CS325: Private Data Analysis **Fall 2022**

Lecture 12: Releasing Multiple Linear Queries Privately: II

Lecturer: Di Wang *Scribes: Di Wang*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

In the previous lecture, we provide a factorization and projetion mechanism, whose error is $O((\frac{\sigma_{\epsilon,\delta}\log^{1/2}|\mathcal{D}|}{n})^{\frac{1}{2}})$ to answer k linear queries over the data universe $\mathcal{D}$. Equivalently, to achieve an error of α, the number of samples we need should be $n = O(\frac{\sqrt{\log|\mathcal{D}|\log 1/\delta}}{\alpha^2\epsilon})$. In this lecture, we will consider the problem of answering multiple queries as a online two-player game and provide an algorithm whose sample complexity is $n = O(\frac{\log k\sqrt{\log|\mathcal{D}|\log 1/\delta}}{\alpha^2\epsilon})$, or equivalently, the error will be $n = O(\frac{\log k\sqrt{\log|\mathcal{D}|\log 1/\delta}}{\alpha^2\epsilon})$. The algorithm is called private multiplicative weight (PMW) algorithm, introduced by [1].

12.1 Non-private Multiplicative Weights

Lets start quite distant from differential privacy and answering linear queries.

12.1.1 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 doesnt 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 theres an algorithm which makes at most $\log N$ mistakes.

Lemma 12.1 *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.

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 dont make a mistake, or half the experts are wrong and they get removed. More precisely, if we make a mistake at step t, then $|S_{t+1}|\leq \frac{|S_t|}{2}$. 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

Algorithm 1 Perfect Expert

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

best expert (in hindsight).

12.1.2 A Best Expert

This isnt the most realistic scenario no one can truly predict the future. Well assume that no theres no perfect expert, but our goal is relaxed as well: were just trying to perform competitively with the best expert, who makes OPT mistakes. Its easy to see that the exact same strategy wont 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.

Lemma 12.2 *There is an algorithm that makes at most* $2.4(OPT + \log N)$ *mistakes.*

This time, the difference between the number of mistakes and the number made by the best expert is $1.4OPT + 2.4 \log N$.

Algorithm 2 Weighted Majority Algorithm

Set $w_i^1 = 1$ for all $i \in [N]$.
for $t = 1, \cdots, T$ **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 $p_i^t \neq s^t$, let $w_i^{t+1} = \frac{w_i^t}{2}$.
end for

Proof: Let W^T be the total wright 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 OP T 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 OP T mistakes thus, their weight remains at least $(1/2)^{OPT}$. Thus, we have $(1/2)^{OPT} \leq W^T$. On the other hand, we know that every time the algorithm makes a mistake, the total weight drops by a factor of 3/4 at most: 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: $W^T \leq N(\frac{3}{4})^M$. In total we have $(1/2)^{OPT} \leq N(\frac{3}{4})^M$, that is $M \leq \frac{OPT + \log N}{\log(4/3)}$. ■

12.1.3 Multiplicative Weights Algorithm

The previous approaches give us some of the key ideas we need: multiplicatively reward or penalize experts based on whether theyre right or wrong. Well 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 algorithms total loss is $L_A^T = \sum_{t=1}^T \ell_A^t$, and expert is 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: $L_A^T - \min_{i\in[N]} L_i^T$.

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. 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:

Algorithm 3 Multiplicative Weights Algorithm

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

Theorem 12.3 *For an arbitrary sequence of losses, and any expert i,*

$$\mathbb{E}[L_A^T] - L_i^T \leq 2\sqrt{T \log N}. \tag{12.1}$$

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. 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.

Algorithm 4 Multiplicative Weights Algorithm

1: Set $w_i^1 = 1$ for all $i \in [N]$.
2: **for** $t = 1, \cdots, T$ **do**
3: Let $W^t = \sum_{i=1}^N w_i^t$
4: Set the probability distribution p^t where $p_i^t = \frac{w_i^t}{W^t}$.
5: Update $w_i^{t+1} = w_i^t(1 - \gamma \ell_i^t)$,where γ is some parameter to be set.
6: **end for**

Theorem 12.4 *For an arbitrary sequence of loss functions:*

$$\sum_{t=1}^T \ell^t \cdot p^t - \sum_{t=1}^T \ell^t \cdot p \leq 2\sqrt{T \log N}, \tag{12.2}$$

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

$$W^{t+1} = \sum_{i=1}^N w_i^t(1 - \gamma \ell_i^t) = W^t(1 - \ell^t \cdot p^t) \tag{12.3}$$

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

$$W^{T+1} = N\Pi_{t=1}^T(1 - \gamma \ell^t \cdot p^t). \tag{12.4}$$

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

$$\ln W^{T+1} = \ln N + \sum_{t=1}^T \ln(1 - \gamma \ell^t \cdot p^t) \leq \ln N - \gamma \sum_{t=1}^T \ell^t \cdot p^t. \tag{12.5}$$

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 $-1/2 \leq x \leq 1/2$,

$$\ln W^{T+1} \geq \ln w_i^{T+1} = \sum_{t=1}^T \ln(1 - \gamma \ell_i^t) \geq -\sum_{i=1}^T \gamma \ell_i^t - \sum_{t=1}^T (\ell_i^t)^2 \geq -\gamma L_i^T - \gamma^2 T. \tag{12.6}$$

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

$$\ln W^{T+1} \geq -\gamma \sum_{t=1}^T p \cdot \ell^t - \gamma^2 T^2. \tag{12.7}$$

Thus,

$$\sum_{t=1}^T \ell^t \cdot p^t - \sum_{t=1}^T p \cdot \ell^t \leq \gamma T + \frac{\log N}{\gamma}. \tag{12.8}$$

Choose $\gamma = \sqrt{\log N / T}$ we have the result. ■

12.2 Private Multiplicative Weights

12.2.1 Multiplicative Weights for Queries

We now have some key ideas in place, and its 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.

Suppose we have k queries. Just as the previous lecture, we can represent each query f_i on D as $\langle f_i, h_D \rangle$, where $f_i \in [0,1]^{\mathcal{D}}$ and h_D is the histogram representation of the dataset D.

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 f^t in $\{f_1, \cdots, f_k\}$.
- The algorithm selects a distribution p^t over $\mathcal{D}$.

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

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

$$g(p^t) = |\langle f^t, p^t \rangle - \langle f^t, h_D \rangle|.$$

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

$$g(h_D) \geq g(p^t) + \langle \nabla g(p^t), h_D - p^t \rangle.$$

Rearranging, and using the fact that $g(h_D) = 0$ we have

$$g(p^t) \leq \langle \nabla g(p^t), p^t - h_D \rangle.$$

Summing over all t, we get

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

This is quite similar as the result in Theorem 12.4 with $\ell^t = \nabla g(p^t)$. Specifically, this gives a vector $[-1,1]^{|\mathcal{D}|}$ such that $\ell_i^t = f^t(i)$, if $\langle f^t, p^t \rangle \geq \langle f^t, h_D \rangle$ and $\ell_i^t = -f^t(i)$ otherwise.

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

$$\sum_{t=1}^{T} |\langle f^t, p^t \rangle - \langle f^t, h_D \rangle| \leq \sum_{t=1}^{T} \langle \nabla g(p^t), p^t - h_D \rangle \leq 2\sqrt{T \log |\mathcal{D}|}. \tag{12.9}$$

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 $|\langle f^t, p^t\rangle - \langle f^t, h_D\rangle| > \alpha$. Suppose for all T time steps, the adversary chooses a function which causes the algorithm to make a mistake: specifically, at every time t$|\langle f^t, p^t\rangle - \langle f^t, h_D\rangle| > \alpha$. 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 $2\sqrt{T \log|\mathcal{D}|}$. Thus we have $T \leq \frac{4\log|\mathcal{D}|}{\alpha^2}$. This implies that the adversary can only choose queries from all queries which result in a mistake up to $\frac{4\log|\mathcal{D}|}{\alpha^2}$ rounds. After this, no query would cause a mistake, meaning that it can answer all of them with error α.

This gives an approach for arriving at a distribution which can simultaneously answer all queries accurately. Specifically, we act as the adversary, repeatedly choosing queries f^t which cause the algorithm to make a mistake. Again, it seems very roundabout, since we could have just picked $p^t = h_D$, but we will see that all of the steps in this algorithm are amenable to privacy.

Algorithm 5 Multiplicative Weights Algorithm

1: Set $p_i^1 = \frac{1}{|\mathcal{D}|}$ for all $i \in [N]$.
2: **for** $t = 1, \cdots, T$ **do**
3: Choose a query f^t among the k queries with $|\langle f^t, p^t\rangle - \langle f^t, h_D\rangle| > \alpha$.
4: Set the probability distribution p^{t+1} where $p_i^{t+1} \propto p_i^t(1 - \eta \text{sign}(\langle f^t, p^t\rangle - \langle f^t, h_D\rangle) f_i^t)$ with $\eta = \sqrt{\frac{\log|\mathcal{D}|}{T}}$.
5: **end for**

Corollary 12.5 *Algorithm 5 can only run for at most $\frac{4\log|\mathcal{D}|}{\alpha^2}$ rounds until it is no longer able to select a query which causes a mistake.*

12.2.2 Private Multiplicative Weights for Queries

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. See Algorithm 6 for details. Let us first analyze the privacy of this approach. We can see that each iteration is $2\epsilon_0$-DP.

Algorithm 6 Private Multiplicative Weights Algorithm

1: Set $p_i^1 = \frac{1}{|\mathcal{D}|}$ for all $i \in [N]$.
2: **for** $t = 1, \cdots, T$ **do**
3: Use the exponential mechanism to choose a query f^t among the k queries with $|\langle f^t, p^t\rangle - \langle f^t, h_D\rangle| > \alpha$ using ϵ_0-DP.
4: Compute $y^t = \langle f^t, p^t\rangle - \langle f^t, h_D\rangle + \text{Lap}(\frac{1}{\epsilon_0 n})$.
5: If $|y^t| \leq 2\alpha$ return p^t.
6: Set the probability distribution p^{t+1} where $p_i^{t+1} \propto p_i^t(1 - \eta \text{sign}(y^t) f_i^t)$ with $\eta = \sqrt{\frac{\log|\mathcal{D}|}{T}}$.
7: **end for**

By basic composition, the overall algorithm is $2\epsilon_0 T$-DP. However, using advanced composition, this will be $(O(\epsilon_0\sqrt{T\log 1/\delta}), \delta)$-DP. Thus, take $\epsilon_0 = O(\frac{\epsilon}{\sqrt{T\log 1/\delta}})$, the algorithm will be (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 $\frac{\alpha}{100}$ of the large possible. Similarly, because of the Laplace mechanism, we incur error at most $\frac{\alpha}{100}$ 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 $O(\alpha)$ regret. and we can prove a similar result to bound $T = O(\frac{\log|\mathcal{D}|}{\alpha^2})$. Analyzing the accuracy of both of these is standard.3 The Laplace mechanism incurs error $O(\frac{1}{\epsilon_0 n})$ for each invocation. Upper bounding this by $\frac{\alpha}{100}$ gives the requirement that $n \geq \Omega(1/\epsilon\alpha) = \Omega(\sqrt{\log|\mathcal{D}|\log 1/\delta}/\alpha^2\epsilon)$. Similarly, the exponential mechanism incurs error $O(\frac{\log|\mathcal{D}|}{\epsilon_0 n})$ and bounding this by $\frac{\alpha}{100}$ requires that $n \geq \Omega(\log k\sqrt{\log|\mathcal{D}|\log 1/\delta}/\alpha^2\epsilon)$. Thus, in total we have

Theorem 12.6 *The Private Multiplicative Weights algorithm can answer a set of k linear queries on a database of size n to accuracy α under (ϵ, δ)-DP, given $n = \tilde{O}(\frac{\log k\sqrt{\log|\mathcal{D}|\log 1/\delta}}{\alpha^2\epsilon})$ examples.*

References

[1] Moritz Hardt and Guy N Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 61–70. IEEE, 2010.