Data Stream Analysis

Big Data Management

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

DTIM
www.essi.upc.edu/dtim

Knowledge objectives

1. Explain the difference between generic one-pass algorithms and stream processing
2. Name the two challenges of stream processing
3. Name two solutions to limited processing capacity
4. Name three solutions to limited memory capacity

Understanding Objectives

1. Decide the probability of keeping a new element or removing an old one from memory to keep equi-probability on load shedding
2. Decide the parameters of the hash function to get a representative result on load shedding
3. Decide the optimum number of hash functions in a Bloom filter
4. Approximate the probability of false positives in a Bloom filter
5. Calculate the weighted average of an attribute considering an exponentially decaying window
6. Decide if heavy hitters will show false positives

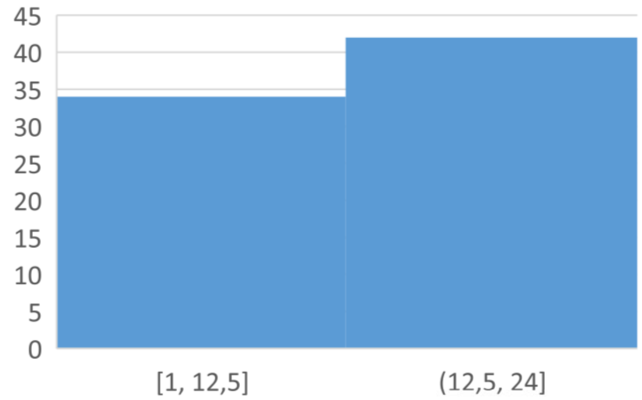
Challenges and approaches

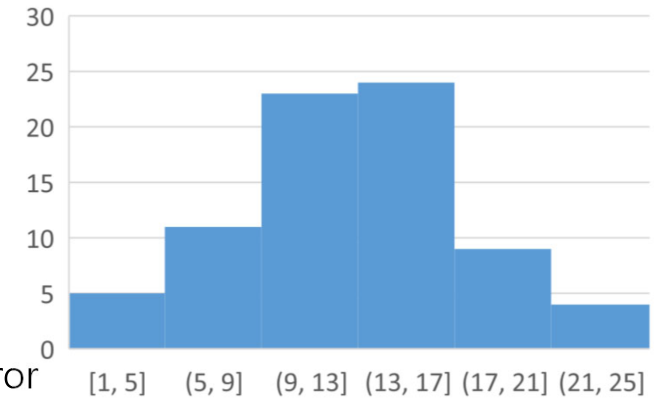
Constraints

- Data cannot be stored
  - One-pass algorithms with
    - Bounded processing time
    - Bounded resources (i.e., memory)
      - At most, logarithmic on the size of the stream
    - Answer available at any time
- Processing must be on-line
  - Bounded response time for both
    a) Summary update
    b) Response retrieval

Challenges and approaches

- Limited computation capacity
  - Sampling (i.e., Load shedding)
    - Probabilistically drop stream elements
  - Filtering (i.e., Bloom filters)
- Limited memory capacity
  - Sliding window -> Discard elements
    - Aging (use only most recent data)
  - Exponentially decaying window -> Weight elements
  - Synopsis -> Approximate solutions
    - Examples:
      - Histograms - Works under uniform distribution of values in a bucket
      - Concise sampling - Works under a limited number of distinct values
      - Heavy hitters - Uses logarithmic memory space
      - Sketching - Space needed depends on error and probability of that error





Load shedding

Sampling data streams

Load shedding (Keeping equi-probability)

- Mistakes in case of infinite streams:
  a) Fix the values at the beginning
  b) Remove old values from memory
- Goal (Uniform Random Sampling):
  - Any subset of elements has the same probability of being in memory at any time
  - Do not want to store any additional information
- Definitions:
  - Memory positions: *p*
  - Elements seen: *n*

- Solution (Reservoir sampling):
  - Probability of keeping the new element *n+1*
    - p/(n+1)

      n= number of elements seen so far
      n+1 = current element position
  - Probability of removing an element from memory
    - 1/p

Load shedding (Statement)

"Select a subset of the stream so that answering ad-hoc queries gives a statistically representative result."

Example: *Given a stream of tuples [user, query, time], we can store 10% of the tuples. If we randomly keep **1/10 of the tuples**, then we would get the wrong answer to "Percentage of duplicate queries for a user"!!!*

how many times user passes same query in search engine

Definitions:

- s = *#queries issued once by any user*
- d = *#queries issued twice by any user*
- *No queries issued more than twice*

The sample will contain:

- s/10+18d/100 *queries issued once*
- d/100 *queries issued twice*

The answer would be:

- *(d/100)/(*s*10/100+d*18/100+d/100*)* = d/(10s+19d) ≠ d/(s+d)

Solution:

- *Keep* 1/10 *of the users*

Before/After	Twice	Once	None
Once	0	s*1/10	s*9/10
Twice	d*1/100	d*(9/10*1/10+1/10*9/10)	d*9/10*9/10
Total	d*1/100	s*10/100+d*18/100	…

How many words can appear after sampling once if they appeared once before sampling?

A. Rajaraman and J. Ullman

Load shedding (Generalization)

- Queries may need different grouping keys or the key can be compound
  - Use the “group by” set in the hash function
- Memory is limited
  - Take a hash function to a large number of values M and keep only elements mapping to a value bellow *t*
    - Dynamically reduce *t* as you are running out of memory

h(GB) = f(GB) mod M < 4

more threshold- remove

U2W1

Bloom Filters

Filtering data streams

Adding number of hash function reduces number of false positives up to some extent

Bloom filters (Statement)

"Accept those elements in the stream that meet a criterion (based on looking for membership in a set), others are dropped."

- *Example*
  - *Given an e-mail stream of tuples [address,text], we have a list of 10^9 allowed addresses (20 bytes each) and only 1GB of memory available.*
- *Solution*
  - *Use the memory as an array of bits and map the addresses by means of a hash function (h: address -> bit position)*
- *Note: Some spam will get through the filter*

A. Rajaraman and J. Ullman

Bloom filters (Example with one hash function)

Key values = {IP_1, IP_2}
Hash function = {h}
Array of bits → 0 0 1 0 0 0 1 0 0 0
Build
  $h(IP_1) = 3$
  $h(IP_2) = 7$
Probe
  IP_3 → $h(IP_3) = 5$
  IP_4 → $h(IP_4) = 3$ **FALSE POSITIVE!**

Bloom filters (Example with two hash functions)

Key values = {IP_1, IP_2}
Hash functions = {h_1, h_2}
Array of bits ➡ 0 0 1 0 1 0 1 0 0 0
Build 0 0 0 0 0 0 0 0 0 0

$h_1(IP_1) = 3$ $h_2(IP_1) = 5$
$h_1(IP_2) = 7$ $h_2(IP_2) = 5$

Probe

IP_3 ➡ $h_1(IP_3) = 5$ $h_2(IP_3) = 9$

IP_4 ➡ $h_1(IP_4) = 3$ $h_2(IP_4) = 7$ **FALSE POSITIVE!**

Bloom filters (Generalization)

- Elements:
  - A set of *m* key values
  - A list of *k* hash functions (h_i: key → *n*)
  - One array of *n* bits (*n*>>*m*)
- Build:
  - For each element in the probing set, apply all *k* hash functions and set to 1 the corresponding bits
- Probing:
  - For each element in the stream, apply all *k* hash functions, it will pass only if all corresponding bits are set to 1
- False positives:
  - $(1-e^{-km/n})^k$
- Optimal
  - $k = (n/m)\cdot\ln 2 \rightarrow (1-e^{-km/n})^k = (1/2)^k \approx 0.618^{n/m}$

Bloom filters (Rationale)

- Probability of a bit being set by a hash function at build phase
  $1/n$
- Probability of a bit NOT being set by a hash function at build phase
  $1-1/n$
- Probability of a bit NOT being set by a hash function of ANY key at build phase
  $(1-1/n)\cdot(1-1/n)\cdot \ldots \cdot(1-1/n) = (1-1/n)^m = (1-1/n)^{n(m/n)}$
  - A good approximation of $(1-\epsilon)^{1/\epsilon}$ for small ϵ is $1/e$
    $(1/e)^{m/n} = (e^{-1})^{m/n} = e^{-m/n}$
- Probability of a bit NOT being set by ANY hash function of ANY key at build phase
  $(e^{-m/n})^k$
- Probability of a bit set by SOME hash function of ANY key at build phase
  $1-(e^{-m/n})^k = 1-e^{-km/n}$
- Probability of all hash functions finding the bit set in the probing phase
  $(1-e^{-km/n})^k$

Exponentially decaying window

Exponentially decaying window (Statement)

"Do not make a distinction between old and young element, but just weight them."

- *Example*
  - *Find the* ***currently*** *most popular movie/topic.*
- *Solution:*
  - *Keep one weighted* ***counter per movie/topic***
- *Definitions:*
  - *c = small constant (e.g., 10^{-6} or 10^{-9})*
  - *T = current time*
  - *f(t) = a_t= element at time t (or 0 if there is no element)*
  - *g(T-t) = $(1\text{-}c)^{(T\text{-}t)}$ = weight at time T of an item obtained at time t*
- *Value: $\Sigma f(i)\cdot g(T\text{-}i)$ = $\sum_{i=0}^{T} a_i (1-c)^{T-i}$*
- *Process: Multiply the current counter by* (1-c) *and add a_t*
  - Counter(T+1)=$\sum_{i=0}^{T+1} a_i (1-c)^{T+1-i}$ = $\left(\sum_{i=0}^{T} a_i (1-c)^{T-i}\right) * (1-c) + a_{T+1} = Counter(T) * (1-c) + a_{T+1}$
- *Optimizations*

  a) *Being X the time since the last update, we can multiply by $(1\text{-}c)^X$*, instead

  b) We might define a threshold to remove from memory the elements when the counter is too small

Exponentially decaying window (Example)

c=0.5

Counter = 0.28125

Stream

0 1 0 0 1 0 0 …

Heavy Hitters
(or Frequent Items)

Heavy hitters (Statement)

“Given a stream, identify the items that occur more than a given percentage (θ) of times.”

- Example:
  - Find the most frequent destinations in a router
  - Find the most frequent queries in a search engine
- Problem:
  - We do not know which will be frequent enough
  - We cannot store all items
    - An exact solution needs to store all items seen
      - O(n log(N)) in the worst case
- Solution – Approximate with <u>false positives</u>
  - Structure:
    - Set of 1/θ pairs [element, counter]
  - Actions on receiving an element:
    - If the element is in the structure, increase its counter
    - If the element is not in the structure, insert it with counter 1
    - If the set overflows, decrease all counters and remove those with value zero

Heavy hitters (example)

Required frequency: 33%

Heavy hitters: a b c

a b a b a c c c a a b d e f g h

Summary (capacity: 1/0.33 = 3)

[a,4]

[b,3]

[b,2]

[e,1]

a:	5/16	=	31.25%	
b:	3/16	=	18.75%	
c:	3/16	=	18.75%	
d:	1/16	=	6.25%	
e:	1/16	=	6.25%	
f:	1/16	=	6.25%	
g:	1/16	=	6.25%	
h:	1/16	=	6.25%	

Closing

Summary

- Stream processing techniques
  - Load shedding
  - Bloom filters
  - Exponentially decaying window
  - Heavy hittters

References

- J. Leskovec, A. Rajaraman and J. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011
  - http://www.mmds.org
- C. C. Aggrawal editor. *Data Streams, models and algorithms*. Springer, 2007
- I. Botan et al. *Flexible and Scalable Storage Management for Data-intensie Stream Processing*. EDBT, 2009
- R. Karp et al. *A simple algorithm for finding frequent elements in streams and bags*. ACM Transactions on Database Systems 28 (1), 2003
- A. Z. Broder and M. Mitzenmacher: *Network Applications of Bloom Filters: A Survey*. Internet Mathematics 1(4), 2003