Database Systems Research on Data Mining

Carlos Ordonez *
University of Houston
Houston, USA

Javier García-García
Universidad Nacional Autónoma de México
Mexico City, Mexico

ABSTRACT

Data mining remains an important research area in database systems. We present a review of processing alternatives, storage mechanisms, algorithms, data structures and optimizations that enable data mining on large data sets. We focus on the computation of well-known multidimensional statistical and machine learning models. We pay particular attention to SQL and MapReduce as two competing technologies for large scale processing. We conclude with a summary of solved major problems and open research issues.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications—*Data mining*

General Terms

Algorithms, Languages, Performance, Theory

1. INTRODUCTION

DBMSs represent the dominating technology to manage and query structured data represented by tables. On the other hand, there is an explosion of semistructured data on the Internet represented by documents, web pages, files and so on, which are stored and queried by search engines. In both worlds, a major challenge is to efficiently analyze large data sets, where data mining plays a central role. Data mining research is extensive [1]: there exist many efficient algorithms, data structures, and optimizations to analyze large data sets. However, most of them work on flat files. This is because most data mining algorithms perform complex and demanding mathematical computations, which make their integration with a DBMS difficult. We present a review of processing alternatives, storage mechanisms, algorithms, data structures and optimizations for data mining on large data sets. We pay particular attention to SQL [1, 2] and MapReduce [4] as two competing technologies.

2. PROCESSING ALTERNATIVES

We distinguish three fundamental solutions to compute data mining models on large data sets, going from external processing on flat files to extending the DBMS internal source code: (1) Analyzing large data sets outside a DBMS, creating an efficient program in a standard programming language (e.g. C++, Java) that works directly on flat files. (2) Performing processing on top of the DBMS with SQL code generation [2] and User-Defined Functions (UDFs) [3]. (3) Modifying and extending the DBMS source code, yielding algorithms tightly coupled to the DBMS.

Alternative (1) is the most flexible to develop new algorithms and optimizations and hence it is the most popular. However, exporting or importing large data sets is a bottleneck, even with bulk utilities. Statistical and data mining packages working on exported flat files fall into alternative (1). Alternative (2) is based on computing model equations combining SQL queries and User-Defined Functions, if available. Alternative (2) leverages DBMS functionality, but it provides less programming flexibility and it is slower due to DBMS overhead and architecture. Some data mining packages have the capability of pushing some demanding computations into a DBMS with SQL queries or UDFs. Alternative (3) is the preferred choice for commercial DBMSs, because it can match speed of alternative (1) and gives DBMS developers programming flexibility. Nevertheless, it hinders users from adding new algorithms or improving existing ones.

Both SQL and MapReduce exploit parallel processing by partitioning the input data set. Therefore, both technologies benefit from even load balancing. However, SQL requires relational tables as input, whereas MapReduce can work on files. In SQL a relational operator is generally evaluated with data parallelism and query evaluation becomes a sequence of physical operators acting on tables. UDFs extend SQL by allowing efficient computation of mathematical equations in a high-level programming language (e.g. C, C++) with arrays and flow control statements (if-then-else, for, while). Unfortunately, their features vary widely across DBMSs. On the other hand, in MapReduce functions are programmed in a high-level programming language. MapReduce has two phases: in the first phase a mathematical computation is distributed and evaluated in parallel, whereas in the second phase partial results are combined or summarized. MapReduce can be categorized into each alternative, depending on the level of integration with the DBMS.

3. DATA MINING ALGORITHMS

Even though it is difficult to generalize properties of data mining algorithms, they typically have the following characteristics: (1) Algorithms to compute models have an iterative behavior, which requires performing multiple passes over the data set. (2) The input data set contains records

*Supported by NSF grants CCF 0937562 and IIS 0914861.

http://doi.acm.org/10.1145/1807167.1807335

with a combination of numeric (dimension) and categorical (discrete) attributes. Data set dimensionality makes the problem mathematically more difficult, whereas data set size can significantly increase computation time due to access to secondary storage. (3) Models are computed and represented with vectors, matrices and histograms. Most models require complex mathematical calculations.

We explain algorithms in two groups: unsupervised (clustering, dimensionality reduction, association rules) and supervised (classification, regression, feature selection, neural nets). Association rule discovery and OLAP are deeply related problems solved by combinatorial search algorithms. Bayesian models, a trend in statistics, are generally solved with stochastic methods (MCMC methods), which in general require exploring a huge probabilistic space and many more iterations than classical methods.

4. STORAGE AND OPTIMIZATIONS

We consider two major database systems research aspects: storage and optimizations. We further categorize them as general optimizations (algorithmic, hardware) and specific optimizations (in SQL or MapReduce).

Storage mechanisms include data layout on disk and indexing. Storage is analyzed for the data set, model matrices and intermediate results. There exist three main storage layouts: row-based, column-based and cell-based. Cell-based storage works at the intersection of a row and a column identifiers (i.e. one value per record). Row-based storage is the default storage for large data sets in a DBMS, whereas cell-based is the most common in search engines. Column-based storage is a more recent alternative to support analytics. Indexing is generally defined on vector and matrix subscripts. Most algorithms assume row-based storage.

There are two kinds of general optimizations: (1) algorithmic, which decrease mathematical operations, reduce I/O, exploit parallel processing or accelerate convergence. (2) exploiting hardware, like large RAM, cache memory, multicore CPUs, alternative technologies to disk storage and so on. Algorithmic optimizations have received more attention, given their generality. Sufficient statistics represent summaries of the data set, which help reducing the number of passes and developing incremental algorithms with faster convergence. Sufficient statistics have been widely used in clustering, PCA, regression and decision trees. Data stream mining represents an extreme case requiring a single pass. However, convergence to a stable solution and parallel processing become more difficult. Sampling allows model computation in main memory with small subsets of the data set, but error in estimations needs to be controlled, especially with skewed distributions. Sampling has been key to accelerate clustering, decision trees and association rules. Hardware optimizations have been a less popular direction because they generally require programming in a low level (assembly) language and they are architecture dependent (less general). Caching matrices or representative subsets of the data set minimize access to secondary storage.

We now discuss specific optimizations in SQL and MapReduce. SQL query optimizations include forcing hash–based or merge-sort joins to join large tables, using clustered indices for efficient join and aggregation, denormalization to avoid joins and reduce I/O, and pushing aggregation before joins to compress intermediate tables. Aggregate UDFs can help computing summary matrices on the entire data set in one pass. Quadratic sufficient statistics computation has been pushed into the DBMS to accelerate PCA and linear regression. Table UDFs return tables instead of values and enable data stream-like processing. Extending SQL with new clauses is not a promising alternative, because SQL syntax is extensive and access to the DBMS source code is required. On the other hand, in MapReduce sometimes it is better to delay the reduce phase so that multiple jobs can exploit the same map phase. Another important MapReduce optimization is to avoid parsing text files, by reading binary files with a simple structure. Full scans on large files can be avoided by following naming conventions on file names for data subsets based on selection predicates or dates.

5. CONCLUSIONS

Data mining research is extensive, but most algorithms and techniques work on flat files outside a DBMS. The storage layout is related to the I/O pattern and optimized memory usage. Reducing the number of passes over a large data set, exploiting summary matrices, developing incremental algorithms, sampling for model approximation and creating data structures for indexing or summarization remain fundamental optimizations. SQL and MapReduce are two competing technologies for data mining on large data sets, both based on automatic data parallelism on a shared-nothing architecture. SQL and UDFs provide less programming flexibility than MapReduce. Exporting or importing large data sets is a bottleneck in SQL, whereas MapReduce allows faster load and processing. When performing data mining inside a DBMS the user can enjoy querying, but also security and recovery. Most importantly, the export bottleneck can be avoided.

There are many research issues. SQL and MapReduce can be combined as recent work has shown. Incremental algorithms are difficult to develop in SQL or MapReduce. DBMSs should provide more efficient mechanisms to export data sets, especially when SQL or UDFs are not acceptable. Column stores can accelerate data mining, especially for high dimensional models. From the mathematical side, the list is extensive. Hidden Markov Models, Support Vector Machines, time series, non-linear regression and neural networks remain difficult to compute with SQL or MapReduce. Adapting Bayesian methods to work with SQL or MapReduce is an ambitious goal.

6. REFERENCES

[1] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2006.

[2] 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.

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

[4] M. Stonebraker, D. Abadi, D.J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. MapReduce and parallel DBMSs: friends or foes? *Commun. ACM*, 53(1):64–71, 2010.