Kalman Filters and Adaptive Windows for Learning in Data Streams *

Albert Bifet and Ricard Gavaldà

Universitat Politècnica de Catalunya, Barcelona, Spain
{abifet,gavalda}@lsi.upc.edu

July 12th, 2006

Abstract. We study the combination of Kalman filter and a recently proposed algorithm for dynamically maintaining a sliding window, for learning from streams of examples. We integrate this idea into two well-known learning algorithms, the Naïve Bayes algorithm and the k-means clusterer. We show on synthetic data that the new algorithms do never worse, and in some cases much better, than the algorithms using only memoryless Kalman filters or sliding windows with no filtering.

1 Introduction

We deal with the problem of distribution and concept drift when learning from streams of incoming data. We study the combination of a classical estimation method in automatic control theory, the Kalman filter, with the also classical idea in machine learning of using a window of recently seen data items for learning. Many of the previous works in the machine learning area use windows of a fixed length. We use instead an algorithm that we proposed recently [3] for adaptively changing the size of the window in reaction to changes observed in the data.

In automatic control theory, many modern complex systems may be classed as estimation systems, combining several sources of (often redundant) data in order to arrive at an estimate of some unknown parameters. Among such systems are terrestrial or space navigators for estimating such parameters as position, velocity, and altitude, and radar systems for estimating position and velocity.

Estimation is often characterized as prediction, filtering or smoothing, depending on the intended objectives and the available observational information. Prediction usually implies the extension in some manner of the domain of validity of the information. Filtering usually refers to the extraction of the true signal from the observations. Smoothing usually implies the elimination of some noisy or useless component in the observed data.

One of the most widely used estimation algorithms is the Kalman filter, an algorithm that generates estimates of variables of the system being controlled by

* Partially supported by the 6th Framework Program of EU through the integrated project DELIS (#001907), by the EU PASCALNetwork of Excellence, IST-2002-506778, and by the DGICYT MOISES-BAR project, TIN2005-08832-C03-03.

processing available sensor measurements. Kalman filtering and related estimation algorithms have proved tremendously useful in a large variety of settings. Automatic machine learning is but one of them; see [17, 10] among many others. There is however an important difference in the control theory and machine learning settings: In automatic control, we assume that system parameters are known or easily detectable; these parameters are physical properties of devices, and therefore fixed. In contrast, in most machine learning situations the distribution that generates the examples is totally unknown, and there is no obvious way to measure any of its statistics, other than estimating them from the data. In addition, these statistics may vary impredictably over time, either continuously at a slow rate, or abruptly from time to time.

Besides the occasional use of filtering, most previous work on learning and time change has used variations of the sliding-window idea: at every moment, one window (or more) is kept containing the most recently read examples, and only those examples are considered relevant for learning. A critical point in this approach is the choice of a window size, and several strategies have been proposed.

The first and easiest strategy is deciding (or asking the user for) a window size W and keeping it fixed through the execution of the algorithm (see e.g. [6, 7, 18]). In order to detect change, one can keep a reference window with data from the past, also of some fixed size, and decide that change has occurred if some statistical test indicates that the distributions in the reference and current windows differ.

This strategy can work well if information on the time scale of change is available – which may be too much too ask in many contexts, and especially too much to ask from the user. Furthermore, there may be no "right" time scale: a single data source may experience periods of sudden changes, periods with slow but continuous change, and periods where the distribution remains almost stationary.

In general, the fixed-size approach is caught in the following trade-off, discussed explicitly in [6]: we would like to choose a "large" value for W so that in periods with little change we accumulate many examples to work on, which leads to more accurate learning; on the other hand, we would like a "small" W, so that we can react quickly to changes. Conversely, too large a W will make algorithms insensitive to change, and too small a W will lead to large variance, hence inaccuracy in the learned model.

A second approach [8, 12] uses windows of variable length, and monitors the evolution of the model's error to decide whether change has occurred. In this case, the a-priori assumption on the rate of change is usually hidden in the way that the decision is made. E.g., one maintains a *fixed-size* window of most recent examples on which the "current error rate" is measured. Still another approach is to consider that examples "decay" over time [4]: in this case, the a-priori assumption on the time-scale of change is the choice of a "decay constant" or "decay function" to weight the examples.

We have recently proposed another strategy [3]: keeping a window whose length is adjusted dynamically to reflect changes in the data. When change seems to be occurring, as indicated by some statistical test, the window is shrunk to keep only data items that still seem to be valid. When data seems to be stationary, the window is enlarged to work on more data and reduce variance. Our algorithm, called `ADWIN` for Adaptive Windowing, can be implemented with a low amount of memory and time per data item, and experimentally outperforms every window of a fixed size S when the time scale of change is either much smaller or much larger than S. In a way, `ADWIN` adjusts its window size to "best" one for the data it is seeing.

1.1 Proposal and Results of this Paper

In this paper, we combine `ADWIN` and Kalman filter and compare experimentally the performance of the resulting algorithm, `K-ADWIN`, with other estimator algorithms. The intuition why this combination should be better than `ADWIN` alone or the Kalman filter alone is as follows.

The Kalman filter is a memoryless algorithm, and it can benefit from having a memory aside. In particular, running a Kalman filter requires knowledge of at least two parameters of the system, named *state covariance* and *measurement covariance*, that should be estimated a priori. These are generally difficult to measure in the context of learning from a data stream, and in addition they can vary over time. The window that `ADWIN` maintains adaptively is guaranteed to contain up-to-date examples from which the current value of these covariances can be estimated and used in the Kalman filter.

On the other hand, `ADWIN` is somewhat slow in detecting a gradual change, because it gives the same weight to all examples in the window – it is what we will call a *linear* estimator. If there is a slow gradual change, the most recent examples should be given larger weight. This is precisely what the Kalman filter does in its estimation.

As in [3], we test `K-ADWIN` on two well-known learning algorithms where it is easy to observe the effect of distribution drift: the Naïve Bayes classifier and the k-means clusterer. We also perform experiments that directly compare the ability of different estimators to track the average value of a stream of real numbers that varies over time. We use synthetic data in order to control precisely the type and amount of distribution drift. The main conclusions are:

- In all three types of experiments (tracking, Naïve Bayes, and k-means), `K-ADWIN` either gives best results or is very close in performance to the best of the estimators we try. And each of the other estimators is clearly outperformed by `K-ADWIN` in at least some of the experiments. In other words, no estimator ever does much better than `K-ADWIN`, and each of the others is outperformed by `K-ADWIN` in at least one context.
- More precisely, for the tracking problem, `K-ADWIN` and `ADWIN` automatically do about as well as the Kalman filter with the best set of fixed covariance parameters (parameters which, in general, can only be determined after a

good number of experiments). And these three do far better than any fixed-size window.
- In the Naïve Bayes experiments, `K-ADWIN` does somewhat better than `ADWIN` and far better than any memoryless Kalman filter. This is, then, a situation where having a memory clearly helps.
- In the k-means case, again `K-ADWIN` performs about as well as the best (and difficult to find) Kalman filter, and they both do much better than fixed-size windows.

The paper is structured as follows: In Section 2 we describe a general framework for discussing estimator algorithms, as made of three modules: Memory, Estimator, and Change detector. In Section 3 we describe the Kalman filter (an example of Estimator) and the CUSUM test (an example of Change Detector). In Section 4 we review the `ADWIN` algorithm and its theoretical guarantees of performance. In Section 5 we describe how `ADWIN` and the Kalman filter can be combined, and describe the tracking experiments. In Sections 6 and 7 we describe the experiments with Naïve Bayes and k-means, respectively.

2 Time Change Detectors and Predictors: A General Framework

Most approaches for predicting and detecting change in streams of data can be discussed in the general framework: The system consists of three modules: a Memory module, an Estimator Module, and a Change Detector or Alarm Generator module. These three modules interact as shown in Figure 1, which is analogous to Figure 8 in [16].

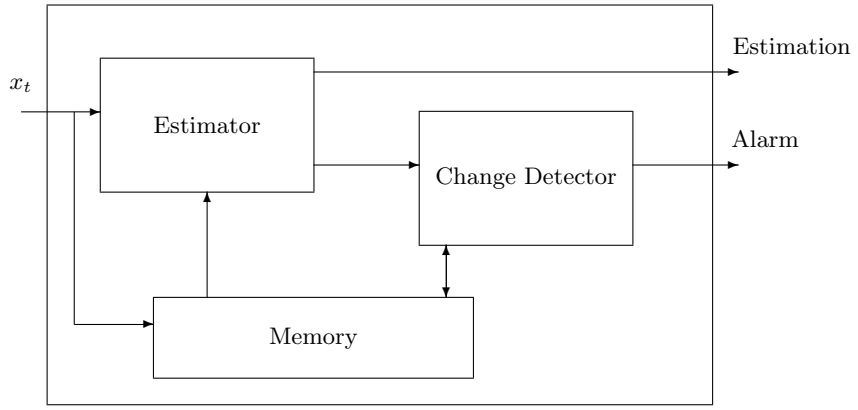

Fig. 1. General Framework

In general, the input to this algorithm is a sequence $x_1, x_2, \ldots, x_t, \ldots$ of data items whose distribution varies over time in an unknown way. The outputs of the algorithm are, at each time step

– an estimation of some important parameters of the input distribution, and
– a signal alarm indicating that distribution change has recently occurred.

In this paper we consider a specific, but very frequent case, of this setting: that in which all the x_t are real values. The desired estimation is usually the expected value of the current x_t, and less often another distribution statistics such as the variance. The only assumption on the distribution is that each x_t is drawn independently from each other.

Memory is the component where the algorithm stores all the sample data or summary that considers relevant at current time, that is, that presumably shows the current data distribution.

The Estimator component is an algorithm that estimates the desired statistics on the input data, which may change over time. The algorithm may or may not use the data contained in the Memory. The simplest Estimator algorithm for the expected is the *linear estimator,* which simply returns the average of the data items contained in the Memory. Other examples of run-time efficient estimators are Auto-Regressive, Auto Regressive Moving Average, and Kalman filters.

The change detector component outputs an alarm signal when it detects change in the input data distribution. It uses the output of the Estimator, and may or may not in addition use the contents of Memory.

In Table 1 we classify these predictors in four classes, depending on whether Change Detector and Memory modules exist:

Table 1. Types of Time Change Predictor and some examples

	No memory	**Memory**
No Change Detector	*Type I* Kalman Filter	*Type III* Adaptive Kalman Filter
Change Detector	*Type II* Kalman Filter + CUSUM	*Type IV* `ADWIN` Kalman Filter + `ADWIN`

– *Type I: Estimator only.* The simplest one is modelled by

$$\hat{x}_k = (1 - \alpha)\hat{x}_{k-1} + \alpha \cdot x_k.$$

The linear estimator corresponds to using $\alpha = 1/N$ where N is the width of a virtual window containing the last N elements we want to consider.

Otherwise, we can give more weight to the last elements with an appropriate constant value of α. The Kalman filter tries to optimize the estimation using a non-constant α (the K value) which varies at each discrete time interval.
- *Type II: Estimator with Change Detector.* An example is the Kalman Filter together with a CUSUM test change detector algorithm, see for example [10].
- *Type III: Estimator with Memory.* We add Memory to improve the results of the Estimator. For example, one can build an Adaptive Kalman Filter that uses the data in Memory to compute adequate values for the process variance Q and the measure variance R. In particular, one can use the sum of the last elements stored into a memory window to model the Q parameter and the difference of the last two elements to estimate parameter R.
- *Type IV: Estimator with Memory and Change Detector.* This is the most complete type. Two examples of this type, from the literature, are:
  - A Kalman filter with a CUSUM test and fixed-length window memory, as proposed in [16]. Only the Kalman filter has access to the memory.
  - A linear Estimator over fixed-length windows that flushes when change is detected [11], and a change detector that compares the running windows with a reference window.

In [3], we proposed another Type IV algorithm called `ADWIN`, which is a new Memory+Change detector module together with a linear estimator. We showed it performs advantageously with respect to strategies that maintain a fixed-length window and linear estimators. The goal of this paper is to propose an even better Type IV algorithm by combining the best of both worlds: a Kalman filter as a "good" Estimation module and `ADWIN` as "good" Memory+Change detector modules.

3 The Kalman Filter and the CUSUM Test

One of the most widely used Estimation algorithms is the Kalman filter. We give here a description of its essentials; see [19] for a complete introduction.

The Kalman filter addresses the general problem of trying to estimate the state $x \in \Re^n$ of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_k = Ax_{k-1} + Bu_k + w_{k-1}$$

with a measurement $z \in \Re^m$ that is

$$Z_k = Hx_k + v_k.$$

The random variables w_k and v_k represent the process and measurement noise (respectively). They are assumed to be independent (of each other), white, and with normal probability distributions

$$p(w) \sim N(0, Q)$$

$$p(v) \sim N(0, R).$$

In essence, the main function of the Kalman filter is to estimate the state vector using system sensors and measurement data corrupted by noise.

The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter fall into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimates for the next time step.

$$x_k^- = Ax_{k-1} + Bu_k$$
$$P_k^- = AP_{k-1}A^T + Q$$

The measurement update equations are responsible for the feedback, i.e. for incorporating a new measurement into the a priori estimate to obtain an improved a posteriori estimate.

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$
$$x_k = x_k^- + K_k(z_k - Hx_k^-)$$
$$P_k = (I - K_k H)P_k^-.$$

There are extensions of the Kalman filter (Extended Kalman Filters, or EKF) for the cases in which the process to be estimated or the measurement-to-process relation is nonlinear. We do not discuss them here.

In our case we consider the input data sequence of real values $z_1, z_2, \ldots, z_t, \ldots$ as the measurement data. The difference equation of our discrete-time controlled process is the simpler one, with $A = 1, H = 1, B = 0$. So the equations are simplified to:

$$K_k = P_{k-1}/(P_{k-1} + R)$$
$$X_k = X_{k-1} + K_k(z_k - X_{k-1})$$
$$P_k = P_k(1 - K_k) + Q.$$

The performance of the Kalman filter depends on the accuracy of the a-priori assumptions:

- linearity of the difference stochastic equation
- estimation of covariances Q and R, assumed to be fixed, known, and follow normal distributions with zero mean.

When applying the Kalman filter to data streams that vary arbitrarily over time, both assumptions are problematic. The linearity assumption for sure, but also the assumption that parameters Q and R are fixed and known – in fact, estimating them from the data is itself a complex estimation problem.

The cumulative sum (CUSUM algorithm), which was first proposed in [15], is a change detection algorithm that gives an alarm when the mean of the input

data is significantly different from zero. The CUSUM input can be any filter residual, for instance the prediction error from a Kalman filter.

The CUSUM test is as follows:

$$g_0 = 0$$

$$g_t = \max\,(0, g_{t-1} + \epsilon_t - \upsilon)$$

$$\text{if } g_t > h \text{ then alarm and } g_t = 0$$

The CUSUM test is memoryless, and its accuracy depends on the choice of parameters υ and h.

4 The ADWIN Algorithm

In this section we review ADWIN, an algorithm for estimating, detecting change, and dynamically adjusting the length of a data window. For details see [3].

4.1 Maintaining Updated Windows of Varying Length

The inputs to the algorithm 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 the expected value 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 $a \leq x_t \leq b$ with probability 1. Nothing else is known about the sequence of distributions D_t; in particular, μ_t is unknown for all t.

Algorithm ADWIN uses a sliding window W with the most recently read x_i. Let n denote the length of W, $\hat{\mu}_W$ the (known) average of the elements in W, and μ_W the (unknown) average of μ_t for $t \in W$. Note that these quantities should be indexed by t, but in general t will be clear from the context.

The idea behind ADWIN 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. Formally, "large enough" and "distinct enough" translate into the computation of a cut value ϵ_c (which depends on δ, the length of the subwindows, and the averages of their contents). 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.

The computation of this ϵ_c (not discussed here) is crucial to the performance of the algorithm. In particular, the main technical result in [3] about the performance of ADWIN is:

Theorem 1. *[3] At every time step we have:*

1. (Few false positives guarantee) *If μ_t remains constant within W, the probability that* ADWIN *shrinks the window at this step is at most δ.*

2. (Few false negatives guarantee) *If for any partition W in two parts W_0W_1 (where W_1 contains the most recent items) we have $|\mu_{W_0} - \mu_{W_1}| > \epsilon$, and if*

$$\epsilon \geq 4 \cdot \sqrt{\frac{3 \max\{\mu_{W_0}, \mu_{W_1}\}}{\min\{n_0, n_1\}} \ln \frac{4n}{\delta}}$$

then with probability $1 - \delta$ `ADWIN` *shrinks W to W_1, or shorter.*

Thus, `ADWIN` is in a sense conservative: it is designed to sustain the null hypothesis "no change" for as long as possible. This leads to a very good false positive rate (essentially 0 in all experimental settings) but a somewhat slow reaction time to slow, gradual changes. Abrupt changes, on the other hand, are detected very quickly.

4.2 Improving Time and Memory Requirements

This 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 presented in [3] a new version, `ADWIN2` that uses ideas developed in data stream algorithmics [1, 13, 2, 5] to find a good cutpoint quickly. We summarize the behavior of this policy in the following theorem.

Theorem 2. *The* `ADWIN2` *algorithm maintains a data structure with the following properties:*

- *It can provide the exact counts of 1's for all the subwindows whose lengths are of the form $\lfloor (1+\epsilon)^i \rfloor$, for some design parameter ϵ, in $O(1)$ time per point.*
- *It uses $O(\frac{1}{\epsilon} \log W)$ memory words (assuming a memory word can contain numbers up to W).*
- *The arrival of a new element can be processed in $O(1)$ amortized time and $O(\log W)$ worst-case time.*

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). The choice of the internal parameter ϵ controls the amount of memory used and the desired density of checkpoints. It 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-invariant.

In the sequel, whenever we say `ADWIN` we really mean its efficient implementation, `ADWIN2`.

5 K-ADWIN = ADWIN + Kalman Filtering

ADWIN is basically a linear Estimator with Change Detector that makes an efficient use of Memory. It seems a natural idea to improve its performance by replacing the linear estimator by an adaptive Kalman filter, where the parameters Q and R of the Kalman filter are computed using the information in ADWIN's memory.

We have set $R = W^2/50$ and $Q = 200/W$, where W is the length of the window maintained by ADWIN. While we cannot rigorously prove that these are the optimal choices, we have informal arguments that these are about the "right" forms for R and Q, on the basis of the theoretical guarantees of ADWIN.

Let us sketch the argument for Q. Theorem 1, part (2) gives a value ϵ for the maximum change that may have occurred within the window maintained by ADWIN. This means that the process variance within that window is at most ϵ^2, so we want to set $Q = \epsilon^2$. In the formula for ϵ, consider the case in which $n_0 = n_1 = W/2$, then we have

$$\epsilon \geq 4 \cdot \sqrt{\frac{3(\mu_{W_0} + \epsilon)}{W/2} \cdot \ln \frac{4W}{\delta}}$$

Isolating from this equation and distinguishing the extreme cases in which $\mu_{W_0} \gg \epsilon$ or $\mu_{W_0} \ll \epsilon$, it can be shown that $Q = \epsilon^2$ has a form that varies between c/W and d/W^2. Here, c and d are constant for constant values of δ, and $c = 200$ is a reasonable estimation. This justifies our choice of $Q = 200/W$. A similar, slightly more involved argument, can be made to justify that reasonable values of R are in the range W^2/c to W^3/d, for somewhat large constants c and d.

When there is no change, ADWIN window's length increases, so R increases too and K decreases, reducing the significance of the most recent data arrived. Otherwise, if there is change, ADWIN window's length reduces, so does R, and K increases, which means giving more importance to the last data arrived.

5.1 Experimental Validation of K-ADWIN

We compare the behaviours of the following types of estimators:

- *Type I*: Kalman filter with different but fixed values of Q and R. The values $Q = 1$, $R = 1000$ seemed to obtain the best results with fixed parameters.
- *Type I*: Exponential filters with $\alpha = 0.1, 0.25, 0.5$. This filter is similar to Kalman's with $K = \alpha, R = (1 - \alpha)P/\alpha$.
- *Type II*: Kalman filter with a CUSUM test Change Detector algorithm. We tried initially the parameters $\upsilon = 0.005$ and $h = 0.5$ as in [10], but we changed to $h = 5$ which systematically gave better results.
- *Type III*: Adaptive Kalman filter with R as the difference of $x_t - x_{t-1}$ and Q as the sum of the last 100 values obtained in the Kalman filter. We use a fixed window of 100 elements.

- *Types III and IV*: Linear Estimators over fixed-length windows, without and with flushing when changing w.r.t. a reference window is detected. Details are as in [3].
- *Type IV*: `ADWIN` and `K-ADWIN`. `K-ADWIN` uses a Kalman filter with $R = W^2/50$ and $Q = 200/W$, where W is the length of the `ADWIN` window.

We build a framework with a stream of synthetic data consisting of some triangular wavelets, of different periods, some square wavelets, also of different periods, and a staircase wavelet of different values. We generate 10^6 points and feed all them to all of the estimators tested. We calculate the mean L_1 distances from the prediction of each estimator to the original distribution that generates the data stream. Finally, we compare these measures for the different estimators.

Table 2 shows the results for $\delta = 0.3$ and L_1. In each column (a test), we show in boldface the result for `K-ADWIN` and for the best result.

A summary of the results is as follows: The results for `K-ADWIN`, `ADWIN`, the Adaptive Kalman filter, and the best fixed-parameter Kalman filter are the best ones in most cases. They are all very close to each other and they outwin each other in various ways, always by a small margin. They all do about as well as the best fixed-size window, and in most cases they win by a large amount. The exception are wavelets of very long periods, in which a very large fixed-size window wins. This is to be expected: when change is extremely rare, it is best to use a large window. Adaptivity necessarily introduces a small penalty, which is a waste in this particular case.

6 Example 1: 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.

It is then extremely easy to incorporate our time-change management: simply add an instance of an estimator for each count $N_{i,j,c}$, and one for each value c

Table 2. Comparative of different estimators using L_1 and $\delta = 0.3$.

	Stair	Triangular					Square					
	5000	128	512	2048	8192	32768	128	512	2048	8192	32768	131072
ADWIN	0.04	0.16	0.09	**0.05**	**0.03**	0.03	0.16	**0.07**	**0.03**	**0.02**	0.02	0.02
Kalman $Q = 1, R = 1000$	0.05	0.14	**0.08**	0.06	0.05	0.05	0.22	0.10	0.05	0.04	0.04	0.04
Kalman $Q = 1, R = 100$	0.08	**0.11**	0.09	0.08	0.08	0.08	**0.13**	0.09	0.08	0.07	0.07	0.07
Kalman $Q = .25, R = .25$	0.28	0.27	0.27	0.27	0.27	0.27	0.22	0.22	0.22	0.22	0.22	0.22
Exp. Estim. $\alpha = .1$	0.09	**0.11**	0.09	0.09	0.09	0.09	**0.13**	0.09	0.08	0.07	0.07	0.07
Exp. Estim.$\alpha = .5$	0.23	0.23	0.23	0.23	0.23	0.23	0.19	0.19	0.19	0.19	0.19	0.19
Exp. Estim. $\alpha = .25$	0.15	0.15	0.14	0.14	0.14	0.14	0.14	0.13	0.12	0.12	0.12	0.12
Adaptive Kalman	**0.03**	0.16	0.11	0.06	0.04	0.03	0.28	0.17	0.06	0.04	0.03	0.03
CUSUM Kalman	0.08	0.15	0.12	0.08	0.06	0.05	0.24	0.18	0.11	0.06	0.04	0.04
K-ADWIN	**0.05**	**0.14**	**0.10**	**0.06**	**0.04**	**0.04**	**0.17**	**0.09**	**0.05**	**0.04**	**0.03**	**0.03**
Fixed-sized $W = 32$	0.07	0.13	**0.08**	0.07	0.07	0.07	0.18	0.09	0.06	0.06	0.06	0.06
Fixed-sized $W = 128$	0.04	0.17	0.12	0.06	0.04	0.03	0.30	0.16	0.06	0.04	0.03	0.03
Fixed-sized $W = 512$	0.06	0.16	0.16	0.11	0.04	**0.02**	0.30	0.30	0.16	0.05	0.02	0.02
Fixed-sized $W = 2048$	0.17	0.16	0.16	0.16	0.11	0.04	0.30	0.30	0.30	0.15	0.04	0.02
Fixed-sized $W = 8192$	0.16	0.16	0.16	0.16	0.16	0.11	0.30	0.30	0.30	0.30	0.15	0.04
Fix. flushing $W = 32$	0.07	0.14	0.09	0.07	0.07	0.07	0.20	0.09	0.07	0.06	0.06	0.06
Fix. flushing $W = 128$	0.04	0.17	0.12	0.06	0.04	0.03	0.30	0.13	0.05	0.03	0.03	0.03
Fix. flushing $W = 512$	0.05	0.16	0.16	0.08	0.04	**0.02**	0.30	0.30	0.07	0.03	0.02	**0.01**
Fix. flushing $W = 2048$	0.10	0.16	0.16	0.16	0.05	**0.02**	0.30	0.30	0.30	0.04	**0.01**	**0.01**
Fix. flushing $W = 8192$	0.15	0.16	0.16	0.16	0.16	0.03	0.30	0.30	0.30	0.30	0.02	**0.01**

of C. When a labelled example is processed, add a 1 to the window for $N_{i,j,c}$ 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, use the output of the estimator associated to $N_{i,j,c}$; this will be simply the average of the corresponding window, if the estimator is linear, or a more elaborate value if it incorporates e.g. a Kalman filter.

The experiments with synthetic data use a changing concept based on a rotating hyperplane explained in [9]. 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 test our algorithms on a classical Naïve Bayes predictor. We use 2 classes, 8 attributes, and 2 values per attribute. The different weights w_i of the hyperplane vary over time, at different moments and different speeds for different

Table 3. Naïve Bayes benchmark

	Width	%Static	%Dynamic	% Dynamic/Static
ADWIN		83,36%	80,30%	96,33%
Kalman $Q = 1, R = 1000$		83,22%	71,13%	85,48%
Kalman $Q = 1, R = 1$		83,21%	56,91%	68,39%
Kalman $Q = .25, R = .25$		83,26%	56,91%	68,35%
Exponential Estimator $\alpha = .1$		83,33%	64,19%	77,03%
Exponential Estimator $\alpha = .5$		83,32%	57,30%	68,77%
Exponential Estimator $\alpha = .25$		83,26%	59,68%	71,68%
Adaptive Kalman		83,24%	76,21%	91,56%
CUSUM Kalman		83,30%	50,65%	60,81%
K-ADWIN		**83,24%**	**81,39%**	**97,77%**
Fixed-sized Window	32	83,28%	67,64%	81,22%
Fixed-sized Window	128	83,30%	75,40%	90,52%
Fixed-sized Window	512	83,28%	80,47%	96,62%
Fixed-sized Window	2048	83,24%	**82,19%**	**98,73%**
Fixed-sized flushing Window	32	83,28%	67,65%	81,23%
Fixed-sized flushing Window	128	83,29%	75,57%	90,73%
Fixed-sized flushing Window	512	83,26%	80,46%	96,64%
Fixed-sized flushing Window	2048	83,25%	82,04%	98,55%

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 Naïve Bayes predictor we do the following: At every time t we build a static Naïve Bayes model M_t using a data set of 1000 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 2000 fresh points, and compute the error rate of both this static model M_t and the different sliding-window models built from the t points seen so far. The ratio of these error rates is averaged over all the run.

Table 3 shows accuracy results. The "%Static" column shows the accuracy of the statically built model – it is the same for all rows, except for small variance. The "%Dynamic" column is the accuracy of the dynamically built model, 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 estimator-based model Naïve Bayes model with respect that of the statically computed one. Again, in each column (a test), we show in boldface the result for K-ADWIN and for the best result.

The results can be summarized as follows: K-ADWIN outperforms plain ADWIN by a small margin, and they both do much better than all the memoryless Kalman filters. Thus, having a memory clearly helps in this case. Strangely enough, the winner is the longest fixed-length window, which achieves 98.73% of the static performance compared to K-ADWIN's 97.77%. We have no clear explanation of this fact, but believe it is an artifact of our benchmark: the way in which we vary the attributes' distributions might imply that simply taking the

average of an attribute's value over a large window has best predictive power. More experiments with other change schedules should confirm or refute this idea.

7 Example 2: *k*-means Clustering

We adapt in essence the incremental version from [14]. 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 [14] to be the square root of the number of points seen so far. In our case, the latter rule is extended to deal with distribution changes, as discussed in [3].

We change this algorithm to deal with time change in the following way. We create an estimator E_{ij} for every attribute centroid i and every attribute j. The algorithm still interleaves phases in which centroids are just incrementally modified with incoming points and phases where global recomputation of centroids takes place. Recomputation phases occur for two reasons. First, when any of the E_{ij} shrinks its window (for `ADWIN`-type estimators), we take this as a signal that the position of centroid i may have changed and recompute. In the case of estimators that use windows of a fixed size s, we recompute whenever this window becomes full. Second, when the average point distance to closest centroid has changed more than an ϵ factor, where ϵ is user-specified. This is taken as an indication that a certain number of points might move to another cluster if recomputation took place now.

The synthetic data used in our experiments consist of a sample of 10^5 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.

On this data stream, we run one instance of the incremental k-means clusterer with each of the estimator types we want to test. Each instance of the clusterer uses itself an estimator for each centroid coordinate. At every time step, we feed the current example to each of the clusterers, we generate a sample of points from the current distribution (which we know) and use a traditional k-means clusterer to cluster this sample. Then, we compute the sum of the distances of each data point to each centroid assigned, for this statically built clustering and for each of the clustering dynamically built using different estimators. The statically built clustering is thus a baseline on how good the clustering could be without distribution drift.

Table 4 shows the results of computing the distance from 100 random points to their centroids. Again, in each column (a test), we show in boldface the result for `K-ADWIN` and for the best result.

The results can be summarized as follows: The winners are the best fixed-parameter Kalman filter and, for small variance, `K-ADWIN`. `ADWIN` follows closely in all cases. These three do much better than any fixed-size window strategy, and somewhat better than Kalman filters with suboptimal fixed-size parameters.

Table 4. k-means sum of distances to centroids, with $k = 5$, 10^5 samples and change's velocity of 10^{-3}.

		$\sigma = 0.15$		$\sigma = 0.3$		$\sigma = 0.6$	
	Width	Static	Dynamic	Static	Dynamic	Static	Dynamic
ADWIN		9,72	21,54	19,41	28,58	38,83	46,48
Kalman $Q = 1, R = 1000$		9,72	19,72	19,41	27,92	38,83	**46,02**
Kalman $Q = 1, R = 100$		9,71	17,60	19,41	**27,18**	38,77	46,16
Kalman $Q = .25, R = .25$		9,71	22,63	19,39	30,21	38,79	49,88
Exponential Estimator $\alpha = .1$		9,71	21,89	19,43	27,28	38,82	46,98
Exponential Estimator $\alpha = .5$		9,72	20,58	19,41	29,32	38,81	46,47
Exponential Estimator $\alpha = .25$		9,72	17,69	19,42	27,66	38,82	46,18
Adaptive Kalman		9,72	18,98	19,41	31,16	38,82	51,96
CUSUM Kalman		9,72	18,29	19,41	33,82	38,85	50,38
K-ADWIN		**9,72**	**17,30**	**19,40**	**28,34**	**38,79**	**47,45**
Fixed-sized Window	32	9,72	25,70	19,40	39,84	38,81	57,58
Fixed-sized Window	128	9,72	36,42	19,40	49,70	38,81	68,59
Fixed-sized Window	512	9,72	38,75	19,40	52,35	38,81	71,32
Fixed-sized Window	2048	9,72	39,64	19,40	53,28	38,81	73,10
Fixed-sized Window	8192	9,72	43,39	19,40	55,66	38,81	76,90
Fixed-sized Window	32768	9,72	53,82	19,40	64,34	38,81	88,17
Fixed-sized flushing Window	32	9,72	35,62	19,40	47,34	38,81	65,37
Fixed-sized flushing Window	128	9,72	40,42	19,40	52,03	38,81	70,47
Fixed-sized flushing Window	512	9,72	39,12	19,40	53,05	38,81	72,81
Fixed-sized flushing Window	2048	9,72	40,99	19,40	56,82	38,81	75,35
Fixed-sized flushing Window	8192	9,72	45,48	19,40	60,23	38,81	91,49
Fixed-sized flushing Window	32768	9,72	73,17	19,40	84,55	38,81	110,77

8 Conclusions and Future Work

The experiments on synthetic data give strong indications that the combination of Kalman filtering and a system that dynamically manages a window of examples has good potential for learning from data streams. More precisely, it seems to give better results than either memoryless Kalman Filtering or sliding windows with linear estimators. Furthermore, it tunes itself to the data stream at hand, with no need for the user to hardwire or precompute parameters that describe how the data stream changes over time.

Because of lack of space we have not discussed the time and memory used by K-ADWIN. These are comparable to those of ADWIN2, reported in [3], and quite reasonable. For example, the experiments using 10^6 sample points and about a dozen estimators take the order of 20 seconds to execute on a standard PC.

There are two features of the Kalman filter that we have not discussed here and can be important in the future. One is that it filters noise in the signal very well – that is in fact one of the main motivations for its use. This could help in dealing with attribute or class noise, in the learning context. Another feature is that it can continue to give reasonable predictions even if the input signal (e.g.,

from sensors) is temporarily not available. This could help in various learning situations: for example if there may be delays in the arrival of examples or, on the other extreme, if examples are arriving too fast and some of them have to be skipped.

Other future work goes in two directions: on the one hand, these ideas should be tested on real-world, not only synthetic data. This is a notoriously difficult evaluation problem, since it is generally difficult to assess the real drift present in a real-world data set, hence compare meaningfully the performance of different strategies. On the other hand, other learning algorithms should be tried; clear candidates are algorithms for induction of decision trees.

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. A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. Technical report, Universitat Politècnica de Catalunya, 2006. Available from `www.lsi.upc.edu/~abifet`.
4. E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *Proc. 22nd ACM Symposium on Principles of Database Systems*, 2003.
5. M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 14(1):27–45, 2002.
6. G. Dong, J.H., Laks V.S. Lakshmanan, J.P., H. Wang, and P.S. Yu. Online mining of changes from data streams: Research problems and preliminary results. In *ACM SIGMOD Workshop on Management and Processing of Data Streams*, 2003.
7. Wei Fan. Streamminer: A classifier ensemble-based engine to mine concept-drifting data streams. In *Proc. 30th VLDB Conf., Toronto, Canada*, 2004.
8. J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *SBIA Brazilian Symposium on Artificial Intelligence*, pages 286–295, 2004.
9. 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.
10. K. Jacobsson, N. Möller, K.-H. Johansson, and H. Hjalmarsson. Some modeling and estimation issues in control of heterogeneous networks. In *16th Intl. Symposium on Mathematical Theory of Networks and Systems (MTNS2004)*, 2004.
11. D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proc. 30th VLDB Conf., Toronto, Canada*, 2004.
12. R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *Proc. 17th Intl. Conf. on Machine Learning*, pages 487 – 494, 2000.
13. S. Muthukrishnan. Data streams: Algorithms and applications. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.
14. C. Ordonez. Clustering binary data streams with k-means. In *ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, 2003.
15. E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

16. T. Schön, A. Eidehall, and F. Gustafsson. Lane departure detection for improved road geometry estimation. Technical Report LiTH-ISY-R-2714, Dept. of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, Dec 2005.
17. M. Severo and J. Gama. Change detection with Kalman Filter applied to apnoeas disorder. In *2nd. Intl. Workshop on Knowledge Discovery from Data Streams*, Porto (Portugal), 2005.
18. H. Wang, W. Fan, P. Yun, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *ACM SIGKDD*, 2003.
19. G. Welch and G. Bishop. An introduction to the Kalman Filter. Technical report, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995.