Lecture 3. Sampling. Finding frequent elements. The CM-sketch

Ricard Gavaldà

MIRI Seminar on Data Streams, Spring 2015

Contents

Sampling

Sampling

… p # q q 9 8 g r k % & q 3 4 n 3 1 i @ g a a …

At time t, process element t with probability $\alpha(t)$
Compute your query on the sampled elements only

Example: computing an average

- $\alpha(t) = \alpha$, constant: error $\simeq 1/\sqrt{\alpha t} \to 0$
- $\alpha(t) \simeq 1/(\varepsilon^2 t)$: error $\simeq \varepsilon$, constant over time

But a sampled element remains in the sample forever

Reservoir Sampling [Vitter85]

Uniform sampling

Fix k. We want to keep a sample of size k such that after t steps, each of the first t elements in the stream is in the sample with equal probability k/t

Reservoir Sampling [Vitter85]

- Add first k stream elements to the sample
- Choose to sample t-th item with probability k/t
- If sampled, replace any element in the sample with same probability

Reservoir Sampling: why does it work?

Claim: for every t, for every $i \leq t$,

$$P_{i,t} = \Pr[s_i \text{ in sample at time } t] = k/t$$

Suppose true at time t. At time $t+1$,

$$P_{t+1,t+1} = \Pr[s_{t+1} \text{ sampled}] = k/(t+1)$$

and for $i \leq t$, s_i is in the sample if it was before, and not (s_{t+1} sampled and it kicks out exactly s_i)

$$\begin{aligned} P_{i,t+1} &= \frac{k}{t} \cdot \left(1 - \frac{k}{t+1} \cdot \frac{1}{k}\right) = \frac{k}{t} \cdot \left(1 - \frac{1}{t+1}\right) \\ &= \frac{k}{t} \cdot \frac{t}{t+1} = \frac{k}{t+1} \end{aligned}$$

Skip counting [Vitter85]

Instead of deciding whether or not to sample each x_t

- suppose we sample x_t
- compute randomly $m = f(t, k)$
- skip next m records without any processing, process $(m+1)$-th

The distribution of m is computed so that it matches the equations in the previous page (somewhat tricky)

Avoids computation at each step, e.g. random number generation

Observation: $m \to \infty$ as t grows

Finding heavy hitters

Finding Frequent Elements

Heavy Hitters, Elephants, Hotlist analysis, Iceberg queries

Finding frequent elements

Given a sequence *S* of *t* elements, threshold θ,

- Heavy hitters: Find all elements with frequency $> \theta t$
- Top-*k*: Find the *k* most frequent elements

Good sources: [Berinde+09], [Cormode+08]

Finding Frequent Elements

Approximate versions:

- Find a list of elements including all those with frequency $> \theta t$ and none with frequency $< (1-\varepsilon)\theta t$
- Find a list of L of k elements such that if $i \in L$ and $j \notin L$ then $f_i > (1-\varepsilon) f_j$

Sampling?

Intuition: Frequencies in sample $\simeq$ frequencies in stream

Use e.g. reservoir sampling to keep uniform sample

Problems:

- Doesn't work for top-k queries
- For θ-heavy hitters, sample size $\simeq 1/\theta^2$ is required (try it, using Hoeffding)

We present 3 solutions for θ-heavy hitters with memory $O(1/\theta)$

KPS, a simple algorithm for heavy hitters

[Karp-Papadimitriou-Shenker03]

generalizing [Boyer-Moore80, Fischer-Salzberg82, Boyer-Moore82, Misra-Gries82]

- Def: x is a heavy hitter at time t if $f_{x,t} > \theta t$
- There are at most $1/\theta$ of these
- Producing them exactly in 1 pass requires (the obvious) large memory
- Fact: A list containing all θ-heavy hitters of size at most $1/\theta$ can be produced using $O(1/\theta)$ words
- No *false negatives*; maybe *false positives*

A Simple Two-Pass Algorithm

Init(k):

```
Create associative table (K,count):
  - K = the empty set of keys
  - count is a vector of size k, indexed by K, initially 0
```

Update(x):

```
if (x is in K) count[x]++
else
    insert x in K with count 1
    if (|K| = k+1) // K full; discount all items
          for (a in K) do
             count[a]--
             if (count[a] = 0) delete a from (K,count)
```

Query:

```
return the set K
```

Why Does This Work?

- Let $k = 1/\theta - 1$
- Consider an item x not in K at the end of the algorithm
- Each occurrence of x was discounted together with k occurrences of other items
- So at least $(k+1) \cdot f_{x,t}$ items discounted in total
- But number of discounted items at time t can't exceed t
- Therefore $f_{x,t}/\theta = (k+1) \cdot f_{x,t} \leq t$, i.e., x is not θ-heavy hitter
- Contrapositive: all θ-heavy hitters are in K

KPS, memory and time

- k keys, k counts
- with some care, $O(k)$ words for hashing, lists, bookeeping

- $O(1)$ operations per update when no discounting
- there can be at most t/k discounting rounds up to time t (think why)
- and each one takes time $O(k)$
- so $O(1)$ time on average

The Space Saving sketch [Metwally+05]

- KPS followed by many other counter-based methods
  - Lossy Counter, Frequent, Sticky Sampling, GroupTest, ...
- Space-saving:
  - Good update time
  - Some guarantee on count error
  - No false negatives; may have false positives

The Space Saving sketch

Init(k): Create

set of keys K := $\emptyset$

vector *count*, indexed by K

Update(x):

if x is in K then $count[x]++$;

else, if $|K| < k$, add x to K and set $count[x] = 1$;

else, replace an item with lowest count with x

and increase its count by 1

Query:

return the set K;

Why Does This Work?

Claims:

1. If $f_t(x) \geq t/k$, then $x \in K$ at time t
2. For every $x \in K$, $f_t(x) \leq count_t[x] \leq f_t(x) + t/k$

In particular, all items with frequency over t/k are in K

And non-heavy-hitters will have count at most $2t/k$

The bound is most meaningful for frequencies $\gg t/k$

Why Does This Work?

Proof:

- At all times t, $\sum_x count_t[x] = t$
- The minimum count at time t is $\leq t/k$
- Now suppose x not in K a time t
- Either x was never in K (so not frequent)
- Or it was in K but was removed from K. Let $t' \leq t$ the last time it was removed
- Because it was removed, $count_{t'}[x] \leq t'/k \leq t/k$
- I.e. part 1: x not $1/k$-frequent at time t
- For 2, distinguish whether x was in K at time $t-1$ or not and assume (by induction) that 1, 2 true at time $t-1$

Exercise 1

Understand this proof, particularly completing the proof of 2. Not to be delivered.

More on Space Saving

- We omit discussion of efficient implementation - StreamSummary data structure
- Appropriate for very skewed distributions
- Very frequent elements large counters; unfrequent elements low counters
- $\rightarrow$ good approximation of frequent element frequencies
- Paper contains space analysis for powerlaw - Zipf distributions

Exercise 2

Without looking into the paper, propose a data structure to have fast update & query time. Should still use $O(k) = O(1/\theta)$ words, pointers, etc.

Top-k Elements

[Charikar-Chen-(Farach-Colton)04]

- Hash-based (like CM-sketch), not count-based (like Space-Saving)
- Assume $f_1 \geq f_2 \geq f_3 \geq \cdots \geq f_n$
- Given (k, ε), finds a list of k elements such that

$$\text{if } i \in L \text{ then } f_i \geq (1-\varepsilon) f_k$$

- Memory

$$O\left(k \log \frac{t}{\delta} + \frac{\sum_{i=k+1}^{n} f_i^2}{\varepsilon^2 f_k^2} \log \frac{t}{\delta}\right)$$

- I.e., depends on tail. Better for more skewed distributions

The CM-Sketch

The Count-Min Sketch

[Cormode-Muthukrishnan 04]
Like Space Saving:

- Provides an approximation f'_x to f_x, for every x
- Can be used (less directly) to find θ-heavy hitters
- Uses memory $O(1/\theta)$

Unlike Space Saving:

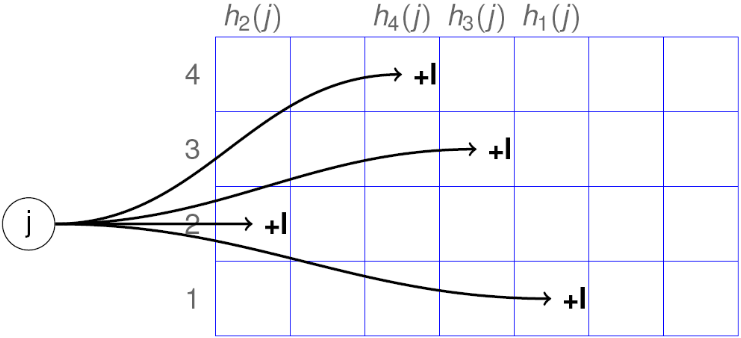
- It is randomized - hash functions instead of counters
- Supports additions and deletions
- Can be used as basis for several other queries

The Count-Min Sketch

- Vector $F[n]$. Assumes $F[i] \geq 0$ for all i, at all times
- Provides estimations F' of F such that
  1. $F[i] \leq F'[i]$ for all i
  2. $F'[i] \leq F[i] + \varepsilon |F|_1$ for all i, with probability $\geq 1 - \delta$

  where $|F|_1 = \sum_i F[i]$
- Note: $|F|_1$ may be $\ll$ stream length, if subtractions allowed
- Uses $O(\frac{1}{\varepsilon} \ln \frac{n}{\delta})$ memory words, $O(\ln \frac{n}{\delta})$ update time

The Count-Min Sketch



source: A. Bifet,

`http://albertbifet.com/comp423523a-2012-stream-data-mining/`

The Count-Min Sketch

- d independent hash functions $h_1 \ldots h_d$: $[1..n] \rightarrow [1..w]$
- one "memory cell" for each $h_j(i)$
- On instruction "$F[i]$ += v", do $h_j(i)$ += v for all $j \in 1 \ldots d$
- Estimation:

$$F'[i] = \min\{\, h_j(i) \mid j = 1..d \,\}$$

The Count-Min Sketch

$F'[i] = \min\{\, h_j(i) \mid j = 1..d \,\}$

- $F'[i] \geq F[i]$
  For each instruction involving i, we update all counts $h_j(i)$
  $F[i] \geq 0$ at all times for all i

- $F'[i] = F[i]$?
  No: cell $h_j(i)$ is also incremented by $k \neq i$ if $h_j(k) = h_j(i)$
- But it is unlikely that this occurs very often
- min instead of average $\rightarrow$ Markov instead of Chebyshev or Hoeffding

The Count-Min Sketch: Proof of main bound

- Fix j. Def random var $I_{ijk} = 1$ if $h_j(i) = h_j(k)$, 0 otherwise
- If h good hash function

$$E[I_{ijk}] \leq 1/\text{range}(h_j) = 1/w$$

- Def $X_{ij} = \sum_k I_{ijk} F[k]$. Then

$$E[X_{ij}] = \sum_k E[I_{ijk}] F[k] \leq |F|_1/w$$

The Count-Min Sketch: Proof of main bound (2)

Then by Markov's inequality and pairwise independence:

$$\Pr[X_{ij} \geq \varepsilon|F|_1] \leq E[X_{ij}]/(\varepsilon|F|_1) \leq (|F|_1/w)/(\varepsilon|F|_1) \leq 1/2$$

if $w = 2/\varepsilon$. Then:

$$\begin{aligned}
& \Pr[F'[i] \geq F[i] + \varepsilon|F|_1] \\
= \quad & \Pr[\forall j \ : \ F[i] + X_{ij} \geq F[i] + \varepsilon|F|_1] \\
= \quad & \Pr[\forall j \ : \ X_{ij} \geq \varepsilon|F|_1] \\
\leq \quad & (1/2)^d = \delta \qquad \text{if } d = \log(1/\delta)
\end{aligned}$$

for one fixed i. To have good estimates for all i simultaneously, use $d = \log(n/\delta)$ and use union bound

The Count-Min Sketch: Summary

- Memory is $\frac{2}{\varepsilon} \log \frac{1}{\delta}$ words
- Update time $O(\log \frac{1}{\delta})$
- Replace $\log(1/\delta)$ with $\log(n/\delta)$ if the bound needs to hold for all i simultaneously

  "Pr[for all $i, \ldots] \leq \delta$" instead of "for all i, $\Pr[\ldots] \leq \delta$"
- Error for $F[i]$ is ε relative to $|F|_1$, not to $F[i]$
- This is bad for counts $F[i]$ small w.r.t. $|F|_1$

- Any problem where we care about large $F[i]$'s only?

Applications of the CM-sketch

Heavy Hitters, revisited

- i is a θ-heavy hitter if $F[i] \geq \theta t$
- The CM-sketch with width θ guarantees

$$F[i] \leq F'[i] \leq F[i] + \theta t$$

- So: If we output all i s.t. $F'[i] \geq \theta t$, we output all heavy hitters; no false negatives

But we can't cycle through all n candidates one by one!

Range-Sum queries

Range-sum query

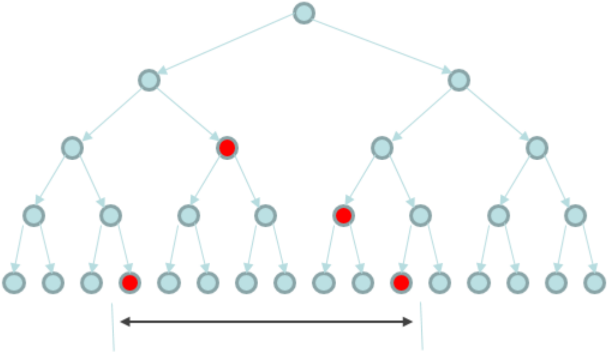
Given a, b, return $\sum_{i=a}^{b} F[i]$

Example: how many packets received came from the IP range 172.16.xxx.xxx?

We show:

- A variant of CM-sketch supports range-sum queries efficiently
- Answering range-sum queries efficiently $\longrightarrow$ finding heavy hitters efficiently

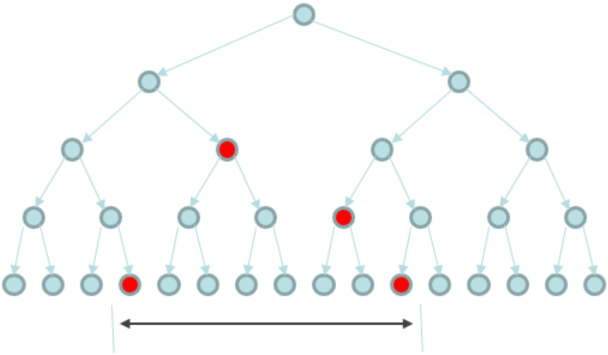
Fom CM-sketch to range-sum queries



For $p = 0 \dots \log n$, for each $j = ...$, keep the value of $sum(j2^p \dots (j+1)2^p - 1)$

Any interval $[a, b]$ is the sum of $O(\log n)$ such values

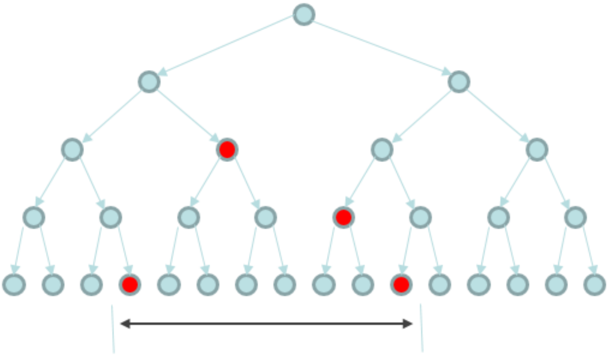
From CM-sketch to range-sum queries



Keep

- One CM-sketch for each 2^p to store $sum(j2^p \ldots (j+1)2^p - 1)$ for each j
- (Perhaps?) Or a single CM-sketch whose set of items is the set of intervals indexed by pairs (p, j)

From CM-sketch to range-sum queries



When receiving i, update the counts for ranges where i lies = ancestors of i in the tree

When queried $sum(a..b)$, decompose $[a..b]$ as sum of such intervals, retrieve and add their sums

From Range-sum queries to heavy hitters

- Adaptively search for heavy hitters in the tree
- if a node has count $< \theta t$, do not explore its children: no heavy hitters below
- if a node has count $\geq \theta t$, explore both children
- when reaching a leaf, we know whether it's a heavy hitter
- the sum of counts at any one level of the tree is t
- no more than $1/\theta$ of them may have frequency $\geq \theta t$
- Efficiency: no more than $1/\theta$ nodes of each level are expanded

Exercise 3

Formalize the algorithms above:

- For computing range-sum queries given CM-sketch
- Form finding all heavy hitters using range-sum queries

and tell their memory usage and update time

Inner product, revisited

Run CM-sketches for u and v using *same* hash functions

Estimate $IP(u,v) = \sum_i u_i v_i$ by

$$\min_j \sum_r \text{count}^u(j,r) \cdot \text{count}^v(j,r)$$

Inner product, revisited

- Observe that

$$u \cdot v = \sum_i u(i)v(i) = \sum_r \sum_{i:h_j(i)=r} u(i)v(i)$$

- Let $\mathrm{count}^u(j,r)$, $\mathrm{count}^v(j,r)$ be the cells for j-th function, r-th value in CM-sketches for u and v. Intuitively,

$$\mathrm{count}^u(j,r) \cdot \mathrm{count}^v(j,r) \cong \sum_{i:h_j(i)=r} u(i)v(i)$$

- So we estimate $u \cdot v$ by

$$\min_j \sum_r \mathrm{count}^u(j,r) \cdot \mathrm{count}^v(j,r)$$

  in time $O(\frac{1}{\varepsilon} \log \frac{1}{\delta})$
- Formally the proof is as in CM-sketch

Inner product, revisited

[AMS]

- $|AMS - IP(u,v)| \leq \varepsilon\, IP(u,v)$
- memory $O(\frac{1}{\varepsilon^2} \ln \frac{1}{\delta})$ words
- update time $O(\frac{1}{\varepsilon^2} \ln \frac{1}{\delta})$

[CountMin]

- $|CM - IP(u,v)| \leq \varepsilon\, |u|_1 |v|_1$
- memory $O(\frac{1}{\varepsilon} \ln \frac{1}{\delta})$ words
- update time $O(\frac{1}{\varepsilon} \ln \frac{1}{\delta})$

$\therefore$ CM-Sketch better memory and update time, but absolute instead of relative, approximation

Quantile computation

Given i, θ, compute all the θ-quantiles
I.e., find for all k the $q(k)$ such that

$$\sum_{i=1}^{q(k)} F[i] = k\theta \sum_{i=1}^{n} F[i]$$

- Can be done from the CM-sketch with $O(n)$ estimation time
- Can be done faster using Range-Sum queries

Histogram Computation. Inverse distributions

- We have asked the question

  "given i, what is the frequence of i?"
- Inverse question:

  "given f, how many i's have frequence f?"

- Can be done in space $O(\frac{1}{\varepsilon^2} \ln \frac{1}{\delta})$
- The plot for all f's is a histogram

Sketches & random linear projections

- AMS, CM-Sketch, Cohen's counter, ..., sketch vector F as

$$\sum_{i=1}^{n} F[i]h_j(i) \quad \text{for hash functions } h_j,\ j = 1 \dots d$$

- Equivalently, $S = HF$, $F \in \mathbb{R}^n$, $H \in \mathbb{R}^{d \times n}, S \in \mathbb{R}^d$
- A linear projection from dimension n to dimension d
- Vectors that are close remain close
- Vectors that are far most likely remain far
- More next week