Time Complexity and Parallel Speedup to Compute the Gamma Summarization Matrix

Carlos Ordonez, Yiqun Zhang

Department of Computer Science, University of Houston, USA

Abstract. We study the serial and parallel computation of Γ (Gamma), a comprehensive data summarization matrix for linear Gaussian models, widely used in big data analytics. Computing Gamma can be reduced to a single matrix multiplication with the data set, where such multiplication can be evaluated as a sum of vector outer products, which enables incremental and parallel computation, essential features for scalable computation. By exploiting Gamma, iterative algorithms are changed to work in two phases: (1) Incremental-parallel data set summarization (i.e. in one scan and distributive); (2) Iteration in main memory exploiting the summarization matrix in intermediate matrix computations (i.e. reducing number of scans). Most intermediate computations on large matrices collapse to computations based on Gamma, a much smaller matrix. We present specialized database algorithms for dense and sparse matrices, respectively. Assuming a distributed memory model (i.e. shared-nothing) and a larger number of points than processing nodes, we show Gamma parallel computation has close to linear speedup. We explain how to compute Gamma with existing database systems processing mechanisms and their impact on time complexity.

1 Introduction

Machine learning models generally require demanding iterative computations on matrices. This problem becomes significantly more difficult when input matrices (i.e. the data set) are large, residing on secondary storage, and when they need to be processed in parallel. We propose the Γ (Gamma) data summarizaton matrix [7] for linear Gaussian models [8]. From a computational perspective Γ has outstanding features: it produces an exact solution, it can be incrementally computed in one pass and it can be computed in parallel, with low communication overhead in a distributed memory model (i.e. shared-nothing [1]). We show computing Γ can be reduced to a single matrix self-multiplication. We study time and space complexity and parallel speedup of such multiplication.

2 Definitions

We first define X, the input data set. Let $X = \{x_1, ..., x_n\}$, where x_i is a column-vector in $\mathbf{R}^d$. In other words, X is a $d \times n$ matrix; intuitively, X can be visualized

as a wide rectangular matrix. Supervised (predictive) models require an extra attribute. For regression models X is augmented with a $(d+1)$th dimension containing an output variable Y. On the other hand, for classification models, there is an extra discrete attribute G, where G is in general binary (e.g. false/true, bad/good). We use $i = 1 \ldots n$ and $j = 1 \ldots d$ as matrix subscripts. The model, a "bag" of vectors and matrices, is called Θ. Therefore, Θ can represent principal component analysis (PCA) [4], linear regression [5], K-means clustering [3] and Naive Bayes classification [6], among others.

3 The Gamma Data Summarization Matrix

We introduce a general augmented matrix $\mathbf{Z}$, by prefixing it with a row of 1s and appending a $[d+2]$th row to $\mathbf{X}$, containing vector Y (in transposed form). Since X is $d \times n$, $\mathbf{Z}$ is $(d+2) \times n$, where row $[1]$ are 1s and row $[d+2]$ is Y. For classification and clustering models X is partitioned using a discrete attribute G (e.g. class 0/1, cluster $1 \ldots k$). In this paper, we focus on $\mathbf{Z}$ extended with Y.

Multidimensional sufficient statistics [4] summarize statistical properties of a data set: $n = |X|$, $L = \sum_{i=1}^{n} x_i$, $Q = XX^T$. Intuitively, n counts points, L is a simple "linear" sum (a vector) and Q is a "quadratic" matrix. We now introduce Γ, our main contribution, which generalizes previous sufficient statistics.

$$\Gamma = \begin{bmatrix} n & L^T & \mathbf{1}^T \cdot Y^T \\ L & Q & XY^T \\ Y \cdot \mathbf{1} & YX^T & YY^T \end{bmatrix} \qquad = \begin{bmatrix} n & \sum x_i^T & \sum y_i \\ \sum x_i & \sum x_i x_i^T & \sum x_i y_i \\ \sum y_i & \sum y_i x_i^T & \sum y_i^2 \end{bmatrix}$$

The fundamental property of Γ is that it can be computed by a single matrix multiplication with $\mathbf{Z}$: $\Gamma = \mathbf{Z}\mathbf{Z}^T$. Notice $\mathbf{Z}\mathbf{Z}^\mathbf{T} \neq \mathbf{Z}^\mathbf{T}\mathbf{Z}$, a property of Gramian matrices [2]. Matrix Γ is fundamentally the result of "squaring" matrix $\mathbf{Z}$. Γ is much smaller than X for large n: $O(d^2) \ll O(dn)$, symmetric and computable via vector outer products. The property below gives two equivalent forms to summarize the data set: one based on matrix-matrix multiplication and a second one based on vector-vector multiplication, which is more efficient for incremental and parallel processing.

Proposition: Γ can be equivalently computed as follows:

$$\Gamma = \mathbf{Z}\mathbf{Z}^T = \sum_{i=1}^{n} z_i \cdot z_i^T. \tag{1}$$

4 Serial Computation

Recall from Section 2 that Θ represents a model. We start by presenting the standard iterative algorithm to compute a model Θ in two phases. Phase 1: initialize Θ; Phase 2: Iterate until covergence (i.e. until Θ does not change)

reading X from secondary storage at every iteration (once or several times). Our proposed equivalent optimized iterative 2-phase algorithm based on Γ follows. Phase 1: Compute Γ, updating Γ in main memory reading X from secondary storage. Initialize Θ from Γ; Phase 2: Iterate until convergence exploiting Γ in main memory in intermediate matrix computations.

Lemma (Succinctness–asymptotically small size): Given a large data set X where $d \ll n$, matrix Γ size is $O(d^2) \ll O(dn)$ as $n \to \infty$.

Our time and space analysis is based on the 2-phase algorithm introduced above. Phase 1, which corresponds to the summarization matrix operator, is the most important. Computing Γ with a dense matrix multiplication is $O(d^2n)$. On the other hand, computing Γ with a sparse matrix multiplication is $O(k^2n)$ for the average case assuming k entries from x_i are non-zero; assuming X is hyper-sparse $k^2 = O(d)$ then the time is $O(dn)$ on average. In Phase 2 we take advantage of Γ to accelerate computations involving X. Since we are computing matrix factorizations derived from Γ time is $\Omega(d^3)$, which for a dense matrix it may approach $O(d^4)$, when the number of iterations in the factorization numerical method is proportional to d. In short, time complexity for Phase 2 for the models we consider does not depend on n. I/O cost is just reading X with the corresponding number of I/Os in time $O(dn)$ for dense matrix storage and $O(kn)$ for sparse matrix storage.

5 Parallel Computation

Let N be the number of processing nodes, under a distributed memory architecture (shared-nothing [1]). That is, each node has its own main memory and secondary storage. We assume $d \ll n$ and $N \ll n$, but d is independent from N: $d \ll N$ or $N \ll d$. Our parallel algorithm follows. Phase 1 with N nodes: Compute Γ in parallel, updating Γ in main memory reading X from secondary storage. Phase 2 with 1 node: Iterate until convergence exploiting Γ in main memory in intermediate matrix computations to compute Θ.

Recall we assume $d < n$ and $n \to \infty$. For Phase 1 each $x_i \in X$ is hashed to N processors. Then our matrix operator can compute each outer product $z_i \cdot z_i^T$ in parallel, resulting in $O(d^2n/N)$ work per processor for a dense matrix and $O(dn/N)$ work per processor for a sparse matrix. That is, both dense and sparse matrix multiplication algorithms become optimal as N increases or $n \to \infty$. Given its importance from a time complexity point of view, we present the following result as a theorem:

Theorem (linear speedup): Let T_j be processing time using j nodes, where $1 \leq j \leq N$. Under our main assumption and Θ fits in main memory then our optimized algorithm gets close to optimal speedup $T_1/T_N \approx O(N)$.

For completeness, we provide an analysis of a parallel system where N is large. That is, $N = O(n)$.

Lemma (sequential communication bottleneck): Assume only one node can receive partial summarizations. Then time complexity with N messages to a single coordinator is $O(d^2n/N + Nd^2)$.
Lemma (improved communication: global summarization with a balanced binary tree): Assume partial results $\Gamma^{[I]}$ can be sent to other workers in a binary tree fashion. Then $T(d, n, N) \geq O(d^2N + \log_2(N)d^2)$ (a lower communication bound).

6 Computing Γ in a Database System

Here we connect our matrix-based parallel computation with a database system with: (1) relational queries; (2) extending it with matrix functions. We assume X can be evenly partitioned by i, a reasonable assumption when $d \ll n$.

We now treat X as a table in order to process with relational algebra, the formalization of SQL. In order to simplify presentation we use X as the table name and we treat Y as another dimension. There are two major alternatives: (1) Defining a table with d columns, plus primary key i $X(i, X_1, X_2, \ldots, X_d)$, ideal for dense matrices. (2) Defining a table with triples, using the matrix subscripts as primary key: $X(i, j, v)$, where i means point id, j dimension subscript and v is the matrix entry value, excluding zeroesm, where $v = 0$ (i.e. sparse matrix). Alternative (1) requires expressing matrix product as d^2 aggregations, which requires a wrapper program to generate the query. Notice X_j is a column name: we cannot assume there is subscript-based access. Alternative (2) can express the computation as a single query, making the solution expressible within relational algebra and it is natural for sparse matrices. In the following query $X1$ and $X2$ are "alias" of X because it is referenced twice: $\Gamma = \pi_{X1.j,X2.j,sum(X1.v*X2.v)}(X1 \bowtie_{X1.i=X2.i} X2)$. The time complexity to evaluate the Gamma query depends on the join algorithm. Therefore, it can be: $O(d^2n)$ (hash join, balanced buckets), $O(d^2n\log(n))$ (sort-merge join, merge join), to $O(d^2n^2)$ (nested loop join). These bounds can be tighter if the join operation can be avoided using a subscript based mechanism in main memory when Γ fits in main memory.

The previous query solution requires an expensive join ($\bowtie$), which can be slow. Therefore, our goal is to eliminate the join operation by enabling efficient subscript-based access. Our proposal is to introduce a matrix function in an extended database model, where tuples are transfomed into vectors and a matrix can be the result of the query. This function bridges relational tables on one side with matrices on the other side. Let $Gamma()$ be an aggregation which returns a $(d + 2) \times (d + 2)$ matrix. There are two elegant solutions: (1) $\Gamma = Gamma(X_1, X_2, \ldots, X_d)(X)$ for X with a "horizontal" schema. (2) $\Gamma = Gamma(X)$ for a "vertical" schema, where the function assume X has the schema defined above. In this case time complexity ranges from $O(d^2n)$ to $O(d^2n\ \log(n))$, depending on the aggregation algorithm. The price we are paying is that the query is no longer expressible within relational algebra and therefore it is not feasible to treat Γ as a new relational operator. This second mechanism corresponds to so-called SQL UDFs [4, 9] or user-defined array operators [7, 10].

References

1. D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35(6):85–98, 1992.
2. G.H. Golub and C.F. Van Loan. *Matrix computations*. Johns Hopkins University, 3rd edition, 1996.
3. C. Ordonez. Integrating K-means clustering with a relational DBMS using SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 18(2):188–201, 2006.
4. C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(12):1752–1765, 2010.
5. C. Ordonez, C. Garcia-Alvarado, and V. Baladandayuthapani. Bayesian variable selection in linear regression in one pass for large datasets. *ACM TKDD*, 9(1):3, 2014.
6. C. Ordonez and S. Pitchaimalai. Bayesian classifiers programmed in SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(1):139–144, 2010.
7. C. Ordonez, Y. Zhang, and W. Cabrera. The Gamma matrix to summarize dense and sparse data sets for big data analytics. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2016.
8. S. Roweis and Z. Ghahramani. A unifying review of linear Gaussian models. *Neural Computation*, 11:305–345, 1999.
9. M. Stonebraker, D. Abadi, D.J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. MapReduce and parallel DBMSs: friends or foes? *Commun. ACM*, 53(1):64–71, 2010.
10. M. Stonebraker, P. Brown, D. Zhang, and J. Becla. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science and Engineering*, 15(3):54–62, 2013.