

Embedded Predictive Modeling in a Parallel Relational Database

A. Dorneich
IBM Software Group
Boeblingen, Germany
dorneich@ de.ibm.com

R. Natarajan
IBM Thomas J. Watson Research Center
P.O. Box 218, Yorktown Heights NY
nramesh@ us.ibm.com

E. Pednault
IBM Thomas J. Watson Research Center
P.O. Box 218, Yorktown Heights NY
pednault@ us.ibm.com

F. Tipu
IBM Thomas J. Watson Research Center
P.O. Box 218, Yorktown Heights NY
fateh@ us.ibm.com

ABSTRACT

A methodology for embedding predictive modeling algorithms in a commercial parallel database is described; specifically, the parallel editions of IBM DB2 Universal Database, although many aspects of the overall approach can be used with other commercial parallel databases. This parallelization approach was implemented in the Version 8.2 release of DB2 Intelligent Miner Modeling to support a new predictive modeling algorithm called Transform Regression. This database-embedded mining algorithm provides all the usual benefits, including easier integration into large enterprise applications, the ability to perform entire data mining workflows directly from an SQL-based programming interface, reduced data transfer costs between the database and the data mining application, and faster, parallel data access during query processing. However, in addition to these benefits, a significant part of the data mining computations are also parallelized without the use of any sophisticated parallel programming constructs, or any specialized message passing and parallel synchronization libraries.

Categories and Subject Descriptors

H.2.8 [Database Applications]: Data Mining

General Terms

Algorithms, Design, Performance

Keywords

Data Mining, Parallel Databases, Predictive modeling, Embedded Analytics, Transform Regression.

1. Introduction

A significant inhibitor to the use of data mining in large enterprise applications is the expertise required to learn specialized data mining software, and the difficulty in obtaining a robust suite of data mining

algorithms that can be robustly integrated into the workflow of large applications. In addition, many enterprise applications need to analyze data that is primarily stored and aggregated in relational databases and data warehouses, while specialized data mining software often requires data to be stored either in-memory or in a software-specific persistent workspace (e.g., flat files). Such requirements lead to excessive overheads for data transfer and format conversion when transferring data and results between the databases and external data mining software. Furthermore, many data mining algorithms are not yet designed to scale to the massive data sets required in enterprise applications. By contrast, there has been much progress in optimizing parallel data access in database query processing, and this feature is now supported in many commercial databases. Therefore, database-embedded data mining is not only useful for improving workflow management and reducing data-transfer overhead, but also because it enables the parallel query optimizing capabilities of databases to be exploited in order to obtain better data locality and parallelism in the data mining computations.

The database-embedding methodology described in this paper is particularly well-suited for the new predictive modeling algorithm called Transform Regression (described elsewhere [1]), in the Version 8.2 release of IBM DB2 Intelligent Miner Modeling. An important feature of this implementation is that the parallelization scheme does not require any specialized, machine-dependent libraries for message passing or parallel synchronization. Instead, it relies solely on the underlying parallelization mechanisms provided by the database query optimizer. The degree of parallelization is thus completely under the control of the database administrator using the same control options that are available for query parallelization.

The outline of the paper is as follows. In Section 1, the related previous work in embedded data mining is reviewed and the SQL/MM and PMML standards that were used for the current implementation are described. In Section 2, the generic methodology for implementing embedded data mining kernels, which is based on the implicit parallelism of user-defined functions, is described. Section 3 describes aspects of the Transform Regression algorithm implemented in the DB2 IM Modeling product for which the present methodology is well suited. Section 4 discusses some of the experiences with the scalability and performance of the parallel implementation, and the likely impact of future database enhancements. Section 5 concludes with a summary of the paper.

1.1 Background and Review

The primary interest is in statistical or machine-learning-based data mining algorithms such as classification, clustering, and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'06, April, 23-27, 2006, Dijon, France.

Copyright 2006 ACM 1-59593-108-2/06/0004...\$5.00

prediction [2]. These algorithms typically comprise two steps, *modeling* and *scoring*. The *modeling* step is used to build and validate models from training data, while the *scoring* step is subsequently used to make inferences on new data using these models. Depending on the amount of data, both of these steps may be computationally expensive, but it is typically the *modeling* step that presents algorithmic and scaling challenges for massive data sets.

Many data mining *modeling* algorithms do not scale well for large data sets and must therefore rely on data sampling for computational tractability. However, the accuracy and quality of the resulting models is usually improved by using the largest possible training data sets, with the improvements being reflected in better model prediction accuracy on validation data and smaller variances of the model parameter estimates.

Most data mining software will typically require that training data be extracted from relational database prior to the computations for the *modeling* step. In other, more database-aware data-mining applications, client programs establish database connections and retrieve training data records using the cursor interface to the result sets of SQL *select* queries. Sometimes, these data records may be cached as disk files, which is particularly useful whenever multiple data passes are required or whenever the data must be reorganized in a more convenient format for the algorithm [3]. A further improvement in performance is obtained by reconfiguring a client application as a stored procedure running on the database server, which reduces the network data transfer costs considerably. An empirical performance evaluation of these different implementations for a data mining algorithm can be found in [4].

Many database vendors now support integrated mining capabilities on their platforms, with a small class of robust and widely-used data mining kernels being implemented as stored procedures. In addition, all the data mining tables and workflow objects are also managed within the database. Vendors will typically provide facilities to store, retrieve, and manage collections of historical data mining models and model results in the database, which is advantageous for large-scale data mining operations [5], [6].

In many data mining applications, the relational data in data warehouses is often already stored in a form that is suitable for mining. Data warehouses are typically created for analytical purposes after performing data cleansing, verification, transformation, aggregation, merging, and feature extraction operations on raw data feeds from operational databases or from external data sources. In addition, data warehouses are often implemented in parallel database systems to obtain scalable storage and query performance for large data tables. Therefore, wherever possible, database-embedded data mining algorithms should take advantage of the existing data organization and parallel environments to improve efficiency and performance.

For example, the DB2 Universal Database [7] supports two different kinds of parallel query access on single or multi-partition databases. The INTRA parallel mode, which is available on all versions of DB2, uses multiple processes or threads within a single partition and is therefore well suited for a shared-memory parallel (SMP) system. The INTER parallel mode, which is supported in the Extended Enterprise Edition of DB2, uses separate processes on each node partition of partitioned database and is therefore suitable when the data partitions reside on separate nodes of distributed, shared-nothing parallel systems, such as the IBM SP2 [8]. Furthermore, both parallel modes are possible when the individual nodes of a multi-partitioned distributed parallel system are themselves SMP nodes.

In the INTRA mode, a DB2 register parameter called *current_degree* can be dynamically set to specify the number of

threads that will participate in the access plan for a specific query. In the INTER mode, a given database table or materialized view can be partitioned across a set of processor nodes specified in a “node configuration” file for the database instance (a partitioning key defined at table loading time is used to hash into a table to give the partition index for each row). All standard SQL queries using standard built-in functions can take advantage of parallelism without any special modification, and the query optimizer ensures that the parallel synchronization and inter-processor communication overheads are minimized for best query performance. For simple queries that access every record in a database table or materialized view, the INTRA mode can use dynamic load balancing via the “bag-of-tasks” approach with an appropriately-chosen task granularity comprising a small set of records. The INTER mode uses a static load balancing via the explicit and user-defined partitioning key and control over the distribution of data via that key.

1.2 Overview of SQL/MM standards

The use of de-facto standards is a major enabler for embedding mining kernels into databases. An example is the ISO SQL/MM Data Mining standard, which is an SQL-based API for specifying and executing data mining tasks in a database [9].

The SQL/MM API defines a set of user-defined types (UDT’s) and associated methods by which a data mining run can be defined and executed. Schematically, these UDT’s include for example, MiningData (which represents the table containing the training or test data set, and the table columns along with their SQL type specification), LogicalDataSpec (which defines mining types and valid ranges for the fields used for mining, and name mappings and taxonomies for certain field values), Settings (which sets up mining features and kernel-specific information such as the response variable for prediction), BuildTask (which gathers all the metadata and contains all the information to compute a specific data mining model), TestTask (which validates a mining model on new data), MiningModel (which contains all the model details for querying and model export), and ModelResult (which contains the results from applying a computed model to a given data set). This API, which can be implemented as a database extender, allows the entire data mining process to be carried out from an SQL interface, and makes it possible for data mining to be easily incorporated into the workflow of database-centric enterprise applications.

1.3 Related Work

The implementation of well-known mining algorithms (such as association rules, k-means clustering and decision trees) for database-resident data has been widely studied. As mentioned earlier, many of these implementations are client applications or stored procedures that use a cursor interface to the result set of an SQL *select* query to obtain the data.

A different approach for embedded data mining kernels where parts of the algorithm are implemented as user-defined functions (UDF’s) is described for association rules in [10], [11]. This approach avoids the process context switching and data copying overheads in the stored procedure implementation. These UDF’s can be implemented in either *fenced* mode (i.e., as a separate process or thread) or in *unfenced* mode (i.e., within the address space of the database). In the case of association rules, performance efficiencies are obtained only in the *unfenced* mode, with the results showing a factor of two improvement in performance.

Other approaches to database embedding include the use of data aggregation queries to directly obtain all relevant sufficient statistics required at each step of a mining algorithm [12], [13]. For example,

[14] describes a decision tree algorithm for which, at each step of model refinement, a complex database query returns the entire set of bivariate contingency tables involving the target feature for all potential node-splits of the current tree. These sufficient statistics are then directly used to determine the best node splits without the decision tree algorithm itself having to retrieve and process individual data records. The expressivity of these aggregate queries is improved by defining new SQL operators [15], which allow an entire collection of low-dimensional aggregations to be computed without specifying an exhaustive list of predicates. In addition, efficient query plans for these new operators can be generated either by the database query optimizer or by a specialized external pre-optimizer [16]. Alternatively, these aggregate queries can also be implemented using user-defined aggregates (UDA) or using scratchpad user-defined functions. However, UDA's are not yet natively supported in many commercial databases [17], [18].

2. Parallel Implementation Details

2.1 Background

Many commercial parallel databases provide built-in parallel single-column aggregation functions for a variety of common operators (such as MAX, MIN, COUNT, AVG and SUM), but in general they do not provide an API for implementing general-purpose, multi-column, parallel user-defined aggregates [19]. Nevertheless, UDA's with these properties can be implemented using the API for scratchpad UDF's, which on DB2 can run in either INTRA or INTER parallel modes, or in the combined mode using independent scratchpads for each separate thread or thread-within-partition involved. Various strategies can be used to globally aggregate the individual results in the scratchpad at the end of a data scan. For example with DB2, the communication for this final aggregation could be performed using shared memory primitives for INTRA mode, or using message passing primitives for INTER mode. However, both of these approaches would require special-purpose parallel programming libraries that might not be supported across all combinations of machine architecture, operating system, and database platforms. We have instead opted for a simpler but highly robust and portable approach that uses a shared file system for communication of aggregation results.

2.2 Implementation details

The parallel implementation is based on an abstract data mining kernel class, with the following four interface methods: **BeginDataScan**, **ScanDataPoint**, **MergeWithKernel** and **DataScanComplete**. These four methods, which must be implemented by all concrete derived kernel objects, makes it possible to compute a large class of data mining models that satisfy the following two conditions. First, it should be possible to divide the model training computations into 2 phases; the first phase consisting of the computation of the model sufficient statistics from the training data, and the second phase consisting of the model parameter computations based on the sufficient statistics obtained in the first phase. Second, it should be possible to combine the model sufficient statistics of two independent subsets of the training data to obtain the model sufficient statistics of the combined subsets without any loss of information. These properties are shown below for the case when the mining kernel is linear regression with feature selection. The properties likewise hold for the more complex transform regression mining kernel that is described later in the paper.

For the concrete derived objects of the mining kernel class, the **BeginDataScan** method initializes the kernel data structures for accumulating the model sufficient statistics. The **ScanDataPoint**

method updates the model sufficient statistics as it is supplied with a stream of data records. The **MergeWithKernel** method combines the sufficient statistics of two models that have seen disjoint subsets of the training data records. The **DataScanComplete** method carries out the model parameter computations from the accumulated sufficient statistics, and updates internal data structures as needed to set up for additional data scans if they are needed.

In the serial version of a model training algorithm, the client program or the stored procedure will instantiate a mining kernel object, then call **BeginDataScan** to initialize the object, then loop over all the records in the database (using the cursor interface) passing each individual record to **ScanDataPoint**, and finally call **DataScanComplete** to update the model and to decide whether an additional data scan is required. Some algorithms (like decision trees) may require multiple iterations over the data to converge on a final model; hence, the **DataScanComplete** method returns a flag to indicate whether training is now complete or whether additional data scans are required.

Our approach to parallelizing algorithms that are implemented using the above interface is to have the **ScanDataPoint** method called from within a scratchpad UDF that is invoked from an SQL query that is invoked from within a stored procedure. The query optimizer can then automatically parallelize the data-scanning portion of the algorithm as a function of how the database is configured. The **BeginDataScan** and **DataScanComplete** methods are not parallelized and are called from within the stored procedure. This parallelization scheme requires that the mining kernel object be exchanged between the stored procedure and the UDF parameter list—i.e., between a program variable and a database binary large object (BLOB) type—once at the beginning and once at the end of each data scan. This transfer is achieved through an abstract class with a Reader/Writer interface to the BLOB type, with the actual implementation using an explicit BLOB for forward communication from the stored procedure to the UDF, and the reverse communication using a reference to a file in the shared file-system containing the BLOB (as below).

For scratchpad UDF's in DB2, each function call includes a parameter flag that is set to indicate whether it is the FIRST, NORMAL, or FINAL call. In the FIRST call, the mining kernel object is allocated with its pointer stored in scratchpad memory, and the object is then initialized by reading from the input BLOB. The FIRST call and each subsequent NORMAL call then updates the state of the kernel object. In the FINAL call, the updated mining kernel object is written as an output BLOB into a temporary binary file. The filename for the return BLOB is the return value of the FIRST call to the UDF, with the return value of each subsequent NORMAL call being a database NULL. These filenames are then returned to the stored procedure as the result set of the SQL query that was invoked to perform a data scan. The stored procedure then uses the filenames to rematerialize the final (parallel) states of the (distributed) mining kernel from the result BLOBs. In order to avoid race conditions, 0-byte shadow files are used to signal the points at which the various parallel invocations of the UDF have finished writing their result BLOB files during FINAL call. We note that it is not possible to explicitly return a BLOB in the reverse communication from the UDF because its value is finalized only after the last record has been seen by the UDF. The UDF becomes aware that it has seen the last record only in the subsequent FINAL call, and any value returned by the FINAL call is ignored by the query processor (unlike the FIRST and NORMAL calls).

The above approach allows the query optimizer to automatically parallelize data scans because each separate database thread or partition invokes a separate instance of the UDF with its own

scratchpad memory, which then returns a unique and private reference to a file containing the BLOB for the kernel updates for the corresponding subset of data records. These filename references can be stored in a temporary database table or returned to the stored procedure as the result set of the parallel UDF query. In either case, a sequence of calls to the **MergeWithKernel** is used to aggregate all the model sufficient statistics from each subset. Finally, **DataScanComplete** method is invoked on the aggregated kernel object to complete the parameter estimation calculations for the current data scan and the set up for an additional data scan should one be needed. These sequential steps are illustrated schematically in Figure 1.

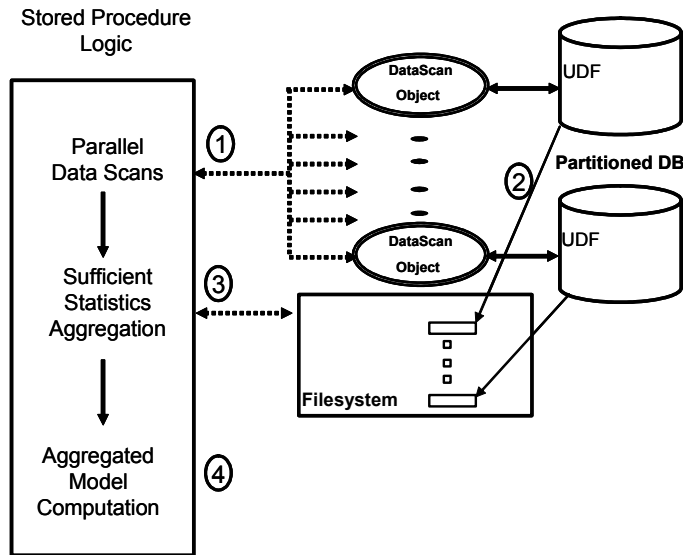


Figure 1: Schematic of parallel data mining implementation

Note that the **MergeWithKernel** method is currently invoked in serial fashion from within the stored procedure. For databases that provide full support for user-defined aggregates (UDA's), the **MergeWithKernel** method could instead be incorporated into the merge/combine function of a UDA, in which case the merge operations could also be performed in parallel at the discretion of the query optimizer.

The processing steps for the case when the mining kernel is linear regression is very similar to that for the serial, file-based algorithm described in [20]. The sufficient statistics for computing the regression model are the training data mean and (non-normalized) covariance. These quantities, which would be allocated and initialized in **BeginDataScan**, are updated in **ScanDataPoint** using special recursions, and this step would be performed in parallel in the UDF's as described above. The combining of two separate instances of this mining kernel with sufficient statistics obtained from disjoint sets of training records, which is required for the parallel merge, is also computed using special update formulas in **MergeWithKernel**. Finally, the regression coefficients are computed by the normal equations methods in **DataScanComplete**, using feature selection and cross-validation for obtaining stable estimates of the coefficients [20].

This approach can be directly extended for computing ensembles of regression models; for example, as required in ProbE linear regression trees [21] and in the transform regression algorithm described below. From a performance perspective, for either linear regression or the associated ensemble modeling techniques, the complexity of the **ScanDataPoint** method is $O(NJ^2)$, the

MergeWithKernel method is $O(J^2)$, and the **DataScanComplete** method is $O(J^3)$, where N is the number of records in the database and J is the number of features. Since typically, $J \ll N$, and since I/O and database access overheads contribute to large constant factors in the lower order terms, the overall cost tends to be dominated by the fully parallel operations in **ScanDataPoint**. Therefore, given a suitably load-balanced partitioning of the training data tables, this approach always yields very good parallel speedups.

3. Transform Regression Kernel

The transform regression algorithm that is implemented in DB2 Intelligent Miner Modeling V8.2 generalizes conventional linear regression by introducing feature transformation mechanisms that automatically handle nonlinearities and cross-product interactions. In so doing, it provides a general-purpose, nonlinear regression modeling capability that is highly suited for automated predictive modeling applications.

Transform regression [1] combines aspects of gradient boosting methodology [22], [23] with aspects of generalized additive models [24] and aspects of ProbE linear regression trees [21]. Like generalized additive models (GAM's), input features are nonlinearly transformed in order to take nonlinear relationships into account. However, unlike conventional GAM methods [24], an alternative model-fitting approach based on Friedman's gradient boosting methodology is used for iterative model refinement. In addition, unlike conventional GAM's, ProbE linear regression tree methods are used to construct piecewise-linear feature transformations. While this has many benefits from the modeling perspective [1], a major implementation benefit is that the feature transformation calculations can be parallelized because the ProbE linear regression tree algorithms are implemented using the interface methods described above. The transformation refinements of all features are also computed at each iteration of the Transform regression algorithm, therefore a large number of piecewise-linear regression models are required to be computed to derive the transformed features. The final model converges in a small number of iterations (typically no more than 4 or 5) for many problems we have studied. The number of iterations is comparable to the full ProbE linear regression tree implementation [21], but each iteration is much computationally faster. In addition, all the data is required in each iteration, and therefore the algorithm stays load-balanced, whereas in regression tree algorithms, the data can get fragmented in later iterations because of early convergence along some branches in the regression tree.

4. Performance Review

Several data sets were used for confirming the correctness and consistency of the serial and parallel versions of the transform regression algorithm, and for code optimization and performance improvement. One set of performance measurements were carried on an IBM xSeries 450 machine consisting of four Itanium II, 64 bit, 1.4 GHz CPU's with 4GB RAM and 200 GB disk, running the Red Hat Linux 3 operating system. For example, with the 1998 KDD cup data set [25] consisting of 95412 records and 481 features, the parallel implementation scaled quite well and on 4 CPU's there was a speedup of about 3.4 when compared to 1 CPU for the entire transform regression modeling step.

For the same data set, the single node performance of the serial, stored-procedure version was about 2.8 times faster than the parallel UDF based version. It is instructive to examine the reasons for this

performance disparity. First, for this modestly-sized data set (file size around 120 Mbytes), the serial version triggers an option that allows the data to be cached in temporary binary files during the first data scan, so that subsequent data scans are considerably speeded up. However, this caching approach, which is possible only in the serial code, is unsuitable for larger data sets for which the temporary disk space may be insufficient. Furthermore, except in the first pass, this approach does not benefit from the multi-threaded database I/O (obtained by setting the *current_degree* parameter in DB2) which may provide performance that is comparable to binary file I/O. Second, a large portion of the overhead in the parallel version comes from the use of a built-in DB2 function called *rec2xml* [7] for packaging a variable number of data fields into a single, generic XML-style string for the data record, which is passed as a parameter to the parallel UDF and parsed to extract the individual field values. Although the parsing of the *rec2xml* string can be implemented reasonably efficiently, alternative built-in functions that provide the data records without requiring string operations or format conversions in the UDF would remove most of the performance disparity between the single node serial and parallel implementations.

5. Summary

The embedding of data mining kernels in commercial databases has several potential benefits for the widespread adoption of data mining within enterprise and business applications.

A direct SQL-based interface to the data mining modeling and scoring capability makes it easier for developers to integrate these algorithms into the workflow of end-user applications without the complexity and data transfer overheads of using a separate, external data mining tool. Furthermore, these embedded data mining kernels can directly exploit the numerous built-in capabilities of databases. These include for example, optimized functions for random sampling directly from large tables using fewer disk accesses and lower data transfer cost [26]. Finally, as described in this paper, it is possible to transparently derive the benefits of parallel and distributed computing, both in the data access and query processing, as well as in the computational parts of the data mining algorithm.

The management of the data mining process within the database is greatly aided by the development of the standards-based SQL/MM API, which enables the data mining task and data specifications, as well as the model results, to be stored, modified, and queried in the database using an SQL interface. Model results can also be materialized in a portable XML format called PMML, which allows the computed models to be redeployed on the scoring engines of different database platforms.

This paper has described an approach for embedded, scalable parallel implementation of data mining algorithms using database user-defined functions and stored procedures. The approach takes full advantage of the parallel query scheduler and optimizer without requiring any other specialized parallel programming software. Specifically, the Transform regression algorithm, a new predictive modeling algorithm for DB2 Intelligent Miner modeling, benefits greatly from this approach, as described in the paper.

6. References

- [1] E. Pednault, "Transform Regression and the Kolmogorov Superposition Theorem," IBM Research Report RC 23227, IBM Research Division, Yorktown Heights, NY 10598, 2004.
- [2] C. Apte, B. Liu, E. P. D. Pednault and P. Smyth, "Business Applications of Data Mining," *Communications of the ACM*, Vol. 45, No. 8, August 2002.
- [3] J. Shafer, R. Agrawal and M. Mehta, "SPRINT: A Scalable Parallel Classifier for Data Mining," *Proceedings of the 22nd International Conference on Very Large Databases*, 1996, 544-555.
- [4] S. Sarawagi, S. Thomas and R. Agrawal, "Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications," *Data Mining And Knowledge Discovery*, Vol. 4, pp. 89-125, 2000.
- [5] G. Saarevirta, "Operational Data Mining," *DB2 Magazine*, Vol. 6, (2001).
- [6] A. Netz, S. Chaudhuri, J. Bernhardt, and U. Fayyad, "Integration of data mining and relational databases," *Proceeding of the 26th Conference on Very Large Databases*, 2000, pp. 719-722.
- [7] The IBM DB2 Universal Database V8.1, <http://www.ibm.com/software/data/db2>, 2004.
- [8] D. Chamberlain, *A Complete Guide to the DB2 Universal Database*, Morgan Kaufman, San Francisco CA, 1998.
- [9] ISO/IEC 13249 Final Committee Draft. Information Technology – Database Languages – SQL Multimedia and Application Packages. <http://www.iso.org>, 2002.
- [10] R. Agrawal and K. Shim, "Developing Tightly-Coupled Data Mining Applications on a Relational Database System," *Proceedings Second International Conference on Knowledge and Data Mining*, pp. 287-290, 1996.
- [11] K. Rajamani, A. Cox, B. Iyer and A. Chadha, "Efficient Mining for Association Rules with Relational Database Systems," *Proceedings of the 1999 International Symposium on Database Engineering and Applications 1999*, pp. 148-155.
- [12] G. Graefe, U. Fayyad and S. Chaudhuri, "On the efficient gathering of sufficient statistics for classification from large SQL databases," *Proceedings Fourth International Conference on Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, 1998, pp.204-208.
- [13] M. Wang, B. Iyer, and J. S. Vitter, "Scalable Mining for Classification Rules in Relational Databases," *Proceedings of the 1998 International Symposium on Database Engineering and Applications 1998*, pp. 58-67.
- [14] S. Chaudhuri, V. R. Narsayya and S. Sarawagi, "Efficient evaluation of queries with mining predicates," *Proceedings of the 18th International Conference on Data Engineering*, IEEE Computer Society, 2002, pp. 529-540.
- [15] A. Hinneburg, W. Lehner, and D. Habich, "COMBI-Operator: Database Support for Data Mining Applications," *Proceedings of the 29th Conference on Very Large Databases*, 2003, pp. 429-439.
- [16] S., Nestorov and S. Tsur, "Integrating Data Mining with Relational DBMS: A Tightly-Coupled Approach," *Fourth International Workshop Next Generation Information Technologies and Systems*, Zikhron-Yaakov, Israel, pp. 295-311, 1999.
- [17] H. Wang and C. Zaniolo, "Using SQL to build new aggregates and extenders for object-relational systems," *Proceedings for the 26th International Conference on Very Large Data Bases*, Cairo, Egypt, 2000, pp. 166-175.
- [18] K. Stuzle, "User Defined Aggregate Functions in the IBM DB2 Database," IBM DeveloperWorks, <http://www-128.ibm.com/developerworks/db2/library/techarticle/0309stolze/0309stolze.html>, 2003.
- [19] M. Jaedicke and B. Mitschang, "On Parallel Processing of Aggregate and Scalar Functions in Object-Relational DBMS," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seattle, WA, 1988, pp. 379-389.
- [20] C. Apte, R. Natarajan, E. Pednault, F. Tipu, "A Probabilistic Framework for Predictive Modeling Analytics," *IBM Systems Journal*, v. 41(3), 2002.

- [21] R. Natarajan and E. P. D. Pednault, "Segmented Regression Estimators for Massive Data Sets," *Proc. Second SIAM Conference on Data Mining*, Crystal City VA, 2002.
- [22] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of Statistics*, v. 29, 2001, 1189-1232.
- [23] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics and Data Analysis*, v. 38, 2002, 367-378.
- [24] T. Hastie and R. Tibshirani, *Generalized Additive Models*, Chapman and Hall, New York, 1990.
- [25] KDD Cup 1998, <http://www.kdnuggets.com/meetings/kdd98/kdd-cup-98.html>, 1998.
- [26] P. J. Haas and C. Konig, "A bi-level Bernoulli scheme for database sampling," *Proceedings of the 2004 ACM SIGMOD Conference on the Management of Data*, 2004, pp. 275-286.