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 pairwise indedependent hash function $h : [n] \to [m]$ satifies that for all $x_1, x_2 \in [n]$ and $y_1, y_2 \in [m]$:

$$\Pr[h(x_1) = y_1, h(x_2) = y_2] = \Pr[h(x_1) = y_1] \Pr[h(x_2) = y_2] .$$

3 CountMin sketch

The CountMin sketch is a solution to the heavy hitters problem developed by Cormode and Muthukrishnan '05. 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 $C(h_j(i))$ for all $j \in [d]$.

Algorithm 1: CountMin

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

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

3.1 Analysis

Lemma 1 *The estimator* $\hat{f}_i$ *has the following property:* $\hat{f}_i \geq f_i$ *and with probability at least* $1 - \left(\frac{1}{2}\right)^d$, $\hat{f}_i \leq f_i + \frac{2}{w} \cdot m$, *where* m *is the length of the stream.*

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

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

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

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 $\Pr[h_j(s) = h_j(i)] \leq 1/w$).

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

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

Since the count-min 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 $\Pr\left(\hat{f}_i \geq f_i + \epsilon m\right) \leq \delta$.

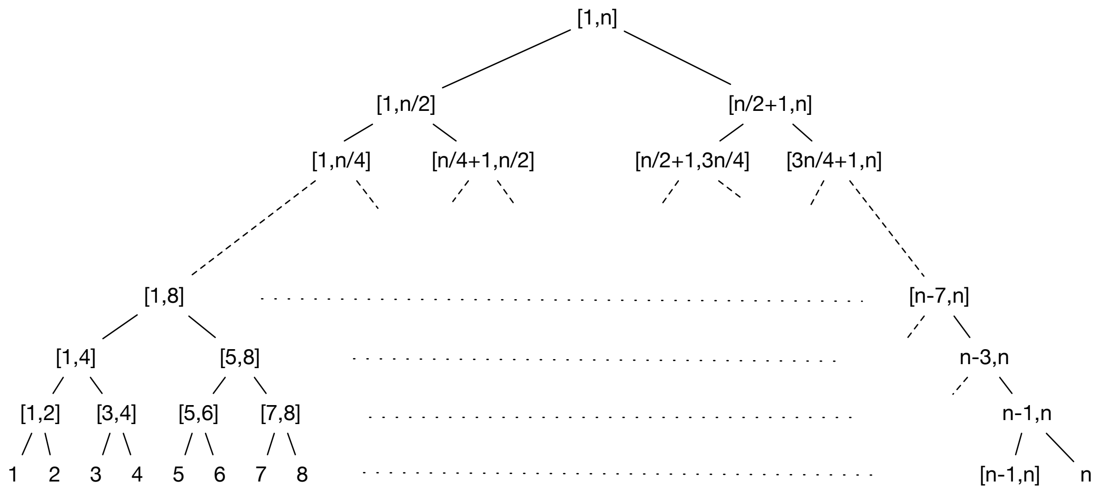

Figure 1: Tree of dyadic intervals

Theorem 2 *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.*

The space usage of the CountMin sketch is $dw = \frac{2}{\epsilon} \lg \frac{1}{\delta}$. The query and processing time is $O(d) = O(\lg \frac{1}{\delta})$.

3.2 Extensions to CountMin

We will now see how to use the CountMin sketch 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).

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 $0 \leq i \lg m$ and all $0 \leq j \leq 2^i - 1$. See Figure **??**.

For each level of the tree in Figure **??** 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.

3.2.1 Heavy hitters

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 frequency at least m/k and return the leaves whose 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 There are $\lg n$ CountMin sketches (one for each level in the tree). Thus the total space usage is $O(\frac{1}{\epsilon}\lg\left(\frac{1}{\delta}\right)\lg n)$.

For any given row, the sum over all frequencies in that row is m. Thus, in any row, there are at most k intervals with frequency m/k. Therefore, we only explore the children of at most k intervals in any given row, so the total number of intervals queried is $O(k \log n)$. The total query time is $O(k \log n \cdot \lg \frac{1}{\delta})$.

4 CountSketch

Algorithm 2: CountSketch

```
Initialize d independent hash functions h_j : [n] → [w].
Initialize d independent hash functions s_j : [n] → {±1}.
Set counter C[j, b] = 0 for all j ∈ [n] and b ∈ [w].
if Insert(x) then
    while Stream S not empty do
        for j = 1 ... n do
            C[j, h_j(x)] =+ s_j(i)
        end
    end
else if Frequency(i) then
    f̂_ij = C(h_j(i)) · s_j(i)
    return f̃_ij = median_{j∈[d]} f̂_ij
end
```