Data stream statistics

Filippo Geraci

A set of data analytics problems

1. How many different users accessed the server this week?
2. Was john among them?
3. How many times john accessed the server?
4. What is the usage trend on the server?
5. Who are the most active users on this server?

Cardinality Estimation

- Easy when I want to count all
- Counting the distinct elements of a stream:
  - Sort data and find unique keys
  - Use hash tables

- Sorting takes O(n log n) time
- Both require O(n) space

Cardinality Estimation: Linear Counting

```
class LinearCounter {
    BitSet mask = new BitSet(m) // m is a design parameter

    void add(value) {
        int position = hash(value) // map the value to the range 0..m
        mask.set(position) // sets a bit in the mask to 1
    }
}
```

$$Load\ factor = \frac{\#\ distinct\ elements}{m}$$

- LF << 1 → low probability of collision
- LF ~ 1 → Estimation can be corrected based on the probability of collision
- LF >> 1 → Estimation cannot be corrected

Cardinality Estimation: Linear Counting

```
class LinearCounter {
    BitSet mask = new BitSet(m) // m is a design parameter

    void add(value) {
        int position = hash(value) // map the value to the range 0..m
        mask.set(position) // sets a bit in the mask to 1
    }
}
```

- Estimation of c’ can be adjusted according to the the number m of bits and the number c of bits set to 1

$$c' = -m \ ln\frac{m-c}{m}$$

Cardinality Estimation: Linear Counting

Load factor	Real count	Estimated count	Relative Error
10%	1	1.050	5%
20%	2	2.230	11.5%
30%	3	3.560	18.6%
40%	4	5.100	27.5%
50%	5	6.930	38.6%
60%	6	9.160	52.6%
70%	7	12.030	71.8%
80%	8	16.090	101.1%
90%	9	23.020	155.7%

Linear counting with m=10 and no collisions

How big should be the bit vector?

Number of elements in the stream	Size for an error rate of 1%
100	5034
1000	5329
7000	7132
8000	7412
10000	7960
100000	26729
1000000	154171
10000000	1096582
100000000	8571013

- http://dblab.kaist.ac.kr/Publication/pdf/ACM90_TODS_v15n2.pdf

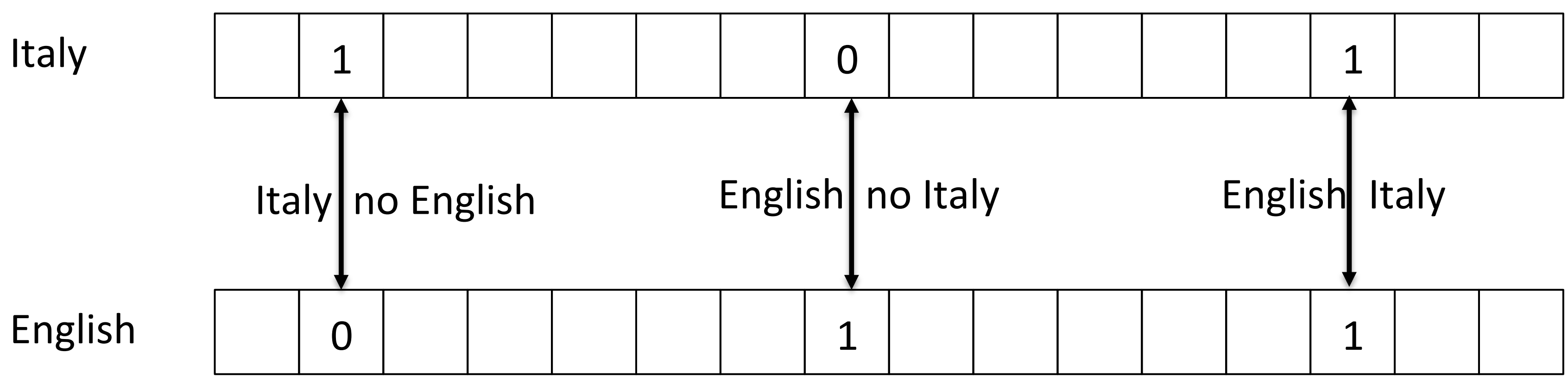
Cardinality Estimation: Linear Counting – Complex queries

- Case study: I have tweets tagged with country and language
  - Question: how many tweets from Italy are in English?

Cardinality Estimation: Linear Counting – Complex queries

- Case study: I have tweets tagged with country and language
  - Question: how many tweets from Italy are in English?

- I can keep two counters one for country and one for language
  - Answer: OR of the two counters

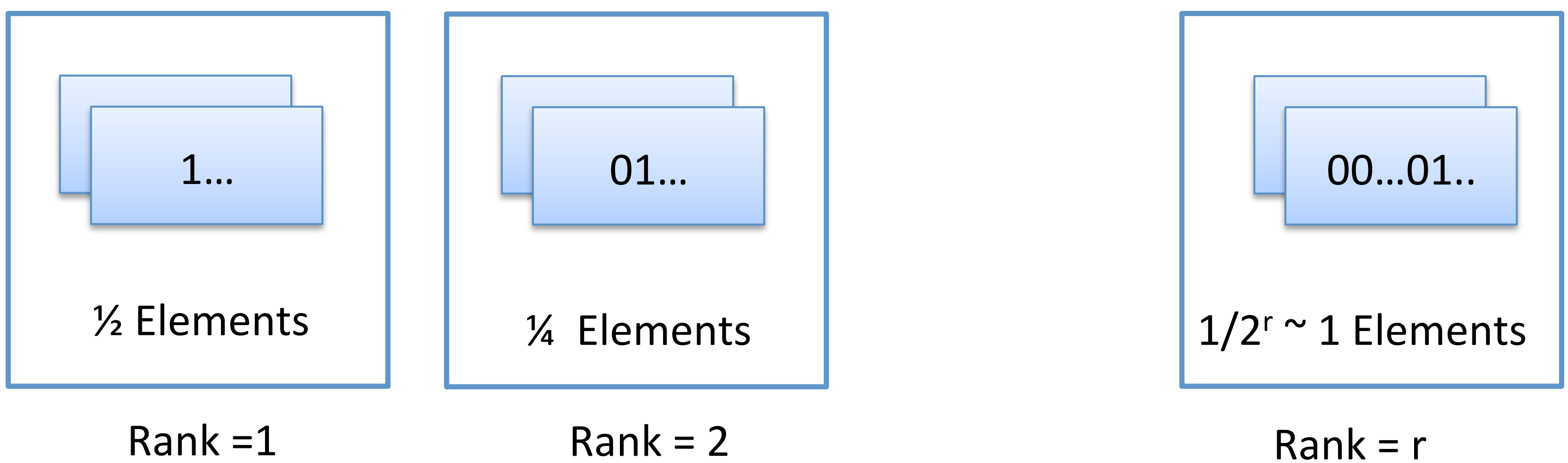


Cardinality Estimation: Linear Counting

- Keeping the error rate under control still requires linear growth of memory with the dataset

Loglog counters

- Assuming each element is hashed as a H bit vector
  - Let ρ(y) be the rank (i.e. the position of the leftmost bit set to 1) of the hash of the element y



Loglog counters

- Given a hash function where the bits are uniformly distributed we can estimate that:

$$|X = \{y' \colon \rho(y') = r\}| = \frac{1}{2^r} \cong 1$$

Imply

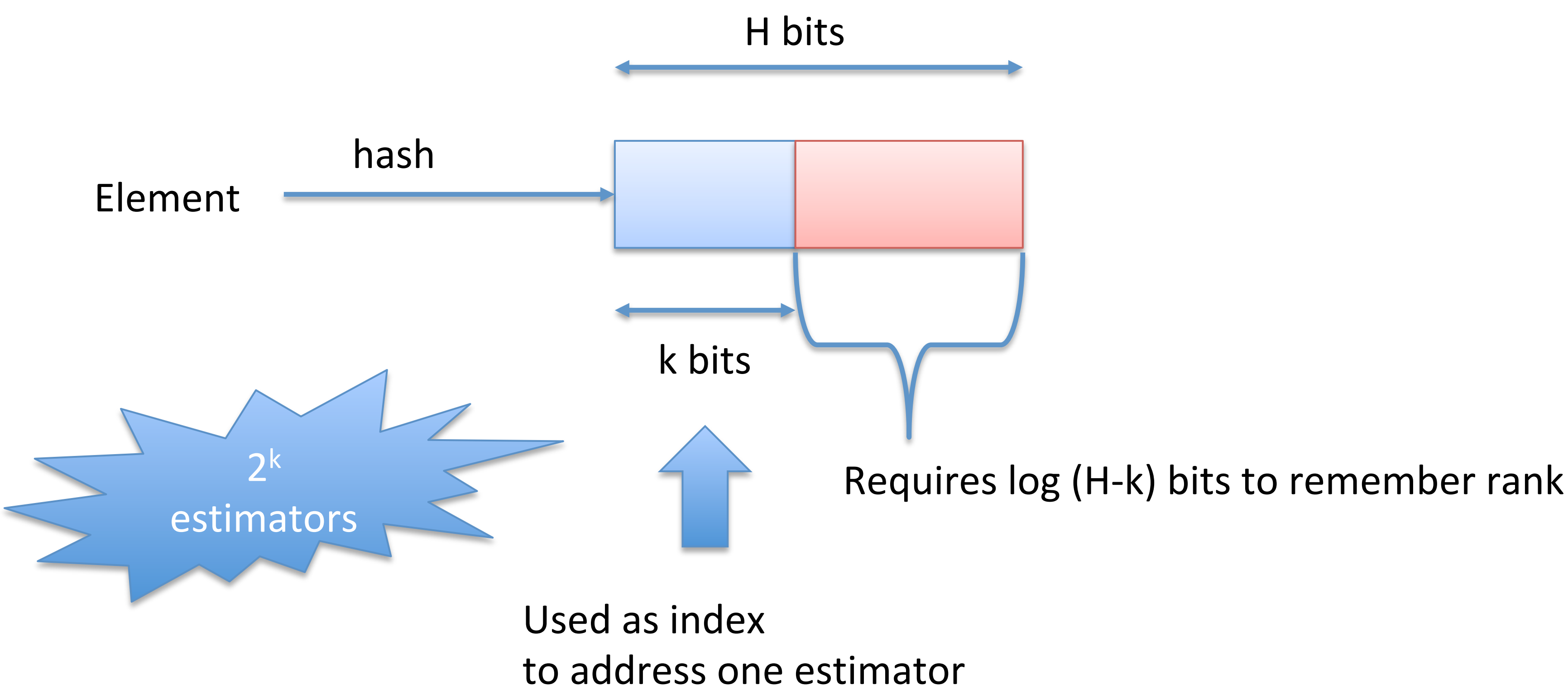
$$\max \rho\ (y) = log_2 n$$

thus

$$n = 2^{\max \rho\ (y)}$$

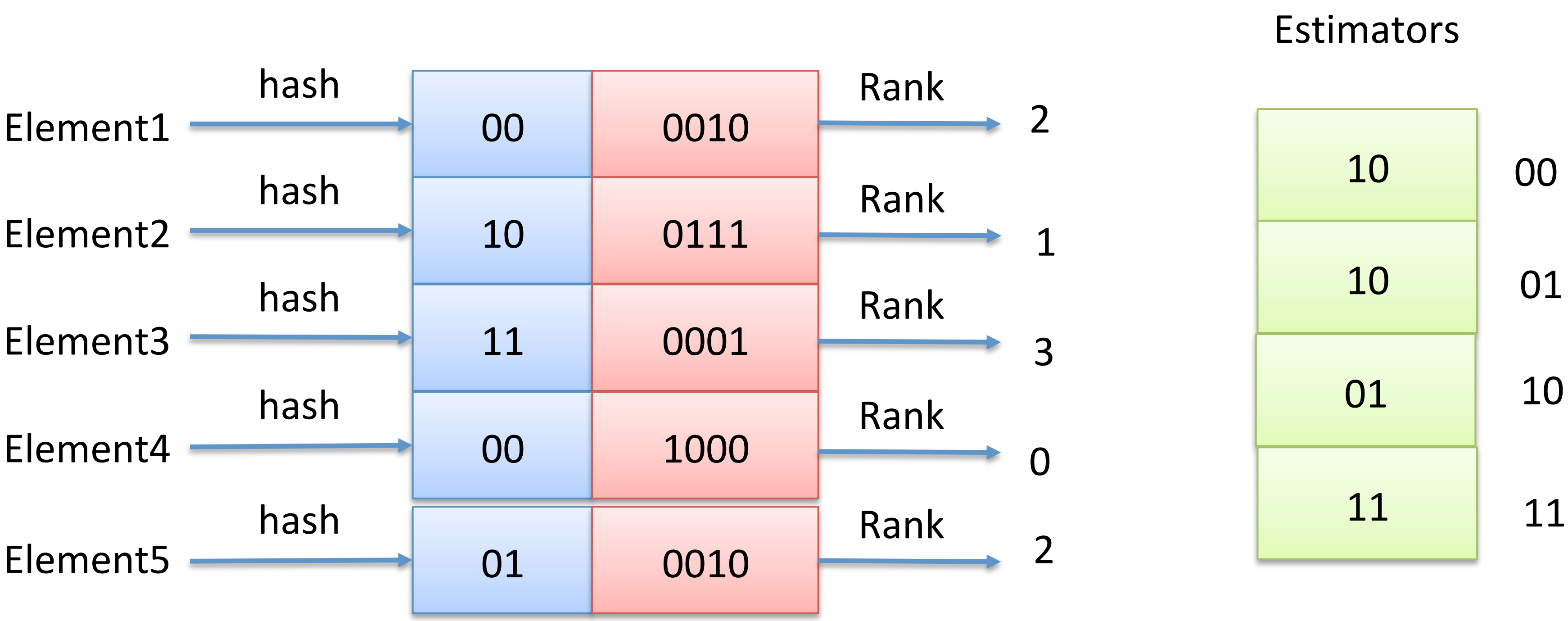
Loglog counters

- Using one estimator would be unstable



Loglog counters

- Example H=6, k=2 (2^2=4 estimators each of which with of 2 bits = 8 bits overall)



Loglog counters

```
class LogLogCounter {
   int H            // H is a design parameter
   int m = 2^k          // k is a design parameter
   etype[] estimators = new etype[m] // etype is a design parameter

   void add(value) {
       hashedValue = hash(value)
       bucket = getBits(hashedValue, 0, k)
       estimators[bucket] =
           max (estimators[bucket], rank( getBits(hashedValue, k, H) );
       )
   }
   int count (void) {
       int sum = 0;
       for (i=0; i < m; i ++) sum += estimators[i];
       return m * 2 ^ (1/m * sum);
   }
}
```

Loglog counters - performance

- Given m=256 (k=8) H=16 -> max rank () stored in 4 bits
  - The data structure is 256 * 4bit = 128 bytes
  - Count the number of distinct words in Shakespeare's writings with an error rate of 9.4%
  - 30,897 instead of 28,239
- The HyperLogLog algorithm can count > 10^9 elements using 1.5kB of memory with error rate less than 2%

Loglog counters – in practice

```
import hyperloglog, random

h = hyperloglog.HyperLogLog (0.005)
print "k = ", h.p
s = set ()
diff = maxdiff = 0
tot_elems = 10000
while (len (s) < tot_elems):
    x = random.randint (0, 10 * tot_elems)
    h.add (x)
    s.add (x)
    if abs (len (h) - len (s)) != diff:
        diff = abs (len (h) - len (s))
        print len (h), len (s), diff
        if abs (diff) > maxdiff:
            maxdiff = diff
print len (h), len (s), diff
print maxdiff
print float (maxdiff) / tot_elems
```

Resources

- Python Imlementation of Loglogcounters
  – https://github.com/svpcom/hyperloglog
- Original work:
  – http://algo.inria.fr/flajolet/Publications/DuFl03-LNCS.pdf
- Several references can be found in the Wikipedia article
  – https://en.wikipedia.org/wiki/HyperLogLog

A step further

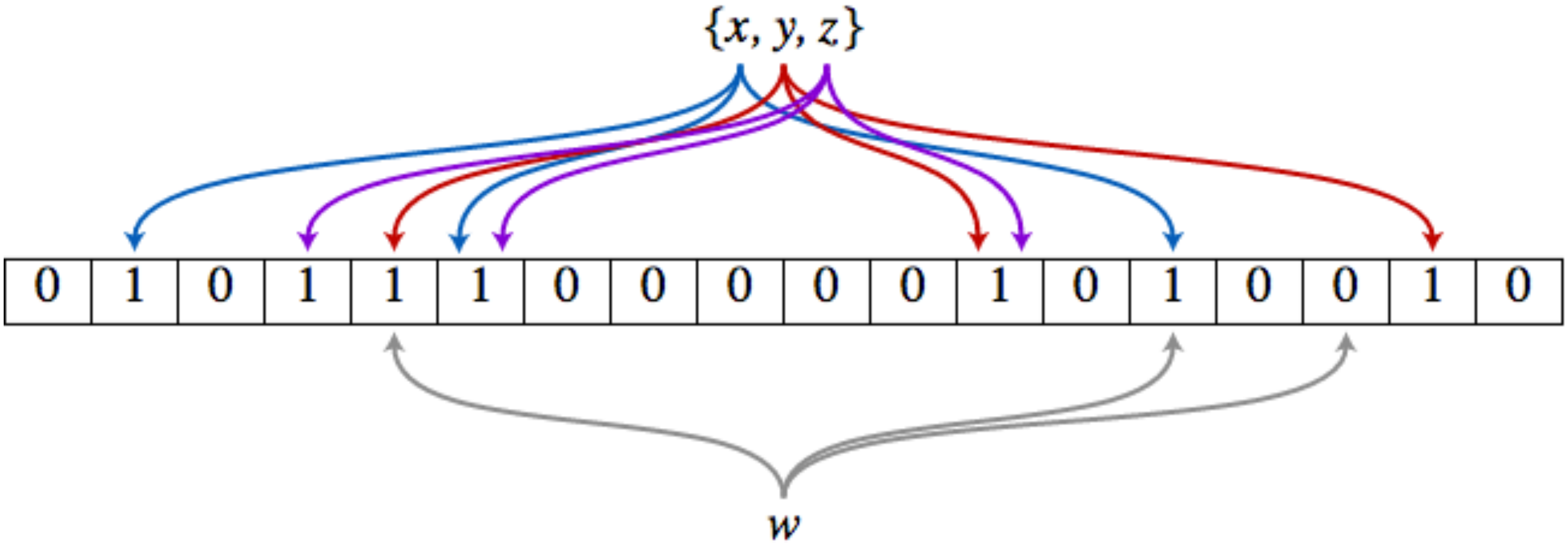
- Now I know how many different elements in a multiset.
- I want to know if an element belongs to the set

A step further

- Now I know how many different elements in a multiset.
- I want to know if an element belongs to the set

- Bloom filters answer:
  - I strongly think the element is in set
  - Definitely not in set

Bloom filters



- Bloom filters reduce to linear counting when using only one hash function.

Improving Bloom filters

- Bloom filters are somehow similar to linear counters but cannot answer to the cardinality query
  - The number of bits set for each insertion is not proportional to the number of hash functions
    - Case 1: collisions with other elements
    - Case 2: collisions among the hash functions
- A possible solution: partition the bloom filter so no case 2 collisions are possible

Bloom filters' extensions

- Deletable Bloom filters: enable probabilistic removal of elements without false negatives and with minimal additional memory
  - C. E. Rothenberg, C. A. B. Macapuna, F. L. Verdi, and M. Magalhaes, "The deletable Bloom filter: a new member of the Bloom family," IEEE Communications Letters, vol. 14, no. 6, pp. 557–559, June 2010. [Online]. Available: http://arxiv.org/abs/1005.0352

Bloom filters' extensions

- Dynamic Bloom filters (DBF) allow to extend bloom filters when the load factor exceeds a given threshold
  - D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo, "The dynamic Bloom filters," IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 1, pp. 120–133, 2010.

Bloom filters: a practical example of use

- Consider you have a proxy that caches web pages. You may want not to cache a page that will be visited only once

Bloom filters: a practical example of use

- Consider you have a proxy that caches web pages. You may want not to cache a page that will be visited only once
  - Solution: use a bloom filter. Once you have a request first check whether it has already be seen. If YES cache the page, otherwise NO. ANYWAY add the page to the Bloom filter.

The bloom filter version of the spell checker

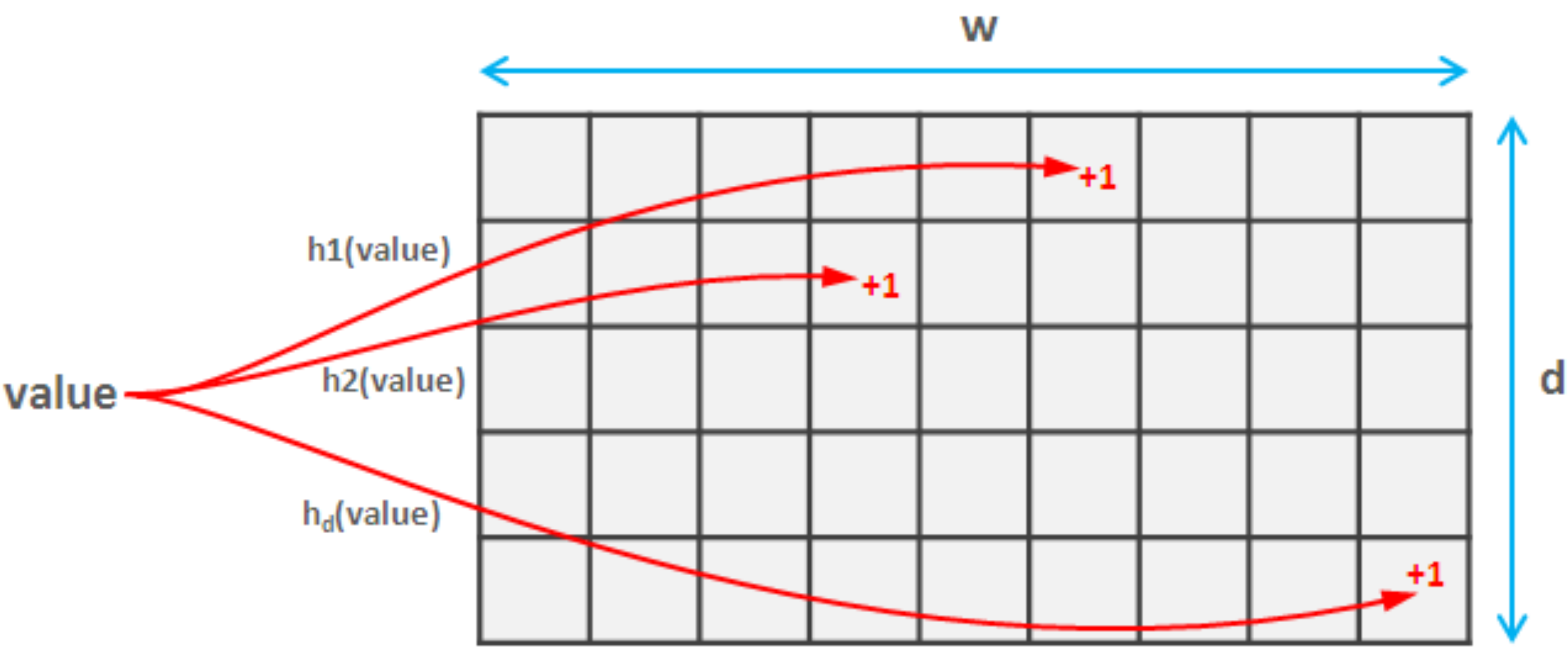
```
from pybloom import BloomFilter
import sys

bf = BloomFilter(capacity=466544, error_rate=0.01)

f = open ("english.txt")
for line in f:
    line = line[:-1]
    bf.add (line)
f.close ()

f = open (sys.argv[1])
for line in f:
    line = line[:-1]
    line = line.split (" ")
    for elem in line:
        if elem in bf:
            print elem, "True"
        else:
            print elem, "False"
f.close ()
```

Number of words in the dictionary

Practical usage

- A python implementation:
  - https://github.com/jaybaird/python-bloomfilter
- Two parameters:
  - Capacity (i.e. expected number of elements to insert)
  - Error rate [0, 1]
- Compare speed versus space of bloom filters and hash sets

Estimation of the number of occurrences: the count mean sketch



A simple python-ish implementation

```
class CountMinSketch:
  int CMS[d][w], a[d], b[d];
  int p = (2^31) - 1; # a convenient prime number

  def initializeHashes () {
    for i in range (d):
      a[i] = random (p) # random in range 1..p
      b[i] = random (p)

  def hash (value, i):
    return ((a[i] * value + b[i]) mod p) mod w

  def add (value):
    for i in range (d):
      CMS[i][ hash (value, i) ] += 1

  def estimateFrequency (value):
    return min ([CMS[i][ hash(value, i) for i in range (d)])
```

What about computing distributions?

- Given highly skewed data I want to measure the frequency at least of the top elements
- Facts:
  - Counters are expected to be higher because of the contribution of other elements
  - CM returns the counter with less noise
- Idea
  - Estimate the contribution of noise for a specific counter

CMM – Count Mean-Min sketch

```
class CountMeanMinSketch {
    // initialization and addition procedures as in CountMinSketch
    // n is total number of added elements

    long estimateFrequency(value) {
        long e[] = new long[d]
        for(i = 0; i < d; i++) {
            sketchCounter = estimators[i][ hash(value, i) ]
            noiseEstimation = (n - sketchCounter) / (w - 1)
            e[i] = sketchCounter – noiseEstimator
        }
        return median(e)
    }
}
```

- Noise as the average value of the other elements of the row

Heavy hitters

- All the above data structures allow counting or membership evaluation.
- How to know the most represented keys in a stream?
- Until now:
  - I can count how many keys exist,
  - I can check if a particular key is present
  - I can count the number of its occurrences
  - …but I can't do anything if I don't know it

Bad news

- Naïve solution:
  - Sort data

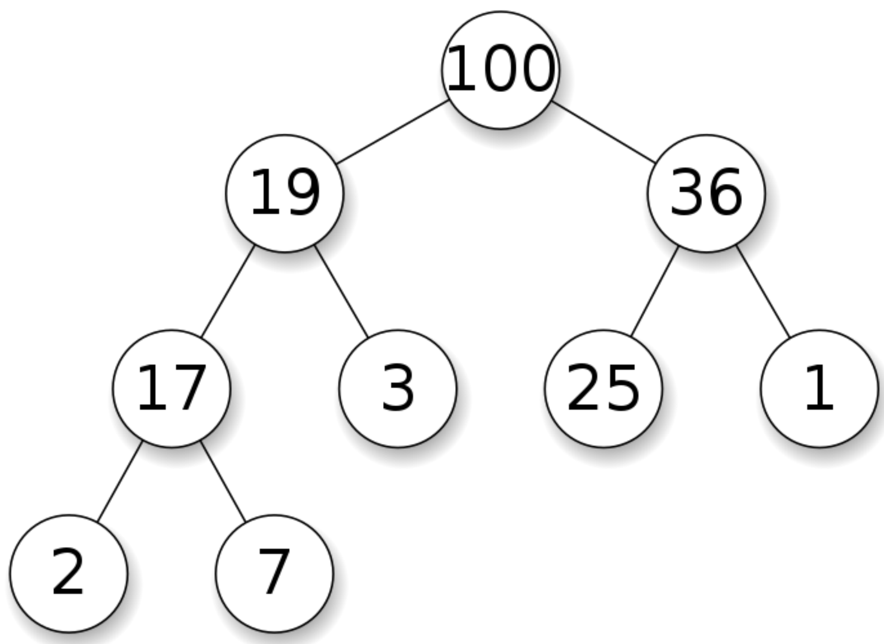
- There is no algorithm that solves the Heavy Hitters problem in one pass while using a sublinear amount of auxiliary space

A simple algorithm

- Problem: find the elements that occur more than N/k times (N is the stream length, k is a free parameter)
- Solution:
  - Maintain a CM and a max-heap (with k elements) of the top elements
- Process:
  1. Add the element in the CM and estimate its frequency
  2. If frequency >= N/k insert the element in the heap
  3. Note: the number of elements in the heap must be at most k

max-heap review

- A tree where the parent node is higher than the descendent nodes



https://en.wikipedia.org/wiki/Heap_(data_structure)#/media/File:Max-Heap.svg

The Space saving algorithm - build

```
Algorithm: Space-Saving(m counters, stream S)
begin
  for each element, e, in S{
    If e is monitored,
      increment the counter of e;
    else{
      let e_m be the element with least hits, min
      Replace e_m with e;
      Increment count_m;
      Assign ε_m the value min;
    }
  }// end for
end;
```

The Space saving algorithm - query

Algorithm: QueryFrequent(m counters, support ϕ)

```
begin
  Bool guaranteed = true;
  Integer i = 1;
  while (count_i > φN AND i ≤ m){
    output e_i;
    If ((count_i − ε_i) < φN)
      guaranteed = false;
    i++;
  }// end while
  return( guaranteed )
end;
```

Note that counts are sorted in descending order in this implementation

References

- New Estimation Algorithms for Streaming Data: Count-min Can Do More
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.449&rep=rep1&type=pdf
- Efficient Computation of Frequent and Top-k Elements in Data Streams
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.8360&rep=rep1&type=pdf
- PROBABILISTIC DATA STRUCTURES FOR WEB ANALYTICS AND DATA MINING
  - https://highlyscalable.wordpress.com/2012/05/01/probabilistic-structures-web-analytics-data-mining/

References

- Guo, Deke, et al. "The dynamic bloom filters." *IEEE Transactions on Knowledge and Data Engineering* 22.1 (2010): 120-133.

Datasets

- Free Twitter datasets
  - http://followthehashtag.com/datasets/
- Stackexchange Q&A website
  - https://archive.org/download/stackexchange