Stream Data Mining

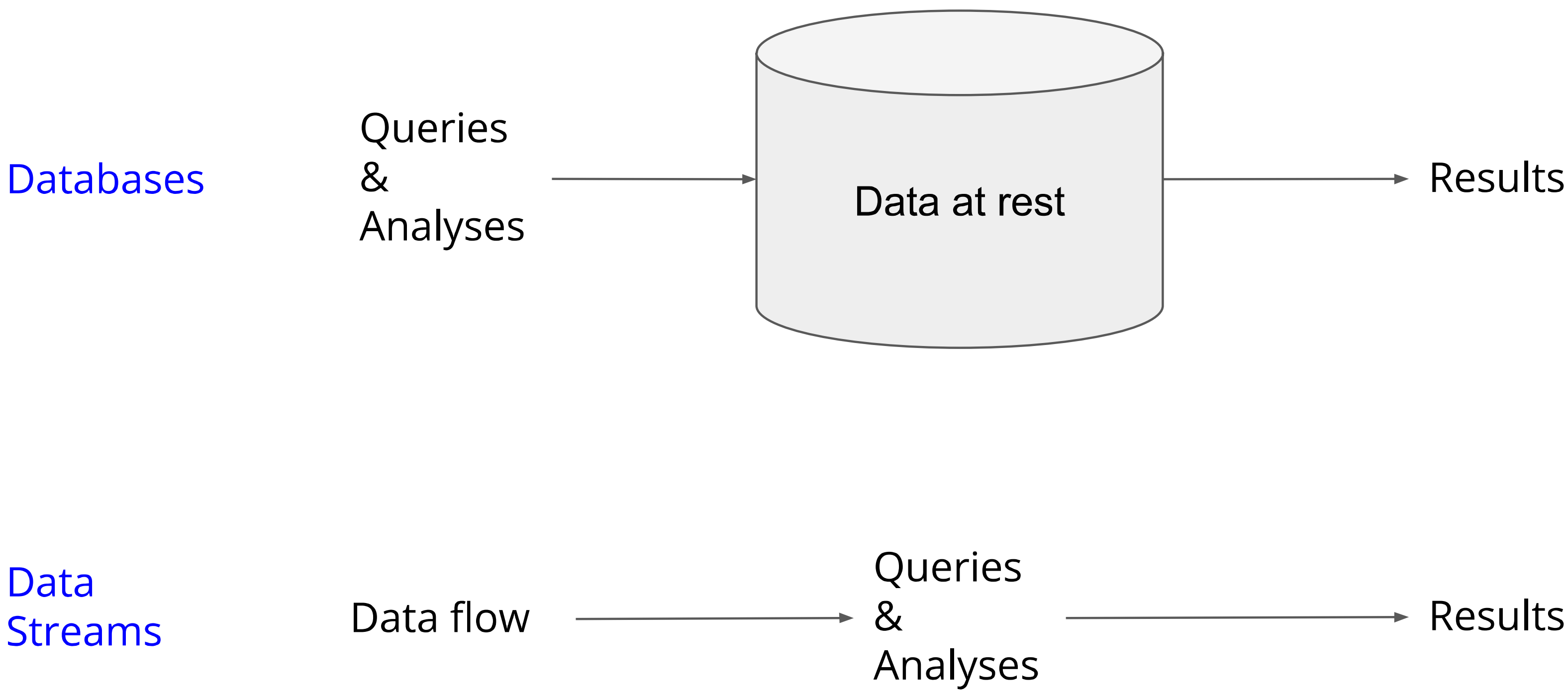
Data Streams

Many modern applications generate huge amounts of fast data streams:

- Credit card transactions
- Wearable sensors
- Connected vehicles
- Industry 4G
- IoT

hpe.com

Data Streams V.S. Databases



Data Stream Challenges

- One pass constraint
  - Data size is the assumed to be infinite. No chance to store all, and do second pass. So what ? Think of k-means
- Concept drift
  - Data evolves over time, and also its statistical properties. So what ? Think of outlier mining
- Resource constraints
  - Variable arrival rate forces that algorithms have to be very efficient
- Massive domain
  - Some data attributes might have large number of distinct values (massive domain), e.g., social network connections. So what ? Think of frequent itemset mining

Tools for your data scientist toolbox

- Have I seen you before ?
- How many time did I see you ?
- How many persons have attended my lecture ?
    - Solution 1, 3

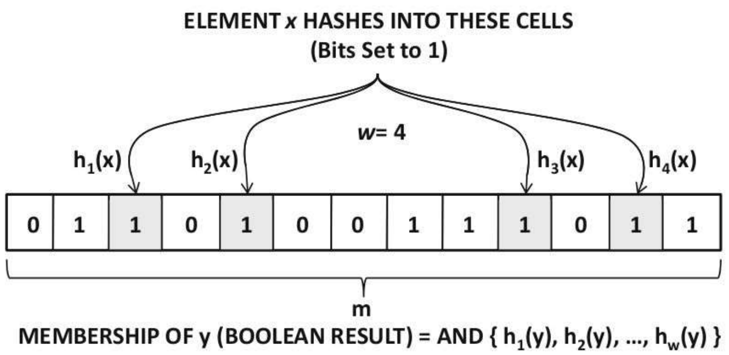
Synopsis structures for massive domain

Bloom Filter

False Positive (incorrectly predicted as positive) is possible i.e. element is not there but it says element is present
False Negative is not possible

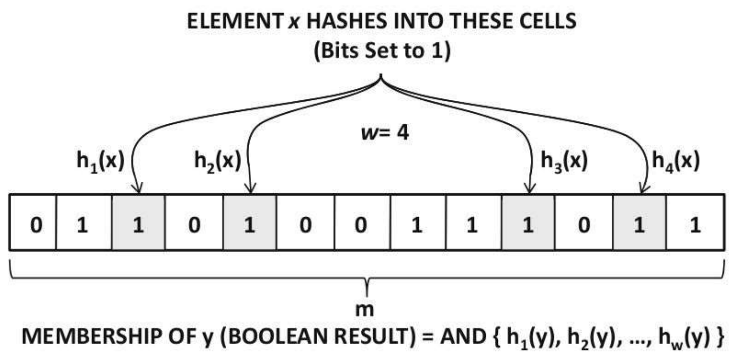
Given a particular element, has it ever occurred in the data stream?

- A Bloom filter is a synopsis that gives an answer of this query with a probabilistic bound on the accuracy.
- If the bloom filter reports that an element does not belong to the stream, then this will always be the case. But there can be false positives.
- A bloom filter consists of:
  - a binary bit array of length m, [0, ..., m-1]
  - w independent hash functions $h_1(.) \ldots h_w(.)$



Bloom Filter

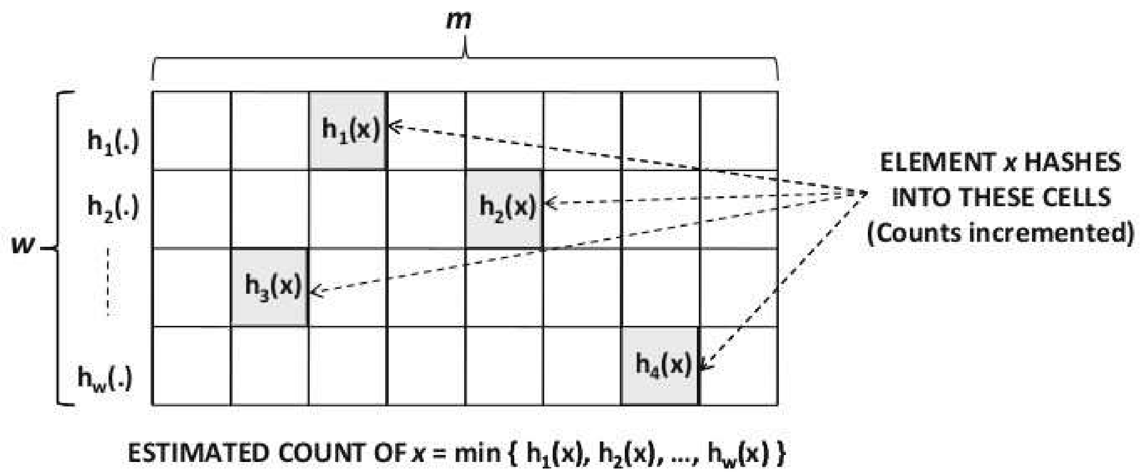
Algorithm *BloomConstruct*(Stream: $\mathcal{S}$, Size: m, Num. Hash Functions: w)
begin
  Initialize all elements in a bit array $\mathcal{B}$ of size m to 0;
  repeat
    Receive next stream element $x \in \mathcal{S}$;
    for $i = 1$ to w **do**
      Update $h_i(x)$th element in bit array $\mathcal{B}$ to 1;
  until end of stream $\mathcal{S}$;
  return $\mathcal{B}$;
end



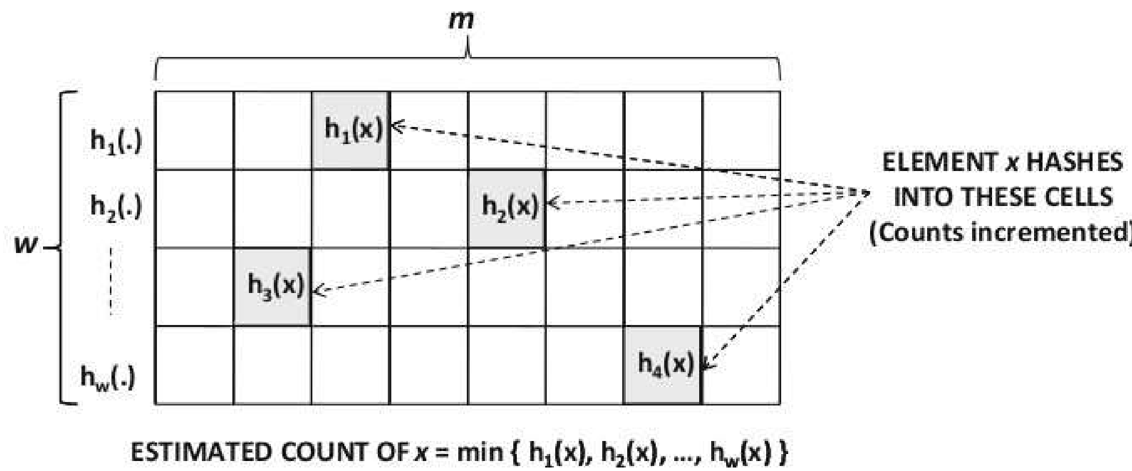
Count-Min Sketch

Given a particular element, how many times did it appear before in the data stream?

- A count-min sketch consists of:
  - a set of *w* different numeric arrays, each of which has a length *m*
  - *w* independent hash functions $h_1(.) \ldots h_w(.)$

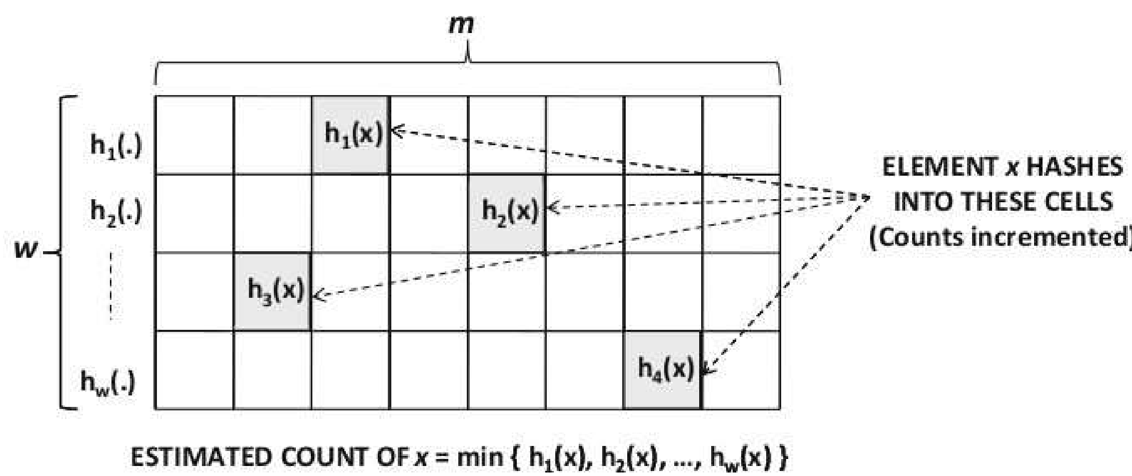


Construction



Algorithm *CountMinConstruct*(Stream: $\mathcal{S}$, Width: w, Height: m)
begin
  Initialize all entries of $w \times m$ array $\mathcal{CM}$ to 0;
  repeat
    Receive next stream element $x \in \mathcal{S}$;
    for $i = 1$ to w **do**
      Increment $(i, h_i(x))$th element in $\mathcal{CM}$ by 1;
  until end of stream $\mathcal{S}$;
  return $\mathcal{CM}$;
end

Querying



Algorithm *CountMinQuery*(Element: y, Count-min Sketch: $\mathcal{CM}$)
begin
  Initialize $Estimate = \infty$;
  for $i = 1$ to w **do**
    $Estimate = \min\{Estimate, V_i(y)\}$;
  { $V_i(y)$ is the count of the $(i, h_i(y))$th element in $\mathcal{CM}$ }
  return $Estimate$;
end

Flajolet–Martin Algorithm

How many persons are attending Justin Bieber concert

- Use a hash function h(·): stream element x -> integer in the range $[0, 2^{L} - 1]$
- Usually, the value L is selected to be 64. We need that 2^{L} > number of distinct elements
- Take the binary representation of h(x) 0000101110101101010000000000
- Look at the number of 0's in the tail 9
- Keep the maximum of the number of 0s in the tail

Flajolet–Martin Algorithm

0000101110101101010000000000

The intuition is that Probability of finding a tail of r zeros:

- Goes to 1 if $n \gg 2^r$
- Goes to 0 if $n \ll 2^r$

It has been proven that the expected maximum number of trailing zeros over all stream elements is

$E\,[R_{max}] = \log_2 (0.77351\, n)$, n is number of distinct elements in the stream

Reversing it:

$n = 2^{R_{max}} / 0.77351$

Flajolet–Martin Algorithm - improvements

000010111010110101000000000

What if the first guy to enter my lecture hits a record of trailing zeros ?

Flajolet-Martin will estimate that my lecture has much more people than Justin Bieber concert

Solution ?

Flajolet–Martin Algorithm - improvements

000010111010110101000000000

What if the first guy to enter my lecture hits a record of trailing zeros ?

Flajolet-Martin will estimate that my lecture has much more people than Justin Bieber concert

Solution ? Multiple hash functions can be used, and the average value of R_{max} over the different hash functions

Multiple Hash Functions:
Instead of applying a single hash function, use multiple independent hash functions.
Each hash function will produce a different result for the same input.

Combine Results:
For each hash function, apply the Flajolet-Martin algorithm independently
to get an estimate of the count of trailing zeros (R) for that hash function.

Average Results:
Compute the average of the R values obtained from different hash functions.
This averaged value can be used as the final estimation.

Flajolet–Martin Algorithm - improvements

0000101110101101010000000000

What if the concert hall has many doors ?

How to synchronize multiple Flajolet-Martin R_{max}

Solution ?

Multiple Streams:
If you have data streams from multiple doors (or multiple sources), apply the Flajolet–Martin algorithm independently to each stream.

Calculate max Rmax for Each Stream:
For each stream, calculate the max Rmax value using the Flajolet–Martin algorithm.

Aggregate Results:
Take the maximum max Rmax value across all streams.
This aggregated max Rmax represents the maximum number of trailing zeros observed across all streams.

Final Estimation:
The final estimation of the cardinality can then be obtained using the aggregated max Rmax value.

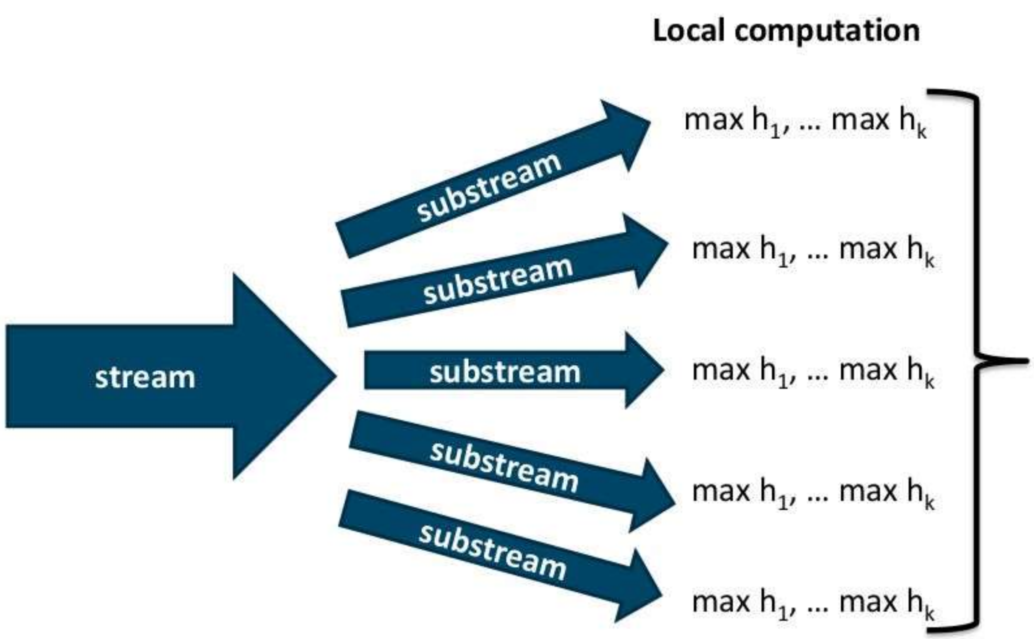
Flajolet–Martin Algorithm - improvements

0000101110101101010000000000

What if the concert hall has many doors ?

How to sum multiple Flajolet-Martin R_{max}

Solution ?



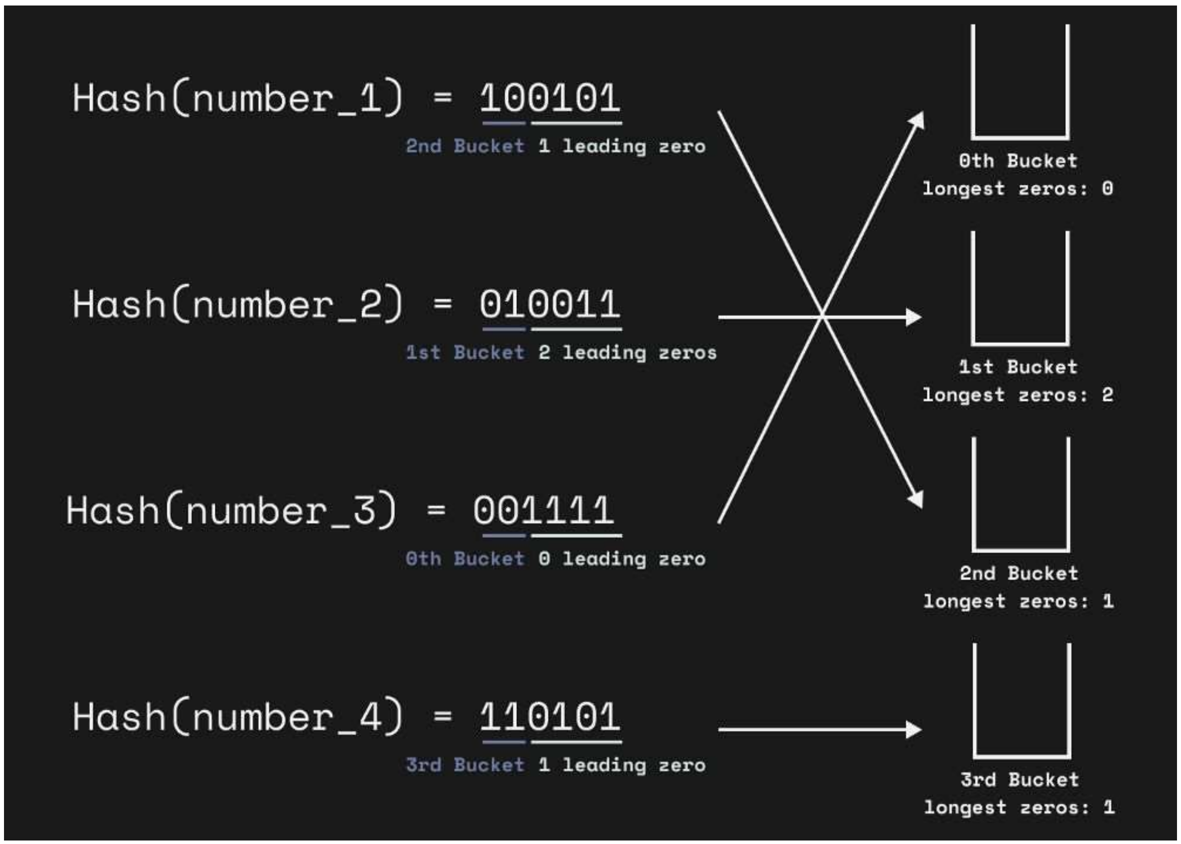
max (max(A), max(B)) = max(AB)

Hyperloglog

Flajolet–Martin[1] -> loglog[2] -> hyperloglog[3]

1. Flajolet, Philippe; Martin, G. Nigel (1985). "Probabilistic counting algorithms for data base applications"
2. Durand, Marianne; Flajolet, Philippe (2003). "Loglog Counting of Large Cardinalities"
3. Flajolet, Philippe; Fusy, Éric; Gandouet, Olivier; Meunier, Frédéric (2007). HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm

Hyperloglog



towardsdatascience.com

Hyperloglog

- Workhorse when it comes to cardinality counting
- Avoids the need for many hash-functions to reduce error
- Standard error 1.04/*m* (*m* is the number of buckets)
- Great demo at:

  http://content.research.neustar.biz/blog/hll.html

Credits Filter - Analysis

These slides, except when explicitly stated, use material from:

- Charu C.Aggarwal. Data Mining The Textbook, Springer.
- Toon Calders, Tools for a data scientist's toolbox: Data Stream Processing and others, eBISS summer school 2017