Mining Data Streams II

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University
June 21, 2023

TODAY

Mining Data Streams II: Overview

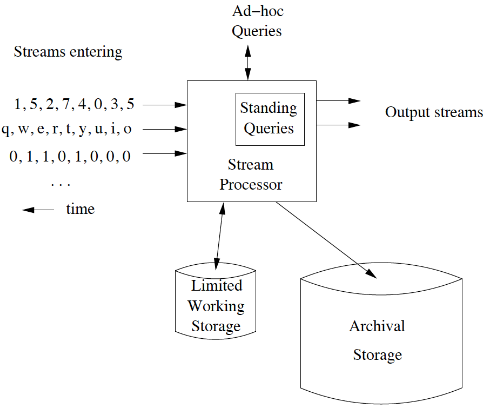
- Counting Ones in a Window:
  ☞ Datar-Gionis-Indyk-Motwani algorithm
- Decaying Windows

Learning Goals: Understand these topics and get familiarized

Counting Ones in a Window

The Datar-Gionis-Indyk-Motwani Algorithm

Data Stream Management System



A data stream management system

Adopted from `mmds.org`

Data Stream Queries

Issues

- Streams deliver elements rapidly: need to act quickly
- Thus, data to work on should fit in main memory
- New techniques required:
- ☞ Compute approximate, not exact answers
- ☞ Hashing is a useful technique

Counting Ones in Window: Problem

- Let $x_i \in \{0, 1\}$, earliest = leftmost = x_1 (unlike before)

$$x_1, ..., x_t,\ \underbrace{x_{t+1}, ..., \overbrace{x_{t+N-k+1}, ..., x_{t+N}}^{k \leq N}}_{\text{window of length } N} \tag{1}$$

- *Situation:*
  - We have a window of length N on a binary stream
  - Query: "how many ones are there in the last $k \leq N$ bits?"
  - We cannot afford to store entire window
  - Approximate algorithms required
- Present solution for binary streams first
- Thereafter extension for summing numbers

The Cost of Exact Counts

- One needs to store N bits to answer count-one-queries for arbitrary $k \leq N$:
  - Assume one could use less than N bits
  - We need 2^N different representations to represent all possible 2^N bit strings of length N
  - Since we use less than N bits, there are two different bit strings $w \neq x$, for which we use the same representation
  - Let k be the first bit from the right where w and x disagree
  - *Example:*
    - For $w = 0101, x = 1010$, we have $k = 1$
    - For $w = 1001, x = 0101$, we have $k = 3$

The Cost of Exact Counts

- One needs to store N bits to answer count-one-queries for arbitrary $k \leq N$:
  - Let k be the first bit from the right where w and x disagree
  - *Example:*
    - For $w = 0101, x = 1010$, we have $k = 1$
    - For $w = 1001, x = 0101$, we have $k = 3$
  - So the counts of ones in the window of length k for w and x differ
  - But because we use identical representations for w and x, we will output the same count
  - This proves that one needs the full N bits to represent bit strings for exact count-one-queries.

THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM

- *Situation:*
  - We consider a binary stream: elements are *bits*
  - Let each element of the stream have a *timestamp*
  - The first, *leftmost* element has timestamp 1, the second leftmost has timestamp 2, and so on; i is timestamp for x_i

$$\underbrace{x_1, \ldots, x_t, x_{t+1}, \ldots, x_M}_{\text{timestamps: } 1,\ldots,t,t+1,\ldots,M} \tag{2}$$

- *Goal:* We like to count the ones among the N most recent (rightmost) elements/bits

The Datar-Gionis-Indyk-Motwani Algorithm

- *Goal:* We like to count the ones among the N most recent (rightmost) elements/bits
- *Space requirements:*
  - Storing timestamps modulo N, and
  - marking rightmost timestamp as most recent
  - allows to store positions of individual bits using $\log_2 N$ bits
- *Illustration:* Let $t = mN + r$.

$$x_1, \ldots, \underbrace{\overbrace{\underbrace{x_{t-N+1}}_{r+1}, \underbrace{x_{t-N+2}}_{r+2}, \ldots, \underbrace{x_{t-r-1}}_{N-1}, \underbrace{x_{t-r}}_{0}, \underbrace{x_{t-r+1}}_{1}, \ldots, \underbrace{x_{t-1}}_{r-1}, \underbrace{x_t}_{r}}^{\text{window of length } N}}_{\text{stored timestamps}} \tag{3}$$

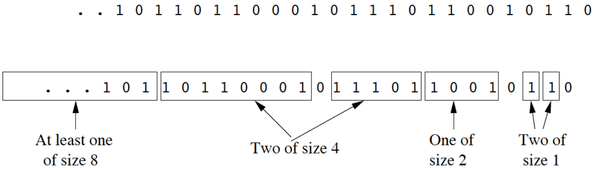
THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM

- *Algorithm:* Divide window into *buckets*, contiguous bit substrings
- *Bucket Representation:* Store
  - The timestamp (TS) of its right end (figure: $t-7$), and
  - The *size* of the bucket, as the number of 1's in the bucket
  - The size is supposed to be a power of 2 (figure: 2^2)

$$\overbrace{0\ \underbrace{[1011\ \underbrace{1}_{x_{t-7}}]}_{\substack{\text{size } 4=2^2 \\ TS:\, t-7}}\ 011000\ \underbrace{1}_{x_t}}^{N} \tag{4}$$

- *Bucket Space Requirements:*
  - Storing buckets: (TS, size)
  - TS between 0 and $N-1$, so requires $\log_2 N$ bits
  - Size: Storing $\log_2 j, 0 \leq j \leq \log_2 N$ amount to $\log_2 \log_2 N$ bits
  - Requires $O(\log N)$ bits overall

DATAR-GIONIS-INDYK-MOTWANI BUCKET RULES



Bit stream divided into buckets following DGIM rules

From `mmds.org`

- Buckets do not overlap; right end always is a 1
- Every 1, but not necessarily every 0 of window in some bucket
- Either one or two buckets for each possible size
- Size cannot decrease on earlier timestamps (moving to the left)

THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM

Key Ideas / Considerations

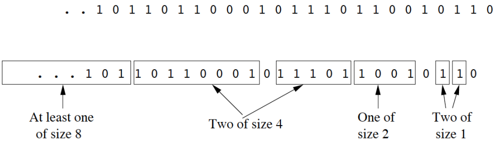
- Number of buckets representing a window must be small
- Estimate number of 1's in last k bits by exploiting known (because stored) bucket structure
- For any k, estimate has error of no more than 50%
- How to re-establish DGIM Bucket Rules quickly, on new bits arriving?

The Datar-Gionis-Indyk-Motwani Algorithm

Storage Requirements Overall

- Each bucket represented using $O(\log N)$ bits (see before)
- Let 2^j be size of largest bucket: $2^j < N$ implies $j \leq \log_2 N$
- So there are at most 2 buckets of sizes $2^j, j = \log_2 N, ..., 1$
- This means that there are $O(\log N)$ buckets overall
- $O(\log N)$ buckets of $O(\log N)$ bits: $O(\log^2 N)$ space overall

THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM



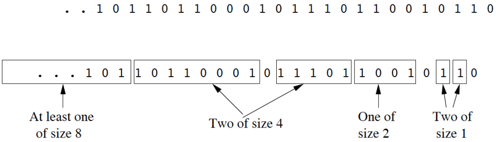
Bit stream divided into buckets following DGIM rules

From `mmds.org`

Answering Queries

- Let $1 \leq k \leq N$: how many 1's are among the last k bits?
- *Answer:*
  - Find leftmost (= with earliest timestamp) bucket b containing some of last k bits
  - *Estimate:* Sum of sizes of buckets right of b plus half the size of b

THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM



Bit stream divided into buckets following DGIM rules

From `mmds.org`

Example

- Let $k = 10$: how many 1's are among 0110010110?
- Let t be timestamp of rightmost bit
- Buckets of timestamps $t-1$, $t-2$ and size 1 fully included in k righmost bits
- Bucket of size 2 with timestamp $t-4$ is also included
- Bucket of size 4 with timestamp $t-8$ is only partially included
- Estimate: $1 + 1 + 2 + (1/2 \times 4) = 6$, one more than true count

DGIM: Error of Estimate

Case 1: estimate is less than c

- Let c be true count; let leftmost bucket b be of size 2^j
- *Worst case:* all 1's in b are among k most recent bits
- *Worst case example:*

$$\overbrace{\underbrace{[1\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\,1]}_{\text{leftmost bucket b}}\;0\,[1\,0\,1\ldots}^{\text{k most recent bits}} \tag{5}$$

  - Left timestamp of leftmost bucket is $t - k + 1$
  - All ones of left most bucket (here: 8) belong to true count
  - Estimate counts only half of them (here: 4)
- Because $c \geq 2^j$, error is at most half of c:

$$\frac{\text{estimate}}{\text{true count}} = \frac{c - 2^{j-1}}{c} = 1 - \frac{2^{j-1}}{c} \overset{c \geq 2^j}{\geq} 1 - \frac{2^{j-1}}{2^j} = \frac{1}{2} \tag{6}$$

DGIM: Error of Estimate

Case 2: estimate is larger than c

- Let c be true count; let leftmost bucket b be of size 2^j
- *Worst case:* only rightmost bit of b is among k most recent bits, and
- There is only one bucket for each of sizes $2^{j-1}, \ldots, 1$
- *Worst case example:*

$$\left[1\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\;\underbrace{\overbrace{1}^{t-k+1}\;\Big]\,0\,\Big[1\,0\,1\,0\,1\,1\Big]\,0\,\Big[1\,0\,0\,1\Big]\,0\,\Big[1\Big]\,\overbrace{0}^{t}}_{\text{k most recent bits} \quad \text{only one bucket of each size}}\right. \tag{7}$$

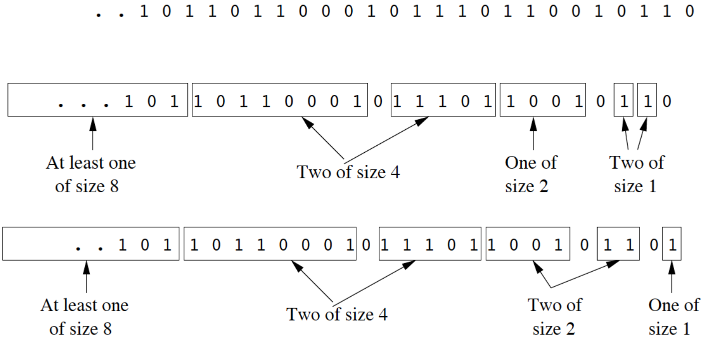
  - Timestamp of leftmost bucket is $t-k+1$
  - Estimate counts half of ones (here: 4); true count is 1
- That yields $c = 1 + 2^{j-1} + \ldots + 1 = 1 + 2^j - 1 = 2^j$
- Estimate is $2^{j-1} + 2^{j-1} + \ldots + 1 = 2^{j-1} + 2^j - 1$, so
- Error $\frac{2^{j-1}+2^j-1}{2^j}$ is no greater than 50% of true count

Maintaining DGIM Rules

Upon a new bit with timestamp t having arrived:

- Check timestamp s of leftmost bucket b:
  - if $s \leq t - N$, drop b from list of buckets
- If the new bit is 0, do nothing
- If the new bit is 1, do
  - Create new bucket with timestamp t and size 1
  - On increasing size, while there are three buckets of the same size, do
    - keep the rightmost bucket of that size as is
    - join the two left buckets into one of double the size
    - where the timestamp is that of the rightmost bit
  - *For example:* joining the two left of the three buckets of size 1 into a bucket of size 2 may create a third bucket of size 2, and so on
- *Runtime:* Need to look at $O(\log N)$ buckets, joining is constant time, so processing new bit requires $O(\log N)$ time overall

THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM
PART VI



Buckets following DGIM rules (top), with new 1 arriving (bottom)

From mmds.org

DGIM Algorithm: Reducing the Error

- For some $r > 2$, allow either r or $r-1$ buckets of the same size
- Allow this for all but size 1 and largest size, whose numbers may be any of 1, ..., r
- Compute estimate as before
- Extend maintaining the DGIM Bucket Rules in the obvious way
- *Recall:* largest error $\frac{2^{j-1}+2^{j}-1}{2^{j}}$ was made when only one 1 from leftmost bucket b was within window

DGIM ALGORITHM: REDUCING THE ERROR

- *Recall:* largest error $\frac{2^{j-1}+2^j-1}{2^j}$ was made when only one 1 from leftmost bucket b was within window
- *New error:*
  - True count is at most $1+(r-1)(2^{j-1}+...+1)=1+(r-1)(2^j-1)$
  - Estimate is $2^{j-1}+(r-1)(2^j-1)$, difference between estimate and true count is $2^{j-1}-1$, so fractional error is

$$\frac{2^{j-1}-1}{1+(r-1)(2^j-1)}$$

    which is upper bounded by $1/2(r-1)$
  - Picking large r can limit error to any $\epsilon > 0$

DGIM Algorithm: Extensions

- DGIM can be extended to integers instead of bits
- Question is to estimate the sum of last $k \leq N$ integers from a window of N integers overall
- However, DGIM cannot be extended to streams containing negative integers
- Consider case of integers in range of 0 to $2^m - 1$, represented by m bits
- *Example:* $m = 3$, integers 0 to 7

$$\begin{array}{rcccccccc} \text{integer stream}: & 2 & 4 & 3 & 1 & 6 & 7 & \ldots \\ \text{bit stream}: & 010 & 100 & 011 & 001 & 110 & 111 & \ldots \end{array} \tag{8}$$

DGIM Algorithm: Extensions

- Consider case of integers in range of 0 to $2^m - 1$, represented by m bits
- *Example:* $m = 3$, integers 0 to 7

$$\begin{array}{rcccccccc} \text{integer stream}: & 2 & 4 & 3 & 1 & 6 & 7 & \ldots \\ \text{bit stream}: & 010 & 100 & 011 & 001 & 110 & 111 & \ldots \end{array} \tag{9}$$

- *Solution:*
  - Treat each bit of integers as separate stream
  - *Example from above:*

$$\begin{array}{ccccccccc} c_0: & \text{rightmost bit} & 0 & 0 & 1 & 1 & 0 & 1 & \ldots \\ c_1: & \text{middle bit} & 1 & 0 & 1 & 0 & 1 & 1 & \ldots \\ c_2: & \text{leftmost bit} & 0 & 1 & 0 & 0 & 1 & 1 & \ldots \end{array} \tag{10}$$

DGIM Algorithm: Extensions

- *Solution:*
  - Treat each bit of integers as separate stream
  - *Example from above:*

$$\begin{array}{lcccccccc} c_0: & \text{rightmost bit} & 0 & 0 & 1 & 1 & 0 & 1 & \ldots \\ c_1: & \text{middle bit} & 1 & 0 & 1 & 0 & 1 & 1 & \ldots \\ c_2: & \text{leftmost bit} & 0 & 1 & 0 & 0 & 1 & 1 & \ldots \end{array} \tag{11}$$

  - Apply DGIM algorithm to each of m streams ☞ estimate c_i for i-th stream
  - *Overall estimate:*

$$\sum_{i=0}^{m-1} c_i 2^i$$

  - If error is at most ϵ for all i, overall error is also at most ϵ

Most Common Elements

Decaying Windows

Decaying Windows: Motivation

- *Stream:* Movie tickets purchased all over the world
- *Goal:* Listing currently most "popular" movies
- *Currently popular:*
  - Movie that sold plenty of tickets years ago not to be listed
  - Movie that sold $2n$ tickets last week, for large n, currently popular
  - Movie that sold n tickets in last 10 weeks is even more popular
  - How to grasp that idea?

Decaying Windows: Motivation

- *Stream:* Movie tickets purchased all over the world
- *Goal:* Listing currently most “popular” movies
- *Possible solution:*
  - One bit stream for each movie
  - i-th bit in a movie stream is 1 if i-th ticket was for that movie
  - *Example:* Three movies M_1, M_2, M_3

$$\begin{array}{lccccc} M_1: & 0 & 0 & 0 & 1 & \dots \\ M_2: & 1 & 0 & 0 & 0 & \dots \\ M_3: & 0 & 1 & 1 & 0 & \dots \end{array} \tag{12}$$

    First ticket to M_2, second and third ticket to M_3, fourth ticket to M_1
  - Pick window of size N, where N is to reflect tickets to be recent

Decaying Windows: Motivation

- *Possible solution:*
  - *Example:* Three movies M_1, M_2, M_3

$$\begin{array}{lccccc} M_1: & 0 & 0 & 0 & 1 & \dots \\ M_2: & 1 & 0 & 0 & 0 & \dots \\ M_3: & 0 & 1 & 1 & 0 & \dots \end{array} \tag{13}$$

    First ticket to M_2, second and third ticket to M_3, fourth ticket to M_1
  - Pick window of size N, where N is to reflect tickets to be recent
  - Estimate number of ones in each stream
    - E.g. use Datar-Gionis-Indyk-Motwani (DGIM) algorithm
    - Estimates number of tickets sold for each movie
  - Rank movies by the estimated counts

DECAYING WINDOWS: MOTIVATION

- *Possible solution, summary:*
  - One bit stream for each movie
  - i-th bit in a movie stream is 1 iff i-th ticket was for that movie
  - Count number of ones in each stream...
  - ... counts tickets for each movie
  - Rank movies by ticket counts
- Works for movies, because there only thousands of movies
- *Drawback:*
  - Does not work for items at Amazon or tweets per Twitter-user
  - ☞ too many items or users

Decaying Windows: Motivation

- *Stream:* Movie tickets purchased all over the world
- *Goal:* Listing currently most "popular" movies
- *Alternative approach:*
  - Do not count ones in fixed-size window
  - Rather, compute "smooth aggregation" of *all* ones in stream
  - Smooth: use weights to rate stream elements in terms of recentness
  - The further back in the stream, the less weight given
  - *Example:* a_t most recent stream element

$$\begin{array}{rccccc} \text{Stream}: & a_1 & a_2 & \cdots & a_{t-1} & a_t \\ \text{Weights}: & w_1 & w_2 & \cdots & w_{t-1} & w_t \end{array} \tag{14}$$

where

$$w_1 \leq w_2 \leq ... \leq w_{t-1} \leq w_t$$

Exponentially Decaying Window: Definition

Definition [Exponentially Decaying Window]:

- Let $a_1, a_2, ..., a_t$ be a stream, with a_t most recent element
- Let c be small constant, e.g. $c \in [10^{-9}, 10^{-6}]$

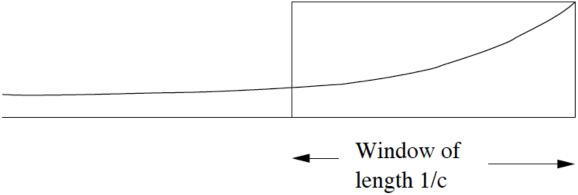
The *exponentially decaying window* for the stream is defined to be

$$\sum_{i=0}^{t-1} a_{t-i}(1-c)^i \tag{15}$$

Weight is $(1-c)^i$, it holds

$$(1-c)^0 \geq (1-c) \geq (1-c)^2 \geq (1-c)^3 \geq ... \geq (1-c)^{t-1}$$

Exponentially Decaying Window: Definition



Decaying window and fixed-length window of equal weight

From mmds.org

- Decaying window puts weight $(1-c)^i$ on $(t-i)$-th element
- A window of length $1/c$ puts equal weight 1 on the first $1/c$ elements
- Both principles distribute the same weight to stream elements overall

Updating Exponentially Decaying Windows

Upon arrival of a new element a_{t+1}, one updates the exponentially decaying window $\sum_{i=0}^{t-1} a_{t-i}(1-c)^i$ by

1. multiplying the current window by $(1-c)$, yielding

$$\sum_{i=0}^{t-1} a_{t-i}(1-c)^{i+1}$$

2. adding a_{t+1}, yielding

$$\sum_{i=0}^{t-1} a_{t-i}(1-c)^{i+1} + a_{t+1} = \sum_{i=0}^{(t+1)-1} a_{(t+1)-i}(1-c)^i$$

Exponentially Decaying Windows: Finding Most Popular Movies

- *Most Popular Movies: Idea*
  - Have a bit stream for each movie, as before
  - Use e.g. $c = 10^{-9}$ ($\approx$ sliding window of size $1/c = 10^9$)
  - On incoming movie ticket sale, update all decaying windows, as described above
    - First, multiply all decaying windows by $1 - c$
    - Add 1 for stream of the movie of the ticket; if there is no stream for that movie, create one
    - Do nothing (add 0) for all other streams
  - If any decaying window drops below threshold of $1/2$, drop window
  - Because the sum of all scores is $1/c$, there cannot be more than $2/c$ movies with score of $1/2$ or more
  - So, $2/c$ is limit on number of movies being tracked at any time
  - In practice, there should be much less movies counted
- *Therefore,* one can apply the technique also for Amazon items and Twitter-users

Materials / Outlook

- See *Mining of Massive Datasets*, chapter 4.6, 4.7
- As usual, see `http://www.mmds.org/` in general for further resources
- Next lecture: "Mining Data Streams III / Social Networks I"
  - See *Mining of Massive Datasets* 4.5; 10.1, 10.2, 10.3