# Data Mining and Database Systems: Where is the Intersection?

Surajit Chaudhuri
Microsoft Research
Email: surajitc@microsoft.com

## 1 Introduction

The promise of decision support systems is to exploit enterprise data for competitive advantage. The process of deciding what data to collect and how to clean such data raises nontrivial issues. However, even after a data warehouse has been set up, it is often difficult to analyze and assimilate data in a warehouse. OLAP takes an important first step at the problem by allowing us to view data multidimensionally as a giant spreadsheet with sophisticated visual tools to browse and query the data (See [3] for a survey). Data Mining promises a giant leap over OLAP where instead of a power OLAP user navigating data, the mining tools will automatically discover interesting patterns. Such functionality will be very useful in enterprise databases that are characterized by a large schema as well as large number of rows.

Data Mining involves data analysis techniques that have been used by statisticians and machine learning community for quite some time now (generically referred to data analysts in this paper). This raises the question as to what role, if any, database systems research may contribute to area of data mining. In this article, I will try to present my biased view on this issue and argue that (1) we need to focus on *generic* scalability requirements (rather than on features tuned to specific algorithms) wherever possible and (2) we need to try to build data mining systems that are not just scalable, but "SQL-aware".

## 2 Data Mining Landscape

We can categorize the ongoing work in data mining area as follows:

- *Inventing* new data analysis techniques
- *Scaling* data analysis technique over large data sets

### 2.1 Inventing New Data Analysis Techniques

In my opinion, discovery of new data analysis technique is to a large extent an expertise that requires insight in statistical and machine learning and related algorithmic areas. Examples of well-known techniques include decision-tree classification, clustering (see [5] for an overview of known techniques). Innovating in this space requires establishing statistical merit of a proposed technique and appears to have little interaction with database system issues. On a more pragmatic note, we seem to have a large number of established techniques in this space.


**Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**

## 2.2 Scaling Data Analysis Techniques

In contrast to inventing new data analysis techniques, the problem of scaling analysis techniques seems far more familiar for us. Although data analysis experts have worked for quite some time on the problems where the number of dimensions (data attributes) is large, they have been much less concerned with the number of data records. In part, this is because it has been traditional in data analysis work to make assumptions about the data distributions that model a data set. Assumptions on data distribution help reduce the size of the necessary data sets. In contrast, in most databases, little a priori inference on data distribution may be assumed. Thus, while most statistical and machine learning schemes assume a single-level store, over large data sets, we recognize the reality of multi-level store. This leads to two possible consequences:

- Develop efficient algorithms that take into account the fact that the data set is large.
- Restrict the scope of analysis objectives.

Scalability requirements lead to algorithms that carefully *stage computation* when data do not fit in memory. This is an area we have been leveraging. Nonetheless, we have to guard against the following dangers when we consider scalable implementations:

- *Restricting choice of data mining tasks:* While there has been an impressive amount of work related to association rules (see [1] for an overview) and their generalizations. Relatively less work has been done in the context of other classical data analysis technique, e.g., clustering, classification.
- *Scaling specific algorithms:* There are literally many variants of classification or clustering algorithms. The specific choice of an algorithm will depend on an application. Therefore, instead of focusing on how to scale a specific algorithm, we should try to identify common *data centric* steps in a broad class of algorithms. For example, all decision tree classifiers are driven by data-centric operations of building *counts* for distinct values of attributes and then *partitioning* the data set. Therefore, any decision tree classifier can be minimally modified so that whenever counting or partitioning steps are needed, the classifier uses an *interface* to invoke a *generic* middleware that optimizes scalable implementations of those operations by leveraging the way most decision tree classifiers grow a tree [4]. As a consequence, we are able to exploit the middleware for the entire set of decision tree classifiers, instead of requiring to build a specific scalable implementations for many variants.
- *Ignoring sampling as a scaling methodology:* Another generic way to scale the data over large data set is to use *sampling*. Whether sampling is appropriate for a class of data analysis technique, and even when appropriate, how much to sample and how, will become increasingly important question for data analysts. Use of sampling directly brings us to the related issue of restricting the scope of analysis objectives.

In designing the scalable implementations, some of the algorithms have made assumptions that ignore the fact that a datawarehouse will service not just data mining, but also traditional query processing. For example, some of the algorithms discuss how the physical design of the database may be tuned for a specific data mining task. However, in many cases, the physical design of a data warehouse is unlikely to be guided solely by the requirement of a single data analysis algorithm. Many of the scalable implementations also do not consider SQL database as the repository of data. In the next section, I will discuss this issue in somewhat more detail.

The problem of *restricting the scope of analysis* objective is motivated by the desire to strike a balance between accuracy and exhaustiveness of analysis with the desire to be efficient. While not a novel concept by any means, of direct interest to us will be techniques to efficiently cut corners that are motivated specifically by the large database (records and schema) scenarios. Restricting the scope of the analysis can take different forms. First, the analysis can be less than exhaustive. For example, support and confidence parameters are used to restrict the set of association rules that are mined. Next, the guarantee of the analysis can be probabilistic. A wide

class of sampling based algorithms can take advantage of such an approximate analysis. This is clearly an area that is rich with past work by AI/statistics community. To ensure that we avoid pitfalls in cutting corners that severely affect quality of analysis, a database systems person needs to carefully *work with* a data analysis expert.

# 3 Motivation for SQL-aware Data Mining Systems

I will discuss two obvious reasons why we need to consider implementation of data mining algorithms that are "SQL-aware". However, from my point of view, *ad-hoc mining* provides the most compelling reason to consider SQL-aware data mining systems.

**Data is in the warehouse**

Data warehouses are deploying relational database technology for storing and maintaining data. Furthermore, data in datawarehouse will not be exclusively used for data mining, but will be shared also by OLAP and other database utilities. Therefore, for pragmatic reasons, the data mining utilities *must* assume a relational backend.

**SQL Systems can be leveraged**

Apart from the reality that SQL databases hold enterprise data, it is also true that SQL database management systems provide a rich set of primitives for data retrieval that the mining algorithms can exploit instead of developing all required functionality from scratch. It is surprising that although scalability of mining algorithms has been an active area of work, few significant pieces of work have looked at the issue of data mining algorithms for SQL systems. A nice study of a SQL-aware scalable implementation of association rules appear in [8].

**Ad-hoc Mining**

Today's data mining algorithms are invoked on a materialized disk-resident data set. If data mining were to succeed, data mining must evolve to *ad-hoc data mining* where the data set which is mined is specified on-the-fly. In other words, mining may be invoked on a data set that has been created on-the-fly by the powerful query tools. This allows us to mine an arbitrary query, not necessarily just base data. Thus, it is possible for a power user to use OLAP tools to specify a subset of the data and then to invoke mining tools on that data (cf. [7]). Likewise, it is possible to exploit the data reduction capabilities of the data mining tools to identify a subset of data that is interesting and then the OLAP tools can explore the subset of the data using query features. As an example, mining tool may be used to reduce the dimensionality of data. For ad-hoc mining, requiring the query to be materialized will result in unacceptable performance in many cases. A far more sensible approach is to cleverly exploit the interaction of the mining operator with the SQL operators. Similar interactions are possible with data visualization tools.

# 4 Building SQL-aware Data Mining Systems

In this section, I will use the example of *decision-tree classification* as an example of a data analysis technique, to illustrate the issues related to integration. The decision-tree classification process begins with the root node of the tree representing the entire data set. For each data value for each attribute of the data set, the counts of tuples are computed. These counts are used to determine a criteria to either partition the data set into a set of a disjoint partitions based on values of a specific attribute or to conclude that the node (the root in this case) is a leaf node in the decision tree. This "count and split" cycle is repeated until no new partitions are possible.

Recently, we built a scalable classifier [4] at Microsoft Research. Our approach exemplifies one of the several ways in which the problem of building SQL-aware systems may be approached. We started with a classical main-memory implementation of a decision tree classifier and Microsoft SQL Server. We augmented this set-up with a middleware to enhance performance. In particular, we modified the in-memory classifier such that it invokes the middleware whenever it needed to generate counts for each active node. In our first implementation, our goal was to get the SQL backend to do all the work in generating counts. The implementation helped us identify key bottlenecks and led to implementation changes for optimal use of server functionality as well as led to needs for SQL extensions. In the rest of this section, I will briefly discuss the issues related to exploiting the SQL backend as well as issues related to extensions to SQL for data mining. Where appropriate, I will draw examples from the decision-tree classifier and association rule implementations.

### 4.1 Using SQL Backend

Effectively using a SQL backend for data mining applications is a nontrivial problem since using the SQL backend as much as possible in an obvious way may hurt performance. The problem is analogous to what ROLAP providers faced in building their middleware over SQL engines. In particular, instead of generating a single complex SQL statement against the backend, they often generate multi-statement SQL that may be executed more efficiently. Similar considerations will be needed in the context of data mining.

On the other hand, we need to exploit the functionality in the SQL subsystem that can indeed be leveraged. Physical database design[1] and query processing subsystem including its use of parallelism are examples of functionality that data mining applications can exploit. Although the above seems too obvious to mention, there are few implementations of data mining algorithms today that take advantage of these functionality. The goal of harnessing the above functionality often lead to novel ways of *staging computation*. In [4], we discuss how we can batch servicing multiple active nodes (i.e., nodes that are still being grown) of a scalable decision tree classification algorithm and exploit data structures in the database server.

### 4.2 SQL Extensions

As we implement mining algorithms that generate SQL efficiently, we also will identify primitives that need to be incorporated in SQL. Once again, we can draw similarities with the OLAP world. Generation of SQL queries against the backend clearly benefits from the CUBE construct [6]. We can identify two goals for studying possible extensions to SQL, extensions that:

1. strongly interact with core SQL primitives and can result in significant performance improvement.
2. encapsulate a set of useful data mining primitives.

We feel that extensions that belong to (1) are extremely useful. An example of an operator which belongs there is the ability to sample a relation and more generally a query. This functionality can be exploited by many data mining algorithms, especially algorithms that provide probabilistic guarantees. However, the operation to sample a query is also a feature that strongly interacts with the query system. In particular, a sampling operator can be pushed down past a selection and interacts with other relational operators. While there has been substantial past research in this area, implementation issues related to processing sampling along with other relational operators continue to be an active area.

In our recent work on building scalable classification algorithms over SQL subsystems [4], we recognized that there is strong performance incentive to do *batch aggregation*. Intuitively, batch aggregation helps fully leverage a single data scan by evaluating multiple aggregation over the same data (or, query). This functionality

[1] It is important to emphasize that data mining algorithms need to *exploit* the physical design, but should not assume that such algorithms will singularly dictate such designs.

is important in classification since while growing a decision-tree classifier, for every active (non-leaf) node, we must collect the count of the number of tuples for every value of every attribute of the data table. This corresponds to a set of single-block aggregation queries where all the queries share the same *From* and *Where* clauses, i.e., differ only on *Group By* and *Select* clauses. Having the ability to do multi-statement optimization of the above set of related queries and to exploit a single data scan to evaluate them greatly speed up the classification algorithms. Such batch aggregation functionality goes beyond the CUBE operator [6]. Both sampling and batch aggregation strongly interact with core SQL primitives and thus with the SQL relational engine implementations.

We distinguish the above set of new operators with those that do not strongly interact with core SQL but presents a set of useful encapsulated procedures, perhaps supported via a mining extender/cartridge/blade or simply via extended system-provided stored procedures (depending on the database vendor). The purpose of such a set of operators is to make it easy to develop new data mining applications. However, for the set of primitives in this class to be useful, it is important that the operations be *unbundled* so that they may be shared. This issue is best illustrated through a recent SQL extension that has been proposed for association rules [2]. In that proposal, an extension is proposed to generate association rules. However, note that an alternative would have been to consider specifying *frequent itemsets* instead. An association rule can be derived easily with frequent itemsets as the primitive. Furthermore, the construct for frequent itemset may be exploited more generally for different variants of association rules. Thus, building extenders that directly map one-on-one to individual data mining algorithms may not be ideal.

# 5 Conclusion

In this article, I have reviewed various facets of data mining. There is an opportunity to work closely with data analysts to develop approximations of classical data analysis that are scalable. There is a need to look for generic scalability extensions for each class of data mining algorithms, rather than for specific scalable algorithms in each class. Another important direction is to consider scalable implementations over SQL systems. Such an effort will lead not only to changes in scalable algorithms, but also lead to new extensions for SQL that will make it better suited to support a variety of data mining utilities. Finally, it is well understood that core data mining algorithms by themselves are not sufficient, but needs to be integrated with other database tools. In particular, it is necessary to augment them with visualization support, as has been done in OLAP.

**Acknowledgement**

We thank Umesh Dayal, Usama Fayyad, Goetz Graefe, and Jim Gray for many fruitful discussions.

# References

[1] Agrawal R. et. al. “Fast Dicovery of Association Rules,” pp. 307-328 in [5].

[2] Meo R., P. Giuseppe, Ceri S., “A new SQL-like Operator for Mining Association Rules,” in *Proc. of VLDB96*, pp. 122-133, Mumbai, India.

[3] Chaudhuri S., Dayal U. “An Overview of Datawarehousing and OLAP Technology,” in *Sigmod Record*, March 1997.

[4] Chaudhuri S., Fayyad U., Bernhardt J. “Scalable Classifier over SQL Databases,” in preparation.

[5] Fayyad U. et. al. *Advances in Knowledge Discovery and Data Mining*, MIT Press, 1996.

[6] Gray et al. “Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub Totals,”in *Data Mining and Knowledge Discovery*, 1(1), pp. 29-53, 1997.

[7] Han J. “Towards On-Line Analytical Mining in Large Databases,”to appear.

[8] Sarawagi S., Thomas S., Agrawal R. “Integrating Mining with Relational Database Systems: Alternatives and Implications,” in *Proc. of ACM Sigmod 98*, To appear.