5

Generalizations

In this section we generalize the query release algorithms of the previous section. As a result, we get bounds for arbitrary low sensitivity queries (not just linear queries), as well as new bounds for linear queries. These generalizations also shed some light on a connection between query release and machine learning.

The SmallDB offline query release mechanism in Section 4 is a special case of what we call the *net mechanism*. We saw that both mechanisms in that section yield *synthetic databases*, which provide a convenient means for approximating the value of any query in $\mathcal{Q}$ on the private database: just evaluate the query on the synthetic database and take the result as the noisy answer. More generally, a mechanism can produce a *data structure* of arbitrary form, that, together with a fixed, public, algorithm (independent of the database) provides a method for approximating the values of queries.

The Net mechanism is a straightforward generalization of the SmallDB mechanism: First, fix, independent of the actual database, an α-net of data structures such that evaluation of any query in $\mathcal{Q}$ using the released data structure gives a good (within an additive α error) estimate of the value of the query on the private database. Next, apply

the exponential mechanism to choose an element of this net, where the quality function minimizes the maximum error, over the queries in $\mathcal{Q}$, for the elements of the net.

We also generalize the online multiplicative weights algorithm so that we can instantiate it with any other *online learning algorithm* for learning a database with respect to a set of queries. We note that such a mechanism can be run either online, or offline, where the set of queries to be asked to the "online" mechanism is instead selected using a "private distinguisher," which identifies queries on which the current hypothesis of the learner differs substantially from the real database. These are queries that would have yielded an update step in the online algorithm. A "distinguisher" turns out to be equivalent to an agnostic learning algorithm, which sheds light on a source of hardness for efficient query release mechanisms.

In the following sections, we will discuss *data structures* for classes of queries $\mathcal{Q}$.

Definition 5.1. A data structure D drawn from some class of data structures $\mathcal{D}$ for a class of queries $\mathcal{Q}$ is implicitly endowed with an evaluation function $\text{Eval} : \mathcal{D} \times \mathcal{Q} \to \mathbb{R}$ with which we can evaluate any query in $\mathcal{Q}$ on D. However, to avoid being encumbered by notation, we will write simply $f(D)$ to denote $\text{Eval}(D, f)$ when the meaning is clear from context.

5.1 Mechanisms via α-nets

Given a collection of queries $\mathcal{Q}$, we define an α-net as follows:

Definition 5.2 (α-net)**.** An α-net of data structures with respect to a class of queries $\mathcal{Q}$ is a set $\mathcal{N} \subset \mathbb{N}^{|\mathcal{X}|}$ such that for all $x \in \mathbb{N}^{|\mathcal{X}|}$, there exists an element of the α-net $y \in \mathcal{N}$ such that:

$$\max_{f \in \mathcal{Q}} |f(x) - f(y)| \leq \alpha \,.$$

We write $\mathcal{N}_\alpha(\mathcal{Q})$ to denote an α-net of minimum cardinality among the set of all α-nets for $\mathcal{Q}$.

That is, for every possible database x, there exists a member of the α-net that "looks like" x with respect to all queries in $\mathcal{Q}$, up to an error tolerance of α.

Small α-nets will be useful for us, because when paired with the exponential mechanism, they will lead directly to mechanisms for answering queries with high accuracy. Given a class of functions $\mathcal{Q}$, we will define an instantiation of the exponential mechanism known as the *Net* mechanism. We first observe that the Net mechanism preserves ε-differential privacy.

Algorithm 7 The Net Mechanism

NetMechanism$(x, \mathcal{Q}, \varepsilon, \alpha)$

Let $\mathcal{R} \leftarrow \mathcal{N}_\alpha(\mathcal{Q})$

Let $q : \mathbb{N}^{|\mathcal{X}|} \times \mathcal{R} \to \mathbb{R}$ be defined to be:

$$q(x, y) = -\max_{f \in \mathcal{Q}} |f(x) - f(y)|$$

Sample And Output $y \in \mathcal{R}$ with the exponential mechanism $\mathcal{M}_E(x, q, \mathcal{R})$

Proposition 5.1. The Net mechanism is $(\varepsilon, 0)$ differentially private.

Proof. The Net mechanism is simply an instantiation of the exponential mechanism. Therefore, privacy follows from Theorem 3.10. □

We may similarly call on our analysis of the exponential mechanism to begin understanding the utility guarantees of the Net mechanism:

Proposition 5.2. Let $\mathcal{Q}$ be any class of sensitivity $1/\|x\|_1$ queries. Let y be the database output by NetMechanism$(x, \mathcal{Q}, \varepsilon, \alpha)$. Then with probability $1 - \beta$:

$$\max_{f \in \mathcal{Q}} |f(x) - f(y)| \leq \alpha + \frac{2\left(\log\left(|\mathcal{N}_\alpha(\mathcal{Q})|\right) + \log\left(\frac{1}{\beta}\right)\right)}{\varepsilon \|x\|_1}.$$

Proof. By applying Theorem 3.11 and noting that $S(q) = \frac{1}{\|x\|_1}$, and that $\mathrm{OPT}_q(D) \leq \alpha$ by the definition of an α-net, we find:

$$\Pr\left[\max_{f \in \mathcal{Q}} |f(x) - f(y)| \geq \alpha + \frac{2}{\varepsilon \|x\|_1} \left(\log\left(|\mathcal{N}_\alpha(\mathcal{Q})|\right) + t\right)\right] \leq e^{-t}.$$

Plugging in $t = \log\left(\frac{1}{\beta}\right)$ completes the proof. □

We can therefore see that an upper bound on $|\mathcal{N}_\alpha(\mathcal{Q})|$ for a collection of functions $\mathcal{Q}$ immediately gives an upper bound on the accuracy that a differentially private mechanism can provide simultaneously for *all* functions in the class $\mathcal{Q}$.

This is exactly what we did in Section 4.1, where we saw that the key quantity is the VC-dimension of $\mathcal{Q}$, when $\mathcal{Q}$ is a class of linear queries.

5.2 The iterative construction mechanism

In this section, we derive an offline generalization of the private multiplicative weights algorithm, which can be instantiated with any properly defined learning algorithm. Informally, a database update algorithm maintains a sequence of data structures $D^1, D^2, \ldots$ that give increasingly good approximations to the input database x (in a sense that depends on the database update algorithm). Moreover, these mechanisms produce the next data structure in the sequence by considering only one query f that *distinguishes* the real database in the sense that $f(D^t)$ differs significantly from $f(x)$. The algorithm in this section shows that, up to small factors, solving the query-release problem in a differentially private manner is equivalent to solving the simpler *learning* or *distinguishing* problem in a differentially private manner: given a private distinguishing algorithm and a non-private database update algorithm, we get a corresponding private release algorithm. We can plug in the exponential mechanism as a canonical private distinguisher, and the multiplicative weights algorithm as a generic database update algorithm for the general linear query setting, but more efficient distinguishers are possible in special cases.

Syntactically, we will consider functions of the form $U : \mathcal{D} \times \mathcal{Q} \times \mathbb{R} \to \mathcal{D}$, where $\mathcal{D}$ represents a class of data structures on which queries in $\mathcal{Q}$ can be evaluated. The inputs to U are a data structure in $\mathcal{D}$, which represents the current data structure D^t; a query f, which represents the distinguishing query, and may be restricted to a certain set $\mathcal{Q}$; and also a real number, which estimates $f(x)$. Formally, we define a *database update sequence*, to capture the sequence of inputs to U used to generate the database sequence $D^1, D^2, \ldots$.

Definition 5.3 (Database Update Sequence)**.** Let $x \in \mathbb{N}^{|\mathcal{X}|}$ be any database and let $\{(D^t, f_t, v_t)\}_{t=1,\ldots,L} \in (\mathcal{D} \times \mathcal{Q} \times \mathbb{R})^L$ be a sequence of tuples. We say the sequence is a $(U, x, \mathcal{Q}, \alpha, T)$-*database update sequence* if it satisfies the following properties:

1. $D^1 = U(\perp, \cdot, \cdot)$,
2. for every $t = 1, 2, \ldots, L$, $|f_t(x) - f_t(D^t)| \geq \alpha$,
3. for every $t = 1, 2, \ldots, L$, $|f_t(x) - v_t| < \alpha$,
4. and for every $t = 1, 2, \ldots, L-1$, $D^{t+1} = U(D^t, f_t, v_t)$.

We note that for all of the database update algorithms we consider, the approximate answer v_t is used only to determine the *sign* of $f_t(x) - f_t(D^t)$, which is the motivation for requiring that the estimate of $f_t(x)$ (v_t) have error smaller than α. The main measure of efficiency we're interested in from a database update algorithm is the maximum number of updates we need to perform before the database D^t approximates x well with respect to the queries in $\mathcal{Q}$. To this end we define a database update algorithm as follows:

Definition 5.4 (Database Update Algorithm)**.** Let $U : \mathcal{D} \times \mathcal{Q} \times \mathbb{R} \to \mathcal{D}$ be an update rule and let $T : \mathbb{R} \to \mathbb{R}$ be a function. We say U is a $T(\alpha)$-*database update algorithm for query class* $\mathcal{Q}$ if for every database $x \in \mathbb{N}^{|\mathcal{X}|}$, every $(U, x, \mathcal{Q}, \alpha, L)$-database update sequence satisfies $L \leq T(\alpha)$.

Note that the definition of a $T(\alpha)$-database update algorithm implies that if U is a $T(\alpha)$-database update algorithm, then given any maximal $(U, x, \mathcal{Q}, \alpha, U)$-database update sequence, the final database D^L must satisfy $\max_{f \in \mathcal{Q}} \left| f(x) - f(D^L) \right| \leq \alpha$ or else there would exist

another query satisfying property 2 of Definition 5.3, and thus there would exist a $(U, x, \mathcal{Q}, \alpha, L+1)$-database update sequence, contradicting maximality. That is, the goal of a $T(\alpha)$ database update rule is to generate a maximal database update sequence, and the final data structure in a maximal database update sequence necessarily encodes the approximate answers to every query $f \in \mathcal{Q}$.

Now that we have defined database update algorithms, we can remark that what we really proved in Theorem 4.10 was that the Multiplicative Weights algorithm is a $T(\alpha)$-database update algorithm for $T(\alpha) = 4 \log |\mathcal{X}|/\alpha^2$.

Before we go on, let us build some intuition for what a database update algorithm is. A $T(\alpha)$-database update algorithm begins with some initial guess D^1 about what the true database x looks like. Because this guess is not based on any information, it is quite likely that D^1 and x bear little resemblance, and that there is some $f \in \mathcal{Q}$ that is able to distinguish between these two databases by at least α: that is, that $f(x)$ and $f(D^1)$ differ in value by at least α. What a database update algorithm does is to update its hypothesis D^t given evidence that its current hypothesis D^{t-1} is incorrect: at each stage, it takes as input some query in $\mathcal{Q}$ which distinguishes its current hypothesis from the true database, and then it outputs a new hypothesis. The parameter $T(\alpha)$ is an upper bound on the number of times that the database update algorithm will have to update its hypothesis: it is a promise that after at most $T(\alpha)$ distinguishing queries have been provided, the algorithm will finally have produced a hypothesis that looks like the true database with respect to $\mathcal{Q}$, at least up to error α.[1] For a database update algorithm, smaller bounds $T(\alpha)$ are more desirable.

Database Update Algorithms and Online Learning Algorithms: We remark that database update algorithms are essentially *online learning*

[1] Imagine that the database update algorithm is attempting to sculpt x out of a block of clay. Initially, its sculpture D^1 bears no resemblance to the true database: it is simply a block of clay. However, a helpful distinguisher points out to the sculptor places in which the clay juts out much farther than the true target database: the sculptor dutifully pats down those bumps. If the distinguisher always finds large protrusions, of magnitude at least α, the sculpture will be finished soon, and the distinguisher's time will not be wasted!

algorithms in the *mistake bound model*. In the setting of online learning, unlabeled examples arrive in some arbitrary order, and the learning algorithm must attempt to label them.

Background from Learning Theory. In the *mistake bound model of learning*, labeled examples $(x_i, y_i) \in \mathcal{X} \times \{0, 1\}$ arrive one at a time, in a potentially adversarial order. At time i, the learning algorithm A observes x_i, and must make a prediction $\hat{y}_i$ about the label for x_i. It then sees the true label y_i, and is said to *make a mistake* if its prediction was wrong: i.e., if $y_i \neq \hat{y}_i$. A learning algorithm A for a class of functions C is said to have a mistake bound of M, if for all $f \in C$, and for all adversarially selected sequences of examples $(x_1, f(x_1)), \ldots, (x_i, f(x_i)), \ldots$, A never makes more than M mistakes. Without loss of generality, we can think of such a learning algorithm as maintaining some hypothesis $\hat{f} : \mathcal{X} \to \{0, 1\}$ at all times, and updating it only when it makes a mistake. The adversary in this model is quite powerful — it can choose the sequence of labeled examples adaptively, knowing the current hypothesis of the learning algorithm, and its entire history of predictions. Hence, learning algorithms that have finite mistake bounds can be useful in extremely general settings.

It is not hard to see that mistake bounded online learning algorithms always exist for finite classes of functions C. Consider, for example, the *halving algorithm*. The halving algorithm initially maintains a set S of functions from C consistent with the examples that it has seen so far: Initially $S = C$. Whenever a new unlabeled example arrives, it predicts according to the majority vote of its consistent hypotheses: that is, it predicts label 1 whenever $|\{f \in S : f(x_i) = 1\}| \geq |S|/2$. Whenever it makes a mistake on an example x_i, it updates S by removing any inconsistent function: $S \leftarrow \{f \in S : f(x_i) = y_i\}$. Note that whenever it makes a mistake, the size of S is cut in half! So long as all examples are labeled by *some* function $f \in C$, there is at least one function $f \in C$ that is never removed from S. Hence, the halving algorithm has a mistake bound of $\log |C|$.

Generalizing beyond boolean labels, we can view database update algorithms as online learning algorithms in the mistake bound model:

here, examples that arrive are the queries (which may come in adversarial order). The labels are the approximate values of the queries when evaluated on the database. The database update algorithm hypothesis D^t makes a *mistake* on query f if $|f(D^t) - f(x)| \geq \alpha$, in which case we learn the label of f (that is, v_t) and allow the database update algorithm to update the hypothesis. Saying that an algorithm U is a $T(\alpha)$-database update algorithm is akin to saying that it has a mistake bound of $T(\alpha)$: no adversarially chosen sequence of queries can ever cause it to make more than $T(\alpha)$-mistakes. Indeed, the database update algorithms that we will see are taken from the online learning literature. The multiplicative weights mechanism is based on an online learning algorithm known as *Hedge*, which we have already discussed. The Median Mechanism (later in this section) is based on the *Halving Algorithm*, and the Perceptron algorithm is based (coincidentally) on an algorithm known as *Perceptron*. We won't discuss Perceptron here, but it operates by making *additive* updates, rather than the multiplicative updates used by multiplicative weights.

A database update algorithm for a class $\mathcal{Q}$ will be useful together with a corresponding *distinguisher*, whose job is to output a function that behaves differently on the true database x and the hypothesis D^t, that is, to point out a mistake.

Definition 5.5 ($(F(\varepsilon), \gamma)$-Private Distinguisher)**.** Let $\mathcal{Q}$ be a set of queries, let $\gamma \geq 0$ and let $F(\varepsilon) : \mathbb{R} \to \mathbb{R}$ be a function. An algorithm $\text{Distinguish}_\varepsilon : \mathbb{N}^{|\mathcal{X}|} \times \mathcal{D} \to \mathcal{Q}$ is an $(F(\varepsilon), \gamma)$-Private Distinguisher for $\mathcal{Q}$ if for every setting of the privacy parameter ε, on every pair of inputs $x \in \mathbb{N}^{|\mathcal{X}|}$, $D \in \mathcal{D}$ it is $(\varepsilon, 0)$-differentially private with respect to x and it outputs an $f^* \in \mathcal{Q}$ such that $|f^*(x) - f^*(D)| \geq \max_{f \in \mathcal{Q}} |f(x) - f(D)| - F(\varepsilon)$ with probability at least $1 - \gamma$.

Remark 5.1. In machine learning, the goal is to find a function $f : \mathcal{X} \to \{0, 1\}$ from a class of functions $\mathcal{Q}$ that *best labels* a collection of labeled examples $(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \{0, 1\}$. (Examples $(x, 0)$ are known as *negative examples*, and examples $(x, 1)$ are known as *positive examples*). Each example x_i has a *true* label y_i, and a function f *correctly labels* x_i if $f(x_i) = y_i$. An *agnostic learning algorithm* for a class $\mathcal{Q}$ is an algorithm that can find the function in $\mathcal{Q}$ that labels

all of the data points approximately as well as the best function in $\mathcal{Q}$, even if no function in $\mathcal{Q}$ can perfectly label them. Note that equivalently, an agnostic learning algorithm is one that maximizes the number of positive examples labeled 1 minus the number of negative examples labeled 1. Phrased in this way, we can see that a *distinguisher* as defined above is just an agnostic learning algorithm: just imagine that x contains all of the "positive" examples, and that y contains all of the "negative examples." (Note that it is ok of x and y are not disjoint — in the learning problem, the same example can occur with both a positive and a negative label, since agnostic learning does not require that any function perfectly label every example.) Finally, note also that for classes of linear queries $\mathcal{Q}$, a distinguisher is simply an optimization algorithm. Because for linear queries f, $f(x) - f(y) = f(x - y)$, a distinguisher simply seeks to find $\arg\max_{f \in \mathcal{Q}} |f(x - y)|$.

Note that, *a priori*, a differentially private distinguisher is a weaker object than a differentially private release algorithm: A distinguisher merely finds a query in a set $\mathcal{Q}$ with the approximately largest value, whereas a release algorithm must find the answer to every query in $\mathcal{Q}$. In the algorithm that follows, however, we reduce release to optimization.

We will first analyze the IC algorithm, and then instantiate it with a specific distinguisher and database update algorithm. What follows is a formal analysis, but the intuition for the mechanism is simple: we simply run the iterative database construction algorithm to construct a hypothesis that approximately matches x with respect to the queries $\mathcal{Q}$. If at each round our distinguisher succeeds in finding a query that has high discrepancy between the hypothesis database and the true database, then our database update algorithm will output a database that is β-accurate with respect to $\mathcal{Q}$. If the distinguisher ever fails to find such a query, then it must be that there are no such queries, and our database update algorithm has already learned an accurate hypothesis with respect to the queries of interest! This requires at most T iterations, and so we access the data only $2T$ times using $(\varepsilon_0, 0)$-differentially private methods (running the given distinguisher, and then checking its answer with the Laplace mechanism). Privacy will therefore follow from our composition theorems.

Algorithm 8 The Iterative Construction (IC) Mechanism. It takes as input a parameter ε_0, an $(F(\varepsilon_0), \gamma)$-Private Distinguisher Distinguish for $\mathcal{Q}$, together with an $T(\alpha)$-iterative database update algorithm U for $\mathcal{Q}$.

IC$(x, \alpha, \varepsilon_0, \text{Distinguish}, U)$:
  Let $D^0 = U(\perp, \cdot, \cdot)$.
  for $t = 1$ to $T(\alpha/2)$ **do**
    Let $f^{(t)} = \text{Distinguish}(x, D^{t-1})$
    Let $\hat{v}^{(t)} = f^{(t)}(x) + \text{Lap}\left(\frac{1}{\|x\|_1 \varepsilon_0}\right)$.
    if $|\hat{v}^{(t)} - f^{(t)}(D^{t-1})| < 3\alpha/4$ **then**
      Output $y = D^{t-1}$.
    else
      Let $D^t = U(D^{t-1}, f^{(t)}, \hat{v}^{(t)})$.
    end if
  end for
  Output $y = D^{T(\alpha/2)}$.

The analysis of this algorithm just involves checking the technical details of a simple intuition. Privacy will follow because the algorithm is just the composition of $2T(\alpha)$ steps, each of which is $(\varepsilon_0, 0)$-differentially private. Accuracy follows because we are always outputting the last database in a maximal database update sequence. If the algorithm has not yet formed a maximal Database Update Sequence, then the distinguishing algorithm will find a distinguishing query to add another step to the sequence.

Theorem 5.3. The IC algorithm is $(\varepsilon, 0)$-differentially private for $\varepsilon_0 \leq \varepsilon / 2T(\alpha/2)$. The IC algorithm is (ε, δ)-differentially private for $\varepsilon_0 \leq \frac{\varepsilon}{4\sqrt{T(\alpha/2)\log(1/\delta)}}$.

Proof. The algorithm runs at most $2T(\alpha/2)$ compositions of ε_0-differentially private algorithms. Recall from Theorem 3.20 that ε_0 differentially private algorithms are $2k\varepsilon_0$ differentially private under $2k$-fold composition, and are (ε', δ) private for $\varepsilon' = \sqrt{4k\ln(1/\delta')}\varepsilon_0 + 2k\varepsilon_0(e^{\varepsilon_0} - 1)$. Plugging in the stated values for ε_0 proves the claim. □

Theorem 5.4. Given an $(F(\varepsilon), \gamma)$-private distinguisher, a parameter ε_0, and a $T(\alpha)$-Database Update Algorithm, with probability at least $1-\beta$, the IC algorithm returns a database y such that: $\max_{f \in \mathcal{Q}} |f(x)-f(y)| \leq \alpha$ for any α such that where:

$$\alpha \geq \max\left[\frac{8\log(2T(\alpha/2)/\beta)}{\varepsilon_0\|x\|_1}, 8F\left(\varepsilon_0\right)\right]$$

so long as $\gamma \leq \beta/(2T(\alpha/2))$.

Proof. The analysis is straightforward.

Recall that if $Y_i \sim \mathrm{Lap}(1/(\varepsilon\|x\|_1))$, we have: $\Pr[|Y_i| \geq t/(\varepsilon\|x\|_1)] = \exp(-t)$. By a union bound, if $Y_1, \ldots, Y_k \sim \mathrm{Lap}(1/(\varepsilon\|x\|_1))$, then $\Pr[\max_i |Y_i| \geq t/(\varepsilon\|x\|_1)] \leq k\exp(-t)$. Therefore, because we make at most $T(\alpha/2)$ draws from $\mathrm{Lap}(1/(\varepsilon_0\|x\|_1))$, except with probability at most $\beta/2$, for all t:

$$|\hat{v}^{(t)} - f^{(t)}(x)| \leq \frac{1}{\varepsilon_0\|x\|_1}\log\frac{2T(\alpha/2)}{\beta} \leq \frac{\alpha}{8}.$$

Note that by assumption, $\gamma \leq \beta/(2T(\alpha/2))$, so we also have that except with probability $\beta/2$:

$$\begin{aligned}|f^{(t)}(x) - f^{(t)}(D^{t-1})| &\geq \max_{f\in\mathcal{Q}} |f(x) - f(D^{t-1})| - F(\varepsilon_0) \\ &\geq \max_{f\in\mathcal{Q}} |f(x) - f(D^{t-1})| - \frac{\alpha}{8}.\end{aligned}$$

For the rest of the argument, we will condition on both of these events occurring, which is the case except with probability β.

There are two cases. Either a data structure $D' = D^{T(\alpha/2)}$ is output, or data structure $D' = D^t$ for $t < T(\alpha/2)$ is output. First, suppose $D' = D^{T(\alpha/2)}$. Since for all $t < T(\alpha/2)$ it must have been the case that $|\hat{v}^{(t)} - f^{(t)}(D^{t-1})| \geq 3\alpha/4$ and by our conditioning, $|\hat{v}^{(t)} - f^{(t)}(x)| \leq \frac{\alpha}{8}$, we know for all t: $|f^{(t)}(x) - f^{(t)}(D^{t-1})| \geq \alpha/2$. Therefore, the sequence $(D^t, f^{(t)}, \hat{v}^{(t)})$, formed a maximal $(U, x, \mathcal{Q}, \alpha/2, T(\alpha/2))$-Database Update Sequence (recall Definition 5.3), and we have that $\max_{f\in\mathcal{Q}} |f(x) - f(x')| \leq \alpha/2$ as desired.

Next, suppose $D' = D^{t-1}$ for $t < T(\alpha/2)$. Then it must have been the case that for t, $|\hat{v}^{(t)} - f^{(t)}(D^{t-1})| < 3\alpha/4$. By our conditioning, in

this case it must be that $|f^{(t)}(x) - f^{(t)}(D^{t-1})| < \frac{7\alpha}{8}$, and that therefore by the properties of an $(F(\varepsilon_0), \gamma)$-distinguisher:

$$\max_{f \in \mathcal{Q}} |f(x) - f(D')| < \frac{7\alpha}{8} + F(\varepsilon_0) \leq \alpha$$

as desired. □

Note that we can use the exponential mechanism as a private distinguisher: take the domain to be $\mathcal{Q}$, and let the quality score be: $q(D, f) = |f(D) - f(D^t)|$, which has sensitivity $1/\|x\|_1$. Applying the exponential mechanism utility theorem, we get:

Theorem 5.5. The exponential mechanism is an $(F(\varepsilon), \gamma)$ distinguisher for:

$$F(\varepsilon) = \frac{2}{\|x\|_1 \varepsilon} \left(\log \frac{|\mathcal{Q}|}{\gamma} \right).$$

Therefore, using the exponential mechanism as a distinguisher, Theorem 5.4 gives:

Theorem 5.6. Given a $T(\alpha)$-Database Update Algorithm and a parameter ε_0 together with the exponential mechanism distinguisher, with probability at least $1 - \beta$, the IC algorithm returns a database y such that: $\max_{f \in \mathcal{Q}} |f(x) - f(y)| \leq \alpha$ where:

$$\alpha \leq \max \left[\frac{8 \log(2T(\alpha/2)/\beta)}{\varepsilon_0 \|x\|_1}, \frac{16}{\|x\|_1 \varepsilon_0} \left(\log \frac{|\mathcal{Q}|}{\gamma} \right) \right]$$

so long as $\gamma \leq \beta/(2T(\alpha/2))$.

Plugging in our values of ε_0:

Theorem 5.7. Given a $T(\alpha)$-Database Update Algorithm, together with the exponential mechanism distinguisher, the IC mechanism is ε-differentially private and with probability at least $1 - \beta$, the IC algorithm returns a database y such that: $\max_{f \in \mathcal{Q}} |f(x) - f(y)| \leq \alpha$ where:

$$\alpha \leq \frac{8T(\alpha/2)}{\|x\|_1 \varepsilon} \left(\log \frac{|\mathcal{Q}|}{\gamma} \right)$$

and (ε, δ)-differentially private for:

$$\alpha \leq \frac{16\sqrt{T(\alpha/2)\log(1/\delta)}}{\|x\|_1 \varepsilon}\left(\log\frac{|\mathcal{Q}|}{\gamma}\right)$$

so long as $\gamma \leq \beta/(2T(\alpha/2))$.

Note that in the language of this section, what we proved in Theorem 4.10 was exactly that the multiplicative weights algorithm is a $T(\alpha)$-Database Update Algorithm for $T(\alpha) = \frac{4\log|\mathcal{X}|}{\alpha^2}$. Plugging this bound into Theorem 5.7 recovers the bound we got for the online multiplicative weights algorithm. Note that now, however, we can plug in other database update algorithms as well.

5.2.1 Applications: other database update algorithms

Here we give several other database update algorithms. The first works directly from α-nets, and therefore can get non-trivial bounds even for nonlinear queries (unlike multiplicative weights, which only works for linear queries). The second is another database update algorithm for linear queries, but with bounds incomparable to multiplicative weights. (In general, it will yield improved bounds when the dataset has size close to the size of the data universe, whereas multiplicative weights will give better bounds when the dataset is much smaller than the data universe.)

We first discuss the median mechanism, which takes advantage of α-nets. The median mechanism does not operate on databases, but instead on median data structures:

Definition 5.6 (Median Data Structure)**.** A median data structure $\mathbf{D}$ is a collection of databases: $\mathbf{D} \subset \mathbb{N}^{|\mathcal{X}|}$. Any query f can be evaluated on a median data structure as follows: $f(\mathbf{D}) = \text{Median}(\{f(x) : x \in \mathbf{D}\})$.

In words, a median data structure is just a set of databases. To evaluate a query on it, we just evaluate the query on every database in the set, and then return the median value. Note that the answers given by the median data structure need not be consistent with *any* database! However, it will have the useful property that whenever it makes an

error, it will rule out at least half of the data sets in its collection as being inconsistent with the true data set.

The median mechanism is then very simple:

Algorithm 9 The Median Mechanism (MM) Update Rule. It inputs and outputs a median data structure. It is instantiated with an α-net $\mathcal{N}_\alpha(\mathcal{Q})$ for a query class $\mathcal{Q}$, and its initial state is $\mathbf{D} = \mathcal{N}_\alpha(\mathcal{Q})$

$MM_{\alpha,\mathcal{Q}}(\mathbf{D}^t, f_t, v_t)$:
  if $\mathbf{D}^t = \perp$ **then**
    Output $\mathbf{D}^0 \leftarrow N_\alpha(\mathcal{Q})$.
  end if
  if $v_t < f_t(\mathbf{D}^t)$ **then**
    Output $\mathbf{D}^{t+1} \leftarrow \mathbf{D}^t \setminus \{x \in \mathbf{D} : f_t(x) \geq f_t(\mathbf{D^t})\}$.
  else
    Output $\mathbf{D}^{t+1} \leftarrow \mathbf{D}^t \setminus \{x \in \mathbf{D} : f_t(x) \leq f_t(\mathbf{D^t})\}$.
  end if

The intuition for the median mechanism is as follows. It maintains a set of databases that are consistent with the answers to the distinguishing queries it has seen so far. Whenever it receives a query and answer that differ substantially from the real database, it updates itself to remove all of the databases that are inconsistent with the new information. Because it always chooses its answer as the median database among the set of consistent databases it is maintaining, every update step removes at least half of the consistent databases! Moreover, because the set of databases that it chooses initially is an α-net with respect to $\mathcal{Q}$, there is always some database that is never removed, because it remains consistent on all queries. This limits how many update rounds the mechanism can perform. How does the median mechanism do?

Theorem 5.8. For any class of queries $\mathcal{Q}$, The Median Mechanism is a $T(\alpha)$-database update algorithm for $T(\alpha) = \log |N_\alpha(\mathcal{Q})|$.

Proof. We must show that any sequence $\{(D^t, f_t, v_t)\}_{t=1,\ldots,L}$ with the property that $|f^t(\mathbf{D}^t) - f^t(x)| > \alpha$ and $|v_t - f^t(x)| < \alpha$ cannot have $L > \log |\mathcal{N}_\alpha(\mathcal{Q})|$. First observe that because $\mathbf{D}^0 = \mathcal{N}_\alpha(\mathcal{Q})$ is an α-net

for $\mathcal{Q}$, by definition there is at least one y such that $y \in \mathbf{D}^t$ for all t (Recall that the update rule is only invoked on queries with error at least α. Since there is guaranteed to be a database y that has error less than α on all queries, it is never removed by an update step). Thus, we can always answer queries with $\mathbf{D}^t$, and for all t, $|\mathbf{D}^t| \geq 1$. Next observe that for each t, $|\mathbf{D}^t| \leq |\mathbf{D}^{t-1}|/2$. This is because each update step removes at least half of the elements: all of the elements at least as large as, or at most as large as the median element in $\mathbf{D}^t$ with respect to query f_t. Therefore, after L update steps, $|\mathbf{D}^L| \leq 1/2^L \cdot |\mathcal{N}_\alpha(\mathcal{Q})|$. Setting $L > \log |\mathcal{N}_\alpha(\mathcal{Q})|$ gives $|\mathbf{D}^L| < 1$, a contradiction. □

Remark 5.2. For classes of linear queries $\mathcal{Q}$, we may refer to the upper bound on $\mathcal{N}_\alpha(\mathcal{Q})$ given in Theorem 4.2 to see that the Median Mechanism is a $T(\alpha)$-database update algorithm for $T(\alpha) = \log |\mathcal{Q}| \log |\mathcal{X}|/\alpha^2$. This is worse than the bound we gave for the Multiplicative Weights algorithm by a factor of $\log |\mathcal{Q}|$. On the other hand, nothing about the Median Mechanism algorithm is specific to linear queries — it works just as well for any class of queries that admits a small net. We can take advantage of this fact for nonlinear low sensitivity queries.

Note that if we want a mechanism which promises (ε, δ)-privacy for $\delta > 0$, we do not even need a particularly small net. In fact, the trivial net that simply includes every database of size $\|x\|_1$ will be sufficient:

Theorem 5.9. For every class of queries $\mathcal{Q}$ and every $\alpha \geq 0$, there is an α-net for databases of size $\|x\|_1 = n$ of size $\mathcal{N}_\alpha(\mathcal{Q}) \leq |\mathcal{X}|^n$.

Proof. We can simply let $\mathcal{N}_\alpha(\mathcal{Q})$ be the set of all $|\mathcal{X}|^n$ databases y of size $\|y\|_1 = n$. Then, for every x such that $\|x\|_1 = n$, we have $x \in \mathcal{N}_\alpha(\mathcal{Q})$, and so clearly: $\min_{y \in \mathcal{N}_\alpha(\mathcal{Q})} \max_{f \in \mathcal{Q}} |f(x) - f(y)| = 0$. □

We can use this fact to get query release algorithms for *arbitrary* low sensitivity queries, not just linear queries. Applying Theorem 5.7 to the above bound, we find:

Theorem 5.10. Using the median mechanism, together with the exponential mechanism distinguisher, the IC mechanism is (ε, δ)-differentially private and with probability at least $1 - \beta$, the IC algorithm returns a database y such that: $\max_{f \in \mathcal{Q}} |f(x) - f(y)| \leq \alpha$ where:

$$\alpha \leq \frac{16\sqrt{\log|\mathcal{X}| \log \frac{1}{\delta}} \log\left(\frac{2|\mathcal{Q}|n \log|\mathcal{X}|}{\beta}\right)}{\sqrt{n}\varepsilon},$$

where $\mathcal{Q}$ can be *any* family of sensitivity $1/n$ queries, not necessarily linear.

Proof. This follows simply by combining Theorems 5.8 and 5.9 to find that the Median Mechanism is a $T(\alpha)$-Database Update Algorithm for $T(\alpha) = n \log|\mathcal{X}|$ for databases of size $\|x\|_1 = n$ for every $\alpha > 0$ and every class of queries $\mathcal{Q}$. Plugging this into Theorem 5.7 gives the desired bound. □

Note that this bound is almost as good as we were able to achieve for the special case of linear queries in Theorem 4.15! However, unlike in the case of linear queries, because arbitrary queries may not have α-nets which are significantly smaller than the trivial net used here, we are not able to get nontrivial accuracy guarantees if we want $(\varepsilon, 0)$-differential privacy.

The next database update algorithm we present is again for linear queries, but achieves incomparable bounds to those of the multiplicative weights database update algorithm. It is based on the *Perceptron* algorithm from online learning (just as multiplicative weights is derived from the *hedge* algorithm from online learning). Since the algorithm is for linear queries, we treat each query $f_t \in \mathcal{Q}$ as being a vector $f_t \in [0,1]^{|\mathcal{X}|}$. Note that rather than doing a multiplicative update,

Algorithm 10 The Perceptron update rule

Perceptron$_{\alpha,\mathcal{Q}}(x^t, f_t, v_t)$:

If: $x^t = \perp$ **then:** output $x^{t+1} = 0^{|\mathcal{X}|}$

Else if: $f_t(x^t) > v_t$ **then:** output $x^{t+1} = x^t - \frac{\alpha}{|\mathcal{X}|} \cdot f_t$

Else if: $f_t(x^t) \leq v_t$ **then:** output $x^{t+1} = x^t + \frac{\alpha}{|\mathcal{X}|} \cdot f_t$

as in the MW database update algorithm, here we do an additive update. In the analysis, we will see that this database update algorithm has an exponentially worse dependence (as compared to multiplicative weights) on the size of the universe, but a superior dependence on the size of the database. Thus, it will achieve better performance for databases that are large compared to the size of the data universe, and worse performance for databases that are small compared to the size of the data universe.

Theorem 5.11. Perceptron is a $T(\alpha)$-database update algorithm for:

$$T(\alpha) = \left(\frac{\|x\|_2}{\|x\|_1}\right)^2 \cdot \frac{|\mathcal{X}|}{\alpha^2}.$$

Proof. Unlike for multiplicative weights, it will be more convenient to analyze the Perceptron algorithm without normalizing the database to be a probability distribution, and then prove that it is a $T(\alpha')$ database update algorithm for $T(\alpha') = \frac{\|x\|_2^2|\mathcal{X}|}{\alpha'^2}$. Plugging in $\alpha' = \alpha\|x\|_1$ will then complete the proof. Recall that since each query f_t is linear, we can view $f_t \in [0,1]^{|\mathcal{X}|}$ as a vector with the evaluation of $f_t(x)$ being equal to $\langle f_t, x\rangle$.

We must show that any sequence $\{(x^t, f_t, v_t)\}_{t=1,\ldots,L}$ with the property that $|f_t(x^t) - f_t(x)| > \alpha'$ and $|v_t - f_t(x)| < \alpha'$ cannot have $L > \frac{\|x\|_2^2|\mathcal{X}|}{\alpha'^2}$.

We use a potential argument to show that for every $t = 1, 2, \ldots, L$, x^{t+1} is significantly closer to x than x^t. Specifically, our potential function is the L_2^2 norm of the database $x - x^t$, defined as

$$\|x\|_2^2 = \sum_{i\in\mathcal{X}} x(i)^2.$$

Observe that $\|x - x^1\|_2^2 = \|x\|_2^2$ since $x^1 = 0$, and $\|x\|_2^2 \geq 0$. Thus it suffices to show that in every step, the potential decreases by $\alpha'^2/|\mathcal{X}|$. We analyze the case where $f_t(x^t) > v_t$, the analysis for the opposite case will be similar. Let $R^t = x^t - x$. Observe that in this case we have

$$f_t(R^t) = f_t(x^t) - f_t(x) \geq \alpha'.$$

Now we can analyze the drop in potential.

$$\begin{aligned}
\|R^t\|_2^2 - \|R^{t+1}\|_2^2 &= \|R^t\|_2^2 - \|R^t - (\alpha'/|\mathcal{X}|) \cdot f_t\|_2^2 \\
&= \sum_{i \in \mathcal{X}} ((R^t(i))^2 - (R^t(i) - (\alpha'/|\mathcal{X}|) \cdot f_t(i))^2) \\
&= \sum_{i \in \mathcal{X}} \left(\frac{2\alpha'}{|\mathcal{X}|} \cdot R^t(i) f_t(i) - \frac{\alpha'^2}{|\mathcal{X}|^2} f_t(i)^2 \right) \\
&= \frac{2\alpha'}{|\mathcal{X}|} f_t(R^t) - \frac{\alpha'^2}{|\mathcal{X}|^2} \sum_{i \in \mathcal{X}} f_t(i)^2 \\
&\geq \frac{2\alpha'}{|\mathcal{X}|} f_t(R^t) - \frac{\alpha'^2}{|\mathcal{X}|^2} |\mathcal{X}| \\
&\geq \frac{2\alpha'^2}{|\mathcal{X}|} - \frac{\alpha'^2}{|\mathcal{X}|} = \frac{\alpha'^2}{|\mathcal{X}|}.
\end{aligned}$$

This bounds the number of steps by $\|x\|_2^2 |\mathcal{X}|/\alpha'^2$, and completes the proof. □

We may now plug this bound into Theorem 5.7 to obtain the following bound on the iterative construction mechanism:

Theorem 5.12. Using the perceptron database update algorithm, together with the exponential mechanism distinguisher, the IC mechanism is (ε, δ)-differentially private and with probability at least $1 - \beta$, the IC algorithm returns a database y such that: $\max_{f \in \mathcal{Q}} |f(x) - f(y)| \leq \alpha$ where:

$$\alpha \leq \frac{4\sqrt{4}\sqrt{\|x\|_2} \left(4|\mathcal{X}| \ln(1/\delta)\right)^{1/4} \sqrt{\frac{\log(2|\mathcal{Q}\|\mathcal{X}| \cdot \|x\|_2^2)}{\beta}}}{\sqrt{\epsilon}\|x\|_1},$$

where $\mathcal{Q}$ is a class of linear queries.

If the database x represents the edge set of a graph, for example, we will have $x_i \in [0, 1]$ for all i, and so:

$$\frac{\sqrt{\|x\|_2}}{\|x\|_1} \leq \left(\frac{1}{\|x\|_1} \right)^{3/4}.$$

Therefore, the perceptron database update algorithm will outperform the multiplicative weights database update algorithm on dense graphs.

5.2.2 Iterative construction mechanisms and online algorithms

In this section, we generalize the iterative construction framework to the online setting by using the NumericSparse algorithm. The online multiplicative weights algorithm which saw in the last chapter is an instantiation of this approach. One way of viewing the online algorithm is that the NumericSparse algorithm is serving as the private distinguisher in the IC framework, but that the "hard work" of distinguishing is being foisted upon the unsuspecting user. That is: if the user asks a query that does not serve as a good distinguishing query, this is a good case. We cannot use the database update algorithm to update our hypothesis, but we don't need to! By definition, the current hypothesis is a good approximation to the private database with respect to this query. On the other hand, if the user asks a query for which our current hypothesis is not a good approximation to the true database, then by definition the user has found a good distinguishing query, and we are again in a good case — we can run the database update algorithm to update our hypothesis!

The idea of this algorithm is very simple. We will use a database update algorithm to publicly maintain a hypothesis database. Every time a query arrives, we will classify it as either a hard query, or an easy query. An easy query is one for which the answer given by the hypothesis database is approximately correct, and no update step is needed: if we know that a given query is easy, we can simply compute its answer on the publicly known hypothesis database rather than on the private database, and incur no privacy loss. If we know that a query is hard, we can compute and release its answer using the Laplace mechanism, and update our hypothesis using the database update algorithm. This way, our total privacy loss is not proportional to the number of queries asked, but instead proportional to the number of *hard* queries asked. Because the database update algorithm guarantees that there will not need to be many update steps, we can be guaranteed that the total privacy loss will be small.

Theorem 5.13. OnlineIC is (ε, δ)-differentially private.

Algorithm 11 The Online Iterative Construction Mechanism parameterized by a $T(\alpha)$-database update algorithm U. It takes as input a private database x, privacy parameters ε, δ, accuracy parameters α and β, and a stream of queries $\{f_i\}$ that may be chosen adaptively from a class of queries $\mathcal{Q}$. It outputs a stream of answers $\{a_i\}$.

OnlineIC$_U(x, \{f_i\}, \varepsilon, \delta, \alpha, \beta)$

Let $c \leftarrow T(\alpha)$,
if $\delta = 0$ **then**
  Let $T \leftarrow \frac{18c(\log(2|\mathcal{Q}|)+\log(4c/\beta))}{\epsilon||x||_1}$
else
  Let $T \leftarrow \frac{(2+32\sqrt{2})\sqrt{c\log\frac{2}{\delta}}(\log k+\log\frac{4c}{\beta})}{\epsilon||x||_1}$
end if
Initialize NumericSparse$(x, \{f'_i\}, T, c, \varepsilon, \delta)$ with a stream of queries $\{f'_i\}$, outputting a stream of answers a'_i.
Let $t \leftarrow 0$, $D^0 \in x$ be such that $D^0_i = 1/|\mathcal{X}|$ for all $i \in [|\mathcal{X}|]$.
for each query f_i **do**
  Let $f'_{2i-1}(\cdot) = f_i(\cdot) - f_i(D^t)$.
  Let $f'_{2i}(\cdot) = f_i(D^t) - f_i(\cdot)$
  if $a'_{2i-1} = \perp$ and $a'_{2i} = \perp$ **then**
    Let $a_i = f_i(D^t)$
  else
    if $a'_{2i-1} \in \mathbb{R}$ **then**
      Let $a_i = f_i(D^t) + a'_{2i-1}$
    else
      Let $a_i = f_i(D^t) - a'_{2i}$
    end if
    Let $D^{t+1} = U(D^t, f_i, a_i)$
    Let $t \leftarrow t + 1$.
  end if
end for

Proof. This follows directly from the privacy analysis of NumericSparse, because the OnlineIC algorithm accesses the database only through NumericSparse. □

Theorem 5.14. For $\delta = 0$, With probability at least $1 - \beta$, for all queries f_i, OnlineIC returns an answer a_i such that $|f_i(x) - a_i| \leq 3\alpha$ for any α such that:

$$\alpha \geq \frac{9T(\alpha)(\log(2|\mathcal{Q}|) + \log(4T(\alpha)/\beta))}{\epsilon||x||_1}.$$

Proof. Recall that by Theorem 3.28 that given k queries and a maximum number of above-threshold queries of c, Sparse Vector is (α, β)-accurate for:

$$\alpha = \frac{9c(\log k + \log(4c/\beta))}{\epsilon||x||_1}.$$

Here, we have $c = T(\alpha)$ and $k = 2|\mathcal{Q}|$. Note that we have set the threshold $T = 2\alpha$ in the algorithm. First let us assume that the sparse vector algorithm does not halt prematurely. In this case, by the utility theorem, except with probability at most β, we have for all i such that $a_i = f_i(D^t)$: $|f_i(D) - f_i(D^t)| \leq T + \alpha = 3\alpha$, as we wanted. Additionally, for all i such that $a_i = a'_{2i-1}$ or $a_i = a'_{2i}$, we have $|f_i(D) - a'_i| \leq \alpha$.

Note that we also have for all i such that $a_i = a'_{2i-1}$ or $a_i = a'_{2i}$: $|f_i(D) - f_i(D')| \geq T - \alpha = \alpha$, since $T = 2\alpha$. Therefore, f_i, a_i form a valid step in a database update sequence. Therefore, there can be at most $c = T(\alpha)$ such update steps, and so the Sparse vector algorithm does not halt prematurely. □

Similarly, we can prove a corresponding bound for (ε, δ)-privacy.

Theorem 5.15. For $\delta > 0$, With probability at least $1 - \beta$, for all queries f_i, OnlineIC returns an answer a_i such that $|f_i(x) - a_i| \leq 3\alpha$ for any α such that:

$$\alpha \geq \frac{(\sqrt{512} + 1)(\ln(2|\mathcal{Q}|) + \ln \frac{4T(\alpha)}{\beta})\sqrt{T(\alpha) \ln \frac{2}{\delta}}}{\epsilon||x||_1}$$

We can recover the bounds we proved for online multiplicative weights by recalling that the MW database update algorithm is a $T(\alpha)$-database update algorithm for $T(\alpha) = \frac{4 \log |\mathcal{X}|}{\alpha^2}$. More generally, we have that *any* algorithm in the iterative construction framework can be converted into an algorithm which works in the interactive setting without loss in accuracy. (i.e., we could equally well plug in

the median mechanism database update algorithm or the Perceptron database update algorithm, or any other). Tantalizingly, this means that (at least in the iterative construction framework), there is no gap in the accuracy achievable in the online vs. the offline query release models, despite the fact that the online model seems like it should be more difficult.

5.3 Connections

5.3.1 Iterative construction mechanism and α-nets

The Iterative Construction mechanism is implemented differently than the Net mechanism, but at its heart, its analysis is still based on the existence of small α-nets for the queries C. This connection is explicit for the median mechanism, which is parameterized by a net, but it holds for all database update algorithms. Note that the database output by the iterative database construction algorithm is entirely determined by the at most T functions $f_1, \ldots, f_T \in \mathcal{Q}$ fed into it, as selected by the distinguisher while the algorithm is running. Each of these functions can be indexed by at most $\log |\mathcal{Q}|$ bits, and so every database output by the mechanism can be described using only $T \log |\mathcal{Q}|$ bits. In other words, the IC algorithm itself describes an α-net for $\mathcal{Q}$ of size at most $\mathcal{N}_\alpha(\mathcal{Q}) \leq |\mathcal{Q}|^T$. To obtain error α using the Multiplicative Weights algorithm as an iterative database constructor, it suffices by Theorem 4.10 to take $T = 4 \log |\mathcal{X}|/\alpha^2$, which gives us $\mathcal{N}_\alpha(\mathcal{Q}) \leq |\mathcal{Q}|^{4 \log |\mathcal{X}|/\alpha^2} = |\mathcal{X}|^{4 \log |\mathcal{Q}|/\alpha^2}$. Note that up to the factor of 4 in the exponent, this is exactly the bound we gave using a different α-net in Theorem 4.2! There, we constructed an α-net by considering all collections of $\log |\mathcal{Q}|/\alpha^2$ data points, each of which could be indexed by $\log |\mathcal{X}|$ bits. Here, we considered all collections of $\log |\mathcal{X}|/\alpha^2$ functions in $\mathcal{Q}$, each of which could be indexed by $\log |\mathcal{Q}|$ bits. Both ways, we got α-nets of the same size! Indeed, we could just as well run the Net mechanism using the α-net defined by the IC mechanism, to obtain the same utility bounds. In some sense, one net is the "dual" of the other: one is constructed of databases, the other is constructed of queries, yet both nets are of the same size. We will see the same phenomenon in the

"boosting for queries" algorithm in the next section — it too answers a large number of linear queries using a data structure that is entirely determined by a small "net" of queries.

5.3.2 Agnostic learning

One way of viewing what the IC mechanism is doing is that it is reducing the seemingly (information theoretically) more difficult problem of *query release* to the easier problem of *query distinguishing* or *learning*. Recall that the distinguishing problem is to find the query $f \in \mathcal{Q}$ which varies the most between two databases x and y. Recall that in *learning*, the learner is given a collection of labeled examples $(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \{0, 1\}$, where $y_i \in \{0, 1\}$ is the *label* of x_i. If we view x as representing the *positive examples* in some large data set, and y as representing the *negative examples* in the same data set, then we can see that the problem of distinguishing is exactly the problem of *agnostic learning*. That is, a distinguisher finds the query that best labels the positive examples, even when there is no query in the class that is guaranteed to perfectly label them (Note that in this setting, the same example can appear with both a positive and a negative label — so the reduction still makes sense even when x and y are not disjoint). Intuitively, learning should be an information-theoretically easier problem than query release. The query release problem requires that we release the approximate value of every query f in some class $\mathcal{Q}$, evaluated on the database. In contrast, the agnostic learning problem asks only that we return the evaluation and identity of a single query: the query that best labels the dataset. It is clear that information theoretically, the learning problem is no harder than the query release problem. If we can solve the query release problem on databases x and y, then we can solve the distinguishing problem without any further access to the true private dataset, merely by checking the approximate evaluations of every query $f \in \mathcal{Q}$ on x and y that are made available to us with our query release algorithm. What we have shown in this section is that the reverse is true as well: given access to a private distinguishing or agnostic learning algorithm, we can solve the query release problem by making a small (i.e., only $\log |\mathcal{X}|/\alpha^2$) number of calls to the

private distinguishing algorithm, *with no further access to the private dataset*.

What are the implications of this? It tells us that up to small factors, the information complexity of agnostic learning is equal to the information complexity of query release. Computationally, the reduction is only as efficient as our database update algorithm, which, depending on our setting and algorithm, may or may not be efficient. But it tells us that any sort of information theoretic bound we may prove for the one problem can be ported over to the other problem, and vice versa. For example, most of the algorithms that we have seen (and most of the algorithms that we know about!) ultimately access the dataset by making linear queries via the Laplace mechanism. It turns out that any such algorithm can be seen as operating within the so-called *statistical query* model of data access, defined by Kearns in the context of machine learning. But agnostic learning is very hard in the statistical query model: even ignoring computational considerations, there is no algorithm which can make only a polynomial number of queries to the dataset and agnostically learn conjunctions to subconstant error. For query release this means that, *in the statistical query model*, there is no algorithm for *releasing* conjunctions (i.e., contingency tables) that runs in time polynomial in $1/\alpha$, where α is the desired accuracy level. If there is a privacy preserving query release algorithm with this run-time guarantee, it must operate outside of the SQ model, and therefore look very different from the currently known algorithms.

Because privacy guarantees compose linearly, this also tells us that (up to the possible factor of $\log |\mathcal{X}|/\alpha^2$) we should not expect to be able to privately learn to significantly higher accuracy than we can privately perform query release, and vice versa: an accurate algorithm for the one problem automatically gives us an accurate algorithm for the other.

5.3.3 A game theoretic view of query release

In this section, we take a brief sojourn into game theory to interpret some of the query release algorithms we have (and will see). Let us consider an interaction between two adversarial players, Alice and Bob.

Alice has some set of actions she might take, $\mathcal{A}$, and Bob has a set of actions $\mathcal{B}$. The game is played as follows: simultaneously, Alice picks some action $a \in \mathcal{A}$ (possibly at random), and Bob picks some action $b \in \mathcal{B}$ (possibly at random). Alice experiences a cost $c(a, b) \in [-1, 1]$. Alice wishes to play so as to minimize this cost, and since he is adversarial, Bob wishes to play so as to *maximize* this cost. This is what is called a *zero sum game*.

So how should Alice play? First, we consider an easier question. Suppose we handicap Alice and require that she announce her randomized strategy to Bob before she play it, and allow Bob to respond optimally using this information? If Alice announces that she will draw some action $a \in \mathcal{A}$ according to a probability distribution $\mathcal{D}_A$, then Bob will respond optimally so as to maximize Alice's expected cost. That is, Bob will play:

$$b^* = \arg\max_{b \in \mathcal{B}} \mathbb{E}_{a \sim \mathcal{D}_A}[c(a, b)].$$

Hence, once Alice announces her strategy, she knows what her cost will be, since Bob will be able to respond optimally. Therefore, Alice will wish to play a distribution over actions which *minimizes her cost once Bob responds*. That is, Alice will wish to play the distribution $\mathcal{D}_A$ defined as:

$$\mathcal{D}_A = \arg\min_{\mathcal{D} \in \Delta\mathcal{A}} \max_{b \in \mathcal{B}} \mathbb{E}_{a \sim \mathcal{D}}[c(a, b)].$$

If she plays $\mathcal{D}_A$ (and Bob responds optimally), Alice will experience the lowest possible cost that she can guarantee, with the handicap that she must announce her strategy ahead of time. Such a strategy for Alice is called a *min-max* strategy. Let us call the cost that Alice achieves when playing a min-max strategy Alice's *value* for the game, denoted v^A:

$$v^A = \min_{\mathcal{D} \in \Delta\mathcal{A}} \max_{b \in \mathcal{B}} \mathbb{E}_{a \sim \mathcal{D}}[c(a, b)].$$

We can similarly ask what Bob should play if we instead place *him* at the disadvantage and force him to announce his strategy first to Alice. If he does this, he will play the distribution $\mathcal{D}_B$ over actions $b \in \mathcal{B}$ that *maximizes* Alice's expected cost when Alice responds optimally. We call such a strategy $\mathcal{D}_B$ for Bob a *max-min* strategy. We can define

Bob's value for the game, v^B, as the maximum cost he can ensure by any strategy he might announce:

$$v^B = \max_{\mathcal{D} \in \Delta\mathcal{B}} \min_{a \in \mathcal{A}} \mathbb{E}_{b \sim \mathcal{D}}[c(a, b)].$$

Clearly, $v^B \leq v^A$, since announcing one's strategy is only a handicap.

One of the foundational results of game theory is Von-Neumann's min-max Theorem, which states that in any zero sum game, $v^A = v^B$.[2] In other words, there is no disadvantage to "going first" in a zero sum game, and if players play optimally, we can predict exactly Alice's cost: it will be $v^A = v^b \equiv v$, which we refer to as the value of the game.

Definition 5.7. In a zero sum game defined by action sets $\mathcal{A}, \mathcal{B}$ and a cost function $c : \mathcal{A} \times \mathcal{B} \to [-1, 1]$, let v be the value of the game. An α-approximate min-max strategy is a distribution $\mathcal{D}_A$ such that:

$$\max_{b \in \mathcal{B}} \mathbb{E}_{a \sim \mathcal{D}_A}[c(a, b)] \leq v + \alpha$$

Similarly, an α-approximate max-min strategy is a distribution $\mathcal{D}_B$ such that:

$$\min_{a \in \mathcal{A}} \mathbb{E}_{b \sim \mathcal{D}_B}[c(a, b)] \geq v - \alpha$$

If $\mathcal{D}_A$ and $\mathcal{D}_B$ are both α-approximate min-max and max-min strategies respectively, then we say that the pair $(\mathcal{D}_A, \mathcal{D}_B)$ is an α-approximate Nash equilibrium of the zero sum game.

So how does this relate to query release?

Consider a particular zero sum-game tailored to the problem of releasing a set of linear queries $\mathcal{Q}$ over a data universe $\mathcal{X}$. First, assume without loss of generality that for every $f \in \mathcal{Q}$, there is a query $\hat{f} \in \mathcal{Q}$ such that $\hat{f} = 1 - f$ (i.e., for each $\chi \in \mathcal{X}$, $\hat{f}(\chi) = 1 - f(\chi)$). Define Alice's action set to be $\mathcal{A} = \mathcal{X}$ and define Bob's action set to be $\mathcal{B} = \mathcal{Q}$. We will refer to Alice as the *database player*, and to Bob as the *query player*. Finally, fixing a true private database x normalized to be a probability distribution (i.e., $\|x\|_1 = 1$), define the cost function $c : \mathcal{A} \times \mathcal{B} \to [-1, 1]$

[2] Von Neumann is quoted as saying "As far as I can see, there could be no theory of games ... without that theorem ... I thought there was nothing worth publishing until the Minimax Theorem was proved" [10].

to be: $c(\chi, f) = f(\chi) - f(x)$. Let us call this game the "Query Release Game."

We begin with a simple observation:

Proposition 5.16. The value of the query release game is $v = 0$.

Proof. We first show that $v^A = v \leq 0$. Consider what happens if we let the database player's strategy correspond to the true database: $\mathcal{D}_A = x$. Then we have:

$$\begin{aligned} v^A &\leq \max_{f \in \mathcal{B}} \mathbb{E}_{\chi \sim \mathcal{D}_A}[c(\chi, f)] \\ &= \max_{f \in \mathcal{B}} \sum_{i=1}^{|\mathcal{X}|} f(\chi_i) \cdot x_i - f(x) \\ &= f(x) - f(x) \\ &= 0. \end{aligned}$$

Next we observe that $v = v^B \geq 0$. For point of contradiction, assume that $v < 0$. In other words, that there exists a distribution $\mathcal{D}_A$ such that *for all* $f \in \mathcal{Q}$

$$\mathbb{E}_{\chi \sim \mathcal{D}_A} c(\chi, f) < 0.$$

Here, we simply note that by definition, if $\mathbb{E}_{\chi \sim \mathcal{D}_A} c(\chi, f) = c < 0$ then $\mathbb{E}_{\chi \sim \mathcal{D}_A} c(\chi, \hat{f}) = -c > 0$, which is a contradiction since $\hat{f} \in \mathcal{Q}$. □

What we have established implies that for any distribution $\mathcal{D}_A$ that is an α-approximate min-max strategy for the database player, we have that for all queries $f \in \mathcal{Q}$: $|\mathbb{E}_{\chi \sim \mathcal{D}_A} f(\chi) - f(x)| \leq \alpha$. In other words, the distribution $\mathcal{D}_A$ can be viewed as a synthetic database that answers every query in $\mathcal{Q}$ with α-accuracy.

How about for nonlinear queries? We can repeat the same argument above if we change the query release game slightly. Rather than letting the database player have strategies corresponding to universe elements $\chi \in \mathcal{X}$, we let the database player have strategies corresponding to *databases* themselves! Then, $c(f, y) = |f(x) - f(y)|$. Its not hard to see that this game still has value 0 and that α-approximate min-max strategies correspond to synthetic data which give α-accurate answers to queries in $\mathcal{Q}$.

So how do we compute approximate min-max strategies in zero sum games? There are many ways! It is well known that if Alice plays the game repeatedly, updating her distribution on actions using an online-learning algorithm with a no-regret guarantee (defined in Section 11.2), and Bob responds at each round with an approximately-cost-maximizing response, then Alice's distribution will quickly converge to an approximate min-max strategy. Multiplicative weights is such an algorithm, and one way of understanding the multiplicative weights mechanism is as a strategy for Alice to play in the query release game defined in this section. (The private distinguisher is playing the role of Bob here, picking at each round the query that corresponds to approximately maximizing Alice's cost). The median mechanism is another such algorithm, for the game in which Alice's strategies correspond to databases, rather than universe elements, and so is also computing an approximate min-max solution to the query release game.

However, there are other ways to compute approximate equilibria as well! For example, *Bob*, the query player, could play the game using a no-regret learning algorithm (such as multiplicative weights), and *Alice* could repeatedly respond at each round with an approximately-cost-minimizing database! In this case, the *average* over the databases that Alice plays over the course of this experiment will converge to an approximate min-max solution as well. This is exactly what is being done in Section 6, in which the private base-sanitizer plays the role of Alice, at each round playing an approximately cost-minimizing database given Bob's distribution over queries.

In fact, a third way of computing an approximate equilibrium of a zero-sum game is to have *both* Alice and Bob play according to no-regret learning algorithms. We won't cover this approach here, but this approach has applications in guaranteeing privacy not just to the database, but also to the set of queries being asked, and to privately solving certain types of linear programs.

5.4 Bibliographical notes

The Iterative Construction Mechanism abstraction (together with the perception based database update algorithm) was formalized by

Gupta et al. [39], generalizing the median mechanism of Roth and Roughgarden [74] (initially presented as an online algorithm), the online private multiplicative weights mechanism of Hardt and Rothblum [44], and its offline variant of Gupta et al. [38]; see also Hardt et al. [41]. All these algorithm can be seen to be instantiations. The connection between query release and agnostic learning was observed in [38]. The observation that the median mechanism, when analyzed using the composition theorems of Dwork et al. [32] for (ε, δ) privacy, can be used to answer arbitrary low sensitivity queries is due to Hardt and Rothblum. The game theoretic view of query release, along with its applications to analyst privacy, is due to Hsu, Roth, and Ullman [48].