Bloom Filters

Christian Wulff-Nilsen
Algorithmic Techniques for Modern Data Models
DTU

September $5, 2025$

Overview for today

- Independent random variables
- Hash functions for Bloom filters
- Problem definition
- Description of Bloom filter
- Performance
- Comparison to lower bound

Independent random variables

- Random variables $X_1, \ldots, X_n : A \to B$ are *independent* if

$$\begin{aligned} &\mathrm{P}[X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n] \\ &= \mathrm{P}[X_1 = x_1] \cdot \mathrm{P}[X_2 = x_2] \cdot \ldots \cdot \mathrm{P}[X_n = x_n] \end{aligned}$$

  for all $x_1, \ldots, x_n \in B$
- This property also holds for every subset of variables
- Example:
  - n coin tosses, $X_i = 1$ if the ith toss is heads and $X_i = 0$ otherwise
  - These random variables are independent so the probability that the first, third, and fourth toss are all heads is

$$\begin{aligned} &\mathrm{P}[X_1 = 1, X_3 = 1, X_4 = 1] \\ &= \mathrm{P}[X_1 = 1] \cdot \mathrm{P}[X_3 = 1] \cdot \mathrm{P}[X_4 = 1] \end{aligned}$$

Hash functions for Bloom filters

- A hash function is a mapping $h : U \to M$ from a universe U of size u to a set $M = \{1, \ldots, m\}$; typically, $m \ll u$
- For the analysis of Bloom filters, we need certain properties of k hash functions $h_1, \ldots, h_k$:
  - (**Uniform hashing**) Each h_i maps each element $x \in U$ to M uniformly at random:

$$\mathrm{P}[h_i(x) = j] = \frac{1}{m} \text{ for } j = 1, \ldots, m$$

  - (**Independence**) The ku random variables $h_i(x)$ for $i = 1, \ldots, k$ and $x \in U$ are independent
  - For instance, for any $x, y \in U$:

$$\mathrm{P}[h_1(x) = 2, h_2(y) = 4] = \mathrm{P}[h_1(x) = 2] \cdot \mathrm{P}[h_2(y) = 4]$$

Problem definition

- We are given a universe U of size u and a subset $X = \{x_1, \ldots, x_n\}$ of U of size n
- We need to support two types of operations:
  - Inserting an element of $U \setminus X$ into X
  - Answer a query of the form "Is $x \in X$?" for any query element $x \in U$

Bloom filter

- A Bloom filter for representing a set $X \subseteq U$ consists of:
  - A bit array M of length m with indices $1, \ldots, m$
  - k hash functions, $h_1, \ldots, h_k : U \to \{1, \ldots, m\}$
- We assume the hash functions have the properties stated earlier (uniformity, independence)

Bloom filter

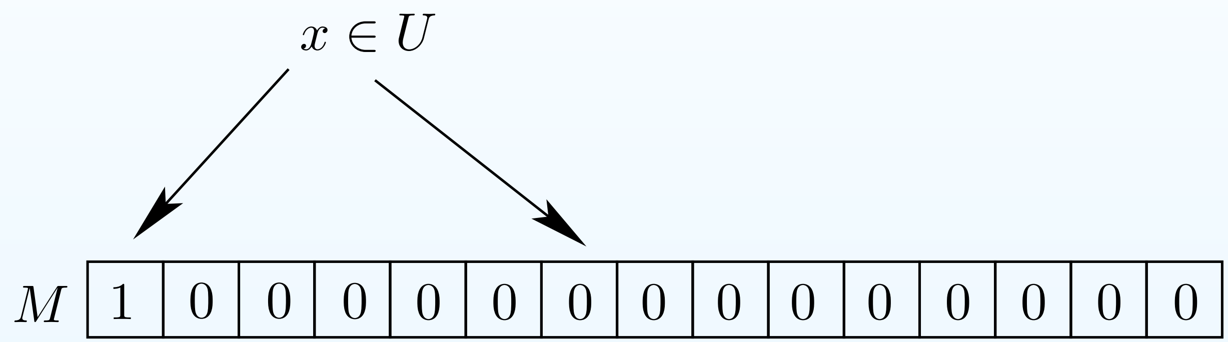
- To represent X, the bits of M are set as follows:
  - Initialize all bits to 0: $M[j] \leftarrow 0$ for $j = 1, \ldots, m$
  - For each $x \in X$ and each $i = 1, \ldots, k$, set $M[h_i(x)] \leftarrow 1$
  - Example with $X = \{x_1, x_2\}$ and $k = 2$ hash functions:
- This makes the insertion of a new element x straightforward:
  - $M[h_i(x)] \leftarrow 1$ for $i = 1, \ldots, k$
- We therefore focus on analyzing queries

Answering a query

- Recall that the Bloom filter should answer queries of the form "Is $x \in X$?" for any $x \in U$
- This is done as follows:
  - If $M[h_i(x)] = 1$ for every $i = 1, \ldots, k$, answer "Yes"
  - Otherwise, answer "No"

Are queries answered correctly?

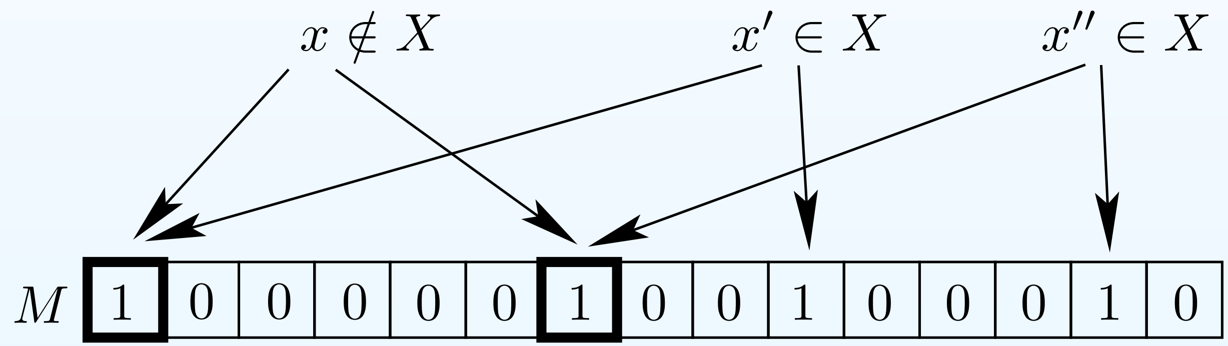
- If the Bloom filter answers “No” to a query for x, we must have $x \notin X$:



- This follows since:
  - $M[h_i(x)] = 0$ for at least one i (since the answer is “No”)
  - No element of X has this property (by construction of the Bloom filter)
- Conclusion: the Bloom filter is always correct when it answers “No”

Are queries answered correctly?

- If the Bloom filter answers “Yes”, we cannot be sure that $x \in X$:
  - We could have $x \notin X$ and all bits $h_i(x)$ of M were set to 1 by elements of X:



- Thus, the Bloom filter might be wrong when it answers “Yes”
- In summary, it can have false positives but not false negatives
- We want to bound the probability of false positives

Bounding the chance of false positives

- Suppose a ρ-fraction of the bits of M are 0; call this event $\mathcal{E}$
- Given $x \notin X$, we want to bound the probability that the Bloom filter answers “Yes” to x
- Random variables $h_1(x), \ldots, h_k(x)$ are
  - uniformly distributed in $\{1, \ldots, m\}$,
  - independent of $\mathcal{E}$ (x is not considered in the construction of M)
- This implies that for any i,

$$\mathrm{P}[M[h_i(x)] = 1] = 1 - \mathrm{P}[M[h_i(x)] = 0] = 1 - \rho$$

- Using independence of $h_1(x), \ldots, h_k(x)$, the probability that the Bloom filter incorrectly answers “Yes” for x is thus

$$\mathrm{P}[M[h_1(x)] = 1, \ldots, M[h_k(x)] = 1] = (1 - \rho)^k$$

Expressing the failure probability in terms of m, n and k

- Our goal: given m and n, pick the number k of hash functions to minimize the chance of false positives
- Let p' be the probability that a specific bit j of M is 0, i.e., that none of the k hash functions h_i map any of the n elements of X to that bit:

$$p' = \left(1 - \frac{1}{m}\right)^{kn}$$

- We have $E[\rho] = p'$ (exercise)
- Example: if each bit has a $p' = 50\%$ chance of being 0, we expect the fraction ρ of 0-bits in M to be $\frac{1}{2}$
- It can be shown that ρ is concentrated around its expectation, meaning that with high probability, $\rho \approx E[\rho] = p'$
- Thus, the probability of a false positive is

$$(1-\rho)^k \approx (1-p')^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k$$

Simplifying the expression for the false positive probability

- We showed that the probability of a false positive is approximately

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

- We simplify this with the approximation $1+x \approx e^{x}$ for x close to 0
- Setting $x=-1/m$, the approximate probability then becomes:

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

- We want to pick k to minimize this expression

Minimizing the chance of false positives

- Goal: pick k to minimize the function $f(k) = \exp(k \ln(1 - e^{-\frac{kn}{m}}))$
- Standard calculus shows that this occurs for $k_{\min} = \frac{m}{n} \ln 2$
- The minimum probability of a false positive is thus approximately

$$\begin{aligned} f(k_{\min}) &= \exp\left(\left(\frac{m}{n} \ln 2\right) \ln(1 - e^{-\left(\frac{m}{n} \ln 2\right)\frac{n}{m}})\right) \\ &= \exp\left(\left(\frac{m}{n} \ln 2\right) \ln(1 - 1/2)\right) \\ &= \exp\left(-\frac{m}{n} (\ln 2)^2\right) \\ &= 2^{-\frac{m}{n} \ln 2} = 2^{-k_{\min}} = (1/2)^{k_{\min}} \end{aligned}$$

- Equivalently, this is

$$2^{-\frac{m}{n} \ln 2} = (2^{-\ln 2})^{\frac{m}{n}} \approx 0.6185^{\frac{m}{n}}$$

Space requirement for $k = k_{\min}$

- Fix $k = k_{\min} = \frac{m}{n} \ln 2$ to minimize the chance of false positives
- As we showed, the false positive rate ϵ is:

$$\epsilon = (1/2)^{k_{\min}} = (1/2)^{(m/n) \ln 2} \Leftrightarrow 1/\epsilon = 2^{(m/n) \ln 2}$$

- To analyze space, take the logarithm and isolate m:

$$\log_2(1/\epsilon) = (m/n) \ln 2 \Leftrightarrow m = \frac{n \log_2(1/\epsilon)}{\ln 2} = n \log_2 e \log_2(1/\epsilon)$$

  where we used $\log_2 e = \ln e / \ln 2 = 1/\ln 2$
- Thus, the number m of bits stored is $n \log_2 e \log_2(1/\epsilon)$

Better data structure than the Bloom filter?

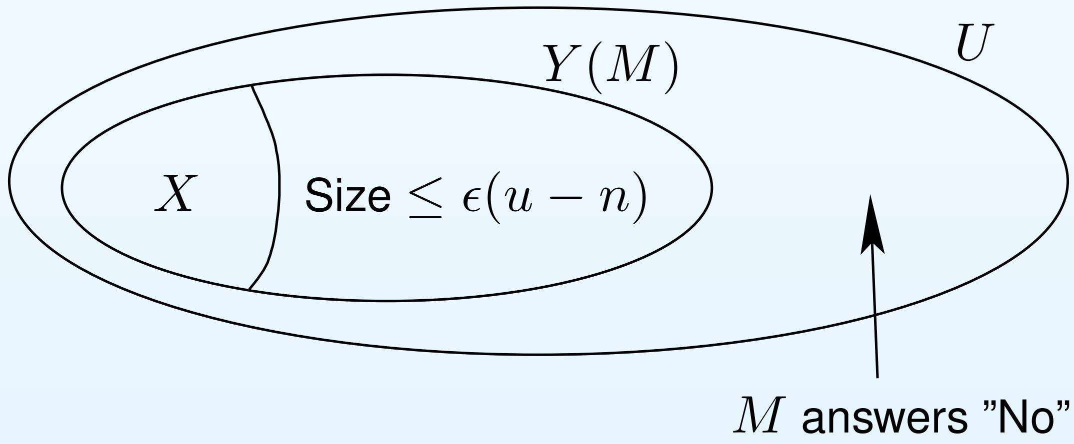
- Is there a data structure requiring significantly less space than a Bloom filter if we allow no false negatives and allow false positives for at most an ϵ fraction of elements of $U \setminus X$?
- We will show that this is not the case: only minor improvements in space are possible

Lower bound

- Consider any such data structure and let m be the number of bits it requires
- Each instance X gives rise to such an m-bit string and we say that X is *represented* by this string

Lower bound

- Consider an m-bit string M (one instance of the data structure)
- Let $Y(M)$ be the set of elements of U that M answers “Yes” to
- For any X represented by M, we must have $X \subseteq Y(M)$ (no false negatives)
- We allow at most a false positive rate of ϵ for $U \setminus X$
- Thus, $Y(M)$ contains at most $\epsilon(u-n)$ elements in addition to X



- It follows that $|Y(M)| \leq n + \epsilon(u-n)$

Lower bound

- $Y(M)$: elements that M answers "Yes" to
- Have shown: $|Y(M)| \leq n + \epsilon(u-n)$
- M can thus not represent more than $\binom{n+\epsilon(u-n)}{n}$ subsets X since they all need to be contained in $Y(M)$
- There are 2^m choices of m-length bit string M and each represents at most $\binom{n+\epsilon(u-n)}{n}$ sets X
- Hence, the data structure cannot represent more sets than

$$2^m \binom{n+\epsilon(u-n)}{n}$$

- However, it needs to represent *all* of the $\binom{u}{n}$ sets X so

$$2^m \binom{n+\epsilon(u-n)}{n} \geq \binom{u}{n}$$

Lower bound

- We have shown $2^m \binom{n+\epsilon(u-n)}{n} \geq \binom{u}{n}$
- Taking the logarithm and assuming $n \ll \epsilon u$, we isolate m:

$$m \geq \log_2 \left(\frac{\binom{u}{n}}{\binom{n+\epsilon(u-n)}{n}} \right) \approx \log_2 \left(\frac{\binom{u}{n}}{\binom{\epsilon u}{n}} \right)$$

$$\approx \log_2 \left(\frac{\left(\frac{u^n}{n!}\right)}{\left(\frac{(\epsilon u)^n}{n!}\right)} \right) = \log_2 \left((1/\epsilon)^n \right) = n \log_2(1/\epsilon)$$

- For the second approximation, we used that $n \ll \epsilon u$ implies

$$\binom{\epsilon u}{n} = \frac{(\epsilon u)!}{n!(\epsilon u - n)!} = \frac{(\epsilon u)(\epsilon u - 1) \cdots (\epsilon u - n + 1)}{n!} \approx \frac{(\epsilon u)^n}{n!}$$

- Similarly $\binom{u}{n} \approx \frac{u^n}{n!}$ since $n \ll \epsilon u \leq u$

Bloom filter compared to lower bound

- Have shown lower bound on m of $n \log_2(1/\epsilon)$ bits
- Recall that the Bloom filter requires $n \log_2 e \log_2(1/\epsilon)$ bits of space
- We see that the space requirement of the Bloom filter is within a factor $\log_2 e \approx 1.44$ of the lower bound
- More complicated data structures with better space bounds exist, for instance compressed Bloom filters

Drawback of our analysis

- Our analysis relied on hash functions with strong independence guarantees
- It is not known how to ensure such guarantees without using a lot of space (around $n \log n$ bits)
- Fortunately, Bloom filters work well using much more practical hash functions with weaker guarantees