CS 591, Lecture 7
Data Analytics: Theory and Applications
Boston University

Babis Tsourakakis

February 13th, 2017

Bloom Filter

- Approximate membership problem
- Highly space-efficient randomized data structure
- Its analysis shows an interesting tradeoff between space and error probability
- **The Bloom filter principle** [Broder and Mitzenmacher, 2004]:

  Wherever a list or set is used, and space is at a premium,consider using a Bloomfilter if the effect of false positives can be mitigated.

Bloom Filter – Applications

Historically, Bloom filter was developed in the context of **dictionary applications** when space resources were scarce.

- Burton H. Bloom introduced Bloom filters (1970) for an application related to hyphenation programs.
- Bloom filters were also used in early UNIX spell-checker (space savings were crucial for functionality)
- Avoid weak passwords.

Bloom Filter – Applications

Content Delivery in P2P networks [Byers et al., 2002]

- Peer A has items S_A
- Peer B has items S_B
- B wants to obtain items in $S_A - S_B$
- Communication efficient approach: B send a Bloom filter to A

Bloom Filter – Applications

More important applications in

- **Networks**
- **Distributed Caching**
- **Databases** (e.g., Bloomjoin algorithm)
- ...

For a great survey on Bloom filters, read [Broder and Mitzenmacher, 2004].

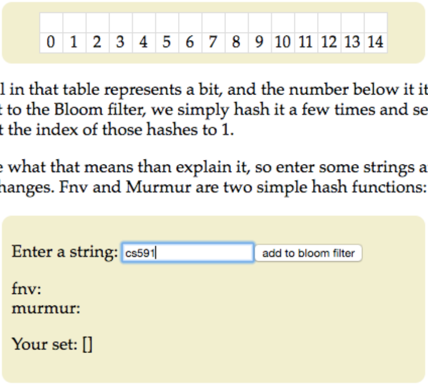
Bloom Filter – Description

1. A vector of m bits
2. k independent hash functions $h_1, \ldots, h_k$
3. A set S of n keys
4. To store key x, we set $A[h_i(x)] = 1$ for all $i \in [k]$
5. Lookup(x): if $A[h_i(x)] = 1$ for all $i \in [k]$, then $x \in S$.
6. No false negatives, but false positives may exist.

Bloom filter in Python – Pybloom library

```
>>> from pybloom import BloomFilter
>>> f = BloomFilter(capacity=1000, err=0.001)
>>> [f.add(x) for x in range(10)]
>>> all([(x in f) for x in range(10)])
True
>>> 10 in f
False
>>> 5 in f
True
```

Bloom Filters – Example



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Each empty cell in that table represents a bit, and the number below it its index. To add an element to the Bloom filter, we simply hash it a few times and set the bits in the bit vector at the index of those hashes to 1.

It's easier to see what that means than explain it, so enter some strings and see how the bit vector changes. Fnv and Murmur are two simple hash functions:

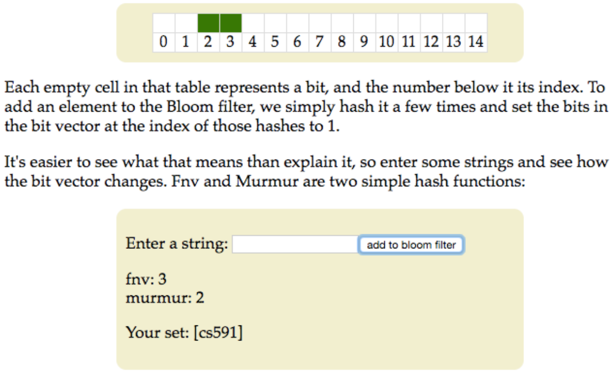
Enter a string: cs591

fnv:
murmur:

Your set: []

Demo: Bloom Filters by Example

Bloom Filters – Example



		🟩	🟩											
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Each empty cell in that table represents a bit, and the number below it its index. To add an element to the Bloom filter, we simply hash it a few times and set the bits in the bit vector at the index of those hashes to 1.

It's easier to see what that means than explain it, so enter some strings and see how the bit vector changes. Fnv and Murmur are two simple hash functions:

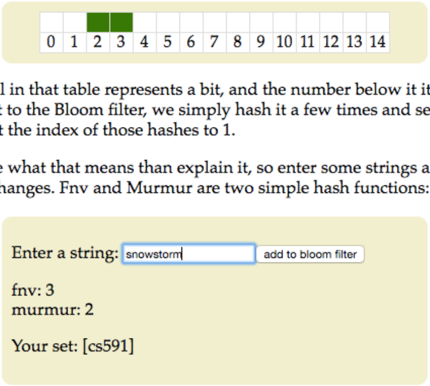
Enter a string:

fnv: 3
murmur: 2

Your set: [cs591]

Demo: Bloom Filters by Example

Bloom Filters – Example



		🟩	🟩											
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Each empty cell in that table represents a bit, and the number below it its index. To add an element to the Bloom filter, we simply hash it a few times and set the bits in the bit vector at the index of those hashes to 1.

It's easier to see what that means than explain it, so enter some strings and see how the bit vector changes. Fnv and Murmur are two simple hash functions:

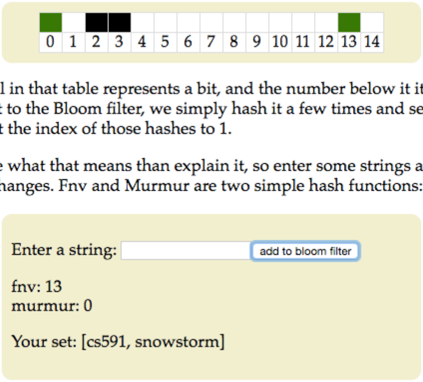
Enter a string: snowstorm

fnv: 3
murmur: 2

Your set: [cs591]

Demo: Bloom Filters by Example

Bloom Filters – Example



Each empty cell in that table represents a bit, and the number below it its index. To add an element to the Bloom filter, we simply hash it a few times and set the bits in the bit vector at the index of those hashes to 1.

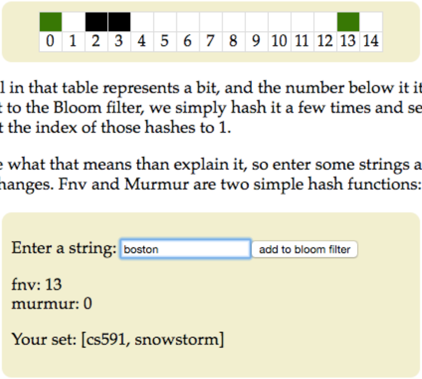
It's easier to see what that means than explain it, so enter some strings and see how the bit vector changes. Fnv and Murmur are two simple hash functions:

fnv: 13
murmur: 0

Your set: [cs591, snowstorm]

Demo: Bloom Filters by Example

Bloom Filters – Example



🟩		■	■										🟩	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Each empty cell in that table represents a bit, and the number below it its index. To add an element to the Bloom filter, we simply hash it a few times and set the bits in the bit vector at the index of those hashes to 1.

It's easier to see what that means than explain it, so enter some strings and see how the bit vector changes. Fnv and Murmur are two simple hash functions:

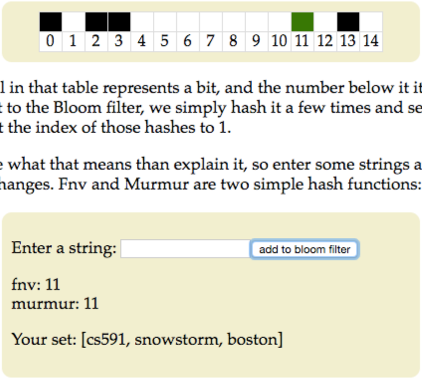
Enter a string: boston

fnv: 13
murmur: 0

Your set: [cs591, snowstorm]

Demo: Bloom Filters by Example

Bloom Filters – Example



■		■	■								🟩		■	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Each empty cell in that table represents a bit, and the number below it its index. To add an element to the Bloom filter, we simply hash it a few times and set the bits in the bit vector at the index of those hashes to 1.

It's easier to see what that means than explain it, so enter some strings and see how the bit vector changes. Fnv and Murmur are two simple hash functions:

Enter a string: [ ] add to bloom filter

fnv: 11
murmur: 11

Your set: [cs591, snowstorm, boston]

Demo: Bloom Filters by Example

Bloom Filters – False positives

- **Assumption:** h_i are close to being independent hash functions, probes are uniform
- **Claim:** $\mathbf{Pr}\left[A(i) = 1\right] = 1 - (1 - \frac{1}{m})^{kn}$
- **Why?**

Bloom Filters – False positives

- How can we upper bound $\mathbf{Pr}\left[A(i)\right]$?

$$\mathbf{Pr}\left[A(i)=1\right]=1-(1-\frac{1}{m})^{kn}\leq 1-e^{kn/m}.$$

Where is the mistake?

Bloom Filters – False positives

- The inequality goes the wrong way, since $1 - x \leq e^{-x}$.
- We will use instead $1 - x \geq e^{-x-x^2}$ which holds for $x \in [0, 0.683]$.

$$\mathbf{Pr}\left[A(i) = 1\right] = 1 - (1 - \frac{1}{m})^{kn} \leq 1 - e^{-\frac{kn}{m}(1+\frac{1}{m})} \\ = 1 - e^{-k\alpha'},$$

where $\alpha = \frac{n}{m}$ is the load factor, and $\alpha' = (1 + \frac{1}{m})\alpha$.

Bloom Filters – False positives

- What is the probability of a false positive, i.e., x that was never inserted by satisfies $A(h_i(x)) = 1$ for all $i \in [k]$?

$$\Big(\frac{\sum A(i)}{m}\Big)^k.$$

- Here, for simplicity we will assume that the false positive probability is

$$(1 - e^{-k\alpha'})^k$$

- **Question:** Fix the load factor α, what is the value of k that minimizes the error probability?

Bloom Filters – False positives

- We will minimize the error probability by taking the derivative wrt $x = e^{-k\alpha'}$.
- Solving for $k = -\frac{1}{\alpha'} logx$

$$\frac{d}{dx}\log\left((1-x)^k\right) = \frac{d}{dx}k\log(1-x) = \frac{d}{dx}\Big(-\frac{1}{\alpha'}logx\log(1-x)\Big)$$

$$= -\frac{1}{\alpha'}\Big(\frac{\log 1-x}{x} - \frac{\log x}{1-x}\Big) = 0.$$

Bloom Filters – False positives

- By simplifying the above we obtain that

$$x^* \log x^* = (1 - x^*) \log (1 - x^*) \Rightarrow x^* = \frac{1}{2}.$$

- In retrospect, this outcome is intuitive (why?)
- $k = \frac{1}{\alpha'} \log 2 = \frac{1}{\alpha(1+m^{-1})} \log 2$

- For a given false positive rate ϵ

$$2^{-\frac{\log 2}{\alpha'}} \leq \epsilon \Rightarrow \alpha \leq \frac{\log^2 2}{(1 + \frac{1}{m}) \log(\frac{1}{\epsilon})}.$$

Bloom Filters – Do we need k hash functions?

Double hashing works!

- Instead of using k random hash functions, one can choose two sufficiently random hash functions h, h' and then set

$$h_i(x) = h(x) + ih'(x) \bmod m.$$

- This was proved [Kirsch and Mitzenmacher, 2006].

- Dillinger and Manolios had earlier suggested

$$h_i(x) = h(x) + ih'(x) + i^2 \bmod m,$$

  as an effective heuristic.

references I

Broder, A. and Mitzenmacher, M. (2004).
Network applications of bloom filters: A survey.
Internet mathematics, 1(4):485–509.

Byers, J., Considine, J., Mitzenmacher, M., and Rost, S. (2002).
Informed content delivery across adaptive overlay networks.
ACM SIGCOMM Computer Communication Review, 32(4):47–60.

Kirsch, A. and Mitzenmacher, M. (2006).
Less hashing, same performance: building a better bloom filter.
In *European Symposium on Algorithms*, pages 456–467. Springer.