# Online Non-Stationary Boosting

Adam Pocock, Paraskevas Yiapanis, Jeremy Singer,
Mikel Luján, and Gavin Brown

School of Computer Science, University of Manchester, UK
{apocock,pyiapanis,jsinger,mlujan,gbrown}@cs.manchester.ac.uk

**Abstract.** Oza's Online Boosting algorithm provides a version of AdaBoost which can be trained in an online way for stationary problems. One perspective is that this enables the power of the boosting framework to be applied to datasets which are too large to fit into memory. The online boosting algorithm assumes the data distribution to be independent and identically distributed (i.i.d.) and therefore has no provision for concept drift. We present an algorithm called Online Non-Stationary Boosting (ONSBoost) that, like Online Boosting, uses a static ensemble size without generating new members each time new examples are presented, and also adapts to a changing data distribution. We evaluate the new algorithm against Online Boosting, using the STAGGER dataset and three challenging datasets derived from a learning problem inside a parallelising virtual machine. We find that the new algorithm provides equivalent performance on the STAGGER dataset and an improvement of up to 3% on the parallelisation datasets.

## 1 Introduction

Many real-world problems change their characteristics over time. This is known as learning in non-stationary environments, where the function which maps from the inputs to the outputs changes over time [4, 5, 7, 10]. This change is known as *concept drift*. Ensemble frameworks provide a modular style where components can be added, removed or modified which is particularly useful when tracking changing concepts. The aim of this work is to demonstrate a novel online learning algorithm, capable of being deployed in a *real-time* environment, requiring online learning with thousands of examples per second. The domain we work with is that of *adaptive Java compilers*, which dynamically optimise how Java code is executed on modern *multi-core* computers.

We are investigating the application of ML techniques to *automatic parallelisation* problems, running inside a Java virtual machine. Automatic parallelisation problems have several interesting characteristics that constrain the space of applicable learning algorithms. Training data is taken from runtime analysis of Java benchmarks, recording properties of the machine state at the start/end of each method execution. The prediction task is, given the machine state, and the source code for a new Java method, *should this method be executed in parallel?* This is an important task, as parallel execution can potentially fail if it causes a

resource conflict with other currently running methods; and solving this problem well can potentially halve the runtime of a piece of Java code.

Since decisions are taken at the start of every method execution, this results in a dataset generating potentially millions of examples *per second*, rendering many standard offline training algorithms infeasible. The exemplar of this is our DaCapo Bloat data [1], which generates $1,273,359$ datapoints in 386 milliseconds of runtime. Predictions required by the parallelisation system therefore need to be generated on the order of microseconds otherwise the parallelisation opportunity is lost in the overhead created by the ML system. Additionally, a training cycle must not 'lock' the ML system preventing it from providing a prediction. A further problem is that the data is not independent and identically distributed, as decisions made by the parallelisation system using ML predictions will alter the future behaviour of the system. In summary we require a ML system which:

- Learns in a 'true' online fashion[1].
- Can adapt to a changing distribution.
- Can generate predictions quickly.
- Can be updated without preventing the generation of predictions.

We propose a system based Oza's Online Boosting [8], with modifications to enable the updating of ensemble members, adapting the system to changing data. Online Boosting meets the requirements using simple base models which can generate predictions quickly, and while each ensemble member is being trained, the remainder of the ensemble members could be used to generate a prediction.

The layout of this paper is as follows: Section 2 provides a description of the related literature. Section 3 provides a description of our algorithm and how it relates to the literature. Section 4 provides our experimental setup and testing results, and Section 5 contains the conclusions and future directions for our algorithm.

## 2 Related Work

Learn$^{++}$.NSE [2, 7] provides an algorithm for generating a boosted classifier on streaming data. It generates a series of classifiers using batches of examples, by converting the online datastream into a series of chunks of a fixed size. At each time step one new classifier is trained on a batch of new examples, using an example weighting distribution similar to AdaBoost based upon the performance of the current ensemble, then all the ensemble members are reweighted according to their performance on the current batch of examples. After each classifier has been trained it becomes immutable, though its weight in the majority vote may change dependent on its current performance.

Knowledge-based Sampling Stream (KBS-Stream) [10] is a boosting algorithm similar to Learn$^{++}$.NSE, as it generates a series of classifiers by creating

[1] By 'true' online, we mean a system which learns from and then discards each training example one-by-one.

batches of examples from an online datastream. A key difference from Learn$^{++}$ is the classifier weighting scheme, based on a probabilistic correlation measure. Another important difference is the way it makes use of new data: in Learn$^{++}$, a new classifier trained every $K$ examples, whereas in KBS-Stream, a new classifier is only trained if the data distribution is deemed to have drifted.

It is important to note that both Learn$^{++}$.NSE and KBS-Stream assume that data in each batch are independent and identically distributed. In our parallelisation problem the distribution is unknown, and thus no such guarantee can be made. This leads to a problem if we were to apply these algorithms to the parallelisation datasets described in Section 4.1, as each new example would hypothetically require the generation of a new classifier. In contrast to such 'batching' algorithms are algorithms which update the classifiers on presentation of each example, which we term 'true' online learning algorithms.

Online boosting developed by Oza [8] provides a boosting algorithm which mimics the sampling with replacement variant of AdaBoost [3] when applied to an online stream of examples. Each example is presented to an ensemble member $r$ times, where $r$ is drawn from a Poisson($\lambda$) where $\lambda$ is a weight derived from previous performance on that example. Effectively, if the example is misclassified by an earlier classifier it is presented more often to classifiers later on in the ensemble. This assumes that the ensemble members are capable of learning incrementally and that repeated training on the same example will have a cumulative effect on the classifier. The ensemble is initialised with a fixed number of members which remain throughout the training process. The algorithm is proven to return the same ensemble as AdaBoost in the limit of infinite examples, when using a Naive Bayes classifier as the weak learning algorithm. At each stage the algorithm is approximating the performance of AdaBoost trained upon the same examples, which limits the performance when the distribution is changing because offline classification tasks are not subject to concept drift.

AdaBoost performs a greedy forward search in the space of classifiers. Alternative search methods which are less greedy can be fitted in the place of this forward search. One such algorithm is FloatBoost [6] which incorporates a sequential forward floating search [9] in place of the greedy forward search. This enables the removal of classifiers which are hindering the performance of the ensemble or are made redundant by a combination of other ensemble members. Like AdaBoost, it is an offline binary classification algorithm. The sequential forward floating search increases the runtime of the algorithm significantly whilst providing an increase in the accuracy.

The properties of the various online algorithms described in this section are summarised in Table 1.

## 3 Non-Stationary Boosting

We present an algorithm that, like Online Boosting, uses a static ensemble size without generating new members each time new examples are presented, and also adapts to a changing data distribution. It is based upon a combination of

| Algorithm | "True" Online | Fixed Ensemble Size | Concept Drift |
|---|---|---|---|
| Learn$^{++}$.NSE | ✗ | ✗ | ✔ |
| KBS-Stream | ✗ | ✗ | ✔ |
| Online Boosting | ✔ | ✔ | ✗ |
| ONSBoost | ✔ | ✔ | ✔ |

**Table 1.** Comparison of related algorithms and ONSBoost. By "True Online" we mean that the algorithm can be trained on single examples, without chunking them and assuming i.i.d. within the batch.

FloatBoost [6] and Online Boosting [8] which we call Online Non-Stationary Boosting, or ONSBoost. It incorporates a floating search into the Online Boosting algorithm, which enables the addition of new classifiers and the removal of poorly performing classifiers. This lets the algorithm follow a changing data distribution.

The algorithm, ONSBoost, provides a simple extension of Online Boosting to allow the resetting of outdated and inaccurate classifiers. The key parameters are: $K$, determining how often the classifiers are checked to see if a reset is necessary; $W$, the size of the window used to determine if a reset is necessary; and $P$, a "protection" period, where a classifier cannot be reset. The protection period is necessary to allow a classifier sufficient training examples to learn the new concept.

The algorithm reduces to Oza's Online Boosting in the case when $K$ is greater than the number of examples in the training dataset[2]. When $K$ is less than the number of examples $N$ there are $\lfloor \frac{N}{K} \rfloor$ classifier removal steps.

### Window Size

In a true online environment the base assumption is that there is an infinite stream of examples being generated for training and classification. This means it is not feasible to store all the previous examples and use those to determine performance. A common technique for evaluating online classifiers is to use a window of the most recent examples to determine the current performance [4]. This can also improve the accuracy on data with concept drift as it forces the classifier to adapt to the most recent data. In common with these other techniques we use a window to determine the current performance of the ensemble and to decide which, if any, members need replacing to improve the ensemble.

### Update Period

A parameter is introduced to control how often a search is performed of the classifiers to check if the ensemble performance is being hindered. Ideally this step would be performed after each example has been presented for training, however this introduces problems. The backwards search requires $n \times j$ evaluations, where $n$ is the window size and $j$ is the number of classifiers which makes it very computationally intensive compared to training. Thus performing a search after each example would greatly increase the time complexity of each training cycle.

[2] Note that in our current implementation we have used a modified pseudo-count method for error estimates

This parameter effectively controls how quickly poorly performing classifiers are replaced, and thus has an effect on the speed with which the algorithm adapts to concept drift. The searches are then limited so they are only performed after the algorithm has seen $K$ new examples. For example, if the parameter is set to 50, then each time 50 examples have been processed then a backwards search for poorly performing classifiers will be executed.

**Protecting New Classifiers**
One further parameter is added to the system, which is necessitated by the fact that newly created classifiers will perform poorly until they have been trained on a sufficient number of examples. They are "protected" from removal by the backwards search until they have been sufficiently trained. This is not a problem in an offline algorithm as each new classifier is trained on the whole dataset. It is parameterised because the amount of time a classifier needs to train to a given standard on a dataset is in general unknown.

**How to place classifiers?**
A subtle point in all boosting algorithms, but particularly in Oza's Online Boosting is that the classifier ordering matters whilst the system is being trained. In Online Boosting each classifier is implicitly dependent on the output of all classifiers before it, as they are trained in a fixed sequential order. The performance of a previous classifier dictates the number of times that the current classifier is trained on any given example. To modify Online Boosting to allow for the replacement of classifiers thought needs to be given to the placement of the new classifier.

The current algorithm places the reset classifier at the end of the ensemble, and the alternative is in place replacement, where the reset classifier directly replaces the old one. The former makes the $(i+1)$th classifier adapt to the distribution of examples that the $i$th classifier was learning when the $i$th classifier is replaced. In the latter the new $i$th classifier will initially have poor performance and thus make all examples have higher weights for any classifiers which are sequentially after the $i$th classifier. The choice between these two behaviours is interesting, but not explored in this paper. The two different methods are shown in Figure 1.

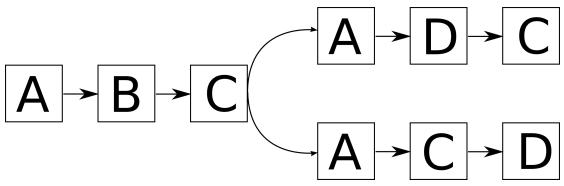



**Fig. 1.** Classifier placement. If classifier B is replaced with a fresh classifier D, D can be put in two different places.

**Algorithm 1** ONSBoost

Variables: $H$ = the ensemble, $h$ is an ensemble member, $\epsilon(H)$ is the ensemble error on the window
Parameters: $M$ = number of classifiers, $K$ = update period, $W$ = window size $P$ = number of examples to "protect" a new classifier
Initialisation: $\forall m \in \{1, 2, ..., M\}, \lambda_m^{sc} = 0, \lambda_m^{sw} = 0, k = 0$

**ONSBoost** ($H$, train,$(x, y)$)
**PHASE I: Oza's Online Boosting**
Set the current example's "weight" $\lambda = 1$
Increment the example counter $k = k + 1$
**for all** $h_m, (m \in \{1, 2, ..., M\})$ in $H$ **do**
  Set $r$ sampled from $Poisson(\lambda)$
  **for** $i = 0, i < r$ **do**
    $h_m \leftarrow \text{train}(h_m, (x, y))$
  **end for**
  **if** $y = h_m(x)$ **then**
    $\lambda_m^{sc} \leftarrow \lambda_m^{sc} + \lambda$
    $\epsilon_m \leftarrow \frac{\lambda_m^{sw}+1}{\lambda_m^{sc}+\lambda_m^{sw}+1}$
    $\lambda \leftarrow \lambda(\frac{1}{2(1-\epsilon_m)})$
  **else**
    $\lambda_m^{sw} \leftarrow \lambda_m^{sw} + \lambda$
    $\epsilon_m \leftarrow \frac{\lambda_m^{sw}+1}{\lambda_m^{sc}+\lambda_m^{sw}+1}$
    $\lambda \leftarrow \lambda(\frac{1}{2\epsilon_m})$
  **end if**
**end for**

**PHASE II: ONSBoost modification**
Move window along datastream
**if** $k = K$ **then**
  $H_P$ = all classifiers which have trained on at least $P$ examples
  $h' = \arg\max_{h \in H_P} \epsilon(H - h)$
  **if** $\epsilon(H - h') < \epsilon(H)$ **then**
    Remove $h'$ from $H$, and add a new $h$ to the end of $H$
    Set $\lambda_h^{sc} = 0, \lambda_h^{sw} = 0, \epsilon_h = 0$
  **end if**
  $k \leftarrow 0$
**end if**

**return** $H(x) \leftarrow \arg\max_{c \in Y} \sum_{m:h_m(x)=c} \log \frac{1-\epsilon_m}{\epsilon_m}$

## 4 Experimental Results

An obvious question is "how should we fix the parameters $K$, $W$ and $P$?" In this section we empirically evaluate the proposed algorithm with particular emphasis on characterising the parameter space.

### 4.1 Datasets

We use 4 datasets in the presented experiments, STAGGER and 3 datasets taken from an automatic parallelisation problem which have concept drift and thousands or millions of examples.

The parallelisation datasets all use a set of features derived from an offline inspection of Java bytecode. The Java applications are taken from the DaCapo benchmarks suite [1]. The datasets are described in detail in [11]. Each dataset comprises a set of method features, and if the method was successfully executed in parallel with its parent method. The class concept is subject to concept drift as the underlying virtual machine state is not fully captured in the feature set. This is a *hidden context* [4] which can be subject to gradual, cyclical, and abrupt forms of concept drift. Abrupt and cyclical behaviour can be generated as the feature set does not include information on all currently executing methods, and these can cause conflicts which cause the parallel execution to fail. Gradual behaviour can be generated as variables in the application change over time and cause different memory access patterns, which affect the parallel execution.

The STAGGER dataset is described in [12]. A standard methodology with this dataset is to generate 100 test examples sampled from the current distribution which are used to test the performance of any given algorithm. This does not reflect a true online learning scenario as there is not generally an opportunity to sample a testing set from the same distribution As a result dataset size was increased to 600 to improve the measure of accuracy, with 1/3 still taken from each of the three concepts. The format and features of the datasets are described in Table 2.

| Dataset | Examples | Features | Feature Type | Class Skew |
|---|---|---|---|---|
| bloat | 1,273,359 | 38 | Binary | 7% +ve |
| antlr | 169,578 | 38 | Binary | 22% +ve |
| pmd | 38,994 | 38 | Binary | 48% +ve |
| STAGGER | 600 | 3 | Ternary | 29% +ve |

**Table 2.** Dataset Properties

### 4.2 Comparison With Online Boosting

Due to the problems Learn$^{++}$.NSE and KBS-Stream have with data which is not i.i.d. at the example level, we compare the new algorithm with Online Boosting (without priming). In all cases the base learner used is a categorical Naive Bayes with pseudocounts. The number of classifiers was kept constant at 30 when using

both ONSBoost and Online Boosting. Each experiment was repeated 10 times, to eliminate some of the randomness inherent in the Poisson distribution used in both algorithms. Results on STAGGER are presented, with additional results using datasets collected by our automatic parallelisation framework. Each experiment tests the online accuracy of a classifier. Online accuracy is the percentage of the training data that the classifier predicts correctly, before it has been trained on that example. The value are given with 95% confidence intervals.

| Dataset | Online Boosting | ONSBoost | ONSBoost Parameters |
|---|---|---|---|
| antlr | 80.89% ± 0.23 | **81.92% ± 0.05** | $K = 200, W = 50$ |
| bloat | 89.32% ± 0.85 | **91.93% ± 0.03** | $K = 200, W = 50$ |
| pmd | 76.19% ± 0.18 | **78.10% ± 0.18** | $K = 200, W = 50$ |
| STAGGER | 95.97% ± 0.36 | **96.10% ± 0.36** | $K = 10, W = 10$ |

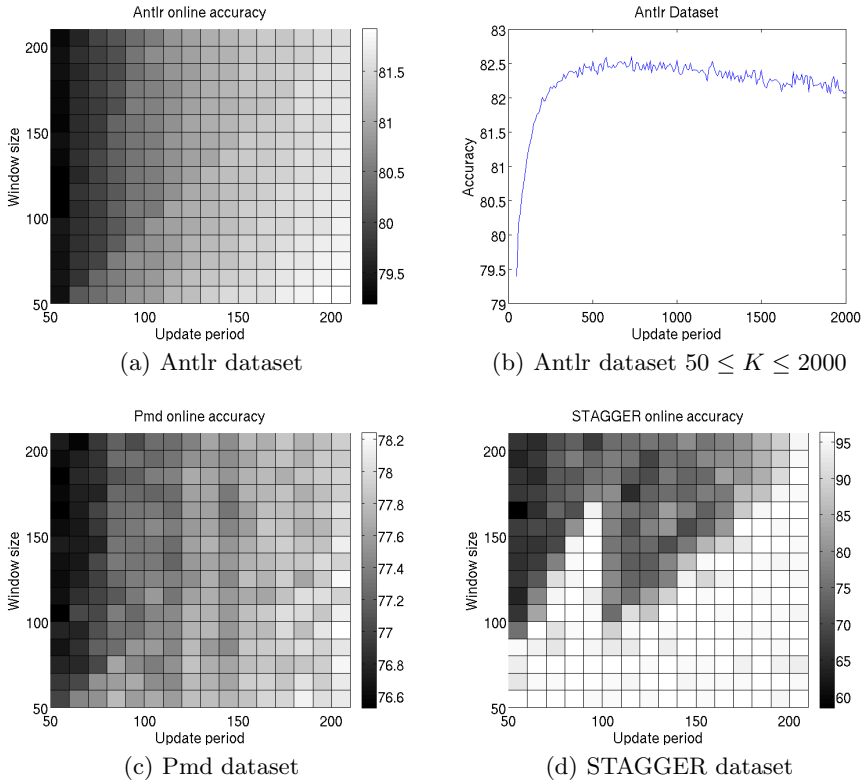
**Table 3.** Online accuracy comparison between ONSBoosting and Online Boosting.

The parameters for the comparison with Online Boosting were chosen as these give the best result across the most data. This can be seen in the parameter exploration in Section 4.3. With the STAGGER dataset an Update Period of 200 examples means that the algorithm considers classifiers for removal based upon a concept which changes with a hard boundary at the example immediately afterwards. For this reason a small update period was chosen, with a consequently smaller window size. In these experiments we perform better than Online Boosting in the parallelisation datasets, and equivalently in the STAGGER dataset. Even small improvements in accuracy are important for our task, as each incorrect prediction has associated costs, and a 1% improvement in accuracy results in an extra 12,000 correctly classified examples in the bloat dataset.

### 4.3 Exploration of Parameter Space

We now vary the parameters of ONSBoost to see how sensitive the performance is to parameter choice. Results are presented in Figure 2 varying the update period ($K$) and window size ($W$) parameters of ONSBoost. The number of examples to protect a new classifier ($P$) was fixed at 100 for all experiments. Figure 2(b) shows the accuracy for one line of the heat maps, fixing the window size to 50, and varying the update period from 50 to 2000 in steps of 10. This shows how the performance increases as this parameter increases before decreasing again once past a data dependent threshold, and the performance will converge to the performance of Online Boosting when the update period is equal to the size of the dataset.

From the exploration of the parameter space it appears that small window sizes and a relatively high number of steps before a backward search are the best parameters, as these consistently provide the highest accuracies. Using a window size which is much greater than the number of steps before a backwards search causes a decrease in performance, as new classifiers are penalised by the window as they have not been trained upon the examples it contains.




(a) Antlr dataset

(b) Antlr dataset $50 \leq K \leq 2000$

(c) Pmd dataset

(d) STAGGER dataset

**Fig. 2.** Online accuracy results for ONSBoosting.

## 5 Conclusion & Future Work

We presented an algorithm capable of dealing with continuous concept drift in a resource constrained environment over millions of examples. The algorithm is motivated by the need to develop a fast online learning algorithm for an automatic parallelisation problem, which imposes constraints on the types of algorithms which can be used. The algorithm replaces classifiers based upon their impact on the ensemble's performance rather than simply removing the oldest classifier in the ensemble, so it can cope with some measure of cyclic behaviour in the concept drift. We compared this new algorithm to Online Boosting, and found an improvement in performance on our automatic parallelisation problem, and comparable performance on a standard problem. The key idea is to maintain a fixed number of classifiers which are updated online, and to replace classifiers if they are negatively impacting the performance of the ensemble.

Our algorithm is based upon Online Boosting but will not converge to offline AdaBoost given the limit of infinite training examples. This is because the data is assumed to be i.i.d. in AdaBoost. Concept drift data is not i.i.d. and thus

Online Boosting is a sub-optimal choice of learning algorithm in a concept drift environment. ONSBoost provides a way to cope with concept drift in streaming data while maintaining the useful properties of Online Boosting, namely the ability to deal with incremental learning of streaming data, and the fixed number of classifiers, and thus fixed memory usage.

An area of further research is to develop a system to enable ONSBoost to cope better with cyclical drift. Learn$^{++}$.NSE can turn off classifiers based upon their current performance, but keep them for reuse later. A way of mimicking this ability in the ONSBoost system would be to select new ensemble members from a pool which included all previously removed classifiers, and a new ensemble member with no prior knowledge of the system. Other possible extensions include using a variable number of classifiers to decrease the ensemble size when the data is simple to classify and to increase the ensemble size when the data is difficult to classify.

## References


1. Blackburn, S.M., et al.: The DaCapo benchmarks: Java benchmarking development and analysis. In: OOPSLA '06: Proc. of the 21st annual ACM SIGPLAN conference on Object-Oriented Programing, Systems, Languages, and Applications. pp. 169–190. ACM Press, New York, NY, USA (Oct 2006)
2. Elwell, R., Polikar, R.: Incremental Learning of Variable Rate Concept Drift. In: Proc. of the 8th International Workshop on Multiple Classifier Systems. p. 151. Springer (2009)
3. Freund, Y., Schapire, R.: Experiments with a new boosting algorithm. In: Machine Learning: Proc. of the Thirteenth International Conference. pp. 148–156 (1996)
4. Kuncheva, L.: Classifier Ensembles for Changing Environments. In: Proc. of the 5th International Workshop on Multiple classifier systems (MCS 2004). pp. 1–15. Springer-Verlag New York Inc (2004)
5. Kuncheva, L.: Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. In: Proc. of the 2nd Workshop SUEMA 2008 (ECAI 2008) (2008)
6. Li, S., Zhang, Z., Shum, H., Zhang, H.: FloatBoost learning for classification. Advances in Neural Information Processing Systems pp. 1017–1024 (2003)
7. Muhlbaier, M., Polikar, R.: An ensemble approach for incremental learning in nonstationary environments. Lecture Notes in Computer Science 4472, 490 (2007)
8. Oza, N.C.: Online Ensemble Learning. Ph.D. thesis, The University of California, Berkeley, CA (Sep 2001)
9. Pudil, P., Novoviová, J., Kittler, J.: Floating search methods in feature selection. Pattern recognition letters 15(11), 1119–1125 (1994)
10. Scholz, M., Klinkenberg, R.: Boosting classifiers for drifting concepts. Intelligent Data Analysis 11(1), 3–28 (2007)
11. Singer, J., Pocock, A., Yiapanis, P., Brown, G., Luján, M.: Fundamental Nano-Patterns to Characterize and Classify Java Methods. In: Proc. Workshop on Language Descriptions, Tools and Applications (2009)
12. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine learning 23(1), 69–101 (1996)