A glimpse of auction theory (continued) + distinct elements

Anna Karlin

Agenda

- Finish up glimpse of auction theory
- Distinct Elements

Auctions

- Companies like Google and Facebook make most of their money by selling ads.
- The ads are sold via auction.

Facebook Ads bidding... 🤔 Is this an auction?

Yes! That's the first thing you need to understand to master bidding management of Facebook Ads. **When you're creating a new campaign, you're joining a huge, worldwide auction**.

You'll be competing with hundreds of thousands of advertisers to buy what Facebook is selling: Real estate on the News Feed, Messenger, Audience Network, and mobile apps to display your ads to the users.

An auction is a ...

- Game
    - Players: advertisers
    - Strategy choices for each player: possible bids
    - Rules of the game – made up by Google/Facebook/whoever is running the auction

- What do we expect to happen? How do we analyze mathematically?

Special case: Sealed Bid single item auction

- Say I decide to run an auction to sell my laptop and I let you be the bidders.
- If I want to make as much money as possible – what should the rules of the auction be?

Some possibilities:

- **First price auction:** highest bidder wins; pays what they bid.
- **Second price auction:** highest bidder wins; pays second highest bid.
- **All pay auction:** highest bidder wins: all bidders pay what they bid.

Which of these will make me the most money?

Bidder model

Each bidder has a value, say v_i for bidder i.

Bidder is trying to maximize their "utility" –
the value of the item they get – price they pay.

Distinct elements

Anna Karlin

With many slides by Luxi Wang, Shreya Jayaraman,
Alex Tsun and Jeff Ullman

Data mining

- In many data mining situations, the data is not known ahead of time.

- Examples:
    - Google queries
    - Twitter or Facebook status updates
    - Youtube video views

- In some ways, best to think of the data as an infinite stream that is non-stationary (distribution changes over time)

Stream model

- Input elements (e.g. Google queries) enter/arrive one at a time.
- We cannot possibly store the stream.

Question: How do we make critical calculations about the data stream using a limited amount of memory?

Sources of this kind of data

- Sensor data
    - E.g. millions of temperature sensors deployed in the ocean
- Image data from satellites or surveillance camers
    - E.g. London
- Internet and web traffic
    - E.g. millions of streams of IP packets
- Web data
    - E.g. Search queries on Google, clicks on Bing, etc.

Example Applications

- Mining query streams
  - Google wants to know which queries are more frequent today than yesterday.
- Mining click streams
  - Facebook wants to know which of its ads are getting an unusual number of hits in the last hour.
- Mining social network news feeds
  - E.g., looking for trending topics on Twitter and Facebook, trending videos on TikTok

More Applications

- Sensor networks
  - Many sensors feeding into a central controller.
- IP packets
  - Gather congestion information for optimal routing
  - Detect denial-of-service attacks

Problem

- Input: sequence of N elements $x_1, x_2, \ldots, x_N$ from a known universe U (e.g., 8-byte integers).

- Goal: perform a computation on the input, in a single left to right pass where
  - Elements processed in real time
  - Can’t store the full data. => use minimal amount of storage while maintaining working “summary”

What can we compute?

32, 12, 14, 32, 7, 12, 32, 7, 32, 12, 4,

- Some functions are easy:
  - Min
  - Max
  - Sum
  - Average

Counting distinct elements

32, 12, 14, 32, 7, 12, 32, 7, 32, 12, 4,

Applications:

- IP packet streams: How many distinct IP addresses or IP flows (source+destination IP, port, protocol)
  - Anomaly detection, traffic monitoring
- Search: How many distinct search queries on Google on a certain topic yesterday
- Web services: how many distinct users (cookies) searched/browsed a certain term/item
  - Advertising, marketing trends, etc.

Another application

You are the content manager at YouTube, and you are trying to figure out the **distinct** view count for a video. How do we do that?

Note: A person can view their favorite videos several times, but they only count as 1 **distinct** view!

Counting distinct elements

32, 12, 14, 32, 7, 12, 32, 7, 32, 12, 4,

- Want to compute number of **distinct** keys in the stream.
- *How to do this without storing all the elements?*
- *Yet another super cool application of probability (and hashing)*

A naive solution, counting!

Store the n **distinct** user IDs in a hash table.

Space requirement: O(n)

CONSIDERING THE NUMBER OF USERS OF YOUTUBE, AND THE NUMBER OF VIDEOS ON YOUTUBE, THIS IS NOT FEASIBLE.

Consider a hash function $h : \mathcal{U} \rightarrow [0, 1]$
For distinct values in $\mathcal{U}$, the function maps to iid (independent and identically distributed) Unif(0,1) random numbers.

Note that, if you were to feed in two equivalent elements, the function returns the **same** number.

32, 12, 14, 32, 7, 12, 32, 7, 32, 12, 4,

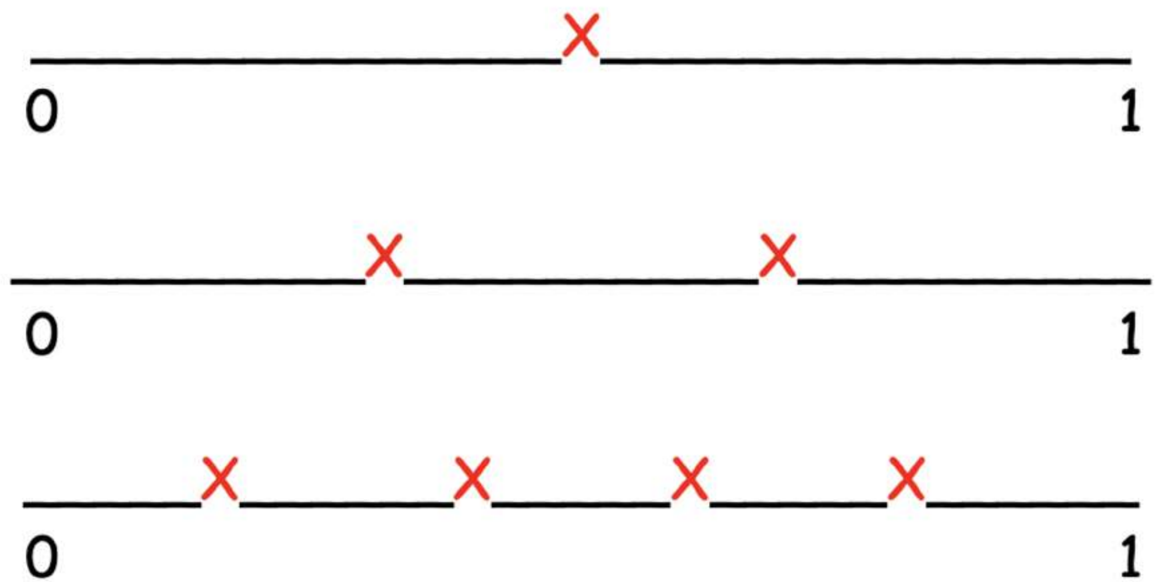
Min of IID Uniforms

If $Y_1, \ldots, Y_m$ are iid $Unif(0,1)$, where do we "expect" the points to end up?

$m = 1$

$m = 2$

$m = 4$

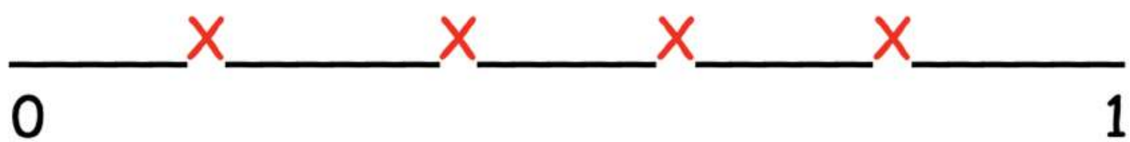


Min of IID Uniforms

If $Y_1, \ldots, Y_m$ are iid $Unif(0,1)$, where do we "expect" the points to end up?

$$E[\min\{Y_1, \ldots, Y_4\}] = \frac{1}{4+1} = \frac{1}{5}$$

$m = 4$



Min of IID Uniforms

If $Y_1, \ldots, Y_m$ are iid $Unif(0,1)$, where do we "expect" the points to end up?

A super duper clever idea

32 5 17 32 14 5 32 32 17

The Distinct Elements Algorithm

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
  val $\leftarrow \infty$
function UPDATE(x)
  val $\leftarrow \min\{\text{val}, \text{hash}(x)\}$
function ESTIMATE()
  return round $\left(\frac{1}{\text{val}} - 1\right)$
for $i = 1, \ldots, N$: **do** ▷ Loop through all stream elements
  update(x_i) ▷ Update our single float variable
return estimate() ▷ An estimate for n, the number of distinct elements.

Distinct Elements Example

```
Stream:  13,   25,   19,   25,   19,   19
```

```
Hashes: 0.51, 0.26, 0.79, 0.26, 0.79, 0.79
```

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
  val $\leftarrow \infty$
function UPDATE(x)
  val $\leftarrow$ min {val, hash(x)}
function ESTIMATE()
  return round $\left(\frac{1}{\text{val}} - 1\right)$
for $i = 1, \ldots, N$: **do** ▷ Loop through all stream elements
  update(x_i) ▷ Update our single float variable
return estimate() ▷ An estimate for n, the number of distinct elements.

```
val = infty
```

Distinct Elements Example

Stream: 13, 25, 19, 25, 19, 19

h

Hashes: 0.51, 0.26, 0.79, 0.26, 0.79, 0.79

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
  val $\leftarrow \infty$
function UPDATE(x)
  val $\leftarrow$ min {val, hash(x)}
function ESTIMATE()
  return round $\left(\frac{1}{\text{val}} - 1\right)$
for $i = 1, \ldots, N$: **do** ▷ Loop through all stream elements
  update(x_i) ▷ Update our single float variable
return estimate() ▷ An estimate for n, the number of distinct elements.

val = infty

Distinct Elements Example

Stream: $\boxed{13}$, 25, 19, 25, 19, 19

h

Hashes: $\boxed{0.51}$, 0.26, 0.79, 0.26, 0.79, 0.79

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
  val $\leftarrow \infty$
function UPDATE(x)
  val $\leftarrow \min\{\text{val}, \text{hash(x)}\}$
function ESTIMATE()
  return round $\left(\frac{1}{\text{val}} - 1\right)$

for $i = 1, \ldots, N$: **do** ▷ Loop through all stream elements
  update(x_i) ▷ Update our single float variable
return estimate() ▷ An estimate for n, the number of distinct elements.

val = 0.51

Distinct Elements Example

```
Stream:  13,   25,   19,   25,   19,   19
                h
Hashes: 0.51, 0.26, 0.79, 0.26, 0.79, 0.79
```

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
  val $\leftarrow \infty$
function UPDATE(x)
  val $\leftarrow \min\{\text{val}, \text{hash}(x)\}$
function ESTIMATE()
  return round $\left(\frac{1}{\text{val}} - 1\right)$
for $i = 1, \ldots, N$: **do** ▷ Loop through all stream elements
  update(x_i) ▷ Update our single float variable
return estimate() ▷ An estimate for n, the number of distinct elements.

val = 0.26

Distinct Elements Example

Stream: 13, 25, $\boxed{19,}$ 25, 19, 19

h

Hashes: 0.51, 0.26, $\boxed{0.79}$, 0.26, 0.79, 0.79

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
  val $\leftarrow \infty$
function UPDATE(x)
  val $\leftarrow$ min {val, hash(x)}
function ESTIMATE()
  return round $\left(\frac{1}{\text{val}} - 1\right)$
for $i = 1, \ldots, N$: **do** ▷ Loop through all stream elements
  update(x_i) ▷ Update our single float variable
return estimate() ▷ An estimate for n, the number of distinct elements.

val = 0.26

Distinct Elements Example

```
Stream:  13,   25,   19,   25,   19,   19
                            h
Hashes: 0.51, 0.26, 0.79, 0.26, 0.79, 0.79
```

Algorithm 2 Distinct Elements Operations

---

function INITIALIZE()

  val $\leftarrow \infty$

function UPDATE(x)

  val $\leftarrow$ min {val, hash(x)}

function ESTIMATE()

  return round $\left(\frac{1}{\text{val}} - 1\right)$

for $i = 1, \ldots, N$: **do** ▹ Loop through all stream elements

  update(x_i) ▹ Update our single float variable

return estimate() ▹ An estimate for n, the number of distinct elements.

---

val = 0.26

Distinct Elements Example

Stream: 13, 25, 19, 25, $\boxed{19}$, 19

h

Hashes: 0.51, 0.26, 0.79, 0.26, $\boxed{0.79}$, 0.79

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
  val $\leftarrow \infty$
function UPDATE(x)
  val $\leftarrow$ min {val, hash(x)}
function ESTIMATE()
  return round $\left(\frac{1}{\text{val}} - 1\right)$

for $i = 1, \ldots, N$: **do** ▷ Loop through all stream elements
  update(x_i) ▷ Update our single float variable
return estimate() ▷ An estimate for n, the number of distinct elements.

val = 0.26

Distinct Elements Example

```
Stream:  13,   25,   19,   25,   19,   19
                                     h
Hashes: 0.51, 0.26, 0.79, 0.26, 0.79, 0.79
```

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
  val $\leftarrow \infty$
function UPDATE(x)
  val $\leftarrow$ min {val, hash(x)}
function ESTIMATE()
  return round $\left(\frac{1}{\text{val}} - 1\right)$

for $i = 1, \ldots, N$: **do** ▹ Loop through all stream elements
  update(x_i) ▹ Update our single float variable
return estimate() ▹ An estimate for n, the number of distinct elements.

val = 0.26

Distinct Elements Example

Stream: 13, 25, 19, 25, 19, $\boxed{19}$

$\downarrow$ h $\downarrow$

Hashes: 0.51, 0.26, 0.79, 0.26, 0.79, $\boxed{0.79}$

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
  val $\leftarrow \infty$
function UPDATE(x)
  val $\leftarrow \min\{\text{val}, \text{hash(x)}\}$
function ESTIMATE()
  return round $\left(\frac{1}{\text{val}} - 1\right)$
for $i = 1, \ldots, N$: **do** ▷ Loop through all stream elements
  update(x_i) ▷ Update our single float variable
return estimate() ▷ An estimate for n, the number of distinct elements.

val = 0.26

Return
round(1/0.26 - 1) =
round(2.846) =
3

DIY: Distinct Elements Example II

Stream: 11, 34, 89, 11, 89, 23

Hashes: 0.5, 0.21, 0.94, 0.5, 0.94, 0.1

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
  val $\leftarrow \infty$
function UPDATE(x)
  val $\leftarrow$ min {val, hash(x)}
function ESTIMATE()
  return round $\left(\frac{1}{\text{val}} - 1\right)$

for $i = 1, \ldots, N$: **do** ▷ Loop through all stream elements
  update(x_i) ▷ Update our single float variable
return estimate() ▷ An estimate for n, the number of distinct elements.

val = 0.1

Return= 9

Summary so far

Problem

How can we reduce the variance?

Coding on pset 6

You will use a hash function $h : \mathcal{U} \rightarrow [0, 1]$
For distinct values in $\mathcal{U}$, the function maps to iid (independent and identically distributed) Unif(0,1) random numbers.

Note that, if you were to feed in two equivalent elements, the function returns the **same** number.

We will implement the hash function for you! Just know that you can consider it an iid uniform continuous random variables for each of the values being hashed.

To do better...

1. we will keep track of K DistElts classes each with its own independent hash function
2. take the mean of our K mins to get a better estimate of the min
3. and then apply the same trick as earlier to give an estimate for the number of distinct elements based on this min that we saw.