中国科学院大学
University of Chinese Academy of Sciences

Mining Massive Datasets
Mining Data Stream

Outline

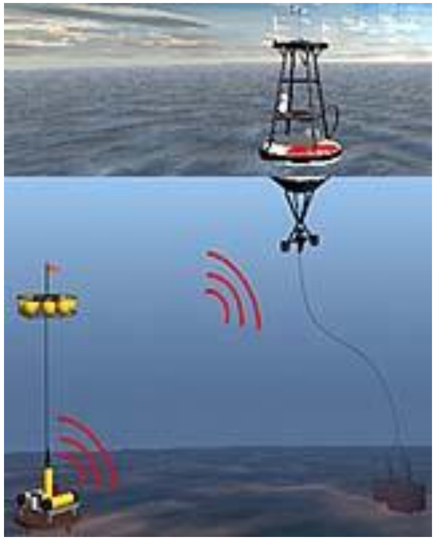
1.1 Intro of Data Stream?

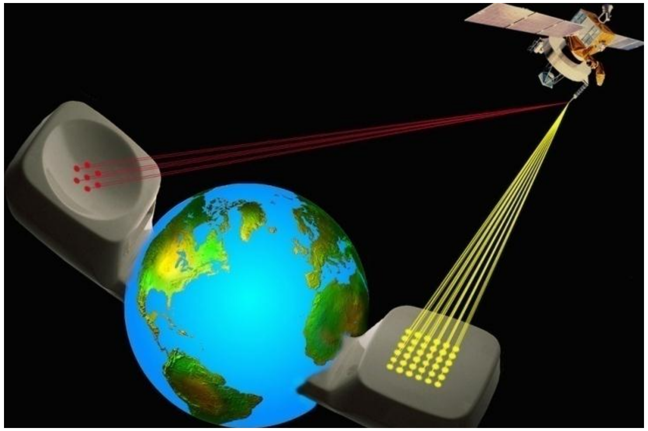
1.2 Sampling from a Data Stream

1.3 Queries over a Sliding Window

Data Stream

- where dose it come from?
  - sensor data: hospital, ocean, war
  - image data: satellites, surveillance cameras
  - web site: Google, twitter





Application

- Mining click streams

  Google wants to know what queries are more frequent today than yesterday

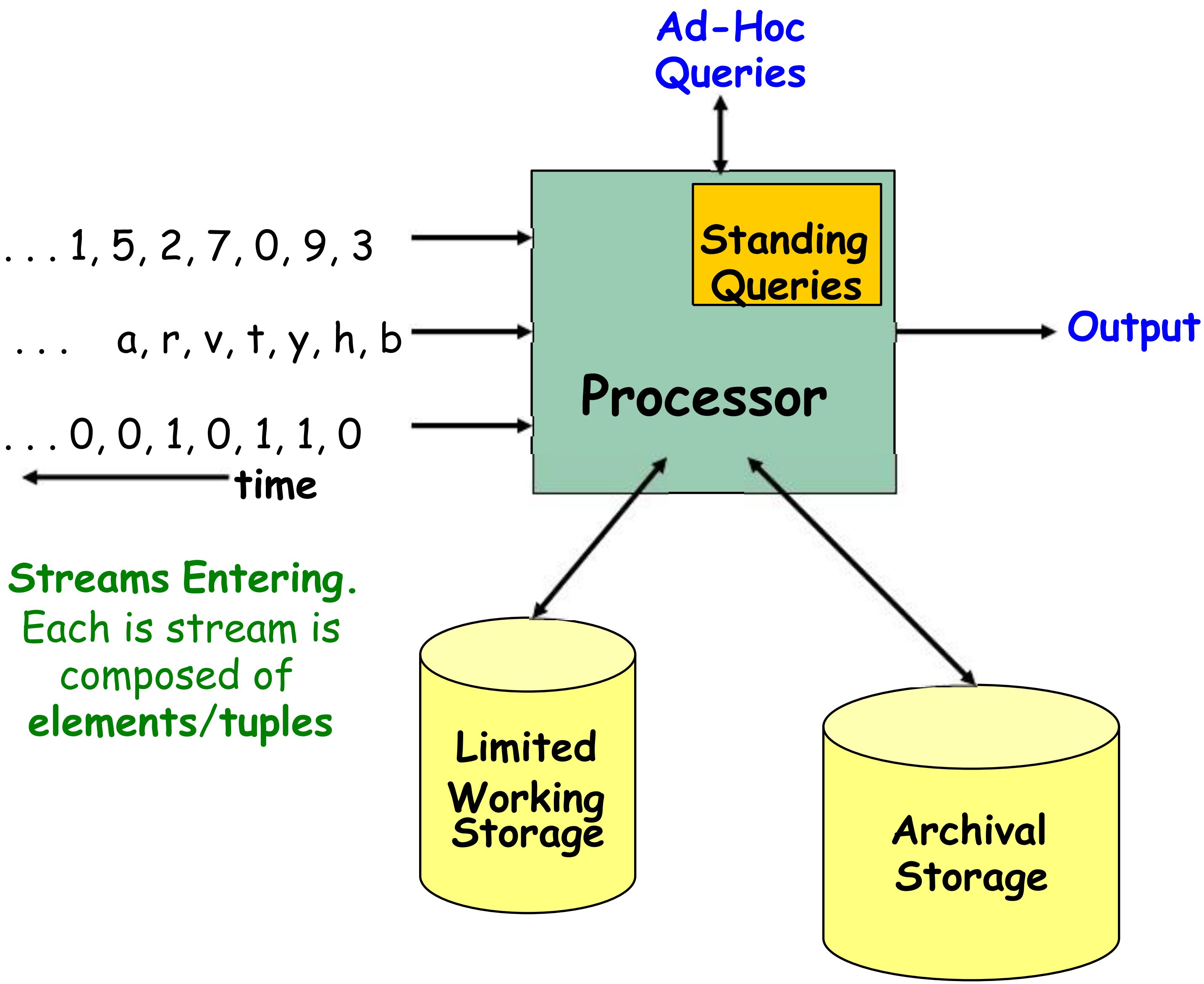
- Mining query streams

  Yahoo wants to know which of its pages are getting an unusual number of hits in the past hour

- Mining social network news feeds

  look for trending topics on Twitter, Facebook

General Stream Processing Model



Data Streams

- feature: infinite (non-stop), non-stationary (the distribution changes over time)
- before: database—all data is available when and if we want
- now: data arrives so rapidly that we can't store it all if it's not processed immediately or stored, then it's lost forever

$\Downarrow$

summarization

Data Streams cont'd

- summarization
  - sample
  - fixed-length window

Outline

1.1 Intro of Data Stream?

<u>1.2 Sampling from a Data Stream</u>

1.3 Queries over a Sliding Window

Sampling from a Data Stream

- Two different problems:

(1) Sample a fixed proportion of elements in the stream (say 1 in 10)

(2) Maintain a random sample of fixed size over a potentially infinite stream

Sampling a Fixed Proportion

- Scenario: Search engine query stream
  - Stream of tuples: (user, query, time)
  - questions: How often did a user run the same query in a single days
- Naïve solution:
  - Generate a random integer in [0..9] for each query
  - Store the query if the integer is 0, otherwise discard
  - fixed proportion: 1/10

Problem with Naïve Approach

- question: What fraction of queries by an average search engine user are duplicates?
- suppose: each user issues x queries once and d queries twice (total of $x+2d$ queries), no search queries more than twice
- Correct answer: $\frac{d}{x+d}$
- sample-based answer: $\frac{d}{10x+19d}$

Problem with Naïve Approach cont'd

- of the x queries issued once
  - x/10 of the search queries appear once

- of the d queries issued twice
  - d/100 appear twice ← $C_2^2 \frac{1}{10} \times \frac{1}{10} \times d$
  - 18d/100 appear once ← $C_2^1 \frac{1}{10} \times \frac{9}{10} \times d$

$$\frac{d/100}{(d/100)+(x/10)+(18d/100)} = \frac{d}{10x+19d}$$

$$\neq \frac{d}{x+d}$$

Maintaining a fixed-size sample

- we need to maintain a random sample *S* of size exactly *s* (e.g., main memory size constraint)
- suppose at time *t* we have seen *n* items, Each item is in the sample *S* with equal prob. *s/n*
- *for example: s=2,*stream: a x c y z k c d e g…

- At t= 5, each of the fist 5 tuples is included in the sample S with equal prob = 2/5
- At t= 7, each of the first 7 tuples is included in the sample S with equal prob = 2/7

Maintaining a fixed-size sample cont'd

- problem?
- Don't know length of stream in advance,
- so need to store all the *n* tuples seen so far
  - Ensure each item is in the sample S with equal prob. s/n

Solution: Fixed Size Sample

- Algorithm:
  - Store all the first ***s*** elements of the stream to ***S***
  - Suppose we have seen ***n-1*** elements, and now the ***n**th* element arrives (***n*** **>** ***s***)
    - With probability ***s/n***, keep the ***n**th* element, else discard it
    - If we picked the ***n**th* element, then it replaces one of the ***s*** elements in the sample ***S***, picked uniformly at random
- Claim: This algorithm maintains a sample *S* with the desired property

Proof: By Induction

- Property : after n elements, the sample contains each element seen so far with probability s/n
- Base case:
  - After we see n=s elements, Each out of n=s elements is in the sample with probability $s/s = 1$
- Inductive hypothesis:
  - After n elements, the sample S contains each element seen so far with prob. s/n
- Now element $n+1$ arrives

Proof: By Induction cont'd

- Inductive step: For elements already in *S*, probability of remaining in *S* is: $(1-\frac{s}{n+1})+(\frac{s}{n+1})(\frac{s-1}{s})=\frac{n}{n+1}$

- Time *n* to *n+1*, tuple stayed in *S* with prob. n/(n+1)
- so prob. Tuple is in s at time n+1 = $\frac{s}{n}\times\frac{n}{n+1}=\frac{s}{n+1}$

Outline

1.1 Intro of Data Stream?

1.2 Sampling from a Data Stream

1.3 Queries over a Sliding Window

Sliding Window

- Model: queries are about a *window* of length *N* the *N* most recent elements received

- Interesting case:
  N is so large it cannot be stored in memory or even on disk Or, there are so many streams that windows for all cannot be stored

Sliding Window: 1 Stream

- Sliding window on a single stream: N=6

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past   Future →

Counting Bits(1)

- Problem:

  Given a stream of 0s and 1s

  Be prepared to answer queries of the form

  How many 1s are in the last k bits? where $k \leq N$

- Obvious solution:

  Store the most recent N bits

  When new bit comes in, discard the N+1st bit

0 1 0 0 1 1 0 1 1 1 0 1 0 1 0 1 1 0 $\boxed{1\ 1\ 0\ 1\ 1\ 0}$ Suppose N=6

← Past Future →

Counting Bits(2)

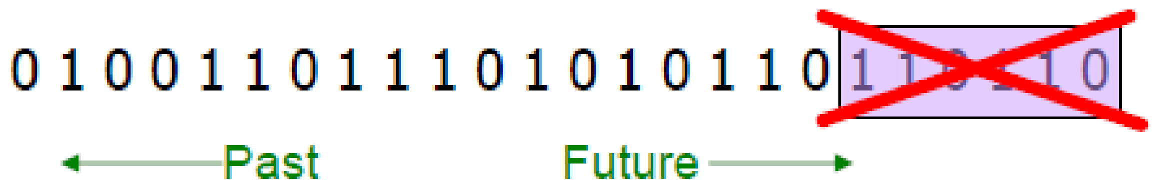
- You can not get an exact answer without storing the entire window

- Real Problem:

What if we cannot afford to store *N* bits?

E.g., we're processing 1 billion streams and

N = 1 billion



- But we are happy with an approximate answer

An attempt: Simple solution

- How many 1s are in the last N bits?
- Simple solution that does not really solve our problem: Uniformity assumption

← N →

0 1 0 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 1 0 1 0

← Past   Future →

- Maintain 2 counters:

  S: number of 1s from the beginning of the stream

  Z: number of 0s from the beginning of the stream

- How many 1s are in the last N bits $N \cdot \frac{S}{S+Z}$

- But, what if stream is non-uniform?

  What if distribution changes over time?

DGIM Method

- DGIM solution that does <u>not</u> assume uniformity
- We store $O(\log^2 N)$ bits per stream
- Solution gives approximate answer

  Error factor can be reduced to any fraction > 0, with more complicated algorithm and proportionally more stored bits

DGIM: Timestamps

- Each bit in the stream has a *timestamp*, starting 1, 2, …
- Record timestamps modulo N (the window size), so we can represent any relevant timestamp in $O(\log_2 N)$ bits

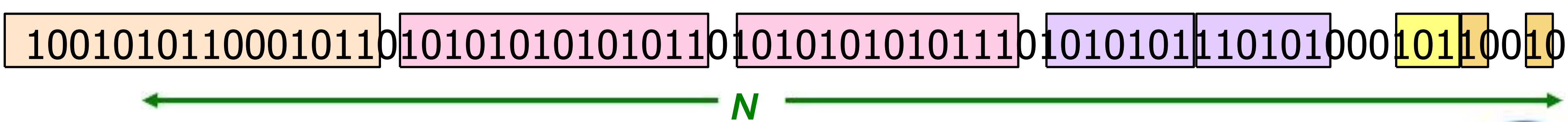
DGIM: Buckets

- A *bucket* in the DGIM method is a record consisting of:

1. The timestamp of its end [O(log N) bits]
2. The number of 1s between its beginning and end [O(log log N) bits]

- *Constraint on buckets:*
  Number of 1s must be a power of 2

- *That explains the O(log log N) in 2.*

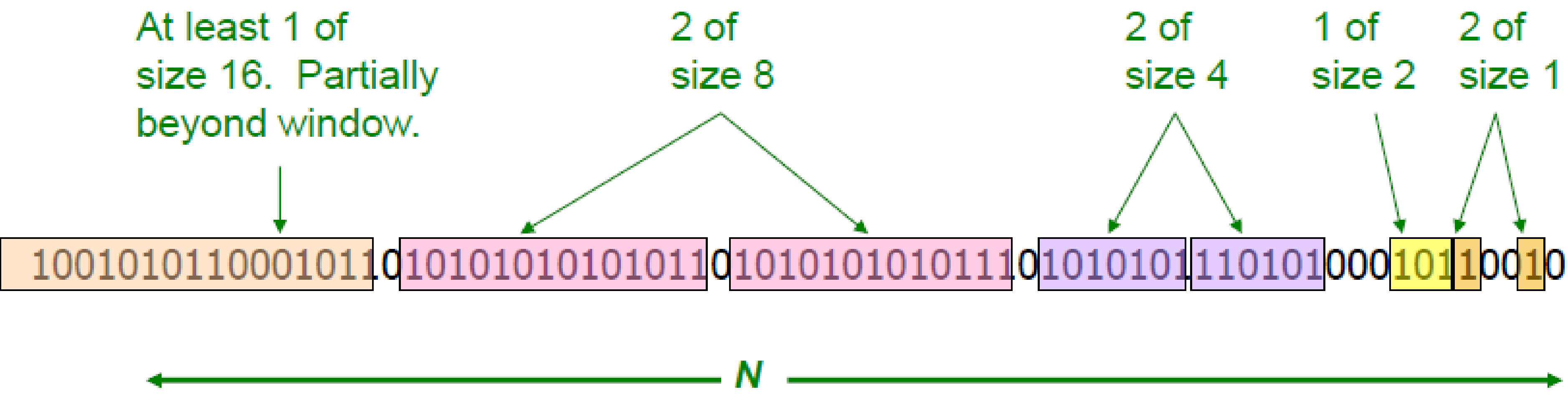


Representing a Stream by Buckets

- Either one or two buckets with the same power-of-2 number of 1s
- Buckets do not overlap in timestamps
- Buckets are sorted by size

  Earlier buckets are not smaller than later buckets
- Buckets disappear when their end-time is > N time units in the past

Example: Bucketized Stream



Three properties of buckets that are maintained:

- Either one or two buckets with the same power-of-2 number of 1s
- Buckets do not overlap in timestamps
- Buckets are sorted by size

Updating Buckets (1)

- When a new bit comes in, drop the last (oldest) bucket if its end-time is prior to N time units before the current time

- 2 cases: Current bit is 0 or 1

- If the current bit is 0: no other changes are needed

Updating Buckets (2)

- If the current bit is 1:

(1) Create a new bucket of size 1, for just this bit

End timestamp = current time

(2) If there are now three buckets of size 1, combine the oldest two into a bucket of size 2

(3) If there are now three buckets of size 2, combine the oldest two into a bucket of size 4

(4) And so on …

Example: Updating Buckets

Current state of the stream:

100101011000101101010101010101101010101010111010101011101010001011 0010

Bit of value 1 arrives

0010101100010110101010101010110101010101011101010101110101000101100101

Two orange buckets get merged into a yellow bucket

0010101100010110101010101010110101010101011101010101110101000101100101

Bit 1 arrives, new orange bucket is created, then 0 comes, then 1:

0101100010110101010101010110101010101011101010101110101000101100101101

Buckets get merged…

0101100010110101010101010110101010101011101010101110101000101100101101

State of the buckets after merging

0101100010110101010101010110101010101011101010101110101000101100101101

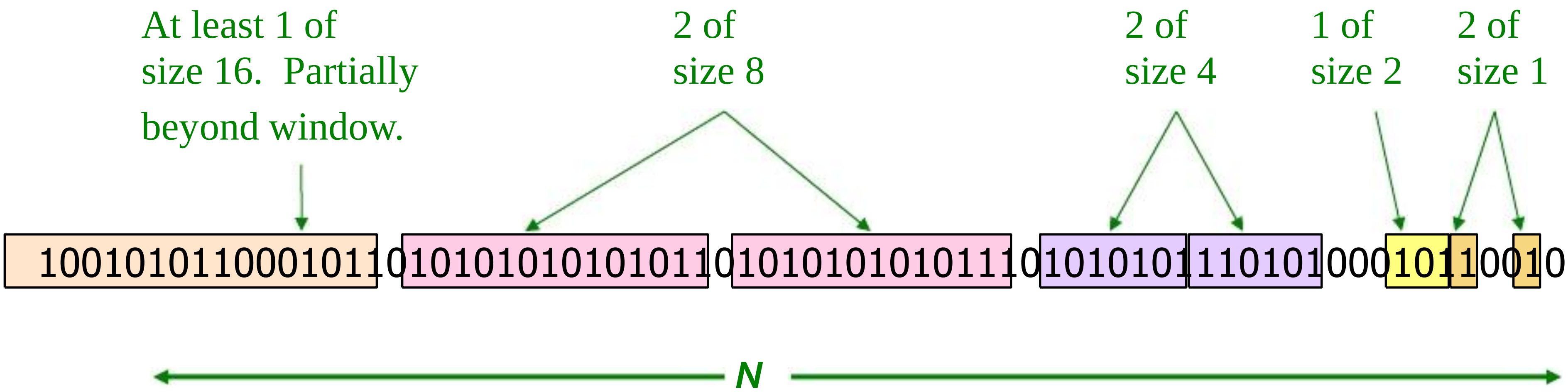
How to Query

- To estimate the number of 1s in the most recent N bits:

1. Sum the sizes of all buckets but the last
   (note "size" means the number of 1s in the bucket)
2. Add half the size of the last bucket

- Remember: We do not know how many 1s of the last bucket are still within the wanted window

Example: Bucketized Stream



Error Bound: Proof

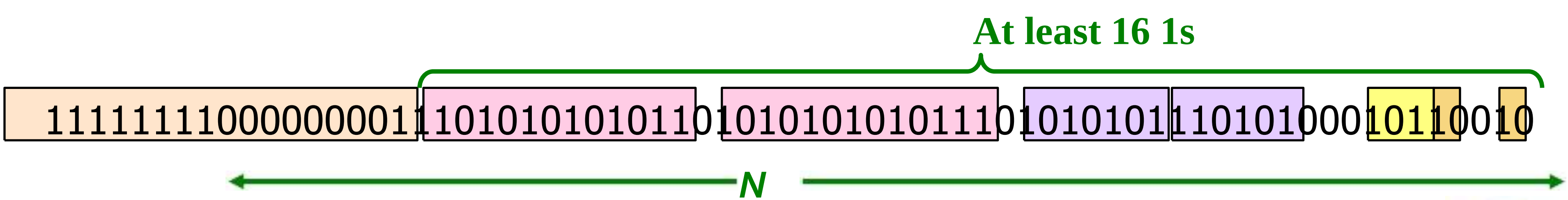
- Why is error 50%? Let's prove it!
- Suppose the last bucket has size 2^r
- Then by assuming 2^{r-1} (i.e., half) of its 1s are

  still within the window, we make an error of at most 2^{r-1}
- Since there is at least one bucket of each of

  the sizes less than 2^r, the true sum is at least

  $1 + 2 + 4 + .. + 2^{r-1} = 2^r-1$

- Thus, error at most 50%

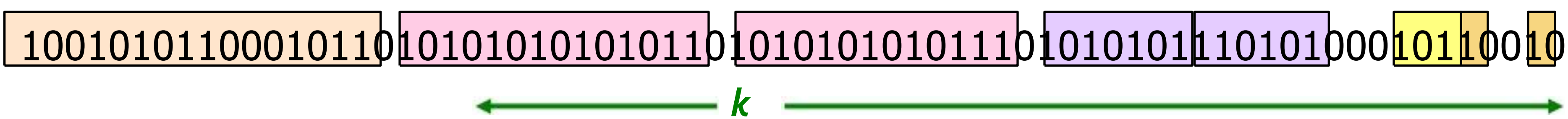


Extensions

- Can we use the same trick to answer quer*ies*

  How many 1's in the last k? where k < N?

  - A: Find earliest bucket B that at overlaps with k. Number of 1s is the sum of sizes of more recent buckets + ½ size of B



- Can we handle the case where the stream is not bits, but integers, and we want the sum of the last k elements?

Further Reducing the Error

- Instead of maintaining 1 or 2 of each size bucket, we allow either r-1 or r for $r > 2$

  Except for the largest size buckets; we can have any

  number between 1 and r of those

- Error is at most 1/(r)
- By picking r appropriately, we can tradeoff between number of bits we store and the error

Extensions cont'd

- Stream of positive integers
- We want the sum of the last k elements
- Solution:

  (1) If you know all integers have at most m bits

  Treat m bits of each integer as a separate stream

  Use DGIM to count 1s in each integer c_i …estimated count for i-th bit

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

  (2) Use buckets to keep partial sums

  Sum of elements in size b bucket is at most 2b

More Algorithms on Data Streams

Types of queries one wants on answer on a data stream:

Filtering a data stream: Bloom Filters

Select elements with property *x* from the stream

Counting distinct elements: Flajolet-Martin

Number of distinct elements in the last *k* elements of the stream

Estimating moments: *AMS method*

Estimate avg./std. dev. of last *k* elements

Finding frequent elements

Description

- Each element of data stream is a tuple
- Given a list of good keys S
- Determine which tuples of stream are in *S*
- Obvious solution: Hash table
  - But suppose we do not have enough memory to store all of *S* in a hash table
  - E.g., we might be processing millions of filters on the same stream

Application

- Email spam filtering
  - We know 1 billion "good" email addresses
  - If an email comes from one of these, it is **NOT** spam
  - About 80% emails are spam
- Publish-subscribe systems
  - You are collecting lots of messages (news articles)
  - People express interest in certain sets of keywords
  - Determine whether each message matches user's interest

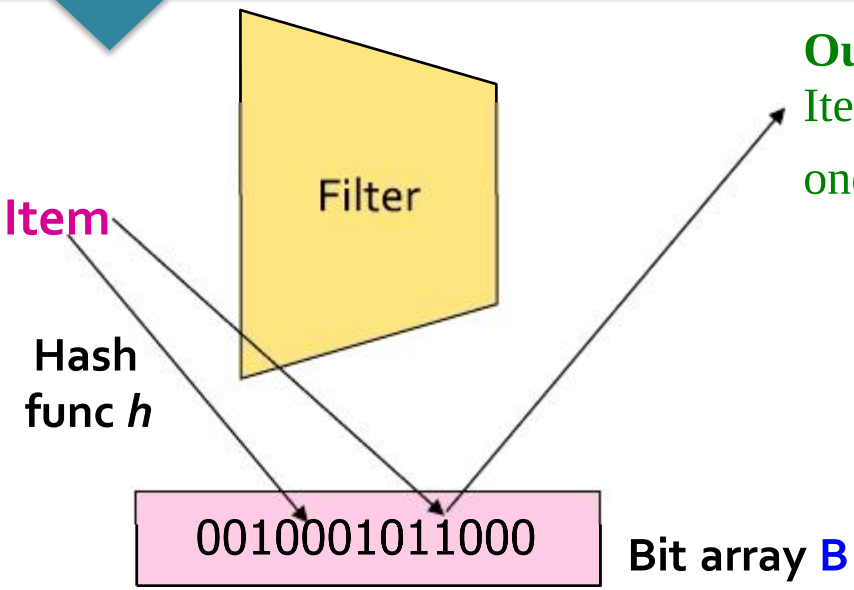
Application

- How many different words are found among the Web pages being crawled at a site?
- How many different Web pages does each customer request in a week?
- How many distinct products have we sold in the last week?

First Cut Solution

- Given a set of keys *S* that we want to filter
- Create a bit array B of n bits, initially all 0s
- Choose a hash function h with range [0,n)
- Hash each member of $s \in S$ to one of n buckets, and set that bit to 1, i.e., B[h(s)]=1
- Hash each element a of the stream and output only those that hash to bit that was set to 1, i.e., output a if B[h(a)] == 1

First Cut Solution cont'd



Output the item since it may be in *S*.
Item hashes to a bucket that at least one of the items in **S** hashed to.

Drop the item.
It hashes to a bucket set to **0** so it is surely not in ***S***.

- Creates false positives but no false negatives
- If the item is in S we surely output it, if not we may still output it

Prob. of False Negatives

- $|S|$ = 1 billion email addresses
  $|B|$= 1GB = 8 billion bits

- Approximately 1/8 of the bits are set to 1

- About 1/8 of the addresses not in S get through to the output (false positives)

- Actually, less than 1/8th, because more than one address might hash to the same bit

Throwing Darts

- Consider: Throw m darts into n targets equally, what is the probability that a target gets at least one dart?

- In our case:
  - Targets = bits/buckets
  - Darts = hash values of items

- Prob. a given dart will not hit a given target: (n-1)/n
- Prob. None of the m darts will hit a given target:

$$\left(\frac{n-1}{n}\right)^m -> \left(1-\frac{1}{n}\right)^{n\left(\frac{m}{n}\right)} \approx e^{-m/n}$$

Throwing Darts cont'd

- Fraction of 1s in the array B == probability of false positive == 1 – $e^{-m/n}$

- |S| = 1 billion email addresses
  |B|= 1GB = 8 billion bits

$$1 - e^{-\frac{m}{n}} = 1 - e^{-\frac{1}{8}} = 0.1175 \approx 0.125$$

Bloom Filter

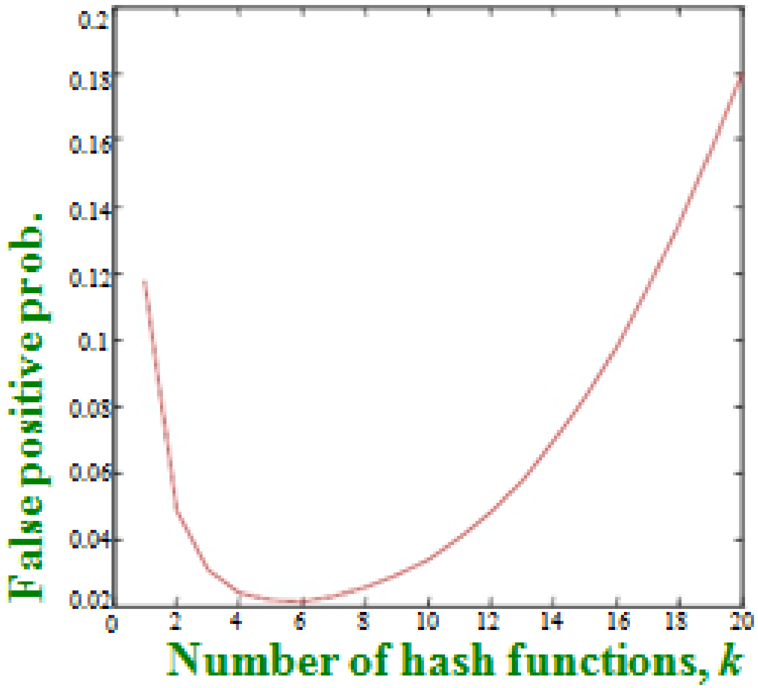
- Consider: $|S| = m$, $|B| = n$
- Use k independent hash functions $h1, \ldots, hk$
- Initialization:
  - Set B to all 0s
  - Hash each element $s \in S$ using each hash function hi ,set $B[hi(s)] = 1$ (for each $i = 1, \ldots, k$)
- Run-time:
  - When a stream element with key x arrives Hash
  - If $B[hi(x)] = 1$ for all $i = 1, \ldots, k$ then x is in S
  - Otherwise discard the element

Bloom Filter cont'd

- What fraction of the bit vector B are 1s?
  - Throwing k·m darts at n targets
  - So fraction of 1s is $(1 - e^{-km/n})$
- But we have k independent hash functions
- So, false positive probability = $(1 - e^{-km/n})^k$

Bloom Filter cont'd

- m = 1 billion, n = 8 billion
  - k = 1: (1 – e-1/8) = 0.1175
  - k = 2: (1 – e-1/4)2 = 0.0493



- What happens as we keep increasing k?

- "Optimal" value of k: (n/m) ln(2)
  - In our case: Optimal k = 8 ln(2) = 5.54 ≈ 6

Outline

1.1 Filtering a data stream?

1.2 Counting distinct elements

1.3 Estimating moments

1.4 Counting frequent items

Description

- Data stream consists of a universe of elements chosen from a set

- Maintain a count of the number of distinct elements seen so far

- Obvious solution:
  maintain the set of elements seen so far
  - That is, keep a hash table of all the distinct elements seen so far
  - What if we do not have space to maintain the set of elements seen so far?
    - estimate the count in unbiased way
    - Accept that the count may have a little error but limit the probability that the error is large

Flajolet-Martin Algorithm

- Pick a hash function h that maps each of the N elements to at least $\log_2 N$ bits
- For each stream element a, let r(a) be the number of trailing 0s in h(a)
  - say h(a) = 12, then 12 is 1100 in binary, so r(a) = 2
- Record R = the maximum r(a) seen
- Estimated number of distinct elements = 2^R

Why it works: Intuition

- $h(a)$ hashes a with equal prob. to any of N values

- Then $h(a)$ is a sequence of $\log_2 N$ bits, where 2^{-r} fraction of all a have a tail of r zeros
  - About 50% of as hash to $***0$
  - About 25% of as hash to $**00$
  - So, if we saw the longest tail of $r=2$ (i.e., item hash ending $*100$) then we have probably seen about 4 distinct items so far

- So, it takes to hash about 2^r items before we see one with zero-suffix of length r

Why it works: Formally

- Now we show why M-F works

- Formally, we will show that probability of NOT finding a tail of r zeros:
  - Goes to 1 if $m \gg 2^r$
  - Goes to 0 if $m \ll 2^r$
  - where m is the number of distinct elements seen so far in the stream

Why it works: Formally cont'd

- h(a) hashes elements uniformly at random
- Prob. that a random number ends in at least r zeros is 2^{-r}
- The, the probability of NOT seeing a tail of length r among m elements:

$$\left(\boxed{1 - 2^{-r}}\right)^m$$

Prob. all end in fewer than ***r*** zeros.

Prob. that given **h(a)** ends in fewer than ***r*** zeros

Why it works: Formally cont'd

- Note: $(1-2^{-r})^m = (1-2^{-r})^{2^{r}(m2^{-r})} \approx e^{-m2^{-r}}$
- If m << 2^r, $(1-2^{-r})^m \approx e^{-m2^{-r}} = 1$
  - So, the probability of finding a tail of length r tends to 0
- If m >> 2^r, $(1-2^{-r})^m \approx e^{-m2^{-r}} = 0$
  - So, the probability of finding a tail of length r tends to 1
- Thus, 2^R will almost always be around m

Combining Estimates

- Consider, using many hash functions hi and getting many samples of R_i
- How are samples R_i combined?
  - Average? What if one very large value 2^{R_i}?
  - Median? All estimates are a power of 2
- Solution:
  - Partition your samples into small groups
  - Take the average of groups
  - Then take the median of the averages

Outline

1.1 Filtering a data stream?

1.2 Counting distinct elements

<u>1.3 Estimating moments</u>

1.4 Counting frequent items

Generalization

- What is moment?
- Suppose a stream has elements chosen from a set *A* of N ordered values
- Let m_i be the number of times value i occurs in the stream
- The kth-order moment is

$$\sum_{i \in A} (m_i)^k$$

Special case

$$\sum_{i \in A} (m_i)^k$$

- 0^{th} moment = number of distinct elements
  - The problem just considered
- 1st moment = count of the numbers of elements = length of the stream
  - Easy to compute
- 2nd moment = surprise number S = a measure of how uneven the distribution is

Surprise Number

- Stream of length 100
- 11 distinct values
- Item counts: 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9
  Surprise S = 910
- Item counts: 90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1
  Surprise S = 8,110

Alon-Matias-Szegedy Algorithm

- We will begin with the 2nd moment S
- We keep track of many variables X
- For each variable X we store X.el and X.val
  - X.el corresponds to the item i
  - X.val corresponds to the count of item i
  - Note this requires a count in main memory, so number of Xs is limited
- Our goal is to compute:

$$S = \sum_i m_i^2$$

Alon-Matias-Szegedy Algorithm

- How to set X.val and X.el?
  - Assume stream has length n (we relax this later)
  - Pick some random time t (t<n) to start, so that any time is equally likely
  - Let at time t the stream have item i. We set X.el = i
  - Then we maintain count c (X.val = c) of the number of is in the stream starting from the chosen time t

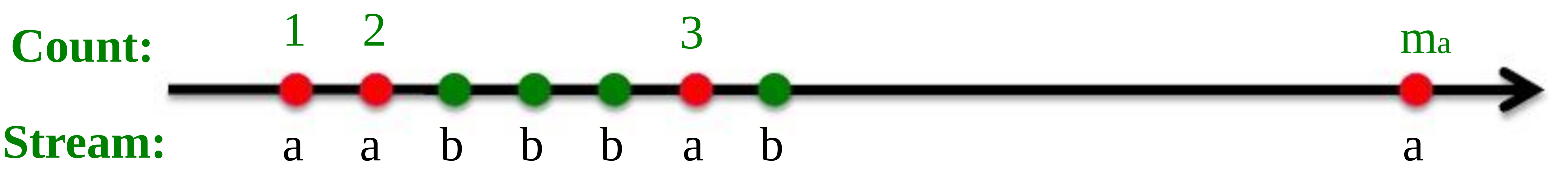
- Then the estimate of the 2nd moment is:

$$S = f(x) = n(2c - 1)$$

- Note, we keep track of multiple Xs, (X1, X2,…Xk), and our final estimate will be

$$S = \frac{1}{k}\sum_j f(X_j)$$

Expectation Analysis



■ c_t … number of times record at time t appears from that time on ($c_1=m_a$, $c_2=m_a-1$, $c_3=m_b$)

$$E[f(x)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1)$$

$$= \frac{1}{n}\sum_{a} n(1+3+5+\ldots+2m_a - 1)$$

$$= \sum_{a}(m_a)^2$$

$$\sum_{a=1}^{m_a}(2i-1) = 2\frac{m_a(m_a+1)}{2} - m_a = (m_a)^2$$

Example

- Stream: a,b,c,b,d,a,c,d,a,b,d,c,a,a,b
- length of the stream n = 15
- a: 5, b: 4, c: 3, d: 3, surprise num:$5^2 + 4^2 + 3^2 + 3^2 = 59$
- Three variables X1, X2, X3
- Random pick 3rd, 8th, 13th position to define X1,X2,X3
  - X1.el = c, X1.val = 3
  - X2.el = d, X2.val = 2
  - X3.el = a, X3.val = 2
  - an estimate for any variable X: n*(2*X.val – 1)
  - 75,45,45→55

Higher Order Moment

- For estimating kth moment we essentially use the same algorithm but change the estimate:
  - For k=2 we used n (2·c – 1)
  - For k=3 we use: n (3·c^2 – 3c + 1)

- Why?
  - For k=2: Remember we had 1+3+5+⋯+2m_i−1 and we showed terms 2c-1 (for c=1,…,m) sum to m^2
  - So: $$\sum_{c=1}^{m} 2c-1 = \sum_{c=1}^{m} c^2 - \sum_{c=1}^{m} (c-1)^2 = m^2$$
  - For k=3: c^3 - (c-1)^3 = 3c^2 - 3c + 1

- Generally: Estimate = $n(c^k - (c-1)^k)$

Combining Samples

- In practice:
  - For k=2 we used n (2·c - 1)
  - In practice, Compute $f(X) = n(2c - 1)$ for as many variables X as you can fit in memory
  - Average them in groups
  - Take median of averages

- Problem: Streams never end
  - We assumed there was a number n, the number of positions in the stream
  - But real streams go on forever, so n is a variable – the number of inputs seen so far

Stream never ends

- The variables X have n as a factor – keep n separately; just hold the count in X

- Suppose we can only store k counts. We must throw some Xs out as time goes on:

- Objective: Each starting time t is selected with probability k/n

- Solution: (fixed-size sampling!)
  - Choose the first k times for k variables
  - When the nth element arrives (n > k), choose it with probability k/n
  - If you choose it, throw one of the previously stored variables X out, with equal probability

Chapter 4: Mining Data Stream 2

Outline

1.1 Filtering a data stream?

1.2 Counting distinct elements

1.3 Estimating moments

1.4 Counting frequent items

Counting items

- New Problem: Given a stream, which items appear more than s times in the window?

- Possible solution
  - Think of the stream of baskets as one binary stream per item,1 = item present; 0 = not present
  - Use DGIM to estimate counts of 1s for all items
  - One stream per itemset, Number of itemsets is way too big

Exponentially Decaying Windows

- Exponentially decaying windows:
  - What are "currently" most popular movies?
  - Instead of computing the raw count in last N elements
  - Compute a smooth aggregation over the whole stream

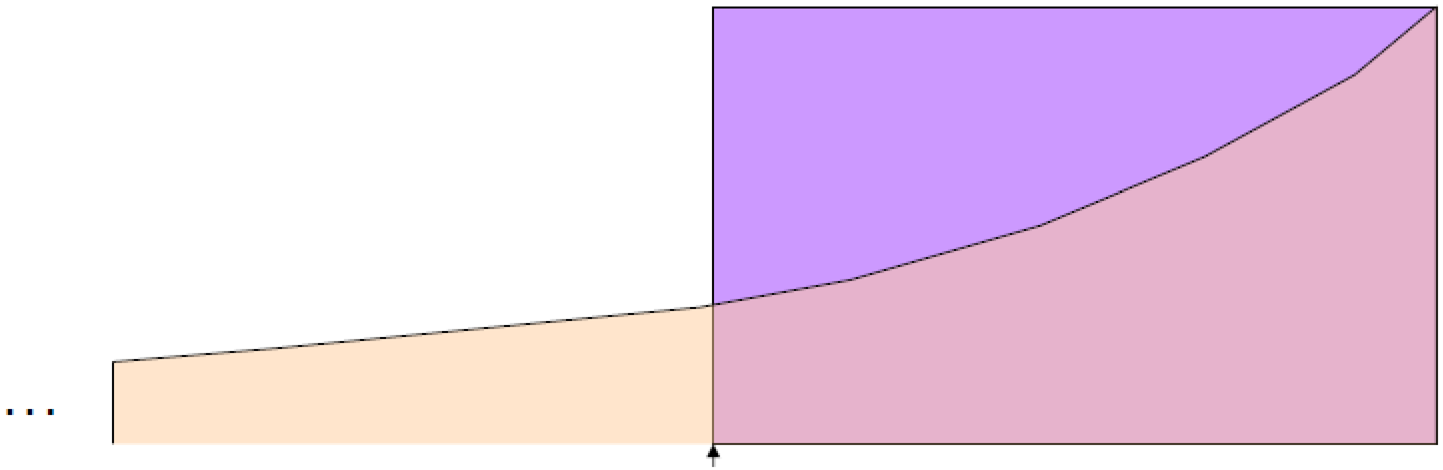
- If stream is a1, a2,… and we are taking the sum of the stream, take the answer at time t to be: $\sum_{i=1}^{t} a_i (1-c)^{t-i}$
c is a constant, presumably tiny, like 10^{-6} or 10^{-9}

- When new at+1 arrives:
Multiply current sum by (1-c) and add at+1

Example: Counting Items

- Imagine that for each item x we have a binary stream (1 if x appears, 0 if x does not appear)
- Calculate: $\sum_{i=1}^{t} \delta_i (1-c)^{t-i}$ where δ_i=1 if a_i =x, and 0 otherwise
- New item x arrives:
  - Multiply all counts by (1-c)
  - Add +1 to count for element x
- Call this sum the "weight" of item x

Decaying Window



Finding the Most Popular Elements

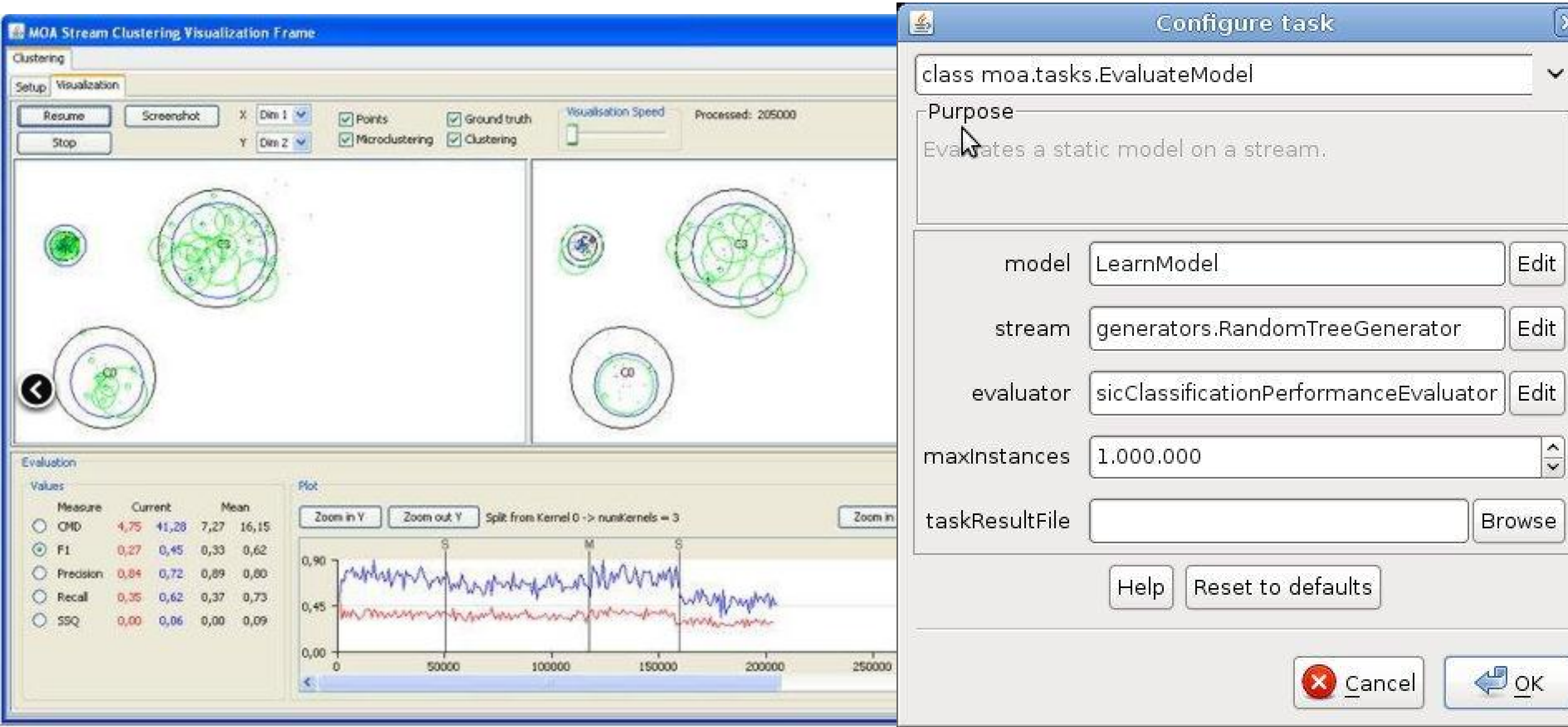
- Score Threshold : ½

- When a new element m comes in:
  - Multiply all counts by (1-c)
  - Add 1 to count of element m
  - For uncounted element m, create one and initialize it to 1
  - Drop score < ½

- There cannot be more than 2/c movies with weight of ½ or more

Data Stream Application

- MOA(Massive Online Analysis): ree open-source software specific for mining data streams



Data Stream Application

- Monitoring IP addresses

Exercise 4. We have a stream of financial transactions including securities, options etc. collected from multiple sources. What are interesting data analysis queries on such data streams?

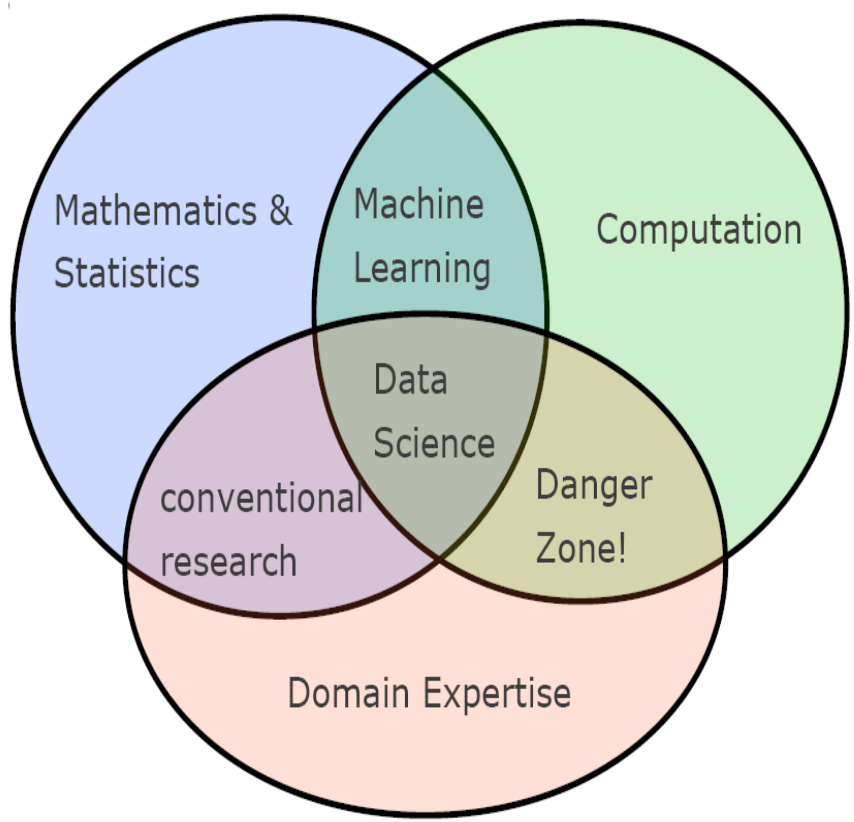
Exercise 5. We have continuous physical observations – temperature, pressure, EMG/ECG/EEG signals from humans, humidity – from an ad hoc network of sensors. What are interesting data analysis queries on such data streams?

中国科学院大学
University of Chinese Academy of Sciences

Short Review of course part II

Review of Mining Massive Datasets

- **Data Science: automatically extracting knowledge from data**
  - Mathematics & Statistics
  - Machine Learning
  - Domain Expertise
- **Applications in Business**
  - Lots and lots
- **Applications in the Sciences**
  - Astronomy, Cosmology
  - High-energy Physics
  - Biology, Genomics
  - Neuroscience
  - The Social Sciences
- **Medicine, education**
- **Government**



What have been ?

- **Based on different types of data:**
  - Data is **high dimensional**
  - Data is **never-ending**
  - **Data is Graph**
- We learned various "tools" and how to solve real-world problems:
  - DHT
  - Recommender systems,
  - Linear algebra (SVD, Rec. Sys., Communities)
  - Optimization (stochastic gradient descent)
  - Hashing (LSH, Bloom filters)

What we have been ? DHT

- How do you search in O(log(n)) time?
  - Binary search
- You need an ordered array
- How can you order nodes in a network and data items?

Hash function

What we have been ? DHT

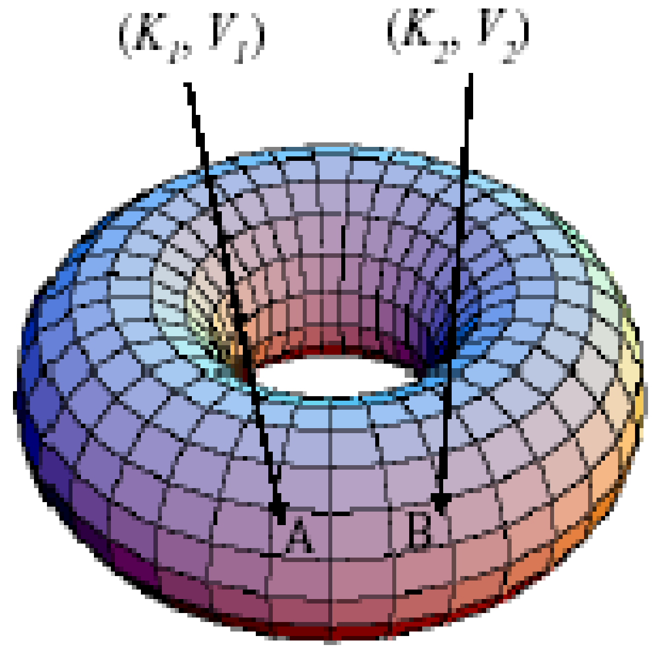
- Hash table spread over many nodes
  – Distributed over a wide area
- Main design goals
  – *Decentralization*
    - no central coordinator
  – *Scalability*
    - efficient even with large # of nodes
  – *Fault tolerance*
    - tolerate nodes joining/leaving

What Makes a Good DHT Design

- For each object, the node(s) responsible for that object should be reachable via a “short” path (small diameter)
  - The different DHTs differ fundamentally only in the routing approach
- The number of neighbors for each node should remain “reasonable” (small degree)
- DHT routing mechanisms should be decentralized (no single point of failure or bottleneck)
- Should gracefully handle nodes joining and leaving
  - Repartition the affected keys over existing nodes
  - Reorganize the neighbor sets
  - Bootstrap mechanisms to connect new nodes into the DHT
- To achieve good performance, DHT must provide low stretch
  - Minimize ratio of DHT routing vs. unicast latency

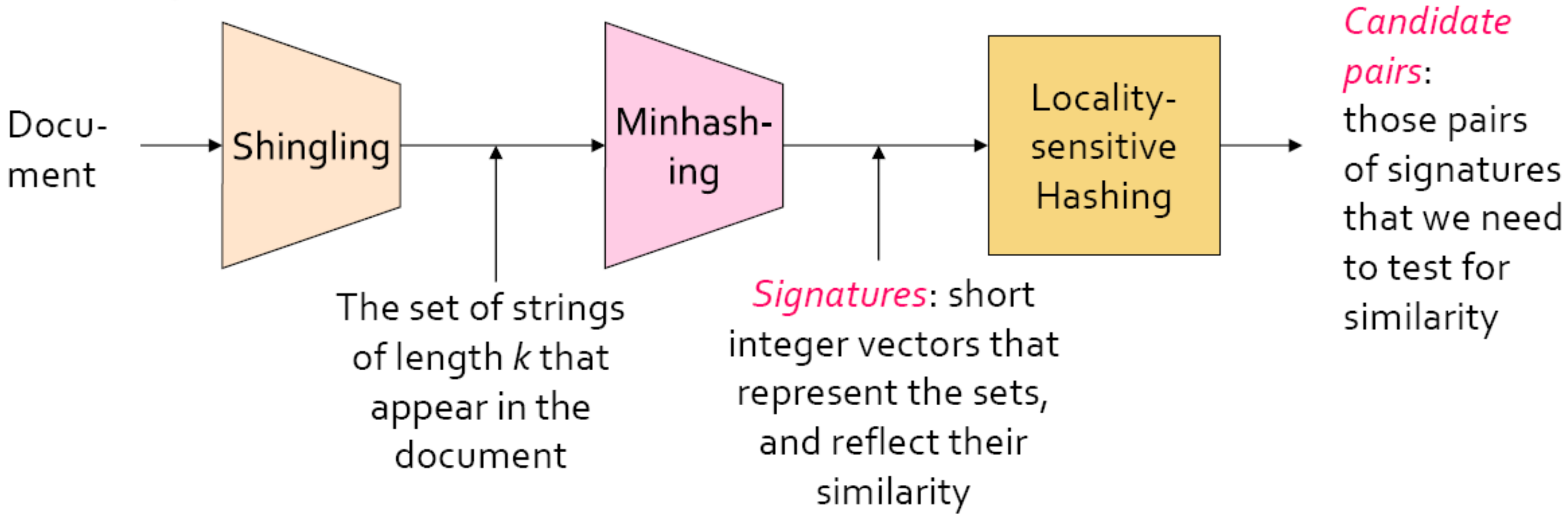
CAN: Zone and Key

- Use a virtual *d*-dimensional coordinate space
  - $[0, 1]^d = [0, 1] \times [0, 1] \times .. \times [0, 1]$
  - called a d-torus
- A peer is mapped to a "zone" of this d-torus and said to "own this zone"
- Each file F is identified with key K_F
- A hash function h maps a key to a point in the d-torus $K \rightarrow (x_1, x_2, .., x_d) \in [0, 1]^d$, where $0 \leq x_2 \leq 1$.



2-torus

Data is High-dimensional



1. Shingling: Convert documents to sets

2. Min-Hashing: Convert large sets to short signatures, while preserving similarity

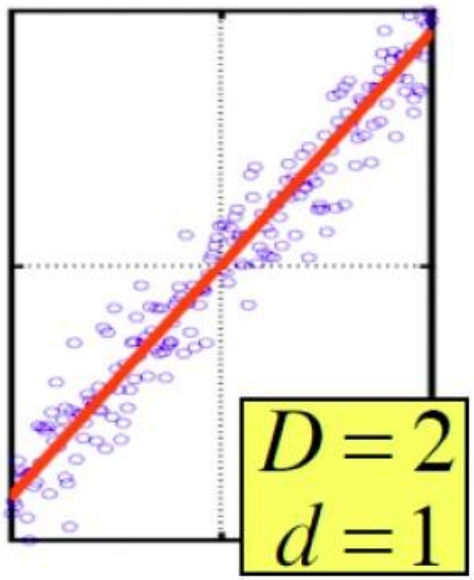
3. Locality-Sensitive Hashing: Focus on pairs of signatures likely to be from similar documents

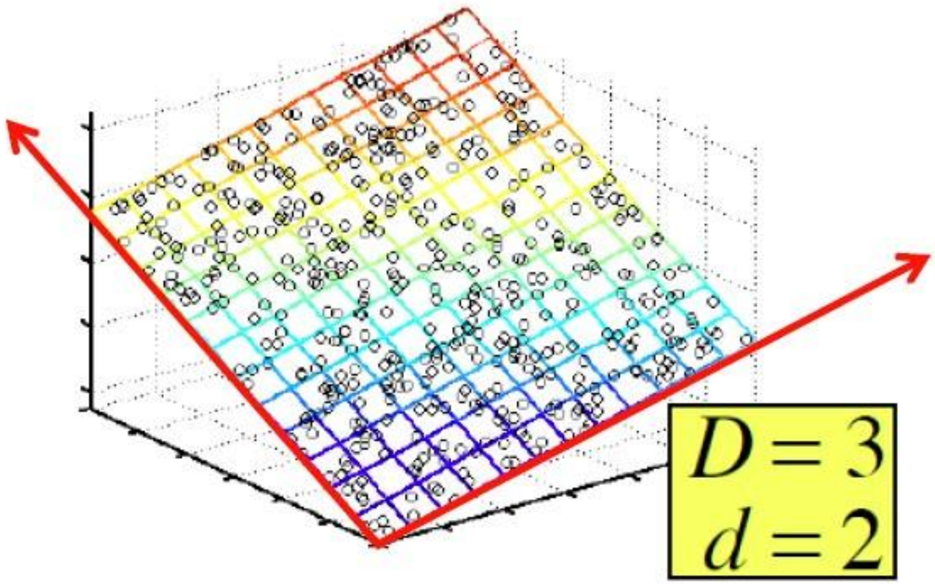
Locality -Sensitive(LS) Families

- Suppose we have a space S of points with a distance measure *d(x,y)*

- A family *H* of hash functions is said to be *(d1, d2, p1, p2)-sensitive* if for any *x* and *y* in *S*:
  - 1. If $d(x, y) < d1$, then the probability over all $h \in H$, that $h(x) = h(y)$ is <u>at least</u> *p1*
  - 2. If $d(x, y) > d2$, then the probability over all $h \in H$, that $h(x) = h(y)$ is <u>at most</u> *p2*

Dimensionality Reduction

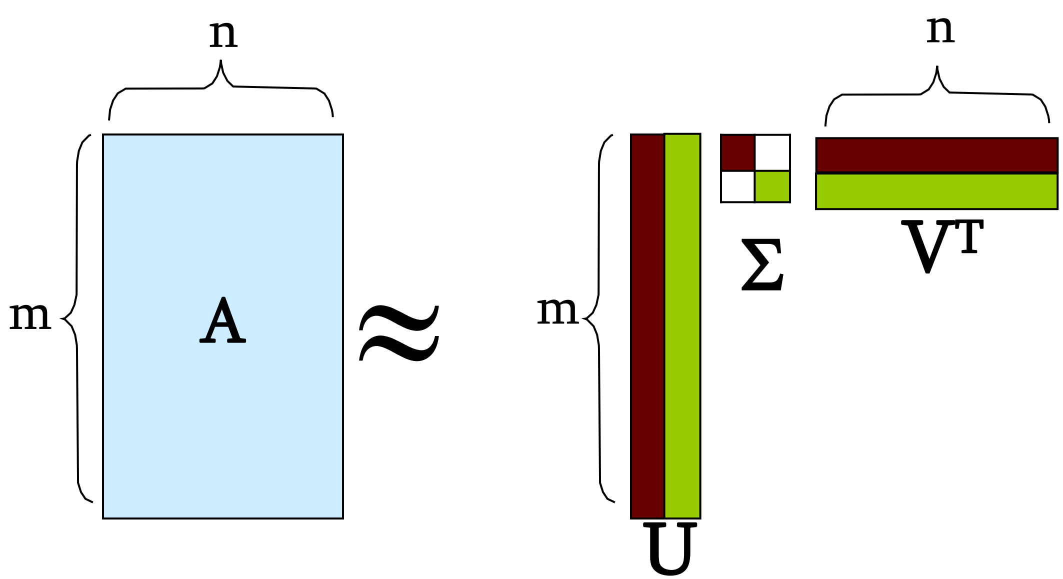




- **Assumption:** Data lies on or near a low *d*-dimensional subspace
- Axes of this subspace are effective representation of the data

SVD

$$\mathbf{A} \approx \mathbf{U\Sigma V}^T = \sum_i \sigma_i \mathbf{u}_i \circ \mathbf{v}_i^{\mathsf{T}}$$



SVD-Properties

It is **always** possible to decompose a real matrix A into $A = U \sum V^T$, where

- U, Σ, V: unique
- U, V: column orthonormal
  - $U^T U = I$; $V^T V = I$ (I: identity matrix)
  - (Columns are orthogonal unit vectors)
- Σ: diagonal
  - Entries (**singular values**) are positive,

    and sorted in decreasing order ($\boldsymbol{\sigma_1} \geq \boldsymbol{\sigma_2} \geq \ldots \geq \mathbf{0}$)

SVD-Best Low Rank Approx.

- $A = U \Sigma V^T$, $B = U S V^T$ $(\sigma_1 \geq \sigma_2 \geq \ldots \geq 0,\ \mathrm{rank}(A)=r)$
  - S = diagonal $n \times n$ matrix where $s_i=\sigma_i$ ($i=1\ldots k$) else $s_i=0$

  then B is solution to $\min_B \|A-B\|_F$, $\mathrm{rank}(B)=k$
- Why?

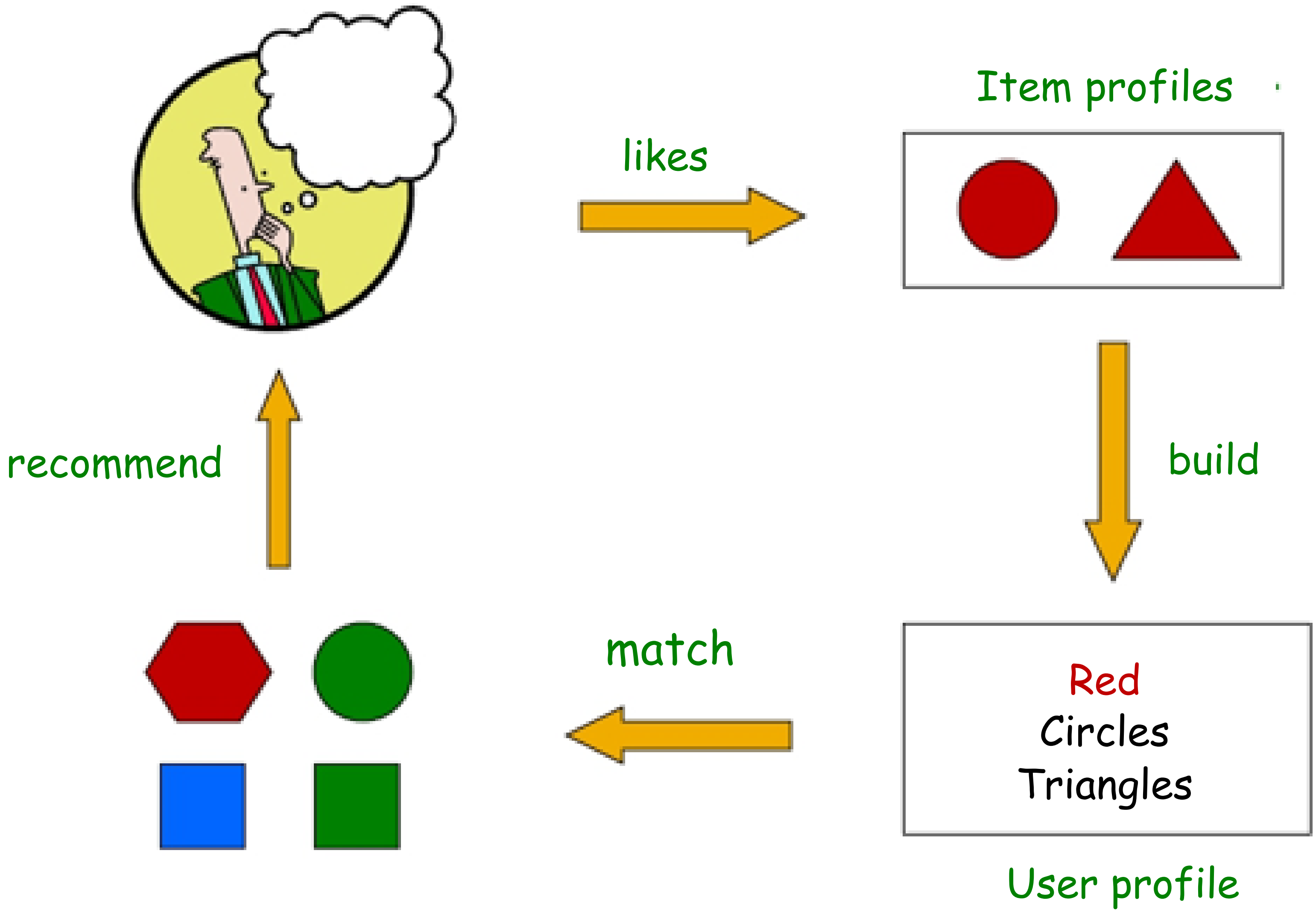
$$\min_{B,\, rank(B)=k} \|A - B\|_F = \min \|\Sigma - S\|_F = \min_{s_i} \sum_{i=1}^{r} (\sigma_i - s_i)^2$$

We used: $U \Sigma V^T - U S V^T = U (\Sigma - S) V^T$

- We want to choose s_i to minimize $\sum_i (\sigma_i - s_i)^2$
- Solution is to set $s_i=\sigma_i$ ($i=1\ldots k$) and other $s_i=0$

$$= \min_{s_i} \sum_{i=1}^{k} (\sigma_i - s_i)^2 + \sum_{i=k+1}^{r} \sigma_i^2 = \sum_{i=k+1}^{r} \sigma_i^2$$

Content based

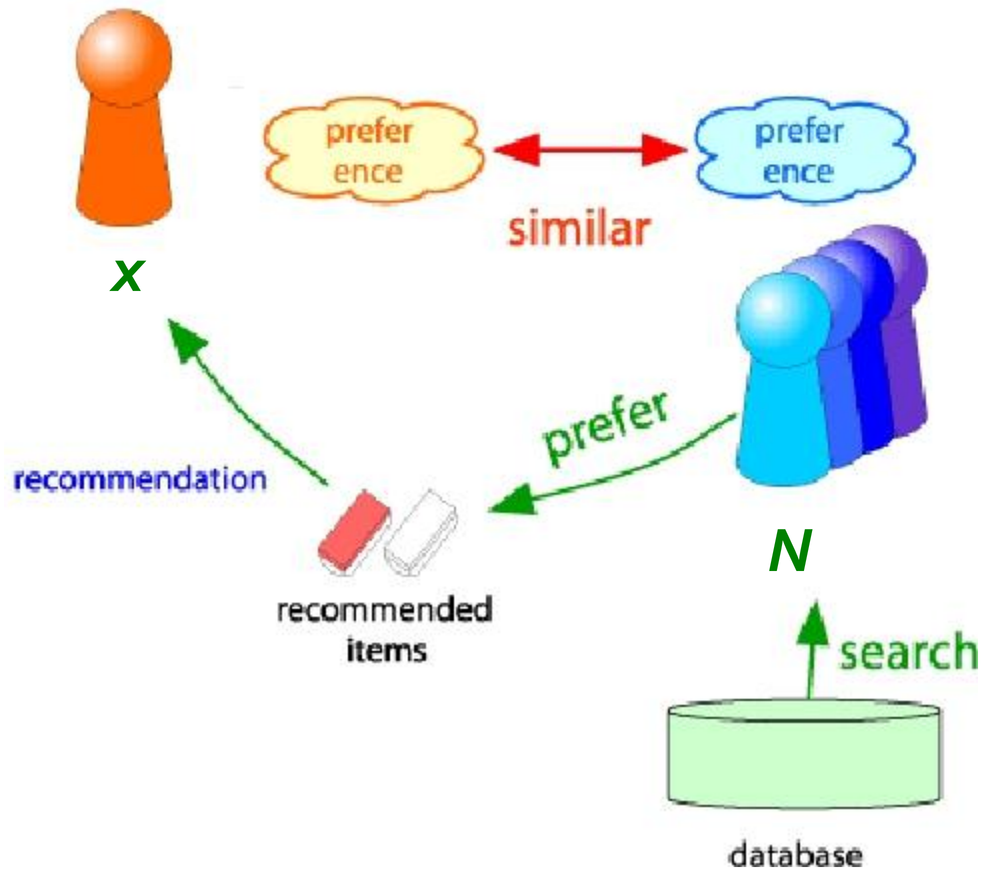


Collaborative Filtering

- Consider user *x*

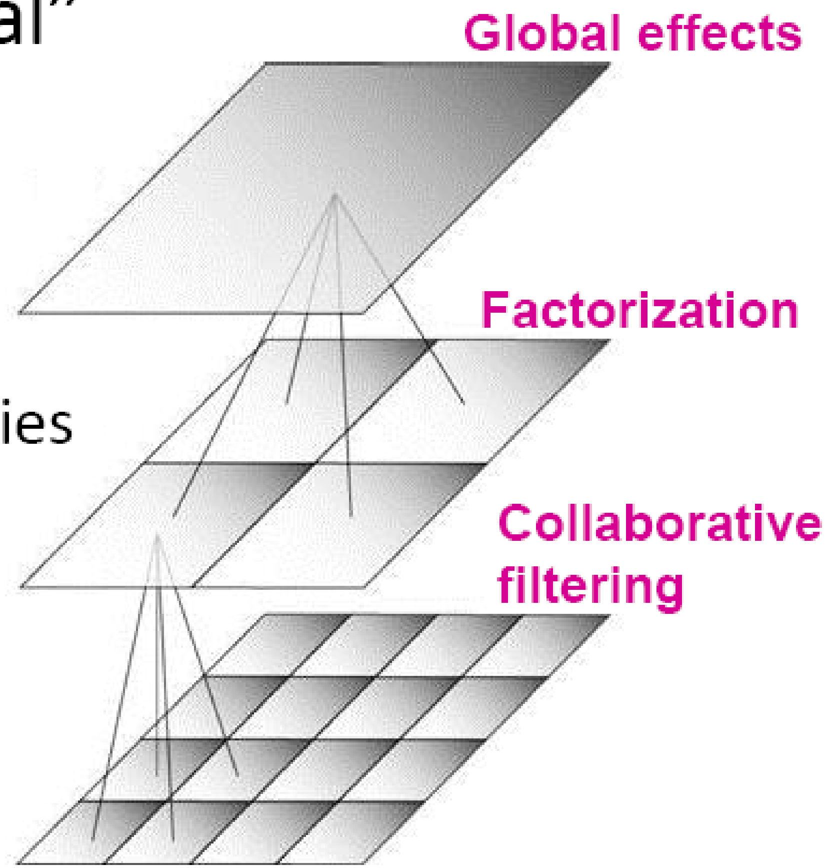
- Find set *N* of other users whose ratings are "similar" to *x*'s ratings

- Estimate *x*'s ratings based on ratings of users in *N*



Bellkor Recommender System

- **The winner of the Netflix Challenge**
- **Multi-scale modeling of the data:** Combine top level, "regional" modeling of the data, with a refined, local view:
  - **Global:**
    - Overall deviations of users/movies
  - **Factorization:**
    - Addressing "regional" effects
  - **Collaborative filtering:**
    - Extract local patterns



Fitting the New Model

- **Solve:**

$$\min_{Q,P} \sum_{(x,i)\in R} \left(r_{xi} - (\mu + b_x + b_i + q_i\, p_x^T) \right)^2$$

goodness of fit

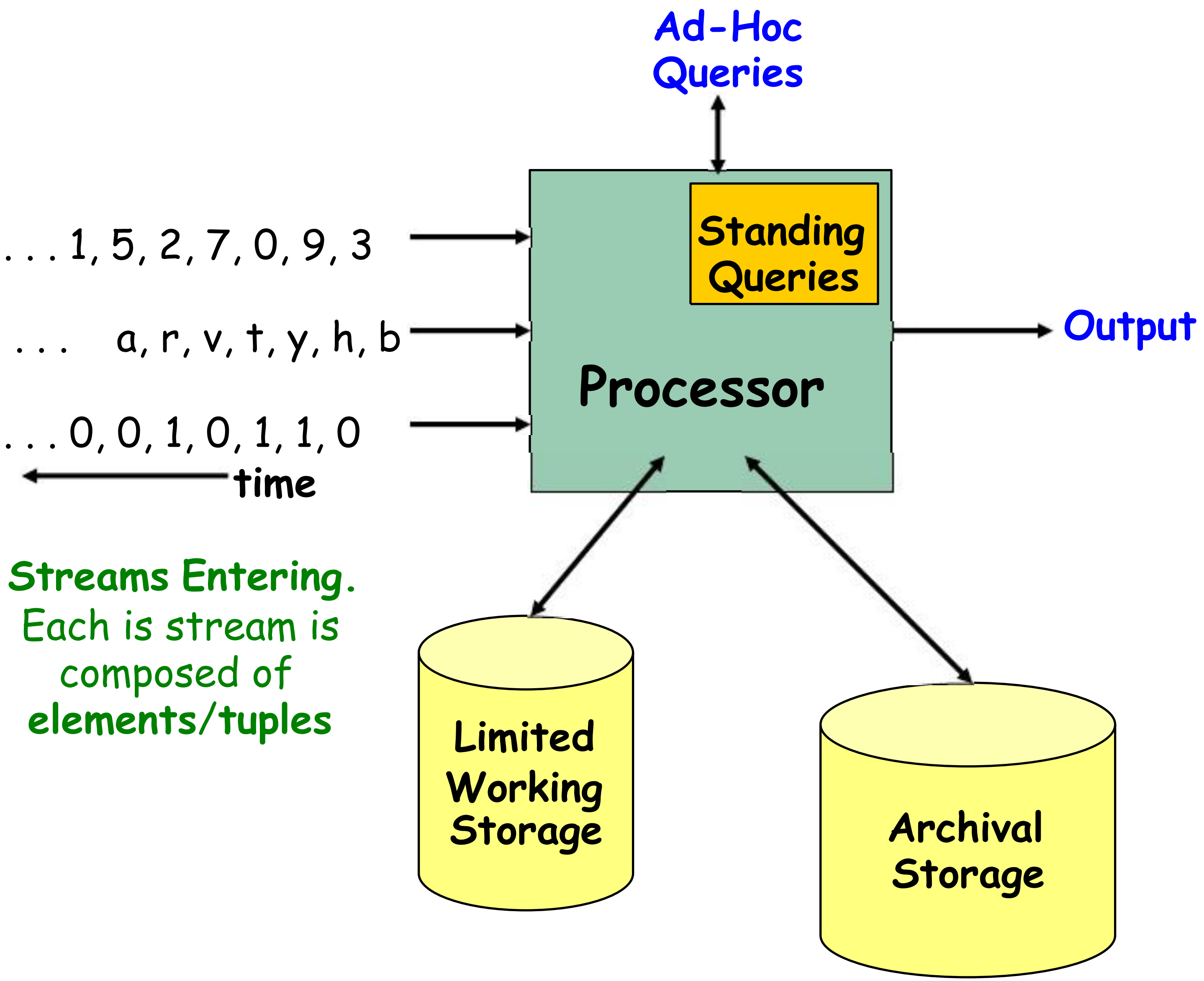
$$+ \lambda \left(\sum_i \|q_i\|^2 + \sum_x \|p_x\|^2 + \sum_x \|b_x\|^2 + \sum_i \|b_i\|^2 \right)$$

regularization

λ is selected via grid-search on a validation set

- **Stochastic gradient decent to find parameters**
  - **Note:** Both biases $\boldsymbol{b_u}$, $\boldsymbol{b_i}$ as well as interactions $\boldsymbol{q_i}$, $\boldsymbol{p_u}$ are treated as parameters (we estimate them)

General Stream Processing Model



Sample data from stream

- Algorithm:
  - Store all the first ***s*** elements of the stream to ***S***
  - Suppose we have seen ***n-1*** elements, and now the ***n***th element arrives (***n*** > ***s***)
    - With probability ***s/n***, keep the ***n***$_{th}$ element, else discard it
    - If we picked the ***n***$_{th}$ element, then it replaces one of the ***s*** elements in the sample ***S***, picked uniformly at random

Claim: Each element is included with prob. s/n

Queries from Sliding Window

- Model: queries are about a *window* of length *N*
  the *N* most recent elements received

- Interesting case:
  N is so large it cannot be stored in memory or even on disk Or, there are so many streams that windows for all cannot be stored

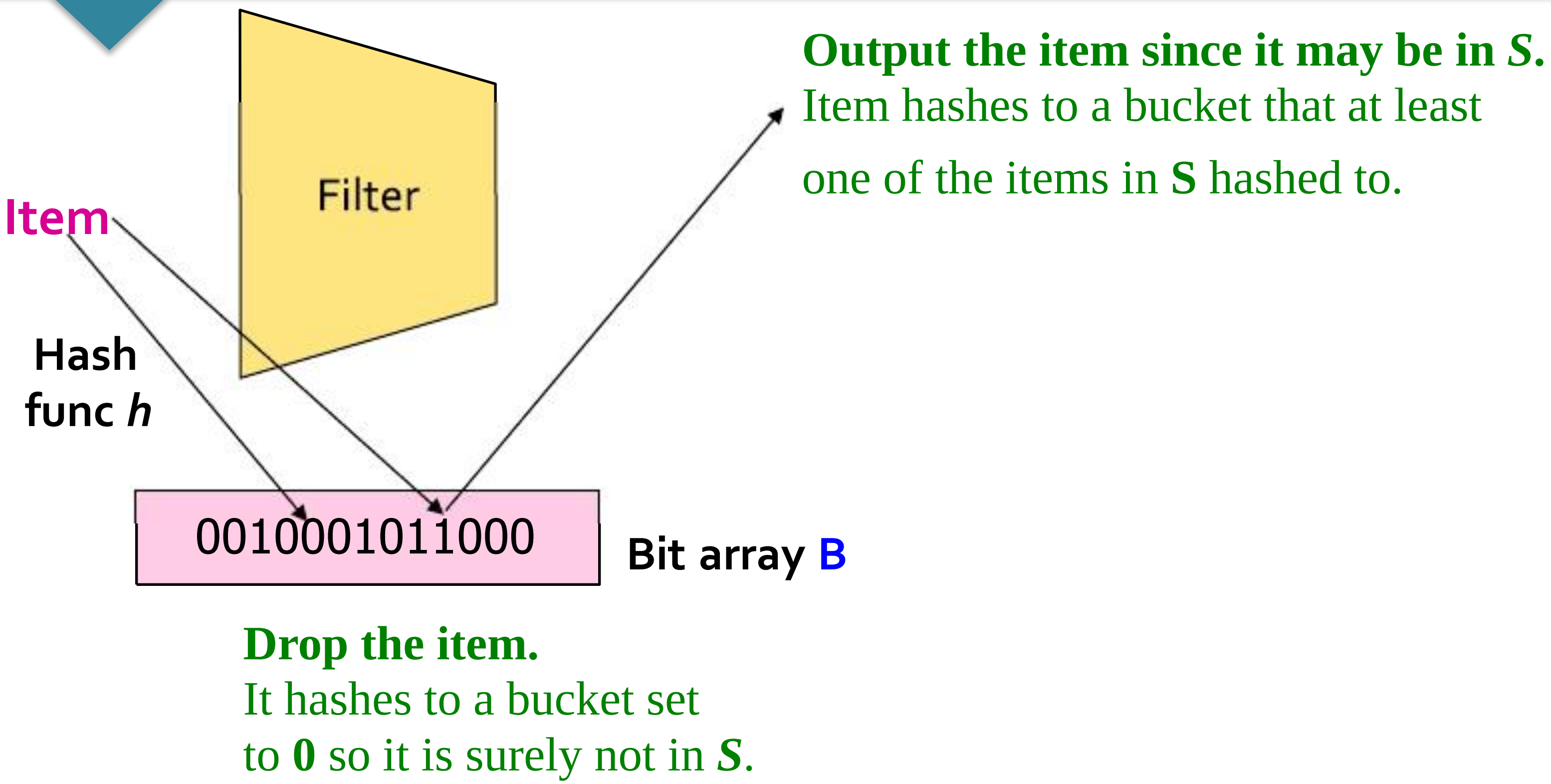
How many 1s are in last *k* bits?

10010101100010110101010101010110101010101011101010101110101000101100010

Filtering a data stream: Bloom filters

- Given a set of keys *S* that we want to filter
- Create a bit array B of n bits, initially all 0s
- Choose a hash function h with range [0,n)
- Hash each member of $s \in S$ to one of n buckets, and set that bit to 1, i.e., B[h(s)]=1
- Hash each element x of the stream and output only those that hash to bit that was set to 1, i.e., output x if B[h(a)] == 1

Filtering a data stream: Bloom filters



- Creates false positives but no false negatives
- If the item is in S we surely output it, if not we may still output it

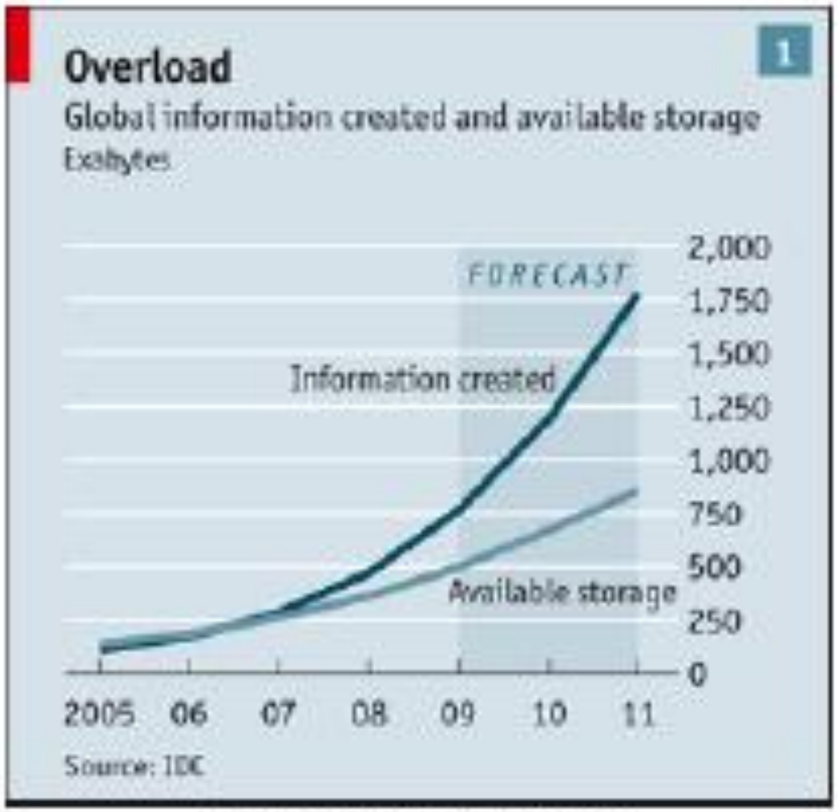
Counting distinct elements in data stream

- Data stream consists of a universe of elements chosen from a set

- Maintain a count of the number of distinct elements seen so far

- Flajolet-Martin Algorithm
  - For each stream element a, let r(a) be the number of railing 0s in h(a) say h(a) = 12, then 12 is 1100 in binary, so r(a) = 2
  - Record R = the maximum r(a) seen so far
  - Estimated number of distinct elements = 2^R

- **We are producing more data than we are able to store!**



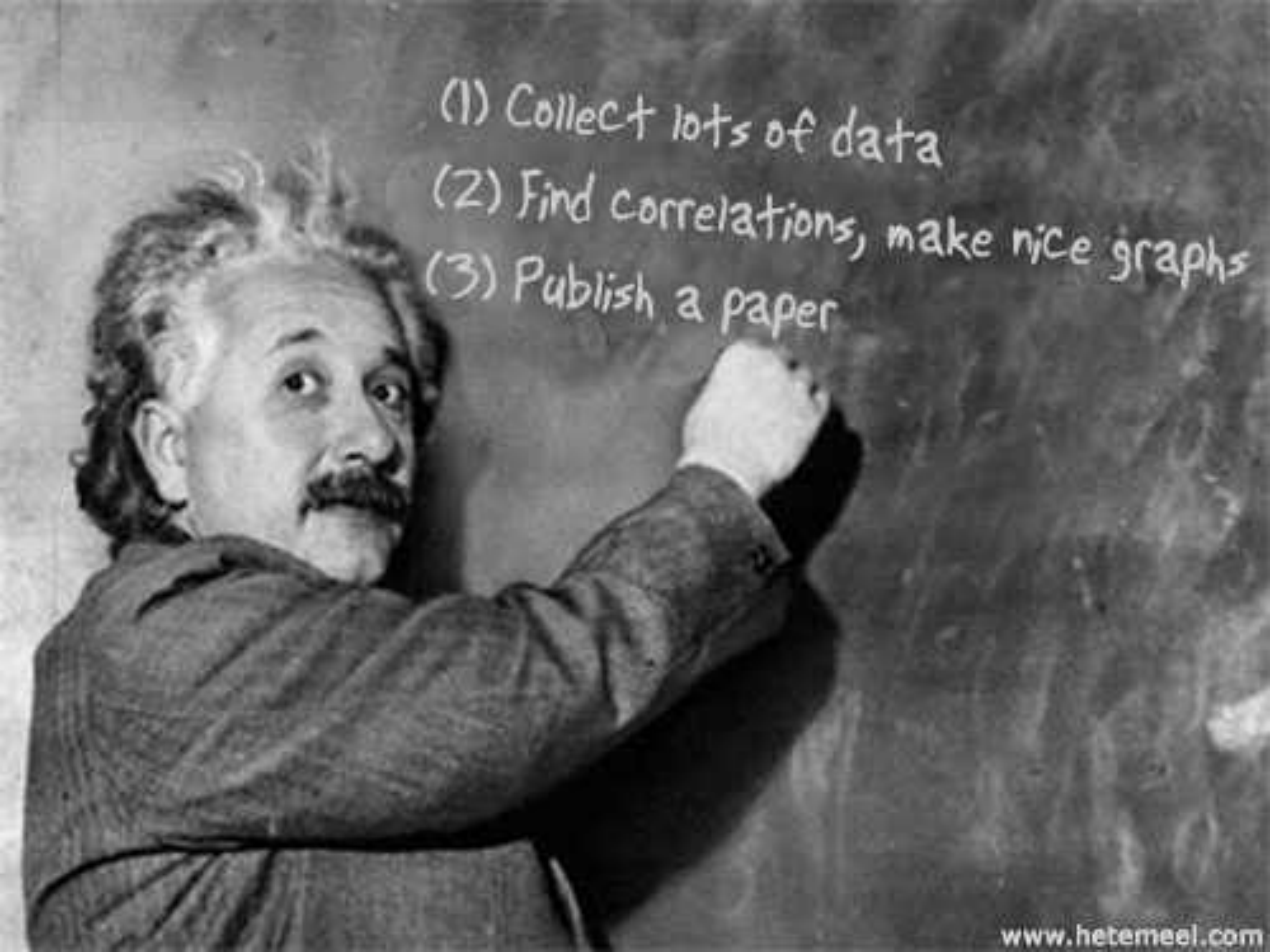
[The economist, 2010]


Spongecell
Trumba
eskobo
Pageflakes
vimeo
skobee
shadows
groovee
YouTube
Zimbra
FURL
smugmug
newsgator
Blogniscient
TinyFinger
shutterfly
PodDater
Feedster
favoor
ZAZZLE
Tailrank
TagWorld
nuvvo
dogear
yokalike
grouper
ODDPOST
iNods
Lulu
blish
flagr
FireAnt
simplyhired
Veoh
theadcloud
rbloc.com
cafepress
Renkoo
pixagogo
gather
browsr
oyogi
standpoint
meebo
last.fm
Jotspot
Frappr!
jeteye
dabble db
YEDDA
Writeboard
SHOUTWIRE
iKarma
AirSet
tech.memeorandum
CalendarHub
Suprglu
pando
zingtag
Findory
backfence
clipmarks
wayfaring
gOFFICE
AllPeers
Rallypoint
Zoozio
blogbeat
Ziggs
Zooto
vSocial
Boltfolio
wink
riya
Wordcast
Opinity
reddit
measuremap
gumshoe
bluepulse
STREAMLOAD
Ta-da Lists
FeedSky
jellyBam.INC
nativetext
CONGOO
PODZINGER
RSS MAD
FeedTier
phanfare
WIKIPEDIA
Fruitcast
PubSub
dPolls
flickr
Ning
Ookles
Strongspace
zoominfo
CASTPOST
yubnub
BLOOP
ProjectSpaces
FeedBurner
Bloglines
FOTOLOG
ourmedia
Yub.com
Spot Runner
myspace
NewsAlloy
Allmydata.com
gabbrcom
Gcast
blinkx
openomy
riffs
ajchat
jambo
ROLLYO
ClipShack
chatsum
PANDORA
looklater
PROTOPAGE
squish
WebJay
PLAZES
Noodly
30 BETA
wundir
diigo
box
Jots
Xdrive
vizu
digg
del.icio.us
AlmondRocks
Tagyu
30 Boxes
writely
Simpy
Gtalkr
TRUVEO
egoSurf
mozy
Quimble
pegasus
remember the milk
SQUIDOO
picturecloud
newsvine
Clipfire
Basecamp
measuremap
facebook
Netvibes
Goowy
My Video Karaoke
Lexxe alpha
LOOKSTER
PXN8
43 Things
Technorati
Platial
Backpack
GMail
yelp
Smarkets
inform
magnolia
ShoZu
feedmarker
filangy
mercora MusicSearch
Meet With Approval.com
HOMEPORTALS
Spin spy

Big Data trends

- The rise of the Industrial Internet
  - The faster you analyze data the great the predictive value
  - Using more diverse data, not just more data
- It is going to be cloudy: Big-Data-as-a-Service solutions
- Businesses get serious about big data privacy and security.
- Education and Medicine will be essential for success

How do you want that data?


(1) Collect lots of data
(2) Find correlations, make nice graphs
(3) Publish a paper
www.hetemeel.com

Tell me and I forget.
Show me and I remember.
Involve me and I understand.

Thank you!   Q&A