Learning from Time-Changing Data with Adaptive Windowing *

Albert Bifet Ricard Gavaldà
Universitat Politècnica de Catalunya
{abifet,gavalda}@lsi.upc.edu

17 October 2006

Abstract

We present a new approach for dealing with distribution change and concept drift when learning from data sequences that may vary with time. We use sliding windows whose size, instead of being fixed *a priori*, is recomputed online according to the rate of change observed from the data in the window itself: The window will grow automatically when the data is stationary, for greater accuracy, and will shrink automatically when change is taking place, to discard stale data. This delivers the user or programmer from having to guess a time-scale for change.

Contrary to many related works, we provide rigorous guarantees of performance, as bounds on the rates of false positives and false negatives. In fact, for some change structures, we can formally show that the algorithm automatically adjusts the window to a statistically optimal length.

Using ideas from data stream algorithmics, we develop a time- and memory-efficient version of this algorithm, called `ADWIN2`. We show how to incorporate this strategy easily into two learning algorithms, the Naïve Bayes predictor and the k-means clusterer, chosen since it is relatively easy to observe their behaviour under time change. We combine `ADWIN2` with the Naïve Bayes (NB) predictor, in two ways: one, using it to monitor the error rate of the current model and declare when revision is necessary and, two, putting it inside the NB predictor to maintain up-to-date estimations of conditional probabilities in the data. We test our approach using synthetic and real data streams and compare them to both fixed-size and variable-size window strategies with good results.

Keywords: Data Streams, Time-Changing Data, Concept and Distribution Drift, Naïve Bayes

1 Introduction

Dealing with data whose nature changes over time is one of the core problems in data mining and machine learning. To mine or learn such data, one needs strategies for the following three tasks, at least: 1) detecting when change occurs 2) deciding which examples to keep and which ones to forget (or, more in general, keeping updated sufficient statistics), and 3) revising the current model(s) when significant change has been detected.

Most strategies use variations of the *sliding window* idea: a window is maintained that keeps the most recently read examples, and from which older examples are dropped according to some set of rules. The contents of the window can be used for the three tasks above: 1) to detect change (e.g., by using some statistical test on different subwindows), 2) obviously, to obtain updated statistics from the recent examples, and 3) to have data to rebuild or revise the model(s) after data has changed.

The simplest rule is to keep a window of some fixed size, usually determined *a priori* by the user. This can work well if information on the time-scale of change is available, but this is rarely the case. Normally, the user is caught in a tradeoff without solution: choosing a small size (so that the window reflects accurately the current distribution) and choosing a large size (so that many examples are available to work on, increasing accuracy in periods of stability). A different strategy uses a *decay function* to weight the importance of examples according to their age (see e.g. [5]). In this case, the tradeoff shows up in the choice of a decay constant that should match the unknown rate of change.

Less often, it has been proposed to use windows of variable size. In general, one tries to keep examples as long as possible, i.e., while not proven stale. This delivers the users from having to guess *a priori* an unknown parameter such as the time scale of change. However, most works along these lines that we know of (e.g., [7, 13, 14, 17]) are heuristics and have no rigorous guarantees of performance. Some works in computational learning theory (e.g. [3, 9, 10]) describe strategies with rigorous performance bounds, but to our knowledge they have never been tried in real learning/mining contexts and often assume a known bound on the rate of change.

In addition, window strategies have been used in conjunction with learning/mining algorithms in two ways: one, externally to the learning algorithm; the window is used to monitor the error rate of the cur-

*Partially supported by the 6th Framework Program of EU through the integrated project DELIS (#001907), by the EU PASCAL Network of Excellence, IST-2002-506778, and by the DGICYT MOISES-BAR project, TIN2005-08832-C03-03. Home pages: `http://www.lsi.upc.edu/~{abifet, gavalda}`

rent model, which under stable distributions should keep decreasing or at most stabilize; when instead this rate grows significantly, change is declared and the base learning algorithm is invoked to revise or rebuild the model with fresh data. Note that in this case the window contains bits or real numbers (not full examples). The other way is to embed the window system *inside* the learning algorithm, to maintain the statistics required by the learning algorithm continuously updated; it is then the algorithm's responsibility to keep the model in synchrony with these statistics.

In this paper, we present a new algorithm (`ADWIN`, for ADaptive WINdowing) for maintaining a window of variable size containing bits or real numbers. The algorithm automatically grows the window when no change is apparent, and shrinks it when data changes. Unlike many related works, we provide rigorous guarantees of its performance, in the form of bounds on the rates of false positives and false negatives. In fact, it is possible to show that for some change structures, `ADWIN` automatically adjusts its window size to the optimum balance point between reaction time and small variance. Since `ADWIN` keeps bits or real numbers, it can be put to work together with a learning algorithm in the first way, that is, to monitor the error rate of the current model.

The first version of `ADWIN` is inefficient in time and memory. Using ideas from data-stream algorithmics, we provide another version, `ADWIN2`, working in low memory and time. In particular, `ADWIN2` keeps a window of length W with $O(\log W)$ memory and update time, while keeping essentially the same performance guarantees as `ADWIN` (in fact, it does slightly better in experiments). Because of this low time and memory requirements, it is thus possible to use `ADWIN2` in the second way: a learning algorithm can create many instances of `ADWIN2` to maintain updated the statistics (counts, averages, entropies, ...) from which it builds the model.

To test our approach, we perform two types of experiments. In the first type, we test the ability of `ADWIN2` to track some unknown quantity, independent of any learning. We generate a sequence of random bits with some hidden probability p that changes over time. We check the rate of false positives (% of claimed changes when p does not really change) and false negatives (% of changes missed when p does change) and in this case the time until the change is declared. We compare `ADWIN2` with a number of fixed-size windows and show, as expected, that it performs about as well or only slightly worse than the best window for each rate of change, and performs far better than each windows of any fixed-size W when the change of rate is very different from W. We also compare to one of the recently proposed variable-size window methods [7] and show that it performs better, for moderately large quantities of data.

Then we test `ADWIN2` in conjunction with two learning algorithms. In this first work, we choose the Naïve Bayes (NB) predictor and a k-means clusterer since it is easiest to observe their reactions to time changes. We are currently working on the application to decision tree induction. We try both using `ADWIN2` "outside", monitoring NB's error rate, and "inside", providing accurate statistics to NB. We compare them to fixed-size windows and the variable-length window strategy in [7]. We perform experiments both on synthetic and real-life data. The second combination (`ADWIN2` inside NB) performs best, sometimes spectacularly so. The first combination performs about as well as [7] in some cases, and substantially better in others. We propose a version of k-means able to deal with centroids that move at different velocities over time, by adding an instance of `ADWIN2` to track each coordinate of each centroid. We compare the performance of this algorithm over synthetic data consisting of sums of k-gaussians. Experiments indicate that our algorithm is able to track the true centroids more accurately than any fixed-window method.

2 Comparison with Previous Work

It is impossible to review here the whole literature on dealing with time change in machine learning and data mining. We discuss only a few of the references using sliding windows that seem most related to ours. Among those using fixed-size windows, the work of Kifer et al. [12] is probably the closest in spirit to ours. They detect change by keeping two windows of fixed size, a "reference" one and "current" one, containing whole examples. The focus of their work is on comparing and implementing efficiently different statistical tests to detect change, with provable guarantees of performance. In our case, our windows contain simple information (bits or numbers), so change tests are really simple.

Among the variable-window approaches, best known are the work of Widmer and Kubat [17] and Klinkenberg and Joachims [13]. These works are essentially heuristics and are not suitable for use in data-stream contexts since they are computationally expensive. In particular, [13] checks all subwindows of the current window, like our first algorithm `ADWIN` does, and is specifically tailored to work with SVMs. The work of Last [14] uses info-fuzzy networks or IFN, as an alternative to learning decision trees. The change detection strategy is embedded in the learning algorithm, and used to revise parts of the model, hence not easily applicable to other learning methods. Domingos et al. [11] use the "monitoring" strategy in their CVFDT

system for decision-tree induction. Very roughly, each example is stored in the node it reaches. When the error rate at any given node seems to degrade, new examples are used to build an alternative subtree at that node.

Our approach is closest to that of Gama et al. [7], so we should explain the difference in more detail. Their approach is as follows: the window is used to monitor the current model's error rate. In periods of no change, that error should decrease or at most stabilize. Therefore, one keeps track of the initial subwindow where error has been lowest so far. When the error rate over the whole current window significantly exceeds this lowest error rate, change is declared. In contrast, we compare the averages of all pairs of subwindows obtained by partitioning the current one, and this should lead to faster change detection. Indeed, suppose that a sudden change occurs after T time steps without any change. In Gama's approach, we have to wait until time $T + T'$ so that the difference in averages between the windows $1..T$ and $1..(T+T')$ is significant. If T is large, it will take a comparably large T' to override the influence of the past. In contrast, in our approach, we test (among others) the partition $1..T$ and $(T+1)..(T+T')$. Even for large T, the difference in distribution will become clear with a much smaller T'.

3 Maintaining Updated Windows of Varying Length

In this section we describe our algorithms for dynamically adjusting the length of a data window, make a formal claim about its performance, and derive an efficient variation.

3.1 Setting The inputs to the algorithms are a confidence value $\delta \in (0,1)$ and a (possibly infinite) sequence of real values $x_1, x_2, x_3, \ldots, x_t, \ldots$ The value of x_t is available only at time t. Each x_t is generated according to some distribution D_t, independently for every t. We denote with μ_t and σ_t^2 the expected value and the variance of x_t when it is drawn according to D_t. We assume that x_t is always in $[0,1]$; by an easy rescaling, we can handle any case in which we know an interval $[a,b]$ such that $a \le x_t \le b$ with probability 1. Nothing else is known about the sequence of distributions D_t; in particular, μ_t and σ_t^2 are unknown for all t.

3.2 First algorithm: `ADWIN` Our algorithm keeps a sliding window W with the most recently read x_i. Let n denote the length of W, $\hat{\mu}_W$ the (observed) average of the elements in W, and μ_W the (unknown) average of μ_t for $t \in W$. Strictly speaking, these quantities should be indexed by t, but in general t will be clear from the context.

Since the values of μ_t can oscillate wildly, there is no guarantee that μ_W or $\hat{\mu}_W$ will be anywhere close to the instantaneous value μ_t, even for long W. However, μ_W is the expected value of $\hat{\mu}_W$, so μ_W and $\hat{\mu}_W$ *do* get close as W grows.

Algorithm `ADWIN` is presented in Figure 1. The idea is simple: whenever two "large enough" subwindows of W exhibit "distinct enough" averages, one can conclude that the corresponding expected values are different, and the older portion of the window is dropped. In other words, W is kept as long as possible while the null hypothesis "μ_t has remained constant in W" is sustainable up to confidence δ.[1] "Large enough" and "distinct enough" above are made precise by choosing an appropriate statistical test for distribution change, which in general involves the value of δ, the lengths of the subwindows, and their contents. We choose one particular statistical test for our implementation, but this is not the essence of our proposal – many other tests could be used. At every step, `ADWIN` simply outputs the value of $\hat{\mu}_W$ as an approximation to μ_W.

The value of ϵ_{cut} for a partition $W_0 \cdot W_1$ of W is computed as follows: Let n_0 and n_1 be the lengths of W_0 and W_1 and n be the length of W, so $n = n_0 + n_1$. Let $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ be the averages of the values in W_0 and W_1, and μ_{W_0} and μ_{W_1} their expected values. To obtain totally rigorous performance guarantees we define:

$$m = \frac{1}{1/n_0 + 1/n_1} \text{ (harmonic mean of } n_0 \text{ and } n_1),$$
$$\delta' = \frac{\delta}{n}, \text{ and } \quad \epsilon_{cut} = \sqrt{\frac{1}{2m} \cdot \ln \frac{4}{\delta'}}\,.$$

Our statistical test for different distributions in W_0 and W_1 simply checks whether the observed average in both subwindows differs by more than the threshold ϵ_{cut}. The role of δ' is to avoid problems with multiple hypothesis testing (since we will be testing n different possibilities for W_0 and W_1 and we want global error below δ). Later we will provide a more sensitive test based on the normal approximation that, although not 100% rigorous, is perfectly valid in practice.

Now we state our main technical result about the performance of `ADWIN`:

THEOREM 3.1. *At every time step we have*

1. (False positive rate bound). *If μ_t remains constant within W, the probability that* `ADWIN` *shrinks the window at this step is at most δ.*

[1] It would easy to use instead the null hypothesis "there has been no change greater than ϵ", for a user-specified ϵ expressing the smallest change that deserves reaction.

ADWIN: Adaptive Windowing Algorithm

1 Initialize Window W
2 **for** each $t > 0$
3 **do** $W \leftarrow W \cup \{x_t\}$ (i.e., add x_t to the head of W)
4 **repeat** Drop elements from the tail of W
5 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_{cut}$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 output $\hat{\mu}_W$

Figure 1: Algorithm ADWIN.

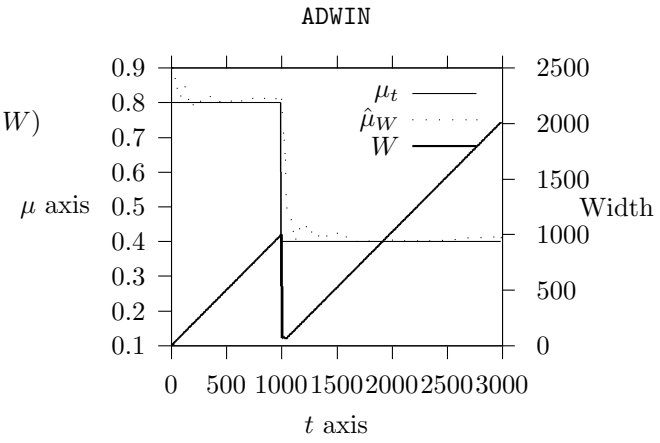

Figure 2: Output of algorithm ADWIN with abrupt change.

2. (False negative rate bound). *Suppose that for* some *partition of W in two parts W_0W_1 (where W_1 contains the most recent items) we have $|\mu_{W_0} - \mu_{W_1}| > 2\epsilon_{cut}$. Then with probability $1-\delta$* ADWIN *shrinks W to W_1, or shorter.*

The proof of the theorem is given in Appendix A. In practice, the definition of ϵ_{cut} as above is too conservative. Indeed, it is based on the Hoeffding bound, which is valid for all distributions but greatly overestimates the probability of large deviations for distributions of small variance; in fact, it is equivalent to assuming always the worst-case variance $\sigma^2 = 1/4$. In practice, one can observe that $\mu_{W_0} - \mu_{W_1}$ tends to a normal distribution for large window sizes, and use

$$\epsilon_{cut} = \sqrt{\frac{2}{m} \cdot \sigma_W^2 \cdot \ln \frac{2}{\delta'}} + \frac{2}{3m} \ln \frac{2}{\delta'}, \tag{3.1}$$

where σ_W^2 is the observed variance of the elements in window W. Thus, the term with the square root is essentially equivalent to setting ϵ_{cut} to k times the standard deviation, for k depending on the desired confidence δ, as is done in [7]. The extra additive term protects the cases where the window sizes are too small to apply the normal approximation, as an alternative to the traditional use of requiring, say, sample size at least 30; it can be formally derived from the so-called Bernstein bound. Additionally, one (somewhat involved) argument shows that setting $\delta' = \delta/(\ln n)$ is enough in this context to protect from the multiple hypothesis testing problem; anyway, in the actual algorithm that we will run (ADWIN2), only $O(\log n)$ subwindows are checked, which justifies using $\delta' = \delta/(\ln n)$. Theorem 3.1 holds for this new value of ϵ_{cut}, up to the error introduced by the normal approximation. We have used these better bounds in all our implementations.

Let us consider how ADWIN behaves in two special cases: sudden (but infrequent) changes, and slow gradual changes. Suppose that for a long time μ_t has remained fixed at a value μ, and that it suddenly jumps to a value $\mu' = \mu + \epsilon$. By part (2) of Theorem 3.1 and Equation 3.1, one can derive that the window will start shrinking after $O(\mu \ln(1/\delta)/\epsilon^2)$ steps, and in fact will be shrunk to the point where only $O(\mu \ln(1/\delta)/\epsilon^2)$ examples prior to the change are left. From then on, if no further changes occur, no more examples will be dropped so the window will expand unboundedly.

In case of a gradual change with slope α following a long stationary period at μ, observe that the average of W_1 after n_1 steps is $\mu + \alpha n_1/2$; we have $\epsilon (= \alpha n_1/2) \geq O(\sqrt{\mu \ln(1/\delta)/n_1})$ iff $n_1 = O(\mu \ln(1/\delta)/\alpha^2)^{1/3}$. So n_1 steps after the change the window will start shrinking, and will remain at approximately size n_1 from then on. A dependence on α of the form $O(\alpha^{-2/3})$ may seem odd at first, but one can show that this window length is actually optimal in this setting, even if α is known: it minimizes the sum of variance error (due to short window) and error due to out-of-date data (due to long windows in the presence of change). Thus, in this setting, ADWIN provably adjusts automatically the window setting to its optimal value, up to multiplicative constants.

Figures 2 and 3 illustrate these behaviors. In Figure 2, a sudden change from $\mu_{t-1} = 0.8$ to $\mu_t = 0.4$ occurs, at $t = 1000$. One can see that the window size grows linearly up to $t = 1000$, that ADWIN cuts the window severely 10 steps later (at $t = 1010$), and that the window expands again linearly after time $t = 1010$. In Figure 2, μ_t gradually descends from 0.8 to 0.2 in the range $t \in [1000..2000]$. In this case, ADWIN cuts the window sharply at t around 1200 (i.e., 200 steps after the slope starts), keeps the window length bounded

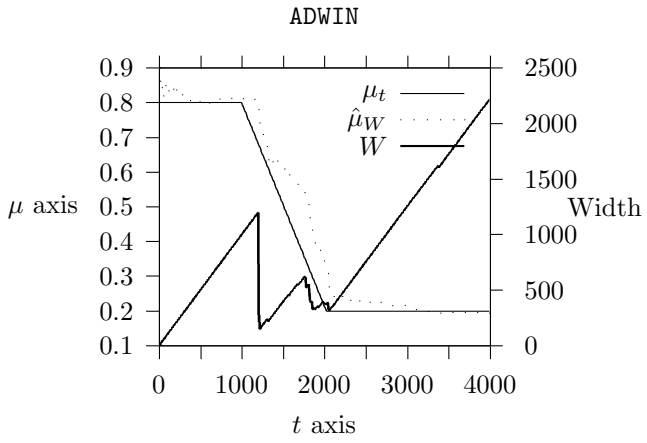

Figure 3: Output of algorithm ADWIN with slow gradual changes.

(with some random fluctuations) while the slope lasts, and starts growing it linearly again after that. As predicted by theory, detecting the change is harder in slopes than in abrupt changes.

3.3 Improving time and memory requirements

Our first version of ADWIN is computationally expensive, because it checks exhaustively all "large enough" subwindows of the current window for possible cuts. Furthermore, the contents of the window is kept explicitly, with the corresponding memory cost as the window grows. To reduce these costs we present a new version ADWIN2 using ideas developed in data stream algorithmics [1, 15, 2, 6] to find a good cutpoint quickly. We next provide a sketch of how these data structures work.

Our data structure is a variation of exponential histograms [6], a data structure that maintains an approximation of the number of 1's in a sliding window of length W with logarithmic memory and update time. We adapt this data structure in a way that can provide this approximation simultaneously for about $O(\log W)$ subwindows whose lengths follow a geometric law, *with no memory overhead* with respect to keeping the count for a single window. That is, our data structure will be able to give the number of 1s among the most recently $t-1$, $t-\lfloor c \rfloor$, $t-\lfloor c^2 \rfloor$,..., $t-\lfloor c^i \rfloor$, ...read bits, with the same amount of memory required to keep an approximation for the whole W. Note that keeping exact counts for a fixed-window size is provably impossible in sublinear memory. We go around this problem by shrinking or enlarging the window strategically so that what would otherwise be an approximate count happens to be exact.

More precisely, to design the algorithm one chooses a parameter M. This parameter controls both 1) the amount of memory used (it will be $O(M \log W/M)$ words, and 2) the closeness of the cutpoints checked (the basis c of the geometric series above, which will be about $c = 1+1/M$). Note that the choice of M does *not* reflect any assumption about the time-scale of change: Since points are checked at a geometric rate anyway, this policy is essentially scale-independent.

More precisely, in the boolean case, the information on the number of 1's is kept as a series of buckets whose size is always a power of 2. We keep at most M buckets of each size 2^i, where M is a design parameter. For each bucket we record two (integer) elements: *capacity* and *content* (size, or number of 1s it contains).

Thus, we use about $M \cdot \log(W/M)$ buckets to maintain our data stream sliding window. ADWIN2 checks as a possible cut every border of a bucket, i.e., window lengths of the form $M(1+2+\ldots+2^{i-1})+j\cdot 2^i$, for $0 \le j \le M$. It can be seen that these $M \cdot \log(W/M)$ points follow approximately a geometric law of basis $\cong 1+1/M$.

Let's look at an example: a sliding window with 14 elements. We register it as:

1010101	101	11	1	1
Content: 4	2	2	1	1
Capacity: 7	3	2	1	1

Each time a new element arrives, if the element is "1", we create a new bucket of *content* 1 and *capacity* the number of elements arrived since the last "1". After that we compress the rest of buckets: When there are $M+1$ buckets of size 2^i, we merge the two oldest ones (adding its capacity) into a bucket of size 2^{i+1}. So, we use $O(M \cdot \log W/M)$ memory words if we assume that a word can contain a number up to W. In [6], the window is kept at a fixed size W. The information missing about the last bucket is responsible for the approximation error. Here, each time we detect change, we reduce the window's length deleting the last bucket, instead of (conceptually) dropping a single element as in a typical sliding window framework. This lets us keep an exact counting, since when throwing away a whole bucket we know that we are dropping exactly 2^i "1"s.

We summarize these results with the following theorem.

THEOREM 3.2. *The* ADWIN2 *algorithm maintains a data structure with the following properties:*

- *It uses $O(M \cdot \log(W/M))$ memory words (assuming a memory word can contain numbers up to W).*
- *It can process the arrival of a new element in $O(1)$ amortized time and $O(\log W)$ worst-case time.*

- *It can provide the exact counts of* 1*'s for all the subwindows whose lengths are of the form* $\lfloor(1 + 1/M)^i\rfloor$*, in* $O(1)$ *time per query.*

Since ADWIN2 tries $O(\log W)$ cutpoints, the total processing time per example is $O(\log W)$ (amortized) and $O(\log^2 W)$ (worst-case).

In our example, suppose $M = 2$, if a new element "1" arrives then

1010101	101	11	1	1	1
Content: 4	2	2	1	1	1
Capacity: 7	3	2	1	1	1

There are 3 buckets of 1, so we compress it:

1010101	101	11	11	1
Content: 4	2	2	2	1
Capacity: 7	3	2	2	1

and now as we have 3 buckets of size 2, we compress it again

1010101	10111	11	1
Content: 4	4	2	1
Capacity: 7	5	2	1

And finally, if we detect change, we reduce the size of our sliding window deleting the last bucket:

10111	11	1
Content: 4	2	1
Capacity: 5	2	1

In the case of real values, we also maintain buckets of two elements: *capacity* and *content*. We store at *content* the sum of the real numbers we want to summarize. We restrict *capacity* to be a power of two. As in the boolean case, we use $O(\log W)$ buckets, and check $O(\log W)$ possible cuts. The memory requirement for each bucket is $\log W + R + \log\log W$ bits per bucket, where R is number of bits used to store a real number.

Figure 4 shows the output of ADWIN2 to a sudden change, and Figure 5 to a slow gradual change. The main difference with ADWIN output is that as ADWIN reduces one element by one each time it detects changes, ADWIN2 deletes an entire bucket, which yields a slightly more jagged graph in the case of a gradual change. The difference in approximation power between ADWIN and ADWIN2 is almost negligible, so we use ADWIN2 exclusively for our experiments.

4 Experimental Validation of ADWIN2

We construct the following experiments to test the performance of our algorithms. We use, somewhat arbitrarily, $M = 5$ for all experiments.

In a first experiment, we investigate the rate of false positives of ADWIN2. This is a very important measure, specially when there is a cost associated with a reported change. To do this, we feed ADWIN2 a data stream of 100,000 bits, generated from a stationary Bernoulli

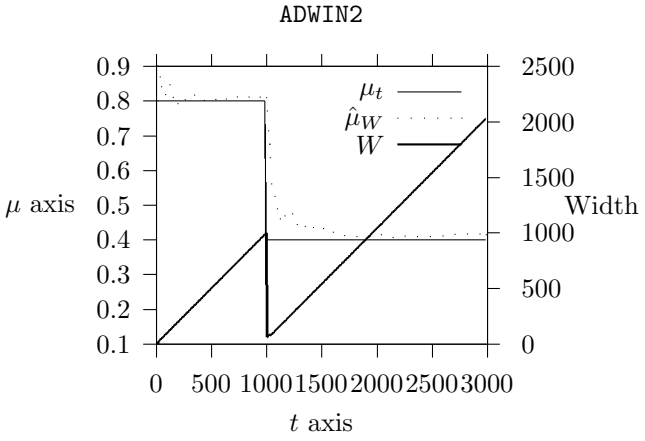

Figure 4: Output of algorithm ADWIN2 with abrupt change

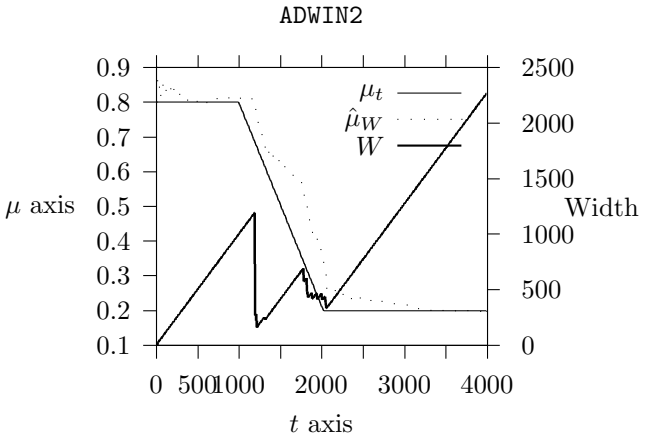

Figure 5: Output of algorithm ADWIN2 with slow gradual changes

distribution with parameter μ, and different confidence parameters δ.

Table 1 shows the ratio of false positives obtained. In all cases, it is below δ as predicted by the theory, and in fact much smaller for small values of μ.

Table 1: Rate of false positives

μ	$\delta = 0.05$	$\delta = 0.1$	$\delta = 0.3$
0.01	0.0000	0.0000	0.0000
0.1	0.0001	0.0002	0.0018
0.3	0.0008	0.0017	0.0100
0.5	0.0012	0.0030	0.0128

In a second set of experiments, we want to compare ADWIN2 as an estimator with estimations obtained from fixed-size window, and with fixed-size window which are flushed when change is detected. In the last case, we use a pair of windows (X, Y) of a fixed size W. Window X is used as a reference window that contains the first W elements of the stream that occurred after the last detected change. Window Y is a sliding window that contains the latest W items in the data stream. To detect change we check whether the difference of the averages of the two windows exceeds threshold ϵ_{cut}. If it does, we copy the content of window Y into reference window X, and empty the sliding window Y. This scheme is as in [12], and we refer to it as "fixed-size windows with flushing".

We build a framework with a stream of synthetic data, and estimators of each class: an estimator that uses ADWIN2, an array of estimators of fixed-size windows for different sizes, and also an array of fixed-size windows with flushing. Our synthetic data streams consist of some triangular wavelets, of different periods, some square wavelets, also of different periods, and a staircase wavelet of different values. We test the estimator's performance over a sample of 10^6 points, feeding the same synthetic data stream to each one of the estimators tested. We compute the average distance (both L_1 and L_2) from the true probability generating the data stream to the estimation. Finally, we compare these measures for the different estimators. Tables 2, 3, 4 and 5 shows these results using L_1 and L_2 distances and $\delta = 0.1, 0.3$. For the ADWIN2 estimator, besides the distance to the true distribution, we list as information the window length averaged over the whole run.

The general pattern for the triangular or square wavelets is as follows. For any fixed period P, the best fixed-size estimator is that whose window size is a certain fraction of P. ADWIN2 usually does sometimes does worse than this best fixed-size window, but only slightly, and often does better than even the best fixed size that we try. Additionally, it does better than any window of fixed size W when P is much larger or much smaller than W, that is, when W is a "wrong" time scale. The explanation is simple: if W is too large the estimator does not react quickly enough to change, and if W is too small the variance within the window implies a bad estimation. One can check that ADWIN2 adjusts its window length to about $P/4$ when P is small, but keeps it much smaller than P for large P, in order again to minimize the variance / time-sensitivity tradeoff.

In a third type of experiments, we use small probabilities to generate input to estimators. We compare again ADWIN2 to fixed-size strategies in the situation when getting a 1 is a rare event. To deal with this case nonadaptively, one should decide *a priori* on a very large fixed window size, which is a waste if it turns out that there happen to be many 1s. We measure the relative error of the estimator, that is |True Probability - Estimated Probability|/ True Probability. Table 6 shows the results. ADWIN2 beats almost all fixed-size window estimators, with or without flushing. This confirms that ADWIN2's capacity of shrinking or enlarging its window size can be a very useful tool for to accurately track the probability of infrequent events.

In a fourth type of experiments, we test ADWIN2 as a change detector rather than as an estimator, and compare it to Gama's method [7]. The measures of interest here are the rate of changes detected and the mean time until detection.

To do this, we feed ADWIN2 and Gama's change detector with four data streams of lengths $L = 2,000$, $10,000$, $100,000$ and $1,000,000$ bits, generated from a Bernoulli distribution of parameter μ. We keep $\mu = 0.2$ stationary during the first $L-1,000$ time steps, and then make it increase linearly during the last $1,000$ steps. We try different slopes: 0 (no change), 10^{-4}, $2{\cdot}10^{-4}$, $3{\cdot}10^{-4}$, and $4 \cdot 10^{-4}$.

To compare the rate of false negatives on an equal foot, we adjust ADWIN2 confidence parameter δ to have the same rate of false positives as Gama's method.

Table 7 shows the results. Rows are grouped in four parts, corresponding to the four values of L that we tested. For each value of L, we give the number of changes detected in the last $1,000$ samples (summed over all runs) and the mean and standard distribution of the time until the change is detected, in those runs where there is detection.

The first column gives the ratio of false positives. One observation we made is that Gama's method tends to detect many more changes early on (when the window is small) and less changes as the window grows. This explains that, on the first column, even if the ratio of false positives is the same, the average time until

Table 2: Comparative of ADWIN2 with other estimators using L_1 and $\delta = 0.1$

		ADWIN2		Fixed-sized Window					Fixed-sized flushing Window				
Stream	Period		Width	32	128	512	2048	8192	32	128	512	2048	8192
Scale	5000	0,05	503	0,07	0,04	0,06	0,17	0,16	0,07	0,04	0,05	0,11	0,15
Triangular	128	0,15	74	0,13	0,17	0,16	0,16	0,16	0,13	0,17	0,16	0,16	0,16
Triangular	512	0,09	140	0,08	0,12	0,16	0,16	0,16	0,09	0,12	0,16	0,16	0,16
Triangular	2048	0,05	314	0,07	0,06	0,11	0,16	0,16	0,07	0,06	0,09	0,16	0,16
Triangular	8192	0,03	657	0,07	0,04	0,04	0,11	0,16	0,07	0,04	0,04	0,06	0,16
Triangular	32768	0,02	935	0,07	0,03	0,02	0,04	0,11	0,07	0,03	0,02	0,02	0,03
Triangular	131072	0,02	1.099	0,07	0,03	0,02	0,02	0,03	0,07	0,03	0,02	0,01	0,01
Triangular	524288	0,02	1.107	0,07	0,03	0,02	0,01	0,01	0,07	0,03	0,02	0,01	0,01
Triangular	43	0,17	148	0,20	0,17	0,16	0,16	0,16	0,20	0,17	0,16	0,16	0,16
Triangular	424	0,10	127	0,09	0,13	0,15	0,16	0,16	0,10	0,13	0,15	0,16	0,16
Triangular	784	0,07	180	0,08	0,09	0,19	0,16	0,16	0,08	0,10	0,15	0,16	0,16
Triangular	5000	0,03	525	0,07	0,04	0,06	0,16	0,17	0,07	0,04	0,05	0,09	0,15
Square	128	0,14	45	0,18	0,30	0,30	0,30	0,30	0,20	0,30	0,30	0,30	0,30
Square	512	0,06	129	0,09	0,16	0,30	0,30	0,30	0,09	0,14	0,30	0,30	0,30
Square	2048	0,03	374	0,06	0,06	0,16	0,30	0,30	0,07	0,06	0,08	0,30	0,30
Square	8192	0,02	739	0,06	0,04	0,05	0,15	0,30	0,06	0,04	0,03	0,04	0,30
Square	32768	0,02	1.144	0,06	0,03	0,02	0,04	0,15	0,06	0,03	0,02	0,01	0,02
Square	131072	0,02	1.248	0,06	0,03	0,02	0,02	0,04	0,06	0,03	0,02	0,01	0,01

Table 3: Comparative of ADWIN2 with other estimators using L_1 and $\delta = 0.3$

		ADWIN2		Fixed-sized Window					Fixed-sized flushing Window				
Stream	Period		Width	32	128	512	2048	8192	32	128	512	2048	8192
Scale	5000	0,05	213	0,07	0,04	0,06	0,17	0,16	0,07	0,04	0,05	0,10	0,15
Triangular	128	0,13	48	0,13	0,17	0,16	0,16	0,16	0,13	0,17	0,16	0,16	0,16
Triangular	512	0,08	93	0,08	0,12	0,16	0,16	0,16	0,09	0,12	0,16	0,16	0,16
Triangular	2048	0,06	156	0,07	0,06	0,11	0,16	0,16	0,07	0,06	0,09	0,16	0,16
Triangular	8192	0,05	189	0,07	0,04	0,04	0,11	0,16	0,07	0,04	0,04	0,06	0,16
Triangular	32768	0,05	218	0,07	0,03	0,02	0,04	0,11	0,07	0,03	0,02	0,02	0,03
Triangular	131072	0,05	215	0,07	0,03	0,02	0,02	0,03	0,07	0,03	0,02	0,01	0,01
Triangular	524288	0,05	217	0,07	0,03	0,02	0,01	0,01	0,07	0,03	0,02	0,01	0,01
Triangular	43	0,17	49	0,20	0,17	0,16	0,16	0,16	0,20	0,17	0,16	0,16	0,16
Triangular	424	0,09	85	0,09	0,13	0,15	0,16	0,16	0,10	0,14	0,15	0,16	0,16
Triangular	784	0,07	109	0,08	0,09	0,19	0,16	0,16	0,08	0,10	0,15	0,16	0,16
Triangular	5000	0,05	184	0,07	0,04	0,06	0,16	0,17	0,07	0,04	0,05	0,09	0,15
Square	128	0,12	37	0,18	0,30	0,30	0,30	0,30	0,20	0,30	0,30	0,30	0,30
Square	512	0,07	93	0,09	0,16	0,30	0,30	0,30	0,09	0,14	0,30	0,30	0,30
Square	2048	0,05	175	0,06	0,06	0,16	0,30	0,30	0,07	0,06	0,08	0,30	0,30
Square	8192	0,05	220	0,06	0,04	0,05	0,15	0,30	0,06	0,04	0,03	0,04	0,30
Square	32768	0,05	273	0,06	0,03	0,02	0,04	0,15	0,06	0,03	0,02	0,02	0,02
Square	131072	0,04	250	0,06	0,03	0,02	0,02	0,04	0,06	0,03	0,01	0,01	0,01

Table 4: Comparative of ADWIN2 with other estimators using L_2 and $\delta = 0.1$

		ADWIN2		Fixed-sized Window					Fixed-sized flushing Window				
Stream	Period		Width	32	128	512	2048	8192	32	128	512	2048	8192
Scale	5000	0.06	501	0.08	0.04	0.06	0.19	0.19	0.08	0.05	0.07	0.13	0.17
Triangular	128	0.19	75	0.18	0.19	0.19	0.19	0.19	0.18	0.19	0.19	0.19	0.19
Triangular	512	0.12	140	0.12	0.17	0.19	0.19	0.19	0.12	0.17	0.19	0.19	0.19
Triangular	2048	0.08	320	0.09	0.09	0.16	0.19	0.19	0.09	0.10	0.14	0.19	0.19
Triangular	8192	0.05	666	0.08	0.06	0.09	0.16	0.19	0.09	0.06	0.08	0.11	0.19
Triangular	32768	0.04	905	0.08	0.05	0.05	0.08	0.16	0.08	0.05	0.04	0.06	0.08
Triangular	131072	0.04	1,085	0.08	0.04	0.03	0.04	0.08	0.08	0.04	0.03	0.03	0.04
Triangular	524288	0.04	1,064	0.08	0.04	0.02	0.02	0.03	0.08	0.04	0.02	0.02	0.02
Triangular	43	0.20	146	0.23	0.20	0.19	0.19	0.19	0.23	0.20	0.19	0.19	0.19
Triangular	424	0.13	126	0.12	0.18	0.17	0.19	0.19	0.13	0.18	0.17	0.19	0.19
Triangular	784	0.11	181	0.11	0.14	0.22	0.19	0.19	0.11	0.14	0.19	0.19	0.19
Triangular	5000	0.06	511	0.09	0.07	0.11	0.20	0.20	0.09	0.07	0.10	0.14	0.19
Square	128	0.21	45	0.25	0.30	0.30	0.30	0.30	0.26	0.30	0.30	0.30	0.30
Square	512	0.12	129	0.14	0.25	0.30	0.30	0.30	0.15	0.23	0.30	0.30	0.30
Square	2048	0.07	374	0.09	0.13	0.25	0.30	0.30	0.10	0.12	0.18	0.30	0.30
Square	8192	0.05	765	0.08	0.07	0.12	0.24	0.30	0.08	0.07	0.09	0.13	0.30
Square	32768	0.04	1,189	0.07	0.05	0.06	0.12	0.24	0.07	0.04	0.05	0.07	0.09
Square	131072	0.04	1,281	0.07	0.04	0.03	0.06	0.12	0.07	0.04	0.03	0.03	0.05

Table 5: Comparative of ADWIN2 with other estimators using L_2 and $\delta = 0.3$

		ADWIN2		Fixed-sized Window					Fixed-sized flushing Window				
Stream	Period		Width	32	128	512	2048	8192	32	128	512	2048	8192
Scale	5000	0.08	210	0.08	0.04	0.06	0.18	0.19	0.08	0.05	0.07	0.13	0.17
Triangular	128	0.17	48	0.18	0.19	0.19	0.19	0.19	0.18	0.19	0.19	0.19	0.19
Triangular	512	0.12	93	0.12	0.17	0.19	0.19	0.19	0.12	0.17	0.19	0.19	0.19
Triangular	2048	0.09	153	0.09	0.09	0.16	0.19	0.19	0.09	0.10	0.14	0.19	0.19
Triangular	8192	0.08	193	0.08	0.06	0.09	0.16	0.19	0.09	0.06	0.08	0.11	0.19
Triangular	32768	0.08	213	0.08	0.05	0.05	0.08	0.16	0.08	0.05	0.04	0.06	0.08
Triangular	131072	0.08	223	0.08	0.04	0.03	0.04	0.08	0.08	0.04	0.03	0.03	0.04
Triangular	524288	0.08	222	0.08	0.04	0.02	0.02	0.03	0.08	0.04	0.02	0.02	0.02
Triangular	43	0.21	49	0.23	0.19	0.19	0.19	0.19	0.23	0.19	0.19	0.19	0.19
Triangular	424	0.12	85	0.12	0.18	0.17	0.19	0.19	0.13	0.18	0.17	0.19	0.19
Triangular	784	0.11	109	0.11	0.14	0.22	0.19	0.19	0.11	0.14	0.19	0.19	0.19
Triangular	5000	0.08	181	0.09	0.07	0.11	0.20	0.20	0.09	0.07	0.10	0.14	0.19
Square	128	0.18	37	0.25	0.30	0.30	0.30	0.30	0.26	0.30	0.30	0.30	0.30
Square	512	0.12	93	0.14	0.25	0.30	0.30	0.30	0.15	0.23	0.30	0.30	0.30
Square	2048	0.09	174	0.09	0.13	0.25	0.30	0.30	0.10	0.12	0.18	0.30	0.30
Square	8192	0.08	243	0.08	0.07	0.12	0.25	0.30	0.08	0.07	0.09	0.13	0.30
Square	32768	0.08	262	0.07	0.05	0.06	0.12	0.24	0.07	0.05	0.05	0.07	0.09
Square	131072	0.08	253	0.07	0.04	0.03	0.06	0.12	0.07	0.04	0.03	0.03	0.04

Table 6: Relative error using small probabilities

Prob.	ADWIN2	Fixed-sized Window					Fixed-sized flushing Window				
		32	128	512	2048	8192	32	128	512	2048	8192
1/32	0.06	0.72	0.38	0.20	0.10	0.05	0.72	0.38	0.20	0.10	0.05
1/64	0.04	1.21	0.53	0.27	0.14	0.07	1.21	0.53	0.27	0.14	0.07
1/128	0.02	1.56	0.73	0.39	0.20	0.10	1.56	0.73	0.39	0.20	0.10
1/256	0.02	1.76	1.21	0.53	0.28	0.14	1.76	1.21	0.53	0.28	0.14
1/512	0.03	1.89	1.56	0.74	0.40	0.22	1.89	1.56	0.74	0.40	0.22
1/1024	0.04	1.89	1.72	1.18	0.52	0.28	1.89	1.72	1.18	0.52	0.28
1/2048	0.04	1.97	1.88	1.55	0.70	0.36	1.97	1.88	1.55	0.70	0.36
1/4096	0.10	1.97	1.93	1.76	1.22	0.55	1.97	1.93	1.76	1.22	0.55
1/8192	0.10	1.93	1.91	1.83	1.50	0.66	1.93	1.91	1.83	1.50	0.66
1/16384	0.22	2.08	2.06	2.02	1.83	1.31	2.08	2.06	2.02	1.83	1.31
1/32768	0.37	1.85	1.85	1.83	1.75	1.49	1.85	1.85	1.83	1.75	1.49

the first false positive is produced is much smaller for Gama's method.

The last four columns describe the results when change does occur, with different slopes. ADWIN2 detects change more often, with the exception of the $L = 2,000$ experiment. As the number of samples increases, the percentage of changes detected decreases in Gama's methodology; as discussed early, this is to be expected since it takes a long time for Gama's method to overcome the weight of past examples. In contrast, ADWIN2 maintains a good rate of detected changes, largely independent of the number of the number of past samples $L-1,000$. One can observe the same phenomenon as before: even though Gama's method detects less changes, the average time until detection (when detection occurs) is smaller.

5 Example 1: Naïve Bayes Predictor

5.1 An Incremental Naïve Bayes Predictor Let $x_1,\ldots,\ x_k$ be k discrete attributes, and assume that x_i can take n_i different values. Let C be the class attribute, which can take n_C different values. Recall that upon receiving an unlabelled instance $I = (x_1 = v_1, \ldots, x_k = v_k)$, the Naïve Bayes predictor computes a "probability" of I being in class c as:

$$\begin{aligned}\Pr[C = c|I] &\cong \prod_{i=1}^{k} \Pr[x_i = v_i|C = c] \\ &= \Pr[C = c] \cdot \prod_{i=1}^{k} \frac{\Pr[x_i = v_i \wedge C = c]}{\Pr[C = c]}\end{aligned}$$

The values $\Pr[x_i = v_j \wedge C = c]$ and $\Pr[C = c]$ are estimated from the training data. Thus, the summary of the training data is simply a 3-dimensional table that stores for each triple (x_i, v_j, c) a count $N_{i,j,c}$ of training instances with $x_i = v_j$, together with a 1-dimensional table for the counts of $C = c$. This algorithm is naturally incremental: upon receiving a new example (or a batch of new examples), simply increment the relevant counts. Predictions can be made at any time from the current counts.

We compare two time-change management strategies. The first one uses a static model to make predictions. This model is rebuilt every time that an external change detector module detects a change. We use Gama's detection method and ADWIN2 as change detectors. Gama's method generates a warning example some time before actually declaring change; see [7] for the details; the examples received between the warning and the change signal are used to rebuild the model. In ADWIN2, we use the examples currently stored in the window to rebuild the static model.

The second one is incremental: we simply create an instance $A_{i,j,c}$ of ADWIN2 for each count $N_{i,j,c}$, and one for each value c of C. When a labelled example is processed, add a 1 to $A_{i,j,c}$ if $x_i = v \wedge C = c$, and a 0 otherwise, and similarly for N_c. When the value of $\Pr[x_i = v_j \wedge C = c]$ is required to make a prediction, compute it using the estimate of $N_{i,j,c}$ provided by $A_{i,j,c}$. This estimate varies automatically as $\Pr[x_i = v_j \wedge C = c]$ changes in the data.

Note that different $A_{i,j,c}$ may have windows of different lengths at the same time. This will happen when the distribution is changing at different rates for different attributes and values, and there is no reason to sacrifice accuracy in all of the counts $N_{i,j,c}$, only because a few of them are changing fast. This is the intuition why this approach may give better results than one monitoring the global error of the predictor: it has more accurate information on at least some of the statistics that are used for the prediction.

Table 7: Change detection experiments. Each entry contains "x (y)" where x is average and y is standard deviation.

Slope	0	10^{-4}	$2 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$4 \cdot 10^{-4}$
$2 \cdot 10^3$ **samples,** 10^3 **trials**					
Detection time (Gama)	854 (462)	532 (271)	368 (248)	275 (206)	232 (178)
%runs detected (Gama)	10.6	58.6	97.2	100	100
Detection time ADWIN2	975 (607)	629 (247)	444 (210)	306 (171)	251 (141)
%runs detected (ADWIN2)	10.6	39.1	94.6	93	95
10^4 **samples, 100 trials**					
Detection time (Gama)	2,019 (2,047)	498 (416)	751 (267)	594 (287)	607 (213)
%runs detected (Gama)	14	13	38	71	84
Detection time ADWIN2)	4,673 (3142)	782 (195)	595 (100)	450 (96)	367 (80)
%runs detected (ADWIN2)	14	40	79	90	87
10^5 **samples, 100 trials**					
Detection time (Gama)	12,164 (17,553)	127 (254)	206 (353)	440 (406)	658 (422)
%runs detected (Gama)	12	4	7	11	8
Detection time ADWIN2)	47,439 (32,609)	878 (102)	640 (101)	501 (72)	398 (69)
%runs detected (ADWIN2)	12	28	89	84	89
10^6 **samples, 100 trials**					
Detection time (Gama)	56,794 (142,876)	1 (1)	1 (0)	1 (0)	180 (401)
%runs detected (Gama)	22	5	5	3	5
Detection time ADWIN2)	380,738 (289,242)	898 (80)	697 (110)	531 (89)	441 (71)
%runs detected (ADWIN2)	22	15	77	80	83

5.2 Synthetic data Experiments The experiments with synthetic data use a changing concept based on a rotating hyperplane explained in [11]. A hyperplane in d-dimensional space is the set of points x that satisfy

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

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 use 2 classes, $d = 8$ attributes, and 2 values (0 and 1) per attribute. The different weights w_i of the hyperplane vary over time, at different moments and different speeds for different attributes i. All w_i start at 0.5 and we restrict to two w_i's varying at the same time, to a maximum value of 0.75 and a minimum of 0.25.

To test the performance of our two Naïve Bayes methodologies we do the following: At every time t, we build a static Naïve Bayes model M_t using a data set of 10,000 points generated from the distribution at time t. Model M_t is taken as a "baseline" of how well a Naïve Bayes model can do on this distribution. Then we generate 1000 fresh points from the current distribution and use them to compute the error rate of both the static model M_t and the different models built dynamically from the points seen so far. The ratio of these error rates is averaged over all the run.

Table 8 shows accuracy results. The "%Static" column shows the accuracy of the static model M_t – it is the same for all rows, except for small random fluctuations. The "%Dynamic" column is the accuracy of the model built dynamically using the estimator in the row. The last column in the table shows the quotient of columns 1 and 2, i.e., the relative accuracy of the dynamically vs. statically built models. In all NB experiments we show in boldface the result for ADWIN2 and the best result. It can be seen that the incremental time-change management model (using one instance of ADWIN2 per count) outperforms fixed-size windows and the models based on detecting change and rebuilding the model. Among these, the one using ADWIN2 as a change detector is better than that using Gama's method.

5.3 Real-world data experiments We test the performance of our Naïve Bayes predictors using the Electricity Market Dataset described by M. Harries [8] and used by Gama [7]. This dataset is a real-world

Table 8: Naïve Bayes, synthetic data benchmark

	Width	%Static	%Dynamic	% Dynamic/Static
Gama Change Detection		94.74%	58.02%	61.24%
ADWIN2 Change Detection		94.73%	70.72%	74.66%
ADWIN2 for counts		94.77%	**94.16%**	**99.36%**
Fixed-sized Window	32	94.74%	70.34%	74.24%
Fixed-sized Window	128	94.76%	80.12%	84.55%
Fixed-sized Window	512	94.73%	88.20%	93.10%
Fixed-sized Window	2048	94.75%	92.82%	97.96%
Fixed-sized flushing Window	32	94.74%	70.34%	74.25%
Fixed-sized flushing Window	128	94.75%	80.13%	84.58%
Fixed-sized flushing Window	512	94.73%	88.17%	93.08%
Fixed-sized flushing Window	2048	94.72%	92.86%	98.03%

dataset where we do not know when drift occurs or if there is drift, hence it is not possible to build a static model for comparison as we did before.

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 45312 instances dated from 7 May 1996 to 5 December 1998. 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. Each example on the dataset has 5 fields, the day of week, the time stamp, the NSW electricity demand, the Vic electricity demand, the scheduled electricity transfer between states and the class label. 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.

At each time step, we train a static model using the last 48 samples received. We compare this static model with other models, also on the last 48 samples. Table 9 shows accuracy results of the different methods on this dataset. Again, in each column (a test), we show in boldface the result for ADWIN2 and for the best result.

The results are similar to those obtained with the hyperplane data: ADWIN2 applied in the incremental time-change model (to estimate probabilities) does much better than all the others, with the exception of the shortest fixed-length window, which achieves 86.44% of the static performance compared to ADWIN2 s 83.62%. The reason for this anomaly is due to the nature of this particular dataset: by visual inspection, one can see that it contains a lot of short runs (length 10 to 20) of identical values, and therefore a myopic strategy (i.e., a short window) gives best results. ADWIN2 behaves accordingly and shortens its window as much as it can, but the formulas involved do not allow windows as short as 10 elements. In fact, we have tried replicating each instance in the dataset 10 times (so there are runs of length 100 to 200 of equal values), and then case ADWIN2 becomes the winner again.

We also test the prediction accuracy of these methods. We compare, as before, a static model generated at each time t to the other models, and evaluate them asking to predict the instance that will arrive at time $t+1$. The static model is computed training on the last 24 samples. The results are in Table 10. In this experiment, ADWIN2 outperforms clearly other time-change models. Generally, the incremental time-change management model does much better than the static model that refreshes its NB model when change is detected.

6 Example 2: k-means Clustering

6.1 An Incremental k-means clusterer In contrast to Naïve Bayes, it is not completely obvious how to give an incremental version of the k-means clustering algorithm.

We adapt in essence the incremental version from [16]. In that version, every new example is added to the cluster with nearest centroid, and every r steps a recomputation phase occurs, which recomputes both the assignment of points to clusters and the centroids. To balance accuracy and computation time, r is chosen in [16] to be the square root of the number of points seen so far. In our case, this latter rule is extended to react to changes in the data distribution.

We incorporate adaptive windowing to this algorithm in the following way. Let k and d be the number of centroids and attributes. We add an instance W_{ij} of our algorithm for every attribute centroid i and every attribute j, hence kd instances. The algorithm still interleaves phases in which centroids are just incremen-

Table 9: Naïve Bayes, Electricity data benchmark, testing on last 48 items

	Width	%Static	%Dynamic	% Dynamic/Static
Gama Change Detection		91.62%	45.94%	50.14%
ADWIN2 Change Detection		91.62%	60.29%	65.81%
ADWIN2 for counts		91.62%	**76.61%**	**83.62%**
Fixed-sized Window	32	91.55%	**79.13%**	**86.44%**
Fixed-sized Window	128	91.55%	72.29%	78.97%
Fixed-sized Window	512	91.55%	68.34%	74.65%
Fixed-sized Window	2048	91.55%	65.02%	71.02%
Fixed-sized flushing Window	32	91.55%	78.57%	85.83%
Fixed-sized flushing Window	128	91.55%	73.46%	80.24%
Fixed-sized flushing Window	512	91.55%	69.65%	76.08%
Fixed-sized flushing Window	2048	91.55%	66.54%	72.69%

Table 10: Naïve Bayes, Electricity data benchmark, testing on next instance

	Width	%Static	%Dynamic	% Dynamic/Static
Gama Change Detection		94.40%	45.87%	48.59%
ADWIN2 Change Detection		94.40%	46.86%	49.64%
ADWIN2 for counts		94.39%	**72.71%**	**77.02%**
Fixed-sized Window	32	94.39%	71.54%	75.79%
Fixed-sized Window	128	94.39%	68.78%	72.87%
Fixed-sized Window	512	94.39%	67.14%	71.13%
Fixed-sized Window	2048	94.39%	64.25%	68.07%
Fixed-sized flushing Window	32	94.39%	71.62%	75.88%
Fixed-sized flushing Window	128	94.39%	70.12%	74.29%
Fixed-sized flushing Window	512	94.39%	68.02%	72.07%
Fixed-sized flushing Window	2048	94.39%	65.60%	69.50%

tally modified with incoming points and phases where global recomputation of centroids takes place. The second type of phase can occur each time we detect change. We use two criteria. First, when any of the $W_{i\ell}$ windows shrinks, we take this as a signal that the position of centroid i may have changed. In the case of estimators that use windows of a fixed size s, when any of the windows is full of new s elements we take this as as indicator of change in the position of centroids. And in the estimators that use windows of a fixed size with change detection, every time it detects change, we use this as a signal that the position of a centroid may have changed.

The second criterion is to recompute when the average point distance to theirs centroids has changed more than an ϵ factor where ϵ is user-specified. This is taken as an indication that a certain number of points may change from cluster i to cluster j or vice-versa if recomputation takes place now.

6.2 Experiments We build a model of k-means clustering, using a window estimator for each centroid coordinate. We compare the performance of our model with a static one, measuring the sum of the distances of each data point to each centroid assigned.

The synthetic data used in our experiments consist of a sample of 10^6 points generated from a k-gaussian distribution with some fixed variance σ^2, and centered in our k moving centroids. Each centroid moves according to a constant velocity. We try different velocities v and values of σ in different experiments. Tables 11 and 12 shows the results of computing the distance from 100 random points to their centroids. We observe that `ADWIN2` outperforms other estimators in essentially all settings.

7 Time and Memory Requirements

In the experiments above we have only discussed the performance in terms of error rate, and not time or memory usage. Certainly, this was not our main goal in this paper, and we have in no way tried to optimize our implementations in either time or memory (as is clearly indicated by the choice of Java as programming language). Let us, however, mention some rough figures about time and memory, since they suggest that our approach can be fairly competitive after some optimization work.

All programs were implemented in Java Standard Edition. The experiments were performed on a 3.0 GHz Pentium PC machine with 1 Gigabyte main memory, running Microsoft Windows XP. The Sun Java 2 Runtime Environment, Standard Edition (build 1.5.0 06-b05) was used to run all benchmarks.

Consider first the experiments on `ADWIN2` alone. A bucket formed by an integer plus a real number uses 9 bytes. Therefore, about 540 bytes store a sliding window of 60 buckets. In the boolean case, we could use only 5 bytes per bucket, which reduces our memory requirements to 300 bytes per window of 60 buckets. Note that 60 buckets, with our choice of $M = 5$ suffice to represent a window of length about $2^{60/5} = 4096$.

In the experiment comparing different estimators (Tables 2,3,4 and 5), the average number of buckets used by `ADWIN2` was 45,11, and the average time spent was 23 seconds to process the 10^6 samples, which is quite remarkable. In the Naïve Bayes experiment (Table 8), it took an average of 1060 seconds and 2000 buckets to process 10^6 samples by 34 estimators. This means less than 32 seconds and 60 buckets per estimator. The results for k-means were similar: We executed the k-means experiments with $k = 5$ and two attributes, with 10 estimators and 10^6 sample points using about an average of 60 buckets and 11.3 seconds for each instance of `ADWIN2`.

8 Conclusions

We have described a new method for dealing with distribution change and concept drift when learning from data sequences that may vary with time. We developed an algorithm `ADWIN` using sliding windows whose size is recomputed online according to the rate of change observed from the data in the window itself. This delivers the user from having to choose any parameter (for example, window size), a step that most often ends up being guesswork. So, client algorithms can simply assume that `ADWIN` stores the currently relevant data.

We proposed a time- and memory-efficient version of our algorithm, `ADWIN2`, that checks $O(\log W)$ cutpoints, uses $O(\log W)$ memory words, and whose processing time per example is $O(\log^2 W)$ (worst-case) and $O(\log W)$ (amortized).

We tested our approach using synthetic, time-changing data streams and a real dataset, showed the improvements of our estimator over fixed-size windows and one of the most recently proposed variable-length window strategies [7]. These tests studied `ADWIN2` as an estimator, as a change detector, and in combination with the Naïve-Bayes predictor. In one line, the results indicate that `ADWIN2` does almost as well as the best in all contexts, and much better than any other in at least one context. This shows that `ADWIN2` really adapts its behavior to the characteristics of the problem at hand.

Future work must include more thorough experimentation, both with more real-world datasets and in combination with other learning algorithms. As mentioned before, we are currently working in the integration of `ADWIN2` in decision-tree induction methods.

Table 11: k-means sum of distances to centroids, with $k = 5$, 10^6 samples and centroid velocities of 10^{-5} and different variances.

		$\sigma = 0.15$		$\sigma = 0.3$		$\sigma = 0.6$	
	Width	Static	Dynamic	Static	Dynamic	Static	Dynamic
ADWIN2		9.72	16.63	19.42	26.71	38.83	47.32
Fixed-sized Window	32	9.72	18.46	19.42	27.92	38.83	48.79
Fixed-sized Window	128	9.72	26.08	19.42	35.87	38.83	58.65
Fixed-sized Window	512	9.72	28.20	19.42	38.13	38.83	61.22
Fixed-sized Window	2048	9.72	29.84	19.42	39.24	38.83	61.96
Fixed-sized Window	8192	9.72	32.79	19.42	40.58	38.83	63.09
Fixed-sized Window	32768	9.72	35.12	19.42	40.93	38.83	64.40
Fixed-sized flushing Window	32	9.72	29.29	19.42	34.19	38.83	57.54
Fixed-sized flushing Window	128	9.72	31.49	19.42	39.06	38.83	61.18
Fixed-sized flushing Window	512	9.72	30.10	19.42	39.47	38.83	62.44
Fixed-sized flushing Window	2048	9.72	29.68	19.42	39.38	38.83	62.01
Fixed-sized flushing Window	8192	9.72	31.54	19.42	39.86	38.83	62.82
Fixed-sized flushing Window	32768	9.72	36.21	19.42	41.11	38.83	65.54

Table 12: k-means sum of distances to centroids, with $k = 5$, 10^6 samples with variance $\sigma = 0.3$ and different centroid velocities.

		$v = 10^{-3}$		$v = 10^{-2}$		$v = 0$	
	Width	Static	Dynamic	Static	Dynamic	Static	Dynamic
ADWIN2		19.41	28.13	19.41	28.60	19.41	27.63
Fixed-sized Window	32	19.41	30.60	19.41	29.89	19.41	28.62
Fixed-sized Window	128	19.41	39.28	19.41	37.62	19.41	36.41
Fixed-sized Window	512	19.41	41.74	19.41	39.47	19.41	38.32
Fixed-sized Window	2048	19.41	42.36	19.41	39.76	19.41	38.67
Fixed-sized Window	8192	19.41	42.73	19.41	40.24	19.41	38.21
Fixed-sized Window	32768	19.41	44.13	19.41	41.81	19.41	37.12
Fixed-sized flushing Window	32	19.41	38.82	19.41	34.92	19.41	29.44
Fixed-sized flushing Window	128	19.41	41.30	19.41	38.79	19.41	42.72
Fixed-sized flushing Window	512	19.41	42.14	19.41	39.80	19.41	44.04
Fixed-sized flushing Window	2048	19.41	42.43	19.41	40.37	19.41	44.37
Fixed-sized flushing Window	8192	19.41	43.18	19.41	40.92	19.41	44.45
Fixed-sized flushing Window	32768	19.41	44.94	19.41	70.07	19.41	44.47

On another line, we are investigating the possibility of making better predictions from the contents of the window. Right now, the prediction at every time step is simply the average of the elements in the window. But if change is occurring, it may be better to give more weight to most recent examples – the question is then how much more weight? We have recently reported on using Kalman filters together with an earlier (less sensitive) version of `ADWIN2`, where the Kalman filter is adaptively tuned to provide better estimations [4].

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. 21st ACM Symposium on Principles of Database Systems*, 2002.

[2] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[3] P. Bartlett, S. Ben-David, and S. Kulkarni. Learning changing concepts by exploiting the structure of change. *Machine Learning*, 41(2):153–174, 2000.

[4] A. Bifet and R. Gavaldà. Kalman filters and adaptive windows for learning in data streams. In *Proc. 9th Intl. Conference on Discovery Science*, volume 4265, pages 28–40. Springer-Verlag Lecture Notes in Artificial Intelligence, 2006.

[5] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *Proc. 22nd ACM Symposium on Principles of Database Systems*, 2003.

[6] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 14(1):27–45, 2002.

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

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

[9] D. Helmbold and P. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14(1):27–45, 1994.

[10] M. Herbster and M. K. Warmuth. Tracking the best expert. In *Intl. Conf. on Machine Learning*, pages 286–294, 1995.

[11] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *7th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, 2001. ACM Press.

[12] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proc. 30th VLDB Conf., Toronto, Canada*, 2004.

[13] R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *Proc. 17th Intl. Conf. on Machine Learning*, pages 487 – 494, 2000.

[14] M. Last. Online classification of nonstationary data streams. *Intelligent Data Analysis*, 6(2):129–147, 2002.

[15] S. Muthukrishnan. Data streams: Algorithms and applications. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[16] C. Ordonez. Clustering binary data streams with k-means. In *ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, 2003.

[17] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

A Appendix: Proof of Theorem 1

Part 1. Assume $\mu_{W_0} = \mu_{W_1} = \mu_W$ as null hypothesis. We show that for any partition W as W_0W_1 we have probability at most δ/n that `ADWIN` decides to shrink W to W_1, or equivalently,

$$\Pr[\,|\hat{\mu}_{W_1} - \hat{\mu}_{W_0}| \geq \epsilon_{cut}\,] \leq \delta/n.$$

Since there are at most n partitions W_0W_1, the claim follows by the union bound. Note that, for every real number $k \in (0,1)$, $|\hat{\mu}_{W_1} - \hat{\mu}_{W_0}| \geq \epsilon_{cut}$ can be decomposed as

$$\Pr[\,|\hat{\mu}_{W_1} - \hat{\mu}_{W_0}| \geq \epsilon_{cut}\,] \leq \Pr[\,|\hat{\mu}_{W_1} - \mu_W| \geq k\epsilon_{cut}\,] + \Pr[\,|\mu_W - \hat{\mu}_{W_0}| \geq (1-k)\epsilon_{cut})\,]. \tag{1.2}$$

Applying the Hoeffding bound, we have then

$$\Pr[\,|\hat{\mu}_{W_1} - \hat{\mu}_{W_0}| \geq \epsilon_{cut}\,] \leq 2\exp(-2(k\,\epsilon_{cut})^2\, n_0) + 2\exp(-2((1-k)\,\epsilon_{cut})^2\, n_1) \tag{1.3}$$

To approximately minimize the sum, we choose the value of k that makes both probabilities equal, i.e. such that

$$(k\,\epsilon_{cut})^2\, n_0 = ((1-k)\,\epsilon_{cut})^2\, n_1.$$

which is $k = \sqrt{n_1/n_0}/(1+\sqrt{n_1/n_0})$. For this k, we have precisely

$$(k\,\epsilon_{cut})^2\, n_0 = \frac{n_1 n_0}{(\sqrt{n_0}+\sqrt{n_1})^2}\,\epsilon_{cut}^2 \leq \frac{n_1 n_0}{(n_0+n_1)}\,\epsilon_{cut}^2 = m\,\epsilon_{cut}^2.$$

Therefore, in order to have

$$\Pr[\,|\hat{\mu}_{W_1} - \hat{\mu}_{W_0}| \geq \epsilon_{cut}\,] \leq \frac{\delta}{n}$$

it suffices to have

$$4\exp(-2m\,\epsilon_{cut}^2) \leq \frac{\delta}{n}$$

which is satisfied by

$$\epsilon_{cut} = \sqrt{\frac{1}{2m}\,\ln\frac{4n}{\delta}}\,.$$

Part 2) Now assume $|\mu_{W_0} - \mu_{W_1}| > 2\epsilon_{cut}$. We want to show that $\Pr[\,|\hat{\mu}_{W_1} - \hat{\mu}_{W_0}| \leq \epsilon_{cut}\,] \leq \delta$, which means that with probability at least $1-\delta$ change is detected and the algorithm cuts W to W_1. As before, for any $k \in (0,1)$, we can decompose $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \leq \epsilon_{cut}$ as

$$\begin{aligned}\Pr[\,|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \leq \epsilon_{cut}\,] &\leq \Pr[\,(|\hat{\mu}_{W_0} - \mu_{W_0}| \geq k\epsilon_{cut}) \cup (|\hat{\mu}_{W_1} - \mu_{W_1}| \geq (1-k)\epsilon_{cut})\,] \\ &\leq \Pr[\,|\hat{\mu}_{W_0} - \mu_{W_0}| \geq k\epsilon_{cut}\,] + \Pr[\,|\hat{\mu}_{W_1} - \mu_{W_1}| \geq (1-k)\epsilon_{cut}\,].\end{aligned}$$

To see the first inequality, observe that if $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \leq \epsilon_{cut}$, $|\hat{\mu}_{W_0} - \mu_{W_0}| \leq k\epsilon_{cut}$, and $|\hat{\mu}_{W_1} - \mu_{W_1}| \leq (1-k)\epsilon_{cut}$ hold, by the triangle inequality we have

$$|\mu_{W_0} - \mu_{W_1}| \leq |\hat{\mu}_{W_0} + k\epsilon_{cut} - \hat{\mu}_{W_1} + (1-k)\epsilon_{cut}| \leq |\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| + \epsilon_{cut} \leq 2\epsilon_{cut},$$

contradicting the hypothesis. Using the Hoeffding bound, we have then

$$\Pr[\,|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_{cut}\,] \leq 2\,\exp(-2(k\,\epsilon_{cut})^2\, n_0) + 2\,\exp(-2((1-k)\,\epsilon_{cut})^2\, n_1).$$

Now, choose k as before to make both terms equal. By the calculations in Part 1 we have

$$\Pr[\,|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_{cut}\,] \leq 4\exp(-2\,m\,\epsilon_{cut}^2) \leq \frac{\delta}{n} \leq \delta,$$

as desired.