Improving Adaptive Bagging Methods for Evolving Data Streams

Abstract. We propose two new improvements for bagging methods on evolving data streams. Recently, two new variants of Bagging were proposed [5] : `ADWIN` Bagging and Adaptive-Size Hoeffding Tree (ASHT) Bagging. ASHT Bagging uses trees of different sizes, and `ADWIN` Bagging uses `ADWIN` as a change detector to decide when to discard underperforming ensemble members. We improve `ADWIN` Bagging using Hoeffding Adaptive Trees, trees that can adaptively learn from data streams that change over time. To speed up the time for adapting to change of Adaptive-Size Hoeffding Tree (ASHT) Bagging, we add an error change detector for each classifier. We test our improvements by performing an evaluation study on synthetic and real-world datasets comprising up to ten million examples.

1 Introduction

Data streams pose several challenges on data mining algorithm design. First, algorithms must make use of limited resources (time and memory). Second, by necessity they must deal with data whose nature or distribution changes over time. In turn, dealing with time-changing data requires strategies for detecting and quantifying change, forgetting stale examples, and for model revision. Fairly generic strategies exist for detecting change and deciding when examples are no longer relevant. Model revision strategies, on the other hand, are in most cases method-specific.

The following constraints apply in the Data Stream model:

1. Data arrives as a potentially infinite sequence. Thus, it is impossible to store it all. Therefore, only a small summary can be computed and stored.
2. The speed of arrival of data is fast, so each particular element has to be processed essentially in real time, and then discarded.
3. The distribution generating the items may change over time. Thus, data from the past may become irrelevant (or even harmful) for the current prediction.

Under these constraints the main properties of an ideal classification method are the following: high accuracy and fast adaption to change, low computational cost in both space and time, theoretical performance guarantees, and a minimal number of parameters.

Ensemble methods are combinations of several models whose individual predictions are combined in some manner (for example, by averaging or voting) to form a final prediction. Often, ensemble learning classifiers provide superior predictive performance and they are easier to scale and parallelize than single classifier methods.

In [5] two new state-of-the-art bagging methods were presented: ASHT Bagging using trees of different sizes, and `ADWIN` Bagging using a change detector to decide

when to discard underperforming ensemble members. This paper improves on ASHT Bagging by speeding up the time taken to adapt to changes in the distribution generating the stream. It improves on ADWIN Bagging by employing Hoeffding Adaptive Trees, trees that can adaptively learn from evolving data streams. The paper is structured as follows: the state-of-the-art Bagging methods are presented in Section 2. Improvements to these methods are presented in Section 3. An experimental evaluation is conducted in Section 4. Finally, conclusions and suggested items for future work are presented in Section 5.

2 Previous Work
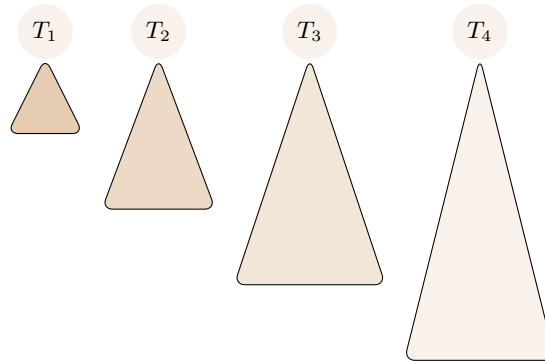
2.1 Bagging using trees of different size

Fig. 1. An ensemble of trees of different size

In [5], a new method of bagging was presented using Hoeffding Trees of different sizes. A *Hoeffding tree* [8] is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams, assuming that the distribution generating examples does not change over time. Hoeffding trees exploit the fact that a small sample can often be enough to choose an optimal splitting attribute. This idea is supported mathematically by the Hoeffding bound, which quantifies the number of observations (in our case, examples) needed to estimate some statistics within a prescribed precision (in our case, the goodness of an attribute). More precisely, the Hoeffding bound states that with probability $1 - \delta$, the true mean of a random variable of range R will not differ from the estimated mean after n independent observations by more than:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}.$$

A theoretically appealing feature of Hoeffding Trees not shared by other incremental decision tree learners is that it has sound guarantees of performance. Using the Hoeffd-

ing bound one can show that its output is asymptotically nearly identical to that of a non-incremental learner using infinitely many examples. See [8] for details.

The Adaptive-Size Hoeffding Tree (ASHT) is derived from the Hoeffding Tree algorithm with the following differences:

– it has a value for the maximum number of split nodes, or *size*
– after one node splits, if the number of split nodes of the ASHT tree is higher than the maximum value, then it deletes some nodes to reduce its size

The intuition behind this method is as follows: smaller trees adapt more quickly to changes, and larger trees perform better during periods with little or no change, simply because they were built on more data. Trees limited to size s will be reset about twice as often as trees with a size limit of $2s$. This creates a set of different reset-speeds for an ensemble of such trees, and therefore a subset of trees that are a good approximation for the current rate of change. It is important to note that resets will happen all the time, even for stationary datasets, but this behaviour should not have a negative impact on the ensemble's predictive performance.

When the tree size exceeds the maximun size value, there are two different delete options:

– delete the oldest node, the root, and all of its children except the one where the split has been made. After that, the root of the child not deleted becomes the new root.
– delete all the nodes of the tree, that is, restart from a new root.

In [5] a new bagging method was presented that uses these Adaptive-Size Hoeffding Trees and that sets the size for each tree. The maximum allowed size for the n-th ASHT tree is twice the maximum allowed size for the $(n-1)$-th tree. Moreover, each tree has a weight proportional to the inverse of the square of its error, and it monitors its error with an exponential weighted moving average (EWMA) with $\alpha = .01$. The size of the first tree is 2.

With this new method, the authors attempted to improve bagging performance by increasing tree diversity. It has been observed [15] that boosting tends to produce a more diverse set of classifiers than bagging, and this has been cited as a factor in increased performance.

2.2   Bagging using `ADWIN`

`ADWIN` [3] is a change detector and estimator that solves in a well-specified way the problem of tracking the average of a stream of bits or real-valued numbers. `ADWIN` keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window".

More precisely, an older fragment of the window is dropped if and only if there is enough evidence that its average value differs from that of the rest of the window. This has two consequences: first, that change is reliably declared whenever the window shrinks; and second, that at any time the average over the existing window can be reliably taken as an estimate of the current average in the stream (barring a very small

or very recent change that is still not statistically visible). A formal and quantitative statement of these two points (in the form of a theorem) appears in [3].

`ADWIN` is parameter- and assumption-free in the sense that it automatically detects and adapts to the current rate of change. Its only parameter is a confidence bound δ, indicating how confident we want to be in the algorithm's output, a property inherent to all algorithms dealing with random processes.

Also important for our purposes, `ADWIN` does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique. This means that it keeps a window of length W using only $O(\log W)$ memory and $O(\log W)$ processing time per item.

Bagging using `ADWIN` is implemented as `ADWIN` Bagging where the Bagging method is the online bagging method of Oza and Rusell [17] with the addition of the `ADWIN` algorithm as a change detector. When a change is detected, the worst classifier of the ensemble of classifiers is removed and a new classifier is added to the ensemble.

3   Improvements for Adaptive Bagging Methods

In this section we propose two new improvements for the adaptive bagging methods explained in the previous section.

3.1   `ADWIN` **bagging using Hoeffding Adaptive Trees**

The basic idea is to use adaptive Hoeffding trees instead of non-adaptive Hoeffding trees as the base classifier for the bagging ensemble method. We use the Hoeffding Adaptive Trees proposed in [4], where a new method for managing alternate trees is proposed. The general idea is simple: we place `ADWIN` instances at every node that will raise an alert whenever something worth attention happens at the node.

We use the variant of the *Hoeffding Adaptive Tree* algorithm (HAT for short) that uses `ADWIN` as a change detector. It uses one instance of `ADWIN` in each node, as a change detector, to monitor the classification error rate at that node. A significant increase in that rate indicates that the data is changing with respect to the time at which the subtree was created. This was the approach used by Gama *et al.* in [10], using another change detector.

When any instance of `ADWIN` at a node detects change, we create a new alternate tree without splitting any attribute. Using two `ADWIN` instances at every node, we monitor the average error of the subtree rooted at this node and the average error of the new alternate subtree. When there is enough evidence (as witnessed by `ADWIN`) that the new alternate tree is doing better than the original decision subtree, we replace the original decision subtree by the new alternate subtree.

3.2   **DDM Bagging using trees of different size**

We improve the Bagging using trees of different size, by adding a change detector for each Hoeffding tree to speed up the adaption to the evolving stream.

We use two drift detection methods (DDM and EDDM) proposed by Gama et al. [9] and Baena-García et al. [2]. These methods control the number of errors produced by the learning model during prediction. They compare the statistics of two windows: the first contains all of the data, and the second contains only the data from the beginning until the number of errors increases. Their methods do not store these windows in memory. They keep only statistics and a window of recent errors data.

The drift detection method (DDM) uses the number of errors in a sample of n examples, modelled by a binomial distribution. For each point i in the sequence that is being sampled, the error rate is the probability of misclassifying (p_i), with standard deviation given by $s_i = \sqrt{p_i(1 - p_i)/i}$. They assume (as stated in the PAC learning model [16]) that the error rate of the learning algorithm (p_i) will decrease while the number of examples increases if the distribution of the examples is stationary. A significant increase in the error of the algorithm, suggests that the class distribution is changing and, hence, the actual decision model is supposed to be inappropriate. Thus, they store the values of p_i and s_i when $p_i + s_i$ reaches its minimum value during the process (obtaining p_{pmin} and s_{min}). It then checks when the following conditions are triggered:

- $p_i + s_i \geq p_{min} + 2 \cdot s_{min}$ for the warning level. Beyond this level, the examples are stored in anticipation of a possible change of context.
- $p_i + s_i \geq p_{min} + 3 \cdot s_{min}$ for the drift level. Beyond this level the concept drift is supposed to be true, the model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level triggered. The values for p_{min} and s_{min} are reset too.

This approach is good at detecting abrupt changes and gradual changes when the gradual change is not very slow, but has difficulties when the change is gradual and slow. In that case, the examples will be stored for a long time, the drift level then takes too long to trigger and the examples in memory can be exceeded.

Baena-García et al. proposed a new method EDDM [2] in order to improve DDM. It is based on the estimated distribution of the distances between classification errors. The window resize procedure is governed by the same heuristics.

4   Comparative Experimental Evaluation

Massive **O**nline **A**nalysis (MOA) [13] is a software environment for implementing algorithms and running experiments for online learning from data streams. The data stream evaluation framework and all algorithms evaluated in this paper were implemented in the Java programming language extending the MOA software. MOA includes a collection of offline and online methods as well as tools for evaluation. In particular, it implements boosting, bagging, and Hoeffding Trees, all with and without Naïve Bayes classifiers at the leaves.

One of the key data structures used in MOA is the description of an example from a data stream. This structure borrows from WEKA, where an example is represented by an array of double precision floating point values. This provides freedom to store all necessary types of value – numeric attribute values can be stored directly, and discrete attribute values and class labels are represented by integer index values that are stored as

floating point values in the array. Double precision floating point values require storage space of 64 bits, or 8 bytes. This detail can have implications for memory usage.

We use the new experimental framework for concept drift presented in [5]. Considering data streams as data generated from pure distributions, we can model a concept drift event as a weighted combination of two pure distributions that characterizes the target concepts before and after the drift. This framework defines the probability that every new instance of the stream belongs to the new concept after the drift. It uses the sigmoid function, as an elegant and practical solution.
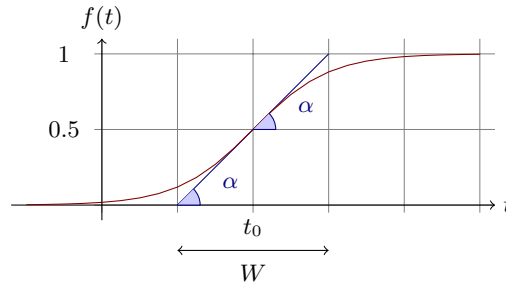

Fig. 2. A sigmoid function $f(t) = 1/(1 + e^{-s(t-t_0)})$.

We see from Figure 2 that the sigmoid function

$$f(t) = 1/(1 + e^{-s(t-t_0)})$$

has a derivative at the point t_0 equal to $f'(t_0) = s/4$. The tangent of angle α is equal to this derivative, $\tan \alpha = s/4$. We observe that $\tan \alpha = 1/W$, and as $s = 4 \tan \alpha$ then $s = 4/W$. So the parameter s in the sigmoid gives the length of W and the angle α. In this sigmoid model we only need to specify two parameters : t_0 the point of change, and W the length of change.

Definition 1. *Given two data streams a, b, we define $c = a \oplus_{t_0}^{W} b$ as the data stream built joining the two data streams a and b, where t_0 is the point of change, W is the length of change and*

- $\Pr[c(t) = a(t)] = e^{-4(t-t_0)/W}/(1 + e^{-4(t-t_0)/W})$
- $\Pr[c(t) = b(t)] = 1/(1 + e^{-4(t-t_0)/W})$.

In order to create a data stream with multiple concept changes, we can build new data streams joining different concept drifts:

$$(((a \oplus_{t_0}^{W_0} b) \oplus_{t_1}^{W_1} c) \oplus_{t_2}^{W_2} d) \ldots$$

4.1 Datasets for concept drift

Synthetic data has several advantages – it is easier to reproduce and there is little cost in terms of storage and transmission. For this paper we use the data generators most commonly found in the literature.

SEA Concepts Generator This artificial dataset contains abrupt concept drift, first introduced in [20]. It is generated using three attributes, where only the two first attributes are relevant. All three attributes have values between 0 and 10. The points of the dataset are divided into 4 blocks with different concepts. In each block, the classification is done using $f_1 + f_2 \leq \theta$, where f_1 and f_2 represent the first two attributes and θ is a threshold value. The most frequent values are 9, 8, 7 and 9.5 for the data blocks. In our framework, SEA concepts are defined as follows:

$$(((SEA_9 \oplus_{t_0}^{W} SEA_8) \oplus_{2t_0}^{W} SEA_7) \oplus_{3t_0}^{W} SEA_{9.5})$$

Rotating Hyperplane It was used as testbed for CVFDT versus VFDT in [14]. A hyperplane in d-dimensional space is the set of points x that satisfy

$$\sum_{i=1}^{d} w_i x_i = w_0 = \sum_{i=1}^{d} w_i$$

where x_i, is the ith coordinate of x. Examples for which $\sum_{i=1}^{d} w_i x_i \geq w_0$ are labeled positive, and examples for which $\sum_{i=1}^{d} w_i x_i < w_0$ are labeled negative. Hyperplanes are useful for simulating time-changing concepts, because we can change the orientation and position of the hyperplane in a smooth manner by changing the relative size of the weights. We introduce change to this dataset adding drift to each weight attribute $w_i = w_i + d\sigma$, where σ is the probability that the direction of change is reversed and d is the change applied to every example.

Random RBF Generator This generator was devised to offer an alternate complex concept type that is not straightforward to approximate with a decision tree model. The RBF (Radial Basis Function) generator works as follows: A fixed number of random centroids are generated. Each center has a random position, a single standard deviation, class label and weight. New examples are generated by selecting a center at random, taking weights into consideration so that centers with higher weight are more likely to be chosen. A random direction is chosen to offset the attribute values from the central point. The length of the displacement is randomly drawn from a Gaussian distribution with standard deviation determined by the chosen centroid. The chosen centroid also determines the class label of the example. This effectively creates a normally distributed hypersphere of examples surrounding each central point with varying densities. Only numeric attributes are generated. Drift is introduced by moving the centroids with constant speed. This speed is initialized by a drift parameter.

LED Generator This data source originates from the CART book [6]. An implementation in C was donated to the UCI [1] machine learning repository by David Aha. The goal is to predict the digit displayed on a seven-segment LED display, where

each attribute has a 10% chance of being inverted. It has an optimal Bayes classification rate of 74%. The particular configuration of the generator used for experiments (led) produces 24 binary attributes, 17 of which are irrelevant.

Data streams may be considered infinite sequences of (x, y) where x is the feature vector and y the class label. Zhang et al. [21] observe that $p(x, y) = p(x|t) \cdot p(y|x)$ and categorize concept drift in two types:

- *Loose Concept Drifting (LCD)* when concept drift is caused only by the change of the class prior probability $p(y|x)$,
- *Rigorous Concept Drifting (RCD)* when concept drift is caused by the change of the class prior probability $p(y|x)$ and the conditional probability $p(x|t)$

Note that the Random RBF Generator has RCD drift, and the rest of the dataset generators have LCD drift.

4.2   Real-World Data

It is not easy to find large real-world datasets for public benchmarking, especially with substantial concept change. The UCI machine learning repository [1] contains some real-world benchmark data for evaluating machine learning techniques. We consider three of the largest: Forest Covertype, Poker-Hand, and Electricity.

Forest Covertype Contains the forest cover type for 30 x 30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. It contains $581,012$ instances and 54 attributes, and it has been used in several papers on data stream classification [11, 18].

Poker-Hand Consists of $1,000,000$ instances and 11 attributes. Each record of the Poker-Hand dataset is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes (suit and rank), for a total of 10 predictive attributes. There is one Class attribute that describes the "Poker Hand". The order of cards is important, which is why there are 480 possible Royal Flush hands instead of 4.

Electricity Is another widely used dataset described by M. Harries [12] and analysed by Gama [9]. This data was collected from the Australian New South Wales Electricity Market. In this market, the prices are not fixed and are affected by demand and supply of the market. The prices in this market are set every five minutes. The ELEC dataset contains $45,312$ instances. Each example of the dataset refers to a period of 30 minutes, i.e. there are 48 instances for each time period of one day. The class label identifies the change of the price related to a moving average of the last 24 hours. The class level only reflect deviations of the price on a one day average and removes the impact of longer term price trends.

The size of these datasets is small, compared to tens of millions of training examples of synthetic datasets: $45,312$ for ELEC dataset, $581,012$ for CoverType, and $1,000,000$ for Poker-Hand. Another important fact is that we do not know when drift occurs or indeed if there is any drift. We may simulate RCD concept drift, joining the

	Hyperplane Drift .0001			Hyperplane Drift .001			SEA W= 50000		
	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.
BagADWIN 10 HAT	3025.87	90.80	9.91	2834.85	90.02	1.23	154.91	**89.02**	2.35
DDM Bag10 ASHT W	1321.64	**91.57**	0.85	1351.16	**91.23**	2.10	44.02	88.80	0.65
EDDM Bag10 ASHT W	1362.31	91.39	3.15	1371.46	90.96	2.77	48.95	88.75	0.90
NaiveBayes	86.97	84.37	0.01	86.87	73.69	0.01	5.52	83.87	0.00
NBADWIN	308.85	91.40	0.06	295.19	90.68	0.06	12.40	87.58	0.02
HT	157.71	86.39	9.57	159.43	80.70	10.41	7.20	84.87	0.33
HT DDM	174.10	89.28	0.04	180.51	88.48	0.01	7.88	88.07	0.16
HT EDDM	207.47	88.95	13.23	193.07	87.64	2.52	8.52	87.64	0.06
HAT	500.81	89.88	1.72	431.6	88.72	0.15	20.96	88.32	0.18
BagADWIN 10 HT	1306.22	91.16	11.40	1308.08	90.48	5.52	53.15	88.53	0.88
Bag10 HT	1236.92	87.68	108.75	1253.07	81.80	114.14	30.88	85.34	3.36
Bag10 ASHT	1060.37	91.11	2.68	1070.44	90.08	2.69	35.30	87.57	0.91
Bag10 ASHT W	1055.87	91.40	2.68	1073.96	90.65	2.69	35.69	87.91	0.91
Bag10 ASHT R	995.06	91.47	2.95	1016.48	90.61	2.14	33.74	88.07	0.84
Bag10 ASHT W+R	996.52	**91.57**	2.95	1024.02	90.94	2.14	33.56	88.30	0.84
Bag5 ASHT W+R	551.53	90.75	0.08	562.09	90.57	0.09	20.00	87.99	0.05
OzaBoost	974.69	87.01	130.00	959.14	82.56	123.75	39.97	86.17	4.00

Table 1. Comparison of algorithms. Accuracy is measured as the final percentage of examples correctly classified over the 1 or 10 million test/train interleaved evaluation. Time is measured in seconds, and memory in MB. The best individual accuracies are indicated in boldface. Note that due to the large number of test examples all differences are statistically significant, but these differences may not be meaningful in practise.

three datasets, merging attributes, and supposing that each dataset corresponds to a different concept.

$$\text{CovPokElec} = (\text{CoverType} \oplus_{581,012}^{5,000} \text{Poker}) \oplus_{1,000,000}^{5,000} \text{ELEC}$$

As all examples need to have the same number of attributes, we simple concatenate all the attributes, and we set a number of classes that is the maximum number of classes of all the datasets.

4.3 Results

We use the datasets for evaluation explained in the previous sections. The experiments were performed on a 2.0 GHz Intel Core Duo PC machine with 2 Gigabyte main memory, running Ubuntu 8.10. The evaluation methodology used was Interleaved Test-Then-Train: every example was used for testing the model before using it to train. This interleaved test followed by train procedure was carried out on 10 million examples from the hyperplane and RandomRBF datasets, and one million examples from the SEA dataset. Tables 1 and 2 reports the final accuracy, and speed of the classification models induced on synthetic data. Table 3 shows the results for real datasets: Forest CoverType, Poker Hand, and CovPokElec. The results for the Electricity dataset were structurally similar
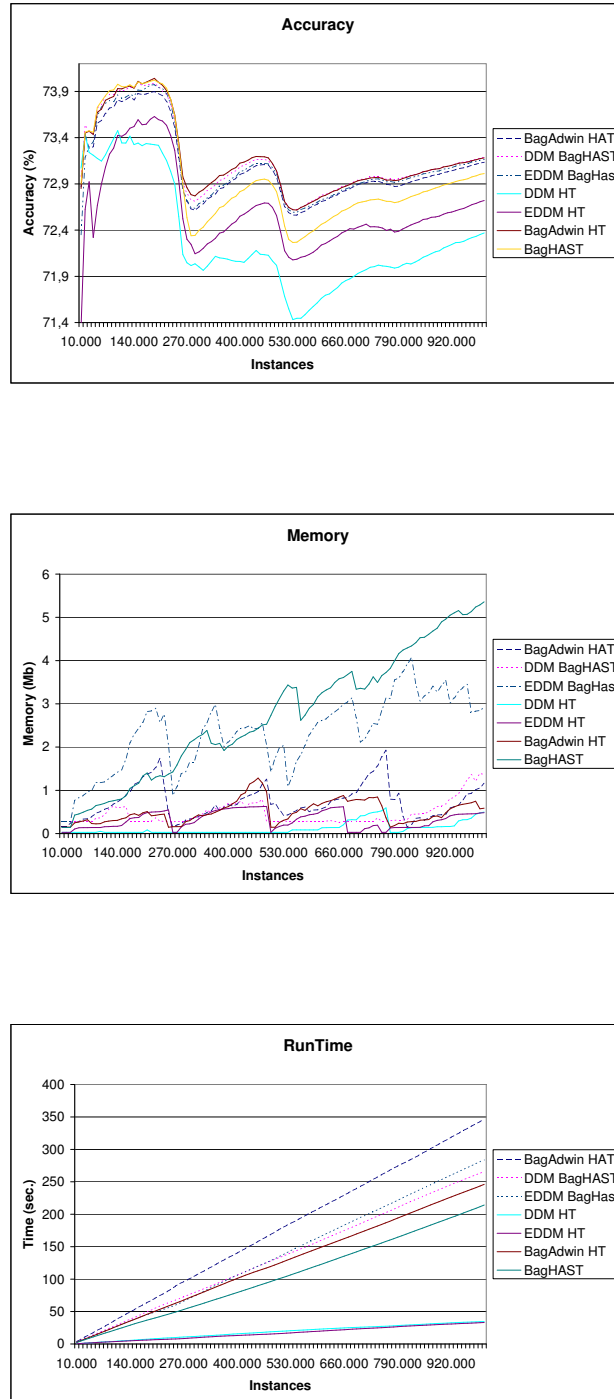
Fig. 3. Accuracy, runtime and memory on dataset LED with three concept drifts.

	RandomRBF No Drift 50 centers			RandomRBF Drift .0001 50 centers			RandomRBF Drift .001 50 centers		
	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.
BagADWIN 10 HAT	5378.75	**95.40**	119.17	2706.91	**86.73**	0.51	1976.62	69.09	0.10
DDM Bag10 ASHT W	1230.43	93.01	3.21	1349.65	83.42	0.53	1441.22	69.85	3.09
EDDM Bag10 ASHT W	1317.58	93.29	3.76	1366.65	84.30	0.71	1422.31	**70.29**	0.71
NaiveBayes	111.12	72.04	0.01	111.47	53.21	0.01	113.37	53.17	0.01
NBADWIN	396.01	72.04	0.08	272.58	68.07	0.05	249.1	62.20	0.04
HT	154.67	93.64	6.86	189.25	63.64	9.86	186.47	55.48	8.90
HT DDM	185.15	93.64	13.72	199.95	76.49	0.02	206.41	64.09	0.03
HT EDDM	185.89	93.66	13.81	214.55	75.55	0.09	203.41	64.00	0.02
HAT	794.48	93.63	9.28	413.53	79.09	0.09	294.94	65.29	0.01
BagADWIN 10 HT	1238.50	95.29	67.79	1326.12	85.23	0.26	1354.03	67.18	0.03
Bag10 HT	995.46	95.30	71.26	1362.66	71.08	106.20	1240.89	58.15	88.52
Bag10 ASHT	1009.62	85.47	3.73	1124.40	76.09	3.05	1133.51	66.36	3.10
Bag10 ASHT W	986.90	93.76	3.73	1104.03	76.61	3.05	1106.26	66.94	3.10
Bag10 ASHT R	913.74	91.96	2.65	1069.76	84.28	3.74	1085.99	67.83	2.35
Bag10 ASHT W+R	925.65	93.57	2.65	1068.59	84.71	3.74	1101.10	69.27	2.35
Bag5 ASHT W+R	536.61	85.47	0.06	557.20	81.69	0.09	587.46	68.19	0.10
OzaBoost	964.75	94.82	206.60	1312.00	71.64	105.94	1266.75	58.20	88.36

Table 2. Comparison of algorithms. Accuracy is measured as the final percentage of examples correctly classified over the 1 or 10 million test/train interleaved evaluation. Time is measured in seconds, and memory in MB.

to those for the Forest CoverType dataset and therefore not reported. Additionally, the learning curves and model growth curves for LED dataset are plotted (Figure 3).

The first, and baseline, algorithm (HT) is a single Hoeffding tree, enhanced with adaptive Naive Bayes leaf predictions. Parameter settings are $n_{min} = 1000$, $\delta = 10^{-8}$ and $\tau = 0.05$, as used in [8]. The HT DDM and HT EDDM are Hoeffding Trees with drift detection methods as explained in Section 3.2.

Bag10 is Oza and Russell online bagging using ten classifiers and Bag5 only five. BagADWIN is the online bagging version using ADWIN explained in Section 2.2. As described earlier, we implemented the following new variants of bagging:

- ADWIN Bagging using Hoeffding Adaptive Trees.
- Bagging ASHT using the DDM drift detection method
- Bagging ASHT using the EDDM drift detection method

In general terms, we observe that ensemble methods perform better than single classifier methods, and that explicit drift detection is better. However, these improvements come at a cost of runtime and memory. In fact, the results indicate that memory is not as big an issue as the runtime accuracy tradeoff.

RCD drift produces much more drama than LCD - for example, the best result in Table 2 goes down 16% when increasing the drift from 0.0001 to 0.001 - in Hyperplane the same change in drift elicits only a fractional change in accuracy. For RCD drift all

	Cover Type			Poker			CovPokElec		
	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.
BagADWIN 10 HAT	317.75	85.48	0.2	267.41	**88.63**	13.74	1403.40	**87.16**	0.62
DDM Bag10 ASHT W	249.92	**88.39**	6.09	128.17	75.85	1.51	876.18	84.54	19.74
EDDM Bag10 ASHT W	207.10	86.72	0.37	141.96	85.93	12.84	828.63	84.98	48.54
NaiveBayes	31.66	60.52	0.05	13.58	50.01	0.02	91.50	23.52	0.08
NBADWIN	127.34	72.53	5.61	64.52	50.12	1.97	667.52	53.32	14.51
HT	31.52	77.77	1.31	18.98	72.14	1.15	95.22	74.00	7.42
HT DDM	40.26	84.35	0.33	21.58	61.65	0.21	114.72	71.26	0.42
HT EDDM	34.49	86.02	0.02	22.86	72.20	2.30	114.57	76.66	11.15
HAT	55.00	81.43	0.01	31.68	72.14	1.24	188.65	75.75	0.01
BagADWIN 10 HT	247.50	84.71	0.23	165.01	84.84	8.79	911.57	85.95	0.41
Bag10 HT	138.41	83.62	16.80	121.03	87.36	12.29	624.27	81.62	82.75
Bag10 ASHT	213.75	83.34	5.23	124.76	86.80	7.19	638.37	78.87	29.30
Bag10 ASHT W	212.17	85.37	5.23	123.72	87.13	7.19	636.42	80.51	29.30
Bag10 ASHT R	229.06	84.20	4.09	122.92	86.21	6.47	776.61	80.01	29.94
Bag10 ASHT W+R	198.04	86.43	4.09	123.25	86.76	6.47	757.00	81.05	29.94
Bag5 ASHT W+R	116.83	83.79	0.23	57.09	75.87	0.44	363.09	77.65	0.95
OzaBoost	170.73	85.05	21.22	151.03	87.85	14.50	779.99	84.69	105.63

Table 3. Comparison of algorithms on real data sets. Time is measured in seconds, and memory in MB.

methods drop significantly. The bagging methods have the most to lose and go down between 14-17% (top three) and 10-18% (bottom of table). The base methods have less to lose going down (0-14%).

For all datasets one of the new methods always wins in terms of accuracy. Specifically, on the nine analysed datasets: `BagADWIN 10 HAT` wins five times out of nine, `DDM Bag10 ASHT W` wins three times, `EDDM Bag10 ASHT W` wins once; this relationship is consistent in the following way: whenever the single `HAT` beats `Bag10 ASHT W`, then `BagADWIN 10 HAT` beats `DDM Bag10 ASHT W`, and vice versa. Note that the bad result for `DDM Bag10 ASHT W` on the Poker dataset must be due to too many false positive drift predictions that wrongly keep the model too small (this can be verified by observing the memory usage in this case), which is mirrored by the behavior of `HT DDM` on the Poker dataset.

The new improved ensemble methods presented in this paper are slower than the old ones, with `BagADWIN 10 HAT` being worst, because it pays twice: through the addition of `ADWIN` and `HAT`, the latter being slower than `HT` by a factor between two and five. Change detection can be time-consuming, the extreme case being `Naive Bayes` vs. `NBAdwin`, where `Naive Bayes` can be up to six times faster. However, change detection helps accuracy, with improvements of up to 20 percentage points (see, for example, CovPokElec).

Non-drift-aware non-adaptive ensembles like `Bag10 HT` and `OzaBoost` usually need the most memory, sometimes by a very large margin. `Bag10 ASHT W+R` needs a lot more memory than `Bag5 ASHT W+R`, because the last trees in a `Bag N ASHT W+R` needs as much space as the full `Bag N-1 ASHT W+R` ensemble.

Recall that data stream evaluation is fundamentally three-dimensional. When adding adaptability to evolving data streams using change detector methods, we increase the run-time, obtaining more accurate methods. For example, adding a change detector DDM to HT, or to `Bag10 ASHT W`, in Table 1, we observe a higher cost in runtime, but also an improvement in accuracy.

5   Conclusions and Future Work

We have presented two new improvements for bagging methods, using Hoeffding Adaptive Trees and change detection methods. In general terms, we observe that using explicit drift detection methods we improve accuracy. However, these improvements come at a cost of runtime and memory. It seems that the cost of improving accuracy in bagging methods for data streams, is large in runtime, but small in memory.

As future work, we would like to build new ensemble methods that perform with an accuracy similar to the methods presented in this paper, but using less runtime. These new methods could be a boosting ensemble method, or a bagging method using new change detection strategies. We think that a boosting method could improve bagging performance by increasing tree diversity, as it is shown by the increased performance of boosting for the traditional batch learning setting. This could be a challenging topic since in [5], the authors didn't find any boosting method in the literature [17, 19, 7] that outperformed bagging in the streaming setting.

References

[1] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[2] M. Baena-García, J. D. Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno. Early drift detection method. In *Fourth International Workshop on Knowledge Discovery from Data Streams*, 2006.

[3] A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, pages 443–448, 2007.

[4] A. Bifet and R. Gavaldà. Adaptive learning from evolving data streams. In *IDA*, 2009.

[5] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *KDD '09*. ACM, 2009.

[6] L. Breiman et al. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.

[7] F. Chu and C. Zaniolo. Fast and light boosting for adaptive mining of data streams. In *PAKDD*, pages 282–292. Springer Verlag, 2004.

[8] P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[9] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *SBIA Brazilian Symposium on Artificial Intelligence*, pages 286–295, 2004.

[10] J. Gama, P. Medas, and R. Rocha. Forest trees for on-line data. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 632–636, New York, NY, USA, 2004. ACM Press.

[11] J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *KDD '03*, pages 523–528, August 2003.

[12] M. Harries. Splice-2 comparative evaluation: Electricity pricing. Technical report, The University of South Wales, 1999.

[13] G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis. `http://sourceforge.net/projects/ moa-datastream`. 2007.

[14] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *KDD'01*, pages 97–106, San Francisco, CA, 2001. ACM Press.

[15] D. D. Margineantu and T. G. Dietterich. Pruning adaptive boosting. In *ICML '97*, pages 211–218, 1997.

[16] T. Mitchell. *Machine Learning*. McGraw-Hill Education (ISE Editions), October 1997.

[17] N. Oza and S. Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics 2001*, pages 105–112. Morgan Kaufmann, 2001.

[18] N. C. Oza and S. Russell. Experimental comparisons of online and batch versions of bagging and boosting. In *KDD '01*, pages 359–364, August 2001.

[19] R. Pelossof, M. Jones, I. Vovsha, and C. Rudin. Online coordinate boosting. *http://arxiv.org/abs/0810.4553*, 2008.

[20] W. N. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *KDD '01*, pages 377–382, New York, NY, USA, 2001. ACM Press.

[21] P. Zhang, X. Zhu, and Y. Shi. Categorizing and mining concept drifting data streams. In *KDD '08*, pages 812–820. ACM, 2008.