SQLEM: Fast Clustering in SQL using the EM Algorithm

Carlos Ordonez
Georgia Institute of Technology
Atlanta, GA, USA

Paul Cereghini
Teradata, NCR
Rancho Bernardo, CA, USA

ABSTRACT

Clustering is one of the most important tasks performed in Data Mining applications. This paper presents an efficient SQL implementation of the EM algorithm to perform clustering in very large databases. Our version can effectively handle high dimensional data, a high number of clusters and more importantly, a very large number of data records. We present three strategies to implement EM in SQL: horizontal, vertical and a hybrid one. We expect this work to be useful for data mining programmers and users who want to cluster large data sets inside a relational DBMS.

1. INTRODUCTION

1.1 Clustering

Clustering data is a well researched topic in Statistics [7, 13]. Unfortunately the proposed statistical algorithms are generally inefficient and do not work well with *large* data sets. Most of the work done on clustering by the database community attempts to make clustering algorithms more efficient in order to handle large data sets. Clustering algorithms can be broadly classified into distance-based and density-based; most of them work only with numerical data. BIRCH [17] is an important precursor in clustering for large databases. BIRCH is a distance-based algorithm based on the CF tree. It is linear in database size and the number of passes over the data is determined by a user-supplied accuracy, but it is sensitive to noisy data and is not designed to handle high dimensionality. CLARANS [14] and DBSCAN [10] are also important clustering algorithms that work on spatial data. CLARANS, using a distance-based approach, uses randomized search and represents clusters by their medioids (most central point). DBSCAN clusters data points in dense regions separated by low density regions. CLIQUE [2] is a density-based clustering algorithm that can discover clusters in subspaces of multidimensional data and which exhibits several advantages with respect to performance, order of data and initialization over other clustering algorithms but is bad handling high dimensional data as it prunes most subspaces and finds only low dimensional embedded clusters. There is recent work on the problem of selecting subsets of dimensions being relevant to all clusters; this problem is called the projected clustering problem and the proposed algorithm is called PROCLUS [1]. This approach is specially useful to analyze sparse high dimensional data focusing on a *few* dimensions. An outstanding recent clustering algorithm is OptiGrid [11]. In this paper the authors develop a new technique that succesively partitions dimensions by hyperplanes in an optimal manner to discover dense regions. This algorithm effectively handles noisy data of high dimensionality and has a very good performance.

1.2 The EM clustering algorithm

EM is a well established clustering algorithm in the Statistics community. It was first introduced in the seminal paper [6] and there has been extensive work in Machine Learning and Computer Vision to apply it and extend it [4, 12, 15, 16]. EM is a distance-based algorithm that assumes the data set can be modeled as a linear combination of multivariate normal distributions and the algorithm finds the distribution parameters that maximize a model quality measure, called loglikelihood. EM was our choice to cluster data for the following reasons among others. It has a strong statistical basis, it is linear in database size, it is robust to noisy data, it can accept the desired number of clusters as input, it provides a cluster membership probability per point, it can handle high dimensionality and it converges fast given a good initialization. We must stress that EM does have some disadvantages. Sometimes the normal distribution assumption does not hold for some dimensions, given a poor initialization convergence can be slow, the algorithm may stop at a suboptimal solution, undefined computations may appear and interpreting results can be hard. All these issues will be addressed throughout the paper.

1.3 Motivation to implement EM in SQL

There are several reasons for which implementing EM in SQL turned out to be an interesting problem. One of the main points is that this task looked trivial at first sight, but it turned out to be challenging when we faced the problem of handling large data sets with high dimensionality. Here we list some of the reasons that motivated us to do this work:

- Most of the Database research papers on clustering concentrate on developing efficient algorithms in some high-level programming language, such as C++, but very few concentrate on the problem of actually deploying the solution inside a relational DBMS. Besides, many of those clustering algorithms lack a strong statistical foundation.
- Transferring data out of a large data warehouse for processing can be time consuming, error-prone and difficult. In our case transferring tables having more than

100 million records to a workstation to perform clustering uncovered many unexpected problems. The two alternatives to storing the data in a workstation were storing the data in a large text file, or in a database. First, having a big text file containing those 100 million records became a problem because of access speed (just sequential access) and susceptibility to read errors. So text files had to be limited in size in order to be used without errors. Second, we considered the problem of putting the data inside a local DBMS but that turned out to be a problem similar to just trying to cluster the data inside the data warehouse itself.

- SQL is a high-level data manipulation language available in most important DBMS's. SQL is well understood and standarized. Moreover, using SQL can save a great deal of programming.
- Data management is easier if one relies on the DBMS. A clustering program should be freed from managing data.
- In our case most of the times the access plan generated by the query optimizer was inefficient. Ideally, the EM implementation should run as fast as possible regardless of the order of the input data, the access plan produced by the optimizer, the dimensionality of the data and the presence of noise. In other words, the algorithm should have a guaranteed performance.

Having identified all these issues we decided to implement EM in SQL thinking it would be an easy task. However, we soon faced the following difficulties. SQL is based on the relational model and can execute relational operators but it does not support linear algebra operations. Inside a relational DBMS all the data is stored in tables; therefore there are no arrays. For the same reason there is no direct and efficient way to perform linear algebra operations which are required by the EM algorithm.

Any relational DBMS has practical limitations. Just to name a few, there is usually a maximum number of columns per table (as of today in the neighborhood of 1000), the string length of some SQL query cannot be more than a few kilobytes, many optimizations are only available internally. To handle high dimensional data table joins are required, and how the corresponding queries are formulated becomes critical to performance. A joinless solution was not feasible for the following reasons. There is a maximum number of columns per table and we required a number beyond that limit. There is a maximum length for a SQL expression that the SQL interpreter/parser can handle and some mathematical expressions were much longer. Lastly, in our case, the DBMS executes queries in parallel and results are returned in an unpredictable order.

Many times the DBMS becomes slower as record size and block size grow but that may vary depending on the particular DBMS. However, there is a tradeoff with the performance of indexes when these two numbers decrease as we shall see. Some operations have higher overhead than others. In some cases updates are better than inserts but in many other cases inserts are faster. Deletes turned out to be slow. Some database operations tend to be more efficient as the number of columns increases and some others tend to be faster as the number of records increases. The performance of the algorithm can degrade considerably if the SQL statement execution requires many joins and data needs to be sorted.

1.4 Contributions and outline of the paper

Our main contributions are the following:

- Explain how to create a simple SQL code generator to implement the EM clustering algorithm: SQLEM.
- Provide a solution that has good performace. Scale well with database size. Scale well with high dimensional data.
- Keep the basic behavior of the EM algorithm unchanged. This is important to check correctness and debugging.
- Prove that SQL can be used to perform complex mathematical calculations with a reasonable performance when queries are properly formulated.
- Provide a solution that does not require user-defined functions or external data structures (matrices, trees, hash tables). This improves portability and applicability.
- Produce SQL statements that can be easily optimized and executed in parallel by the DBMS.
- Perform most of the work inside the DBMS, having a small program in a workstation to control execution.

This work is organized as follows. Section 2 provides the basic background to understand our implementation. Section 3 describes in detail the three alternatives we came up with to implement EM in SQL and argues which was the best. Section 4 gives a detailed performance evaluation with synthetic and retail data to do customer segmentation. The paper ends with section 5; here we summarize our most important results and indicate directions for future research.

2. STATISTICAL AND DATABASE BACKGROUND

2.1 Mixture of Gaussians

EM assumes the data can be fitted by a linear combination (mixture) of normal (Gaussian) distributions. The probability density function (pdf) for the normal distribution on one variable x [8] is:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp[\frac{-(x-\mu)^2}{2\sigma^2}].$$

This pdf has expected values: $E[X] = \mu, E[(x-\mu)^2] = \sigma^2$. The mean of the distribution is μ and its variance is σ^2. Samples from points having this distribution tend to form a cluster around the mean. The points scatter around the mean is measured by σ^2.

The multivariate normal pdf for p-dimensional space is a generalization of the previous function [8]. The multivariate normal density for a p-dimensional vector $x = x_1, x_2, \ldots, x_p$ is:

$$p(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} exp[-\frac{1}{2}(x-\mu)^t \Sigma^{-1}(x-\mu)],$$

Size	value
k	number of clusters
p	dimensionality
n	number of data points

Figure 1: Matrices sizes

Matrix	size	contents
C	$p \times k$	means (μ)
R	$p \times p$	covariances (Σ)
W	$k \times 1$	weights (w_i)

Figure 2: Gaussian Mixture parameters

where μ is the mean and Σ is the covariance matrix; μ is a p-dimensional vector and Σ is a $p \times p$ matrix. $|\Sigma|$ is the determinant of Σ and the t superscript indicates transposition. The quantity δ^2 is called the squared Mahalanobis distance: $\delta^2 = (x-\mu)^t\Sigma^{-1}(x-\mu)$. This formula will be our basic ingredient to implement EM in SQL.

EM assumes the data is formed by the mixture of k multivariate normal distributions on p variables. The Gaussian (normal) mixture model probability function is given by:

$$p(x) = \sum_{i=1}^{k} w_i p(x|i),$$

where $p(x|i)$ is the normal distribution for each cluster and w_i is the fraction (weight) that cluster i represents from the entire database. It is important to note that we will focus on the case that there are k different clusters each having their corresponding vector μ but all of them having the same covariance matrix Σ. However, it is not hard to extend this work to handle a different Σ for each cluster.

2.2 Outline of the EM algorithm

There are two basic approaches to perform clustering: based on distance and based on density. Distance-based approaches identify those regions in which points are close to each other according to some distance function. On the other hand, density-based clustering finds those regions which are more highly populated than adjacent regions. Clustering algorithms can work in a top-down (hierarchical [13]) or a bottom-up (agglomerative) fashion. Bottom-up algorithms tend to be more accurate but slower.

The Expectation-Maximization (EM) algorithm [16] is an algorithm based on distance computation. It can be seen as a generalization of clustering based on computing a mixture of probability distributions. It works by succesively improving the solution found so far. The algorithm stops when the quality of the current solution becomes stable; this is measured by a monotonically increasing statistical quantity called *loglikelihood* [6]. The goal of the EM algorithm is to estimate the means C, the covariances R and the mixture weights W of the Gaussian mixture probability function described above. The parameters estimated by the EM algorithm are stored in the matrices described in Figure 2 whose sizes are shown in Figure 1. The popular K-means clustering algorithm [16] is a particular case of EM when W and R are fixed: $W = 1/k, R = I$. It is trivial to simplify SQLEM to do clustering based on K-means and therefore we do not describe it.

The EM algorithm starts from an approximation to the solution. This solution can be randomly chosen or it can be set by the user (when there is some idea about potential clusters). A common way to initialize the parameters is to set $C \leftarrow \mu\ random(), R \leftarrow I$ and $W \leftarrow 1/k$; where μ is the global mean. It must be pointed out that this algorithm can get stuck in a locally optimal solution depending on the initial approximation. So one of the disadvantages of EM is that it is sensitive to the initial solution and sometimes it cannot reach the global optimal solution. Nevertheless, EM offers many advantages besides being efficient and having a strong statistical basis. One of those advantages is that EM is robust to noisy data and missing information. In fact, EM was born to handle incomplete data as explained in [6].

2.3 The EM algorithm

The EM algorithm, shown in Figure 3, has two major steps: the *Expectation* step and the *Maximization* step. EM executes the E step and the M step as long as the change in global loglikelihood (called llh inside pseudo-code below) is greater than ϵ or as long as the maximum number of iterations has not been reached. Setting a maximum no. of iterations is important to guarantee performance. The global loglikelihood is computed as $llh = \sum_{i=1}^{n} ln(sump_i)$. The variables δ, P, X are $n \times k$ matrices storing Mahalanobis distances, normal probabilities and responsibilities repectively

```
• Input: $k$, # of clusters. $Y = \{y_1 \ldots y_n\}$ a set of $n$ $p$-dimensional points. $\epsilon$, a tolerance for loglikelihood. $maxiterations$, a maximum number of iterations.
• Output: $C, R, W$, the matrices containing the updated mixture parameters. $X$, a matrix with cluster membership probabilities.
1. Initialize. Set initial values for $C, R, W$ (random or approximate solution from sample)
2. WHILE change in loglikelihood $llh$ is greater than $\epsilon$ and $maxiterations$ has not been reached DO E and M steps
   E step
      $C' = 0, R' = 0, W' = 0, llh = 0$
      for $i = 1$ to $n$
          $sump_i = 0$
          for $j = 1$ to $k$
           $\delta_{ij} = (y_i - C_j)^t R^{-1} (y_i - C_j)$
           $p_{ij} = \frac{w_j}{(2\pi)^{p/2}|R|^{1/2}} exp[-\frac{1}{2}\delta_{ij}]$
           $sump_i = sump_i + p_{ij}$
          endfor
          $x_i = p_i/sump_i, \quad llh = llh + ln(sump_i)$
          $C' = C' + y_i x_i^t, \quad W' = W' + x_i$
      endfor
   M step
      for $j = 1$ to $k$
       $C_j = C'_j/W'_j$
        for $i = 1$ to $n$    $R' = R' + (y_i - C_j) x_{ij} (y_i - C_j)^t$
      endfor
      endfor
      $R = R'/n, W = W'/n$
```

Figure 3: Pseudo code for EM algorithm

for each of the n points.

This is the basic framework of the EM algorithm and this will be the basis to do the translation into SQL. There are several important observations. C', R' and W' are temporary matrices used in computations. Note that they are not the transpose of the corresponding matrix. $||W|| = 1$, that is, $\sum_{i=1}^{k} w_i = 1$. Each column of C is a cluster; C_j is the jth column of C. y_i is the ith data point. R is a diagonal matrix in the context of this paper (statistically meaning that covariances are independent); that is, $R_{ij} = 0$ for $i \neq j$. The diagonality of R is a key assumption to make linear gaussian models work with EM [16]. Therefore, its determinant and its inverse can be computed in time $O(p)$. Note that under these assumptions the EM algorithm has complexity $O(kpn)$. The diagonality of R is a key assumption for the SQL implementation. Having a non-diagonal matrix would change the time complexity to $O(kp^2n)$.

2.4 Simplifying and optimizing computations

The first important substep in the E step is computing the Mahalanobis distances δ_{ij} [6]. Remember that we assume R is diagonal. A careful inspection of the expression reveals that when R is diagonal the Mahalanobis distance of point y to cluster mean C having covariance R is

$$\delta^2 = (y - C)^t R^{-1} (y - C) = \sum_{i=1}^{p} \frac{(y_i - C_i)^2}{R_i}.$$

This is because $R_{ii}^{-1} = 1/R_{ii}$. For a non-singular diagonal matrix R^{-1} is easily computed by taking the mutiplicative inverses of the elements in the diagonal and being R^{-1} diagonal all the products $(y_i - C_i)R_j^{-1} = 0$ when $i \neq j$. A second observation is that R being diagonal can be stored as a vector saving space, but more importantly speeding up computations. So we will index R with just one subscript from now on. Since R does not change during the E step its determinant can be computed only once, making probability computations (p_{ij}) faster. For the M step since R is diagonal the covariance computation gets simplified. Elements off the diagonal in the computation $(y_i - C_j)x_{ij}(y_i - C_j)^t$ become zero. In simpler terms, $R_i = R_i + x_{ij}(y_{ij} - C_{ij})^2$ is faster to compute. The rest of the computations cannot be further optimized mathematically.

2.5 Dealing with null probabilities and null covariances

In practice $p_{ij} = 0$ sometimes, as computed in the E step. This may happen because $exp[-\frac{1}{2}\delta_{ij}] = 0$ when $\delta_{ij} > 600$; that is, when the Mahalanobis distance is big. There is a simple and practical reason for this: the numeric precision available in the computer. In most DBMS's and current computers the maximum accuracy available for numeric computations is double precision which uses 8 bytes. For this precision the $exp(x)$ mathematical function is zero when $x < -1200$.

A big Mahalanobis distance for one point can be the result of noisy data, poor cluster initialization or the point belonging to an outlier. So this problem needed to be solved in order to make SQLEM a practical solution. We must stress that this happens because the computer cannot keep the required accuracy, but not because EM is making a wrong computation. So we needed to have an alternative for δ_{ij} when the distances were big and we solved it like this:

$$p_{ij} = \frac{1/\delta_{ij}}{\Sigma_{l=1}^{k} 1/\delta_{il}}, j \in \{1 \ldots k\}.$$

Note that this computation gives a higher probability to points closer to cluster j and is *never* undefined as long as distances are not zero. Also, if some distance δ_{ij} is zero then $exp(\delta_{ij}) = exp(0)$ is indeed defined (being equal to 1) and thus it can be used without any problem. In our current implementation this alternative computation solved the problem.

In many cases the individual covariance for some dimensions (variables) becomes zero in some clusters or more rarely in all the clusters. This can happen for a number of reasons. Missing information, in general, leaves numerical values equal to zero; clusters involving categorical attributes tend to have the same value on the corresponding column. Remember that the E step computes $p_{ij} = \frac{w_j}{(2\pi)^{p/2}|R|^{1/2}} exp[-0.5\delta_{ij}]$ for $i = 1 \ldots n, j = 1 \ldots k$. As we can see the computation for p_{ij} requires dividing by $\sqrt{|R|}$ and computing R^{-1} for Mahalanobis distances δ_{ij}. Therefore, the problem is really a division by zero which is undefined and computing R^{-1} which is also undefined. But our EM implementation uses only one *global* covariance matrix for all the clusters and then $R = \sum_{i=1}^{k} R^i$, where R^i is the corresponding covariance matrix for cluster i. This can clearly be seen in the M step. In short, *one global covariance matrix R* solves the problem. We have found in practice that as k grows the chance of having $R_i = 0$ is very small, but it may happen. Having only one global covariance matrix R solves the problem in part, but there is a price to pay: we sacrifice cluster description accuracy a bit.

In the event that $\exists i$, s.t. $i \in \{1 \ldots k\}$ and $R_i = 0$ we do the following to compute $|R|$ and R^{-1}. To compute the Mahalanobis distances we skip variables whose covariance is zero and then we avoid dividing by zero ($R_i = 0$). Having a null covariance means all the points have zero distance between them in the corresponding dimensions and then this does not affect δ_{ij}. In other words, we compute R^{-1} for the subspace in which covariances are not zero. To compute $|R|$ we do an analogous thing. Remember that noise independence implies $|R| = \Pi_{i=1}^{p} R_i$ and then we can also skip null covariances. Therefore, $|R| = \Pi_{i=1, R_i \neq 0}^{p} R_i$. But again, there is a price to pay: loglikelihood computation is affected. Skipping null covariances solves the problem of undefined computations but we have observed that loglikelihood decreases sometimes. We believe this is the way to solve the problem but it requires further research.

2.6 Database background

The relational model represents data as relations, each having a primary key and a number of attributes. Each attribute has a simple data type. Arrays, for instance, are not allowed. This gets translated into SQL having tables with a number of columns. A subset of those columns will be the primary key. In general in a relational DBMS the primary key has a corresponding physical index to search data rows efficiently. Relation tuples become data rows in a table. Relational operations such as select, project and join get translated into SQL queries.

SQL is a standarized data manipulation language used in databases. SQL can save a considerable amount of programming and is effective to write high-level queries. However, SQL is neither efficient nor adequate to do linear algebra operations, but we managed to get around that problem by converting matrices to relational tables and using arithmetic operators (+-*/) and functions (exp(x) ln(x)) available in our DBMS. The most important SQL commands we used in our implementation were the following: CREATE TABLE, used to define a table and its corresponding primary index, DROP TABLE, to delete tables, INSERT INTO [table] SELECT, used to add data rows to one table from a select expression, DELETE, used to delete a number of rows from a table and UPDATE, to set columns to different values.

3. SQLEM: THE EM ALGORITHM PROGRAMMED IN SQL

This is the most important part of this work. The reader is referred to the EM description given in the previous section to understand the explanations given. Also, many optimizations and improvements to EM are assumed to be understood from the previous section.

3.1 Overview of alternatives

After analyzing and experimenting we discovered two basic strategies to implement EM in SQL: horizontal and vertical. These two strategies represent two extreme points to implement EM in SQL and there are tradeoffs regarding performance, flexibility and functionality. Experimental evaluation and practical constraints lead to a third hybrid approach.

The first challenge is to compute the k squared Mahalanobis distances for each point to each cluster. The next problem is to compute the k probabilities and k responsibilities. These are computed by evaluating the normal density function with the corresponding distance for each cluster. After responsibilities are computed we just need to update the mixture parameters; this requires computing several relational aggregate functions. Updating C and R requires several matrix products that are expressed as aggregate SQL sums of arithmetic expressions. Updating W requires only doing a SUM on computed responsibilities.

For the three approaches we will present we assume that in general $k \leq p$ (for high-dimensional data) and $p << n$. These assumptions are important for performance. In any case our solution will work well for big n as long as $p \leq 100, k \leq 100$. For the horizontal and vertical approaches we only show the SQL code for the E step. Since the hybrid approach turned out to be the best one it is analyzed in more detail and we show both the E and the M steps in SQL. The SQL statements required to create/drop tables and their indexes, to delete rows, and to transpose C, R are omitted for brevity.

Given a good initialization SQLEM converges fast; as seen before, we either initialize clusters to random values or better to parameters obtained from a sample (usually 5% for large data sets or 10% for medium data sets). Nevertheless, in some cases SQLEM does not converge soon because of noisy data or bad initialization. To avoid making useless computations we limit the maximum number of iterations to some fixed number. For large data sets we have found that 10 iterations is a good number. In some cases we may run SQLEM up to 20 iterations, but for large data sets we never let the program go beyond that limit. Otherwise, convergence can become a bottleneck for performance.

Table	PK	columns	#	Contents
Y	RID	y1,y2...yp	n	data points
YD	RID	d1,d2...dk	n	distances
YP	RID	p1,p2...pk,sump	n	probabilities
YX	RID	x1,x2...xk,llh	n	responsib's
C1..CK	-	y1,y2...yp	1	means
R	-	y1,y2...yp	1	covariances
W	-	w1,w2...wk,llh	1	weights
GMM	-	n,twopipdiv2	1	other
	-	sqrtdetR		parameters

Figure 4: Horizontal approach SQL tables

3.2 Storing data points and mixture parameters in tables

The data points and the Gaussian mixture parameters must be stored in tables. Following the notation we defined before we will add a few more conventions for naming columns in SQL. Column name i will indicate the cluster number, i.e. $i \in \{1 \dots k\}$, column name v will indicate the variable number; that is, $v \in \{1 \dots p\}$. val will be the value of the corresponding column. w_i will indicate the ith cluster weight. RID stands for row id and it is a unique identifier for each data point. Please refer to Tables 4,6 and 8 to understand these naming conventions.

All remaining parameters needed for computations are stored in the table called GMM. This includes all the matrix sizes p, k, n,the constant needed in the density function computation twopipdiv2=$(2\pi)^{p/2}$, the square root of determinant of the covariance matrix sqrtdetR=$\sqrt{|R|}$ and number of iterations. The table YX stores the loglikelihood for each point as well as a *score*, which is the index of the cluster with highest membership probability for that point; in our case *score* is used to classify/segment retail data.

3.3 Horizontal approach

The first way to solve the problem is called the *horizontal* approach. Here we compute the Mahalanobis distances in k terms of a SELECT statement. Each of the k terms is a sum of squared distances divided by the corresponding covariance as seen in the previous section. This is very efficient since all k squared Mahalanobis distances (δ^2) are computed in one table scan but has a major drawback: since there are no arrays in SQL the sum has to be expanded to a long string to sum the p terms. For high dimensional data we found that the parser of the SQL interpreter could not handle such long statements. Even having user-defined functions would not solve the problem because of expression size. For the same reason many computations required by EM have to be broken down into several simpler SQL statements.

The time/space complexity for computing the k Mahalanobis distances for each of the n points is $O(kp)$. This expression size can be a practical problem in almost any relational DBMS. Just to give an example think about computing parameters for 50 clusters for data having 100 dimensions ($k = 50, p = 100$). We need to compute squared differences on p terms, each being in the best case about 10 characters, add them p times, and then put k of those expressions in *only one* SQL statement. All in all, we end up with an expression having approximately $10 \times 50 \times 100 \approx 50,000$ characters. So far, we haven't seen any DBMS handling an expression this long.

```
INSERT INTO YD SELECT
  RID,(Y.y1-C1.y1)**2/R.y1+...+(Y.yp-C1.yp)**2/R.yp,
      (Y.y1-C2.y1)**2/R.y1+...+(Y.yp-C2.yp)**2/R.yp,
      ...
      (Y.y1-Ck.y1)**2/R.y1+...+(Y.yp-Ck.yp)**2/R.yp
  FROM   Y,C1,C2...CK,R;

INSERT INTO YP SELECT
  RID,w1/(twopipdiv2*sqrtdetR)*exp(-0.5*d1) AS p1,
      w2/(twopipdiv2*sqrtdetR)*exp(-0.5*d2) AS p2,
      ...
      wk/(twopipdiv2*sqrtdetR)*exp(-0.5*dk) AS pk,
      p1+p2+...+pk AS sump            FROM  YD,GMM,W;

INSERT INTO YX SELECT
  RID,p1/sump,p2/sump,...,pk/sump,ln(sump)  FROM   YP;
```

Figure 5: Horizontal approach SQL for E step

The tables required to implement this approach in the most efficient way are given in Figure 4. It is important to note that C is stored in k tables to avoid having k different select statements to compute distances. This avoids launching k statements in parallel which would be slower, or doing the k selects sequentially. The code is shown in Figure 5.

To update mixture paramaters C, R, W we proceed as follows. First of all, there is no need to create separate tables C', W', R' as seen on the pseudo code for EM; all temporary results are stored in the corresponding tables C, R, W. To update C we need to execute k select statements (updating C_j) each of them computing the product $y_i x_{ij}$ for $j = 1 \dots k$ and then making a SUM over all n rows to update cluster means from cluster j: table C_j. These k SELECT statements join tables Y and YX by the primary key RID multiplying y_i by x_{ij}. Updating weights in W' is straightforward; first we have to sum the responsibilities and log-likelihood stored in YX and then dividing by n; this is done just by one SELECT statement using the SQL aggregate function SUM. Having computed C' and W' as described in the pseudo code for EM we can update $C_j = C'_j/W'_j$. Now since C is updated we can proceed to compute covariances R by launching k SELECT statements, each computing $R' = R' + (y_i - C_j)x_{ij}(y_i - C_j)^t$ with $j = 1 \dots k$. The last step involves updating R and W. We update covariances and weights: $R = R'/n$ and $W = W'/n$; n is stored in the table GMM.

3.4 Vertical approach

We call the second approach *vertical*. Here the n points are copied into a table having pn rows. And then the Mahalanobis distances are computed using joins. The tables used for this approach in the most efficient way are given in Figure 6. In this case C is stored in one table.

Note that we have to perform separate inserts to compute distances, probabilities and responsibilities because aggregate functions cannot be combined with non-aggregate expressions in the same SQL select statement. YSUMP.sump= $\sum_{i=1}^{k} p_i$ and it is computed using the SUM(column) SQL aggregate function. The code is shown in Figure 7. Note that the first SELECT statement computes distances. Once distances are computed we can obtain probabilities by evaluating the multivariate normal distribution on each distance; this is done in the 2nd SELECT statement shown. Finally, the 3rd SELECT statement shown computes responsibilities x_{ij} by dividing $p_{ij}/sump$ for $j = 1 \dots k$. These responsibilities are the basic ingredient to update mixture parameters C, R, W.

Table	PK	columns	#	Contents
Y	RID,v	value	pn	points
YD	RID,i	d	kn	distances
YP	RID,i	p	kn	probabilities
YX	RID,i	x	kn	responsib's
C	i,v	value	pk	means
R	v	value	p	covariances
W	i	w	k	weights
GMM	-	n,twopipdiv2 sqrtdetR	1	remaining parameters

Figure 6: Vertical approach SQL tables

```
INSERT INTO YD SELECT
   RID,C.i,sum( (Y.val-C.val)**2/R.val ) AS d
FROM      Y,C,R   WHERE  Y.v = C.v AND C.v = R.v
GROUP BY  RID,C.i;

INSERT INTO YP SELECT
   RID,YD.i,w/(twopipdiv2*sqrtdetR)*exp(-0.5*d) AS p
   FROM   YD,W,GMM  WHERE  YD.i = W.i;

INSERT INTO YX SELECT
   RID,C.I,p/YSUMP.sump
FROM   YP,YSUMP  WHERE  YP.RID=YSUMP.RID;
```

Figure 7: Vertical approach SQL for E step

To update mixture paramaters C, R, W we proceed as follows. The first challenge is to compute the product $y_i x_i^t$. Each of the p coordinates for y_i are stored in one row in table Y, and each of the k responsibilities are in a different row in table YX. Therefore, to compute this matrix product $y_i x_i^t$ we need to perform a JOIN between Y and YX only on RID multiplying *value* by x . This JOIN will produce pk rows for each of the n points; the corresponding temporary table YYX will have kpn rows, in general a much bigger number than n. Then to compute C' we need to use the SUM function over all rows of YYX grouping by RID and inserting the aggregated pk rows into table C. To update weights we have to add responsibilities in YX. To that end, we use SUM on x grouping by RID on table YX inserting results into W. With these two summations we can easily compute $C_j = C'_j/W'_j$ (as specified in the pseudo code for EM) by joining tables C and W on column i, dividing *value* by w. Once means C are recomputed we just need to recompute covariances R: we need to JOIN Y and C on v performing a substraction of their corresponding *value* columns, and squaring the difference, storing results on temp table YC. Once these squared differences are computed we perform a JOIN with tables YC and YX on RID, multiplying the squared difference by x and then SUM over all rows. This will effectively recompute R. Finally, we just need to divide both W and R by n (stored in table GMM).

3.5 Solution: a hybrid approach

Here we combine the benefits from both the horizontal and the vertical approaches. The horizontal one is the most efficient since it minimizes I/O, while the vertical is the most flexible but has highest overhead in temporary tables created for joins. For each of the n points this solution computes

Table	PK	columns	#	Contents
Z	RID	y1,y2...yp	n	points
Y	RID,v	value	pn	points
YD	RID	d1,d2...dk	n	distances
YP	RID	p1,p2...pk	n	probabilities
		sump,suminvd	n	
YX	RID	x1,x2...xk,	n	responsibil's
		llh,score		
C	i	y1,y2...yp	k	means
R	-	y1,y2...yp	1	global
				covariances
RK	i	y1,y2...yp	1	covariances/
				cluster
CR	v	C1,C2...Ck,R	p	C^t, R^t
W	i	w1,w2...wk	1	weights
		llh	1	
GMM	-	n,twopipdiv2	1	remaining
		sqrtdetR	1	parameters
X	RID,i	x	kn	responsibil's
				vertically
XMAX	RID	maxx	n	max(x) for/
				n points

Figure 8: Hybrid approach vertical tables

```
UPDATE GMM SET detR=R.y1*R.y2*...*R.yp,
               sqrtdetR=detR**0.5;
INSERT INTO YD SELECT
   RID,sum( (Y.val - CR.C1)**2/CR.R ),
       sum( (Y.val - CR.C2)**2/CR.R ),...
       sum( (Y.val - CR.Ck)**2/CR.R )
  FROM  Y,CR WHERE  Y.v=C.v AND C.v=R.v GROUP BY RID;

INSERT INTO YP SELECT
   RID,
   w1/(twopipdiv2*sqrtdetR)*exp(-0.5*d1) AS p1,
   w2/(twopipdiv2*sqrtdetR)*exp(-0.5*d2) AS p2,...
   wk/(twopipdiv2*sqrtdetR)*exp(-0.5*dk) AS pk,
   p1+p2+..+pk        AS sump,
   1/(d1+1.0E-100)+1/(d2+1.0E-100)+...+1/(dk+1.0E-100)
                            AS suminvd  FROM YD,GMM,W;
INSERT INTO YX SELECT
   RID,
   CASE WHEN sump>0 THEN p1/sump ELSE (1/d1)/suminvd END,
   CASE WHEN sump>0 THEN p2/sump ELSE (1/d2)/suminvd END,
   ...,
   CASE WHEN sump>0 THEN pk/sump ELSE (1/dk)/suminvd END,
   CASE WHEN sump>0 THEN ln(sump) END, 0    FROM  YP;
```

Figure 9: Hybrid approach SQL for E step

the k distances vertically using SQL aggregate functions (SUM(column)) and computes probabilities, responsibilities and mixture parameters horizontally projecting expressions having p or k terms. The tables required for this approach are shown in Figure 8. The SQL code for the E and M steps is shown in Figures 9 and 10 respectively.

Before each insertion (in general INSERT INTO table SELECT exp.) the tables C, R, CR, W, YD, YP, YX are left empty; that is, their respective rows are deleted. These drop/deletion SQL statements are not shown to keep the SQL code understandable and shorter. The C and R matrices are transposed and copied into CR. This is currently done by launching several UPDATE statements in parallel but could be improved if there was a transposition statement available, similar to the SQL extensions to do Knowledge Discovery proposed in [5]. W stores the cluster weights as well as the loglikelihood. The first SQL statement in the E step computes $|R|$. The SQL code shown takes care of null probabilities by using the approximation $\frac{1/d_i}{suminvd}$ described before. Null covariances are handled by inserting a 1 instead of zero in the tables CR and R, but as we mentioned before this has an impact in loglikelihood accuracy and thus needs further research.

For this solution the computation of distances requires a join creating a temporary table with pn rows and k columns. The computation of probabilities and responsibilities requires joins with only n rows at any intermediate step. C and R are updated by inserting rows from k separate SELECT staments. This was necesary for two reasons: first expression size could be a problem again if we updated all parameters in one wide table and for high dimensional data we could easily exceed the maximum number of columns in our DBMS. shwon

Analyzing the query costs incurred by the hybrid approach we have the following. For each iteration the computation of distances requires scanning a table having pn rows. The computation of probabilities and responsibilites each require one scan on n rows. Updating C and R require k table scans on n rows each. Updating W requires only one scan on n rows. All other computation involve only scans on tables having less than p or k rows. Overall one iteration of EM requires $2k + 3$ scans on tables having n rows, and one scan on a table having pn rows. The theoretical minimum of table scans on n rows required by EM is only 4: two for the E step and two for the M step; the E step reads the input points Y and writes the responsibilities X, and the M step reads the input table Y as well as the responsibilities in the table X. This points out to several optimizations that can be made in the algorithm if scans can be synchronized and more computations are done in a single SELECT statement. The problem is that many optimizations happen automatically inside the DBMS and are not available through SQL commands.

Given current practical constraints found in the Teradata DBMS we expect this implementation to be useful for clustering problems having $n \leq 1.0E + 8$, $p \leq 100$, $k \leq 100$ and $pk \leq 1000$. These numbers clearly represent big problem sizes.

3.6 Important optimizations

Even though we used a parallel DBMS having a good query optimizer, several optimizations were needed to make our SQL code effective. The most important optimizations are the following.

Updates are slower than inserts. The reason for this is that updates involve 2 I/Os for each data block and inserts only one. Whenever there are two or more tables in one statement the access plan requires joins, even if some of the tables have only one row. This is particularly troublesome to compute distances because if Y is one table and C is stored in k tables with only one row then joins are performed anyway. To reduce the overhead of joins only one itermediate join produces pn rows: the SELECT statement to compute distances. All the remaining joins always produce intermediate tables having only n rows. Our DBMS uses a fast hash-based join approach and that is why times scale linearly as we shall see in the next section. As we have pointed out before the Teradata DBMS returns rows in a normal query

```
INSERT INTO C SELECT
   1,sum(Z.y1*x1)/sum(x1),sum(Z.y2*x1)/sum(x1),...
     sum(Z.yp*x1)/sum(x1)
FROM  Z,YX   WHERE Z.RID=YX.RID;
...
INSERT INTO C SELECT
   k,sum(Z.y1*xk)/sum(xk),sum(Z.y2*xk)/sum(xk),...
     sum(Z.yp*xk)/sum(xk)
FROM  Z,YX   WHERE Z.RID=YX.RID;

INSERT INTO W SELECT
   sum(x1),sum(x2),...sum(xk),sum(llh) FROM YX;
UPDATE W SET w1=w1/GMM.n,w2=w2/GMM.n,...,wk/GMM.n;

INSERT INTO RK SELECT
   1,sum(x1*(Z.y1-C.y1)**2),...,
     sum(x1*(Z.yp-C.yp)**2)    FROM  Z,C,YX;
...
INSERT INTO RK SELECT
   k,sum(xk*(Z.y1-C.y1)**2),...,
     sum(xk*(Z.yp-C.yp)**2)    FROM  Z,C,YX;
INSERT INTO R  SELECT
  sum(y1/GMM.n),sum(y2/GMM.n,...,
  sum(yp/GMM.n)                FROM RK;
```

Figure 10: Hybrid approach SQL for M step

in an unpredictable order unless the user specifies that rows should be ordered (by primary key); this happens because queries are executed in parallel in several AMPS (processors) and results are assembled together in one processor. However, our solution does not require ordering results in any SELECT statements; this is crucial to keep the time complexity of EM unchanged. We accomplished this by joining tables always using the column RID.

For a big table, that is, a table storing n rows, it is faster to drop and create a table than deleting all the records. This is not true if the table is small (C, R, W) since the overhead to drop/create a table is greater than just deleting a few rows. As we mentioned the DBMS we used executes queries in parallel. One way to speedup the process is to make data block size smaller; in this way there is a finer grain for parallelism and the optimizer can better balance load among processors.

3.7 Practical considerations

EM offers many advantages besides having a strong statistical basis and being efficient. One of those advantages is that EM is robust to noisy data and missing information. In fact, EM was born to handle incomplete data as explained in [6]. SQLEM can be extended to cluster categorical data by converting each categorical value to a binary field. The cluster centroids C will then give the probability or percentage of points in some cluster having a particular categorical value. These findings can be further explained by looking at the covariance matrix R. The drawback is that this extension increases dimensionality.

A covariance matrix having entries equal to zero just affects the determinant $|R|$, R^{-1} and loglikelihood computation as explained before; zero entries are skipped to compute $|R|$ and loglikelihood is rescaled accordingly. Sampling can be used to obtain a good initial approximation to initialize the cluster centroids and the covariance matrix; note that sampling is not good enough to cluster the entire data set as standard error is proportional to the inverse of the square root sample size [9] and then such clustering results obtained from a sample are not reliable.

4. EXPERIMENTAL EVALUATION

We made our experiments on an NCR 4800 parallel computer running Unix MP-RAS (Unix System V). This machine has 2 nodes connected by a high speed interconnect network. Each node has 4 CPU's running at 400Mhz. The relational DBMS we used was NCR Teradata. The SQL code generator was written in the Java language and the connection to the DBMS was done using the JDBC library. We concentrated on benchmarking our hybrid solution.

4.1 Experiments with retail data

We used Market Basket data from a retailer involving sales for one month in several stores. We chose the number of clusters to be $k = 9$ based on customer's requirements. The data had the following characteristics: no categorical data; all numerical variables. Based on business requirements we focused on p=6 variables including: hour of the transaction, total sales per basket, total discount per basket, total cost per basket, distinct product quantity per basket, distinct categories of product per basket. The total number of baskets to analyze was n=1,545,075. SQLEM took around 31 minutes on 5 iterations to converge on a good solution. The EM algorithm uncovered the following as evidenced by the means and covariances found in the data.

This particular retailer had about 71% of its clientele in two clusters, that can be described as customers that would come into the store for an average of 1 to 3 low price products and would not take advantage of any discount promotions. The main discriminator between the two clusters was that one shopped around noon and the other shopped in the late afternoon, maybe after work. In other two clusters about 12% of the customers showed core behavior. These individuals had a tendency to have baskets that were higher in sales and not quite as low in discount. But the overriding characteristic was that they shopped for an average of 9 products from an average of 6 different sections in the store. These two clusters seemed to have shopping time as a main discriminator. Around 10% of the baskets were transacted by customers that seemed interested in lunch, since they shopped around noon time, yet they looked for possibly more than just lunch given the fact that they on average purchased 5 products from 4 different sections in the store. Maybe lunch was the main reason for their visit to the store but they realized that they needed something else while in the store.

A small cluster comprising 3% of the baskets showed the same characteristics as the 10% mentioned above except for the fact that these customers seemed to take advantage of promotions. The rest of the baskets show three distinct clusters. One that has convenience shoppers that like to shop later in the day. And two clusters that exhibited "cherry picking" behavior. High sales, high discounts due to promotions and low number of products per baskets.

With minimal analysis time we could start making some educated assumptions about customer behavior that with further analysis could turn into valuable data to a business. We were able to analyze a significant amount of data in a very short amount of time. As Data Warehousing becomes mainstream the availability of massive volumes of data for analysis will become more common place. We see SQLEM

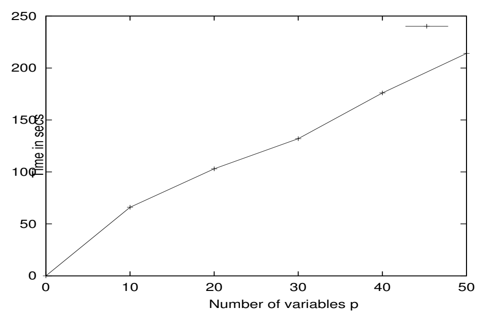

Figure 11: Time per iteration with varying dimensionality p,$k = 20$,$n = 10k$

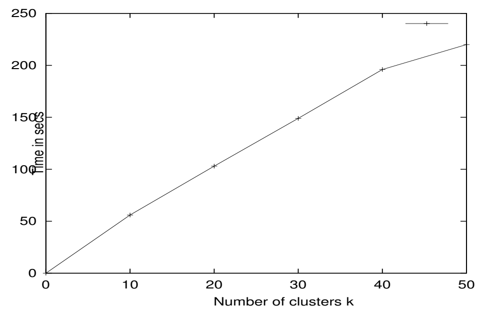

Figure 12: Time per iteration with varying no. of clusters k,$p = 20$,$n = 10k$

as a way to enable mining of large data volumes in Data Warehousing.

4.2 Experiments with synthetic data

We generated data by evaluating a mixture density of k Gaussian distributions on p variables. We varied the number of clusters k , the number of variables p and the number of points n to test scalability. We added 20% of n points as noise. The covariances were kept uniform across clusters.

We concentrated on benchmarking the time per iteration. To cluster big data sets in general we use sampling with about 10% of the data points to obtain several possible ”good” initial solutions. With a good initial approximation EM usually converges in a few iterations (less than 10). In any case at each iteration EM always finds a solution that is guaranteed not to decrease loglikelihood from the previous one. Otherwise the classical version of the EM algorithm may not scale for a database having hundreds of millions of records. However, as our times show it may be feasible to cluster a data set with a few million records starting from a random initialization of mixture parameters.

As can be seen from the graphs 11,12,13 SQLEM scales linearly in the 3 dimensions we analyzed: p, k and n. It should be noted that the graph for p is not quite linear in data sets having lower dimensionality. A similar problem happens with database size n. This is because the execution overhead of the SQL statements is higher on smaller problem sizes. The times per iteration shown are at least as good as the times achieved by SEM [3]. However, a direct comparison is not possible since they compress the data and make most of the work in a workstation memory, whereas we rely heavily on using only efficient SQL code inside a relational DBMS. Another important note is that our times may vary in another relational DBMS if joins are not hash-based. A nested-loop join approach would definetely make our solution slower. Overall the execution time of SQLEM can be bounded by $O(p^2n)$ if p and k are in the same order of magnitude since we do not change the complexity of the algorithm.

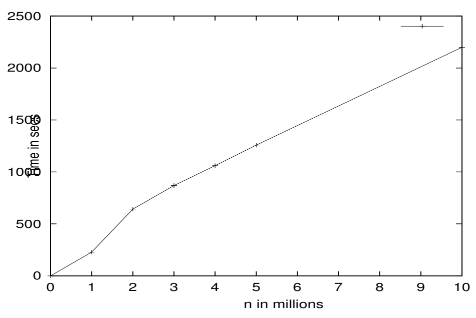

Figure 13: Time per iteration for different database sizes n,$p = 10$,$k = 10$

4.3 Related work

To the best of our knowledge the only other work that analyzes the problem of scaling EM to large data sets is [3]. Their clustering algorithm is called Scalable EM (SEM). However, this work was not inspired by SEM. It was not our intent to improve their solution since the performance results the authors report are very good and their approach is very different from ours. We just needed a clustering algorithm which had a strong statistical basis and EM was the choice. EM may not be the best clustering algorithm currently available but it offers advantages not available in other clustering algorithms published in the literature.

Here we give a brief summary of SEM. SEM can compute clusters in one database scan by updating several mixture models concurrently (usually around 10) and by doing data compression in two phases. Their solution relies on having special data structures to update the mixture parameters in memory. The algorithm makes iterations in memory, avoiding repeated table scans. It is important to note that our solution does not preclude incorporating some of the improvements proposed by SEM, but that requires further research.

5. CONCLUSIONS

We presented an efficient implementation of the EM algorithm in SQL to cluster large data sets inside a relational DBMS. The SQL statements presented are easy to optimize and to execute in parallel. Our implementation is fast and scalable. We used plain ANSI SQL which provides portability and applicability in most relational DBMS's.

We presented three basic appraoches to solve the problem. A first *horizontal* approach was to project big arithmetic expressions and create SELECT statements involving k terms and returning n rows. This approach is very efficient but the Mahalanobis distance computation turned out to be a problem because of expression size: $O(kp)$. At the other extreme we developed a *vertical* approach in which the data points and the mixture parameters are stored in long tables having a compound key and only one data column.

The table storing Y had pn rows. This is the most flexible approach, but also the most inefficient because the join to compute Mahalanobis distance produced a temporary table having kpn rows. The remaining joins produced tables having pn or kn rows slowing down EM even further. From these two approaches we devised a *hybrid* one that computes Mahalanobis distances vertically, but computes responsibilities and mixture parameters horizontally. This turned out to be a still flexible solution without seriously compromising performance. We explored general SQL query optimizations. Inserts are preferred over updates. Dropping/creating a table is preffered over deleting all its data rows for tables having n rows. A table having n rows and k columns is more efficient than a table having kn rows for join processing. The select operation to compute distances requires a join with pn rows in a temporary table. The remaining joins involve only n rows. Smaller block sizes give better granularity for parallel query execution.

Regarding performance. SQLEM scales linearly with dimensionality p, the number of clusters k, and more importantly with the number of data points n. The overall execution time makes clustering feasible for tables containing several millions of data rows and high dimensionality ($pk \leq 1000$). We presented experiments with synthetic and retail data to support our claims. Our clustering algorithm compares reasonably well with other approaches proposed in the literature as shown by our experiments.

In the future we would like to explore further improvements to make SQLEM more efficient and more robust to noisy data. This includes synchronizing operations to decrease table scans, avoiding computations that do not change mixture parameters in consecutive iterations, improving parallel execution and caching for small tables. We also want to apply SQLEM to a large collection of segmented images to continue our previous work on association rules obtained from an image collection [15].

Acknowledgements

The first author was supported in part by grant LM 06726-02 from the National Library of Medicine. The first author would also like to acknowledge the support from Teradata, a division of NCR Corporation, to accomplish this work. We thank the help provided by Scott Cunningham to understand the statistical concepts behind the EM algorithm. We thank Robert MacDonald, Ken Rhames and Roman Diaz for helping us improve the presentation of this paper. We thank the anonymous referees for their helpful comments. The first author would like to specially thank Edward Omiecinski, from Georgia Tech, for his encouragement to work on clustering algorithms.

6. REFERENCES

[1] C. Aggarwal, C. Procopiuc, J. Wolf, P. Yu, and Jong Park. Fast algorithms for projected clustering. In *ACM SIGMOD Conference*, 1999.

[2] R. Agrawal, J. Gehrke, D. Gunopolos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *ACM SIGMOD Conference*, pages 94–105, 1998.

[3] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. ACM KDD Conference*, pages 9–15, 1998.

[4] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Region-based image querying. In *IEEE Workshop on Content-Based Access of Image and Video Libraries*, 1997.

[5] J. Clear, D. Dunn, B. Harvey, M.L. Heytens, and P. Lohman. Non-stop SQL/MX primitives for knowledge discovery. In *ACM KDD Conference*, pages 425–429, 1999.

[6] A.P. Dempster, N.M. Laird, and D. Rubin. Maximum likelihood estimation from incomplete data via the EM algorithm. *Journal of The Royal Statistical Society*, 39(1):1–38, 1977.

[7] R. Dubes and A.K. Jain. *Clustering Methodologies in Exploratory Data Analysis.* Academic Press, New York, 1980.

[8] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* J. Wiley and Sons, New York, 1973.

[9] W. DuMouchel, C. Volinski, T. Johnson, and D. Pregybon. Squashing flat files flatter. In *Proc. ACM KDD Conference*, 1999.

[10] M. Easter, H.P. Kriegel, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *IEEE ICDE Conference*, 1996.

[11] A. Hinneburg and D. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality. In *VLDB Conference*, pages 506–517, 1999.

[12] M. Jordan and R. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, 1994.

[13] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 1983.

[14] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *VLDB Conference*, pages 144–155, 1994.

[15] C. Ordonez and E. Omiecinski. Discovering association rules based on image content. In *IEEE Advances in Digital Libraries Conference (ADL'99)*, pages 38–49, 1999.

[16] S. Roweis and Z. Ghahramani. A unifying review of linear Gaussian models. *Neural Computation*, 11:305–345, 1999.

[17] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proc. ACM SIGMOD Conference*, pages 103–114, 1996.