

# Kalman Filters and Adaptive Windows for Learning in Data Streams

Albert Bifet   Ricard Gavaldà

Universitat Politècnica de Catalunya

DS'06 Barcelona

# Outline

- 1 Introduction
- 2 The Kalman Filter and the CUSUM Test
- 3 The ADWIN Algorithm
- 4 General Framework
- 5 K-ADWIN
- 6 Experimental Validation of K-ADWIN
- 7 Conclusions

# Introduction

- Data Streams
  - Sequence potentially infinite
  - High amount of data:
    - sublinear space
  - High Speed of arrival:
    - small constant time per example
- Estimation and prediction
- Distribution and concept drift
- K-ADWIN : Combination
  - Kalman filter
  - ADWIN : Adaptive window of recently seen data items.

# Introduction

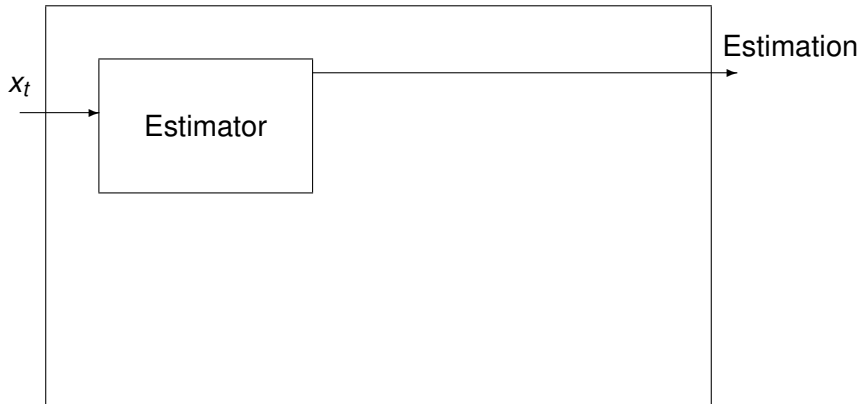
## Problem

Given an input sequence  $x_1, x_2, \dots, x_t, \dots$  we want to output at instant  $t$  a prediction  $\hat{x}_{t+1}$  minimizing prediction error:

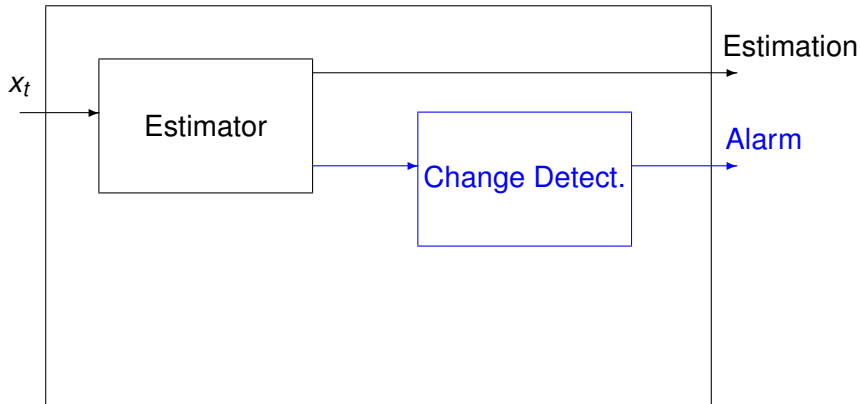
$$|\hat{x}_{t+1} - x_{t+1}|$$

considering distribution changes overtime.

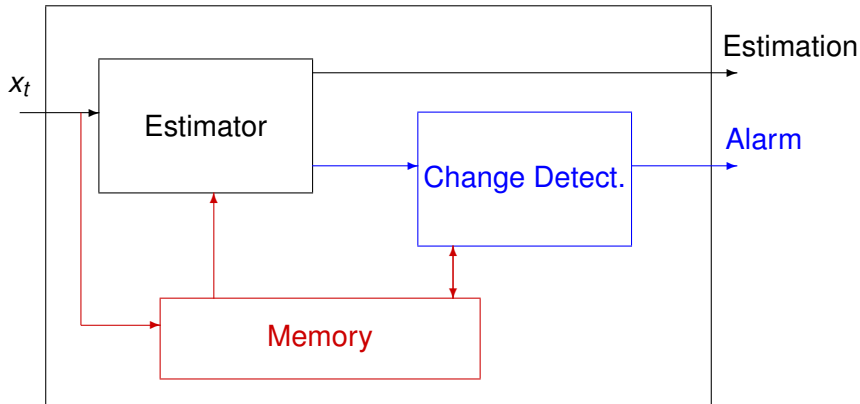
# Time Change Detectors and Predictors: A General Framework



# Time Change Detectors and Predictors: A General Framework



# Time Change Detectors and Predictors: A General Framework



# Introduction

## Our generic proposal:

- Use change detector
- Use memory

## Our particular proposal: $K$ -ADWIN

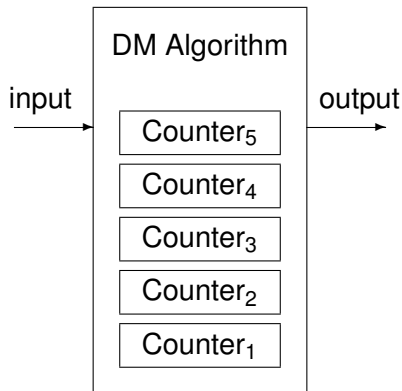
- Kalman filter as estimator
- Use ADWIN as change detector with memory [BG06]

## Application

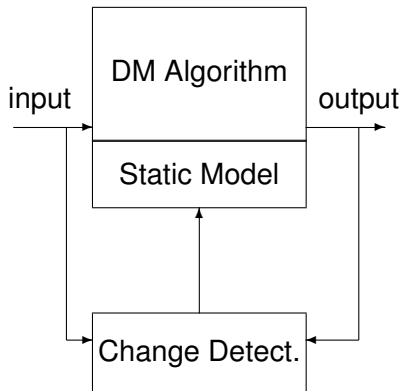
- Estimate statistics from data streams
- In Data Mining Algorithms based on counters, replace them for estimators.

# Data Mining Algorithms with Concept Drift

## No Concept Drift

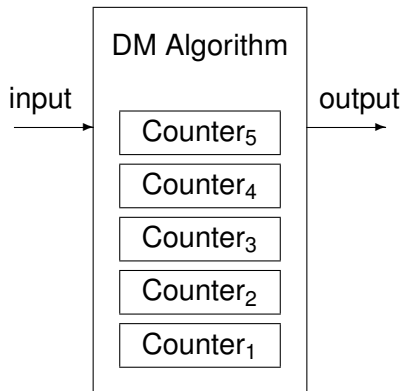


## Concept drift

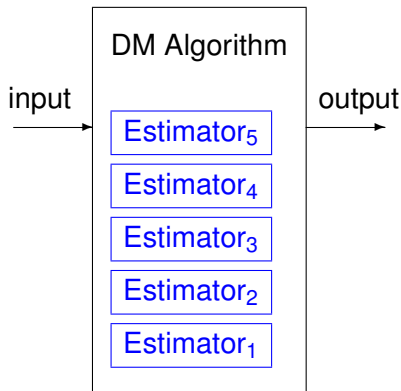


# Data Mining Algorithms with Concept Drift

## No Concept Drift



## Concept Drift



# The Kalman Filter

- Optimal recursive algorithm
- Minimum mean-square error estimator
- Estimate the state  $x \in \mathfrak{R}^n$  of a discrete-time controlled process

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

with a measurement  $z \in \mathfrak{R}^m$  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

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

# The Kalman Filter

The difference equation of our discrete-time controlled process is

$$K_k = P_{k-1} / (P_{k-1} + R)$$

$$X_k = X_{k-1} + K_k(z_k - X_{k-1})$$

$$P_k = P_k(1 - K_k) + Q$$

# The Kalman Filter

The difference equation of our discrete-time controlled process is

$$K_k = P_{k-1} / (P_{k-1} + R)$$

$$X_k = X_{k-1} + K_k(z_k - X_{k-1})$$

$$P_k = P_k(1 - K_k) + Q$$

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.

# The CUSUM Test

- The cumulative sum (CUSUM algorithm), is a change detection algorithm that gives an alarm when the mean of the input data is significantly different from zero.
- The CUSUM test is memoryless, and its accuracy depends on the choice of parameters  $v$  and  $h$ . It is as follows:

$$g_0 = 0, \quad g_t = \max(0, g_{t-1} + \epsilon_t - v)$$

if  $g_t > h$  then alarm and  $g_t = 0$

## Algorithm ADWIN [BG06]

## Example

 $W =$  101010110111111

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6             for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$$W = 101010110111111$$

$$W_0 = 1 \quad W_1 = 01010110111111$$

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6     for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$W =$  101010110111111  
 $W_0 =$  10  $W_1 =$  1010110111111

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6     for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$$W = 101010110111111$$

$$W_0 = 101 \quad W_1 = 010110111111$$

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6         for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$$W = 101010110111111$$

$$W_0 = 1010 \quad W_1 = 10110111111$$

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6     for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$$W = 101010110111111$$

$$W_0 = 10101 \quad W_1 = 0110111111$$

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6     for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$$W = 101010110111111$$

$$W_0 = 101010 \quad W_1 = 110111111$$

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6         for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$$W = 101010110111111$$

$$W_0 = 1010101 \quad W_1 = 10111111$$

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6     for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$W =$  101010110111111  
 $W_0 =$  10101011  $W_1 =$  0111111

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6     for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$W =$  101010110111111  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  : CHANGE DETECTED!  
 $W_0 =$  101010110  $W_1 =$  111111

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6     for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$W =$  101010110111111 Drop elements from the tail of  $W$

$W_0 =$  101010110  $W_1 =$  111111

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6         for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

## Algorithm ADWIN [BG06]

## Example

$W =$  01010110111111 Drop elements from the tail of  $W$

$W_0 =$  101010110  $W_1 =$  111111

## 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**  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$  holds
- 6         for every split of  $W$  into  $W = W_0 \cdot W_1$
- 7     Output  $\hat{\mu}_W$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$   $1011011$   $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$   $1111$

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{1}$   $\boxed{01010110111111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{10}$   $\boxed{1010110111111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{101}$   $\boxed{010110111111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{1010}$   $\boxed{10110111111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{10101}$  |  $\boxed{0110111111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{101010}$   $\boxed{110111111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$   $1011011$   $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$   $1111$

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{1010101}$   $\boxed{10111111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{10101011}$   $\boxed{0111111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{101010110}$   $\boxed{111111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{1010101101}$   $\boxed{11111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{10101011011}$   $\boxed{1111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{101010110111}$   $\boxed{111}$

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$\boxed{1010}$  1011011  $\boxed{1111}$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$\boxed{10101011011}$  1111

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$\boxed{1010101101111}$  | 11

# Window Management Models

$$W = \boxed{101010110111111}$$

Equal & fixed size subwindows

$$\boxed{1010} \boxed{1011011} \boxed{1111}$$

*D. Kifer, S. Ben-David, and J. Gehrke.* Detecting change in data streams. 2004

Total window against subwindow

$$\boxed{\boxed{10101011011} 1111}$$

*J. Gama, P. Medas, G. Castillo, and P. Rodrigues.* Learning with drift detection. 2004

ADWIN: All Adjacent subwindows

$$\boxed{10101011011111} \boxed{1}$$

# Algorithm ADWIN [BG06]

ADWIN has rigorous guarantees

- On ratio of false positives
- On ratio of false negatives
- On the relation of the size of the current window and change rates

# Algorithm ADWIN [BG06]

## Theorem

At every time step we have:

- 1 (Few false positives guarantee) *If  $\mu_t$  remains constant within  $W$ , the probability that ADWIN shrinks the window at this step is at most  $\delta$ .*
- 2 (Few false negatives guarantee) *If for any partition  $W$  in two parts  $W_0 W_1$  (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.*

# Data Streams Algorithm $ADWIN_2$ [BG06]

$ADWIN_2$  using a Data Stream Sliding Window Model,

- can provide the exact counts of 1's in  $O(1)$  time per point.
- tries  $O(\log W)$  cutpoints
- uses  $O(\frac{1}{\epsilon} \log W)$  memory words
- the processing time per example is  $O(\log W)$  (amortized) and  $O(\log^2 W)$  (worst-case).

## Sliding Window Model

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

# Algorithm ADWIN2

ADWIN2 using a Data Stream Sliding Window Model,

- can provide the exact counts of 1's in  $O(1)$  time per point.
- tries  $O(\log W)$  cutpoints
- uses  $O(\frac{1}{\epsilon} \log W)$  memory words
- the processing time per example is  $O(\log W)$  (amortized) and  $O(\log^2 W)$  (worst-case).

Insert new Item

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

# Algorithm ADWIN2

ADWIN2 using a Data Stream Sliding Window Model,

- can provide the exact counts of 1's in  $O(1)$  time per point.
- tries  $O(\log W)$  cutpoints
- uses  $O(\frac{1}{\epsilon} \log W)$  memory words
- the processing time per example is  $O(\log W)$  (amortized) and  $O(\log^2 W)$  (worst-case).

Insert new Item

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

# Algorithm ADWIN2

ADWIN2 using a Data Stream Sliding Window Model,

- can provide the exact counts of 1's in  $O(1)$  time per point.
- tries  $O(\log W)$  cutpoints
- uses  $O(\frac{1}{\epsilon} \log W)$  memory words
- the processing time per example is  $O(\log W)$  (amortized) and  $O(\log^2 W)$  (worst-case).

## Compressing Buckets

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

# Algorithm ADWIN2

ADWIN2 using a Data Stream Sliding Window Model,

- can provide the exact counts of 1's in  $O(1)$  time per point.
- tries  $O(\log W)$  cutpoints
- uses  $O(\frac{1}{\epsilon} \log W)$  memory words
- the processing time per example is  $O(\log W)$  (amortized) and  $O(\log^2 W)$  (worst-case).

## Compressing Buckets

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

# Algorithm ADWIN2

ADWIN2 using a Data Stream Sliding Window Model,

- can provide the exact counts of 1's in  $O(1)$  time per point.
- tries  $O(\log W)$  cutpoints
- uses  $O(\frac{1}{\epsilon} \log W)$  memory words
- the processing time per example is  $O(\log W)$  (amortized) and  $O(\log^2 W)$  (worst-case).

## Compressing Buckets

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

# Algorithm ADWIN2

ADWIN2 using a Data Stream Sliding Window Model,

- can provide the exact counts of 1's in  $O(1)$  time per point.
- tries  $O(\log W)$  cutpoints
- uses  $O(\frac{1}{\epsilon} \log W)$  memory words
- the processing time per example is  $O(\log W)$  (amortized) and  $O(\log^2 W)$  (worst-case).

## Compressing Buckets

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

# Algorithm ADWIN2

ADWIN2 using a Data Stream Sliding Window Model,

- can provide the exact counts of 1's in  $O(1)$  time per point.
- tries  $O(\log W)$  cutpoints
- uses  $O(\frac{1}{\epsilon} \log W)$  memory words
- the processing time per example is  $O(\log W)$  (amortized) and  $O(\log^2 W)$  (worst-case).

Detecting Change: Delete last Bucket

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

# Algorithm ADWIN2

ADWIN2 using a Data Stream Sliding Window Model,

- can provide the exact counts of 1's in  $O(1)$  time per point.
- tries  $O(\log W)$  cutpoints
- uses  $O(\frac{1}{\epsilon} \log W)$  memory words
- the processing time per example is  $O(\log W)$  (amortized) and  $O(\log^2 W)$  (worst-case).

Detecting Change: Delete last Bucket

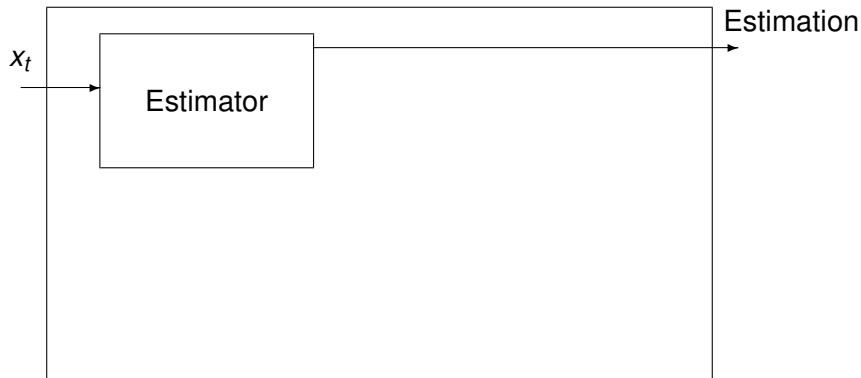
10111	11	1
-------	----	---

Content:            4      2    1

Capacity:          5      2    1

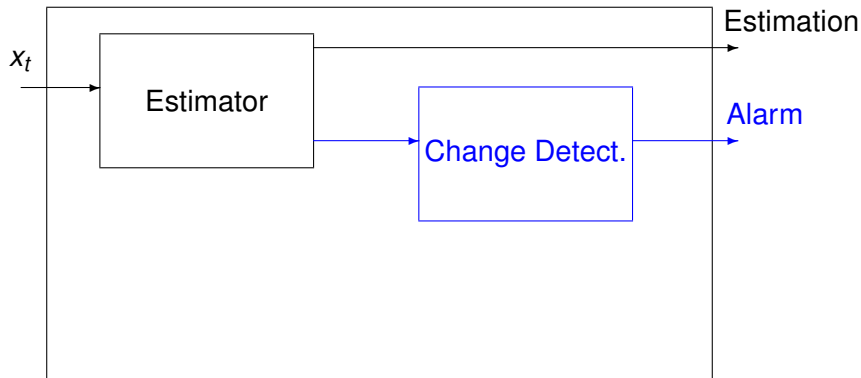
# General Framework Time Change Detectors and Predictors : Type I

## Example (Kalman Filter)



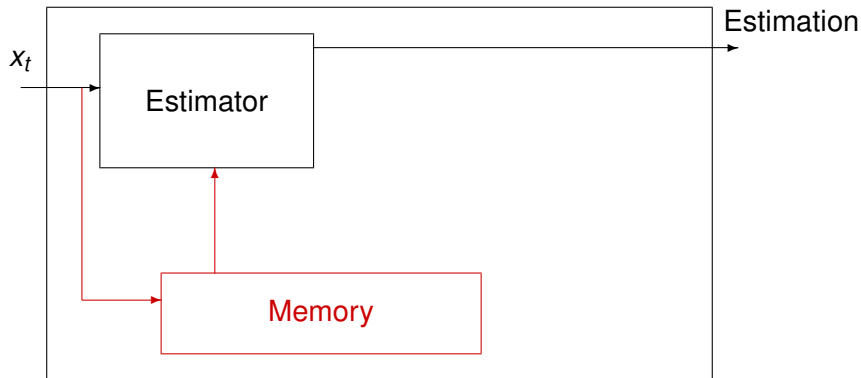
# General Framework Time Change Detectors and Predictors : Type II

## Example (Kalman Filter + CUSUM)



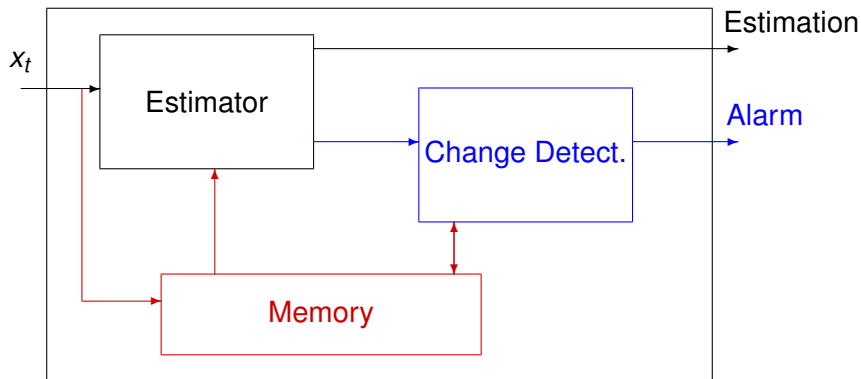
# General Framework Time Change Detectors and Predictors : Type III

## Example (Adaptive Kalman Filter)



# General Framework Time Change Detectors and Predictors : Type IV

Example (ADWIN, Kalman Filter+ADWIN)



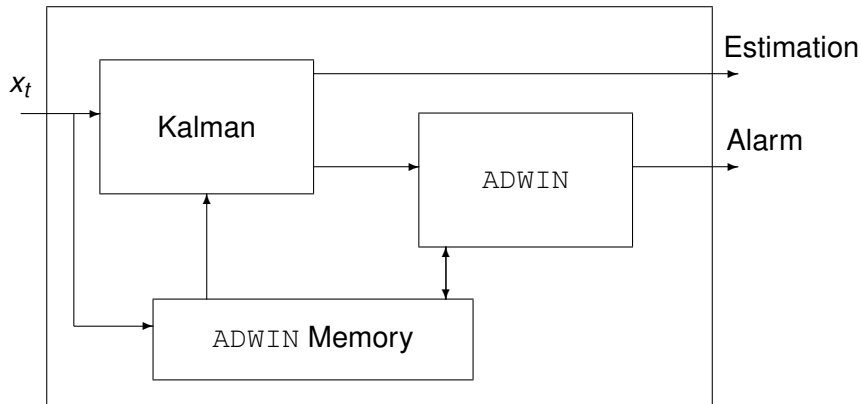
# Time Change Detectors and Predictors: A General Framework

	<b>No memory</b>	<b>Memory</b>
<b>No Change Detector</b>	<i>Type I</i> Kalman Filter	<i>Type III</i> Adaptive Kalman Filter
<b>Change Detector</b>	<i>Type II</i> Kalman Filter + CUSUM	<i>Type IV</i> ADWIN Kalman Filter + ADWIN

# Time Change Detectors and Predictors: A General Framework

	No memory	Memory
No Change Detector	<i>Type I</i> Kalman Filter	<i>Type III</i> Adaptive Kalman Filter Q,R estimated from window
Change Detector	<i>Type II</i> Kalman Filter + CUSUM	<i>Type IV</i> ADWIN Kalman Filter + ADWIN Q,R estimated from window

# K-ADWIN = ADWIN + Kalman Filtering



$R = W^2/50$  and  $Q = 200/W$ , where  $W$  is the length of the window maintained by ADWIN.

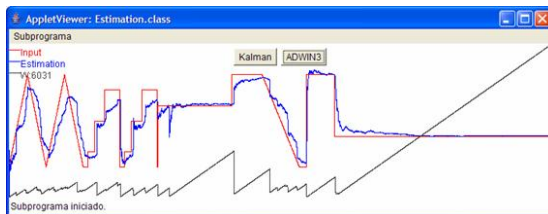
# Tracking Experiments

KALMAN:  $R=1000; Q=1$  Error= 854.97

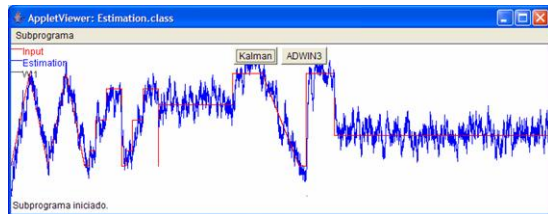


# Tracking Experiments

ADWIN : Error= 674.66

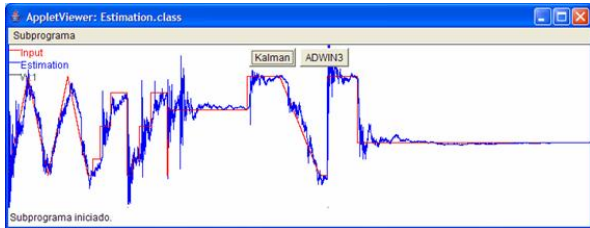


KALMAN: R=1000;Q=1 Error= 854.97

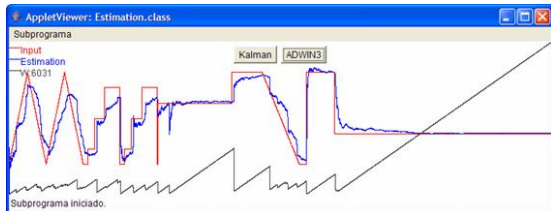


# Tracking Experiments

$K$ -ADWIN Error= 530.13



ADWIN : Error= 674.66



# Naïve Bayes

## Example

Data set that describes the weather conditions for playing some game.

<b>outlook</b>	<b>temp.</b>	<b>humidity</b>	<b>windy</b>	<b>play</b>
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes

Assume we have to classify the following new instance:

<b>outlook</b>	<b>temp.</b>	<b>humidity</b>	<b>windy</b>	<b>play</b>
sunny	cool	high	true	?

# Naïve Bayes

- Assume we have to classify the following new instance:

<b>outlook</b>	<b>temp.</b>	<b>humidity</b>	<b>windy</b>	<b>play</b>
sunny	cool	high	true	?

- We classify the new instance:

$$\nu_{NB} = \arg \max_{\nu \in \{yes, no\}} P(\nu_j) P(sunny|\nu_j) P(cool|\nu_j) P(high|\nu_j) P(true|\nu_j)$$

- Conditional probabilities can be **estimated** directly as frequencies:

$$P(a_i|\nu_j) = \frac{\text{number of instances with attribute } a_i \text{ and class } \nu_j}{\text{total number of training instances with class } \nu_j}$$

- Create one **estimator** for each frequency that needs estimation

# Experimental Validation of $\kappa$ -ADWIN

- We test Naïve Bayes Predictor and k-means clustering
- Method: replace counters by estimators
- Synthetic data where change is controllable
- **Naïve Bayes**: We compare accuracy of
  - Static model: Training of 1000 samples every instant
  - Dynamic model: replace probabilities counters by estimatorscomputing the ratio  $\frac{\%Dynamic}{Static}$  with tests using 2000 samples.

# Naïve Bayes Predictor

	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%
Adaptive Kalman		83,24%	76,21%	91,56%
CUSUM Kalman		83,30%	50,65%	60,81%
$K$ -ADWIN		<b>83,24%</b>	<b>81,39%</b>	<b>97,77%</b>
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%	<b>82,19%</b>	<b>98,73%</b>

# k-means Clustering

	Width	$\sigma = 0.15$	
		Static	Dynamic
ADWIN		9,72	21,54
Kalman $Q = 1, R = 1000$		9,72	19,72
Kalman $Q = 1, R = 100$		9,71	17,60
Kalman $Q = .25, R = .25$		9,71	22,63
Adaptive Kalman		9,72	18,98
CUSUM Kalman		9,72	18,29
$\kappa$ -ADWIN		<b>9,72</b>	<b>17,30</b>
Fixed-sized Window	32	9,72	25,70
Fixed-sized Window	128	9,72	36,42
Fixed-sized Window	512	9,72	38,75
Fixed-sized Window	2048	9,72	39,64
Fixed-sized Window	8192	9,72	43,39
Fixed-sized Window	32768	9,72	53,82

# Results

- No estimator ever does much better than  $K$ -ADWIN
- $K$ -ADWIN does much better than every other estimators in at least one context.
- Tracking problem  $K$ -ADWIN and ADWIN automatically do about as well as the Kalman filter with the best set of fixed covariance parameters.
- Naïve Bayes and  $k$ -means:  $K$ -ADWIN does somewhat better than ADWIN and far better than any memoryless Kalman filter.

# Conclusions and Future Work

- $K$ -ADWIN tunes itself to the data stream at hand, with no need for the user to hardwire or precompute parameters.
- Better results than either memoryless Kalman Filtering or sliding windows with linear estimators.
- Future work :
  - Tests on real-world, not only synthetic data.
  - Other learning algorithms: algorithms for induction of decision trees.