CS60021: Scalable Data Mining

Streaming Algorithms

Sourangshu Bhattacharya

Frequent count

Streaming model revisited

- Data is seen as incoming sequence
  - can be just element-ids, or (id, frequency update) tuple
- Arrival only streams
- Arrival + departure
  - Negative updates to frequencies possible
  - Can represent fluctuating quantities, e.g.

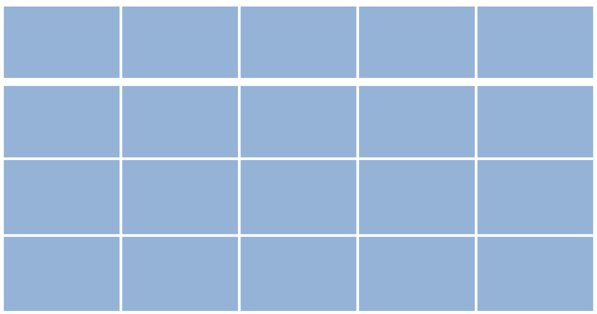
Review: Frequency Estimation in one pass

- Given input stream, length m, want a sketch that can answer frequency queries at the end
  - For give item x, return an estimate of the frequency

- Algorithms seen
  - Deterministic counter based algorithms: Misra-Gries, SpaceSaving
  - Count-Min sketch

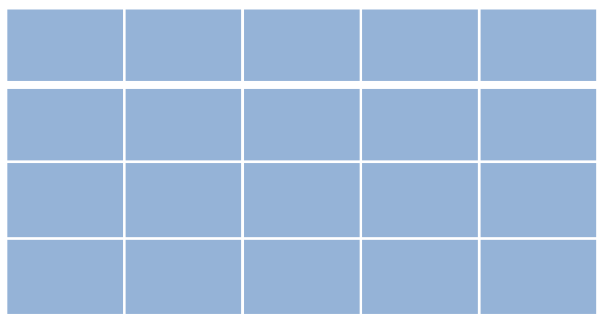
Recall: Count-min sketch

- Model input stream as a vector over U
  - f_x is the entry for dimension x
- Creates a small summary $w \times d$
- Use w hash functions, each maps $U \rightarrow [1, d]$



Count-sketch

- Model input stream as a vector over U
  - f_x is the entry for dimension x
- Creates a small summary $w \times d$
- Use w hash functions, $h_i: U \rightarrow [1, d]$
- w sign hash function, each maps $\mathrm{g_i}$: $U \rightarrow \{-1, +1\}$



Count Sketch

Initialize

– Choose $h_1, .., h_w,\ A[w, d] \leftarrow 0$

Process(x, c):

– For each $i \in [w],\ A[i, h_i\,(x)] \mathrel{+}= c \times g_i(x)$

Query(q):

– Return median$\{g_i(x)A[i, h_i(x)]\}$

Example

h1			
h2			

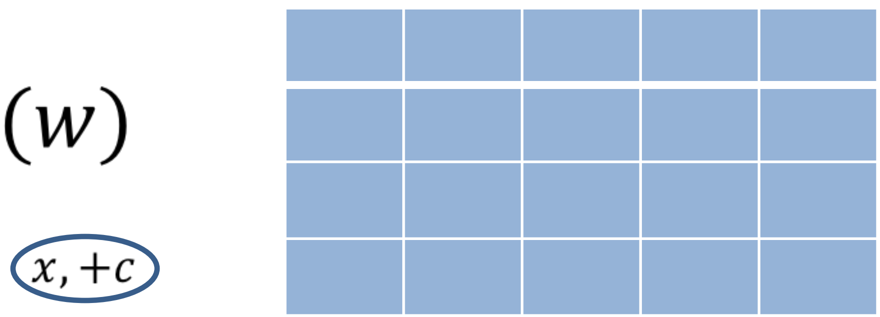
	h1,g1	h2,g2
	2,+	1,+
	3,-	2,+
	1,+	3,-
	2,-	3,+

8

Guarantees

Space = $O(wd)$

Update time = $O(w)$



Each item is mapped to one bucket per row

Guarantees

- $w = \frac{2}{\epsilon^2} \quad \mathrm{d} = \log\left(\frac{1}{\delta}\right)$

$Y_1 \ldots . Y_w$ be the w estimates, i.e. $Y_i = g_i(x) A[i, h_i(x)], \quad \widehat{f_x} = \underset{i}{\mathrm{median}}\, Y_i$

$$E[Y_i] = E\left[g_i(x)\, A[i, h_i(x)]\right] = E\left[g_i(x) \sum_{h_i(y)=h_i(x)} f_y\, g_i(y)\right]$$

Guarantees

$$E[Y_i] = E\big[g_i(x)\, A[i, h_i(x)]\big] = E\left[g_i(x) \sum_{h_i(y)=h_i(x)} f_y\, g_i(y)\right]$$

Notice that for $x \neq y$, $E[g_i(x)\; g_i(y)] = 0$!

$$E[Y_i] = g_i(x)^2 f_x = f_x$$

We analyse the variance in order to bound the error

For simplicity assume hash functions all independent

Variance analysis

Using simple algebra, as well as independence of hash functions, $|f|_2^2 = \sum_x f_x^2$

$$var(Y_i) = \frac{\left(\sum_y f_y^2 \; - \; f_x^2\right)}{d} \leq \frac{|f|_2{}^2}{d}$$

Using Chebyshev's inequality

$$\Pr[\, |Y_i \; - f_x| > \epsilon |f|_2] \leq \frac{1}{d\epsilon^2} \leq \frac{1}{3} \qquad\qquad d = \frac{3}{\epsilon^2}$$

Finally, use analysis of median-trick with $w = \log\left(\frac{1}{\delta}\right)$

Final Guarantees

- Using space $O\left(\frac{1}{\epsilon^2}\log\left(\frac{1}{\delta}\right)\log(n)\right)$, for any query x, we get an estimate, with prob $1\ -\delta$

$$f_x\ \ -\epsilon|f|_2 \leq f_x \leq f_x + \epsilon|f|_2$$

Comparisons

Algorithm	$\widehat{f_x} - f_x$	Space $\times log(n)$	Error prob	Model
Misra-Gries	$[-\epsilon\|f\|_1, 0]$	$1/\epsilon$	0	Insert Only
SpaceSaving	$[0, \epsilon\|f\|_1]$	$1/\epsilon$	0	Insert Only
CountMin	$[0, \epsilon\|f\|_1]$	$\log\left(\frac{1}{\delta}\right)/\epsilon$	δ	Insert
CountSketch	$[-\epsilon\|f\|_2, \epsilon\|f\|_2]$	$\log\left(\frac{1}{\delta}\right)/\epsilon^2$	δ	Insert+Delete

(3) Computing Moments

Generalization: Moments

- **Suppose a stream has elements chosen from a set *A* of *N* values**

- **Let m_i be the number of times value *i* occurs in the stream**

- **The k^{th} *moment* is**

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

Special Cases

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

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

Example: 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

AMS method

- AMS method works for all moments
- Gives an unbiased estimate.
- We will just concentrate on the 2nd moment S.
- We pick and keep track of many variables X:
  - For each variable X, 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 $S = \Sigma_i m_i^2$

One random variable (X)

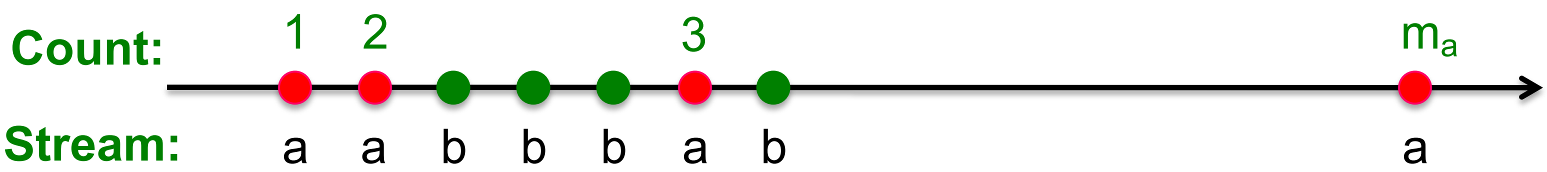
- How to set X.val and X.el ?
  - Assume stream has length n (we relax this later)
  - Pick some random time **t (t<n)** to start, so that any time is equally likely
  - Let at time t the stream have item **i**. We set ***X.el = i***
  - Then we maintain count **c** (***X.val = c***) of the number of is in the stream starting from the chosen time **t**
- Then the estimate of the 2^nd^ moment ($\Sigma_i m_i^2$) is:

$$S = f(X) = n\,(2\,c - 1)$$

  - Note, we will keep track of multiple **Xs, (X_1, X_2,... X_k)** and our final estimate will be:

$$S = 1/k\ \Sigma_j\, f(X_j)$$

Expectation Analysis



- 2nd moment is $\mathbf{S = \Sigma_i\, m_i^2}$
- $\mathbf{C_t}$ - number of times item at time **t** appears from time **t** onwards $(\mathbf{c_1=m_a\ ,\ c_2=m_a\text{-}1,\ c_3=m_b})$
- $E[f(X)] = 1/n\ \Sigma_{t=1}^{n}\ n\ (2c_t - 1)$

  $= 1/n\ \Sigma_i\ n\ (1 + 3 + 5 + \ldots + 2\ m_i - 1)$

m_i … total count of item i in the stream (we are assuming stream has length **n**)

Group times by the value seen

Time t when the last i is seen (c_t=1)

Time t when the penultimate i is seen (c_t=2)

Time t when the first i is seen (c_t=m_i)

Higher-Order Moments

- 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·c² – 3c + 1)** (where c=X.val)
- Why?
  - For **k=2**: Remember we had $\mathbf{(1+3+5+\cdots+(2m_i-1))}$ and we showed terms **2c-1 (for c=1,...,m)** sum to $\mathbf{m^2}$
  - $\boldsymbol{2c - 1 = c^2 - (c-1)^2}$
  - For **k=3**: $\boldsymbol{c^3 - (c-1)^3 = 3c^2 - 3c + 1}$
- Generally: Estimate = $\mathbf{n\,(c^k - (c-1)^k)}$

Combining Samples

- **In practice:**
  - Compute $f(X) = n(2\,c - 1)$ 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

Streams Never End: Fixups

- **(1)** The variables ***X*** have ***n*** as a factor – keep ***n*** separately; just hold the count in ***X***
- **(2)** Suppose we can only store ***k*** counts. We must throw some ***X***s 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 n^{th} 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

AMS algorithm

Initialize : $(m, r, a) \leftarrow (0, 0, 0)$

Process j:

$m \leftarrow m + 1$ $\beta \leftarrow$ random bit with $\Pr[\beta = 1] = 1/m$ **if** $\beta = 1$ **then**

$\quad a \leftarrow j$ $r \leftarrow 0$

if $j = a$ **then**

$\quad r \leftarrow r + 1$

Output : $m(r^k - (r-1)^k)$

AMS ALGORITHM USING SKETCHES

Generalization of AMS Algorithm

- Stream of pair (i,c), i € {1,…,U} and c is positive integer.
- ***x[i] = x[i] + c*** for each update
- Join size: $\boldsymbol{x.y = \Sigma_{i=1}^{U} (x[i]\, y[i])}$
- Pth Moment: $\boldsymbol{F_P(x) = \Sigma_{i=1}^{U} x[i]^2}$

$$\|\boldsymbol{x} - \boldsymbol{y}\|_2 = \sqrt{F_2(\boldsymbol{x} - \boldsymbol{y})}.$$

- h : {1,…U} ➔ {+1,-1}

Generalization of AMS Algorithm

UPDATE(i, c, z)

Input: item i, count c, sketch z

for $j = 1$ to w **do**
    for $k = 1$ to d **do**
        $z[j][k] += h_{j,k}(i) * c$

ESTIMATE$F_2(z)$

Input: sketch z

Return ESTIMATEJS(z, z)

ESTIMATEJS(x, y)

Input: sketch x, sketch y

Output: estimate of $\boldsymbol{x} \cdot \boldsymbol{y}$

for $j = 1$ to w **do**
    $avg[j] = 0;$
    for $k = 1$ to d **do**
        $avg[j] += x[j][k] * y[j][k]/w;$
Return(median(avg))

Fig. 1 AMS algorithm for estimating join and self-join size

Generalization of AMS Algorithm

Lemma 1 $\mathsf{E}(Z^2) = F_2(\boldsymbol{x})$

Proof

$$\begin{aligned}
\mathsf{E}(Z^2) &= \mathsf{E}\left(\left(\sum_{i=1}^{U} h(i)\boldsymbol{x}[i]\right)^2\right) \\
&= \mathsf{E}\left(\sum_{i=1}^{U} h(i)^2\boldsymbol{x}[i]^2\right) + \mathsf{E}\sum_{1\le i<j\le U} 2h(i)h(j)\boldsymbol{x}[i]\boldsymbol{x}[j] \\
&= \sum_{i=1}^{U}\boldsymbol{x}[i]^2 + 0 = F_2(\boldsymbol{x}).
\end{aligned}$$

Generalization of AMS Algorithm

- ***Var(Z²) ≤ 2F₂(x)²***

$$\mathsf{Var}(Z^2) = \mathsf{E}(Z^4) - \mathsf{E}(Z^2)^2$$

$$= \mathsf{E}\left(\left(\sum_{i=1}^{U} h(i)\boldsymbol{x}[i]\right)^4\right) - \left(\sum_{i=1}^{U} \boldsymbol{x}[i]^2\right)^2$$

Generalization of AMS Algorithm

$$
\begin{aligned}
= \mathsf{E}\Bigg(\Bigg(&\sum_{i=1}^{U} h(i)^4 \boldsymbol{x}[i]^4 + \sum_{1\le i<j\le U} 6h(i)^2 h(j)^2 \boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2 \\
&+ \sum_{i,i\neq j\neq k} 12h(i)^2 h(j)h(k)\boldsymbol{x}[i]^2 \boldsymbol{x}[j]\boldsymbol{x}[k] \\
&+ \sum_{1\le i\neq j\le U} 4h^3(i)h(j)\boldsymbol{x}[i]^3 \boldsymbol{x}[j] \\
&+ \sum_{1\le i<j<k<l\le U} 12h(i)h(j)h(k)h(l)\boldsymbol{x}[i]\boldsymbol{x}[j]\boldsymbol{x}[k]\boldsymbol{x}[l]\Bigg) \\
&- \Bigg(\sum_{i=1}^{U} \boldsymbol{x}[i]^4 + \sum_{1\le i<j\le U} 2\boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2\Bigg)
\end{aligned}
$$

Generalization of AMS Algorithm

$$
\begin{aligned}
&= \sum_{i=1}^{U} \boldsymbol{x}[i]^4 + \sum_{1 \le i < j \le U} 6\boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2 \\
&\quad - \left(\sum_{i=1}^{U} \boldsymbol{x}[i]^4 + \sum_{1 \le i < j \le U} 2\boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2 \right) \\
&= 4 \sum_{1 \le i < j \le U} \boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2 \Bigg) \le 2F_2^2.
\end{aligned}
$$

Generalization of AMS Algorithm

Fact 1 (Variance Reduction) *Let X_i be independent and identically distributed random variables. Then*

$$\mathsf{Var}\left(\sum_{i=1}^{w} \frac{X_i}{w}\right) = \frac{1}{w}\mathsf{Var}(X_1).$$

Fact 2 (The Chebyshev Inequality) *Given a random variable X,*

$$\mathsf{Pr}\left[\left|X - \mathsf{E}(X)\right| \geq k\right] \leq \frac{\mathsf{Var}(X)}{k^2}.$$

Generalization of AMS Algorithm

Theorem 1 *An (ϵ, δ)-approximation of F_2, the self-join size, can be computed in space $O(\frac{1}{\epsilon^2}\log 1/\delta)$ machine words in the streaming model. Each update takes time $O(\frac{1}{\epsilon^2}\log 1/\delta)$.*

Proof Applying the Chebyshev inequality to the average of $w = \frac{16}{\epsilon^2}$ copies of the estimate Z generates a new estimate Y such that

$$\Pr\left[|Y - F_2| \leq \epsilon F_2\right] \leq \frac{\mathsf{Var}(Y)}{\epsilon^2 F_2^2} = \frac{\mathsf{Var}(Z)}{c\epsilon^2 F_2^2} = \frac{2F_2^2}{(16/\epsilon^2)\epsilon^2 F_2^2} = \frac{1}{8}.$$

Generalization of AMS Algorithm

Fact 3 (Application of Chernoff Bounds) *Let R be a range of values $R = [R_{\min}..R_{\max}]$, and let Y_i be $d = 4 \log 1/\delta$ independent and identically distributed random variable such that* $\Pr[Y_i \notin R] \leq \frac{1}{8}$. *Then*

$$\Pr\left[\left(\text{median}_{i=1}^{d} Y_i\right) \notin R\right] \leq \delta,$$

that is, if there is constant probability that each Y_i falls within the desired range R, then taking the median of $O(\log 1/\delta)$ copies of Y_i reduces the failure probability to δ.

Hence, applying the Chernoff bound result from Fact 3 to the median of $4 \log 1/\delta$ copies of the average Y gives the probability of the results being outside the range of ϵF_2 from F_2 as δ. The space required is that to maintain $O(\frac{1}{\epsilon^2} \log 1/\delta)$ copies of the original estimate. Each of these requires a counter and a 4-wise independent hash function, both of which can be represented with a constant number of machine words under the standard RAM model. $\square$

RANGE QUERIES

Dyadic Intervals

- Define $\lg n$ partitions of $[n]$

$$
\begin{aligned}
\mathcal{I}_0 &= \{1,2,3,4,5,6,7,8,\ldots\} \\
\mathcal{I}_1 &= \{\{1,2\},\{3,4\},\{5,6\},\{7,8\},\ldots\} \\
\mathcal{I}_2 &= \{\{1,2,3,4\},\{5,6,7,8\},\ldots\} \\
\mathcal{I}_3 &= \{\{1,2,3,4,5,6,7,8\},\ldots\} \\
\vdots \quad & \vdots \quad \vdots \\
\mathcal{I}_{\lg n} &= \{\{1,2,3,4,5,6,7,8,\ldots,n\}\}
\end{aligned}
$$

- *Exercise:* Any interval $[i,j]$ can be written as the union of $\leq 2\lg n$ of the above intervals. E.g., for $n = 256$,

  $[48,107] = [48,48] \cup [49,64] \cup [65,96] \cup [97,104] \cup [105,106] \cup [107,107]$

  Call such a decomposition, the *canonical decomposition*.

Range Queries and Quantiles

- *Range Query:* For $1 \leq i \leq j \leq n$, estimate $f_{[i,j]} = f_i + f_{i+1} + \ldots + f_j$
- *Approximate Median:* Find j such that

$$\begin{aligned} f_1 + \ldots + f_j &\geq m/2 - \epsilon m \quad \text{and} \\ f_1 + \ldots + f_{j-1} &\leq m/2 + \epsilon m \end{aligned}$$

  Can approximate median via binary search of range queries.

- *Algorithm:*
  1. Construct $\lg n$ Count-Min sketches, one for each $\mathcal{I}_i$ such that for any $I \in \mathcal{I}_i$ we have an estimate $\tilde{f}_I$ for f_I such that

  $$\mathbb{P}\left[f_I \leq \tilde{f}_I \leq f_I + \epsilon m\right] \geq 1 - \delta \ .$$

  2. To estimate $[i,j]$, let $I_1 \cup I_2 \cup \ldots \cup I_k$ be canonical decomposition. Set

  $$\tilde{f}_{[i,j]} = \tilde{f}_{I_1} + \ldots + \tilde{f}_{I_k}$$

  3. Hence, $\mathbb{P}\left[f_{[i,j]} \leq \tilde{f}_{[i,j]} \leq 2\epsilon m \lg n\right] \geq 1 - 2\delta \lg n$.

Range Sum Example

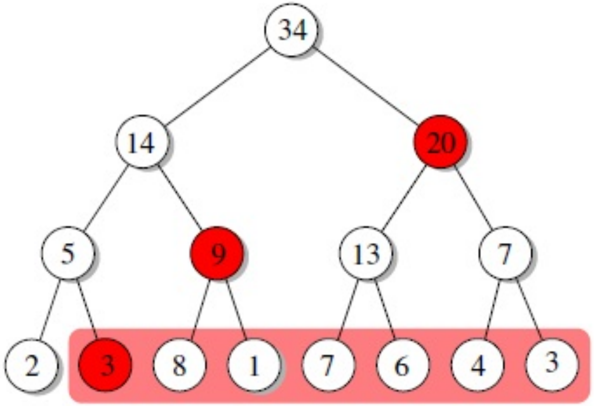
- AMS approach to this, the error scales proportional to $\sqrt{F_2(f)\ F_2(f')}$

So here the error grows proportional to the square root of the length of the range.

- Using the Count-Min sketch approach, the error is proportional to $F_1(h-l+1)$, i.e. it grows proportional to the length of the range

- Using the Count-Min sketch to approximate counts, the accuracy of the answer is proportional to $(F_1 \log n)/w$. For large enough ranges, this is an exponential improvement in the error.

e.g. To estimate the range sum of [2…8], it is decomposed into the ranges [2…2], [3…4], [5…8], and the sum of the corresponding nodes in the binary tree as the estimate.



Using dyadic ranges to answer a range query

Theorem 4 $a[l,r] \le \hat{a}[l,r]$

$$\Pr[\hat{a}[l,r] > a[l,r] + 2\varepsilon \log n \|\vec{a}\|_1] \le \delta$$

Proof : **Theorem 1** $\xrightarrow{a_i \le \hat{a}_i}$ $a[l,r] \le \hat{a}[l,r]$

E(Σ error for each estimator) $= 2\log n$ E(error for each estimator)

$$\le 2\log n \frac{\varepsilon}{e} \|\vec{a}\|_1$$

$$\Pr[\hat{a}[l,r] - a[l,r] > 2\log n \|\vec{a}\|_1] < e^{-d} \le \delta$$

Analysis

Time to produce the estimate $O\left(\log(n)\log\frac{1}{\delta}\right)$

Space used $O\left(\frac{\log(n)}{\varepsilon}\log\frac{1}{\delta}\right)$

Time for updates $O\left(\log(n)\log\frac{1}{\delta}\right)$

Remark : the guarantee will be more useful when stated without terms of $\log n$ In the approximation bound.

References:

- Primary references for this lecture
  - Lecture slides by Graham Cormode
    http://dmac.rutgers.edu/Workshops/WGUnifyingTheory/Slides/cormode.pdf
  - Lecture notes by Amit Chakrabarti: http://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf
  - Sketch techniques for approximate query processing, Graham Cormode.
    http://dimacs.rutgers.edu/~graham/pubs/papers/sk.pdf