Towards Expectation-Maximization by SQL in RDBMS

Kangfei Zhao[1], Jeffrey Xu Yu[1], Yu Rong[2], Ming Liao[1], and Junzhou Huang[3]

[1] The Chinese University of Hong Kong, Hong Kong S.A.R.,
{kfzhao, yu, mliao}@se.cuhk.edu.hk
[2] Tencent AI Lab, China, yu.rong@hotmail.com
[3] University of Texas at Arlington, United States, jzhuang@uta.edu

Abstract. Integrating machine learning techniques into *RDBMS*s is an important task since many real applications require modeling (e.g., business intelligence, strategic analysis) as well as querying data in *RDBMS*s. Without integration, it needs to export the data from *RDBMS*s to build a model using specialized *ML* toolkits and frameworks, and import the model trained back to *RDBMS*s for further querying. Such a process is not desirable since it is time-consuming and needs to repeat when data is changed. In this paper, we provide an *SQL* solution that has the potential to support different *ML* models in *RDBMS*s. We study how to support unsupervised probabilistic modeling, that has a wide range of applications in clustering, density estimation, and data summarization, and focus on Expectation-Maximization (EM) algorithms, which is a general technique for finding maximum likelihood estimators. To train a model by EM, it needs to update the model parameters by an E-step and an M-step in a while-loop iteratively until it converges to a level controled by some thresholds or repeats a certain number of iterations. To support EM in *RDBMS*s, we show our solutions to the matrix/vectors representations in *RDBMS*s, the relational algebra operations to support the linear algebra operations required by EM, parameters update by relational algebra, and the support of a while-loop by *SQL* recursion. It is important to note that the *SQL*'99 recursion cannot be used to handle such a while-loop since the M-step is non-monotonic. In addition, with a model trained by an EM algorithm, we further design an automatic in-database model maintenance mechanism to maintain the model when the underlying training data changes. We have conducted experimental studies and will report our findings in this paper.

1 Introduction

Machine learning (*ML*) plays a leading role in predictive and estimation tasks. It needs to fully explore how to build, utilize and manage *ML* models in *RDBMS*s for the following reasons. First, data are stored in a database system. It is time-consuming of exporting the data from the database system and then feeding it into models, as well as importing the prediction and estimation results back to the database system. Second, users need to build a model as to query data in

*RDBMS*s, and query their data by exploiting the analysis result of the models trained as a part of a query seamlessly in *RDBMS*s. A flexible way is needed to train/query an *ML* model together with data querying by a high-level query language (e.g., *SQL*). Third, the data maintained in *RDBMS*s are supposed to change dynamically. The analysis result of the *ML* models trained may be outdated, which requires repeating the process of exporting data from *RDBMS*s followed by importing the model trained into *RDBMS*s. It needs to support *ML* model update automatically in *RDBMS*s.

There are efforts to support *ML* in *RDBMS*s [12,21]. Model-based views [8,13] are proposed to support classification and regression analysis in database systems. In brief, [8,13] use an ad-hoc create view statement to declare a classification view. In this create view statement, [8] specifies the model by an as...fit...bases clause, and the training data is fed by an *SQL* query, while [13] specifies a model explicitly with using svm clause, where the features and labels are fed by feature function and labels, respectively. Here, feature function takes database attributes as the input features, and labels are database attributes. Although these approaches provide an optimized implementation for classification models, their create view statement is lack of generality and deviates from the regular *SQL* syntax. In addition, the models supported are limited and implemented in a low-level form in a database system, which makes it difficult for ordinary database end-users to develop new models swiftly. In this work, we demonstrate that our SQL recursive query can define a model-based view in an explicit fashion to support many *ML* models. Different from [8,13], we focus on unsupervised models in the application of in-database clustering, density estimation, and data summarization.

The main contributions of this work are summarized below. First, we study how to support Expectation-Maximization (EM) algorithms [15] in *RDBMS*s, which is a general technique for finding maximum likelihood estimators. We discuss how to represent data in *RDBMS*s, how to compute E-step and M-step of EM using relational algebra operations, how to update parameters using relational algebra operations, and how to support its while-loop using *SQL* recursive queries. Note that mutual recursion is needed for both E/M-step in EM, where the E-step is to compute the conditional posterior probability by Bayesian inference, which is a monotonic operation, whereas the M-step is to compute and update the parameters of the model given a closed-form updating formula, which can be non-monotonic. This fact suggests that *SQL*'99 recursion cannot be used to support EM, since *SQL*'99 recursion (e.g., recursive with) only supports stratified negation, and therefore cannot support non-monotonic operations. We adopt the enhanced *SQL* recursion (e.g., with+) given in [28] based on *XY*-stratified [4, 26, 27], to handle non-monotonic operations. We have implemented our approach on top of *PostgreSQL*. We show how to train a batch of classical statistical models [5], including Gaussian Mixture model, Bernoulli Mixture model, the mixture of linear regression, Hidden Markov model, Mixtures of Experts. Third, given a model trained by EM, we further design an automatic in-database model maintenance mechanism to maintain the model

Algorithm 1: EM Algorithm for Mixture Gaussian Model

```
1: Initialize the means $\boldsymbol{\mu}$, covariances $\boldsymbol{\sigma}$ and mixing coefficients $\boldsymbol{\pi}$;
2: Compute the initial log-likelihood $L$; $i \leftarrow 0$;
3: while $\Delta L > \epsilon$ or $i <$ maxrecursion do
4:     E-step: compute the responsibilities $p(z_{ik})$ based on current $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ and $\boldsymbol{\pi}$ by Eq. (3);
5:     M-step: re-estimate $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ and $\boldsymbol{\pi}$ by Eq. (4)-(6);
6:     re-compute the log-likelihood $L$; $i \leftarrow i + 1$;
7: end while
8: return $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$, $\boldsymbol{\pi}$;
```

when the underlying training data changes. Inspired by the online and incremental EM algorithms [14,17], we show how to obtain the sufficient statistics of the models to achieve the incremental even decremental model updating, rather than re-training the model by all data. Our setting is different from the incremental EM algorithms which are designed to accelerate the convergence of EM. We maintain the model dynamically by the sufficient statistics of partial original data in addition to the delta part. Fourth, we have conducted experimental studies and will report our findings in this paper.

Organization In Section 2, we introduce the preliminaries including the EM algorithm and the requirements to support it in database systems. We sketch our solution in Section 3 and discuss EM training in Section 4. In Section 5, we design a view update mechanism, which is facilitated by triggers. We conduct extensive experimental studies in Section 6 and conclude the paper in Section 7.

2 Preliminaries

In this paper, we focus on unsupervised probabilistic modeling, which has broad applications in clustering, density estimation and data summarization in database and data mining area. Specifically, the unsupervised models aim to reveal the relationship between the observed data and some latent variables by maximizing the data likelihood. The expectation-maximization (EM) algorithm, first introduced in [7], is a general technique for finding maximum likelihood estimators. It has a solid statistical basis, robust to noisy data and its complexity is linear in data size. Here, we use the Gaussian mixture model [5], a widely used model in data mining, pattern recognition, and machine learning, as an example to illustrate the EM algorithm and our approach throughout this paper.

Suppose we have an observed dataset $X = \{x_1, x_2, \cdots, x_n\}$ of n data points where $x_i \in \mathbb{R}^d$. Given $\mathcal{N}(x|\boldsymbol{\mu}, \boldsymbol{\sigma})$ is the probability density function of a Gaussian distribution with mean $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance $\boldsymbol{\sigma} \in \mathbb{R}^{d \times d}$, the density of Gaussian mixture model is a simple linear super-position of K different Gaussian components in the form of Eq. (1).

$$p(x_i) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k) \tag{1}$$

K	$\boldsymbol{\pi}$	$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$
1	π_1	μ_1	σ_1
2	π_2	μ_2	σ_2

(a) relation GMM

ID	x
1	x_1
2	x_2

(b) relation X

ID	K	p
1	1	$p(z_{11})$
1	2	$p(z_{12})$
2	1	$p(z_{21})$
2	2	$p(z_{22})$

(c) relation R

ID	x
1	[1.0, 2.0]
2	[3.0, 4.0]

(d) row-major representation X_c

Table 1. The relation representations

Here, $\pi_k \in \mathbb{R}$ is the mixing coefficient, i.e., the prior of a data point belonging to component k and satisfies $\sum_{i=1}^{K} \pi_k = 1$. To model this dataset X using a mixture of Gaussians, the objective is to maximize the log-likelihood function in Eq. (2).

$$lnp(\boldsymbol{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \sum_{i=1}^{n} ln[\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k)] \quad (2)$$

Algorithm 1 sketches the EM algorithm for training the Gaussian Mixture Model. First, in line 1-2, the means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\sigma}_k$ and the mixing coefficients $\boldsymbol{\pi}_k$ of K Gaussian distributions are initialized, and the initial value of the log-likelihood (Eq. (2)) is computed. In the while loop of line 3-7, the Expectation-step (E-step) and Maximization-step (M-step) are executed alternatively. In the E-step, we compute the responsibilities, i.e., the conditional probability that x_i belongs to component k, denoted as $p(z_{ik})$ by fixing the parameters based on the Bayes rule in Eq. (3).

$$p(z_{ik}) = \frac{\pi_k \mathcal{N}(x_i|\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j)} \quad (3)$$

In the M-step, we re-estimate a new set of parameters using the current responsibilities by maximizing the log-likelihood (Eq. (2)) as follows.

$$\boldsymbol{\mu}_k^{new} = \frac{1}{n_k} \sum_{i=1}^{n} p(z_{ik}) x_i \quad (4)$$

$$\boldsymbol{\sigma}_k^{new} = \frac{1}{n_k} \sum_{i=1}^{n} p(z_{ik})(x_i - \boldsymbol{\mu}_k^{new})(x_i - \boldsymbol{\mu}_k^{new})^T \quad (5)$$

$$\boldsymbol{\pi}_k^{new} = \frac{n_k}{n} \quad (6)$$

where $n_k = \sum_{i=1}^{n} p(z_{ik})$. At the end of each iteration, the new value of log-likelihood is evaluated and used for checking convergence. The algorithm ends when the log-likelihood converges or a given iteration time is reached. In *RDBMS*, the learned model, namely the parameters of K components, can be persisted in a relation of K rows as shown in Table 1(a). Suppose 1-dimensional dataset X as Table 1(b) is given, the posterior probability of x_i belongs to component k can be computed as Table 1(c) and clustering can be conducted by assigning x_i to component with the maximum $p(z_{ik})$.

To fulfill the EM algorithm in database systems, there are several important issues that need to be concerned, including (1) the representation and storage of high dimensional data in the database, (2) the relation algebra operation used to perform linear algebra computation in EM, (3) the approach for iterative parameter updating, (4) the way to express and control the iteration of EM algorithm, and (5) the mechanism to maintain the existing model when underlying data evolves.

3 Our Solution

In this section, we give a complete solution to deal with above issues in applying the EM algorithm and building model-based views inside *RDBMS*.

High Dimensional Representation. Regarding the issue of high dimensional data, different from [20], we adopt the row-major representation, as shown in Table 1(d), which is endorsed by allowing array/vector data type in the database. With it, we can use the vector/matrix operations to support complicated linear algebra computation in a concise *SQL* query.

Consider computing the means $\boldsymbol{\mu}$ in the M-step (Eq. (4)) with the representation X_c, in Table 1(d). Suppose the responsibilities are in relation $R(ID, K, p)$, where ID, K and p are the identifier of data point, component, and the value of $p(z_{ik})$. The relational algebra expression to compute Eq. (4) is shown in Eq. (7).

$$M_c \leftarrow \rho_{(K,\text{mean})}({}_{K}\mathcal{G}_{\mathsf{sum}(p\cdot x)}(R \underset{R.ID=X_c.ID}{\bowtie} X_c)) \tag{7}$$

It joins X and R on the ID attribute to compute $p(z_{ik}) \cdot x_i$ using the operator $\cdot$ which denotes a scalar-vector multiplication. With the row-major representation which nesting separate dimension attributes into one vector-type attribute, the $\cdot$ operator for vector computation, Eq. (4) is expressed efficiently (Eq. (7)).

Relational Algebra to Linear Algebra. On the basis of array/vector data type and the derived statistical function and linear algebra operations, the complicated linear algebra computation can be expressed by basic relational algebra operations (selection (σ), projection (Π), union ($\cup$), Cartesian product ($\times$), and rename (ρ)), together with group-by & aggregation. Let V and E (E') be the relation representation of vector and matrix, such that $V(ID, v)$ and $E(F, T, e)$. Here ID is the tuple identifier in V. F and T, standing for the two indices of a matrix. [28] introduces two new operations to support the multiplication between a matrix and a vector (Eq. (8)) and between two matrices (Eq. (9)) in their relation representation.

$$E \overset{\oplus(\odot)}{\underset{T=ID}{\bowtie}} V = {}_{F}\mathcal{G}_{\oplus(\odot)}(E \underset{T=ID}{\bowtie} V) \tag{8}$$

$$E \overset{\oplus(\odot)}{\underset{E.T=E'.F}{\bowtie}} E' = {}_{E.F,E'.T}\mathcal{G}_{\oplus(\odot)}(E \underset{E.T=E'.F}{\bowtie} E') \tag{9}$$

The matrix-vector multiplication (Eq. (8)) consists of two steps. The first step is computing $v \odot e$ between a tuple in E and a tuple in V under the join condition $E.T = V.ID$. The second step is aggregating all the $\odot$ results by the operation

```
with R as
    select ··· from R_{1,j}, ··· computed by ··· (Q_1)
    union by update
    select ··· from R_{2,j}, ··· computed by ··· (Q_2)
```

Fig. 1. The general form of the enhanced recursive with

of $\oplus$ for every group by grouping-by the attribute $E.F$. Similarly, the matrix-matrix multiplication (Eq. (9)) is done in two steps. The first step computes $\odot$ between a tuple in E and a tuple in E' under the join condition $E.T = E'.F$. The second step aggregates all the $\odot$ results by the operation of $\oplus$ for every group-by grouping by the attributes $E.F$ and $E'.T$. The formula of re-estimating the means $\boldsymbol{\mu}$ (Eq. (4)) is a matrix-vector multiplication if data is 1-dimensional or a matrix-matrix multiplication otherwise. When high dimensional data is nested as the row-major representation (Table 1(d)), the matrix-matrix multiplication is reduced to matrix-vector multiplication, as shown in Eq. (7).

Re-estimating the covariance/standard deviation $\boldsymbol{\sigma}$ (Eq. (5)) involves the element-wise matrix multiplication if data is 1-dimensional or a tensor-matrix multiplication otherwise. The element-wise matrix multiplication can be expressed by joining two matrices on their two indices to compute $E.e \odot E'.e$. An extra aggregation is required to aggregate on each component k as shown in Eq. (10).

$$E \overset{\oplus(\odot)}{\underset{\substack{E.F=E'.F\\E.T=E'.T}}{\bowtie}} E' = {}_{E.F}\mathcal{G}_{\oplus(\odot)}(E \underset{\substack{E.F=E'.F\\E.T=E'.T}}{\bowtie} E') \tag{10}$$

Similarly, when $\odot$ and $\oplus$ are vector operation and vector aggregation, Eq. (10) is reduced to high dimensional tensor-matrix multiplication.

Value Updating. We need to deal with parameter update when training the model in multiple iterations. We use union by update, denoted as $\uplus$, defined in [28] (Eq. (11)) to address value update.

$$R \uplus_A S = (R - (R \underset{R.A=S.A}{\ltimes} S)) \cup S \tag{11}$$

Suppose t_r is a tuple in R and t_s is a tuple in S. The union by update updates t_r by t_s if t_r and t_s are identical by some attributes A. If t_s does not match any t_r, t_s is merged into the resulting relation. We use $\uplus$ to update the relation of parameters in Table 1(a).

Iterative Evaluation. *RDBMS*s have provided the functionality to support recursive queries, based on *SQL*'99 [11,16], using the with clause in *SQL*. This with clause restricts the recursion to be a stratified program, where non-monotonic operation (e.g., $\uplus$) is not allowed. To support iterative model update, we follow [28], which proves $\uplus$ (union by update) leads to a fixpoint in the enhanced recursive *SQL* queries by *XY*-stratification. We prove that the vector/matrix data type, introduced in this paper, can be used in *XY*-stratification. We omit the details due to the limited space.

The general syntax of the enhanced recursive with is sketched in Fig. 1. It allows union by update to union the result of initial query Q_1 and recursive query

Q_2. Here, the computed by statement in the enhanced with allows users to specify how a relation $R_{i,j}$ is computed by a sequence of queries. The queries wrapped in computed by must be non-recursive.

4 The EM Training

We show the details of supporting the model-based view using *SQL*. First, we present the relational algebra expressions needed followed by the enhanced recursive query. Second, we introduce the queries for model inference.

Parameter Estimation: For simplicity, here we consider the training data point x_i is 1-dimensional scalar. It is natural to extend the query to high dimensional input data when matrix/vector data type and functions are supported by the database system. We represent the input data by a relation $X(ID, x)$, where ID is the tuple identifier for data point x_i and x is a numeric value. The model-based view, which is persisted in the relation GMM(K, pie, mean, cov), where K is the identifier of the k-th component, and 'pie', 'mean', and 'cov' denote the corresponding parameters, i.e., mixing coefficients, means and covariances (standard deviations), respectively. The relation representations are shown in Table 1. The following relational algebra expressions describe the E-step (Eq. (12)), M-step (Eq. (13)-(16)), and parameter updating (Eq. (17)) in one iteration.

$$R \leftarrow \rho_{(ID,K,p)} \Pi_{(ID,K,f)}(GMM \times X) \tag{12}$$

$$N \leftarrow \rho_{(K,\mathrm{pie})}(R \overset{\mathsf{sum}(p)}{\underset{R.ID=X.ID}{\bowtie}} X) \tag{13}$$

$$M \leftarrow \rho_{(K,\mathrm{mean})}(R \overset{\mathsf{sum}(p*x)/\mathsf{sum}(p)}{\underset{R.ID=X.ID}{\bowtie}} X) \tag{14}$$

$$T \leftarrow \Pi_{ID,K,\mathsf{pow}(x-\mathrm{mean})}(X \times N) \tag{15}$$

$$C \leftarrow \rho_{(K,\mathrm{cov})}\, {}_{K}\mathcal{G}_{\mathsf{sum}(p*t)}(T \underset{\substack{R.ID=T.ID \\ R.K=T.K}}{\bowtie} R) \tag{16}$$

$$GMM \leftarrow \rho_{(K,\mathrm{pie},\mathrm{mean},\mathrm{cov})}(N \underset{N.K=M.K}{\bowtie} M \underset{M.K=C.K}{\bowtie} C) \tag{17}$$

In Eq. (12), by performing a Cartesian product of GMM and X, each data point is associated with the parameters of each component. The responsibilities are evaluated by applying an analytical function f to compute the normalized probability density (Eq. (3)) for each tuple, which is the E-step. The resulting relation $R(ID, K, p)$ is shown in Fig. 1(c). For the M-step, the mixing coefficients 'pie' (Eq. (13)), the means 'mean' (Eq. (14)) and the covariances 'cov' (Eq. (15)-(16)) are re-estimated based on their update formulas in Eq. (4)-(6), respectively. In the end, in Eq. (17), the temporary relations N, M and C are joined on attribute K to merge the parameters. The result is assigned to the recursive relation GMM.

Fig. 2 shows the enhanced with query to support Gaussian Mixture Model by EM algorithm. The recursive relation GMM specifies the parameters of k

```
 1.  with
 2.  GMM(K, pie, mean, cov) as (
 3.        (select K, pie, mean, cov from INIT_PARA)
 4.        union by update K
 5.        (select N.K, pie/n, mean, sqrt (cov/pie)
 6.        from N, C where N.K = C.K
 7.        computed by
 8.        R(ID, K, p) as select ID, k, norm(x, mean, cov) * pie /
 9.                         (sum(norm(x, mean, cov) * pie) over (partition by ID))
10.                         from GMM, X
11.        N(K, pie, mean) as select K, sum(p), sum(p * x) / sum(p)
12.                            from R, X where R.ID = X.ID
13.                            group by K
14.        C(K, cov) as select R.K, sum(p * T.val) from
15.                        (select ID, K, pow(x-mean) as val from X, N) as T, R
16.                        where T.ID = R.ID and T.K = R.K
17.                        group by R.K)
18.        maxrecursion 10)
19.  select * from GMM
```

Fig. 2. The enhanced recursive *SQL* for Gaussian Mixtures

Gaussian distributions. In line 3, the initial query loads the initial parameters from relation INI_PARA. The new parameters are selected by the recursive query (line 5-6) evaluated by the computed by statement and update the recursive relation by union by update w.r.t. the component index K. It wraps the queries to compute E-step and M-step of one iteration EM.

We elaborate on the queries in the computed by statement (line 8-17). Specifically, the query in line 8-10 performs the E-step, as the relational algebra in Eq. (12). Here, norm is the Gaussian (Normal) probability density function of data point x given the mean and covariance as input. We can use the window function, introduced in *SQL*'03 to compute the responsibility by Bayes rule in Eq. (3). In line 9, sum() over (partition by()) is the window function performing calculation across a set of rows that are related to the current row. As it does not group rows, where each row retains its separate identity, many *RDBMS*s allow to use it in the recursive query, e.g., *PostgreSQL* and *Oracle*. The window function partitions rows of the Cartesian product results in partitions of the same *ID* and computes the denominator of Eq. (3). In line 11-13, the query computes the means (Eq. (4)) and the mixing coefficients together by a matrix-matrix multiplication due to their common join of R and X. Then, line 14-17 computes the covariances of Eq. (5). First, we compute the square of $x_i - \boldsymbol{\mu}_k$ for each x_i and k, which requires a Cartesian product of N and R (Eq. (15)). Second, the value is weighted by the responsibility and aggregated as specified in Eq. (16). The new parameters in the temporary relation N and C will be merged by joining on the component index K in line 6. The depth of the recursion can be controlled by maxrecursion clause, adapted from *SQL Server* [2]. The maxrecursion clause can effectively prevent infinite recursion caused by infinite fix point, e.g., '**with** $R(n)$ **as** ((**select values**(0)) **union all** (**select** $n+1$ **from** R))' , a legal *SQL*'99 recursion.

Model Inference: Once the model is trained by the recursive query in Fig. 2, it can be materialized in a view for online inference. In the phase of inference, users can query the view by *SQL* to perform clustering, classification and density estimation. Given a batch of data in relation X and a view GMM computed by

Fig. 2, the query below computes the posterior probability that the component K generated the data with index ID. The query is similar to computing the E-step (Eq. (3)) in line 5-7 of Fig. 2.

```
create table R as select ID, K,
norm(x, mean, cov) * pie / (sum(norm(x, mean, cov) * pie)
over (partition by ID)) as p from GMM, X
```

Based on relation $R(ID, K, p)$ above, we can further assign the data into K clusters, where x_i is assigned to cluster k if the posterior probability $p(z_{ik})$ is the maximum among the $\{p(z_{i1}), \cdots, p(z_{iK})\}$. The query below creates a relation CLU(ID, K) to persist the clustering result where ID and K are the index of the data point and its assigned cluster, respectively. It first finds the maximum $p(z_{ik})$ for each data point by a subquery on relation R. The result is renamed as T and is joined with R on the condition of $R.ID = T.ID$ and $R.p = T.p$ to find the corresponding k.

```
create table CLU as select ID, K from R,
(select ID, max (p) as p from R group by ID) as T,
where R.ID = T.ID and R.p = T.p
```

It is worth noting that both of the queries above only access the data exactly once. Thereby, it is possible to perform the inference on-the-fly and only for interested data. Besides density estimation and clustering, result evaluation, e.g., computing the purity, normalized mutual information (NMI) and Rand Index can be conducted in the database by *SQL* queries.

5 Model Maintenance

In this section, we investigate the automatic model/view updating. When the underlying data X changes, a straightforward way is to re-estimate the model over the updated data. However, when only a small portion of the training data are updated, the changes of the corresponding model are slight, it is inefficient to re-estimate the model on-the-fly. Hence, a natural idea arises that whether we can update the existing model by exploring the "incremental variant" of the EM algorithm. And this variant can be maintained by the newly arriving data and a small portion of data extracted from the original dataset. As the statistical model trained by an *SQL* query can be represented by its sufficient statistics, the model is updated by maintaining the model and sufficient statistics.

Maintaining Sufficient Statistics: The sufficient statistic is a function of data X that contains all of the information relevant to estimate the model parameters. As the model is updated, the statistics of data is also updated followed by the changing of the posterior probability $p(z_{ik})$. This process repeats until the statistics converge. We elaborate on the sufficient statistics updating rules below.

Suppose the training dataset of model $\boldsymbol{\theta}$ is $\{x_1, x_2, \cdots, x_n\}$. Let $\boldsymbol{s}$ be the sufficient statistics of $\boldsymbol{\theta}$, based on the Factorization Theorem [10], we can obtain

$$\boldsymbol{s} = \sum_{i=1}^{n} \sum_{\boldsymbol{z}} p(\boldsymbol{z}|x_i, \boldsymbol{\theta})\phi(x_i, \boldsymbol{z}) \tag{18}$$

where $\boldsymbol{z}$ is the unobserved variable, ϕ denotes the mapping function from an instance $(x_i, \boldsymbol{z})$ to the sufficient statistics contributed by x_i. The inserted data is $\{x_{n+1}, x_{n+2}, \cdots, x_m\}$. Let the model for overall data $\{x_1, \cdots, x_n, x_{n+1}, \cdots, x_m\}$ be $\widetilde{\boldsymbol{\theta}}$ and the corresponding sufficient statistics be $\widetilde{\boldsymbol{s}}$. The difference of $\widetilde{\boldsymbol{s}} - \boldsymbol{s}$, denoted as $\Delta\boldsymbol{s}$ is

$$\begin{aligned}
\Delta\boldsymbol{s} &= \sum_{i=1}^{n+m} \sum_{\boldsymbol{z}} p(\boldsymbol{z}|x_i, \widetilde{\boldsymbol{\theta}})\phi(x_i, \boldsymbol{z}) - \sum_{i=1}^{n} \sum_{\boldsymbol{z}} p(\boldsymbol{z}|x_i, \boldsymbol{\theta})\phi(x_i, \boldsymbol{z}) \\
&= \sum_{i=1}^{n+m} \sum_{\boldsymbol{z}} [p(\boldsymbol{z}|x_i, \widetilde{\boldsymbol{\theta}}) - p(\boldsymbol{z}|x_i, \boldsymbol{\theta})]\phi(x_i, \boldsymbol{z}) \qquad (19) \\
&+ \sum_{i=n+1}^{m} \sum_{\boldsymbol{z}} p(\boldsymbol{z}|x_i, \boldsymbol{\theta})\phi(x_i, \boldsymbol{z}) \qquad (20)
\end{aligned}$$

According to above equations, we observe that the delta part of the sufficient statistics $\Delta\boldsymbol{s}$ consists of two parts: (1) changes of the sufficient statistics for the overall data points $\{x_1, x_2 \cdots x_m\}$ in Eq. (19), and (2) the additional sufficient statistics for the newly inserted data points $\{x_{n+1}, \cdots x_m\}$ in Eq. (20). Consider to retrain a new model $\widetilde{\boldsymbol{\theta}}$ over $\{x_1, x_2, \cdots, x_m\}$ in T iterations by taking $\boldsymbol{\theta}$ as the initial parameter, i.e., $\boldsymbol{\theta}^{(0)} = \boldsymbol{\theta}$ and $\boldsymbol{\theta}^{(T)} = \widetilde{\boldsymbol{\theta}}$. We have

$$\begin{aligned}
\Delta\boldsymbol{s} &= \sum_{i=1}^{n+m} \sum_{\boldsymbol{z}} [p(\boldsymbol{z}|x_i, \boldsymbol{\theta}^{(T)}) - p(\boldsymbol{z}|x_i, \boldsymbol{\theta}^{(0)})]\phi(x_i, \boldsymbol{z}) \qquad (21) \\
&+ \sum_{i=n+1}^{m} \sum_{\boldsymbol{z}} p(\boldsymbol{z}|x_i, \boldsymbol{\theta}^{(0)})\phi(x_i, \boldsymbol{z}) \qquad (22) \\
&= \sum_{t=1}^{T} \sum_{i=1}^{n+m} \sum_{\boldsymbol{z}} [p(\boldsymbol{z}|x_i, \boldsymbol{\theta}^{(t)}) - p(\boldsymbol{z}|x_i, \boldsymbol{\theta}^{(t-1)})]\phi(x_i, \boldsymbol{z}) \qquad (23) \\
&+ \sum_{i=n+1}^{m} \sum_{\boldsymbol{z}} p(\boldsymbol{z}|x_i, \boldsymbol{\theta}^{(0)})\phi(x_i, \boldsymbol{z}) \qquad (24)
\end{aligned}$$

Above equations indict how to compute $\Delta\boldsymbol{s}$. For the inserted data $\{x_{n+1}, \cdots x_m\}$, the delta can be directly computed by evaluating the original model $\boldsymbol{\theta}^{(0)}$ as Eq. (24), while for original data, the delta can be computed by updating the model $\boldsymbol{\theta}^{(t)}$ iteratively using all the data $\{x_1, x_2 \cdots x_m\}$ as Eq. (23). Since most of the computational cost is on the iteration of Eq. (23), we approximate the computation. First, we use the stochastic approximation algorithm, where the parameters are updated after the sufficient statistics of each new data point x_i is

```
1. create trigger T1 before insert on $X$
2. for each statement
3. execute procedure DATA_SELECTION

4. create trigger T2 before insert on $X$
5. for each row
6. execute procedure DATA_INSERTION

7. create trigger T3 after insert on $X$
8. for each statement
9. execute procedure MODEL_UPDATE
```

Fig. 3. The triggers for incremental update

computed, instead of the full batch dataset [14,18,22]. The second is discarding the data points which are not likely to change their cluster in the future, as the scaling clustering algorithms adopt for speedup [6]. We discuss our strategy of selecting partial original data in $\{x_1, x_2, \cdots, x_n\}$ for model update. There is a tradeoff between the accuracy of the model and the updating cost. We have two strategies: a distance-based and a density-based strategy. For the distance-based strategy, we use Mahalanobis distance [9] to measure the distance between a data point and a distribution. For each data x_i, we compute the Mahalanobis distance, $D_k(x_i)$, to the k-th component with mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\sigma}_k$.

$$D_k(x_i) = \sqrt{(x_i - \boldsymbol{\mu}_k)^T \boldsymbol{\sigma}_k^{-1} (x_i - \boldsymbol{\mu}_k)} \tag{25}$$

We can filter the data within a given thresholding radius with any component. Another density-based measurement is the entropy of the posterior probability for data x_i as in Eq. (26), where $p(z_{ik})$ is evaluated by parameter $\boldsymbol{\theta}^{(0)}$. The larger the entropy, the lower the possibility of assigning x_i to any one of the components.

$$E(x_i) = -\sum_{k=1}^{K} p(z_{ik}) ln\ p(z_{ik}) \tag{26}$$

Similarly, considering deleting m data points $\{x_{n-m+1}, \cdots x_n\}$ from $\{x_1, x_2 \cdots x_n\}$, the difference of the sufficient statistics, $\Delta \boldsymbol{s}$ is

$$\begin{aligned}\Delta \boldsymbol{s} = &\sum_{t=1}^{T} \sum_{i=1}^{n-m} \sum_{\boldsymbol{z}} [p(\boldsymbol{z}|x_i, \boldsymbol{\theta}^{(t)}) - p(\boldsymbol{z}|x_i, \boldsymbol{\theta}^{(t-1)})] \phi(x_i, \boldsymbol{z}) \\ &- \sum_{i=n-m+1}^{n} \sum_{\boldsymbol{z}} p(\boldsymbol{z}|x_i, \boldsymbol{\theta}^{(0)}) \phi(x_i, \boldsymbol{z})\end{aligned} \tag{27}$$

A Trigger-based Implementation: In *RDBMS*s, the automatic model updating mechanism is enabled by triggers built on the relation of the input data. There are three triggers built on the relation of training data X, whose definitions are shown in Fig. 3. Before executing the insertion operation, two triggers T1 (line 1-3 in Fig. 3) and T2 (line 4-6 in Fig. 3) prepare the data for model updating in a temporary relation X'. Here, T1 performs on each row to select a subset from original data in $\{x_1, x_2, \cdots, x_n\}$ based on a selection criterion. Additionally, T2 inserts all the newly arrived data $\{x_{1+n}, x_2, \cdots, x_m\}$ to relation X'. After the data preparation finished, another trigger T3 (line 7-9 in Fig. 3) will call a *PSM* to compute the $\Delta \boldsymbol{s}$ by X'. In the *PSM*, first, the delta of the newly inserted data (Eq. (23)) is computed to reinitialize the parameters of the model. Then, T iterations of scanning relation X' is performed. Where in each iteration. X' is randomly shuffled and each data point is used to update the sufficient statistics it contributes as well as the model instantly. It is worth mentioning that the data selection in trigger T1 can be performed offline, i.e., persisting a subset of training data with a fixed budget size for model updating in the future. In addition, the sufficient statistics for original model $\boldsymbol{\theta}^0$ can be precomputed. Those will improve the efficiency of online model maintenance significantly. The actions of these triggers are transparent to the database users.

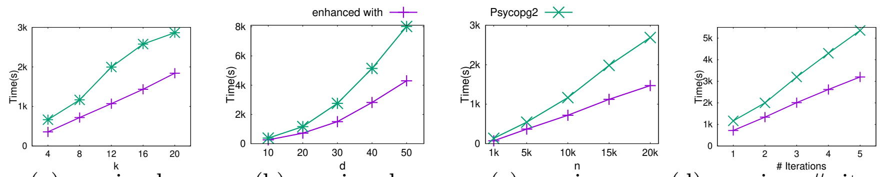
6 Experimental Studies

In this section, we present our experimental studies of supporting model-based view training, inference, and maintenance in *RDBMS*. We conduct extensive experiments to investigate the following: (a) to compare the performance of our enhanced with and looping control by a host language, (b) to test the scalability of the recursive queries for different models, and (c) to validate the efficiency of our model maintenance mechanism.

Experimental Setup: We report our performance studies on a PC with Intel(R) Xeon(R) CPU E5-2697 v3 (2.60GHz) with 96GB RAM running Linux CentOS 7.5 64 bit. We tested the enhanced recursive query on *PostgreSQL* 10.10 [3]. The statistical function and matrix/vector computation function are supported by Apache MADlib 1.16 [12]. All the queries we tested are evaluated in a single thread *PostgreSQL* instance.

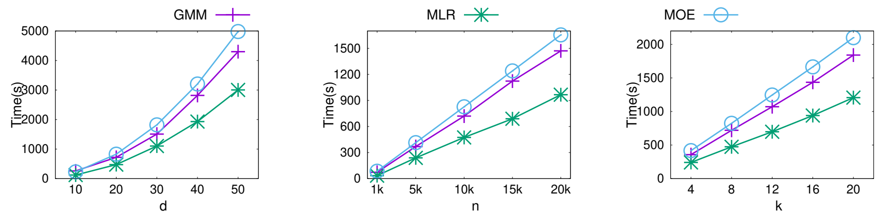
with+ **vs.** *Psycopg2*: We compare the enhanced with, which translates the recursive *SQL* query to *SQL*/*PSM* with the implementation of using a host language to control the looping, which is adopted in previous EM implementation [20]. We implement the latter by *Psycopg2* [1], a popular *PostgreSQL* adapter for the python language. Regarding the EM algorithm, the E-step, M-step, and parameter updating are wrapped in a python for-loop, and executed by a cursor alternatively. We compare the running time of these two implementations, i.e., enhanced with and *Psycopg2* for training Gaussian Mixture Model by varying the dimension d of data point (Fig. 4(b)), the scale of the training data n (Fig. 4(c)), the number of components k (Fig. 4(a)) and the number of iterations (Fig. 4(d)). The training data is evenly generated from 10 Gaussian distributions.

The evaluated time is the pure query execution time where the costs of database connection, data loading and parameter initialization are excluded. The experiments show that enhanced with outperforms *Psycopg2* significantly, not only for multiple iterators in Fig. 4(d) but also for per iteration in Fig. 4(b)-4(a). For one thing, the implementation of *Psycopg2* calls the database multiple



(a) varying k (b) varying d (c) varying n (d) varying # iterations

Fig. 4. with+ vs. *Psycopg2*



(a) k = 8, n = 10000 (b) k = 8, d = 20 (c) d = 20, n = 10000

Fig. 5. Scalability Test

times per iteration, incurring much client-server communication and context switch costs. For the other, the issued queries from client to server will be parsed, optimized and planned on-the-fly. These are the general problems of calling *SQL* queries by any host language. Meanwhile, we implement the hybrid strategy of *SQLEM* [19] in *PostgreSQL*. For Gaussian Mixture model, one iteration for 10,000 data points with 10 dimensions fails to terminate within 1 hour. In their implementation, $2k$ separate *SQL* queries evaluate the means and variances of k components respectively, which is a performance bottleneck.

Experiments on synthetic data: We train Gaussian Mixture model (GMM) [5], mixture of linear regression (MLR) [23] and a neural network model, mixture of experts (MOE) [25] by evaluating *SQL* recursive queries in *PostgreSQL*. Given the observed dataset as $\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, the MLR models the density of y given x as

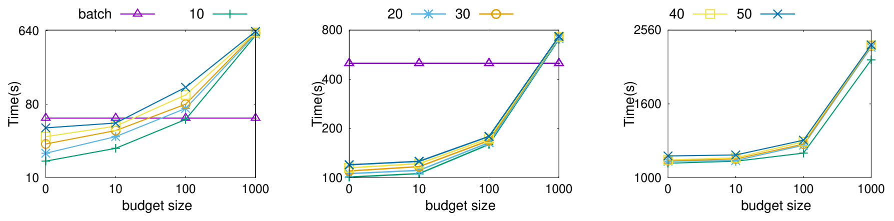
$$p(y_i|x_i) = \sum_{k=1}^{K} \pi_k \mathcal{N}(y_i|x_i^T \boldsymbol{\beta}_k, \boldsymbol{\sigma}_k) \tag{28}$$

And the MOE models the density of y given x as

$$p(y_i|x_i) = \sum_{k=1}^{K} g_k(x_i) \mathcal{N}(y_i|x_i^T \boldsymbol{\beta}_k, \boldsymbol{\sigma}_k) \tag{29}$$

where $\boldsymbol{\beta}_k \in \mathbb{R}^d$ is the parameters of a linear transformer, $\mathcal{N}$ is the probability density function of a Gaussian given mean $x_i^T \boldsymbol{\beta}_k \in \mathbb{R}$ and standard deviation $\boldsymbol{\sigma}_k \in \mathbb{R}$. In Eq. (29), $g_k(x)$ is called the gating function, given by computing the softmax in Eq. (30) where $\boldsymbol{\theta} \in \mathbb{R}^d$ is a set of linear weights on x_i.

$$g_k(x_i) = \frac{e^{x_i \boldsymbol{\theta}_k}}{\sum_{j=1}^{K} e^{x_i \boldsymbol{\theta}_j}} \tag{30}$$



(a) n = 100K (b) n = 1M (c) n = 10M

Fig. 6. Insert maintenance

The intuition behind the gating functions is a set of 'soft' learnable weights which determine the mixture of K local models. We adopt the single loop EM algorithm [24] to estimate the parameters of MOE, which uses least-square regression to compute the gating network directly. For GMM, the training data is evenly drawn from 10 Gaussian distributions. For MLR and MOE, the training data is generated from 10 linear functions with Gaussian noise. The parameters of the Gaussians and the linear functions are drawn from the uniform distribution $[0, 10]$. And the initial parameters are also randomly drawn from $[0, 10]$.

Fig. 5 displays the training time per iteration of the 3 models by varying the data dimension d (Fig. 5(a)), the scale of the training data n (Fig. 5(b)) and the number of clusters k (Fig. 5(c)). In general, for the 3 models, the training time grows linearly as n and k increase, while the increment of data dimension d has a more remarkable impact on the training time. When increasing n and k, the size of intermediate relations, e.g., relation R for computing the responsibilities in Eq. (12) grow linearly. Therefore the training cost grows linearly with regards to n and k. However, in the 3 models, we need to deal with $d \times d$ dimensional matrices in the M-step. For GMM, it needs to compute the probability density of the multivariable Gaussians and reestimate the covariance matrices. For MLR and MOE, they need to compute the matrix inversion and least square regression. The training cost grows regarding the size of the matrix. The comparison shows it is still hard to scale high-dimensional analysis in a traditional database system. However, efficiency can be improved on a parallel/distributed platform and new hardware.

Incremental Maintenance: Finally, we test the performance of our trigger-based model updating mechanism. First, we train GMM for 1-dimensional data generated from 2 Gaussian distributions. The original models are trained over 100k, 1M and 10M data points, respectively with 15 iterations. The overall training time is recorded as the 'batch' mode training time, which is 54s, 501s and 4,841s respectively. After the model is trained and persisted. We insert 10, 20, 30, 40, 50 data points to the underlying data by varying the budget size of selected data from 0 to 1,000.

Fig. 6 shows the insertion time w.r.t. the budget size of the selected data for the 3 models. The insertion time is the collapsed time from the insert command issuing to the transaction commit, including the cost of data selection with the density-based strategy and computing initial sufficient statistics. As the number of processed tuples increases, the insertion time grows linearly. Compare to the retraining cost, i.e., the batch training time, it is not always efficient to update the existing model. The choice depends on two factors, the size of overall data points, and the budget size plus insertion size, i.e., the numbers of data points to be processed in the updating. The updating mechanism may not be efficient and effective when the overall data size is small or there is a large volume of insertion. That is because, for the batch training mode, computation of parameter evaluation dominates the cost. While for the model updating, since the sufficient statistics and the model are updated when processing each data point, the updating overhead becomes a main overhead. Meanwhile, we notice that the collapsed time of data selection and computing initial sufficient statistics take about 10s, 100s and 1,000s for data size of 100k, 1M and 10M, respectively. Precomputing and persisting these results will benefit for a larger dataset.

7 Conclusion

Integrating machine learning techniques into database systems facilitates a wide range of applications in industrial and academic fields. In this paper, we focus on supporting EM algorithm in *RDBMS*. Different from the previous approach, our approach wraps the E-step and M-step in an enhanced *SQL* recursive query to reach an iterative fix point. We materialize the learned model as a database view to query. Furthermore, to handle model updates, we propose an automatic view updating mechanism by exploiting the incremental variant of the EM algorithm. The extensive experiments we conducted show that our approach outperforms the previous approach significantly, and can support multiple mixture models by EM algorithm, as well as the efficiency of the incremental model update. The *SQL* implementation can be migrated to parallel and distributed platforms, e.g., *Hadoop* and *Spark*, to deploy large scale machine learning applications. These directions deserve future explorations.

Acknowledgement

This work is supported by the Research Grants Council of Hong Kong, China under No. 14203618, No. 14202919 and No. 14205520.

References

1. `http://initd.org/psycopg/docs/index.html`.
2. Microsoft SQL documentation. `https://docs.microsoft.com/en-us/sql/`.
3. Postgresql. `https://www.postgresql.org`.
4. F. Arni, K. Ong, S. Tsur, H. Wang, and C. Zaniolo. The deductive database system LDL++. *TPLP*, 3(1), 2003.
5. C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.
6. P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. of KDD'98*, pages 9–15, 1998.

7. A. P. Dempster. Maximum likelihood estimation from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 39:1–38, 1977.
8. A. Deshpande and S. Madden. Mauvedb: supporting model-based user views in database systems. In *Proc. of SIGMOD'06*, pages 73–84, 2006.
9. R. O. Duda and P. E. Hart. *Pattern classification and scene analysis*. A Wiley-Interscience publication. Wiley, 1973.
10. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2 edition, 2001.
11. S. J. Finkelstein, N. Mattos, I. Mumick, and H. Pirahesh. Expressing recursive queries in SQL. *ISO-IEC JTC1/SC21 WG3 DBL MCI*, (X3H2-96-075), 1996.
12. J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The madlib analytics library or MAD skills, the SQL. *PVLDB*, 5(12):1700–1711, 2012.
13. M. L. Koc and C. Ré. Incrementally maintaining classification using an RDBMS. *PVLDB*, 4(5):302–313, 2011.
14. P. Liang and D. Klein. Online EM for unsupervised models. In *Proc. of NAACL'09*, pages 611–619, 2009.
15. G. McLachlan and T. Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007.
16. J. Melton and A. R. Simon. *SQL: 1999: understanding relational language components*. Morgan Kaufmann, 2001.
17. R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.
18. R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. 1998.
19. C. Ordonez. Optimization of linear recursive queries in SQL. *IEEE Trans. Knowl. Data Eng.*, 22(2), 2010.
20. C. Ordonez and P. Cereghini. SQLEM: fast clustering in SQL using the EM algorithm. In *Proc. of SIGMOD*, pages 559–570, 2000.
21. P. Tamayo, C. Berger, M. M. Campos, J. Yarmus, B. L. Milenova, A. Mozes, M. Taft, M. F. Hornick, R. Krishnan, S. Thomas, M. Kelly, D. Mukhin, R. Haberstroh, S. Stephens, and J. Myczkowsji. Oracle data mining - data mining in the database environment. In *The Data Mining and Knowledge Discovery Handbook.*, pages 1315–1329. 2005.
22. D. M. Titterington. Recursive parameter estimation using incomplete data. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(2):257–267, 1984.
23. K. Viele and B. Tong. Modeling with mixtures of linear regressions. *Statistics and Computing*, 12(4):315–330, 2002.
24. Y. Yang and J. Ma. A single loop EM algorithm for the mixture of experts architecture. In *Advances in Neural Networks - ISNN 2009, 6th International Symposium on Neural Networks, ISNN 2009, Proceedings, Part II*, pages 959–968, 2009.
25. S. E. Yuksel, J. N. Wilson, and P. D. Gader. Twenty years of mixture of experts. *IEEE Trans. Neural Netw. Learning Syst.*, 23(8):1177–1193, 2012.
26. C. Zaniolo, N. Arni, and K. Ong. Negation and aggregates in recursive rules: the LDL++ approach. In *Proc. of DOOD*, 1993.
27. C. Zaniolo, S. Stefano, Ceri, C. Faloutsos, R. T. Snodgrass, V. S. Subrahmanian, and R. Zicari. *Advanced database systems*. Morgan Kaufmann, 1997.
28. K. Zhao and J. X. Yu. All-in-one: Graph processing in rdbmss revisited. In *Proc. of SIGMOD'17*, 2017.