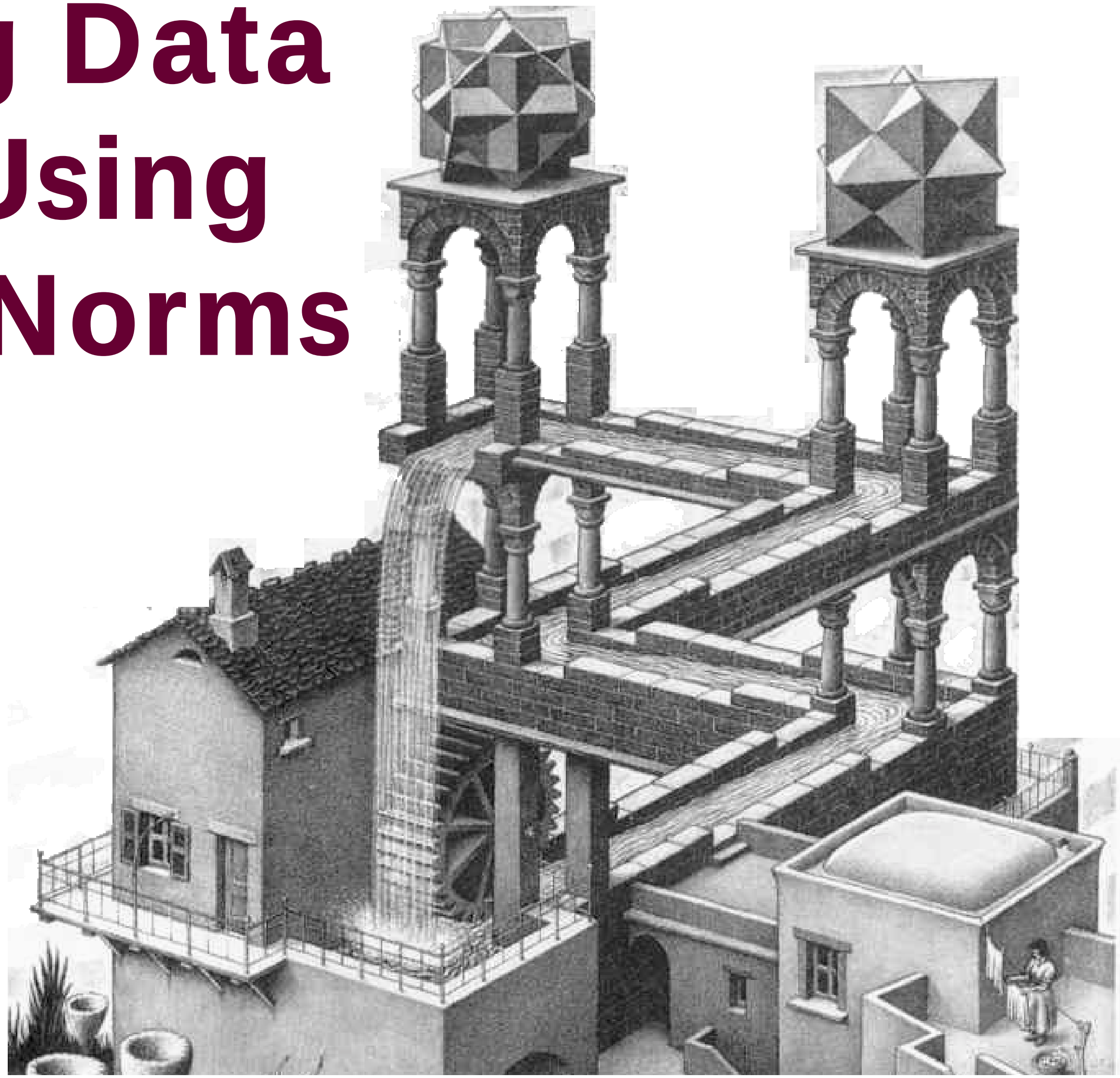
Comparing Data Streams Using Hamming Norms



Graham Cormode,
Mayur Datar,
Piotr Indyk,
S. Muthukrishnan
graham@cormode.org

Data Streams

Data streams occur everywhere:

- Network streams
  - IP packet flow records, phone call records
- Environmental observations
  - Weather readings, other sensor values
- Other streams of values
  - Web clickstreams, stock values…

Streams from IP Networks

Many network flows between (source, dest) pairs

Want a snapshot at time t of the flows

This defines a (massive) vector, and we ask:

- Summarise the current state
- How does state at time t compare with at t'?
- Which past situation does this most resemble, etc.?

Processing Constraints

Network devices have small memory, limited processing power

Want solutions which have fast per-item processing, minimal memory requirements

Backtracking on the input is impossible without explicitly storing it

Informally the "datastream" model of computation

How to measure streams?

The state at any time defines a massive vector

- **Hamming norm:** $\Sigma\ (x_i \neq 0)$

  Number of non-zero entries of the vector

- **Union Size:** $\Sigma\ (x_i + y_i \neq 0)$
- **Hamming difference:** $\Sigma\ ((x_i - y_i) \neq 0) = \Sigma\ (x_i \neq y_i)$

This is the number of places where the vectors differ - a fundamental concept.

Hamming Norm for Counting Distinct Values

Application 1: Maintaining number of distinct values in a relation with inserts and deletes

Important to know number of values for query optimization, approximate query answering, join size estimation etc.

Fully dynamic case, with inserts and deletes: sampling has been shown to be inaccurate.

The Hamming Norm of the stream of updates gives the number of distinct values.

Application to Networks

Application 2: Many questions possible about network streams:

- How many packet flows between distinct pairs of (source, destination)?
- How many flows are losing packets (where packets in one side of network not equal to packets out)?
- Denial of service attacks signalled by large numbers of requests (from spoofed IPs) — so many distinct sources.

All these can be solved by computing Hamming norms.

Our approach

An exact answer is not possible in small space, so we find an approximate answer with probability guarantees.

We will use statistical distributions with provable properties.

Assume an general form of a data stream:

- Pairs (i, j) arrive (meaning "add j to location i")
- The total of values x_i is bounded $|x_i| < U$ for some U.

We will create a small summarizing "sketch" for the stream that allows Hamming Norm, Difference and Union to be approximated.

Hamming Norm of a Stream

Vectors are assumed to be massive, too large to store explicitly. Entries are updated dynamically:

(5,+3), (2, -1), (3, +2), (7, +9), (5, -2), (6, -1), (6, -3), (2, +1), (4, +2), (3, -2), (7, -5), (5, +2), (6, -2), (4, -3), (5, -1)

1	2	3	4	5	6	7	8
0	0	0	-1	2	-3	4	0

Hamming norm of the stream is 4 (4 non-zero entries)

Zeroing in on the Hamming Norm

We can approximate the Hamming norm by finding the Lp norm to the power p for small enough p

Hamming norm of vector **a** is $|\mathbf{a}|_H = \Sigma\, |a_i|^0$
where 0^0 defined $= 0$

Lp norm of a vector is $(\Sigma\, |a_i|^p)^{1/p}$

$$|\mathbf{a}|_H = \Sigma\, |a_i|^0 \leq \Sigma\, |a_i|^p \leq \Sigma\, U^p\, |a_i|^0 \leq U^p\, \Sigma\, |\mathbf{a}|_H$$

Setting $U^p = (1+\varepsilon)$ means $|\mathbf{a}|_H \leq \Sigma\, |a_i|^p \leq (1+\varepsilon)\, |\mathbf{a}|_H$

This fixes $p = \varepsilon / \log U$, allowing us to approximate the Hamming Norm

Finding Lp norm

Relies on results from Indyk '00 on Stable Distributions:

We can use Stable distributions to approximate the Lp norm:

Fact: if $X_i \sim \text{Stable}(p, 0)$ then $\Sigma_i\, a_i\, X_i \sim (\Sigma|a_i^p|)^{1/p}\, \text{Stable}(p,0)$

Create vector $\mathbf{x}$ where each entry is drawn from Stable(p,0)

Compute $|\hat{\mathbf{a}}_H| = \Sigma\, \mathbf{a}_i\, \mathbf{x}_i$ — this quantity has the correct expectation

Can be computed on the stream: with each update (i, j), then update $|\hat{\mathbf{a}}_H| \leftarrow |\hat{\mathbf{a}}_H| + j\mathbf{x}_i$

Guaranteed Accuracy

One estimate is not accurate (variance is high), so repeat several times independently: keep k copies based on independent drawings of the vector $\mathbf{x}$.

Store the values of $\hat{\mathbf{a}}_H$ in a short L_0 *sketch*, sk[1…k].

Find $\text{median}_i(|sk[i]|)$, and scale by $\text{median}(|\text{Stable}(p,0)|) = m$.

Fix $k = O(1/\varepsilon^2 \log 1/\delta)$. Then

$(1-\varepsilon)\ |\mathbf{a}|_H \leq \text{median}(sk)/m \leq (1+\varepsilon)^2\ |\mathbf{a}|_H$ with probability $1-\delta$

Implementation Details

Don't store **x** explicitly — it would take too much space.

Instead, compute each $\mathbf{x}_i$ as a pseudo-random function of i (so use a pseudo-random number generator, initialized by i), and known methods to generate values from Stable Distributions from uniform distributions.

Also need to compute |median(Stable(p,0))| in advance — can do this empirically or numerically.

Properties

Space usage is small: the L_0 sketch consists of $O(1/\varepsilon^2 \log 1/\delta)$ counters

Time per item is to update each counter, $O(1/\varepsilon^2 \log 1/\delta)$

Difference and union of streams is easy to compute:

$$sk(\mathbf{a} + \mathbf{b}) = sk(\mathbf{a}) + sk(\mathbf{b})$$

$$sk(\mathbf{a} - \mathbf{b}) = sk(\mathbf{a}) - sk(\mathbf{b})$$

by linearity of dot product, so can approximate $|\mathbf{a} - \mathbf{b}|_H$ and $|\mathbf{a} + \mathbf{b}|_H$ with the same accuracy.

Complete Algorithm

```
initialize sk[1…k] = 0.0
for all tuples (i,j) do
  initialize random with i
  for s = 1 to k do
    r1 = random(); r2 = random()
    sk[s] = sk[s]+j*stable(r1,r2,p)

for s = 1 to k do
  sk[s] = absolute(sk[s])^p
return median(sk)*scalefactor(p)
```

Simple to implement, can run quickly with small space

Experimental Evaluation

Data Sets

- Generated synthetic data from Zipf distributions with a range of parameters
- Took real Netflow data from one of AT&T's networks
- Each data stream was around 20Mb, working space was around a few Kb.

Parameters We fixed $p = 0.02$ (as small as possible), this sets the scale factor, $\text{median}(|\text{Stable}(0.02,0)|) = 1.425$

Existing Techniques

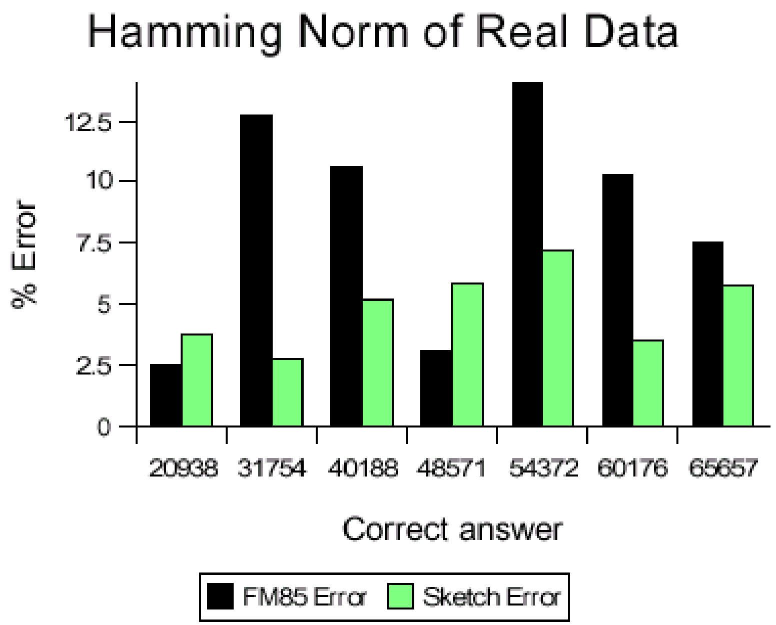
Compared against the "probabilistic counting" algorithm of Flajolet and Martin

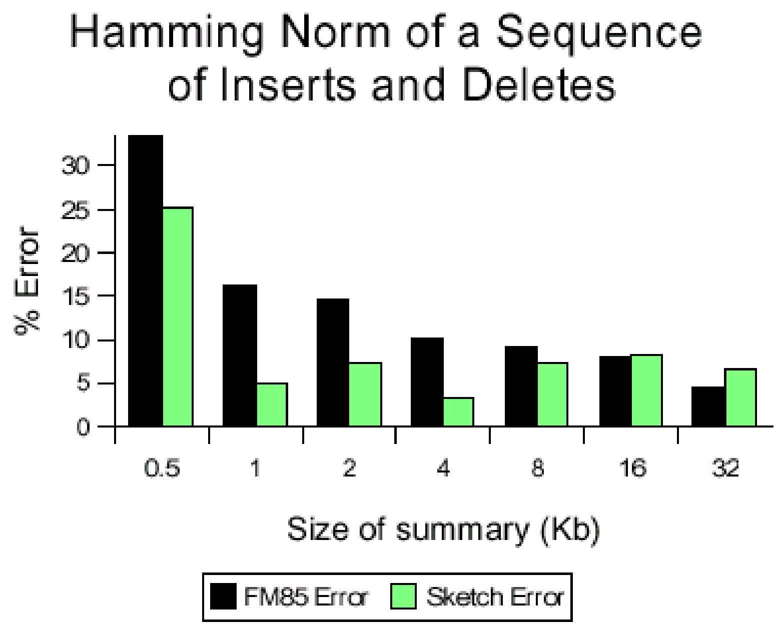
+ Uses a similar amount of space
+ Operates in the data stream model
+ Fast per-item processing

– Can't cope with all situations (eg negative values)

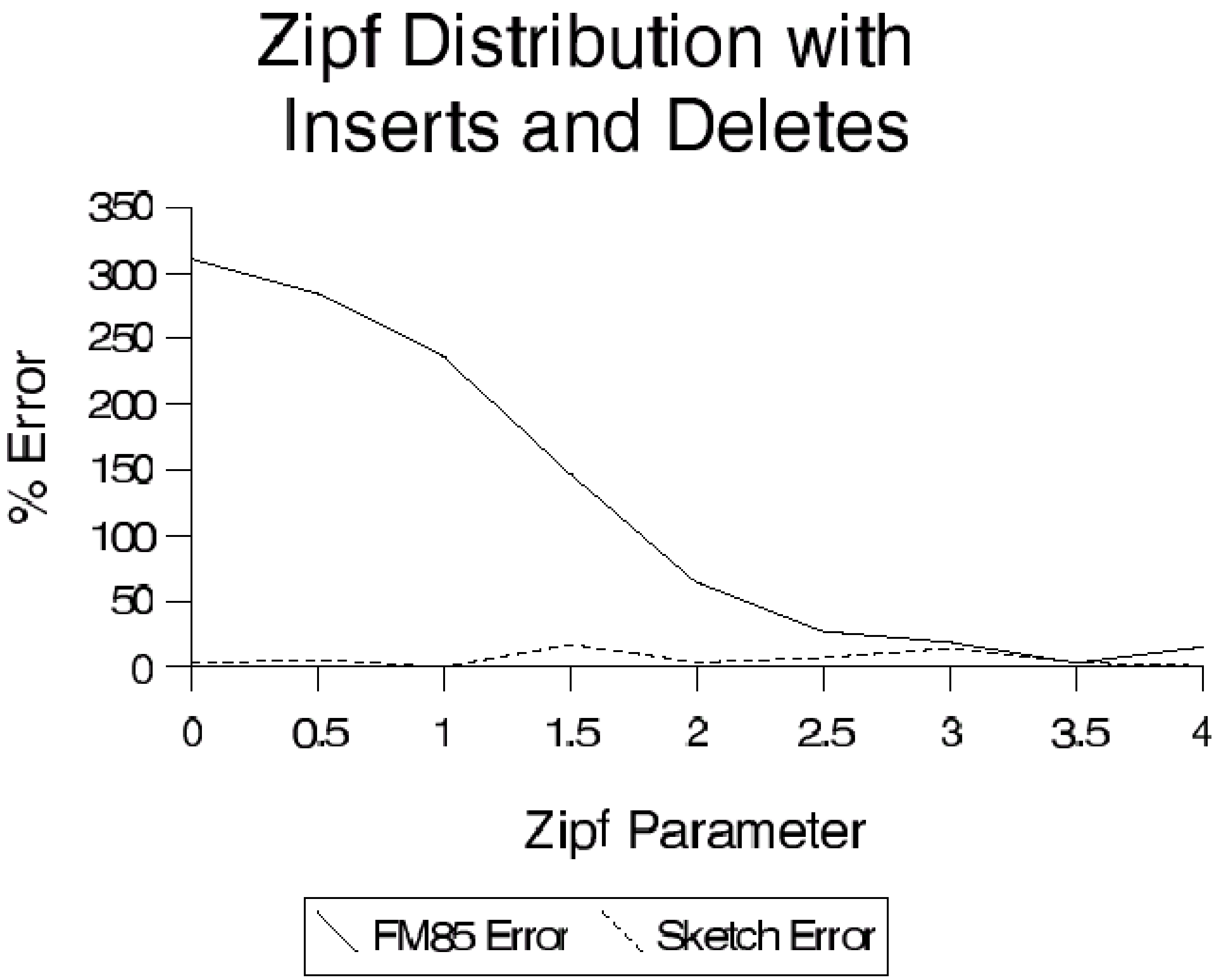
– Can't find the difference between two streams

Hamming Norm Tests

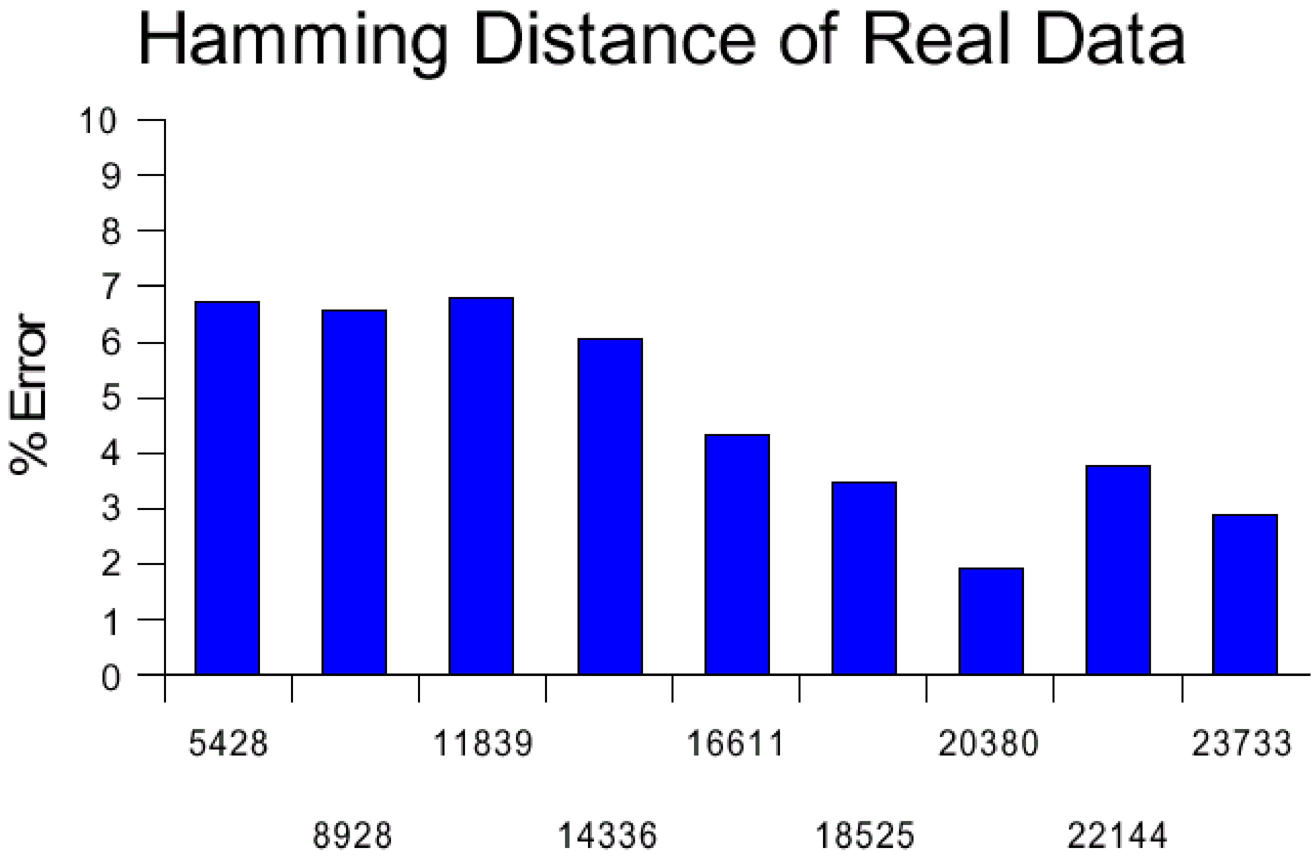




- Performance of our algorithm is better than FM85
- Improves with more workspace
- Slightly slower in practice



- Shows that FM85 can’t cope when values are allowed to be negative, but L_0 sketches retain their accuracy.



- Good performance (~7% error), small memory cost
- Performance of finding union of streams (not shown) also good.

Conclusions

We give a new technique for data stream analysis

Can approximate the Hamming norm, Number of Distinct Items, Hamming difference with only a few kb of space

Suitable for indexing streams

The “L_0 sketch” can be used as a surrogate for the stream in other computations: clustering, searching, querying, all based only on the sketches