Algorithmic Design / Algorithmic Data Mining

Data Mining - Lecture 5

Tommaso Padoan <tommaso.padoan@units.it>

Probabilistic data structures

Probabilistic data structures provide efficient (in space and/or time) ways to store approximated information about some data. They are called probabilistic because the corresponding approximation error is guaranteed to be less than a given bound with some probability, usually depending on the size of the data structure and the total amount of data to be stored inside. A good probabilistic data structure trades a reasonably low error rate for a significant gain in performance and/or efficiency, thus being useful for applications to data mining and big data.

Bloom filter

A Bloom filter is a space-efficient data structure to approximate a set of data items, to be used for *membership queries*. Indeed, it does not record the actual data items, just their existence. Formally, given the universe U of all possible data items and a subset S of them, a Bloom filter for S provides an **over-approximation** B such that $S \subseteq B \subseteq U$, with some probabilistic guarantees about the approximation error $B \setminus S$.

Bloom filters use hash functions to build simple hash tables without actual items inside. In particular, a Bloom filter is an array of m bits (0 or 1 values), initially all set to 0, equipped with k hash functions $h_1, \dots, h_k : U \to \{0, \dots, m-1\}$. Whenever an item x is added to the filter, the bits indexed by $h_1(x), \dots, h_k(x)$ are set to 1 (independently of their previous value). On the other hand, when queried for the membership of some item y, it returns 1 (*true*) if **all* bits $h_1(y), \dots, h_k(y)$ are currently 1, otherwise, it returns 0 (*false*) if any of them is 0. This ensures that negative answers are always correct, while positive ones may sometime not be. In the latter case, we say that we have a *false-positive*, due to the approximation. It is worth noting that, differently from normal dynamic data structures, items cannot be removed from a Bloom filter, because of the possible collisions with other added items. Indeed, after removing an item, by setting the bits indexed by its hashings to 0, false-negatives would also become possible, compromising the guarantees of the filter.

The probability of a false-positive can be calculated, assuming that the hash functions have been chosen so that they *uniformly* distribute keys and are *pair-wise independent*. Let n be the number of items that have been added to the filter, m the size of the array B of bits, and k the number of hash functions. A false-positive is an item y such that $B[h_i(y)] = 1$ for all $1 \leq i \leq k$, but that was never added to the filter. For this to happen, each one of those bits must have been set to 1 by some hash function after the addition of one of the n items. With the afore mentioned assumptions, the probability for a bit to be set to 1 by a hash fuction after the addition of an item is $1/m$. After the addition of n items using k hash functions for every addition, the probability of a bit still being 0 is $(1 - \frac{1}{m})^{nk}$, thus a bit will be 1 with probability $1 - (1 - \frac{1}{m})^{nk}$. Therefore, the probability of a false-positive y is $(1 - (1 - \frac{1}{m})^{nk})^k$ since it requires all bits indexed by $h_1(y), \ldots, h_k(y)$ to be 1.

Note that $(1 - \frac{1}{m})^{nk}$ increases for larger m and decreases for larger n and k, but $(1-p)^k$ decreases for larger k. So, the probability of a false-positive will increase for larger load factor $\alpha = n/m$ and for too small or too large k. Rewriting the probability in terms of e, we obtain:

$$\left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^k = \left(1 - e^{\ln(1-\frac{1}{m})^{nk}}\right)^k = \left(1 - e^{nk\ln(1-\frac{1}{m})}\right)^k \approx$$
$$\approx \left(1 - e^{-\frac{nk}{m}}\right)^k = \left(1 - e^{-\alpha k}\right)^k = \left(1 - \frac{1}{e^{\alpha k}}\right)^k$$

where we approximated $\ln(1 - \frac{1}{m})$ to $-1/m$ using the Mercator series, since $0 < 1/m < 1$. For a fixed load factor α, we can find the optimal k, the one minimising the false-positive probability, where the derivative is 0, that is, when $k = \alpha^{-1} \ln 2 = \frac{m}{n} \ln 2$, or rather $\left\lceil \frac{m}{n} \ln 2 \right\rceil$ since it must be integer. The table below shows optimal values of k and corresponding false-positive probabilities for Bloom filters of different sizes m used for approximating a set of $n = 1000$ items.

m	500	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
k^*	1	1	2	3	3	4	5	5	6	7	7
p[f.p.]	0.865	0.632	0.400	0.253	0.147	0.092	0.058	0.035	0.022	0.013	0.008

When working with sets, union and intersection are common operations. It is worth seeing how they interact with Bloom filters. Let B_1 and B_2 be two Bloom filters with same size m and hash functions, obtained through the addition of the elements of sets S_1 and S_2, respectively. The Bloom filter B_U that would be produced by adding every element of the union $S_1 \cup S_2$ of the two sets, can be obtained by simply taking the bit-wise *OR*/*max* of B_1 and B_2, that is, $B_U[i] = B_1[i] \vee B_2[i]$ for all $0 \leq i < m$. This works because size and hash functions are the same, and every time an item is added to the Bloom filter every bit is actually set to the *OR* of what was already, and 1 or 0 depending on the hashing. On the other hand, the same cannot be done for intersection, at least not without losing accuracy. Indeed, the

Bloom filter obtained by computing the bit-wise *AND/min* $B_I[i] = B_1[i] \wedge B_2[i]$ for all $0 \leq i < m$, is not the same as the one produced by adding every element in the intersection $S_1 \cap S_2$ of the two sets to a new filter. B_I will still provide a correct over-approximation of the intersection of the two sets, that is, $S_1 \cap S_2 \subseteq B_I \subseteq B_1, B_2$. In fact, B_I will contain at least as many bits set to 1 as the actual Bloom filter produced by the intersection $S_1 \cap S_2$, plus those obtained from 1 -bits in B_1 and B_2 not from actual shared items. Therefore, the false-positive probability will also be larger than that of the filter built directly from the intersection.

Count-min sketch

When we are interested in the *frequency* of data items in a stream of data, abstractly a sequence or multi-set, we need to count the occurrences of every data item added to the data structure. Count-min sketches extend the ideas of Bloom filters to record counters instead, again in an approximate way, as well as providing guaranties on the possible error of approximation. Moreover, similarly to Bloom filters, the approximation is always an over-approximation. Formally, given the universe U of all possible data items and a stream (sequence or multi-set) S of those items, a count-min sketch for S provides, for every item $x \in U$, a value that is larger or equal than the number f_x of its occurrences in S.

A count-min sketch is a matrix (2-dimensional array) C of size $d \times w$ of natural numbered counters, initially all set to 0, equipped with d hash functions $h_0, \ldots, h_{d-1} : U \rightarrow \{0, \ldots, w-1\}$. Whenever an item x is added to the sketch, the counters $C[0, h_0(x)], \ldots, C[d-1, h_{d-1}(x)]$ indexed by the hashings of the item are all increased by 1. On the other hand, when queried for the occurrencies of some item y, it returns the **minimum** of the counters $C[0, h_0(y)], \ldots, C[d-1, h_{d-1}(y)]$. Such a value is always larger or equal than the actual number f_y of occurrences of y among all the added items, that is $f_y \leq \min\limits_{0 \leq i < d} C[i, h_i(y)]$.

Assuming as usual that the adopted hash functions are uniform and independent, the error on the estimation can be shown to be bounded by a value ϵn, proportional to the number n of added items, with some probability, depending on the size d, w of the sketch. First, when considering only a generic row i of the sketch, the error of estimation on the number of occurrences of a generic item x is the difference $C[i, h_i(x)] - f_x$. Note that, if larger than 0, such an error must be due to collisions of other added items with same hashing of x. The expected value of the error is thus $\sum_{y \neq x} f_y / w \leq n/w$, since n is the total number of occurrences of any added item (including x) and a collision happens with probability $1/w$. Then, by Markov's inequality, the probability of the estimation error being larger than ϵn in a row of the sketch is at most its expected value divided by the bound, which is at most

$\frac{n/w}{\epsilon n} = \frac{1}{\epsilon w}$. Extending the analysis to the whole sketch, since the actual estimation is the minimum of the counters of every row, for the actual estimation error to be larger than ϵn, all must be larger, thus the corresponding probability will be at most $\left(\frac{1}{\epsilon w}\right)^d$. The latter can be used to find the size of the sketch when we want such probability to be at most some desired value δ. In particular, this happens if we take $w = 2/\epsilon$ and $d = \log_2 \delta^{-1}$, indeed

$$\left(\frac{1}{\epsilon w}\right)^d = \left(1/\left(\frac{2\epsilon}{\epsilon}\right)\right)^{\log_2 \delta^{-1}} = \left(\frac{1}{2}\right)^{\log_2 \delta^{-1}} = \frac{1}{2^{\log_2 \delta^{-1}}} = \frac{1}{\delta^{-1}} = \delta$$

Given the previous upper-bound δ, we deduce that the probability of the opposite event, that is, the estimation error being less than ϵn, is at least $1-\delta$. So, after the addition of n items to a count-min sketch, we have that
$f_x \leq \min\limits_{0\leq i<d} C[i, h_i(x)] \leq f_x + \epsilon n$ with probability $1-\delta$, for the desired ϵ and δ, if the size of the sketch is chosen so that $w = \lceil 2/\epsilon \rceil$ and $d = \left\lceil \log_2 \frac{1}{\delta} \right\rceil$, which means that asymptotically the sketch must contain $\Theta(\epsilon^{-1} \log_2 \delta^{-1})$ counters. The table below shows some examples of those values.

ϵ	δ	$1-\delta$	d	w	$d \cdot w$
0.1	0.1	0.9	4	20	80
0.01	0.01	0.99	7	200	1400
0.001	0.001	0.999	10	2000	20000
0.0001	0.0001	0.9999	14	20000	280000
0.00001	0.00001	0.99999	17	200000	3400000

For the same reasons of Bloom filters, items cannot be removed/subtracted from count-min sketches. Moreover, also (multi-set) union and intersection produce similar results when applied to count-min sketches. Given two multi-sets and corresponding count-min sketches, with same size and hash functions, their union (taking pairwise *max* of corresponding counters) will produce a count-min sketch that still provides a correct over-approximation of the occurrences of items in the union of the multi-sets. Furthermore, such a sketch will have the same (or better) accuracy of the one obtained by adding every item in the union of the multi-sets to a new empty sketch, despite possibly not resulting exactly the same. Therefore, even though the result is not actually the same, union can be safely applied to count-min sketches. On the other hand, like Bloom filters, the intersection of two count-min sketches (taking pairwise *min* of corresponding counters) will also produce a correct over-approximation of the occurrences of items in the intersection of the corresponding multi-sets, but with less accuracy than the one obtained by adding those items to a new empty sketch. Thus, intersection cannot be safely applied to count-min sketches. Another common operation when working with data streams, is *concatenation*. Interestingly, count-min sketches, with same size and hash functions, can be combined, preserving all guarantees and accuracy, to obtain the exact sketch for the concatenation of their data streams, by simply pairwise *adding* the corresponding counters together.