Efficient computation of PCA with SVD in SQL

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

ABSTRACT

PCA is one of the most common dimensionality reduction techniques with broad applications in data mining, statistics and signal processing. In this work we study how to leverage a DBMS computing capabilities to solve PCA. We propose a solution that combines a summarization of the data set with the correlation or covariance matrix and then solve PCA with Singular Value Decomposition (SVD). Deriving the summary matrices allow analyzing large data sets since they can be computed in a single pass. Solving SVD without external libraries proves to be a challenge to compute in SQL. We introduce two solutions: one based in SQL queries and a second one based on User-Defined Functions. Experimental evaluation shows our method can solve larger problems in less time than external statistical packages.

Categories and Subject Descriptors

G.1.3 [**Mathematics of Computing**]: Numerical Linear Algebra—*Singular value decomposition*; G.3 [**Probability and Statistics**]: Multivariate statistics; H.2.4 [**Database Management**]: Systems—*Relational databases*; H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms

Algorithms, Performance, Theory

Keywords

PCA, SVD, SQL, DBMS

1. INTRODUCTION

Principal component analysis or PCA is a popular technique in statistics and data mining for dimensionality reduction. It is a valuable tool to reveal hidden patterns, compress and extract relevant information from complex data sets. Several applications for image processing [6], data compression [3], pattern recognition [23], clustering [5], classification [9] and time series prediction [22] have developed over time around PCA. By successfully applying linear algebra, PCA finds the optimal linear scheme, in sense of least square errors, to reduce dimensions of a data set. Traditionally PCA is computed by exporting data as flat files compatible with statistical tool specifications. Since the complete data set is uploaded into main memory, handling large amount of information is a major concern for analysis. Few proposals have taken into account the integration of statistical methods into the DBMS, due to its limitations to perform complex matrix and vector operations. Constructing statistical models efficiently inside the DBMS is a key factor to add data mining capabilities into the DBMS [16]. Previous research has used user defined functions (UDFs) to enrich the DBMS with PCA [20]. We have extended past work by successfully implementing all steps participating in PCA with SQL statements. Compared with UDFs, SQL is portable among DBMS providers. Also the query optimizer manipulates disc and memory I/O operations to run without data size limitations. Additionally this article includes a comprehensive analysis of implementation alternatives to optimize execution time without affecting correctness of the results. Our focus is to deal with large amount of data and overcome limitations of statistical applications.

The article is organized as follows. Section 2 presents definitions. Section 3 presents our contributions, how PCA is solved inside the DBMS with SQL statements, along with some implementation choices. Section 4 presents experiments comparing different methods of correlation analysis, SVD implementation inside and outside the DBMS, SVD optimizations to reduce execution time. Section 5 includes related work. In Section 6 we present conlusions and future work.

DMMT'09, June 28, 2009, Paris.

2. PRELIMINARIES

In this section we explain PCA, the relation with SVD and the algorithm to solve the eigen-problem. The initial step is to compute the correlation matrix used as input for PCA. Next, we explain the fundamentals of dimensionality reduction, together with the connection to the eigen-problem. We show how the QR algorithm performs the Householder tridiagonalization of the correlation matrix and iteratively the QR decomposition, returning the principal components.

2.1 Definitions

Let $X = \{x_1, ..., x_n\}$ be the input data with n points, where each point has d dimensions. X is a $d \times n$ matrix, where the point x_i is represented by a column vector (equiv-

alent to a $d \times 1$ matrix). We use the subscripts i and j to indicate position of a value in a matrix, thus x_{ij} is the value at the row i and the column j of X. The subscripts k and s are used to identify the number of the current execution step. Matrix transposition is denoted by T, likewise norm of a vector as $||x_i||$. The matrix Q, from the summary matrices L and Q, is not to be confused with Q from the QR decomposition. The storage of the data set X and table definitions in a DBMS, can be found in Section 3.1.

2.2 Data Preprocessing

In order to perform PCA, the first step is either to compute the covariance or the correlation matrix of the input data set with data centered on the mean (substracting μ). The covariance matrix of X is a $d \times d$ matrix with measures of how the dimensions change together and a main diagonal of variances. It is defined as follows:

$$C_X = \frac{1}{n-1} X X^T \tag{1}$$

Correlation coefficients are more desirable because they indicate the strength and direction of the linear relationship between two variables in the range $[-1, 1]$. Therefore, a correlation matrix is normalized, symmetric $d \times d$ and populated with the degree of correlation among dimensions. Such matrix can be calculated using the summary matrices: L and Q [16]. Let L (see Equation 2) be a column vector $d \times 1$ with the linear sum of points with elements of the form L_i where $i = 1, 2, .., d$. At the same time, let Q (see Equation 3) be the quadratic sum of points, in the sense that the $d \times d$ matrix with values Q_{ij} is the sum of the cross-products of each point with its transpose. These summary matrices allow us to compute several statistical techniques efficiently, due to its small size compared to X when $d << n$. Thus, elements in the correlation matrix are given by Equation 4.

$$L = \sum x_i \tag{2}$$

$$Q = \sum x_i {x_j}^T \tag{3}$$

$$\rho_{ij} = \frac{n Q_{ij} - L_i L_j}{\sqrt{n Q_{ii} - L_i^2}\sqrt{n Q_{jj} - L_j^2}} \tag{4}$$

2.3 PCA Overview

In PCA the objective is to find a change of basis with fewer attributes while retaining as much of the variations present in the original data set as possible. The transformation U^T that maps the original data X into a new dimensionally reduced space $U^T X$ is called the principal components of X. Such projection of the sample space is defined by a linear basis where the variance of each direction is maximized and the covariance between directions is minimized. Therefore, each linear transformation or column vector of U is associated to the variance of its projection. After sorting the linear transformations accordingly to variance, we can pick the k^{th} most representative vectors to be the principal components [11]. Moreover, PCA can be used to identify the most representative features in the data set [1] by solving the column subset selection problem.

The singular value decomposition (SVD) is used to solve PCA. The duality of change of basis establishes the relationship between the right, the left eigenvalue decompositions and the input data set. Given a matrix X the equation for the SVD is as follows,

$$X = UEV^T \tag{5}$$

Solving the SVD of X produces two orthogonal basis, one defined by the left singular vectors U, and the second given by the right singular vectors V. Since U^T is the linear transformation of PCA, we need to solve the left eigenvalue decomposition problem as shown in Equation (6). The equation is obtained by substituting X by its SVD (5), where E^2 is the diagonal matrix of eigenvalues.

$$XX^T = UEV^T VEU^T = UE^2U^T \tag{6}$$

Let us consider the covariance matrix of the projection $U^T X$ given by (7). As $U^T = U^{-1}$ (U is orthogonal), we can use the eigenvalue decomposition (6), such that $U^T XX^T U = U^T UE^2U^T U = (U^{-1}U)E^2(U^{-1}U) = E^2$. Consequently, the variance of the dimensions in the projection $U^T X$ are proportional to the eigenvalues in E^2 and for the algebraic solution of the decomposition, the covariance measures between dimensions are zero. Such characteristics explain the selection of the eigenvectors in U^T as the principal component scores of X.

$$C_{U^T X} = \frac{1}{n-1} U^T X (U^T X)^T = \frac{1}{n-1} E^2 \tag{7}$$

Solving PCA is equivalent to solve the eigen-problem for the covariance or the correlation matrix. In our experiments we use correlation as it is already normalized and it requires minimum preprocessing. The eigenvalue decomposition method is performed by using Householder tridiagonialization and QR factorization. Such algorithm is explained next.

2.4 QR algorithm to solve PCA

Here we present in detail the steps of a QR based algorithm to solve the eigen-problem. Given either the correlation or the covariance matrix as an approximation of XX^T, the eigenvectors U and E^2 are computed. In Section 3, we will match every step in the algorithm with optimized operations in the DBMS, revealing guidelines to implement efficiently QR based algorithms for the eigenvalue decomposition problem. To solve SVD and PCA the approach applied in our system (see Figure 1) uses Householder tridiagonalization followed by the QR algorithm [21].

1	Compute summarization matrices
2	Calculate correlation matrix
3	Perform Householder tridiagonalization
4	Execute QR algorithm

Figure 1: Solving PCA with SVD overiew.

The summary of steps is as follows: (1) Start by computing a correlation matrix of X, $A_0 = XX^T$, the correlation matrix on the d space. (2) Apply Householder method by reducing the correlation matrix to an equivalent tridiagonal matrix, with a linear transformation T. $B_0 = T^T A_0 T$, where $T^{-1} = T^T$. (3) 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 criteria. (4) Finally, we can combine both decompositions to get $XX^T = (TC_s)B_s(TC_s)^T$. As a result, the diagonal matrix of eigenvalues E^2 is constructed by the diagonal elements of B_s and columns of $U = TC_s$ are the eigenvectors of XX^T.
As shown in Section 2.3, the computation of SVD can be done by solving the eigenvalue decomposition of the correlational matrix. Given $A_0 = XX^T$, the Householder tridiagonalization is done in $d-2$ steps. A rotation P_k is generatated to obtain $A_k = P_k A_{k-1} P_k$, where $k = 1, 2, .., d-2$. Consequently, we have that $T = P_1 P_2 ... P_{k-2}$. The name of QR algorithm is due to the iterative use of the QR decomposition. Thus it finds values for the matrices Q_k and R_k, given a matrix B_{k-1}, such that $B_{k-1} = Q_k R_k$. Starting from $B_0 = A_{d-2}$, in every iteration a value $B_k = R_k Q_k$ is computed; where $k = 1, 2, .., s$ and s is the iteration at which the error criteria is overcome. Once the algorithm has converged, the eigenvectors of B_0 are the rows of the matrix $C_s = Q_1 Q_2 ... Q_s$. Finally, the matrices E^2 and U are computed as $E^2 = diag(B_s)$ and $U = TC_s = P_1 P_2 ... P_{d-2} Q_1 Q_2 ... Q_s$.
The implementation receives XX^T as a parameter and computes E^2 together with U. The matrix E^2 is calculated by performing transformations over the original input correlation matrix getting a tridiagonal matrix and later the diagonal matrix of eigenvalues, this matrix will be called A. Meanwhile, the matrix U is computed at the same time, starting from an identity matrix. These two tables will keep their names during the execution of the algorithm, and they will use the subindex k to specify the step counter.

1	For k=1 to d-2
2	$\alpha = -sign(a_{k+1,k})(\sum_{j=k+1}^{d}(a_{jk})^2)^{\frac{1}{2}}$
3	$r = (\frac{1}{2}\alpha^2 - \frac{1}{2}\alpha a_{k+1,k})^{\frac{1}{2}}$
4	$w_1 = w_2 = ... = w_k = 0$
5	$w_{k+1} = \frac{a_{k+1,k} - \alpha}{2r}$
6	$w_j = \frac{a_{jk}}{2r}$ for $j = k+2, k+3, ..., d$
7	$P_k = I - 2ww^T$
8	$A_k = P_k A_{k-1} P_k$
9	$U_k = U_{k-1} P_k$
10	End

Figure 2: Householder Implementation.

The linear transformation defined by Householder [21] is composed by $d-2$ rotations P_k derived with the scalars α and r, and the vector w of n cardinality. Initially $A_0 = XX^T$ and $U_0 = I$ where I is the identity matrix. The implementation is shown in Figure 2.

1	Do
2	Find Q_k and R_k for $A_{k-1} = Q_k R_k$
3	$U_k = U_{k-1} Q_k$
4	$A_k = R_k Q_k$
5	While($error > \epsilon$)

Figure 3: QR Algorithm Implementation.

For the QR algorithm (see Figure 3), the number of steps required to reach a solution depends on an error criteria ϵ, which is computed out of the diagonal elements of A_k. The loop starts with $k = 0$ and at each step $A_{k-1} = Q_k R_k$ is calculated using the QR Factorization (see Figure 4), it becomes a nested loop of the iterative QR algorithm [21]; initially $Q_k = A_{k-1} = \{v_0, v_1, ..., v_d\}$, and the values of R_k are given by $R_k = \{r_{i,j}\}$. Notice that Householder is finished before the QR algorithm starts, therefore their step counters k are not the same.

1	For $i = 1$ to d
2	$r_{ii} = \|\|v_i\|\|$
3	$v_i = v_i / r_{ii}$
4	For $j = i+1$ to d
5	$r_{ij} = v_i \cdot v_j$
6	$v_j = v_j - r_{ij} v_i$
7	End
8	End

Figure 4: QR Factorization Implementation.

After completion of the main steps the eigenvalues need to be positive. Thus for every negative value $a_{i,i}$ in the diagonal of A, $a_{i,i} = -a_{i,i}$ and $u_{j,i} = -u_{j,i}$; where $j = 1, 2, ..., d$. Finally, $E^2 = \{a_{i,i}\}$ and $E^{-1} = \{\frac{1}{\sqrt{a_{i,i}}}\}$. In the next sections the detailed implementation of the main steps of SVD with SQL statements as well as the technical overview of the coding inside the database is exposed.

3. SVD IN SQL

In this section we explain the techniques for matrix multiplication, matrix transposition, along with the fundamental operations used in our SQL implementation. The detailed explanation of the steps of the algorithm can be found in Section 2.4, our SQL implements such definition. Further analysis of space, I/O and complexity of our SQL queries is performed for each step of the algorithm.

3.1 Table Definitions

In order to represent matrices as tables in a DBMS, we can define the columns as dimensions and rows as records. For the input matrix X the storage table, `tX`, contains the attributes $\{X_1, X_2, ..., X_d\}$. This matrix is used to compute the input correlation matrix of the algorithm in order to find its decomposition (see Section 2.4). Matrix operations, such as multiplication, can be performed with this representation by pivoting the left or the right operator to pair rows of one matrix with columns in the other and obtain the resulting table with an AGGREGATE statement.
Another way to store matrices is vertically [14]. Vertical layout matrices $\{i, j, val\}$ are implemented to perform multiplications and other matrix operations needed to solve the eigen-problem. There is no need to pivot multiplying matrices and it is a very convenient method to implement operations in the algorithm steps. For instance, the multiplication of a pair of matrices A and B, which are stored with vertical layout, could be done in one SQL statement with one JOIN condition and an AGGREGATE function. Moreover trivial values are not required to be stored, such as, zero values of an sparse matrix or values which can be computed from or are equal to other elements in the same matrix. We also exploit the fact that for a multiplication result, the element $\{i, j, val\}$ can only exist if there exists at least one element $\{i, k, val\}\}$ for the left operator and at least one match $\{k, j, val\}$ for the right operator. This is

used to avoid unnecessary pairing and to reduce the number of values involved in the AGGREGATE. Notice that the computation of a value during multiplication of two $d \times d$ matrices will have the cost of joining (pairing), a product operation and an aggregation of d values. Pursuing to alleviate the computation time of JOINs, which are the most expensive computation in SQL, our code uses constraints that cut down the unnecessary records during operations.
For PCA (see Section 2.4), the resulting principal components and their associated variances will be stored on tables `tU` and `tA` respectively. Table 1 has the summary of tables used for storage and their scope. In the QR factorization the tables `tQ` and `tR` have an extenal scope in the sence that they exist outside the loop, then the QR algorithm uses them to compute new values for the tables `tA` and `tU`. Therefore, tables `tQ` and `tR` are not needed beyond the scope of an iteration of the QR algorithm. Thus, even though names overlap for local table definitions, the only common tables between steps are `tU` and `tA`.

Table 1: Summary of tables in SQL.

Step	Global	Local
Correlation	tX, tA	tL, tQ
Householder	tA, tU	tW, tP, tH, tT
QR algorithm	tA, tU	tQ , tR, tT

Further optimization is done to take advantage of SQL when performing Householder and QR factorization. The details about how SQL statements and tables map the steps of the algorithm will be shown in the next sections.

3.2 Correlation Computation

The computation of n, L, Q are sufficient statistics to compute the correlation matrix (see Section 2.2). When the data set is in vertical layout, the table stores one correlation value per row as $\{i, j, \rho_{ij}\}$. Since the matrix is symmetric, it is sufficient compute a triangular matrix with the number of rows of the resulting matrix equal to the total number of correlation pairs ($d(d+1)/2$). The number of INSERT statements to compute n is 1, L is d and Q is $\frac{d(d+1)}{2}$. Retrieving every singular value with an AGGREGATE function involving a scan of `tX`, can be avoided with a more conservative approach [15, 17]. It is a single SELECT with $1+d+\frac{d(d+1)}{2}$ AGGREGATE statements using `tX`.

```
SELECT sum(1.0) AS n
,sum(X_1),sum(X_2)...,sum(X_d) /*L*/
,sum(X_1 * X_1) /*Q*/
,sum(X_2 * X_1),sum(X_2 * X_2)
⋮
,sum(X_n * X_1),sum(X_n * X_2),...,sum(X_n * X_n)
FROM tX;
```

The resulting query is stored on a temporary table and then inserted with vertical layout in the tables `tL` and `tQ`. Finally, the values of the correlation matrix are computed from the summary matrices without performing more scans on the original data set. The final table storing the correlation coefficients, `tA`, has 3 columns and $(d(d+1)/2)$ records.

3.3 Householder Transformation

Our implementation of Householder in SQL uses vertical matrices to exploit the storage of sparse matrices, and also other optimizations of operations. The resulting tables `tA` storing A_{k-2} and `tV` storing V_{d-2} are obtained using only INSERT statements. We use the auxiliary table `tH` to store the part of the matrix A_k that will be rotated during following iterations and the table `tT` to store the sub matrix of V_k that will continue to be involved in further operations. Such implementation is possible as the matrix P_k is composed of a diagonal identity matrix of k elements and a square matrix starting at the position $(k+1, k+1)$. Consequently, the values of P_k that are not trivial have the form $\{[k+1, d], [k+1, d], val\}$ and stored on the table `tP`. During each iteration instances of the tables `tH`, `tT`, `tP` and `tW` are computed, along with the variables `alpha` and `r`. Meanwhile the values inserted in `tA` and `tV` are never updated.
First `tH` is instantiated with the input correlation matrix. Values for the variables `alpha` and `r` are computed with SELECT and AGGREGATE statements over the table `tH`. The table `tW`, storing the vector w, has the form $\{[k+1, d], val\}$, not including zero values. Since `tW` has $d-k-1$ values and the table `tP` is symetric, only the upper or bottom diagonal matrix is stored. `tP` has size $((d-k+1)(d-k+2)/2)+1$, which is calculated with two statements: one to insert the record $\{k, k, 1\}$, necessary to include the column and the row k in our operation results, and other to compute the difference between I and the cross product of the values in `tW`. As shown in Figure 5, during iteration k only the sub

$$\begin{bmatrix}
a & a & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
a & a & a & \dots & 0 & 0 & 0 & \dots & 0 \\
0 & a & a & \dots & 0 & 0 & 0 & \dots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \dots & a_{k,k} & a_{k,k+1} & 0 & \dots & 0 \\
0 & 0 & 0 & \dots & a_{k+1,k} & h & h & \dots & h \\
0 & 0 & 0 & \dots & 0 & h & h & \dots & h \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \dots & 0 & h & h & \dots & h
\end{bmatrix}$$

Figure 5: Storage of matrix A at step k, in `tA`:$\{i, j, a_{ij}\}$ and `tH`:$\{i, j, h_{ij}\}$.

matrix $\{[k, d], [k, d], value)\}$ changes from A_{k-1} to A_k. The rotation `tP` of `tH` from the previous iteration with the form $\{[k, d], [k, d], val\}$ has the elements $a_{k,k}$, $a_{k+1,k}$ and $a_{k,k+1}$ of the final matrix A_{n-2} to be stored on `tA`, also the sub matrix $\{[k+1, d], [k+1, d], val\}$ to be the new instance of `tH`. The matrices stored on `tH`, `tP`, `tA` are symetric, therefore only the triangular upper or lower matrix is stored, moreover computations are cut down only for such values. Nevertheless, it is important to include the values not stored at the moment of doing the aggregations of matrix multiplications.

Since P_k is applied as a transformation at the right side of U_{k-1}, it only alters values for the columns of the range $[k+1, d]$. At each iteration a new instance for the table `tT` is generated, which stores a sub matrix of U_k (see Figure 6) with tuples of the form $\{[2, d], [k+2, d], val\}$ obtained from the multiplication between the instance of `tT` at it-

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & u & u & \dots & u_{2,k+1} & t & t & \dots & t \\ 0 & u & u & \dots & u_{3,k+1} & t & t & \dots & t \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & u & u & \dots & u_{k+1,k+1} & t & t & \dots & t \\ 0 & u & u & \dots & u_{k+2,k+1} & t & t & \dots & t \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & u & u & \dots & u_{d,k+1} & t & t & \dots & t \end{bmatrix}$$

Figure 6: Storage of matrix U at step k, in `tU`:$\{i, j, u_{ij}\}$ and `tT`:$\{i, j, t_{ij}\}$.

eration $k-1$ and the values in tP. The remaining tuples $\{[2, d], k+1, val\}$ are to be inserted in `tU`.

3.4 QR Factorization

As described in Section 2.4, the QR factorization is iteratively called by the QR algorithm. In each step, a row of R_k is computed and the columns v_j of Q_k where $j \geq i$ are modified. As shown in Figure 7, Q_k is stored on `tT` and `tQ`. The table `tT` is initiated with a copy of the input matrix A_{k-1}. For the step i, v_i and $r_{i,i}$ are computed, both to be inserted in `tQ` and `tR` respectively. Vertical matrix representation allows us to compute the internal loop with three SQL statements. One for the column of R_k $\{[i], [i, d], val\}$, with the AGGREGATE of the JOIN between v_i and `tT` on the row value. The second to compute the new instance of `tT`, with a join between the previous `tT`, v_i and `tR`. Notice that this matching will result on records of the form $\{[1, i+1], [i+1, d], val\}$, consequently the remaining $\{[i+2, d], [i+1, d], val\}$ from the old `tT` is inserted with a third statement. Given $@i = i$ and $@r = r_{i,i}$, the SQL is as follows,

```
/*New values for tQ*/
INSERT INTO tQ
SELECT tT.i, tT.j, tT.val / @r
FROM tT
WHERE tT.j = @i;

/*New values for tR*/
INSERT INTO tR
SELECT @i, @i, @r
UNION ALL
SELECT tQ.j, tT.j, sum(tQ.val * tT.val)
FROM tQ,tT
WHERE tQ.j=@i AND tT.j>@i AND tQ.i=tT.i
GROUP BY tQ.j, tT.j;

/*New instance of tT*/
INSERT INTO new_tT
SELECT tT.i, tT.j, tT.val - tQ.val * tR.val
FROM tQ, tT, tR
WHERE tT.j>@i AND tQ.j=@i AND tR.i=@i AND tR.j=tT.j
      AND tQ.i=tT.i
UNION ALL
SELECT i, j, val
FROM tT
WHERE tT.j>@i and tT.i>@i+1
```

Finally, we have that no UPDATE statement is required to obtain Q_k and R_k. Which is very close from the database point of view. And the only table that needs to be recomputed per iteration is `tT`.

$$\begin{bmatrix} q & q & q & \dots & q_{1,i} & t & t & t & \dots & t \\ q & q & q & \dots & q_{2,i} & t & t & t & \dots & t \\ 0 & q & q & \dots & q_{3,i} & t & t & t & \dots & t \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & q_{i,i} & t & t & t & \dots & t \\ 0 & 0 & 0 & \dots & q_{i+1} & t & t & t & \dots & t \\ 0 & 0 & 0 & \dots & 0 & t & t & t & \dots & t \\ 0 & 0 & 0 & \dots & 0 & 0 & t & t & \dots & t \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & t \end{bmatrix}$$

Figure 7: Storage of matrix Q_k at step i, in `tQ`:$\{i, j, q_{ij}\}$ and `tT`:$\{i, j, t_{ij}\}$.

3.5 QR Algorithm

During each iteration of the QR algorithm, there is a QR factorization, two matrix multiplications and an *error* computation. Since the convergence iteration is not known beforehand, new solution tables `tU` and `tA` are computed each stage. The SQL for QR factorization was explained in Section 3.4. As for the multiplications, only optimization of the computation A_k is feasible. Since A_k is a tridiagonal symetric matrix, the SQL is constrained to calculate the main and one of side diagonals of the multiplication between the matrices in the tables `tR` and `tQ`, result to be stored on `tA`. The table `tU` with the value of U_k is computed with a query of the previous instance of `tU` and `tQ`. A value for the variable *error* is computed with a SELECT of the maximum difference of the diagonal elements between the tables storing A_{k-1} and A_k. Finally, the loop will conclude when the value of the *error* is less or equal the parameter ϵ.

3.6 Implementation Alternatives

Data access and manipulation in a DBMS could be done using different tools, hence, PCA is not restricted to SQL statements. Here we explain how external libraries, UDFs and Database Connectivity APIs are used to implement the main steps of PCA: correlation computation and eigenvalue decomposition. Even though external libraries are specialized pieces of software that have been developed over time to optimize execution time of their algorithms, most of them are not designed to overcome memory heap constraints. Such limitation is evident when attempting to apply large data sets to these external libraries for statistical and data mining analysis [7]. APIs, like JDBC or ODBC, allow the user to access and manipule data from external programs. This collection of libraries grant data mining applications connectivity to data sources from different DBMS providers and they can be used to minimize memory usage. Database connectivity components are extremely portable, for most of them few properties need to be changed along with the driver to switch the DBMS. Since the execution speed depends on the communication channel and for security reasons, it is desirable to minimize the amount of data transmitted through the network. Other exploitable feature

of a DBMS are user defined functions or UDFs, which are compiled to be embedded into the DBMS and can be called using SQL statements. Therefore, a UDF executes inside the DBMS without delay transmitting information. On the other hand, UDFs depend on the DBMS specification, making them not portable among providers and sometimes not feasible due to access constraints. Our focus is to optimize performance of PCA computation for large data sets by minimizing memory allocation and execution time, without altering correctness of the results.
So far we have explained how to perform each step of SVD with SQL statements(see Section 3). To avoid redundancy, the structural components of the other implementation alternatives besides SQL are generalized with pseudo-code. The correlation matrix can be computed with one pass over the original data set [16]. Reading one record in the table at a time, the summary matrices are computed as an aggregation. Thus, a data structure is used to store n, L and Q, where the values obtained by combining attributes of the current record are accumulated. After finished over the whole or part (sampling) of the table, the correlation matrix is generated as specified in Section 2.2. Finally, the steps for eigenvalue decomposition, given the correlation matrix as an array of two dimensions, are mapped for most programming languages [19, 4, 10]. For our experiments we use the algorithm specification in Section 2.4.

3.7 Time and Space complexity

For the computation of the summary matrices L and Q, along with the covariance matrix, the complexity of operations in the algorithm (Section 3.2) are $O(dn)$ and $O(d^2)$ respectively, the same complexity is shared by the SQL, Java and the UDF implementations. In the other hand, for the SVD (Section 2.4) all implementations maintain the same steps for the algorithm. However, for SQL operations, we need to take into account the complexity of cross product and joins. This problem is alleviated with reduction of records participating in computations by constraints and by not physically represent elements with predictable values. Complexity of the SVD [19] is analyzed by operation and iteration. Solving Householder is $O(d^3)$, with $n-2$ steps of $O(d^2)$ each. Table 2 shows the number of operations for Java and the UDF, along with the records involved in operations for SQL. Even though each step of the QR factorization is $O(d^2)$ (see table 3), after completion of d steps the overall complexity is $O(d^3)$. Once again the representation in SQL incorporates equal or less records for operations. The QR algorithm includes matrix multiplications besides the QR factorization, preserving complexity of $O(d^3)$ per step. However, with the number of iterations for convergence s, the QR algorithm is $O(sd^3)$. In table 3 we observe that eventhough the records for U are not reduced in SQL, only the elements on the three diagonals of A are computed.

As for the implementation alternatives to SQL, space requirements are static. Operations of matrices are done by changing values already allocated in memory. In SQL the space requirements are closely related to the number of records used in operations or query statements. Therefore, tables in Householder and the QR factorization reduce on size when the step counter increases. Reducing the numerosity of records taking part in the join statements, improves the execution speed of this approach when compared to the counter scenario were every single value is computed for each step of the algorithm.

Table 2: Householder I/O per step.

Matrix	UDF & Java	SQL
A	$\frac{(d+1)(d)}{2}$	$\frac{(d-k+2)(d-k+1)}{2}$
U	d^2	$(d-k+1)^2$

Table 3: QR algorithm I/O per iteration.

Matrix	UDF & Java	SQL
Q_k	$\sum_{i=1}^{d} d(d-i)+d$	$\sum_{i=1}^{d} i(d-i)+d$
A	$d \times d$	$2d-1$
U	$d \times d$	$d \times d$

4. EXPERIMENTAL EVALUATION

In this section we present the experimental evaluation on SQL Server DBMS. The server had an Intel Dual Core CPU, 2.6GHz, 4GB of memory and 1TB on disk. The DBMS ran under the Windows XP operating system and has implementations of the UDFs compiled. We also used a workstation with a 1.6 GHz CPU, 256 MB of main memory and 40GB on disk with JDBC connection, a SQL script generator, an implementation of SVD, JAMA and the R application for comparison purposes. We implemented three UDFs: one to compute the correlation matrix, a second to compute the SVD given a matrix and a third that computes the correlation matrix together with its SVD decomposition to return the principal components of the given table. Our optimizations combine UDFs and SQL to improve execution time. Likewise, the optimizations are also implemented with JDBC by executing some operations inside the DBMS with SQL statements and others in the workstation (see Section 3.6). For R, all the data sets were transformed into flat files, the time of exportation was not taken into account during speed measurements. JAMA has a JDBC connection to access data in the DBMS, therefore measurements include time taken to import data.
We used three real data sets from the UCI Machine Learning Repository. This information comes from different backgrounds which require dimensionality reduction. The first data set contains general information on Internet users with a combination of categorical and numerical data(n=10000, d=72). A second data set is of cartographic variables for forest cover type prediction of 30×30 meter cells (n=100000, d=54). The last data set is of US Census Data with numeric and categorical attributes (n=100000, d=62).
We varied the number of dimensions d for time performance comparisons and to test scalability. Since the results are numerical approximations, we use the maximum expected relative error δ of the most representative eigenvalue as measure of precision. Notice that the only parameter for execution is the convergence criteria ϵ which can be modified to warranty any desired expected maximum error. The external Java package JAMA (A Java Matrix Package) and R do not

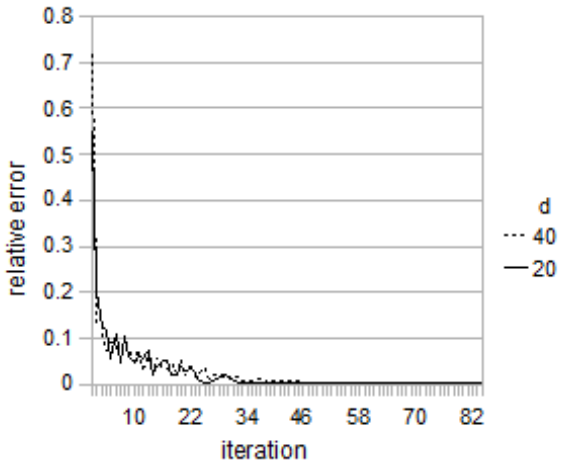

Figure 8: PCA with SQL relative error (δ).

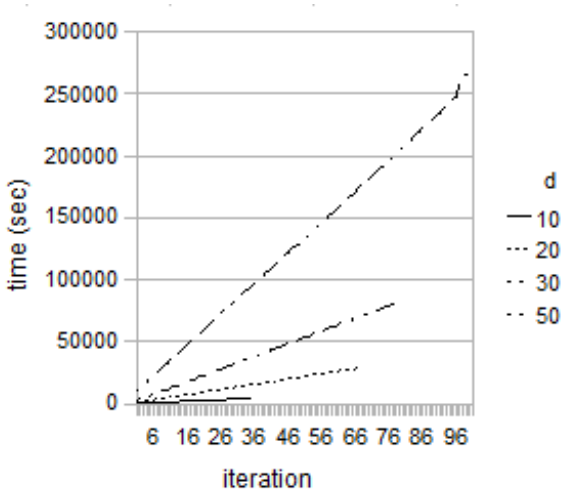

Figure 9: PCA with SQL time.

require to set any parameter to compute the eigenvalue decomposition. Considering that all the implementations converge to the same direction, yet all solutions are numerical approximation, we report correctness based on the maximum error expectation. Given an input table of $d \times n$, the resulting principal components or eigenvectors matrix will have size $d \times d$. Finally, we have that each experiment is repeated 5 times and the average is reported.

4.1 Result Correctness

Our goal is to minimize the execution time without affecting the correctness of solutions. In Section 3, we showed the implementation of our QR algorithm (see Section 2.4) in SQL, without modifying any step or main operation. Optimizations are required not only to minimize the storage requirements of the matrices, but also to reduce accumulated error; since trivial or known zero values are cut off the operations. As expected, roundoff errors can swamp the process from the true solution. Also in SQL we are bound to work with the floating point operation precision of the DBMS. However, results show that the optimizations efficiently help to prevent accumulated roundoff and floating point operation errors.

Convergence of the SQL implementation solving SVD over the US Census data set ($n = 1000$) is presented in Figure 8. The plot shows how the relative error of the eigenvalue diagonal rapidly decreases during the first steps of the algorithm eventually converging bellow the desired error criteria. As presented in Figure 9 the execution time of the SQL implementation quickly increases with the number of dimensions d. Since SVD operates over a $d \times d$ matrix, the eigenvalue decomposition is not affected by any change on the number of instances in the data set. We explained in Section 3.1 how the implementation reduces the number of values to be computed during each iteration to the minimum. Also how we take advantage of SQL and our matrix representation to acomodate matrix operations in such a way that only sub matrices interact during each step. Moreover, a value computed will only be used no more than two times before the table where it is contained is replaced by a new instance. However, the volatile nature of the matrices and the number of cross joins performed during every execution are not to be avoided. The remarkable advantages of an SQL implementation is to be linearly scalable with n and not to have a memory heap limitation. Therefore, this approach can be used to compute PCA for a table of any size. Finally, more analysis needs to be done comparing the results against different external tools. From our experiments we have that our implementations and external libraries converge to the same direction, yet all the values vary on decimal places. Such analysis would focus on the impact of roundoff errors in the SQL floating point operations and how different algorithms to solve SVD perform better for certain datasets.

4.2 Implementation comparison

Based on the algorithm, the first step is to compute the correlation matrix. Table 4 shows a computation time of the correlation matrix in SQL and the time required by an external library to generate the same matrix in main memory. The JAMA package outperforms SQL for all the cases of $n = 10$. Since L requires d and Q a number $d \times (d+1)/2$ INSERT queries, it is not beneficial to use SQL when the number of records is relatively small. However, for all the cases with $n \geq 100$, the advantage of executing operations inside the database is evident. The database not only overcame the memory issue, but also surpassed Java execution time. When correlation is computed vertically, the number of scans done to the original data set increases with d. For convenience we improved our vertical correlation matrix computation using one horizontal aggregation query with all the values for n, L and Q (see Section 3.2). Therefore, no more than one scan of the data is needed. The time of inserting the resulting attributes of the query, one at a time in the vertical result table, is insignificant when compared to the multiple scans of a table with a large number of rows. SQL Server limits the number of columns based on the total size of the row, which is capped at 8KB. As a result, when only using floating point columns, the maximum number of dimensions that can be computed with one scan is 42. A larger number of dimensions would require additional scans on the original table because we need to split the aggregate in order to fit the maximum row size allowed by the DBMS.

This explains why the horizontal aggregate is outperformed by the vertical aggregate when $n = 10$ and $d \geq 50$. The UDF implementation (see Section 3.6) looks promising in comparison to the SQL when $n \leq 100$, yet not enough to have faster results than JAMA if the data set is relatively small $n = 10$. Finally, we have that the horizontal aggregate dominates for $n \geq 100$, due to DBMS optimizations to perform aggregates fast on large tables.

Table 4: Correlation computation time in seconds with JAMA using JDBC (*out of memory).

$n \times 1000$	d	Vertical aggregate	Horizontal aggretate	UDF	JAMA
10	30	7	2	3	5
10	50	20	21	16	11
10	70	40	59	34	12
100	10	3	1	2	13
100	20	10	7	7	25
100	30	23	16	16	32
100	50	57	53	47	*
1000	20	>1K	33	72	*
1000	40	>1K	154	261	*

Table 5 compares the execution time of SVD for our SQL and UDF implementations. They use the correlation matrix from the previous step to generate a numeric SVD decomposition. The relative error δ with respect to the most representative eigenvalue is set the same to compare speed. Since both implementations overcome the error criteria after certain number of steps s, there is no precision lost when executing operation in SQL. However, the price to pay is on the number of iterations executed. Considering that all the operations are done in main memory, it was expected for the UDF implementation to surpass SQL in speed. Notice that the time to compute the correlation matrix matrix was not taken in account during measurement. In order to optimize execution speed of PCA, we can combine the different implementations, this comparison is analyzed bellow.

Table 5: SVD computation time in seconds with s iterations for convergence.

$n \times 1000$	d	δ	SQL	s	UDF	s
10	30	3.76^{-3}	92	78	1	80
10	50	3.65^{-3}	481	146	3	115
10	70	2.92^{-3}	978	172	5	122
100	10	3.49^{-5}	6	34	<1	34
100	20	2.71^{-4}	30	66	2	66
100	30	2.52^{-3}	82	77	4	76
100	50	2.44^{-3}	506	174	10	126
1000	20	2.07^{-4}	39	84	<1	84
1000	40	1.15^{-3}	113	64	2	62

4.3 Optimizations

In this section we present strategies to improve the computation of PCA. For comparison purposes we used the statistic package R and the Java library JAMA. Since the execution of R is independent from the DBMS and data files are used as input, there is no importation cost or connection delay with the database. R is dominant as long as memory limitations are not reached. It has the reading cost from the hard drive and I/O operations at memory level which increases linearly with the data size $n \times d$. Our results show that we can get similar execution speed with minimal memory requirements and without compromising solutions. In the other hand, JAMA is used with a JDBC connection to import the data into main memory before execution. It has I/O operation cost of the JDBC connection, plus execution time in main memory. Such implementation point up the main limitation of data mining applications, to find the appropriate way to manipulate data inside the DBMS. Like R, JAMA has memory heap allocation constraints. It is important to analyze the strength of each implementation since an optimal configuration depends on environment constraints and data set size. Table 6 presents total execution time of our implementations. The first of them has every step of PCA with SQL statements. We also combined the SQL correlation computation with SVD as a UDF in the DBMS, and outside with JDBC and Java. Furthermore, PCA is implemented as a UDF which receives an input table and returns its principal components as tables in the DBMS.

Table 6: PCA execution time comparison in seconds against R without flat file creation time cost, and JAMA with JDBC (*out of memory)(*/* for hybrid methods).

$n\times$ 1000	d	SQL	SQL/ UDF	SQL/ Java	UDF	R	JAMA
10	30	96	3	6	3	3	8
10	50	501	22	23	17	4	11
10	70	1020	47	50	34	6	13
100	10	7	2	4	2	5	14
100	20	37	9	9	8	8	27
100	30	98	20	20	17	13	34
100	50	559	62	67	47	21	*
1000	20	72	34	39	72	*	*
1000	40	267	156	158	262	*	*

The implementation that uses JDBC and JAMA outperformed our approaches when the data set size is relatively small $n = 10$. However, for larger data sets, the external libraries suffer because the memory limitation prevents them from analyzing these data sets. Likewise, time difference with R is more evident for the same data set when $d = 50$ and $d = 70$. The reason for this results is that we focused on improving execution time for large data, with the flaw of not performing so fast for small data sets.

The main weakness of our SQL implementation is data set dimensionality. To solve this issue, we also implemented a version of the program with DBMS connection. It uses JDBC to execute queries that compute the correlation matrix, extracts it from the database and solves the eigenvalue decomposition. The combination of this two techniques yields a desirable implementation for a data mining implementation which performs remarkably for large data sets. Results show growth that is similar to the SQL/UDF implementation, which uses a similar procedure to calculate PCA with correlation computation inside the DBMS and the UDF to compute SVD. The difference between both implementations is due to the transmission velocity of the data containing the correlation matrix. SQL/Java requires the transmission of data through the communication network. Unlike our dedicated server for experiments, transmition

time can increase on a concurrent network. The two implementations assume the correlation matrix can be stored on main memory. Therefore, computations of SVD, the $O(sd^3)$ numerical iterative process, execute extremely fast. PCA implemented as UDF has better result for data with moderate size. With the data set of $n = 100$ its results are similar to our SQL implementation. Having the UDF to perform only one scan of the data set is efficient and has fewer speed impact when the number of dimensions increases. However, optimizations inside the DBMS to execute aggregate queries are the more efficient approach for large data sets. Our results show performance, along with a guideline to select the best combination, for speed and correctness, depending on the data set characteristics. Since we present the impact of the data set size n and dimensionality d on the options available to perform PCA over information stored on a DBMS. The current work gives a precise perspective of the weaknesses and strengths of each implementation alternative to solve SVD and PCA.

Table 7: Execution time percentage (%) of PCA computation (*/* for hybrid methods).

		Correlation			SVD		
$n\times$			SQL/	SQL/		SQL/	SQL/
1000	d	SQL	UDF	Java	SQL	UDF	Java
10	30	2	67	33	98	33	67
10	50	3	95	91	97	5	9
10	70	3	85	80	97	15	20
100	10	14	50	25	86	50	75
100	20	19	78	78	81	22	22
100	30	43	56	56	57	44	44
100	50	9	85	79	81	15	21
1000	20	45	97	85	55	3	15
1000	40	58	99	97	42	1	3

5. RELATED WORK

There is a wide range of sucessful applications based on PCA for image processing [6] and pattern recognition [23]. Data mining has also exploited dimensionality reduction for data compression [3], clustering [5] and classification [9]. Although there has been considerable amount of work in machine learning and data mining to develop efficient and accurate techniques, most data mining work has concentrated on proposing efficient algorithms assuming the data set is a flat file outside the DBMS. Statistics and machine learning have paid little attention to large data sets, whereas that has been the primary focus of data mining.

Most research work has focused on modifying existing algorithms for sparse data, and to optimize methodologies to solve SVD [4]. One of the recent research extensions of PCA is feature selection [1], to define a subset of the original set of attributes that represents characteristics of the data with minimal information lost. Considering that nowadays most of the information is captured using a DBMS, this methodologies lack tools to appropiatetly do the preprocessing required to execute its algorithms.

There are some research in databases involving visualization [12] and text mining using PCA. SVD has been used in database research to perform Collaborative Filtering (CF) [18] and information retrieval [2]. PCA [8] and correlation analysis are two common techniques to analyze data. Investigation has been done to get statistical models of the data inside the DBMS fast [16, 13, 17]. In order to add PCA capabilities to a DBMS, a recent approach was to use UDFs to implement the complex matrix operations involved in SVD for microarray data analysis [20]. We have gone beyond integrating such methodologies with UDFs in a non declarative programming language such as C++ or Basic. Our approach is to implement efficiently every operation in PCA with SQL statements, providing guidelines of time performance, correctness and scalability of the alternatives to incorporate PCA into the DBMS.

6. CONCLUSIONS

Our proposal is about integrating PCA computation capabilities with a DBMS using SQL queries and UDFs. We focus on solving PCA with singular value decomposition (SVD). We explored techniques in order to improve execution time, get correct results and achieve scalability. Since memory size is a limitation of statistical packages, our methods overcome memory limitation without compromising accuracy and improving performance. To build a summarization of the data set the number of scans was reduced down to one, and the resulting summarization is used to compute the correlation matrix. Computing a decomposition of the correlation matrix to find the principal components is a complex methodology that requires several matrix operations. We studied how efficiently compute SVD with different alternatives, including SQL statements, UDFs and external libraries. We proposed some optimizations for each alternative to improve time performance, we tested several combinations to overcome limitations of our implementation alternatives. The results show that our scheme can execute fast for large data sets compared to statistical packages and to find solutions when they have reached memory limitations. In order to test the data mining client experience, our experiments are done from a workstation with a database connectivity API to execute operations outside the DBMS. We experimentally compared all the alternatives, we found that computing the correlation matrix with SQL statements is faster for large data sets, due to sufficient summarization matrices. UDFs are faster to compute SVD, since the algorithm used is iterative. Consecuently, the hybrid method of computing the correlation matrix with SQL and SVD with a UDF, has the best time performance for large data sets. The scalability is linear in data set size and cubical in dimensionality. Our solution can work efficiently completely inside the DBMS.

These are several issues for future research. Statistical and data mining techniques can benefit from our approach to compute PCA inside the DBMS. Deep analysis of how to best implement numerical methods with minimal I/O against the DBMS. The column subset selection problem is to find a subset of the most important attributes of a data set, this can be done exploiting our scheme to compute PCA, more analysis of SQL and UDFs to compute the NP-complete procedure is still required. Moreover, we have only considered the problem when the input data set has more records than dimensions, and focused on optimizing performance for large data sets. Future work must include number of dimensions greater than records, for which our implementations are not efficient in execution time.

7. REFERENCES

[1] C. Boutsidis, W.M. Mahoney, and P. Drineas. Unsupervised feature selection for principal components analysis. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–69, New York, NY, USA, 2008. ACM.

[2] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228, 2002.

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

[4] A. d'Aspremont, F. Bach, and L. Ghaoui. Optimal solutions for sparse principal component analysis. *J. Mach. Learn. Res.*, 9:1269–1294, 2008.

[5] C. Ding and X. He. K-means clustering via principal component analysis. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 29, New York, NY, USA, 2004. ACM.

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

[7] J. Han and M. Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, September 2000.

[8] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning*. Springer, New York, 1st edition, 2001.

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

[10] N. Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *J. Mach. Learn. Res.*, 6:1783–1816, 2005.

[11] S. Mosci, L. Rosasco, and A. Verri. Dimensionality reduction and generalization. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 657–664, New York, NY, USA, 2007. ACM.

[12] V. Nadimpally and M.J. Zaki. A novel approach to determine normal variation in gene expression data. *SIGKDD Explor. Newsl.*, 5(2):6–15, 2003.

[13] C. Ordonez. Horizontal aggregations for building tabular data sets. In *DMKD '04: Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 35–42, New York, NY, USA, 2004. ACM.

[14] C. Ordonez. Vertical and horizontal percentage aggregations. In *SIGMOD Conference*, pages 866–871, 2004.

[15] C. Ordonez. Optimizing recursive queries in sql. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 834–839, New York, NY, USA, 2005. ACM.

[16] C. Ordonez. Building statistical models and scoring with udfs. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1005–1016, New York, NY, USA, 2007. ACM.

[17] C. Ordonez and J. García-García. Vector and matrix operations programmed with udfs in a relational dbms. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 503–512, New York, NY, USA, 2006. ACM.

[18] H. Polat and W. Du. Svd-based collaborative filtering with privacy. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 791–795, New York, NY, USA, 2005. ACM.

[19] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 2007.

[20] W. Rinsurongkawong and C. Ordonez. Microarray data analysis with pca in a dbms. In *DTMBIO '08: Proceeding of the 2nd international workshop on Data and text mining in bioinformatics*, pages 13–20, New York, NY, USA, 2008. ACM.

[21] S. Salleh, A.Y Zomaya, and A.B. Sakhinah. *Computing for numerical methods using visual C++*. Wiley, Hoboken, NJ, 2008.

[22] Polyxeni Zacharouli, Michalis Titsias, and Michalis Vazirgiannis. Web page rank prediction with pca and em clustering. In *WAW '09: Proceedings of the 6th International Workshop on Algorithms and Models for the Web-Graph*, pages 104–115, Berlin, Heidelberg, 2009. Springer-Verlag.

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