Data Sketches

Lecturer: Jiaheng Lu
Autumn 2016
Data Streams

Traditional DBMS: data stored in *finite, persistent data sets*

Data Streams: distributed, continuous, unbounded, rapid, time varying, noisy, . . .

Data-Stream Management: variety of modern applications

- Network monitoring and traffic engineering
- Telecom call-detail records
- Financial applications
- Manufacturing processes
- Web logs and clickstreams
- Other massive data sets…
Massive Data Streams

• Data is *continuously growing* faster than our ability to store or index it

• There are 3 Billion **Telephone Calls** in US each day, 30 Billion emails daily, 1 Billion SMS, IMs

• **Scientific data**: NASA's observation satellites generate billions of readings each per day

• **IP Network Traffic**: up to 1 Billion packets per hour per router. Each ISP has many (hundreds) routers!

• Whole **genome sequences** for many species now available: each megabytes to gigabytes in size
Massive Data Stream Analysis

Must analyze this massive data:

- Scientific research (monitor environment, species)
- System management (spot faults, drops, failures)
- Business intelligence (marketing rules, new offers)
- For revenue protection (phone fraud, service abuse)
Three models of data streams

- **Time-series** model: the update arrives in sorted order

- **Cash-register** model: the update arrives in some arbitrary order

- **Turnstile** model: items which have previously been observed can be subsequently been removed
Sketches for data streams

- Not every problem can be solved with sampling
  - Example: counting how many distinct items in the dataset
  - If a large fraction of items aren’t sampled, don’t know if they are all same or all different

- Other techniques take advantage that the algorithm can “see” all the data even if it can’t “remember” it all
Main properties of a sketch

- Queries supported
  - Distinct Value, Min, Max, Average

- Sketch size

- Update speed

- Query time

- Sketch initialization
Bloom Filters

• **Bloom filters** compactly encode set membership
  • E.g. store a list of many long URLs compactly
  • random hash functions map items to a bit vector
• **Update**: Set all hashed entries to 1 to indicate item is present
• **Query**: Is $x$ in the set?
  – Return “yes” if all $k$ bits $x$ maps to are 1.
• Duplicate insertions do not change Bloom filters
• Can be **merged** by OR-ing vectors (of same size)
### Example

**Initial State**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Add x(0,2,6)**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Add y(0,3,9)**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Contain m(1,3,9)**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Contain n(0,2,9)**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**False Positives**
Bloom Filters Analysis

- If item is in the set, then always return “yes”
  - No false negative
- If item is not in the set, may return “yes” with a probability.
  - Prob. that a bit is 0: \( \left(1 - \frac{1}{m}\right)^{kn} \)
  - Prob. that all \( k \) bits are 1:
    \[
    \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \exp(k \ln(1 - e^{kn/m}))
    \]
  - Setting \( k = \frac{m}{n} \ln 2 \) minimizes this probability to \( 0.6185m/n \).
- For false positive prob \( \delta \), Bloom filter needs \( O(n \log \frac{1}{\delta}) \) bits.
Summary of Bloom Filters

- Given a large set of elements $S$, efficiently check whether a new element is in the set.
- Bloom filters use hash functions to check membership
  - If $a$ is in $S$, return TRUE with probability 1
  - If $a$ is not in $S$, return FALSE with high probability
- False positive error depends on $|S|$, number of bits in the memory and number of hash functions
Bloom Filters Applications

- Bloom Filters widely used in “big data” applications
  - Many problems require storing a large set of items
- Can generalize to allow deletions
  - Swap bits for counters: increment on insert, decrement on delete
  - If no duplicates, small counters suffice: 4 bits per counter
- Bloom Filters are still an active research area
  - Often appear in networking conferences
  - Also known as “signature files” in DB
Count-Min Sketch

- Count Min sketch encodes item counts
  - Allows estimation of frequencies (e.g. for selectivity estimation)
  - Some similarities in appearance to Bloom filters
- Model input data as a matrix
  - Create a small summary as an array of $w \times d$ in size
  - Use $d$ hash function to map vector entries to $[1..w]$
Count-Min Sketch

- **Query**: estimate by taking $\min_{1 \leq j \leq d} \{C_j[h_j(i)]\}$
- Never underestimate
- Will bound the error of overestimation
### Count-Min example

<table>
<thead>
<tr>
<th></th>
<th>Initial</th>
<th>Insert (x,5)</th>
<th>Insert (y,3)</th>
<th>Insert (z,1)</th>
<th>Query x? min = 5</th>
<th>Query z? min = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0</td>
<td>0 0 5</td>
<td>0 0 8</td>
<td>0 0 9</td>
<td>0 0 9</td>
<td>0 0 9</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
<td>5 0 0</td>
<td>5 0 3</td>
<td>5 0 4</td>
<td>5 0 4</td>
<td>5 0 4</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
<td>5 0 0</td>
<td>5 3 0</td>
<td>6 3 0</td>
<td>6 3 0</td>
<td>6 3 0</td>
</tr>
</tbody>
</table>

- **h1(x) = 2**
- **h2(x) = 0**
- **h3(x) = 0**
- **h1(y) = 2**
- **h2(y) = 2**
- **h3(y) = 1**
- **h1(z) = 2**
- **h2(z) = 2**
- **h3(z) = 0**

Over-estimation
Count-Min Sketch Analysis

• Focusing on first row:
  • $x[i]$ is added to $CM[1, h_1(i)]$
  • but $x[j], j \neq i$ is added to $CM[1, h_1(i)]$ with prob. $1/w$
  • The expected error is $x[j]/w$
  • The total expected error is $\frac{\sum_{j \neq i} x[j]}{w} \leq \frac{||x||_1}{w}$
  • By Markov inequality, $Pr[\text{error} > \frac{2 \cdot ||x||_1}{w}] < \frac{1}{2}$

• By taking the minimum of $d$ rows, this prob. is $\left(\frac{1}{2}\right)^d$

• To give an $\varepsilon ||x||_1$ error with prob. $1 - \delta$, the sketch needs to have size $\frac{2}{\varepsilon} \times \log \frac{1}{\delta}$
Count Sketch

- Second hash function: \( g_k \) maps each \( i \) to +1 or -1
### Count-Sketch example

<table>
<thead>
<tr>
<th>Insert (x, 5)</th>
<th>Insert (y, 3)</th>
<th>Insert (z, 1)</th>
<th>Query z? median = 1</th>
<th>Query x? median = −4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 5</td>
<td>0 0 8</td>
<td>0 0 7</td>
<td>0 0 7</td>
<td>0 0 7</td>
</tr>
<tr>
<td>-5 0 0</td>
<td>-5 0 3</td>
<td>-5 1 3</td>
<td>-5 1 3</td>
<td>-5 1 3</td>
</tr>
<tr>
<td>-5 0 0</td>
<td>-5 -3 0</td>
<td>-4 -3 0</td>
<td>-4 -3 0</td>
<td>-4 -3 0</td>
</tr>
</tbody>
</table>

- \( h_1(x) = 2 \)
- \( h_2(x) = 0 \)
- \( h_3(x) = 0 \)
- \( g_1(x) = 1 \)
- \( g_2(x) = -1 \)
- \( g_3(x) = -1 \)

- \( h_1(y) = 2 \)
- \( h_2(y) = 2 \)
- \( h_3(y) = 1 \)
- \( g_1(y) = 1 \)
- \( g_2(y) = 1 \)
- \( g_3(y) = -1 \)

- \( h_1(z) = 2 \)
- \( h_2(z) = 1 \)
- \( h_3(z) = 0 \)
- \( g_1(z) = -1 \)
- \( g_2(z) = 1 \)
- \( g_3(z) = 1 \)

**under-estimation**
Count Sketch Analysis

- Focusing on first row:
  - $x[i]g_1(i)$ is always added to $C[1, h_1(i)]$
    - We return $C[1, h_1(i)]g_1(i) = x[i]$
  - $x[j]g_1(j), j \neq i$ is added to $C[1, h_1(i)]$ with prob. $1/w$
  - The expected error is $x[j]E[g_1(j)]/w = 0$
  - $E[\text{total error}] = 0, E[|\text{total error}|] \leq \frac{||x||_2}{\sqrt{w}}$
    - By Chebyshev inequality, $\Pr[|\text{total error}|] > \frac{2||x||_2}{\sqrt{w}} < \frac{1}{4}$
    - By taking the median of $d$ rows, this prob. is $\left(\frac{1}{4}\right)^O(d)$
  - To give an $\varepsilon||x||_2$ error with prob. $1 - \delta$, the sketch needs to have size $O\left(\frac{1}{\varepsilon^2 \log \frac{1}{\delta}}\right)$
Count Sketch Analysis

- $E[|\text{total error}|] \leq \frac{||x||_2}{\sqrt{w}}$

- Chebyshev inequality
  - Let $X$ be a random variable with finite expected value $\mu$ and finite non-zero variance $\sigma^2$. Then for any real number $k > 0$,
    - $\Pr(|x - \mu| \geq k\delta) \leq \frac{1}{k^2}$
  - By Chebyshev inequality, $\Pr[|\text{total error}|] > \frac{2 \cdot ||x||_2}{\sqrt{w}} < \frac{1}{4}$
  - By taking the median of $d$ rows, this prob. is $\left(\frac{1}{4}\right)^{O(d)}$
  - To give an $\varepsilon||x||_2$ error with prob. $1 - \delta$, the sketch needs to have size $O\left(\frac{1}{\varepsilon^2 \log \frac{1}{\delta}}\right)$
Count-Min Sketch vs Count Sketch

- **Count-Min:**
  - Size: $O(wd)$
  - Error: $||x||_1/w$

- **Count Sketch:**
  - Size: $O(wd)$
  - Error: $||x||_2/\sqrt{w}$

Count Sketch is better when $||x||_2 < ||x||_1/\sqrt{w}$

Other benefits:

- Unbiased estimator
Join size estimation

Consider two data sets F and G given as pairs (key, frequency):

- \( F\{(1,2),(0,1),(4,1),(3,2)\} \) and \( G\{(2,1),(3,1),(0,2)\} \)

Please estimate the size of join \( |F \bowtie G| \) of two sets using Count-sketch with a \( 2 \times 2 \) matrix.
## Count-Min join example

<table>
<thead>
<tr>
<th></th>
<th>Insert (1,2)</th>
<th>Insert (0,1)</th>
<th>Insert (4,1)</th>
<th>Insert (3,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial</strong></td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>h1(0) = 0</td>
<td>h1(0) = 0</td>
<td>h1(4) = 0</td>
<td>h1(3) = 1</td>
</tr>
<tr>
<td></td>
<td>h2(0) = 0</td>
<td>h2(0) = 0</td>
<td>h2(4) = 1</td>
<td>h2(3) = 1</td>
</tr>
<tr>
<td></td>
<td>h3(0) = 1</td>
<td>h3(0) = 1</td>
<td>h3(4) = 1</td>
<td>h2(3) = 0</td>
</tr>
</tbody>
</table>

Estimation is \( \min(12, 15, 12) = 12 \),

But real number is 4.
Distinct Value Estimation

- **Problem**: Find the *number of distinct values* in a stream of values with domain.
- **Applications**
  - Statistics: number of *species or classes* in a population
  - Databases: Important for query optimizers
  - *Network monitoring*: distinct destination IP addresses, source/destination pairs, requested URLs, etc.
- **Example (N=64)**
  - Data: 3 2 5 3 2 1 7 5 1 2 3 7
  - *Number of distinct values*: 5
- **Count-Min is not good for this job. Why?**
FM Sketch  [Flajolet, Martin’85]

- Estimates number of distinct inputs (count distinct)
- Uses hash function mapping input items to $i$ with prob
  - i.e. $\Pr[h(x) = 1] = \frac{1}{2}$, $\Pr[h(x) = 2] = \frac{1}{4}$, $\Pr[h(x)=3] = \frac{1}{8} \ldots$
- Easy to construct $h()$ from a uniform hash function by counting trailing zeros
- Maintain FM Sketch = bitmap array of $L = \log N$ bits
  - Initialize bitmap to all 0s
  - For each incoming value $x$, set $FM[h(x)] = 1$

$$x = 5 \rightarrow h(x) = 3$$

```
0 0 0 1 0 0
```
FM Sketch  [Flajolet, Martin’85]

• Estimates number of distinct inputs (\textit{count distinct})
• Uses hash function mapping input items to $i$ with prob
  • i.e. $\Pr[h(x) = 1] = \frac{1}{2}$, $\Pr[h(x) = 2] = \frac{1}{4}$, $\Pr[h(x)=3] = \frac{1}{8}$ ...

• Define $\text{Tail}(h(x)) = \text{number of trailing consecutive 0}$
  • $\text{Tail}(101001) = 0$
  • $\text{Tail}(101010) = 1$
  • $\text{Tail}(101100) = 2$
Algorithm

- For all x
  - Compute $k(x) = \text{Tail}(h(x))$
  - Let $R = \max k(x) + 1$
  - Return $F = c = 2^R$
FM Sketch Analysis

- If \( d \) distinct values, expect \( d/2 \) map to \( \text{FM}[1] \), \( d/4 \) to \( \text{FM}[2] \)...

- Let \( R \) = position of rightmost zero in FM, indicator of \( \log(d) \)
- Basic estimate \( d = c2^R \) for scaling constant
FM Sketch Properties

- With $O(1/\varepsilon^2 \log 1/\delta)$ copies, get $(1 \pm \varepsilon)$ accuracy with probability at least $1 - \delta$ [Bar-Yossef et al.’02], [Ganguly et al.’04]

- 10 copies gets $\approx 30\%$ error, 100 copies $< 10\%$ error
Practicality

- Algorithms discussed here are quite simple and very fast
  - Sketches can easily process millions of updates per second on standard hardware
  - Limiting factor in practice is often I/O related
- Implemented in several practical systems:
  - AT&T’s Gigascope system on live network streams
  - Sprint’s CMON system on live streams
  - Google’s log analysis
Conclusions

• **Data Streaming:** Major departure from traditional persistent database paradigm
  • Fundamental re-thinking of models, assumptions, algorithms, system architectures, …

• Simple tools from *approximation and/or randomization* play a critical role in effective solutions
  • Sampling, sketches (CM, FM, …), …
  • Simple, yet powerful, ideas with great reach