Sketching

Inge Li Gørtz

These notes are heavily inspired by the lecture notes by Moses Charikar and Chandra Chekuri on the same subject.

1 Sketches

Informally, a data sketch is a smaller description of a stream of data that enables the calculation or estimate of a property of the data. In other words a compact summary of the data.

An important attribute of sketches is that they are composable. Suppose we have data streams S_1 and S_2 with corresponding sketches $sk(S_1)$ and $sk(S_2)$. We wish there to be an efficiently computable function f where

$$sk(S_1 \cup S_2) = f(sk(S_1), sk(S_2)) \ .$$

2 Hashing

A hash function $h : U \to [m]$ is pairwise independent if for all $x \neq y \in U$ and $q, r \in [m]$:

$$P[h(x) = q \wedge h(y) = r] = \frac{1}{m^2} \ .$$

Equivalently, the following two conditions hold:

- for any $x \in U$, $h(x)$ is uniform in $[m]$,
- for any $x \neq y \in U$, $h(x)$ and $h(y)$ are independent.

3 CountMin sketch

The CountMin sketch is a solution to the heavy hitters problem developed by Cormode and Muthukrishnan [1]. The idea of the CountMin sketch is to use a collection of pairwise independent hash functions to hash each element in the stream, keeping track of the number of times each bucket is hashed to.

Initialization Initialize d pairwise independent hash functions $h_j : [n] \to [w]$ with w buckets each for $j \in [d]$. For each bucket b of each hash function j, store a counter $C_j(b)$ initially set to 0.

Building the data structure: For each element i of the stream, hash i using each hash function and increment $C_j(h_j(i))$ for all $j \in [d]$.

Algorithm 1: CountMin

Initialize d independent pairwise independent hash functions $h_j : [n] \to [w]$.
Set counter $C_j(b) = 0$ for all $j \in [d]$ and $b \in [w]$.
while *Stream S not empty* **do**
  if *Insert(x)* **then**
    for $j = 1 \dots d$ **do**
      $C_j(h_j(x)) = +1$
    end
  else if *Frequency(i)* **then**
    return $\hat{f}_x = \min_{j \in [d]} C_j(h_j(x))$.
  end
end

Querying the data structure Given element i, return $\hat{f}_i = \min_{j \in [d]} C_j(h_j(i))$.

3.1 Analysis

In this section we will show that the by choosing d and w in the right way, we can obtain the following guarantees on the estimated values.

Theorem 1. *The estimator $\hat{f}_i$ has the following property: $\hat{f}_i \geq f_i$ and with probability at least $1 - \delta$, $\hat{f}_i \leq f_i + \epsilon m$, where m is the length of the stream.*

To show this we first analyze $\hat{f}_i$ wrt the parameters d and w. First we give a lower bound on $\hat{f}_i$.

Lemma 2. *The estimator $\hat{f}_i$ has the property $\hat{f}_i \geq f_i$.*

Proof. Clearly for any $i \in [n]$ and $1 \leq j \leq d$, it holds that $C(h_j(i)) \geq f_i$ and hence $\hat{f}_i \geq f_i$. □

We now give an upper bound on $\hat{f}_i$. This bound is probabilistic, i.e., it holds with a certain probability, which depends on d—the number of hash functions in the CountMin sketch.

Lemma 3. *With probability at least $1 - \left(\frac{1}{2}\right)^d$, we have $\hat{f}_i \leq f_i + \frac{2}{w} \cdot m$, where m is the length of the stream.*

Proof. Fix an element $i \in [n]$ and let $Z_j = C_j(h_j(i))$ be the value of the counter in row j to which i is hashed. Let $b = h_j(i)$ be the bucket that i hashes to in row j. We can compute the expectation of the value Z_j as follows:

$$\mathbb{E}[Z_j] = \mathbb{E}\left[\sum_{s:h_j(s)=b} f_s\right] = f_i + \frac{1}{w}\sum_{s:s\neq i} f_s \leq f_i + \frac{m}{w}$$

since the sum of all frequencies is m (the number of elements in the stream), and each element has probability $1/w$ of mapping to a particular bucket (pairwise independence of h_j gives us that $\Pr[h_j(s) = b] \leq 1/w$).

We now want to bound the probability that $Z_j \geq f_i + \frac{2}{w} \cdot m$. We have

$$\Pr\left(Z_j \geq f_i + \frac{2m}{w}\right) = \Pr\left(Z_j - f_i \geq \frac{2m}{w}\right)$$

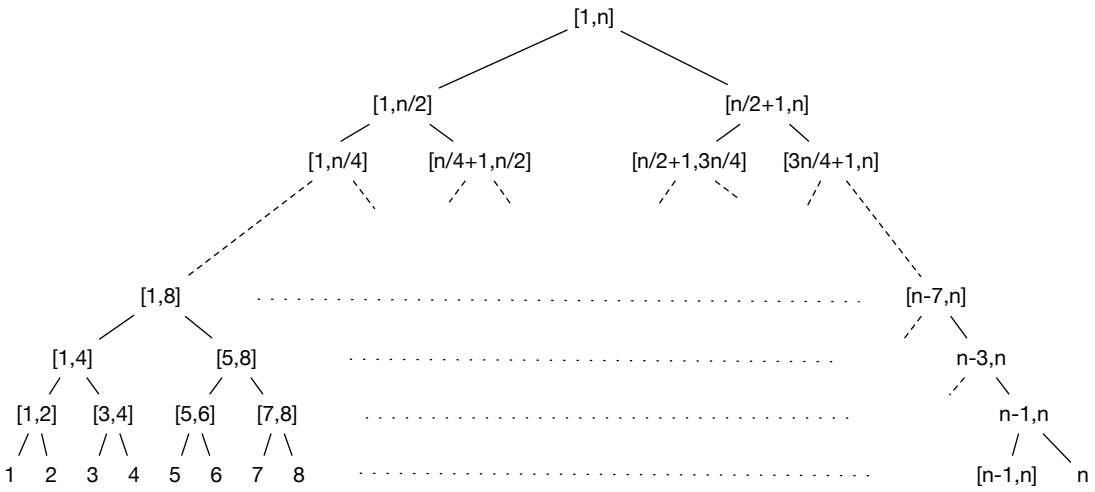

Figure 1: Tree of dyadic intervals

Since the CountMin sketch only overestimates frequencies implying $Z_j - f_i \geq 0$, we can use Markov's inequality to get

$$\Pr\left(Z_j - f_i \geq \frac{2m}{w}\right) \leq \frac{\mathbb{E}[Z_j - f_i]}{2\frac{m}{w}} = \frac{\mathbb{E}[Z_j] - f_i}{2\frac{m}{w}} \leq \frac{(f_i + \frac{m}{w}) - f_i}{2\frac{m}{w}} \leq \frac{1}{2}\,.$$

Since we select each hash function independently, we have that

$$\Pr\left(\hat{f}_i \geq f_i + \frac{2m}{w}\right) = \prod_{j\in[d]} \Pr\left(Z_j \geq f_i + \frac{2m}{w}\right) \leq \left(\frac{1}{2}\right)^d\,.$$

□

Setting $w = \frac{2}{\epsilon}$ and $d = \lg\frac{1}{\delta}$ we get $\Pr\left(\hat{f}_i \geq f_i + \epsilon m\right) \leq \delta$. This concludes the proof of Theorem 1.

Space and time The space usage of the CountMin sketch is $O(dw) = O(\frac{2}{\epsilon}\lg\frac{1}{\delta})$ words, i.e., $O(\frac{2}{\epsilon}\lg\frac{1}{\delta}(\lg m + \lg n)$ bits. The query and update time is $O(d) = O(\lg\frac{1}{\delta})$.

4 Applications of CountMin Sketch

The CountMin sketch can be used to efficiently support the following queries:

- Range queries: "How many elements in the stream have value between a and b?
- Heavy hitters: listing all heavy hitters (elements with frequency at least m/k).

This can be done using CountMin sketches over *dyadic intervals*.

Dyadic intervals The dyadic intervals of $[1,\ldots,m]$ are the set of intervals of the form $[j\frac{m}{2^i} + 1,\ldots,(j+1)\frac{n}{2^i}]$ for all $0 \leq i \leq \lg m$ and all $0 \leq j \leq 2^i - 1$. See Figure 1.

4.1 Range Queries

For each level of the tree in Figure 1 we store a separate CountMin sketch data structure. For level j the jth CountMin sketch treats two elements that fall into the same interval in level j as the same element. For all intervals i in the tree, let $C(i)$ denote the value that the appropriate CountMin sketch returns for i. Let the frequency of interval i denote the sum of the frequencies over all elements in interval i.

Any range query can be reduced to at most $2 \lg n$ dyadic range queries (see exercises), which each can be reduced to a single point query (frequency query of an element). Each lement i is a member of $\lg n$ different ranges, corresponding to nodes in the tree from the leaf corresponding to i to the root. We update each of the $\lg n$ sketches whenever an element arrives in the stream. Given a range query we compute the at most $2 \lg n$ ranges covering our range query, and pose that many frequency queries to our sketches, returning the sum of the queries as the estimate.

This gives us the following lemma.

Lemma 4. *Let $f_{[a,b]}$ be the number of elements in the stream with value in the range $[a, b]$, and let $\hat{f}_{[a,b]}$ be the value returned by our data structure. Then $f_{[a,b]} \leq \hat{f}_{[a,b]}$ and with probability at least $1 - \delta$,*

$$\hat{f}_{[a,b]} \leq f_{[a,b]} + 2\epsilon \lg m \ .$$

Proof. That $f_{[a,b]} \leq \hat{f}_{[a,b]}$ follows directly from Lemma 2. The expected additive error for each estimator used to compute $\hat{f}_{[a,b]}$ is m/w. By linearity of expectation the total expected error over tje $2 \lg n$ ranges is thus $2 \lg n(m/w)$. Using the same Markov inequality argument as before, we get that the probability that this additive error is more than $2\epsilon m \lg n$ is less than $1/2$. Thus the probability that all of them are greater than this is at most δ. □

In the exercises we will show how to scale the error to get a better bound on $\hat{f}_{[a,b]}$.

The space for the data structure is the space for $\lg n$ CountMin sketches, i.e., $O(\frac{\lg n}{\epsilon} \lg \frac{1}{\delta})$. The query and update time is. The query and update time is $O(d) = O(\lg n \lg \frac{1}{\delta})$.

4.2 Heavy Hitters

Using dyadic intervals we can solve the heavy hitters problem in the turnstile model (where updates are both positive and negative, with the restriction that count of any item is never less than zero). There is a more simple solution using heaps in the cash register model (all updates are positive).

4.2.1 Heavy hitters in the cash register model

We maintain a CountMin sketch, a min-heap and the number m of elements seen so far. When a new element i arrives, we increment m by one, update the sketch, and perform a Frequency(i) query to get the value of $\hat{f}_i$. If $\hat{f}_i > m/k$ we insert i into the heap with value $\hat{f}_i$. If i is already in the heap we delete it before we insert it with the new value. To keep the heap small we then check if the minimum element in the heap has a key less than m/k (with the new updated m). If this is the case we extract the minimum element from the heap. To return all heavy hitters we run through the heap and return all elements with estimated frequency greater than m/k. To ensure this we make a frequency query for each element in the heap.

Analysis Assume that we set $\epsilon = 1/2k$. Assume for simplicity that the CountMin sketches makes no large errors[1], i.e., that $\hat{f}_i \leq f_i + \epsilon m$ for all i. This implies that every element x with $\hat{f}_i \geq m/k$ has true frequency f_i at least $m/k - \epsilon m = \frac{m}{2k}$. There are at most $2k$ intervals with frequency $m/(2k)$ and thus the number of elements in the heap is no more than $O(k)$.

The space is $O(dw) = O(\frac{1}{\epsilon} \lg \frac{1}{\delta}) = O(k \lg \frac{1}{\delta})$ words for the CountMin sketch and $O(k)$ space for the heap. Update time is $O(\frac{1}{\delta} + \lg k)$.

4.2.2 Heavy hitters in the turnstile model

As in the range queries we store a separate CountMin sketch data structure for each level of the tree in Figure 1. For level j the jth CountMin sketch treats two elements that fall into the same interval in level j as the same element. For all intervals i in the tree, let $C(i)$ denote the value that the appropriate CountMin sketch returns for i. Let the frequency of interval i denote the sum of the frequencies over all elements in interval i.

To find the heavy hitters we traverse the tree from the root only traversing the children whose intervals have estimated frequency at least m/k and return the leaves whose estimated frequency is at least m/k. Since the frequency of an interval is at least that of its children and the CountMin sketch overestimates the frequencies, we will reach all leaves with frequency at least m/k.

Analysis Assume $\epsilon = \frac{1}{2k}$. There are $\lg n$ CountMin sketches (one for each level in the tree). Thus the total space usage is $O(\frac{1}{\epsilon} \lg \left(\frac{1}{\delta}\right) \lg n) = O(k \lg \left(\frac{1}{\delta}\right) \lg n)$ words. The update time is $O(\log n \cdot \lg \frac{1}{\delta})$.

Assume for simplicity that the CountMin sketches makes no large errors[2], i.e., that $\hat{f}_i \leq f_i + \epsilon m$ for all i. This implies that every element x with $\hat{f}_i \geq m/k$ has true frequency f_i at least $m/k - \epsilon m = \frac{m}{2k}$. For any given level, the sum over all frequencies in that level is m. Thus, in any level, there are at most $2k$ intervals with frequency $m/2k$. Therefore, we only explore the children of at most $O(k)$ intervals in any given level, so the total number of intervals queried is $O(k \log n)$. The total query time is $O(k \log n \cdot \lg \frac{1}{\delta})$.

[1] For a version with a more precise analysis see [1].

[2] For a version with a more precise analysis see [1].

5 CountSketch

Algorithm 2: CountSketch

Initialize d independent hash functions $h_j : [n] \to [w]$.
Initialize d independent hash functions $s_j : [n] \to \{\pm 1\}$.
Set counter $C[j, b] = 0$ for all $j \in [d]$ and $b \in [w]$.
while *Stream S not empty* **do**
  if *Insert(x)* **then**
    for $j = 1 \dots d$ **do**
      $C[j, h_j(x)] =+ s_j(i)$
    end
  else if *Frequency(i)* **then**
    $\hat{f}_{ij} = C(h_j(i)) \cdot s_j(i)$
    return $\widetilde{f}_{ij} = \text{median}_{j \in [d]} \hat{f}_{ij}$
  end
end

References

[1] Cormode, G. and Muthukrishnan, S., 2005. An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms, 55(1), pp.58-75.