3.3 Variance and Standard Deviation recap

Anna Karlin
Most Slides by Alex Tsun

Agenda

- Variance
- Independence of random variables
- Properties of variance

Variance and Standard Deviation (SD)

Variance: The variance of a random variable X is

More Useful

$$Var(X) = E[(X - E[X])^2] \boxed{= E[X^2] - (E[X])^2}$$

The variance is always nonnegative since we take an expectation of a nonnegative random variable $(X - E[X])^2$. We can also show that for any scalars $a, b \in \mathbb{R}$,

$$Var(aX + b) = a^2\, Var(X)$$

Standard Deviation (SD): The standard deviation of a random variable X is

$$\sigma_X = \sqrt{Var(X)}$$

We want this because the units of variance are squared in terms of the original variable X, and this "undo's" our squaring, returning the units to the same as X.

Random variables and independence

Random variable X and event E are independent if the event E is independent of the event {X=x} (for any fixed x), i.e.

$$\forall x \; P(X = x \text{ and } E) = P(X=x) \bullet P(E)$$

Two random variables X and Y are independent if the events {X=x} and {Y=y} are independent for any fixed x, y, i.e.

$$\forall x, y \; P(X = x \text{ and } Y=y) = P(X=x) \bullet P(Y=y)$$

Intuition as before: knowing X doesn't help you guess Y or E and vice versa.

Independent vs dependent r.v.s

- Dependent r.v.s can reinforce/cancel/correlate in arbitrary ways.
- Independent r.v.s are, well, independent.

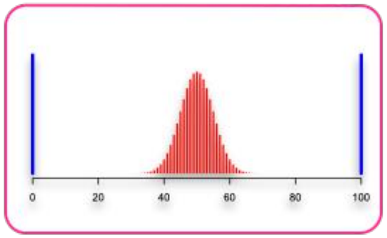
Example:

$Z = X_1 + X_2 + \ldots. + X_n$

X_i is indicator r.v. with probability 1/2 of being 1.

versus

$W = n X_1$



Important facts about independent random variables

Theorem: If X & Y are independent, then $E[X \bullet Y] = E[X] \bullet E[Y]$

Theorem: If X and Y are independent, then

$$Var[X + Y] = Var[X] + Var[Y]$$

Corollary: If $X_1 + X_2 + \ldots + X_n$ are mutually independent then

$$Var[X_1 + X_2 + \ldots + X_n] = Var[X_1] + Var[X_2] + \ldots + Var[X_n]$$

E[XY] FOR INDEPENDENT RANDOM VARIABLES

- Theorem: If X & Y are independent, then E[X•Y] = E[X]•E[Y]
- Proof:

Let $x_i, y_i, i = 1, 2, \ldots$ be the possible values of X, Y.

$$\begin{aligned} E[X \cdot Y] &= \sum_i \sum_j x_i \cdot y_j \cdot P(X = x_i \wedge Y = y_j) \quad \text{independence} \\ &= \sum_i \sum_j x_i \cdot y_j \cdot P(X = x_i) \cdot P(Y = y_j) \\ &= \sum_i x_i \cdot P(X = x_i) \cdot \left(\sum_j y_j \cdot P(Y = y_j) \right) \\ &= E[X] \cdot E[Y] \end{aligned}$$

Note: *NOT* true in general; see earlier example $E[X^2] \neq E[X]^2$

x

Variance of a sum of independent r.v.s

Theorem: If X and Y are independent, then

Var[X + Y] = Var[X] + Var[Y]

Proof:

$$Var[X+Y]$$

$$= E[(X+Y)^2] - (E[X+Y])^2$$

$$= E[X^2 + 2XY + Y^2] - (E[X] + E[Y])^2$$

$$= E[X^2] + 2E[XY] + E[Y^2] - ((E[X])^2 + 2E[X]E[Y] + (E[Y])^2)$$

$$= E[X^2] - (E[X])^2 + E[Y^2] - (E[Y])^2 + 2(E[XY] - E[X]E[Y])$$

$$= Var[X] + Var[Y] + 2(E[X]E[Y] - E[X]E[Y])$$

$$= Var[X] + Var[Y]$$

x

Bloom Filters

Anna Karlin
Most slides by Shreya Jayaraman, Luxi Wang, Alex Tsun

Hashing

Basic Problem

Problem: Store a subset S of a large set U.

Example. U = set of 128 bit strings $|U| \approx 2^{128}$

S = subset of strings of interest $|S| \approx 1000$

Two goals:

1. **Constant-time** answering of queries "Is $x \in S$?"
2. **Minimize storage** requirements.

Naïve Solution – Constant Time

Idea: Represent S as an array A with 2^{128} entries.

$$\mathrm{A}[x] = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases}$$

$S = \{0,2, \dots, \mathrm{K}\}$

0	1	2	…	*K*	…		
1	0	1	0	1	…	0	0

Membership test: To check. $x \in S$ just check whether A$[x] = 1$.

$\longrightarrow$ **constant time!** 👍 😀

Storage: Require storing 2^{64} bits, even for small S. 👎 😢

Naïve Solution – Small Storage

Idea: Represent S as a list with $|S|$ entries.

$S = \{0, 2, \ldots, K\}$

Storage: Grows with $|S|$ <u>only</u> 👍 😀

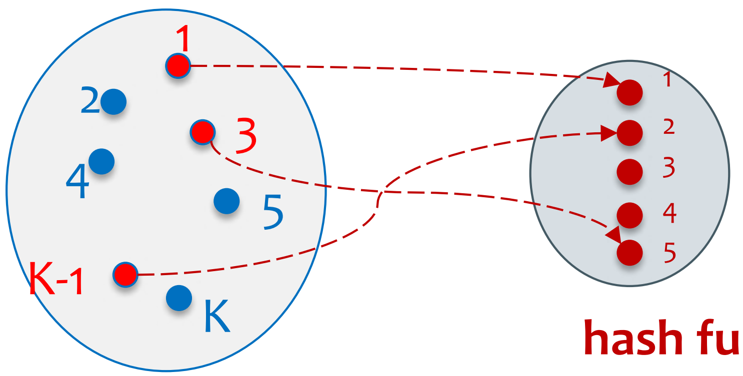
Membership test: Check $x \in S$ requires time linear in $|S|$
(Can be made logarithmic by using a tree) 👎 😢

Hash Table

Idea: Map elements in S into an array A using a hash function

Membership test: To check $x \in S$
just check whether $A[\mathbf{h}(x)] = x$

Storage: n elements



hash function h: $[\mathrm{U}] \rightarrow [n]$

Hash Table

Idea: Map elements in S into an ar

Membership test: To check $x \in S$
just check whether $A[\mathbf{h}(x)] = x$

Storage: n elements

Challenge 1: Ensure
$\mathbf{h}(x) \neq \mathbf{h}(y)$
for most $x, y \in S$

Challenge 2: Ensure
$n = O(|S|)$

Hashing – collisions

- **Collisions** occur when two elements of set map to the same location in the hash table.
- Common solution: chaining – at each location (bucket) in the table, keep linked list of all elements that hash there.

- Want: hash function that distributes the elements of S well across hash table locations. Ideally uniform distribution!

Summary

Hash Tables

- They store the data itself
- With a good hash function, the data is well distributed in the table and lookup times are small.
- However, they need at least as much space as all the data being stored
- E.g. storing strings, or IP addresses or long DNA sequences.

Bloom Filters: Motivation

- Large universe of possible data items.
- Data items are large (say 128 bits or more)
- Hash table is stored on disk or across network, so any lookup is expensive.
- Many (if not nearly all) of the lookups return “Not found”.

Altogether, this is bad. You’re wasting **a lot of time and space** doing lookups for items that aren’t even present.

Bloom Filters: Motivation

- Large universe of possible data items.
- Hash table is stored on disk or in network, so any lookup is expensive.
- Many (if not most) of the lookups return “Not found”.

Altogether, this is bad. You’re wasting **a lot of time and space** doing lookups for items that aren’t even present.

Examples:

- Google Chrome: wants to warn you if you’re trying to access a malicious URL. Keep hash table of malicious URLs.
- Network routers: want to track source IP addresses of certain packets, .e.g., blocked IP addresses.

Bloom Filters: Motivation

- Probabilistic data structure.
- Close cousins of hash tables.
- Ridiculously space efficient
- To get that, make occasional errors, specifically false positives.

Typical implementation: only 8 bits per element!

Bloom Filters

Bloom Filters

- Stores information about a set of elements.
- Supports two operations:
  1. **add(x)** - adds x to bloom filter
  2. **contains(x)** - returns true if x in bloom filter, otherwise returns false
     a. If return false, **definitely** not in bloom filter.
     b. If return true, **possibly** in the structure (some false positives).

Bloom Filters

- Why accept false positives?
  - **Speed** - both operations very very fast.
  - **Space** - requires a miniscule amount of space relative to storing all the actual items that have been added.

- Often just 8 bits per inserted item!

Bloom Filters: Initialization

Number of hash functions

Size of array associated to each hash function.

function INITIALIZE(k,m)

  for $i = 1, \ldots, k$: **do**

    t_i = new bit vector of m 0's

for each hash function, initialize an empty bit vector of size m

Bloom Filters: Example

bloom filter t with m = 5 that uses k = 3 hash functions

function INITIALIZE(k,m)
  for $i = 1, \ldots, k$: **do**
    t_i = new bit vector of m 0's

Index →	0	1	2	3	4
$\mathbf{t_1}$	0	0	0	0	0
$\mathbf{t_2}$	0	0	0	0	0
$\mathbf{t_3}$	0	0	0	0	0

Bloom Filters: Add

function ADD(X)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

for each hash function h_i

$h_i(x)$ → result of hash function h_i on x

Bloom Filters: Add

function ADD(X)

  for $i = 1, \ldots, k$: **do**

    $t_i[h_i(x)] = 1$

for each hash function h_i

Index into ith bit-vector, at index produced by hash function and set to 1

Bloom Filters: Example

bloom filter t with m = 5 that uses k = 3 hash functions

add(“thisisavirus.com”)

function ADD(x)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

Index →	**0**	**1**	**2**	**3**	**4**
t_1	0	0	0	0	0
t_2	0	0	0	0	0
t_3	0	0	0	0	0

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

function ADD(X)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

add("thisisavirus.com")

h_1("thisisavirus.com") → 2

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	0	0	0	0
t_3	0	0	0	0	0

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

function ADD(X)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

add("thisisavirus.com")

h_1("thisisavirus.com") → 2
h_2("thisisavirus.com") → 1

Index →	0	1	2	3	4
t_1	0	0	1	0	0
t_2	0	1	0	0	0
t_3	0	0	0	0	0

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

function ADD(X)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

add("thisisavirus.com")

h_1("thisisavirus.com") → 2
h_2("thisisavirus.com") → 1
h_3("thisisavirus.com") → 4

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Contains

function CONTAINS(x)
    return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

Returns True if the bit vector for each hash function has bit 1 at index determined by that hash function, otherwise returns False

Bloom Filters: Example

bloom filter t with m = 5 that uses k = 3 hash functions

contains("thisisavirus.com")

function CONTAINS(x)
  return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("thisisavirus.com")

h_1("thisisavirus.com") → 2

function CONTAINS(x)
  return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("thisisavirus.com")

h_1("thisisavirus.com") → 2

h_2("thisisavirus.com") → 1

function CONTAINS(x)
return $t_1[h_1(x)] == 1 \land t_2[h_2(x)] == 1 \land \cdots \land t_k[h_k(x)] == 1$

True True

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("thisisavirus.com")

h_1("thisisavirus.com") → 2

h_2("thisisavirus.com") → 1

h_3("thisisavirus.com") → 4

function CONTAINS(x)
  return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True   True   True

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("thisisavirus.com")

h_1("thisisavirus.com") → 2

h_2("thisisavirus.com") → 1

h_3("thisisavirus.com") → 4

function CONTAINS(x)
  return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True **True** **True**

Since all conditions satisfied, returns True (correctly)

Index →	**0**	**1**	**2**	**3**	**4**
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: False Positives

bloom filter t of length m = 5 that uses k = 3 hash functions

add(“totallynotsuspicious.com”)

function ADD(x)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

Index →	0	1	2	3	4
t_1	0	0	1	0	0
t_2	0	1	0	0	0
t_3	0	0	0	0	1

Bloom Filters: False Positives

bloom filter t of length m = 5 that uses k = 3 hash functions

add("totallynotsuspicious.com")

h_1("totallynotsuspicious.com") → 1

function ADD(x)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

Index →	**0**	**1**	**2**	**3**	**4**
$\mathbf{t_1}$	0	**1**	1	0	0
$\mathbf{t_2}$	0	1	0	0	0
$\mathbf{t_3}$	0	0	0	0	1

Bloom Filters: False Positives

bloom filter t of length m = 5 that uses k = 3 hash functions

function ADD(x)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

add("totallynotsuspicious.com")

h_1("totallynotsuspicious.com") → 1

h_2("totallynotsuspicious.com") → 0

Index →	0	1	2	3	4
t_1	0	**1**	1	0	0
t_2	**1**	1	0	0	0
t_3	0	0	0	0	1

Bloom Filters: False Positives

bloom filter t of length m = 5 that uses k = 3 hash functions

function ADD(x)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

add("totallynotsuspicious.com")

h_1("totallynotsuspicious.com") → 1

h_2("totallynotsuspicious.com") → 0

h_3("totallynotsuspicious.com") → 4

Index →	0	1	2	3	4
t_1	0	**1**	1	0	0
t_2	**1**	1	0	0	0
t_3	0	0	0	0	1

Collision, is already set to 1

Bloom Filters: False Positives

bloom filter t of length m = 5 that uses k = 3 hash functions

function ADD(x)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

add("totallynotsuspicious.com")

h_1("totallynotsuspicious.com") → 1

h_2("totallynotsuspicious.com") → 0

h_3("totallynotsuspicious.com") → 4

Index →	0	1	2	3	4
t_1	0	**1**	1	0	0
t_2	**1**	1	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("verynormalsite.com")

function CONTAINS(x)
  return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

Index →	0	1	2	3	4
t_1	0	1	1	0	0
t_2	1	1	0	0	0
t_3	0	0	0	0	1

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("verynormalsite.com")

h_1("verynormalsite.com") → 2

function CONTAINS(x)

return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True

Index →	0	1	2	3	4
t_1	0	1	**1**	0	0
t_2	1	1	0	0	0
t_3	0	0	0	0	1

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains(“verynormalsite.com”)

h_1(“verynormalsite.com”) → 2

h_2(“verynormalsite.com”) → 0

function CONTAINS(x)
    return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True **True**

Index →	0	1	2	3	4
t_1	0	1	**1**	0	0
t_2	**1**	1	0	0	0
t_3	0	0	0	0	1

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("verynormalsite.com")

h_1("verynormalsite.com") → 2

h_2("verynormalsite.com") → 0

h_3("verynormalsite.com") → 4

function CONTAINS(x)

return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True **True** **True**

Index →	**0**	**1**	**2**	**3**	**4**
t_1	0	1	**1**	0	0
t_2	**1**	1	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("verynormalsite.com")

h_1("verynormalsite.com") → 2

h_2("verynormalsite.com") → 0

h_3("verynormalsite.com") → 4

function CONTAINS(x)
  return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True True True

Since all conditions satisfied, returns True (incorrectly)

Index →	0	1	2	3	4
t_1	0	1	**1**	0	0
t_2	**1**	1	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Summary

- An empty bloom filter is an empty k x m bit array with all values initialized to zeros
  - k = number of hash functions
  - m = size of each array in the bloom filter
- add(x) runs in O(k) time
- contains(x) runs in O(k) time
- requires O(km) space (in bits!)
- Probability of false positives from collisions can be reduced by increasing the size of the bloom filter

Bloom Filters: Application

- Google Chrome has a database of malicious URLs, but it takes a long time to query.
- Want an in-browser structure, so needs to be efficient and be space-efficient
- Want it so that can check if a URL is in structure:
  - If return False, then definitely not in the structure (don't need to do expensive database lookup, website is safe)
  - If return True, the URL may or may not be in the structure. Have to perform expensive lookup in this rare case.

False positive probability

Comparison with Hash tables - Space

- Google storing 5 million URLs, each URL 40 bytes.
- Bloom filter with k=8 and m = 10,000,000.

Hash Table

Bloom Filter

Comparison with Hash tables - Time

- Say avg user visits 100,000 URLs in a year, of which 2,000 are malicious.
- 0.5 seconds to do lookup in the database, 1ms for lookup in Bloom filter.
- Suppose the false positive rate is 2%

Hash Table

Bloom Filter

Bloom Filters: Many Applications

- Any scenario where space and efficiency are important.
- Used a lot in networking
- In distributed systems when want to check consistency of data across different locations, might send a Bloom filter rather than the full set of data being stored.
- Google BigTable uses Bloom filters to reduce the disk lookups for non-existent rows and columns
- Internet routers often use Bloom filters to track blocked IP addresses.
- And on and on…

Bloom Filters typical example...

of randomized algorithms and randomized data structures.

- Simple
- Fast
- Efficient
- Elegant
- Useful!

- You'll be implementing Bloom filters on pset 4. Enjoy!