWM adds and removes classifiers according to its global performance: a classifier is added when the ensemble commits an error; the weight of each classifier is reduced when it commits an error; and a classifier is removed when its weight is very low, indicating it presented low accuracy on many examples.

Diversity for Dealing with Drifts (DDD) [14], [15] uses four ensembles with high and low diversity, before and after a concept drift is detected. It tries to select the best weighted majority of ensembles before and after concept drifts detected by the Early Drift Detection Method (EDDM) [16].

A. ADOB

ADOB [6] is based on ADWIN OZABOOST and uses a different strategy to speed up the recovery of the experts after the concept drifts. Algorithm 1 shows the main ADOB pseudo-code as published in [6].

Notice that ADOB sorts the experts by accuracy before processing each instance (line 4). This affects the way diversity is distributed to the classifiers and tends to slightly improve the accuracy of the ensemble just after the concept drifts, especially when these drifts are abrupt.

When an instance d arrives, initially the expert with less accuracy will be selected. If d is correctly classified, it is assumed the other (more accurate) experts also have good chances of correctly classifying it (an error is unlikely).

Algorithm 1: Adaptable Diversity-based Online Boosting

Input: ensemble size M, instance d, ensemb. h

1 $minPos \leftarrow 1$; $maxPos \leftarrow M$;
2 $correct \leftarrow$ false;
3 $\lambda \leftarrow 1.0$; $\lambda^{sc} \leftarrow 0.0$; $\lambda^{sw} \leftarrow 0.0$;
4 **sort** h **by** accuracy **in** ascending order;
5 **for** $m \leftarrow 1$ **to** M **do**
6 **if** $correct$ **then**
7 $pos \leftarrow maxPos$;
8 $maxPos \leftarrow maxPos$ - 1;
9 **else**
10 $pos \leftarrow minPos$;
11 $minPos \leftarrow minPos$ + 1;
12 **end**
13 $K \leftarrow$ Poisson(λ);
14 **for** $k \leftarrow 1$ **to** K **do**
15 $h_{pos} \leftarrow$ Learning(h_{pos}, d);
16 **end**
17 **if** $h_{pos}(d)$ *was correctly classified* **then**
18 $\lambda_m^{sc} \leftarrow \lambda_m^{sc} + \lambda$;
19 $\lambda \leftarrow \lambda \left(\frac{N}{2\lambda_m^{sc}}\right)$;
20 $correct \leftarrow$ true;
21 **else**
22 $\lambda_m^{sw} \leftarrow \lambda_m^{sw} + \lambda$;
23 $\lambda \leftarrow \lambda \left(\frac{N}{2\lambda_m^{sw}}\right)$;
24 $correct \leftarrow$ false;
25 **end**
26 **end**
27 **return** h;

Also, observe that λ is reduced when the classification is correct and increased if it is incorrect (lines 17 to 25). This makes the influence of an unlikely error on λ decrease as more instances are processed, because the next selected expert will be the one with the best accuracy (lines 6 to 8). So, experts with the worst accuracies, and most likely to make mistakes, will only be selected at the end.

III. Proposed Heuristics and BOLE

This section provides detailed descriptions of the heuristic modifications proposed to online boosting algorithms, instantiated and tested in both ADOB and OZABOOST, aimed at improving their accuracies, particularly in datasets with frequent and/or abrupt concept drifts.

As already mentioned, many online boosting methods implement AdaBoost's requirements for a classifier to vote, i.e., its error must be below 50%, a condition that is often too strong when the problem is not binary. Besides, these methods also discard the instance when one of the classifiers fails this condition, which is probably a poor strategy in online environments, since each instance is read only once. We call this the *50%-Break* voting strategy.

Even though the pseudo-code presented in their paper does not explicitly reproduce the aforementioned behavior of Adaboost.M1, Oza and Russell write nothing about doing anything differently regarding these features. Thus, in this case, it is reasonable to assume OZABOOST should replicate Adaboost.M1. And indeed this is how it is implemented in its corresponding code available in the MOA framework [17].

Similarly, the aforementioned boosting behavior is not explicitly written in the ADOB paper [6] but it is also included in the corresponding implementation code available at https://sites.google.com/site/moamethods.

It is worth saying that some online boosting methods (e.g. OSBoost [18]) are not based on AdaBoost.M1 and/or do not use such 50% condition and/or do not discard any instances.

Algorithm 2 shows a very abstract and deliberately much simplified pseudo-code corresponding to the java method *getVotesForInstance*, which implements the original voting computation of both tested boosting algorithms as implemented in the MOA framework.

Algorithm 2: Online boosting voting computation

```
Input: ensemble size M, instance x
for m ← 1 to M do
    Calculates the error ε_m
    if ε_m <= 0.5 then
        Calculates the member weight w_m
        Calc. the member weighted vote wv_m
        Combines wv_m with other votes of instance x
    else
        break
    end
end
return highest weighted combined vote for x
```

Notice that the **if** statement written in the pseudo-code corresponds to the part of the code that only permits a classifier (weak hypothesis) to vote if its error is up to 50% and that its corresponding **else** clause contains a **break** which prevents an instance from being processed by the remaining classifiers after one of them fails the 50% condition.

It is worth pointing out that the corresponding **if** statement in the actual java code of method *getVotesForInstance* uses a different test condition, but this condition is met exactly when the error is greater than 0.5%, which is explicitly tested in method *getEnsembleMemberWeight*.

Following from these observations, we decided to try heuristic modifications, to examine different strategies. More specifically, we empirically test three different modifications to generate ensembles aimed at beating the precision accuracies of the corresponding online boosting methods. And to pursue higher accuracies, we were prepared, perhaps boldly, to risk giving up boosting theoretical guarantees. As part of this process we also compare some reasonable combinations.

The first two changes weaken this behavior and permit more classifiers to vote. Initially, we maintain the 50% error requirement but do not stop processing the instances. This modification is simple and does not change the general idea of boosting. We named it the *50%-Continue* voting strategy.

The second scenario lets the error bound assume other values and more classifiers vote. Notice that, in the boosting original calculation, the weight of a classifier is positive when its classification error is below 50%. So, this modification requires some other arrangement in the calculation of the weights of the classifiers to prevent negative weights.

Finally, the third change we tested was to replace the drift detection method internally used from ADWIN to DDM. In this case, the change in the code is a mere parametrization of code already available in the MOA framework.

A. The BOLE implementation

Because we are envisaging a single BOLE implementation, one of the first design decisions we made was to add new parameters to its implementation whenever necessary.

The implementation of the first modification is very simple and could be carried out by simply removing the **else** clause and the **break** command of the method *getVotesForInstance*. However, we created a new parameter called *breakVotes*, with possible values being 'y' and 'n': when it is set to 'y', BOLE will behave just like the original method; otherwise, the **break** command is never executed.

Similarly, for the second scenario, we created the parameter *errorBound*, which expects a positive value between 0.5 and 1.0. Values greater than 0.5 lets more classifiers vote, whereas values below 0.5 would impose stronger restrictions on the voting process.

To avoid negative weights in the classifiers allowed to vote, we adopted a simple shift strategy in the weights of *all* classifiers. Likewise, the value used in this shift strategy is a parameter, called *weightShift*, and its expected values are in the [0.0, 5.0] interval. When its value is set to 0.0, BOLE will calculate the weights just like the original method.

To minimize the shift strategy interference in the weights, *weightShift* should be the smallest value that avoids negative weights. For example, if the error bound is changed to 60%, *weightShift* should be *at least* 0.4055 because the original weighting function would generate a weight of approximately -0.4055 when the error of a classifier is 0.6. For the extreme error bound of 100%, the corresponding shift should be 5.0.

Notice *weightShift* can also be used to generate different weighting strategies in the ensemble, while maintaining the rest of the method behaving just like boosting. Higher values of *weightShift* would make the more accurate classifiers have comparatively smaller effects on the final results. When its value is high enough, i.e. it is greater than most of the classifiers original weights, the weighting function would tend to become very similar to a simple majority vote.

Nevertheless, we have *not* thoroughly explored this direction of investigation nor the combined effect of such strategy with the use of other values for the error bound.

Algorithm 3 details BOLE's voting computation, again using a very abstract and much simplified pseudo-code, which represents our modified version of the java method *getVotesForInstance*.

Algorithm 3: BOLE's voting computation

Input: ensemble size M, instance x, $breakVotes$, $errorBound$, $weightShift$

```
for m ← 1 to M do
    Calculates the error ε_m
    if ε_m <= errorBound then
        Calculates the member weight w_m
        w_m ← w_m + weightShift
        Calc. the member weighted vote wv_m
        Combines wv_m with other votes of instance x
    else
        if breakVotes = 'y' then
            break
        end
    end
end
return highest weighted combined vote for x
```

IV. Datasets

This section presents the datasets that were chosen for the experiments described in Section VI, designed to analyze the performance of the methods using the most meaningful combinations of the three proposed improvements and to compare them to other ensemble methods available in the MOA framework website at http://moa.cs.waikato.ac.nz/.

We selected artificial datasets, useful since it is possible to define the number, position, and size of the concept drifts and thus simulate different scenarios, as well as real-world datasets, which bring unpredictability and volume of data. All the chosen datasets are also freely available, most of them in the MOA website.

A. Artificial Datasets

We picked four artificial datasets, two of them with abrupt concept drifts and two with gradual concept drifts. These are: Stagger [10], [19], Agrawal [20], [21], Mixed [2], [9], and Waveform [10], [22].

Stagger has three attributes $color \in \{green, blue, red\}$, $shape \in \{triangle, circle, rectangle\}$, and $size \in \{small, medium, large\}$, and three concepts: (1) $color = red \wedge size = small$; (2) $color = green \vee shape = circle$; and (3) $size = medium \vee size = large$. This dataset is usually employed to simulate abrupt concept drifts.

Agrawal has information from people aiming to receive a certain amount of loan. From this data, they are classified as belonging to group A or group B. The attributes are: salary, commission, age, education level, zip code, value of the house, etc. To perform the classification, Agrawal et al. [20] propose ten functions, each with different forms of evaluation. In addition, it is possible to add noise.

Composed of two boolean (v and w) and two numeric attributes (x and y), the data in the Mixed dataset can be classified as positive or negative. A positive example has two of the following three conditions satisfied: v, w, and $y < 0.5 + 0.3\sin(3\pi x)$. To simulate a concept drift, the labels of the aforementioned conditions are reversed.

Waveform has three-classes and 40 numerical attributes, with the last 19 used to produce noise. The goal is to detect the waveform generated by combining two of three base waves. To perform changes, the positions of the attributes, which represents a certain context, are reversed.

B. Real Datasets

We also selected three real-world datasets, with very different number of instances and complexity, and previously used in the area. In these datasets, the number and position of the drifts are unknown.

The Electricity dataset [9], [12], [15] stores data from the Australian New South Wales Electricity Market where prices depend on market demand and supply – it has 45,312 instances and eight attributes. The prices are set every five minutes and the class label identifies the change of the price related to a moving average of the last 24 hours. The goal is to predict if the price will increase or decrease. It is probably the most used real dataset in the data streams area.

Covertype [2], [10], [21] stores information on the forest cover type for 30x30 meter cells obtained from US Forest Service (USFS) Region 2 data and contains 581,012 instances and 54 attributes (numeric and categoric). The goal is to predict the forest cover type from cartographic variables. This real-world dataset is also frequently used in the area.

The Poker Hand dataset [10], [22] represents the problem of identifying the value of five cards in the Poker game. It is constituted of five categoric and five numeric attributes and one categoric class with 10 possible values informing the value of the hand: one pair, two pairs, a sequence, etc. In this original and harder to classify version of the dataset, with 1,000,000 instances, the cards are not ordered.

V. Experiments Configuration

This section describes the experiments designed to test and evaluate our ideas. Specifically, some of the meaningful combinations of the three proposed modifications implemented in ADOB (and also OzaBoost) were tested among themselves, as well as against other ensembles aimed at learning from data streams with concept drifts: DDD, DWM, and LevBag.

To evaluate accuracy, we use the Interleaved Test-Then-Train methodology: each incoming instance is first tested and, then, it is used for training. This guarantees that every instance is used both for testing and training and avoids the problem of training before testing on any given instance.

Since neither ADOB nor OzaBoost use much execution time or memory, and also because our modifications should not change this scenario, the methods are only compared in terms of accuracy.

The computer used to execute the experiments was an Intel Core i5 4210U processor, with 4GB of main memory, and running the Ubuntu 14.04 LTS 64 bits operating system.

A. Configuration of the Artificial Datasets

We generated three versions of each of the four artificial datasets (12 in total), with 10, 40, and 80 concept changes, respectively. They are all composed of 10,000 instances and have the concept drifts distributed at regular intervals.

The three versions of both Stagger and Agrawal have abrupt drifts whereas all versions of Mixed and Waveform have gradual changes. In all these gradual datasets, the length of the concept drifts was set to 50 instances. In the Agrawal datasets, 1% of noise was inserted in each of the six numeric attributes.

Finally, to compute the precision of the methods in the artificial datasets, the experiments were executed 40 times and the average results were computed alongside with 95% confidence intervals.

B. Parametrization of the Methods

As several methods have common parameters, these were all set similarly, for a fair comparison of their results. Likewise, the chosen base learner was a Hoeffding Tree [23] and the number of experts was set to 10 in all of them.

To detect concept drifts, ADWIN OZABOOST, ADOB, and LEVBAG all use ADWIN. The only formal parameter of ADWIN is δ, the maximum global error, and its default value at MOA is 0.002. However, the ADWIN code available in the MOA framework has an informal parameter as an internal variable, which is the minimum number of processed instances necessary to reduce the window size ($mintClock$, set to 32). We refer to this configuration as ADWIN$_{OLD}$.

On the other hand, based on partial results of ongoing and unpublished research, we believe these are not the best parameter values for ADWIN when many concept drifts are expected. Thus, we decided to run our experiments using the detectors with a more sensitive to concept drifts parametrization, despite this making them more likely to raise false alarms. The chosen parametrization for ADWIN is: $\delta = 0.58$ and $mintClock = 70$. Notice that (a) changing the value of δ directly influences the sensibility of ADWIN and (b) increasing $mintClock$ avoids consecutive detections, notably during gradual concept drifts.

In addition, aiming to separate the effect of this different configuration from those of the proposed modifications to the boosting methods, we also tested the original versions of ADOB and OZABOOST using this new setting.

The DDM implementation available in the MOA framework has one formal parameter, the minimum number of processed instances before a drift can be detected (n), with default value 30, and two others hard-coded, representing the number of standard deviations to raise warnings ($w = 2$) and to detect drifts ($d = 3$). As in the case of ADWIN, we chose to use a different, more sensitive, configuration: $n = 7$, $w = 1.2$, and $d = 1.95$.

The parameters of the other methods were always set to their default values, as specified by their authors and their specific values are given below.

The parameters of DDD are W, which controls its robustness to false alarms and was set to 1, and λ_l and λ_h, which are the values that represent ensembles with low and high diversity, respectively set to 1 and 0.1.

DDD uses EDDM to detect changes. The parameters of EDDM with their respective default values are the number of instances ($n = 30$) and of errors ($e = 30$) before starting to detect changes, and the confidence levels to activate the warning level ($w = 0.95$) and to detect drifts ($d = 0.9$).

DWM uses three parameters: the time needed to verify if any expert will be added or removed and to update the weights of classifiers that incorrectly classifies the actual instance ($p = 50$); the decrement applied to the expert when it makes a mistake ($\beta = 0.5$); and the minimum value an expert must have to stay in the ensemble ($\theta = 0.01$) [12].

Finally, LEVBAG uses λ, which controls the weight of resampling and was set to 6.

C. Tested Versions of the Methods

Seven versions based on each of the two methods were tested. The first versions, named ADOB and OZABOOST, respectively, are their original versions, using the traditional boosting voting strategy (*50%-Break*) and ADWIN$_{OLD}$, the default configuration of ADWIN. Versions 2 of both methods are similar but use the new parametrization of ADWIN.

The other five versions were named using BOLE and OZABOLE, respectively, as they use the modified implementation code. Versions 1 use the *50%-Continue* voting strategy (breakVotes = 'n', errorBound=0.5, weightShift=0.0), the first proposed modification to the original methods, also adopting the new parametrization of ADWIN.

The BOLE$_2$ and OZABOLE$_2$ versions both implement the two proposed modifications, again using the new parametrization of ADWIN. The parameter values chosen to let more classifiers vote were: breakVotes = 'n', errorBound=0.6, and weightShift=1.0. We named this combination the *60%-Continue* voting strategy. The second modification on its own (the *60%-Break* voting strategy) was not tested in this round of experiments.

Finally, versions 3, 4 and 5 of the new methods are similar to the respective versions 2 of the original methods, and versions 1 and 2 of the new methods, respectively, except for they all use DDM and its new parametrization, which is our preferred drift detector and configuration for BOLE.

VI. RESULTS AND ACCURACY ANALYSIS

Tables I and II present the accuracies obtained for each of the seven variations based on ADOB and OZABOOST, respectively, and Table III gives the results of the other methods, all tested on the artificial and real-world datasets. In each dataset, the overall best result is written in **bold** and the best local results in the other tables are written in *italics*. Also, Rank$_{ALL}$ is the average of the rank positions that each

TABLE I

AVERAGE ACCURACIES IN PERCENTAGE (%), WITH 95% CONFIDENCE INTERVALS IN ARTIFICIAL DATASETS, FOR ADOB/BOLE USING DIFFERENT CONFIGURATIONS OF CONCEPT DRIFT DETECTOR, ITS PARAMETERS (IN THE SPECIFIC CASE OF ADWIN), AND VOTING STRATEGY.

	ADOB ADWIN$_{OLD}$ 50%-Break	ADOB$_2$ ADWIN 50%-Break	BOLE$_1$ ADWIN 50%-Cont.	BOLE$_2$ ADWIN 60%-Cont.	BOLE$_3$ DDM 50%-Break	BOLE$_4$ DDM 50%-Cont.	BOLE$_5$ DDM 60%-Cont.
Stag.$_{10D}$	98.43±0.04	98.54±0.04	98.53±0.04	98.52±0.04	98.96±0.04	98.96±0.04	**98.97±0.04**
Stag.$_{40D}$	90.05±0.26	93.48±0.20	93.48±0.20	93.45±0.19	96.91±0.07	96.90±0.07	**96.92±0.07**
Stag.$_{80D}$	71.39±0.52	79.51±0.75	79.63±0.74	80.03±0.73	93.88±0.14	94.00±0.13	**94.08±0.12**
Agr.$_{10D}$	77.38±0.53	77.92±0.49	78.01±0.48	*78.19±0.44*	77.68±0.24	77.69±0.24	77.87±0.27
Agr.$_{40D}$	67.44±0.42	69.52±0.35	69.61±0.32	69.79±0.32	71.98±0.31	71.98±0.31	**72.07±0.29**
Agr.$_{80D}$	63.80±0.39	66.30±0.43	66.30±0.43	66.32±0.45	69.12±0.34	69.12±0.34	**69.21±0.33**
Mix.$_{10D}$	84.16±0.26	84.42±0.29	84.46±0.27	84.27±0.28	85.79±0.24	**85.81±0.23**	85.71±0.22
Mix.$_{40D}$	55.12±0.57	62.11±1.02	63.03±0.78	63.31±0.79	76.53±0.52	**76.55±0.52**	76.36±0.53
Mix.$_{80D}$	50.88±0.25	50.93±0.28	50.78±0.29	50.72±0.31	64.79±0.82	*64.81±0.82*	64.78±0.84
Wave$_{10D}$	77.08±0.36	77.06±0.36	77.17±0.35	76.76±0.36	77.45±0.36	*77.69±0.31*	77.55±0.31
Wave$_{40D}$	72.26±0.76	72.61±0.78	72.92±0.74	71.52±0.71	76.83±0.56	*77.08±0.56*	77.01±0.55
Wave$_{80D}$	70.72±1.05	71.24±1.03	71.56±1.03	71.04±1.02	76.14±0.92	*76.36±0.92*	76.31±0.93
Electric.	88.64	89.32	89.34	89.25	89.91	89.93	**90.04**
Covert.	85.52	85.26	85.29	85.25	*90.04*	*90.04*	90.03
Pokerh.	53.03	52.92	53.65	53.03	50.71	**53.70**	53.10
Rank$_{\text{BOLE}}$	6.3000	5.1333	4.3333	5.3000	3.0000	**1.8666**	2.0666
Rank$_{ALL}$	12.8333	10.2666	9.0000	10.4333	4.7333	**2.7333**	3.2666

TABLE II

AVERAGE ACCURACIES IN PERCENTAGE (%), WITH 95% CONFIDENCE IN ARTIFICIAL DATASETS, FOR OZABOOST/OZABOLE, USING SIMILAR CONFIGURATIONS TO THOSE OF TABLE I.

	OZABOOST ADWIN$_{OLD}$ 50%-Break	OZABOOST$_2$ ADWIN 50%-Break	OZABOLE$_1$ ADWIN 50%-Cont.	OZABOLE$_2$ ADWIN 60%-Cont.	OZABOLE$_3$ DDM 50%-Break	OZABOLE$_4$ DDM 50%-Cont.	OZABOLE$_5$ DDM 60%-Cont.
Stag.$_{10D}$	97.51±0.92	97.61±0.92	98.48±0.06	98.51±0.05	98.02±0.92	98.89±0.05	*98.91±0.04*
Stag.$_{40D}$	89.84±0.49	92.60±0.44	93.08±0.19	93.24±0.17	96.30±0.42	96.78±0.08	*96.81±0.08*
Stag.$_{80D}$	71.55±0.61	81.15±0.61	81.98±0.58	82.36±0.59	93.00±0.21	93.74±0.13	*93.87±0.12*
Agr.$_{10D}$	76.35±0.46	77.24±0.46	77.22±0.47	77.29±0.43	*77.61±0.27*	77.56±0.27	77.58±0.22
Agr.$_{40D}$	67.64±0.37	69.64±0.32	69.63±0.33	69.74±0.29	*71.83±0.31*	71.81±0.31	71.78±0.30
Agr.$_{80D}$	63.63±0.30	66.08±0.39	66.06±0.38	65.99±0.38	*68.79±0.36*	*68.79±0.36*	68.75±0.36
Mix.$_{10D}$	83.47±0.31	83.55±0.34	83.59±0.35	83.52±0.29	85.51±0.27	*85.55±0.25*	84.92±0.26
Mix.$_{40D}$	55.30±0.51	63.17±0.81	63.30±0.84	63.84±0.87	76.05±0.49	*76.09±0.49*	75.76±0.50
Mix.$_{80D}$	50.73±0.21	50.63±0.27	50.70±0.25	50.75±0.29	65.14±0.75	*65.20±0.76*	65.12±0.77
Wave$_{10D}$	77.56±0.39	77.51±0.38	77.82±0.34	78.20±0.40	77.59±0.34	77.79±0.32	**78.21±0.33**
Wave$_{40D}$	71.19±0.91	71.61±0.90	72.77±0.83	72.59±0.90	76.66±0.58	77.02±0.53	**77.61±0.53**
Wave$_{80D}$	69.77±1.05	70.53±0.97	71.30±0.94	71.87±0.97	75.58±0.92	76.05±0.88	**76.78±0.91**
Electric.	88.45	88.88	88.92	89.47	89.24	89.31	*89.75*
Covert.	85.45	84.90	84.91	85.63	89.70	89.73	**90.16**
Pokerh.	53.07	53.07	*53.67*	53.63	52.43	52.96	52.29
Rank$_{OZ.}$	6.4333	5.7000	4.6666	3.8666	2.9666	2.2333	**2.1333**
Rank$_{ALL}$	14.0333	12.3666	10.6000	9.2666	6.8333	5.4333	4.7333

configuration achieved over the 15 datasets, considering all 17 tested configurations.

Note that ADWIN$_{OLD}$ with the *50%-Break* voting strategy corresponds to the methods' original configurations and the other ranks (Rank$_{\text{BOLE}}$ and Rank$_{OZ.}$) are the averages of the rank positions of each configuration within each table.

In all Stagger datasets, all the modifications improved the accuracy of both methods and their performances were close. The biggest increases came with OZABOLE whereas the best accuracies were achieved by BOLE. Also, the improvements were higher in the versions with more concept drifts.

In Agrawal$_{10D}$, the accuracies remained similar in all configurations of both methods. In the other two data sets, the accuracies have increased 2%-3% in the versions using ADWIN with the new parametrization, and another 2%-3% when using DDM. BOLE$_5$ achieved the best accuracies in these Agrawal datasets with more concept drifts.

In Mixed$_{10D}$, the accuracies in both methods were similar, being slightly higher when using DDM. In Mixed$_{40D}$, the results increased by approximately 8% with the new ADWIN parametrization, and another 11% to 13% with DDM. In Mixed$_{80D}$ the versions using ADWIN had similar results but the change to DDM increased the results by more than 14%. BOLE$_4$ was the best classifier in the first two datasets.

Similarly, the accuracies in Waveform with fewer drifts improved slightly when using the proposed configurations. In the other versions, the accuracies increased a little with the new ADWIN parametrization and voting strategies. The change to DDM improved the results in about 4%. Here, OZABOLE$_5$ was the best classifier in all three versions.

In the real-world datasets, the improvements were usually small and were not present in all combinations but, once again, the best result in each individual dataset was obtained by one of the modified ensembles.

TABLE III
AVERAGE ACCURACY IN PERCENTAGE (%) WITH 95% CONFIDENCE IN ARTIFICIAL DATASETS, FOR THE OTHER SELECTED ENSEMBLES.

	DDD	DWM	LEVBAG
Stagger_{10D}	95.24±0.29	*95.75*±0.34	90.77±0.32
Stagger_{40D}	*88.11±0.36*	86.46±0.44	81.46±0.32
Stagger_{80D}	76.22±0.31	*77.19±0.40*	72.26±0.43
Agrawal_{10D}	76.44±0.30	73.10±0.42	**80.07±0.56**
Agrawal_{40D}	69.87±0.73	67.30±0.34	*70.64±0.31*
Agrawal_{80D}	*66.60±0.47*	64.47±0.38	64.95±0.46
Mixed_{10D}	*83.94±0.44*	83.70±0.37	75.49±0.68
Mixed_{40D}	*74.14±0.52*	72.34±0.65	53.72±0.73
Mixed_{80D}	**65.84±0.71**	61.60±0.58	44.64±0.35
Wave_{10D}	*77.70±0.35*	72.59±0.43	76.99±0.50
Wave_{40D}	*76.18±0.62*	71.60±0.70	74.71±0.62
Wave_{80D}	*74.62±0.99*	69.18±0.95	72.61±0.96
Electricity	86.17	88.52	*89.71*
Covertype	83.86	87.00	*88.13*
Pokerhand	*52.97*	46.36	52.18
Rank_{ALL}	10.4666	14.0000	12.0000

One interesting and promising result was the fact that the higher error bound for the votes of the classifiers also improved results in binary datasets, e.g. in stagger and electricity. This was a somewhat surprising result.

Notice that, in the average ranks, all versions of BOLE have a better ranking than DDD, DWM, and LEVBAG. The same occurred with OZABOLE, except for OZABOLE_1. The best overall classifier was BOLE_4, closely followed by BOLE_5, and the worst were OZABOOST and DWM.

We also used a statistic named F_F [24], which is a variation of the Friedman test, and the post-hoc Nemenyi test to verify which of the classifiers are statistically superior to the others. We computed F_F three times: one to compare the seven versions of Table I, another for the seven versions of Table II, and a third time comparing all the 17 configurations, generating the three rankings.

To find out which methods are statistically superior, the Nemenyi-test uses a critical difference (CD) as a reference. The calculated CD value with 95% confidence for the comparison of all 17 versions was approximately 6.376. This means that each pair of methods with a ranking difference greater than 6.376 are statistically different.

BOLE_4 presented significant differences when compared to both versions of ADOB and OZABOOST, to BOLE_2, OZABOLE_1, and OZABOLE_2, as well as to DDD, DWM, and LEVBAG. BOLE_5 was also statistically superior to all these classifiers, except for OZABOLE_2. Notice that BOLE_4 and BOLE_5 were the only two configurations to statistically outperform DDD, and that BOLE_3, OZABOLE_5 and OZABOLE_4 followed just behind, with several significant differences.

Figure 1 graphically represents these results, but some configurations with intermediate results have been omitted to improve the presentation.

VII. CONCLUSION

This paper proposed different strategies aimed at increasing the accuracy of online boosting methods, particularly in scenarios where concept drifts are frequent and/or abrupt.

More specifically, we studied the effects of (a) lessening the precondition that controls which experts are allowed to vote and of (b) replacing the concept drift detection method that is often used within several online learning methods. In addition, we tried a more aggressive parametrization on these detectors, making them more sensitive to concept drifts, in spite of making them likely to raise more false alarms.

The results suggest that each of the proposed modifications are more effective than the others in different scenarios. In most cases, they contributed to improve the accuracies of both tested methods and, together, they statistically outperformed most other configurations in the tested datasets.

So, we claim the substitution of ADWIN for DDM, the different parametrization of both drift detectors, and the less strict application of the 50% precondition were all very successful, making BOLE_4 achieve the best ranking overall.

Although lowering the 50% bound to 60% (permitting more classifiers to vote) has not been as successful, most of the time it did not hurt the results either. In fact, OZABOLE_5 ended in front of OZABOLE_4 in the overall ranking and, despite the differences being small, BOLE_5 outperformed BOLE_4 in all six datasets with abrupt concept drifts. An interesting, and perhaps surprising, result is that the use of 60% as error bound also improved the results in binary datasets, for example in the stagger and electricity datasets.

An indication of their effectiveness was the remarkable results of the modified versions of both methods in several datasets, in particular the versions with more concept drifts and two of the real-world ones. For instance, in Stagger_{80D}, the differences to the other methods were enormous – up to 22.6%. In Covertype, the obtained accuracies are among the best previously reported. Additionally, the 90.03% accuracy obtained in the Electricity dataset is, to the best of our knowledge, the highest ever achieved by an online method in this dataset. Also, BOLE_4 and BOLE_5 were *statistically* more accurate than (a) the other three tested ensemble methods and (b) all configurations of the original methods ADOB and OZABOOST. Furthermore, the global performances of BOLE_3, OZABOLE_4, and OZABOLE_5 were also very solid and reasonably close to that of BOLE_4.

Note these heuristic strategies are general and applicable to other variations of online boosting. So, we might investigate other methods such as Online Coordinate Boosting (OCBOOST) [25], Fast and Light Classifier (FLC) [26], Online Non-Stationary Boosting (ONSB) [27], etc. and select some of them to implement and further test our ideas.

The impact of the parametrization of the drift detectors in the final accuracy over different kinds of datasets could also be the subject of investigation and, so, it is possible that these obtained accuracies could be improved further.

Another possible direction of investigation would be to try some other values as percentage limits for permitting the

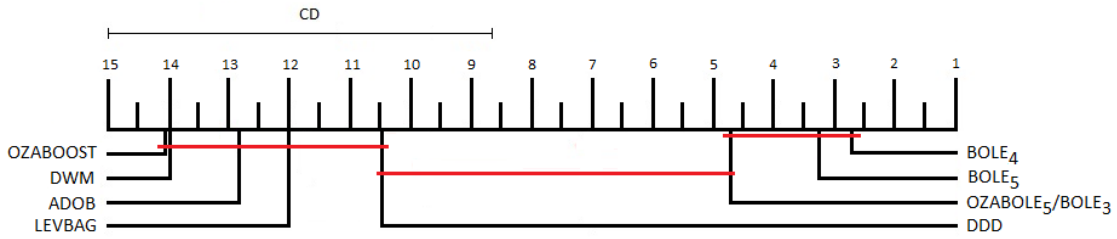

Fig. 1. Comparison results using the Nemenyi test with 95% confidence: groups of classifiers that are not significantly different are connected.

classifiers to vote. It might be that the best choice depends on how hard the problem is, for example, on the number of classes.

Also, we have not thoroughly analysed the impact of changing the 50% bound and of shifting the values of the classifiers' weights in the distribution of diversity among the experts and in the final accuracies. Other functions could also be considered to prevent the use of negative weights.

Finally, it is important to explicitly say we used an empirical approach that, to some extent, could be seen as an experimental exploration of the algorithmic solution space which might lead to provably better boosting algorithms in the future. In addition, both BOLE and OZABOLE were implemented in the MOA framework and their codes will soon be freely available.

ACKNOWLEDGEMENTS

Silas Santos is a Ph.D. student supported by a CNPq postgraduate grant.

REFERENCES

[1] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, pp. 44:1–37, 2014.

[2] P. M. Gonçalves, Jr. and R. S. M. Barros, "RCD: A recurring concept drift framework," *Pattern Recognition Letters*, vol. 34, no. 9, pp. 1018–1025, 2013.

[3] Y. Freund, "Boosting a weak learning algorithm by majority," *Information and Computation*, vol. 121, no. 2, pp. 256–285, 1995.

[4] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *International Conference on Machine Learning*, vol. 96, 1996, pp. 148–156.

[5] N. C. Oza and S. Russell, "Online bagging and boosting," in *Artificial Intelligence and Statistics*. Morgan Kaufmann, 2001, pp. 105–112.

[6] S. G. T. C. Santos, P. M. Gonçalves, Jr., G. D. S. Silva, and R. S. M. Barros, "Speeding up recovery from concept drifts," in *Machine Learning and Knowledge Discovery in Databases*, ser. LNCS. Springer, 2014, vol. 8726, pp. 179–194.

[7] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 7th SIAM International Conference on Data Mining*, ser. SDM '07, Lake Buena Vista, FL, USA, 2007, pp. 443–448.

[8] P. M. Gonçalves, Jr., S. G. T. C. Santos, R. S. M. Barros, and D. C. L. Vieira, "A comparative study on concept drift detectors," *Expert Systems with Applications*, vol. 41, no. 18, pp. 8144–8156, 2014.

[9] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence – SBIA 2004*, ser. LNCS. Springer, 2004, vol. 3171, pp. 286–295.

[10] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New ensemble methods for evolving data streams," in *Proceedings of 15th ACM International Conference on Knowledge Discovery and Data Mining*, New York, 2009, pp. 139–148.

[11] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Machine Learning and Knowledge Discovery in Databases*, ser. LNCS. Springer, 2010, vol. 6321, pp. 135–150.

[12] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, 2007.

[13] A. Blum, "Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain," *Machine Learning*, vol. 26, no. 1, pp. 5–23, 1997.

[14] L. L. Minku, A. P. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 5, pp. 730–742, 2010.

[15] L. L. Minku and X. Yao, "DDD: A new ensemble approach for dealing with concept drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 4, pp. 619–633, 2012.

[16] M. Baena-Garcia, J. Del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, "Early drift detection method," in *International Workshop on Knowledge Discovery from Data Streams*, ser. IWKDDS '06, 2006, pp. 77–86.

[17] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive online analysis," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[18] S. Chen, H. Lin, and C. Lu, "An online boosting algorithm with theoretical justifications," in *Proceedings of the 29th International Conference on Machine Learning*, ser. ICML'12, 2012.

[19] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Machine Learning*, vol. 1, no. 3, pp. 317–354, 1986.

[20] R. Agrawal, T. Imielinski, and A. N. Swami, "Database mining: a performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.

[21] B. I. F. Maciel, S. G. T. C. Santos, and R. S. M. Barros, "A lightweight concept drift detection ensemble," in *Proceedings of 27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, Vietri sul Mare, Italy, 2015, pp. 1061–1068. [Online]. Available: http://dx.doi.org/10.1109/ICTAI.2015.151

[22] S. G. T. C. Santos, R. S. M. Barros, and P. M. G. Jr, "Optimizing the parameters of drift detection methods using a genetic algorithm," in *Proceedings of 27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, Vietri sul Mare, Italy, 2015, pp. 1077–1084. [Online]. Available: http://dx.doi.org/10.1109/ICTAI.2015.153

[23] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01, New York, USA, 2001, pp. 97–106.

[24] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[25] R. Pelossof, M. Jones, I. Vovsha, and C. Rudin, "Online coordinate boosting," in *Computer Vision Workshops (ICCV Workshops)*. IEEE, 2009, pp. 1354–1361.

[26] V. Attar, P. Sinha, and K. Wankhade, "A fast and light classifier for data streams," *Evolving Systems*, vol. 1, no. 3, pp. 199–207, 2010.

[27] A. Pocock, P. Yiapanis, J. Singer, M. Luján, and G. Brown, "Online non-stationary boosting," in *Multiple Classifier Systems*, ser. LNCS. Springer, 2010, vol. 5995, pp. 205–214.