Privacy in Statistics and Machine Learning — Spring 2021
Lecture 7: The Binary Tree Mechanism

Adam Smith and Jonathan Ullman

1 Query Release

When we set up the problem of reconstruction, we looked at examples where some organization, such as the Census Bureau, holds a sensitive dataset and wants to release a large number of counts or other simple statistics on that data. Often each one of these statistics is a low sensitivity function that can be answered privately with very little noise—for example each count in Figure 1 is a 1-sensitive function. However, the for large numbers of statistics, the sensitivity of the entire table quickly becomes prohibitively large, proportional to the number of statistics.

		Age		
Statistic	**Group**	**Count**	**Median**	**Mean**
1A	Total Population	7	30	38
2A	Female	4	30	33.5
2B	Male	3	30	44
2C	Black or African American	4	51	48.5
2D	White	3	24	24
3A	Single Adults	(D)	(D)	(D)
3B	Married Adults	4	51	54
4A	Black or African American Female	3	36	36.7
4B	Black or African American Male	(D)	(D)	(D)
4C	White Male	(D)	(D)	(D)
4D	White Female	(D)	(D)	(D)
5A	Persons Under 5 Years	(D)	(D)	(D)
5B	Persons Under 18 Years	(D)	(D)	(D)
5C	Persons 64 Years or Over	(D)	(D)	(D)

Note: Married persons must be 15 or over

Figure 1: An example of census tabulations from [GAM19].

The problem of releasing large numbers of counts, often somewhat inscrutably called *linear query release* has been one of the central problems in differential privacy for many years [BCD+07], and has driven many of the research questions and applications.

1.1 The Query Release Problem

We'll start with a dataset $\mathbf{x} = (x_1, \ldots, x_n)$ where each $x_i \in \mathcal{U}$. We're interested in asking *linear queries*, which are statistics of the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} \varphi(x_i) \text{ for } \varphi : \mathcal{U} \to \{0, 1\} \tag{1}$$

As we've seen before, these queries capture statistics like "how many individuals in the dataset are married adults age 64 and over?" We are interested in taking a large set of queries $f_1, \ldots, f_k$ and releasing a table of answers $a_1, \ldots, a_k$ such that

$$\max_{j=1}^{k} \left| a_j - f_j(\mathbf{x}) \right| \leq \alpha n \tag{2}$$

Note that the way we've constructed this set of queries, if we let $F : \mathcal{U}^n \to \mathbb{R}^k$ be the function that takes $\mathbf{x}$ and outputs a vector of answers to every query, then the global sensitivity of F is never more than k, so we can answer every query using Laplace noise with scale $\frac{1}{\varepsilon}k$. By our standard analysis of the Laplace distribution, doing so will guarantee error $\alpha n = O(\frac{1}{\varepsilon}k \log k)$. This is a baseline we can always start with for any query release problem.

However, we can often do considerably better! One example where we've already seen this is *histograms*, which is a set of k statistical queries that we can answer with error $O(\frac{1}{\varepsilon}\log k)$ using the fact that the queries have a special structure that makes their global sensitivity just 2 instead of k. But histograms are by no means the only example, and there are many clever algorithms for releasing large sets of queries, even when their global sensitivity is not small. In this lecture, we're going to focus on one simple, but surprisingly rich example a query release problem, involving *threshold queries*.

2 Answering Interval Queries

So far we've seen "straightforward" approaches to achieving differential privacy based on computing the global sensitivity Δ of some statistic f and adding noise proportional to Δ. In this lecture we'll take a look at an important example of a statistic—*threshold queries*—where we can come up with a more "clever" approach that allows us to add considerably less noise than the global sensitivity. This example is both important in its own right, and also a prelude to many other powerful and surprising differentially private algorithms that we'll see later in the course.

2.1 Interval Queries

Suppose we're given a dataset $\mathbf{x} = (x_1, \ldots, x_n)$ where each user's data is a number $x_i \in \{1, \ldots, D\}$. We'll use the notation $[D] = \{1, \ldots, D\}$ for convenience. For $t \in [D]$, we can define a *threshold query* that asks for the number of users whose data is $x_i \leq t$. The vector of *interval queries* is a function $\Phi : [D]^n \to \mathbb{R}^{\binom{D}{2}}$ that asks for the answer to $f_{s,t}$ for every pair $1 \leq s \leq t \leq D$. Specifically,

$$f_{s,t}(\mathbf{x}) = \#\{i : s \leq x_i \leq t\} \tag{3}$$

$$F(\mathbf{x}) = (f_{s,t}(\mathbf{x}))_{1 \leq s \leq t \leq D} \tag{4}$$

These queries are useful for many applications, and capture natural statistical quantities like the cumulative distribution function and quantiles of the data. The *cumulative distribution function* is simply the function $\Phi : [D] \to \mathbb{R}$ defined by

$$\Phi(t) = \#\{i : x_i \leq t\} \tag{5}$$

and the *q-quantile* of the data is just the number t such that $\Phi(t) = qn$, or $\Phi^{-1}(qn)$.[1] In particular, the *median* is just the $1/2$-quantile of the data, or t such that $\Phi(t) = n/2$. Threshold queries are also called *range queries* in the field of databases.

Baseline: The Laplace Mechanism. Since threshold queries are a set of $\binom{D}{2}$ count statistics, the global ℓ_1-sensitivity of F is $\Delta \leq \binom{D}{2} = O(D^2)$. Thus we can answer these queries by sampling Laplace random variables $Z_{s,t}$ independently, with scale $\lambda = \binom{D}{2}/\varepsilon$, for every $1 \leq s \leq t \leq D$, and returning $a_{s,t} = f_{s,t}(\mathbf{x}) + Z_{s,t}$ for each query. Using our bounds on the Laplace distribution,

$$\mathbb{E}\left(\max_{1 \leq s \leq t \leq D} \left|a_{s,t} - f_{s,t}(\mathbf{x})\right|\right) = O\left(\frac{D^2 \log D}{\varepsilon}\right) \tag{6}$$

[1] Strictly speaking, since the data is discrete, Φ wont be continuous, and we need to redefine the quantiles a bit more generally, but it's not hard to come up with a reasonable way to do so.

and this bound on the sensitivity is tight up to a constant factor.

Exercise 2.1. Prove that the global ℓ_1 sensitivity of F is $\Omega(D^2)$.

A Simple Improvement. Just to see why answering every query with independent Laplace noise is not the best possible algorithm, suppose we only answered the set of D queries $f_{1,t}$ for $1 \le t \le D$. Since there are now only D queries, we can answer each of these queries with expected maximum error $O(\frac{1}{\varepsilon} D \log D)$, which is a big improvement over before. However, now that we have an answer $a_{1,t} \approx f_{1,t}(\mathbf{x})$ for every t, we can get an answer

$$a_{s,t} = a_{1,t} - a_{1,s-1} \approx f_{s,t}(\mathbf{x}) \tag{7}$$

for every query $1 \le s \le t \le D$. Taking the difference between two noisy answers can increase the error by at most a factor of 2, so we still answer all interval queries with error $O(\frac{1}{\varepsilon} D \log D)$! One important thing to note about this approach is that, the overall result is a *correlated* distribution of the noise over all $\binom{D}{2}$ queries. In principle we could have just come up with this complex distribution over $\binom{D}{2}$ from scratch, it's a lot easier to come up with this distribution by add independent noise to the subset of the queries and then obtaining the correlated noise by recombining the answers.

This example may seem almost trivial, but it identifies a crucial strategy that we're going to see both in the binary tree mechanism and later in the course:

Try to answer an easier set of queries and recover the answers to the queries you want!

In this example, the easy set of queries was just a subset of all the queries we were interested in. Although we won't see any example in this lecture, in general the easy set of queries can be completely different from the queries we actually want to answer.

2.2 The Binary Tree Mechanism

Next we're going to see an even better way to reduce the noise for answering interval queries, that reduces the error *exponentially* to $O(\frac{1}{\varepsilon} \log^3 D)$. As before, the idea is to only answer a carefully chosen *subset* of all the interval queries that simultaneously has two properties:

- The sensitivity of answering all the queries in the subset is small.
- We can reconstruct the missing queries by combining a small number of the queries in this subset.

For comparison, in the simple improvement above, the sensitivity was D and the number of queries we had to combine to reconstruct was 2. Now we're going to see an even better subset where the sensitivity drops all the way down to $O(\log D)$ and the number of queries to reconstruct goes up just a bit but is still $O(\log D)$. As we'll see, this balancing act gives a much better overall guarantee on the error.

The resulting mechanism is often called the *binary tree mechanism*, and it was discovered a few different times in different contexts [DNPR10, CSS11].

For convenience, let's assume D is a power of 2 so that $\log_2 D$ is an integer. If this isn't the ase we can just increase D until it's a power of 2, which can increase the size of the domain by a factor of at most 2, and wont change the final error bound significantly. The interval queries we'll include in our subset are going to be all the intervals of length 1, $f_{1,1}, f_{2,2}, \ldots, f_{D,D}$ as well as all consecutive intervals of length 2, $f_{1,2}, f_{3,4}, \ldots, f_{D-1,D}$ as well as all consecutive intervals of length 4, $f_{1,4}, f_{5,8}, \ldots, f_{D-3,D}$ and so on for lengths $1, 2, 4, 8, \ldots, D$ until we have just a single interval $f_{1,D}$. It's much easier to understand the mechanism if we visualize the set of intervals we're choosing as a binary tree, where the integers $1, \ldots, D$

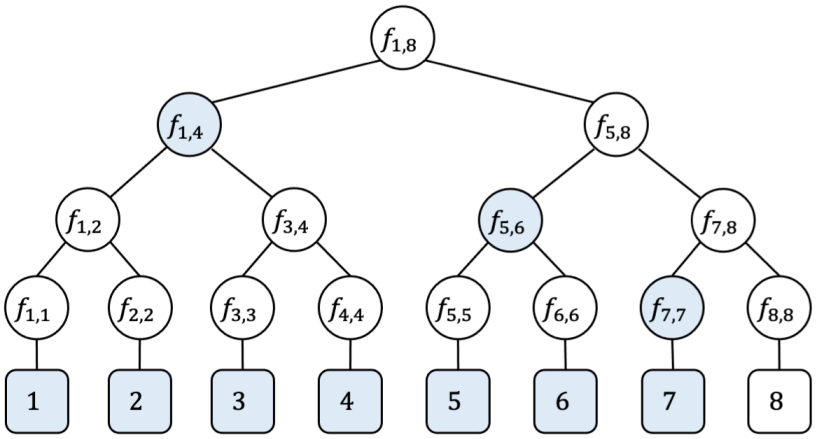

Figure 2: A diagram showing the queries in $\mathcal{T}$ for the domain $D = 8$. The highlighted squares represent the interval $\{1, \ldots, 7\}$ and the highlighted circles show the decomposition of this interval into a union of three intervals in $\mathcal{T}$.

are at the leaves, and every node represents an interval consisting of the union of its two children.[2] To help with the analysis, we'll use $\mathcal{T}$ to denote the set of intervals contained in the tree. Formally,

$$\mathcal{T} = \left\{(u, v) : u = j \cdot 2^{\ell-1} + \text{ and } v = (j+1) \cdot 2^{\ell-1} \text{ for } 1 \le \ell \le \log_2 D \text{ and } 1 \le j \le D/2^{\ell-1}\right\} \tag{8}$$

although it's much easier to look at the picture in Figure 2.

Note that the tree has $\log_2 D$ levels and each level has $D/2^\ell$ nodes, so

$$|\mathcal{T}| = \sum_{\ell=1}^{\log_2 D} \frac{D}{2^{\ell-1}} = 2D - 1$$

Thus, just looking at the size of $\mathcal{T}$ doesn't tell us that these queries have low sensitivity. However, the key property of the binary tree is that if we add a user with data $x_i = t$, then it only changes the value of the intervals in the tree that are *ancestors* of the leaf t, and there are exactly $\log_2 D$ of these! We can formalize this idea with the following claim.

Claim 2.2. *The global sensitivity of* $F^{BT}(\mathbf{x}) = (f_{u,v}(\mathbf{x}))_{(u,v)\in\mathcal{T}}$ *is at most* $2\log_2 D$.

Proof. If we change one datapoint from x_i to x_i' then we reduce the answer to all queries that are ancestors of x_i in the binary tree and increase the answer to all queries that are ancestors of x_i'. Since every leaf in the tree has d ancestors, the total number of queries that can change is at most $2\log_2 D$. Thus the change to the whole vector of queries in ℓ_1 is at most $2\log_2 D$. □

Claim 2.2 means that we can release every query ε-DP by adding Laplace noise with scale $\lambda = \frac{2\log_2 D}{\varepsilon}$ to each coordinate of F^{BT}. Thereby we obtain

$$a_{u,v} = f_{u,v}(\mathbf{x}) + Z_{u,v} \qquad Z_{u,v} \sim \text{Lap}\left(\frac{2}{\varepsilon}\log_2 D\right) \tag{9}$$

[2] In practice we actually get better bounds using a shallower tree with a larger branching factor, but it wont make any difference for the $O(\cdot)$ analysis.

for every $(u,v) \in \mathcal{T}$, where the random variables $Z_{u,v}$ are independent. By our standard analysis of the Laplace distribution we also have

$$\mathbb{E}\left(\max_{(u,v)\in\mathcal{T}} |Z_{u,v}|\right) = \frac{2\log_2(D)(\ln(2D-1)+1)}{\varepsilon} = O\left(\frac{1}{\varepsilon}\log^2 D\right) \tag{10}$$

Recovering the Answers. So far we have managed to answer all of the interval queries contained in $\mathcal{T}$ with expected maximum error $O(\log^2(D)/\varepsilon)$. For example, if we want to answer the threshold query $f_{5,8}$, then we can just look up

$$f_{5,8}(\mathbf{x}) \approx a_{5,8} \tag{11}$$

and get an accurate answer.

But what if we want to obtain, say, $f_{1,7}(\mathbf{x})$, which is the number of individuals whose data x_i is at most 7? This isn't one of the intervals in the tree, but we can reconstruct it by combining three different intervals. The key idea is to write the interval $\{1,\ldots,7\}$ as a union of intervals that are contained in the binary tree, specifically

$$\{1,2,3,4,5,6,7\} = \{1,2,3,4\} \cup \{5,6\} \cup \{7\} \tag{12}$$

Thus, we can reconstruct an approximate answer to $f_7(\mathbf{x})$ from the output of the binary tree as follows

$$f_{1,7}(\mathbf{x}) = f_{1,4}(\mathbf{x}) + f_{5,6}(\mathbf{x}) + f_{7,7}(\mathbf{x}) \approx a_{1,4} + a_{5,6} + a_{7,7} \tag{13}$$

Note that since we are summing up three different noisy answers, we might get three times as much noise, but as long as we don't have to sum too many noisy, we won't increase the noise by too much. Let's see how we can do this for any threshold query $f_{1,t}$. Note that we also want answers to queries $f_{s,t}$, but as we've seen we can obtain those as the difference $f_{s,t} = f_{1,t} - f_{1,s-1}$.

The key claim is that for any threshold query $f_{1,t}$, we can write it as the sum of at most $\log_2 D$ intervals in the set $\mathcal{T}$.

Claim 2.3. *For every $1 \le t \le D$, there exists $\mathcal{S} \subseteq \mathcal{T}$ of size $|\mathcal{S}| \le \log_2 D$ such that*

$$f_{1,t}(\mathbf{x}) = \sum_{(u,v)\in\mathcal{S}} f_{u,v}(\mathbf{x}).$$

Exercise 2.4. Prove Claim 2.3

By the claim, for every interval query $f_{1,t}$, for $1 \le t \le D$, we can compute

$$f_{1,t}(\mathbf{x}) = \sum_{(u,v)\in\mathcal{S}} f_{u,v}(\mathbf{x}) \approx \sum_{(u,v)\in\mathcal{S}} a_{u,v} \tag{14}$$

Now let's see why the error will be small for every query. Let $b_{1,t} \sum_{(u,v)\in\mathcal{S}} a_{u,v}$ be the noisy answer we recover for $f_t(\mathbf{x})$. Then for every threshold $1 \le t \le D$ we have

$$\begin{aligned}
\left|b_{1,t} - f_{1,t}(\mathbf{x})\right| = \left|\sum_{(u,v)\in\mathcal{S}} a_{u,v} - f_{1,t}(\mathbf{x})\right| &= \left|\sum_{(u,v)\in\mathcal{S}} Z_{u,v}\right| \\
&\le \sum_{(u,v)\in\mathcal{S}} |Z_{u,v}| \\
&\le |\mathcal{S}| \cdot \max_{(u,v)\in\mathcal{T}} |Z_{u,v}| \\
&\le \log_2 D \cdot \max_{(u,v)\in\mathcal{T}} |Z_{u,v}|
\end{aligned}$$

Therefore we also have

$$\mathbb{E}\left(\max_{1\le t\le D}\left|b_{1,t}-f_{1.t}(\mathbf{x})\right|\right)\le \mathbb{E}\left(\log_2 D\cdot\max_{(u,v)\in\mathcal{T}}|Z_{u,v}|\right)=O\left(\frac{1}{\varepsilon}\log^3 D\right) \tag{15}$$

where we have used the fact that $\mathbb{E}\left(\max_{(u,v)\in\mathcal{T}}|Z_{s,t}|\right)=O(\frac{1}{\varepsilon}\log^2 D)$. Remember that we can get answers $b_{s,t}$ for $1\le s\le t\le D$ by taking $b_{s,t}=b_{1,t}-b_{1,s-1}$.

We can summarize with the following theorem

Theorem 2.5. *There is an ε-DP mechanism that answers all $\binom{D}{2}$ interval queries over the universe $\{1,\dots,D\}$ such that*

$$\mathbb{E}\left(\max_{1\le s\le t\le D}\left|f_{s,t}(\mathbf{x})-b_{s,t}\right|\right)=O\left(\frac{1}{\varepsilon}\log^3 D\right)$$

A few notes about this theorem are in order:

- Since any interval query $g_{s,t}(\mathbf{x})$ can be written as $f_t(\mathbf{x})-f_s(\mathbf{x})$, we can answer all $\binom{D}{2}$ interval queries with (at most) the error as we get for all the threshold queries.
- The analysis we did was a bit loose, in particular in the step where we wrote $|\sum Z_{u,v}|\le\sum|Z_{u,v}|$. That is, since the signs of each $Z_{u,v}$ are random and independent, the sum should be more like $(\sum Z_{u,v}^2)^{1/2}$ rather than $\sum|Z_{u,v}|$, and the former will typically be much smaller. The overall effect of a more careful analysis would be to get error more like $O(\frac{1}{\varepsilon}\log^{2.5} D)$.
- Our reconstruction procedure is actually not the only way to reconstruct the answers! In fact, the binary tree contains multiple ways of estimating a threshold $f_t(\mathbf{x})$. For example,

  $$f_6=g_{1,4}+g_{5,6}=g_{1,8}-g_{7,8}.$$

  By combining the different estimates we can improve the error significantly in practice [Hon15].
- The true answers to threshold queries $f_{1,t}$ are *monotonic*, meaning $t\le t'$ implies $f_t(\mathbf{x})\le f_{t'}(\mathbf{x})$. However, because of noise, the reconstruction procedure might give answers that are not monotone. However, since differential privacy is closed under post-processing, we can take the answers $a_1,\dots,a_D$ and replace them with a new set of answers $\tilde{a}_1,\dots,\tilde{a}_D$ that are monotone and are as close as possible to $a_1,\dots,a_D$. Not only will this result in a set of plausibly correct answers, it will actually reduce the overall error significantly! This phenomenon arises in many settings, and was (perhaps?) first explicitly studied in [HRMS10].

The Domain Size. One other thing that deserves mention is the role of the domain size D. Often D is not actually given to us, but rather something we choose. For example, if the data consists of arbitrary bounded real numbers $x_i\in[0,B]$ then we might choose to *discretize* the data by rounding it to the nearest integer value in $\{0,1,\dots,B\}$, so that we can apply the binary tree mechanism with $D=B+1$. The rounding will also introduce some error, and this error may or may not be significant depending on the nature of the data itself. For example, if most of x_i lie between 10 and 11, then rounding will lose most of the information about the dataset. Thus, perhaps we will choose to discretize to multiples of some small parameter γ, in which case we'll have $D=B/\gamma+1$, which will reduce the error from rounding but also increase the error from the binary tree mechanism. Finding the optimal way to discretize data to apply the binary tree mechanism is tricky, and cannot be done without some prior knowledge of what the data looks like.

A natural question is whether we really have to choose D? Perhaps we can just use a domain of all real numbers, or take D to be so large that it captures every number our computer can represent? The answer turns out to be quite complex. There are many more algorithms for threshold queries with improved error, notably [DNRR15, BNS13]. These best of these gives a strange looking error bound of about $\frac{1}{\varepsilon}(\log^* D)^{3/2}$, although these algorithms are not currently practical. Given that the iterated logarithm function[3] $\log^* D$ grows with D so slowly, you would be tempted to think that surely there is a solution with error independent of D, or even one that works for data over the infinite domain of all real numbers. However, you're be wrong [BNSV15]! Actually any algorithm for answering interval queries requires error $\Omega(\log^* D)$.

One final thought is that this is probably the most "clever" algorithm that we've seen so far, and it's worth emphasizing that there are many interesting differentially private algorithms, even for very simple looking problems, and surely many more yet to be discovered!

Additional Reading and Watching

-

References

[BCD+07] Boaz Barak, Kamalika Chaudhuri, Cynthia Dwork, Satyen Kale, Frank McSherry, and Kunal Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *Proceedings of the 26th Annual ACM Symposium on Principles of Database Systems*, PODS '07, 2007. ACM.

[BLR08] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to non-interactive database privacy. In *Annual ACM Symposium on the Theory of Computing*, STOC '08, 2008.

[BNS13] Amos Beimel, Kobbi Nissim, and Uri Stemmer. Private learning and sanitization: Pure vs. approximate differential privacy. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, RANDOM-APPROX '13. Springer, 2013.

[BNSV15] Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil Vadhan. Differentially private release and learning of threshold functions. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, 2015.

[CSS11] T-H Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Transactions on Information and System Security (TISSEC)*, 14(3):26, 2011.

[DNPR10] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Symposium on Theory of Computing (STOC)*. ACM, 2010.

[DNRR15] Cynthia Dwork, Moni Naor, Omer Reingold, and Guy N Rothblum. Pure differential privacy for rectangle queries via private partitions. In *International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '15. Springer, 2015.

[3] The function $\log^* D$ is the *Iterated logarithm function*, and is defined by the recurrence $\log^*(2^D) = \log^*(D) + 1$. In more operational terms, it's the number of times you hit the log button on your calculator, starting from T, before the answer is smaller than 1. It technically goes to ∞ as $D \to \infty$, but for any number you can dream up, it's at most 6.

[GAM19] Simson Garfinkel, John M Abowd, and Christian Martindale. Understanding database reconstruction attacks on public data. *Communications of the ACM*, 62(3):46–53, 2019.

[Hon15] James Honaker. Efficient use of differentially private binary trees. 2015. https://hona.kr/papers/files/privatetrees.pdf.

[HRMS10] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proceedings of the VLDB Endowment*, 3(1), 2010. https://arxiv.org/abs/0904.0942.