Fast PCA and Bayesian Variable Selection for Large Data Sets Based on SQL and UDFs [*]

Mario Navas
University of Houston
Dept. of Computer Science
Houston, TX 77204, USA

Carlos Ordonez
University of Houston
Dept. of Computer Science
Houston, TX 77204, USA

Veerabhadran Baladandayuthapani
UT MD Anderson
Houston, TX 77030, USA

ABSTRACT

Large amounts of data are stored in relational DBMSs. However, statistical analysis is frequently performed outside the DBMS using statistical tools, such as the well-known R package, leading to slow processing when data sets cannot fit in main memory and going through a file export bottleneck. In this article, we propose algorithms for large data set processing of principal component analysis (PCA) and stochastic search variable selection (SSVS) that can work entirely inside a DBMS, using SQL queries and User-Defined Functions (UDFs). Both of our algorithms consist of two main phases: a first phase to compute sufficient statistics in one pass with SQL queries and a second one to derive the model from such such sufficient statistics, in main memory with UDFs. PCA is efficiently solved with SVD via UDFs in main memory after sufficient statistics are derived. On the other hand, the traditional SSVS algorithm requires multiple passes to compute a model. In contrast, our improved Bayesian algorithm performs a single table scan on the input data set and then the UDF performs thousands of iterations on small matrices. In addition, we incorporate optimizations that exploit DBMS multi-threaded processing capabilities to compute multidimensional aggregates in data summarization. Specifically, we present low-level optimizations to distribute the workload among multiple cores, accessing records by block and caching in main memory. Experiments with large data sets results demonstrate the efficiency of our optimizations to compute sufficient statistics and show our algorithms have linear scalability on the size of the data set. Finally, a detailed comparison against R, the standard open-source package for statistical research, shows correctness and superior speed of our DBMS-based algorithms to process very large datasets.

[*]This work was partially supported by NSF grants CCF 0937562 and IIS 0914861.

Categories and Subject Descriptors

G.1.3 [**Mathematics of Computing**]: Numerical Linear Algebra—*Singular value decomposition*; G.3 [**Probability and Statistics**]: Multivariate statistics; H.2.8 [**Database Management**]: Database Applications—*Data mining*

Keywords

PCA, Bayesian statistics, variable selection, SQL

1. INTRODUCTION

The efficient computation of very large datasets is a major concern of data mining, machine learning and statistics. Exploratory techniques, such as principal component analysis (PCA), are widely used to evaluate attributes of a dataset. PCA not only finds linear hidden relationships between variables, but also it is used for unsupervised dimensionality reduction. On the other hand, when the purpose is to find the explanation for certain output, variable selection is required. The traditional methodology is stepwise regression, where one variable is added or removed from the model at a time. The best set of explanatory variables is selected after an exhaustive search through all possible combination of variables, which can be computationally demanding and often untenable in large datasets with many variables. In contrast, stochastic search variable selection (SSVS) evaluates models based on a stochastic probabilistic search over the possible model configurations best explaining the output variable. Based on the configuration yielding the highest posterior probabilities, the best model can be evaluated without conducting an exhaustive search over the model space.
Even though, DBMSs are extremely fast to process large amounts of data, the majority of work in statistics and data mining focuses on processing flat files outside the DBMS with statistical packages like R. The main limitations for external tools are disk I/O and the amount of the data they can handle. Moreover, exporting large datasets for analysis is time consuming, and it compromises security of sensitive data. Nowadays, specific functionality can be attached to modern DBMSs [7]. Furthermore, DBMSs provide application programming interfaces (APIs) [4] for their user-defined functions (UDFs) [16] that support multi-threading. Our contributions are to extend the DBMS functionality with a combination of SQL and UDFs for data mining operations, while exploiting the efficient computation of aggregate functions [22]. We focus on computing PCA and SSVS[10] with only one scan over the input table, and maintaining exact results. The algorithms proposed split the process in

the major steps of data summarization with aggregate functions, and computing the model using the summarization matrices. Even though we consider only data residing in a DBMS, our algorithm can be implemented in other storage systems with programming languages like C, or tools like MapReduce [29]. We also explore caching, efficient memory manage, and multi-threading in order to make the most of current state of hardware technology.
This paper is organized as follows. Section 2 presents an overview of definitions. Section 3 discusses our algorithms for efficiently computing PCA and SSVS for large datasets. Experimental results, in Section 4, evaluate accuracy and performance of different methodologies when integrated to a DBMS. Section 5 discloses related work. Finally, Section 6 concludes the article.

2. PRELIMINARIES

2.1 Definitions

Assume the input data $X = \{x_1, \ldots, x_n\}$, with d dimensions and n data points x_i as column vectors. Consequently, the matrix X has size $d \times n$. The response Y, or variable of interest, is a matrix $1 \times n$. To indicate matrix transposition we use the superscript T. The a^{th} dimension is referred with the subscript X_a, while a superscript γ^I is used for the I^{th} iteration in stochastic search variable selection, and $X^{(I)}$ for the I^{th} iteration in singular value decomposition. We will use the terminology 1_n to represent a column unit vector of n elements (as in [18]). Finally, $\mathcal{X}$ is used for the augmented version of X in linear regression.

2.2 Data Summarization

There are three summarization computations shown to be essential for data mining and statistical models [13]: n, L and Q (see Equation 1, 2). The value n is a count of points in the input data set X. L is the linear sum of the d dimension in X, and forms a vector of d values. Since the matrix Q is the quadratic sum of the cross-products of points, it has a size $d \times d$. Y can be included in the data summarization to compute: XY^T, YY^T, and $Y1_n$. Let the data set X be stored in a table inside the DBMS, which has an attribute or column for each dimension, and a row entry for every data point. Also, the output Y is stored as an additional attribute in the same table. Only one scan over the input data is needed to obtain the data summarization. Moreover, aggregations in n, L and Q are distributive [14]; thus, they can be computed in parallel over different sections of the data, where the global aggregation is given by the addition of all partial results.

$$L = X1_n = \sum_{i=1}^{n} x_i \tag{1}$$

$$Q = XX^T = \sum_{i=1}^{n} x_i \cdot x_i^T \tag{2}$$

Since the summary matrices can be computed efficiently with aggregate functions, they are used in data mining to generate several statistical models [22]. Our focus is to exploit the summarization matrices n, L and Q for dimensionality reduction, and variable selection of large datasets.

2.3 Dimensionality Reduction with PCA

Principal Component Analysis (PCA) is the most popular statistical technique used for dimensionality reduction. PCA is a set of linear combinations or vectors, each of them associated to the variability of data it holds. Therefore, by selecting the top k vectors, we can project the data into a space with less dimensions than the original data while maintaining the most of the information present in the original data. The linear combination of PCA can be found by performing eigenvalue decomposition on either the covariance or the correlation matrices. Both of this matrices are $d \times d$ matrices which values are measurements of the linear dependency between two dimensions of the original data. Equation 3 shows the eigenvalue decomposition problem of PCA. Using the covariance or correlation matrix for XX^T, the problem is to find a set of eigenvectors U and the set of eigenvalues in the diagonal elements of Λ^2. Consequently, the linear transformation U^T can be applied to the original data set to have its projection in the space of its principal components. Moreover, the subset of k vectors of U^T, with the largest eigenvalues, will be the linear combination that maximizes the variance kept of the original data in terms of least square error.

$$XX^T = U\Lambda^2U^T \tag{3}$$

Since PCA is a tool with exploratory purposes, it helps finding hidden relationships between attributes or variables of the input data. However, many statistical applications require finding the relationship between the explanatory variables and certain response. We avoid having to evaluate all the possible subsets of explanatory variables by doing a stochastic search based on Markov chains, and the Gibbs sampler [11].

2.4 Bayesian Variable Selection

Linear regression models the relationship between an output or response variable Y, and a set a set of explanatory variables or estimators X. The regression model $Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_d X_d + \epsilon$, is defined by $\beta = (\beta_0, \beta_1, \ldots, \beta_d)$, the regression coefficients and a Gaussian white noise error ϵ. Since increasing d (the number of estimators) not necessarily increases the quality of the model, the problem of variable selection is to find a set of explanatory variables that best predicts the output variable Y, among the 2^d possible subsets of X. In variable selection via Gibbs sampler [11], a model $\mathcal{M}_\gamma$ is represented by a $\gamma \in \Gamma = \{0,1\}^d$, where the variable X_a is part of the model if $\gamma_a = 1$. The estimation for the probability distribution $\pi(\gamma|Y, X)$ obtained by convolving the likelihood and the prior distribution is computed with the number of appearances of each model $\mathcal{M}_\gamma$ taken S iterations. Therefore, stochastic search variable selection (SSVS) generates a sequence γ^I, where $I = 1, \ldots, S$. Given that $\gamma^I_{(a)} = (\gamma^I_1, \ldots, \gamma^I_{a-1}, \gamma^{I-1}_{a+1}, \ldots, \gamma^{I-1}_d)$, the sample γ^I is obtained by evaluating the probabilities of $\gamma^I_a = 0$ and $\gamma^I_a = 1$ for each explanatory variable a, using the probability function $\pi(\gamma^I_a|Y, X, \gamma^I_{(a)})$.
In contrast to stepwise variable selection which does an exhaustive search to find the model that better explains the output variable, SSVS is a stochastic search which finds this subset of variables based on posterior probabilities. Our contribution pursues to compute SSVS using summarization matrices, without more than one pass over the input dataset.

Operations are done completely inside the DBMS with aggregate functions, and extensibility alternatives; thus, resulting in high gains in computational efficiency.

2.5 DBMS Extensibility

Most DBMSs provide alternatives to attach code or functionality to the database. We assume the capability to either implement aggregate functions, store procedures, or table valued functions not programmed in standard SQL, yet with C or C-like languages. By generalizing database connectivity features to different programming languages, we expect such APIs to include tools for: (1) executing SQL queries, and (2) accessing results sets. A result set gives access to rows in a table one at a time. Consequently, records are read sequentially and oblivious of the size of the table until reaching the end of the result set. In order to return values of a computation, we can use SQL statements to create one or various tables storing results. However, there are also table-valued functions (TVFs) [4], which return a table instead of a single value.
The set of functions that need to be implemented by the user to define an aggregate user-defined function (UDF) are [28]: initialize, accumulate, merge, and terminate. With the definition of a UDF, several working threads are created. All threads compute an aggregation on a partition of the input table. The global aggregation is obtained by merging the aggregations on pairs, its elements are used to compute the value of the function [7]. Aggregate UDFs are a powerful framework to control multi-threading, and leaving to the DBMS the control of the working threads However, the user is constrained to the parameterization defined by the interface, and little can be done to introduce specific optimization driven to the optimal use of hardware resources. Finally, aggregate functions in the DBMS are used to compute the summarization matrices efficiently, and extensibility alternatives to compute PCA and SSVS, completely inside the DBMS.

3. ALGORITHMS

In this section we propose the algorithms to compute PCA and Bayesian variable selection. The algorithms are divided in two main phases: (1) summarize the input data with n, L, Q, and (2) compute the specific model. The algorithms developed exploit aggregate functions of DBMSs to compute sufficient statistics. Furthermore, operations in PCA and SSVS are expressed in terms of the summary matrices n, L, and Q, so only one scan to the input table is needed to obtain the exact model. We include optimizations for large scale processing inside the DBMSs with SQL and UDFs. Finally, we present optimizations to exploit hardware with multi-threading, accessing records by blocks, and caching in memory.

3.1 Efficient Computation of PCA

PCA can be computed using the covariance or the correlation matrix of the input data. The loading vectors of PCA are obtained by solving the eigenvalue decomposition of either of these two matrices. We use the covariance Σ or the correlation matrix ρ_{ab}, since Σ is equivalent to XX^T when X is centered to the mean, and when X is standardized with a z-score it is equivalent to ρ_{ab}. In order to have only one scan over the input dataset, we use the summarization matrices n, L, and Q to express the covariance matrix

```
Algorithm 1: Principal Component Analysis
Input: Table X
1 : Compute data summarization: $n$, $L$, and $Q$ .
2 : Obtain $\Sigma$ or $\{\rho_{ab}\}$ from $n$, $L$, $Q$.
3 : $A^{(0)} \leftarrow \Sigma$ or $A^{(0)} \leftarrow \{\rho_{ab}\}$
4 : $(U, \Lambda^2) \leftarrow \mathrm{SVD}(A^{(0)})$
Return: loadings $U_a$, and variances $\Lambda_{a,a}$
```

as in Equation 4, or the correlation matrix as in Equation 5. To solve the eigenvalue decomposition problem we use an algorithm based on the QR factorization.

$$\Sigma = \frac{1}{n}Q - \frac{1}{n^2}LL^T \tag{4}$$

$$\rho_{ab} = \frac{nQ_{ab} - L_aL_b}{\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2}} \tag{5}$$

The steps to compute singular value decomposition or SVD: (1) Refer to Σ or $\{\rho_{ab}\}$ as $A^{(0)}$, and Apply Householder for reducing the correlation matrix to an equivalent tridiagonal matrix, with a linear transformation T. $B^{(0)} = T^TA^{(0)}T$, where $T^{-1} = T^T$. (2) Find the eigenvalue decomposition of the tridiagonal matrix $B^{(0)} = C^{(S)}B^{(S)}C^{(S)T}$ by applying the QR factorization method, where the diagonal of $B^{(S)}$ is the matrix of eigenvalues, $C^{(S)}$ is the orthogonal matrix and S is the number of iterations taken to satisfy the error criterion. (3) Finally, we can combine both decompositions to get $A^{(0)} = (TC^{(S)})B^{(S)}(TC^{(S)})^T$. As a result, the diagonal matrix of eigenvalues Λ^2 is constructed by the diagonal elements of $B^{(S)}$ and rows in $U = TC^{(S)}$ are the eigenvectors or principal components of X.

Summary of Contributions
PCA is computed with only one scan over the input dataset. The summarization step of the algorithm is a key factor to efficiently process a large number of records. As it can be seen in Algorithm 1, after summarization, the model is computed by operating matrices of dimensionality d. Finally, we have that aggregate functions, and DBMS extensibility options can be used by other data mining techniques, such as variable selection.

3.2 Bayesian Variable Selection

Data summarization can be used to compute the posterior probability distributions in variable selection with only one table scan over the input data. In order to include XY^T, $Y1_n$, and YY^T in the summarization step, the explanatory variable Y is included in the summarization as another attribute of the table X. Therefore, we compute n, L and Q for $(X_1, \ldots, X_d, Y)$. Since $\beta \in \mathbb{R}^{d+1}$, we use the augmented $\mathcal{X}$, defined in Equation 6, to represent linear regression as $Y = \beta^T\mathcal{X} + \epsilon$. For the model $\mathcal{M}_\gamma$, we have that $\beta_\gamma = (\beta_0, \{\beta_a : \gamma_a = 1\})$, the number of dimension in the model $d_\gamma = 1_n^T\gamma$, and the explanatory variables $X_\gamma = \{X_a : \gamma_a = 1\}$. By expressing probability density functions in stochastic search variable selection (SSVS) with summarization matrices, we overcome the main disadvantage, especially for large datasets, of having to scan the input dataset multiple times to compute a model.

$$\mathcal{X} = \begin{bmatrix} 1_n^T \\ X \end{bmatrix} \tag{6}$$

Algorithm 2: Stochastic Search Variable Selection

```
Input: Table $(X_1, \ldots, X_d, Y)$, iterations $S$
1 : Compute data summarization: $n$, $L$, and $Q$
2 : Pick a random $\gamma^{(0)}$ .
3 : For $I \leftarrow 1$ to $S$
4 :     For $a \leftarrow 1$ to $d$
5 :         $p_0 \leftarrow \pi(\gamma_a^I = 0|Y, X, \gamma_{(a)}^I)$*
6 :         $p_1 \leftarrow \pi(\gamma_a^I = 1|Y, X, \gamma_{(a)}^I)$*
8 :         If Rand([0,1]) $< p_0/(p_0 + p_1)$
9 :             $\gamma_a^I \leftarrow 0$
10:         else
11:             $\gamma_a^I \leftarrow 1$
Return: $\pi(\gamma|X, Y)$ from $\gamma^1, \ldots, \gamma^S$
```

*$\pi(\gamma_a^I|Y, X, \gamma_{(a)}^I)$ is evaluated, as shown in Equation 8, where X is substituded by n, L, and Q.

We use Zellner's G-prior [18], which is conditional on the error variance, σ^2, a constant c and a hyperparameter $\tilde{\beta}$ is defined as $\beta|\sigma^2, X \backsim \mathcal{N}(\tilde{\beta}, c\sigma^2(\mathcal{X}\mathcal{X}^T)^{-1})$. To complete the hierarchical formulations we define another prior on the error variance as $\sigma^2 \backsim \pi(\sigma^2|X) = \sigma^{-2}$, i.e. a non-informative prior distribution. Therefore, the prior distribution of β_γ for the model $\mathcal{M}_\gamma$ is shown in Equation 7, where $\tilde{\beta}_\gamma = (\mathcal{X}_\gamma\mathcal{X}_\gamma^T)^{-1}\mathcal{X}_\gamma\mathcal{X}^T\tilde{\beta}$. Even though there are multiple prior functions available, our choice is due the feasibility to express the probability function $\pi(\gamma|Y, X)$ (see Equation 8) of the Zellner's informative G-prior in a closed form and in terms of summarization matrices, which greatly aids our Bayesian computations.

$$\beta_\gamma|\gamma, \sigma^2 \backsim \mathcal{N}(\tilde{\beta}_\gamma, c\sigma^2(\mathcal{X}_\gamma\mathcal{X}_\gamma^T)^{-1}) \tag{7}$$

$$\begin{aligned}\pi(\gamma|Y, X) \quad \propto \quad & (c+1)^{-(d_\gamma+1)/2}[YY^T \\ & -\frac{c}{c+1}Y\mathcal{X}_\gamma^T(\mathcal{X}_\gamma\mathcal{X}_\gamma^T)^{-1}\mathcal{X}_\gamma Y^T \\ & -\frac{1}{c+1}\tilde{\beta}_\gamma^T\mathcal{X}_\gamma\mathcal{X}_\gamma^T\tilde{\beta}_\gamma]^{-n/2}\end{aligned} \tag{8}$$

Since the multiplication $X_\gamma 1_n = \{L_a : \gamma_a = 1\}^T$, and the cross-product $X_\gamma X_\gamma^T$ has as elements $\{Q_{ab} : \gamma_a = 1 \wedge \gamma_b = 1\}$, the cross-product $\mathcal{X}_\gamma{\mathcal{X}_\gamma}^T$ of the augmented set of variables of the model $\mathcal{M}_\gamma$ can be derived from summarization matrices (see Equation 9). Likewise, the matrix multiplication $\mathcal{X}_\gamma\mathcal{X}^T$ (see Equation 10), in $\tilde{\beta}_\gamma$, is constructed knowing that $X_\gamma X^T = \{Q_{a,[1:d]} : \gamma_a = 1\}$. The remaining matrix multiplication in the probability distribution, $\mathcal{X}_\gamma Y^T$ or $(Y\mathcal{X}_\gamma^T)^T$, is derived from the fact that $X_\gamma Y^T = \{Q_{a,d+1} : \gamma_a = 1\}$ (see Equation 11). Finally, the dot product of the explanatory variable $YY^T = Q_{d+1,d+1}$.

$$\mathcal{X}_\gamma{\mathcal{X}_\gamma}^T = \begin{bmatrix} 1_n^T 1_n & 1_n^T X_\gamma^T \\ X_\gamma 1_n & X_\gamma X_\gamma^T \end{bmatrix} = \begin{bmatrix} n & {L_b}^T \\ L_a & Q_{ab} \end{bmatrix} \tag{9}$$

$$\mathcal{X}_\gamma\mathcal{X}^T = \begin{bmatrix} 1_n^T 1_n & 1_n^T X^T \\ X_\gamma 1_n & X_\gamma X^T \end{bmatrix} = \begin{bmatrix} n & L_{[1:d]}^T \\ L_a & Q_{a,[1:d]} \end{bmatrix} \tag{10}$$

$$\mathcal{X}_\gamma Y^T = \begin{bmatrix} 1_n^T Y^T \\ X_\gamma Y^T \end{bmatrix} = \begin{bmatrix} L_{d+1} \\ Q_{a,d+1} \end{bmatrix} \tag{11}$$

Summary of Contributions

In our algorithm to compute SSVS, results are exact (see Algorithm 2). Only one scan to the input table is required, after that, operations are done using the summarization matrices n, L, and Q; without reading X every iteration. SSVS is computed efficiently for very large datasets, because the number of records only affects performance of the summarization step. Since summarization is fundamental for our algorithms, it is important to optimize aggregate operations inside the DBMS.

3.3 SQL Optimizations

Data summarization can be expressed in terms of the SUM aggregation. A single SQL statement is used to calculate all values of the summary matrices. A property of Q is to be a symmetric matrix, so it is enough to include the $d(d+1)/2$ upper or lower triangular elements. When including all summarization values, we have the following SQL statement:

```
SELECT SUM(1.0) /*n*/
    ,SUM($X_1$),SUM($X_2$)...,SUM($X_d$) /*L or $X1_n$*/
    ,SUM($X_1 * X_1$) /*Q or $XX^T$*/
    ,SUM($X_2 * X_1$),SUM($X_2 * X_2$)
    ⋮
    ,SUM($X_d * X_1$),SUM($X_d * X_2$),...,SUM($X_d * X_d$),
/*Variable of interest*/
    ,SUM($Y$) /*$Y1_n$*/
    ,SUM($Y * Y$) /*$YY^T$*/
    ,SUM($X_1$*$Y$),SUM($X_2$*$Y$),...,SUM($X_d$*$Y$) /*$XY^T$*/
FROM X;
```

The result table has a single row and a column for each value in n, L, and Q. Since a limitation exists for the number of columns in a table depending on the DBMS provider, there is also a limit for the number of dimensions that can be summarized with a single scan. By using only the SUM aggregate function, data summarization is computed without modifications to the DBMS.

Aggregate UDFs are used to overcome the limitation on the number of dimensions that can be summarized with a single scan [25]. Only one function which computes summarization of a given set of dimensions is defined. Since the accumulate function receives only one parameter, dimensions are packed with a user-defined type. Values for the d attributes of a data point have to be stored as a binary object which can be accessed following the user-defined type definition. Therefore, to implement data summarization, the required methods are: (1) a parsing function, and (2) a serialization functions for reading and writing the binary objects. The parsing function creates a value of the type when given a string value concatenating all the values of the dimensions. The aggregate UDF for summarization receives all values for the dimensions of a data point packed as a binary object, and returns all the elements in n, L, and Q also packed using the user-defined type. We have the following SQL statement to compute the aggregation:

```
SELECT $Agg$(CAST(CAST($X_1$ AS VARCHAR)+','+
    CAST($X_2$ AS VARCHAR)+','+
    ⋮
    CAST($X_d$ AS VARCHAR) AS $Type$))
FROM $X$;
```

Execution performance of the aggregate UDF can be improved by materializing a table with a single column storing the packed dimensions with the user-defined type. Even though the aggregation can be computed efficiently when data is already in binary format, creating such table is a time consuming pre-step which can be avoided with specific optimizations for data summarization.

3.4 Hardware Optimizations

The algorithm design focuses on using multithreading to distribute the workload, while keeping the hard drive access sequential for performance, and following the common API of DBMSs to access query result sets. Furthermore, concurrent threads have to guarantee deterministic results without race conditions, deadlocks or starvation.

On the design of aggregate functions, we included several changes that increase the speed for hardware configurations where computing the aggregations is faster than retrieving records from the storage device. To obtain sequential reading, one thread is uniquely in charge of retrieving records from the input table, caching blocks of records in main memory, and calling a monitor to dispatch the job to other thread that actually performs the calculations or working thread. All the threads share memory to update the global aggregate computation. Moreover, we define techniques to control the number of threads executing simultaneously, and the amount of memory used by the aggregation process.

Since each worker thread is assigned the task of computing the aggregation of one block, portions of the data are cached at all the time during execution. The reading process is oblivious to the multithreaded execution since its only task is to allocate memory space to fit a fixed number of rows and fill the current block with records from the input table. The characteristic difference with a parallel aggregate is the way the threads access the data. Instead of having the threads requesting blocks from the input device, threads are assigned a block as workload by the monitor process. Reading from the input and monitoring the process, both tasks, are done by the main thread. Thus, additionally to the cost of reading the input sequentially and allocating blocks in main memory, we have to take into account the overhead of dispatching the worker threads. There is little overhead caused by the monitor calls, due to the speed difference between reading records from disk and computing flops by the processor.

In order to manage the multiple worker threads, we include a monitor process. The monitor executes as part of the main thread; it is in charge of dispatching workload, and terminating the execution when all worker threads have finished their computations. We propose three monitor alternatives: (1) create a thread for every upcoming task (2) create a thread pool (3) use a fixed number of threads.

The simplest approach for the monitor is to create a thread for every request of dispatching a workload, and then add the thread to a list. Such configuration has the reading process allocating blocks in memory, with disregard whether the processing power is enough to complete the tasks before causing stack or memory overflow. Moreover, the policy of the operating system assigns time slices of the processors to threads. A higher priority could not be necessarily given to tasks closer to finish, and uncompleted tasks will keep holding memory space until done.

Table 1: Time Compexity of n, L and Q

	n	L	Q
elements	1	d	$\frac{(d+1)\times d}{2}$
flops per accumulate step	1	d	$(d+1)\times d$
flops per merging	1	d	$\frac{(d+1)\times d}{2}$
overall time complexity	$O(n)$	$O(nd)$	$O(nd^2)$

To control the number of threads executing in the system and to have a FIFO policy for the upcoming workload, our second alternative includes a thread pool managed by the monitor process. With such configuration, all tasks created by the monitor are added to the thread pool. As such, whenever a thread finishes its current task, it is assigned the next task in queue. Even though completed tasks free memory space, each task in the thread pool queue has a data block associated to it. Moreover, if the waiting queue grows big enough it could cause memory overflow.

The maximum amount of memory used for caching can be controlled by our third alternative for the monitor process. While this approach does have a circular list to keep a fixed number of working threads, the queue is eliminated since there can be at most one task waiting to be executed. Although the circular list limits the amount of memory used for caching, the reading process has to be stopped every time the list is full. Finally, stopping the sequential read for long periods of time could severely impact the performance of the algorithm.

3.5 Time and Space Complexity

Time complexity, and the number of operation for the summarization matrices can be seen in Table 1. Even though, the overall time complexity of the summarization process is $O(nd^2)$, with multi-threading the problem becomes disk bandwidth bound. Therefore, the lowest bound for summarization is an execution time close to a table scan or $O(nd)$. On the other hand, the space complexity of the aggregation process has the global aggregation invariant, and it fluctuates depending on the number of working threads existing in the system at any time. Thus, we have that the amount of memory used is given by:

$$Memory\ Used = ((t+u)((b\times d)+e)+e)f, \qquad (12)$$

where t is the number of threads executing in the system, u is the number of tasks created and waiting for a thread, b is the block size, d is the number of dimensions, e is the number of elements in the aggregation and f are the bytes used to store a number.

In addition to the table scan to compute the covariance or correlation matrix in PCA, it has a Householder decomposition, and d QR factorizations. Solving Householder is $O(d^3)$, with $d-2$ steps of $O(d^2)$ each. Each QR factorization is $O(d^2)$, and computed d times S iterations. Consequently, the global complexity of solving the eigenvalue decomposition in PCA is $O(d^3)$ (see Table 2). On the other hand, time complexity of SSVS also includes the table scan to calculate sufficient statistics. Each probability estimation $\pi(\gamma_a^I|Y,X,\gamma_{(a)}^I)$ has a complexity of $O(d^3)$. For each iteration I, $2\times d$ probabilities are computed. Therefore the global complexity of SSVS is $O(nd^2+d^4)$. In the next section, we present our experimental results, and the impact of efficient data summarization when computing PCA and SSVS.

Table 2: Time Compexity PCA and SSVS

Technique	Summarization	Model
PCA	$O(nd^2)$	$O(d^3)$
SSVS	$O(nd^2)$	$O(d^4)$

Table 3: PCA Loading for the *wpbc* Dataset

	R		DBMS	
	U_1	U_2	U_1	U_2
X_1	0.310	-0.397	0.311	-0.396
X_2		-0.146	0.008	-0.146
X_3	0.335	-0.371	0.336	-0.370
X_4	0.312	-0.392	0.313	-0.392
X_5	0.254	0.370	0.253	0.371
X_6	0.375	0.261	0.374	0.262
X_7	0.442		0.441	0.091
X_8	0.456		0.456	-0.024
X_9	0.247	0.309	0.246	0.309
X_{10}	0.159	0.469	0.158	0.470

4. EXPERIMENTAL RESULTS

For this paper, we conducted our experiments on a commercial DBMS installed on a server with an Intel Core 2 Quad CPU with four cores of 2.83 GHz each, and 3.24GB of RAM. The hard drive had 320GB of capacity, with a SATA interface of 3.0Gb/s, and 7200 RPM. We used the implementation of PCA already included in R, and for SSVS we used the code given by [18]. DBMS extensibility was done using aggregate user-defined functions (UDFs), table-valued functions (TVFs), and store procedures (SPs). We use one real dataset *wpbc*, of breast cancer diagnostic information, from the UCI Machine Learning Repository [1]. Other datasets used were created following normal distributions, and they are test cases for constructing linear models. Finally, we have that all experiments were repeated five times before reporting an average.

Experiments are organized as follows. First, we perform PCA and SSVS on the breast cancer dataset. We compare the accuracy of our algorithms in the DBMS with results obtained by the statistical package R, which is a standard in statistics and data mining. The second set of experiments shows the performance of our optimizations to control multi-threading, caching in memory, and accessing records by blocks during the data summarization step. Also, we include a performance analysis of all the different alternatives to compute data summarization in the DBMS. Finally, the efficiency of our algorithms is shown in a comparison of the time measurements of computing PCA and SSVS with R.

4.1 Evaluation of Model Accuracy

In this section we analyze the accuracy of the models computed with our algorithms. For comparison purposes, we have executed PCA over 10 dimension of the *wpbc* dataset picked at random. In both R and our implementation, the eigenvalues λ_a are greater than the unit only for the first two loading vectors. Table 3 shows the loading of these two principal components for each variable. Clearly, results are the same; the only difference with R is because it gets rid of the less significant loadings.

Table 4: SSVS of the *wpbc* Dataset

R		DBMS	
$a:\gamma_a=1$	$\pi(\gamma\vert X,Y)$	$a:\gamma_a=1$	$\pi(\gamma\vert X,Y)$
2,7,10	0.0748	2,7,10	0.0828
1,2,3,10	0.0724	1,2,3,10	0.0686
2,6,10	0.0387	2,6,10	0.0423
2,3,9	0.0289	2,7,9,10	0.0287
2,7,9,10	0.0273	2,3,9	0.0275
2,3	0.0271	2,3	0.0237
1,2	0.0247	2,3,10	0.0234
2,4	0.0233	1,2	0.0228
2,6,9,10	0.0221	2,4	0.0226
2,3,10	0.0198	2,6,9,10	0.0205

We also used the *wpbc* dataset to find the best subset of dimensions or explanatory variables to predict the recurrence time for the patients. Estimations for the probability distribution $\pi(\gamma|X,Y)$ are presented in Table 4. The distribution was computed over 10000 iterations, after a burning period of 10000 iterations. For the priors, we used $\tilde{\beta} = 0_{11}$ and $c = 100$. Our results match with R in the top 10 most frequent samples, and the same ranking order is shown for the three models with highest probability estimation. Even though, SSVS is a stochastic search based in probabilities, the results obtained by the DBMS are equivalent to R. Nevertheless, the optimizations we introduced to efficiently compute aggregates have a significant impact on time performance to compute the models.

4.2 Evaluation of Optimizations

In this section, we analyze the performance impact of the alternatives to control multi-threading by: (1) creating a thread for every task or block to aggregate (MT-UDF), (2) using a maximum fixed number of threads with at most one task waiting to start (FT-TVF), and (3) using a thread pool with fixed number of threads and a queue of waiting tasks (TP-TVF). The parameters for summarization are the aggregate computation *agg*, the block size *b*, and the number of working threads *t*. All the alternatives are compared against the execution time of a table scan, using the sequential data access primitive given by the connectivity interface of the DBMS.

Results in Figure 1 show the execution performance of the three alternatives when computing the Q aggregation, $n = 1M$, and working threads $t = 4$. We can see that the monitoring process of multi-threading adds little or no representative overhead to the table scan. For the cases when $d \leq 32$, the impact of performing the aggregation while reading the table is minimal. On the other hand, the trend becomes evident when $d \geq 64$; there is a difference on performance given by the monitoring policy used for the working threads. Nevertheless, the time impact of any of the policies is not enough to consider a higher complexity than a table scan. The biggest overhead between the policies is caused when the OS is left to manage all the threads. Such results were somehow expected, because there is no memory control, and the policy to assign workload is suboptimal for aggregations. The scheduler will try to assign equal time-slides to the existing threads in the system, adding context switching to the picture, and letting the system to saturate with unfinished workload. The best performance is acquired by TP-TVF

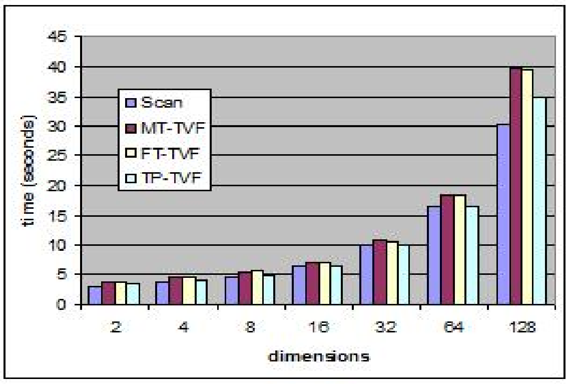

Figure 1: Concurrency Control Comparison (task=Q, n=1M, d=64, t=4)

because there is little saturation to the OS components by the number of threads to manage. As for FT-TVF, controlling the maximum memory used by the aggregation does not representatively affect the execution time, and the performance is comparable to a table scan while being faster than MT-TVF.

Caching data blocks of the input table, with disregard of the amount of memory used for this matter, could cause memory overflow issues. Aware of this potential catch, we have proposed methods to limit memory usage, and implemented them in FT-TVF. Since our experimental study focuses on $d << n$, such limitation is never reached by varying values of $d \leq 128$. Furthermore, we found that in all our experimental cases, a reduced number of threads are enough to compute sufficient statistics efficiently. Table 5 displays the performance of our monitor policies, when varying the number of working threads additional to the reading thread. For $d = 64$, execution time is almost invariant by increasing the number of threads assigned to the accumulate part of the aggregation. Such behavior is caused because the time to read a record of size d from the input table is greater than the time it takes to the processor to calculate the $O(d^2)$ flops in the accumulate step. For instance, in the data set of $d = 128$, there are about 16 Mflops per record to aggregate, and the Core 2 Quad executes about 90 Gflops/second. The reading time of accessing records with the database programmability API is expected to be higher than the theoretical disk bandwidth of 3 Gbits/second. When d increases, we need more threads to catch up with the reading speed. This is the case for $d = 128$, the performance of FT-TVF reaches a pick when $t = 3$. When $t < 3$, the sequential read has to be stopped every time the limit for the number of working threads (t) is reached. In contrast, TP-TVF does not stop whenever there are t working threads busy, instead it places all waiting workload in a queue. Furthermore, the cases when $d = 128$ with $t > 3$ for FT-TVF, and $t > 1$ for TP-TVF show the tradeoff caused by using more threads than the required by the aggregation. Since data summarization is bound by the disk bandwidth, scalability due to the number of threads is very limited. Even though summarization with one reading and one working thread has very similar time to a table scan, the difference with executing both tasks with a single thread is evident.

Table 5: Execution time when varying the number of threads (time in seconds) (n=10M)

d	control threads	working threads (t)	FT-TVF	TP-TVF
64	1	0	311.80	311.80
64	1	1	172.98	155.23
64	1	2	173.26	155.35
64	1	3	173.24	155.40
64	1	4	173.03	155.68
128	1	0	893.50	893.50
128	1	1	385.90	324.44
128	1	2	380.60	324.43
128	1	3	378.81	326.07
128	1	4	379.87	329.68

4.3 Time Complexity of Summarization Aggregates

For comparison purposes, we tested standard SQL aggregations, aggregate user-defined functions, and multithreaded TVFs. We include a comparison with a TVF that does not use multi-threading to distribute workload (REG-TVF). Table 6 contains the execution performance when solving the Q aggregation for $n = 1M$. FT-T VF, and TP-TVF have the same number of threads $t = 4$. The three TVFs with multi-threading have a block size $b = 1000$.

Execution performance of REG-TVF is severely affected by d; only for the cases when $d \leq 8$ is the execution time comparable with the multithreaded TVFs. The impact of multithreading is shown for $d = 128$ where the time of the single-threaded TVF (REG-TVF) is more than double the time of the other TVF implementations. On the other hand, BIN-UDF aggregates rows packed in binary structures that follow the user-defined type definition. It has a better performance than the multithreaded TVFs for $d \geq 32$. However, BIN-UDF is better only if we do not consider the pre-step of physically materializing the table inside the DBMS. The second implementation of aggregate UDFs (STR-UDF) moves the CAST function, for packing rows into a user-define type object, into the SQL statement of the aggregation. Yet, STR-UDF still has higher execution time than any of the TVFs, because of the overhead of the parsing function in the user-defined type.

It can be seen in Figure 2 that tendencies hold when increasing the size of n. Although, SQL is the fastest way to compute the aggregates in all the cases when $d \leq 16$, for $d = 32$ the performance decreases in a slight amount. Unfortunately, limitations of the number of rows in a query, prevents experimenting with $d \geq 64$. Since the complexity of REG-TVF is $O(nd^2)$, the performance of TP-TVF demonstrates its efficiency to exploit primitives for accessing data in the DBMS. Consequently, the time difference with REG-TVF increases with d, while TP-TVF remains comparable with the aggregate step of BIN-UDF.

The final set of experiments is to verify linear execution time with respect to the size of the input table. The execution results for Q, with a number of dimensions $d = 16$, are presented in Figure 3. The multithreaded monitoring algorithm with a tread pool (TP-TVF) is parameterized with threads $t = 4$, and a block size of $b = 1000$. The three methods

Table 6: Muti-threading Comparison with Aggregate User-define Functions (time in seconds) (agg=Q, n=1M, t=4, b=1000)

d	Plain SQL	REG-TVF	BIN-UDF Pre-step	BIN-UDF	STR-UDF	MT-TVF	FT-TVF	TP-TVF
2	0.8	3.1	18.1	6.0	8.0	3.7	3.3	3.2
4	0.7	3.4	21.3	6.0	8.6	3.9	3.7	3.7
8	1.4	4.5	28.1	6.1	10.5	4.8	4.8	4.5
16	4.9	7.0	41.6	6.4	14.2	6.4	6.3	6.3
32	86.3	13.4	68.4	7.4	21.2	9.8	9.6	9.4
64	*	31.8	126.7	10.7	38.1	17.4	17.5	16.1
128	*	89.7	244.9	22.9	78.3	38.0	37.9	33.1

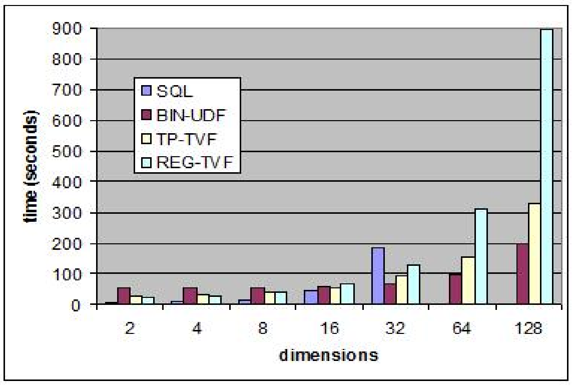

Figure 2: Aggregate Comparison when Varying d ($agg = Q$, $n = 10M$, $t = 4$, $b = 1000$)

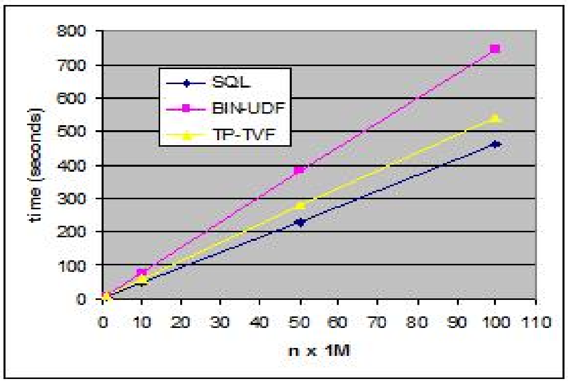

Figure 3: Aggregate Comparison when Varying n ($agg = Q$, d=16, t=4, b=1000)

(SQL. BIN-UDF and TP-TVF) show linear scalability with respect to the data size n. Finally, experimental results have provided evidence that our algorithms for multi-threading scale linearly in both n and d. Such achievement is obtained by efficiently taking advantage of DBMS primitives, and integrating them with multithreaded aggregates.

4.4 Comparison with R

The current work has focused on optimizing aggregate operations to compute data summarization. The performance impact of efficiently calculating sufficient statistics is evident for data analysis of large datasets or $n >> d$. Only one table scan is needed to obtain the principal components or a probability distribution of the best subset of attributes to predict an output variable. For the TVF implementation, we have used the monitor with a fixed number of threads, and at most one task waiting to be assigned a working thread (FT-TVF); the number of threads is $t = 4$, and the block size $b = 1000$. The SQL implementation uses standard SQL to compute the summarization step. Also, aggregate user-define functions or UDFs are defined without a pre-computing a table for the attributes packed with the user-defined type (STR-UDF). Finally, we have that summary matrices computed with these methodologies are used by implementations inside the DBMS to perform PCA and SSVS.

Table 7 presents the execution time performance of solving PCA with our algorithms inside the DBMS, and outside the DBMS with the R statistical package. Experimental results show that data summarization is a key step for analyzing large datasets. The performance tendencies, seen during the analysis of optimization, hold for PCA. The FT-TVF has linear scalability with the size of the input data. On the other hand, summarization with standard SQL has good scalability with the size of the input data n, yet poor with the increase of the number of dimensions d. The main disadvantage of the STR-UDF is having to parse attributes packed with the user-defined type during each accumulate step. Therefore, performance of the STR-UDF scales poorly on n, but it has good scalability with the increase of the number of dimensions d. In all the cases, we have that our algorithms outperform execution outside the DBMS.

Table 7: Execution Performance for Solving PCA (time in seconds) (* out of memory)

			DBMS		
n	d	R	FT-TVF	SQL	STR-UDF
100k	8.0	6.3	2.0	1.5	4.8
100k	16.0	13.3	2.4	4.4	7.1
100k	32.0	31.4	2.7	11.5	10.8
1M	16.0	218.1	8.6	7.4	21.6
1M	32.0	*	14.8	94.9	30.4
10M	16.0	*	65.1	51.4	204.5
10M	32.0	*	106.5	198.6	291.6

Not only PCA benefits from efficient data summarization, Table 8 contains the execution performance of solving SSVS for several test cases. We can see the performance improvement of our algorithm to compute SSVS for large datasets by comparing with R. Since the time taken in R to load the

Table 8: **Execution Performance for Solving SSVS ($S = 1K$) (time in seconds) (* out of memory)**

			DBMS		
n	d	R	FT-TVF	SQL	STR-UDF
100	8	16.5	0.6	0.7	0.9
1K	8	605.2	0.7	0.9	1.1
10K	8	>2K	0.7	0.9	1.5
100K	8	>2K	1.3	1.6	5.4
1M	8	>2K	5.5	2.7	17.8
10M	8	*	47.2	18.1	169.2
1M	16	>2K	8.6	9.1	23.5
1M	32	*	29.7	131.4	49.5
1M	64	*	302.6	*	328.4

Table 9: Time Percentage of Summarization and the Model Computation with FT-TVF ($S = 1K$)

		PCA		SSVS	
n	d	n,L,Q	Model	n,L,Q	Model
1M	8	76%	24%	94%	13%
1M	64	90%	10%	6%	98%
10M	8	96%	4%	97%	2%
10M	64	98%	2%	38%	32%

dataset in main memory is minimal compared to computing SSVS, there is a significant drop in execution time in our methodologies caused by the use of summarization matrices. Furthermore, benefits of choosing a specific optimization, depending on the input dataset, remain for a relatively small number of iterations. However, the time difference between optimizations for the summarization step becomes less representative when S increases, due the execution time taken up by the $O(d^4)$ iterative step. It can be seen in Table 9, the time percentage summarize the data, and to compute the model from n, L, and Q. We can see the benefit from our algorithm to compute PCA and SSVS, especially for very large datasets. Finally, we have that models are constructed from the complete data set, and the results obtained with our algorithms are exact.

5. RELATED WORK

There is a wide range of related work to apply PCA in image processing [12], pattern recognition [30], data compression [6], clustering [8] and classification [15]; thus, the importance to efficiently compute PCA for large dataset. Even though, the use of sufficient statistics was previously introduced for linear models [22], our novelty is to extend the use of the summarization matrices for Bayesian statistics, Markov chains, and the Gibbs sampler [11]. Variable or feature selection is well-research topic in both data mining and statistical literature, leading to a variety of algorithms for searching the model space and selection criteria for choosing between competing models [19]. In the Bayesian framework, Markov chain Monte Carlo methods can be used for inference and estimation, such in [3]. Thus, the model selection problem is transformed to the form of parameter estimation, in which rather than searching for the single optimal model, attempts to estimate the posterior probability of all models within the considered class of models (in our case, we focus on linear regression [2]). The primary task then is to estimate the marginal posterior probability that a given set of variables should be in the model. Such a setup has been used in various variable selection settings by Mitchell and Beauchamp [20], Clyde, Desimone and Parmigiani [17], George and McCulloch [10], Smith and Kohn [27] and Chipman et. al. [5] but were limited to cases when the size and dimensionality of the data is relative small. The DBMS approach we propose here is not only highly computationally efficient but also scales well to large databases.
Integrating data mining and statistical techniques into a DBMS has been paid little attention by the research community. Due to the importance of sufficient statistics to process large matrices [26, 9], it is a desirable capability to be integrated into the DBMS. Some of the applicability of constructing models based on correlation and clustering is presented in [23]. Resent work has focused on optimizing the computation of sufficient statistics by exploiting caching and sampling [25]. In contrast, we propose specific changes to the aggregation algorithm to avoid intermediate steps, and concentrating on hardware performance. Our present article extends our previous work computing SVD with SQL and TVFs [24, 21]. In our new article we show both PCA (based on SVD) and SSVS (based on Gibbs sampler) can be solved with the same underlying sufficient statistics n, L, and Q. Solving both models highlights the importance of pushing computation of summary matrices using SQL and UDFs. Also, we incorporate OS level optimizations: multi-threaded processing and RAM memory management for threads, which turned out to be challenging to incorporate into UDFs.

6. CONCLUSIONS

In this paper, we have studied how DBMS functionality to compute aggregates can be exploited in linear models. By seeing the importance of data summarization, we proposed algorithms to yield DBMSs with the crucial functionality of PCA and SSVS. Experimental results show that results are not compromised by representing operation of both techniques in terms of summary matrices. Therefore, after one table scan to summarize the input data, the models computed with our algorithms are the same than the ones obtained with the R statistical package. Our optimizations show how to efficiently execute aggregate operations to summarize data in large data sets. The performance of the algorithms was tested with standard SQL and aggregate UDFs. Even though for most cases an efficient summarization can be done using few threads, there is a significant difference with the summarization performance without multithreading. Our algorithms for data summarization exhibit linear scalability on both the data size n and on the number of dimensions d. On the other hand, the time to compute the model depends uniquely on the number of dimensions d, and their performance does not get affected by the increase of the number of records in the dataset. By efficiently using DBMSs capabilities to process large datasets, it is possible to achieve better performance, and overcome limitations of external tools for statistics and data mining.
There are several issues for future research. Since not all data fits a linear model, it is important to investigate methods to improve the models obtained with SSVS to fit non

linear distributions of the data. Since information with high dimensionality is of great interest for diverse applications, other problem to improve SSVS for large datasets and $d > n$. Our optimizations for performing aggregate operations fit the MapReduce paradigm; further research to speed up data summarization can be done under this framework, which is an alternative to UDFs. Also, we have proposed several changes to the steps in a multithreaded UDF. Although, our modifications are specific to compute data summarization, we think that further research should adjust the definition of multithreaded UDFs to solve a broader set of problems.

7. REFERENCES

[1] D. N. A. Asuncion. UCI machine learning repository, 2007.

[2] V. Baladandayuthapani, R. Carroll, and B. Mallick. Spatially adaptive bayesian penalized regression splines (p-splines). *Journal of Computational & Graphical Statistics*, Volume 14:378–394, 2005.

[3] V. Baladandayuthapani, B. Mallick, M. Hong, J. Lupton, N. Turner, and R. Carroll. Bayesian hierarchical spatially correlated functional data analysis with application to colon carcinogenesis. *Biometrics*, 64(1):64–73, March 2008.

[4] J. Blakeley, V. Rao, I. Kunen, A. Prout, M. Henaire, and C. Kleinerman. .net database programmability and extensibility in microsoft sql server. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1087–1098, New York, NY, USA, 2008. ACM.

[5] H. Chipman, E. I. George, and R. E. Mcculloch. The practical implementation of bayesian model selection. In *Institute of Mathematical Statistics*, pages 65–134, 2001.

[6] S. Chitroub, A. Houacine, and B. Sansal. A new PCA-based method for data compression and enhancement of multi-frequency polarimetric sar imagery. *Intelligent Data Analysis*, 6(2):187–207, 2002.

[7] S. Cohen. User-defined aggregate functions: bridging theory and practice. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 49–60, New York, NY, USA, 2006. ACM.

[8] C. Ding and X. He. K-means clustering via principal component analysis. In *Proc. ICML Conference*, page 29, 2004.

[9] P. Drineas and M. W. Mahoney. A randomized algorithm for a tensor-based generalization of the singular value decomposition. *Linear Algebra and its Applications*, 420(2-3):553 – 571, 2007.

[10] E. George and R. McCulloch. Approaches for bayesian variable selection. *Statistica Sinica*, 7:339–374, 1997.

[11] E. I. George and R. E. Mcculloch. Variable selection via gibbs sampling. *Journal of the American Statistical Association*, 88(423):881–889, 1993.

[12] J. Gerbrands. On the relationships between svd, klt and pca. *Pattern Recognition*, 14(1-6):375–381, 1981.

[13] G. Graefe, U. Fayyad, and S. Chaudhuri. On the efficient gathering of sufficient statistics for classification from large SQL databases. In *Proc. ACM KDD Conference*, pages 204–208, 1998.

[14] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-total. In *ICDE Conference*, pages 152–159, 1996.

[15] M. Hubert and S. Engelen. Robust pca and classification in biosciences. *Bioinformatics*, 20(11):1728–1736, 2004.

[16] M. Jaedicke and B. Mitschang. On parallel processing of aggregate and scalar functions in object-relational DBMS. In *ACM SIGMOD Conference*, pages 379–389, 1998.

[17] H. D. M. Clyde and G. Parmigiani. Prediction via orthogonalized model mixing. *Journal of the American Statistical Associatio*, 91(435).

[18] J. M. Marin and C. P. Robert. *Bayesian Core: A Practical Approach to Computational Bayesian Statistics*. Springer Publishing Company, Incorporated, 2007.

[19] A. Miller. *Subset Selection in Regression*. Chapman & Hall/CRC, Boca Raton, Florida, U.S.A., 2002.

[20] T. J. Mitchell and J. J. Beauchamp. Bayesian variable selection in linear regression. *Journal of the American Statistical Association*, 83(404):1023–1032, 1988.

[21] M. Navas and C. Ordonez. Efficient computation of PCA with SVD in SQL. In *KDD Workshop on Data Mining using Tensors and Matrices*, 2009.

[22] C. Ordonez. Building statistical models and scoring with UDFs. In *Proc. ACM SIGMOD Conference*, pages 1005–1016, 2007.

[23] C. Ordonez. Models for association rules based on clustering and correlation. *Intelligent Data Analysis*, 13(2):337–358, 2009.

[24] C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22, 2010.

[25] C. Ordonez and S. Pitchaimalai. Fast UDFs to compute sufficient statistics on large data sets exploiting caching and sampling. *Data & Knowledge Engineering*, 69(4):383 – 398, 2010.

[26] M. W. M. P. Drineas and S. Muthukrishnan. Subspace sampling and relative-error matrix approximation: column-row-based methods. In *ESA'06: Proceedings of the 14th conference on Annual European Symposium*, pages 304–314, London, UK, 2006. Springer-Verlag.

[27] M. Smith and R. Kohn. Nonparametric regression using bayesian variable selection. *Journal of Econometrics*, (75):317–344, 1996.

[28] H. Wang, C. Zaniolo, and C. Luo. ATLaS: A small but complete SQL extension for data mining and data streams. In *Proc. VLDB Conference*, pages 1113–1116, 2003.

[29] H. C. Yang, A. Dasdani, R. L. Hsiao, and D. S. Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040, New York, NY, USA, 2007. ACM.

[30] X. Zhuang and D. Dai. Improved discriminate analysis for high-dimensional data and its application to face recognition. *Pattern Recogn.*, 40(5):1570–1578, 2007.