Consiglio Nazionale delle Ricerche

Declarative Knowledge Extraction with Iterative User-Defined Aggregates

Fosca Giannotti, Giuseppe Manco

Rapporto
CNUCE-B4-2000-015

Pisa

Declarative Knowledge Extraction with Iterative User-Defined Aggregates

Fosca Giannotti and Giuseppe Manco

CNUCE Institute
Pisa Research Area
Via Alfieri 1
I56010 Ghezzano (PI) - Italy
e-mail : {F.Giannotti, G.Manco}@cnuce.cnr.it

Abstract. We present the notion of Iterative User-Defined Aggregates as an extension of the notion of user-defined aggregates in deductive databases. Such an extension provides a versative mechanism for defining complex aggregation functions, that are not definable as distributive aggregates. As a result, we show how such a mechanism can be applied to the specification of complex data mining tasks as user-defined aggregates. The resulting formalism provides a flexible way to customize, tune and reason on both the evaluation functions and the extracted knowledge.

1 Introduction and Motivations

Most knowledge-intensive data analysis applications require the combination of two kinds of activities: knowledge acquisition, and reasoning on the acquired knowledge according to the expert rules that characterize the business. Data mining techniques are an answer to the first issue, in that they extract from raw data knowledge that is implicit and, more importantly, that is at a higher abstraction level. However, the ability of combining the results of knowledge extraction with expert rules is a key factor of success when building decision support systems. Many applications of the above mentioned kind, in domains such as market basket analysis [9], are challenging applications, for a variety of reasons.

A key point is the management of the KDD process, and its tailoring to specific domains. The role of domain, or background, knowledge is relevant at each step of the KDD process: which attributes discriminate best, how can we characterize a correct/useful profile, what are the interesting exception conditions, etc., are all examples of domain dependent notions. Notably, in the evaluation phase we need to associate with each inferred knowledge structure some quality function that evaluates its information content in the specific domain. More generally, a logically uniform representation of data, domain knowledge and extracted knowledge is required, in order to express easily high-level business rules.

Various proposals in the current literature have shown that the knowledge discovery process can take great advantage of a powerful knowledge-

representation and reasoning formalism [13,14,2,9]. As an example, the notion of *inductive database*, proposed in [1,10], is an attempt to formalize the notion of interactive mining process. An inductive database provides a unified and transparent view of both inferred (deductive) knowledge, and all the derived patterns, (the induced knowledge) over the data. The user does not care about whether he is dealing with inferred or induced knowledge, and whether the requested knowledge is materialized or not. The only detail he is interested in stems in the high-level specification of the query involving both deductive and inductive knowledge. In this direction moves the notion of Inductive Databases to the case of rule-based languages presented in [4,5].

The central point of the paper is about the "declarative" nature of many data mining tasks, since they can be conceived as purely functional operators on a data set. Many data mining algorithms can be defined by means of simple iterative steps, in which each iteration consists in the computation of an aggregate. It is the case, for example, of the Apriori association rule mining algorithm, in which each iteration consists in finding the frequent itemsets from a set of candidate itemsets generated by the previous iteration. We propose a model for declaratively integrating data mining tasks in a deductive environment. Directly specifying the mining algorithms in a deductive environment gives us the oppportunity to apply both database optimizations and application/domain dependent optimizations. We can, for example, directly specify constraints, and apply optimized data structures only when they are needed. Moreover, from an implementation viewpoint, the major drawback of using a purely declarative specification can be overcome using the techniques described in [9], which allow to define and use, by means of the mechanism of external functions, specialized data structures.

In this paper we extend the approach shown in [4,5,9], following two main directions:

- we introduce the concept of *iterative user-defined aggregate* (IUDAs), as an extension of the concept of user-defined aggregate [17]. We define the formal semantics of IUDAs in a Datalog++ framework [8,7,16].
- We use such a notion as a means for formalizing a logically uniform representation of data, domain knowledge and extracted knowledge in a declarative way.

The paper is organized as follows. In section 2 we briefly sketch the mechanism of *user defined aggregates* provided by $\mathcal{LDL}$++. Section 3 describes the generalization of such a notion to the notion of Iterative user-defined aggregates. In section 4 we show how the mechanism is powerful enough to express different phases of the knowledge discovery process. A short final discussion in section 5 remarks both the actual achievements and the future developments that can make the system viable for practical application development.

2 User-Defined Aggregates

In this paper we refer to Datalog++ and its current implementation $\mathcal{LDL}$++, a highly expressive language which includes among its features recursion and a powerful form of stratified negation [16], and is viable for efficient query evaluation [8].

A remarkable capability of such a language is that of expressing distributive aggregates, which are definable by the user [17]. For example, the following rule illustrates the use of a sum aggregate, which aggregates the values of the relation sales along the dimension Dealer:

$$\texttt{supplierTot(Time, Product, sum(Sales))} \leftarrow \texttt{sales(Time, Product, Store, Sales)}.$$

Such rule corresponds to the SQL statement

```
SELECT Time, Product, SUM(Sales)
FROM sales
GROUP BY Time, Product
```

From a semantic viewpoint, the above rule is a syntactic sugar for a program that exploits the notions of nondeterministic choice and XY stratification [3,16,8]. The evaluation of the aggregation rule

$$\texttt{q(aggr(X))} \leftarrow \texttt{p(X, Y)}.$$

exploits the capability of defining "recursive" aggregates, i.e., aggregates inductively defined,

$$f(\{x\}) = g(x) \tag{1}$$
$$f(S \cup \{x\}) = h(f(S), x) \tag{2}$$

by defining the base and inductive cases by means of ad-hoc user-defined predicates `single` and `multi`.

In [17], a further extension to the approach is proposed, in order to deal with more complex aggregation functions. Practically, we can manipulate the results of the aggregation function by means of two predicates `freturn` and `ereturn`.

Example 1 ([17]). The sample variance of a set of values is defined as $s = \frac{1}{n-1}(\sum_i x_i - \frac{1}{n}(\sum_j x_j)^2)$. The aggregate that computes such a measure over a column of a table is the following:

$$\texttt{single(var, X, (X, 1))}.$$
$$\texttt{multi(var, X, (C, N), (C + X, N + 1))}.$$
$$\texttt{freturn(var, (C, N), 1/(N - 1) \times (C - 1/N \times C \times C))}.$$

□

3 Iterative User-Defined Aggregates

The main deficiency of the previous model is the impossibility of defining non-distributive aggregates within datalolg++. In many cases, even simple aggregates require multiple steps over data in order to be computed.

Example 2. The absolute deviation $S_n = \sum_x |\overline{x} - x|$ of a set of n elements is defined as the sum of the absolute difference of each element with the mean value $\overline{x} = 1/n \sum_x x$ of the set. In order to compute such an aggregate, we need to scan the available data twice: first, to compute the mean value, and second to compute the sum of the absolute difference. □

A simple way of coping with the problem of multiple scans over data can be done by extending the declarative rewriting of the aggregation rule

$$\mathtt{q}(\mathtt{aggr}\langle \mathtt{X} \rangle) \leftarrow \mathtt{p}(\mathtt{X}) \tag{3}$$

in order to impose some user-defined conditions for iterating the scan over data. More specifically, the extension we propose can be specified as an iterative function `aggr`

```
aggr(S)
{
    init(res);
    do
        res = f(res, S)
    while (iterate(res));

    return res;
}
```

where f is defined, as usual, as a fold operator

$$f(v, \{\}) = g(v)$$
$$f(v, S \cup \{x\}) = h(f(v, S), x)$$

and `iterate` is a user-defined condition specifying whether the aggregate computation can be stopped. By exploiting recursion, it is easy to define such an iteration directly as an aggregate computation:

```
cagr_p(nil, Aggr, nil, New) ← empty(Aggr, New).

cagr_p(nil, Aggr, X, New) ← next_p(nil, X), X ≠ nil,
                  single(Aggr, X, New), ¬empty(Aggr, _).
cagr_p(I, Aggr, Y2, New) ← next_p(Y1, Y2), Y2 ≠ nil,
                  cagr_p(I, Aggr, Y1, Old), multi(Aggr, Old, Y2, New).
cagr_p(s(I), Aggr, nil, New) ← next_p(X, Y), ¬next_p(Y, _),
                  cagr_p(I, Aggr, Y, Old), iterate(Aggr, Old, New).
```

The main difference w.r.t. the schema shown in the previous section is in the last rule, that specifies the condition for iterating the aggregate computation. The activation (and evaluation) of such a rule is subject to the successful evaluation of the user-defined predicate `iterate`, so that any failure in evaluating it results in the termination of the computation. This guarantees that the proposed schema is conservative: any aggregate specified by `single`, `multi` and `return` is computed in the same way as shown in the previous section.

Rule 3 is then rewritten in the following rules,

$$\begin{aligned}
\mathtt{q(S)} \leftarrow\ & \mathtt{next_p(X,Y)}, \neg\mathtt{next_p(Y,_)}, \mathtt{cagr_p(I,Aggr,Y,C)},\\
& \neg\mathtt{cagr_p(s(I),Aggr,nil,C)}, \mathtt{freturn(Aggr,C,S)}.\\
\mathtt{q(S)} \leftarrow\ & \mathtt{next_p(X,Y)}, \mathtt{Y} \neq \mathtt{nil}, \mathtt{cagr_p(I,Aggr,X,C)},\\
& \mathtt{ereturn(Aggr,C,Y,S)}.
\end{aligned}$$

so that the aggregate produces a final result only when iteration stops.

Example 3. By adopting the above schema, the absolute deviation S_n can be defined in the following way:

$$\mathtt{single(abserr,X,(nil,X,1))}.$$

$$\mathtt{multi(abserr,(nil,S,C),X,(nil,S+X,C+1))}.$$

$$\begin{aligned}
&\mathtt{multi(abserr,(M,D),X,(M,D+(M-X)))} \leftarrow \mathtt{M} > \mathtt{X}.\\
&\mathtt{multi(abserr,(M,D),X,(M,D+(X-M)))} \leftarrow \mathtt{M} \leq \mathtt{X}.
\end{aligned}$$

$$\mathtt{iterate(abserr,(nil,S,C),(S/C,0))}.$$

$$\mathtt{freturn(abserr,(M,D),D)}.$$

Notice how the combined use of `multi` and `iterate` allows to define two scans over the data: the first scan in order to compute the mean value, and the second one in order to compute the sum of the absolute difference with the mean value. □

4 Mining Aggregates

The iterative schema shown so far is common in many data mining algorithms. Usually, a typical data mining algorithm is an instance of an iterative schema. At each iteration, some statistics are gathered from the data, and the termination condition determines whether the extracted statistics are sufficient to the purpose of the task, or no new statistics can be computed instead.

Example 4. The Apriori algorithm for the computation of frequent itemsets is based on the following criteria:

1. at each iteration, frequencies of candidate itemsets are computed, and unfrequent candidate sets are removed. New candidate itemsets are computed by combining the frequent itemsets.

2. The cycle terminates when no further candidate itemsets are generated.

The two items identify the necessary conditions for specifying the computation of frequent itemsets by means of the iterative schema shown before. □

Example 5. The K-means algorithm for the computation of clusters of tuples is defined by the following steps:

1. initially, K random cluster centers are chosen.
2. at each iteration, each tuple is assigned to its most similar cluster center, according to some similarity measure. Hence, cluster centers are recomputed averaging the values of each cluster component.
3. no more iteration is performed if the cluster centers are stable, i.e., if they do not change.

Clearly, the specification of such operations within the iterative schema is immediate. □

Example 6. A general schema for decision-tree classification algorithms is the following:

BuildTree(Node n, Datapartition D, Splitting_criterion $\mathcal{CL}$)
  $A = \mathcal{CL}$.`find_best_classifier_attribute`(D)
  $k = \mathcal{CL}$.`split_dataset`(D, A)
  if $k > 0$
    for each partition D_i, $1 \leq i \leq k$
      create new children n_i of n
      BuildTree(n_i,D_i,$\mathcal{CL}$)
    endfor
  endif

Such a recursive schema can be easily fitted into the above shown iterative schema. Notice in fact that, although the recursive call is made on a partition of the dataset, there are as many recursive calls as the partitions are. □

From these examples it is clear that the iterative schema shown in the previous section is a good candidate for specifying data mining tasks, and, moreover, for integrating them in a deductive environment.

From an implementation viewpoint, the major drawback of using a purely declarative specification can be overcome using the techniques described in [9]. However, directly specifying the mining algorithms in a deductive environment gives us the opportunity to apply both database optimizations and application/domain dependent optimizations. We can, for example, directly specify constraints, and apply optimized data structures whenever they are needed in order to speed-up the computation.

The rest of the section is devoted at showing how the model is suitable enough to cover some important tasks of the data mining process. In particular, we analyze two different tasks: classification and association rule mining. The full version [6] of this paper extensively covers further data mining tasks.

4.1 Classification

We are interested in developing a classification construct based on the notion of user-defined aggregate. We concentrate here on bayesian classification, that is among the most practical approaches to many types of learning problems [11,12]. To summarize, we aim at computing the function

$$\max_c \quad Prob(C = c | A_1 = a_1, \ldots, A_n = a_n)$$

where c is a value of the target attribute and $a_1, \ldots, a_n$ are possible values of the attributes of a relation with schema $\mathbf{R} = \{Z_1, \ldots, Z_k\}$ such that $\{A_1, \ldots, A_n, C\} \subseteq \{Z_1, \ldots, Z_k\}$. By assuming that, for each i, j such that $i \neq j$, A_i and A_j are independent, the above expression is maximized by the same value c that maximizes the expression

$$\max_c \quad Prob(C = c) \cdot \prod_1^n Prob(A_i = a_i | C = c)$$

$Prob(A|B)$ can be estimated as $count(A \wedge B)/count(B)$; hence, we can then define the pair $(Prob(A_i = a_i|C = c), Prob(C = c))$ as a user-defined aggregate[1]. Practically, we define a predicate s as

$$\mathtt{s}(\mathtt{X}_1, \ldots, \mathtt{X}_m, \mathtt{bayes}\langle(\mathtt{P}, [\mathtt{A}_1, \ldots, \mathtt{A}_n], \mathtt{C})\rangle) \leftarrow \mathtt{r}(\mathtt{Z}_1, \ldots, \mathtt{Z}_k).$$

where the variables $\mathtt{X}_1, \ldots, \mathtt{X}_m, \mathtt{P}, \mathtt{A}_1, \ldots, \mathtt{A}_n, \mathtt{C}$ are a (possibly rearranged) subset of $\mathtt{Z}_1, \ldots, \mathtt{Z}_k$ and $\mathtt{r}(\mathtt{Z}_1, \ldots, \mathtt{Z}_k)$ is either an extensional or an intensional predicate. The result of such an evaluation is the set of conditional probabilities of each of the possible values of $\mathtt{A}_i$, given any possible value of C, and a weight P associated to the tuple $\mathtt{A}_1, \ldots, \mathtt{A}_n$.

We can define, as usual, the `single`, `multi` and `freturn` predicates:

$\mathtt{single}(\mathtt{bayes}, (\mathtt{P}, \mathtt{F}, \mathtt{C}), (\mathtt{A}, \mathtt{C})) \leftarrow \mathtt{member}(\mathtt{A}, \mathtt{F}).$
$\mathtt{multi}(\mathtt{bayes}, (\mathtt{P}, \mathtt{F}, \mathtt{C}), (\mathtt{A}, \mathtt{C}_1), (\mathtt{A}, \mathtt{C}_1)).$
$\mathtt{multi}(\mathtt{bayes}, (\mathtt{P}, \mathtt{F}, \mathtt{C}), \_, (\mathtt{A}, \mathtt{C})) \leftarrow \mathtt{member}(\mathtt{A}, \mathtt{F}).$

$\mathtt{multi}(\mathtt{bayes}, (\mathtt{P}, \mathtt{F}, \mathtt{C}), (\mathtt{A}, \mathtt{P}_\mathtt{A}, \mathtt{C}, \mathtt{P}_\mathtt{C}, \mathtt{N}), (\mathtt{A}, \mathtt{P}_\mathtt{A} + \mathtt{P}, \mathtt{C}, \mathtt{P}_\mathtt{C} + \mathtt{P}, \mathtt{N} + 1)) \leftarrow \mathtt{member}(\mathtt{A}, \mathtt{F}).$
$\mathtt{multi}(\mathtt{bayes}, (\mathtt{P}, \mathtt{F}, \mathtt{C}), (\mathtt{A}, \mathtt{P}_\mathtt{A}, \mathtt{C}, \mathtt{P}_\mathtt{C}, \mathtt{N}), (\mathtt{A}, \mathtt{P}_\mathtt{A}, \mathtt{C}, \mathtt{P}_\mathtt{C} + \mathtt{P}, \mathtt{N} + 1)) \leftarrow \neg\mathtt{member}(\mathtt{A}, \mathtt{F}).$

$\mathtt{iterate}(\mathtt{bayes}, (\mathtt{A}, \mathtt{C}), (\mathtt{A}, 0, \mathtt{C}, 0, 0)).$

$\mathtt{freturn}(\mathtt{bayes}, (\mathtt{A}, \mathtt{P}_\mathtt{A}, \mathtt{C}, \mathtt{P}_\mathtt{C}, \mathtt{N}), (\mathtt{A}, \mathtt{C}, \mathtt{P}_{\mathtt{A},\mathtt{C}}, \mathtt{P}_\mathtt{C})) \leftarrow \mathtt{P}_{\mathtt{A},\mathtt{C}} = \mathtt{P}_\mathtt{A}/\mathtt{P}_\mathtt{C}, \mathtt{P}_\mathtt{C} = \mathtt{P}_\mathtt{C}/\mathtt{N}.$

[1] Notice that such an approach fails to deal with 0 probabilities. However, simple corrections can be done, as suggested, e.g., in [12, chapter 6]. For example, we can weight the proposed estimate of $Prob(A|B)$ with prior uniform probabilities.

Let us consider the extensional predicate:

$$\texttt{playTennis}(\texttt{Outlook}, \texttt{Temperature}, \texttt{Humidity}, \texttt{Wind}, \texttt{Play})$$

A simple classifier on such a relation is built by means of the rule

$$\texttt{classifier}(\texttt{bayes}\langle(1, [\texttt{Outlook}, \texttt{Temp}, \texttt{Humidity}, \texttt{Wind}], \texttt{Play})\rangle) \leftarrow \texttt{playTennis}(\texttt{Outlook}, \texttt{Temp}, \texttt{Humidity}, \texttt{Wind}, \texttt{Play}).$$

Example 7. A query $\texttt{classifier}(\texttt{C}, \texttt{F}, \texttt{P}_\texttt{C}, \texttt{P}_\texttt{F})$ against such a database returns as answers tuples such as

(no, sunny, 0.357143, 0.6)	(no, hot, 0.357143, 0.4)
(no, weak, 0.357143, 0.4)	(no, strong, 0.357143, 0.6)
(yes, hot, 0.642857, 0.222222)	(yes, high, 0.642857, 0.333333)
(yes, rain, 0.642857, 0.444444)	(yes, mild, 0.642857, 0.444444)

Such tuples represent a classification model that can be used to classify any new tuple. □

4.2 Association Rules Mining

In [4,9] we defined the patterns aggregate for computing frequent itemsets, which are basically needed for computing association rules from a given relation. Practically, the patterns aggregate is defined in rules

$$\texttt{p}(\texttt{patterns}\langle(\texttt{min\_supp}, \texttt{S})\rangle) \leftarrow \texttt{q}(\texttt{Z}_1, \ldots, \texttt{Z}_\texttt{m}).$$

where S denotes a set of values computed by q. The avaluation of the above rule returns a set of predicates $p(s)$, where $s = \{l_1, \ldots, l_l\}$ is a frequent itemset which satisfies the min_supp minimum support requirement.

Examples of complex queries whithin the resulting logic language are very easy to express.

Example 8. "Find patterns with at least 3 occurrences from the daily transactions of each customer":

$$\texttt{frequentPatterns}(\texttt{patterns}\langle(3, \texttt{S})\rangle) \leftarrow \texttt{transSet}(\texttt{D}, \texttt{C}, \texttt{S}).$$

$$\texttt{transSet}(\texttt{D}, \texttt{C}, \langle\texttt{I}\rangle) \leftarrow \texttt{transaction}(\texttt{D}, \texttt{C}, \texttt{I}, \texttt{P}, \texttt{Q}).$$

By querying $\texttt{frequentPatterns}(\texttt{S})$ we obtain, among the answers, the tuples $(\{pasta\})$ and $(\{pasta, wine\})$. □

Example 9. "Find patterns with at least 3 occurrences from the transactions of each customers":

$$\texttt{frequentPatterns}(\texttt{patterns}\langle(3, \texttt{S})\rangle) \leftarrow \texttt{transSet}(\texttt{C}, \texttt{S}).$$

$$\texttt{transSet}(\texttt{C}, \langle\texttt{I}\rangle) \leftarrow \texttt{transaction}(\texttt{D}, \texttt{C}, \texttt{I}, \texttt{P}, \texttt{Q}).$$

Differently from the previous example, where transactions were grouped by customer and by date, the previous rules group transactions by customer. We then compute the frequent patterns on the restructured transactions

```
transSet(cust1, {beer, chips, jackets, pasta, wine})
transSet(cust2, {beer, chips, col_shirts, jackets, pasta, wine})
transSet(cust3, {beer, brown_shirts, chips, col_shirts, pasta, wine})
```

obtaining, e.g., the pattern ({beer, chips, pasta, wine}). □

The above aggregate can be defined using the iterative schema schown in section 3. The interested reader is referred to the full version [6] of this paper for its definition.

5 Conclusions

Iterative user-defined aggregates extend the expressive power of user-defined aggregates, by providing the basic tool for defining aggregates that need several scans over data. A natural application of such iterative aggregates is in the specification of data mining task. The advantage of such specifications is twofold:

- from a conceptual level, we achieve a semantic integration of domain knowledge and extracted knowledge, by means of a uniform knowledge representation layer supporting both deduction and induction
- from a phisical level, such an integration gives us the opportunity of directly integrating specific optimizations to the application of data mining algorithms, that can be treated no more as "black boxes", but allow interaction in the various phases of the computational process.

Efficiency issues were out of the scope of this paper, and were not specifically studied. A simple yet attractive way of using specialized data mining structures mantaining the current aggregate specification is shown in [9]. Another interesting way of coping with efficiency is that of identifying a set of relevant features that can be transferred into more specialized and efficient languages. As an example, [15] study how to provide relational database systems with the mechanism of user-defined aggregates.

References

1. J-F. Boulicaut, M. Klemettinen, and H. Mannila. Querying Inductive Databases: A Case Study on the MINE RULE Operator. In *Proc. 2nd European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD98)*, volume 1510 of *Lecture Notes in Computer Science*, pages 194–202, 1998.

2. U.M. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI Press/the MIT Press, 1996.
3. F. Giannotti, D. Pedreschi, and C. Zaniolo. Semantics and Expressive Power of Non Deterministic Constructs for Deductive Databases. Technical Report C96-04, The CNUCE Institute, 1996. Submitted.
4. F. Giannotti and G. Manco. Querying Inductive Databases via Logic-Based User-Defined Aggregates. In J. Rauch and J. Zitkov, editors, *Procs. of the European Conference on Principles and Practices of Knowledge Discovery in Databases*, number 1704 in Lecture Notes on Artificial Intelligence, pages 125–135, September 1999.
5. F. Giannotti and G. Manco. Making Knowledge Extraction and Reasoning Closer. In T. Terano, editor, *Procs. of the Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining*, number 1805 in Lecture Notes in Computer Science, April 2000.
6. F. Giannotti and G. Manco. *Declarative Knowledge Extraction with Iterative User-Defined Aggregates*. CNUCE-CNR Technical report, June 2000.
7. F. Giannotti, G. Manco, M. Nanni, and D. Pedreschi. Query Answering in Nondeterministic, Nonmonotonic, Logic Databases. In T. Andreasen, H. Christiansen, and H. Larsen, editors, *Procs. of the Workshop on Flexible Query Answering*, number 1395 in Lecture Notes on Artificial Intelligence, pages 175–187, march 1998.
8. F. Giannotti, G. Manco, M. Nanni, and D. Pedreschi. Nondeterministic, Nonmonotonic Logic Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2000. To appear. Available at `http://www-kdd.di.unipi.it`.
9. F. Giannotti, G. Manco, M. Nanni, D. Pedreschi, and F. Turini. Using Deduction for Intelligent Data Analysis. Technical Report B4-1999-02, CNUCE Institute of CNR, January 1999. A revised version is submitted to International Journal of Knowledge Discovery and Data Mining.
10. H. Mannila. Inductive databases and condensed representations for data mining. In *International Logic Programming Symposium*, pages 21–30, 1997.
11. D. Michie, D.J. Spiegelhalter, and C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, New York, 1994.
12. J. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
13. S. Ceri R. Meo, G. Psaila. A new sql-like operator for mining association rules. In *Proceedings of The Conference on Very Large Databases*, pages 122–133, 1996.
14. W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for Data Mining. In *Advances in Knowledge Discovery and Data Mining*, pages 375–398. AAAI Press/The MIT Press, 1996.
15. H. Wang and C. Zaniolo. Using SQL to Build New Aggregates and Extenders for Object-Relational Systems. In *Procs. of the International Conference on Very Large Data Bases (VLDB200)*, 2000. To appear.
16. C. Zaniolo, N. Arni, and K. Ong. Negation and Aggregates in Recursive Rules: The $\mathcal{LDL}$++ Approach. In *Proc. 3rd Int. Conf. on Deductive and Object-Oriented Databases (DOOD93)*, volume 760 of *Lecture Notes in Computer Science*, 1993.
17. C. Zaniolo and H. Wang. Logic-Based User-Defined Aggregates for the Next Generation of Database Systems. In K.R. Apt, V. Marek, M. Truszczynski, and D.S. Warren, editors, *The Logic Programming Paradigm: Current Trends and Future Directions*. Springer Verlag, 1998.