Microarray Data Analysis with PCA in a DBMS

Waree Rinsurongkawong *
University of Houston
Department of Computer Science
Houston, TX 77204, USA

Carlos Ordonez
University of Houston
Department of Computer Science
Houston, TX 77204, USA

ABSTRACT

Microarray data sets contain expression levels of thousands of genes. The statistical analysis of such data sets is typically performed outside a DBMS with statistical packages or mathematical libraries. In this work, we focus on analyzing them inside the DBMS. This is a difficult problem because microarray data sets have high dimensionality, but small size. First, due to DBMS limitations on a maximum number of columns per table, the data set has to be pivoted and transformed before analysis. More importantly, the correlation matrix on tens of thousands of genes has millions of values. While most high dimensional data sets can be analyzed with the classical PCA method, small, but high dimensional, data sets can only be analyzed with Singular Value Decomposition (SVD). We adapt the Householder tridiagonalization and QR factorization numerical methods to solve SVD inside the DBMS. Since these mathematical methods require many matrix operations, which are hard to express in SQL, query optimizations and efficient UDFs are developed to get good performance. Our proposed techniques achieve processing times comparable with those from the R package, a well-known statistical tool. We experimentally show our methods scale well with high dimensionality.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Singular value decomposition*

General Terms

Algorithms, Performance

Keywords

DBMS, PCA, SQL, gene

*This work has been partially conducted by the author at the MD Anderson Cancer Center.

DTMBIO'08, October 30, 2008, Napa Valley, California, USA.

1. INTRODUCTION

Microarray technology enables rapid analysis of expression levels of thousands of genes. Data sets generated from microarray experiments are typically stored in a DBMS in the form of a very large matrix that is organized by genes versus tissue samples from patients. Due to the lack of comprehensive statistical tools inside a DBMS and the high-dimensional nature of the data set, researchers generally export data sets from the DBMS to a client machine to perform analysis with external statistical packages. Previous research has shown it is possible to extend the statistical capabilities of the DBMS with SQL code generation and User Defined Functions (UDFs) [14]. This eliminates the time-consuming process of exporting the data, thus reducing client and network overhead. Such approach exploits the computational resources of the server, which is typically more powerful both in terms of processing power and available memory. Furthermore, performing analysis inside the DBMS enhances data security.

We propose a system that efficiently performs the two most common statistical analyses for microarray data inside a DBMS (see Figure 1): correlation analysis and Principal Component Analysis (PCA)[10]. The correlation matrix can be used as input for PCA, a technique to reduce the number of dimensions. This is especially useful when applied to microarray data sets due to their high dimensionality. Microarray data set typically have a high number of dimensions obtained from a few records. Interestingly, the number of correlation coefficients derived from such microarray data is the square of the number of dimensions, a value that could be as high as 100 million values. Such size evidently creates challenges to analyze the data set inside the DBMS: The data set has to be pivoted to overcome limitations on the maximum number of columns. This paper explains how to solve PCA using Singular Value Decomposition instead of traditional methods. SVD is a method to compute PCA more efficiently for ill-conditioned data sets, such as the microarray data set. SVD requires solving the Householder tridiagonalization and QR factorization [3, 7], (an eigen-value decomposition method), two techniques that require numerical analysis computations. We introduce several optimizations in SQL and UDFs, which are necessary to improve the efficiency of the SVD implementation.

The article is organized as follows. Section 2 presents definitions. Section 3 presents our contributions and includes univariate and bivariate analysis, PCA with SVD implementation, and component scores and loadings. Section 4 presents experiments comparing different methods of cor-

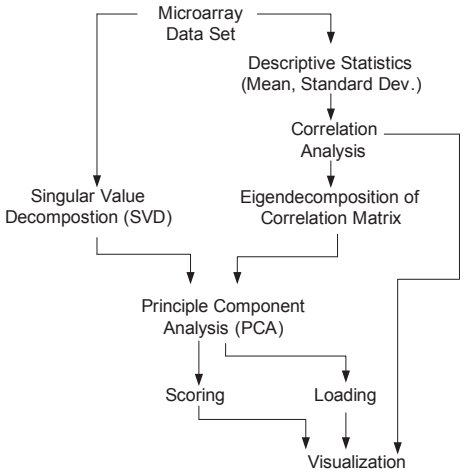

Figure 1: Analysis Diagram.

relation analysis, SVD implementation inside and outside DBMS, SVD optimizations, and time complexity. Section 5 discusses related work. Section 6 concludes the article.

2. DEFINITIONS

Let $\tilde{X} = \{\tilde{x_1}, \ldots, \tilde{x_n}\}$ be the input data set (microarray data set) and $X = \{x_1, \ldots, x_n\}$ be the the transformed microarray data set having zero mean (i.e. centered at the origin). X has n records (tissue samples), d dimensions (gene expression levels), an additional disease binary variable, and k rank, where $d \geq n \geq k$. The data set dimensions need to be scaled especially when the units of measurement are different. We use the subscript i for records and subscripts h, a, b for the dimensions. For one dimension, each x_{hi} represents the gene expression level of the tissue sample. X may have an additional target variable (cancer disease in our case), where y_i indicates whether the tissue sample is diseased or not.

3. MICROARRAY DATA ANALYSIS

In this section, we explain how to calculate the correlation matrix in SQL. The correlation matrix is used as an input matrix for PCA. We explain why traditional methods cannot solve PCA due to microarray data sets having more dimensions than records. Based on such limitation we explain how to implement the Singular Value Decomposition (SVD) method combining optimized SQL queries and efficient UDFs.

3.1 Correlation Analysis

A correlation matrix describes the degree of correlation among d dimensions or the degree of correlation among n records. It is a symmetric $d \times d$ or $n \times n$ matrix consisting of correlation coefficients. Since the matrix is symmetric, it is sufficient compute a triangular matrix.

The correlation matrix can be calculated using two fundamental matrices, L and Q. Let $L = \sum x_i$, resulting in a $d \times 1$ matrix. Let $Q = \sum x_i x_i^T$ be the quadratic sum of points for each pair of dimensions across all records resulting in a $d \times d$ matrix. Thus, the $d \times d$ linear correlation matrix

Table 1: Number of rows and columns of the Vertical, L-Horizontal, and I-Horizontal initial tables.

Initial Tables	No. of rows	No. of columns
Vertical	dn	3
L-Horizontal	n	d
I-Horizontal	d	n

can be expressed in terms of L and Q as:

$$\rho_{ab} = \frac{nQ_{ab} - L_a L_b}{\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2}} \quad (1)$$

SQL queries are used to compute n, L, Q. There are three alternatives to generate the correlation matrix: SQL-Vertical, SQL-L-Horizontal, and SQL-I-Horizontal. For comparison purposes, we also compute the correlation matrices using aggregate UDFs for the L-Horizontal table and scalar UDF for the I-Horizontal table. The critical issues for correlation analysis of high-dimensional data are memory space consumption and execution time.

When the data set is in vertical format, the table stores one value per row (see Table 1). The number of rows of the resulting correlation matrix is equal to the total number of correlation pairs ($d(d-1)/2$ rows). In the SQL-Vertical method, the L and Q matrices are first created. In order to create the Q matrix, a self-join operation is performed on the input table to get the set of different pairs of dimensions without duplication. Afterwards, L and Q are computing using an aggregation. We now present the L-Horizontal and I-Horizontal Methods in more detail.

L-Horizontal Method

When d is small, the SQL-L-Horizontal method is the most efficient approach to get the Q matrix[14]. This approach takes an $n \times d$ input matrix (see Table 1) and returns a table with $1 \times (d(d-1)/2)$ columns. The method requires just a single table scan to compute L and Q. Unfortunately, database tables typically have a limit on the number of columns. In the case of high-dimensional data, this method presents important drawbacks since the data set has to be partitioned into several tables. To work within DBMS limitations, the SQL-L-Horizontal method takes the $n \times d$ data set and partitions it vertically into several "lean" tables. However, within the correlation matrix, each dimension needs to be paired with all other dimensions, not only within the same table, but also with those in the other tables. Therefore, every table must be joined with every other table, rendering the method impractical due to excessive memory resources.

I-Horizontal Method

The I-Horizontal output table is a $d \times n$ table is obtained by pivoting the vertical format table to get records as rows and dimensions as columns. The SQL-I-Horizontal method is an optimized version of the SQL-Vertical method. Since the I-Horizontal table has less rows than the vertical format table, joining the tables is more efficient. As the number of dimensions increases, the SQL-I-Horizontal method takes significantly less execution time than the SQL-Vertical method. In addition, the I-Horizontal method is more efficient than the L-Horizontal method because it requires only one self-join, whereas the L-Horizontal method needs several joins for all

different pairs of small tables. The computation with SQL queries is shown below:

```
SELECT h, (A.rec1 + ... + A.recN) FROM X;   /*L*/

SELECT A.h, B.h
 ,(A.rec1 * B.rec1 + ... + A.recN * B.recN) /*Q*/
FROM X A CROSS JOIN X B
WHERE A.h >= B.h;
```

The SQL statements produces the number of result values for a triangular matrix not for the full matrix, which eliminates the duplicate pairs from the calculation. The UDF-I-Horizontal method is implemented similar to the SQL-I-Horizontal method. It accepts n, L and Q of a dimension pair as parameters and returns a correlation coefficient of the dimension pair. The UDF is called $d(d-1)/2$ times to produce correlation coefficients of all dimension pairs. The method requires a self-join operation similar to SQL-I-Horizontal method. Though the UDF-I-Horizontal method requires less execution time than the SQL-Vertical method, it is faster than the SQL-I-Horizontal method only when the number of dimensions is small (less than 400). As the number of dimensions involved in the calculation rises, the SQL-I-Horizontal method eventually matches and then outperforms the UDF-I-Horizontal method.

3.2 PCA Overview

We are now ready to solve PCA, the classical technique used to reduce dimensionality of a high-dimensional data set with the goal of representing the statistical structure of the data set with fewer variables [15]. This method requires a covariance or a correlation matrix as an input matrix and computes their eigen-decomposition to get the diagonal matrix of eigenvalues and the orthogonal matrix of eigenvectors. These eigenvectors that correspond to the k largest eigenvalues are the principal components of the data set. For our experiments, PCA is derived from the correlation matrix. The eigen-problem to solve PCA can be stated as:

$$XX^T = US^2U^T \tag{2}$$

$$X^TX = VS^2V^T \tag{3}$$

To solve the eigen-problem of equation (2), XX^T, the large $d \times d$ correlation matrix, is needed to obtain the left eigenvector matrix U. On the other hand, the right eigenvector matrix V can be obtained by solving the eigen decomposition of the smaller $n \times n$ correlation matrix, X^TX. PCA on a $d \times d$ matrix is slow when the number of dimensions d is larger than the number of records n. We overcome such limitation by solving PCA using SVD on an $n \times n$ matrix.

3.3 Solving PCA with SVD

For microarray data sets, PCA is typically solved using SVD [8], which can deal with ill-conditioned matrices. A matrix becomes ill-conditioned when the number of dimensions is larger than the number of records. For an iterative method, small errors in the dimensions can result in much larger errors in the model. The solution of SVD for X produces two orthonormal bases, one defined by the right singular vectors, V, and a second one given by left singular vectors, U. The right singular vectors, V, spans d-space of the dimensions and the left singular vectors U span n-space. S is a diagonal matrix whose diagonal elements are eigenvalues. If the problem is to understand linear relationships among dimensions, then the principal component matrix or loading matrix is the left singular matrix, U. The matrix SV^T or U^TX contains the principal component scores, which are the transformed coordinates of input records in the space of principal components. The loading of records is given by the column vectors of V. The matrix US or XV contains the principal component scores, which are the coordinates of the dimensions in the space of principal components. The equation for SVD of X is as follows.

$$X = USV^T \tag{4}$$

For a square, symmetric matrix, SVD is equivalent to the eigen-decomposition of a square matrix. The problem can be solved from either XX^T or X^TX (See Equation (5) and (6)).

$$XX^T = USV^TVSU^T = US^2U^T \tag{5}$$

$$X^TX = VSU^TUSV^T = VS^2V^T \tag{6}$$

For a data set where d is larger than the number of records n, computing SVD from an $n \times n$ matrix, X^TX, is more efficient than from a $d \times d$ matrix, XX^T. This method first calculates the eigen-decomposition of X^TX (equation 7) to produce V^T and S, and then to calculate $d \times n$ matrix U (equation 8). This approach alleviates the array size limit problem in UDFs because we avoid using the large $d \times d$ matrix, XX^T. As a result, this mathematically optimized method dramatically reduces execution time and required memory compared with the classical PCA method. Note that Equations 5 and 6 of SVD are equivalent to the Equation 2 and 3 in the classical PCA method. In our experiments, the eigen-decomposition method is performed by using Householder tridiagonalization and QR factorization.

$$X^TX = VS^2V^T \tag{7}$$

$$U = XVS^{-1} \tag{8}$$

3.4 SVD Method

The SVD method has considerable advantages in terms of execution speed and memory space over traditional PCA methods for high d data with very few records. The method exploited in our system uses Householder tridiagonalization followed by QR factorization [3, 7]. The main algorithm is as follows:

1. Start by centering the data at the global mean and computing a covariance matrix or a correlation matrix, $A : A = X^TX$, where X is the centered data.
2. Apply Householder method by reducing the covariance or correlation matrix to a tridiagonal matrix, B_0, with an orthogonal matrix
   $P_1 : B_0 = A_{n-2} = P_{n-2}A_{n-3}P_{n-2}^T$
3. Find the eigen-decomposition of $B_0 : B_0 = Q_sB_sQ_s^T$ by applying the QR factorization method, where B_s is the diagonal matrix of eigenvalues and Q is the orthogonal matrix.
4. Combine both decompositions to get
   $X^TX = (PQ)B_n(QP)^T$. As a result, columns of $V = PQ$ are the eigenvectors X^TX.

The details of mathematics computation and SVD programming are explained later in Section 3.5. We will first focus on the basic matrix operations used for solving problems of calculating SVD by using SQL and UDFs.

3.5 SVD Implementation

The main algorithm of SVD is explained in section 3.4. In this section we explain the techniques for matrix multiplication and matrix transposition, since those are the fundamental operations used in our SVD implementation. Later, the detailed implementation is explained in mathematical terms, and then we will explain the SQL and scalar UDF implementation of each step. Finally, we present the method using SQL and table-valued user-defined function (TVF).

Transposing Matrix for Multiplication

Matrix transposition is needed for SVD calculation, particularly to perform matrix multiplication, in which matrix transposition is applied to conveniently pair a row from one matrix to a column of another matrix. To transform rows of a matrix to columns, we unpivot a table into a long and slim vertical table indexed by a row and column, and then pivot the unpivoted table to a transposed table where its rows are transformed into columns.

Matrix Multiplication

The most frequently used matrix operation in SVD is matrix multiplication. Several approaches can be exploited to implement matrix multiplication with SQL queries and UDFs. Suppose we want to multiply matrices A and B, then one approach is to transpose the table A before calculating the sum of products by using an aggregation. Another approach is to transpose the table B instead of A, and then sum the products of elements of each row in matrix A and B. Our method of choice is the second approach since the first method uses aggregate sum, which are considerably slower than row summation. We also explored another method to performing matrix multiplication where the result table can be transposed without any extra effort. For example, when computing PAP, we first compute AP, and then transpose AP to compute PAP. The matrix transposition of AP can be performed by altering the pivoting command in the multiplication process by interchanging rows to columns. This allows the matrices to be multiplied and transposed in a single step.

Mathematical Computation of SVD

1. The system first computes a zero-mean data set and then compute a correlation matrix, A. In here, we use a correlation matrix instead of a covariance matrix since correlation matrix scales dimensions better.

2. The $n \times n$ correlation matrix, A, is used to calculate Householder tridiagonalization to get B_0. The process of reducing A to a symmetric tridiagonal form costs $O(n^3)$. The process takes $n-2$ iterations.

3. We then compute $P_r = I - 2v_r v_r^T$.

4. Then an iterative step is executed to obtain a tridiagonal matrix $B_0 = A_{n-2}$ such that $A_r = P_r A_{r-1} P_r$, where $r = 1, ..., n-2$.

   After $n-2$ successive steps, matrix A_i eventually becomes a tridiagonal matrix. The matrix A_{n-2} is renamed as B_0 and it will be used for QR factorization in the next step.

5. matrix $T : T = P_1 P_2 ... P_{n-2}$ is computed to multiply with matrices Q in the final step to obtain eigenvector matrix, V.

   $B_0 = P_{n-2} A_{n-3} P_{n-2}$

   $B_0 = P_{n-2} P_{n-3} ... P_1 A_0 P_1 ... P_{n-3} P_{n-2}$

   Let $T = P_1 P_2 ... P_{n-2}$

   $B_0 = TAT = T^{-1}AT$

6. We are now ready to compute the stepwise QR factorization method. In the first step, B_0 is factorized as $Q_0 R_0$: $B_0 = Q_0 R_0$, where Q_0 is orthogonal and R_0 is upper triangular. Then B_1 is obtained by computing $B_1 = R_0 Q_0$. In the second step, B_1 is factorized as $Q_1 R_1$. Then again B_2 is obtained by computing $B_2 = R_1 Q_1$. In general, the stepwise computation continues as $B_S = Q_S R_S$ and then at each iteration the algorithm computes $B_{S+1} = R_S Q_S$ until convergence.

   The algorithm starts by multiplying B_0 from the left by a Givens [6] rotation matrix C_j, where $j = 2, ..., n$, n-1 times. To obtain R_0 we factorize and multiply by B_0 as follows.

   $$C_n C_{n-1} ... C_3 C_2 B_0 = R_0 \tag{9}$$

   The Givens rotation matrix, C_j, contains the following submatrix in rows $j-2$ and $j-1$ and columns $j-2$ and $j-1$.

   $$\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

7. Then R_0 can be obtained from $R_0 = C_n ... C_3 C_2 B_0$.

8. The program then continues calculating $B_1 = R_0 Q_0$. Similarly in the second step, the program factorizes $B_1 = Q_1 R_1$ obtaining Q_1 and R_1 followed by getting $B_2 = R_1 Q_1$. The program continues its execution in the same way for the next k steps until convergence. Finally, the program produces B_k, a diagonal matrix with eigenvalues in its diagonal. The right eigenvectors, V, can be easily computed from the product of $Q_k Q_{k-1} ... Q_0$ and the matrix T obtained from the Householder method. Subsequently, the matrix with left eigenvectors, U, can be obtained from $U = XVB_k^{-1}$, where $B_k^{-1} = Diag(\frac{1}{\lambda_1}, \frac{1}{\lambda_2} ... \frac{1}{\lambda_n})$, λ_i denotes an eigenvalue of B_k.

SQL and scalar UDF Implementation

1. The detailed implementation of the zero-mean data and correlation analysis was explained in section 3.1.

2. Compute unit vector V_r

   To calculate v_r, we first transform A to an unpivoted form, $AUnpvt(rowID, colID, val)$, then we compute the required variables, $@S_r$.

   The alternative method to compute v_r is to pass all the elements in the rth column of A matrix as parameters to a UDF, which will compute and then return v_r as a result.

3. Compute P_r: $P_r = I - 2v_r v_r^T$, $r = 1, ..., n-2$

   Matrix P can be computed row by row with $n-2$ queries and n entries of the matrix per query.

4. Compute $A_i = P_i A_{i-1} P_i$, where $i = 1, .., n-2$

   To efficiently compute the iterative step, our algorithm performs query optimization by multiplying and transposing $A_{i-1}P_i$ in a single step as explained earlier in Section 3.5. The matrix, $A_{i-1}P_i$ is transposed to multiply to the right of P_i obtaining a matrix, A_i. The queries are performed iteratively to obtain A_n, which will be called B_0, and used as an input matrix for the QR factorization method.

5. Compute $T = P_1 P_2 ... P_{n-2}$

   $T = P_1 P_2 ... P_{n-2}$ can be computed by a single step matrix multiplication and transposition in for every iteration, except for the last iteration where the transposition is not needed.

6. Compute $Q = C_2^T C_3^T ... C_n^T$

   To optimize the computation of Q, matrix multiplication can be avoided since we only need to update two columns for each iteration. In the first iteration, a base matrix Q: Q = C_2 is created. Then the Q matrix is updated with a stored procedure.

7. Compute $R_0 = (C_n ... C_3 C_2) B_0$.

   Then R_0 can be obtained from $R_0 = (C_n ... C_3 C_2) B_0$. Similarly, this calculation does not need a matrix multiplication of the whole matrix. To obtain R_0 we only need to update two rows. In the jth iteration, the jth and the $(j+1)th$ rows are deleted and two updated rows are inserted with the same subscripts.

8. Compute the eigenvalues, right singular vectors and left singular vectors using matrix multiplication and matrix transposition, as explained in Section 3.5.

TVF Implementation

A single microarray experiment can generate thousands of measurements. Existing microarray data sets typically consist of less than one hundred records, but in the future they can consist of a few hundreds. Since a $d \times n$ microarray data set X, has a small number of records (n), the covariance matrix and correlation matrix on the data set are small enough to allow the implementation of SVD using UDFs as available in a modern DBMS. This method requires a table-valued function (TVF), which is a User-Defined Function (UDF) that returns a table. The table returned by the UDF can be referenced in the FROM clause of a query. In steps 1-7 of SVD computation, the method is implemented inside the TVF, which then returns the diagonal matrix S, containing the eigenvalues and the right singular matrix V that can be obtained and stored in the database. Thus, the large $d \times n$ left singular matrix U ($U = XVS^{-1}$) and data set scoring can be calculated using SQL queries.

4. EXPERIMENTAL EVALUATION

Experiments on SVD analysis were conducted on SQL Server 2005. The server had an Intel Dual Core CPU, 2.6 GHz, 4 GB of memory, and 160 GB on disk. The DBMS ran under the Windows XP operating system. For comparison purposes we also used R, a popular statistical package. We conducted experiments with the R statistical software running on the same machine as SQL Server.

Table 2: Execution time of different correlation analysis methods (* denotes "Out of memory".) (time in seconds).

d	SQL-V	SQL-I	UDF-I	SQL-L	UDF-L
100	10	8	2	12	43
200	20	10	4	67	170
400	61	17	17	*	725
600	130	29	39	*	1760
800	224	49	68	*	*
1,000	346	71	105	*	*
2,000	2287	240	417	*	*
3,000	5580	521	942	*	*
4,000	10256	899	1678	*	*
5,000	16687	1363	2641	*	*

4.1 Data Sets

We used real and synthetic data sets to evaluate our system. The microarray datasets were obtained from *GEO Gene Expression Omnibus*, a well-known gene expression and gene molecular repository. The dataset has 176 samples of $36,864$ genes. Samples are from kidney tissue with renal clear cell carcinoma (malignant tumor) and non cancer tissue (benign tissue). For our correlation analysis, the number of genes was reduced to $d = 100 - 5,000$ genes. We also generated synthetic data sets varying n (50, 100, 200 and 300 records) and d (15,000, 30,000, and 60,000 dimensions) to study performance optimizations and to test scalability. The values were randomized between 0 and 1.

4.2 Correlation Analysis Optimizations

In order to compute a correlation matrix of ten thousands by ten thousands dimensions, optimizations are essential (see Table 2). The data set is initially formatted vertically, but by pivoting the table, the I-Horizontal table can be obtained. The system performance is greatly improved by pivoting the table because the I-Horizontal table has fewer rows than the vertical table, and is faster to execute the self-join operations. As another optimization, the UDF-I-Horizontal method was created to optimize the SQL-I-Horizontal. The UDF performs faster than the regular SQL implementation with small sets; however, the SQL implementation outperforms the UDF when the number of dimensions exceeds 400. The least efficient method for high-dimensional data is the SQL-L-Horizontal which involves joining all possible pairs of small input tables. Similarly, the UDF-L-Horizontal requires more execution time compared with the SQL-L-Horizontal method.

4.3 SVD Implementation Alternatives

The SVD computation for microarray data analysis, is usually performed outside the DBMS using statistical packages. For performance comparison with our implementation inside databases, we conducted experiments on the R statistical software. Data sets are exported from DBMS to a tab delimited file before importing them into R for analysis.

The classical PCA method reduces dimensionality by ap-

Table 3: Execution time to calculate classical PCA of $d \times d$ correlation matrix using TVF (* denotes "Out of memory".) (time in seconds).

d	PCA with TVF
1K	68
2K	590
3K	2273
4K	*

Table 4: Execution time of SVD using R, TVF, and SQL and scalar UDF (* denotes "Out of memory".) (time in seconds).

		Outside DB	Inside DB	
d	n	Using R	TVF	SQL and scalar UDF
30K	50	20	3	176
30K	100	33	8	783
30K	200	57	26	4176
30K	300	172	54	12212
60K	50	33	6	188
60K	100	119	14	829
60K	200	231	48	4398
60K	300	*	103	12733

Table 5: Execution time of SVD using R (* denotes "Out of memory", time in seconds).

d	n	Export	R Import	X^TX+SVD	TOTAL
30k	50	2	14	4	20
30k	100	4	21	8	33
30k	200	8	36	13	57
30k	300	13	137	22	172
60k	50	4	20	9	33
60k	100	10	98	11	119
60k	200	15	194	22	231
60k	300	28	304	*	*

Table 6: Execution time of SVD using Table-Valued Function (time in seconds).

d	n	X^TX	SVD	TOTAL
30k	50	2	1	3
30k	100	7	1	8
30k	200	24	2	26
30k	300	50	4	54
60k	50	5	1	6
60k	100	13	1	14
60k	200	46	2	48
60k	300	99	4	103

Table 7: Execution time of SVD using SQL, and aggregate and scalar UDFs (time in seconds).

d	n	X^TX	HH	QR	TOTAL
30k	50	13	18	145	176
30k	100	50	108	625	783
30k	200	248	1288	2640	4176
30k	300	514	5298	6400	12212
60k	50	25	18	145	188
60k	100	97	108	624	829
60k	200	468	1289	2641	4398
60k	300	1012	5301	6420	12733

plying eigen-decomposition of the $d \times d$ correlation matrix among dimensions(see Table 3). If the data set is high dimensional, PCA computation can be very expensive. PCA computation using TVF can compute data sets up to only 3,000 dimensions and failed to compute a data set with 4,000 dimensions due to insufficient memory. Table 4 compares performance of three methods that include a method using SQL commands and scalar UDFs, a method using table valued function, and a method using R statistical packages. For the approach that uses R packages (see Table 5), we assume that the data is originally in the database. Thus, the time to export microarray data sets out of a DBMS and the time to import the data set into R are included in the time to calculate SVD. The method using Table Valued Function (see Table 6) outperforms the other two methods in all cases. When the number of dimensions reaches 60,000 and the number of record reaches 300, the method using R produces an out of memory error, which means it cannot allocate very large vectors. Such error does not occur in the other two methods. Even though the TVF method is the fastest, it is less portable than the method using SQL queries and scalar UDF (see Table 7). In real-life applications, microarray data sets are very small due to the cost to produce them. In fact, statisticians almost never analyze a data set where the number of records exceeds $n = 100$. However, that is not to say the number of records cannot be in the hundreds, but one will find it difficult to find a data set larger than $n = 300$. Therefore, the three methods are compared varying the number of records between 50 to 300. Figure 3 shows that computing SVD in R does not scale well when the number of records, n, grows.

4.4 Time Complexity

The Householder method reduces $A : A = X^TX$ to a symmetric tridiagonal form from T costing $\frac{8}{3}n^3 + O(n^2)$. In addition, to find all the eigenvectors and eigenvalues of the tridiagonal matrix costs $O(6n^3 + n^2)$. The total cost of the Householder tridiagonalization and QR factorization is $8\frac{2}{3}n^3 + O(n^2)$. Figure 2 shows that the running time grows linearly when varying the number of dimensions. The number of dimensions d only impacts the time to obtain the zero-mean data and to create the initial correlation matrix. After we obtain a $n \times n$ correlation matrix, d no longer has any effect on the Householder tridiagonalization or the QR factorization calculation. The total time to compute SVD in all three methods is quadratic. Therefore the running time in axis Y is represented in logarithmic scale. However, in a real-life application the number of records rarely reaches hundreds. The small number of records, n, also gives us

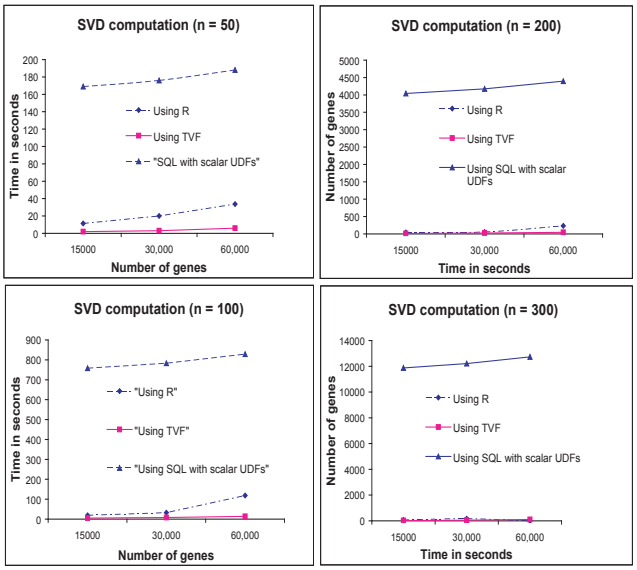

Figure 2: Running time of SVD computation varying the number of dimensions (d).

the opportunity to exploit the $O(n^3)$ computation of the Householder tridiagonalization and the QR factorization in TVF. Then SQL queries are used for the computations that are impacted by the high number of dimensions, such as the computation to obtain the $d \times n$ left eigenvector and U, the scores of PCA.

5. RELATED WORK

PCA [6, 8] and correlation analysis [11] are two common techniques to analyze microarray data. The correlation matrix is also used as an input matrix for PCA. There is a wide range of applications based on PCA, such as, clustering[10], significant genes selection [1], and gene shaving [5, 4]. SVD has been combined with clustering [13], and also with support vector machines and least-squares regression [17]. Microarray data have not been gained enough attention from the database community. Most researchers put microarray data in flat files without a database system to keep the data in more organized manner. There are less tools available inside the DBMS for the statistical analysis than outside.

There exists work on computing statistical models inside a DBMS [15, 14]; this paper explains how to compute sufficient statistics n, L, Q in order to to enable fast computation of several linear statistical models. The L-Horizontal method for the correlation analysis extended the idea from that paper and was adapted for the high dimensional microarray data. Our research shows that although the L-Horizontal method performs the best when the number of dimensions is low, our novel I-Horizontal method performs best for high dimensional microarray data sets. Compared to previous research, our paper focuses on the application involving PCA using SVD for microarray data, which is much different from the classical PCA. We delved into the details of SVD implementation in both SQL and UDFs including the detailed implementation of Householder tridiagonalization and QR factorization. Furthermore, the system performance is compared with a statistical tool outside the DBMS (using the R statistical package). Several SQL and UDF optimizations were also exploited and discussed. SVD has been used in database research, such as Collaborative Filtering (CF) [16] and information retrieval[2]. There are some works in microarray database involving visualization using PCA [12]. The microarray database is also being studied extensively in text mining[9]. To the best of our knowledge, analyzing microarray data sets implementing numerical methods with SQL and UDFs is an unexplored topic.

6. CONCLUSIONS

We presented a system that can analyze microarray data sets inside a DBMS exploiting optimized SQL queries and UDFs. The problem is challenging given the high dimensionality, but small size, of microarray data sets. We explained how to efficiently compute the correlation matrix and dimensionality reduction models with Principal Component Analysis (PCA). Unfortunately, traditional PCA methods cannot be applied with microarray data sets because the eigen-value problem is ill-conditioned due to having more dimensions than rows in the data set. This limitation motivated adapting numerical methods to solve PCA through the Singular Value Decomposition (SVD) based on a Householder tridiagonalization followed by a QR factorization. Our SVD solution is based on a combination of optimized SQL queries and User-Defined Functions, tailored to exploit the DBMS performance. We experimentally compared our system with a state-of-de-art statistical package. We showed our system outperformed the statistical package including export times from the database and was competitive based purely on processing time. We showed our SQL-based system was capable of analyzing data sets at the highest dimensionality, whereas the statistical package failed to do so. Our implementation scales well with the high dimensionality found in microarray data sets and has cubic complexity on data set size, which is a minor limitation since dimensionality is much higher than data set size.

There are several important and interesting directions for future work. Our work opens the possibility of performing advanced statistical analysis of high-dimensional data sets inside the DBMS using other similar matrix-based numerical methods. We would like to explore other statistical methods which may benefit from a combination of SQL queries and UDFs. We would like to combine dimensionality reduction methods with methods to fit a mixture of distributions (i.e. clustering) on high dimensional data sets. Given the high dimensionality of text data sets our proposal has promise to be used in information retrieval techniques integrated into a database system.

7. REFERENCES

[1] J.A. Berger, S. Hautaniemi, S.K. Mitra, and J. Astola. Jointly analyzing gene expression and copy number data in breast cancer using data reduction models. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 3(1):2, 2006.

[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.

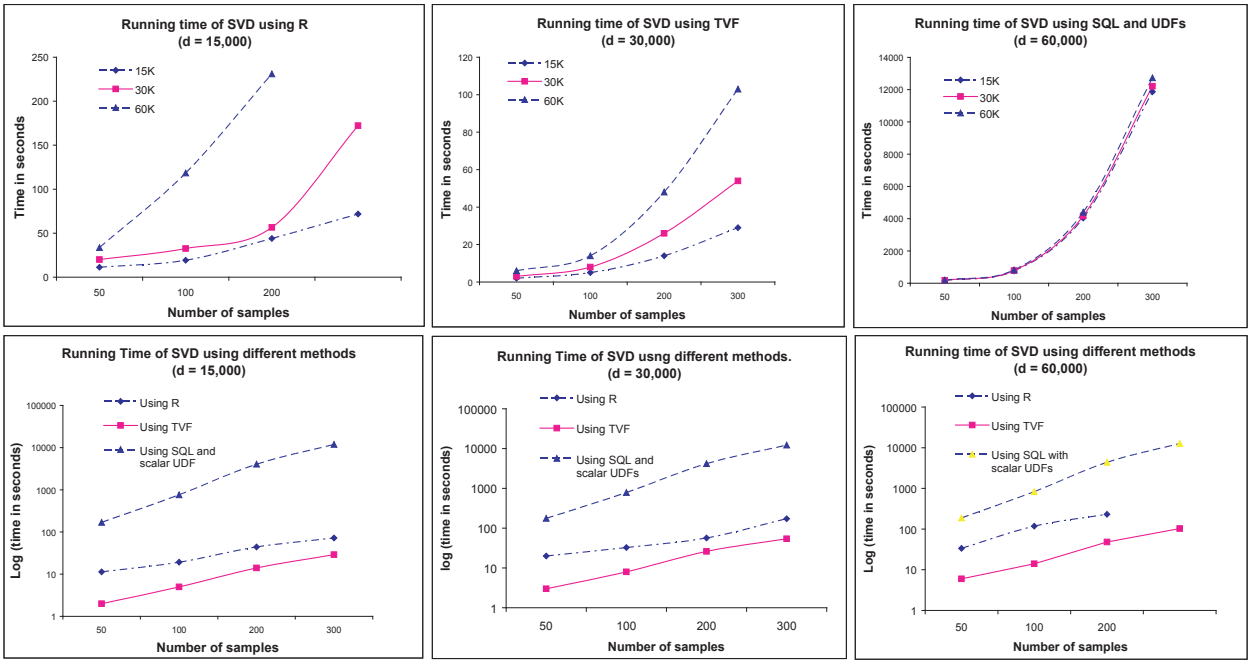

Figure 3: Running time of SVD computation varying the number of records (n).

[3] J.W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[4] K.A. Do, G.J. McLachlan, R. Bean, and S. Wen. Application of gene shaving and mixture models to cluster microarray gene expression data. *Cancer Informatics*, 2:25–43, 2006.

[5] T. Hastie, R. Tibshirani, A. Eisen, R. Levy, L. Staudt, D. Chan, and P. Brown. Gene shaving as a method for identifying distinct sets of genes with similar expression patterns. *Genome Biology 2000*, 1, 2000.

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

[7] E. Kreyszig. *Advanced Engineering Mathematics*. John Wiley and Sons, New York, 1988.

[8] L. Liu, D.M. Hawkins, S. Ghosh, and S.S. Young. Robust singular value decomposition analysis of microarray data. In *Proceedings of the National Academy of Sciences of the United States of America*, pages 13167–13172, 2003.

[9] Y. Liu, S.B. Navathe, J. Civera, V. Dasigi, A. Ram, B.J. Ciliax, and R. Dingledine. Text mining biomedical literature for discovering gene-to-gene relationships: A comparative study of algorithms. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 2(1):62–76, 2005.

[10] Geoffrey J. McLachlan, Kim-Anh Do, and Christophe Ambroise. *Analyzing Microarray Gene Expression Data*. John Wiley and Sons, New Jersey, 2004.

[11] Kuan ming Lin and Jaewoo Kang. Exploiting inter-gene information for microarray data integration. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 123–127, 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] See-Kiong Ng, Zexuan Zhu, and Yew-Soon Ong. Whole-genome functional classification of genes by latent semantic analysis on microarray data. In *APBC '04: Proceedings of the second conference on Asia-Pacific bioinformatics*, pages 123–129, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

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

[15] C. Ordonez and J. García-García. Vector and matrix operations programmed with UDFs in a relational DBMS. In *ACM CIKM Conference*, pages 503–512, 2006.

[16] 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.

[17] L. Shen and E.C. Tan. Dimension reduction-based penalized logistic regression for cancer classification using microarray data. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 2(2):166–175, 2005.