probabilistic algorithms to process MASSIVE data

Jérémie Lumbroso
INRIA Rocquencourt (Algorithms) / LIP6

April 10th, 2012
0. DATA EXPLOSION

- 340 million Tweets a day, 294 billion emails a day
- 35 hours of video uploaded to YouTube per minute
- one human genome: 3.2 billion letters
- NSA wants to build 150 Petabytes (150 million GB) to store personal data on people

Moore’s law: processing power doubles every 18 months

Sedgewick’s principle: Volumes and complexity of data increase faster than processing speed. We need ever better algorithms to keep pace.
April 2012: Google’s index contains 55 billion pages
Google processes 24 petabytes every day
= only fraction of 1 trillion existing web pages (in 2008)
**data stream model**

**Stream:** a (very large) sequence $S$ over (also very large) domain $\mathcal{D}$

$$S = s_1 \ s_2 \ s_3 \ \cdots \ s_\ell, \quad s_j \in \mathcal{D}$$

consider $S$ as a multiset

$$\mathcal{M} = m_1^{f_1} \ m_2^{f_2} \ \cdots \ m_n^{f_n}$$

Ex.: $S = \text{run} \ \text{sally} \ \text{run} \ \text{see} \ \text{sally} \ \text{run} \quad \Rightarrow \quad \mathcal{M} = \text{run}^3 \ \text{sally}^2 \ \text{see}^1$

Interested in **estimating** the following **quantitive** statistics:

— **A. Length** $:= \ell$

— **B. Cardinality** $:= \text{card}(m_i) \equiv n$ (distinct values)

— **C. Icebergs** $:= \# \ \text{elem. with relative frequency} \ f_v/\ell > \theta$

  [ where $\theta$ is any fixed threshold, like 50% ]

**Constraints:**

- very little processing memory
- on the fly (single pass + simple main loop)
- no statistical hypothesis
- accuracy within a few percentiles
Prelude: you need $\log_2 N$ bits to count up to $N$. 

The smallest unit of information, either 0 or 1, with 10 fingers/bits you can count up to $2^{10} - 1 = 512 + 256 + ... + 1$. 

$\frac{5}{17}$
Prelude: you need $\log_2 N$ bits to count up to $N$

**bit:** smallest unit of information, either 0 or 1

with 10 fingers/bits you can count up to $2^{10} - 1 = 512 + 256 + \ldots + 1$
1 + 8 + 16 = 25
6/17
$1 + 8 + 16 = 25$
1. **Approximate Counting (count length $\ell$)**

With 8 bits, can count up to $2^8 - 1 = 255$ elements.

**Question:** is it possible to count more??

⇒ YES, with coin flips!

**First idea:** increment every other time

- **Initialize:** $C := 0$
- **Increment:** with probability $1/2$, $C := C + 1$
- **Output:** $2 \cdot C$

$E[2 \cdot C] = n$ and only 3% error

**Limitation:** only save 1 bit (with 8 bits count to $2^{8+1} - 1 = 511$)

[ not very interesting, be honest! ]
Second idea: generalize, and increment 1 out of $2^k$

\[ \text{prob } 1/2^k = \text{flip } k \text{ coins, and all equal to 1} \]

- Initialize: $C := 0$
- Increment: with probability $1/2^k$, $C := C + 1$
- Output: $2^k \cdot C$

better: saves $k$ bits, i.e., count up to $2^{8+k}$ with 8 bits

Limitations:
- only saves linear number of bits
- for $k = 8$, error is 55%
- worst: always inaccurate for small values $< 2^k$

\[ \text{because smallest value returned is } 2^k \cdot C \]
Third idea: probability of increment depends on value of counter $C$

- Initialize: $C := 0$
- Increment: with probability $1/2^C$, $C := C + 1$
- Output: $2^C - 1$

Gets “harder” to increment:

Finally:
- accurate for small values
- with 8 bits count up to $2^{16}$ with 15% error

Morris 1978, Flajolet 1985
**application** to genetics: finding patterns in genomes

**Genome:** long sequence of letters \{\textit{A, C, G, T}\}

- count occurrences of all subwords of size \(k\)
  - [interested in non-occurring words + very frequent words]

**Example:**  \texttt{A A C T A A C G T A A A} for \(k = 2\)

<table>
<thead>
<tr>
<th>AA</th>
<th>AC</th>
<th>AG</th>
<th>AT</th>
<th>CA</th>
<th>CC</th>
<th>CG</th>
<th>CT</th>
<th>GA</th>
<th>GC</th>
<th>GG</th>
<th>GT</th>
<th>TA</th>
<th>TC</th>
<th>TG</th>
<th>TT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- AA occurs 4 times
- ...  
- GA is absent, and is called a \textit{nullomer}\ (as are AG, AT, etc.)
  - [significant: because in total randomness \textit{all patterns} would appear]

**Limitations of exact count:**
- for \(k = 13\), requires \(2\ GB\) of memory
- \(k > 14\) requires Approx. Counting!
patterns in **anthrax bacteria** genome (5.23 M)

distribution of sequences of nucleotides of size $k = 7, 8, 9, 10$

patterns in \textbf{anthrax bacteria} genome (5.23 M)

distribution of 5.23 M \textbf{random} strings

distribution of sequences of nucleotides of size $k = 7, 8, 9, 10$ source: Csúrös 2007, http://www.iro.umontreal.ca/~csuros/spectrum/
2. **DISTINCT elements**

Back to our stream:

\[ S = s_1 \ s_2 \ s_3 \ \cdots \ s_\ell, \quad s_j \in \mathcal{D} \]

We want the number \( n \) of **distinct elements** used.

**Ex.:** \( S = \text{run sally run see sally run} \) (3 distinct elements)

- **idea 1:** sort data; then same elements next to each other; scan sorted data and count distinct elements
- **idea 2:** have bag, for each \( s_j \), if not in bag, add it; then count number of elements in bag

**Bad ideas:** too much memory is used; at minimum \( O(n) \).
A weird way to use hash functions!!

**Definition:** a hash function $h$ is defined as

$$h : A^* \rightarrow [0, 1].$$

**Main idea.** With “good enough” hash functions, our data is uniformized.
Two neat things about the minimum

**Fact 1:** the minimum *not sensitive to repetitions*

\[
\min\{0.83, 0.32, 0.83, 0.83, 0.95, 0.74\} = \min\{0.83, 0.32, 0.95, 0.74\} = 0.32
\]

**Fact 2:** \(n\) uniform random variables in \([0, 1]\) have min. \(M \approx \frac{1}{n+1}\)

\[
\mathbb{E}_n[M] = \int_0^1 x \cdot n(1 - x)^{n-1} dx = \frac{1}{n+1}.
\]

**Minimum-counting algorithm (Bar-Yossef et al. 2002, L. 2010):**

- hash elements of stream to \([0, 1]\) values
- take the minimum \(M\)
- return \(1/M - 1\)

With some optimizations, you can obtain 3% error with only 4kB!
AltaVista: remove near-duplicates

Broder (1997) uses similarity measure

\[ S(A, B) = \frac{|A \cap B|}{|A \cup B|} \]

- \( S(A, B) = 0 \): sets disjoint
- \( S(A, B) = 1 \): sets overlap

if \( S(A, B) > 0.99 \), consider documents \( A \) and \( B \) are same

⇒ eliminate near duplicates!
with Minimum-counting algorithm... easy

\[ S(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cup B| - |A| - |B|}{|A \cup B|} \]

and, if you note \( h(A) \) and \( h(B) \) the streams where you apply the hash function \( h \) to \( A \) and \( B \),

- \( |A| = 1/ \min(h(A)) - 1 \)
- \( |B| = 1/ \min(h(B)) - 1 \)
- \( |A \cup B| = 1/ \min(h(A), h(B)) - 1 \)

so only need to keep the minimum for each document, then only \( O(1) \) operations to compare to documents!

compare 10^5 documents of size 10^5 with each-other in only minutes instead of days
3. EPILOGUE

Many other applications:

- **Network security:**
  detect attacks (denial of service), or the spreading of worms/spam,...

- **Data mining:** document classification, ...

- **Databases:** query optimization

- **Distributed:** censor networks