Mining from Data Streams: Issues and Challenges

João Gama
LIAAD-INESC Porto,
University of Porto, Portugal
jgama@fep.up.pt

1. Motivation
2. Data Streams
   - Approximate Answers
   - Count-Min Sketch
3. Basic Methods
   - Estimating statistics over windows
   - Sampling
4. Illustrative Applications
   - Hot-Lists
   - Distributed Streams
5. References

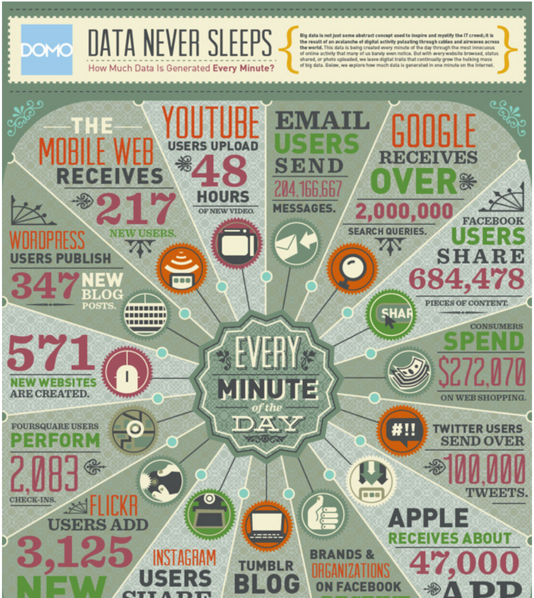
Outline

1. Motivation
2. Data Streams
   - Approximate Answers
   - Count-Min Sketch
3. Basic Methods
   - Estimating statistics over windows
   - Sampling
4. Illustrative Applications
   - Hot-Lists
   - Distributed Streams
5. References

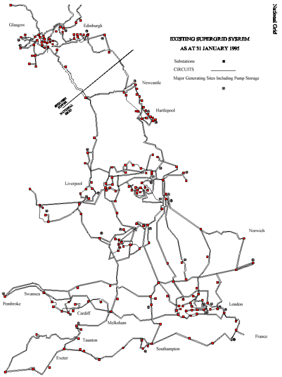
Tribute to Sir Ronald Fisher

Nowadays ...

Data Never Sleeps ...



Scenario



Electrical power Network: Sensors all around network monitor measurements of interest.

Scenario

- Sensors produce continuous flow of data at high speed:
  - Send information at different time scales;
  - Act in adversary conditions: they are prone to noise, weather conditions, battery conditions, etc;
- Huge number of Sensors, variable along time
- Geographic distribution:
  - The topology of the network and the position of the sensors are known.

Illustrative Learning Tasks:

- Cluster Analysis
  - Identification of Profiles: Urban, Rural, Industrial, etc.

Illustrative Learning Tasks:

- Cluster Analysis
  - Identification of Profiles: Urban, Rural, Industrial, etc.
- Predictive Analysis
  - Predict the value measured by each sensor for different time horizons.
  - Prediction of peaks on the demand.

Illustrative Learning Tasks:

- Cluster Analysis
  - Identification of Profiles: Urban, Rural, Industrial, etc.
- Predictive Analysis
  - Predict the value measured by each sensor for different time horizons.
  - Prediction of peaks on the demand.
- Monitoring Evolution
  - Change Detection
    - Detect changes in the behavior of sensors;
    - Detect Failures and Abnormal Activities;
  - Extreme Values, Anomalies and Outliers Detection
    - Identification of peaks on the demand;
    - Identification of **critical points** in load evolution;

Standard Approach:

This problem has been addressed time ago:

Strategy

- Select a finite sample
- Generate a static model (cluster structure, neural nets, Kalman filters, Wavelets, etc)
- Very good performance in next month!
- Six months later: Retrain everything!

Standard Approach:

This problem has been addressed time ago:

Strategy

- Select a finite sample
- Generate a static model (cluster structure, neural nets, Kalman filters, Wavelets, etc)
- Very good performance in next month!
- Six months later: Retrain everything!

What is the Problem?

The world is not static!
Things change over time.

The Data Stream Phenomenon

- Highly detailed, automatic, rapid data feeds.
  - Radar: meteorological observations.
  - Satellite: geodetics, radiation,.
  - Astronomical surveys: optical, radio,.
  - Internet: traffic logs, user queries, email, financial,
  - Sensor networks: many more *observation points* ...
- Most of these data will never be seen by a human!
- Need for near-real time analysis of data feeds.
- Monitoring, intrusion, anomalous activity Classification, Prediction, Complex correlations, Detect outliers, extreme events, etc

Data Streams

Continuous flow of data generated at **high-speed** in **Dynamic**, **Time-changing** environments.

The usual approaches for *querying*, *clustering* and *prediction* use **batch procedures** cannot cope with this streaming setting.

Machine Learning algorithms assume:

- Instances are independent and generated at random according to some probability distribution $\mathcal{D}$.
- It is required that $\mathcal{D}$ is stationary

Practice: *finite* training sets, *static* models.

Data Streams

We need to maintain **Decision models** in **real time**.

Decision Models must be capable of:

- **incorporating** new information at the speed data arrives;
- **detecting** changes and **adapting** the decision models to the most recent information.
- **forgetting** outdated information;

Unbounded training sets, dynamic models.

Outline

1 Motivation

2 Data Streams
- Approximate Answers
- Count-Min Sketch

3 Basic Methods
- Estimating statistics over windows
- Sampling

4 Illustrative Applications
- Hot-Lists
- Distributed Streams

5 References

Data Streams Models

Continuous flow of data generated at **high-speed** in **Dynamic**, **Time-changing** environments.

The input elements $a_1, a_2, \ldots, a_j, \ldots$ arrive sequentially, and describe an underlying function A:

1. Insert Only Model: once an element a_i is seen, it can not be changed;
2. Insert-Delete Model: elements a_i can be deleted or updated.

The domain of variables can be huge.

DBMS / DSMS

Data Base Management Systems

- Persistent relations
- One-time queries
- Random access
- Access plan determined by query processor and physical DB design

Data Streams Management Systems

- Transient streams (and persistent relations)
- Continuous queries
- Sequential access
- Unpredictable data characteristics and arrival patterns

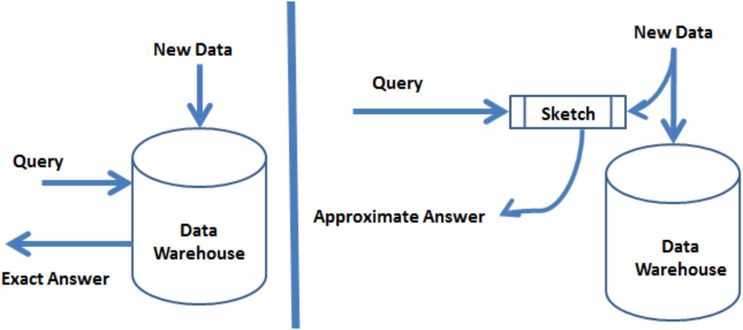
Traditional / Stream Processing

	Traditional	**Stream**
Nr. of Passes	Multiple	Single
Processing Time	Unlimited	Restricted
Memory Usage	Unlimited	Restricted
Type of Result	Accurate	Approximate
Distributed	No	Yes

Massive Data Sets

- Data analysis is **complex**, **interactive**, and **exploratory** over very large volumes of historic data.
- Traditional pattern discovery process **requires on-line ad-hoc queries**, not previously defined, that are successively refined.
- Due to the exploratory nature of these queries, an exact answer may not be required. A user may prefer a **fast approximate answer**.

Massive Data Sets



Approximate Answers

Approximate answers:

Actual answer is within 5 ± 1 *with probability* ≥ 0.9.

- Approximation: find an answer correct within some factor
  - Find an answer that is within 10% of correct result
  - More generally, a $(1 \pm \epsilon)$ factor approximation
- Randomization: allow a small probability of failure
  - Answer is correct, except with probability 1 in 1000
  - More generally, success probability $(1 - \delta)$
- Approximation **and** Randomization: (ϵ, δ)-approximations

The constants ϵ and δ have great influence in the space used. Typically the space is $O(1/\epsilon^2 log(1/\delta))$.

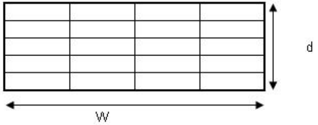
An Illustrative Example: Count-Min Sketch

Cormode & Muthukrishnan. *An improved data stream summary: The count-min sketch and its applications*. Journal of Algorithms, 2005.

Used to approximately solve: Point Queries, Range Queries, Inner Product queries.

Simple sketch idea

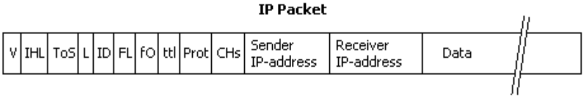
- Creates a small summary as an array of $w \times d$ in size $W = 2/\epsilon$, $d = log(1/\delta)$
- Use d hash functions to map vector entries to $[1..w]$
- Works on Insert-only and Insert-Delete model streams



$$W = 2/\epsilon, \; d = log(1/\delta)$$

Count-Min Sketch

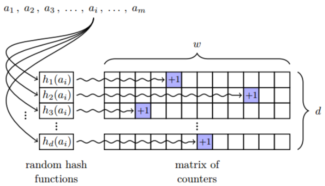
Example: Count the number of packets from the set of IPs that cross a server in a network.



CM Sketch Update

Update:

Each entry in vector x is mapped to one cell per row. Increment the corresponding counter: $CM[k, h_k(j)] += 1$.

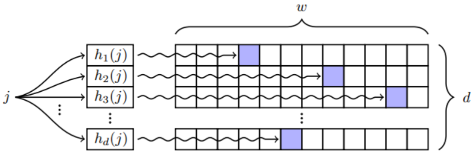


Count-Min Sketch

Example: Count the number of packets from the set of IPs that cross a server in a network.

CM Sketch Query

Query: How many packets from IP j?
Estimate $\hat{x}[j]$ by taking $min_k CM[k, h_k(j)]$



return the minimum of values in blue cells

The estimate guarantees:

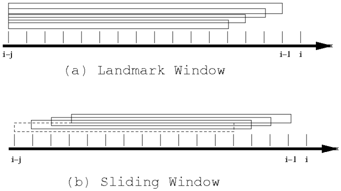
- $x[j] \leq \hat{x}[j]$
- $\hat{x}_i \leq \epsilon \times ||x_i||_1$, with probability $1 - \delta$.

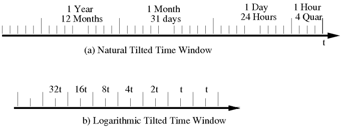
Outline

1. Motivation
2. Data Streams
   - Approximate Answers
   - Count-Min Sketch
3. Basic Methods
   - Estimating statistics over windows
   - Sampling
4. Illustrative Applications
   - Hot-Lists
   - Distributed Streams
5. References

Time Windows

- Instead of computing statistics over all the stream ...
- use only the most recent data points.
- Most recent data is more relevant than older data
- Several Window Models: **Landmark**, **Sliding**, **Tilted** Windows.
  - time based
  - sequence based





Landmark Windows

- The recursive version of the sample mean:

$$\bar{x}_i = \frac{(i-1) \times \bar{x}_{i-1} + x_i}{i} \tag{1}$$

- Incremental version of the standard deviation:

$$\sigma_i = \sqrt{\frac{\sum x_i^2 - \frac{(\sum x_i)^2}{i}}{i-1}} \tag{2}$$

- Recursive correlation coefficient.

$$corr(x,y) = \frac{\sum(x_i \times y_i) - \frac{\sum x_i \times \sum y_i}{n}}{\sqrt{\sum x_i^2 - \frac{\sum x_i^2}{n}}\sqrt{\sum y_i^2 - \frac{\sum y_i^2}{n}}} \tag{3}$$

Sliding Windows

- Computing these statistics in sliding windows:
  requires to maintain all the observations inside the window.
- Simple Moving Average:

$$SMA_t = \frac{(x_t + x_{t-1} + \ldots + x_{t-(n-1)})}{n}$$

  where n is the window size
- Weighted moving average
  use multiplicative factors to give different weights to different data points

$$WMA_t = \frac{nx_t + (n-1)x_{t-1} + \cdots + 2x_{t-n+2} + x_{t-n+1}}{n + (n-1) + \cdots + 2 + 1}$$

Exponential moving average

The weight for each data point decreases exponentially, giving more importance to recent observations while still not discarding older observations entirely.

$$S_t = \alpha \times Y_{t-1} + (1 - \alpha) \times S_{t-1}$$

where α represents the degree of weighting decrease, a constant smoothing factor between 0 and 1. A higher α discounts older observations faster.

Sliding Windows

Maintaining Stream Statistics over Sliding Windows,
M.Datar, A.Gionis, P.Indyk, R.Motwani; ACM-SIAM;2002
The basic idea:

- Use buckets of exponentially growing size $(2^0, 2^1, 2^2 \ldots 2^h)$ to hold the data
- Each bucket has a time-stamp associated with it
- It is used to decide when the bucket is out of the window

Data Structures for Exponential Histograms:

- Buckets: counts and time stamp
- LAST: stores the size of the last bucket.
- TOTAL: keeps the total size of the buckets.

Exponential Histograms

When a new data element arrives:

- Create a new bucket of size 1 with the current time-stamp, and increment the counter TOTAL.
- Given a relative error,ϵ, if there are $|1/\epsilon|/2 + 2$ buckets of the same size, merge the oldest two of the same-size into a single bucket of double size.
- The larger time-stamp of the two buckets is then used as the time-stamp of the newly created bucket.

Whenever we want to estimate the moving sum:

- Check if the oldest bucket is within the sliding window.
- If not, we drop that bucket and subtract its size from the variable TOTAL
- Repeat the procedure until all the buckets with timestamps outside of the sliding window are dropped.
- The estimate of 1's in the sliding window is TOTAL-LAST$/2$.

Exponential Histograms: Example

Count the number of 1's in a sliding window

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$;
- Merge if $\lceil 1/0.5 \rceil /2 + 2 = 3$ buckets of the same size.

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 1, x = 1

EH: $\boxed{1_1}$
TOTAL: 1

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 2, x = 1

EH: | 1_1 | 1_2 |

TOTAL: 2

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 3, x = 1

EH: | 1_1 | 1_2 | 1_3 |

TOTAL: 3

Merge

EH: | 2_2 | 1_3 |

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 4, x = 1

EH: 2_2 | 1_3 | 1_4

TOTAL: 4

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 5, x = 0

EH: 2_2 | 1_3 | 1_4

TOTAL: 4

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 6, x = 1

EH: 2_2 | 1_3 | 1_4 | 1_5

TOTAL: 5

Merge

EH: 2_2 | 2_4 | 1_6

TOTAL: 5

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 7, x = 0

EH: | 2_2 | 2_4 | 1_6 |

TOTAL: 5

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 8, x = 1

EH: | 2_2 | 2_4 | 1_6 | 1_8 |

TOTAL: 6

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 9, x = 1

EH: | 2_2 | 2_4 | 1_6 | 1_8 | 1_9 |

TOTAL: 7

Merge

EH: | 4_4 | 2_8 | 1_9 |

TOTAL: 7

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 10, x = 1

EH: 4_4 | 2_8 | 1_9 | 1_{10}

TOTAL: 8

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 11, x = 1

EH: | 4_4 | 2_8 | 2_{10} | 1_{11} |

TOTAL: 9

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 12, x = 1

EH: | 4_4 | 2_8 | 2_{10} | 1_{11} | 1_{12} |

TOTAL: 10

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 13, x = 1

EH: 4_4 | 4_{10} | 2_{12} | 1_{13}

TOTAL: 11

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 14, x = 1

EH:

4_4	4_{10}	2_{12}	1_{13}	1_{14}

TOTAL: 12

Exponential Histograms: Example

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Element	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0

- Window length=10;
- Relative Error $\epsilon = 0.5$ Merge if 3 buckets of the same size.

Time = 15, x = 0

EH: 4_4 | 4_{10} | 2_{12} | 1_{13} | 1_{14}

TOTAL: 12

Removing outdated buckets

EH: 4_{10} | 2_{12} | 1_{13} | 1_{14}

TOTAL: 8

Time	Data	Buckets	Total	Last
T1	1	1_1	1	1
T2	1	$1_1, 1_2$	2	1
T3	1	$1_1, 1_2, 1_3$	3	1
(merge)		$2_2, 1_3$	3	2
T4	1	$2_2, 1_3, 1_4$	4	2
T5	0	$2_2, 1_3, 1_4$	4	2
T6	1	$2_2, 1_3, 1_4, 1_6$	5	2
		$2_2, 2_4, 1_6$	5	2
T7	0	$2_2, 2_4, 1_6$	5	2
T8	1	$2_2, 2_4, 1_6, 1_8$	6	2
T9	1	$2_2, 2_4, 1_6, 1_8, 1_9$	7	2
		$4_4, 2_8, 1_9$	7	4
T10	1	$4_4, 2_8, 1_9, 1_{10}$	8	4
T11	1	$4_4, 2_8, 2_{10}, 1_{11}$	9	4
T12	1	$4_4, 2_8, 2_{10}, 1_{11}, 1_{12}$	10	4
T13	1	$4_4, 4_{10}, 2_{12}, 1_{13}$	11	4
T14	1	$4_4, 4_{10}, 2_{12}, 1_{13}, 1_{14}$	12	4
(Removing outdated)				
T15	0	$4_{10}, 2_{12}, 1_{13}, 1_{14}$	8	4

Exponential Histograms: Analysis

- The size of the buckets grows exponentially: $2^0, 2^1, 2^2 \ldots 2^h$
- Need only $O(logN)$ buckets.
- It is shown that, for N 1's in the sliding window, we only need $O(logN/\epsilon)$ buckets to maintain the moving sum.
- The error in the oldest bucket **only**.
- The moving sum is proven to be bounded within the given relative error, ϵ.

Sampling

To obtain an unbiased sampling of the data, we need to know the length of the stream.

In Data Streams, we need to modify the approach!

When and How often should we sample?

Strategy

- Sample instances at periodic time intervals
- Useful to *slow down* data.
- Involves *loss* of information.

Sampling

To obtain an unbiased sampling of the data, we need to know the length of the stream.

In Data Streams, we need to modify the approach!

When and How often should we sample?

Strategy

- Sample instances at periodic time intervals
- Useful to *slow down* data.
- Involves *loss* of information.

Methods

- Reservoir Sampling, Vitter, 1985
- Min-Wise Sampling, Broder, *et al.*, 00

The reservoir Sample Technique

Vitter, J.; *Random Sampling with a Reservoir*, ACM, 1985.

- Creates uniform sample of fixed size k;
- Insert first k elements into sample
- Then insert ith element with prob. $p_i = k/i$
- Delete an instance at random.

Analysis

Analyze the simplest case: sample size m = 1

Probability i'th item is the sample from a stream length n:

$$\frac{1}{2} \times \frac{2}{3} \ldots \times \frac{i}{i+1} \times \ldots \times \frac{n-2}{n-1} \times \frac{n-1}{n}$$

$$= 1/n$$

Analysis

Known Problems

Low probability of detecting:

- Changes
- Anomalies

Hard to parallelize

Outline

1. Motivation
2. Data Streams
   - Approximate Answers
   - Count-Min Sketch
3. Basic Methods
   - Estimating statistics over windows
   - Sampling
4. Illustrative Applications
   - Hot-Lists
   - Distributed Streams
5. References

Illustrative Problem I

A (real) data warehouse problem

- Suppose you have a retail data warehouse
- 3 TB of data
- 100s GB new sales records updated daily
- Millions of different items

Problem: hot-list

Identify *hot items*: the top-20 items in popularity

Restricted memory: Can have a memory of 100s-1000s bytes only

Illustrative Examples

- We see a large number of individual transactions.
  - What are the top sellers today?
- We are monitoring network traffic.
  - Which hosts/subnets are responsible for most of the traffic?
- We have a network of satellites monitoring events over large areas.
  - Which areas are experiencing the most activity over a week / day /hour?

The Top-k Elements Problem

Count the top-K most frequent elements in a stream.

First Approach

Maintain a count for each element of the alphabet.
Return the *k* first elements in the sorted list of counts.

Problems

Exact and Efficient solution for small alphabets.
Large alphabets: Space inefficient – large number of zero counts.

The Space Saving Algorithm

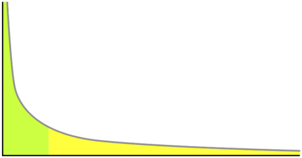
Metwally, D. Agrawal, A. Abbadi, *Efficient Computation of Frequent and Top-k Elements in Data Streams*, ICDT 2005

Maintain partial information of interest; monitor only a subset m of elements.

- For each element e in the stream
  - If e is monitored: Increment $Count_e$
  - Else
    - Let e_m be the element with least hits min.
    - Replace e_m with e with $count_e = min + 1$

The Space Saving Algorithm: Properties

- Efficient for skewed data!
- Ensures no false negatives are kept in the top-k list: no non frequent item is in the top-k list.
- It allows false positive in the list: some non frequent items appear in the list.
- If the popular elements evolve over time, the elements that are growing more popular will gradually be pushed to the top of the list.



Illustrative Problem II

Air Quality Monitoring

- Sensors monitoring the concentration of air pollutants.
- Each sensor holds a data vector comprising measured concentration of various pollutants (CO_2, SO_2, O_3, etc.).
- A function on the average data vector determines the Air Quality Index (AQI)
- Issue an alert in case the AQI exceeds a given threshold.

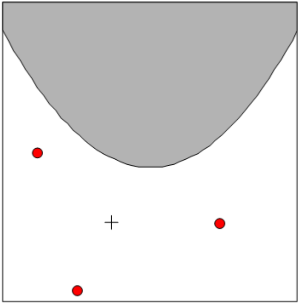
Distributed Monitoring:

- Given:
  - A function over the average of the data vectors
  - A predetermined threshold
- Continuous Query: Alert when function crosses the threshold
- Goal: Minimize communication during query execution

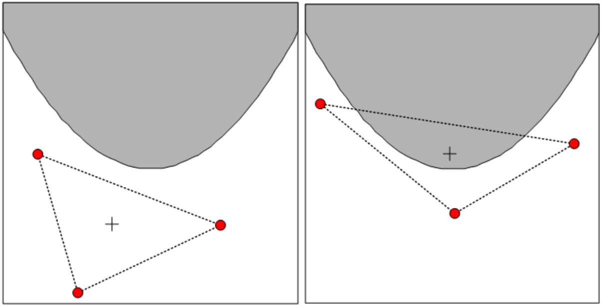
Example: Geometric Approach

I. Sharfman, A. Schuster, D. Keren, *A Geometric Approach to Monitoring Distributed DataStreams*, SIGMOD 2006

- Geometric Interpretation:
  - Each node holds a statistics vector
  - Coloring the vector space :
    - Grey:
      function $>$ threshold
    - White:
      function $\leq$ threshold
- Goal: determine color of global data vector (average).

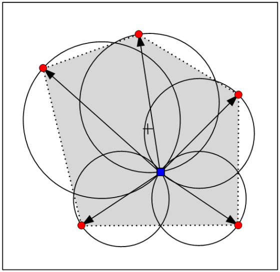


Monitoring Threshold Functions



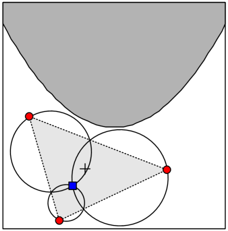
The Bounding Theorem

- A reference point is known to all nodes
- Each vertex constructs a sphere
- Theorem: convex hull is bounded by the union of spheres
    - Local constraints!



Basic Algorithm

- An initial estimate vector is calculated;
- Nodes compute spheres and check its color;
    - Drift vector is the diameter of the sphere
- If any sphere non monochromatic: node triggers re-calculation of estimate vector



Monitoring Threshold Functions

Analysis

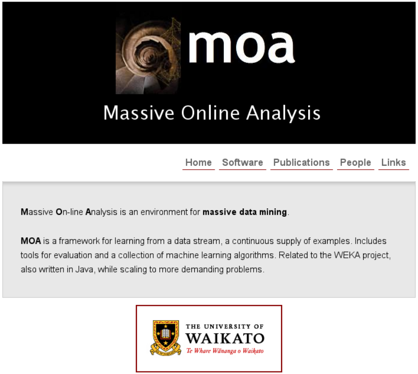
- Mostly Local Computations
- Minimum communications

Outline

1. Motivation
2. Data Streams
   - Approximate Answers
   - Count-Min Sketch
3. Basic Methods
   - Estimating statistics over windows
   - Sampling
4. Illustrative Applications
   - Hot-Lists
   - Distributed Streams
5. References

Software

http://moa.cms.waikato.ac.nz/

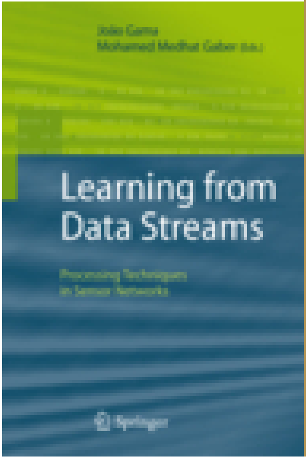


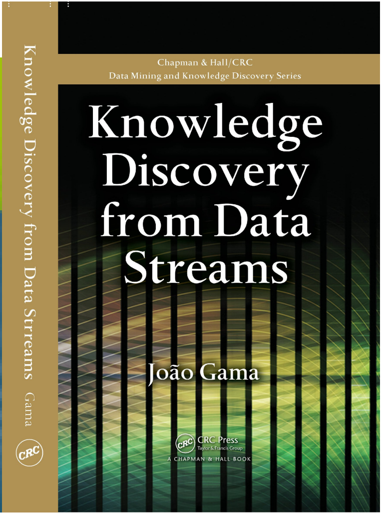
Resources

- Massive Data Analysis
  http://dimacs.rutgers.edu/∼graham/
- Distributed Data Mining
  Maintained by Hillol Kargupta
- UCR Time-Series Data Sets
  Maintained by Eamonn Keogh, UCR, US
  http://www.cs.ucr.edu/∼eamonn/time_series_data
- Mining Data Streams Bibliography
  Maintained by Mohamed Gaber
  http://www.csse.monash.edu.au/∼mgaber/
  WResources.html

Data Stream Management Systems

- Niagara (OGI/Wisconsin) – Internet XML databases
- Aurora (Brown/MIT) – sensor monitoring, dataflow
  `http://www-db.stanford.edu/sdt`
- Stream (Stanford) – general-purpose DSMS
  `http://www-db.stanford.edu/stream/index.html`
- COUGAR (Cornell)
- GigaScope and Hancock (At&T)
- Medusa (Brown University)


Learning from Data Streams


Knowledge Discovery from Data Streams
Gama
Chapman & Hall/CRC
Data Mining and Knowledge Discovery Series
Knowledge Discovery from Data Streams
João Gama
CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Master References

- J. Gama, *Knowledge Discovery from Data Streams*, CRC Press, 2010.
- S. Muthukrishnan *Data Streams: Algorithms and Applications*, Foundations & Trends in Theoretical Computer Science, 2005.
- B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems, in Proc. PODS, 2002.
- Gaber, M, M., Zaslavsky, A., and Krishnaswamy, S., Mining Data Streams: A Review, in ACM SIGMOD Record, Vol. 34, No. 1, 2005.
- C. Aggarwal, *Data Streams: Models and Algorithms*, Ed. Charu Aggarwal, Springer, 2007
- J. Gama, M. Gaber (Eds), *Learning from Data Streams – Processing Techniques in Sensor Networks*, Springer, 2007.

Sampling and Synopsis

- Cormode & Muthukrishnan. *An improved data stream summary: The count-min sketch and its applications*. Journal of Algorithms, 2005.
- A. Arasu, G. Manku, *Approximate Counts and Quantiles over Sliding Windows*, in PODS 2004.
- M. Datar, A. Gionis, P. Indyk, R. Motwani; *Maintaining Stream Statistics over Sliding Windows*, in the ACM-SIAM Symposium on Discrete Algorithms (SODA) 2002.
- J. Vitter.*Random sampling with a reservoir*, ACM Transactions on Mathematical Software, 1985.
- A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher; *Min-wise independent permutations*, Journal of Computer and System Sciences, 2000
- A. Chakrabarti and G. Cormode and A. McGregor, *A Near-Optimal Algorithm for Computing the Entropy of a Stream*, SIAM, 2007
- J. Gama, C.Pinto, *Discretization from data streams: applications to histograms and data mining*. SAC 2006.

Bibliography on Frequent Item's

- *Probabilistic Counting Algorithms for DataBase Applications*, Flajolet and Martin; JCSS, 1983
- *Finding repeated elements*, J. Misra and D. Gries. Science of Computer Programming, 1982.
- *What's Hot and What's Not: Tracking Most Frequent Items Dynamically*, by G. Cormode, S. Muthukrishnan, PODS 2003.
- *Dynamically Maintaining Frequent Items Over A Data Stream*, by C. Jin, W. Qian, C. Sha, J. Yu, A. Zhou; CIKM 2003.
- *Processing Frequent Itemset Discovery Queries by Division and Set Containment Join Operators*, by R. Rantzau, DMKD 2003.
- *Approximate Frequency Counts over Data Streams*, by G. Singh Manku, R. Motawani, VLDB 2002.
- *Finding Hierarchical Heavy Hitters in Data Streams*, by G. Cormode, F. Korn, S. Muthukrishnan, D. Srivastava, VLDB 2003.
- *A Geometric Approach to Monitoring Distributed DataStreams*, I. Sharfman, A. Schuster, D. Keren, SIGMOD 2006