Bloom Filters

1 Characteristics

Bloom filters are built for fast inserts and lookups of objects, must like hash tables. However, unlike a hash table, a bloom filter can only tell the user whether or not an object exists in the collection (no storage of associated objects), can't have entries deleted once they are added, and presents a small false positive probability.

Bloom filters are much more space-efficient than hash tables however, and are thus used for applications like spellcheckers, forbidden password lists, and network routers to block IP addresses and prevent DDoS attacks.

2 Implementation

Bloom filters are made of an array of n bits `A` and k hash functions $h_1, \ldots, h_k$, where k is some small constant. Bloom filters support two operations, insertions and lookups.

```
insert(x):
    for i in [1, k]:
        A[h_i(x)] = 1

lookup(x):
    return all(A[h_i(x)] for i in [1, k])
```

3 Heuristic Analysis

Assume all $h_i(x)$ are uniformly random and independent, and a bloom filter with n bits and a data set S already inserted.

$$\forall i \in [1, n], P(A[i]) = 1 - (1 - \frac{1}{n})^{k|S|} \tag{1}$$

$$= O(1 - e^{-\frac{k|S|}{n}}) \tag{2}$$

$$= O(1 - e^{-\frac{k}{b}}), \quad b = \frac{|S|}{n} \tag{3}$$

Thus the false-positive probability ϵ is given by

$$\epsilon \leq (1 - e^{-\frac{k}{b}})^k \tag{4}$$

For a fixed ratio b, ϵ is minimized by $k \approx (\ln 2) \cdot b$. After some more algebra,

$$\epsilon \approx \frac{1}{2}^{(\ln 2)b}, \qquad b \approx 1.44 \log_2 \left(\frac{1}{\epsilon}\right) \tag{5}$$