Finding Frequent Items in Data Streams

Graham Cormode
graham@research.att.com

Marios Hadjieleftheriou (AT&T)
S. Muthukrishnan (Rutgers)
Radu Berinde & Piotr Indyk (MIT)
Martin Strauss (U. Michigan)
Data Streams

- Many large sources of data are best modeled as data streams
  - E.g. streams of network packets, defining traffic distributions
- Impractical and undesirable to store and process all data exactly
- Instead, seek algorithms to find approximate answers
  - With one pass over data, quickly build a small summary
- Active research area for last decade, history goes back 30 years
The Frequent Items Problem

♦ The Frequent Items Problem (aka Heavy Hitters): given stream of N items, find those that occur most frequently
♦ E.g. Find all items occurring more than 1% of the time
♦ Formally “hard” in small space, so allow approximation
♦ Find all items with count $\geq \phi N$, none with count $< (\phi - \epsilon)N$
  – Error $0 < \epsilon < 1$, e.g. $\epsilon = 1/1000$
  – Related problem: estimate each frequency with error $\pm \epsilon N$
Why Frequent Items?

♦ A natural question on streaming data
  – Track bandwidth hogs, popular destinations etc.
♦ The subject of much streaming research
  – Scores of papers on the subject
♦ A core streaming problem
  – Many streaming problems connected to frequent items
    (itemset mining, entropy estimation, compressed sensing)
♦ Many practical applications
  – Search log mining, network data analysis, DBMS optimization
This Talk

♦ A brief history of the frequent items problem
♦ A tour of some of the most popular algorithms
  – Counter-based algorithms: Frequent, LossyCounting, SpaceSaving
  – Sketch algorithms: Count-Min Sketch, Count Sketch
♦ Experimental comparison of algorithms
♦ Extensions, new results and future directions
Data Stream Models

- We model data streams as sequences of simple tuples
- Complexity arises from massive length of streams
- **Arrivals only streams:**
  - Example: \((x, 3), (y, 2), (x, 2)\) encodes the arrival of 3 copies of item \(x\), 2 copies of \(y\), then 2 copies of \(x\).
  - Could represent eg. packets on a network; power usage
- **Arrivals and departures:**
  - Example: \((x, 3), (y, 2), (x, -2)\) encodes final state of \((x, 1), (y, 2)\).
  - Can represent fluctuating quantities, measure differences between two distributions, or represent general signals
The Start of The Problem?

[J.Alg 2, P208-209] Suppose we have a list of $n$ numbers, representing the “votes” of $n$ processors on the result of some computation. We wish to decide if there is a majority vote and what the vote is.

♦ Does not require a streaming solution, but first solutions were
MAJORITY algorithm

- **MAJORITY** algorithm solves the problem in arrivals only model.
- **Start** with a counter set to zero. For each item:
  - If counter is zero, pick up the item, set counter to 1
  - Else, if item is same as item in hand, increment counter
  - Else, decrement counter
- If there is a majority item, it is in hand
  - **Proof outline**: each decrement pairs up two different items and cancels them out
  - Since majority occurs \( > \frac{N}{2} \) times, not all of its occurrences can be canceled out.
"Frequent" algorithm

<table>
<thead>
<tr>
<th>6</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
</table>

- **FREQUENT** generalizes **MAJORITY** to find up to \( k \) items that occur more than \( 1/k \) fraction of the time
- Keep \( k \) different candidates in hand. For each item in stream:
  - If item is monitored, increase its counter
  - Else, if \(< k \) items monitored, add new item with count 1
  - Else, decrease all counts by 1
Frequent Analysis

- **Analysis**: each decrease can be charged against $k$ arrivals of different items, so no item with frequency $N/k$ is missed
- Moreover, $k=1/\varepsilon$ counters estimate frequency with error $\varepsilon N$
  - Not explicitly stated until later [Bose et al., 2003]

- **Some history**: First proposed in 1982 by Misra and Gries, rediscovered twice in 2002
  - Later papers showed how to make fast implementations
Lossy Counting

- **LossyCounting** algorithm proposed in [Manku, Motwani ’02]
- **Simplified version:**
  - Track items and counts
  - For each block of $1/\varepsilon$ items, merge with stored items and counts
  - Decrement all counts by one, delete items with zero count
- Easy to see that counts are accurate to $\varepsilon N$
- Analysis shows $O(1/\varepsilon \log \varepsilon N)$ items are stored
- Full version keeps extra information to reduce error
SpaceSaving Algorithm

“SpaceSaving” algorithm [Metwally, Agrawal, El Abaddi 05] merges Lossy Counting and FREQUENT algorithms

- Keep $k = \frac{1}{\varepsilon}$ item names and counts, initially zero
- Count first $k$ distinct items exactly
- On seeing new item:
  - If it has a counter, increment counter
  - If not, replace item with least count, increment count
SpaceSaving Analysis

- Smallest counter value, \( \min \), is at most \( \varepsilon n \)
  - Counters sum to \( n \) by induction
  - \( 1/\varepsilon \) counters, so average is \( \varepsilon n \): smallest cannot be bigger
- True count of an uncounted item is between \( 0 \) and \( \min \)
  - Proof by induction, true initially, \( \min \) increases monotonically
  - Hence, the count of any item stored is off by at most \( \varepsilon n \)
- Any item \( x \) whose true count \( > \varepsilon n \) is stored
  - By contradiction: \( x \) was evicted in past, with count \( \leq \min_t \)
  - Every count is an overestimate, using above observation
  - So est. count of \( x > \varepsilon n \geq \min \geq \min_t \), and would not be evicted

So: Find all items with count \( > \varepsilon n \), error in counts \( \leq \varepsilon n \)
Experimental Comparison

♦ Implementations of all these algorithms (and more!) at http://www.research.att.com/~marioh/frequent-items

♦ Experimental comparison highlights some differences not apparent from analytic study
  – All counter algorithms seem to have similar worst-case performance ($O(1/\varepsilon)$ space to give $\varepsilon N$ guarantee)
  – Algorithms are often more accurate than analysis would imply

♦ Compared on a variety of web, network and synthetic data
Counter Algorithms Experiments

Two implementations of SpaceSaving (SSL, SSH) achieve perfect accuracy in small space (10KB – 1MB)

Very fast: 20M – 30M updates per second
Counter Algorithms Summary

♦ Counter algorithms very efficient for arrivals-only case
  – Use $O(1/\varepsilon)$ space, guarantee $\varepsilon N$ accuracy
  – Very fast in practice (many millions of updates per second)

♦ Similar algorithms, but a surprisingly clear “winner”
  – Over many data sets, parameter settings, SpaceSaving algorithm gives appreciably better results

♦ Many implementation details even for simple algorithms
  – “Find if next item is monitored”: search tree, hash table...?
  – “Find item with smallest count”: heap, linked lists...?

♦ Not much room left for improvement in core problem?
Outline

♦ Problem definition and background
♦ “Counter-based” algorithms and analysis
♦ “Sketch-based” algorithms and analysis
♦ Further Results
♦ Conclusions
Sketch Algorithms

- Counter algorithms are for the “arrivals only” model, do not handle “arrivals and departures”
  - Deterministic solutions not known for the most general case
- Sketch algorithms compute a summary that is a linear transform of the frequency vector
  - Departures are naturally handled by such algorithms
- Sketches solve core problem of estimating item frequencies
  - Can then use to find frequent items via search algorithm
Count-Min Sketch

- **Count-Min Sketch** proposed in [C, Muthukrishnan ’04]
- Model input stream as a vector $x$ of dimension $U$
  - $x[i]$ is frequency of item $i$
- Creates a small summary as an array of $w \times d$ in size
- Use $d$ hash function to map vector entries to $[1..w]$

![Array CM[i,j]](image-url)
Count-Min Sketch Structure

- Each entry in vector \( x \) is mapped to one bucket per row.
- Estimate \( x[j] \) by taking \( \min_k CM[k, h_k(j)] \)
  - Guarantees error less than \( \varepsilon \|x\|_1 \) in size \( O(1/\varepsilon \log 1/\delta) \)
  - Probability of more error is less than \( 1-\delta \)
Count-Min Sketch Analysis

Approximate \( x'[j] = \min_k CM[k,h_k(j)] \)

- **Analysis:** In \( k \)'th row, \( CM[k,h_k(j)] = x[j] + X_{k,j} \)
  - \( X_{k,j} = \sum x[i] \mid h_k(i) = h_k(j) \)
  - \( \mathbb{E}(X_{k,j}) = \sum x[k] \cdot \Pr[h_k(i) = h_k(j)] \)
    \( \leq \Pr[h_k(i) = h_k(k)] \cdot \sum a[i] \)
    \( = \varepsilon \|x\|_1/2 \) by pairwise independence of \( h \)
  - \( \Pr[X_{k,j} \geq \varepsilon \|x\|_1] = \Pr[X_{k,j} \geq 2\mathbb{E}(X_{k,j})] \leq 1/2 \) by Markov inequality

- So, \( \Pr[x'[j] \geq x[j] + \varepsilon \|x\|_1] = \Pr[\forall k. X_{k,j} > \varepsilon \|x\|_1] \leq 1/2^{\log 1/\delta} = \delta \)

- **Final result:** with certainty \( x[j] \leq x'[j] \) and
  with probability at least \( 1-\delta \), \( x'[j] < x[j] + \varepsilon \|x\|_1 \)
  - Estimate is biased, can correct easily
Count Sketch

- **Count Sketch** proposed in [Charikar, Chen, Farach-Colton ’02]
- Uses extra hash functions $g_1 \ldots g_{\log \frac{1}{\delta}} \{1 \ldots U\} \rightarrow \{+1,-1\}$
- Now, given update $(j,+c)$, set $CM[k,h_k(j)] += c^* g_k(j)$

![Diagram](image-url)
Count Sketch Analysis

- Estimate $x'_{k[j]} = CM[k, h_k(j)] * g_k(j)$
- Analysis shows estimate is correct in expectation
- Bound error by analyzing the variance of the estimator
  - Apply Chebyshev inequality on the variance
- With probability $1-\delta$, error is at most $\varepsilon \|x\|_2 < \varepsilon N$
  - $\|x\|_2$ could be much smaller than $N$, at cost of $1/\varepsilon^2$
Hierarchical Search

- Sketches estimate the frequency of a single item
  - How to find frequent items without trying all items?
- Divide-and-conquer approach limits search cost
  - Impose a binary tree over the domain
  - Keep a sketch of each level of the tree
  - Descend if a node is heavy, else stop
- Correctness: all ancestors of a frequent item are also frequent
- Alternate approach based on “group testing”
  - Use sketches to determine identities of frequent items by running multiple tests.
Sketch Algorithms Experiments

- Less clear which sketch is best: depends on data, parameters
- Speed less by factor of 10, size more by factor 10:
  - A necessary trade off for flexibility to handle departures?
Outline

♦ Problem definition and background
♦ “Counter-based” algorithms and analysis
♦ “Sketch-based” algorithms and analysis
♦ Further Results
♦ Conclusions
Tighter Bounds

♦ **Observation:** algorithms outperform worst case guarantees

♦ **Analysis:** can prove stronger guarantees than $\varepsilon N$
  
  – Define $n_1 = $ highest frequency, $n_2 = $ second highest, etc.
  
  – Then define $F_{1 \text{res}(k)} = N -(n_1 + n_2 + \ldots n_k)$, $\ll N$ for skewed dbns
  
  – Result [Berinde, C, Indyk, Strauss, ’09]:
    
    Frequent, SpaceSaving (and others) guarantee $\varepsilon F_{1 \text{res}(k)}$ error

♦ **Similar bounds for sketch algorithms**
  
  – CountMin sketch also has $F_{1 \text{res}(k)}$ bound
  
  – Count sketch has $(F_{2 \text{res}(k)})^{1/2} = (\sum_{i=k+1}^{m} n_i^2)^{1/2}$ bound
  
  – Related to results in Compressed Sensing for signal recovery
Weighted Updates

- **Weighted case**: find items whose total weight is high
  - Sketch algorithms adapt easily, counter algs with effort
- **Simple solution**: all weights are integer multiples of small $\delta$
- **Full solution**: define appropriate generalizations of counter algs to handle real valued weights [Berinde et al ’09]
  - Straightforward to extend *SpaceSaving* analysis to weighted case
  - Frequent more complex, action depends on smallest counter value
  - No positive results known for *LossyCounting*
Mergability of Summaries

- Want to merge summaries, to summarize the union of streams
- Sketches with shared hash fns are easy to merge together
  - Via linearity, sum of sketches = sketch of sums
- Counter-based algorithms need new analysis [Berinde et al’09]
  - Merging two summaries preserves accuracy, but space may grow
  - With pruning of the summary, can merge indefinitely
  - Space remains bounded, accuracy degrades by at most a constant
Other Extensions

♦ Heavy Changers
  – Which items have largest (absolute, relative) change over two streams?

♦ Assumptions on frequency distribution, order
  – Give tighter space/accuracy tradeoff for skewed distributions
  – Worst case arrival order vs. random arrival order

♦ Distinct Heavy Hitters
  – E.g. which sources contact the most distinct addresses?

♦ Time Decay
  – “Weight” of items decay (exponentially, polynomially) with age
Conclusions

♦ Finding the frequent items is one of the most studied problems in data streams
  – Continues to intrigue researchers
  – Many variations proposed
  – Algorithms have been deployed in Google, AT&T, elsewhere...
♦ Still some room for innovation, improvements

♦ Survey and experiments in VLDB [C, Hadjieleftheriou ’08]
  – Code, synthetic data and test scripts at http://www.research.att.com/~marioh/frequent-items
  – Shorter, broader write up in CACM 2009