Mining Databases and Data Streams with Query Languages and Rules

Carlo Zaniolo

Computer Science Department
UCLA
Los Angeles, CA 90095
zaniolo@cs.ucla.edu

Abstract. Among data-intensive applications that are beyond the reach of traditional Data Base Management Systems (DBMS), data mining stands out because of practical importance and the complexity of the research problems that must be solved before the vision of Inductive DBMS can become a reality. In this paper, we first discuss technical developments that have occurred since the very notion of Inductive DBMS emerged as a result of the seminal papers authored by Imielinski and Mannila a decade ago. The research progress achieved since then can be subdivided into three main problem subareas as follows: (i) language (ii) optimization, and (iii) representation. We discuss the problems in these three areas and the different approaches to Inductive DBMS that are made possible by recent technical advances. Then, we pursue a language-centric solution, and introduce simple SQL extensions that have proven very effective at supporting data mining. Finally, we turn our attention to the related problem of supporting data stream mining using Data Stream Management Systems (DSMS) and introduce the notion of Inductive DSMS. In addition to continuous query languages, DSMS provide support for synopses, sampling, load shedding, and other built-in functions that are needed for data stream mining. Moreover, we show that Inductive DSMS can be achieved by generalizing DSMS to assure that their continuous query languages support efficiently data stream mining applications. Thus, DSMS extended with inductive capabilities will provide a uniquely supportive environment for data stream mining applications.

1 Introduction

Data Base Management Systems (DBMS) and their enabling technology have evolved successfully to deal with most of the data-intensive application areas that have emerged anew during the last twenty years. For instance, in response to the growing importance of decision-support applications, relational DBMS and SQL were quickly extended to support OLAP queries—a remarkable exploit from both technical and commercial viewpoints. On the other hand, there have also been significant failures, with data mining applications representing the

most egregious of such failures. Therefore, databases today are still mined using primarily a cache-mining approach, whereby the data is first moved from the database to a memory cache, which is then processed using mining methods written in a procedural programming language. Indeed, most mining functions cannot be expressed efficiently using SQL:2003, which represents the standard query language of DBMS.

Research on Inductive Inductive DBMS aims at changing this state of affairs and make it easy to mine databases by their query languages. The emergence of Inductive DBMS as a well-defined research area can be traced back to the seminal papers by Imielinski and Manilla [1, 2] who introduced the lofty notion of a DBMS where complex mining tasks can be expressed with ease using the query language of the system[1]. According to [2], Inductive DBMS should also assure efficient execution of such high-level mining queries via powerful query optimization techniques—although the enabling technology for such a task was not available at the time[2].

Early attempts to realize the lofty notion of Inductive DBMS have produced mining languages such as MSQL[3], DMQL[4] and the Mine Rule [5]. These projects propose SQL extensions to specify the data to be mined and the kind of patterns to be derived, along with other parameters needed for the task, such the support and confidence required. As discussed in a comprehensive comparative study [6], these projects have made a number of contributions, by exploring and demonstrating some of the key features required in a Inductive DBMS, including

(i) the ability of specifying constraints, meta-patterns, and concept hierarchies to sharpen the search process,
(ii) the ability to apply the derived rules to the original data for verification and analysis (crossing over),
(iii) the closure property whereby the query language can be used to operate on the results produced by the mining query.

The research contributions brought by these approaches have not led to significant commercial deployments, because of practical limitations. A first limitation is that these approaches are primarily intended for association rule mining, although DMQL consider other patterns besides association rules.

A second and more serious concern is that of performance: the projects discussed in [3–5] do not claim to have achieved performance levels that are comparable to those achievable with the cache-mining approach, nor they claim to have identified a query-optimization approach that can be reasonably expected to take them there. This in line with the view of Imielinski and Manilla[2] that sophisticated optimizers are needed to achieve good performance and developing such technology represents a long-term research challenge for which no quick solution should be expected. Furthermore, experience with query optimizers has

[1] 'There is no such thing as real discovery, just a matter of the expressive power of the query languages' [2].

[2] 'KDD query optimization will be more challenging than relational query optimization ... It took more than 20 years to develop efficient query optimization and execution methods for relational query languages' [2].

shown that it is very difficult to extend relational optimizers to handle more powerful constructs, such as recursive queries, or richer data models and their query languages, i.e., XML and XQuery. Therefore, optimizers for data mining queries require novel techniques, not just extensions of relational optimizer technology. Such a task could take many years, although progress on this difficult problem has been achieved in the last few years [7–10]. Once these techniques will be incorporated into systems supporting declarative mining queries, the lofty vision of [2] will then be realized, at least for associative rule mining.

In order to provide data mining functions to their users, commercial database vendors are instead taking a quite different approach. Typically, vendors have been implementing a suite of data mining functions on top of their DBMS, along with graphical interfaces to drive these packages [11–13]. While only providing a predefined set of built-in mining functions, the Microsoft DB serve is however achieving a closer integration and better interoperability of the mining task with the SQL query task, by using the descriptive+predictive mining model of OLE DB DM [13]. Thus the descriptive task generates an internal representation (a mining model) as a special table that is populated (learned) by executing the mining task on the training data. Then, a special operator called prediction join is provided that can be used to predict unknown attribute values for new data [13]. It is also possible to inspect the descriptive model and export it into an XML-based representation called PMML (Predictive Model Markup Language). PMML is a markup language proposed to represent statistical and data mining information [14].

Therefore, OLE DB DM goes beyond the mining-language approach by addressing the need to support the multiple steps of the DM process with well-defined representations linking the various steps. Ideally, this should lead to the notion of open Inductive DBMS, where, for instance, descriptive models can be imported into the system and used for prediction (or exported and used for predictive tasks in a second system).

In addition to the mining-language approach and the DM approach of OLE DB, there is also a third approach that we will call the middle-road approach. This offers interesting promises both in terms of mining data bases and data streams, and is discussed in the next two sections.

2 Query Languages and Data Mining

The mining-language approach proposed in [3–5] defines a very ambitious high-road path toward Inductive DBMS, since users only need to provide high-level declarative queries specifying their mining goals. Then, the Inductive DBMS optimizer is left with the responsibility of selecting an algorithm to accomplish those goals—a task that, in general, exceeds the capabilities of current technology.

At the opposite end of the spectrum, we find the low-road approach discussed in the prize-winning paper presented in [15]. In said study, a task force of researchers with deep expertise on mining methods and the IBM DB2 O-R DBMS tried to implement efficiently the APriori algorithm, exploring several

implementation alternatives that only use the DBMS as is, using the standard SQL version supported by DB2. An acceptable level of performance was achieved through the use of specialized join techniques and user-defined functions (UDFs), at the price of excessive difficulties in programming and debugging [15]. We will characterize the approach taken in [15] as a 'low-road' path toward Inductive DBMS. While the work presented in [15] established the inadequacy of SQL in supporting complex data mining algorithms such as Apriori, it provided no clear indication how to proceed to overcome these inadequacy.

Once we compare the high-road approach against the low road we see that the first makes unrealistic demands upon the system, while the second makes unrealistic demands on the users. Given this situation, it is only natural to pursue middle-road approaches that explore extensions of SQL and DBMS that are realizable with current technology and make the task of writing mining algorithms simple for common mortals. We next describe the ATLaS system that is taking such middle-road path to Inductive DBMS.

As described by Arno Siebes in his invited talk [16], data mining success stories in the real world, frequently employ the simplest mining methods, e.g., Naive Bayesian Classifiers (NBCs). NBCs are also significant for the very subject of this paper, since they provide a unique example of on data mining algorithm that current DBMS can support as well as full-fledged Inductive DBMS would.

Take for instance the well-known Play-Tennis example of Table 1: we want to predict the value of **Play** as a 'Yes' or a 'No' given a training set consisting of tuples similar to the three shown in Table 1.

RID	**Outlook**	**Temp**	**Humidity**	**Wind**	**Play**
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	Yes
3	Overcast	Hot	High	Weak	Yes
...	...	...	...	..	...

Table 1. The relation **PlayTennis**

The first step is to convert the training set into column/value pairs whereby the first two tuples in Table 1 are now represented by the eight tuples shown in Table 2.

This verticalization can be implemented using a table function, which is a very useful SQL:2003 construct now supported by most DSMS. From this representation, we can now build a Bayesian classifier by simply counting the occurrences of Yes and No with a statement as follows:

RID	Column	Value	Dec
1	1	Sunny	No
1	2	Hot	No
1	3	High	No
1	4	Weak	No
2	1	Sunny	Yes
2	2	Hot	Yes
2	3	High	Yes
2	4	Strong	Yes
...	...	...	...

Table 2. A Column-oriented representation for **PlayTennis**

Example 1 *Disassemble a relation into column/value pairs.*

```
SELECT Column, Value, Dec, count(Dec) as mycount
    FROM traningset
    GROUP BY Col, Value, Dec
```

We can then add up the counts for each column, and use it to normalize the values of **mycount** (by dividing by the total number of 'Yes' and 'No'). Finally, we take the absolute value of the log of the results and thus obtain a descriptor table as follows:

DescriptorTbl(Col: int, Value: int, Dec: int, Log: real)

Now, the set of tuples submitted for prediction will also be collected in a table called, say **TestTuples** having the same format as Table 2, except that the column **Dec** is missing. Then, the Naive Bayesian classifier is implemented using the results of the following query:

Example 2 *Probabilities for each tuple to be predicted*

```
SELECT t.RID, d.Dec, sum(d.Log)
FROM DescriptorTbl AS d, TestTuples AS t
WHERE d.Val=t.Val AND d.Col=t.Col
    GROUP BY t.RID, d.Dec
```

Thus, for each test tuple, and each class label we multiply (sum the logs of) the relative frequencies for each column value supporting this class label. The final step would consist in selecting for each RID the class label with the least sum, a step that in SQL requires finding first the min value and then the columns where such min value occurs (such a min maximizes the probability since we use absolute values of logarithms).

Observe that so far, we have only described the core descriptive and predictive tasks and not discussed other tasks such as data preparation, testing the model accuracy, and boosting it. However, these tasks can normally be expressed by rather simple SQL queries on our basic relational representation. For instance, if we want to build an ensemble of classifiers, we only need to add to the descriptor table a new column containing the classifier name: then voting operations can be

reduced to counting the number of individual classifiers for each (i.e., grouped by each) **Dec** value and then selecting the decision supported by most votes. Here again relational tables are used to describe both the data and the induced model.

The example of Naive Bayesian Classifiers illustrates the superior computational environment that DBMS can bring to the data mining process once their query languages are capable of expressing such applications. Therefore, a very natural middle-road approach can be that of preserving the basic relational representation for the data sets and the induced models, but providing extensions to SQL:2003 to turn it into a more powerful language—one that is capable of expressing complex mining algorithms. In the past, aggregates extended with more general GROUP BY constructs enabled SQL-compliant DSMS to support decision support functions via OLAP and data cubes. More recently, in our ATLaS project, we have shown that User-Defined Aggregates (UDAs) natively defined in SQL can turn SQL into a powerful (Turing-complete [17]) language for data mining applications [18].

The ATLaS middle-road approach allows users to write data mining algorithms in SQL extended with natively defined UDAs. For instance, we will now write a simple UDA that computes the correct classification from a table storing the results of Example 2. If we were restricted to standard SQL, things would be more complex, since we would need to nest a statement that finds the minimum into another statement that finds all the points where this occur. Moreover, to break ties, we will have to find again the min (or max) among the such points (ordered by lexicographically). Alternatively, we can use the following UDA:

Example 3 *Defining the standard aggregate minpoint*

```
AGGREGATE mincol(inCol Int, inValue Real) : Int
{    TABLE current( CrCol Int, CrValue Int);
     INITIALIZE : {
        INSERT INTO current VALUES (inCol, inValue);
     }
     ITERATE : {
        UPDATE current SET CrCol=inCol, CrValue=inValue;
        WHERE CrValue <= inValue
     }
     TERMINATE : {
        INSERT INTO RETURN SELECT CrCol FROM current;
     }
}
```

In this case, we have an internal table which only contains one tuple that is always updated to the incoming **inCol**, **inValue** pair when **inValue** is less or equal to the current minimum (but in a situation where we want to find the top K values/points our table would instead contain K tuples). Observe the stream-oriented computation is specified in the three steps: (i) when the first tuple arrives (initialize), (ii) when the successive tuples arrive (iterate), and (iii) after the input is exhausted (terminate). A number of commercial DBMS

support UDAs where the computations in these three states can be defined in an external procedural language. However, as shown by our simple example, these computations can be naturally defined in SQL itself, an approach that has three important advantages:

- UDAs can be invoked from other UDAs,
- UDAs can access the database tables besides their internal tables, and
- any impedance mismatch problem is eliminated.

In a nutshell, we obtain a rich programming environment, which brings native extensibility and Turing-completeness to SQL [17] which can be used in a number of other applications besides data mining. For data mining, however, UDAs afford the ability of expressing concisely and efficiently all data mining methods, including Apriori [18].

For instance, a basic-decision tree classifier might start by computing the gini index (or entropy gain) instead of the probabilities used for NBCs. Then, to decide where to split, we will have to find where a minimum occurs. For instance, for a multiway split we will count for each column and each value in the column the number of Yes and No, and we use those to compute a gini index. If store the pairs (column, gini-value) in a temporary table, the next step consists in selecting the column where we have the least gini index by calling the UDA of Example 3, above.

This would generate the first level of nodes in our decision tree. We can now partition the training set according to these node numbers, and then we can call the same UDA grouped by this node number. Thus, a classifier can be written as a UDA consisting of fourteen ATLaS-SQL lines [18].

Not surprising, given the experience described in [15], writing an efficient implementation of Apriori proved a tougher test, one that required forty-five lines of ATLaS-SQL code. In terms of performance, the key issue proved to be the support for data structures such as prefix trees, which we were able to support via the use of in-memory tables and SQL reference data types that, for in-memory tables, can be used to point to other tuples [18]. The performance and scalability so obtained are comparable to those obtainable with the cache-mining approach, and normally better than those of java-based data mining libraries [19].

The ability of working directly with SQL represents a practical advantage of this approach over others using new special algebras [20]. Moreover, the stream-oriented definition mechanism of UDAs makes them particularly effective at mining data streams, as discussed next.

3 Inductive Data Stream Management Systems

There is a great deal of interest in managing high volumes of information that is exchanged as data streams that, because of high arrival rates or immediate response requirements, cannot be managed via DBMS. Therefore, Data Stream Management Systems (DSMS) are being developed to manage streaming information by supporting data streams applications via continuous queries [21]. In

particular, data stream mining applications have been the focus of much recent interest [22–24] raising the issue of designing the best DSMS to support such applications. Therefore, in this section, we introduce the notion of an Inductive DSMS which falls naturally at the intersection of the two research areas. In most general terms, we will define Inductive DSMS as DSMS designed to supports and facilitate the task of data stream mining.

While many approaches are possible to the design of management systems that support publish & subscribe OR data streams, a very popular research approach consists in using query languages and operators similar to those of databases [21, 25–28] and extend them with operators and constructs specifically designed for data streams. Typical extensions include windows or other synoptic structures, sampling, and load shedding [21, 25]. Moreover, Inductive DSMS are often used to support mining algorithms that are similar to those of Inductive DBMS, as demonstrated by the fact that stream mining algorithms are often fast&light, one-pass adaptation of the original algorithms designed to work on stored data. Therefore, approaching Inductive DSMS and Inductive DBMS as two closely related technical topics is natural and likely to be beneficial from a research viewpoint. In terms of practical issues, however, we see that the two areas are different and face somewhat complementary concerns, which are briefly discussed next.

The fact that DSMS are far from the level of maturity and standardization achieved by DBMS represents a clear disadvantage for Inductive DSMS, which however, also enjoy major advantages, because of the number of built-in functions they support, and because cache mining might no longer represent an appealing alternative for data streams. For instance, the typical approach used for mining data streams consists in dividing the incoming data into windows. By comparing the statistics of successive windows we can (i) detect concept shift/drift, and when none is detected (ii) use bagging and boosting techniques to improve the predictive accuracy of our model [23, 24]. DSMS support a rich assortment of window constructs that can be utilized very effectively in these tasks [29, 27, 21].

Sampling represents another basic function that is useful for mining data streams [30] and is well-supported in DSMS [26]. For instance, sampling can be used to find the center of clusters [31] or frequent item sets for mining association rules [32]. Moreover, building classifier ensembles via multiple samples of the data can result in improved accuracy [33]. Also, interesting techniques have been proposed to improve the accuracy of aggregates and mining methods on sample data using past information on the stream behavior [34]. In principle, a cache-mining programmer could code these sampling techniques or import them from some library, but in practice, an Inductive DSMS that supports windows and sampling as built-ins could be hard to resist for our opportunistic data stream miner.

The reasons for using an Inductive DBMS become even more compelling as we move from the language level to the system level, since DSMS provide unique functions such as load balancing, scheduling, and shedding, which are

designed to assure quality-of-service and prompt response in the presence of multiple users and bursty arrivals [35]. By taking advantage of computing grids, or distributed computing platforms, DSMS can provide highly reliable, non-stop service [36]. Thus data mining applications seeking uninterrupted service, reliability, robustness, and sharing by multiple applications will need Inductive DSMS (unlike database mining applications that can live without the support for transaction, recovery, and data independence provided by DBMS).

In summary, Inductive DBMS can deliver to the data stream miner great practical benefits—possibly even greater than those of Inductive DBMS in traditional mining applications. Moreover this research area also offers interesting opportunities, since techniques and solutions developed for Inductive DBMS can be naturally transferred to Inductive DSMS and vice versa. In particular, we have extended the middle-road approach to Inductive DBMS described in the previous section and applied to Inductive DSMS, by extending the UDAs of ATLaS with powerful primitives for windows, sampling, and time-stamp management. The Expressive Stream Language (ESL) so obtained, can express every non-blocking function expressible by a Turing machine and it is supported efficiently in our Stream Mill prototype [37]. In data streams applications, windows are often used in conjunction with aggregates, to overcome their blocking behavior and to summarize the past history of the data stream. Unlike other DSMS that only support windows on built-in aggregates, ESL supports a vast assortment of windows on arbitrary UDAs. For instance, a classifier UDA can be called on tumbles, i.e., windows that partition the input stream into disjoint segment, and the results produced by few recent tumbles can be used to build a classifier ensemble [38]. A sliding window aggregate is instead one that recomputes the value of the aggregate when new tuples arrive or leave the window, using differential maintenance techniques. The development of such techniques for the various mining methods represent an interesting topic of ongoing research. In the following example we show how the DBscan algorithm can be concisely written in ESL and applied to an incoming stream partitioned into tumble windows.

Density-Based Clustering In our application, we have a stream of points in a two-dimensional space. In order to study the data and distribution changes, we (i) partition the stream into windows of equal size, (ii) cluster the data in each window, and (iii) compare the sizes of the different clusters in successive windows, along with any appearance of new clusters or disappearance of old ones. For clustering, we employ the density-based clustering algorithm DBScan [39]. The density conditions is defined by the fact that within a radius of **eps**, we find at least **minPts** points; thus, points that occur in a dense area are assigned to the same cluster, while points that fall in a sparse area are instead classified as outliers.

The partition of the incoming stream into windows and the execution of DBscan on each window are accomplished by the following ESL statement that calls the **dbscan** aggregate on input data stream:

Stream_of_Points(Xvalue, Yvalue, TimeStamp).

Example 4 *Applying dbscan with minPts = 10 and eps = 50*

```
        /*call dbscan with minPts = 10 and eps = 50 */
SELECT dbscan(Xvalue, Yvalue, 0, 10, 50)
        OVER(ROWS 999 PRECEDING SLIDE 1000 )
FROM Stream_of_Points
```

Here 10 and 50 are the example values we assign to two important parameters for the DBScan Algorithm, **eps** and **minPts**, respectively. The third argument is for book-keeping purposes. Observe that since the size of the slide is the same as that of the window, this is known as a tumble. Therefore the Stream Mill system will use the base definition of DBscan, shown below. Given the two parameters **eps** and **minPts**, the DBScan algorithm works as follows: pick an arbitrary point **p** and find its neighbors (points that are less than eps distance away). If **p** has more than **minPts** neighbors then form a cluster and call DBScan on all its neighbors recursively. If **p** does not have more than **minPts** neighbors then move to other un-clustered points in the database. Note, this can be viewed as a depth-first search.

```
AGGREGATE dbscan(iX Real, iY Real, Flag Int, minPt Int, eps Int): Int
{       TABLE closepnts(X2 real, Y2 real, C2 Int) MEMORY;
        INITIALIZE: ITERATE: {
           /* Find neighbors of the given point */
           INSERT INTO CLOSEPNTS SELECT X1, Y1, C1 FROM points
           WHERE sqrt((X1-iX)*(X1-iX) + (Y1-iY)*(Y1-iY)) < eps;
           /* If there are more than minPt neighbors, form a cluster */
           UPDATE clusterno SET Cno= Cno+1  /* new cluster number*/
           WHERE Flag=0 AND SQLCODE=0 /* A new cluster */
           AND minPt < (SELECT count(C2) FROM closepnts);
           /* Assign these neighboring points to this cluster */
           UPDATE points SET C1 = (SELECT Cno FROM clusterno)
           WHERE points.C1=0 AND
           EXISTS (SELECT S.X1 FROM closepnts AS S
                WHERE points.X1=S.X2 AND points.Y1=S.Y2 )
                AND minPt < (SELECT count(C2) FROM closepnts);
           /* Call dbscan recursively */
           SELECT dbscan(X2, Y2, 1, minPt, eps)
           FROM  closepnts, points
           WHERE X1 = X2 AND Y1=Y2;
           DELETE FROM closepnts;
        }
};         /*end dbscan*/
```

This density-based clustering was part of a demo presented at the ACM SIGMOD 2005 conference of the Stream Mill System that supports very powerful continuous queries on data streams and applications that span both data streams and databases using a client-server architecture [38]. Another data mining application demonstrated on that occasion, was an ensemble-based classifier where each window was used to build a separate classifier. Sliding windows that

are recomputed after each new tuple arrives in the window, are suitable when incremental computation is feasible—as in the case of mining methods, such as Naive Bayesian Classifiers that are based on count or other algebraic aggregates. Two other important advantages of Stream Mill are (i) support for time series applications, and [40], and (ii) inclusion of streaming XML data along with relational streams [41], as needed e.g., to support PPML data. Indeed, Stream Mill has already taken the first important steps toward becoming an Inductive DSMS.

This example illustrates how the middle-road approach to data mining can be generalized to work with data streams, and in fact the simpler one-one pass algorithms that are prevalent with data streams can be expressed simply and concisely using UDAs. However, other approaches to Inductive DBMS, such as the mining-language approach, or the OLE DB DM approach, can also be extended naturally to support Inductive DSMS and such extensions provide an interesting topic for future research.

4 Conclusion

Ten years after being proposed in concept papers [1, 2], the notion of inductive databases is coming of age in terms of research advances and commercial systems with progress occurring along three largely parallel and independent paths. Progress along the high-road pathway, has been made with the introduction of the first generation of mining languages [6], and with techniques for the optimization of declarative mining queries based on association rules [9]. Progress along the middle-road has delivered SQL extensions based on natively defined UDAs that can be used to write data mining algorithms [18]. On the commercial front, DBMS can now support the combination of descriptive/predictive data mining via a predefined library mining methods [42].

Remarkably these advances are not mutually exclusive but they should instead be integrated to produce more powerful Inductive DBMS. In particular, the libraries of systems such as OLE DB DM should be made extensible, as to accommodate the inclusion of new declarative mining methods and procedural mining methods. As demonstrated by ATLaS, new mining methods can be added to DSMS as UDAs operating on tables. While these UDAs could be written in a foreign language, UDAs natively and concisely written in SQL are preferable, because they are safe, easier to modify, and free of 'impedance mismatch' problems.

In order to get synergy between these different approaches, we must assure their interoperability. Experience with data mining libraries [43, 19] indicates that for flexibility and interoperability, we need to establish well-defined representations between the various steps of the mining process. These representations must, e.g., support import/export of data, metadata and mining models, so that they can be cooperatively exchanged between different systems. The PMML-based approach of OLE DB [42] represents an important first step in the right direction, but it suffers from limitations in terms of power and generality. For instance, while a single classifier can be imported/exported using PMML, it is

not clear how ensembles of such classifiers could be assembled and reimported to perform a predictive task. While more general approaches to the representation of mining artifacts are possible using XML, not all representations are equally desirable. For instance, for large data sets, relational tables have proven to be much more efficient than XML-based ones both in terms of data and query efficiencies. On the other hand, logical rules are clearly the representation of choice in dealing with knowledge. As describe in [44] logical rule are very effective at (i) bringing the domain knowledge to bear upon specific mining task, (ii) driving the mining process by calling procedurally defined UDAs to perform the specific mining tasks, and (iii) combining the results of knowledge extraction with application-expert rules. From a research viewpoint, the success obtained in [45] with a rule-based data mining environment suggests the need for two important enhancements that were not available in the framework of systems [46] originally used in those experiments. One is the ability of using induced rules as if they were deductive rules, and the other is ability of using deductive rules to define UDAs which compare in terms of efficiency with those written in ATLaS SQL which approach those of UDAs written in a procedural language.

Finally, we have shown that the problem of mining data streams is so close to that of mining databases that the two should be pursued together to exploit the considerable opportunities their close relationship offers both in terms of research and commercial applications.

Acknowledgements

I would like to thank Francesco Bonchi and Yan-Nei Law for their comments and suggested improvements on the first version of the manuscript. In addition to their many helpful comments, Haixun Wang, Yijian Bai and Hetal Thakkar must also be credited with building ATLaS and Stream Mill.

References

1. Tomasz Imielinski. A database perspective on knowledge discovery. In *The First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, 1995.
2. Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Communication ACM*, 39(11):58–64, 1996.
3. T. Imielinski and A. Virmani. MSQL: a query language for database mining. *Data Mining and Knowledge Discovery*, 3:373–408, 1999.
4. J. Han, Y. Fu, W. Wang, K. Koperski, and O. R. Zaiane. DMQL: A data mining query language for relational databases. In *Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, pages 27–33, Montreal, Canada, June 1996.
5. R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *VLDB*, pages 122–133, Bombay, India, 1996.
6. Marco Botta, Jean-François Boulicaut, Cyrille Masson, and Rosa Meo. Query languages supporting descriptive rule mining: A comparative study. In *Database Support for Data Mining Applications*, pages 24–51, 2004.

7. Francesco Bonchi, Fosca Giannotti, Alessio Mazzanti, and Dino Pedreschi. Examiner: Optimized level-wise frequent pattern mining with monotone constraint. In *ICDM*, pages 11–18, 2003.
8. Sau Dan Lee and Luc De Raedt. An algebra for inductive query evaluation. In *ICDM*, pages 147–154, 2003.
9. Francesco Bonchi and Claudio Lucchese. Pushing tougher constraints in frequent pattern mining. In *PAKDD*, pages 114–124, 2005.
10. Baptiste Jeudy and Jean-François Boulicaut. Constraint-based discovery and inductive queries: Application to association rule mining. In *Pattern Detection and Discovery*, pages 110–124, 2002.
11. IBM. Db2 intelligent miner, http://www-306.ibm.com/software/data/iminer.
12. ORACLE. Oracle data miner release 10gr2, http://www.oracle.com/technology/products/bi/odm.
13. Z. Tang, J. Maclennan, and P.P. Kim. Building data mining solutions with ole db for dm and xml for analysis. *SIGMOD Record*, 34(2):80–85, 2005.
14. Data Mining Group (DMG). Predictive model markup language (pmml), http://sourceforge.net/projects/pmml.
15. S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *SIGMOD*, 1998.
16. Arno Siebes. Where is the mining in kdid? (invited talk). In *Fourth Int. Workshop on Knowledge Discovery in Inductive Databases (KDID 2005), Porto, Prtugal*, 2005.
17. Yan-Nei Law, Haixun Wang, and Carlo Zaniolo. Data models and query language for data streams. In *VLDB*, pages 492–503, 2004.
18. Haixun Wang and Carlo Zaniolo. Atlas: a native extension of sql for data minining. In *Proceedings of Third SIAM Int. Conference on Data Mining*, pages 130–141, 2003.
19. Weka 3—data mining with open source machine learning software in java http://www.cs.waikato.ac.nz.
20. Theodore Johnson, Laks V. S. Lakshmanan, and Raymond T. Ng. The 3w model and algebra for unified data mining. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 21–32. Morgan Kaufmann, 2000.
21. B. Babcock, S. Babu, M. Datar, R. Motawani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.
22. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *SIGKDD*, pages 97–106, San Francisco, CA, 2001. ACM Press.
23. Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *KDD*, pages 226–235, 2003.
24. Fang Chu, Yizhou Wang, and Carlo Zaniolo. An adaptive learning approach for noisy data streams. In *ICDM*, pages 351–354, 2004.
25. Lukasz Golab and M. Tamer Ozsu. Issues in data stream management. *ACM SIGMOD Record*, 32(2):5–14, 2003.
26. Theodore Johnson, S. Muthukrishnan, and Irina Rozenbaum. Sampling algorithms in a stream operator. In *SIGMOD Conference*, pages 1–12, 2005.
27. D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 12(2):120–139, 2003.
28. C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk, and O. Spatscheck. Gigascope: High performance network monitoring with an sql interface. In *SIGMOD*, page 623. ACM Press, 2002.

29. A. Arasu, S. Babu, and J. Widom. Cql: A language for continuous queries over streams and relations. In *DBPL*, pages 1–19, 2003.
30. Mohamed Medhat Gaber, Arkady B. Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *SIGMOD Record*, 34(2):18–26, 2005.
31. Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. Knowl. Data Eng.*, 15(3):515–528, 2003.
32. Hannu Toivonen. Sampling large databases for association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 134–145. Morgan Kaufmann, 1996.
33. Kagan Tumer and Joydeep Ghosh. Error correlation and error reduction in ensemble classifiers. *Connect. Sci.*, 8(3):385–404, 1996.
34. Yan-Nei Law and Carlo Zaniolo. Improving the accuracy of continuous aggregates and mining queries. In *Submitted for Publication*, 2005.
35. Nesime Tatbul, Ugur Çetintemel, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Load shedding in a data stream manager. In *VLDB*, pages 309–320, 2003.
36. Yanif Ahmad, Bradley Berg, Ugur Çetintemel, Mark Humphrey, Jeong-Hyon Hwang, Anjali Jhingran, Anurag Maskey, Olga Papaemmanouil, Alex Rasin, Nesime Tatbul, Wenjuan Xing, Ying Xing, and Stanley B. Zdonik. Distributed operation in the borealis stream processing engine. In *SIGMOD Conference*, pages 882–884, 2005.
37. Stream mill home. http://wis.cs.ucla.edu/stream-mill.
38. Chang Luo, Hetal Thakkar, Haixun Wang, and Carlo Zaniolo. A native extension of sql for mining data streams. pages 873–875, 2005.
39. Hans-Peter Kriegel Martin Ester, J. Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD 1996*, pages 226–231, 1996.
40. Y. Bai, L. Chang, H. Thakkar, X. Zhou, and C. Zaniolo. Efficient support for time series queries in data stream management systems. In K. Shaw N. Chaudhry and M. Abdelguerfi (eds), editors, *Stream Data Management" Kluwer: Chapter 6.* Kluwer Academic Publishers, 2005.
41. Xin Zhou, Hetal Thakkar, and Carlo Zaniolo. Unifying the processing of xml streams and relational data streams. The 22nd International Conference on Data Engineering April 3-7, Atlanta, GA, 2006, 2005.
42. ZhaoHui Tang, Jamie Maclennan, and Pyungchul (Peter) Kim. Building data mining solutions with ole db for dm and xml for analysis. *SIGMOD Record*, 34(2):80–85, 2005.
43. Clementine http://www.spss.com/clementine/index.htm.
44. F. Giannotti, G. Manco, D. Pedreschi, and F. Turini. Experiences with a logic-based knowledge discovery support environment. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, 1999.
45. Fosca Giannotti, Giuseppe Manco, Dino Pedreschi, and Franco Turini. Experiences with a logic-based knowledge discovery support environment. In *AI*IA*, pages 202–213, 1999.
46. Faiz Arni, KayLiang Ong, Shalom Tsur, Haixun Wang, and Carlo Zaniolo. The deductive database system ldl++. *TPLP*, 3(1):61–94, 2003.