The CLT – Recap

**Theorem. (Central Limit Theorem)** \( X_1, \ldots, X_n \) iid with mean \( \mu \) and variance \( \sigma^2 \). Let \( Y_n = \frac{X_1 + \cdots + X_n - n\mu}{\sigma \sqrt{n}} \). Then,

\[
\lim_{n \to \infty} Y_n \to N(0,1)
\]

One main application:

Use Normal Distribution to Approximate \( Y_n \).

No need to understand \( Y_n \).
Example – $Y_n$ is binomial

We understand binomial, so we can see how well approximation works

We flip $n$ independent coins, heads with probability $p = 0.75$.

$X = \#$ heads $\quad \mu = \mathbb{E}(X) = 0.75n \quad \sigma^2 = \text{Var}(X) = p(1 - p)n = 0.1875n$

\[
\mathbb{P}(X \leq 0.7n) \sim \mathcal{N}(\mu, \sigma^2)
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th>exact</th>
<th>$\mathcal{N}(\mu, \sigma^2)$ approx</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.4744072</td>
<td>0.357500327</td>
</tr>
<tr>
<td>20</td>
<td>0.38282735</td>
<td>0.302788308</td>
</tr>
<tr>
<td>50</td>
<td>0.25191886</td>
<td>0.207108089</td>
</tr>
<tr>
<td>100</td>
<td>0.14954105</td>
<td>0.124106539</td>
</tr>
<tr>
<td>200</td>
<td>0.06247223</td>
<td>0.051235217</td>
</tr>
<tr>
<td>1000</td>
<td>0.00019359</td>
<td>0.000130365</td>
</tr>
</tbody>
</table>
Example – Naive Approximation

Fair coin flipped (independently) $40$ times. Probability of $20$ or $21$ heads?

**Exact.**

$$\mathbb{P}(X \in \{20,21\}) = \left[ \binom{40}{20} + \binom{40}{21} \right] \left( \frac{1}{2} \right)^{40} \approx 0.2448$$

**Approx.**

$X = \#$ heads \hspace{1cm} \mu = \mathbb{E}(X) = 0.5n = 20 \hspace{1cm} \sigma^2 = \text{Var}(X) = 0.25n = 10$

$X \sim \mathcal{N}(20, 10)$

$$\mathbb{P}(20 \leq X \leq 21) = \Phi \left( \frac{20 - 20}{\sqrt{10}} \leq \frac{X - 20}{\sqrt{10}} \leq \frac{21 - 20}{\sqrt{10}} \right)$$

$$\approx \Phi \left( 0 \leq \frac{X - 20}{\sqrt{10}} \leq 0.32 \right)$$

$$= \Phi(0.32) - \Phi(0) \approx 0.1241$$
Example – Even Worse Approximation

Fair coin flipped (independently) \(40\) times. Probability of \(20\) heads?

Exact. \[ 
\mathbb{P}(X = 20) = \binom{40}{20} \left( \frac{1}{2} \right)^{40} \approx 0.1254
\]

Approx. \[ 
\mathbb{P}(20 \leq X \leq 20) = 0 \]

\( \sim \mathcal{N}(20, 10) \)
Solution – Continuity Correction

Round to next integer!

To estimate probability that discrete RV lands in (integer) interval \( \{a, \ldots, b\} \), compute probability continuous approximation lands in interval \( [a - \frac{1}{2}, b + \frac{1}{2}] \).
Example – Continuity Correction

Fair coin flipped (independently) 40 times. Probability of 20 or 21 heads?

**Exact.** \( P(X \in \{20, 21\}) = \left[ \binom{40}{20} + \binom{40}{21} \right] \left( \frac{1}{2} \right)^{40} \approx 0.2448 \)

**Approx.** \( X = \# \text{ heads} \quad \mu = \mathbb{E}(X) = 0.5n = 20 \quad \sigma^2 = \text{Var}(X) = 0.25n = 10 \)

\[
P(19.5 \leq X \leq 21.5) = \Phi\left( \frac{19.5 - 20}{\sqrt{10}} \leq \frac{X - 20}{\sqrt{10}} \leq \frac{21.5 - 20}{\sqrt{10}} \right)
\]

\[
\approx \Phi\left( -0.16 \leq \frac{X - 20}{\sqrt{10}} \leq 0.47 \right)
\]

\[
= \Phi(-0.16) - \Phi(0.47) \approx 0.2452
\]
Example – Continuity Correction

Fair coin flipped (independently) 40 times. Probability of 20 heads?

**Exact.** \( \mathbb{P}(X = 20) = 40 \binom{40}{20} \left( \frac{1}{2} \right)^{40} \approx 0.1254 \)

**Approx.** \( \mathbb{P}(19.5 \leq X \leq 21.5) = \Phi \left( \frac{19.5 - 20}{\sqrt{10}} \leq \frac{X - 20}{\sqrt{10}} \leq \frac{20.5 - 20}{\sqrt{10}} \right) \)

\approx \Phi \left( -0.16 \leq \frac{X - 20}{\sqrt{10}} \leq 0.16 \right) \approx 0.1272
Application: Distinct Elements
(code this in Pset 6)
Data mining – Stream Model

- 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).
- 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?
Problem Setup

- 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
Today: Counting distinct elements

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

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!
Other 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.
## Counting distinct elements

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

N = # of IDs in the stream = 11, \( m = \# \) of distinct IDs in the stream = 5

Want to compute number of **distinct** IDs in the stream.

- **Naïve solution:** As the data stream comes in, store all distinct IDs in a hash table.
- Space requirement \( O(m) \), where \( m \) is the number of distinct IDs
- Consider the number of users of youtube, and the number videos on youtube. This is not feasible.
Counting distinct elements

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

Want to compute number of distinct IDs in the stream.
- How to do this without storing all the elements?

Yet another super cool application of probability
Hash function $h: U \rightarrow [0,1]$

Assumption: For distinct values in $U$, the function maps to iid (independent and identically distributed) $\text{Unif}(0,1)$ random numbers.

Important: if you were to feed in two equivalent elements, the function returns the same number.

• So $m$ distinct elements $\rightarrow m$ iid uniform $y_i$’s
Min of IID Uniforms

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

In general, $E[\min(Y_1, \ldots, Y_m)] = \frac{1}{m+1}$

- $m = 1$
  
  $E[\min(Y_1)] = \frac{1}{1+1} = \frac{1}{2}$

- $m = 2$
  
  $E[\min(Y_1, Y_2)] = \frac{1}{2+1} = \frac{1}{3}$

- $m = 4$
  
  $E[\min(Y_1, \ldots, Y_4)] = \frac{1}{4+1} = \frac{1}{5}$
A super duper clever idea

If $Y_1, \ldots, Y_n$ are iid $\text{Unif}(0,1)$, where do we expect the points to end up?

In general, $\mathbb{E}[\min(Y_1, \ldots, Y_m)] = \frac{1}{m+1}$

Idea: $m = \frac{1}{\mathbb{E}[\min(Y_1, \ldots, Y_m)] - 1}$

Let’s keep track of the value $\text{val}$ of $\min$ of hash values, and estimate $m$ as $\text{Round} \left( \frac{1}{\text{val}} - 1 \right)$
The Distinct Elements Algorithm

**Algorithm 2 Distinct Elements Operations**

```plaintext
function INITIALIZE()
    val ← ∞

function UPDATE(x)
    val ← min {val, hash(x)}

function ESTIMATE()
    return round \( \frac{1}{val} - 1 \)

for \( i = 1, \ldots, N \):
    update \( x_i \)

return estimate()
```

- Loop through all stream elements
- Update our single float variable
- An estimate for \( n \), the number of distinct elements.
Distinct Elements Example

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

Hashes:

Algorithm 2 Distinct Elements Operations

```
function INITIALIZE()
    val ← \infty

function UPDATE(x)
    val ← \min \{val, \text{hash}(x)\}

function ESTIMATE()
    return \text{Round}\left(\frac{1}{\text{val}} - 1\right)

for i = 1, \ldots, N: do
    update(x_i)

return estimate()
```

- Loop through all stream elements
- Update our single float variable
- An estimate for $n$, the number of distinct elements.
Distinct Elements Example

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

Hashes: $0.51,$

Algorithm 2 Distinct Elements Operations

```
function INITIALIZE()
    val ← ∞

function UPDATE(x)
    val ← min {val, hash(x)}

function ESTIMATE()
    return round \left( \frac{1}{val} - 1 \right)

for i = 1, \ldots, N: do
    update(x_I)
    update(x_I)

return estimate()
```

$\text{val} = \text{infty}$
Distinct Elements Example

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

Hashes: 0.51,

val = 0.51

---

**Algorithm 2** Distinct Elements Operations

```plaintext
function INITIALIZE()
    val ← ∞

function UPDATE(x)
    val ← min {val, hash(x)}

function ESTIMATE()
    return round \left( \frac{1}{val} - 1 \right)

for i = 1, ..., N: do
    update(x_i)

return estimate()
```

> Loop through all stream elements
> Update our single float variable
> An estimate for $n$, the number of distinct elements.
Distinct Elements Example

Stream: 13, 25, 19, 25, 19, 19
Hashes: 0.51, 0.26,

\[
\text{val} = 0.26
\]

---

**Algorithm 2 Distinct Elements Operations**

```
function INITIALIZE()
    val ← ∞

function UPDATE(x)
    val ← min {val, hash(x)}

function ESTIMATE()
    return round \( \frac{1}{\text{val}} - 1 \)

for i = 1, \ldots, N: do
    update \( x_i \)
    \( \triangleright \) Update our single float variable

return estimate()
\( \triangleright \) Loop through all stream elements
\( \triangleright \) An estimate for \( n \), the number of distinct elements.
```
Distinct Elements Example

Stream: 13, 25, \textcircled{19}, 25, 19, 19
Hashes: 0.51, 0.26, 0.79

Algorithm 2 Distinct Elements Operations

\begin{verbatim}
function INITIALIZE()
  val ← ∞
function UPDATE(x)
  val ← min \{val, hash(x)\}
function ESTIMATE()
  return round \(\frac{1}{\text{val}} - 1\)
for \(i = 1, \ldots, N\): do
  update(\(x_i\)) \> Loop through all stream elements
  \> Update our single float variable
return estimate() \> An estimate for \(n\), the number of distinct elements.
\end{verbatim}

\text{val} = 0.26
Distinct Elements Example

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

val = 0.26

Algorithm 2 Distinct Elements Operations

function initialize()
    val ← ∞

function update(x)
    val ← min {val, hash(x)}

function estimate()
    return round \left(\frac{1}{val} - 1\right)

for i = 1, \ldots, N: do
    update(x_i)

return estimate()  

▷ Loop through all stream elements
▷ Update our single float variable
▷ 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,

Algorithm 2 Distinct Elements Operations

function INITIALIZE()
    val ← ∞

function UPDATE(x)
    val ← min {val, hash(x)}

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

for \( i = 1, \ldots, N \): do
    update(\( x_i \))

return estimate()

- Loop through all stream elements
- Update our single float variable
- An estimate for \( n \), the number of distinct elements.

val = 0.26
Distinct Elements Example

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

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

val = 0.26

Algorithm 2 Distinct Elements Operations

```plaintext
function initialize()
    val ← ∞

function update(x)
    val ← min {val, hash(x)}

function estimate()
    return round \left( \frac{1}{val} - 1 \right)

for i = 1, . . . , N: do
    update(x_i)
end for

return estimate()
```

- Loop through all stream elements
- Update our single float variable
- An estimate for $n$, the number of distinct elements.

val = 0.26
Distinct Elements Example

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

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 ← ∞

function UPDATE(x)
    val ← min {val, hash(x)}

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

for i = 1, \ldots, N: do
    update(x_i)
return estimate()

\text{val} = 0.1
\text{Return} = 9

\text{Loop through all stream elements}
\text{Update our single float variable}
\text{An estimate for } n, \text{ the number of distinct elements.}
Problem

\[ \text{val} = \min(Y_1, \ldots, Y_m) \]

\[ E[\text{val}] = \frac{1}{m+1} \]

Algorithm:

Track \( \text{val} = \min(h(X_1), \ldots, h(X_N)) = \min(Y_1, \ldots, Y_m) \)

estimate \( m = \frac{1}{\text{val} - 1} \)

But, \( \text{val} \) is not \( E[\text{val}] \)! How far is \( \text{val} \) from \( E[\text{val}] \)?

\[ \text{Var}[\text{val}] \approx \frac{1}{(m + 1)^2} \]

\[ \sigma = \frac{1}{m+1} \]
How can we reduce the variance?

Idea: Repetition to reduce variance!

Use \( k \) independent hash functions \( h^1, h^2, \ldots h^k \)

Keep track of \( k \) independent min hash values

\[
val^1 = \min(h^1(x_1), \ldots, h^1(x_N)) = \min(Y^1_1, \ldots, Y^1_m)
\]

\[
val^2 = \min(h^2(x_1), \ldots, h^2(x_N)) = \min(Y^2_1, \ldots, Y^2_m)
\]

\[
\vdots
\]

\[
val^k = \min(h^k(x_1), \ldots, h^k(x_N)) = \min(Y^k_1, \ldots, Y^k_m)
\]

\[
\overline{val} = \frac{1}{k} \sum_i val_i, \quad \text{Estimate } m = \frac{1}{\overline{val}} - 1
\]