Matrix Representations for Linear Queries in Statistical Databases

Ryan McKenna

June 5, 2018

1 Introduction

Many differential privacy algorithms make heavy use of the data vector representation of the database. This object has one cell for each possible tuple in the domain, and counts the number of occurrences of that tuple in the database. A linear query is just some linear transformation of this data vector. Answering linear queries is a common task considered in the privacy literature, in part because tight bounds on the sensitivity of a collection of linear queries are readily available and easily computable from the query matrix.

For theoretical reasons it is often convenient to think of a query as a vector q and a collection of queries as a matrix Q with one query per row. It is common for algorithms to use this represenation in practice as well, in part because this representation is useful for computing sensitivity and performing inference, which are essential components of many privacy algorithms. It is also common for algorithms to *avoid this representation*, because it is too computationally expensive for large domain sizes. These algorithms have to find other ways to compute sensitivity and perform inference.

2 Matrix Representations

There are a variety of ways to represent a collection of linear queries. Some representations are general enough to encode any linear query, while others have less expressive power but may be able to represent a certain subclass of queries more compactly. Choosing the right matrix representation can have a big impact on the efficiency of answering a collection of linear queries. In these section, we enumerate the representations that have appeared in the literature, and discuss their tradeoffs.

There are a number of important computations that a matrix representation should support, listed below:

Dense Matrix A dense matrix stores $m * n$ values, one for each query and cell of the domain. A dense matrix can encode any collection of linear queries,

Description	Computation
L_1 Sensitivity	$\Delta = \max_j \|\|A_{\cdot j}\|\|_1$
L_2 Sensitivity	$\Delta = \max_j \|\|A_{\cdot j}\|\|_2$
Matrix-Vector product	$y = Ax$
Vector-Matrix product	$x = yA$
Gram Matrix	$X = A^T A$
Least Squares	$\hat{x} = \arg\min_x \|\|Ax - y\|\|_2$
Domain Reduction	$A' = AP$

and it supports all of the above computations.

Sparse Matrix A sparse matrix stores only the nonzero values. It can still encode an arbitrary collection of linear queries, but the efficiency depends on the number of nonzero entries. If $nnz(A) \approx m*n$ then there is not much benefit to the sparse represenation (and the overhead makes the dense represenatation preferable). However, if $nnz(A) << m * n$ there may be large improvements to performance in using this representation. A sparse matrix supports all of the above computations, with the caveat that the least squares problem is most effectively solved using an iterative method such as LSQR (as opposed to a direct method that computes the pseudo inverse or factorizes the matrix).

Range Intervals A range query can be compactly represented as a tuple (i, j), which encodes the query q where $q_k = 1$ if $i \leq k \leq j$. This representation was used in the empirical study DPBench, and it is far more compact than sparse and dense matrices, but it limited in its expressive capability (it can only encode range queries). It is straightforward to calculate the L_1 and L_2 sensitivity, as well as compute matrix-vector products in this representation. The least squares problem can also be solved using LSQR.

With small modifications, we can encode an arbitrary collection of queries. A *union of weighted range intervals* is expressive enough to encode any query, however this representation offers the most benefit (in terms of space usage) for queries that are a union of k things where k is small (e.g., for range queries $k = 1$). This representation has the same benefits as the simpler one – we can compute sensitivity, compute matrix-vector products, etc.

Matrix-Free Linear Transformation Sometimes it is possible (and desirable) to replace an explicit matrix-based representation of a collection of queries with an implicit, matrix-free representation. In order to do this, routines must be specified for computing matrix-vector products, vector-matrix products, and all of the routines listed above. There are a number of query sets where it makes sense to represent them this way.

For example, the hierarchical queries considered by Hay et al. and Qardaji et al. can be represented in a matrix-free fashion, and the routines for computing matrix-vector products, vector-matrix products, and least squares are all linear

in the number of queries (as opposed to quadratic or cubic for the dense matrix representation).

Another example is the prefix queries. These can (and should) be represented in a matrix-free fashion, because the key operations above can be computed in $O(n)$ time as opposed to $O(n^2)$ or $O(n^3)$ time for the sparse and dense reprepsentation of this query set. The Range Interval representation also works for this query set, but computing matrix-vector products still requires $O(n^2)$ time using this representation, even though the required space is only $O(n)$. Additionally, there is no generic formula for solving the least squares problem under the range interval representation, but for prefix queries there is a simple $O(n)$ algorithm for solving it that should be exploited and used.

Another example is the haar wavelet matrix. Linear-time algorithms exist for performing all the key operations, so these specialized algorithms should be used in over the standard algorithms for dense and sparse matrices.

Example: An interesting example collection of queries that could benefit from this representation is an arbitrary collection of range queries. We already discussed a possible representation for these query sets that is better than dense and sparse matrices, but we can more efficiently compute matrix-vector and vector-matrix products using the following observation: *any range query can be expressed as the difference of two Prefix queries.* Thus, a collection of range queries, Q can be writtten as $Q = SP$ where P is the prefix queries and S is a sparse matrix with two nonzero entries per row. We can evalute matrix-vector products $Qx = S(Px)$ using the special $O(n)$ algorithm for computing $z = Px$; computing Sz is also linear in the number of queries since it is so sparse.

Kronecker Product A Kronecker product is capable of representing a cartesian product of predicate counting queries where each predicte is a conjunction of conditions on individual attributes (or disjoint subsets of attributes). It is actually capable of encoding more than that because the entries of the matrix do not have to be binary, but it is difficult to intuitively define precisely the query sets that can be represented this way.

For $n \times n$ matrices A_i, the matrix $A = A_1 \otimes \cdots \otimes A_d$ requires $O(n^{2d})$ space to store in a dense format, but it only requires $O(dn^2)$ space to store the tuple $(A_1, \ldots, A_d)$. Operations on A can often be written in terms of $A_1, \ldots, A_d$ (e.g., matrix multiplication, matrix inversoin, singular value decomposition, sensitivity calculation, etc.).

Marginals