New Ensemble Methods For Evolving Data Streams

Albert Bifet
UPC-Barcelona Tech
Barcelona, Catalonia
abifet@lsi.upc.edu

Geoff Holmes
University of Waikato
Hamilton, New Zealand
geoff@cs.waikato.ac.nz

Bernhard Pfahringer
University of Waikato
Hamilton, New Zealand
bernhard@cs.waikato.ac.nz

Richard Kirkby
University of Waikato
Hamilton, New Zealand
rkirkby@cs.waikato.ac.nz

Ricard Gavaldà
UPC-Barcelona Tech
Barcelona, Catalonia
gavalda@lsi.upc.edu

ABSTRACT

Advanced analysis of data streams is quickly becoming a key area of data mining research as the number of applications demanding such processing increases. Online mining when such data streams evolve over time, that is when concepts drift or change completely, is becoming one of the core issues. When tackling non-stationary concepts, ensembles of classifiers have several advantages over single classifier methods: they are easy to scale and parallelize, they can adapt to change quickly by pruning under-performing parts of the ensemble, and they therefore usually also generate more accurate concept descriptions. This paper proposes a new experimental data stream framework for studying concept drift, and two new variants of Bagging: `ADWIN` Bagging and Adaptive-Size Hoeffding Tree (ASHT) Bagging. Using the new experimental framework, an evaluation study on synthetic and real-world datasets comprising up to ten million examples shows that the new ensemble methods perform very well compared to several known methods.

Categories and Subject Descriptors

H.2.8 [**Database applications**]: Database Applications—*Data Mining*

General Terms

Algorithms

Keywords

Data streams, ensemble methods, concept drift, decision trees

1. INTRODUCTION

Conventional knowledge discovery tools assume that the volume of data is such that we can store all data in memory or local secondary storage, and there is no limitation on processing time. In the *Data Stream* model, we have space and time restrictions. Examples of data streams are sensoring video streams, network event logs, telephone call records, credit card transactional flows, etc. An important fact is that data may be evolving over time, so we need methods that adapt automatically. Under the constraints of the Data Stream model, 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 minimal number of parameters.

These properties may be interdependent: adjusting the time and space used by an algorithm can influence accuracy. By storing more pre-computed information, such as look up tables, an algorithm can run faster at the expense of space. An algorithm can also run faster by processing less information, either by stopping early or storing less, thus having less data to process. The more time an algorithm has, the more likely it is that accuracy can be increased.

Ensemble methods are combinations of several models whose individual predictions are combined in some manner (e.g., averaging or voting) to form a final prediction. Ensemble learning classifiers often have better accuracy and they are easier to scale and parallelize than single classifier methods.

A majority of concept drift research in data streams mining is done using traditional data mining frameworks such as WEKA [26]. As the data stream setting has constraints that a traditional data mining environment does not, we believe that a new framework is needed to help to improve the empirical evaluation of these methods.

We present in Section 2 a novel framework for evaluation of concept drift. Sections 3 and 4 present two novel ensemble methods for handling concept drift, and Section 5 shows a first comprehensive cross-method comparison. We present conclusions in Section 6. Source code and datasets will be made available at `http://sourceforge.net/projects/moa-datastream`.

2. EXPERIMENTAL FRAMEWORK FOR CONCEPT DRIFT

A data stream environment has different requirements from the traditional setting [15]. The most significant are the following:

Requirement 1 Process an example at a time, and inspect it only once (at most)

KDD'09, June 28–July 1, 2009, Paris, France.

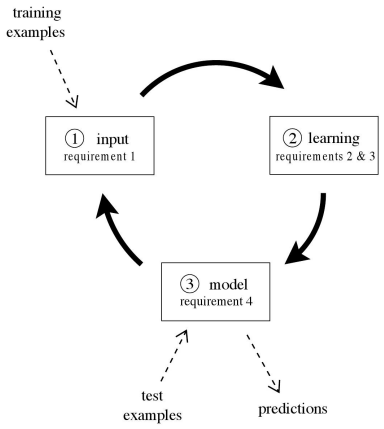

Figure 1: The data stream classification cycle

Requirement 2 Use a limited amount of memory

Requirement 3 Work in a limited amount of time

Requirement 4 Be ready to predict at any time

We have to consider these requirements in order to design a new experimental framework for data streams. Figure 1 illustrates the typical use of a data stream classification algorithm, and how the requirements fit in a repeating cycle:

1. The algorithm is passed the next available example from the stream (requirement 1).

2. The algorithm processes the example, updating its data structures. It does so without exceeding the memory bounds set on it (requirement 2), and as quickly as possible (requirement 3).

3. The algorithm is ready to accept the next example. On request it is able to predict the class of unseen examples (requirement 4).

In traditional batch learning the problem of limited data is overcome by analyzing and averaging multiple models produced with different random arrangements of training and test data. In the stream setting the problem of (effectively) unlimited data poses different challenges. One solution involves taking snapshots at different times during the induction of a model to see how much the model improves.

The evaluation procedure of a learning algorithm determines which examples are used for training the algorithm, and which are used to test the model output by the algorithm. The procedure used historically in batch learning has partly depended on data size. As data sizes increase, practical time limitations prevent procedures that repeat training too many times. It is commonly accepted with considerably larger data sources that it is necessary to reduce the numbers of repetitions or folds to allow experiments to complete in reasonable time. When considering what procedure to use in the data stream setting, one of the unique concerns is how to build a picture of accuracy over time. Two main approaches arise:

- **Holdout**: When traditional batch learning reaches a scale where cross-validation is too time consuming, it is often accepted to instead measure performance on a single holdout set. This is most useful when the division between train and test sets have been predefined, so that results from different studies can be directly compared.

- **Interleaved Test-Then-Train**: Each individual example can be used to test the model before it is used for training, and from this the accuracy can be incrementally updated [15]. When intentionally performed in this order, the model is always being tested on examples it has not seen. This scheme has the advantage that no holdout set is needed for testing, making maximum use of the available data. It also ensures a smooth plot of accuracy over time, as each individual example will become increasingly less significant to the overall average.

As data stream classification is a relatively new field, such evaluation practices are not nearly as well researched and established as they are in the traditional batch setting. The majority of experimental evaluations use less than one million training examples. Some papers use more than this, up to ten million examples, and only very rarely is there any study like Domingos and Hulten [8, 14] that is in the order of tens of millions of examples. In the context of data streams this is disappointing, because to be truly useful at data stream classification the algorithms need to be capable of handling very large (potentially infinite) streams of examples. Demonstrating systems only on small amounts of data does not build a convincing case for capacity to solve more demanding data stream applications.

A claim of this paper is that in order to adequately evaluate data stream classification algorithms they need to be tested on large streams, in the order of tens of millions of examples where possible, and under explicit memory limits. Any less than this does not actually test algorithms in a realistically challenging setting.

2.1 Concept Drift Framework

We present a new experimental framework for concept drift. Our goal is to introduce artificial drift to data stream generators in a straightforward way.

The framework approach most similar to the one presented in this paper is the one proposed by Narasimhamurthy et al. [18]. They proposed a general framework to generate data simulating changing environments. Their framework accommodates the STAGGER and Moving Hyperplane generation strategies. They consider a set of k data sources with known distributions. As these distributions at the sources are fixed, the data distribution at time t, $D^{(t)}$ is specified through $v_i(t)$, where $v_i(t) \in [0, 1]$ specify the extent of the influence of data source i at time t:

$$D^{(t)} = \{v_1(t), v_2(t), \ldots, v_k(t)\}, \sum_i v_i(t) = 1$$

Their framework covers gradual and abrupt changes. Our approach is more concrete, we begin by dealing with a simple scenario: a data stream and two different concepts. Later, we will consider the general case with more than one concept drift events.

Considering data streams as data generated from pure distributions, we can model a concept drift event as a weighted

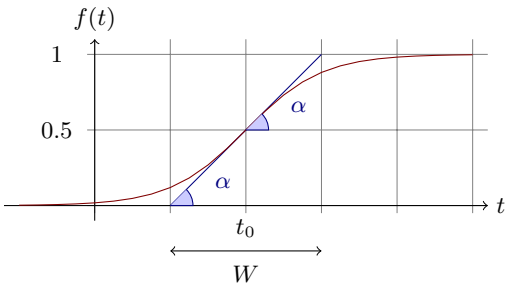

Figure 2: A sigmoid function $f(t) = 1/(1 + e^{-s(t-t_0)})$.

combination of two pure distributions that characterizes the target concepts before and after the drift. In our framework, we need to define the probability that every new instance of the stream belongs to the new concept after the drift. We will use the sigmoid function, as an elegant and practical solution.

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. Note that for any positive real number β

$$f(t_0 + \beta \cdot W) = 1 - f(t_0 - \beta \cdot W),$$

and that $f(t_0 + \beta \cdot W)$ and $f(t_0 - \beta \cdot W)$ are constant values that don't depend on t_0 and W:

$$f(t_0 + W/2) = 1 - f(t_0 - W/2) = 1/(1 + e^{-2}) \approx 88.08\%$$

$$f(t_0 + W) = 1 - f(t_0 - W) = 1/(1 + e^{-4}) \approx 98.20\%$$

$$f(t_0 + 2W) = 1 - f(t_0 - 2W) = 1/(1 + e^{-8}) \approx 99.97\%$$

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*

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

We observe the following properties, if $a \neq b$:

- $a \oplus_{t_0}^{W} b \neq b \oplus_{t_0}^{W} a$
- $a \oplus_{t_0}^{W} a = a$
- $a \oplus_{0}^{0} b = b$
- $a \oplus_{t_0}^{W} (b \oplus_{t_0}^{W} c) \neq (a \oplus_{t_0}^{W} b) \oplus_{t_0}^{W} c$
- $a \oplus_{t_0}^{W} (b \oplus_{t_1}^{W} c) \approx (a \oplus_{t_0}^{W} b) \oplus_{t_1}^{W} c$ if $t_0 < t_1$ and $W \ll |t_1 - t_0|$

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

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

2.2 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 and framework, the data generators most commonly found in the literature have been collected.

SEA Concepts Generator This artificial dataset contains abrupt concept drift, first introduced in [25]. 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:

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

STAGGER Concepts Generator They were introduced by Schlimmer and Granger in [23]. The STAGGER Concepts are boolean functions of three attributes encoding objects: size (small, medium, and large), shape (circle, triangle, and rectangle), and colour (red,blue, and green). A concept description covering either green rectangles or red triangles is represented by (shape= rectangle and colour=green) or (shape=triangle and colour=red).

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 [3] 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.

Waveform Generator It shares its origins with LED, and was also donated by David Aha to the UCI repository. The goal of the task is to differentiate between three different classes of waveform, each of which is generated from a combination of two or three base waves. The optimal Bayes classification rate is known to be 86%. There are two versions of the problem, wave21 which has 21 numeric attributes, all of which include noise, and wave40 which introduces an additional 19 irrelevant attributes.

Function Generator It was introduced by Agrawal et al. in [1], and was a common source of data for early work on scaling up decision tree learners [2, 17, 24, 11]. The generator produces a stream containing nine attributes, six numeric and three categorical. Although not explicitly stated by the authors, a sensible conclusion is that these attributes describe hypothetical loan applications. There are ten functions defined for generating binary class labels from the attributes. Presumably these determine whether the loan should be approved.

Data streams may be considered infinite sequences of (x, y) where x is the feature vector and y the class label. Zhang et al. [27] 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.

2.2.1 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 [3] contains some real-world benchmark data for evaluating machine learning techniques. We will consider three : Forest Covertype, Poker-Hand, and Electricity.

Forest Covertype dataset It 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 [10, 20].

Poker-Hand dataset It 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 dataset Another widely used dataset is the Electricity Market Dataset described by M. Harries [12] and used 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 ELEC2 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 ELEC2 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 if there is any drift. We may simulate RCD concept drift, joining the three datasets, merging attributes, and supposing that each dataset corresponds to a different concept.

$$\text{CovPokElec} = (\text{CoverType} \oplus^{5,000}_{581,012} \text{Poker}) \oplus^{5,000}_{1,000,000} \text{ELEC2}$$

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.

3. NEW METHOD OF BAGGING USING TREES OF DIFFERENT SIZE

In this section, we present a new method of bagging 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

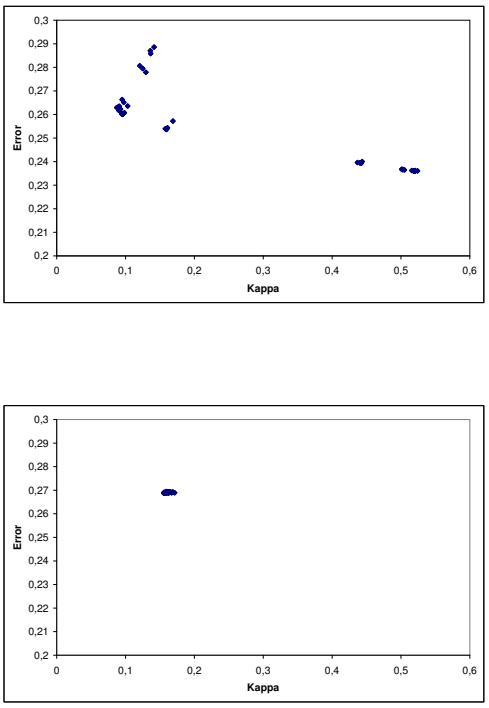

Figure 3: Kappa-Error diagrams for ASHT bagging (top) and bagging (bottom) on dataset RandomRBF with drift, plotting 90 pairs of classifiers.

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 Hoeffding 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.

In this paper, we introduce the Adaptive-Size Hoeffding Tree (ASHT). It is derived from the Hoeffding Tree algorithm with the following differences:

- it has a 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 do better during periods with no or little 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, i.e., restart from a new root.

We present a new bagging method 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, we attempt to improve bagging performance by increasing tree diversity. It has been observed that boosting tends to produce a more diverse set of classifiers than bagging, and this has been cited as a factor in increased performance [16].

We use the Kappa statistic κ to show how using trees of different size, we increase the diversity of the ensemble. Let's consider two classifiers h_a and h_b, a data set containing m examples, and a contingency table where cell C_{ij} contains the number of examples for which $h_a(x) = i$ and $h_b(x) = j$. If h_a and h_b are identical on the data set, then all non-zero counts will appear along the diagonal. If h_a and h_b are very different, then there should be a large number of counts off the diagonal. We define

$$\Theta_1 = \frac{\sum_{i=1}^{L} C_{ii}}{m}$$

$$\Theta_2 = \sum_{i=1}^{L} \left(\sum_{j=1}^{L} \frac{C_{ij}}{m} \cdot \sum_{j=1}^{L} \frac{C_{ji}}{m} \right)$$

We could use Θ_1 as a measure of agreement, but in problems where one class is much more common than others, all classifiers will agree by chance, so all pair of classifiers will obtain high values for Θ_1. To correct this, the κ statistic is defined as follows:

$$\kappa = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2}$$

κ uses Θ_2, the probability that two classifiers agree by chance, given the observed counts in the table. If two classifiers agree on every example then $\kappa = 1$, and if their predictions coincide purely by chance, then $\kappa = 0$.

We use the Kappa-Error diagram to compare the diversity of normal bagging with bagging using trees of different size. The Kappa-Error diagram is a scatterplot where each point corresponds to a pair of classifiers. The x coordinate of the pair is the κ value for the two classifiers. The y coordinate is the average of the error rates of the two classifiers.

Figure 3 shows the Kappa-Error diagram for the Random RBF dataset with drift parameter or change speed equal to 0.001.We observe that bagging classifiers are very similar to one another and that the decision tree classifiers of different size are very diferent from one another.

	Hyperplane Drift .0001			Hyperplane Drift .001			SEA W =50			SEA W= 50000		
	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.
DecisionStump	50.84	65.80	0.01	54.76	62.51	0.01	4.32	66.34	0.01	4.52	66.37	0.01
NaiveBayes	86.97	84.37	0.01	86.87	73.69	0.01	5.32	83.87	0.00	5.52	83.87	0.00
HT	157.71	86.39	9.57	159.43	80.70	10.41	6.96	84.89	0.34	7.20	84.87	0.33
HT DDM	174.10	89.28	0.04	180.51	88.48	0.01	8.30	88.27	0.17	7.88	88.07	0.16
HT EDDM	207.47	88.95	13.23	193.07	87.64	2.52	8.56	87.97	0.18	8.52	87.64	0.06
HOT5	307.98	86.85	20.87	480.19	81.91	32.02	11.46	84.92	0.38	12.46	84.91	0.37
HOT50	890.86	87.37	32.04	3440.37	81.77	32.15	22.54	85.20	0.84	22.78	85.18	0.83
AdaHOT5	322.48	86.91	21.00	486.46	82.46	32.03	11.46	84.94	0.38	12.48	84.94	0.38
AdaHOT50	865.86	87.44	32.04	3369.89	83.47	32.15	22.70	85.35	0.86	22.80	85.30	0.84
Bag10 HT	1236.92	87.68	108.75	1253.07	81.80	114.14	31.06	85.45	3.38	30.88	85.34	3.36
Bag`ADWIN` 10 HT	1306.22	91.16	11.40	1308.08	90.48	5.52	54.51	88.58	1.90	53.15	**88.53**	0.88
Bag10 ASHT	1060.37	91.11	2.68	1070.44	90.08	2.69	34.99	87.83	1.04	35.30	87.57	0.91
Bag10 ASHT W	1055.87	91.40	2.68	1073.96	90.65	2.69	36.15	88.37	1.04	35.69	87.91	0.91
Bag10 ASHT R	995.06	91.47	2.95	1016.48	90.61	2.14	33.10	88.52	0.84	33.74	88.07	0.84
Bag10 ASHT W+R	996.52	**91.57**	2.95	1024.02	**90.94**	2.14	33.20	**88.89**	0.84	33.56	88.30	0.84
Bag5 ASHT W+R	551.53	90.75	0.08	562.09	90.57	0.09	19.78	88.55	0.01	20.00	87.99	0.05
OzaBoost	974.69	87.01	130.00	959.14	82.56	123.75	39.40	86.28	4.03	39.97	86.17	4.00
OCBoost	1367.77	84.96	66.12	1332.94	83.43	76.88	59.12	87.21	2.41	60.33	86.97	2.44
FLBoost	976.82	81.24	0.05	986.42	81.34	0.03	30.64	85.04	0.02	30.04	84.75	0.02

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.

4. NEW METHOD OF BAGGING USING `ADWIN`

`ADWIN`[5] 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".

`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, 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.

`ADWIN` Bagging is the online bagging method of Oza and Rusell [19] with the addition of the `ADWIN` algorithm as a change detector and as an estimator for the weights of the boosting method. When a change is detected, the worst classifier of the ensemble of classifiers is removed and a new classifier is added to the ensemble.

5. 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 compare the following methods: Hoeffding Option Trees, bagging and boosting, and DDM. We review them and their main properties briefly.

5.1 Bagging and Boosting

Bagging and Boosting are two of the best known ensemble learning algorithms. In [19] Oza and Russell developed online versions of bagging and boosting for Data Streams. They show how the process of sampling bootstrap replicates from training data can be simulated in a data stream context. They observe that the probability that any individual example will be chosen for a replicate tends to a Poisson(1) distribution.

For the boosting method, Oza and Russell note that the weighting procedure of AdaBoost actually divides the total example weight into two halves – half of the weight is assigned to the correctly classified examples, and the other half goes to the misclassified examples. They use the Poisson distribution for deciding the random probability that an example is used for training, only this time the parameter changes according to the boosting weight of the example as it is passed through each model in sequence.

Pelossof et al. presented in [21] Online Coordinate Boost-

	RandomRBF No Drift 50 centers			RandomRBF Drift .0001 50 centers			RandomRBF Drift .001 50 centers			RandomRBF Drift .001 10 centers		
	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.
DecisionStump	74.98	58.60	0.01	79.05	50.54	0.01	81.70	50.46	0.01	61.29	62.99	0.01
NaiveBayes	111.12	72.04	0.01	111.47	53.21	0.01	113.37	53.17	0.01	95.25	75.85	0.01
HT	154.67	93.64	6.86	189.25	63.64	9.86	186.47	55.48	8.90	141.63	89.07	6.97
HT DDM	185.15	93.64	13.72	199.95	76.49	0.02	206.41	64.09	0.03	173.31	89.07	13.94
HT EDDM	185.89	93.66	13.81	214.55	75.55	0.09	203.41	64.00	0.02	183.81	89.09	14.17
HOT5	398.82	94.90	23.67	412.38	67.31	27.04	318.07	56.82	18.49	271.22	89.62	15.80
HOT50	1075.74	95.04	32.04	3472.25	71.48	32.16	1086.89	58.88	32.04	949.94	89.98	32.03
AdaHOT5	400.53	94.29	23.82	415.85	71.82	27.21	319.33	59.74	18.60	270.04	89.71	15.90
AdaHOT50	975.40	94.22	32.04	3515.67	79.26	32.16	1099.77	64.53	32.04	951.88	90.07	32.03
Bag10 HT	995.46	**95.30**	71.26	1362.66	71.08	106.20	1240.89	58.15	88.52	1020.18	90.26	74.29
Bag`ADWIN` 10 HT	1238.50	95.29	67.79	1326.12	**85.23**	0.26	1354.03	67.18	0.03	1172.27	**90.29**	44.18
Bag10 ASHT	1009.62	85.47	3.73	1124.40	76.09	3.05	1133.51	66.36	3.10	992.52	84.85	3.28
Bag10 ASHT W	986.90	93.76	3.73	1104.03	76.61	3.05	1106.26	66.94	3.10	983.10	89.58	3.28
Bag10 ASHT R	913.74	91.96	2.65	1069.76	84.28	3.74	1085.99	67.83	2.35	893.55	88.83	2.57
Bag10 ASHT W+R	925.65	93.57	2.65	1068.59	84.71	3.74	1101.10	**69.27**	2.35	901.39	89.53	2.57
Bag5 ASHT W+R	536.61	85.47	0.06	557.20	81.69	0.09	587.46	68.19	0.10	525.83	84.58	0.14
OzaBoost	964.75	94.82	206.60	1312.00	71.64	105.94	1266.75	58.20	88.36	978.44	89.83	172.57
OCBoost	1188.97	92.76	50.88	1501.64	74.69	80.36	1581.96	58.60	87.85	1215.30	89.00	56.82
FLBoost	932.85	71.39	0.02	1171.42	61.85	0.03	1176.33	52.73	0.02	1053.62	74.59	0.03

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.

ing, a new online boosting algorithm for adapting the weights of a boosted classifier, which yields a closer approximation to Freund and Schapire's AdaBoost algorithm. The weight update procedure is derived by minimizing AdaBoost's loss when viewed in an incremental form. This boosting method may be reduced to a form similar to Oza and Russell's algorithm.

Chu and Zaniolo proposed in [7] Fast and Light Boosting for adaptive mining of data streams. It is based on a dynamic sample-weight assignment scheme that is extended to handle concept drift via change detection. The change detection approach aims at significant data changes that could cause serious deterioration of the ensemble performance, and replaces the obsolete ensemble with one built from scratch.

5.2 Adaptive Hoeffding Option Trees

Hoeffding Option Trees [22] are regular Hoeffding trees containing additional option nodes that allow several tests to be applied, leading to multiple Hoeffding trees as separate paths. They consist of a single structure that efficiently represents multiple trees. A particular example can travel down multiple paths of the tree, contributing, in different ways, to different options.

An *Adaptive Hoeffding Option Tree* is a Hoeffding Option Tree with the following improvement: each leaf stores an estimation of the current error. It uses an EWMA estimator with $\alpha = .2$. The weight of each node in the voting process is proportional to the square of the inverse of the error.

5.3 Drift Detection Method

The drift detection method (DDM) proposed by Gama et al. [9] controls the number of errors produced by the learning model during prediction. It compares the statistics of two windows: the first one contains all the data, and the second one contains only the data from the beginning until the number of errors increases. Their method doesn't store these windows in memory. It keeps only statistics and a window of recent errors.

They consider that the number of errors in a sample of examples is modeled by a binomial distribution. 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. They check for a warning level and a drift level. Beyond these levels, change of context is considered.

Baena-García et al. proposed a new method EDDM [4] 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.

5.4 Results

We use a variety of datasets for evaluation, as explained in Section 2.2. 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 LED and SEA datasets. 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, Electricity and CovPokElec. Additionally, the learning curves and model growth curves for LED dataset are plotted (Figure 4). For some datasets the differences in accuracy, as seen in Tables 1, 2 and 3, are marginal.

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 $\tau = 0.05$, used in [8]. The HT DDM and HT EDDM are Hoeffding Trees with drift detection methods as explained in Section 5.3. HOT, is the Hoeffding option tree algorithm, restricted to a maximum of five option paths (HOT5) or fifty option paths (HOT50). AdaHOT is explained in Section 5.2.

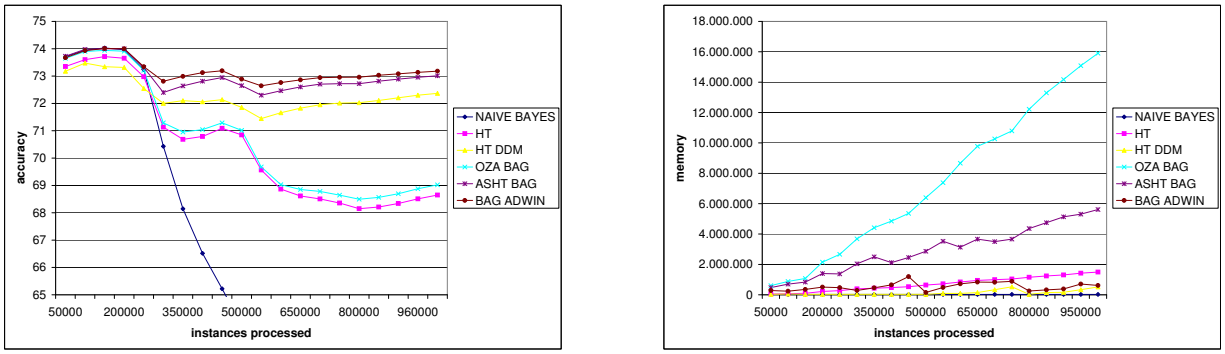

Figure 4: Accuracy and size on dataset LED with three concept drifts.

Bag10 is Oza and Russell online bagging using ten classifiers and Bag5 only five. Bag`ADWIN` is the online bagging version using `ADWIN` explained in Section 4. We implemented the following variants of bagging with Hoeffding trees of different size (ASHT):

- Bag ASHT is the base method, which deletes its root node and all its children except the one where the last split occurred,
- Bag ASHT W uses weighted classifiers,
- Bag ASHT R replaces oversized trees with new ones,
- Bag ASHT W+R uses both weighted classifiers and replaces oversized trees with new ones.

And finally, we tested three methods of boosting: Oza Boosting, Online Coordinate Boosting, and Fast and Light Boosting. The parameters used in the experimental evaluation were found to work well across a range of problems during the PhD of the first author.

Bagging is clearly the best method in terms of accuracy. This superior position is, however, achieved at high cost in terms of memory and time. `ADWIN` Bagging and ASHT Bagging are the most accurate methods for most datasets, but they are slow. `ADWIN` Bagging is slower than ASHT Bagging and for some datasets it needs more memory. ASHT Bagging using weighted classifiers and replacing oversized trees with new ones seems to be the most accurate ASHT bagging method. We observe that bagging using 5 trees of different size may be sufficient, as its error is not much higher than for 10 trees, but it is nearly twice as fast. Also Hoeffding trees using drift detection methods are faster but less accurate methods.

In [22], a range of option limits were tested and averaged across all datasets without concept drift to determine the optimal number of paths. This optimal number of options was five. Dealing with concept drift, we observe that increasing the number of options to 50, we obtain a significant improvement in accuracy for some datasets.

A summary of the best results from the synthetic and real datasets in Tables 1-3 show that of the two novel methods presented here Bag10 ASHT W+R wins five times, Bag`ADWIN` 10 HT wins four times, and Bag10 HT, OzaBoost, and OCBoost win once each. This confirms that the variants proposed in this paper are superior across this collection of datasets.

6. CONCLUSION

Our goal is to build an experimental framework for data streams similar to the WEKA framework, so that it will be easy for researchers to run experimental data stream benchmarks. New 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. These methods compared favorably in a comprehensive cross-method comparison. Data stream evaluation is fundamentally three-dimensional. These comparisons, given your specific resource limitations, indicate the method of preference. For example, on the SEA Concepts and Forest Covertype datasets the best performing method across all three dimensions are arguably HT DDM and HT EDDM, as they are almost the fastest, and almost the most accurate and, by at least an order of magnitude, easily the most memory-efficient methods. On the other hand, if both runtime and memory consumption are less of a concern, then variants of bagging usually produce excellent accuracies.

7. ACKNOWLEDGMENTS

Partially supported by the EU PASCAL2 Network of Excellence (FP7-ICT-216886), and by projects SESAAME-BAR (TIN2008-06582-C03-01), MOISES-BAR (TIN2005-08832-C03-03). Albert Bifet is supported by a FI grant through the SGR program of Generalitat de Catalunya.

8. REFERENCES

[1] R. Agrawal, S. P. Ghosh, T. Imielinski, B. R. Iyer, and A. N. Swami. An interval classifier for database mining applications. In *VLDB '92*, pages 560–573, 1992.

[2] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. on Knowl. and Data Eng.*, 5(6):914–925, 1993.

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

	Cover Type			Poker			Electricity			CovPokElec		
	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.
NaiveBayes	31.66	60.52	0.05	13.58	50.01	0.02	0.92	74.15	0.01	91.50	23.52	0.08
HT	31.52	77.77	1.31	18.98	72.14	1.15	1.16	78.88	0.06	95.22	74.00	7.42
HT DDM	40.26	84.35	0.33	21.58	61.65	0.21	1.36	84.73	0.04	114.72	71.26	0.42
HT EDDM	34.49	86.02	0.02	22.86	72.20	2.30	1.28	85.44	0.00	114.57	76.66	11.15
HOT5	65.69	83.19	5.41	31.60	72.14	1.28	2.36	82.80	0.36	138.20	75.93	13.30
HOT50	143.54	85.29	18.62	31.96	72.14	1.28	10.06	83.29	2.30	286.66	82.78	36.74
AdaHOT5	67.01	83.19	5.42	32.08	72.14	1.28	2.44	82.80	0.36	138.20	75.93	13.31
AdaHOT50	148.85	85.29	18.65	32.18	72.14	1.28	10.04	83.29	2.32	296.54	82.78	36.76
Bag10 HT	138.41	83.62	16.80	121.03	87.36	12.29	3.28	82.16	0.71	624.27	81.62	82.75
Bag`ADWIN` 10 HT	247.50	84.71	0.23	165.01	84.84	8.79	4.96	84.15	0.07	911.57	**85.95**	0.41
Bag10 ASHT	213.75	83.34	5.23	124.76	86.80	7.19	3.92	82.79	0.37	638.37	78.87	29.30
Bag10 ASHT W	212.17	85.37	5.23	123.72	87.13	7.19	3.96	84.16	0.37	636.42	80.51	29.30
Bag10 ASHT R	229.06	84.20	4.09	122.92	86.21	6.47	3.80	83.31	0.42	776.61	80.01	29.94
Bag10 ASHT W+R	198.04	**86.43**	4.09	123.25	86.76	6.47	3.84	84.83	0.42	757.00	81.05	29.94
Bag5 ASHT W+R	116.83	83.79	0.23	57.09	75.87	0.44	2.54	84.44	0.09	363.09	77.65	0.95
OzaBoost	170.73	85.05	21.22	151.03	**87.85**	14.50	3.66	84.95	1.24	779.99	84.69	105.63
OCBoost	230.94	74.39	9.42	172.29	71.15	11.49	4.70	**86.20**	0.59	1121.49	71.94	73.36
FLBoost	234.56	70.29	0.15	19.92	50.12	0.07	2.66	73.08	0.04	368.89	52.92	0.47

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

[4] 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.

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

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

[11] J. Gehrke, R. Ramakrishnan, and V. Ganti. RainForest - a framework for fast decision tree construction of large datasets. In *VLDB '98*, pages 416–427, 1998.

[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] R. Kirkby. *Improving Hoeffding Trees*. PhD thesis, University of Waikato, November 2007.

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

[17] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *EDBT '96*, pages 18–32, London, UK, 1996. Springer-Verlag.

[18] A. Narasimhamurthy and L. I. Kuncheva. A framework for generating data to simulate changing environments. In *AIAP'07*, pages 384–389, 2007.

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

[20] 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.

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

[22] B. Pfahringer, G. Holmes, and R. Kirkby. New options for hoeffding trees. In *AI*, pages 90–99, 2007.

[23] J. C. Schlimmer and R. H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[24] J. C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *VLDB '96*, pages 544–555, 1996.

[25] 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.

[26] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, June 2005.

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