On Querying Inductive Databases to Mine Decision Rules

Omar Nieva-García and Edgard Benítez-Guerrero

Laboratorio Nacional de Informática Avanzada, A. C.
Rebsamen No. 80, Col. Centro, CP 91000, Xalapa, México
ong0027@lania.edu.mx, ebenitez@lania.mx

Abstract. This paper introduces the MINE DECISION RULE extension to SQL for mining classification rules. It allows the user to express his/her mining requirements and to use the resulting rules to classify unseen data. To enable the evaluation of an inductive query Q incorporating a MINE DECISION RULE expression, a typical object-relational algebra has been augmented with the MineDR operator to mine decision rules. To evaluate Q, it is first translated into a query tree with nodes containing operators in this augmented algebra, then the query tree is transformed into an execution plan which is finally executed. A prototype system supporting our approach is also presented.

1 Introduction

The huge amounts of data that are currently produced in digital format represent a challenge for finding useful information. Knowledge Discovery in Databases (KDD) is the non-trivial process of identifying valid, novel, potentially useful, and understandable knowledge (in the form of patterns) in data [1]. The extracted knowledge can then be used to characterize the data or to classify new, unseen data. KDD is an iterative and interactive process with several steps: understanding of the problem domain, data transformation, pattern discovery, and pattern evaluation and usage. Data Mining techniques are applied to discover patterns from raw data. In this paper we are interested in extracting classification (or decision) rules of the form IF-THEN to classify new, uncategorized data into a pre-defined set of classes. These rules then create a classification model for each class based on attributes values. For instance, a rule might say that if weather outlook is overcast then one can play Golf.

There is currently a large number of tools to help the analysts in the KDD process. However, they fail in supporting the complete KDD process adequately: analyzing data is a complicate job because there is no framework to manipulate data and patterns homogeneously. Recently, Inductive Databases (IDBs) have been proposed to remedy this situation. In this framework, a database contain, in addition to the raw data, (implicit or explicit) patterns about the data [2]. The discovery of patterns can then be viewed as a special kind of database querying and, in this context, query languages and associated query evaluation and

© Jesús Olivares, Adolfo Guzmán (Eds.)
Data Mining and Information Systems.
Research in Computer Science 22, 2006, pp. 55-65

optimization techniques are being proposed. Current research in this area has focused on inductive querying of association rules [3, 4], sequential patterns [5] and clusters [6].

The expression and evaluation of queries to mine classification rules have been partially studied on the inductive database context. Some languages have been proposed, such as DMQL [7], which provides primitives for extracting rules and other kinds of patterns, AXL [8] that gives a user the possibility of extending SQL to introduce complex algorithms such as data mining functions, and DMX [9] that also provides expressions to support a variety of mining techniques, including rule induction. These languages are important because they introduce decision rule mining into the traditional database framework. However, in these proposals, it is not clear how a query is processed, how the extracted rules can be manipulated and how they can be used to classify new data.

In this paper, we propose the MINE DECISION RULE extension to SQL for mining classification rules. It enables the user to express his/her rule mining requirements such as the data source and the constraints that a rule must satisfy to be considered in the final result. In order to manipulate homogeneously data and rules, a typical object-relational data model is used [10]. For processing queries using MINE DECISION RULE, we have extended a modified version of the Object-Relational algebra presented in [10] with the MineDR operator (Ξ) to mine decision rules. This operator help us to produce and process algebraic expressions to manipulate data and decision rules in the same framework. To experiment our approach, we have implemented the *DRMiner* prototype system to show the capabilities of our language and test query processing techniques.

This paper is organized as follows. Section 2 overviews related work. Section 3 briefly describes the OR model and introduces the elements extending this model to represent decision rules. Section 4 presents MINE DECISION RULE. Section 5 summarizes the OR algebra and explains the MineDR operator. Section 6 describes the DRMiner prototype system. Finally, Section 7 concludes this paper and introduces our future work.

2 Related Work

Extracting decision rules from small datasets is a problem that has been studied for years. However, mining rules from large databases poses new challenges. In order to discuss relevant related work, we have classified it in two categories: rule learning algorithms, and inductive query languages and processing.

Research on rule learning algorithms has traditionally focused on improving the heuristic search and the functions for rule evaluation. In general, algorithms follow a covering strategy, i.e., an algorithm searches for a rule that explains a part of its training instances (pre-classified data), separates these instances and repeats the search for a rule until no instances remain. Popular rule learning algorithms are R1 [11], PRISM [12], CN2 [13], PFOIL [14] and RIPPER [15]. A useful analysis of these algorithms can be found in [16].

Inductive query languages (such as MSQL [4] and MINE RULE [3]) have mainly focused on association rule mining. Regarding the extraction of decision rules, there are three proposals: DMQL [7], AXL [8] and DMX [9]. DMQL is a language providing expressions to carry out an extensive set of data mining tasks, and in particular it allows the user to generate rules to classify data according to one or more attributes. It also allows the user to select and filter source data from a table. However, DMQL does not provide support for rule filtering and other post-processing operations (such as cross-over) and, for this, ad-hoc tools have to be externally provided.

Another proposal is the AXL language. It allows the user to add some extensions to SQL for expressing data mining tasks such as classification. By defining a new aggregate function to implement a classifier, a classification technique can be expressed in a SQL language expression. However, its main disadvantage is that the mining algorithm has to be implemented, involving significant code rewriting.

DMX (Data Mining Extension), like DMQL, provides several Data Mining techniques. DMX is divided on a Data Definition Language (DDL) and a Data Manipulation Language (DML). Using the DDL, the user has to define the data model schema (using specific data types) and the proper algorithm to use, according to the data mining task. The user then utilizes DML sentences to handle the mining model and to perform prediction tasks. Let us remark that the resulting mining model is a "black box", unless the user queries its description using specific DML sentences.

There are other works that are useful to understand the KDD process and the IDBs framework. These works are related to efficient mapping of patterns in memory [17], development of general primitives for data mining tasks on query languages [18–20] and development of a formal theory for IDBs [21, 22].

3 Data and Rules Model

This section introduces the model for data and decision rules. To represent them homogeneously, the Object-Relational (OR) data model presented in [10] is used. In the following, the OR model is briefly introduced and then the data types proposed to represent rules are explained.

The basic components of the OR model are types. An OR database schema consists of a set of row types $R_1, \ldots, R_m$, and each attribute in a row type is defined on a certain type, which can be a built-in type, an abstract data type (ADT), a collection type, a reference type or another row type. An object-relational database D on database scheme OR is a collection of row type instance sets (called OR tables) $ort_1, \ldots, ort_m$ such that for each OR table ort_i there is a corresponding row type RT_i, and each tuple of ort_i is an instance of the corresponding row type R_i.

Let us consider for instance the database shown in Figure 1. It contains two tables: the Golf table and Xtest table. The Golf table stores a set of weather conditions that can be used to decide if one can play Golf or not. Its row type

Golf

outlook	temperature	humidity	windy	play
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
rainy	mild	normal	FALSE	yes
sunny	mild	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	TRUE	no

Xtest

outlook	temperature	humidity	windy
sunny	hot	high	FALSE
sunny	hot	high	TRUE
overcast	hot	high	FALSE
rainy	mild	high	FALSE
overcast	cool	normal	TRUE
sunny	mild	high	FALSE
sunny	cool	normal	FALSE
rainy	mild	normal	FALSE
sunny	mild	normal	TRUE
overcast	mild	high	TRUE
overcast	hot	normal	FALSE
rainy	mild	high	TRUE
overcast	hot	normal	FALSE
rainy	mild	high	TRUE

Fig. 1. Example database

is composed by the attributes *outlook*, *temperature*, *humidity*, *windy* and *play* (the attribute class) which are all of atomic types. The Xtest table stores data related to weather conditions but without specifying a class for each tuple.

The result of the processing of the MINE DECISION RULE expression is a table named by default *Decision-Rules* which has the following row type:

decision-rule (idrule: *integer*, predicate: *dr-pattern*, class: *string*, support: *float*, confidence: *float*)

where *idrule* is a unique identifier for a rule, *predicate* is the "body" of the rule to predict a class (of type *dr-pattern* which will be defined later), *class* is the label to be assigned and *support* and *confidence* are two accuracy measures of each rule. The type *dr-pattern* is an Abstract Data Type (ADT) defined as:

dr-pattern(Equalityset:set(Equality), Ordered:boolean,
 create: Func(boolean,set(Equality),boolean),
 get-rule: Func(string),
 exist: Func(boolean,string))

A *dr-pattern* has two attributes: *Equalityset* and *Ordered*. *Equalityset* is a non-empty set of values of type *Equality*, where each *Equality* represents a selector of a rule, e.g. <humidity = "high">. The value of the *Ordered* attribute indicates the form in which the rules are tested against the data (more on this in Section 6). There are three functions defined for *dr-pattern*. The *create* function is a constructor that takes as input a set of *Equality* and a boolean value and returns a boolean value indicating success or failure. The *get-rule* function generates as a string the body of a conjunctive rule from the selectors in *Equalityset*. Finally, the *Exists* function searches a string inside *Equalityset* and returns a boolean value indicating the success or failure of the search.

The table shown in Figure 2 contains a set of rules describing the behavior of the Golf data. For instance, rule number one says that when attribute *outlook* is

Num	Predicate	Class	Support	Confidence
1	(<outlook = 'overcast'>, T)	yes	0.28	1.00
2	(<humidity = 'normal'> <windy = 'false'>, T)	yes	0.28	1.00
3	(<temperature = 'mild'> <humidity = 'normal'>, T)	yes	0.14	1.00
4	(<outlook = 'rainy'> <windy = 'false'>, T)	yes	0.21	1.00
5	(<outlook = 'sunny'> <humidity = 'high'>, T)	no	0.21	1.00
6	(<outlook = 'rainy'> <windy = 'true'>, T)	no	0.14	1.00

Fig. 2. A table containing a set of decision rules

overcast then the predicted class is *yes* with a *support* of 0.28 and a *confidence* of 1.0, i.e., one can play Golf.

4 The Mine Decision Rule Extension to SQL

This section introduces our MINE DECISION RULE extension to SQL for extracting classification rules. We have considered some features of the MINE RULE language [3] in the design of MINE DECISION RULE: (a) selection of relevant data, (b) definition of specific structures and (c) definition of conditions to filter rules. The syntax is as follows:

```
MINE DECISION RULE [<target>]
WITH <attribute> AS CLASS
FROM <table> | <sub-query>
[WHERE <conditions>]
```

The MINE DECISION RULE clause produces a new table that can be optionally renamed as indicated by < *target* >. The WITH < *attribute* > AS CLASS clause specifies the < *attribute* > that contains the classes to be predicted. The FROM clause defines the data source (a table or a subquery) for decision rule mining. Finally, the WHERE clause is optional and let the user specify < *conditions* > to filter a set of rules to get only those of interest.

In the following, we present some examples to show how MINE DECISION RULE can be used to express different queries. First, let us consider the query *Retrieve a set of decision rules to know when one can play Golf (or not) considering play as attribute class* (**Q1**).

```
MINE DECISION RULE Xmodel
WITH play AS CLASS
FROM Golf
```

In this inductive query, the resulting table is renamed as *Xmodel*. The WITH clause specifies the attribute that has the classes to be predicted, *play* in this case. This line is mandatory because rules are built around this attribute. In this example, the FROM clause defines as source the Golf table. With this query, the user retrieves all possible rules with their respective support and confidence.

Now consider the query *Retrieve a set of decision rules based on attributes outlook and windy, to know when one can play Golf or not* (**Q2**).

```
MINE DECISION RULE Xmodel
WITH play AS CLASS
FROM ( SELECT outlook, windy, play FROM Golf )
```

In this case a sub-query is used to extract a dataset and the rules will consider only the attributes *outlook* and *windy*. This is specified in the FROM clause.

Let us suppose now that a domain expert wants to consider the query *Retrieve a decision rules set with the minimum support of 0.25, to know when we can play Golf or not* (**Q3**)

```
MINE DECISION RULE Xmodel
WITH play AS CLASS
FROM Golf
WHERE support >= 0.25
```

The resulting table is filtered in the WHERE clause to retrieve the rules having a computed support equal to 0.25 or above.

Now, let us consider the query *Classify the data on the Xtest table based on rules extracted from the Golf table, considering play as class attribute* (**Q4**).

```
SELECT *
FROM Xtest AS XT, ( MINE DECISION RULE
                    WITH play AS CLASS
                    FROM Golf ) AS XM
WHERE PredictionJoin( XT, XM )
```

This query shows that it is possible to join (using a the PredictionJoin function) a table T containing uncategorized data (Xtest in this example) with a table TR (that can be the result of the evaluation of a MINE DECISION RULE expression) containing a set of rules. The result of a query of this kind is a table containing the tuples of T already classified by the rules in TR.

5 Query Processing

For processing inductive queries using MINE DECISION RULE, we have extended the OR algebra introduced in [10] with the MineDR operator to mine decision rules. In the following, we briefly explain the OR algebra and then introduce MineDR.

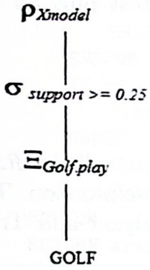
Expressions in the OR algebra consist of OR operands and OR operators. An OR operand is either an OR table, a path expression or the result of another operation. In this section, we simply use the term "table" to refer to all the possible operands, as long as no distinction is necessary. The set of OR operators consists of the object-relational counterparts of basic relational operators – select (σ), join ($\bowtie$), Cartesian product($\times$), project (π) –, set operators – union ($\cup$), difference ($-$), intersection ($\cap$)), nest (ν), unnest (υ)–, and special operators to handle row type object identity – map(ϕ) and cap(δ) –. To the original operator set, we incorporate the operators group-by (F) and rename (ρ).

To this algebra we have added the MineDR operator to mine decision rules, which is noted as follows:

$$\Xi_{A_n}(r)$$

where r is an input table with schema $(A_1, A_2, ..., A_{n-1}, A_n)$ such that A_k $(k = 1 \ldots n-1)$ are general attributes and A_n is a special attribute representing a label or class for each tuple in r. The MineDR operator computes as result a table of row type *decision-rule* (see Section 3) containing a set of rules extracted from table r.

To illustrate the concepts and operations related to the MineDR operator, the evaluation of queries Q_3 and Q_4 is explained in the following. Let us consider first the query Q_3, *Retrieve a decision rules set with the minimum support of 0.25, to know when we can play Golf or not.* In this query, the classification is based on the attribute *play*.



Xmodel

...	Predicate	Class	Suport	Confidence
3	outlook = "overcast"	yes	0.285...	1.00

decision-rule

...	Predicate	Class	Suport	Confidence
3	outlook = "overcast"	yes	0.285...	1.00

decision-rule

...	Predicate	Class	Suport	Confidence
1	outlook = "sunny" | humidity = "high"	no	0.214...	1.00
2	outlook = "rainy" | windy = "TRUE"	no	0.142...	1.00
3	outlook = "overcast"	yes	0.285...	1.00
4	humidity = "normal" | windy = "FALSE"	yes	0.214...	1.00
5	temperature = "mild" | humidity = "normal"	yes	0.071...	1.00
6	outlook = "rainy" | windy = "FALSE"	yes	0.071...	1.00

Fig. 3. Query Tree for Q_1

Figure 3 shows the query tree for Q_3. As input to MineDR (Ξ), it is necessary to indicate the name of the table containing the data set (in this case the Golf table) from which rules will be generated, and the name of the attribute classifying the data (*play* in this example). Next, MineDR produces a table containing a set of decision rules. In this query, the WHERE clause is used to filter the rules to obtain only those that have a support equal to 0.25 or above, and thus it is necessary to introduce the select (σ) operator in the query tree.

Now consider the query Q_4 where a set of decision rules is applied: *Classify the Xtest table based on rules extracted from the Golf table, considering play as the class attribute.* In the query tree in Figure 4, a new table is produced by the MineDR operator by extracting decision rules from the GOLF table. At the left side of the tree is the Xtest table with tuples representing instances to be classified. To obtain the result of the query (classified instances), the Prediction Join of the resulting table of MineDR with Xtest is executed. At the top of the

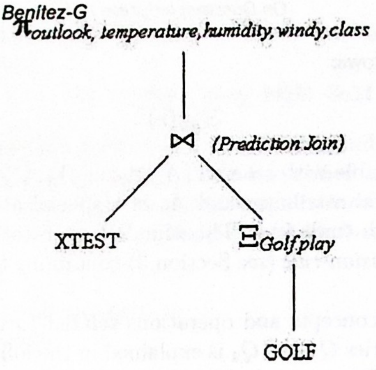


Fig. 4. Query Tree for Q_3

query tree, it is possible to see that all attributes of Xtest plus an attribute containing the assigned class have been projected.

6 The DRMiner Prototype System

We have designed and developed in Java a prototype system, called *DRMiner*, to process queries considering the MINE DECISION RULE expression. The main components of DRMiner are: (a) User interface, (b) Analyzer and Translator, and (c) Evaluation engine (See Figure 5).

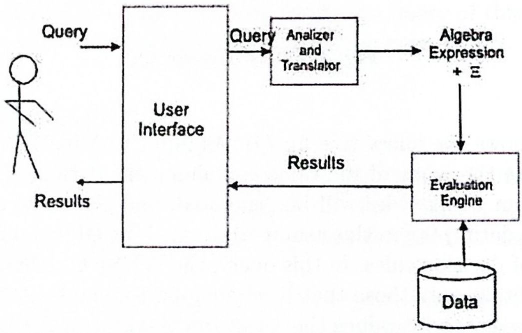


Fig. 5. DRMiner System Architecture

Once a user types a query, the Analyzer verifies its syntax and translate it into a query tree with nodes containing operators in the OR algebra augmented with MineDR. This query tree is passed to the evaluation engine, which uses a rule learning algorithm to implement MineDR to classify data and returns a table containing a set of decision rules as a result of the query. Our evaluation engine currently integrates the PRISM [12] algorithm, because it easy to find standard code implementations and documentation on the Internet. However, it is possible to change this algorithm for another accomplishing the same classification task.

```
Function PREDICTIONJOIN (table T, table M)
Xresult = ∅
FOR each tuple t_T = (a_1,a_2,..,a_n) ∈ T
   classified = false; class = null; C = ∅; xtuple=null;
   FOR each tuple t_M ∈ M
      IF satisfies(t_T,t_M) THEN
         class = t_M.class
         classified = true
         IF ¬t_M.predicate.ordered THEN add(C,class)
         ELSE break
      ENDIF
   ENDFOR
   IF ¬t_M.predicate.ordered AND classified THEN
      class = solve_controversy(C)
   ENDIF
   xtuple = (a_1,a_2,..a_n, class)
   add(Xresult,xtuple)
ENDFOR
RETURN Xresult
```

Fig. 6. Prediction Join Algorithm

The DRMiner evaluation engine also implements a prediction join operation to classify uncategorized data according to a set of rules. Figure 6 shows the algorithm. The input is a table (T) with unseen instances (tuples) and a table (M) with the decision rules to predict the class of each tuple of T. To classify a new tuple, each rule is tested on the tuple and, when the conditions of a rule are satisfied, then a class is found. If rules are ordered then the first class found is assigned to the tuple, otherwise all possible classes are found and then the controversy is solved to assign only one class to the tuple. The variable *xtuple* stores each time a tuple $t_T \in T$ plus its assigned class. If none of rules fires, the algorithm assigns the tuple a *null* class. The result is the $Xresult$ table.

Finally, Figure 7 shows the DRMiner user interface. It is possible to type a query to retrieve a table containing a set of rules. All queries mentioned in this paper can be executed "as is" in *DRMiner*. For instance, in Figure 7, we can see that a basic query to *Retrieve a set of decision based on the Golf table and considering play as attribute class* is introduced and the results are shown.

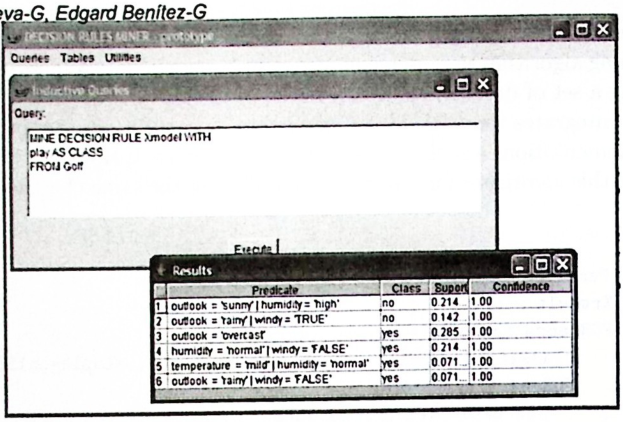


Fig. 7. DRMiner user interface

7 Conclusion and Future Work

This paper introduced the MINE DECISION RULE extension to SQL for mining classification rules. It enables the user to express his/her requirements on the extraction of decision rules from pre-classified data and apply the mined rules over new, unclassified data. To evaluate queries incorporating MINE DECISION RULE, the MineDR operator has been integrated into a typical OR algebra. MineDR takes as input a source table and a specific class attribute, and produces a table containing a set of rules. The DRMiner prototype system has been developed to test our ideas.

Our future work includes research on query optimization. It will be focused on deciding which algorithm, from a possible set of rule learning algorithms, is the most suitable for a classification task, according to data features such as the number of attributes, data volumes and the presence/absence of noise.

Acknowledgements. The authors thank the reviewers of this paper for their useful comments. The work of Mr. Nieva-García has been supported by the Mexican National Council of Science and Technology (CONACYT).

References

1. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: From Data Mining to Knowledge Discovery: An Overview. AAAI/MIT Press (1996)
2. Mannila, H.: Inductive Databases and Condensed Representations for Data Mining. In: Proceedings of the 1997 International Symposium on Logic Programming (ILPS'97), MIT Press (1997) 21–30
3. Meo, R., Psaila, G., Ceri, S.: A New SQL-like Operator for Mining Association Rules. The VLDB Journal (1996) 122–133

4. Imielinski, T., Virmani, A., Abdulghani, A.: Datamine: Application Programming Interface and Query Language for Database Mining. In: Proc. of the 2nd Int'l Conference on Knowledge Discovery and Data Mining (KDD-96). (1996) 256–262
5. Benítez-Guerrero, E., Hernández-López, A.R.: The MineSP Operator for Mining Sequential Patterns in Inductive Databases. MICAI 2006: Advances in Artificial Intelligence, 5th Mexican International Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence **4293** (2006) 684–694 Springer-Verlag.
6. Ordonez, C., Cereghini, P.: SQLEM: Fast Clustering in SQL using the EM Algorithm. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA, ACM (2000) 559–570
7. Han, J., Fu, Y., Wang, W., Chiang, J., Gong, W., Koperski, K., Li, D., Lu, Y., Rajan, A., Stefanovic, N., Xia, B., Zaiane, O.: DBMiner: A System for Mining Knowledge in Large Relational Databases. In: Proc. of the Int'l Conf. on Data Mining and Knowledge Discovery (KDD'96), Portland, Oregon (1996) 250–255
8. Wang, H., Zaniolo, C.: Using SQL to Build New Aggregates and Extenders for Object-Relational Systems. In: Proc. of the 26th International Conference on Very Large Data Bases (VLDB '00), Morgan Kaufmann Publishers Inc. (2000) 166–175
9. Chaudhuri, S., Narasayya, V., Sarawagi, S.: Extracting Predicates from Mining Models for Efficient Query Evaluation. ACM Transactions on Database Systems **29**(3) (2004) 508–544
10. Li, H., Lui, C., Orlowska, M.: A Query Systems for Object-Relational Databases. In: Proc. of the 9th Australasian Database Conference (ADC'98). (1998) 39–50
11. Holte, R.C.: Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. Machine Learning **11** (1993) 63–91
12. Cendrowska, J.: PRISM: An Algorithm for Inducing Modular Rules. International Journal of Man-Machine Studies **27**(4) (1987) 349–370
13. Clark, P., Boswell, R.: Rule Induction with CN2: Some Recent Improvements. In: Proc. Fifth European Working Session on Learning, Springer (1991) 151–163
14. Mooney, R.J.: Encouraging Experimental Results on Learning CNF. Machine Learning **19**(1) (1995) 79–92
15. Cohen, W.W.: Fast Effective Rule Induction. In: Proc. of the 12th Int'l Conference on Machine Learning, Tahoe City, CA, Morgan Kaufmann (1995) 115–123
16. Furnkranz, J., Flach, P.: An Analysis of Rule Learning Heuristics. Technical Report CSTR-03-002, Department of Computer Science, University of Bristol (2003)
17. Raedt, L.D.: A Perspective on Inductive Databases. SIGKDD Explorations Newsletter **4**(2) (2002) 69–77
18. Sattler, K.U., Dunemann, O.: SQL Database Primitives for Decision Tree Classifiers. In: Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM'01), ACM Press (2001) 379–386
19. Hinneburg, A., Lehner, W., Habich, D.: COMBI-Operator: Database Support for Data Mining Applications. In: Proceedings of 29th International Conference on Very Large Data Bases. (2003) 429–439
20. Geist, I., Sattler, K.U.: Towards Data Mining Operators in Database Systems: Algebra and Implementation. In: Proceedings of 2nd International Workshop on Databases (DBFusion 2002). (2002)
21. Boulicaut, J.F., Klemettinen, M., Mannila, H.: Modeling KDD Processes within the Inductive Database Framework. In: Proc. of the First Int'l Conf. on Data Warehousing and Knowledge Discovery (DaWaK '99), Springer (1999) 293–302
22. Raedt, L.D., Jaeger, M., Lee, S., Mannila, H.: A Theory of Inductive Query Answering. In: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02), IEEE Computer Society (2002) 123