Sub-linear Queries Statistical Databases: Privacy with Power

Cynthia Dwork

Microsoft Research
dwork@microsoft.com

Abstract. We consider a statistical database in which a trusted administrator introduces noise to the query responses with the goal of maintaining privacy of individual database entries. In such a database, a query consists of a pair (S, f) where S is a set of rows in the database and f is a function mapping database rows to $\{0, 1\}$. The true response is $\sum_{r \in S} f(DB_r)$, a noisy version of which is released. Results in [3, 4] show that a strong form of privacy can be maintained using a surprisingly small amount of noise, provided the total number of queries is sublinear in the number n of database rows. We call this a sub-linear queries (SuLQ) database. The assumption of sublinearity becomes reasonable as databases grow increasingly large.
The SuLQ primitive – query and noisy reply – gives rise to a calculus of noisy computation. After reviewing some results of [4] on multi-attribute SuLQ, we illustrate the power of the SuLQ primitive with three examples [2]: principal component analysis, k means clustering, and learning in the statistical queries learning model.

1 Introduction

Consider a statistical database in which a trusted administrator introduces noise to the query responses with the goal of maintaining privacy of individual database entries. For concreteness, let the database consist of some number n of rows $DB_1, \ldots, DB_n$, where each row is a d-tuple of Boolean values. A query consists of a pair (S, f) where $S \subseteq [n]$ is a set of rows in the database and $f : \{0,1\}^d \to \{0,1\}$ is a function mapping database rows to $\{0, 1\}$. The true response to the query is $\sum_{r \in S} f(DB_r)$, a noisy version of which is released. That is, the administrator algorithm chooses a random quantity in some range and releases the sum of the true response and the random quantity.

Such databases were studied extensively in the early 1980's (see [1] for an excellent survey of results on these and other techniques for statistical disclosure control), with mixed results. However, results in [3, 4] show that a strong form of privacy can be maintained using a surprisingly small amount of noise – a random quantity whose standard deviation is of order $o(\sqrt{n})$ – *provided the total number of queries is sublinear in the number n of database rows.*

This is significant for the following reason. If we think of each row as a sample from some underlying probability distribution and we wish to gather statistics

A.J. Menezes (Ed.): CT-RSA 2005, LNCS 3376, pp. 1–6, 2005.

on a properties P that occur with possibly small but still constant probability in the population, then the sampling error in our population of size n will be of order $\Omega(\sqrt{n})$. Thus, the noise that is added for the sake of protecting privacy is significantly smaller than the sampling error. In other words, providing privacy need not interfere with accuracy, so long as the number of statistical queries is not too large. The assumption of sublinearity is reasonable as databases grow increasingly large.

The basic SuLQ primitive – noisy sums of arbitrary Boolean functions applied to each row in a set $S \subseteq [n]$ of rows – is powerful: statistics for any d-ary predicate can be very accurately obtained simply by querying the database. It is natural to ask, "Which more complex computations can be expressed using few (in n) SuLQ queries?" We have found this class to be quite rich.

Here, we review the results of [4] on multi-attribute SuLQ databases (Section 3) and then give three examples of the power of the SuLQ primitive (Section 4): principal component analysis, k means clustering, and learning in the statistical queries learning model. The treatment here is informal and without proofs. Rigorous treatment of these and other, related, results, is given in [4, 2].

2 Definitions

We model a database as an $n \times d$ binary matrix $DB = \{DB_{i,j}\}$. Intuitively, the columns in DB correspond to Boolean attributes $\alpha_1, \ldots, \alpha_d$, and the rows in DB correspond to individuals, where $DB_{i,j} = 1$ iff attribute α_j holds for individual i.

Let $\mathcal{D}$ be a distribution on $\{0,1\}^d$. We say that a database $DB = \{DB_{i,j}\}$ is chosen according to distribution $\mathcal{D}$ if every row in DB is chosen according to $\mathcal{D}$, independently of the other rows (in other words, DB is chosen according to $\mathcal{D}^n$). To capture partial information that the adversary may have obtained about individuals prior to interacting with the database, this requirement is relaxed in the privacy analysis, allowing each row i to be chosen from a (possibly) different distribution $\mathcal{D}_i$. In that case we say that the database is chosen according to $\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$.

For a Boolean function $f : \{0,1\}^d \to \{0,1\}$ we let $p_0^{i,f}$ be the *a priori* probability that $f(DB_{i,1}, \ldots, DB_{i,d}) = 1$ and $p_T^{i,f}$ be the *a posteriori* probability that $f(DB_{i,1}, \ldots, DB_{i,d}) = 1$, given the answers to T queries, *as well as* all the values in all the rows of DB other than i: $DB_{i'}$ for all $i' \neq i$.

We define the monotonically-increasing 1-1 mapping $\mathrm{conf} : (0,1) \to \mathbb{R}$ as follows:

$$\mathrm{conf}(p) = \log \frac{p}{1-p}.$$

Note that a small additive change in conf implies a small additive change in p [1].

[1] The converse does not hold: conf grows logarithmically in p for $p \approx 0$ and logarithmically in $1/(1-p)$ for $p \approx 1$.

Let $\text{conf}_0^{i,f} = \log \frac{p_0^{i,f}}{1-p_0^{i,f}}$ and $\text{conf}_T^{i,f} = \log \frac{p_T^{i,f}}{1-p_T^{i,f}}$. We write our privacy requirements in terms of the random variables $\Delta\text{conf}^{i,f}$ defined as[2]:

$$\Delta\text{conf}^{i,f} = |\text{conf}_T^{i,f} - \text{conf}_0^{i,f}|.$$

Definition 1 ((δ, T)-Privacy). *A database access mechanism is (δ, T)-private if for every distribution $\mathcal{D}$ on $\{0,1\}^d$, for every row index i, for every function $f : \{0,1\}^d \to \{0,1\}$, and for every adversary $\mathcal{A}$ making at most T queries*

$$\Pr[\Delta\text{conf}^{i,f} > \delta] \leq \mathsf{neg}(n),$$

where $\mathsf{neg}(n)$ grows more slowly than the inverse of any polynomial in n. The probability is taken over the choice of each row in DB according to $\mathcal{D}$, and the randomness of the adversary as well as the database access mechanism.

The definition of (δ, T)-privacy speaks of the probability that any single function experiences a change in confidence. The next definitions speak about sets of functions that together experience little change in confidence.

A *target set* F is a set of d-ary Boolean functions (one can think of the functions in F as being selected by an adversary; they represent information the adversary may wish to learn about someone). A target set F is *δ-safe* if $\Delta\text{conf}^{i,f} \leq \delta$ for all $i \in [n]$ and $f \in F$. Let F be a target set of size polynomial in n. Definition 1 implies that under a (δ, T)-private database mechanism, F is δ-safe with probability $1 - \mathsf{neg}(n)$.

Claim. [4] Consider a (δ, T)-private database with $d = O(\log n)$ attributes. Let F be the target set containing all the 2^{2^d} Boolean functions over the d attributes. Then, $\Pr[F \text{ is } 2\delta\text{-safe}] = 1 - \mathsf{neg}(n)$.

3 Multi-attribute SuLQ Databases

The multi-attribute SuLQ database of [4] is easy to describe. Let $T = T(n) = O(n^c)$, $c < 1$, and define $R = \left(T(n)/\delta^2\right) \cdot \log^\mu n$ for some $\mu > 0$ (taking $\mu = 6$ will work).

SuLQ Database Algorithm $\mathcal{A}$
Input: a query (S, g).

1. Let $a_{S,g} = \sum_{i \in S} g(DB_i)$.
2. Generate a perturbation value: Let $(e_1, \ldots, e_R) \in_R \{0,1\}^R$ and $\mathcal{E} \leftarrow \sum_{i=1}^{R} e_i - R/2$.
3. Return $\tilde{a}_{S,g} = a_{S,g} + \mathcal{E}$.

[2] Our choice of defining privacy in terms of $\Delta\text{conf}^{i,f}$ is somewhat arbitrary, one could rewrite our definitions (and analysis) in terms of the a priori and a posteriori probabilities. Note however that limiting $\Delta\text{conf}^{i,f}$ in Definition 1 is a stronger requirement than just limiting $|p_T^{i,f} - p_0^{i,f}|$.

Note that $\mathcal{E}$ is a binomial random variable with $\mathbf{E}[\mathcal{E}] = 0$ and standard deviation $\sqrt{R}$. The analysis ignores the case where $\mathcal{E}$ largely deviates from zero, as the probability of such an event is extremely small: $\Pr[|\mathcal{E}| > \sqrt{R}\log^2 n] = \mathsf{neg}(n)$. In particular, this implies that the SuLQ database algorithm $\mathcal{A}$ is within $\tilde{O}(\sqrt{T(n)})$ perturbation, meaning that for every query (S, f)

$$\Pr[|\mathcal{A}(S, f) - a_{S,f}| \leq \mathcal{E}] = 1 - \mathsf{neg}(n).$$

The probability is taken over the randomness of the database algorithm $\mathcal{A}$.

Theorem 1. *[4] Let $T(n) = O(n^c)$ and $\delta = 1/O(n^{c'})$ for $0 < c < 1$ and $0 \leq c' < c/2$. Then the SuLQ algorithm $\mathcal{A}$ is $(\delta, T(n))$-private within $\tilde{O}(\sqrt{T(n)}/\delta)$ perturbation.*

Note that whenever $\sqrt{T(n)}/\delta < \sqrt{n}$, restricting the adversary to $T(n)$ queries allows privacy with perturbation magnitude less than $\sqrt{n}$.

Let $i \in [n]$ and $f : \{0,1\}^d \to \{0,1\}$. The proof analyzes the *a posteriori* probability p_ℓ that $f(DB_i) = 1$ given the answers to the first ℓ queries $(\tilde{a}_1, \ldots, \tilde{a}_\ell)$ *and* $DB^{\{-i\}}$ (where $DB^{\{-i\}}$ denotes the entire database except for the ith row). Let $\mathrm{conf}_\ell = \log_2 p_\ell/(1-p_\ell)$. Note that $\mathrm{conf}_T = \mathrm{conf}_T^{i,f}$, and (due to the independence of rows in DB) $\mathrm{conf}_0 = \mathrm{conf}_0^{i,f}$. Following [3], a random walk on the real line is defined, with $\mathrm{step}_\ell = \mathrm{conf}_\ell - \mathrm{conf}_{\ell-1}$. The proof argues that (with high probability) $T(n)$ steps of the random walk do not suffice to reach distance δ.

4 Computation with the SuLQ Primitive

The basic SuLQ operation – query and noisy reply – can be viewed as a noisy computational primitive which may be used to compute other functions of the database than statistical queries. In this section we describe three examples of the power of the primitive. In this setting, the inputs are reals drawn from the unit d-dimensional cube, and the noise is distributed according to a normal variable $N(0, R)$, where $R = R(n)$ is roughly of order $T(n)\log n \log T(n)$. The privacy analysis in the proof of Theorem 1 must be extended accordingly. A rigorous treatment of this work appears in [2].

4.1 Principal Component Analysis

Principal component analysis [6] is an extremely valuable tool in the (frequent) case in which high-dimensional data lies primarily in a low-dimensional subspace.

The input consists of n points in $\mathcal{U}^d$ (the d-dimensional cube of side length 1) and an integer $k \leq d$. The output will be the k largest eigenvalues of the $d \times d$ covariance matrix (defined below), and their corresponding eigenvectors.

For $1 \leq i \leq d$, we let $\mu_i = E_{r\in[n]}[p_r(i)]$,where $p_r(i)$ denotes the ith coordinate of the input point described by row r. We let the $d \times d$ covariance matrix C be defined by $C = \{c_{ij}\}$, where

$$c_{ij} = E_{r\in[n]}[p_r(i)p_r(j)] - \mu_i\mu_j.$$

PCA is known to be remarkably stable under random noise – so much so, that it is often used with the express intention of *removing* noise.

SuLQ Computation of PCA

1. (d queries) For $0 \leq i \leq d$, let $m_i = SuLQ(F(x) := x(i))/n$. By this we mean that $F(x)$ selects the ith coordinate of each row, so the query sums all the ith coordinates (getting a noisy version of this sum), and the algorithm divides this noisy sum by n. This gives an approximation to μ_i in the pure PCA algorithm described above.
2. (Roughly $d^2/2$ queries) Let $c_{ij} = SuLQ(F(x) = x(i)x(j))/n - m_i m_j$. That is, we first obtain a noisy average of the product of the ith and jth coordinates, and then subtract the product of the estimates of μ_i and μ_j.

Given (an approximation to) the covariance matrix C, the k largest eigenvalues and corresponding eigenvectors can be computed directly, without further queries.

We remark that, using the techniques of [4] for vertically partitioned databases, this computation can be carried out even if each column of the database is stored in a separate, independent, SuLQ database.

4.2 *k* Means

An instance of the k means computation is a set of n points in $\mathcal{U}^d$, together with some number k of initial candidate "means" in $\mathcal{U}^d$. The output will consist of k points in $\mathcal{U}^d$ (the "means"), together with the fraction of points in the database associated with each mean. We next describe the basic step of the k means algorithm.

Basic Step of k Means Algorithm

1. (k queries): For each mean m_i, $1 \leq i \leq k$, count the number of points closer to this mean than to every other mean. This yields cluster sizes. This is approximated via the queries, for $1 \leq i \leq k$,

   $$\text{Size}_i = SuLQ(F(x) := 1 \text{ if } m_i \text{ is the closest mean to } x, \text{ and 0 otherwise}).$$

2. (kd queries): for each mean m_i, $1 \leq i \leq k$, and coordinate j, $1 \leq j \leq d$, compute the sum, over all points in the cluster associated with m_i, of the value of the jth coordinate. Divide by the size of the cluster.
   (a) $\text{Sum}_{ij} = SuLQ(F(x) := x(j)$ if m_i is the closest center to x, and 0 otherwise).
   (b) $m_{ij} = \text{Sum}_{ij}/\text{Size}_i$

The basic step is iterated until some maximum number of queries have been issued. (In practice, this usually converges after a small number of basic steps.) If any cluster size is below a threshhold (say, $\sqrt{T(n)}$), then output an exception.

For clusters that are of size $\Omega(n)$, one step of the (pure) k means computation differs from one step of the SuLQ-based k means computation by a quantity that is roughly Gaussian with mean zero and variance $\tilde{O}(\sqrt{R}/n)$.

4.3 Capturing the Statistical Queries Learning Model

The Statistical Queries Learning model was proposed by Kearns [5]. In this model the goal is to learn a *concept* $c : \{0,1\}^d \to \{0,1\}$. There is a distribution D on strings in $\{0,1\}^d$, and the learning algorithm has access to an oracle, $\text{stat}_{c,D}$, described next.

On query (f, τ), where $f = f(x, \ell)$ is any boolean function over inputs $x \in D$ and label $\ell \in \{0,1\}$, and $\tau = 1/poly(d)$ is an error tolerance, the oracle replies with a noisy estimate of the probability that $f(x, c(x)) = 1$ for a randomly selected element from D; the answer is guaranteed to be correct within additive tolerance τ. Many (but not all, see [5]) concept classes that are PAC learnable can also be learned in the statistical queries learning model.

To fit the statistical queries learning model into our setting, we require that one of the attributes be the value of c applied to the other data in the row, so that a typical row looks like $DB_r = (x, c(x))$. By definition, on input (f, S) the SuLQ database responds with a noisy version of $\sum_{r \in S} f(DB_r)$. Taking $S = [n]$, we have that so long as the noise added by the SuLQ database is within the tolerance τ, the response (divided by n) is a "valid" response of the $\text{stat}_{c,D}$ oracle. In other words, to simulate the query $\text{stat}_{c,D}(f, \tau)$ we compute $\text{SuLQ}(F(x) := f(x))/n$ a total of $\tilde{O}(R/\tau^2 n^2)$ times and return the average of these values.

With high probability the answer obtained will be within tolerance τ. Also, recall that $\tau = 1/poly(d)$; if $d = n^{o(1)}$ then repetition is not necessary.

References

1. N. R. Adam and J. C. Wortmann, Security-Control Methods for Statistical Databases: A Comparative Study, *ACM Computing Surveys 21*(4), pp. 515–556, 1989.
2. A. Blum, C. Dwork, F. McSherry, and K. Nissim, On the Power of SuLQ Databases, *manuscript in preparation*, 2004.
3. I. Dinur and K. Nissim, Revealing information while preserving privacy, *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 202-210, 2003.
4. C. Dwork and N. Nissim, Privacy-Preserving Datamining on Vertically Partitioned Databases, *Proceedings of CRYPTO 2004*
5. M. Kearns, Efficient Noise-Tolerant Learning from Statistical Queries, *JACM 45*(6), pp. 983–1006, 1998. See also *Proc. 25th ACM STOC*, pp. 392–401, 1993
6. M. J. O'Connel, Search Program for Significant Variables, *Comp. Phys. Comm. 8*, 1974.