Towards a Logic Query Language for Data Mining

Fosca Giannotti[1], Giuseppe Manco[2], and Franco Turini[3]

[1] CNUCE-CNR - Via Alfieri 1 - I56010 Ghezzano (PI), Italy
Fosca.Giannotti@cnuce.cnr.it
[2] ICAR-CNR - Via Bucci 41c - I87036 Rende (CS), Italy
manco@icar.cnr.it
[3] Department of Computer Science, Univ. Pisa - C.so Italia 40 - I56125 Pisa, Italy
turini@di.unipi.it

Abstract. We present a logic database language with elementary data mining mechanisms to model the relevant aspects of knowledge discovery, and to provide a support for both the iterative and interactive features of the knowledge discovery process. We adopt the notion of user-defined aggregate to model typical data mining tasks as operations unveiling unseen knowledge. We illustrate the use of aggregates to model specific data mining tasks, such as frequent pattern discovery, classification, data discretization and clustering, and show how the resulting data mining query language allows the modeling of typical steps of the knowledge discovery process, that range from data preparation to knowledge extraction and evaluation.

1 Introduction and Motivations

Research in data mining and knowledge discovery in databases has mostly concentrated on algorithmic issues, assuming a naive model of interaction in which data is first extracted from a database and transformed in a suitable format, and next processed by a specialized inductive engine. Such an approach has the main drawback of proposing a fixed paradigm of interaction. Although it may at first sound appealing to have an autonomous data mining system, it is practically unfeasible to let the data mining algorithm "run loose" into the data in the hope to find some valuable knowledge. Blind search into a database can easily bring to the discovery of an overwhelming large set of patterns, many of which could be irrelevant, difficult to understand, or simply not valid: in one word, uninteresting.

On the other side, current applications of data mining techniques highlight the need for flexible knowledge discovery systems, capable of supporting the user in specifying and refining mining objectives, combining multiple strategies, and defining the quality of the extracted knowledge. A key issue is the definition of *Knowledge Discovery Support Environment* [16], i.e., a query system capable of obtaining, maintaining, representing and using high level knowledge in a unified framework. This comprises representation of domain knowledge, extraction of

R. Meo et al.(Eds.): Database Support for Data Mining Applications,LNAI 2682, pp. 76–94, 2004.

new knowledge and its organization in ontologies. In this respect, a knowledge discovery support environment should be an integrated mining and querying system capable of

- rigorous definition of user interaction during the search process,
- separation of concerns between the specification and the mapping to the underlying databases and data mining tools, and
- understandable representations for the knowledge.

Such an environment is expected to increase the programmer productivity of KDD applications. However, such capabilities require higher-order expressive features capable of providing a tight-coupling between knowledge mining and the exploitation of domain knowledge to share the mining itself.

A suitable approach can be the definition of a set of *data mining primitives*, i.e., a small number of constructs capable of supporting a vast majority of KDD applications. The main idea (outlined in [13]) is to combine relational query languages with data mining primitives in an overall framework capable of specifying data mining problems as complex queries involving KDD objects (rules, clustering, classifiers, or simply tuples). In this way, the mined KDD objects become available for further querying. The principle that query answers can be queried further is typically referred to as *closure*, and is an essential feature of SQL. KDD queries can thus generate new knowledge or retrieve previously generated knowledge. This allows for interactive data mining sessions, where users cross boundaries between mining and querying. Query optimization and execution techniques in such a query system will typically rely on advanced data mining algorithms.

Today, it is still an open question how to realize such features. Recently, the problem of defining a suitable knowledge discovery query formalism has interestend many researchers, both from the database perspective [1,2,13,19,14,11] and the logic programming perspective [21,22,20]. The approaches devised, however, do not explicitly model in a uniform way features such as closure, knowledge extration and representation, background knowledge and interestingness measures. Rather, they are often presented as "ad-hoc" proposals, particularly suitable only for subsets of the described peculiarities.

In such a context, the idea of integrating data mining algorithms in a deductive environment is very powerful, since it allows the direct exploitation of domain knowledge within the specification of the queries, the specification of ad-hoc interest measures that can help in evaluating the extracted knowledge, the modelization of the interactive and iterative features of knowledge discovery in a uniform way. In [5,7] we propose two specific models based on the notion of user-defined aggregate, namely for association rules and bayesian classification. In these approaches we adopt aggregates as an interface to mining tasks in a deductive database, obtaining a powerful amalgamation between inferred and induced knowledge. Moreover, in [4,8] we show that efficiency issues can be takled in an efficient way, by providing a suitable specification of aggregates.

In this paper we generalize the above approaches. We present a logic database language with elementary data mining mechanisms to model the relevant aspects

of knowledge discovery, and to provide a support for both the iterative and interactive features of the knowledge discovery process. We shall refer to the notion of *inductive database* introduced in [17,18,1], and provide a logic database system capable of representing inductive theories. The resulting data mining query language shall incorporate all the relevant features that allow the modeling of typical steps of the knowledge discovery process.

The paper is organized as follows. Section 2 provides an introduction to user-defined aggregates. Section 3 introduces the formal model that allows to incorporate both induced and deduced knowledge in a uniform framework. In section 4 we instantiate the model to some relevant mining tasks, namely frequent patterns discovery, classification, clustering and discretization, and show how the resulting framework allows the modeling of the interactive and iterative features of the knowledge discovery process. Finally, section 5 discusses how efficiency issues can be profitably undertaken, in the style of [8].

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 [23], and is viable for efficient query evaluation [6].

A remarkable capability of such a language is that of expressing distributive aggregates, such as `sum` or `count`. For example, the following clause illustrates the use of the `count` aggregate:

$$\mathtt{p}(\mathtt{X}, \mathtt{count}\langle \mathtt{Y} \rangle) \leftarrow \mathtt{r}(\mathtt{X}, \mathtt{Y}).$$

It is worth noting the semantic equivalence of the above clause to the following SQL statement:

```
SELECT X, COUNT(Y)
FROM r
GROUP BY X
```

Specifying User-defined aggregates. Clauses with aggregation are possible mainly because datalog++ supports nondeterminism [9] and XY-stratification [6,23]. This allows the definition of distributive aggregate functions, i.e., aggregate functions defined in an inductive way (S denotes a set and h is a composition operator):

$$\begin{array}{ll} \textit{Base:} & f(\{x\}) := g(x) \\ \textit{Induction:} & f(S \cup \{x\}) := h(f(S), x) \end{array}$$

Users can define aggregates in datalog++ by means of the predicates `single` and `multi`. For example, the `count` aggregate is defined by the unit clauses:

$$\mathtt{single}(\mathtt{count}, \mathtt{X}, 1).$$
$$\mathtt{multi}(\mathtt{count}, \mathtt{X}, \mathtt{C}, \mathtt{C} + 1).$$

The first clause specifies that the count of a set with a single element x is 1. The second clause specifies that the count of $S \cup \{x\}$ is $c+1$ for any set S such that the count of S is c.

Intuitively, a `single` predicate computes the *Base* step, while the `multi` predicate computes the *Inductive* step. The evaluation of clauses containing aggregates consists mainly in *(i)* evaluating the body of the clause, and nondeterministically sorting the tuples resulting from the evaluation, and *(ii)* evaluating the single predicate on the first tuple, and the multi predicate on the other tuples.

The `freturn` and `ereturn` predicates [25] allow building complex aggregate functions from simpler ones. For example, the average of a set S is obtained by dividing the sum of its elements by the number of elements. If S contains c elements whose sum is s, then $S \cup \{x\}$ contains $c+1$ elements whose sum is $s+x$. This leads to the following definition of the `avg` function:

$$\texttt{single}(\texttt{avg}, \texttt{X}, (\texttt{X}, 1)).$$
$$\texttt{multi}(\texttt{avg}, \texttt{X}, (\texttt{S}, \texttt{C}), (\texttt{S} + \texttt{X}, \texttt{C} + 1)).$$
$$\texttt{freturn}(\texttt{avg}, (\texttt{S}, \texttt{C}), \texttt{S}/\texttt{C}).$$

Iterative User-Defined Aggregates. Distributive aggregates are easy to define by means of the user-defined predicates `single` and `multi` in that they simply require a single scan of the available data. In many cases, however, even simple aggregates require multiple scans of the data. As an example, 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 average 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 average, and second, to compute the sum of the absolute difference.

However, the datalog$_{++}$ language is powerful enough to cope with multiple scans in the evaluation of user-defined aggregates. Indeed, in [4] we extend the semantics of datalog$_{++}$ by introducing the `iterate` predicate, which can be exploited to impose some user-defined conditions for iterating the scans over the data. More specifically, the evaluation of a clause

$$\texttt{p}(\texttt{X}, \texttt{aggr}\langle\texttt{Y}\rangle) \leftarrow \texttt{r}(\texttt{X}, \texttt{Y}). \tag{1}$$

(where `aggr` denotes the name of a user-defined aggregate) can be done according to the following schema:

- Evaluate $\texttt{r}(\texttt{X}, \texttt{Y})$, and group the resulting values associated with `Y` into subsets $S_1, \ldots, S_n$ according to the different values associated with `X`.
- For each subset $S_i = \{x_1, \ldots, x_n\}$, compute the aggregation value r as follows:
  1. evaluate $\texttt{single}(\texttt{aggr}, x_1, \texttt{C})$. Let c be the resulting value associated with `C`.
  2. Evaluate $\texttt{multi}(\texttt{aggr}, x_i, c, \texttt{C})$, for each i (by updating c to the resulting value associated with `C`).

3. Evaluate $\texttt{iterate}(\texttt{aggr}, c, \texttt{C})$. If the evaluation is successful, then update c to the resulting value associated with C, and return to step 2. Otherwise, evaluate $\texttt{freturn}(\texttt{aggr}, c, \texttt{R})$ and return the resulting value r associated with R.

The following example shows how iterative aggregates can be defined in the datalog++ framework. Further details on the approach can be found in [16,5].

Example 1. By exploiting the `iterate` predicate, the `abserr` aggregate for computing the absolute deviation S_n can be defined as follows:

$$\texttt{single}(\texttt{abserr}, \texttt{X}, (\texttt{avg}, \texttt{X}, 1)).$$

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

$$\texttt{multi}(\texttt{abserr}, (\texttt{M}, \texttt{D}), \texttt{X}, (\texttt{M}, \texttt{D} + (\texttt{M} - \texttt{X}))) \leftarrow \texttt{M} > \texttt{X}.$$
$$\texttt{multi}(\texttt{abserr}, (\texttt{M}, \texttt{D}), \texttt{X}, (\texttt{M}, \texttt{D} + (\texttt{X} - \texttt{M}))) \leftarrow \texttt{M} \leq \texttt{X}.$$

$$\texttt{iterate}(\texttt{abserr}, (\texttt{avg}, \texttt{S}, \texttt{C}), (\texttt{S}/\texttt{C}, 0)).$$

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

The first two clauses compute the average of the tuples under examination, in a way similar to the computation of the `avg` aggregate. The remaining clauses assume that the average value has already been computed, and are mainly used in the incremental computation of the sum of the absolute difference with the average. Notice how the combined use of `multi` and `iterate` allows the definition of two scans over the data. □

3 Logic-Based Inductive Databases

A suitable conceptual model that summarizes the relevant aspects discussed in section 1 is the notion of *inductive database* [1,17], that is is a first attempt to formalize the notion of interactive mining process. In the following definition, proposed by Mannila [18,17], the term inductive database refers to a relational database plus the set of all sentences from a specified class of sentences that are true of the data.

Definition 1. *Given an instance* $\mathbf{r}$ *of a relation* $\mathbf{R}$*, a class* $\mathcal{L}$ *of sentences (patterns), and a selection predicate* q*, a pattern discovery task is to find a theory*

$$\mathcal{T}h(\mathcal{L}, \mathbf{r}, q) = \{s \in \mathcal{L} | q(\mathbf{r}, s) \text{ is true}\}$$

□

The main idea here is to provide a unified and transparent view of both deductive knowledge, and all the derived patterns, (the induced knowledge) over

the data. The user does not care about whether he/she is dealing with inferred or induced knowledge, and whether the requested knowledge is materialized or not. The only detail he/she is interested in is the high-level specification of the query involving both deductive and inductive knowledge, according to some interestingness quality measure (which in turn can be either objective or subjective).

The notion of Inductive Database fits naturally in rule-based languages, such as Deductive Databases [7,5]. A deductive database can easily represent both extensional and intensional data, thus allowing a higher degree of expressiveness than traditional relational algebra. Such capability makes it viable for suitable representation of domain knowledge and support of the various steps of the KDD process.

The main problem in a deductive approach is how to choose a suitable representation formalism for the inductive part, enabling a tight integration with the deductive part. More specifically, the problem is how to formalize the specification of the set $\mathcal{L}$ of patterns in a way such that each pattern $s \in \mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$ is represented as an independent (logical) entity (i.e., a predicate) and each manipulation of $\mathbf{r}$ results in a corresponding change in s. To cope with such a problem, we introduce the notion of *inductive clauses*, i.e., clauses that formalize the dependency between the inductive and the deductive part of an inductive database.

Definition 2. *Given an inductive database theory $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$, an inductive clause for the theory is a clause (denoted by s)*

$$H \leftarrow B_1, \ldots B_n$$

such that

- *The evaluation of $B_1, \ldots B_n$ in the computed stable model*[1] *$M_{s \cup \mathbf{r}}$ correspond to the extension $\mathbf{r}$;*
- *there exist an injective function ϕ mapping each ground instance p of H in $\mathcal{L}$;*
- *$\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$ corresponds to the model $M_{s \cup \mathbf{r}}$ i.e.,*

$$p \in M_{s \cup \mathbf{r}} \Longleftrightarrow \phi(p) \in \mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$$

□

As a consequence of the above definition, we can formalize the notion of logic-based knowledge discovery support environment, as a deductive database programming language capable of expressing both inductive clauses and deductive clauses.

Definition 3. *A logic-based knowledge discovery support environment is a deductive database language capable of specifying:*

[1] See [24,6] for further details.

- *relational extensions;*
- *intensional predicates, by means of deductive clauses;*
- *inductive predicates, by means of inductive clauses.*

□

The main idea of the previous definition is that of providing a simple way for modeling the key aspects of a data mining query language:

- the source data is represented by the relational extensions;
- intensional predicates provide a way of dealing with background knowledge;
- inductive predicates provide a representation of both the extracted knowledge and the interestingness measures.

In order to formalize the notion of inductive clauses, the first fact that is worth observing is that it is particularly easy to deal with data mining tasks in a deductive framework, if we use aggregates as a basic tool.

Example 2. A frequent itemset $\{I_1, I_2\}$ is a database pattern with a validity specified by the estimation of the posterior probability $\Pr(I_1, I_2|\mathbf{r})$ (i.e., the probability that items I_1 and I_2 appear together according to $\mathbf{r}$). Such a probability can be estimated by means of *iceberg queries*: an iceberg query is a query containing an aggregate, in which a constraint (typically a threshold constraint) over the aggregate is specified. For example, the following query

```
SELECT R1.Item, R2.Item, COUNT(Tid)
FROM r R1, r R2
WHERE R1.Tid = R2.Tid
      AND R1.Item <> R2.Item
GROUP BY R1.Item, R2.Item
HAVING COUNT(Tid) > thresh
```

computes all the pairs of items appearing in a database of transactions with a given frequency. The above query has a straightforward counterpart in datalog++. The following clauses define typical (two-dimensional) association rules by using the `count` aggregate.

$$\begin{aligned} \texttt{pair}(\texttt{I1}, \texttt{I2}, \texttt{count}\langle\texttt{T}\rangle) &\leftarrow \texttt{basket}(\texttt{T}, \texttt{I1}), \texttt{basket}(\texttt{T}, \texttt{I2}), \texttt{I1} < \texttt{I2}. \\ \texttt{rules}(\texttt{I1}, \texttt{I2}) &\leftarrow \texttt{pair}(\texttt{I1}, \texttt{I2}, \texttt{C}), \texttt{C} \geq 2. \end{aligned}$$

The first clause generates and counts all the possible pairs, and the second one selects the pairs with sufficient support (i.e., at least 2). As a result, the predicate `rules` specifies associations, i.e. rules stating that certain combinations of values occur with other combinations of values with a certain frequency. Given the following definitions of the `basket` relation,

$$\begin{array}{lll} \texttt{basket}(1, \texttt{fish}). & \texttt{basket}(2, \texttt{bread}). & \texttt{basket}(3, \texttt{bread}). \\ \texttt{basket}(1, \texttt{bread}). & \texttt{basket}(2, \texttt{milk}). & \texttt{basket}(3, \texttt{orange}). \\ & \texttt{basket}(2, \texttt{onions}). & \texttt{basket}(3, \texttt{milk}). \\ & \texttt{basket}(2, \texttt{fish}). & \end{array}$$

by querying $\texttt{rules}(\texttt{X}, \texttt{Y})$ we obtain predicates that model the corresponding inductive instances: $\texttt{rules}(\texttt{bread}, \texttt{milk})$ and $\texttt{rules}(\texttt{bread}, \texttt{fish})$. ◁

Example 3. A classifier is a model that describes a discrete attribute, called the *class*, in terms of other attributes. A classifier is built from a set of objects (the training set) whose class values are known. Practically, starting from a table $\mathbf{r}$, we aim at computing the probability $\Pr(C = c|A = a, \mathbf{r})$ for each pair a, c of the attributes A and C [10]. This probability can be roughly estimated by computing some statistics over the data, such as, e.g.:

```
SELECT A, C, COUNT(*)
FROM r
GROUP BY A, C
```

Again, the datalog++ language easily allows the specification of such statistics. Let us consider for example the $\texttt{playTennis}(\texttt{Out}, \texttt{Temp}, \texttt{Hum}, \texttt{Wind}, \texttt{Play})$ table. We would like to predict the probability of playing tennis, given the values of the other attributes. The following clauses specify the computation of the necessary statistics for each attribute of the `playTennis` relation:

$$\begin{aligned}
\texttt{statistics}_{\texttt{Out}}(\texttt{O}, \texttt{P}, \texttt{count}\langle * \rangle) &\leftarrow \texttt{playTennis}(\texttt{O}, \texttt{T}, \texttt{H}, \texttt{W}, \texttt{P}).\\
\texttt{statistics}_{\texttt{Temp}}(\texttt{T}, \texttt{P}, \texttt{count}\langle * \rangle) &\leftarrow \texttt{playTennis}(\texttt{O}, \texttt{T}, \texttt{H}, \texttt{W}, \texttt{P}).\\
\texttt{statistics}_{\texttt{Hum}}(\texttt{H}, \texttt{P}, \texttt{count}\langle * \rangle) &\leftarrow \texttt{playTennis}(\texttt{O}, \texttt{T}, \texttt{H}, \texttt{W}, \texttt{P}).\\
\texttt{statistics}_{\texttt{Wind}}(\texttt{W}, \texttt{P}, \texttt{count}\langle * \rangle) &\leftarrow \texttt{playTennis}(\texttt{O}, \texttt{T}, \texttt{H}, \texttt{W}, \texttt{P}).\\
\texttt{statistics}_{\texttt{Play}}(\texttt{P}, \texttt{count}\langle * \rangle) &\leftarrow \texttt{playTennis}(\texttt{O}, \texttt{T}, \texttt{H}, \texttt{W}, \texttt{P}).
\end{aligned}$$

The results of the evaluation of such clauses can be easily combined in order to obtain the desired classifier. ◁

The above examples show how the simple clauses specifying aggregates can be devised as inductive clauses. Clauses containing aggregates, in languages such as datalog++, satisfy the most desirable property of inductive clauses, i.e., the capability to specify patterns of $\mathcal{L}$ that hold in $\mathcal{T}h$ in a "parameterized" way, i.e., according to the tuples of an extension $\mathbf{r}$. In this paper, we use aggregates as the means to introduce mining primitives into the query language.

As a matter of fact, an aggregate is a "natural" definition of an inductive database schema, in which patterns correspond to the true facts in the computed stable model, as the following statement shows.

Lemma 1. *An aggregate defines an inductive database.*

Proof. By construction. Let us consider the following clause (denoted by r_p):

$$p(X_1, \ldots, X_n, aggr\langle Y_1, \ldots, Y_m \rangle) \leftarrow r(X_1, \ldots, X_n, Y_1, \ldots, Y_m).$$

We then define

$$\mathcal{L} = \{\langle t_1, \ldots, t_n, s\rangle | p(t_1, \ldots, t_n, s) \text{ is ground}\}$$

and

$$q(\mathbf{r}, \langle t_1, \ldots, t_n, s\rangle) = \mathit{true} \text{ if and only if } p(t_1, \ldots, t_n, s) \in M_{r_p \cup \mathbf{r}}$$

that imposes that the only valid patterns are those belonging to the iterated stable model procedure. □

In such a context, an important issue to investigate is the correspondence between inductive schemas and aggregates, i.e., whether a generic inductive schema can be specified by means of an aggregate. Formally, for given an inductive database $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$, we are interested in providing a specification of an aggregate `aggr`, and in defining a clause

$$\mathtt{q}(Z_1, \ldots, Z_k, \mathtt{aggr}\langle X_1, \ldots, X_n\rangle) \leftarrow \mathbf{r}(Y_1, \ldots, Y_m).$$

which should turn out to be a viable inductive clause for $\mathcal{T}h$. The clause should define the format of any valid pattern $s \in \mathcal{L}$. In particular, the correspondence of s with the ground instances of the `q` predicate should be defined by the specification of the aggregate `aggr` (as described in section 2). Moreover, any predicate $\mathtt{q}(\mathtt{t_1}, \ldots, \mathtt{t_k}, \mathtt{s})$ resulting from the evaluation of such a clause and corresponding to s, should model the fact that $s \in \mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$. When such a definition is possible, the "inductive" predicate `q` itself can be used in the definition of more complex queries.

Relating the specification of aggregates with inductive clauses is particularly attractive for two main reasons. First of all, it provides an amalgamation between mining and querying, and hence makes it easy to provide a unique interface capable of specifying source data, knowledge extraction, background knowledge and interestingness specification. Moreover, it allows a good flexibility in the exploitation of mining algorithms for specific tasks. Indeed, we can implement aggregates as simple language interfaces for the algorithms (implemented as separate modules, like in [7]); conversely, we can exploit the notion of iterative aggregate, and explicitly specify the algorithms in datalog++ (like in [8]). The latter approach, in particular, gives two further advantages:

- from a conceptual point of view, it allows the use of background knowledge directly in the exploration of the search space.
- from an efficiency point of view, it provides the opportunity of integrating specific optimizations inside the algorithm.

4 Mining Aggregates

As stated in the previous section, our main aim is the definition of inductive clauses, formally modeled as clauses containing specific user-defined aggregates, and representing specific data mining tasks. The discussion on how to provide efficient implementations of algorithms for such tasks by means of iterative aggregates is given in [8]. The rest of the paper is devoted at showing how the proposed model is suitable to some important knowledge discovery tasks. We shall formalize some inductive clauses by means of aggregates), to formulate data mining tasks in the datalog++ framework. In the resulting framework, the integration of inductive clauses with deductive clauses allows a simple and intuitive formalization of the various steps of the data mining process, in which deductive rules can specify both the preprocessing and the result evaluation phase, while inductive rules can specify the mining phase.

4.1 Frequent Patterns Discovery

Associations are rules that state that certain combinations of values occur with other combinations of values with a certain frequency and certainty. A general definition is the following.

Let $\mathcal{I} = \{a_1, \ldots, a_n\}$ be a set of literals, called *items*. An *itemset* T is a set of items such that $T \subseteq \mathcal{I}$. Given a relation $\mathbf{R} = A_1 \ldots A_n$, a *transaction* in an instance $\mathbf{r}$ of $\mathbf{R}$, associated to attribute A_i (where $dom(A_i) = \mathcal{I}$) according to a *transaction identifier* A_j, is a set of items of the tuples of $\mathbf{r}$ having the same value of A_j.

An association rule is a statement of the form $X \Rightarrow Y$, where $X \subseteq \mathcal{I}$ and $Y \subseteq \mathcal{I}$ are two sets of items. To an association rule we can associate some statistical parameters. The *support* of a rule is the percentage of transactions that contain the set $X \cup Y$, and the *confidence* is the percentage of transactions that contain Y, provided that they contain X.

The problem of association rules mining can be finally stated as follows: given an instance $\mathbf{r}$ of $\mathbf{R}$, find all the association rules from the set of transactions associated to A_i (grouped according to A_j), such that for each rule $A \Rightarrow B[S, C]$, $S \geq \sigma$ and $C \geq \gamma$, where σ is the *support threshold* and γ is the *confidence threshold*.

The following definition provides a formulation of the association rules mining task in terms of inductive databases.

Definition 4. *Let* $\mathbf{r}$ *be an instance of the table* $\mathbf{R} = A_1 \ldots A_n$*, and* $\sigma, \gamma \in [0, 1]$*. For given* $i, j \leq n$*, let*

- $\mathcal{L} = \{A \Rightarrow B | A, B \subseteq dom(\mathbf{R}[A_i])\}$*, and*
- $q(\mathbf{r}, A \Rightarrow B) =$ *true if and only if* $freq(A \cup B, \mathbf{r}) \geq \sigma$ *and* $freq(A \cup B, \mathbf{r})/freq(A, \mathbf{r}) \geq \gamma$*.*

Where $freq(s, \mathbf{r})$ *is the (relative) frequency of* s *in the set of the transactions in* $\mathbf{r}$ *grouped by* A_j*. The theory* $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$ *defines the* frequent patterns discovery *task.* □

The above definition provides an inductive schema for the frequent pattern discovery task. We now specify a corresponding inductive clause.

Definition 5. *Given a relation* `r`*, the* `patterns` *aggregate is defined by the rule*

$$\mathtt{p}(\mathtt{X_1}, \ldots, \mathtt{X_n}, \mathtt{patterns}\langle(\mathtt{m\_s}, \mathtt{m\_c}, \mathtt{Y})\rangle) \leftarrow \mathtt{r}(\mathtt{Z_1}, \ldots, \mathtt{Z_m}) \tag{2}$$

where the variables $\mathtt{X_1}, \ldots, \mathtt{X_n}, \mathtt{Y}$ are a rearranged subset of the variables $\mathtt{Z_1}, \ldots, \mathtt{Z_k}$ of `r`, and the `Y` variable denotes a set of elements. The aggregate `patterns` computes the set of predicates $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n}, \mathtt{l}, \mathtt{r}, \mathtt{f}, \mathtt{c})$ where:

1. $\mathtt{t_1}, \ldots, \mathtt{t_n}$ are distinct instances of the variables $\mathtt{X_1}, \ldots, \mathtt{X_n}$, as resulting from the evaluation of `r`;
2. $\mathtt{l} = \{\mathtt{l_1}, \ldots, \mathtt{l_k}\}$ and $\mathtt{r} = \{\mathtt{r_1}, \ldots, \mathtt{r_h}\}$ are subsets of the value of `Y` in a tuple resulting from the evaluation of `r`;

3. f and c are respectively the support and the confidence of the rule $\mathtt{l} \Rightarrow \mathtt{r}$, such that $f \geq m\_s$ and $c \geq m\_c$.

□

Example 4. Let us consider a sample `transaction` table. The following rule specifies the computation of association rules (with 30% support threshold and 100% confidence threshold) starting from the extension of the table:

$$\begin{array}{ll} \mathtt{transaction}(\mathtt{D},\mathtt{C},\langle \mathtt{I} \rangle) & \leftarrow \mathtt{transaction}(\mathtt{D},\mathtt{C},\mathtt{I},\mathtt{Q},\mathtt{P}). \\ \mathtt{rules}(\mathtt{patterns}\langle(0.3,1.0,\mathtt{S})\rangle) & \leftarrow \mathtt{transaction}(\mathtt{D},\mathtt{C},\mathtt{S}). \end{array} \tag{3}$$

The first clause collects all the transactions associated to attribute `I` and grouped by attributes `D` and `C`. The second clause extracts the relevant patterns from the collection of available transactions. The result of the evaluation of the predicate $\mathtt{rules}(\mathtt{L},\mathtt{R},\mathtt{S},\mathtt{C})$ against such a program yields, e.g., the answer predicate $\mathtt{rules}(\mathtt{diapers},\mathtt{beer},0.5,1.0)$. ◁

It is easy to see how inductive clauses exploiting the `patterns` aggregate allow the specification of frequent pattern discovery tasks. The evidence of such a correspondence can be obtained by suitably specifying the `patterns` aggregate [8,16].

4.2 (Bayesian) Classification

It is particularly simple to specify Naive Bayes classification by means of an inductive database schema. Let us consider a relation $\mathbf{R}$ with attributes $A_1, \ldots, A_n$ and C. For simplicity, we shall assume that all the attributes represent discrete values. This is not a major problem, since

- it is well-known that classification algorithms perform better with discrete-valued attributes;
- supervised discretization [15] combined with discrete classification allows a more effective approach. As we shall see, the framework based on inductive clauses easily allows the specification of discretization tasks as well.

The bayesian classification task can be summarized as follows. Given an instance $\mathbf{r}$ of $\mathbf{R}$, we aim at computing the function

$$\max_c \quad \Pr(C = c | A_1 = a_1, \ldots, A_n = a_n, \mathbf{r}) \tag{4}$$

where $c \in dom(C)$ and $a_i \in dom(A_i)$. By repeated application of Bayes' rule and the assumption that $A_1, \ldots, A_n$ are independent, we obtain

$$\Pr(C = c | A_1 = a_1, \ldots, A_n = a_n, \mathbf{r}) = \Pr(C = c | \mathbf{r}) \times \prod_i \Pr(A_i = a_i | C = c, \mathbf{r})$$

Now, each factor in the above product can be estimated from $\mathbf{r}$ by means of the following equation

$$\tilde{\Pr}(A_j = a_j | C = c, \mathbf{r}) = \frac{freq(c, \sigma_{A_j = a_j}(\mathbf{r}))}{freq(c, \mathbf{r})}$$

Hence, the definition of a classification task can be accomplished by computing some suitable statistics. That is, a suitable inductive theory associates to each possible pair the corresponding statistic.

Definition 6. *Let* $\mathbf{R} = A_1 \dots A_n C$ *be a relation schema. Given an instance* $\mathbf{r}$ *of* $\mathbf{R}$*, we define*

- $\mathcal{L} = \{\langle A_i = a_i \wedge C = c, n_A, n_C\rangle | a_i \in dom(A_i), c \in dom(C)$ *and* n_A, $n_C \in \mathbb{R}\}$.
- $q(\mathbf{r}, \langle A_i = a_i \wedge C = c, n_A, n_C\rangle) = true$ *if and only if* $n_A = \Pr(A_i = a_i | C = c, \mathbf{r})$ *and* $n_C = \Pr(C = c|\mathbf{r})$.

The resulting theory $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$ *formalizes a naive bayesian classification task.* □

Notice that the datalog++ language easily allows the computation of all the needed statistics, by enumerating all the pairs for which we need to count the occurrences (see, e.g., example 3). However, we can associate the inductive theory with a more powerful user-defined aggregate, in which all the needed statistics can be efficiently computed without resorting to multiple clauses.

Definition 7. *Given a relation* `r`*, the* `nbayes` *aggregate is defined by a rule schema*

$$\mathtt{s}(\mathtt{X}_1, \dots, \mathtt{X}_m, \mathtt{nbayes}\langle(\{(\mathtt{1}, \mathtt{A}_1), \dots, (\mathtt{n}, \mathtt{A}_n)\}, \mathtt{C})\rangle) \leftarrow \mathtt{r}(\mathtt{Z}_1, \dots, \mathtt{Z}_k).$$

where

- *The variables* $\mathtt{X}_1, \dots, \mathtt{X}_m, \mathtt{A}_1, \dots, \mathtt{A}_n, \mathtt{C}$ *are a (possibly rearranged) subset of the values of* $\mathtt{Z}_1, \dots, \mathtt{Z}_k$ *resulting from the evaluation of* `r`*;*
- *The result of such an evaluation is a predicate* $\mathtt{s}(\mathtt{t}_1, \dots, \mathtt{t}_n, \mathtt{c}, (\mathtt{i}, \mathtt{a}_i), \mathtt{v}_i, \mathtt{v}_c)$*, representing the set of counts of all the possible values* $\mathtt{a}_i$ *of the i-th attribute* $\mathtt{A}_i$*, given any possible value* `c` *of* `C`*. In particular,* $\mathtt{v}_i$ *represents the frequency of the pair* $\mathtt{a}_i, \mathtt{c}$*, and* $\mathtt{v}_c$ *represents the frequency of* `c` *in the extension* `r`*.*

□

Example 5. Let us consider the toy `playTennis` table defined in example 3. The frequencies of the attributes can be obtained by means of the clause

$$\mathtt{classifier}(\mathtt{nbayes}\langle(\{(\mathtt{1}, \mathtt{O}), (\mathtt{2}, \mathtt{T}), (\mathtt{3}, \mathtt{H}), (\mathtt{4}, \mathtt{W})\}, \mathtt{P})\rangle) \leftarrow \mathtt{playTennis}(\mathtt{O}, \mathtt{T}, \mathtt{H}, \mathtt{W}, \mathtt{P}).$$

The evaluation of the query $\mathtt{classifier}(\mathtt{C}, \mathtt{F}, \mathtt{C}_F, \mathtt{C}_C)$ returns, e.g., the answer predicate $\mathtt{classifier}(\mathtt{yes}, (\mathtt{1}, \mathtt{sunny}), 0.6, 0.4)$. ◁

Again, by suitably specifying the aggregate, we obtain a correspondence between the inductive database schema and its deductive counterpart [16,5].

4.3 Clustering

Clustering is perhaps the most straightforward example of data mining task providing a suitable representation of its results in terms of relational tables. In a relation $\mathbf{R}$ with instance $\mathbf{r}$, the main objective of clustering is that of labeling

each tuple $\mu \in \mathbf{r}$. In relational terms, this correspond in adding a set of attributes $A_1, \ldots, A_n$ to $\mathbf{R}$, so that a tuple $\langle a_1, \ldots a_n \rangle$ associated to a tuple $\mu \in \mathbf{r}$ represents a cluster assigment for μ. For example, we can enhance $\mathbf{R}$ with two attributes C and M, where C denotes the cluster identifier and M denotes a probability measure. A tuple $\mu \in \mathbf{r}$ is represented in the enhancement of $\mathbf{R}$ by a new tuple μ', where $\mu'[A] = \mu[A]$ for each $A \in \mathbf{R}$, and $\mu'[C]$ represents the cluster to which μ belongs, with probability $\mu'[M]$. In the following defintion, we provide a sample inductive schema formalizing the clustering task.

Definition 8. *Given a relation $\mathbf{R} = A_1 \ldots A_n$ with extension $\mathbf{r}$, such that tuples in $\mathbf{r}$ can be organized in k clusters, an inductive database modeling clustering is defined by $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$, where*

- $\mathcal{L} = \{\langle \mu, i \rangle | \mu \in dom(A_1) \times \ldots \times dom(A_n), i \in \mathbb{N}\}$, *and*
- $q(\mathbf{r}, \langle \mu, i \rangle)$ *is true if and only if $\mu \in \mathbf{r}$ is assigned to the i-th cluster.*

□

It is particularly intuitive to specify the clustering data mining task as an aggregate.

Definition 9. *Given a relation* $\mathtt{r}$, *the* `cluster` *aggregate is defined by the rule schema*

$$\mathtt{p}(\mathtt{X_1}, \ldots, \mathtt{X_n}, \mathtt{clusters}\langle(\mathtt{Y_1}, \ldots, \mathtt{Y_k})\rangle) \leftarrow \mathtt{r}(\mathtt{Z_1}, \ldots, \mathtt{Z_m}).$$

where the variables $\mathtt{X_1}, \ldots, \mathtt{X_n}, \mathtt{Y_1}, \ldots, \mathtt{Y_k}$ *are a rearranged subset of the variables* $\mathtt{Z_1}, \ldots, \mathtt{Z_m}$ *of* `r.clusters` *computes the set of predicates* $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n}, \mathtt{s_1}, \ldots, \mathtt{s_k}, \mathtt{c})$, *where:*

1. $\mathtt{t_1}, \ldots, \mathtt{t_n}, \mathtt{s_1}, \ldots, \mathtt{s_k}$ *are distinct instances of the variables* $\mathtt{X_1}, \ldots, \mathtt{X_n}, \mathtt{Y_1}, \ldots \mathtt{Y_k}$, *as resulting from the evaluation of* $\mathtt{r}$;
2. `c` *is a label representing the cluster to which the tuple* $\mathtt{s_1}, \ldots, \mathtt{s_k}$ *is assigned, according to some clustering algorithm.*

□

Example 6. Consider a relation `customer(name, address, age, income)`, storing information about customers, as shown in fig. 1 a). We can define a "clustered" view of such a database by means of the following rule:

$$\mathtt{custCView}(\mathtt{clusters}\langle(\mathtt{N}, \mathtt{AD}, \mathtt{AG}, \mathtt{I})\rangle) \leftarrow \mathtt{customer}(\mathtt{N}, \mathtt{AD}, \mathtt{AG}, \mathtt{I}).$$

The evaluation of such rule, shown in fig. 1 b), produces two clusters. ◁

It is particularly significant to see how the proposed approach allows to directly model closure (i.e., to manipulate the mining results).

Example 7. We can easily exploit the `patterns` aggregate to find an explanation of such clusters:

$$\mathtt{frqPat}(\mathtt{C}, \mathtt{patterns}\langle(0.6, 1.0, \{\mathtt{f\_a}(\mathtt{AD}), \mathtt{s\_a}(\mathtt{I})\})\rangle) \leftarrow \mathtt{custCView}(\mathtt{N}, \mathtt{AD}, \mathtt{AG}, \mathtt{I}, \mathtt{C}).$$

name	address	age	income
cust1	pisa	50	50K
cust2	rome	30	30K
cust3	pisa	45	48K
cust4	florence	24	30K
cust5	pisa	60	50K
cust6	rome	26	30K

(a)

name	address	age	income	cluster
cust1	pisa	50	50K	1
cust2	rome	30	30K	2
cust3	pisa	45	48K	1
cust4	florence	24	30K	2
cust5	pisa	60	50K	1
cust6	rome	26	30K	2

(b)

cluster	left	right	support	conf
1	{s_a(50K)	f_a(pisa)}	2	1.0
2	{f_(rome)	s_a(30K)}	2	1.0

(c)

Fig. 1. a) Sample `customer` table. Each row represents some relevant features of a customer. b) Cluster assignments: a cluster label is associated to each row in the originary `customer` table. c) Cluster explanations with frequent patterns. The first cluster contains customers with high income and living in `pisa`, while the second cluster contains customers with low income and living in `rome`

Notice how the cluster label `C` is used for separating transactions belonging to different clusters, and mining associations in such clusters separately. Table 1 c) shows the patterns resulting from the evaluation of such a rule. As we can see, cluster 1 is mainly composed by high-income people living in Pisa, while cluster 2 is mainly composed by low-income people living in Rome. ◁

4.4 Data Discretization

Data discretization can be used to reduce the number of values for a given continuous attribute, by dividing the range of the attribute into intervals. Interval labels can be used to replace actual data values.

Given an instance $\mathbf{r}$ of a relation $\mathbf{R}$ with a numeric attribute A, the main idea is to provide a mapping among the values of $dom(A)$ and some given labels. More precisely, we define $\mathcal{L}$ as the pairs $\langle a, i\rangle$, where $a \in dom(A)$ and $i \in \mathcal{V}$ represents an interval label (i.e., $\mathcal{V}$ is a representation of all the intervals $[l, s]$ such that $l, s \in dom(A)$). A discretization task of the tuples of $\mathbf{r}$, formalized as a theory $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$, can be defined according to some discretization objective, i.e., a way of relating a value $a \in dom(A)$ to an interval label i.

In such a context, discretization techniques can be distinguished into [12,3] *supervised* or *unsupervised*. The objective of supervised methods is to discretize continuous values in homogeneous intervals, i.e., intervals that preserve a predefined property (which, in practice, is represented by a label associated to each continuous value). The formalization of supervised discretization as an inductive theory can be tuned as follows.

Definition 10. *Let* $\mathbf{r}$ *be an instance of a relation* $\mathbf{R}$*. Given an attribute* $A \in \mathbf{R}$*, and a discrete-valued attribute* $C \in \mathbf{R}$*, an inductive database theory* $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$ *defines a supervised discretization task if*

- *either* $dom(A) \in \mathbb{N}$ *or* $dom(A) \in \mathbb{R}$*;*
- $\mathcal{L} = \{\langle a, C, i\rangle | a \in dom(A), i \in \mathbb{N}\}$*;*
- $q(\mathbf{r}, \langle a, C, i\rangle) =$ *true if and only if there is a discretization of* $\pi_A(\mathbf{r})$ *in homogeneous intervals with respect to the attribute* C*, such that* a *belongs to the* i*-th interval.*

□

The following definition provides a corresponding inductive clause.

Definition 11. *Given a relation* $\mathtt{r}$*, the aggregate* `discr` *defines the rule schema*

$$\mathtt{s}(\mathtt{Y_1}, \ldots, \mathtt{Y_k}, \mathtt{discr}\langle(\mathtt{Y}, \mathtt{C})\rangle) \leftarrow \mathtt{r}(\mathtt{X_1}, \mathtt{X_2}, \ldots, \mathtt{X_n}).$$

where

- $\mathtt{Y}$ *is a continuous-valued variable, and* $\mathtt{C}$ *is a discrete-valued variable;*
- $\mathtt{Y_1}, \ldots, \mathtt{Y_k}, \mathtt{Y}, \mathtt{C}$ *are a rearranged subset of the variables* $\mathtt{X_1}, \mathtt{X_2}, \ldots, \mathtt{X_n}$ *in* $\mathtt{r}$*.*

The result of the evaluation of such rule is given by predicates $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_k}, \mathtt{v}, \mathtt{i})$*, where* $\mathtt{t_1}, \ldots, \mathtt{t_k}, \mathtt{v}$ *are distinct instances of the variables* $\mathtt{Y_1}, \ldots, \mathtt{Y_k}, \mathtt{Y}$*, and* $\mathtt{i}$ *is an integer value representing the* i*-th interval in a supervised discretization of the values of* $\mathtt{Y}$ *labelled with the values of* $\mathtt{C}$*.* □

In the ChiMerge approach to supervised discretization [15], a bottom-up interval generation procedure is adopted: initially, each single value is considered an interval. At each iteraction, two adjacent intervals are chosen and joined, provided that they are sufficiently homogeneous. The degree of homogeneity is measured w.r.t. a class label, and is computed by means a χ^2 statistics. Homogeneity is made parametric to a user-defined significance level α, identifying the probability that two adjacent intervals have independent label distributions. In terms of inductive clauses, this can be formalized as follows:

$$\mathtt{s}(\mathtt{Y_1}, \ldots, \mathtt{Y_k}, \mathtt{discr}\langle(\mathtt{Y}, \mathtt{C}, \alpha)\rangle) \leftarrow \mathtt{r}(\mathtt{X_1}, \mathtt{X_2}, \ldots, \mathtt{X_n}).$$

Here, `Y` represents the attribute to discretize, `C` represents the class label and α is the significance level.

Example 8. The following rule

$$\mathtt{intervals}(\mathtt{discr}\langle(\mathtt{Price}, \mathtt{Beer}, 0.9)\rangle) \leftarrow \mathtt{serves}(\_, \mathtt{Beer}, \mathtt{Price}).$$

defines a 0.9 significance level supervised discretization of the `price` attribute, according to the values of `beer`, of the relation `serves` shown in fig. 2. More precisely, we aim at obtaining a discretization of `price` into intervals preserving the values of the `beer` attribute (as they appear associate to `price` in the tuples of the relation `serves`. For example, the values 100 and 117 can be merged into the interval $[100, 117]$, since the tuples of `serves` in which such values occur contain the same value of the `beer` attribute (the `Bud` value). The results of the evaluation of the inductive clause are shown in fig. 2 c). ◁

bar	beer	price
A	Bud	100
A	Becks	120
C	Bud	117
D	Bud	130
D	Bud	150
E	Becks	140
E	Becks	122
F	Bud	121
G	Bud	133
H	Becks	125
H	Bud	160
I	Bud	135

(a)

value	interval
100	1
120	1
117	1
130	3
150	4
140	4
122	2
121	2
133	3
125	2
160	4
135	3

(b)

Fig. 2. a) The `serves` relation. Each row in the relation represents a brand of beer served by a given bar, with the associated price. b) ChiMerge Discretization: each value of the `price` attribute in `serves` is associated with an interval. Intervals contain values which occur in tuples of `serves` presenting similar values of the `beer` attribute

It is particularly interesting to see how the discretization and classification tasks can be combined, by exploiting the respective logical formalizations.

Example 9. A typical dataset used as a benchmark for classification tasks is the *Iris* classification dataset [15]. The dataset can be represented by means of a relation containing 5 attributes:

$$\texttt{iris}(\texttt{Sepal\_length}, \texttt{Sepal\_width}, \texttt{Petal\_length}, \texttt{Petal\_width}, \texttt{Specie})$$

Each tuple in the dataset describes the relevant features of an iris flower. The first four attributes are continuous-valued attributes, and the `Specie` attribute is a nominal attribute corresponding to the class to which the flower belongs (either *iris-setosa*, *iris-versicolor*, or *iris-virginica*). We would like to characterize species in the `iris` relation according to their features. To this purpose, we may need a preprocessing phase in which continuous attributes are discretized:

$$\begin{aligned}
\texttt{intervals}_{\texttt{SL}}(\texttt{discr}\langle(\texttt{SL}, \texttt{C}, 0.9)\rangle) &\leftarrow \texttt{iris}(\texttt{SL}, \texttt{SW}, \texttt{PL}, \texttt{PW}, \texttt{C}).\\
\texttt{intervals}_{\texttt{SW}}(\texttt{discr}\langle(\texttt{SW}, \texttt{C}, 0.9)\rangle) &\leftarrow \texttt{iris}(\texttt{SL}, \texttt{SW}, \texttt{PL}, \texttt{PW}, \texttt{C}).\\
\texttt{intervals}_{\texttt{PL}}(\texttt{discr}\langle(\texttt{PL}, \texttt{C}, 0.9)\rangle) &\leftarrow \texttt{iris}(\texttt{SL}, \texttt{SW}, \texttt{PL}, \texttt{PW}, \texttt{C}).\\
\texttt{intervals}_{\texttt{PW}}(\texttt{discr}\langle(\texttt{PW}, \texttt{C}, 0.9)\rangle) &\leftarrow \texttt{iris}(\texttt{SL}, \texttt{SW}, \texttt{PL}, \texttt{PW}, \texttt{C}).
\end{aligned}$$

The predicates defined by the above clauses provide a mapping of the continuous values to the intervals shown in fig. 3. A classification task can be defined by exploiting such predicates:

$$\begin{aligned}
\texttt{irisCl}(\texttt{nbayes}\langle(\{\texttt{SL}, \texttt{SW}, \texttt{PL}, \texttt{PW}\}, \texttt{C})\rangle) \leftarrow\ & \texttt{iris}(\texttt{SL1}, \texttt{SW1}, \texttt{PL1}, \texttt{PW1}, \texttt{C}),\\
& \texttt{intervals}_{\texttt{SW}}(\texttt{SW1}, \texttt{SW}),\\
& \texttt{intervals}_{\texttt{SL}}(\texttt{SL1}, \texttt{SL}),\\
& \texttt{intervals}_{\texttt{PW}}(\texttt{PW1}, \texttt{PW}),\\
& \texttt{intervals}_{\texttt{PL}}(\texttt{PL1}, \texttt{PL}).
\end{aligned}$$

interval	setosa	virginica	versicolor
[4.3,4.9)	16	0	0
[4.9,5)	4	1	1
[5,5.5)	25	5	0
[5.5,5.8)	4	15	2
[5.8,6.3)	1	15	10
[6.3,7.1)	0	14	25
[7.1,7.9]	0	0	12

sepal_length

interval	setosa	virginica	versicolor
[2,2.5)	1	9	1
[2.5 ,2.9)	0	18	18
[2.9,3)	1	7	2
[3,3.4)	18	15	24
[3.4,4.4]	30	1	5

sepal_width

interval	setosa	virginica	versicolor
[1,3)	50	0	0
[3,4.8)	0	44	1
[4.8,5.2)	0	6	15
[5.2,6.9]	0	0	34

petal_length

interval	setosa	virginica	versicolor
[0.1,1)	50	0	0
[1,1.4)	0	28	0
[1.4,1.8)	0	21	5
[1.8,2.5]	0	1	45

petal_width

Fig. 3. Bayesian statistics using ChiMerge and the `nbayes` aggregate. For each attribute of the `iris` relation, a set of intervals is obtained, and the distribution of the classes within such intervals is computed. For example, interval $[4.3, 4.9)$ of the `sepal_length` attribute contains 16 tuples labelled as `setosa`, while interval $[7.1, 7.9)$ of the same attribute contains 12 tuples labelled as `versicolor`

The statistics resulting from the evaluation of the predicate defined by the above rule are shown in fig. 3. ◁

5 Conclusions

The main purpose of flexible knowledge discovery systems is to obtain, maintain, represent, and utilize high-level knowledge. This includes representation and organization of domain and extracted knowledge, its creation through specialized algorithms, and its utilization for context recognition, disambiguation, and needs identification. Current knowledge discovery systems provide a fixed paradigm that does not sufficiently supports such features in a coherent formalism. On the contrary, logic-based databases languages provide a flexible model of interaction that actually supports most of the above features in a powerful, simple and versatile formalism. This motivated the study of a logic-based framework for intelligent data analysis.

The main contribution of this paper was the development of a logic database language with elementary data mining mechanisms to model extraction, representation and utilization of both induced and deduced knowledge. In particular, we have shown that aggregates provide a standard interface for the specification of data mining tasks in the deductive environment: i.e., they allow to model mining tasks as operations unveiling pre-existing knowledge. We used such main features to model a set of data mining primitives: frequent pattern discovery, Bayesian classification, clustering and discretization.

The main drawback of a deductive approach to data mining query languages concerns efficiency: a data mining algorithm can be worth substantial optimiza-

tions that come both from a smart constraining of the search space, and from the exploitation of efficient data structures. In this case, the adoption of the datalog++ logic database language has the advantage of allowing a direct specification of mining algorithms, thus allowing specific optimizations. Practically, we can directly specify data mining algorithms by means of iterative user-defined aggregates, and implement the most computationally intensive operations by means of *hot-spot* refinements [8]. Such a feature allows to modularize data mining algorithms and integrate domain knowledge in the right points, thus allowing crucial domain-oriented optimizations.

References

1. J-F. Boulicaut, M. Klemettinen, and H. Mannila. Querying Inductive Databases: A Case Study on the MINE RULE Operator. In *Procs. 2nd European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD98)*, LNCS 1510, pages 194–202, 1998.
2. M.S. Chen, J. Han, and P.S. Yu. Data Mining: An Overview from a Database Perspective. *IEEE Trans. on Knowledge and Data Engineering*, 8(6):866–883, 1996.
3. J. Dougherty, R. Kohavi, and M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *Procs. 12th International Conference on Machine Learning*, pages 194–202, 1995.
4. F. Giannotti and G. Manco. Declarative knowledge extraction with iterative user-defined aggregates. *AI*IA Notizie*, 13(4), December 2000.
5. F. Giannotti and G. Manco. Making Knowledge Extraction and Reasoning Closer. In *Procs. 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2000)*, LNAI 1805, pages 360–371, 2000.
6. F. Giannotti, G. Manco, M. Nanni, and D. Pedreschi. Nondeterministic, Nonmonotonic Logic Databases. *IEEE Trans. on Knowledge and Data Engineering*, 13(5):813–823, 2001.
7. F. Giannotti, G. Manco, D. Pedreschi, and F. Turini. Experiences with a Logic-Based Knowledge Discovery Support Environment. In *Selected Papers of the Sixth congress of the Italian Congress of Artificial Intellingence*, LNCS 1792, 2000.
8. F. Giannotti, G. Manco, and F. Turini. Specifying Mining Algorithms with Iterative User-Defined Aggregates: A Case Study. In *Procs. 5th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2001)*, LNAI 2168, pages 128–139, 2001.
9. F. Giannotti, D. Pedreschi, and C. Zaniolo. Semantics and Expressive Power of Non Deterministic Constructs for Deductive Databases. *Journal of Computer and Systems Sciences*, 62(1):15–42, 2001.
10. G. Graefe, U. Fayyad, and S. Chaudhuri. On the Efficient Gathering of Sufficient Statistics for Classification from Large SQL Databases. In *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD98)*, pages 204–208, 1998.
11. J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A Data Mining Query Language for Relational Databases. In *SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'96)*, 1996.
12. F. Hussain, H. Liu, C. Tan, and M. Dash. Discretization: An Enabling Technique. *Journal of Knowledge Discovery and Data Mining*, 6(4):393–423, 2002.

13. T. Imielinski and H. Mannila. A Database Perspective on Knowledge Discovery. *Communications of the ACM*, 39(11):58–64, 1996.
14. T. Imielinski and A. Virmani. MSQL: A Query Language for Database Mining. *Journal of Knowledge Discovery and Data Mining*, 3(4):373–408, 1999.
15. R. Kerber. ChiMerge: Discretization of Numeric Attributes. In *Proc. 10th National Conference on Artificial Intelligence (AAAI92)*, pages 123–127. The MIT Press, 1992.
16. G. Manco. *Foundations of a Logic-Based Framework for Intelligent Data Analysis*. PhD thesis, Department of Computer Science, University of Pisa, April 2001.
17. H. Mannila. Inductive databases and condensed representations for data mining. In *International Logic Programming Symposium*, pages 21–30, 1997.
18. H. Mannila and H. Toivonen. Levelwise Search and Border of Theories in Knowledge Discovery. *Journal of Knowledge Discovery and Data Mining*, 3:241–258, 1997.
19. R. Meo, G. Psaila, and S. Ceri. A Tightly-Coupled Architecture for Data Mining. In *International Conference on Data Engineering (ICDE98)*, pages 316–323, 1998.
20. L. De Raedt. Data mining as constraint logic programming. In *Procs. Int. Conf. on Inductive Logic Programming*, 2000.
21. 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.
22. D. Tsur et al. Query Flocks: A Generalization of Association-Rule Mining. In *Proc. ACM Conf. on Management of Data (Sigmod98)*, pages 1–12, 1998.
23. 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)*, LNCS 760, 1993.
24. C. Zaniolo, S. Ceri, C. Faloutsos, R.T Snodgrass, V.S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufman, 1997.
25. C. Zaniolo and H. Wang. Logic-Based User-Defined Aggregates for the Next Generation of Database Systems. In *The Logic Programming Paradigm: Current Trends and Future Directions*, Springer Verlag, 1998.