Lecture 16: Differential Privacy (Local Model)
Plan
* Discussion from last time.
* Recap
  - Diff Privacy
  - Laplace Mechanism
* Difficulties in practice
* Local Model
* Applications

Logistics
* HW5 due Friday 11/13 at 5pm via Gradescope
* No class on 11/11? Veterans' Day Holiday
* Guest lecture on Monday 11/16 (Emily Stark, Google)
Recap: Differential Privacy

Two Parts

1. Definition of privacy-preserving systems.
   - Strong, precise, robust
   - Difficult to satisfy

2. Mechanisms - Protocols & systems that satisfy this defn.

So far...

\[ D \xrightarrow{O} M \xrightarrow{M} \text{Census} \xrightarrow{M(D)} \text{Reports} \]
Defin: Mechanism $M$ satisfies $\varepsilon$-DP if 

\[ \Pr_m[M(D) \in S] \leq e^\varepsilon \Pr_m[M(D') \in S]. \]

* Typically $\varepsilon$ is small constant e.g. $\varepsilon = 0.1, 1, 5$
* $\varepsilon = 0 \rightarrow$ Perfect Privacy, $\varepsilon = \infty \rightarrow$ no privacy
* If there’s an output you don’t like (S), if $M$
  satisfies $\varepsilon$-DP, $\Pr$ of output occurring increases by only a little ($\varepsilon \cdot S$) if your data included
* $\varepsilon \Rightarrow$ Mechanism $M$ must be randomized (if non-trivial)

Why we like it: post-processing, composition, group privacy...
Recap: Laplace Mechanism

A simple way to achieve $\varepsilon$-DP in certain cases.

E.g. You take a survey of students, asking whether they voted for candidate $a$ or not.

$x_i \in \{0, 1\}$ if student $i$ voted for $a$

Want to publish $S = \sum_{i=1}^{n} x_i$ w/ $\varepsilon$-DP $\mathcal{L}(\varepsilon/\varepsilon)$.

Idea of Laplace Mechanism: Publish $S + \text{noise}$

Smaller $\varepsilon$ (more privacy) $\implies$ more noise

Bigger $\varepsilon$ (less privacy) $\implies$ less noise

Noise comes from Laplace distribution.

Very simple! So easy to implement!

Are we done?....
Difficulties using DP in practice

→ As you release more statistics, effective E \rightarrow \text{BIG (sums up). Very quickly, the privacy guarantee becomes vacuous.}

→ No good way to “reset” privacy budget

→ Non-sensical outputs. E.g. in census, cities w/ negative population.

→ Data consistency: Need marginals to add up, etc.

→ Analyzing complex mechanisms (e.g. ML training) is very difficult.

→ What is the right value of E?

Take away:

DP is one powerful and important definition of privacy. It doesn’t solve all of our problems. It doesn’t always perfectly capture true privacy leakage. But it is the best we have so far.

→ Central party still has all of your data. (Breach, surveillance, …)
Central Model (so far)

“Will you vote for _____ on Tuesday?”

Citizens

- Pollster

Mechanism

Results + structural noise

...problem with this?

Other examples
- Census data
- Google training ML model

Pros: + Easy to implement (?)

+ Fewer changes to existing processes

Cons: - Pollster sees all of your data

≠ No privacy, i.e., pollster/Census/Google
Local Model ("Randomized response")

Idea: Push mechanism to the edge

0 \to x_i + \text{noise}_i
0 \to x_2 + \text{noise}_2
0 \to x_n + \text{noise}_n

\rightarrow \text{Pollster}
\rightarrow \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} \text{noise}_i

\text{e.g., send } \{ x_i \text{ u.p. } p e \leftarrow \text{Classic randomized response}
\{ -x_i \text{ u.p. } 1 - p e

\text{Pros}
+ No central point of privacy failure
+ DP guarantees mean that arbitrary postprocessing ok

\text{Cons}
- More noise \( \propto \sqrt{n} \) hence more (additive error \( \frac{1}{\sqrt{n}} \))
- Cannot set \( \varepsilon \) too small or else noise blows away signal
- Privacy guarantee for user is weaker than under an MPC implementing central model
  \( \text{Pollster still "learns something" about } x_i \)
  ...can guess \( x_i \) w/ non-negl advantage.
Examples of using Local Model

Apple uses it for collecting telemetry data on iOS and MacOS.

Sofware: 2 submissions/user/day

$\Rightarrow \ 3 = 8 \ \text{for each submission}$

$\Rightarrow \ 3 = 16 \ \text{per day} \ldots \text{after 1 week, not clear that system is buying much in terms of privacy}$

Microsoft uses LDP for collecting # of mins that Windows 10 user use each app

$\Rightarrow \ 3 = 0.7 \ \text{every 6 hours}$

Chrome used k use LDP for collecting telemetry data ("Rappor")

$\Rightarrow \ 3 \approx 1 \ldots \text{As far as I know, on its way out}$
A couple of non-obvious issues...

* Detecting rare events (crash affecting 1% users)
  - Noise is $\frac{1}{\sqrt{n}} / \epsilon$.
  - Set $\epsilon = 0.1$, $\sqrt{n} = 2^{10}$ ($n = 2^{20}$)
  - Error is $\leq 2^{10} \leq 1\%$.
  - Hard to distinguish zero from non-zero (small)

- Partial fix: increase # of users...
  - % noise: $\frac{\epsilon}{\sqrt{n}} = \frac{\epsilon}{\epsilon} \to 0$ as $n \to \infty$

* Collecting statistics other than sums.
  - Common one: Heavy hitters
  - Each client $i$ holds a string $x_i \in \{0,1\}^d$
  - Servers want all strings that more than 1% of clients hold
    (e.g., homepage, URL crashed browser, ...)

≠ URL + noise? ×

Lots of really clean approaches to solve this problem.
General Strategy for Computing Heavy Hitters & Other Statistics w/ LDP

\[ \text{Encode} \quad (\text{e.g., Bloom Filter}) \]

\[ 01011010 \]

\[ \text{noise:} \quad 00110100 \]

\[ \text{noisy data} \]

\[ \text{Server} \]

\[ \text{Server augs noisy vectors, decodes to get output.} \]

\[ \text{UK:} \quad 010110 \]

\[ \text{var:} \quad 11010 \]

\[ \text{SM:} \quad 11010 \]

\[ \text{Decode} \quad \{ \text{nytimes.com, ...} \} \]