One-pass Data Mining Algorithms in a DBMS with UDFs

Carlos Ordonez *
University of Houston
Houston, USA

Sasi K. Pitchaimalai
University of Houston
Houston, USA

ABSTRACT

Data mining research is extensive, but most work has proposed efficient algorithms, data structures and optimizations that work outside a DBMS, mostly on flat files. In contrast, we present a data mining system that can work on top of a relational DBMS based on a combination of SQL queries and User-Defined Functions (UDFs), debuking the common perception that SQL is inefficient or inadequate for data mining. We show our system can analyze large data sets significantly faster than external data mining tools. Moreover, our UDF-based algorithms can process a data set in one pass and have linear scalability.

Categories and Subject Descriptors

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

General Terms

Algorithms, Languages, Performance, Theory

1. INTRODUCTION

The problem of integrating statistical techniques with a DBMS has received scant attention, mostly in industrial DBMSs. Such problem is difficult due to the mathematical nature of models, the (common) lack of access to the DBMS source code and the internal DBMS architecture. Another, more practical, reason is the comprehensive set of techniques available on external statistical (e.g. R package, SAS) and data mining tools (e.g. WEKA). As a consequence, data mining users or the DBMS itself export data sets as flat files to external data mining programs, thereby going through a performance bottleneck (extracting large tables is slow) and losing fundamental data management capabilities provided by the DBMS (query processing, security, concurrency control, fault tolerance). We should mention some commercial DBMSs provide basic data mining algorithms, but they are difficult to customize, extend or optimize because they are either internally integrated (with direct access to the DBMS file system) or they work on internal files exported with fast bulk interfaces (i.e. they are a black box to the user). More importantly, algorithms in commercial DBMSs do no share a common mathematical foundation (like ours) and most of them require multiple passes over the data set. From a practical standpoint, SQL is and will remain the language to query and analyze large databases. 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) [1, 2, 3, 4]. Unlike most research prototypes and existing tools, our system can directly process relational tables instead of flat files. Moroever, our system can analyze large data sets faster than external data mining tools. All our algorithms are optimized to process the data set in one pass and have optimal linear time complexity. Our system has promise of wide applicability as DBMSs remain the main database management technology, relational data warehouses keep growing, CPUs become faster and secondary storage gets larger and faster.

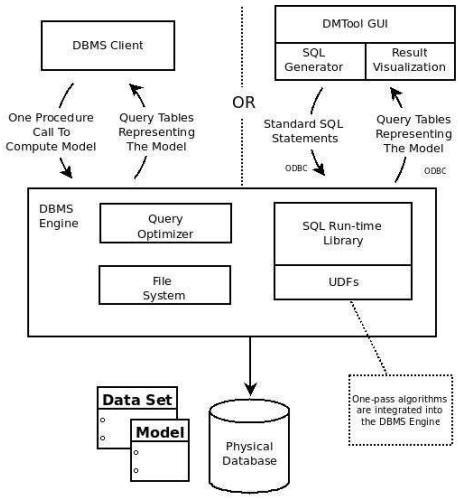

Figure 1: System architecture.

2. SYSTEM DESCRIPTION

2.1 System Architecture

Our system architecture is shown in Figure 1. The input data set, output model matrices and statistical model parameters, are all stored as relational tables. All algorithms can be called directly in the DBMS with stored procedures generating SQL code or by calling UDFs in a SELECT state-

*Supported by NSF grants CCF 0937562 and IIS 0914861.

http://doi.acm.org/10.1145/123456.123456

ment. Alternatively, they can be called from a GUI connecting to the DBMS. Our system avoids exporting large tables. Stored procedures and UDFs are programmed in the host language supported by the DBMS (C,C++,C#). The GUI is programmed in the Java language, ensuring portability. Notice Java speed is irrelevant since the bulk of processing is done by the DBMS. All statistical model tables and intermediate tables (produced by queries and UDFs) can be analyzed with SQL queries.

2.2 Statistical Models and Algorithms

The data set $X = \{x_1, x_2, \ldots, x_n\}$ has n records and d attributes (numeric/categorical). In most models all attributes are numeric (i.e. d is dimensionality), where categorical attributes are mapped to binary dimensions.

Our system provides fundamental statistical models, sharing a common mathematical foundation, which enables common optimizations. Unsupervised models include PCA (to reduce dimensionality; solved with SVD) and K-means clustering (to group similar points). On the other hand, supervised (predictive) models include: linear regression (solved by least squares), and the Naïve Bayes (NB) classifier (estimating joint probability as a product of marginal probabilities). In supervised techniques, X has an additional "target" attribute: a dependent variable Y (numeric in regression) or a discrete (generally binary) "class" attribute G (for Naïve Bayes). In summary, we compute a full spectrum of models.

2.3 Processing with SQL Queries and UDFs

We start by discussing storage and indexing. The data set and model matrices are stored on tables, indexed by specific combinations of matrix subscripts depending on their layout. The system allows two fundamental storage layouts for matrices: horizontal and vertical. The horizontal layout for X is the standard representation which has n rows and d columns. The vertical layout is based on a pivoted table with one attribute value per row (i.e. attribute id, value pairs) and up to dn rows. The vertical layout removes DBMS limitations on the maximum number of columns for high dimensional data and enables efficient manipulation of sparse matrices (i.e. having many zeroes).

Algorithms are programmed with SQL queries or UDFs working on a horizontal or vertical layout, using two sets of parameters. The first parameter set controls statistical model properties. Examples include the number of clusters (k), accuracy tolerance threshold ϵ (i.e. to assess convergence or to stop early), numeric stability, and so on. The second set of parameters is used to manage query and UDF optimizations, which are tailored to each technique (with several common optimizations across multiple models). Our system offers two algorithm flavors: (1) programmed with SQL queries; (2) programmed with aggregate UDFs. SQL queries combine joins, aggregations and arithmetic expressions creating intermediate tables to evaluate matrix equations. SQL queries can be slow to evaluate complex matrix equations (due to joins) and generally require multiple passes over X. On the other hand, algorithms programmed with aggregate UDFs can compute the model in one pass. That is, UDF-based algorithms incorporate incremental model learning instead of iterative behavior. UDFs perform fast matrix processing with arrays in main memory. Algorithms programmed with SQL queries generally require multiple passes over X and join operations, whereas UDFs enable one-pass processing without joins. Notice SQL queries generated by Java (or C++) can evaluate any equation with matrices, no matter how complex. Thus the main reasons to exploit UDFs are faster processing and programming flexibility (i.e. using arrays, data structures in memory and flow control statements). Also, UDFs reduce communication overhead.

We now discuss algorithmic and database optimizations. Sufficient statistics (data set summaries) are an essential ingredient to develop one-pass data mining algorithms. All our models (PCA, K-means clustering, Naive Bayes, linear regression, logistic regression) exploit a family of similar sufficient statistics. In an orthogonal manner, sufficient statistics can be computed on samples, providing fast model approximation, whose accuracy can be controlled by the user. 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): $L = \sum_{i=1}^{n} x_i, Q = \sum_{i=1}^{n} x_i x_i^T$. A fundamental difference with previous research is that our system provides many alternatives to efficiently compute summary matrices. Depending on the model, L and Q can be computed on the entire data set or on multiple subsets and Q can be constrained to be a diagonal matrix (zeroes off the diagonal) or a full matrix. For instance, in K-means L, Q are computed on k data subsets and Q is assumed is diagonal. For PCA and linear regression there is only one set of L, Q and Q is non-diagonal. Summary matrices are efficiently computed with an aggregate UDF or slower, but with better portability, with SQL queries. For all models we developed fast incremental algorithms with aggregate UDFs that can obtain a good approximate model in one pass, being orders of magnitude faster than the standard algorithm; this is particularly difficult to achieve with SQL queries or UDFs. When n is large sufficient statistics can be derived with geometric sampling or bootstrapping techniques (leveraging efficient sampling mechanisms provided by the DBMS), which can get arbitrarily accurate (99%) approximations in much time than a full table scan. For matrix equations evaluated with SQL queries, 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. Whenever possible, queries involving join operations between large tables are rewritten as equivalent queries with fewer or no join operations. Denormalized tables with sufficient statistics avoid joins down the road. Whenever possible, aggregations are pushed (computed before) joins to get smaller tables (i.e. compressing data). Join algorithms are carefully considered. Hash-based joins are the preferred join algorithm on large tables, which nowadays are available in most DBMSs. 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. As a faster alternative, our system exploits aggregate UDFs, which enable maintaining arrays in main memory and incremental model learning. Algorithms programmed with UDFs are generally much faster than SQL queries, exhibit linear scalability and the code is

easier to understand. A small disadvantage about UDFs is that they need to be rewritten for each DBMS, but the underlying parallel scan and aggregation algorithm is the same. Our last optimization worth mentioning is caching, which is automatically applied to sufficient statistics and which can also be applied with data sets that can fit in main memory.

3. SYSTEM DEMONSTRATION

We plan to give an interactive demonstration on a commercial DBMS, running on a portable computer. If Internet is available we can use a remote DBMS server. The user can directly call our algorithms from the DBMS SQL prompt window (i.e. like writing queries) or from a GUI we developed. The database will contain large ($n = 100$K,1M) synthetic data sets (to evaluate performance) and real data sets from the UCI repository (to test accuracy). The same data sets will be also available as flat files to be analyzed by external data mining tools (e.g. Weka, R package) and a fast C++ program (with the same algorithms). The demonstration will flow from a system overview, learning available algorithms and models to technical aspects like storage layouts, SQL queries, UDFs and optimizations. Our demonstration will illustrate the following points: (1) Being able to perform data mining on large data sets truly inside the DBMS (no data record goes out). (2) Exploiting UDFs, queries and stored procedures to evaluate data mining computations instead of extending SQL with new syntax or modifying the DBMS internal subsystems. (3) Efficient computation of accurate models in one pass. Moreover, we will show we can can quickly compute accurate model approximations on large data sets with sampling techniques. (4) Showing our system is much faster than external data mining tools working on flat files. We will even show our UDF-based one-pass algorithms are competitive with C++. (5) Query-like interface by computing a data mining model with one simple stored procedure call (i.e. similar to writing a query). Also, giving the ability to browse the model with queries. (6) Showing exporting large data sets from the DBMS is a bottleneck, even with bulk export utilities, which is counter-intuitive. (7) Highlighting how key mathematical equations are evaluated with UDFs. Also, understanding the impact of query optimizations, turning them on an off. (8) Emphasizing extra benefits like ad-hoc queries, data set transformations and secure access.

In our demonstration the user will be able to experience a complete data mining run, going from choosing a data set to actually querying a model. Initially, the user can get a list of all available models, together with their main procedure call (i.e. a help text). Then the user can see a script with sample procedure calls or the user can type one simple procedure call, specifying model, data set, columns and parameters (if any). Such parameters include optimizations and most parameters have defaults. For supervised 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 will run the algorithm (by submitting the procedure call, like submitting a query), which should finish in a few seconds (for most data sets), The user will be able to verify PCA, K-means clustering, Naïve Bayes, and linear regression can be computed in a single table scan by comparing with a sum() aggregation. The user can optionally exploit sampling to quickly get accurate approximate models on larger data sets (e.g. $n \geq 1$M). More importantly, the user will be able to query model tables, linking them back with the input data set, if necessary. For supervised models the user can evaluate model accuracy on a blind test with an independent set of records (i.e. training and testing). Figure 2 shows two simple stored procedure calls, to compute NB and PCA models as well as queries to browse each model.

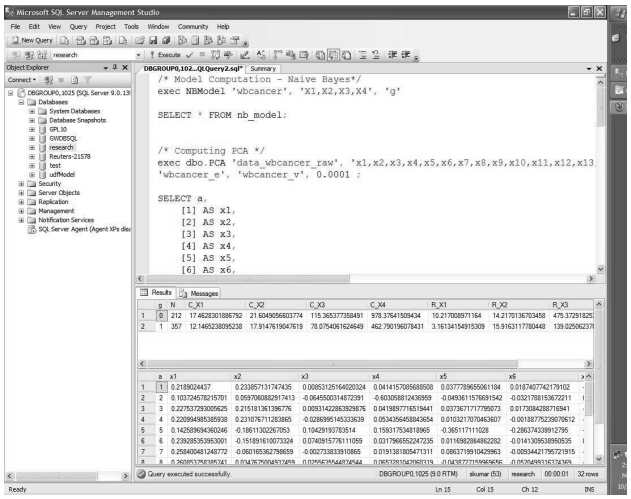

Figure 2: Computing models with simple procedure calls and browsing them with queries in a DBMS.

To test processing performance, the user can compare DBMS processing time with external data mining tools (e.g. Weka, R package) and a C++ program, both working on flat files and computing the same model. UDFs will be shown to be competitive with C++ and much faster than external data mining tools. In addition, the user will be able to understand exporting large data sets, even with bulk utilities, is a bottleneck, making external processing unattractive. The user will be able to understand the impact of optimizations (UDFs, rewritten queries, sampling and caching) turning them on and off. If a user is interested we can go deeper, providing an overview of internals of our system (modules, tables, configuration files) and explaining representative aggregate UDFs and SQL queries. We will explain how UDFs avoid joins, enable incremental model computations and push matrix processing into main memory.

4. REFERENCES

[1] 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.

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

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

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