Lecture 8: Bloom Filters

Prof. Eric Price *Scribe: Aaryan Prakash, Mark Wen*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Bloom Filters

A *bloom filter* is a randomized datastructure to represent a set. We want to be able to insert elements into a set and query if the element exists in the set. Using a hash table, we require $O(1)$ time per operation and $O(n)$ words of space. If our elements come from a set of size U, we need to store $\log U$ bits per element, so the space complexity is actually $O(n \log U)$.

For a deterministic algorithm, we cannot get a better bound. There are $\binom{U}{n}$ possible sets. We can bound the log of the number of sets by

$$\log\left(\frac{U}{n}\right)^n \leq \log\binom{U}{n} \leq \log\left(\frac{eU}{n}\right)^n,$$

which are both $\Theta(n \log U)$ if $u \gg n^2$. If we have less than that many bits, we have to get the wrong answer sometimes.

To build a randomized algorithm, we introduce some probability of failure. Specifically, if $x \in S$, then we always return 1. However, if $x \notin S$, we return 0 with probability $\geq 1 - \delta$. This allows us to build a data structure with space complexity $O\big(n \log \frac{1}{\delta}\big)$.

Bloom filters are useful because we can first run queries on RAM, and then verify on disk if we get a 1. Another example is Chrome checking for malicious websites. A bloom filter of malicious websites is locally stored, and if we get a positive, a request was sent to Google servers to verify that it is a true positive.

1.1 Data Structure Description

We pick hash functions $h_1, \ldots, h_k$ uniformly at random from the set of hash functions. To insert an element, we just set 1 to all of the k hash outputs. To query, we check if each of the k hash outputs are set and return 1 if every bit is set.

Because we want everything to fit on disk, we want to find a k that makes m small. Given n, m, k, we want to find the false positive rate and then optimize from there.

Let Y_j denote if the jth bit is set. For all locations j,

$$\begin{aligned}
\mathbb{P}[Y_j = 0] &= \mathbb{P}[\text{none of } h_\ell(x_i) = j] \\
&= \mathbb{P}[h_1(x_1) \neq j]^{nk} \\
&= \left(1 - \frac{1}{m}\right)^{nk} \\
&= e^{-nk/m} + \widetilde{O}(nk/m^2) \\
&\approx e^{-nk/m}.
\end{aligned}$$

Let $Y = \sum_{j=1}^m \mathbf{1}_{Y_j=1}$. The expected value of this is $m\big(1 - e^{-nk/m}\big)$. If we could apply the Chernoff bound, we would get that $Y = \mathbb{E}[Y] \pm \sqrt{m + \log(1/\delta)}$, which tells us that $Y = \big(1 - e^{-nk/m}m + o(m)\big)$. The issue is that the different Y_is are not independent. If some Y_i is 1, then Y_j is less likely to be 1. Thus, we cannot use the Chernoff bound.

1.2 Negative Association

Definition 1. *A set of random variables $X_1, \dots X_n \in \mathbb{R}$ is* negatively associated *(N.A.) if for all non-decreasing functions $f(X_I)$ and $g(X_{\overline{I}})$, where I is some index set, then*

$$\mathbb{E}[f(X_I)g(X_{\overline{I}})] \le \mathbb{E}[f(X_I)]\mathbb{E}[g(X_{\overline{I}})].$$

If $X_1, \dots, X_n$ are N.A., then the sum satisfies the standard Chernoff bounds. To prove the Chernoff bound, we have $\mathbb{E}[e^{\lambda \sum X_i}] = \prod \mathbb{E}[e^{\lambda X_i}]$ for some $\lambda > 0$. This changes to an inequality when the variables are N.A.

Proposition 2. *If $X_1, \dots, X_n \in \{0, 1\}$ and the sum is always 1, then X is N.A.*

Proof. WLOG assume $f(0) = g(0) = 0$. Therefore, $f(X_I)g(X_{\overline{I}}) = 0$ always since one of the sets will be all zeros. □

Proposition 3. *Monotonic functions of disjoint sets of N.A. variables are N.A. themselves.*

Proposition 4. *If X and Y are independent and separately N.A., then (X, Y) is jointly N.A.*

Suppose we throw n balls into $[m]$. $Z_{ij} = 1$ if ball i went to bin j. We define $Y_j = \max_i Z_{ij}$. $Z_{1,j}$ is N.A., since it only equals 1 for one value of j. This is true for all i, so $\{Z_{ij}\}$ is N.A. all together. Since Y is computed by a monotonic function on disjoint subsets of N.A. random variables, Y is N.A. as well, and thus using the Chernoff bound was actually valid.

1.3 Performance Analysis

Let $z = nk/m$. Suppose $x \notin S$ We know that the probability the query returns 1 is $(1 - e^{-z})^k$. To minimize this probability, we want the minimum over all values of k. This is equivalent to minimizing $(1 - e^{-z})^{zm/n}$, which we can show is minimized at $z = \ln 2$. Therefore, we want to pick $k = \frac{m}{n} \ln 2$.

Plugging this into the probability, we get a failure rate of $2^{-\frac{m}{n}\ln 2} \approx 0.618^{m/n}$. For example, if $m = 8n$ and $k = 6 \approx 8\ln 2$, then we get a failure probability of 2%. If $m = 9.6n$ and $k = 7$, we get a failure probability of 1%. Increasing it up to $m = 20n$ and $k = 14$, we get 7×10^{-5} error. Compared to hash tables, we can have much greater utilization. If we have $4n$ words and 64 bits per word, then we use $256n$ bits in total.

2 Counting Bloom Filter

To support deletion, Y_i needs to be the number of elements that hash to bucket i. Inserting increments the count, and deletion decrements the count. To query, we check if all the bins are at least 1. If we store ℓ bits per bin, we are fine as long as we never overflow.

For the same $k = \frac{m}{n}\ln 2$, the probability that we overflow is $\mathbb{P}[Y_j \geq t]$ for some t. There are $\binom{nk}{t}$ hash outputs, and each of them have a $\frac{1}{m^t}$ chance of going in that bucket. This is bounded by $\left(\frac{enk}{tm}\right)^t = \left(\frac{e\ln 2}{t}\right)^t$. For $t = 16$, this probability is at most 1.4×10^{-15}, and we multiply by m to union bound the probability of any failure happening. If $m < 10^{10}$, then the probability of overflow is still at most 1.4×10^{-5} with 4 bits per counter.

3 Problems with Bloom Filters

1. We need k fully-independent random hash functions.
2. This is not cache efficient if the table doesn't fit in the cache.
3. We assumed that queries are independent of randomness, but we could get many queries that always are false positives.
4. We tried to use for sending anonymous data by sending a bloom filter of the user's data along with some noise, but this still reveals information.