Count-Min Sketch

A probabilistic sub-linear space streaming algorithm which can be used to summarize a data stream in many different ways.

A Count-Min sketch is mostly used to find **Heavy Hitters** in a data set, that is, all elements i whose frequency $a_i > T$. It does seem like the last data structure I studied, the **HyperLogLog** counter, would be appropriate for this task. Alas, HyperLogLog counters cannot accurately (without a decent margin of error) perform frequency estimations on streaming data.

The Count-Min sketch is pretty recent. It was introduced in 2003 and since then, has inspired many applications, extensions, and variations.

This probabilistic data structure has applications in Compressed sensing, Networking, Databases, and Eclectics (NLP, Security, Machine Learning).

"Sketching" Data Structures

Count-Min Sketch belongs to a class of probabilistic data structures I'd call "sketching" data structures that are basically probabilistic data structures that can be applied to input streams. Another example of a "sketching" data structure is the **Bloom Filter** which is more popular than Count-Min sketches.

An "Intuitive" explanation of Count-Min sketch

The problem is to record a numerical value associated with each element, say the number of occurrences of the element in a stream.

The Count-Min sketch works as follows: we have k different hash functions and k different lists which are indexed by the outputs of these functions. Initially, we have all fields in each list set to 0. When we increase the count of an element, we increment all corresponding k fields in the different lists (given by the hash values of the element). To obtain the count of an element, we take the minimum of the k fields that correspond to that element.

Relation to Bloom Filters

It is interesting to note that if we take the Count-Min sketch data structure and make the counters such that they saturate at 1, we obtain a data structure very similar to Bloom Filters in operation and performance.

Some Advantages of Count-Min Sketch

1. Space used is proportional to $1/\epsilon$;
2. The update time is significantly sublinear in the size of the sketch;
3. It requires only pairwise independent hash functions that are simple to construct;

4. This sketch can be used for several different queries and multiple applications;

Count-Min Sketch Internals

The data structure primarily consists of a fixed array of counters, of width w and depth d. The counters are initialized to all zeros. Each row of counters is associated with a different hash function. The hash function maps items uniformly onto the range $\{1, 2, \ldots, w\}$. The hash functions do not need to be particulary strong or as complex as cryptographic hash functions. So you can just use a linear hash. But it is important that the hash function for each row be distinct.

UPDATE(i, c) updates the data structure in a straightforward way. In each row, the corresponding hash function is applied to i to determine a corresponding counter.

For ESTIMATE(i), the process is similar. For each row, the corresponding hash function is applied to i to look up one of the counters. Across all rows, the estimate is found as the minimum of all the probed counters.

Why it works

It's proven in the paper by *Muthukrishnan* and *Cormode* that for a sketch of size $w \times d$ with total count N, it follows that any estimate has error at most $2N/w$, with probability at least $1 - \frac{1}{2}^d$. So setting the parameters w and d large enough allows us to achieve very high accuracy while using relatively little space.

Implementation of Count-Min Sketch

The sketch can be implemented in various ways depending on the application and hardware on which the sketch will run on. As a result, the implementations of the sketch is different in the single-threaded, parallel, and distributed cases. But I'll only consider the single-threaded case here.

Single Threaded Implementation

It is straightforward to implement the sketch in a traditional single CPU environment. Below is the pseudocode for the INIT, UPDATE, and ESTIMATE procedures.

INIT initializes the array C of $w \times d$ counters to 0, and picks values for the hash functions based on the prime p.

Algorithm *INIT*(w, d, p)
1. $C[1, 1] \ldots C[d, w] \leftarrow 0$
2. **for** $j \leftarrow 1$ **to** d
3. Pick a_j, b_j uniformly from $[1 \ldots p]$

4. $N = 0$

UPDATE(i, c) updates the total count N with c, and the loop hashes i to its counter in each row, and updates the counter there.

Algorithm *UPDATE*(i, c)
1. $N = N + c$
2. **for** $j \leftarrow 1$ **to** d
3. $\quad h_j(i) = (a_j \times i + b_j)\ mod\, w$
4. $\quad C[j, h_j(i)] = C[j, h_j(i)] + c$

ESTIMATE(i) is identical to this loop: given i, perform some hashing on i keeping track of the value of the counter in each row, then return the minimum counter. This is the smallest value of $C[j, h_j(i)]$ over the d values of j.

Algorithm *ESTIMATE*(i)
1. $e = \infty$
2. **for** $j \leftarrow 1$ **to** d
3. $\quad h_j(i) = (a_j \times i + b_j)\ mod\, w$
4. $\quad e = \min(e, C[j, h_j(i)])$
5. **return** e

References

- *Count-Min Sketch and its Applications* https://sites.google.com/site/countminsketch
- *Count-Min Sketch on Wikipedia* http://en.wikipedia.org/wiki/Count-Min_sketch
- *Sketching Data Structures* http://lkozma.net/blog/sketching-data-structures
- *Approximating Data with the Count-Min Data Structure* http://dimacs.rutgers.edu/ graham/pubs/papers/cmsoft.pdf
- *An improved Data Stream Summary: The Count-Min Sketch and its Applications* http://www.research.att.com/people/Cormode_Graham/library/publications/CormodeMuthu