Flajolet-Martin Sketches

Omar Zeidan
Count-distinct problem

Given is a stream of elements from a finite dataset:

```
a, c, d, a, i, d, f, m, i, ...
```

How many distinct elements are in the set?

```
SELECT COUNT(DISTINCT x)
FROM table;
```
Naive Approach

- a, c, d, a, i, d, f, m, i, ...

- Have we seen this element?
  - yes
    - Do nothing
  - no
    - Increment counter, remember element

- $O(n)$ in memory $\rightarrow$ large datasets don’t fit into RAM
- Performance depends on data structure
Flajolet-Martin algorithm [1]

Better approach: smart estimation (sketch)

Idea: count leading zeros

<table>
<thead>
<tr>
<th>Bit Pattern</th>
<th>Probability (if uniformly distributed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0XXXXXXX</td>
<td>1/2</td>
</tr>
<tr>
<td>00XXXXXX</td>
<td>1/4</td>
</tr>
<tr>
<td>000XXXXX</td>
<td>1/8</td>
</tr>
</tbody>
</table>

example: 4 leading zeros $\rightarrow 2^4 = 16$
Flajolet-Martin algorithm

1. Hash element (uniformly)
   \[ a \rightarrow 00101101 \]

2. Determine count of leading zeros (clz)
   \[ 00101101 \rightarrow 2 \]

3. Set bitmap at index clz
   \[ \begin{array}{ccccccccc}
   0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
   \end{array} \rightarrow \begin{array}{ccccccccc}
   0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
   \end{array} \]
Flajolet-Martin algorithm

After every element is processed:

Get smallest index that is not set: 4

Final estimate: \( 2^R / \phi \) (\( R \) is the smallest index, \( \phi \) is the correction factor)

In this case: 20,684929736
Flajolet-Martin algorithm

Results:

- Sensible estimate of the ‘cardinality’
- Constant memory footprint (typically 4 or 8 bytes)

But:

- Still some room of improvement for the accuracy
- Results have a high variation (standard deviation $\approx 1.12$)
(Hyper)LogLog [2]

Builds on Flajolet-Martin algorithm

Improvements:

1. Split elements into $2^p$ substreams, where $p$ is the 'precision'

```
first p bits
10111010  ➔  Substream 5
00101101  ➔  Substream 1
```
(Hyper)LogLog

Process every substream similarly as with Flajolet-Martin

2. Start counting the leading zeros after the first $p$ bits

$$\begin{array}{c}
10100110 \\
\rightarrow \\
2
\end{array}$$

3. Remember highest clz for every substream
HyperLogLog

4. Calculate the estimate, the harmonic mean of all maximum clzs:

\[ \text{Harmonic Mean} \rightarrow \alpha_m \cdot m^2 \cdot \left( \sum_{j=1}^{m} 2^{-M[j]} \right) \]

\[ \alpha_m := \left( m \int_0^\infty \left( \log_2 \left( \frac{2 + u}{1 + u} \right) \right)^m du \right)^{-1} \]

(Precalculated in my implementation)
HyperLogLog

Results

“The HyperLogLog algorithm is able to estimate cardinalities of $> 10^9$ with a typical accuracy of 2%, using 1.5 kB of memory.” [2]

- Accuracy is improved, outliers ‘hurt’ the result less
- Memory footprint is still very small:

  Typically $4 \leq p \leq 16 \rightarrow$ max. $2^{16} = 65536$ buckets

  Typically 32 bit hashes $\rightarrow$ 5 bits per bucket (at $p = 16$)

  $\rightarrow$ memory footprint is 40kb at highest precision
Some more improvements:

1. Small range corrections for small $n$ (linear counting)
2. Large range corrections when $n$ starts to approach $2^{32}$
HyperLogLog++ [3]

Does everything HyperLogLog does, and:

- **Uses a 64 bit hash function:**
  - Stays accurate for higher cardinalities, no need for large range corrections

- **Eliminates bias:**
  - HLL is biased for small cardinalities
  - Empirically measure the bias and subtract it from our estimate

Flajolet-Martin Sketches
HyperLogLog++

- Uses the bias corrected estimate OR linear counting, depending on the cardinality

Effects of the bias correction [3]
HyperLogLog++

- Compresses the buckets in a complex manner to further save more memory
Memory footprint

- Flajolet-Martin:
  
  8 bytes

- HyperLogLog:
  
  $2^p \times 5 \text{ bits}$

- (my) HyperLogLog++:
  
  $2^p \times 6 \text{ bits (} + \text{ bias elimination data)}$
Benchmarks: Performance (one element)

At sample size 67,108,864

- Flajolet-Martin:  
  32.6233 ns  25.9636 ns (without hash)

- HyperLogLog/HyperLogLog++:  
  32.4092 ns  25.8448 ns (without hash)
Benchmarks: Accuracy

At cardinality 42000 with 10000 data points and p = 14

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean Relative Error (%)</th>
<th>Standard Deviation of Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flajolet-Martin</td>
<td>59.5372</td>
<td>41.2034</td>
</tr>
<tr>
<td>HyperLogLog</td>
<td>2.17655</td>
<td>0.628657</td>
</tr>
<tr>
<td>HyperLogLog++</td>
<td>0.533857</td>
<td>0.407194</td>
</tr>
</tbody>
</table>
Benchmarks: HyperLogLog++ Accuracy

At cardinality 65536 with 2000 data points

<table>
<thead>
<tr>
<th>p</th>
<th>Mean Relative Error (%)</th>
<th>Standard Deviation of Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>21.8203</td>
<td>17.9068</td>
</tr>
<tr>
<td>6</td>
<td>10.5101</td>
<td>8.26347</td>
</tr>
<tr>
<td>8</td>
<td>5.28637</td>
<td>5.28815</td>
</tr>
<tr>
<td>10</td>
<td>2.68807</td>
<td>2.13092</td>
</tr>
<tr>
<td>12</td>
<td>1.25001</td>
<td>0.935579</td>
</tr>
<tr>
<td>14</td>
<td>0.567714</td>
<td>0.424141</td>
</tr>
<tr>
<td>16</td>
<td>0.257173</td>
<td>0.195455</td>
</tr>
</tbody>
</table>
Sources


[2] Flajolet, Philippe; Fusy, Éric; Gandouet, Olivier; Meunier, Frédéric (2007). "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm"