Data Stream Mining- Lecture 2

Chandresh Kumar Maurya , Assistant Professor

Eötvös Loránd University, Budapest, Hungary

September 24, 2019

Data Synopsis

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

Why we need sliding window?

Sliding window

Why we need sliding window? For capturing recent data

Types of Sliding window

- **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.

Sequence based window: Examples

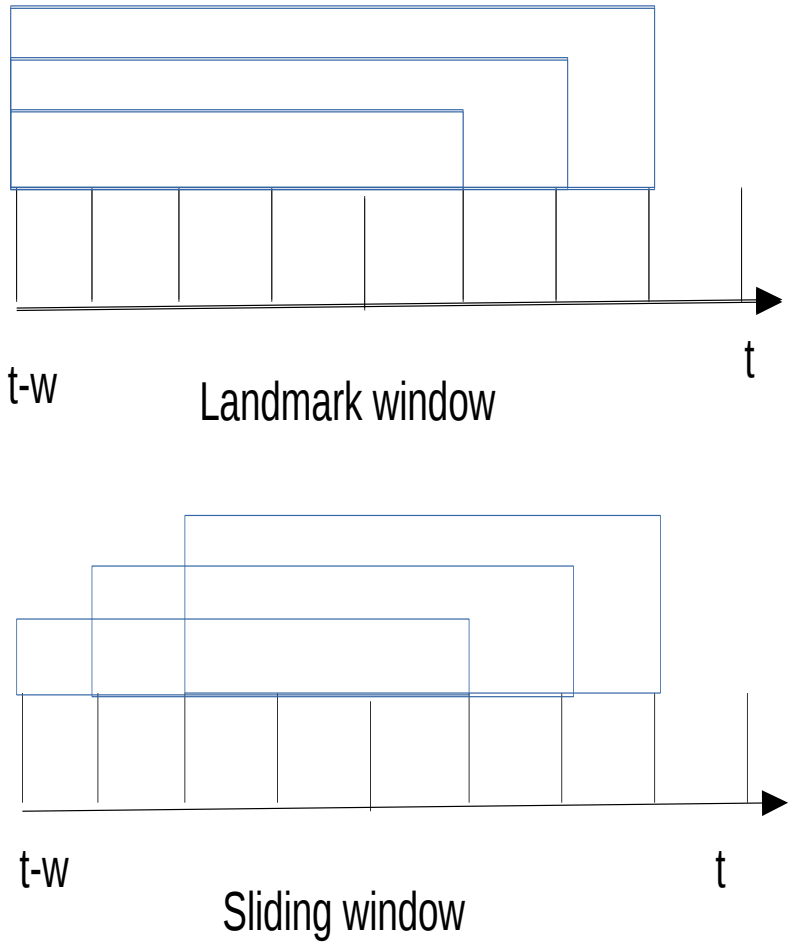


Figure: Sequence based windows. Top figure: Landmark window and bottom figure is sliding window (used in packet transmission)

Timestamp based window: Examples

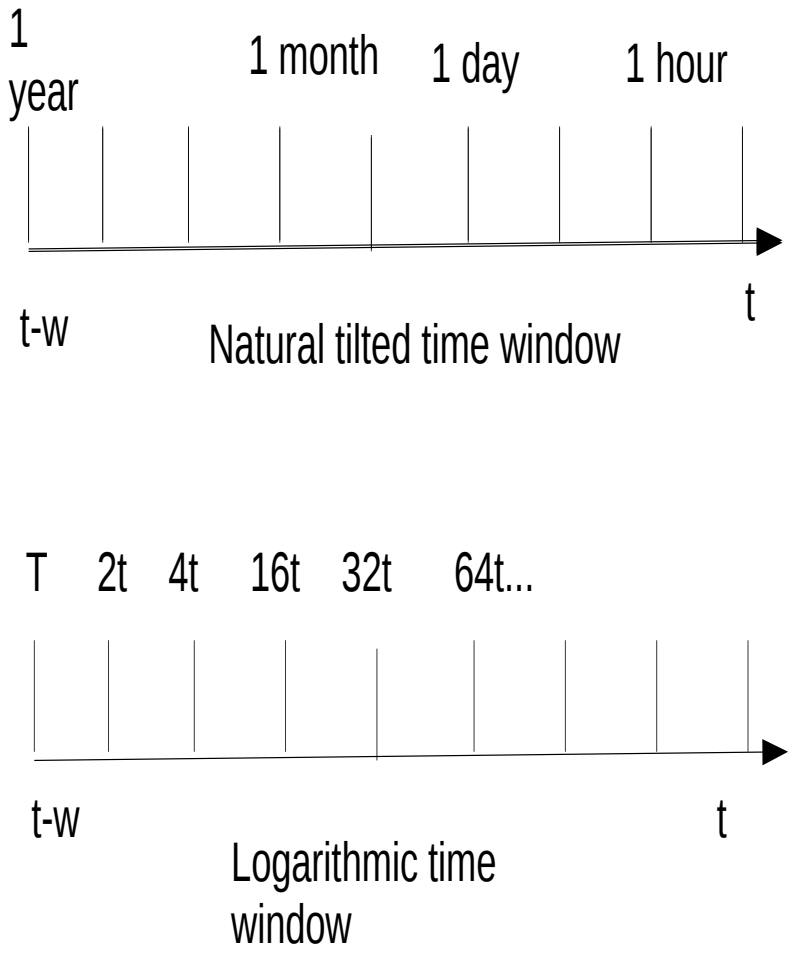


Figure: Timestamp based windows. Top figure: natural tilted window and bottom figure is logarithmic time window

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

Computing Statistics over Sliding Window: The ADWIN algorithm

Algorithm 1: ADWIN

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

1 **for** $t > 0$ **do**
2 $\quad W \longrightarrow W \cup x_t$ (add items to the head of W)
3 $\quad$ **do**
4 $\quad\quad$ Drop elements from the tail of W
5 $\quad$ **while** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \epsilon_{cut}$ *holds for all split of* W *into* W_0 *and* W_1;
6 **end**
7 Output: $\hat{\mu}_W$

Where ϵ_{cut} is given by:

$$\epsilon_{cut} = \sqrt{\frac{1}{2m}.\ln\frac{4|W|}{\delta}}$$ and m is the harmonic mean of W_0 and W_1.

Contd...

How much memory is required to execute ADWIN?

Contd...

How much memory is required to execute ADWIN?
$O(n)$ for window length of n

Contd...

How much memory is required to execute ADWIN?
$O(n)$ for window length of n
Idea: use exponentially increasing window size such as $1, 2, 4, 8, ...$

Contd...

How much memory is required to execute ADWIN?
$O(n)$ for window length of n
Idea: use exponentially increasing window size such as $1, 2, 4, 8, ...$
Time complexity: $O(\log^2 n)$.

ADWIN in action

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

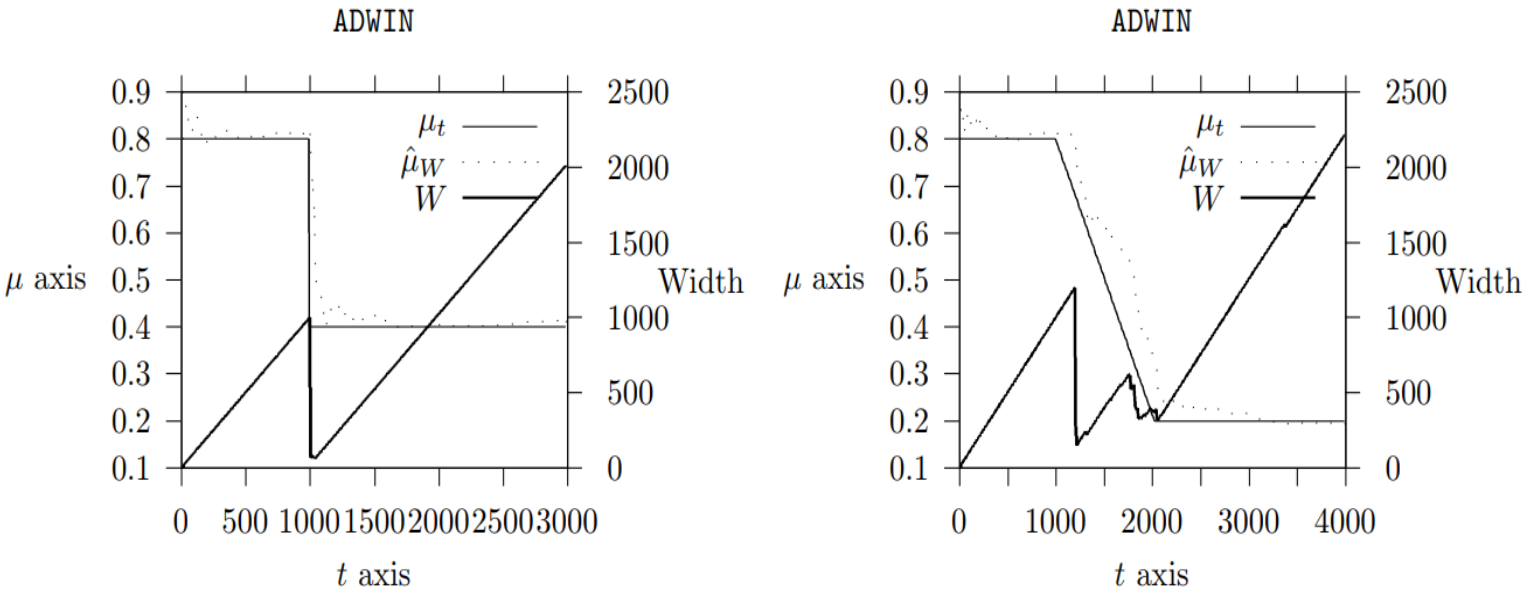


Figure: Output of ADWIN algorithm for different change rates (a) Output with abrupt change (b) output from gradual change.

Count 1's in a Window

The problem: Count 1's in more recent k-bits in the stream of length n:

$$..., 1, 0, 1, 0, 0, 1, 1, 1, 0...$$

Count 1's in a Window

The problem: Count 1's in more recent k-bits in the stream of length n:

$$..., 1, 0, 1, 0, 0, 1, 1, 1, 0...$$

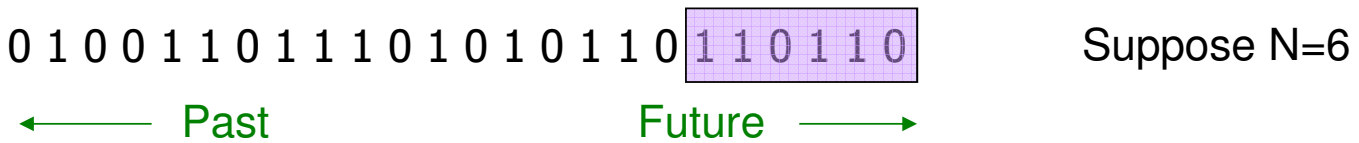
The solution: naive solution requires storing n-bits. But to answer the query takes $O(k)$ times which can be too much time and space. Can you do better without storing all n bits?

Counting Bits (1)

- **Problem:**
  - Given a stream of **0**s and **1**s
  - Be prepared to answer queries of the form
    How many 1s are in the last *k* bits? where $k \leq N$

- **Obvious solution:**

  Store the most recent N bits
  - When new bit comes in, discard the $N+1^{st}$ bit



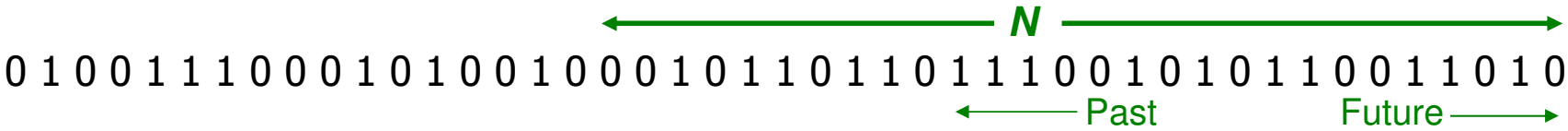
Counting Bits (2)

- **You can not get an exact answer without storing the entire window**
- **Real Problem:**
  What if we cannot afford to store *N* bits?
  - **E.g.**, we're processing 1 billion streams and ***N* = 1 billion**

- **But we are happy with an approximate answer**

An attempt: Simple solution

- **Q: How many 1s are in the last *N* bits?**
- A simple solution that does not really solve our problem: **Uniformity assumption**



- **Maintain 2 counters:**
  - ***S***: number of 1s from the beginning of the stream
  - ***Z***: number of 0s from the beginning of the stream
- **How many 1s are in the last N bits?** $N \cdot \frac{S}{S+Z}$
- **But, what if stream is non-uniform?**
  - What if distribution changes over time?

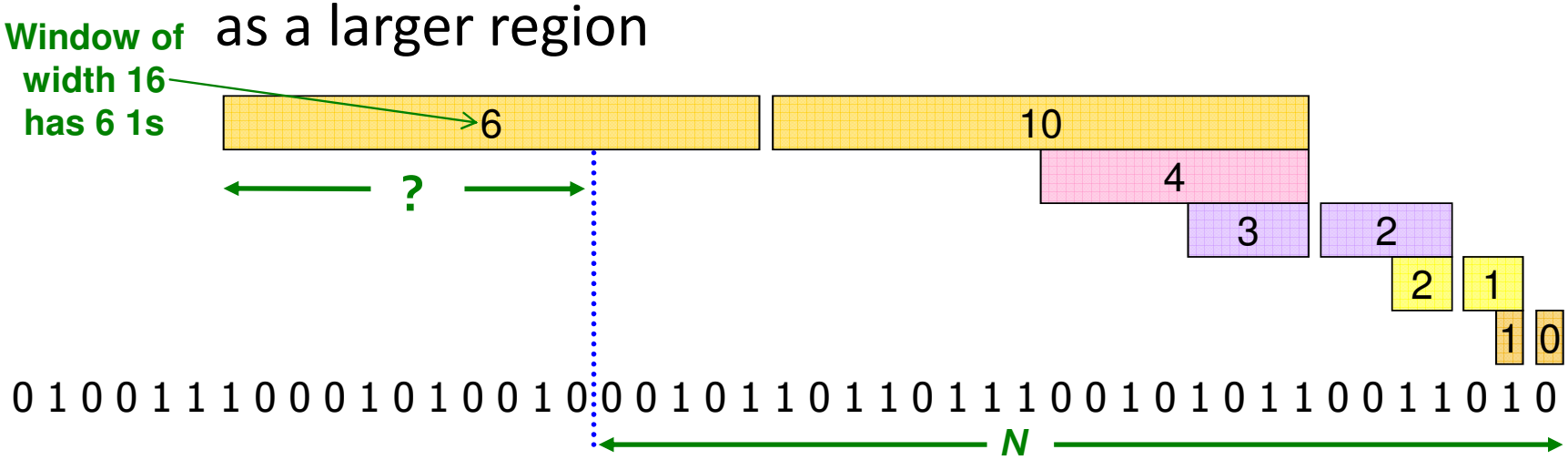
DGIM Method

[Datar, Gionis, Indyk, Motwani]

- **DGIM solution that does not assume uniformity**
- We store $\boldsymbol{O}(\log^2 \boldsymbol{N})$ bits per stream
- **Solution gives approximate answer, never off by more than 50%**
  - Error factor can be reduced to any fraction > 0, with more complicated algorithm and proportionally more stored bits

Idea: Exponential Windows

- **Solution that doesn't (quite) work:**
  - Summarize **exponentially increasing** regions of the stream, looking backward
  - Drop small regions if they begin at the same point as a larger region



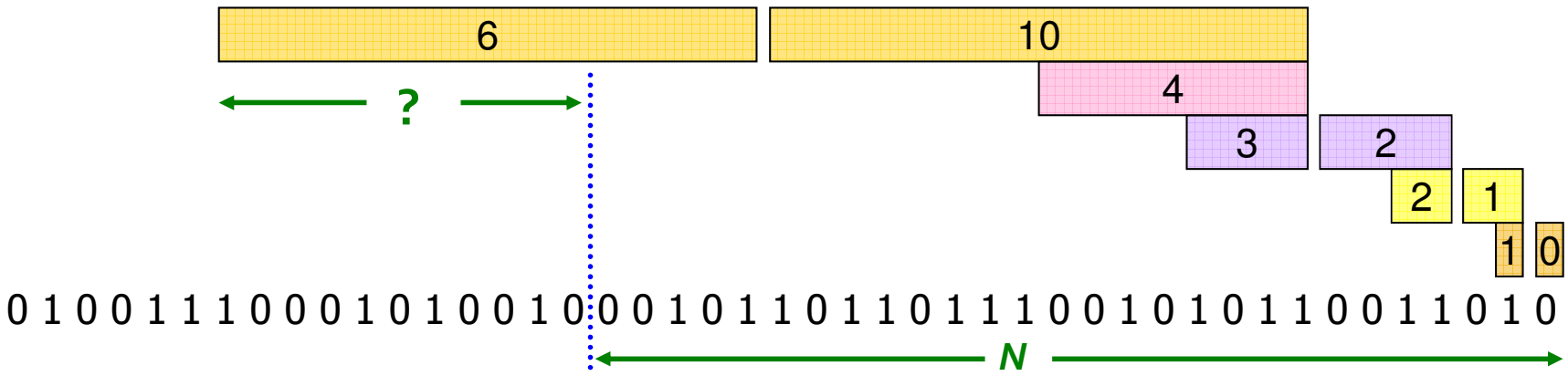
We can reconstruct the count of the last ***N*** bits, except we are not sure how many of the last **6 1s** are included in the ***N***

What's Good?

- **Stores only O($\log^2 N$) bits**
  - $O(\log N)$ counts of $\log_2 N$ bits each
- **Easy update as more bits enter**
- Error in count no greater than the number of **1s** in the "**unknown**" area

What's Not So Good?

- As long as the **1s** are fairly evenly distributed, the error due to the unknown region is small – **no more than 50%**
- But it could be that all the **1s** are in the unknown area at the end
- In that case, **the error is unbounded!**

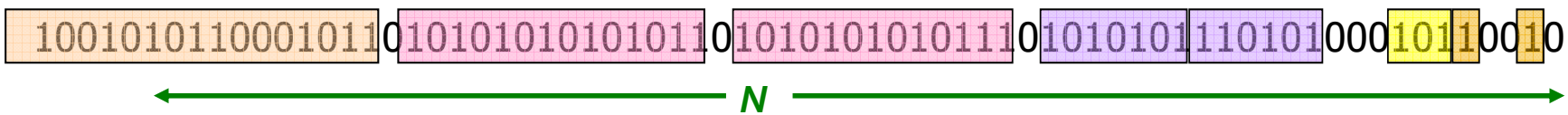


Fixup: DGIM method

[Datar, Gionis, Indyk, Motwani]

- **Idea:** Instead of summarizing fixed-length blocks, summarize blocks with specific number of **1s**:
  - Let the block ***sizes*** (number of **1s**) increase exponentially

- **When there are few 1s in the window, block sizes stay small, so errors are small**



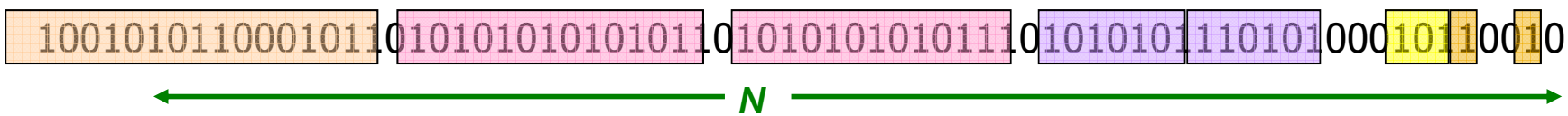
DGIM: Timestamps

- Each bit in the stream has a ***timestamp***, starting **1**, **2,** …
- Record timestamps modulo N (**the window size**), so we can represent any **relevant** timestamp in $O(log_2 N)$ bits

DGIM: Buckets

- A ***bucket*** in the DGIM method is a record consisting of:
  - **(A) The timestamp of its end** [O(log *N*) bits]
  - **(B) The number of 1s between its beginning and end** [O(log log *N*) bits]

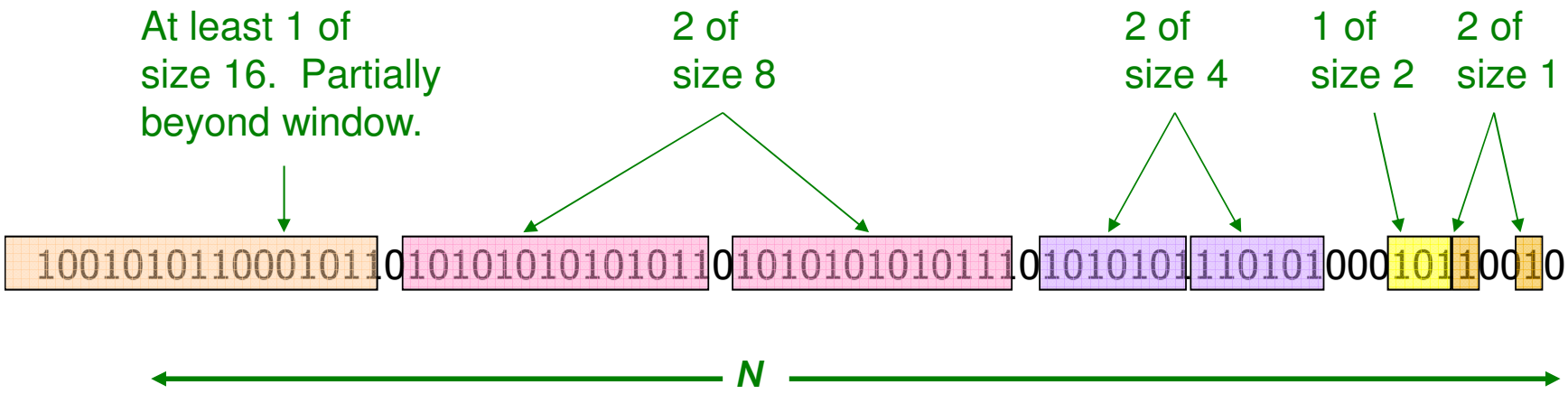
- **Constraint on buckets:**
  Number of **1s** must be a power of **2**
  - That explains the **O(log log *N*)** in **(B) above**



Representing a Stream by Buckets

- Either **one** or **two** buckets with the same **power-of-2 number** of **1s**
- **Buckets do not overlap in timestamps**
- **Buckets are sorted by size**
  - Earlier buckets are not smaller than later buckets
- Buckets disappear when their end-time is **> *N*** time units in the past

Example: Bucketized Stream



Three properties of buckets that are maintained:

- Either **one** or **two** buckets with the same **power-of-2** number of **1s**
- Buckets do not overlap in timestamps
- Buckets are sorted by size

Data Reduction

- Sampling
- Histogram
- Wavelets
- Fourier Transform

Sampling

- Random Sampling
- Reservoir Sampling
- Min-wise Sampling
- Load Shedding

Reservoir Sampling

Intuition: Imagine a swimming pool which can have at most k people at any one time. Swimmers are coming continuously and any new swimmer can go to swim only after old swimmer come out. But, how do you decide which new swimmer to let in?

Reservoir Sampling

Intuition: Imagine a swimming pool which can have at most k people at any one time. Swimmers are coming continuously and any new swimmer can go to swim only after old swimmer come out. But, how do you decide which new swimmer to let in?

Idea: toss a coin (biased) with probability $p = k/n$ where n is the number of swimmers arrived so far. Here assume $k = 1$. If it comes up head, let the swimmer in
otherwise, deny entry.

Reservoir Sampling Algorithm

input: S: stream of values
k: size of the reservoir

```
/* Create uniform sample of fixed size                    */
```

1 Insert first k elements of S to the reservoir.
2 **for** $v \in S$ **do**
3 | Let i be the index of v in S
4 | Generate a random int $j \in [1, i]$
5 | **if** $j < k$ **then**
6 | | Insert v into the reservoir
7 | | Delete an item from the reservoir at random
8 | **end**
9 **end**

Reservoir Sampling Algorithm

What is the probability that the i^{th} item will replace an old item?

Reservoir Sampling Algorithm

What is the probability that the i^{th} item will replace an old item?

Theorem

The probability that the new incoming item will replace the old one is given by $P(a_i \in R) = k/n$ where R denotes the reservoir, k reservoir size and n stream size.

Reservoir Sampling Algorithm

What is the probability that the i^{th} item will replace an old item?

Theorem

The probability that the new incoming item will replace the old one is given by $P(a_i \in R) = k/n$ where R denotes the reservoir, k reservoir size and n stream size.

Proof.

Note that the item i has probability k/i to be inserted into the reservoir R (why?). After that no other item should replace it otherwise it will be gone. Now, next $(i+1)^{th}$ item will not replace it has probability $(1 - \frac{1}{i+1})$, similarly, $(i+2)^{th}$ item will not replace it has probability $(1 - \frac{1}{i+2})$, and so on. Since all of these probabilities are independent (?), we multiply all of them to get the desired result. $P(a_i \in R) = \frac{k}{i} \times \frac{i}{i+1} \times \frac{i+1}{i+2} \times \ldots, \frac{n-1}{n} = \frac{k}{n}$ □

Min-Wise Sampling and Load Shedding

Min-wise

Question: Given a stream of unknwon length, pick k elements randomly so that each elements has the same probability of being chosen.

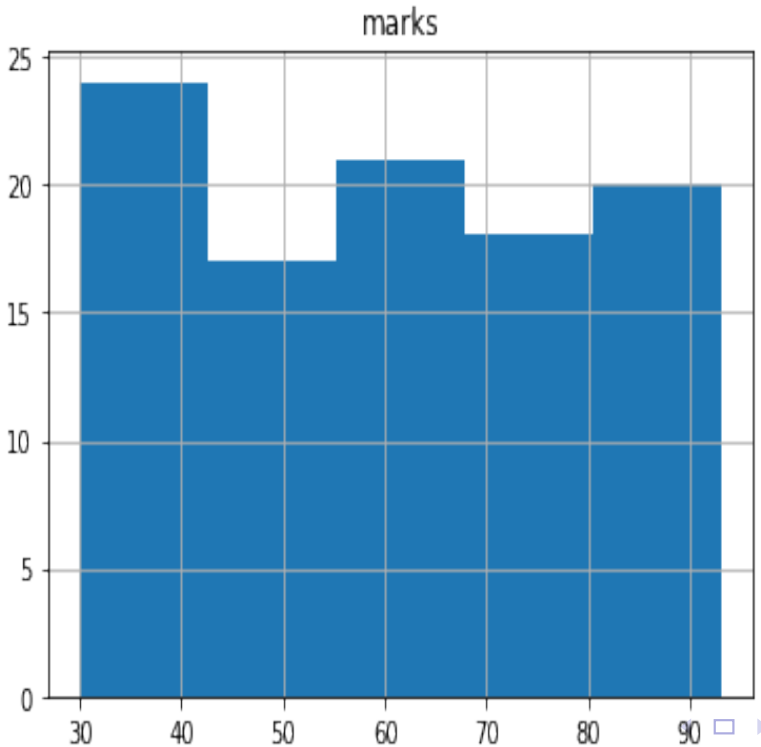
Answer: For each item in the stream, pick a random number in $[0, 1]$, and retain the items with the smallest random number seen so far. Repeat the above process for k smallest number to get a sample of size k.

Load Shedding

Load shedding is used to avoid or control congestion in computer network or server. Whenever arrival rate of the stream is suspected to overload the system, some chunk of data stream is discarded.

Histogram

Summarization technique for values of a feature. Histogram bins values in different buckets whose size is determined by discretizing the values in non-overlapping intervals. Example: Assign grades based on marks. Let's say marks of 100 students are $\{90, 85, 80, 73, 65, 63, 58, 45, 42, 33, ...\}$. Then we can assign grades like as shown in fig.4



Wavelets and Fourier Transforms

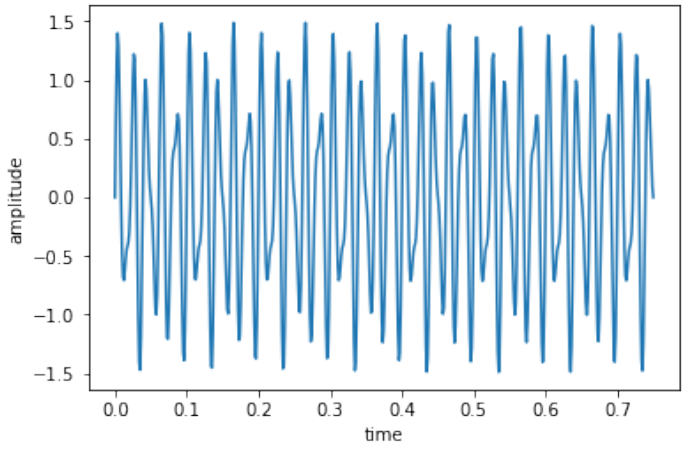
Key points:

- Signal analysis techniques widely used in image processing and data compression.
- Both of them split the data or signal into different components called co-efficients.
- We can reconstruct the signal using these co-efficients.

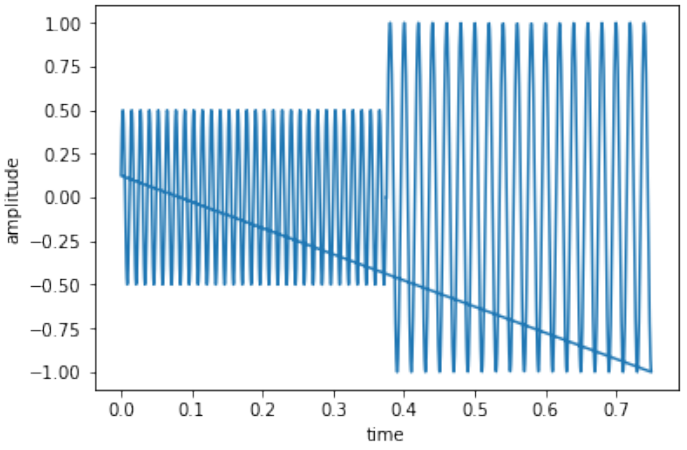
Major differences

Wavelets are used to analyse the non-stationary signal while Fourier transform (FT) are good for analyzing stationary signal.

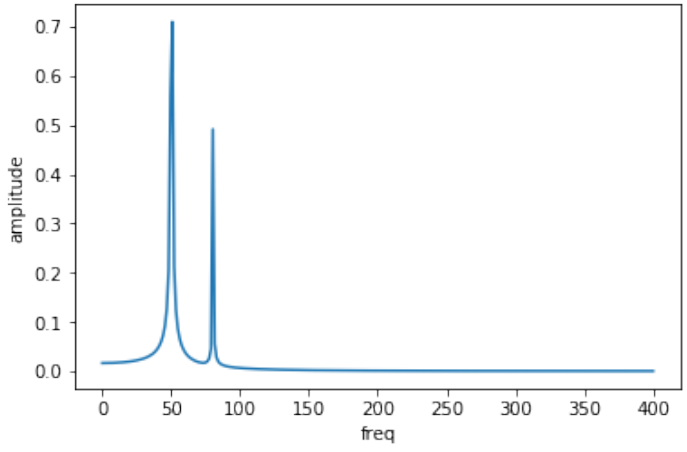
Wavelets and FT differences



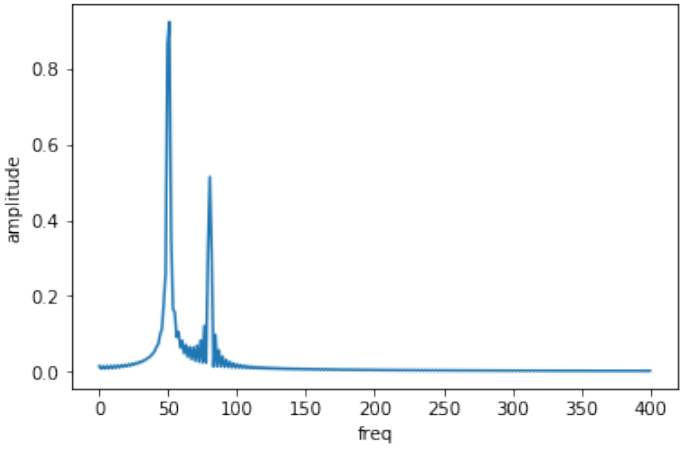
(a) Stationary signal



(b) Non-stationary signal



(c) FFT of fig (a)



(d) FFT of fig (b)

Contd...

Why we need wavelets though FT existed since more than a decade?

Key Differences

Originally signal is represented via time-amplitude plot. But, it has limitations such as can not know how many frequency components are there. Therefore, we devised frequency-amplitude plot which FT gives. Now, if we want to analyse frequency at what point of time, we go for time-frequency plot which is given by wavelets.

Examples

Fourier transform is given by

$$F(\omega) = \int_{-\infty}^{+\infty} f(t) e^{-\omega \pi t} dt \tag{1}$$

Exercise: compute FT of the following functions:

$$f(t) = \begin{cases} 0 & t \leq 0 \\ e^{-t} & t \geq 0 \end{cases}$$

and

$$f(t) = \begin{cases} 0 & t \leq 0 \\ e^{t} & t \geq 0 \end{cases}$$

Contd...

sinusoidal signals: Fourier transform of $f(t) = \cos \omega_0 t$

$$
\begin{aligned}
F(\omega) &= \frac{1}{2}\int_{-\infty}^{\infty} \left(e^{j\omega_0 t} + e^{-j\omega_0 t}\right) e^{-j\omega t}\, dt \\
&= \frac{1}{2}\int_{-\infty}^{\infty} e^{-j(\omega-\omega_0)t} dt + \frac{1}{2}\int_{-\infty}^{\infty} e^{-j(\omega+\omega_0)t} dt \\
&= \pi\delta(\omega-\omega_0) + \pi\delta(\omega+\omega_0)
\end{aligned}
$$

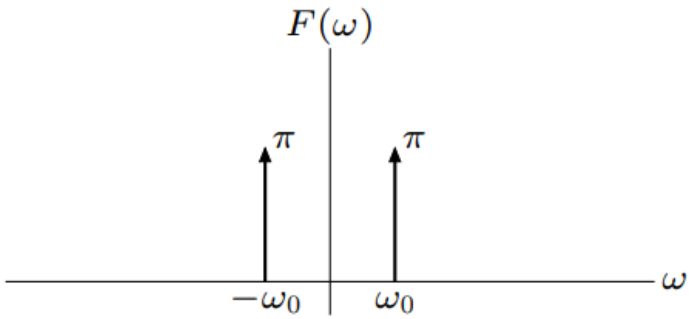


Figure: FT of cosine wave

Wavelets examples

Wavelets use simple functions and some operations such as scaling and translation to express the given function. Simple functions are also called **mother wavelet**.

Translation and scaling

Translation operation on the mother wavelet shifts the location of it to capture the global or slow trends while scaling either compress or elongates the mother signal in order to capture the abrupt changes.

Example: Haar wavelet transform (name given after Hungarian mathematician Haar in 1910).

Contd...

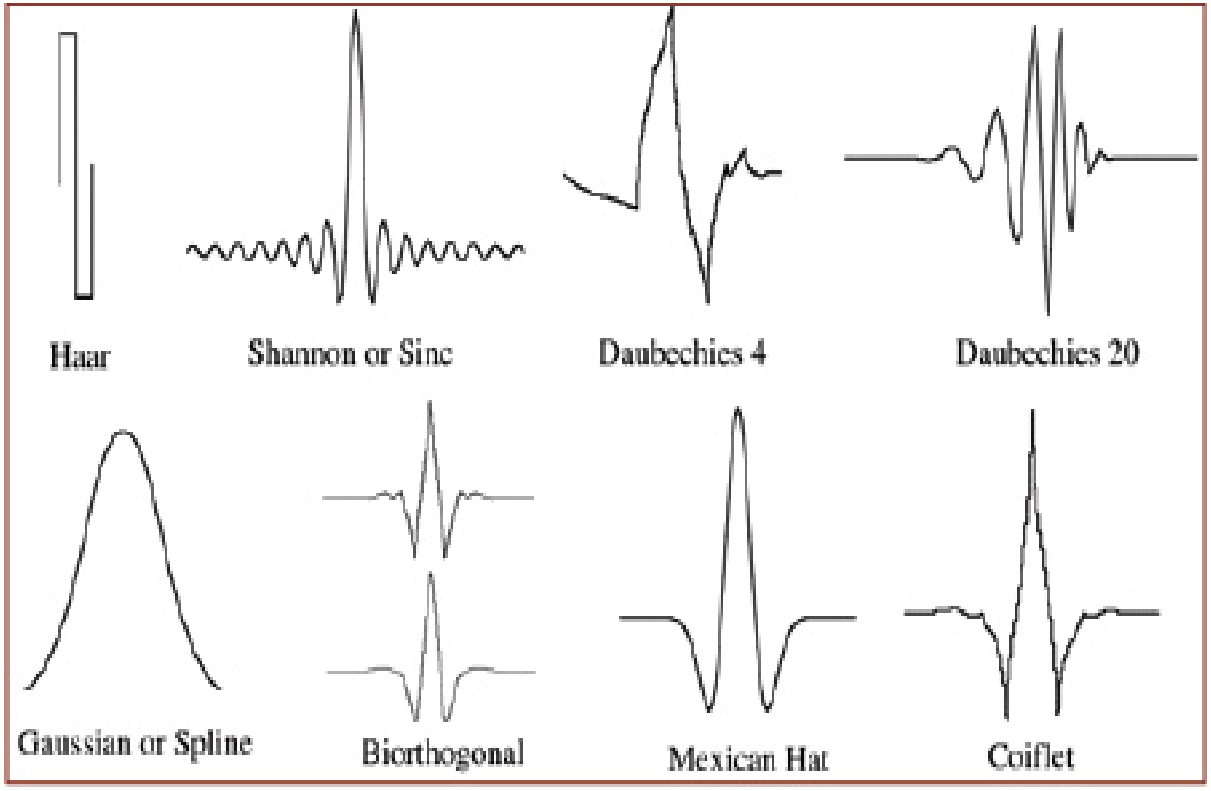


Figure 8

Examples of types of wavelets

Figure: Some standard wavelets

Haar Transform

Haar matrix

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Using the discrete wavelet transform, one can transform any sequence $(a_0, a_1, \ldots, a_{2n}, a_{2n+1})$ of even length into a sequence of two-component-vectors $((a_0, a_1), \ldots, (a_{2n}, a_{2n+1}))$. If one right-multiplies each vector with the matrix H_2, one gets the result $((s_0, d_0), \ldots, (s_n, d_n))$ $((s_0, d_0), \ldots, (s_n, d_n))$ of one stage of the fast Haar-wavelet transform. Usually one separates the sequences s and d and continues with transforming the sequence s. Sequence s is often referred to as the averages part, whereas d is known as the details part.

Contd...

$$\begin{bmatrix} s_i \\ d_i \end{bmatrix} = 1/2 H_2 \begin{bmatrix} x_i \\ x_{i+1} \end{bmatrix} \tag{2}$$

Exercise: compute Haar-transform of the signal $f = (2, 5, 8, 9, 7, 4, -1, 3)$

Example - Haar Wavelets

- Suppose we are given a 1D "image" with a resolution of 4 pixels:

$$[9\ 7\ 3\ 5]$$

- The Haar wavelet transform is the following:

$$[6\ \ 2\ \ 1\ \ -1]$$

$$L_0\ D_1\ D_2\ D_3$$

Example - Haar Wavelets (cont'd)

- Start by averaging the pixels together (pairwise) to get a new lower resolution image:

[8 4] (averaged and subsampled)

- To recover the original four pixels from the two averaged pixels, store some *detail coefficients*.

Resolution	*Averages*	*Detail Coefficients*
4	[9 7 3 5]	[]
2	[8 4]	[1 −1]

Example - Haar Wavelets (cont'd)

- Repeating this process on the averages gives the full decomposition:

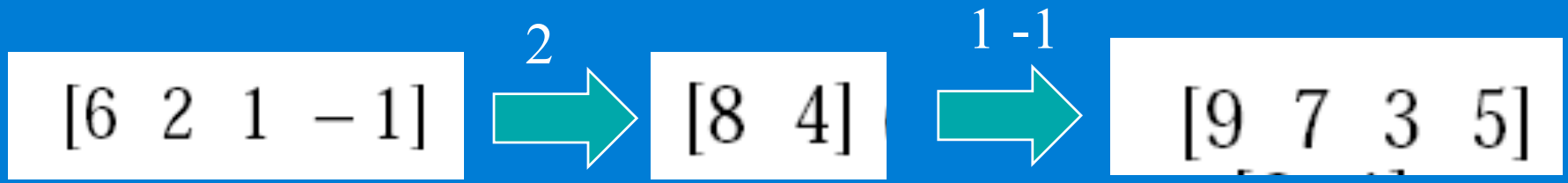
Resolution	*Averages*	*Detail Coefficients*
4	[9 7 3 5]	[]
2	[8 4]	[1 −1]
4	[6]	[2]

Example - Haar Wavelets (cont'd)

- The Harr decomposition of the original four-pixel image is:

$$[6 \quad 2 \quad 1 \quad -1]$$

- We can reconstruct the original image to a resolution by adding or subtracting the detail coefficients from the lower-resolution versions.



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.