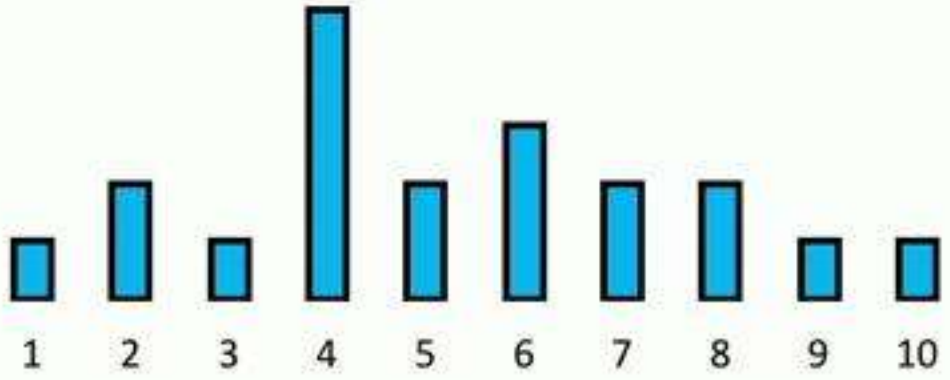
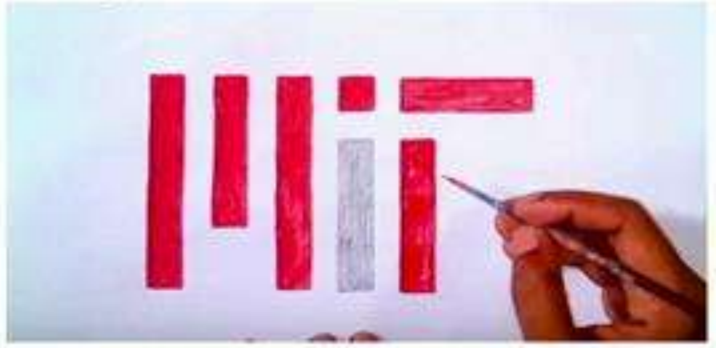


Learning-Based* Sketching Algorithms

Anders Aamand Chen-Yu Hsu Piotr Indyk Dina Katabi
Yang Yuan Ali Vakilian



*A.k.a. Automated / Data-Driven (see Nina Balcan's talk)

Learning-Based Algorithms

- **"Classical" Algorithms (think CLRS)**
  - \+ Worst-case guarantees
  - \- Limited adaptivity to inputs

Learning-Based Algorithms

- **“Classical” Algorithms (think CLRS)**
  - \+ Worst-case guarantees
  - \- Limited adaptivity to inputs

- **Machine Learning Based Approaches**
  - \+ Stronger performance by adapting to inputs
  - \- No worst-case guarantees

Learning-Based Algorithms

- **"Classical" Algorithms (think CLRS)**
  - \+ Worst-case guarantees
  - \- Limited adaptivity to inputs

- **Machine Learning Based Approaches**
  - \+ Stronger performance by adapting to inputs
  - \- No worst-case guarantees
- **"Best of both worlds"**
  - \+ Adaptive
  - \+ (Often) worst case guarantees/analysis

Learning-Based Algorithms

- **“Classical” Algorithms (think CLRS)**
  - \+ Worst-case guarantees
  - \- Limited adaptivity to inputs

- **Machine Learning Based Approaches**
  - \+ Stronger performance by adapting to inputs
  - \- No worst-case guarantees
- **“Best of both worlds”**
  - \+ Adaptive
  - \+ (Often) worst case guarantees/analysis

Learning-Based Algorithms – Rough Overview

- Algorithm configuration - optimizing the knobs/selection

  Leyton-Brown et al 2002-.., Hutter et al, 2011, Gupta & Roughgarden, 2015, Balcan et al, 2017

- "ML oracles"- provide useful guesses about given inputs

  On-line algorithms (Lykouris &Vassilvitskii, 2018; Purohit et al, 2018)

  Data structures e.g., Bloom filters (Kraska et al., 2018; Mitzenmacher, 2018)

  Branch and bound (Balcan et al., 2018, Khalil et al'2017)

- "Learned structures" – structures tailored to the input distribution

  Learning to Hash (Salakhutdinov & Hinton, 2009; Weiss et al., 2009; Jegou et al., 2011;…)

  Compressed sensing (Mousavi et al., 2015; Bora et al., 2017)

- "End-to-end" – algorithms implemented as neural networks

  Scheduling (Mao et al., 2018)

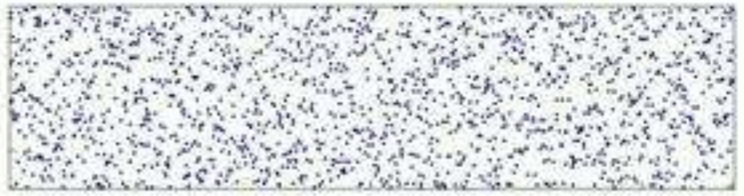
  TSP algorithms (Dai et al., 2017)

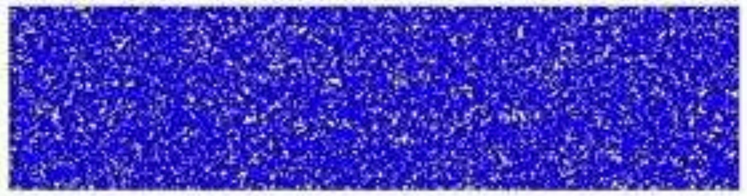
Linear Sketches

- Many algorithms are obtained using **linear sketches**:
  - Input: represented by x (or A)
  - Sketching: compress x into Sx
    - S=sketch matrix
  - Computation: on Sx
- Examples:
  - Dimensionality reduction (e.g., Johnson-Lindenstrauss lemma)
  - Streaming algorithms
  - Compressed sensing
  - Linear algebra (regression, low-rank approximation,..)

Learned Linear Sketches

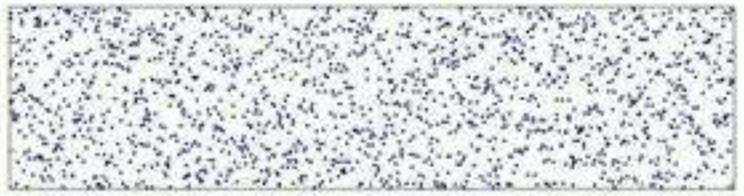
- S is almost always a random matrix
  - Independent, FJLT, sparse,...

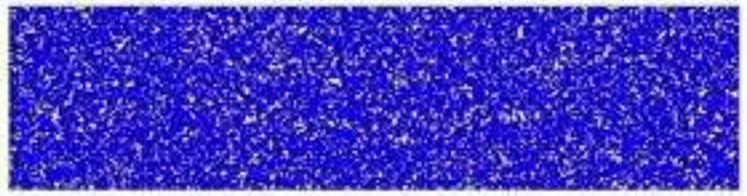




Learned Linear Sketches

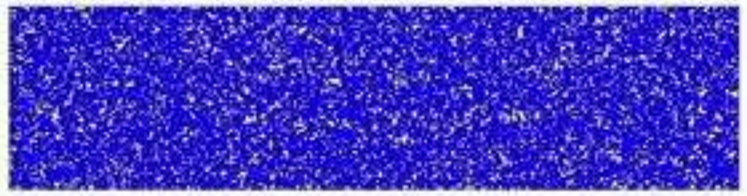
- S is almost always a random matrix
  - Independent, FJLT, sparse,...
  - Pros: simple, efficient, worst-case guarantees
  - Cons: does not adapt to data





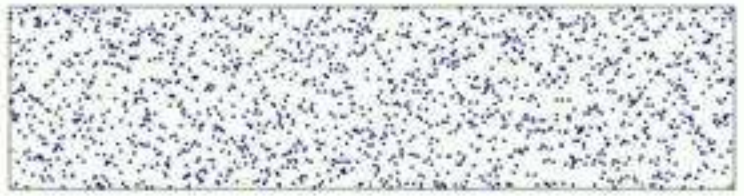
Learned Linear Sketches

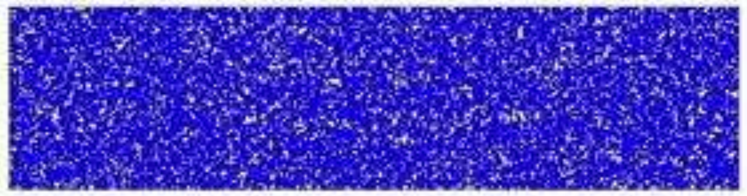
- S is almost always a random matrix
  - Independent, FJLT, sparse,...
  - Pros: simple, efficient, worst-case guarantees
  - Cons: does not adapt to data
- Why not **learn** S from examples ?
  - Dimensionality reduction: e.g., PCA, Isomap (Anna's talk yesterday)
  - Compressed sensing: Mousavi et al, 2015-…
    - Autoencoders: $x \rightarrow Sx \rightarrow x'$
  - Streaming algorithms ?
  - Linear algebra ?



Learned Linear Sketches

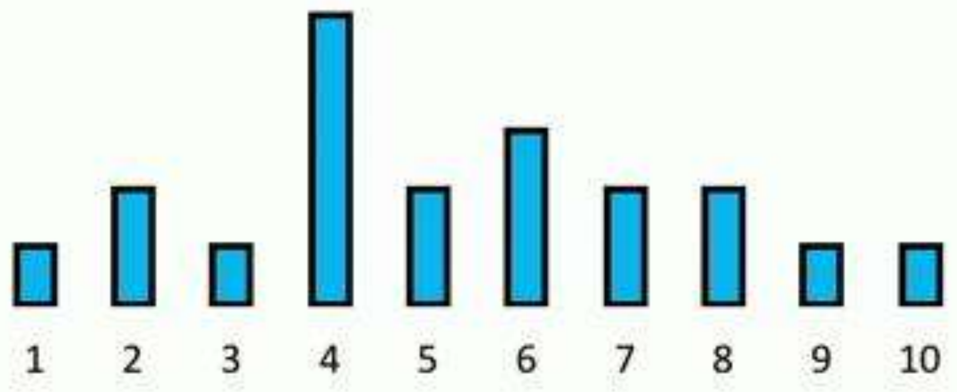
- S is almost always a random matrix
  - Independent, FJLT, sparse,...
  - Pros: simple, efficient, worst-case guarantees
  - Cons: does not adapt to data
- Why not **learn** S from examples ?
  - Dimensionality reduction: e.g., PCA, Isomap (Anna's talk yesterday)
  - Compressed sensing: Mousavi et al, 2015-…
    - Autoencoders: $x \rightarrow Sx \rightarrow x'$
  - Streaming algorithms ?
  - Linear algebra ?





THIS TALK

Learned Streaming Algorithms for Frequency Estimation

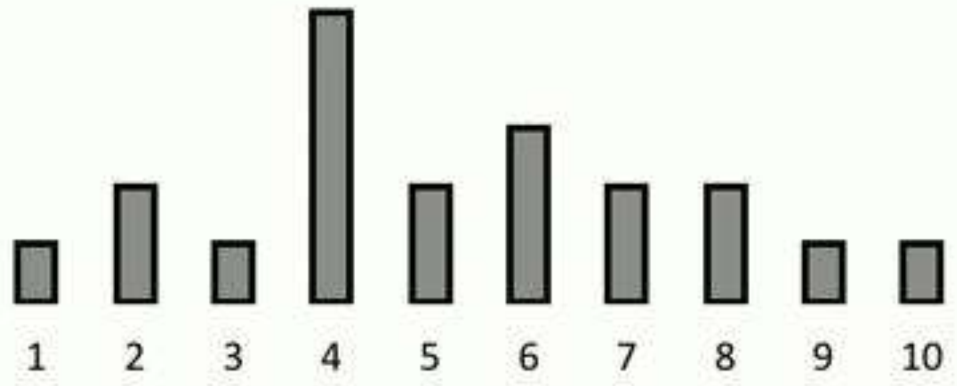


Frequency Estimation Problem

- Stream S: a sequence of items from U

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

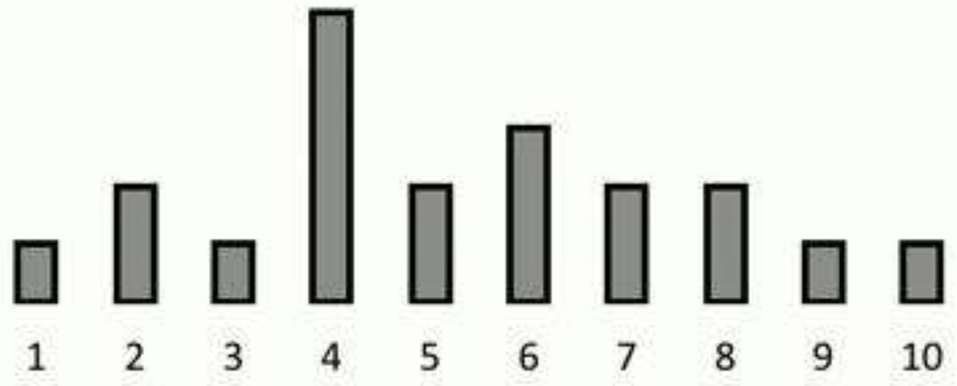


Frequency Estimation Problem

- Stream S: a sequence of items from U

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

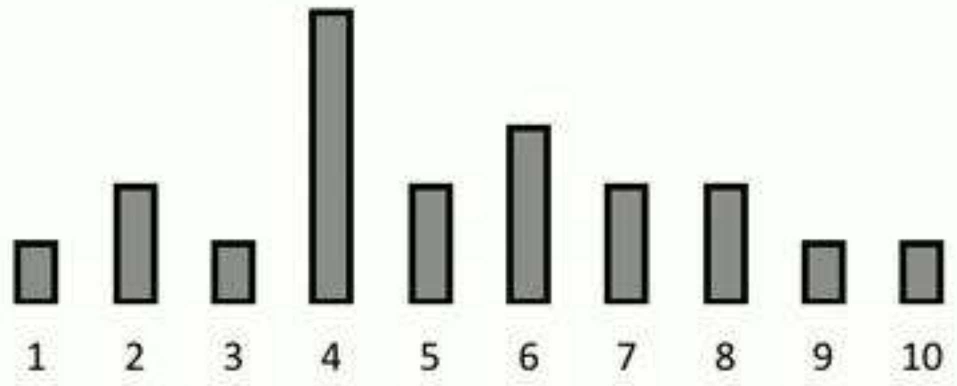


Frequency Estimation Problem

- Stream S: a sequence of items from U

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
- Many applications:
  - Network Measurements
  - Comp bio
  - Machine Learning
  - …

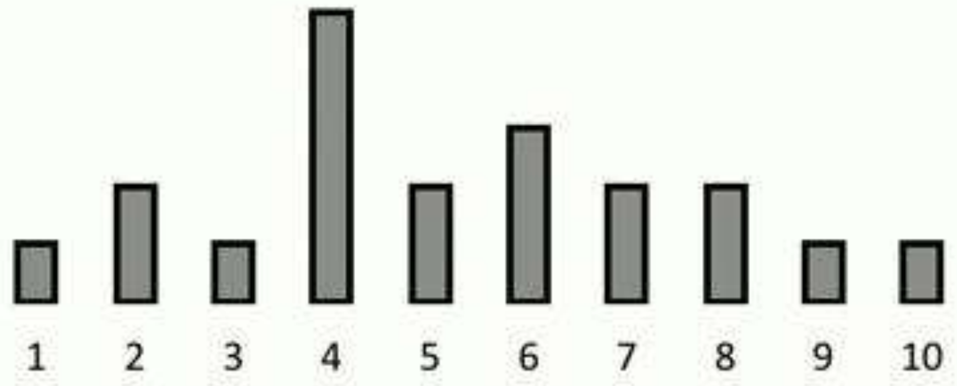


Frequency Estimation Problem

- Stream S: a sequence of items from U

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
- Many applications:
  - Network Measurements
  - Comp bio
  - Machine Learning
  - …
- Easy to do using linear space
- Sub-linear space ?



Count-Min

[Cormode-Muthukrishan'04]; cf. [Estan-Varghese'02,Fan et al'00]

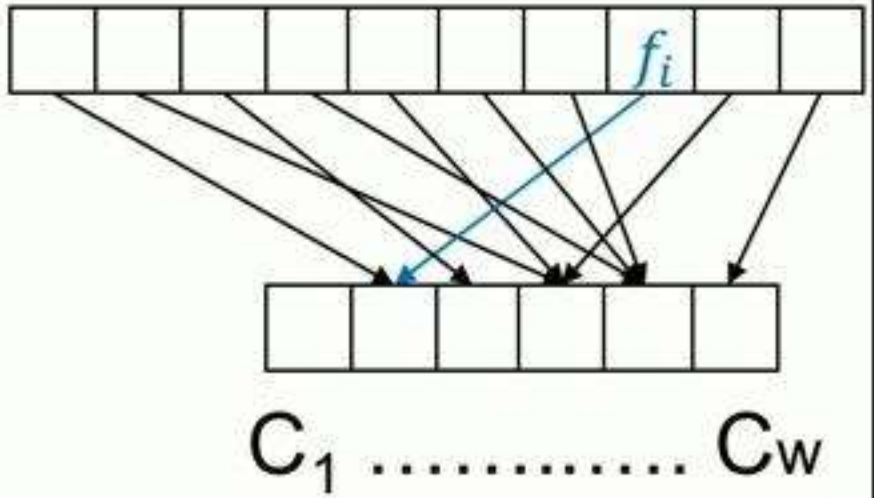
- Basic algorithm:
  - Prepare a random hash function h: U→{1..w}
  - Maintain an array $C=[C_1,\ldots C_w]$ such that

$$C_j=\sum_{i:\ h(i)=j} f_i$$

(if you see element i, increment $C_{h(i)}$)

  - To estimate f_i return

$$\tilde{f}_i = C_{h(i)}$$



Count-Min

[Cormode-Muthukrishan'04]; cf. [Estan-Varghese'02,Fan et al'00]

- Basic algorithm:
  - Prepare a random hash function h: $U \rightarrow \{1..w\}$
  - Maintain an array $C=[C_1,\ldots C_w]$ such that

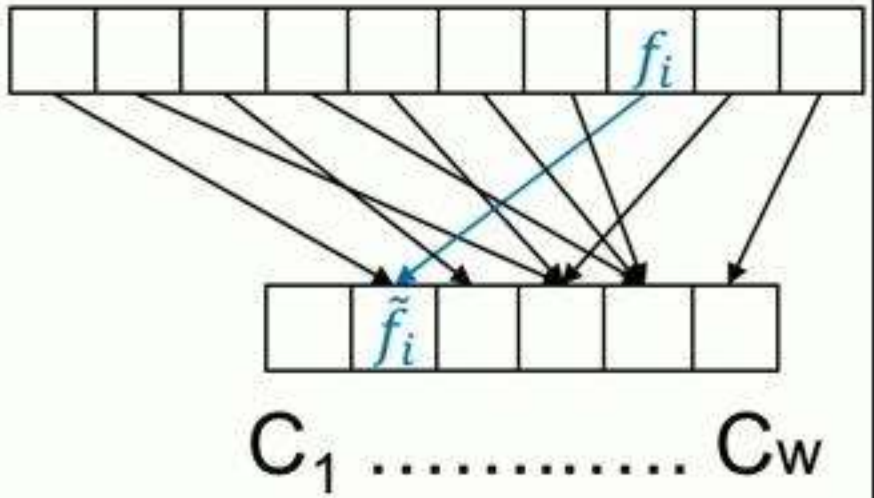
$$C_j = \sum_{i:\ h(i)=j} f_i$$

(if you see element i, increment $C_{h(i)}$)

  - To estimate f_i return

$$\tilde{f}_i = C_{h(i)}$$

- Works with insertions/deletions
- Never underestimates (assuming f_i non-negative)
- Count-Sketch [Charikar et al'02]
  - Arrows have signs, so errors cancel out



(How) can we improve this by learning?

Count-Min

[Cormode-Muthukrishan'04]; cf. [Estan-Varghese'02,Fan et al'00]

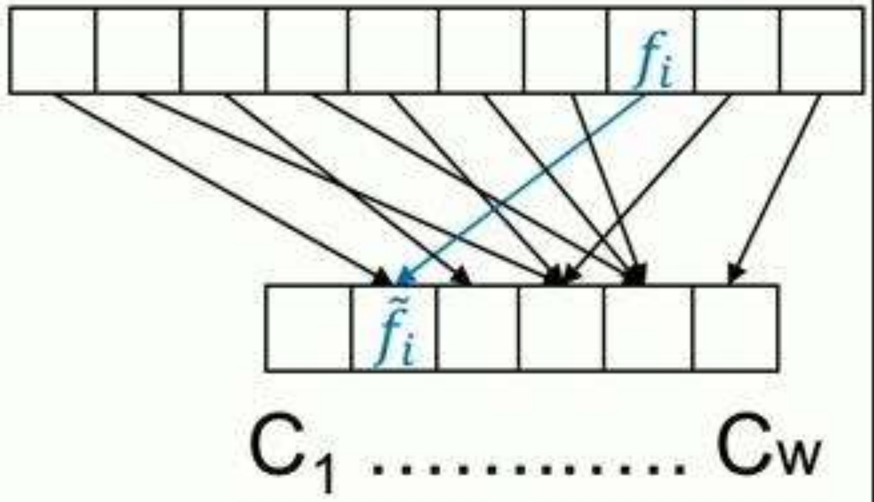
- Basic algorithm:
  - Prepare a random hash function h: $U \rightarrow \{1..w\}$
  - Maintain an array $C=[C_1,\ldots C_w]$ such that

$$C_j=\sum_{i:\ h(i)=j} f_i$$

(if you see element i, increment $C_{h(i)}$)

  - To estimate f_i return

$$\tilde{f}_i = C_{h(i)}$$



- Works with insertions/deletions
- Never underestimates (assuming f_i non-negative)
- Count-Sketch [Charikar et al'02]
  - Arrows have signs, so errors cancel out

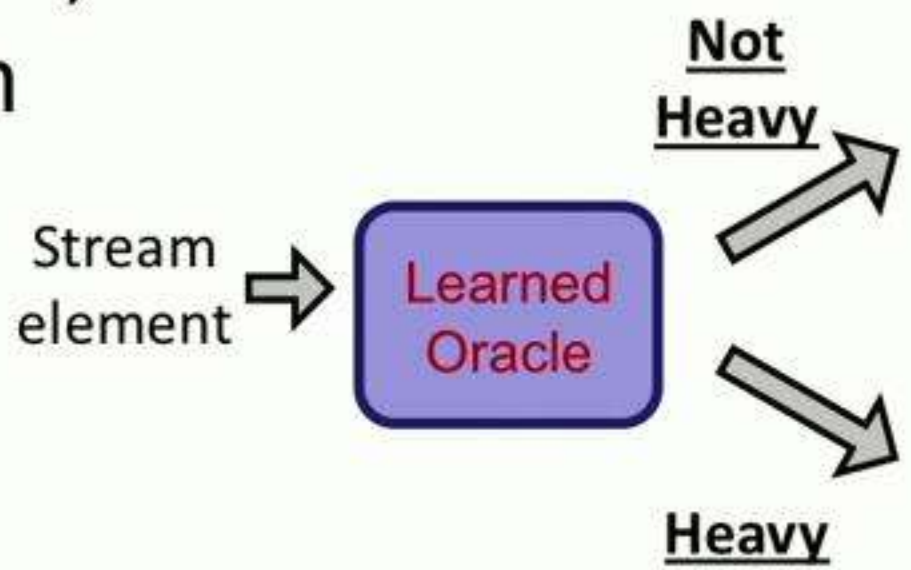
(How) can we improve this by learning?

- 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]

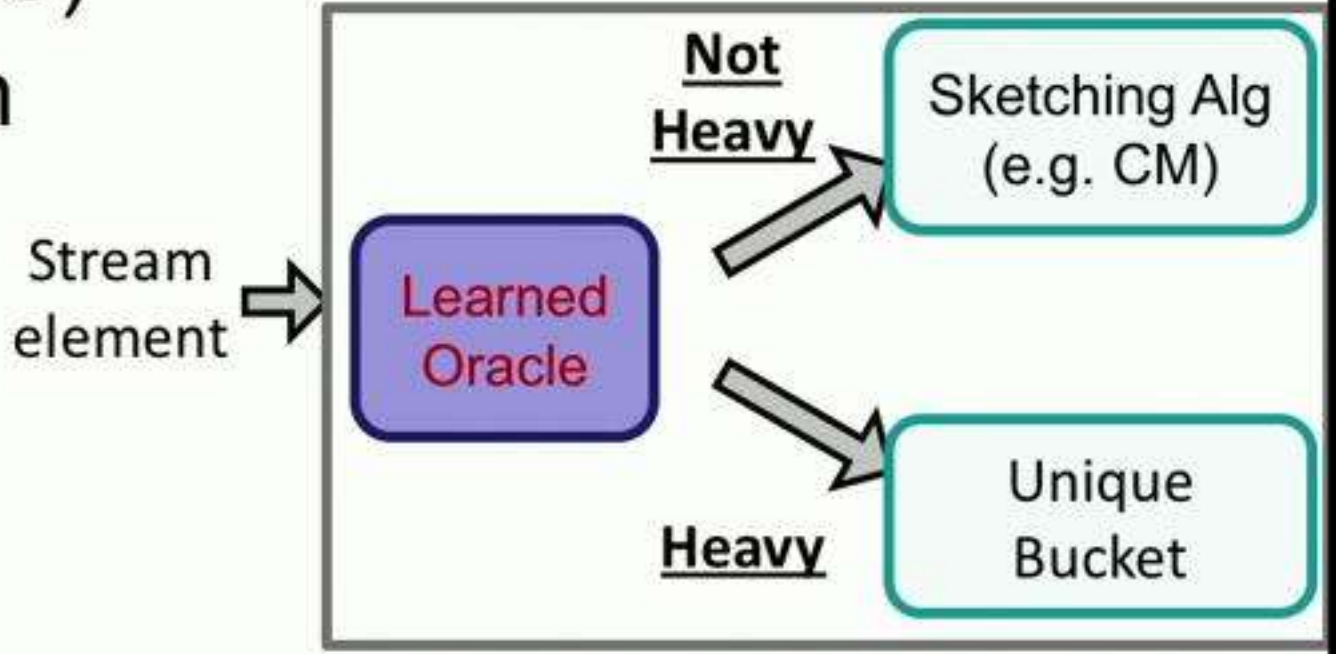
- Inspired by Learned Bloom filters (Kraska et al., 2018)
- Use past data to train an ML classifier to detect "heavy" elements
- Treat heavy elements differently



Learning-Based Frequency Estimation

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

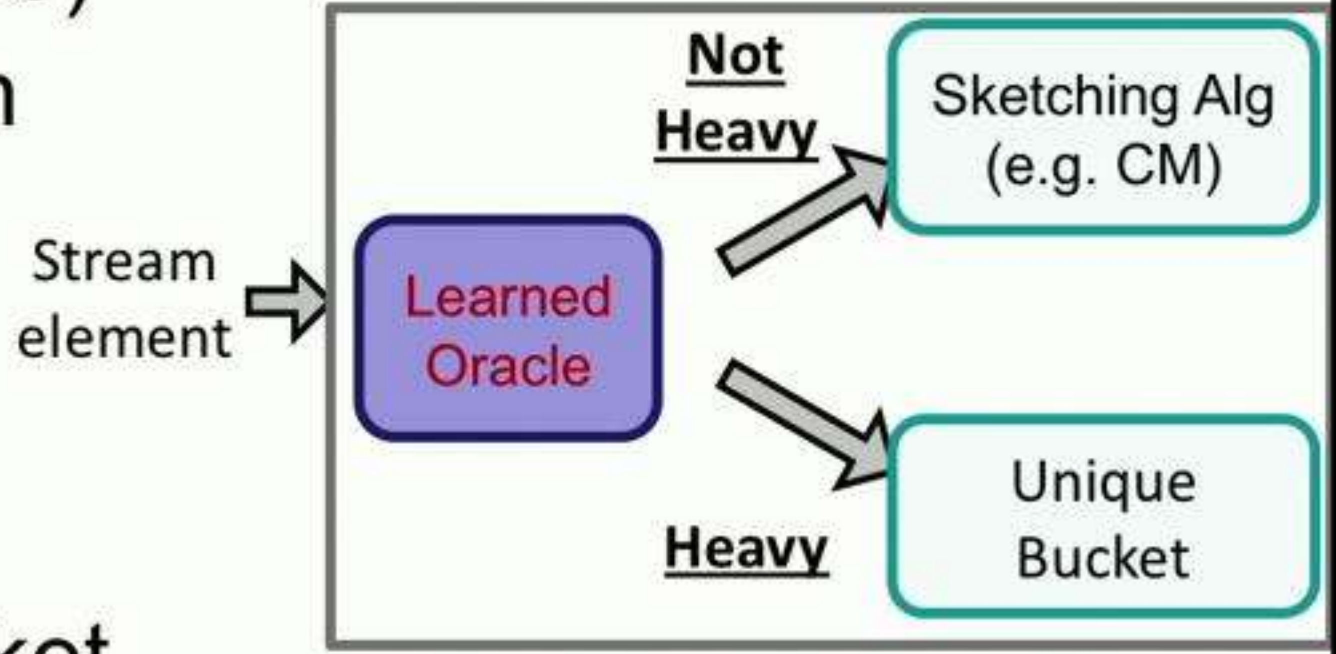
- Inspired by Learned Bloom filters (Kraska et al., 2018)
- Use past data to train an ML classifier to detect "heavy" elements
- Treat heavy elements differently



Learning-Based Frequency Estimation

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

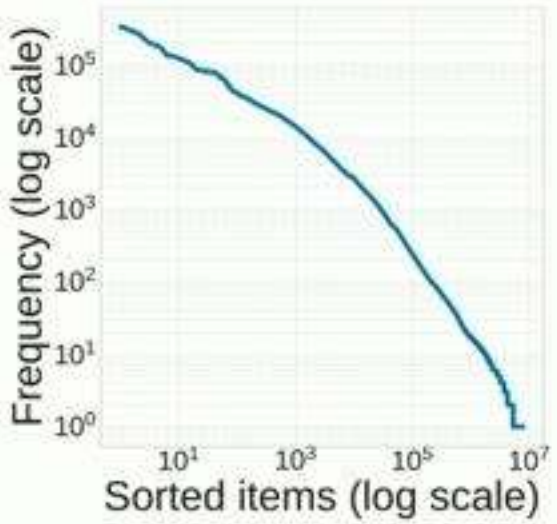
- Inspired by Learned Bloom filters (Kraska et al., 2018)
- Use past data to train an ML classifier to detect "heavy" elements
- Treat heavy elements differently
- Cost model: unique bucket costs 2 memory words
- Algorithm inherits worst case guarantees from the sketching algorithm

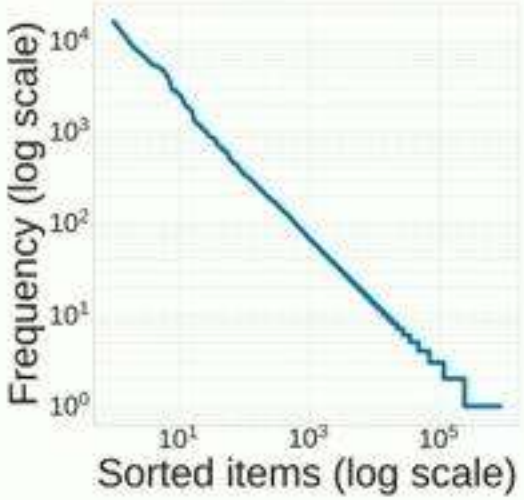


Experiments

- 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 days for validation/testing
- Oracle: Recurrent Neural Network
  - CAIDA: 64 units
  - AOL: 256 units
- Error function

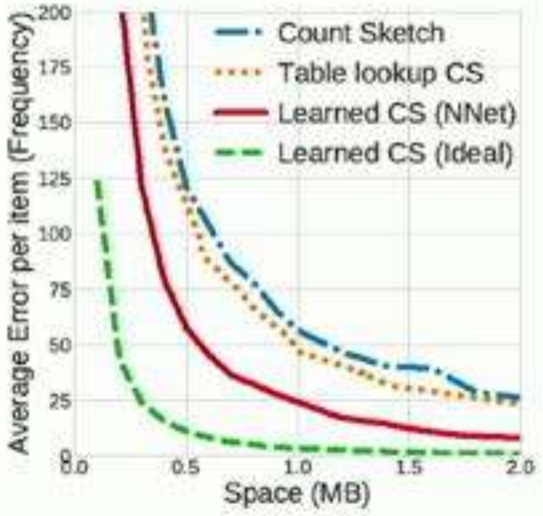
$$\sum_{i \in U} f_i \cdot |\tilde{f}_i - f_i|$$



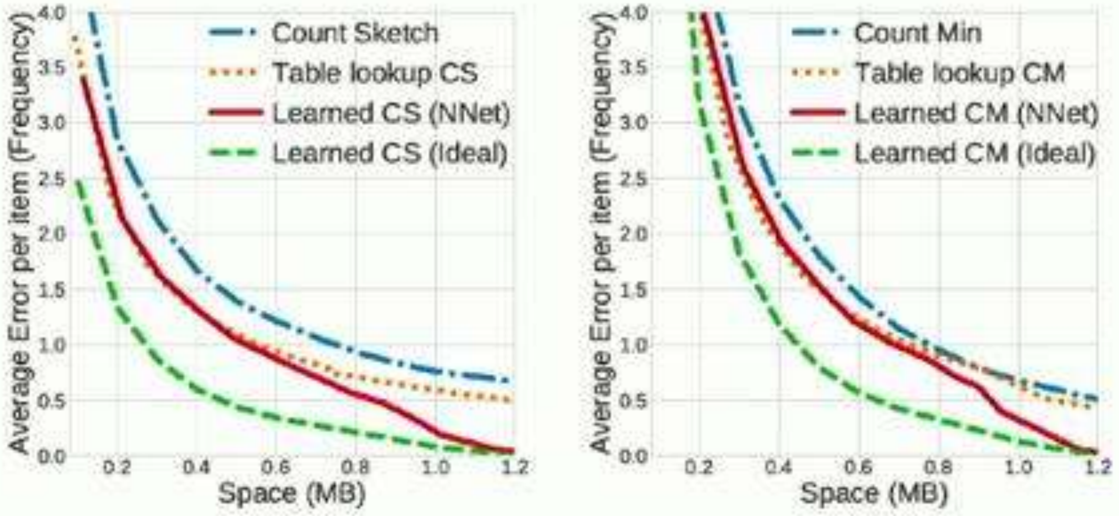


Results

Internet Traffic Estimation (20th minute)



Search Query Estimation (50th day)

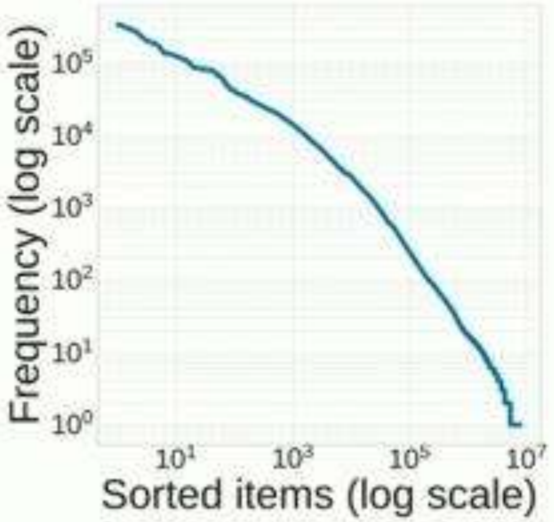


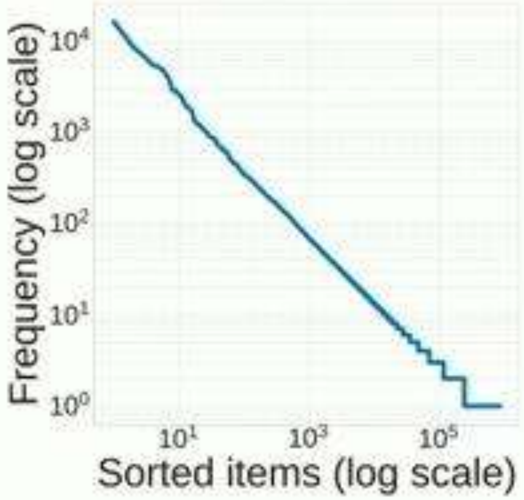
- Table lookup: oracle stores heavy hitters from the training set
- Learning augmented (Nnet): our algorithm
- Ideal: error with a perfect oracle
- Space amortized over multiple minutes (CAIDA) or days (AOL)

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 days for validation/testing
- Oracle: Recurrent Neural Network
  - CAIDA: 64 units
  - AOL: 256 units
- Error function

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





Theoretical Results

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

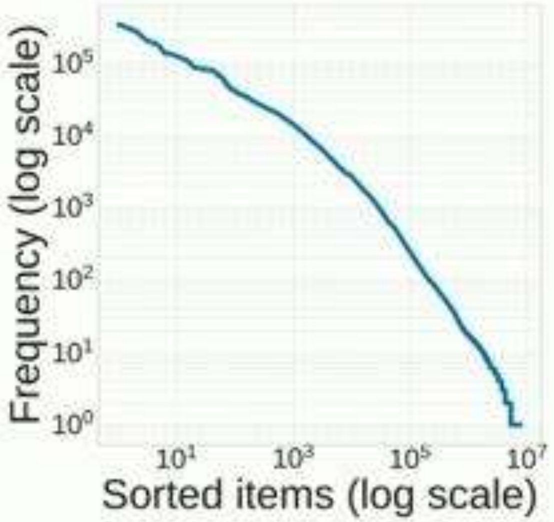
Method	Expected Err
CountMin (*k>1* rows)	$\Theta(\frac{k}{B}\ln n \ln(\frac{kn}{B}))$
Learned CountMin (perfect oracle)	$\Theta(\frac{\ln^2(n/B)}{B})$

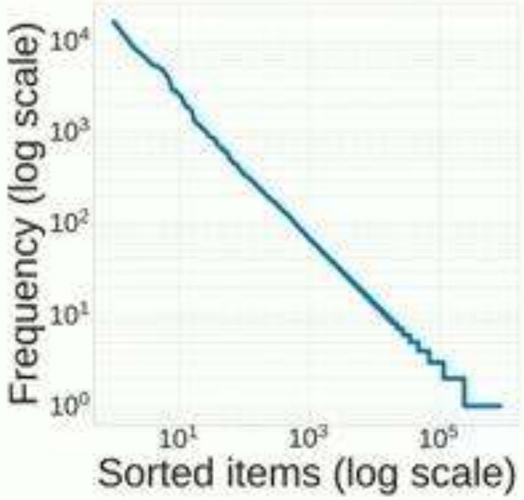
U: universe of the items
n: number of items with non-zero frequency
k: number of hash tables
B: total space (#buckets)
w=B/k: number of buckets per hash table

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 days for validation/testing
- Oracle: Recurrent Neural Network
  - CAIDA: 64 units
  - AOL: 256 units
- Error function

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





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

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

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

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

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

✓Learned CM is asymptotically optimal

Learned Sketching Algorithms for Low-Rank Approximation

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

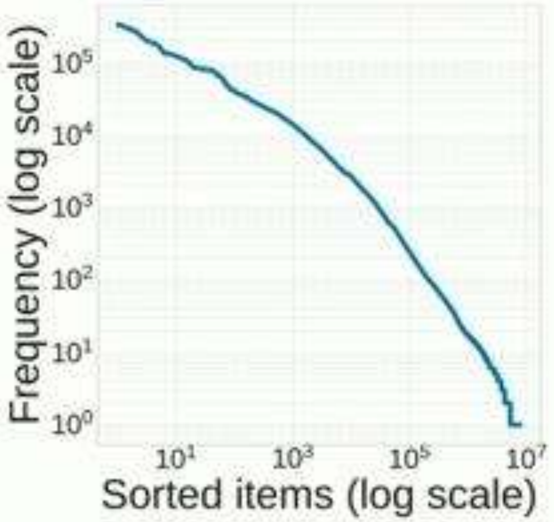
U: universe of the items
n: number of items with non-zero frequency
k: number of hash tables
B: total space (#buckets)
w=B/k: number of buckets per hash table

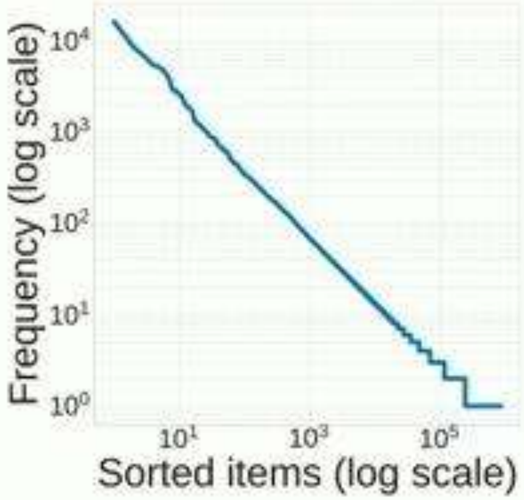
✓Learned CM is asymptotically optimal

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 days for validation/testing
- Oracle: Recurrent Neural Network
  - CAIDA: 64 units
  - AOL: 256 units
- Error function

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





Learned Sketching Algorithms for Low-Rank Approximation

Low Rank Approximation

<u>Singular Value Decomposition (SVD)</u>

Any matrix $A = U \ \Sigma \ V$, where:

- U has orthonormal columns
- Σ is diagonal
- V has orthonormal rows

Rank-k approximation: $A_k = U_k \ \Sigma_k \ V_k$

Equivalently: $A_k = \text{argmin}_{\text{rank-k matrices B}} ||A-B||_F$

$$\left(\quad \mathbf{A} \quad \right) = \left(\quad \mathbf{U}_k \quad \right) \left(\ \Sigma_k \ \right) \left(\quad \mathbf{V}_k \quad \right) + \left(\quad \mathbf{E} \quad \right)$$

Approximate Low Rank Approximation

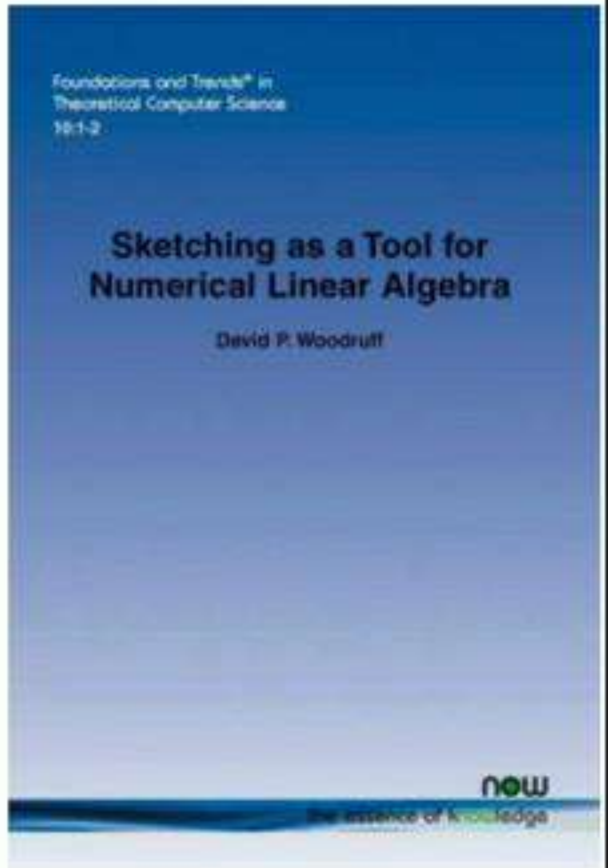
- Instead of

  $$A_k = \mathrm{argmin}_{\text{rank-k matrices } B} \|A-B\|_F$$

  output a rank-k matrix A', so that

  $$\|A-A'\|_F \le (1+\varepsilon) \|A-A_k\|_F$$

- Hopefully more efficient than computing exact A_k
  - Sarlos'06, Clarkson-Woodruff'09,13,Halko et al'11….
  - See Woodruff'14 for a survey
- Most of these algos use linear sketches SA
  - S can be dense (FJLT) or sparse (0/+1/-1)
  - We focus on sparse S



Sarlos-ClarksonWoodruff

- **Streaming** algorithm (two passes):
  - Compute SA (first pass)
  - Compute orthonormal V that spans rowspace of SA
  - Compute AV^T (second pass)
  - Return SCW(S,A):= $[AV^T]_k V$
- Space:
  - Suppose that A is n x d, S is m x n
  - Then SA is m x d, AV^T is n x m
  - Space proportional to m
  - Theory: $m = O(k/\varepsilon)$

Learning-Based Low-Rank Approximation

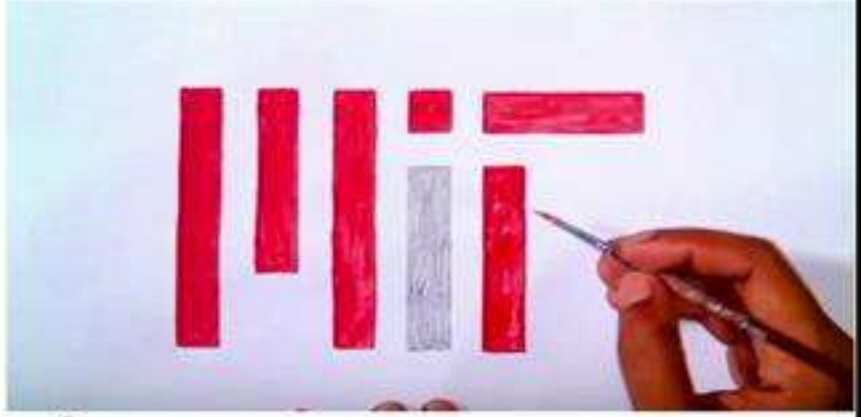
- Sample matrices $A_1 \ldots A_N$
- Find S that minimizes

$$\sum_i ||A_i - SCW(S, A_i)||_F$$

- Use S happily ever after …

  (as long data come from the same distribution)
- "Details":
  - Use sparse matrices S
    - Random support, optimize values
  - Need to differentiate the loss w.r.t. S
  - Represent SVD as a sequence of power-method applications (each is differentiable)

Evaluation



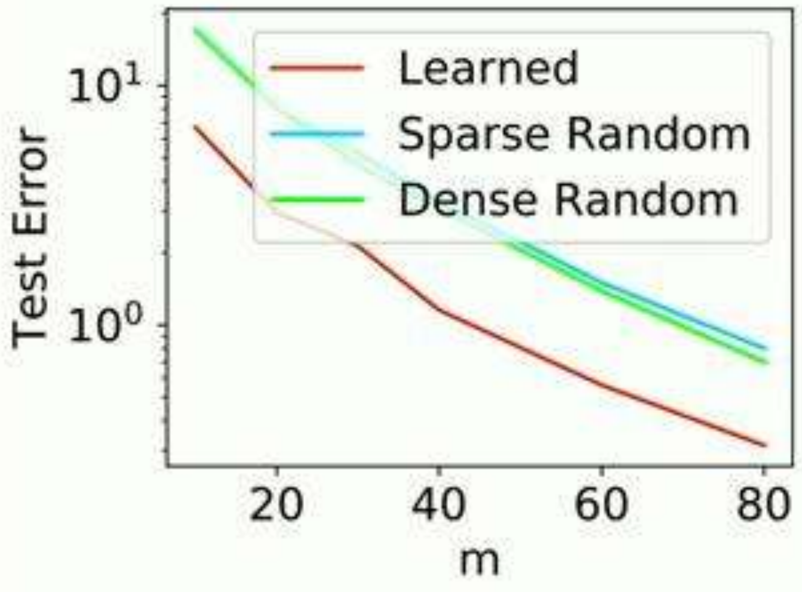
- Datasets:
  - Videos: MIT Logo, Friends, Eagle
  - Hyperspectral images (HS-SOD)
  - TechTC-300
- 200/400 training, 100 testing
- Optimize the matrix S
- Compute the empirical recovery error

$$\sum_i ||A_i - SCW(S, A_i)||_F - ||A_i - [A_i]_k||_F$$

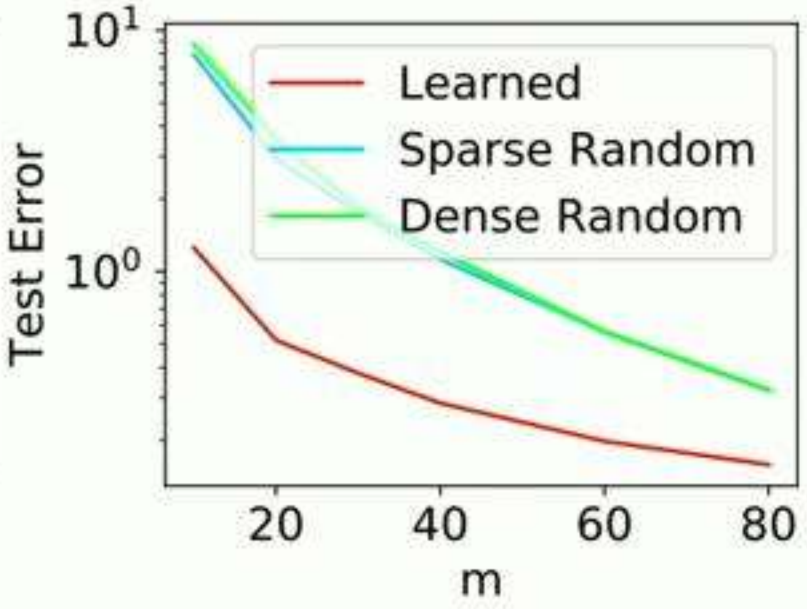
- Compare to random matrices S

Results

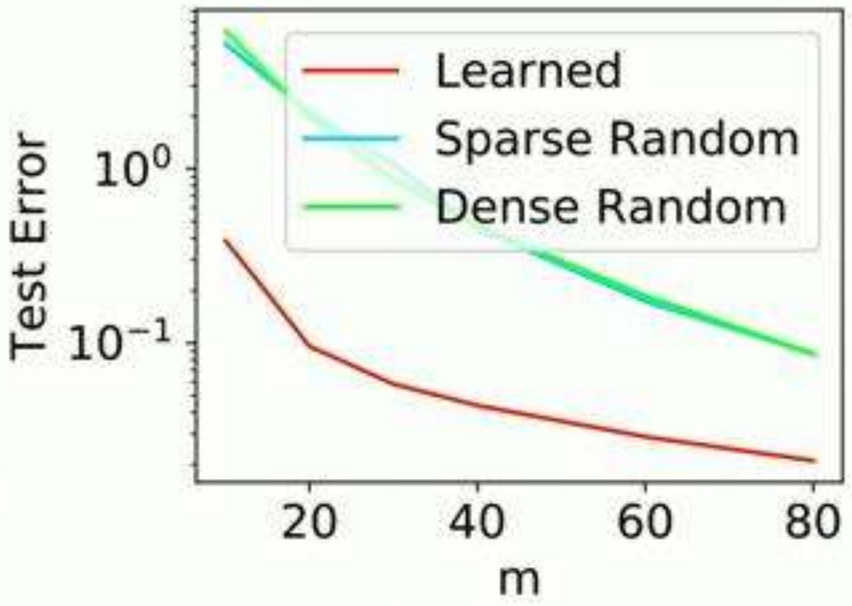
k=10



Tech



Hyper



MIT Logo

Fallback option

- Learned matrices work (much) better, but no guarantees per matrix
- Solution: combine S with **random** rows R
- Lemma: augmenting R with additional (learned) matrix S cannot increase the error of SCW
  - "Sketch monotonicity"
- The algorithm inherits worst-case guarantees from R

Mixed matrices - results

k	m	Sketch	Logo	Hyper	Tech
10	20	Learned	0.1	0.52	2.95
10	20	Mixed	0.2	0.78	3.73
10	20	Random	2.09	2.92	7.99
10	40	Learned	0.04	0.28	1.16
10	40	Mixed	0.05	0.34	1.31
10	40	Random	0.45	1.12	3.28

Conclusions/Questions

- Learned sketches can improve the accuracy/measurement tradeoff for streaming frequency estimation and low-rank approximation
  - **Fallback** options
- Other sketching/streaming problems?
  - Learned Locality-Sensitive Hashing (with Y. Dong, I. Razenshteyn, T. Wagner)
  - Improving **runtime** of low-rank approximation algorithms
- Questions:
  - Sampling complexity
  - Minimizing loss functions (provably)

Learning-Based Low-Rank Approximation

- Sample matrices $A_1 ... A_N$
- Find S that minimizes

$$\sum_i ||A_i - SCW(S, A_i)||_F$$

- Use S happily ever after …
  (as long data come from the same distribution)
- "Details":
  - Use sparse matrices S
    - Random support, optimize values
  - Need to differentiate the loss w.r.t. S
  - Represent SVD as a sequence of power-method applications (each is differentiable)

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
- Taught a class on this topic (with C. Daskalakis)

  https://stellar.mit.edu/S/course/6/sp19/6.890/materials.html
- Insights into "classical" algorithms

Mixed matrices - results

k	m	Sketch	Logo	Hyper	Tech
10	20	Learned	0.1	0.52	2.95
10	20	Mixed	0.2	0.78	3.73
10	20	Random	2.09	2.92	7.99
10	40	Learned	0.04	0.28	1.16
10	40	Mixed	0.05	0.34	1.31
10	40	Random	0.45	1.12	3.28

Fallback option

- Learned matrices work (much) better, but no guarantees per matrix
- Solution: combine S with **random** rows R
- Lemma: augmenting R with additional (learned) matrix S cannot increase the error of SCW
  - "Sketch monotonicity"
- The algorithm inherits worst-case guarantees from R

Mixed matrices - results

k	m	Sketch	Logo	Hyper	Tech
10	20	Learned	0.1	0.52	2.95
10	20	Mixed	0.2	0.78	3.73
10	20	Random	2.09	2.92	7.99
10	40	Learned	0.04	0.28	1.16
10	40	Mixed	0.05	0.34	1.31
10	40	Random	0.45	1.12	3.28

Conclusions/Questions

- Learned sketches can improve the accuracy/measurement tradeoff for streaming frequency estimation and low-rank approximation
  - **Fallback** options
- Other sketching/streaming problems?
  - Learned Locality-Sensitive Hashing (with Y. Dong, I. Razenshteyn, T. Wagner)
  - Improving **runtime** of low-rank approximation algorithms
- Questions:
  - Sampling complexity
  - Minimizing loss functions (provably)

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
- Taught a class on this topic (with C. Daskalakis)

  https://stellar.mit.edu/S/course/6/sp19/6.890/materials.html
- Insights into "classical" algorithms

Fallback option

- Learned matrices work (much) better, but no guarantees per matrix
- Solution: combine S with **random** rows R
- Lemma: augmenting R with additional (learned) matrix S cannot increase the error of SCW
  - "Sketch monotonicity"
- The algorithm inherits worst-case guarantees from R

Approximate Low Rank Approximation

- Instead of

$$A_k = \text{argmin}_{\text{rank-k matrices } B} ||A-B||_F$$

output a rank-k matrix A', so that

$$||A-A'||_F \leq (1+\varepsilon)\, ||A-A_k||_F$$

- Hopefully more efficient than computing exact A_k
  - Sarlos'06, Clarkson-Woodruff'09,13,Halko et al'11….
  - See Woodruff'14 for a survey
- Most of these algos use linear sketches SA
  - S can be dense (FJLT) or sparse (0/+1/-1)