Data Stream Mining- Lecture 3

Statistics on recent data

Chandresh Kumar Maurya, Assistant Professor, Péter Kiss, Teaching Assistant

Eötvös Loránd University, Budapest, Hungary

October 22, 2020

Data Synopsis

Machine Learning / Data Mining - give a model for the data - compress knowledge, statistics.
For data streams

- unbound nature
- concept drift

Need to compute an estimate of the stream due to 1) low memory, 2) fast computation. Data synopsis can be done in two ways:

- Sliding Window
- Data Reduction

Sliding window

Up to now we tried to approximate statistics for the whole stream.
Why we need sliding window?

Sliding window

Up to now we tried to approximate statistics for the whole stream.
Why we need sliding window?
For capturing recent data

Sliding window

Up to now we tried to approximate statistics for the whole stream.
Why we need sliding window?
For capturing recent data
Types:

- **Sequence based**: they contain sequences of data and size of the window is decided based on the number of data sequences they contain.
- **Timestamp based**: The size of the window is decided based on the time interval considered.

Example for query recent statistics

What is the standard deviation of the last 100 element?

$$\text{First moment:} 100 \cdot \mathbb{E}[\overline{X}] (= 100 \cdot \mu) = \sum_{i=n-99}^{n} x_i \quad (1)$$

$$\text{Second moment:} 100 \cdot \mathbb{E}[\overline{X}^2] = \sum_{i=n-99}^{n} x_i^2 \quad (2)$$

$$\text{Standard deviation:} \sigma = \sqrt{\mathbb{E}[\overline{X}^2] - \mathbb{E}[\overline{X}]^2} \quad (3)$$

How to maintain the value?

Example for query recent statistics

What is the standard deviation of the last 100 element?

$$\text{First moment:} 100 \cdot \mathbb{E}[\overline{X}] (= 100 \cdot \mu) = \sum_{i=n-99}^{n} x_i \quad (1)$$

$$\text{Second moment:} 100 \cdot \mathbb{E}[\overline{X}^2] = \sum_{i=n-99}^{n} x_i^2 \quad (2)$$

$$\text{Standard deviation:} \sigma = \sqrt{\mathbb{E}[\overline{X}^2] - \mathbb{E}[\overline{X}]^2} \quad (3)$$

How to maintain the value? any time t a new element appears :

$$\mathbb{E}[\overline{X}]_t = \mathbb{E}[\overline{X}]_{t-1} - \frac{x_{t-100}}{100} + \frac{x_t}{100} \quad (4)$$

$$\mathbb{E}[\overline{X^2}]_{t-1} = \mathbb{E}[\overline{X}^2]_{t-1} - \frac{x_{t-100}^2}{100} + \frac{x_t^2}{100} \quad (5)$$

Thus for this we need to keep 200 elements in memory. Similar problem for timestamp based windowing, based on granularity continuous updates, plus also the size of window changes by time..

Sequence based window: Examples

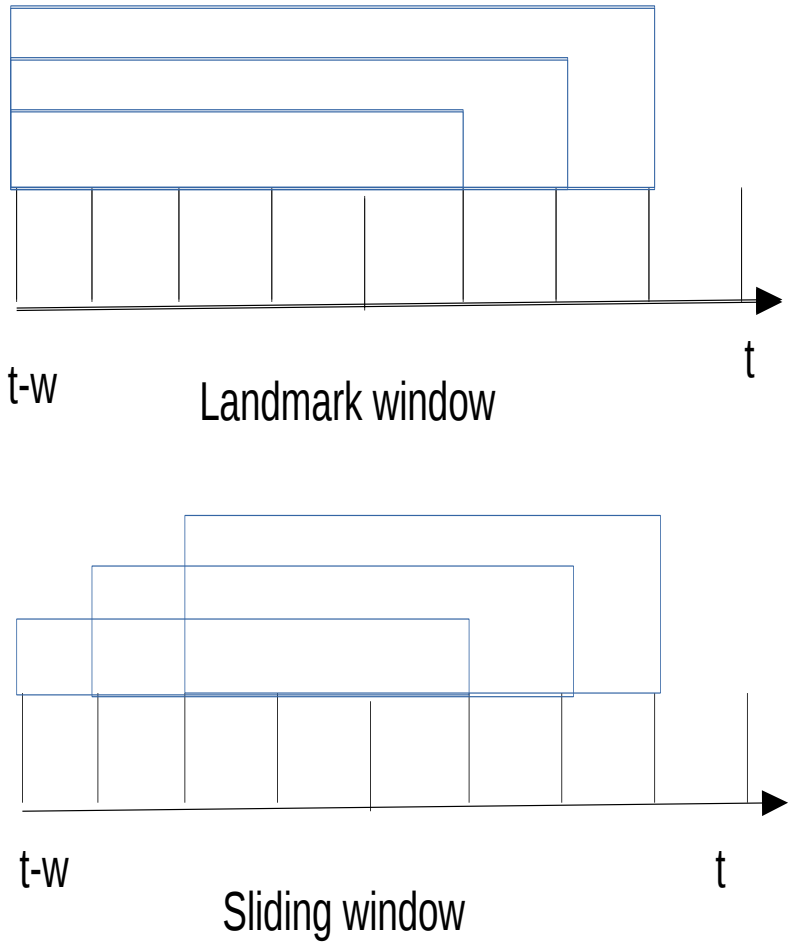


Figure: Sequence based windows. Top figure: Landmark window and bottom figure is sliding window (used in packet transmission)

Timestamp based window: Examples

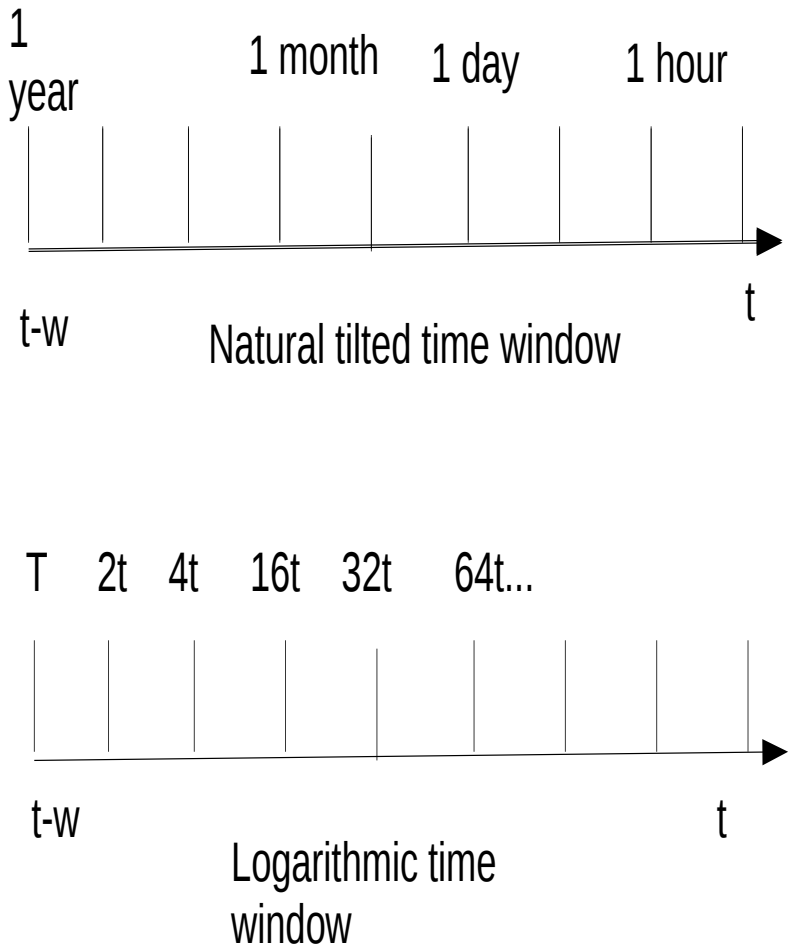


Figure: Timestamp based windows. Top figure: natural tilted window and bottom figure is logarithmic time window

Exponential histograms for sliding windows - DGIM

The Goal: can we reduce space necessary for maintain statistics of the window?

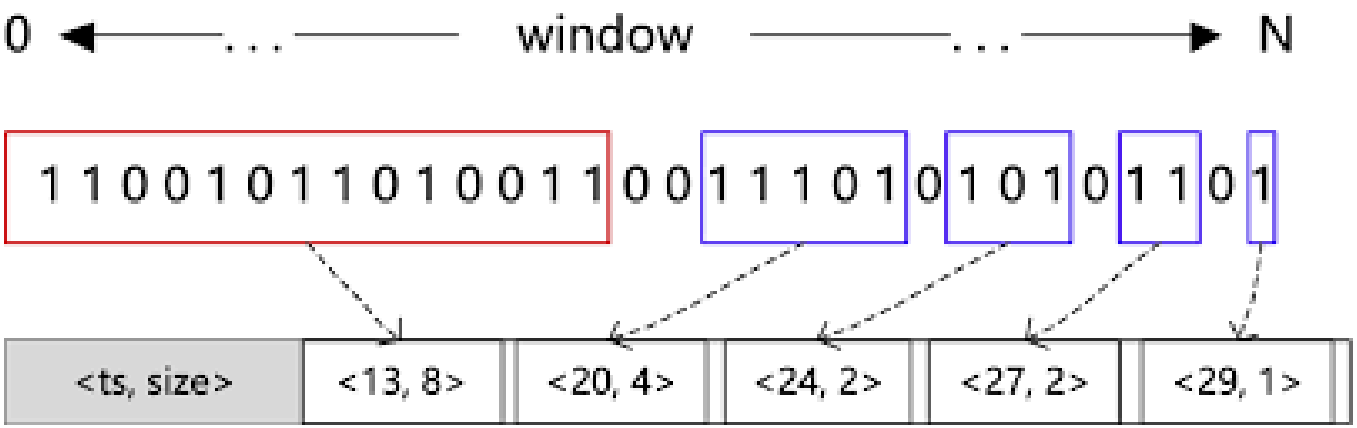
Exponential histograms for sliding windows - DGIM

The Goal: can we reduce space necessary for maintain statistics of the window?
DGIM : maintains approximations of aggregate functions on sliding windows of real numbers on $\mathcal{O}(\log |W|)$ memory vs $\mathcal{O}(|W|)$

Exponential histograms for sliding windows - DGIM

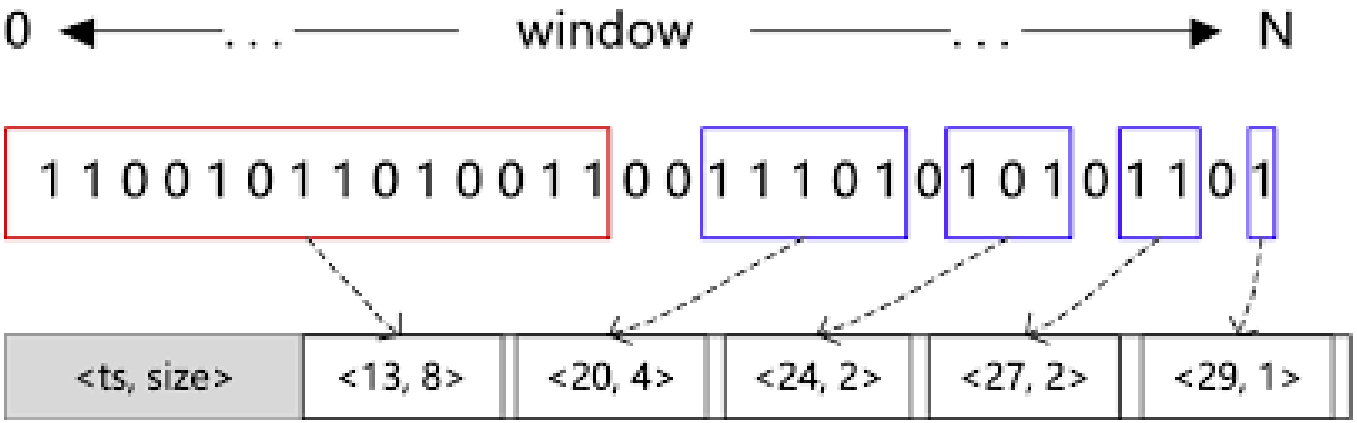
The Goal: can we reduce space necessary for maintain statistics of the window?
DGIM : maintains approximations of aggregate functions on sliding windows of real numbers on $\mathcal{O}(\log |W|)$ memory vs $\mathcal{O}(|W|)$

- each element has a timestamp : the position in which it arrives
- partition W int sequence of subwindows - each is represented by a bucket with a capacity of a power of 2
- timestamp an integer that is the timestamp of the most recent 1 in the bucket



Source: https://link.springer.com/chapter/10.1007/978-981-15-3281-8_12

Frame Title



- parameter r no. of buckets with a given capacity: $1,2,4,...2^m$, with an m for

$$r \sum_{i=0}^{m-1} 2^i < |W| \leq r \sum_{i=0}^{m} 2^i. \quad (6)$$

Exponential histograms for sliding windows - DGIM

DGIM :maintains approximations of aggregate functions on sliding windows of real numbers

- Example for DGIM Query: How many ones in the lask k items?
- aggregate function : sum as and elements are bits
- Answer :
  1. find the bucket b with the earliest timestamp t that includes some of the most recent bits
  2. sum all the capacities of the buckets completely within k plus half of the size of b

Exponential histograms for sliding windows - DGIM

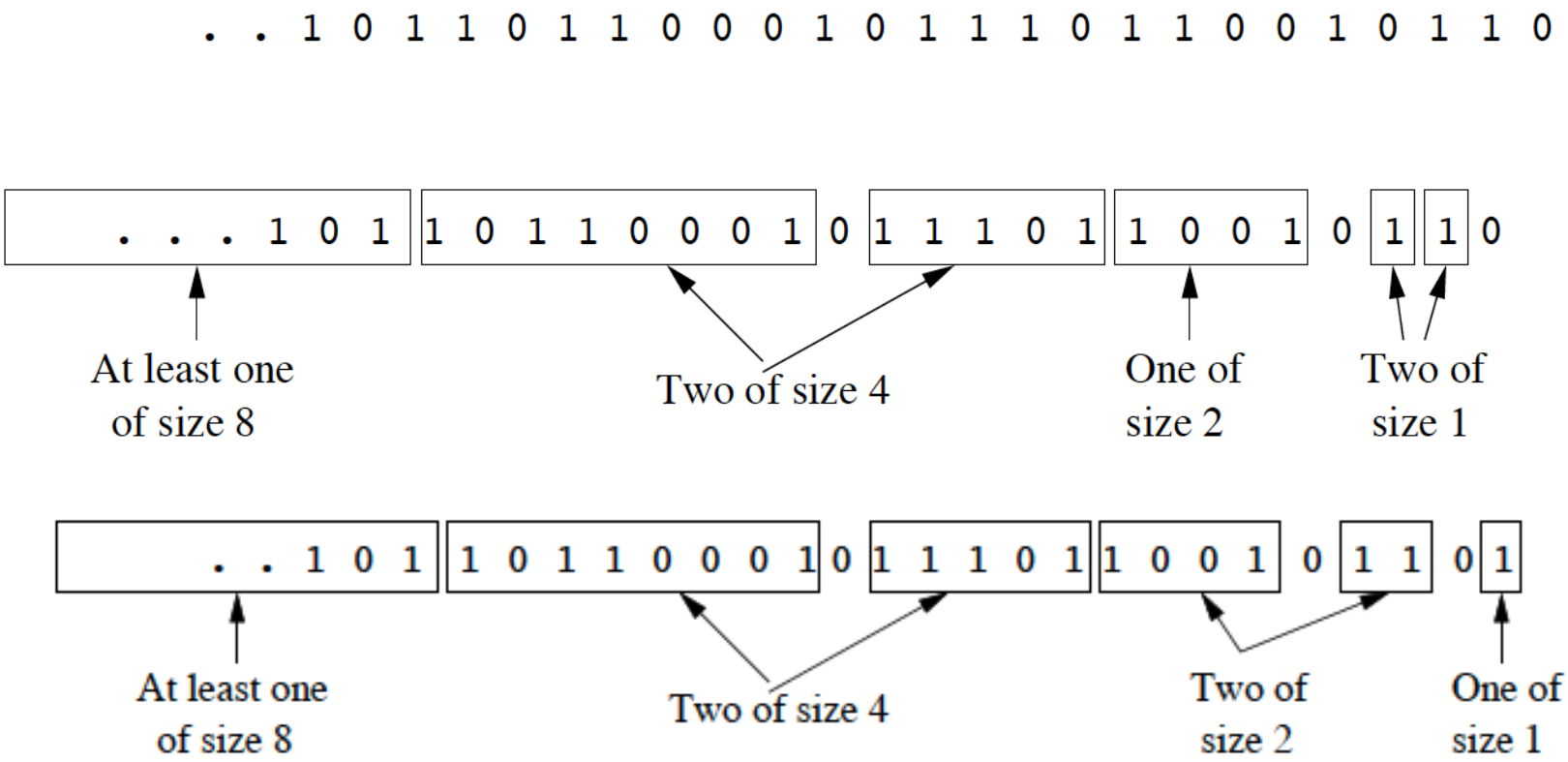
Bucket:	1011100	10100101	100010	11	10	1000
Capacity:	4	4	2	2	1	1
Timestamp:	$t-24$	$t-14$	$t-9$	$t-6$	$t-5$	$t-3$

- Example: the most recent with $t-3$, the least recent with $t-28$ for a desired window size $|W| = 25$,
- our last bucket will be the one that includes the timestamp $t-25$
- so our estimate: $1+1+2+2+4+4/2 = 12$, wile the true count is 11.
- Relative Error: $(12-11)/11 \approx 9\%$. `https://www.cms.waikato.ac.nz/~abifet/book/chapter_4.html`

Reduce Error - trade- off

- The error is introduced only by the oldest bucket b
- it must contain at least one 1 with the timestamp greater than $t - |W|$, otherwise it would have been dropped.
- it also might however contain up to $2^m - 1$ 1s in worst case with a timestamp at most $t - |W|$, that should not be counted.
- Thus the error is at most half of the capacity of the oldest bucket
- There are either k or $k - 1$ buckets of each sizes , but the largest one $\rightarrow$ the relative error is at most: $(2^m)/(2^m + (k-1)(2^{m-1} + \cdots + 1)) \approx 1/2k$.
- with a bound of relative error ϵ, it suffices $k = 1/2\epsilon$. Number of buckets $(\log |W|)/2\epsilon$

Maintaining DGIM condition



Exponential histograms for sliding windows - DGIM

EXPONENTIAL HISTOGRAMS

```
1  Init(k, W):
2      t ← 0
3      create a list of empty buckets
4  Update(b):
5      t ← t + 1
6      if b = 1    ▷ do nothing with 0s
7          let t be the current time
8          create a bucket of capacity 1 and timestamp t
9          i ← 0
10         while there are more than k buckets of capacity 2^i
11             merge the two oldest buckets of capacity 2^i into a
12                 bucket of capacity 2^(i+1): the timestamp of the new bucket
13                 is the largest (most recent) of their two timestamps
14             i ← i + 1
15         remove all buckets whose timestamp is ≤ t − W
16 Query():
17     return the sum of the capacities of all existing buckets
18         minus half the capacity of the oldest bucket
```

Computing Statistics over Sliding Window: The ADWIN algorithm

Why we need to estimate statistics over window? Because can not store all items in the window or want to perform some operation.
Solution: Adaptive Sliding Window Algorithm (ADWIN)[Bifet and Gavalda, 2007] .

ADWIN

A change detector and estimator algorithm using an adaptive size sliding window

Problem

Average of the stream of bits or real values.

ADWIN

Problem

What is the average of the data NOW?

ADWIN

Problem

What is the average of the data NOW?

ADWIN algorithm

- variable length window of recently seen items
- keeps the maximal length window, that is statistically consistent with the hypothesis: there has been no change in the average values $\overline{X}$, and $\overline{X}^2$
- old fragment is dropped iff there is enough evidence that its average differs enough from the other parts

Inputs :

- $\delta \in (0,1)$ confidence value, and $X = x_1, x_2, \ldots, x_t, \ldots$
- x_t is only available at time t, and assume that $x_t \in [0,1]$
- a sample /element $x_t \sim \mathcal{D}_t$ independently, $\mathbb{E}[X]_{X \sim \mathcal{D}_t} = \mu_t$

ADWIN

- $\overline{\mu}_W$ obeserved average over window W, μ_W the unknown true average
- whenever "large enough" (m) subwindows shows "distinct enough"(ϵ_{cut}) averages, we conclude that the corresponding expected values are different.

  m is the harmonic mean of W_0 and W_1:

$$m = \frac{2}{1/|W_0| + 1/|W_1|} \tag{7}$$

$$\epsilon_{cut} = \sqrt{\frac{1}{2m} . \ln \frac{4|W|}{\delta}} \tag{8}$$

- if so, older older portion of the window is dropped

Computing Statistics over Sliding Window: The ADWIN algorithm

Algorithm 1: ADWIN

Input : Sequence $\{x_t\}$ and confidence value δ
Initialization: Window W

for $t > 0$ **do**
  $W \longrightarrow W \cup x_t$ (add items to the head of W)
  do
    Drop elements from the tail of W
  while $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \epsilon_{cut}$ *holds for all split of* W *into* W_0 *and* W_1;
end
Output: $\hat{\mu}_W$

Computing Statistics over Sliding Window: The ADWIN algorithm

Algorithm 2: ADWIN

Input : Sequence $\{x_t\}$ and confidence value δ
Initialization: Window W

for $t > 0$ **do**
  $W \longrightarrow W \cup x_t$ (add items to the head of W)
  do
    Drop elements from the tail of W
  while $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \epsilon_{cut}$ *holds for all split of* W *into* W_0 *and* W_1;
end
Output: $\hat{\mu}_W$

demonstration from : Joao Gama (LIAAD-INESC Porto, University of Porto, Portugal)

Algorithm ADaptive Sliding WINdow

Example

$W =$ 101010110111111

$W_0 =$ 1

ADWIN: Adaptive Windowing Algorithm

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

Algorithm ADaptive Sliding WINdow

Example

$W = \boxed{101010110111111}$

$W_0 = \boxed{1}$ $W_1 = \boxed{01010110111111}$

ADWIN: Adaptive Windowing Algorithm

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

Algorithm ADaptive Sliding WINdow

Example

$W =$ 101010110111111

$W_0 =$ 10 $W_1 =$ 1010110111111

ADWIN: Adaptive Windowing Algorithm

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

Algorithm ADaptive Sliding WINdow

Example

$W =$ 101010110111111

$W_0 =$ 101 $W_1 =$ 010110111111

ADWIN: Adaptive Windowing Algorithm

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

Algorithm ADaptive Sliding WINdow

Example

$W =$ 101010110111111

$W_0 =$ 1010 $W_1 =$ 10110111111

ADWIN: Adaptive Windowing Algorithm

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

Algorithm ADaptive Sliding WINdow

Example

$W =$ 101010110111111

$W_0 =$ 10101 $W_1 =$ 0110111111

ADWIN: Adaptive Windowing Algorithm

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

Algorithm ADaptive Sliding WINdow

Example

$W =$ 101010110111111

$W_0 =$ 101010 $W_1 =$ 110111111

ADWIN: Adaptive Windowing Algorithm

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

Algorithm ADaptive Sliding WINdow

Example

$W =$ 101010110111111

$W_0 =$ 1010101 $W_1 =$ 10111111

ADWIN: Adaptive Windowing Algorithm

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

Algorithm ADaptive Sliding WINdow

Example

$W =$ 101010110111111

$W_0 =$ 10101011 $W_1 =$ 0111111

ADWIN: Adaptive Windowing Algorithm

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

Algorithm ADaptive Sliding WINdow

Example

$W = \boxed{101010110111111}$ $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$: CHANGE DET.!

$W_0 = \boxed{101010110}$ $W_1 = \boxed{111111}$

ADWIN: Adaptive Windowing Algorithm

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

Algorithm ADaptive Sliding WINdow

Example

$W =$ 1**0**1010110111111 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 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
4 **repeat**
5 Drop elements from the tail of W
6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
7 for every split of W into $W = W_0 \cdot W_1$
8 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

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 $W = W \cup \{x_t\}$ (i.e., add x_t to the head of W)
4 **repeat**
5 Drop elements from the tail of W
6 **until** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c$ holds
7 for every split of W into $W = W_0 \cdot W_1$
8 Output $\hat{\mu}_W$

Frame Title

trade-off between reacting quickly to changes and having few false alarm:

1. (False positive rate bound). If μ_t has remained constant within W, the probability that ADWIN shrinks the window at this step is at most δ;
2. (False negative rate bound). Suppose that for some partition of W in two parts $W_0 W_1$ (where W_1 contains the most recent items) we have $|_{0 \cdot 1}| > \epsilon_{\text{cut}}$. Then with probability $1 - \delta$ ADWIN shrinks W to W_1, or shorter.

ADWIN in action

ADWIN can be used for two purposes 1) as change detector and 2) as an estimator of the average.

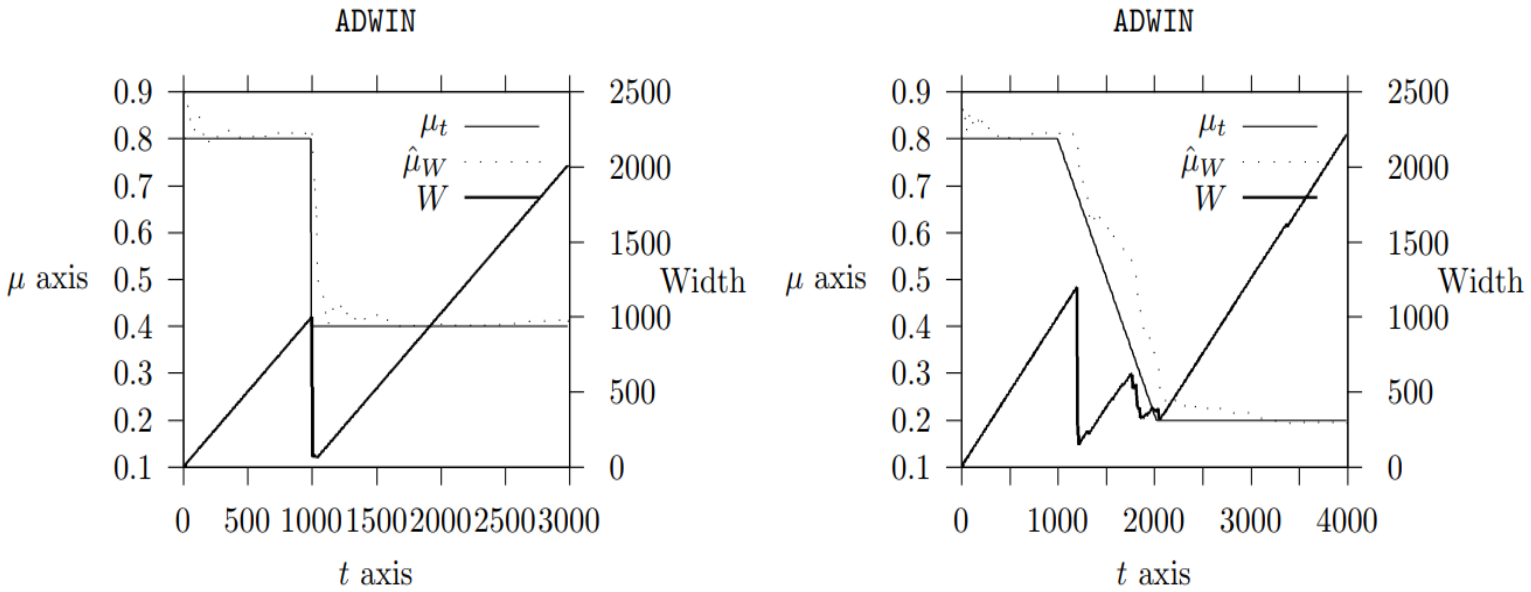


Figure: Output of ADWIN algorithm for different change rates (a) Output with abrupt change (b) output from gradual change.

Bibliography I

Bifet, A. and Gavalda, R. (2007).
Learning from time-changing data with adaptive windowing.
In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM.