Frequency Counts over Data Streams

Gurmeet Singh Manku

Stanford University, USA
The Problem ...

- Identify all elements whose current frequency exceeds support threshold $s = 0.1\%$. 

Stream
A Related Problem ...

Stream

- Identify all subsets of items whose current frequency exceeds $s = 0.1\%$.

Frequent Itemsets / Association Rules
Applications

Flow Identification at IP Router [EV01]

Iceberg Queries [FSGM+98]

Iceberg Datacubes [BR99 HPDW01]

Association Rules & Frequent Itemsets [AS94 SON95 Toi96 Hid99 HPY00 ...]
Presentation Outline ...

1. Lossy Counting
2. Sticky Sampling
3. Algorithm for Frequent Itemsets
Algorithm 1: Lossy Counting

Step 1: Divide the stream into ‘windows’

Is window size a function of support s? Will fix later...
Lossy Counting in Action ...

At window boundary, decrement all counters by 1
Lossy Counting continued ...

At window boundary, decrement all counters by 1
Error Analysis

How much do we undercount?

If current size of stream = $N$
and window-size = $1/e$

then frequency error $\leq \#\text{windows} = eN$

Rule of thumb:
Set $e = 10\%$ of support $s$

Example:
Given support frequency $s = 1\%$,
set error frequency $e = 0.1\%$
Output:
Elements with counter values exceeding $sN - eN$

Approximation guarantees
- Frequencies underestimated by at most $eN$
- No false negatives
- False positives have true frequency at least $sN - eN$

How many counters do we need?
Worst case: $1/e \log (eN)$ counters   [See paper for proof]
Enhancements ...

Frequency Errors
For counter (X, c), true frequency in [c, c+eN]

Trick: Remember window-id’s
For counter (X, c, w), true frequency in [c, c+w-1]

If (w = 1), no error!

Batch Processing
Decrements after k windows
Algorithm 2: Sticky Sampling

- Create counters by sampling
- Maintain exact counts thereafter

What rate should we sample?
Sticky Sampling contd...

For finite stream of length $N$

$$\text{Sampling rate} = \frac{2}{Ne} \log \frac{1}{(s\delta)}$$

Same Rule of thumb:
Set $e = 10\%$ of support $s$

Example:
Given support threshold $s = 1\%$,
set error threshold $e = 0.1\%$
set failure probability $\delta = 0.01\%$

Output:
Elements with counter values exceeding $sN - eN$

Approximation guarantees (probabilistic)
- Frequencies underestimated by at most $eN$
- No false negatives
- False positives have true frequency at least $sN - eN$

Same error guarantees as Lossy Counting
but probabilistic

Same Rule of thumb:
Set $e = 10\%$ of support $s$
Sampling rate?

Finite stream of length $N$

Sampling rate: $\frac{2}{Ne} \log \frac{1}{(s\delta)}$

Infinite stream with unknown $N$

Gradually adjust sampling rate (see paper for details)

In either case,

Expected number of counters $= \frac{2}{\epsilon} \log \frac{1}{s\delta}$

Independent of $N!$
Sticky Sampling Expected: \( \frac{2}{\varepsilon} \log \frac{1}{s \delta} \)
Lossy Counting Worst Case: \( \frac{1}{\varepsilon} \log \varepsilon N \)

Support \( s = 1\% \)
Error \( e = 0.1\% \)
From elements to *sets* of elements ...
Frequent Itemsets Problem ...

Stream

- Identify all *subsets of items* whose current frequency exceeds $s = 0.1\%$.

Frequent Itemsets $\Rightarrow$ Association Rules
Three Modules

- TRIE
- BUFFER
- SUBSET-GEN
Module 1: **TRIE**

Compact representation of frequent itemsets in lexicographic order.

Sets with frequency counts

- 50
- 40
- 30
- 45
- 32
- 42
- 31
- 29
Module 2: BUFFER

In Main Memory

Compact representation as sequence of ints
Transactions sorted by item-id
Bitmap for transaction boundaries
Module 3: SUBSET-GEN

Frequency counts of subsets in lexicographic order
Problem: Number of subsets is exponential!
SUBSET-GEN Pruning Rules

A-priori Pruning Rule

If set S is infrequent, every superset of S is infrequent.

Lossy Counting Pruning Rule

At each 'window boundary' decrement TRIE counters by 1.

Actually, 'Batch Deletion':

At each 'main memory buffer' boundary, decrement all TRIE counters by b.

See paper for details ...
Bottlenecks ...

Consumes main memory

Consumes CPU time
Design Decisions for Performance

TRIE

Main memory bottleneck

Compact linear array
  → (element, counter, level) in preorder traversal
  → No pointers!

Tries are on disk
  → All of main memory devoted to BUFFER

Pair of tries
  → old and new (in chunks)
  mmap() and madvise()

SUBSET-GEN

CPU bottleneck

Very fast implementation
  → See paper for details
Experiments ... 

IBM synthetic dataset T10.I4.1000K

<table>
<thead>
<tr>
<th>N</th>
<th>Avg Tran Size</th>
<th>Input Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Million</td>
<td>10</td>
<td>49MB</td>
</tr>
</tbody>
</table>

IBM synthetic dataset T15.I6.1000K

<table>
<thead>
<tr>
<th>N</th>
<th>Avg Tran Size</th>
<th>Input Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Million</td>
<td>15</td>
<td>69MB</td>
</tr>
</tbody>
</table>

Frequent word pairs in 100K web documents

<table>
<thead>
<tr>
<th>N</th>
<th>Avg Tran Size</th>
<th>Input Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>100K</td>
<td>134</td>
<td>54MB</td>
</tr>
</tbody>
</table>

Frequent word pairs in 806K Reuters newsreports

<table>
<thead>
<tr>
<th>N</th>
<th>Avg Tran Size</th>
<th>Input Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>806K</td>
<td>61</td>
<td>210MB</td>
</tr>
</tbody>
</table>
What do we study?

For each dataset

- Support threshold \( s \)
- Length of stream \( N \)
- BUFFER size \( B \)
- Time taken \( t \)

\[ \text{Set } e = 10\% \text{ of support } s \]

Three independent variables
- Fix one and vary two

Measure time taken
Varying support $s$ and BUFFER $B$

IBM 1M transactions     Reuters 806K docs

Fixed: Stream length $N$
Varying: BUFFER size $B$, Support threshold $s$
Varying length N and support s

IBM 1M transactions

Reuters 806K docs

Fixed: BUFFER size B
Varying: Stream length N
          Support threshold s
Varying BUFFER B and support s

IBM 1M transactions

Reuters 806K docs

Fixed: Stream length N
Varying: BUFFER size B
            Support threshold s
Comparison with fast A-priori

<table>
<thead>
<tr>
<th>Support</th>
<th>A-Priori</th>
<th>Our Algorithm with 4MB Buffer</th>
<th>Our Algorithm with 44MB Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Memory</td>
<td>Time</td>
</tr>
<tr>
<td>0.001</td>
<td>99 s</td>
<td>82 MB</td>
<td>111 s</td>
</tr>
<tr>
<td>0.002</td>
<td>25 s</td>
<td>53 MB</td>
<td>94 s</td>
</tr>
<tr>
<td>0.004</td>
<td>14 s</td>
<td>48 MB</td>
<td>65 s</td>
</tr>
<tr>
<td>0.006</td>
<td>13 s</td>
<td>48 MB</td>
<td>46 s</td>
</tr>
<tr>
<td>0.008</td>
<td>13 s</td>
<td>48 MB</td>
<td>34 s</td>
</tr>
<tr>
<td>0.010</td>
<td>14 s</td>
<td>48 MB</td>
<td>26 s</td>
</tr>
</tbody>
</table>

Dataset: IBM T10.I4.1000K with 1M transactions, average size 10.
Comparison with Iceberg Queries

Query: Identify all word pairs in 100K web documents which co-occur in at least 0.5% of the documents.

[FSGM+98] multiple pass algorithm:
7000 seconds with 30 MB memory

Our single-pass algorithm:
4500 seconds with 26 MB memory

Our algorithm would be much faster if allowed multiple passes!
Lessons Learnt ...

Optimizing for \#passes is wrong!

Small support $s \Rightarrow$ Too many frequent itemsets!  
Time to redefine the problem itself?

Interesting combination of Theory and Systems.
Work in Progress ...

Frequency Counts over Sliding Windows

Multiple pass Algorithm for Frequent Itemsets

Iceberg Datacubes
Summary

Lossy Counting: A Practical algorithm for online frequency counting.

First ever single pass algorithm for Association Rules with user specified error guarantees.

Basic algorithm applicable to several problems.