The Data Stream Model: Sketches and Probability Tools

Ricard Gavaldà
Universitat Politècnica de Catalunya, Barcelona

ECML PKDD 2015 Summer School:
Data Sciences for Big Data - Resource Aware Data Mining
Porto, September 2nd, 2015
http://www.cs.upc.edu/~gavalda
Massive data requires new kind of **algorithmics**

Often, **approximate** answers are OK. That helps!

Focus of this talk:
- (Mostly) Streaming data
- Sketches
- Counting problems
Data streams everywhere

- Telcos - phone calls
- Satellite, radar, sensor data
- Computer systems and network monitoring
- Search logs, access logs
- RSS feeds, social network activity
- Websites, clickstreams, query streams
- E-commerce
- ...

...
Data streams: Concept

- Data arrives as sequence of items
- At high speed
- Forever
- Can’t store them all
- Can’t go back; or too slow
- Evolving, non-stationary reality
The Data stream axioms

Five Data Stream Axioms:

1. Only one pass; $t$-th item available at time $t$ only
2. Small processing time per item
3. Small memory, certainly sublinear in stream length; sketches or summaries
4. Able to provide answers at any time
5. The stream evolves over time
Assumptions & Requirements

- Worst-case, adversarial, input distribution
- Difference with probabilistic assumption:
  - Items are generated probabilistically (often independently), following a probability distribution that may evolve over time
  - Implicit in Data Stream Mining and Machine Learning: Generalization!
- Randomness is in the algorithm
  - Different runs may give different answers
  - But in most runs answer is approximately correct
The Item Counting Problem

How many items have we read so far in the data stream?

To count up to \( t \) elements \textit{exactly}, \( \log t \) bits are \textit{necessary}

Approximate solution using \( \log \log t \) bits
Approximate counting, v1

Init: \( c \leftarrow 0 \)

Update:

\[
\text{draw a random number } x \in [0, 1] \\
\text{if } (x \leq 1/2) \quad c \leftarrow c + 1
\]

Query: return \( 2c \)

\[
E[2c] = t, \quad \sigma \approx \sqrt{t/2}
\]

Space \( \log(t/2) = \log t - 1 \rightarrow \text{we saved 1 bit!} \)
Approximate counting, v2

Init: $c \leftarrow 0$

Update:
   draw a random number $x \in [0, 1]$
   if ($x \leq 2^{-k}$) $c \leftarrow c + 1$

Query: return $2^k c$

$E[c] = t/2^k, \quad \sigma \approx \sqrt{t/2^k}$

Memory log $t - k \rightarrow$ we saved $k$ bits!

$x \leq 2^{-k}$: AND of $k$ random bits, log $k$ memory
Approximate counting: Morris’ counter

Morris’ counter [Morris77]

Init: $c \leftarrow 0$

Update:
- draw a random number $x \in [0, 1]$
  - if $(x \leq 2^{-c})$ $c \leftarrow c + 1$

Query: return $2^c - 2$

$E[c] \approx \log t$, $E[2^c - 2] = t$, $\sigma \approx t/\sqrt{2}$

Memory = bits used to hold $c = \log c = \log \log t$ bits
Morris’ approximate counter

- Can count up to 1 billion with $\log \log 10^9 = 5$ bits

- Problem: large variance, $\sigma \simeq 0.7 t$
Use basis $b < 2$ instead of basis 2:

- Places $t$ in the series $1, b, b^2, \ldots, b^i, \ldots$ (“resolution” $b$)
- $E[b^c] \sim t$, $\sigma \sim \sqrt{(b - 1)/2 \cdot t}$
- Space $\log \log t - \log \log b$ ($> \log \log t$, because $b < 2$)
- For $b = 1.08$, 3 extra bits, $\sigma \sim 0.2 \cdot t$
Run $r$ parallel, independent copies of the algorithm

On Query, average their estimates

$E[\text{Query}] \approx t$, $\sigma \approx t/\sqrt{2r}$ (why?)

Space $r \log \log t$

Time per item multiplied by $r$

Worse performance, but more generic technique
Morris’ counter: A non-streaming application

In [VanDurme+09]

- Counting $k$-grams in a large text corpus
- Number of $k$-grams grows exponentially with $k$
- Highly diverse frequencies
- Should fit in RAM
- Use Morris’ counters (5 bits) instead of standard counters
2. Approximation. Large Deviation Bounds
Reducing the variance, general method

- Variance: \( \text{Var}(X) = E[(X - E[X])^2] = E[X^2] - E[X]^2 \)
- \( \text{Var}(\alpha \cdot X + \beta) = \alpha^2 \cdot \text{Var}(X) \)
- If \( X \) and \( Y \) independent, \( \text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) \)
- In general, if \( X_i \) are all independent and \( \text{Var}(X_i) = \sigma^2 \),
  \[
  \text{Var}
  \left( \frac{1}{n} \sum_{i=1}^{n} X_i \right) = \frac{1}{n^2} (n \sigma^2) = \frac{\sigma^2}{n}
  \]
  Equivalently,
  \[
  \sigma \left( \frac{1}{n} \sum_{i=1}^{n} X_i \right) = \frac{\sigma}{\sqrt{n}}.
  \]
Deviation Bounds

Random variables often described by expectation + variance

Suppose

\[ E[\text{algorithm output}] = \text{desired result}, \quad \text{Var}(\text{algorithm output}) = \sigma^2 \]

We usually want instead

\[ |\text{algorithm output} - \text{desired result}| \leq \text{something} \]
(ε, δ)-approximation

A randomized algorithm \( A(\varepsilon, \delta) \)-approximates a function \( f : X \to \mathbb{R} \) iff for every \( x \in X \), with probability \( \geq 1 - \delta \)

- (absolute approximation) \( |A(x) - f(x)| \leq \varepsilon \)
- (relative approximation) \( |A(x) - f(x)| \leq \varepsilon f(x) \)

\( \varepsilon = \text{accuracy}; \quad \delta = \text{confidence} \)
Often \( \varepsilon, \delta \) given as extra inputs to \( A \)
Markov’s inequality

For a non-negative random variable $X$ and every $k$

$$\Pr[X \geq k \mathbb{E}[X]] \leq 1/k$$

Proof:

$$\mathbb{E}[X] = \sum_x \Pr[X = x] \cdot x \geq \sum_{x \geq k} \Pr[X = x] \cdot x \geq \sum_{x \geq k} \Pr[X = x] \cdot k = k \Pr[X \geq k]$$
Deviation Bounds

**Chebyshev’s inequality**

For every \( X \) and every \( k \)

\[
\Pr[|X - \mathbb{E}[X]| \geq k] \leq \frac{\text{Var}(X)}{k^2}
\]

Equivalently,

\[
\Pr[|X - \mathbb{E}[X]| \geq k \sigma(X)] \leq \frac{1}{k^2}
\]

**Proof:**

\[
\Pr[|X - \mathbb{E}[X]| > k] = \Pr[(X - \mathbb{E}[X])^2 > k^2] \leq (\text{Markov}) \\
\leq \frac{\mathbb{E}[(X - \mathbb{E}[X])^2]}{k^2} = \frac{\text{Var}(X)}{k^2}
\]
Chebyshev gives \((\varepsilon, \delta)\)-approximations

Let algorithm \(A\) be such that \(E[A(x)] = f(x), \ Var(A(x)) \leq \sigma^2\)

Algorithm \(B(x)\) averages \(b\) independent copies of \(A(x)\)

We have \(E[B(x)] = f(x), \ Var(B(x)) \leq \sigma^2/b\)

\[
\Pr[|B(x) - f(x)| > \varepsilon] \leq \frac{\text{Var}(B(x))}{\varepsilon^2} \leq \frac{\sigma^2}{b\varepsilon^2} \leq \delta
\]

if we choose \(b = \sigma^2 \frac{1}{\varepsilon^2} \frac{1}{\delta}\)
Chebyshev gives \((\varepsilon, \delta)\)-approximations

\[
\Pr[|X - E[X]| > k\sigma]
\]

<table>
<thead>
<tr>
<th>(k = 1)</th>
<th>(k = 2)</th>
<th>(k = 3)</th>
<th>(k = 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\leq 1)</td>
<td>(\leq 0.25)</td>
<td>(\leq 0.11)</td>
<td>(\leq 0.07)</td>
</tr>
</tbody>
</table>

But if \(X\) is normally distributed,

<table>
<thead>
<tr>
<th>(k = 1)</th>
<th>(k = 2)</th>
<th>(k = 3)</th>
<th>(k = 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\leq 0.32)</td>
<td>(\leq 0.05)</td>
<td>(\leq 0.003)</td>
<td>(\leq 3 \cdot 10^{-5})</td>
</tr>
</tbody>
</table>
Sums of Independent Variables

\[ \exp(-x^2) \text{ vs. } 1/x^2: \]
Suppose $X = \sum_{i=1}^{n} X_i$, $E[X_i] = p$, $\text{Var}(X_i) = \sigma^2$, all $X_i$ independent and bounded.

By the Central Limit Theorem, $Z_n = (X - np)/\sqrt{n\sigma^2}$ tends to normal $N(0, 1)$ as $n \to \infty$.

And approximating by the normal gives

$$\Pr[ Z_n \geq \alpha ] \approx \exp(-\alpha^2/2)$$

Chebyshev only gives

$$\Pr[ Z_n \geq \alpha ] \leq \frac{1}{\alpha^2}$$
Chernoff-Hoeffding bounds

- $X_1, X_2, \ldots X_n$ be independent random variables,
- $X_i \in [0, 1]$, $E[X_i] = \rho$,
- $X = \sum_{i=1}^{n} X_i$, so $E[X] = \rho n$

**Hoeffding bound (absolute deviation)**

$$\Pr[X - \rho n > \varepsilon n] < \exp(-2\varepsilon^2 n)$$
$$\Pr[X - \rho n < -\varepsilon n] < \exp(-2\varepsilon^2 n)$$

**Chernoff bound (relative deviation)**

For $\varepsilon \in [0, 1]$,

$$\Pr[X - \rho n > \varepsilon \rho n] < \exp(-\varepsilon^2 \rho n / 3)$$
$$\Pr[X - \rho n < -\varepsilon \rho n] < \exp(-\varepsilon^2 \rho n / 2)$$

Note: *Bernstein’s inequality* is more general and (in essence) subsumes both
Example: Approximating the Mean

Input: $\varepsilon$, $\delta$, random variable $X \in [0,1]$ (Important: bounded)
Output: $(\varepsilon, \delta)$-approximation of $E[X]$

Algorithm $A(\varepsilon, \delta)$

- Draw $n = \frac{1}{2\varepsilon^2} \ln \frac{2}{\delta}$ copies of $X$
- Output their average $Y$
Example: Approximating the Mean

- Let $X_i$ be $i$th copy of $X$
- Then $Y = \frac{1}{n} \sum_{i=1}^{n} X_i$, and $E[Y] = E[X]$
- By Hoeffding,

$$\Pr[|Y - E[X]| > \varepsilon] = \Pr[\sum_{i=1}^{n} X_i - E[\sum_{i=1}^{n} X_i] > \varepsilon n]$$

$$< 2 \exp(-2\varepsilon^2 n) = 2 \exp(-\ln(2/\delta)) = \delta$$

- A different, sequential, algorithm gets $(\varepsilon, \delta)$ relative approximation using

$$O\left(\frac{1}{\varepsilon^2 E[X]} \ln \frac{1}{\delta}\right)$$

samples of $X$

[Dagum-Karp-Luby-Ross 95, Lipton-Naughton 95]
Example: Approximating the Median

Input: $\varepsilon$, $\delta$, set $S$ of real numbers (Note: no bound assumed)
Output: some $s \in S$ whose rank in $S$ is $(1/2 \pm \varepsilon)|S|$

Algorithm $A(\varepsilon, \delta)$
- Draw $n = \frac{1}{2\varepsilon^2} \ln \frac{2}{\delta}$ random elements from $S$
- Output the median of these $n$ elements
Example: Approximating the Median

- Let $X_i$ be 1 if $i$th sample has rank $\leq (1/2 - \varepsilon)|S|$, 0 otherwise.
- $E[X_i] = 1/2 - \varepsilon$
- By Hoeffding,

$$\Pr[\geq n/2 \text{ draws give elements with rank } \leq (1/2 - \varepsilon)|S|]$$

$$\leq \Pr[\sum_{i=1}^{n} X_i \geq n/2] = \Pr[\sum_{i=1}^{n} X_i \geq E[\sum_{i=1}^{n} X_i] + \varepsilon n]$$

$$\leq \exp(-2\varepsilon^2 n) = \delta/2$$

- Therefore, with probability $< \delta/2$ we draw $\geq n/2$ elements of rank $\leq (1/2 - \varepsilon)|S|$. Implies median of sample $> (1/2 - \varepsilon)|S|$
- Similarly the other side
Example use in Data Streams: Sampling rate

- Suppose items arrive at so high speed that we have to skip some
- Sample randomly:
  - Choose to process each element with probability $\alpha$
  - Ignore each element with prob. $1 - \alpha$
- At any time $t$, if queried for the median, return the median of the elements chosen so far

Exercise.

Given $\alpha$, $\delta$, determine the probability $\varepsilon_t$ such that at time $t$ the output of the algorithm above is an $(\varepsilon_t, \delta)$-approximation of the median on the first $t$ elements of the stream
Improved $(\varepsilon, \delta)$-approximation: $1/\delta$ to $\ln(1/\delta)$

Let algorithm $A$ be such that $E[A(x)] = f(x)$, $\text{Var}(A(x)) \leq \sigma^2$
(Note: no bound assumed)

$B$: Run $b$ independent copies of $A$ and average results
With $b = 6\sigma^2/\varepsilon^2$ we have

$$\Pr[|B(x) - f(x)| \geq \varepsilon] < 1/6$$

$C$: Run $c$ independent copies of $B$ and take median
With $c = \frac{1}{2(1/2 - 1/6)^2} \ln \frac{2}{\delta}$ we have (Exercise: check!)

$$\Pr[|C(x) - f(x)| > \varepsilon] \leq \delta$$

Memory and runtime blowup is $b \cdot c = 27\sigma^2 \frac{1}{\varepsilon^2} \ln \frac{2}{\delta}$

A better analysis reduces constant 27 to about 4
3. Counting distinct elements

The Distinct Element Counting Problem

How many *distinct* elements have we seen so far in the data stream?
Item spaces and # distinct elements can be large

- I’m a web searcher. How many different queries did I get?
- I’m a router. How many pairs (sourceIP,destinationIP) have I seen?
  - itemspace: potentially $2^{128}$ in IPv6
- I’m a text message service. How many distinct messages have I seen?
  - itemspace: essentially infinite
- I’m an streaming classifier builder. How many distinct values have I seen for this attribute $x$?
Item space $I$, cardinality $n$, identified with range $[n]$

$f_{i,t} = \#$ occurrences of $i \in I$ among first $t$ stream elements

$d_t = \text{number of } i\text{'s for which } f_{i,t} > 0$

Often omit subindex $t$

Solving exactly requires $O(d)$ memory

Approximate solutions using $O(d)$, $O(\log d)$ and $O(\log \log d)$ bits
Linear counting \cite{Whang+90} \sim Bloom filters

\begin{itemize}
  \item Init($d_{\text{max}}, \rho$):
    \begin{itemize}
      \item upper bound $d_{\text{max}} \geq d$
      \item $\rho < 1$, load factor
      \item build a bit vector $B$ of size $s = \rho \cdot d_{\text{max}}$
      \item choose a hash function $f : [n] \rightarrow s$
    \end{itemize}
  \item Update($x$): $B[f(x)] \leftarrow 1$
  \item Query:
    \begin{itemize}
      \item $w = \text{the fraction of 0's in } B$
      \item return $s \cdot \ln(1/w)$
    \end{itemize}
\end{itemize}
Linear counting \cite{Whang90} \simeq Bloom filters

\[
\begin{align*}
w &= \Pr[\text{bucket } i \text{ after } d \text{ distinct elements}] = (1 - 1/s)^d \simeq \exp(-d/s) \\
E[\text{Query}] &\simeq d, \quad \sigma(\text{Query}) = \text{small!}
\end{align*}
\]

**Issue:** What is a “good” hash function?

- \( f(i) \) uniformly distributed, even given all other values of \( f \)
- “Reproducibly random”
- How to get one: Later!
Cohen’s algorithm [Cohen97]

\[ E[\text{gap between two 1's in } B] = \frac{s - d}{d + 1} \approx \frac{s}{d} \]

Query: return \( s / (\text{size of first gap in B}) \)
Cohen’s algorithm [Cohen97]

Trick: Don’t store $B$, remember smallest key inserted in $B$

Init: $\text{posmin} = s$; choose hash function $f : [n] \rightarrow s$

Update($x$): if ($f(x) < \text{posmin}$) $\text{posmin} \leftarrow f(x)$

Query: return $s/\text{posmin}$
Cohen’s algorithm [Cohen97]

\[ E[\text{posmin}] \approx \frac{s}{d} \quad \sigma(\text{posmin}) \approx \frac{s}{d} \]

Memory = (bits to store posmin) =
\[ \log(\text{posmin}) \leq \log s = O(\log d_{\text{max}}) \]
Bloom filter. But: Observe values of hash function $f(i)$, in binary

Idea: To see $f(i) = 0^{k-1}1 \ldots$, about $2^k$ distinct values inserted

And we don’t need to store $B$, just the smallest $k$
Flajolet-Martin probabilistic counter

Init: $p \leftarrow 0$

Update($x$):
- let $b$ be the position of the leftmost 1 bit of $f(x)$
- if ($b > p$) $p \leftarrow b$

Query: return $2^p$

$E[2^p] = d/\varphi$, for a constant $\varphi = 0.77\ldots$

Memory = (bits to store $p$) = $\log p = \log \log d_{\max}$ bits
Solution 1: Use $c$ independent copies, average

- Problem 1: runtime multiplied by $c$
- Problem 2: independent runs = generate *independent* hash functions
- And we don’t know how to generate several independent hash functions
Solution 2:

- Divide stream into \( c = O(\varepsilon^{-2}) \) substreams
- Use first bits of \( f(x) \) to decide substream for \( x \)
- Track \( p \) separately for each substream
- Same \( f \) can be used for all copies
- One sketch update per item

Memory = \( O(c \log \log d_{\text{max}}) = O(\varepsilon^{-2} \log \log d_{\text{max}}) \)
Improving the leading constants

- Original [Flajolet-Martin 85]: Geometric average of estimations
- SuperLogLog [Durand+03]: Remove top 30%, then geometric average
- HyperLogLog [Flajolet+07]: Harmonic average

Standard deviation is $\simeq 1.03/\sqrt{c}$ for HyperLogLog

HyperLogLog: “cardinalities up to $10^9$ can be approximated within say 2% with 1.5 Kbytes of memory”

Implementation aspects: [Heule+13]
Linear or logarithmic?

[Metwaly+08]

- “Why go logarithmic when we can go linear”
- Describe an application where extreme accuracy needed
  - e.g., $10^{-4}$
- For this range, linear counting uses less memory
- My take: I have ML/DM in mind; low accuracy is ok, *and* we will need to maintain *many* counts
4. Finding Frequent Elements

Heavy Hitters, Elephants, Hotlist analysis, Iceberg queries
Finding Frequent Elements

The Heavy Hitter Problem

Given a sequence $S$ of $t$ elements, threshold $\theta$, find all elements with frequency $> \theta t$ - the heavy hitters

Interesting for skewed distributions

There are at most $\left\lfloor 1/\theta \right\rfloor$ heavy hitters

Good sources: [Berinde+09], [Cormode+08]
1. Sampling: Output the heavy hitters computed in a sample
   - Uniform sample can be kept with reservoir sampling technique
   - Doable with sample size $O(1/\theta^2)$ (Hoeffding)

Solutions with memory $O(1/\theta)$:

2. Count based. We cover SpaceSaving Sketch

3. Hash based: Count-Min Sketch
The SpaceSaving sketch [Metwally+05]

- One of many counter-based methods: Karp-Shenker-Papadimitriou, Lossy Counter, Frequent, Sticky Sampling, GroupTest, . . .
- Memory $O(1/\theta)$. Best possible
- Good update time
- Guarantee on count error
- No false negatives; but has false positives
The SpaceSaving sketch

Init(θ): Create

\[ k \leftarrow \lceil 1/\theta \rceil \]

set of keys \( K \leftarrow \emptyset \)

vector \( count \), indexed by \( K \)

Update(\( x \)):

if \( x \) is in \( K \) then \( count[x]++ \)

else, if \(|K| < k\), add \( x \) to \( K \) and set \( count[x] = 1 \)

else, replace an item with lowest count with \( x \) and increase its count by 1

Query:

return the set \( K \)
Why Does This Work?

Claims:
Let $\min_t$ be the minimum value of a counter at time $t > 0$. Then

1. $\min_t \leq t/k$
2. If $f_{x,t} > \min_t$, then $x \in K$ at time $t$
3. For every $x \in K$, $f_{x,t} \leq \text{count}_t[x] \leq f_{x,t} + \min_t$

In particular, all items with frequency over $t/k$ are in $K$

Proof: By joint induction on $t$. Exercise: prove it!
More on SpaceSaving

Efficient implementation: StreamSummary data structure

Exercise
Without looking into the paper, propose an efficient data structure for SpaceSaving. Aim for $O(1)$ update time and $O(k) = O(1/\theta)$ items, counts, pointers, etc.
The Count-Min Sketch

[Cormode-Muthukrishnan 04]
Like SpaceSaving:

- Provides an approximation $f'_x$ to $f_x$, for every $x$
- Can be used (less directly) to find $\theta$-heavy hitters
- Uses memory $O(1/\theta)$

Unlike SpaceSaving:

- It is randomized - hash functions instead of counters
- Supports additions and deletions
- Supports (not trivially) Heavy Hitters
- Can be used as basis for several other queries
The Count-Min Sketch

- Vector $F[n]$. Assumes $F[i] \geq 0$ for all $i$, at all times

- Provides estimations $F'$ of $F$ such that
  1. $F[i] \leq F'[i]$ for all $i$
  2. For every $i \in I$, $F'[i] \leq F[i] + \varepsilon |F|_1$ with probability $\geq 1 - \delta$

  where $|F|_1 = \sum_i F[i]$

- Note: $|F|_1$ may be $\ll$ stream length, if subtractions allowed

- Uses $O\left(\frac{1}{\varepsilon} \ln \frac{1}{\delta}\right)$ memory words, $O(\ln \frac{1}{\delta})$ update time
source: A. Bifet,
The Count-Min Sketch

- $d$ independent hash functions $h_1 \ldots h_d : [1..n] \rightarrow [1..w]$
- one “memory cell” for each $h_j(i)$
- On instruction “$F[i] += v$”, do $h_j(i) += v$ for all $j \in 1 \ldots d$
- Estimation:
  \[ F'[i] = \min \{ h_j(i) \mid j = 1..d \} \]
The Count-Min Sketch

\[ F'[i] = \min\{ h_j(i) \mid j = 1..d \} \]

- \( F'[i] \geq F[i] \)
  
  For each instruction involving \( i \), we update all counts \( h_j(i) \)
  
  \( F[i] \geq 0 \) at all times for all \( i \)

- \( F'[i] = F[i] ? \)
  
  No: cell \( h_j(i) \) is also incremented by \( k \neq i \) if \( h_j(k) = h_j(i) \)
  
  But it is unlikely that this occurs very often

- \( \min \) instead of average → Markov instead of Chebyshev or Hoeffding
Fix $j$. Define random variable $l_{ijk} = 1$ if $h_j(i) = h_j(k)$, 0 otherwise.

If $h$ is a good hash function

$$E[l_{ijk}] \leq \frac{1}{\text{range}(h_j)} = \frac{1}{w}$$

Define $X_{ij} = \sum_k l_{ijk} F[k]$. Then

$$E[X_{ij}] = \sum_k E[l_{ijk}] F[k] \leq |F|_1 / w$$
The Count-Min Sketch: Proof of main bound (2)

Then by Markov’s inequality and pairwise independence:

$$\Pr[X_{ij} \geq \varepsilon | F_1] \leq E[X_{ij}]/(\varepsilon | F_1) \leq (|F_1|/w)/(\varepsilon | F_1|) \leq 1/2$$

if $w = 2/\varepsilon$. Then:

$$\Pr[F'[i] \geq F[i] + \varepsilon | F_1] = \Pr[\forall j : F[i] + X_{ij} \geq F[i] + \varepsilon | F_1] = \Pr[\forall j : X_{ij} \geq \varepsilon | F_1] \leq (1/2)^d = \delta \quad \text{if } d = \log(1/\delta)$$

for one fixed $i$. To have good estimates for all $i$ simultaneously, use $d = \log(n/\delta)$ and use union bound.
The Count-Min Sketch: Summary

- Memory is $\frac{2}{\varepsilon} \log \frac{1}{\delta}$ words
- Update time $O(\log \frac{1}{\delta})$
- Replace $\log(1/\delta)$ with $\log(n/\delta)$ if the bound needs to hold for all $i$ simultaneously
  
  “Pr[for all $i, \ldots] \leq \delta”$ instead of “for all $i$, Pr[\ldots] \leq \delta”

- Error for $F[i]$ is $\varepsilon$ relative to $|F|_1$, not to $F[i]$
Back to Heavy Hitters

- $i$ is a $\theta$-heavy hitter if $F[i] \geq \theta t$
- The CM-sketch with width $\theta$ guarantees

$$F[i] \leq F'[i] \leq F[i] + \theta t$$

- So: If we output all $i$ s.t. $F'[i] \geq \theta t$, we output all heavy hitters; no false negatives

But we can’t cycle through all $n$ candidates one by one!
Range-sum query

Given $a, b$, return $\sum_{i=a}^{b} F[i]$

Example: how many packets received came from the IP range 172.16.xxx.xxx?

We show:

- A variant of CM-sketch supports range-sum queries efficiently
- Answering range-sum queries efficiently $\rightarrow$ finding heavy hitters efficiently
For $p = 0 \ldots \log n$, for each $j = \ldots$, keep the value of
\[
\text{sum}(j2^p \ldots (j+1)2^p - 1)
\]
Any interval $[a, b]$ is the sum of $O(\log n)$ such values. Check it
Keep one CM-sketch for each $2^p$ to store $\sum(j2^p \ldots (j+1)2^p - 1)$ for each $j$
From CM-sketch to range-sum queries

When receiving \( i \), update the counts for ranges where \( i \) lies = ancestors of \( i \) in the tree

When queried \( \text{sum}(a..b) \), decompose \([a..b]\) as sum of such intervals, retrieve and add their sums
Adaptively search for heavy hitters in the tree
if a node has count \(< \theta t\), do not explore its children: no heavy hitters below
if a node has count \(\geq \theta t\), explore both children
when reaching a leaf, we know whether it’s a heavy hitter

the sum of counts at any one level of the tree is \(t\)
no more than \(1/\theta\) of them may have frequency \(\geq \theta t\)
Efficiency: no more than \(1/\theta\) nodes of each level are expanded
Exercise
Formalize the algorithms above:

- For computing range-sum queries given CM-sketch
- Form finding all heavy hitters using range-sum queries

and tell their memory usage and update time
Other uses of CM-Sketch - Range-Sum queries

- Quantile computation: Given $i$, $\theta$, find for all $k$ the $q(k)$ such that

$$
q(k) \sum_{i=1}^{n} F[i] = k \theta \sum_{i=1}^{n} F[i]
$$

- Reverse, histogram computation: Given $f$, how many $i$’s have frequency $f$?

- Inner product of two streams

- ...
5. Counting in Sliding Windows

- Only last $n$ items matter
- Clear way to bound memory
- Natural in applications: emphasizes most recent data
- Data that is too old does not affect our decisions

Examples:
- Study network packets in the last day
- Detect top-10 queries in search engine in last month
- Analyze phone calls in last hours
Statistics on Sliding Windows

- Want to maintain mean, variance, histograms, frequency moments, hash tables, . . .
- SQL on streams. Extension of relational algebra
- Want quick answers to queries at all times
Basic Problem: Counting 1’s

Obvious algorithm, memory $n$:

- Keep window explicitly
- At each time $t$, add new bit $b$ to head, remove oldest bit $b'$ from tail,
- Add $b$ and subtract $b'$ from count

Fact:

$\Omega(n)$ memory bits are necessary to solve this problem exactly
Theorem: Estimating number of 1’s in a window of length $n$ with multiplicative error $\varepsilon$ is possible with $O\left(\frac{1}{\varepsilon} \log n\right)$ counters

$= O\left(\frac{1}{\varepsilon}(\log n)^2\right)$ bits of memory

Example:

- $n = 10^6; \varepsilon = 0.1 \rightarrow 200$ counters, 4000 bits
Idea: Exponential Histograms

Each bit has a timestamp - time at which it arrived
At time $t$, bits with timestamp $\leq t - n$ are expired
We have up to $k$ buckets of capacity 1, 2, 4, 8, ... 
Each bucket contains the number of 1s in a subwindow, up to its capacity
Errors: expired bits in the last bucket
1’s in last bucket $\leq$ (1’s in previous buckets) / $k$
Exponential Histograms

Init: Create empty set of buckets

Query: Return total number of bits in buckets - last bucket / 2
Exponential Histograms

Insert rule(bit \( b \)):

- If \( b \) is a 0, ignore it. Otherwise, if it’s a 1:
- Add a bucket with 1 bit and current timestamp \( t \) to the front
- for \( i = 0, 1, \ldots \)
  - If more than \( k \) buckets of capacity \( 2^i \),
    - merge two oldest as newest bucket of capacity \( 2^{i+1} \),
      with timestamp of the older one
- if oldest bucket timestamp \( < t - n \), drop it (all expired)
Memory Estimate

- Largest bucket needed: $k \sum_{i=0}^{C} 2^i \approx n \rightarrow C \approx \log(n/k)$
- Total number of buckets: $k \cdot (C + 1) \approx k \log(n/k)$
- Each bucket contains a timestamp only (perhaps its capacity, dep. on implementation)
- Timestamps are in $t - n \ldots t$: recycle timestamps mod $n$
- Memory is $O(k \log(n/k) \log n)$ bits; take $k = 1/2\epsilon$
Generalizations

Applies also to other natural aggregates:

- Variance
- Distinct elements (using Flajolet-Martin)
- Max, min
- Histograms
- Hash tables
- Frequency moments

and can be combined with CM-sketch
6. Distributed Sketching

Setting:
- Many sources generating streams concurrently
- No synchrony assumption
- Want to compute global statistics
- Streams can send short summaries to central
Merging sketches

Send the sketches, not the whole stream
Merging sketches

Mergeability

A sketch algorithm is **mergeable** if

- given two sketches $S_1$ and $S_2$ generated by the algorithm on two data streams $D_1$ and $D_2$,
- one can compute a sketch $S$ that answers queries correctly with respect to the concatenation of $D_1$ and $D_2$

Note: For frequency problems,

"for the concatenation" = "for all interleavings"
Merging sketches

All sketches we’ve seen are mergeable efficiently

- Bloom filters, Cohen, Flajolet-Martin, HyperLogLog
- SpaceSaving
- CM-sketch
- Exponential Histograms (though order dependent problem)

May require sites to use common random bits or hash functions
Perfect hash function: $f(i)$ cannot be guessed at all even from all other values of $f$

Storing $f : A \rightarrow B$ unfeasible for large $A$
Wrapping up. Hash functions (2)

- Cryptographic hash functions (MD5, SHA1, SHA256, or MurmurHash) should work well, but are costly.
- Even simpler functions like linear congruential may work well in practice if not in theory — but don’t use 32 bit integers if you plan to count billions!
- $O(\log n)$ bits to store such a function for $|A| = |B| = n$
- But we can’t “generate many of them”, e.g., to reduce variance

- Sometimes, analysis reveals that weaker notions of “good hash function” suffices
- E.g., pairwise independence suffices for CM-sketch: $f(i)$ independent of any other single $f(j)$
- (In general, will work if you use only Chebyshev or Markov)
- We can generate mutually independent, pairwise independent functions
- One can be stored with $O(\log n)$ bits
Wrapping up. Some stuff I left out

- Detecting duplicate documents
- Detecting near duplicates (LSH), minwise hashing, . . .
- Sketches for geometric problems. Clustering
- Graph sketches. Counting subgraphs
- Using HyperLogLog to estimate neighborhood functions of graphs
- Sketches that are linear projections. Metric embeddings. Dimensionality reduction
- Linear algebra. PCA. Singular Value Decomposition
Wrapping up. Last words

Approximation helps

Randomness helps

Some more tools in your toolbox

http://www.cs.upc.edu/~gavalda
8. References and resources

With apologies to all missing papers

General Surveys on Stream Algorithmics:

- **Survey by Liberty and Nelson:** [http://www.cs.yale.edu/homes/el327/papers/streaming_data_mining.pdf](http://www.cs.yale.edu/homes/el327/papers/streaming_data_mining.pdf)
- **A very general bibliography by K. Tufte:** [http://web.cecs.pdx.edu/~tufte/410-510DS/readings.htm](http://web.cecs.pdx.edu/~tufte/410-510DS/readings.htm)
- **Lecture notes by A. Chakrabarti:** [http://www.cs.dartmouth.edu/~ac/Teach/CS85-Fall09/Notes/lecnotes.pdf](http://www.cs.dartmouth.edu/~ac/Teach/CS85-Fall09/Notes/lecnotes.pdf)
- **Survey by G. Cormode:** [http://dimacs.rutgers.edu/~graham/pubs/papers/sk.pdf](http://dimacs.rutgers.edu/~graham/pubs/papers/sk.pdf)
Approximate counting

- The original Morris77 paper: http://dl.acm.org/citation.cfm?id=359627 also available here: http://www.inf.ed.ac.uk/teaching/courses/exc/reading/morris.pdf


- The application of Morris’ counters to counting n-grams, by Van Durme and Lall: http://www.cs.jhu.edu/~vandurme/papers/VanDurmeLallIJCAI09.pdf
8. References and resources

Large deviation bounds

- C. Shalizi list of references (much beyond the scope of this course): http://bactra.org/notebooks/large-deviations.html
Counting distinct elements

- Good general survey of distinct element counting up to 2008: Ahmed Metwally, Divyakant Agrawal, Amr El Abbadi: Why go logarithmic if we can go linear?: Towards effective distinct counting of search traffic. EDBT 2008: 618-629.

- Also general discussion on distinct element counting: http://highscalability.com/blog/2012/4/5/big-data-counting-how-to-count-a-billion-distinct-objects-.html

- Presentation including some sketches I didn’t mention: http://www.cs.upc.edu/~conrado/research/talks/aofa2012.pdf


8. References and resources

HyperLogLog and related for distinct element counting


- Flajolet’s contributions explained beautifully by J. Lumbroso: http://www.stat.purdue.edu/~mdw/ChapterIntroductions/ApproxCountingLumbroso.pdf
8. References and resources

HyperLogLog and related for distinct element counting (2)

- **A live demo of hyperloglog at the web above:** [http://content.research.neustar.biz/blog/hll.html](http://content.research.neustar.biz/blog/hll.html)

- **Important optimizations that I'd like to try:**
  [http://druid.io/blog/2014/02/18/hyperloglog-optimizations-for-real-world-systems.html](http://druid.io/blog/2014/02/18/hyperloglog-optimizations-for-real-world-systems.html). Also here:
8. References and resources

Heavy hitters - count-based approaches


- Good survey of heavy hitter algorithms. Radu Berinde, Graham Cormode, Piotr Indyk, Martin J. Strauss. Space-optimal Heavy Hitters with Strong Error Bounds

- Also very good survey: Graham Cormode, Marios Hadjieleftheriou. Finding Frequent Items in Data Streams. Proc. VLDB Endowment, 2008


8. References and resources

Count-Min sketch and related


- On Frugal Streaming, a neat sketch for estimating quantiles which I did not cover in the course:
  http://research.neustar.biz/2013/09/16/sketch-of-the-day-frugal-streaming/


- https://sites.google.com/site/countminsketch/

- https://tech.shareaholic.com/2012/12/03/the-count-min-sketch-how-to-count-over-large-keyspaces
Counting in Sliding Windows


Mergeability

- Discussions on mergeability are a bit all over. This is sort of an overview: http://research.microsoft.com/en-us/events/bda2013/mergeable-long.pptx
8. References and resources

Others (personal 1-slide selection)

- Computing SVD on streams, this will be important in streaming ML: Mina Ghashami, Edo Liberty, Jeff M. Phillips, David P. Woodruff, Frequent Directions: Simple and Deterministic Matrix Sketching. http://arxiv.org/abs/1501.01711
- This will also be important in streaming ML: Christos Boutsidis, Dan Garber, Zohar Karnin, Edo Liberty: Online Principal Component Analysis, SODA 2015. http://www.cs.yale.edu/homes/el327/papers/opca.pdf
8. References and resources

Resources

- Webgraph. Analysis of large graphs, contains the HyperANF and related code used for the Four-degrees-of-separation paper: http://webgraph.di.unimi.it/
8. References and resources

Resources

I have not used the following, so no guarantees of any kind (including that they still exist)

- **Python**: [https://pypi.python.org/pypi/hyperloglog/0.0.8](https://pypi.python.org/pypi/hyperloglog/0.0.8)
- **Ruby**: [https://rubygems.org/gems/hyperloglog](https://rubygems.org/gems/hyperloglog)
- **Perl**: [http://search.cpan.org/~hideakio/Algorithm-HyperLogLog-0.20/lib/Algorithm/HyperLogLog.pm](http://search.cpan.org/~hideakio/Algorithm-HyperLogLog-0.20/lib/Algorithm/HyperLogLog.pm)
- **JavaScript**: [http://cnpmjs.org/package/hyperloglog](http://cnpmjs.org/package/hyperloglog)
- **node.js**: [https://www.npmjs.org/package/streamcount](https://www.npmjs.org/package/streamcount)
- **https://github.com/eclesh/hyperloglog/blob/master/hyperloglog.go**