Advanced School on Data Exchange, Integration, and Streams - Dagstuhl, November 2010

QUERYING AND MINING DATA STREAMS

Elena Ikonomovska
Jožef Stefan Institute – Department of Knowledge Technologies

Outline

- Definitions
  - Datastream models
  - Similarity measures
- Historical background
- Foundations
  - Estimating the L_2 distance
  - Estimating the Jaccard similarity: Min-Wise Hashing
- Key applications
- Maintaining statistics on streams
  - Hot items
  - Some advanced results (Appendix)
    - Estimating rarity and similarity (the windowed model)
    - Tight bounds for approximate histograms and cluster-based summaries

Data stream models: Time series model

- A stream is a vector / point in space
- Items are arriving in order of their indices:

$$\vec{x} = \{x_1, x_2, x_3, \ldots\}$$

1	2	3	4
x_1	x_2	x_3	x_4

… coordinates of the vector

  - The value of the i-th item is the value of the i-th coordinate of the vector
- Distance (similarity) between two streams is the distance between the two points

Data stream models: Turnstile model

- Each arriving item is an update to some component of the vector:

1	2	3	4
10	5	24	12

$(2, 4) \Rightarrow$

1	2	3	4
10	9	24	12

$(2, x_2^{(5)})$ indicates the *5*-th update to the *2*-nd component of the vector

- *value:* $x_i = x_i^{(1)} + x_i^{(2)} + x_i^{(3)} \ldots$
- positive or negative update

- only nonnegative updates $\Rightarrow$ *cash register model*

L_p distances ($p \geq 0$)

- Stream 1 $\{x_1,x_2,x_3,\ldots\}$ & stream 2 $\{y_1,y_2,y_3,\ldots\}$ in $\{1,\ldots,m\}$

  $$L_p=\Sigma_i|x_i^p-y_i^p|^{1/p}$$

- L_0 distance (Hamming distance) $\Leftrightarrow$ the number of indices *i* such that $x_i \neq y_i$
  - A measure of dis(similarity) of two streams [CDI02]
- $L_\infty = \max_i|x_i - y_i|$
- $L_2=\Sigma_i|x_i^2-y_i^2|^{1/2}$ distance
  - L_2 norm (f_2^2)- for approximating self-join sizes [AGM'99] $Q = COUNT(R \bowtie_A R)$ $|dom(A)| = m$

Basic requirements

- Naïve approach: store the points/vectors in memory and compute any distance/similarity measure or a statistic (norm, frequency moment)
- Typically:
  - Large quantities of data – single pass
  - Memory is constrained – $O(\log m)$
  - Real-time answers – linear time algorithms $O(n)$
- Allowed approximate answers (ε, δ)
  - ε & δ are user-specified parameters

Historical background

- [AMS'96] approximate F_2 (inserts only)
  - [AGM'99] approximate L_2 norm (inserts and deletes)
- [FKS'99] approximate L_1 distance
  - [Indyk'00] approximate L_p distance for $p \in (0,2]$
    - p-stable distributions (Caushy is 1-stable, Gaussian is 2-stable)
- [CDI'02] efficient approximation of L_0 distance
- Approximate distances on windowed streams
  - [DGI'02] approximate L_p distance
  - [Datar-Muthukrishnan'02] approximate Jaccard similarity

Estimating the L_2 distance [AGM'99]

- Data streams $(x_1, x_2 \ldots, x_n)$ and $(y_1, y_2 \ldots y_n)$
- For each $i = 1, 2, \ldots n$ define a i.i.d. random variable X_i $P[X_i = 1] = P[X_i = -1] = 1/2 \;\; \Rightarrow \;\; E[X_i]=0$
- Base idea: Simply maintain $\Sigma_{i=1,\ldots,n} X_i(x_i - y_i)$
  - For some i, j *and items* $(i, x_i^{(i)})$, $(i, y_i^{(i)})$:
    - $X_i \cdot x_i^{(i)}$ is added and $X_i \cdot y_i^{(i)}$ is subtracted

$$E[(\Sigma_{i=1,\ldots,n} X_i(x_i - y_i))^2] =$$

$$E[\Sigma_{i=1,\ldots,n} \overset{1}{X_i^2}(x_i - y_i)^2 + \Sigma_{i \neq j} \overset{0}{X_i X_j}(x_i - y_i)(x_j - y_j)] =$$

$$\Sigma_{i=1,\ldots,n}(x_i - y_i)^2$$

- The problem amounts to obtaining an <u>unbiased estimate</u>

Standard boosting technique

- Run the algorithm in parallel $k=\theta(1/\varepsilon^2)$ times
  1. Maintain sums $\Sigma_{i=1,..,n} X_i(x_i - y_i)$ for k different random assignments for the random var. $\Rightarrow$ $\mathbf{X_{i,k}}$
  2. Take the average of their squares for a given run *r*
     $\Rightarrow v^{(r)}$ (reduce the variance/error!) Chebyshev
  3. Repeat the procedure $l = \theta(\log(1/\delta))$ times $\Rightarrow$ $\mathbf{X_{i,k,l}}$
  4. **Output the median over $\{v^{(1)}, v^{(2)}, \ldots, v^{(l)}\}$** Chernoff
  5. Maintains *nkl* values in parallel for the random variables

Result

The Chebyshev inequality + Chernoff:

$\Rightarrow$ this estimates the square of L_2 within $(1 \pm \varepsilon)$ factor with probability > $(1 - \delta)$

- Random variables needed: nkl !
- The random variables can be four-wise independent
  - This is enough so that Chebyshev still holds [AMS'96]
  - pseudorandomly generated on the fly $\Rightarrow$
  - $O(kl) = O(1/\varepsilon^2 \log(1/\delta))$ words + a logarithmic-length array of seeds $O(\log m)$

Estimating the L_p distance

- *p-stable distributions* [I'00]

  D is a p-stable distribution if:

  - For all real numbers $a_1, a_2, \ldots, a_k$

  If $X_1, X_2, \ldots, X_k$ are i.i.d. random var. drawn from **D**

  $\Rightarrow$ $\boldsymbol{\Sigma a_i X_i}$ has the same distribution as $\boldsymbol{X(\Sigma_i |a_i|^p)^{1/p}}$

  for random variable X with distribution **D**

- Cauchy distribution is 1-stable $\Rightarrow$ L_1
- Gaussian distribution is 2-stable $\Rightarrow$ L_2

The algorithm

$z_1, z_2, \ldots z_n$ is the stream vector

- Again… run in parallel $k=\theta(1/\varepsilon^2 \log(1/\delta))$ procedures & maintain sums $\boldsymbol{\Sigma_i z_i X_i}$ **for each run 1,…k**
- The value of $\boldsymbol{\Sigma_i z_i X_i}$ in the *l*-th run is $\boldsymbol{Z^{(l)}}$
  - $Z^{(l)}$ is a random variable itself
  - Let **D** is *p*-stable:

    $$Z^{(l)} = X^{(l)} (\Sigma_i |z_i|^p)^{1/p}$$

    for some random variable $X^{(l)}$ drawn from D

Estimating the L_p distance cont.

- The output is:

  $(1/\gamma)\textit{median}\{|Z^{(1)}|, |Z^{(2)}|,\ldots, |Z^{(k)}|\}$

  - where γ is the median of $|X|$, for X random variable distributed according to **D**
  - **Chebyshev:** This estimate is within a multiplicative factor $(1\pm\varepsilon)$ of the true norm with probability $(1-\delta)$
- Observation [CDI'02]:
  - L_p is a good approximation of the L_0 norm for *p* sufficiently small
  - $p=\varepsilon/\log(m)$ where m is the maximum absolute value of any item in the stream

The Jaccard similarity

$S_A=\{a_1,a_2,..a_n\}$ $S_B=\{b_1,b_2,...,b_n\}$

- Let A (and B) denote the set of distinct elements

  $|A \cap B|/|A \cup B|$ = Jaccard similarity

- Example: (view sets as columns) m=6

	A	B	
$item_1$	0	1	**$\|A \cup B\|=5$**
$item_2$	1	0	
	1	1	**simJ(A,B) = 2/5 = 0.4**
	0	0	
	1	1	
$item_6$	0	1	

Signature idea

- Represent the sets A and B by signatures Sig(A) and Sig(B)
  - Compute the similarity over the signatures
  - E[simH(Sig(A),Sig(B))]=simJ(A,B)
- Simplest approach
  - Sample the sets (rows) uniformly at random k times to get **k-bit signature Sig** (instead of m bits)
  - Problems!
    - Sparsity – sampling might miss important information

Tool: Min-Wise Hashing

- π - randomly chosen permutation over $\{1,\ldots,m\}$
- For any subset $A \subseteq [m]$ the *min-hash* of A is:
  - $h_\pi(A) = \min_{i \in A}\{\pi(i)\}$
  - Index of the first row with value 1 $\Leftrightarrow$ random permutation of the rows
  - One bit of the k-bit signature of A, Sig(A)
- When π is chosen uniformly at random from the set of all permutations on [m] for any two subsets A,B of [m] then:

$$\mathbf{Pr[h_\pi(A) = h_\pi(B)] = |A \cap B| / |A \cup B|}$$

Example

- Consider the following permutations: for m=5

  k=1 $\pi_1 = (1\ 2\ 3\ 4\ 5)$
  k=2 $\pi_2 = (5\ 4\ 3\ 2\ 1)$
  k=3 $\pi_3 = (3\ 4\ 5\ 1\ 2)$

- And the sets: $A = \{1,3,4\}$ $B = \{1,2,5\}$

  The **min-hash values** are as follows:

  k=1 $h\pi_1(A) = \mathbf{1}$ $h\pi_1(B) = \mathbf{1}$
  k=2 $h\pi_2(A) = 4$ $h\pi_2(B) = 5$
  k=3 $h\pi_3(A) = 3$ $h\pi_3(B) = 5$

$\Rightarrow$ the <u>expectation</u> of the fraction of permutations where min-hash values agree is simJ(A,B)

Estimation of Jaccard similarity

- To get a good estimate of the expectation ⇒
- Run the procedure multiple times (k) in parallel
  - Choose independently k random permutations: $\pi_1, .. \pi_k$
  - Count number of agreements: $|\{j: h_{\pi_j}(A) = h_{\pi_j}(B)\}|$
  - **Output the fraction!**

How many times is good enough?

Lemma

[Datar-Muthukrishnan'02]

Let $\{h_1(A), h_2(A),\ldots,h_k(A)\}$ and $\{h_1(B), h_2(B),\ldots,h_k(B)\}$ be **k** independent **min-hash values** for the sets A and B respectively

Let S(A,B) be the fraction of the min-hash values that they agree on:

$$S(A,B) = |\{j \mid 1 \le j \le k, h_j(A) = h_j(B)\}| / k$$

- For $0 < \varepsilon < 1$, and $k = O(\varepsilon^{-3} \log 1/\delta)$ with success probability at least $1 - \delta$

$$\mathbf{S(A,B) \in (1 \pm \varepsilon) |A \cap B| / |A \cup B|}$$

The algorithm

- Choose k min-hash functions $h_1, h_2, \ldots h_k$ randomly
- Maintain $h_i^*(t) = \min_{a_j, j \leq t} h_i(a_j)$ at every time t
  - For each new a_{t+1} compute the hash value $h_i(a_{t+1})$ under the corresponding permutation *I* (1,..k) and compare with $h_i^*(t)$
  - If $h_i(a_{t+1}) < h_i^*(t)$ update the min-hash value

Storing one π takes O(m log m) space!

$\Rightarrow O(km \log m) = O(\varepsilon^{-3} \log 1/\delta \; m \log m)$

Approximate min-wise hashing

- It suffices to use approximately min-wise independent hash functions (introduces additional error)
- For any hash function **h** chosen randomly from the family of ϵ'*-min-wise independent* functions

  $$Pr\,[h(A) = h(B)] = |A \cap B| / |A \cup B| \pm \epsilon'$$

  $$\mathbf{S(A,B) \in (1 \pm \varepsilon) |A \cap B| / |A \cup B|} \pm \epsilon'$$

  - very efficient in terms of space: $O(\log(1/\epsilon') \log m)$
  - each hash function takes: $O(\log(1/\epsilon'))$ time
- The Lemma still holds, but **k** has to be adjusted

Key applications

- Tracking network traffic
  - Measure and detect large changes
- Query optimization
  - L_2 norm to approximate self-join sizes / for selectivity estimation
  - L_0 norm number of distinct elements
- Genetic data
  - Similarity of two base-pair sequences
- Data mining:
  - Identifying similar entities (purchases, phone calls, IP addresses, Web page visits, bank transactions)

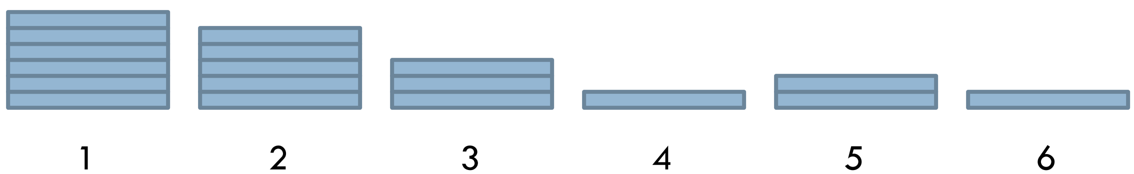
What's Hot and What's Not!

- Problem definition [Cormode-Muthukrishnan'05]
  - What is a hot item?
  - How to dynamically maintain a set of hot items under the presence of delete and insert transactions?
- Preliminaries
  - Lemma on the space lower bound
- Group testing : 2 methods proposed
  - Non-adaptive method
- Results
- Applications - measure of the skew of the data/ iceberg aggregate queries, outliers detection

Hot items

A sequence of n transactions on items, ID's $\in [1, m]$ $m = 6$

1,2,1,3,4,5,1,2,2,3,1,1,3,5,2,6,1,2,…   (turnstile model)



- $n_x(t)$ = #inserted - #deleted $f_x(t) = n_x(t)/\Sigma_{y=1,m} n_y(t)$

  $f_x(t) > 1/(k+1) \Rightarrow$ *hot item*

k=3

$f_1(t)=6/18=1/3 > 1/4$

$f_2(t)=5/18 > 1/4$

$f_3(t)=3/18=1/6$

hot items are only {1,2}

Preliminaries

- If allowed O(m) space (simple heap data structure)
  - Each insert/delete will take O(log m) time
  - All k hot items: O(k log m) time in the worst case
- BUT … if we are to use less than $\Omega(m)$ space:
  - Only approximate answers are possible (ε, δ)!
  - We can guarantee (with success probability $1 - \delta$) that ALL HOT items are output and NO item which has frequency less than $1/(k+1) - \varepsilon$
- **Lemma**: Any algorithm which guarantees to find ALL AND ONLY items which have frequency greater than $1/(k+1)$ must store $\Omega(m)$ bits

Proof (from information theory)

- Let $S \subseteq [1\ldots m]$
  - Transform into a sequence of $n = |S|$ insertions of items
  - x is included only once if and only if $x \in S$
- Insert $\lfloor n/k \rfloor$ copies of x
  - If $x \notin S \Rightarrow$

  $\lfloor n/k \rfloor / (n+\lfloor n/k \rfloor) = \lfloor n/k \rfloor / \lfloor n(k+1)/k \rfloor \leq \lfloor n/k \rfloor / (k+1)\lfloor n/k \rfloor = 1/(k+1)$ x is not output

  - If $x \in S \Rightarrow$

  $(\lfloor n/k \rfloor + 1)/(n+\lfloor n/k \rfloor) > (n/k)/(n+n/k) = 1/(k+1)$ x is output

  So, you can determine whether $x \in S$ or not!

  The set S can be extracted $\Rightarrow$ must store $\Omega(m)$ bits

Puzzle (adaptive GT)

- A man has m coins, where $m = 3^x$, $x > 0$
  - One is slightly heavier than others
- What is the <u>minimum</u> number of weightings with a balance pan required to find the heavier coin?
  - How many coins do we put on each side?
    - Obviously a same amount q ($\leq m/2$)
  - If we place **q** coins on each side:
    - Tip $\Rightarrow$ eliminate all but **q** coins
    - Not tip $\Rightarrow$ eliminate **m-2q** coins
  - **m/2** or **m/3**?
    - Going to m/3
    - Cannot eliminate more than 2m/3!
  - Result: $x = log_3(m)$

Nonadaptive group testing

- Divide all *m* items up into several <u>overlapping</u> groups
  - Each item x is included in several groups
  - Each group is associated with a counter
  - For an insertion of x increment the counters of all groups where it belongs, for a deletion decrement
  - "Weight" each group of items (test each counter) to identify if the group contains a hot item or not (if the set counter exceeds a certain threshold)
- How many groups? (<< m)
- How to represent them in a concise way?
- How to form the tests to obtain the hot items from the results efficiently?

Find the Majority Item (k=1)

- Maintain $\lceil \log_2 m \rceil + 1$ counters : c[0],c[1],...,c[log m]

  bit(x, j) – value of *j-th* bit of the binary representation

  x=13 *bin*: $1101 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

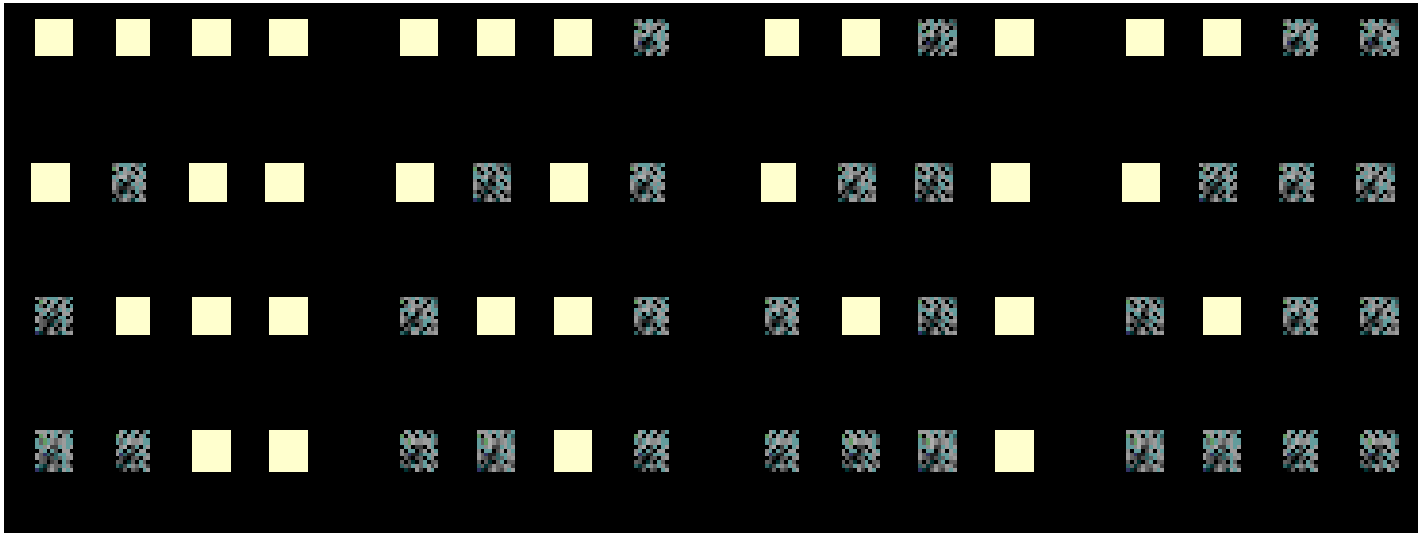
  bit(13, 0)=1, *bit*(13, 1)=0, *bit*(13, 2)=1, …

  d=1 insertion, d=-1 deletion

  - $c[0] \leftarrow c[0] + d$ (how many items are "live")
  - $c[j] \leftarrow c[j] + bit(x, j) \cdot d$ ⇒ *takes **O(log(m))** time*
- The majority *item (if any)* ⇒ $\sum_{j=1,\dots \log(m)} 2^j \, \boldsymbol{gt}(c[j], c[0]/2)$
  - *Deterministic : time **O(log(m))***
- *It there is no majority item it is not possible to distinguish the difference (based on the information stored)*

Illustration

- m=16
- We need 4+1 = 5 counters in total



Finding k Hot items

- To locate k items among m locations : $\log_2 \binom{m}{k} \geq k \log_2(m/k)$
- Suppose a group of items that happened to contain only one hot item
  - Split the group on (*log(m)*) subgroups each associated with a counter
  - Apply the previous algorithm to identify the hot item!
- To identify k hot items ⇒ construct TxW groups
  - For concise representation : Use T hash functions (representation in O(log m) space)

    $$f_{a,b}(x) = ((ax + b) \bmod P) \bmod W, \quad P > m > W$$

  - a and b are drawn randomly from [0 … P-1]

Guarantees

- For appropriate choices of T and W we can:
  1. Ensure that all hot items are being output
  2. Ensure that no items are output which are "far" from being hot
- How?
  1. Using properties of hash functions [Carter-Wegman'79]

Over all choices of a and b, for $x \neq y$,

$$\Pr[f_{a,b}(x) = f_{a,b}(y)] \leq 1/W$$

Update & Test

- TxW number of groups, each split into log(m) subgroups
  - log(m)+1 counters per group ⇒ O(TW log(m)) space
  - T hash functions that map item x onto 0…W-1
  - A group represents the items which are mapped to the same hash value {0 …W-1} by a particular hash function h_i
- Update counters: c[1][0][0] ⟶ c[T][W-1][log m]
  - For i ← 1 to T : Update array $c[i][h_i(x)]$ as previously
  - Update time is now O(T log(m))
- Test: If a group counts more than n/(k+1) items then might contain a hot item
  - Further verification is carried out for each hot item found
  - The search time is $O(T^2 \cdot W \cdot \log(m))$ – a scan of the whole data structure + a check on the hot item

Theorem

- With probability of at least **$(1 - \delta)$** we can find all hot items whose frequency is > **$1/(k+1)$**, and given **$\varepsilon \leq 1/(k+1)$** with probability of at least **$1 - \delta/k$** each item which is output has frequency of at least **$1/(k+1) - \varepsilon$**
  - Using space $O(\log(k/\delta)\ 1/\varepsilon \log(m)) = O(k \log(k) \log(m))$
  - Update time $O(\log(k/\delta) \log(m)) = O(\log(k) \log(m))$
  - Query time $O(\log^2(k/\delta)\ 1/\varepsilon \log(m)) = O(k \log^2(k) \log(m))$
- This follows by setting $W \geq 2/\varepsilon$ and $T = \log(k/\delta)$ + applying 2 other lemmas

Summary (Take Home)

- Intro to data stream models
- The concept of random linear sketches for obtaining reliable (ε,δ) estimates of L_p distances/norms
- Efficient algorithms based on:
  - Min-wise hashing (Jaccard similarity + rarity)
  - The concept of group testing for estimating HOT items in a stream
- Estimating rarity and similarity in a windowed data stream model
- Tight bounds for approximate histograms and the k-center problem

References

- [CDI02] Comparing data streams using Hamming norms (How to zero in)
- [AGM'99] Tracking join and self-join sizes in limited storage
- [Indyk'00] Stable distributions, pseudorandom generators, embeddings, and data stream computation
- [DGI'02] Maintaining stream statistics over sliding windows
- [Vee'09] Stream Similarity Mining
- [Datar-Muthukrishnan'02] Estimating Rarity and Similarity over Data Stream Windows
- [Cormode-Muthukrishnan'05] What's hot and what's not: tracking most frequent items dynamically
- [Guha'09] Tight results for clustering and summarizing data streams
- [Guha-Shim'07] A note on linear time algorithms for maximum error histograms
- [BSS'07] Space efficient streaming algorithms for the maximum error histogram
- [GKS'06] Approximation and streaming algorithms for histogram construction problems
- [Carter-Wegman'79] Universal classes of hash functions

Appendix

Estimating rarity and similarity in the windowed model [Datar-Muthukrishnan'02]

Advanced results from the paper of [Guha'09]

Some advanced topics

- Rarity (Appendix)
  - Definition
  - Base ideas
  - Estimating rarity in the unbounded stream model
- Estimating rarity and similarity in the windowed stream model (Appendix)
- Clustering and summarizing (Appendix)
  - Definitions / Preliminaries
  - Some very tight bounds

Rarity

- An item is α-rare for integer α if it appears precisely α times
  - $\#\alpha$-*rare* number of such items in the window
  - $\rho_\alpha = \#\alpha\text{-}rare / \#distinct$ (α-*rarity*)

S={2,3,2,4,3,1,2,4}	D={1,2,3,4}
1-rare={1}	1-rarity=1/4
2-rare={3,4}	2-rarity=1/2
3-rare={2}	3-rarity=1/4

Base ideas

- R_α - set of α-rare items
- D - set of distinct items
- 2 main observations:

1. $R_\alpha \subseteq D$

   $\Rightarrow |R_\alpha \cap D| / |R_\alpha \cup D| = |R_\alpha| / |D|$

   - Rarity is the fraction of the time min-hash functions for R_α and D have agreed upon
2. $h(R_\alpha)=h(D)$ iff the item in D belongs to R_α
   - Need to maintain the min-hash values <u>only for D</u>

Lemma [Datar-Muthukrishnan'02]

- Let ρ_α' be the fraction of counters $c_i(t)$ that eq. α :

$$\rho_\alpha'(t) = |\{l \,|\, 1 \le l \le k,\ c_i(t) = \alpha\}| / k$$

For $0 < \varepsilon < 1$, $0 < p < 1$ and $k \ge 2\varepsilon^{-3}p^{-1}\log\delta^{-1}$

$$\rho_\alpha'(t) \in (1 \pm \varepsilon)\rho_\alpha(t) + \varepsilon p$$

with success probability at least $1 - \delta$

- Why?

$\Pr[c_i(t) = \alpha] = \Pr[h_i^*(t) = h_i(x) \,|\, x \in R_\alpha] = |R_\alpha(t)| / |D_t|$

- α can be chosen at query time

The windowed data stream model

- Consider the window of the last N observations:

  $a_{t-100}, a_{t-99}, \ldots, a_{t-(N-1)}, a_{t-(N-2)}, \ldots, a_{t-2}, a_{t-1}, a_t$

- The data changes over time
  - Interest over the "recently observed" data elements
  - Eg. How many distinct customers made a call through a given switch in the past 24 hours?
- We cannot store the entire window in memory

  $\{12,89,23,45,34\}$ min=12 ⇒ $\{89,23,45,34,58\}$ min=23

  We need to store each item in the window!
  - Applications: sensor networks, switches, Internet routers,..
  - Computing most functions exactly is impossible

Estimating similarity - windowed

- Maintain k min-hash values for A and B
  - σ – the fraction of min-hash values they agree on
- How to maintain min in a window?
  - d_1, d_2 are items arrived at times t_1 and t_2 ($t_1 < t_2$)
  - If $h_i(d_1) \geq h_i(d_2)$ d_2 *dominates* d_1
  - When both are <u>active</u> the minimum $h_i^*(t)$ is not affected by $h_i(d_1)$ $\Rightarrow$ no need to store $h_i(d_1)$
- For each min-hash function maintain a list:

  $L_i(t) = \{(h_i(a_{j1}), j_1), (h_i(a_{j2}), j_2), \ldots (h_i(a_{jl}), j_l)\}$
  - $j_1 < j_2 < \ldots < j_l$ & $h_i(a_{j1}) < h_i(a_{j2}) < \ldots < h_i(a_{jl})$
  - $h_i^*(t) = h_i(a_{j1})$

Estimating similarity cont.

- Memory allocated $|L_i(t)|$ at time t

10	11	12	13	14	15	16	17	18	19	20
20	12	75	26	23	20	15	29	40	45	32

Min-hash list:

11	16	17	20
12	15	29	32

- With high probability, over the choice of min-hash function h_i, expected $|L_i(t)| = \Theta(\log N)$
  - N is the size of the window
  - $O((\log N)(\log u))$ bits of space
  - $O(\log \log N)$ time per data item

Estimating rarity - windowed

- Keep a **linked-list of "dominant" min-hash** values
  - But since now we need to find α **instances** of an item, we keep several arrival times of the item

    $L_i(t) = \{(h_i(a_{j1}), List^t_{i,j1}), (h_i(a_{j2}), List^t_{i,j2}),\ldots, (h_i(a_{jl}), List^t_{i,jl})\}$

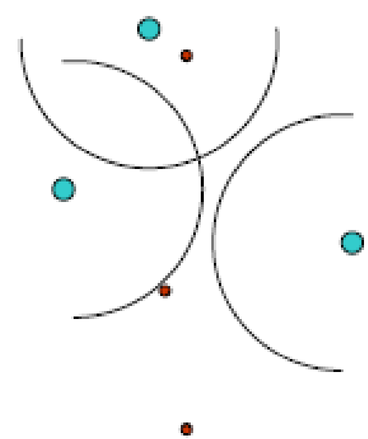
  - Where $List^t_{i,j1}$ is an ordered list of the last α instances mapped to the hash value $h_i(a_{j1})$
  - Concatenate: $List^t_{i,j1} + List^t_{i,j2} + \ldots + List^t_{i,jl} \Rightarrow$ indexes strictly increasing
  - Count the fraction of $List^t_{i,j1}$ over all i that have α elements and agree on the minimal hash value
- The total size of $L_i(t)$ is $O(\alpha \log N)$ with high probability

Clustering and summarizing

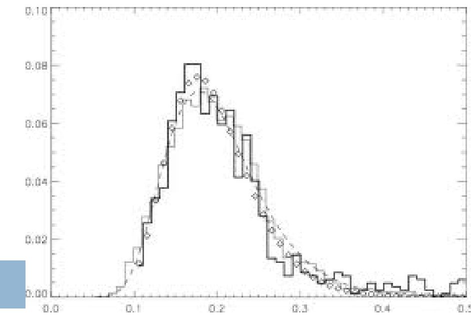
- Definitions
- Preliminaries (the main ideas)
- “Streamstrapping”
- Upper bounds & lower bounds
- <u>Results:</u>
  - Guarantees
  - Applications
    - MinMax objectives
    - MinSum objectives

K-center clustering

- Given n points identify K centers such that the maximal distance for each point from its closest center is minimized
  - Find the smallest radius ε* such that if disks of radius ε* are placed on the chosen centers then every input point is covered
  - Assume an oracle distance model
    - Useful for more complex types of data



Histograms



- Approximate a data distribution using a fixed amount of space while minimizing the overall error
- Given a sequence of *n* numbers $x_1,..,x_n$
  - Construct a piecewise constant representation H with at most B pieces (buckets)
  - The values in a single bucket are estimated using a single value ⇒ we suffer an error
  - Choose the buckets such that an objective function f(X,H) is minimized
    - f(X,H) can be the squared (VOPT) or the maximum error…

Preliminaries - 3 main ideas

- "Thresholded approximation"
  - If there exists a solution of size B' and error ε then we can construct a summary of at most B' such that the error is at most αε (where α ≥ 1)
  - Otherwise, no solution with error ε exists ("fail")
- Run multiple copies (controlled in number) of the algorithm <u>for different estimates of the error</u> ε
  - Try with several values
  - If ε is too small the algorithm will return "fail"
  - Restart with a bigger error estimate
- "Streamstrapping" - bootstrapping streams
  - Use the summarization results from the previous run

“Streamstrapping“ [Guha’09]

- Use a property of *metric errors:*
  - Let $\varepsilon(X,H)$ be summarization error for X using the summary H
  - Let $X_t \circ Y$ a concatenation of input X_t followed by Y
    - Y is $X_t \setminus X_{t-1}$ that is $X_t = X_{t-1} \circ Y$
  - Let $X(H_t)$ is the summarized input X_t using H_t

  $\varepsilon(X_t \circ Y,\ H_t)$ is in the range: $\varepsilon(X(H_{t-1}) \circ Y,\ H_t) \pm \varepsilon(X_{t-1}, H_{t-1})$
- Informs on the correct level of detail we need to be investigating the data

Upper bounds

- when input $\ldots x_i \ldots$ is presented in increasing order of i
- Any $(1+\epsilon)$ approximation algorithm requires:
  - $O((B/\epsilon)\log(1/\epsilon))$ space for maximum error histogram
  - $O((B^2/\epsilon)\log(1/\epsilon))$ space for VOPT error histogram
  - Running time is O(n) plus smaller order terms
- Any $2(1+\epsilon)$ approximation algorithm requires:
  - $O((k/\epsilon)\log(1/\epsilon))$ space for the k-center problem
- First results (for the space bound) that are non-dependent on: the size of the stream N, the precision M, nor the optimal solution ε^*

Lower bounds

- The minimal space that has to be used in order to provide some approximations
- For maximum error histograms: for all $\epsilon \leq 1/(40B)$
  - Any $(1+\epsilon)$ approximation must use $\Omega(B/(\epsilon \log(B/\epsilon)))$ bits of space
  - <u>The first lower bound stronger than $\Omega(B)$</u>
- For k-center single pass deterministic algorithm: for all $\epsilon \leq 1/(10k)$
  - $(2+\epsilon)$ approximation has to store $\Omega(k^2)$ points

The StreamStrap Algorithm

1. Read the first B items in the input. Keep reading as long as the error is 0
2. At the first input that causes a non-zero error $\varepsilon_0 \Rightarrow$ Run $J = O((1/\epsilon) \cdot \log(\alpha/\epsilon))$ copies of the algorithm
   - Each for error $\varepsilon = \varepsilon_0, (1+\epsilon)\varepsilon_0, \ldots (1+\epsilon)^J\varepsilon_0$
3. At some point (for some ε) the algorithm will return "fail", so we know that $\varepsilon^* > \varepsilon$.
   - We terminate the copies for all $\varepsilon' < \varepsilon$ and restart with $(1+\epsilon)\varepsilon'$ <u>using the summarization of ε'</u>
4. Repeat step 2 until end of input

Guarantees

- The answer corresponds to the lowest estimate ε for which a copy of the thresholded algorithm is still running
- If a "thresholded" approximation exists for any $\epsilon < 1/10$
  - The algorithm provides a $\alpha/(1-3\epsilon)^2$ approximation
  - The running time is the time to run $O((1/\epsilon) \cdot \log(\alpha/\epsilon))$ copies of the thresholded algorithm + $O((1/\epsilon) \cdot \log(\alpha\varepsilon^* M))$ initializations

Upper bounds: k-Center

Use the previous guarantees…

- A single pass $2+\epsilon$ approximation for K center problem using
  - $O((K/\epsilon)\log(1/\epsilon))$ space and
  - $O((Kn/\epsilon)\log(1/\epsilon)+(K/\epsilon)\log(M\varepsilon^*))$ time
  - when the points are input in an arbitrary order
- The radius of any cluster is $\pm\epsilon\varepsilon^*$ of the true radius of that cluster using the same center

Upper bounds: Max Error Histogram

- A single pass $1+\epsilon$ streaming approximation for B bucket histogram construction using
  - $O((B/\epsilon)\log(1/\epsilon))$ space and
  - $O(n+(B/\epsilon)\log^2(B/\epsilon)\log(M\varepsilon^*))$ time
  - the input $\ldots x_i \ldots$ is presented in increasing order of i
  - Based on the "thresholded" optimum algorithm [Guha-Shim'07]
- The error of any bucket found is $\pm\epsilon\varepsilon^*$ of the true error of that bucket

Upper bounds – VOPT histogram

- A single pass $1+\epsilon$ streaming approximation for best B-bucket histogram for VOPT error using
  - $O((B^2/\epsilon)\log(1/\epsilon))$ space and
  - $O(n+(B^3/\epsilon^2)\log^2(B/\epsilon)\log(M\varepsilon^*))$ time
  - the input $\ldots x_i \ldots$ is presented in increasing order of i
  - Based on AHIST-B [GKS'06]
- A similar result for the K-median problem
  - Minimize $\sum$ of distances of all points to their closest centers