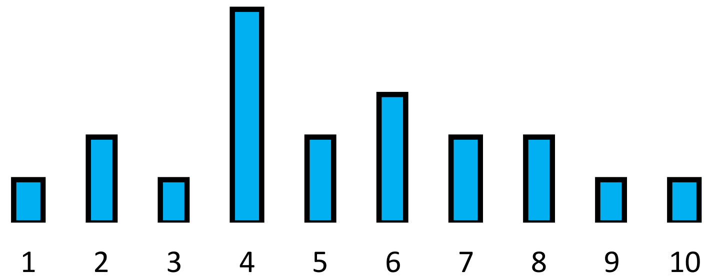


Learning-Based* Frequency Estimation in Data Streams

Chen-Yu Hsu Piotr Indyk Dina Katabi Ali Vakilian

(+Anders Aamand)

MIT

*A.k.a. Automated / Data-Driven

Data Streams

- A data stream is a (massive) sequence of data
  – Too large to store (on disk, memory, cache, etc.)
- Single pass over the data: $i_1, i_2, \ldots, i_n$
- **Bounded storage** (typically n^α or $\log^c n$)

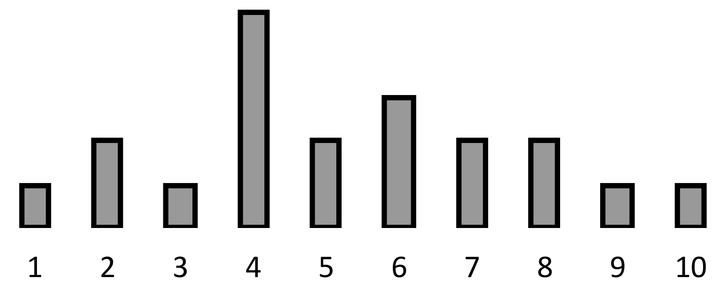
42

8 2 1 9 1 9 2 4 6 3 9 4 2 3 4 2 3 8 5 2 5 6 5 8 6 3 2 9 1

- Many developments, esp. since the 90s
  – Clustering, quantiles, distinct elements, frequency moments, frequency estimation,..

Frequency Estimation Problem

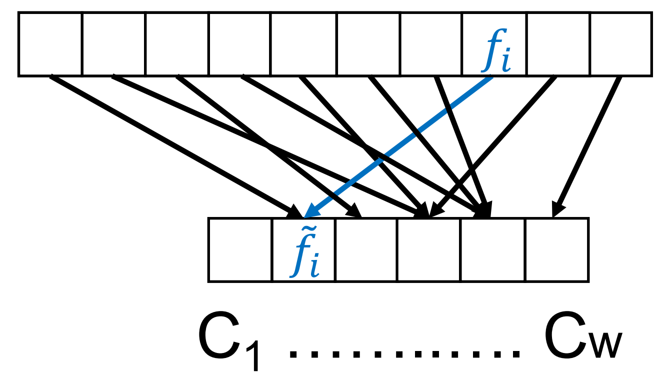
- Data stream S: a sequence of items from U
  - E.g., S=8, 1, 7, 4, 6, 4, 10, 4, 4, 6, 8, 7, 5, 4, 2, 5, 6, 3, 9, 2
- Goal: at the end of the stream, given item $i \in$ U**,** output an estimation $\tilde{f}_i$ of the frequency f_i in S
- Applications in
  - Network Measurements
  - Comp bio (e.g., counting kmers, as in Paul Medvedev's talk on Wed)
  - Machine Learning
  - …
- Easy to do using linear space
- Sub-linear space ?



Count-Min

[Cormode-Muthukrishan'04]; cf. [Estan-Varghese'02]

- Basic algorithm:
  – Prepare a random hash function h: U→{1..w}
  – Maintain an array $C=[C_1,\ldots C_w]$ such that
  $$C_j=\sum_{i:\ h(i)=j} f_i$$
  (if you see element i, increment $C_{h(i)}$)
  – To estimate f_i return
  $$\tilde{f}_i = C_{h(i)}$$



- "Counting" Bloom filters [Fan et al'00]
  – CM never underestimates (assuming f_i non-negative)
- Count-Sketch [Charikar et al'02]
  – Arrows have signs, so errors cancel out

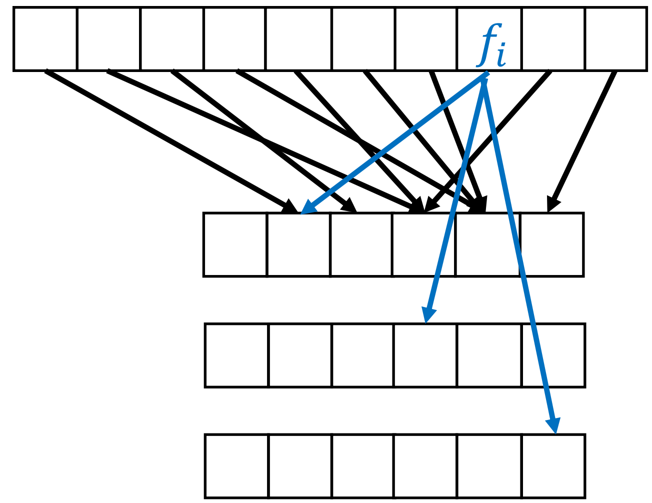
Count-Min ctd.

- Error guarantees (per each f_i):
  - $E[|\tilde{f}_i - f_i|]$
    $= \sum_{l \neq i} \Pr[h(l)=h(i)]\, f_l \leq 1/w\, ||f||_1$
- Actual algorithm:
  - Maintain d vectors $C^1 \ldots C^d$ and functions $h_1 \ldots h_d$
  - Estimator:

$$\tilde{f}_i = \min_t C^t_{h_t(i)}$$

- Analysis:

$$\Pr[\, |\tilde{f}_i - f_i| \geq 2/w\, ||f||_1 \,] \leq 1/2^d$$



(How) can we improve this by learning?

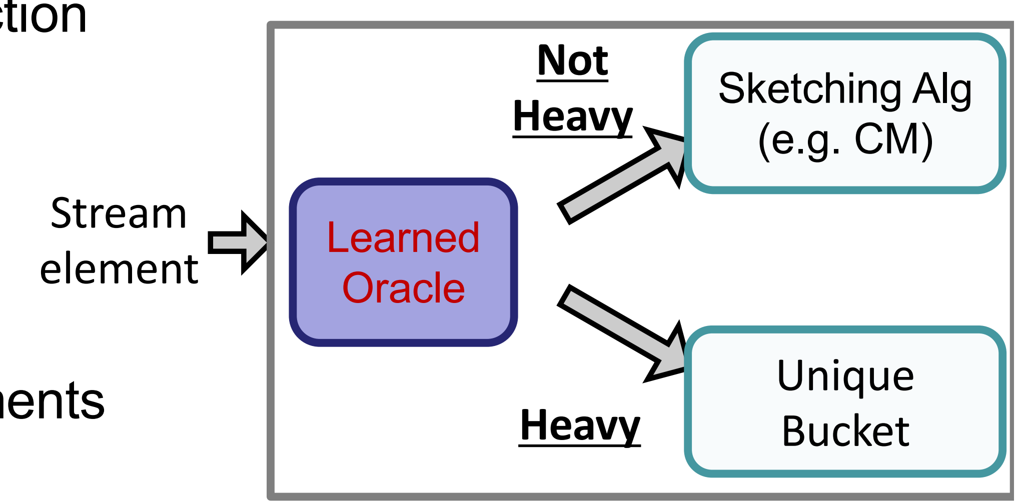
- What is the "structure" in the data that we could adapt to ?
- There is lots of information in the **id** of the stream elements:
  - For word data, it is known that frequency tends to be inversely proportional to the word length rank
  - For network data, some IP addresses (or IP domains) are more popular than others
  - …
- If we could learn these patterns, then (hopefully) we could use them to improve algorithms
  - E.g., try to avoid collisions with/between heavy items

Learning-Based Frequency Estimation

[Hsu-Indyk-Katabi-Vakilian, ICLR'19]

- Inspired by Learned Bloom filters (Kraska et al., 2018)
- Consider "aggregate" error function

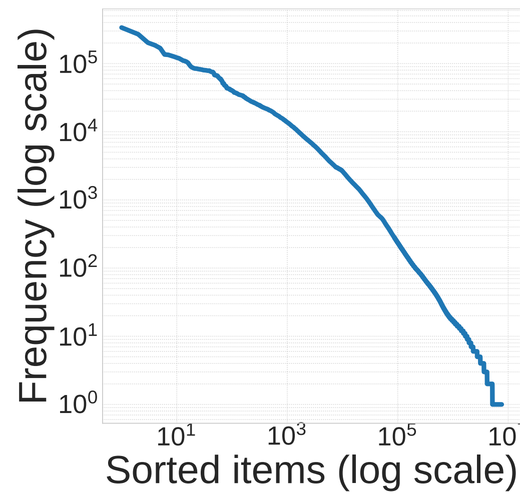
$$\sum_{i \in U} f_i \cdot |\tilde{f}_i - f_i|$$

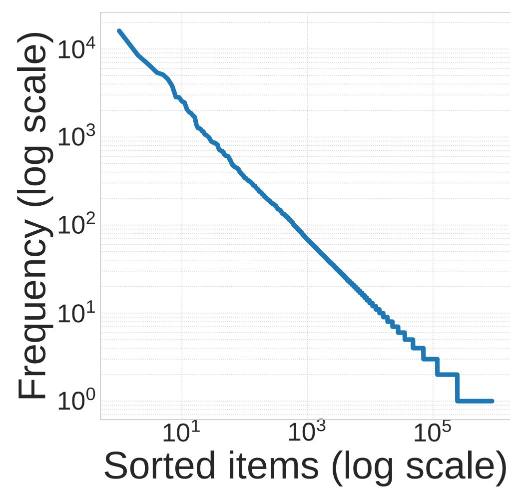


- Use past data to train an ML classifier to detect "heavy" elements
  - "Algorithm configuration"
- Treat heavy elements differently
- Cost model: unique bucket costs 2 memory words
- Algorithm inherits worst case guarantees from the sketching algorithm

Experiments

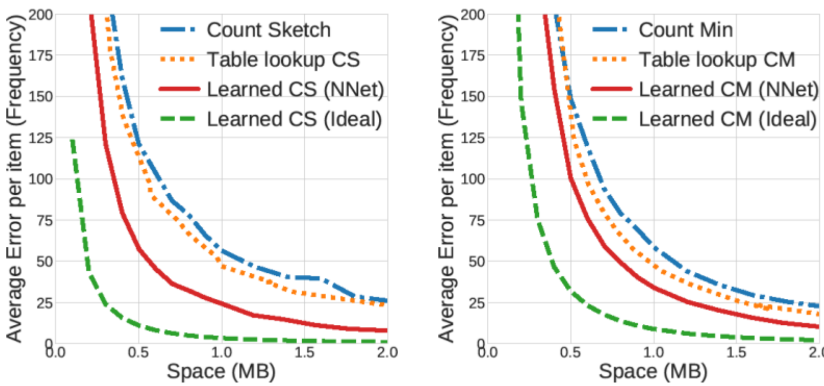
- Data sets:
  - Network traffic from CAIDA data set
    - A backbone link of a Tier1 ISP between Chicago and Seattle in 2016
    - One hour of traffic; 30 million packets per minute
    - Used the first 7 minutes for training
    - Remaining minutes for validation/testing
  - AOL query log dataset:
    - 21 million search queries collected from 650 thousand users over 90 days
    - Used first 5 days for training
    - Remaining minutes for validation/testing
- Oracle: Recurrent Neural Network
  - CAIDA: 64 units
  - AOL: 256 units



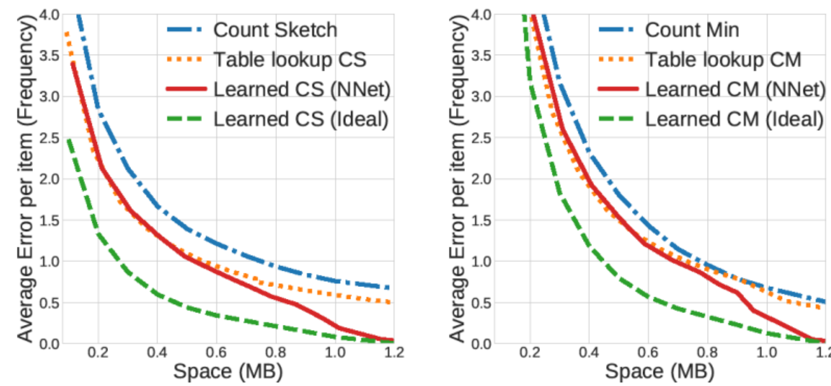


Results

Internet Traffic Estimation (20th minute)



Search Query Estimation (50th day)



- Table lookup: oracle stores heavy hitters from the training set
- Learning augmented (Nnet): our algorithm
- Ideal: error with a perfect oracle
- Space amortized over multiple minutes (CAIDA) or days (AOL)

Theoretical Results

- Assume Zipfian Distribution ($f_i \propto 1/i$)
- Count-Min algorithm

Method	Expected Err
CountMin (*k>1* rows)	$\Theta(\frac{k}{B} \ln n \ln(\frac{kn}{B}))$
Learned CountMin (perfect oracle)	$\Theta(\frac{\ln^2(n/B)}{B})$

A. Aamand

✓Learned CM improves upon CM when B is close to n

✓Learned CM is asymptotically optimal

U: universe of the items
n: number of items with non-zero frequency
k: number of hash tables
w=B/k: number of buckets per hash table

Why ML Oracle Helps ?

- Simple setting: Count-Min with one hash function (i.e., k=1)
  - Standard Count-Min expected error:

$$E[\sum_{i \in U} f_i \cdot |\tilde{f}_i - f_i|] \approx \sum_{i \in U} \frac{1}{i} \cdot \left(\frac{1}{B} \sum_{i \in U} \frac{1}{i} \right) \approx ln^2(n) \,/B$$

  - Learned Count-Min with perfect oracle:
    - Identify heaviest B/2 elements and store separately

$$\sum_{i \in U - [B/2]} \frac{1}{i} \cdot \left(\frac{1}{B/2} \sum_{i \in U - [B/2]} \frac{1}{i} \right) \approx ln^2(n/B) \,/B$$

Optimality of Learned Count-Min

Theorem: If n/B >$e^{4.2}$, then the estimation error of **any** hash function that maps a set of n items following Zipfian distribution to B buckets is $\Omega(\frac{\ln^2(n/B)}{B})$

Observation: For min-of-counts estimator, single hash function is optimal.

Conclusions

- ML helps improve the performance of streaming algorithms
- Some theoretical understanding/bounds, although:
  - ~~Bounds for Count-Min (k>1) not tight~~
  - ~~Count-sketch ?~~
- Other sketching/streaming problems?
  - Learned Locality-Sensitive Hashing

    (with Y. Dong, I. Razenshteyn, T. Wagner)
  - Learned matrix sketching for low-rank approximation

    (with Y. Yuan, A. Vakilian)
  - …

Conclusions ctd

- A pretty general approach to algorithm design
  - Along the lines of divide-and-conquer, dynamic programming etc
- There are pros and cons
  - Pros: better performance
  - Cons: (re-)training time, update time, different guarantees
- Teaching a class on this topic (with C. Daskalakis)
  https://stellar.mit.edu/S/course/6/sp19/6.890/materials.html
- Insights into “classical” algorithms