Inductive Databases: Towards a New Generation of Databases For Knowledge Discovery

Rosa Meo
Dipartimento di Informatica, Università degli Studi di Torino
corso Svizzera 185, Torino, Italy
meo@di.unito.it

Abstract

Data mining applications are typically used in the decision making process. The Knowledge Discovery Process *(KDD process for short) is a typical iterative process, in which not only the raw data can be mined several times, but also the mined patterns might constitute the starting point for further mining on them.*

These are the premises that lead Imielinski and Mannila in [12] to propose the idea of inductive database*, a general-purpose database in which both the data and the patterns can be represented, retrieved and manipulated. The goal of inductive databases is to assist the deployment of the KDD process and integrate several heterogeneous data mining and data analysis tools. In this paper we overview the current state of the art of the research in databases support for KDD. We mean database standards for KDD, APIs for data mining, ad-hoc query languages and constraint-based query optimization. Our look is essentially from an academic point of view but also from an industrial one.*

1. Introduction

Recently, the research literature is emphasizing that the standard, relational or logical databases systems may no longer be sufficient to provide an effective answer to the advanced analysis requirements emerging in complex industrial and scientific applications. Specific systems, for discovering the nuggets of knowledge hidden in huge data sets have been applied as decision support systems to provide the user the specific answers related to the particular problem domain. This situation is reminiscent of the advances in databases in the late sixties when several different ad-hoc systems were being developed, without the unifying knowledge of a general purpose data model that will allow the production of an efficient database technology. The commonality in all these new systems is the goal of knowledge discovery which consists in obtaining useful knowledge from large collections of data. This task is inherently interactive and iterative: the user of a KDD system must have a solid understanding of the domain in order to select from original data meaningful subsets and have the knowledge to apply good criteria for the selection of interesting patterns. The concept of pattern is meant as an interesting knowledge artifact extracted from data.

Discovering knowledge from data is a process containing several steps: (1) understanding the domain, (2) preparing the data set, (3) discovering patterns (the data mining step), (4) postprocessing of discovered patterns and finally (5) deploying the results.

The KDD process is exploratory. Indeed, the user very often performs an analysis of the database in an exploratory way, driven by some hypothesis in mind. If such hypothesis are confirmed by the results, the evidence satisfies the user expectations (or background knowledge). On the contrary, if results do not confirm expectations, a closer look at data and at the obtained patterns is mandatory and some changes are made to some of the steps of the analysis process [26]. Thus, efficient support for such iteration is one important development topic in KDD.

These are the premises that lead Imielinski and Mannila in [12] to propose the idea of *inductive database*, a general-purpose database in which both the data and the patterns can be represented, retrieved and manipulated.

1.1 A Theory of Inductive Databases

Many data mining tasks can be described as the search for interesting and frequently occurring patterns from the data. In other terms, we are given a class $\mathcal{L}$ of patterns or sentences from a language that describe properties of the data, and we are given a database d. We can specify whether an interesting pattern $p \in \mathcal{L}$, occurs frequently enough in d [17].

That is, the generic data mining task is to find the set Th

$Th(d, \mathcal{L}) = \{p \in \mathcal{L} | p \text{ is frequent in } d \wedge p \text{ is interesting}\}$.

Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05)
1529-4188/05 $20.00 © 2005 **IEEE**

This point of view has been used in discovering integrity constraints from databases [14, 15]. An alternative formalism [19] would be to consider a language $\mathcal{L}$ of sentences and view a data mining task as the problem of finding the set of sentences in $\mathcal{L}$ (the theory of the data) that are sufficiently true (i.e., probabilistically) in the data set d. In addition, discovered sentences must fulfill the user's other criteria for interestingness (defined by some evaluation functions expressed by some query constraints or predicate q).

$$Th(d, \mathcal{L}, q) = \{p \in \mathcal{L} | q(p, d)\}.$$

The predicate q indicates whether a sentence of the language is interesting. Its fulfillment in d for a set of discovered patterns p means that it defines the theory of an interesting subgroup of d.

According to this formalization, solving an inductive query needs for the computation of every pattern which satisfies the constrained query q. This model is quite general: beside the itemsets, and sequences, (which are local patterns, with mainly a descriptive function) $\mathcal{L}$ can denote, e.g., the language of partitions over the data set (clusters) or the language of decision trees on a collection of attributes (classifiers). In these cases, classical constraints specify some function optimization. Though the completeness assumption can be satisfied for most of the local pattern discovery tasks (itemsets, sequences, clusters) it is generally necessary to adopt some heuristics or incomplete techniques to compute sub-optimal solutions for the classification task. For the issues related to query optimizations refer to Section 2.1.

2 Query Languages

Imielinski and Mannila pointed out that, within the framework of inductive databases, the process of Knowledge Discovery in Databases should be considered an extended querying process. In their words: *"From the user point of view, there is no such thing as real discovery, just a matter of the expressive power of the available query languages"*. This implies that, in order to discover nuggets of knowledge in databases or datawarehouses, inductive databases must provide powerful and universal query language that can deal either with raw data or patterns and that can be used throughout the whole KDD process across many different applications. An important property that such a query language should satisfy is the closure property, which means that the result of the application of a query to an instance of the inductive database is again an instance of an inductive database. It is necessary to guarantee that the discovery process can be performed in a "closed loop", i.e., that it can be applied on the results produced by previous queries.

At a first glance, a *query language* for an inductive database is an extension of a database query language that includes primitives for supporting every step of the KDD process.

These primitives might concern: (1) the selection of data to be mined (e.g., via standard queries but also by means of sampling, discretization, supporting multi-dimensional data manipulation and multiple data types), (2) the specification of the type of patterns to be mined (e.g., various kind of descriptive rules, clusters or predictive models), (3) the specification of the background knowledge (e.g., the definition of a concept hierarchy), (4) the definition of constraints that the extracted patterns must satisfy. This implies that the language allows the user to define constraints that specify the interestingness (e.g., using measures like frequency, generality, coverage, similarity, etc) on the patterns to be mined. Finally, (5) the post-processing of the generated results, for browsing the collection of patterns (also according to different dimensions), apply selection templates, *cross over* patterns and data, or manipulate results with some aggregate functions.

Some of the structured query languages for data mining and for pattern management include [13, 11, 24]. In [34] a query language for querying multiple sets of rules have been proposed and can be used for post-processing and rule evaluation. In [16] a query language for classification and discretization has been proposed.

When designing logic-based frameworks for data mining, Giannotti et al. have emphasized the possibilities of aggregates in deductive databases [10]. Data manipulation and pattern postprocessing are then performed by some extended Datalog queries while aggregates are used for the integration of the data mining component.

A set of logical primitives for data mining has been studied in [29] for a RDM query language. RDM aims at providing a general query language for database mining and employs patterns in the form of Datalog queries. RDM $-$ Relational Database Mining[8], supports a variety of selection predicates. In addition to providing an operator to manipulate patterns, it also allows to manipulate sets of examples, and provides selection predicates that impose a minimum or maximum frequency threshold. Further primitives in RDM are based on a notion of generality (among patterns) and on coverage (w.r.t. specific examples). RDM queries have been optimized in [30] (RDM's execution mechanism integrates the versionspace approach by Tom Mitchell with the levelwise algorithm).

2.1. Query Optimization Strategies

Scaling algorithms implementing the execution strategy of a query remains important for effective solutions in large data sets, in highly multi-dimensional data and in "dense" datasets (where data are very much correlated) that lead to an exponential grow of search spaces. Good query op-

timization strategies are essential to obtaining on-line response time. The new promising approaches to mining will become really effective only when efficient optimizers for the mining languages will be available, i.e., if it will be possible to execute a query exploiting the available informations, such as the properties of constraints in the query, the schema of the database, the indices or the results of previously executed queries.

In past years, data mining researchers made a big effort to exploit constraints occurring in the query, i.e., for query optimization design. The constraints can be exploited to guide the mining process, by pushing them deeply in the mining algorithms, in order to prune the search space of the patterns as early as possible. In fact, under the completeness assumption, it is well known that a "generate and test" approach is generally practically unfeasible. First, because it first enumerates all the patterns and then tests on them the constraints; second, because enumerating all the solutions has an intrinsic exponentiality in complexity and constraints are evaluated on huge volume of data. Thus, researchers have designed query execution plans for important primitive constraints for patterns such as itemsets and sequences (frequency or syntactic constraints, i.e., constraint directly evaluable on the patterns themselves) [33, 25, 9]. Starting from the work in [25] researchers have done a classification of constraint properties and designed some efficient solving strategies according to these properties, e.g., anti-monotonicity, monotonicity, succinctness.

Briefly, an anti-monotone constraint allows to reduce the search space because if it fails for a pattern p in a given hierarchical relation (e.g., the subset-of relation on sets) it certainly fails also for all the patterns p' that descend from p in the hierarchy. A monotone constraint allows to avoid the constraint check for all the patterns p' that descend from a pattern p on which the check succeeds. A succinct constraint is a constraint that allows to identify the valid portion of the search space in advance.

Along with constraint-based data mining, the concept of condensed representation has emerged as a key concept for inductive querying. The idea is to compute a lossless representation of the result of a query, that would allow at the same time an efficient derivation of the complete set of pattern results and a more efficient way to manage the result (given the restricted volume and the elimination of many patterns, considered redundant) [18, 28, 35].

Due to the computational cost of most data mining queries, optimizing these queries is a major issue. Optimization is possible at the query level or for a sequence of queries.

Single queries. Optimization of a single query can be done in several ways. Effective solutions for primitive constraints have to be designed, first. Many effective algorithms make use of the monotonicity or anti-monotonicity of such constraints, as in [5] and as we have seen above. Some recent algorithms, based on the properties of Galois connections, enable the extraction in dense and highly-correlated data [27]. Second, we still need to study in detail the properties of user defined constraints and their interactions. Third, using the background knowledge in the database (e.g., hierarchies and data dependencies) we might increase both the efficiency and the quality of the extraction phases.

Sequences of queries. Knowledge discovery is an iterative process and many of the queries formulated to the database mining engine will be related. Indeed, one can imagine that various queries are similar except possibly for some parameters such as thresholds, data sets, pattern constraints, etc. The relationships among consecutive queries posed to the data mining engine can provide ample opportunities for optimization. The situation can be afforded in a similar way as views are dealt in databases – they are intensional data defined in the system and might be computed differently according to which data access method is currently most promising in the system. The optimizing strategy could be similar to the one used in data warehousing, where some queries are stored (the materializations) and later used to solve other queries [7]. The advantage of materialization is that new queries are answered much faster whereas the disadvantage is that one needs to recompute or update them whenever something changes in the underlying database. The above cited strategy is known as query rewriting. Query rewriting is usually performed by query optimizers because the execution plan of the DBMS for the query in the rewritten form is better in terms of execution costs than for the original query. In [22] query rewriting has been exploited in an optimizer that discovers equivalence between itemset mining queries. In these cases, no further computation would occur to answer the new query. In fact, it is possible to know in advance which elements are common in the query results without making access the database to evaluate constraints again. In [20] an "incremental" approach to mining with queries, originally proposed in [3], is evaluated. This approach aims at reducing the time spent for performing a series of subsequent queries in which the element that changes incrementally in queries are the constraints specified by the user. In these cases, the optimizer recognizes if the current query is contained in a previous one. In these cases some "incremental" algorithms are launched in order to "adjust" previous result to current query. Incremental algorithms, as described in [21] recompute only the portion of the result that is foreseen to change in new query but leave unchanged the rest, again saving huge workload. Therefore, we suggest that the execution plan of a constraint-based query takes into consideration also the results of previous queries, already executed and readily available.

3. Integration of Different Data Mining Patterns

Inductive databases are intended to be general purpose databases in which both source data and mined patterns can be represented, retrieved and manipulated; however, the heterogeneity of models for mined patterns makes difficult to realize them. In [23] we explored the feasibility of using XML as the unifying framework for inductive databases, introducing a suitable data model called XDM (XML for Data Mining). XDM is designed to describe source raw data, heterogeneous mined patterns and data mining statements, so that they can be stored inside a unique XML-based inductive database. XDM allows the management of semi-structured and complex patterns thanks to the semi-structured nature of the data that can be represented by XML. In XDM the pattern definition is represented together with data. This allows the "reuse" of patterns by the inductive database management system. In particular, XDM explicitly represents the statements that were executed in the derivation process of the pattern. The flexibility of the XDM representation allows extensibility to new pattern models and new mining operators: this makes the framework suitable to build an open system, easily customized by the analyst. Similar functionalities are available also in the XML-based framework proposed in [2] while [4] provides a framework for extracting knowledge from XML documents.

Extensibility and flexibility of a pattern management system are features proposed also in [6] in which object oriented modeling techniques have been applied to obtain a uniform model of a pattern management database. In [31] UML has been applied, instead.

4. Standards

CRoss-Industry Standard Process for Data Mining (http://www.crisp-dm.org) develops a (tool neutral) application and process for the KDD Process. It uses for data access SQL interfaces (ODBC, JDBC). The model generation is described by JDM (a Java Data Mining API that supports primitives to build a data mining model, score it, manage data and meta-data), the SQL/MM package for data mining (part 6) and OLEDB/DM (specific of Microsoft SQL-Server). As a model and pattern representation it allows also PMML (Predictive Model Mark-up Language), an industry standard, supported by many commercial tools, for representing the output of a data mining result.

XML for Analysis (http://xmla.org) is a set of XML message interfaces that use the industry standard SOAP to define the data access interaction between the client application and the analytical server on Internet.

OLE DB for DM is an SQL-like interface and an extension of Microsoft Analysis Server for supporting data mining functionalities. It uses PMML for model description. The main advantage of using OLE DB for DM is that the client application, by issuing SQL commands is able to interact with the data mining server in a rather plain way.

SQL/MM (an SQL-3 package for DM) defines SQL objects for representing, applying and gathering results of data mining models in SQL. In [32] an extensive work has been done to experiment different architectural alternatives for integrating mining with a standard relational database system. Many features of the new SQL standard have been exploited, such as the BLOB and ARRAY data structures, SQL-invokable functions such as the user defined functions, table functions and recursive queries. Thanks to these advanced features inside the SQL statements, the authors claim that improvements of orders of magnitude have been gained.

The **eXtensible Markup Language** (XML) has rapidly become an important standard for representing and exchanging information (see, for instance, the important initiative of XMLW3C). The Predictive Model Markup Language (PMML) is an XML extension which can be used to represent most of the types of patterns. In addition, there are other tools which deal with the mining of XML data. For instance, XML Miner (http://www.metadatamining.com/) searches for relationships within XML code to predict values using fuzzy-logic rules. XML Rule takes the output generated by XML Miner or designed by hand and applies them to web sites.

4.1 Commercial systems

There are a lot of commercial tools for data mining available on the market, namely, among the most renown ones, Clementine by ISL, Darwin by Thinking Machines, Enterprise Miner (EM) by SAS Insitute, Intelligent Miner for Data (IM) by IBM and Pattern Recognition Workbench (PRW) by Unica Technologies, Data Mining Suite by Salford Systems, (Teradata, NCR), Oracle Data Miner in Oracle 10g. They differ for the interface to the user and the type and number of algorithms they offer and a few aspects concerning the querying capabilities. A more detailed analysis of these tools can be found in [1].

5. Conclusions

In this paper we briefly overviewed the state of the art of the research in database support for KDD. We discussed on the theory and definition of an inductive database, constraint-based query languages and their optimization approaches, APIs for data mining, standards and commercial

system support to data mining. This overview puts in evidence that the field is still rapidly evolving, especially towards integration with XML and the WEB environment.

6 Acknowledgements

I with to thank J-F Boulicaut for many useful discussions during the development of the *cInQ* project on Inductive Databases.

References

[1] D. W. Abbott, I. P. Matkovsky, and J. F. E. IV. An evaluation of high-end data mining tools for fraud detection. In *IEEE Int. Conf. on Systems, Man, and Cybernetics*, San Diego, CA, October 1998.

[2] P. Alcamo, F. Domenichini, and F. Turini. An xml based environment in support of the overall kdd process. In *FQAS*, 2000.

[3] E. Baralis and G. Psaila. Incremental refinement of mining queries. In *Proc. of the 1st Int'l Conf. on Data Warehousing and Knowledge Discovery*, Firenze, Italy, 1999.

[4] D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. L. Lanzi. Discovering interesting information in xml data with association rules. In *ACM SAC*, 2003.

[5] C. Bucila, J. Gehrke, D. Kifer, and W. White. Dualminer: a dual-pruning algorithm for itemsets with constraints. In *Proc. of the 8th ACM SIGKDD*, 2002.

[6] B. Catania, M. Maddalena, Mazza, E. Bertino, and S. Rizzi. A framework for data mining pattern management. In *Proc. of ECML-PKDD 2004*, Pisa, Italy, Sept. 2004.

[7] S. Chaudhuri, S. Krishnamurthy, S. Potarnianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of 11th ICDE*, March 1995.

[8] Dzeroski, N. Lavrac, and S. D. (Ed.). *Relational Data Mining*. Springer Verlag, NY, 2001.

[9] M. M. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In *Proc. VLDB'99*, 1999.

[10] F. Giannotti, G. Manco, and F. Turini. Specifying mining algorithms with iterative user-defined aggregates: A case study. In *PKDD*, 2001.

[11] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A data mining query language for relational databases. *In Proceedings of SIGMOD-96 Workshop on Research Issues on Data Mining and knowledge Discovery*, 1996.

[12] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Coomunications of the ACM*, 39(11):58–64, November 1996.

[13] T. Imielinski, A. Virmani, and A. Abdoulghani. Datamine: Application programming interface and query language for database mining. *KDD-96*, pages 256–260, 1996.

[14] J. Kivinen and H. Mannila. Approximate dependency inference from relations. *Theoretical Computer Science*, 149(1), 1995.

[15] W. Kloesgen. Efficient discovery of interesting statements in databases. *Journal of Intelligent Information Systems*, 4(1), 1995.

[16] P. Lanzi and G. Psaila. A relational database mining framework with classification and discretization. *SEBD-99 Sistemi Evoluti per Basi di Dati*, June 1999.

[17] H. Mannila. Methods and problems in data mining. In *Proc. of Int. Conf. on Database Theory*, Delphi, Greece, Jan. 1997.

[18] H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In *Proc. of the 1st ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 1996.

[19] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3), 1997.

[20] M.Botta, R.Esposito, A.Gallo, and R.Meo. Optimizing inductive queries in frequent itemset mining. Technical Report RT79-2004, Università di Torino, Italy, May 2004.

[21] R. Meo. Optimization of a language for data mining. In *Proc. of the 2003 ACM Symposium on Applied Computing*, Melbourne, Florida, 2003.

[22] R. Meo, M. Botta, and R. Esposito. Query rewriting in itemset mining. In *6th Int. Conf. on Flexible Query Answering Systems*, 2004.

[23] R. Meo and G. Psaila. Toward xml-based knowledge discovery systems. In *IEEE Int. Conf. on Data Mining*, 2002.

[24] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proceedings of the 22st VLDB Conference*, Bombay, India, September 1996.

[25] R. Ng, L. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In *Proc. ACM SIGMOD'98*, pages 13–24, Seattle, WA, 1998.

[26] B. Padmanabhan and A. Tuzhilin. A belief-driven method for discovering unexpected patterns. In *KDD*, 1998.

[27] F. Pan, G. Cong, A. K. Tung, J. Yang, and M. J. Zaki. Carpenter: Finding closed patterns in long biological datasets. In *Proc. 8th ACM SIGKDD*, 2003.

[28] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 1(24), 1999.

[29] L. D. Raedt. A logical database mining query language. In *ILP*, 2000.

[30] L. D. Raedt, M. Jaeger, S. D. Lee, and H. Mannila. A theory of inductive query answering. In *ICDM*, 2002.

[31] S. Rizzi. Uml-based conceptual modeling of pattern-bases. In *Proc. of the Intl. Workshop on Pattern Representation and Management*, Heraklion, Hellas, 2004.

[32] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. *Data Min. Knowl. Discov.*, 4(2/3), 2000.

[33] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. KDD'97*, Newport Beach, California, 1997.

[34] A. Tuzhilin and B. Liu. Querying multiple sets of discovered rules. In *Proc. 8th ACM SIGKDD*, 2002.

[35] M. J. Zaki. Generating non-redundant association rules. In *Proc. of 5th Int. Conf. on KDD*, 2000.