CS 860: Algorithms for Private Data Analysis Fall 2020

Lecture 8 — Private Multiplicative Weights

Prof. Gautam Kamath *Scribe: Gautam Kamath*

We describe the setting of linear queries. These are very similar to counting or subset queries which we have discussed before, but with coefficients in the range $[0, 1]$, rather than $\{0, 1\}$. In this setting, we work on a data universe $\mathcal{X} = \{s_1, \ldots, s_{|\mathcal{X}|}\}$. A linear query is a function $q : \mathcal{X} \to [0, 1]$, and we overload the notation to also refer to the function applied to a dataset: letting $X \in \mathcal{X}^n$, $q(X) = \frac{1}{n} \sum_{i \in [n]} q(X_i)$. The core question of answering linear queries is the following: suppose we have a set $\mathcal{Q}$ of linear queries. How large does a dataset have to be in order to answer all of them up to accuracy α under differential privacy? Specifically, we want an algorithm $M : \mathcal{X}^n \to [0, 1]^{|\mathcal{Q}|}$ such that $|M_j(X) - q_j(X)| \leq \alpha$, and M must be differentially private (either pure or approximate).

What do we know about this problem already? The simplest way to solve this is just to use the Laplace mechanism, $M_j(X) = q_j(X) + \text{Lap}(|\mathcal{Q}|/\varepsilon n)$. By basic composition, the overall result will be ε-differentially private. Furthermore, using a tail bound on Laplace random variables and the union bound, it is possible to show that we need $n = O\left(\frac{|\mathcal{Q}| \log |\mathcal{Q}|}{\alpha\varepsilon}\right)$ to attain error α under ε-DP. It is actually possible to show that this this logarithmic factor can be removed, and that we only need $n = O\left(\frac{|\mathcal{Q}|}{\alpha\varepsilon}\right)$ [SU17], but this is outside the scope of this class. If we do a similar argument with the Gaussian mechanism, we get a $n = O\left(\frac{\sqrt{|\mathcal{Q}| \log |\mathcal{Q}| \log(1/\delta)}}{\alpha\varepsilon}\right)$ sample complexity under (ε, δ)-DP. Once again, the logarithmic factors are almost entirely removable [SU17, GZ20] but this is again beyond the scope of this class.[1]

The downside is, both of these bounds are polynomial in $|\mathcal{Q}|$ – is it possible to do better? There is a simple histogram-based approach for this problem, which achieves sample complexity $n = O\left(\frac{\sqrt{|\mathcal{X}| \log \mathcal{Q}}}{\alpha\varepsilon}\right)$ under ε-DP, described in Theorem 2.9 of [Vad17]. This is great when we have a lot of queries over a small domain. But the downside is that it now pays polynomially in the size of the domain, which could potentially be very large. Suppose each individual in the dataset has d binary features – in this case, the size of the domain is $|\mathcal{X}| = 2^d$. Is there an algorithm which pays poly-logarithmically (at worst) in both parameters?

In today's class, we will see the celebrated private multiplicative weights algorithm, introduced by Hardt and Rothblum [HR10], and refined by others [GLM+10, GRU12, HLM12].

Theorem 1. *The Private Multiplicative Weights algorithm can answer a set $\mathcal{Q}$ of linear queries on a database of size n to accuracy α under (ε, δ)-DP, given $n = \tilde{O}\left(\frac{\log |\mathcal{Q}| \sqrt{\log |\mathcal{X}| \log(1/\delta)}}{\alpha^2 \varepsilon}\right)$ datapoints.*

This method fits into the very powerful multiplicative weights framework [AHK12], which sees numerous applications across computer science. We cover the non-private method first, and then describe the application to the differentially private setting. The running time will be $\tilde{O}(|\mathcal{Q}| n |\mathcal{X}| / \alpha^2)$ – this is polynomial time in all parameters, which is not great, but also unavoidable based on standard cryptographic assumptions [UV11].

[1] Though if you can remove them entirely, you are eligible to win a free all-you-can-eat sushi dinner [SU19].

We comment that, a previous work presents the SmallDB algorithm [BLR13], which has a sample complexity of $n = \tilde{O}\left(\frac{\log|\mathcal{Q}|\log|\mathcal{X}|}{\alpha^3\varepsilon}\right)$ under ε-differential privacy, but the running time is $|\mathcal{X}|^{O(\log|\mathcal{Q}|/\alpha^2)}$ which is rather impractical, so we will not cover it. It's recommended reading as a nice application of the exponential mechanism.

(Non-Private) Multiplicative Weights

Let's start quite distant from differential privacy and answering linear queries.

A Perfect Expert

Suppose you have a set of N (so-called) experts. They all claim to have arcane knowledge which allows them to predict the future. All of them are lying – except for one. Naturally, you would like to use their incredible power to choose how to invest in the stock market. Every day t starts by each expert giving you a prediction p_i^t: either U for up or D for down. Based on this advice, you have to somehow decide upon your own prediction for that day: U or D. The true signal for the day s^t is then revealed: if it doesn't match your personal prediction, then you make a *mistake*. Again, we assume (for now) that there is one expert (whose identity is not known beforehand) whose prediction always matches the true signal for the day. All other predictions may be arbitrary. This continues for T days, and the goal is to make as few mistakes as possible.

How well can we do? It turns out pretty well – there's an algorithm which makes at most $\log N$ mistakes.

Claim 2. *There is an algorithm that always makes at most* $\log N$ *mistakes.*

The algorithm is simple: in a sentence, every day we predict in concordance with the majority of experts, and at the end of the day we eliminate everyone who was wrong.

Algorithm 1: An algorithm with a perfect expert

Set $S^1 = [N]$
for $t = 1$ *to* T **do**
  Let $S_U^t = \{i : p_i^t = U\}$ be the set of experts who picked U, and similarly $S_D^t = \{i : p_i^t = D\}$
  If $|S_U^t| > |S_D^t|$ then predict U, otherwise predict D
  Set $S^{t+1} = S_{s^t}^t$
end

The claim is that this algorithm makes at most $\log N$ mistakes. This is not hard to see by the following property: either half the experts are right and we don't make a mistake, or half the experts are wrong and they get removed. More precisely, if we make a mistake at step t, then $|S^{t+1}| \leq |S^t|/2$. Since we start with N experts, there can only be $\log N$ such halvings. Note that we will never eliminate the last expert, who is always correct.

When we run this strategy, we make at most $\log N$ mistakes. On the other hand, the best expert made 0 mistakes. Accordingly, this is called our *regret*: how much worse our performance was, in comparison to the best expert (in hindsight).

A Best Expert

This isn't the most realistic scenario – no one can truly predict the future. We'll assume that no there's no perfect expert, but our goal is relaxed as well: we're just trying to perform competitively with the best expert, who makes OPT mistakes. It's easy to see that the exact same strategy won't work. If the best expert is the only one to err on the first round, they could be eliminated immediately, leaving everyone else to answer incorrectly for all future rounds. As a result, we will choose a *softer* strategy: rather than eliminating experts who make a mistake, we just trust their opinion less. We start with each expert having a weight 1, but each time they err, we divide their weight by 2. When we make predictions, we go along with the weighted majority.

Algorithm 2: Weighted Majority Algorithm

Set $w_i^1 = 1$ for all $i \in [N]$
for $t = 1$ *to* T **do**
  Let $W_U^t = \sum_{i:p_i^t=U} w_i^t$ be the weight of experts who picked U, and similarly $W_D^t = \sum_{i:p_i^t=D} w_i^t$
  If $W_U^t > W_D^t$ then predict U, otherwise predict D
  For all i such that $p_i^t \neq s^t$, set $w_i^{t+1} = \frac{1}{2} w_i^t$
end

This is slightly harder to analyze, but still not bad: we will arrive at the following guarantee.

Claim 3. *There is an algorithm that makes at most* $2.4(OPT + \log N)$ *mistakes.*

Observe that we are in some sense competitive with the best expert. That said, it's still not perfect: before, we had an additive overhead over the best expert (the *regret*) of $\log N$. This time, the difference between the number of mistakes and the number made by the best expert is $1.4OPT + 2.4 \log N$. We will later see how to get rid of this OPT term in the regret, but for now let's analyze this algorithm.

Let W^T be the total weight at the end of the process, time T: $W^T = \sum_i w_i^T$. We will upper and lower bound this quantity to relate OPT and the number of mistakes our algorithm makes, which we denote M. To get a lower bound on the weight: the best expert makes at most OPT mistakes – thus, their weight remains at least $(1/2)^{OPT}$ in the worst case, which is a lower bound on the total weight: $(1/2)^{OPT} \leq W^T$. On the other hand, we know that every time the algorithm makes a mistake, the total weight drops by a factor of 3/4: this is because at least half the total weight corresponds to experts who made a mistake, and their weight is divided by 2. Since the total weight at the start was N, this gives us the following upper bound: $W^T \leq N(3/4)^M$.

At this point, we just combine the upper and lower bounds and use algebra: $(1/2)^{OPT} \leq N(3/4)^M$. Rearranging, we get $(4/3)^M \leq N2^{OPT}$. Taking the log of both sides gives $M \leq \frac{OPT + \log N}{\log(4/3)} \leq 2.4(OPT + \log N)$, as desired.

The Multiplicative Weights Algorithm

The previous approaches give us some of the key ideas we need: multiplicatively reward or penalize experts based on whether they're right or wrong. We'll now generalize and strengthen this core algorithmic idea to achieve the polynomial weights algorithm. Before, each expert was forced to

choose one of two actions – now, each can have their own action. Correspondingly, each expert will experience their own loss at each time step, which is now in $[-1, 1]$ rather than being a binary "right" or "wrong."

We have a sequence of T rounds. In each round, the following process occurs:

- The algorithm first chooses some expert $i^t \in [N]$.
- Every expert experiences some loss ℓ_i^t. The algorithm experiences a loss ℓ_A^t equal to $\ell_{i^t}^t$.

The algorithm's total loss is $L_A^T = \sum_{t=1}^T \ell_A^t$, and expert i's total loss is $L_i^T = \sum_{t=1}^T \ell_i^t$ The goal is to be competitive with the best expert in hindsight, and minimize the regret: $L_A^T - \min_i L_i^T$.

The algorithm for this case is similar to the weighted majority algorithm. The main differences are that it selects an expert randomly, and it rescales weights in a more sophisticated way that takes into account the fact that losses are no longer binary.

Algorithm 3: Polynomial Weights Algorithm

Set $w_i^1 = 1$ for all $i \in [N]$
for $t = 1$ *to* T **do**
  Let $W^t = \sum_{i=1}^N w_i^t$
  Select expert i with probability w_i^t / W^t
  Update $w_i^{t+1} = w_i^t(1 - \gamma \ell_i^t)$, where γ is some parameter to be set.
end

Since this algorithm is randomized, we no longer give a worst-case bound on the regret, we now give one in *expectation*. The guarantees of this algorithm are generally stated in the following form:

Theorem 4. *For an arbitrary sequence of losses, and any expert i,*

$$\mathbf{E}[L_A^T] \leq L_i^T + 2\sqrt{T \ln N}$$

In particular, this holds for the best expert.

Note that this is directly competitive with the best strategy in hindsight, and the regret is minimal: if there are T rounds, it is on the order of $\sqrt{T}$, making it generally a lower order term.

However, we choose to phrase the guarantees slightly differently – rather than thinking about the expected loss of a randomly picked expert, we will think of the loss achieved by a *distribution* over experts. This will be useful for later applications, in which there may not exist any single expert who is good against all loss functions that the adversary might throw at them. Consider a game of rock-paper-scissors: if these three options are your experts, then the only way to do well against any strategy from your opponent is with randomization.

We rephrase the problem using this new perspective. In each of the T rounds:

- The algorithm first chooses a *distribution* p^t over $[N]$.
- The algorithm experiences a loss ℓ_A^t equal to $\ell^t \cdot p^t$.

Note that this can be seen as a generalization of the previous case, as before the adversary could only choose a point mass for p^t. But they are both equivalent if you consider the expected loss of the process.

With this phrasing, we rewrite the algorithm above:

Algorithm 4: Polynomial Weights Algorithm - Distributional Phrasing

Set $w_i^1 = 1$ for all $i \in [N]$
for $t = 1$ *to* T **do**
  Let $W^t = \sum_{i=1}^N w_i^t$
  Select p^t to have $p_i^t = w_i^t / W^t$
  Update $w_i^{t+1} = w_i^t(1 - \gamma \ell_i^t)$, where γ is some parameter to be set.
end

We can give the following guarantee for this algorithm:

Theorem 5. *For an arbitrary sequence of loss functions:*

$$\sum_{t=1}^T \ell^t \cdot p^t \leq \sum_{t=1}^T \ell^t \cdot p + 2\sqrt{T \ln N},$$

where p is any fixed distribution over $[N]$.

Proof. As before, we bound the total weight $W^t = \sum_{i=1}^N w_i^t$ at each stage in two ways. We start with an upper bound, by observing how the total mass decays over time. By inspecting the update rule, we see

$$W^{t+1} = \sum_{i=1}^N w_i^t(1 - \gamma \ell_i^t) = W^t(1 - \gamma \ell^t \cdot p^t)$$

Unrolling this, and using the fact that $W^1 = N$,

$$W^{T+1} = N \prod_{t=1}^T (1 - \gamma \ell^t \cdot p^t).$$

We then take the logarithm of both sides, and use the fact that $\ln(1 - x) \leq -x$.

$$\begin{aligned} \ln W^{T+1} &= \ln N + \sum_{t=1}^T \ln(1 - \gamma \ell^t \cdot p^t) \\ &\leq \ln N - \gamma \sum_{t=1}^T \ell^t \cdot p^t \end{aligned}$$

On the other hand, we can also prove a lower bound, by again considering the weight of any specific

expert i. This time, we will keep in mind the inequality $\ln(1-x) \geq -x - x^2$ for $-1/2 < x < 1/2$.

$$\begin{aligned}
\ln W^{T+1} &\geq \ln w_i^{T+1} \\
&= \sum_{t=1}^{T} \ln(1 - \gamma \ell_i^t) \\
&\geq -\sum_{t=1}^{T} \gamma \ell_i^t - \sum_{t=1}^{T} (\gamma \ell_i^t)^2 \\
&\geq -\gamma L_i^T - \gamma^2 T.
\end{aligned}$$

Since this holds for any particular expert i, it also holds for any fixed distribution p over these experts:

$$-\gamma p \cdot L_i^T - \gamma^2 T = -\gamma \sum_{t=1}^{T} p \cdot \ell_i^t - \gamma^2 T \leq \ln W^{T+1}.$$

We combine this with our previous upper bound on $\ln W^{T+1}$:

$$-\gamma \sum_{t=1}^{T} p \cdot \ell_i^t - \gamma^2 T \leq \ln N - \gamma \sum_{t=1}^{T} \ell_t \cdot p^t.$$

Rearranging and rescaling by γ gives the following:

$$\sum_{t=1}^{T} \ell^t \cdot p^t \leq \sum_{t=1}^{T} \ell^t \cdot p + \gamma T + \frac{\ln N}{\gamma}.$$

Choosing $\gamma = \sqrt{\ln N / T}$ completes the proof. □

Multiplicative Weights for Queries

We now have some key ideas in place, and it's time to bring this back to our problem of answering linear queries. We will fit it into the multiplicative weights framework by phrasing the problem in the right way. For now, we will focus on the non-private setting, which will seem pointlessly indirect. But the translation into the differentially private setting will be much easier as a result.

Recall we have a set of queries $\mathcal{Q}$ where a query $q(X) = \frac{1}{n} \sum_{k \in [n]} q(X_k)$. However, we can rewrite this by grouping points by "type," moving from a dataset into a histogram representation. For each $i \in \mathcal{X}$, we define p_i to be $\frac{|\{k : X_k = i\}|}{n}$: the fraction of the points which are equal to i. This is sometimes called the empirical distribution over the dataset, and we will think of it as a distribution. With this notation in place, we can write query $q(X)$ as $\sum_{i \in \mathcal{X}} q(i) p_i$. We will use the shorthand $\langle q, p \rangle$ to represent this expression.

Suppose we wanted to answer these type of queries in the same type of online setting as above. We are in fact not in the online setting (in that all the queries are known in advance), but this perspective will be useful nonetheless. Specifically, we imagine the following happens repeatedly over T rounds:

- An adversary picks a query $q^t \in \mathcal{Q}$.

- The algorithm selects a distribution p^t over $\mathcal{X}$.

We would like the algorithm to have a low regret in comparison to the best distribution in retrospect: our goal will be to minimize $\sum_{t=1}^{T} |\langle q^t, p^t\rangle - \langle q^t, p\rangle|$. We will analyze this using the multiplicative weights framework above.

This might seem obtuse for two reasons. First, in our setting, the algorithm knows the dataset, and thus it knows p: it could just select $p^t = p$ for all t and just be done with it! However, this is clearly not private – in some sense, the multiplicative weights framework lets us converge to this p[2] by only performing queries of the sort $\langle q, p\rangle$, which is much easier to privatize. Second, it is not clear why we care about the performance of our guesses in this sequential setting. Indeed, in the end we just want to produce a single $\hat{p}$, which performs well on all queries $q \in \mathcal{Q}$ at once. But the strategy will be to argue that, if it makes a lot of mistakes as we feed it queries like this, it will have "used up" all of the budget in the regret bound. Therefore, if we gave it any other query in $\mathcal{Q}$, it couldn't make a mistake without contradicting the guarantees of the multiplicative weights algorithm.

We will run the polynomial weights algorithm (Algorithm 4) in this setting. Our "experts" will be the elements of $\mathcal{X}$. It only remains to specify the losses which the adversary chooses. We will derive this by looking at our objective function:

$$f(p^t) = |\langle q^t, p^t\rangle - \langle q^t, p\rangle|$$

Note that f is a convex function. Exploiting this property tells us that

$$f(p^t) + \nabla f(p^t) \cdot (p - p^t) \leq f(p).$$

Rearranging, and using the fact that $f(p) = 0$:

$$f(p^t) \leq (\nabla f(p^t)) \cdot (p^t - p).$$

Summing over all t, we get

$$\sum_{t=1}^{T} |\langle q^t, p^t\rangle - \langle q^t, p\rangle| \leq \sum_{t=1}^{T} (\nabla f(p^t)) \cdot (p^t - p)$$

The last term should look familiar. Inspect Theorem 5 – it gives control over the inner product of the loss vector and the difference between the algorithm's choice p^t and the best distribution in hindsight p. This guides us on how we should choose the losses to feed into the multiplicative weights algorithm: $\ell^t = \nabla |\langle q^t, p^t\rangle - \langle q^t, p\rangle|$. Specifically, this gives a vector $\in [-1, 1]^{|\mathcal{X}|}$ where $\ell^t_i = q^t(i)$ if $\langle q^t, p^t\rangle \geq \langle q^t, p\rangle$, and $-q^t(i)$ otherwise.

Thus, if we run the multiplicative weights algorithm with loss function $\ell^t = \text{sign}(\langle q^t, p^t - p\rangle) q^t$, we get

$$\sum_{t=1}^{T} |\langle q^t, p^t\rangle - \langle q^t, p\rangle| \leq \sum_{t=1}^{T} (\nabla f(p^t)) \cdot (p^t - p) \leq 2\sqrt{T \ln |\mathcal{X}|},$$

which we state in the following theorem.

[2] Or at least converge in terms of the answers it gives to queries in $\mathcal{Q}$

Theorem 6. *Given an arbitrary sequence of T queries, we have the following regret bound:*

$$\sum_{t=1}^{T} |\langle q^t, p^t \rangle - \langle q^t, p \rangle| \leq 2\sqrt{T \ln |\mathcal{X}|}.$$

While this gives a regret bound, it is convenient for us to convert this to a "mistake bound." This will be useful since it will allow us to convert from guarantees about the performance of the sequence p^t to performance of the final distribution p^{T+1}. We define a mistake as a time t when $|\langle q^t, p^t \rangle - \langle q^t, p \rangle| > \alpha$. Suppose for all T time steps, the adversary chooses a function $q^t \in \mathcal{Q}$ which causes the algorithm to make a mistake: specifically, at every time t, $|q^t(p^t) - q^t(p)| \geq \alpha$. This gives a lower bound on the regret, of $T\alpha$: there are T time steps, and each incurs a regret of at least α. However, the above theorem guarantees that the regret is upper bounded by $2\sqrt{T \ln |\mathcal{X}|}$. Combining these two bounds on the regret says that $T \leq \frac{4 \ln |\mathcal{X}|}{\alpha^2}$. This implies that the adversary can only choose queries from $\mathcal{Q}$ which result in a mistake up to $\frac{4 \ln |\mathcal{X}|}{\alpha^2}$ times. After this, no $q \in \mathcal{Q}$ would cause a mistake, meaning that it can answer all of them with error $\leq \alpha$.

This gives an approach for arriving at a distribution $\hat{p} = p^{T+1}$ which can simultaneously answer all queries accurately. Specifically, we act as the adversary, repeatedly choosing queries $q^t \in \mathcal{Q}$ which cause the algorithm to make a mistake. Again, it seems very roundabout, since we could have just picked $\hat{p} = p$, but we will see that all of the steps in this algorithm are amenable to privacy. We summarize in Algorithm 5 and Corollary 7.

Algorithm 5: A non-private multiplicative weights algorithm for answering linear queries

Set $p_i^1 = 1/|\mathcal{X}|$ for all $i \in \mathcal{X}$
for $t = 1$ *to* T **do**
  Choose a query $q^t \in \mathcal{Q}$ such that $|\langle q^t, p^t \rangle - \langle q^t, p \rangle| \geq \alpha$
  Compute $s = \text{sign}(\langle q^t, p^t \rangle - \langle q^t, p \rangle)$
  Update $p_i^{t+1} \propto p_i^t \left(1 - s\left(\sqrt{\frac{\ln |\mathcal{X}|}{T}}\right) q_i^t\right)$
end

Corollary 7. *Algorithm 5 can only run for at most $4 \ln |\mathcal{X}|/\alpha^2$ timesteps, until it is no longer able to select a $q \in \mathcal{Q}$ which causes a mistake. Consequently, we have that p^{T+1} correctly answers all queries $q \in \mathcal{Q}$ to accuracy $\leq \alpha$.*

Private Multiplicative Weights

Private multiplicative weights is simply a translation of Algorithm 5 to the differentially private setting. There are two steps in each iteration which depend on the dataset: selecting a query which causes the algorithm to err, and checking how much error this query incurs (and the associated sign). The former will be done by the exponential mechanism, and the latter via the Laplace

mechanism. Algorithm 6 describes the procedure more precisely.

Algorithm 6: Private multiplicative weights

Set $p_i^1 = 1/|\mathcal{X}|$ for all $i \in \mathcal{X}$
for $t = 1$ *to* T **do**
  Use the exponential mechanism to choose a query $q^t \in \mathcal{Q}$ with ε_0-DP, using score function $|\langle q^t, p^t\rangle - \langle q^t, p\rangle|$
  Compute $y^t = \langle q^t, p^t\rangle - \langle q^t, p\rangle + \text{Laplace}(1/\varepsilon_0 n)$
  If $|y^t| \leq 2\alpha$, return p^t
  Otherwise, compute $s = \text{sign}(y^t)$
  Update $p_i^{t+1} \propto p_i^t \left(1 - s\sqrt{\frac{\ln|\mathcal{X}|}{T}} q_i^t\right)$
end

Let us first analyze the privacy of this approach. We can see that each iteration is $2\varepsilon_0$-DP. By basic composition, the overall algorithm is $2\varepsilon T$-DP. However, using advanced composition, this will be $(O(\varepsilon_0\sqrt{T\log(1/\delta)}), \delta))$-DP. Setting $\varepsilon_0 = O(\varepsilon/\sqrt{T\log(1/\delta)})$ gives (ε, δ)-DP.

It remains only to reason about the accuracy. The only places that error is incurred due to privacy are in the application of the exponential and Laplace mechanisms. If both of these incur error at most, say $\alpha/100$, then we will be successfully executing a strategy like in Algorithm 5. This is because the exponential mechanism will choose a query which incurs regret which is within an additive $\alpha/100$ of the large possible. Similarly, because of the Laplace mechanism, we incur error at most $\alpha/100$ when measuring how much error it incurs (potentially choosing to return if the answer is small). Combining these appropriately gives that we will always ask a query which incurs $\Omega(\alpha)$ regret, and we can prove a similar result as Corollary 7 to bound $T \leq O(\log|\mathcal{X}|/\alpha^2)$.

Analyzing the accuracy of both of these is standard.[3] The Laplace mechanism incurs error $O(1/\varepsilon_0 n)$ for each invocation. Upper bounding this by $\alpha/100$ gives the requirement that $n \geq \Omega(1/\alpha\varepsilon_0) = \Omega\left(\sqrt{\log|\mathcal{X}|\log(1/\delta)}/\alpha^2\varepsilon\right)$. Similarly, the exponential mechanism incurs error $O\left(\frac{\log|\mathcal{Q}|}{n\varepsilon_0}\right)$, and bounding this by $\alpha/100$ requires that $n \geq \Omega\left(\log|\mathcal{Q}|\sqrt{\log|\mathcal{X}|\log(1/\delta)}/\alpha^2\varepsilon\right)$. These bounds give Theorem 1. They actually give something slightly stronger: a *synthetic dataset* which can answer all the queries in $\mathcal{Q}$.

Notes

The first half of these lecture notes is based heavily off of notes of Aaron Roth [Rot17]. Thanks to Sasho Nikolov and Aaron Roth for advice when preparing this lecture.

References

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: A meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[3] We analyze both of these for only a single round, but a union bound can give the same guarantee for all rounds at the cost of a logarithmic factor in T.

[BLR13] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to noninteractive database privacy. *Journal of the ACM*, 60(2):1–25, 2013.

[GLM+10] Anupam Gupta, Katrina Ligett, Frank McSherry, Aaron Roth, and Kunal Talwar. Differentially private combinatorial optimization. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1106–1125, Philadelphia, PA, USA, 2010. SIAM.

[GRU12] Anupam Gupta, Aaron Roth, and Jonathan Ullman. Iterative constructions and private data release. In *Proceedings of the 9th Conference on Theory of Cryptography*, TCC '12, pages 339–356, Berlin, Heidelberg, 2012. Springer.

[GZ20] Arun Ganesh and Jiazheng Zhao. Privately answering counting queries with generalized gaussian mechanisms. https://people.eecs.berkeley.edu/~arunganesh/papers/generalizedgaussians.pdf, 2020.

[HLM12] Moritz Hardt, Katrina Ligett, and Frank McSherry. A simple and practical algorithm for differentially private data release. In *Advances in Neural Information Processing Systems 25*, NIPS '12, pages 2339–2347. Curran Associates, Inc., 2012.

[HR10] Moritz Hardt and Guy N Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, FOCS '10, pages 61–70, Washington, DC, USA, 2010. IEEE Computer Society.

[Rot17] Aaron Roth. Nets 412: Algorithmic game theory – lecture 5 and 6. https://www.cis.upenn.edu/~aaroth/courses/slides/agt20/lect05.pdf, 2017.

[SU17] Thomas Steinke and Jonathan Ullman. Between pure and approximate differential privacy. *The Journal of Privacy and Confidentiality*, 7(2):3–22, 2017.

[SU19] Thomas Steinke and Jonathan Ullman. Open problem - avoiding the union bound for multiple queries. https://differentialprivacy.org/open-problem-avoid-union/, April 2019.

[UV11] Jonathan Ullman and Salil Vadhan. PCPs and the hardness of generating private synthetic data. In *Proceedings of the 8th Conference on Theory of Cryptography*, TCC '11, pages 400–416, Berlin, Heidelberg, 2011. Springer.

[Vad17] Salil Vadhan. The complexity of differential privacy. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, chapter 7, pages 347–450. Springer International Publishing AG, Cham, Switzerland, 2017.