18.415 Project - Using a Count-Min sketch structure for easier analysis of Bloom filter false positive rates

Yun William Yu

December 11, 2013

Abstract

The classical proof for Bloom filter false positive rates was shown to be incorrect due to a subtle error regarding independence of events. Although the previously computed false positive rate is asymptotically still correct, it is incorrect for small parameter values and is generally only a lower bound [BGK$^+$08]. Indeed, the correct analysis for Bloom filters does not admit a convenient closed form, though efficient computations do exist [CRJ10]. In this paper, we outline the strategy of the new analysis

Furthermore, we demonstrate that for large parameter values, we can use a Count-Min [CM05] data structure to asymptotically recover a classical Bloom filter false positive rate using only the naive analysis scheme. This has no practical applications, as the Bloom filter is strictly better in every sense, but this construction may be useful pedagogically in demonstrating the subtleties involved when working with independence. Lastly, exact expected false positive rates are trivially computable for this structure; though the rates are worse than for regular Bloom filters, this might somehow be useful theoretically.

1 Bloom Filters

Bloom filters are a space efficient probabilistic data structure developed in 1970 for testing set membership [Blo70]. We start off with an empty bit array B, and use independent random hash functions to map items to multiple locations in B. Given our set of interest, we set all of the locations mapped to by items in our set to 1. Then, to test an item for membership in our set, we simply check to see if all the random locations that item maps to in B have been set to 1. Obviously, if the item is in our set, its locations will have been set to 1. However, we also have a chance of false positives because even for items not in our set, there's some chance that its locations will have been set to 1 by chance from the insertion of other items (see figure 1).

More precisely, let Ω be the universe of possible set elements, and let $\Sigma \subset \Omega$ be our set of interest, with $n = |\Sigma|$. Let B be a bit array of size m, and suppose we have k independent uniformly random hash functions $f_1, \ldots, f_k : \Omega \to [m]$. Then, for each item $x \in \Sigma$ set $\forall i \in [k], B(f_i(x)) = 1$. In order to test for set membership, we return "probably in set" if $\forall i \in [k], B(f_i(x)) = 1$. We will denote the false positive rate in what follows by δ.

1.1 Classical Analysis

Let's naively analyze the false positive rate as a function of parameters k, m, n.

	1	2	3	4	5	6	7	8	9	
$x_1 \in \Sigma$			✓				✓			
$x_2 \in \Sigma$			✓						✓	
B	0	0	1	0	0	0	1	0	1	
$y_1 \notin \Sigma$	×						×			TN
$y_2 \notin \Sigma$							×		×	FP

Figure 1: An example Bloom filter B with parameters $k = 2$, $m = 9$, and $n = 2$. Each of the positions corresponding to $x_1, x_2 \in \Sigma$ have their bits set to 1. For most $y_1 \notin \Sigma$, not all of their hash locations will be set, so we'll have a true negative. However, sometimes, we'll get a false positive $y_2 \notin \Sigma$ because all of their hash locations are set to 1 by chance.

Claim 1. *A Bloom filter of length m bits using k hash functions and with n items inserted has a false positive rate*

$$\delta = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k}$$

Proof. By uniformness, the probability that a particular bit $B[j]$ in B is set by the hash function f_i applied to $x \in \Sigma$ is $\frac{1}{m}$. In order for $B[j] = 0$, none of the hash functions can map any of the elements of Σ to y. Since we've assumed that the hash functions are completely independent (this assumption is much stronger than is needed in practice, but makes the analysis easier),

$$\begin{aligned} \mathbb{P}\left(B[j] = 0\right) &= \left(1 - \frac{1}{m}\right)^{kn}, \\ \implies \mathbb{P}\left(B[j] = 1\right) &= 1 - \left(1 - \frac{1}{m}\right)^{kn}. \end{aligned}$$

The false positive rate is exactly the probability that for $x \notin \Sigma$, $B(f_1(x)) = B(f_2(x)) = \cdots = B(f_k(x)) = 1$. Since $f_1, \ldots, f_k$ are independent,

$$\begin{aligned} \delta &= \mathbb{P}(B(f_1(x)) = 1, \ldots, B(f_k(x)) = 1) && (1) \\ &= \mathbb{P}(B(f_1(x)) = 1) \cdots \mathbb{P}(B(f_k(x)) = 1) && (2) \\ &= \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k}. && (3) \end{aligned}$$

□

As an aside, the classical analysis makes a subtle dependence error going from equation 1 to equation 2. We will devote the rest of this paper to fully exploring, characterizing, and working around this error.

Turning things around, let's consider the required number of bits m to store n items in the Bloom filter with error δ in the asymptotic limit as $n \to \infty$.

Claim 2. *In the limit as $n \to \infty$, a Bloom filter storing n items needs $m \approx 1.44 n \log_2 \delta$ bits in order to maintain a false positive rate δ.*

Proof. Obviously, from equation 3, the false positive rate increases with n unless m is also scaled up commensurately. Let $m = cn$, where c is a constant. Then

$$\begin{aligned}\lim_{n\to\infty} \delta &= \lim_{n\to\infty} \left(1 - \left(1 - \frac{1}{cn}\right)^{kn}\right)^k \\ &= \left(1 - \lim_{n\to\infty} \left(1 - \frac{1}{cn}\right)^{kn}\right)^k \\ &= \left(1 - e^{-\frac{k}{c}}\right)^k = \left(1 - e^{-\frac{kn}{m}}\right)^k \qquad (4)\end{aligned}$$

Furthermore, holding n and m constant, we can minimize δ with respect to k. Setting the partial derivative to 0,

$$0 = \frac{\partial \delta}{\partial k} = \left(1 - e^{-\frac{kn}{m}}\right)^k \left[\log\left(1 - e^{-\frac{kn}{m}}\right) + \frac{k \cdot \frac{n}{m} \cdot e^{-\frac{kn}{m}}}{1 - e^{-\frac{kn}{m}}}\right],$$

and with a little work we can recover the extrema $k = 0$ and $k = \frac{m}{n} \log 2$. Although $k = 0$ does indeed globally minimize the false positive rate, it also increases the false negative rate to 100%, and so is rather undesirable as a solution. Thus, $k = \frac{m}{n} \log 2$.

Substituting back into equation 4,

$$\delta = \left(\frac{1}{2}\right)^{\frac{m}{n} \log 2} \implies m = \frac{1}{\log 2} n \log_2 \delta \approx 1.44 n \log_2 \delta.$$

$\square$

Thus, asymptotically, to maintain a δ false positive rate, we need $m = 1.44n \log_2 \delta$ bits using the classical analysis.

1.2 Non-independence error

Unfortunately, there was a subtle error in the classical analysis presented above [BGK+08]. Going from equation 1 to 2 required the assumption that the events $B(f_1(x)) = 1, \ldots, B(f_k(x)) = 1$ are independent. This is *not* the same as independence of the hash functions $f_1, \ldots f_k$, which is what we're given. (As a historical aside, Bloom himself did not make this error, but instead simply started his analysis from the fraction of set bits, without worrying about how to compute the fraction of set bits.)

To see this in an easy example, we fully flesh out an example given in [BGK+08]. Consider a trivial Bloom filter with $k = 2$, $m = 2$, and $n = 1$, for which it's possible to explicitly list out all 16 possibilities resulting from random choice of hash functions in table 1.

Note that since each of the scenarios the table are equally likely given independent uniform random hash functions, we have a false positive rate of $\frac{10}{16}$. However, equation 3 gives $\delta = \frac{9}{16}$. The reason for this discrepancy comes from the fact that although the hash functions themselves are independent, the chances that the hash functions both hit set bits in B is not. In table 1 this phenomenon manifests in that when both bits of B are set, no matter where the hash functions map x_2, there will be a false positive. Thus, the fact that $f_1(x_2) = 1$ tells you that it's more likely than not that $f_2(x_2) = 1$. Specifically, $\mathbb{P}(f_2(x_2) = 1) = \frac{3}{4}$, but $\mathbb{P}(f_2(x_2) = 1 | f_1(x_2) = 1) = \frac{5}{6}$, so the two events are not independent.

$f_1(x_1)$	$f_2(x_1)$	$B(a)$	$B(b)$	$f_1(x_2)$	$f_2(x_2)$	$B(f_1(x_2))$	$B(f_2(x_2))$	FP
a	a	1	0	a	a	1	1	✓
a	a	1	0	a	b	1	0	
a	a	1	0	b	a	0	1	
a	a	1	0	b	b	0	0	
a	b	1	1	a	a	1	1	✓
a	b	1	1	a	b	1	1	✓
a	b	1	1	b	a	1	1	✓
a	b	1	1	b	b	1	1	✓
b	a	1	1	a	a	1	1	✓
b	a	1	1	a	b	1	1	✓
b	a	1	1	b	a	1	1	✓
b	a	1	1	b	b	1	1	✓
b	b	0	1	a	a	0	0	
b	b	0	1	a	b	0	1	
b	b	0	1	b	a	1	0	
b	b	0	1	b	b	1	1	✓

Table 1: The false positives for a Bloom filter with parameters $k = 2$, $m = 2$, and $n = 1$, where we've labelled the two bit positions in B by a, b. Say $x_1 \in \Sigma$ and $x_2 \notin \Sigma$ and we have hash functions f_1, f_2.

Generically, the events $B(f_1(x)) = 1, \ldots, B(f_k(x)) = 1$ are "positively correlated", so equations 1 and 2 need to be modified to

$$\delta = \mathbb{P}(B(f_1(x)) = 1, \ldots, B(f_k(x)) = 1) \geq \mathbb{P}(B(f_1(x)) = 1) \cdots \mathbb{P}(B(f_1(x)) = 1).$$

We will make this notion more precise in the next section, but roughly speaking, all the results in the previous section are therefore only lower bounds.

1.3 Modern analysis

Although the easy way out is simply to do what Bloom did and start the analysis from the fraction of bits set, it's much more useful to have bounds in terms of the number of items inserted into the filter. As such, a more modern analysis was presented in [BGK$^+$08], resolving this issue and explicitly computing the fraction of set bits.

For this analysis, we will first do the easy step of computing the false positive probability given some pattern of set bits in B. Let A be the false positive event that for $x \notin \Sigma$, $B(f_1(x)) = \cdots = B(f_k(x)) = 1$, so $\mathbb{P}(A) = \delta$, the false positive probability.

Lemma 1. *Let $I \subset [m]$, and let E_I be the event that I is exactly all the set bits in B. Then $\mathbb{P}(A|E_I) = \left(\frac{|I|}{m}\right)^k$.*

Proof. A false positive happens when all of the hash functions accidentally hit a set bit. The probability that an individual hash function accidentally hits a set bit is just the fraction of set bits in B, given by $\frac{|I|}{m}$. Since each of the k hash functions is independent, the probabilities multiply, proving this lemma. $\square$

Then, by simply breaking up the probability into components conditioned on the pattern of set bits, we get

$$\mathbb{P}(A) = \sum_{I \subset [m]} \mathbb{P}(A|E_I) \cdot \mathbb{P}(E_I). \tag{5}$$

Thus it remains only to compute $\mathbb{P}(E_I)$.

Lemma 2.

$$\mathbb{P}(E_I) = \frac{1}{m^{kn}} \sum_{j=0}^{l} (-1)^j \binom{l}{j} j^s$$

Proof. Since we do not want to repeat the mistake in the classical analysis, we cannot consider each of the hash functions separately. Instead, we have to count the number of collections of k hash functions $h_i : [n] \to [m]$ that have image exactly I when applied to Σ. Since our hash functions are uniformly random, this is equivalent to counting all surjections from a set of size kn to a set of size $|I|$. From any basic enumerative combinatorics text (e.g. Stanley's *Enumerative Combinatorics*, section 1.9 [Sta11]), recall that the number of ways to partition a set of size s to l unlabeled non-empty subsets is given by the Stirling number of the second kind

$$S(s,l) = \frac{1}{l!} \sum_{j=0}^{l} (-1)^j \binom{l}{j} j^s.$$

However, counting surjections to positions $I \subset B$ carries labels since B itself is labeled. Thus, counting the number of surjections requires us to multiply the Stirling number of the second kind by the number of permutations of the $|I|$ set bits, giving

$$(l)! \cdot S(kn, |I|) = \sum_{j=0}^{l} (-1)^j \binom{l}{j} j^s$$

distinct surjections from a set of size kn to a set of size $|I|$.

Then, we'll need to divide that by the number of all collections of k hash functions $h_i : [n] \to [m]$. This is easy to compute because each hash function randomly returns a location for each element in $[n]$. Thus, there are m^n such hash functions, so the number of all collections of k (possibly non-distinct) hash functions is then simply m^{kn}.

Thus, we get

$$\mathbb{P}(E_I) = \frac{1}{m^{kn}} \sum_{j=0}^{l} (-1)^j \binom{l}{j} j^s.$$

$\square$

Putting all the lemmas together into equation 5, we get

$$\begin{aligned}\delta = \mathbb{P}(A) &= \sum_{I\subset[m]} \left(\frac{|I|}{m}\right)^k \cdot \sum_{j=0}^{|I|}(-1)^j \binom{|I|}{j} j^{kn} \cdot \frac{1}{m^{kn}} \\ &= \sum_{l=1}^{m} \binom{m}{l} \left(\frac{l}{m}\right)^k \cdot \sum_{j=0}^{l}(-1)^j \binom{l}{j} j^{kn} \cdot \frac{1}{m^{kn}} \\ &= \frac{1}{m^{k(n+1)}} \sum_{l=1}^{m} l^k \binom{m}{l} \sum_{j=0}^{l}(-1)^j \binom{l}{j} j^{kn}. \end{aligned} \tag{6}$$

Unfortunately, this cannot be simplified into a simple closed form solution. Furthermore, for arbitrary parameter values, this does not even admit useful upper-bounds. However, [BGK+08] were able to prove

$$p^k < \delta \leq p^k \times \left(1 + O\left(\frac{k}{p}\sqrt{\frac{\log m - k\log p}{m}}\right)\right)$$

for $k \geq 2$, $\frac{k}{p}\sqrt{\frac{\log m - 2k\log p}{m}} \leq c$, $c < 1$, and $p = 1 - \left(1 - \frac{1}{m}\right)^{kn}$. Thus, for large m, $\delta \approx O(p^k)$, which asymptotically matches the bounds we got using the (incorrect) classical analysis.

Furthermore, for smaller parameter values, it can also be necessary to directly compute δ, despite the lack of a closed form solution. [CRJ10] give both iterative and recursive algorithms for numerically computing δ in $O(n)$ time.

Unfortunately, the sorts of results for δ in this modern analysis are not particularly amenable for getting an intuitive understanding of the false positive rate. These difficulties arise from the complicated dependence of events expressed above, and are an intrinsic component of Bloom filters as defined. However, as we'll see later, by modifying a Count-Min sketch, although we lose some on the false positive rate for small parameters, the equivalent classical analysis is correct, making for much more intuitive proofs.

2 Count-Min Bloom filter

[1] Cormode and Muthukrishnan were interested in a more general class of problems than set membership when they developed the Count-Min sketch [CM05]. In particular, the general turnstile problem allows for error on both sides in a generalization of the approximate counting problem. Luckily, the Count-Min sketch only produces noise on the right. Thinking of the approximate set membership problem as approximate counting, we're interested in all items with counts of at least 1, so noise on the right side of the integer line does not matter to us. Naively one might directly drop in a Count-Min sketch for a Bloom filter, but this wastes space because we don't care about keeping track of exact counts. However, the general structure still works, and we can then directly apply the analogue of the classical Bloom filter analysis, this time correctly.

Recalling the setup for a Bloom filter, we do the same thing, but separate the range of each of the hash functions from each other. In Bloom filters, the hash functions all range over the entire bit

[1] In response to reviewer comments, we have completely excised the former section 2, which was an introduction to the Count-Min data structure. Relevant features have been moved into this section instead.

$$B = \begin{array}{|c|c|c|c|c|c|c|c|}
\hline
 & & B_{1,f_1(x)} & & & & & \\
\hline
 & & & & & & B_{2,f_2(x)} & \\
\hline
 & & & B_{3,f_3(x)} & & & & \\
\hline
B_{4,f_4(x)} & & & & & & & \\
\hline
 & & & & & & B_{5,f_5(x)} & \\
\hline
 & & & B_{5,f_5(x)} & & & & \\
\hline
\end{array}$$

Figure 2: An example Count-Min Bloom filter B with parameters $k = 6$, $m = 48$, and $n = 1$. The setup is exactly like the Bloom filter, except each item x maps to one random location in each of the k rows, rather than to k arbitrary locations.

vector; here, we partition the bit-vector into rows corresponding to the range of each hash function, which will give us the independence the classical analysis incorrectly assumed.

Let Ω be the universe of possible set elements, and let $\Sigma \subset \Omega$ be our set of interest, with $n = |\Sigma|$. Create a $k \times w$ bit-table B, or equivalently a bit array of size $m = kw$. Suppose we have k independently uniform random hash functions $f_1, \ldots, f_k : \Omega \to [w]$. Then for each item $x \in \Sigma$, set $\forall i \in [k], B_{i,f_i(x)} = 1$. In order to test for set membership, we again simply check if all those locations are set.

In some ways, this is slightly less space-efficient than a regular Bloom filter, because the hash functions only range over a row, rather than the entire bit-table. However, this restriction does not hurt us asymptotically, and as we'll see, remedies the problem of dependence we ran into earlier in our classical analysis.

Again, let's analyze the false positive rate as a function of parameters k, m, n. By uniformness, the probability that a particular bit $B_{i,y}$ in B is set by the hash function f_i applied to $x \in \Sigma$ is $\frac{1}{w}$. Since only f_i maps to row i, we only need to worry about where f_i maps all the elements of Σ. Thus, in order for $B_{i,y} = 0$, we only need that none of the elements of Σ maps to y under f_i. Making use of our randomness assumption,

$$\begin{aligned}
\mathbb{P}\left(B(y) = 0\right) &= \left(1 - \frac{1}{w}\right)^n, \\
\implies \mathbb{P}\left(B(y) = 1\right) &= 1 - \left(1 - \frac{k}{m}\right)^n.
\end{aligned}$$

The false positive rate is exactly the probability that for $x \notin \Sigma$, $B_{1,f_1(x)} = \cdots = B_{k,f_k(x)} = 1$. Since not only $f_1, \ldots, f_k$ are independent, but those events are completely partitioned from each other (no hash function can set bits in any other row, so there's no information leakage) and therefore independent as well, this time we can actually say that the false positive probability

$$\begin{aligned}
\delta &= \mathbb{P}(B_{1,f_1(x)} = 1, \ldots, B_{k,f_k(x)} = 1) \\
&= \mathbb{P}(B_{1,f_1(x)} = 1) \cdots \mathbb{P}(B_{k,f_k(x)} = 1) \\
&= \left(1 - \left(1 - \frac{k}{m}\right)^n\right)^k \qquad (7)
\end{aligned}$$

This false positive rate is worse than the one given in equation 3 because there's a k term in the numerator over m, rather than in the exponent by n. However, as you'll recall from the limit definition of the exponential function, this does not change anything asymptotically.

Carrying on, by repeating the rest of the classical analysis of Bloom filters, we get for $m = cn$

$$\lim_{n\to\infty} \delta = \left(1 - e^{-\frac{kn}{m}}\right)^k.$$

Again, $k = \frac{m}{n}\log 2$ minimizes δ, and by substituting back in,

$$m = \frac{1}{\log 2} n \log_2 \delta \approx 1.44 n \log_2 \delta.$$

Thus, asymptotically, these "Count-Min Bloom filters" achieve the same space bounds as regular Bloom filters while allowing for much easier analysis.

3 Conclusion

For basically no practical purposes are Count-Min Bloom filters going to replace regular Bloom filters. Although they achieve the same space-efficiency asymptotically, for small parameter values a regular Bloom filter will have a lower false positive rate than the equivalent Count-Min Bloom filter. Indeed, setting parameters to $k = 2, m = 2, n = 1$ to match the toy example exhibiting the non-independence error, the Count-Min Bloom filter will have a false positive rate $\delta = 1$ (but which is correctly predicted by the classical analysis). Furthermore, for a Count-Min Bloom filter as defined, k must divide m. This assumption can be relaxed, but we lose the ease of analysis and might as well use a regular Bloom filter.

However, although the false positive rates are higher, they are also much easier to compute. Whereas correctly estimating false positive rate for a regular Bloom filter requires the complicated surjection counting formula in equation 6, it is trivial to plug parameters into equation 7 for the Count-Min Bloom filter. For most applications, we only care about minimizing the false positive rate. In a hypothetical setting where it is more important to know the expected false positive rate than to minimize it, the Count-Min Bloom filter could be useful.

More importantly though, the analyses in this paper show the dangers that can arise from subtle dependencies between events. Just because we start off with independent hash functions doesn't mean that that independence carries through. This is a both a very important pedagogical point to drive home, and is critical when it comes to actual implementations and we do not have full independence, but rather only k-independence among hash functions, or sometimes not even provable independence, but just some heuristic measure of independence.

Furthermore, this paper demonstrated one way to get around those subtle dependencies when they arise, which is to more carefully separate each of the sources of randomness so that they don't mix in unexpected ways. This will usually come at some cost, in this case to false positive rates at low parameter values, sometimes even to the constant at the end of the analysis. By doing this though, it is often possible to achieve the same asymptotic goals in a rigorous manner while using more naive mathematical machinery. This sort of approach may also be useful pedagogically to prevent the need for "hand-waving" and unjustified (though perhaps accurate) approximations, as was used in the classical analysis of Bloom filters.

Lastly, the authors suggest that when results at low parameter values are of paramount importance, it might be of practical use to relax the separation between sources of randomness. Although it might then become exceedingly difficult to prove good error bounds, empirical bounds are often more useful in implementations anyway. For future work, it would be interesting to examine the relative error rates of the full Count-Min sketch with that of counting Bloom filters at low parameter ranges, as that is the direct generalization of the topics covered in this paper.

References

[BGK$^+$08] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid, and Yihui Tang, *On the false-positive rate of bloom filters*, Information Processing Letters **108** (2008), no. 4, 210–213.

[Blo70] Burton H Bloom, *Space/time trade-offs in hash coding with allowable errors*, Communications of the ACM **13** (1970), no. 7, 422–426.

[CM05] Graham Cormode and S Muthukrishnan, *An improved data stream summary: the count-min sketch and its applications*, Journal of Algorithms **55** (2005), no. 1, 58–75.

[CRJ10] Ken Christensen, Allen Roginsky, and Miguel Jimeno, *A new analysis of the false positive rate of a bloom filter*, Information Processing Letters **110** (2010), no. 21, 944–949.

[Sta11] Richard P Stanley, *Enumerative combinatorics*, vol. 49, Cambridge university press, 2011.