CSE 312

Foundations of Computing II

Lecture 9: Variance and Independence of RVs (continued)

Lecture 10: Bloom Filters

Announcements

- PSet 3 due today
- PSet 2 returned yesterday
- PSet 4 posted this evening
  - <u>Last</u> PSet prior to midterm (midterm is in exactly two weeks from now)
  - Midterm info will follow soon
  - PSet 5 will only come <u>after</u> the midterm in two weeks

Recap Variance – Properties

Definition. The **variance** of a (discrete) RV X is

$$\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \sum_x p_X(x) \cdot (x - \mathbb{E}[X])^2 \quad \geq 0$$

Theorem. For any $a, b \in \mathbb{R}$, $\mathrm{Var}(a \cdot X + b) = a^2 \cdot \mathrm{Var}(X)$

Theorem. $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

Variance

Theorem. $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

Proof: $\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$

Recall $\mathbb{E}[X]$ is a **constant**

$= \mathbb{E}[X^2 - 2\mathbb{E}[X] \cdot X + \mathbb{E}[X]^2]$

$= \mathbb{E}(X^2) - 2\mathbb{E}[X]\mathbb{E}[X] + \mathbb{E}[X]^2$

(linearity of expectation!)

$= \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$\mathbb{E}[X^2]$ and $\mathbb{E}[X]^2$ are different !

Variance of Indicator Random Variables

Suppose that X_A is an indicator RV for event A with $P(A) = p$ so

$$\mathbb{E}[X_A] = P(A) = p$$

Since X_A only takes on values 0 and 1, we always have $X_A^2 = X_A$ so

$$\mathrm{Var}(X_A) = \mathbb{E}[\overset{X_A}{\overset{\shortparallel}{X_A^2}}] - \mathbb{E}[X_A]^2 = \overset{p}{\mathbb{E}[X_A]} - \underset{p}{\mathbb{E}[X_A]^2} = p - p^2 = p(1-p)$$

In General, $\mathrm{Var}(X+Y) \neq \mathrm{Var}(X) + \mathrm{Var}(Y)$

$\mathrm{Var}(X) = \underbrace{\mathbb{E}[X^2]}_{1} - \underbrace{\mathbb{E}[X]^2}_{0}$

$P(X=1)$ (

Proof by counter-example:

- Let X be a r.v. with pmf $P(X=1) = P(X=-1) = 1/2$
  0 1
  - What is $\mathbb{E}[X]$ and $\mathrm{Var}(X)$?
- Let $Y = -X$
  $X + Y = 0$
  - What is $\mathbb{E}[Y]$ and $\mathrm{Var}(Y)$?
    0 1

What is $\mathrm{Var}(X+Y)$? $\neq \mathrm{VAR}(X) + \mathrm{VAR}(Y$

0 1 1

2

Agenda

- Variance
- Properties of Variance
- Independent Random Variables
- Properties of Independent Random Variables
- An Application: Bloom Filters!

Random Variables and Independence

Comma is shorthand for AND

Definition. Two random variables X, Y are **(mutually) independent** if for all $x, y,$

Ω_x Ω_y

$$P(\underbrace{X = x}_{A}, \underbrace{Y = y}_{B}) = P(\underbrace{X = x}_{A}) \cdot P(\underbrace{Y = y}_{B})$$

Intuition: Knowing X doesn't help you guess Y and vice versa

Definition. The random variables $X_1, \ldots, X_n$ are **(mutually) independent** if for all $x_1, \ldots, x_n,$

$$P(X_1 = x_1, \ldots, X_n = x_n) = P(X_1 = x_1) \cdots P(X_n = x_n)$$

Note: No need to check for all subsets, but need to check <u>for all outcomes</u>!

Example

Let X be the number of heads in n *independent* coin flips of the same coin. Let $Y = X \bmod 2$ be the parity (even/odd) of X.

Are X and Y independent?

$Pr[X=0] \neq 0$

$Pr[Y=1] \neq 0$

$Pr[X=0 \;\&\; Y=1]$

$= 0$

Poll:

pollev.com/paulbeame028

A. Yes

B. No ✓

Example

Make $2n$ *independent* coin flips of the same coin.

Let X be the number of heads in the first n flips and Y be the number of heads in the last n flips.

Are X and Y independent?

Poll:
pollev.com/paulbeame028

A. Yes ✓
B. No

Agenda

- Variance
- Properties of Variance
- Independent Random Variables
- Properties of Independent Random Variables
- An Application: Bloom Filters!

Important Facts about Independent Random Variables

Theorem. If X, Y independent, $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$

Theorem. If X, Y independent, $\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$

Corollary. If $X_1, X_2, \ldots, X_n$ mutually independent,

$$\mathrm{Var}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i}^{n} \mathrm{Var}(X_i)$$

n

n^2

Example – Coin Tosses

We flip n *independent* coins, each one heads with probability p

- $X_i = \begin{cases} 1, & i^{\text{th}} \text{ outcome is heads} \\ 0, & i^{\text{th}} \text{ outcome is tails.} \end{cases}$
- Z = number of heads

Fact. $Z = \sum_{i=1}^{n} X_i$

$P(X_i = 1) = p$
$P(X_i = 0) = 1 - p$

$\mathrm{Var}(\sum_{i=1} X_i) = \sum_{i=1}^{n} \underbrace{\mathrm{Var}(X_i)}_{=p(1-p)}$

np

What is $\mathbb{E}[Z]$? What is $\mathrm{Var}(Z)$? $np(1-p)$

$P(Z = k) = \binom{n}{k} p^k (1-p)^{n-k}$

Note: $X_1, \dots, X_n$ are <u>mutually</u> independent! [Verify it formally!]

$$\mathrm{Var}(Z) = \sum_{i=1}^{n} \mathrm{Var}(X_i) = n \cdot p(1-p)$$

Note $\mathrm{Var}(X_i) = p(1-p)$

(Not Covered) Proof of $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$

Theorem. If X, Y independent, $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$

Proof

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

$$\mathbb{E}[X \cdot Y] = \sum_i \sum_j x_i \cdot y_j \cdot P(X = x_i \wedge Y = y_j)$$

independence

$$= \sum_i \sum_j x_i \cdot y_i \cdot P(X = x_i) \cdot P(Y = y_j)$$

$$= \sum_i x_i \cdot P(X = x_i) \cdot \left(\sum_j y_j \cdot P(Y = y_j) \right)$$

$$= \mathbb{E}[X] \cdot \mathbb{E}[Y]$$

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

(Not Covered) Proof of $\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$

Theorem. If X, Y independent, $\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$

Proof

$$Var(X+Y)$$
$$= \mathbb{E}[(X+Y)^2] - (\mathbb{E}[X+Y])^2$$
$$= \mathbb{E}[X^2 + 2XY + Y^2] - (\mathbb{E}[X] + \mathbb{E}[Y])^2$$
$$= \mathbb{E}[X^2] + 2\,\mathbb{E}[XY] + \mathbb{E}[Y^2] - \left(\mathbb{E}[X]^2 + 2\,\mathbb{E}[X]\,\mathbb{E}[Y] + \mathbb{E}[Y]^2\right)$$
$$= \mathbb{E}[X^2] - \mathbb{E}[X]^2 + \mathbb{E}[Y^2] - \mathbb{E}[Y]^2 + 2\,\mathbb{E}[XY] - 2\,\mathbb{E}[X]\,\mathbb{E}[Y]$$
$$= Var(X) + Var(Y) + 2\,\mathbb{E}[XY] - 2\,\mathbb{E}[X]\,\mathbb{E}[Y]$$
$$= Var(X) + Var(Y)$$

linearity

equal by independence


Brain Break

Agenda

- Variance
- Properties of Variance
- Independent Random Variables
- Properties of Independent Random Variables
- An Application: Bloom Filters!

Basic Problem

Problem: Store a subset S of a <u>large</u> set U.

Example. $U =$ set of 128 bit strings $\quad |U| \approx 2^{128}$
$S =$ subset of strings of interest $\quad |S| \approx 1000$

Two goals:

1. **Very fast** (ideally constant time) answers to queries "Is $x \in S$?" for any $x \in U$.
2. **Minimal storage** requirements.

Naïve Solution I – Constant Time

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

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

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

0	1	2	…	K	…		
1	0	1	0	1	…	0	0

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

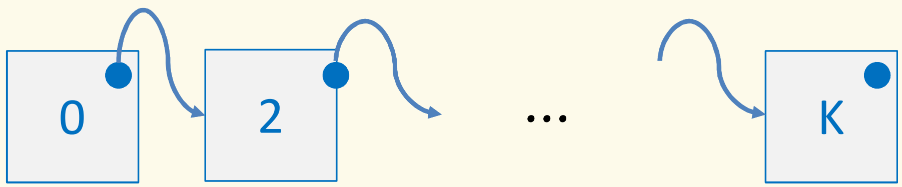
$\longrightarrow$ **constant time!** 👍 😀

Storage: Require storing 2^{128} bits, even for small S. 👎 😢

Naïve Solution II – 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 $|S|$

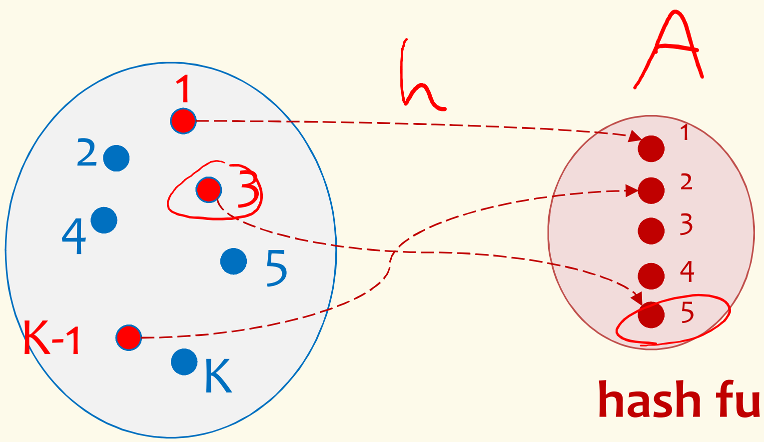
(Can be made logarithmic by using a tree) 👎 😢

Hash Table

Idea: Map elements in S into an array A of size m using a hash function $\mathbf{h}$

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

Storage: m elements (size of array)



hash function h: $U \rightarrow [m]$

Hash Table

Idea: Map elements in S into an array A of size m using a hash function $\mathbf{h}$

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

Storage: m elements (size of array)

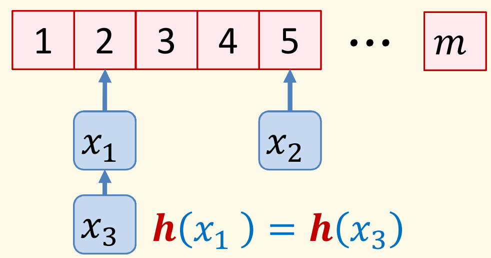
Challenge 1: Ensure $\boldsymbol{h}(x) \neq \boldsymbol{h}(y)$ *for most* $x, y \in S$

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

Hashing: collisions

Collisions occur when $\boldsymbol{h}(x) = \boldsymbol{h}(y)$ for some distinct $x, y \in S$, i.e., two elements of set map to the same location

- Common solution: <u>chaining</u> – at each location (bucket) in the table, keep linked list of all elements that hash there.



Good hash functions to keep collisions low

- The hash function h is good iff it
  - distributes elements uniformly across the m array locations so that
  - pairs of elements are mapped independently

"Universal Hash Functions" – see CSE 332

Hashing: 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, i.e., $m = \Omega(|S|)$

In some cases, $|S|$ is huge, or not known a-priori …

Can we do better!?

Bloom Filters

to the rescue

(Named after Burton Howard Bloom)

Bloom Filters – Main points

- Probabilistic data structure.
- Close cousins of hash tables.
  - But: Ridiculously space efficient
- Occasional errors, specifically false positives.

Bloom Filters

- Stores information about a set of elements $S \subseteq U$.
- Supports two operations:
  1. **add**(x) - adds $x \in U$ to the set S
  2. **contains**(x) – ideally: true if $x \in S$, false otherwise

Instead, relaxed guarantees:

- False $\longrightarrow$ **definitely** not in S
- True $\longrightarrow$ **possibly** in S

  [i.e. we could have *false positives*]

Bloom Filters – Why Accept False Positives?

- **Speed** – both **`add`** and **`contains`** 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!
- **Fallback mechanism** – can distinguish false positives from true positives with extra cost
  - Ok if mostly negatives expected + low false positive rate

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.

Bloom Filters – More 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 disk lookups
- Internet routers often use Bloom filters to track blocked IP addresses.
- And on and on…

What you can't do with Bloom filters

- There is no delete operation
  - Once you have added something to a Bloom filter for S, it stays

- You can't use a Bloom filter to name any element of S

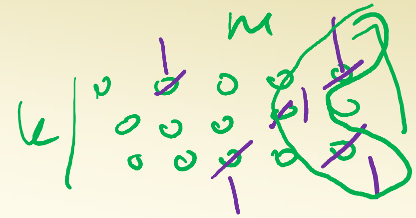
But what you ***can*** do makes them very effective!

Bloom Filters – Ingredients

Basic data structure is a $k \times m$ binary array
"the Bloom filter"

- k rows $t_1, \ldots, t_k$, each of size m
- Think of each row as an m-bit vector

k different hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k : U \to [m]$



Bloom Filters – Three operations

- Set up Bloom filter for $S = \emptyset$

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

- Update Bloom filter for $S \leftarrow S \cup \{x\}$

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

- Check if $x \in S$

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

Bloom Filters - Initialization

Number of hash functions

Size of array associated to each hash function.

$$
\begin{array}{l}
\textbf{function } \text{INITIALIZE}(k, m) \\
\quad \textbf{for } i = 1, \dots, k\text{: } \textbf{do} \\
\quad\quad t_i = \text{new bit vector of } m \text{ 0s}
\end{array}
$$

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