A Data Mining System Based on SQL Queries and UDFs for Relational Databases

Carlos Ordonez
University of Houston
Houston, TX 77204, USA

Carlos Garcia-Alvarado
University of Houston
Houston, TX 77204, USA

ABSTRACT

Most research on data mining has proposed algorithms and optimizations that work on flat files, outside a DBMS, mainly due to the following reasons. It is easier to develop efficient algorithms in a traditional programming language. The integration of data mining algorithms into a DBMS is difficult given its relational model foundation and system architecture. Moreover, SQL may be slow and cumbersome for numerical analysis computations. Therefore, data mining users commonly export data sets outside the DBMS for data mining processing, which creates a performance bottleneck and eliminates important data management capabilities such as query processing and security, among others (e.g. concurrency control and fault tolerance). With that motivation in mind, we developed a novel system based on SQL queries and User-Defined Functions (UDFs) that can directly analyze relational tables to compute statistical models, storing such models as relational tables as well. Most algorithms have been optimized to reduce the number of passes on the data set. Our system can analyze large and high dimensional data sets faster than external data mining tools.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Relational databases*

General Terms

Algorithms, Languages, Performance, Theory

1. INTRODUCTION

Our research studies the problem of performing data mining inside a DBMS, paying attention to large data sets. Data mining research is extensive, but the vast majority of proposals has proposed efficient algorithms and optimizations that work on flat files [7]. Therefore, they are difficult to integrate with a DBMS. However, it is fair to say that despite alternative proposals and technologies, relational DBMSs remain the dominating technology to manage and analyze large databases. Even further, open-source DBMSs (e.g. PostgreSQL, MySQL, SQLite) are becoming increasingly popular. Well-known data mining techniques include K-means clustering [4], PCA [2, 1], regression [1], Bayesian classifiers [8], association rules, and decision trees [1] among others. The problem of integrating machine learning and statistical techniques with a DBMS has received scant attention. The integration of data mining algorithms into a DBMS is difficult due to the mathematical nature of models, the lack of access to the DBMS source code and the complex internal DBMS architecture. Another, more practical, reason, not to use a DBMS for data mining tasks is the comprehensive set of techniques available in external data mining tools. Commercial DBMSs provide basic data mining algorithms, but they are difficult to customize, extend and optimize because algorithms are either internally integrated or work on flat files exported with fast interfaces. Also, from a mathematical point of view, they lack a unifying statistical foundation. As a consequence, users or the DBMS itself generally export data sets as files to an external data mining program, thereby going through a performance bottleneck (extracting large tables is slow) and losing the data management capabilities provided by the DBMS (query processing, security, concurrency control, fault tolerance).

From a practical standpoint, SQL is and will likely remain the language to interact with a DBMS. With that motivation in mind, we present a data mining system that can work on top of a relational DBMS based on SQL queries and User-Defined Functions (UDFs), proving that the common perception that SQL is inefficient or inadequate for data mining is false.

Our system incorporates fundamental machine learning and statistical models whose equations and algorithmic steps are computed with a combination of optimized SQL queries and UDFs. Our system includes, among other techniques, Principal Component Analysis (PCA) [2, 5], clustering with mixtures of distributions [4], linear regression [5], parametric statistical tests, and Bayesian Classifiers [8], covering a wide spectrum of statistical and machine learning models. Unlike most research proposals and existing tools, our system can directly process relational tables, truly performing "in-database" analytics.

Our approach is unique and competitive. Most importantly, matrix equations are computed with SQL queries and UDFs, instead of a traditional programming language. Thus we have been able to integrate techniques without having access to the DBMS internal source code, an option available only to experienced DBMS researchers or developers. Our system provides efficient algorithms, which can analyze large data sets faster than external data mining tools. Several of our SQL-based algorithms can process a large data

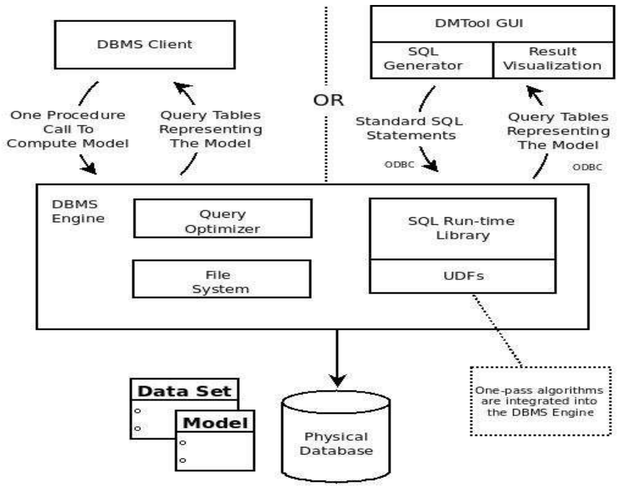


Figure 1: System architecture.

set in only one pass, while the rest can do it in a few passes. Moreover, all our algorithms have linear scalability on data set size and most of them scale linearly on dimensionality. Our system has promise of wide application as DBMSs remain the standard data management platform, databases keep growing and computers become faster.

2. SYSTEM DESCRIPTION

Our system is based on a comprehensive research effort, where different data mining techniques have been adapted and programmed with optimized SQL queries [4] and more recently with UDFs [5, 9]. We start by presenting the system architecture. We then define the input data set, as well as available statistical models. Finally, we provide a technical summary of how we programmed data mining algorithms with optimized SQL queries and UDFs.

2.1 System Architecture

Our system is based on a client-server architecture, shown in Figure 1. The data set, model matrices and model parameters, are all stored as relational tables. A program at the client computer, connected to the DBMS server via ODBC, calls a stored procedure, which in turn generates SQL queries. The system provides the option to evaluate numerically intensive computations such as matrix equations or complex models with precompiled UDFs. In general, the program generates SQL statements (DDL and DML), monitors algorithm progress (convergence, number of iterations, time) and at the end returns a few tables, corresponding to the model at hand.

2.2 Statistical Models and Algorithms

The input data set $X = \{x_1, x_2, \dots, x_n\}$ has n records (points) and d attributes (dimensions, features) plus an extra "supervised" attribute (class G or dependent variable Y). That is, n is the data set size. When all attributes are numeric d represents dimensionality.

Our system provides widely used statistical models sharing a common mathematical foundation. Unsupervised models include PCA (most common method to reduce dimensionality; solved with SVD) and clustering with mixtures (solved with the popular K-means algorithm or the more complex EM method). On the other hand, supervised (predictive) models include: linear regression (solved by least squares), the popular Naïve Bayes classifier [10] (estimating joint probability as a product of marginal probabilities) and a sophisticated Bayesian classifier based on class decomposition (using K-means as a building block). In supervised techniques, X has an additional "target" attribute: a numeric dimension Y (for regression) or a discrete (categorical) "class" attribute G (for Naïve Bayes [10] and the Bayesian classifier [8]). Naïve Bayes accepts a combination of numeric and discrete attributes. All models share a common mathematical foundation related to the multivariate normal density function.

2.3 Processing with SQL Queries and UDFs

We start by discussing storage and indexing [4, 5]. The data set and model matrices are stored on tables, indexed by specific combinations of matrix subscripts depending on their storage layout. In general, given a $d \times k$ matrix it can be stored as a "matrix" table with d rows and k columns, as a wide table with dk columns or as a long table with dk rows having one matrix value each. On the other hand, our system uses two fundamental data set layouts. The first layout for X is a standard tabular representation which has n rows, each having d columns, which generally provides best I/O performance. The second layout is based on a pivoted table with one attribute value per row and up to dn rows, which removes DBMS limitations on the maximum number of columns for high dimensional data and which enables more efficient manipulation of sparse matrices.

Evaluating matrix equations with SQL queries is difficult [4, 6]. It is not possible to create fixed queries that can efficiently evaluate mathematical equations because the table definition for the input data set can vary (especially the horizontal layout with d columns). Therefore, SQL code generation at run-time is mandatory. Statistical methods and data mining algorithms are programmed with SQL queries and UDFs, as shown in Figure 2, based on the input data set and two sets of parameters. The first set of parameters controls the mathematical behavior of each technique. Examples include the number of clusters, tolerance threshold for convergence, numeric stability issues, and so on. The second set of parameters controls database systems optimizations, which are tailored to each algorithm (with some common optimizations across multiple models). SQL queries combine joins, aggregations, arithmetic expressions. Certain computations can be optionally evaluated with UDFs. UDFs generally accelerate aggregations, reduce the number of required tables scans, push more processing into RAM memory and eliminate joins. Therefore, UDFs represent an optimization of SQL queries. More specifically, UDFs extend the DBMS with primitive mathematical functions evaluating demanding matrix equations. We should emphasize SQL code, generated by a host language (e.g. Java, C++), can evaluate any equation with matrices, no matter how complex or slow. Thus the main reason to exploit UDFs is faster execution and to a second extent, programming flexibility.

2.4 Optimizations

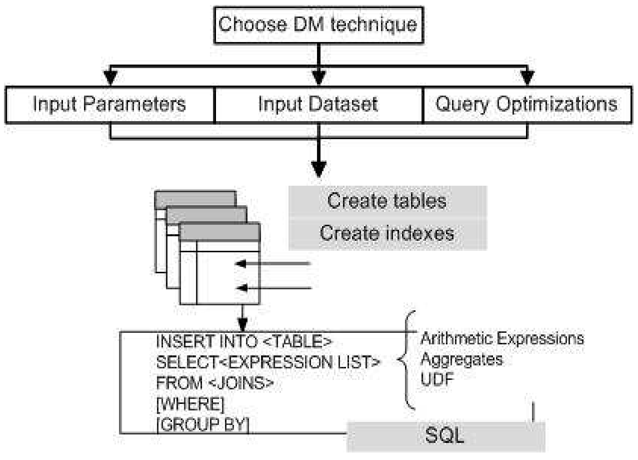


Figure 2: SQL code generation.

We now discuss two main kinds of optimizations: algorithmic and database systems oriented.

We start by discussing algorithmic optimizations, with general applicability. The (multidimensional) sufficient statistics of the multivariate normal distribution are an essential ingredient to accelerate data mining computations. Our system makes extensive use of sufficient statistics to reduce the number of passes on the data set, even one pass. For algorithms like PCA, Naïve Bayes and linear regression sufficient statistics enable computation of the model in one table scan both with SQL queries or UDFs. On the other hand, for K-means clustering and the Bayesian classifier (which are mathematically harder problems), sufficient statistics enable incremental learning, significantly reducing the number of iterations. In an orthogonal manner, sufficient statistics can be computed on samples, providing an approximation to the model, whose accuracy can be controlled. A key difference with previous research is that we can compute sufficient statistics to get either a full covariance or correlation matrix, instead of a diagonal matrix. When the data set is large enough (say $n \geq 100$M) sufficient statistics can be derived with geometric sampling or bootstrapping techniques (leveraging block-based sampling mechanisms provided by the DBMS), which can get arbitrarily accurate ($> 95\%$) approximations in much less time than a full table scan.

Our sufficient statistics are basically the linear sum of points L (a d-dimensional vector) and the quadratic (with cross-products) sum of points Q (a $d \times d$ matrix) [3, 4, 5]:

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

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

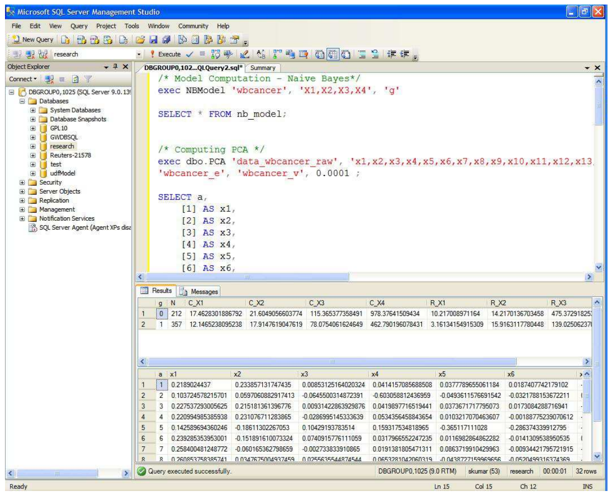
Such summary matrices are independently computed on k subsets of the data set for clustering and the Bayesian classifier. Summary matrices are efficiently computed with an aggregate UDF or slower, but with better portability, with SQL queries. For K-means and EM clustering we developed an incremental algorithm that can obtain an acceptable solution in a few iterations (in some cases one), being much faster than the standard algorithm; this is particularly difficult to achieve with SQL or UDFs.

From a database systems perspective, we carefully analyze the query plan for each data mining computation, identifying optimizations that work well across DBMSs. Since equations are evaluated with SQL queries there are many temporary tables generated on the way. In general, we control how each temporary table is stored and indexed. Query rewriting is essential. The join operation is the main operation that can degrade performance if not carefully managed. Denormalized tables with sufficient statistics avoid joins down the road. Whenever possible, aggregations are pushed (computed before) joins. Hash-based joins are the preferred join algorithm on large tables. To get best performance joined tables are indexed by the same subscripts so that the hashing function maps rows to the same partitions. Alternatively, whenever a hash-join is not possible we exploit merge-sort joins. Sort-Merge joins are typically used when both tables are indexed by different subscripts. Our last optimization worth mentioning is caching, which is automatically applied to sufficient statistics [9] and which can also be applied with data sets that can fit in main memory. This optimization is especially beneficial for iterative processing or to concurrently compute multiple models.

3. SYSTEM DEMONSTRATION

We plan to give an interactive demonstration, considering researchers and students with different backgrounds. We will demonstrate our system with a remote DBMS. The database will contain well-known real data sets (from the UCI ML repository, to illustrate data mining applicability) and several large (n=10K,100K,1M) synthetic data sets (to illustrate time efficiency). The same data sets will be also available as flat files to be analyzed by an external data mining tool (e.g. Weka, R statistical package). The demonstration will flow from an overview of our tool, learning available data mining techniques, input data set, parameters, output and visualization, to technical aspects. The user We will give an overview of available data mining techniques. The user will be able to understand technical aspects of the system such as table layouts, SQL queries, UDFs and query optimizations. Our system demonstration will illustrate the following points: (1) being able to efficiently perform data mining on large data sets entirely inside the DBMS (i.e. no data exported); (2) our system is faster than external data mining tools working on flat files; (3) showing exporting large data sets from the DBMS is a bottleneck with ODBC and even with BULK export utilities; (4) quickly computing accurate approximate models with sampling techniques, especially with very large data sets (for specific models); (5) understanding the impact of optimizations, turning them on an off, as well as highlighting which challenging mathematical equations are evaluated with SQL queries or UDFs; Figure 3 shows stored procedure calls and output for two statistical models in the DBMS.

We first discuss a general demonstration, suitable for researchers working on data mining, information retrieval or database systems. The demonstration will show a complete data mining run going from selecting a data set to actually browsing a model. The user will start by choosing one data mining technique from an on-line help. The user will get an overview of optimizations. For supervised models



```
/* Model Computation - Naive Bayes*/
exec NBModel 'wbcancer', 'X1,X2,X3,X4', 'g'

SELECT * FROM nb_model;


/* Computing PCA */
exec dbo.PCA 'data_wbcancer_raw', 'x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,
'wbcancer_e', 'wbcancer_v', 0.0001 ;

SELECT a,
    [1] AS x1,
    [2] AS x2,
    [3] AS x3,
    [4] AS x4,
    [5] AS x5,
    [6] AS x6,
```

	g	N	C_X1	C_X2	C_X3	C_X4	R_X1	R_X2	R_X3
1	0	212	17.4628301886792	21.6049056603774	115.365377358491	978.37641509434	10.217008971164	14.2170136703458	475.37291825
2	1	357	12.1465238095238	17.9147619047619	78.0754061624649	462.790196078431	3.16134154915309	15.9163117780448	139.0250623

	a	x1	x2	x3	x4	x5	x6
1	1	0.2189024437	0.233857131747435	0.00853125164020324	0.0414157085688508	0.0377789655061184	0.0187407742179102
2	2	0.103724578215701	0.0597060882917413	-0.0645500314872391	-0.603058812436959	-0.0493611576691542	-0.0321788153672211
3	3	0.227537293005625	0.215181361396776	0.00931422863929876	0.0419897716519441	0.0373671717795073	0.0173084288716941
4	4	0.220994985385938	0.231076711283865	-0.0286995145333639	0.0534356458843654	0.0103217070463607	-0.00188775239070612
5	5	0.142589694360246	-0.18611302267053	0.10429193783514	0.159317534818965	-0.365117111028	-0.286374339912795
6	6	0.239285353953001	-0.151891610073324	0.0740915776111059	0.0317966552247235	0.0116982864862282	-0.0141309538950535
7	7	0.258400481248772	-0.060165362798659	-0.002733833910865	0.0191381805471311	0.0863719910429963	-0.00934421795721915
8	8	0.260853758385741	0.0347675004937459	0.0255635544874544	0.0653281042060319	-0.0438727159969656	-0.0520499316374369

```
Query executed successfully.    DBGROUP0,1025 (9.0 RTM)    skumar (53)    research    00:00:01    32 rows
Ln 15    Col 15    Ch 12    INS
```

Figure 3: Calling stored procedures and querying output models.

the demonstration will have two subparts: one to build the model and a second one to test the model on a different set of records (i.e. scoring) on a table with the same columns (i.e. to independently test accuracy). Then the user can run the algorithm, which should quickly finish in a few seconds for most data sets. The user will be able to see Naïve Bayes, PCA, and linear regression can be computed in a single table scan in a matter of seconds. On the other hand, the user can use sampling techniques to get accurate approximate models on larger data sets ($n \geq 1$M) in a few seconds. Finally, the user will be able to browse actual the statistical model, which consists of a few relational tables. In addition, for supervised models the user can evaluate model accuracy on a blind test with an independent set of points (i.e. training and testing). Also, the user can compare the predicted (by our system) and existing class values for the test data set.

We now discuss a technical demonstration, suitable for database systems researchers. For performance comparison purposes we will compare with public domain data mining tools (e.g. Weka, R package) working on flat files, computing the same model. We will temporarily stop the DBMS to be fair: SQL queries and UDFs will be shown to be faster than external data mining tools. We will also show exporting large data sets (ODBC, and even with fast BULK utilities) is a bottleneck, making external processing unattractive. The user will be able to understand the impact of available optimizations (including UDFs, sampling and caching) turning them on and off. For instance, the user can run multiple algorithms on the same data set faster when the data set is cached. We will go deeper, providing an overview of important working tables, explaining their layout and explaining how primary or secondary indices accelerate joins and aggregations. Finally, we will show representative SQL queries and UDFs evaluating matrix equations. When showing SQL queries we will explain how tables are joined (indicating specific join algorithm), how aggregations or SQL arithmetic expressions compute equations, how aggregations are pushed before joins and how the output table of one query becomes the input for the next query. For UDFs we will explain how aggregate UDFs accelerate computations by pushing complex mathematical equations into the DBMS by comparing the UDF code against the slower SQL queries.

Acknowledgments

This work was supported by NSF grants CCF 0937562 and IIS 0914861.

4. REFERENCES

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

[2] M. Navas and C. Ordonez. Efficient computation of PCA with SVD in SQL. In *KDD Workshop on Data Mining using Tensors and Matrices*, Article no. 5, 2009.

[3] M. Navas, C. Ordonez, and V. Baladandayuthapani. On the computation of stochastic search variable selection in linear regression with UDFs. In *Proc. IEEE ICDM Conference*, pages 941–946, 2010.

[4] C. Ordonez. Integrating K-means clustering with a relational DBMS using SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 18(2):188–201, 2006.

[5] C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(12):1752–1765, 2010.

[6] C. Ordonez and P. Cereghini. SQLEM: Fast clustering in SQL using the EM algorithm. In *Proc. ACM SIGMOD Conference*, pages 559–570, 2000.

[7] C. Ordonez and J. García-García. Database systems research on data mining. In *Proc. ACM SIGMOD Conference*, pages 1253–1254, 2010.

[8] C. Ordonez and S.K. Pitchaimalai. Bayesian classifiers programmed in SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(1):139–144, 2010.

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

[10] S.K. Pitchaimalai, C. Ordonez, and C. Garcia-Alvarado. Comparing SQL and MapReduce to compute Naive Bayes in a single table scan. In *Proc. ACM CloudDB*, pages 9–16, 2010.