Streaming II: Norm estimation

Piotr Indyk

The problem

- Frequency moments:
  - Elements $i_1, i_2 \ldots i_n \in \{1 \ldots M\}$ come in a stream S
  - The stream S defines a vector $x = x(S)$, such that x_i is the number of occurrences of i in S
  - The k-th frequency moment $F_k(S) = \sum_i x_i^k$

- Norm estimation:
  - The stream consists of pairs (i_j, a_j), where a_j is the "multiplicity" of i_j
  - We extend $x = x(S)$:

$$x_i = \sum_{(i,a) \in S} a$$

  - The l_k norm of x is just

$$\|x\|_k = (\sum_i |x_i|^k)^{1/k}$$

Is it useful for anything ?

- $k = 0$
  - $F_0(S)$ is equal to the number of distinct elements in S (need to assume $0^0 = 0$)
  - $\|x(S)\|_0^{0/1} = \sum_i |x_i|^k$ is equal to the number of non-zero elements in $x(S)$
    Note: the ugly exponent $0/1$ is used to trump the equally ugly exponent $1/0$ in the definition of $\|\cdot\|_0$
  - Assume the elements i can be inserted or deleted from a database
  - Representation via S: insertion $(i, 1)$, deletion $(i, -1)$
  - $\|x(S)\|_0^{0/1}$ maintains the *current* number of distinct elements

- $k = 1$: $F_1(S) = n$

- $k = 2$: self-join size, an important quantity in DB

Example: Syntactic Clustering

- Finding similar documents:
  - Document $\rightarrow$ set of words
  - Find all pairs A, B such that $\frac{|A \cap B|}{|A \cup B|}$ large
- [BGMZ'97]:
  - Find all intersecting pairs $< A, B >$
  - Keep only pairs A, B such that $\frac{|A \cap B|}{|A \cup B|}$ is large

Algorithm details

- List all pairs $< DOC, WORD >$ s.t. DOC contains $WORD$
- Sort by $WORD$

  $< W_1, D_1 >, < W_1, D_{20} >, ..., < W_2, D_{100} >, < W_2, D_{110} >$
- Generate all triples $< DOC_1, DOC_2, WORD >$ s.t. $WORD$ appears in both DOC_1 and DOC_2
  Note: this is precisely the self-join of the $< DOC, WORD >$ relation.
- Group the triples according to $< DOC_1, DOC_2, \cdot >$

  $< D_1, D_2, W_1 >, < D_1, D_2, W_3 >, ..., < D_1, D_3, W_3 > ...$
- Compute the intersections

How to estimate the number of triples ?

- Sorting by $WORD$ - space/time consuming
- Instead:
  - For each $< DOC, WORD >$, output $WORD$ into stream S
  - $x(S)_{WORD}$ holds the number of documents containing $WORD$
  - The number of triples is equal to

$$\sum_{WORD} x(S)^2_{WORD}$$

    (or less, if we ignore symmetric triples)
  - Can compute $F_2(S)$ in one pass
- This allows us to estimate the running time of the syntactic clustering algorithm, without actually running it

- If the (estimated) running time is too high, we need to readjust some parameters, e.g.,remove frequently occurring words or use sentences instead of words, to reduce the number of triples

Motivation, ctd.

- $\|x(S)\|_\infty$ - the frequency of the most frequently occurring element in the stream

Algorithms

- From Lecture 1 we know the exact $F_0(S)$ is hard to compute
- ..but can be approximated up to any $(1+\epsilon)$
- Unfortunately, $\|x(S)\|_\infty$ is hard to approximate (next slide)

$\|x(S)\|_\infty$ hard to approximate

Theorem: In order to distinguish between $\|x(S)\|_\infty$ equal to 1 or 2, any deterministic* algorithm needs $\Omega(n)$ bits of space.

Sketch of the proof:

- By contradiction, assume there is a *deterministic* algorithm A using $M = o(n)$ bits of space
- Take any $S \subset \{1 \ldots m\}$; $n \le m$.
- Feed S to A; denote the state of A at that point by X
- We can now check if $i \in S$ for any $i \in \{1 \ldots M\}$
  - Feed A with i
  - The max occurrence increased to 2 iff $i \in S$
- This means we can recover S from X $\Rightarrow$ the number of different X's must be at least 2^m

Time for good news

Theorem: There is a constant B such that for any $\epsilon, \delta > 0$ and any $x \in \Re^d$, if we take a $l \times d$ matrix A with entries drawn independently from $N(0,1)$ where $l = B\log(1/\delta)/\epsilon^2$, then

$$median(|Ax|)/m = \|x\|_2(1 \pm \epsilon)$$

with probability at least $1 - \delta$.

Fact: The median estimator can be used to estimate $\|x\|_p$, $p \in (0,2]$ (by using stable distributions).

Conclusion: If we can maintain Ax under changes to x, we can use it to estimate $\|x\|_p$ using only $O(l)$ space

Algorithm for norm estimation

- Algorithm:
  - Initialize $z = (z^1 \dots z^l) = 0$;
    we will maintain $z = Ax(S)$
  - For each (i, a) in S, $z^j = z^j + a \cdot A[j, i]$
  - At the end, retrieve $\|x(S)\|_p$ from z
- Space: $O(l)$ numbers.

What about $A[j,i]$'s ?

Answers:

- Creative accounting: not really a "space", contains only random bits
- Practical: $A[j,i] = YFPRG(j,i)$
- Theoretical: there is a mapping $Nisan : \{0,1\}^{S \log R} \rightarrow \{0,1\}^R$, such that any algorithm that uses R random bits and S bits of space, works "as well" when given pseudorandom bits $Nisan(seed)$, where $seed$ is a random string of $S \log R$ bits.
  - Our algorithm uses small space S and $R = n^{O(1)}$ random bits
  - Can choose and store random $seed$ of length $S \log R$
  - Use $Nisan(seed)$ to define $A[j,i]$

Other goodies

- Assume we have *two* streams S_1 and S_2, in different locations
- Want to compute their sketches locally, transmit them to the central location, and compute

$$\|x(S_1 \cup S_2)\|_p = \|x(S_1) + x(S_2)\|_p$$

- Easy:
  - Compute (locally) and send $Ax(S_1)$ and $Ax(S_2)$
  - In the central location, compute

$$A[x(S_1) + x(S_2)] = Ax(S_1) + Ax(S_2)$$

  - Estimate $\|x(S_1)+x(S_2)\|_p$ from $A[x(S_1)+x(S_2)]$