Logical Languages for Data Mining

Fosca Giannotti[1], Giuseppe Manco[2], and Jef Wijsen[3]

[1] ISTI, Institute of CNR
Via Moruzzi 1, 56124 Pisa, Italy
[2] ICAR, Institute of CNR
Via Bucci 41c, 87036 Rende (CS), Italy
[3] University of Mons-Hainaut, Institut d'Informatique
Avenue du Champ de Mars 6, B-7000 Mons, Belgium

Abstract. Data mining focuses on the development of methods and algorithms for such tasks as classification, clustering, rule induction, and discovery of associations. In the database field, the view of data mining as advanced querying has recently stimulated much research into the development of data mining query languages. In the field of machine learning, inductive logic programming has broadened its scope toward extending standard data mining tasks from the usual attribute-value setting to a multirelational setting. After a concise description of data mining, the contribution of logic to both fields is discussed. At the end, we indicate the potential use of logic for unifying different existing data mining formalisms.

1 Introduction

Data mining uses induction to infer understandable and useful knowledge (rules, patterns, regularities) from large data sets. As such knowledge inference is beyond the expressive power of classical query languages, stronger query paradigms are needed. Logic can contribute in several respects to fulfilling this need. From a database perspective, logic can be used as a unifying formalism for integrating input data sources with data mining tasks and discovered knowledge. From a machine learning perspective, the main challenge lies in upgrading existing "propositional" data mining techniques to first-order logic. We briefly look at each perspective in turn.

Database Perspective. The success of SQL relies mainly on a small number of primitives sufficient to support a large majority of database applications. Unfortunately, these primitives are not sufficiently expressive to support the emerging field of knowledge discovery in databases (KDD). Imielinski and Mannila [42] compare the current situation in KDD to the situation in database management systems in the early 1960s, when each application had to be built from scratch, without the benefit of dedicated database primitives. They note that today's data mining techniques would more appropriately be described as "file mining" because of the loose coupling between data mining engines and databases. To remedy this situation, they propose 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.

These appealing ideas of closure and crossing boundaries in a KDD query system are often summarized by the term *inductive database*. Today, it is still an open question how to realize inductive databases. The route taken in this chapter is to use logic as a unifying framework to "glue" input data with KDD objects and queries.

Machine Learning Perspective. Most classical data mining techniques can work only on a single table and can discover only rules where all atoms are of the form "attribute=constant." Such techniques are commonly called *propositional* because the problems solved can be expressed in propositional logic. It turns out, however, that many real-life data mining problems contain multiple tables and ask for first-order rules containing variables. One approach is to transform these problems first into a propositional format and then tackle them by propositional techniques. Nevertheless, "propositionalization" is often undesirable, because it may lose information or result in an awkward problem specification.

Rather than downgrading the problem, it is more attractive to upgrade existing techniques so that they can directly tackle first-order problems. As a consequence, there is currently a growing interest in upgrading propositional data mining tools, like classifiers, to first-order logic. The induction of first-order rules from examples is at the center of a subdomain of machine learning called inductive logic programming (ILP).

The chapter is organized as follows. Section 2 gives a general overview of the tasks and challenges in data mining. Section 3 takes a database perspective on the use of logic as a unifying framework for integrating data sources with data mining tasks and discovered knowledge. Section 4 explains the original foundations of ILP, and Sect. 5 indicates how ILP has recently broadened its scope from induction of logic programs to standard data mining tasks. Section 6 outlines a possible approach toward the integration of several concepts presented in this chapter. Finally, Sect. 7 contains concluding remarks. Throughout the chapter, we will use the eaters database of Fig. 1 as a running example. The three tables of this database capture their intuitive semantics.

Serves

Restaurant	*Food*	*City*
Pronto	Italian	Pisa
Trevi	Italian	Pisa
Naxos	Greek	Pisa
Taco	Mexican	Mons

Dishes

Food	*Aspect*
Italian	Light
Greek	Light
Mexican	Spicy
Mexican	Fat
Junk	Fat

Visits

Eater	*Restaurant*	*Date*
Ron	Trevi	13-02-2002
Ron	Pronto	14-02-2002
Ron	Pronto	21-02-2002
Ron	Naxos	15-02-2002
Sam	Naxos	14-02-2002
Tim	Pronto	14-02-2002
Tim	Trevi	15-02-2002
Tim	Taco	16-02-2002

$$\forall r, f, c, f', c' \ (\ (\mathit{Serves}(r, f, c) \wedge \mathit{Serves}(r, f', c')) \rightarrow (f = f' \wedge c = c')\ )$$
$$\forall r, f, c \ (\ \mathit{Serves}(r, f, c) \rightarrow \exists x(\mathit{Dishes}(f, x))\ )$$
$$\forall e, r, d \ (\ \mathit{Visits}(e, r, d) \rightarrow \exists f, c(\mathit{Serves}(r, f, c))\ )$$

Fig. 1. Eaters database with integrity constraints

2 The Search for Knowledge in Databases

Knowledge discovery in databases (KDD) is interdisciplinary: it draws many of its basic principles from mature concepts in databases, machine learning, statistics, economics, and sociology. By means of KDD, interesting knowledge (rules, regularities, patterns) can be extracted from large data sets and investigated from different angles. The knowledge discovered can be applied to information management, query processing, decision making, process control, and many other areas.

Fayyad et al. [26] define KDD concisely as *"the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data."* By pattern, we mean any statement or property that characterizes a set of objects within a database. KDD is a *multistep process*, composed of four main phases: data consolidation, data preparation, data mining, and result interpretation. The KDD process is *interactive* and *iterative* in the sense that each step requires a strong user interaction and may incite rectifications of the preceding steps. Textbooks in the domains of KDD, data mining, and machine learning include [7,23,26,38,41,55,82,83].

2.1 A General Data Mining Framework

In general, we can identify a number of main "dimensions" in a knowledge discovery task [55]: the source data to explore, the patterns to discover,

the criterion for determining the usefulness of patterns, and the background knowledge. We briefly discuss each dimension.

Primary Knowledge Sources. Most data mining techniques have focused on extracting knowledge from simple files. Currently, there is a growing interest in mining from more complex data sources, such as multirelational databases, XML documents, and unstructured text.

A famous motto in knowledge discovery cites *"Garbage in ⇒ Garbage out,"* meaning that noisy, inconsistent, insignificant data can hardly produce relevant results. This simple statement highlights the importance of the *preprocessing phase,* which combines

- *data integration,* where multiple heterogeneous data sources are integrated into one;
- *data selection,* where data relevant to the analysis task are retrieved from the database;
- *data cleaning,* which deals with noisy, missing, and irrelevant data;
- *data transformation*, where summary and aggregation operations are applied to transform and consolidate data into forms appropriate for mining.

Each of these aspects has been extensively studied in the literature [5,25,66] and are beyond the scope of this chapter. Hereafter, we will use the symbol Σ to denote primary knowledge.

Pattern Search Space. Patterns are sentences about primary knowledge sources, expressing rules, regularities, or models that hold in the underlying data. The language for expressing patterns is fixed in advance. We will use the symbol $\mathcal{L}$ to denote the set of all legal patterns.

Search Criteria. A search criterion is defined relative to Σ and determines whether a given pattern in $\mathcal{L}$ is "potentially useful." In [51,52], this construct is captured by a binary predicate q, whose first argument is a knowledge source Σ and whose second argument is some pattern $l \in \mathcal{L}$. If $q(\Sigma, l)$ is true, then l is potentially useful. The KDD task then is to find the set $\{l \in \mathcal{L} \mid q(\Sigma, l)$ is true$\}$. The above definition provides no insight into the actual data mining tasks one may be interested in. Sections 2.2–2.4 provide an overview of common data mining tasks.

Background Knowledge. This term denotes the domain knowledge that is already known in advance and that can be incorporated in each of the preceding dimensions. It can be used to enrich the primary knowledge or to derive good initial hypotheses to start the pattern search. Background knowledge can be exploited in the search strategy for pruning the pattern space, or it can serve as a reference for interpreting discovered knowledge. Furthermore, it is

difficult to define adequate statistical measures for subjective concepts such as novelty, usefulness, and understandability. Background knowledge can be helpful in capturing such concepts more accurately.

Transcending Dimensions. Each of the above dimensions can be described in a different language. However, using different languages results in an "impedance mismatch" between dimensions, which can be a serious impediment to efficiency and efficacy. A major challenge in building data mining systems is smooth cooperation between different dimensions. Obviously, incorporating background into primary knowledge is easier if both are expressed in the same language. Exploiting background knowledge in the search strategy is facilitated if the search criterion can be expressed in the same language as the background knowledge. The "closure principle" asks for a common data model to represent both primary knowledge and discovered patterns. Finally, a unified language may permit integration of mining and querying in the style of [42].

A coherent formalism capable of dealing uniformly with all dimensions would represent a breakthrough in the design and development of decision support systems in diverse application domains. The advantages of such an integrated formalism include the ability to formalize the overall KDD process and the possibility of tailoring a methodology to a specific application domain. The amalgamation of different data mining dimensions is at the center of Sect. 3.

2.2 Data Mining Tasks

The term data mining is used in this context to describe a set of algorithms and methods capable of extracting patterns from a structured data source. We search in the data source for regularities, i.e. combinations of values for certain attributes shared by facts in the database. A regularity can be regarded as a high-level summary of information in the data under consideration.

Different classification schemes can be used to categorize data mining methods, based on the kinds of databases to be studied, the kinds of knowledge to be discovered, and the kinds of techniques to be used [17]. It is common to distinguish between two general categories of data mining techniques:

- *Predictive modeling* (or *directed knowledge discovery*), where samples of past experiences with known answers are examined and generalized to future cases. Examples of predictive modeling techniques are classification and regression.
- *Descriptive modeling* (or *undirected knowledge discovery*), where the task is to identify patterns in the data that may be significant. Examples of descriptive modeling techniques are clustering and the discovery of association rules.

Prediction problems are described in terms of specific goals, which are related to past records with known answers. These are used to project new cases. Undirected knowledge discovery problems usually involve a stage prior to prediction, where information is insufficient for predictive analysis and has to be integrated with further knowledge.

2.3 Predictive Modeling

The main goal of predictive modeling is to predict the values of some target attribute based on the values of other attributes. If the attribute being predicted is numeric, then the prediction problem is a *regression* problem. If the target attribute is categorical, then it is a *classification* problem. The general problem is to estimate the most probable value for the target attribute, given the other attribute values, the training data in which the target value is given for each observation, and a set of assumptions representing one's prior knowledge of the problem, i.e. one's background knowledge. Various estimation techniques can be used, including

1. Probability density estimation using, for example, Bayesian models.
2. Metric-space based methods, in which we define a distance measure on data points and base the prediction on proximity to data points in the training set. An example is the k-nearest neighbor method.
3. Projection into decision regions: divide the attribute space into decision regions, and associate a prediction with each. Examples are decision trees that make a piecewise constant approximation of the decision surface, neural networks that find nonlinear decision surfaces, and classification rules.
   The antecedent of a classification rule is generally a conjunction of tests, whereas the consequence gives the class. Often the tests are restricted to comparisons of an attribute with a constant, so the rules have the following format:

$$(A_1 = a_1) \wedge (A_2 = a_2) \wedge \ldots \wedge (A_n = a_n) \Rightarrow C = c \ ,$$

   where C is the target attribute to be predicted. Rules of this form are called *propositional*. Because the number of possible attribute-value comparisons is finite and known in advance, the rules can be treated in propositional logic. Upgrading classification rules to first-order logic is at the center of ILP and multirelational data mining, discussed in Sects. 4–5.

2.4 Descriptive Modeling

Descriptive modeling differs from predictive modeling in that there is no target attribute. The data mining tool explores the data in search of interesting rules valid in the underlying database. In this context, the process of

interpretation and refinement of the results plays a crucial role, as understandability and usefulness of the results have to be defined a posteriori. The most common techniques studied in the literature are *data summarization* and *clustering*.

Data Summarization

The goal is to extract compact patterns that describe subsets of the data. We focus on *association rules*, which are defined as propositional classification rules introduced in Sect. 2.3, except that there is no single target attribute to predict, i.e. the consequent can be any conjunction of tests. An example is the rule "*City* = Pisa ⇒ *Food* = Italian" about the *Serves* relation, expressing that restaurants in Pisa tend to serve Italian food.

In the database community, much attention has been spent on the discovery of Boolean association rules [2,4,40,71]. The starting point for this data mining task is a set of *items*. A *transaction* (or *itemset*) is a finite set of items, and a *transaction base* is a finite multiset of transactions. It is customary to think of a transaction as the items purchased together in a single sales transaction of a supermarket. A *Boolean association rule* is a statement $X \Rightarrow Y$, where X and Y are transactions. The "interestingness" of a Boolean association rule is defined relative to a given transaction base **TB**. The *support* of a transaction X is defined as the fraction of transactions in **TB** that contain X; the *support* of the rule $X \Rightarrow Y$ is defined as the support of $X \cup Y$. The *confidence* of $X \Rightarrow Y$ equals the support of $X \cup Y$ divided by the support of X. Then, given a transaction base **TB**, a support threshold s^*, and a confidence threshold c^*, the data mining task of interest is to find all association rules whose support and confidence exceed s^* and c^*, respectively. An important subtask of this problem is finding all *frequent itemsets*, where a frequent itemset is defined as an itemset with support greater than s^*.

Clearly, transaction data can be easily stored in a relational database. For the *Visits* relation of the eaters database, we can introduce for each eater a transaction defined as the set of restaurants visited by that eater. The confidence of the Boolean association rule $\{\text{Pronto}, \text{Naxos}\} \Rightarrow \{\text{Taco}\}$ then represents the conditional probability that a person visits the Taco restaurant, provided that he visits the restaurants Pronto and Naxos.

The basic framework for Boolean association rules has been extended in several ways to improve the expressive power and quality of the mined rules. When data are time-stamped, then temporal constraints can be put on the association rules to be discovered. For example, it is natural to require that the restaurants in the antecedent of a rule be visited prior to those in the consequent.

The use of concept hierarchies in association rule mining and other KDD tasks has been extensively studied, for example, in [14,75]. A concept hierarchy defines a series of mappings from a set of low-level concepts to more

general, higher level concepts. For example, beer and wine can both be categorized as "beverage," a subcategory of "food." Concept hierarchies provide a simple and powerful way of dealing with domain knowledge in data mining.

Most algorithms for association rule mining rely on the simple notions of support and confidence. Nevertheless, more sophisticated quality measures, including statistically significant correlation, have appeared in the literature [6,39,62,74].

Clustering

Clustering aims at separating the tuples of a given relation into subsets (called clusters) such that the similarity of tuples of the same subset is maximized and the similarity of tuples of different subsets is minimized [45]. Clustering is also called *unsupervised learning* because the right number of clusters is not known in advance. This is unlike the supervised learning task of classification, where the input consists of training examples with known class labels. Clustering methods can be categorized as follows [38]:

- *Partition-based methods* attempt to decompose the data directly into clusters that are scored and adjusted according to some criterion.
- *Hierarchical methods* group objects in a cluster tree, called a *dendrogram*. Hierarchical methods are categorized as *agglomerative* or *divisive* depending on whether the dendrogram is constructed bottom-up or top-down.
- *Density-based methods* assume a reachability relation among data points. A point is dense if the number of points reachable from it exceeds a specified threshold number. A point is added to a cluster if it is reachable from a dense point that already belongs to that cluster.
- *Model-based methods* hypothesize a model for each of the clusters and then attempt to optimize the fit between the data and that model.
- *Grid-based methods* partition the data into a finite number of cells that form a grid structure and then perform clustering on the grid structure.

Particularly relevant in the context of logic is *conceptual clustering* [77], in which an intentional description (a concept) is associated with each extensionally formed cluster. For example, the *Serves* relation of the eaters database can be divided into two clusters corresponding to the concepts "light" and "fat," that is, the first cluster satisfies $\forall r, f, c(Serves(r, f, c) \rightarrow Dishes(f, \text{Light}))$, and the second $\forall r, f, c(Serves(r, f, c) \rightarrow Dishes(f, \text{Fat}))$. Depending on the concepts that can be captured, a conceptual clustering technique can be categorized as propositional or first-order.

3 Inductive Databases

In this section, we first explain the concept of an inductive database and then show how it has been accomplished in extensions of SQL and in Datalog++. We discuss the notion of inductive query and its implementation by means of aggregates.

3.1 Deduction Versus Induction

As argued at the end of Sect. 2.1, there is a need to combine knowledge discovery (induction) with background reasoning and querying (deduction). The construct of an *inductive database* [42] has been conceived to fulfill this need. Intuitively, an inductive database refers to a relational database extended with the set of all patterns of a specified class that hold in the database. The induced patterns are available to the users as views, which can be materialized or virtual. User queries can then combine base tables and induced patterns, thereby crossing the boundaries between deduction and induction. The user does not care whether he is dealing with deduced or induced knowledge, or whether the requested knowledge is materialized or virtual.

The following definitions, inspired by [12,51], capture the ideas of inductive database and inductive query. An *inductive schema* is a triple $(R, \mathcal{L}, f)$, where

- R is a relational schema [and let $inst(R)$ denote the set of all relations over R],
- $\mathcal{L}$ is a set of patterns defined relative to R, and
- f is a total function with domain $inst(R) \times \mathcal{L}$ that assigns an *interestingness measure* to each element of its domain.

Next, an *inductive relation* over the inductive schema $(R, \mathcal{L}, f)$ is a triple $(I, L, ind(I, L))$, where

- I is a relation over R [i.e. $I \in inst(R)$],
- $L \subseteq \mathcal{L}$, and
- $ind(I, L) := \{\langle l, f(I, l)\rangle \mid l \in L\}$.

Note that the component $ind(I, L)$ can be computed from I and L using the function f specified in the inductive schema. As with relational views, the component $ind(I, L)$ may be materialized or virtual. In either case, the user can simply assume that the component is present.

An *inductive query* is a mapping from inductive relations to inductive relations over the same inductive schema. Such a mapping can be specified by a classical relational query Q mapping I to $Q(I)$, extended as the identity on L. Note that the schema of $Q(I)$ must be equal to the schema of I, or else $ind(Q(I), L)$ would be undefined. Alternatively, we can specify a mapping from L to a new set of patterns, extended as the identity on I. For example, if we assume that the output of f is the set of real numbers between zero and one, then L may be mapped to $\{l \in L \mid f(I, l) \geq 0.5\}$. In principle, it is even conceivable to have queries that change both I and L. In all cases, the part $ind(I, L)$ is updated accordingly. Query evaluation results in traditional query processing for the component I and in the execution of mining algorithms for the component $ind(I, L)$.

Figure 2 shows $ind(Visits, L)$, where L is the set of all Boolean association rules among restaurants (see Sect. 2.4), and $f(Visits, l)$ outputs the support

and the confidence for each $l \in L$. The query Q asks for restaurant visits prior to 15-02-2002; the support and confidence figures change accordingly. Finally, L is restricted to L' by requiring that the consequent of each rule be an Italian restaurant and the confidence exceed 0.75. Means of expressing such restrictions will be discussed later on in this chapter.

$ind(Visits, L)$

Association	Support	Confidence
Pronto ⇒ Trevi	0.66	1.00
Trevi ⇒ Pronto	0.66	1.00
Pronto ⇒ Naxos	0.33	0.50
Naxos ⇒ Pronto	0.33	0.50
...		

$Q =$
```
SELECT *
FROM Visits
WHERE Date < 15-02-2002
```

$Q(Visits)$

Eater	*Restaurant*	*Date*
Ron	Trevi	13-02-2002
Ron	Pronto	14-02-2002
Sam	Naxos	14-02-2002
Tim	Pronto	14-02-2002

$ind(Q(Visits), L)$

Association	Support	Confidence
Pronto ⇒ Trevi	0.33	0.50
Trevi ⇒ Pronto	0.33	1.00
Pronto ⇒ Naxos	0.00	0.00
Naxos ⇒ Pronto	0.00	0.00
...		

$ind(Q(Visits), L')$

Association	Support	Confidence
Trevi ⇒ Pronto	0.33	1.00

Fig. 2. Example of querying inductive relations

One can think of an inductive database as a data mining task, where the search space ($\mathcal{L}$) and the interestingness measure (f) are defined by the inductive schema. Knowledge discovery is modeled as an interactive process in which users can query both base tables and induced patterns to gain insight about the data. Query languages capable of dealing with base data and induced patterns in a uniform way may be called *inductive query languages.* Past efforts for developing such languages can be classified into two categories:

- Development of mining tools tightly integrated with SQL DBMSs, representing both the source data and the induced rules in database relations. Mining requests are specified in an SQL-like language.

- Development of a formalism capable of dealing with all requirements of knowledge discovery applications, including pattern specification, significance evaluation, and ad hoc exploitation of background knowledge. The focus has generally been on Datalog-based languages.

We now discuss each approach in turn.

3.2 SQL Extensions

The query language proposed in [54] extends SQL with the new operator `MINE RULE`, which allows computing and encoding association rules in a relational format. The following query defines a table `Associations` to store associations among restaurants. In particular, it looks for Boolean association rules where transactions are sets of restaurants visited by the same eater; transactions with less than three items are ignored. Moreover, the body and the head of the rule are constrained to restaurants in Pisa and Mons, respectively, and the restaurants in the body should have been visited prior to the single restaurant in the head.

```
MINE RULE Associations AS
SELECT   DISTINCT 1..n Restaurant AS BODY,
                  1..1 Restaurant AS HEAD,
                  SUPPORT, CONFIDENCE
WHERE    BODY.City = "Pisa" AND HEAD.City = "Mons"
AND      BODY.Date < HEAD.Date
FROM     Visits NATURAL JOIN Serves
GROUP BY Eater
HAVING   COUNT(*) > 2
EXTRACTING RULES WITH SUPPORT: 0.3, CONFIDENCE: 0.7 .
```

Let us analyze the components of the above query according to the dimensions discussed in Sect. 2.1. First, the primary knowledge source Σ from which to mine can be thought of as the result of the following SQL query:

```
CREATE VIEW Source AS
SELECT Eater, Restaurant, Date, Food, City
FROM   Visits NATURAL JOIN Serves
WHERE  Eater IN ( SELECT   Eater
                  FROM     Visits NATURAL JOIN Serves
                  GROUP BY Eater
                  HAVING   COUNT(*) > 2 ) .
```

Second, starting from the *Source* view, the `MINE RULE` operator specifies the mining of Boolean association rules. The rules are stored in a relational table with attributes `BODY`, `HEAD`, `SUPPORT`, and `CONFIDENCE`. Transactions are constructed by grouping on *Eater*, as specified by the `GROUP BY` clause. The `SELECT` clause restricts the pattern search space to rules with multiple (`1..n`)

restaurants in the body and a single (`1..1`) restaurant in the head. Finally, the search criterion, called *mining condition* in [54], includes the usual support and confidence constraints, as specified by the `EXTRACTING` clause. In addition, the `WHERE` clause requires that restaurants in the body and head be in Pisa and Mons, respectively, and that the restaurants in the body be visited prior to that in the head.

Note that the `MINE RULE` primitive achieves the closure principle by storing association rules in (possibly non-1NF) relational tables that can serve as the input for further data mining steps. MSQL [43,44] introduces SQL extensions to query such rules in tabular format. An example is

```
SELECT *
FROM   Serves
WHERE  VIOLATES ALL (  GETRULES(Serves)
                       WHERE BODY IS {(City=*)}
                       AND   CONSEQUENT IS {(Food=*)}
                       AND   CONFIDENCE > 0.5 ) .
```

In this query, the `GETRULES` subquery generates, from the *Serves* relation, all association rules with confidence > 0.5 of the form $City = a \Rightarrow Food = b$, where a is a *City*-value in *Serves* and b a *Food*-value. Such a rule is violated by a tuple t of *Serves* if $t(City) = a$ but $t(Food) \neq b$.

Other SQL extensions for data mining purposes have been proposed, aiming at richer data preparation prior to data mining and at more complex data mining tasks. In the `MINE RULE` approach, manipulation of source relations is limited to SQL queries. The online analytical mining (OLAM) approach [36,39] and the corresponding DMQL language [37] permit specifying OLAP operations prior to mining. Efficient processing of such operations is studied in [18,73]. The language primitives proposed in [37,57,69] address tasks beyond mining associations: discovery of sequential patterns, classification and prediction, and clustering.

3.3 Deductive Approaches

The `MINE RULE` operator combines, in a single query, the manipulation of source data (deduction) followed by the mining of association rules (induction). The approach satisfies the closure property to the extent that the discovered rules are stored in a relational table. Two important limitations of such linguistic extensions to SQL are, first, their restriction to a predefined family of data mining tasks (association rules for `MINE RULE`), and second, the difficulty of introducing and taking advantage of background knowledge. To overcome these limitations, several authors (for example, [13,28]) propose using Datalog extensions to build inductive database systems. As opposed to the `MINE RULE` approach, deductive databases permit encoding ad hoc

data mining tasks as well as specifying background knowledge, and, importantly, allow using one and the same language for both. As already argued in Sect. 2.1, this permits exploiting background knowledge in the search strategy used to solve a given data mining task. Some efforts were made to equip SQL with such capabilities [3,16,35,70,81]; such efforts, however, aimed at efficiently computing specific data mining tasks within SQL databases and in general failed to provide a flexible and expressive query paradigm. In this section, our focus will be on Datalog++, which we believe is more promising to this extent.

Datalog++, and its current implementation $\mathcal{LDL}$++, is a highly expressive language, including recursion and stratified negation [85], and permits efficient query evaluation [30]. This language allows the user to define distributive aggregates [86], such as *Count* or *Sum*. The following Datalog++ clause, for example, counts the number of restaurants visited by each eater:

$$RestaurantCount(e, Count\langle r\rangle) \leftarrow Visits(e, r, d) \ .$$

The clause is equivalent to the SQL query,

```
SELECT   Eater, COUNT(DISTINCT Restaurant)
FROM     Visits
GROUP BY Eater .
```

Clauses with aggregation are possible mainly because Datalog++ supports nondeterminism [32] and XY stratification [30,85]. This allows the definition of distributive aggregate functions, i.e. aggregate functions defined inductively (S denotes a set and h is a composition operator):

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

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

$$\begin{aligned} Single(Count, x, 1) &\leftarrow \\ Multi(Count, x, c, c+1) &\leftarrow \ . \end{aligned}$$

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.

The *Freturn* predicate allows 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:

$$\begin{aligned} Single(Avg, x, (x, 1)) &\leftarrow \\ Multi(Avg, x, (s, c), (s+x, c+1)) &\leftarrow \\ Freturn(Avg, (s, c), s/c) &\leftarrow \ . \end{aligned}$$

Further predicates also provide the capability of defining nondistributive aggregates. A detailed treatment of such aggregates in Datalog++ is given in [50,86]

We now illustrate that inductive databases conceived in Datalog++ permit exploiting background knowledge. Suppose we are interested in finding pairs of restaurants such that at least three specific eaters have visited both restaurants, or, using the association rule jargon, finding frequent ("at least three") itemsets of size two. The following Datalog++ program performs this task (we use the predicate *ans* for the query answer):

$$\begin{aligned} \mathit{Pairs}(r_1, r_2, \mathit{Count}\langle e\rangle) &\leftarrow \mathit{Visits}(e, r_1, d_1), \mathit{Visits}(e, r_2, d_2), r_1 \neq r_2 \\ \mathit{ans}(r_1, r_2) &\leftarrow \mathit{Pairs}(r_1, r_2, c), c \geq 3 \ . \end{aligned}$$

Now it is easy to see that if a restaurant r has been visited by less than three eaters, then this restaurant can never appear in the answer set. This property, known as the a priori principle, can be considered background knowledge and can be exploited to optimize the preceding program. Before looking at pairs of restaurants, we first apply a filter that retains only restaurants with at least three visitors. Restaurants that do not pass the filter, can be safely ignored. In this way, fewer restaurant pairs have to be examined. The resulting program is

$$\begin{aligned} \mathit{Singleton}(r, \mathit{Count}(e)) &\leftarrow \mathit{Visits}(e, r, d) \\ \mathit{Filter}(r) &\leftarrow \mathit{Singleton}(r, c), c \geq 3 \\ \mathit{Pairs}(r_1, r_2, \mathit{Count}\langle e\rangle) &\leftarrow \mathit{Visits}(e, r_1, d_1), \mathit{Visits}(e, r_2, d_2), \\ &\quad \mathit{Filter}(r_1), \mathit{Filter}(r_2), r_1 \neq r_2 \\ \mathit{ans}(r_1, r_2) &\leftarrow \mathit{Pairs}(r_1, r_2, c), c \geq 3 \ . \end{aligned}$$

Such queries with a filter condition have been called *query flocks* [78]. The clever exploitation of filter conditions in association rule mining has been investigated in [33,39,58,76]. Characterizing queries that allow an a priori style of filtering is at the center of [15].

3.4 Logic-Based Inductive Queries

As shown above, Datalog++ can contribute to inductive databases in several respects: first, it allows defining ad hoc complex aggregates, useful for specifying data mining tasks; second, it provides the possibility of "guiding" the search in the pattern space, thus providing the capability of exploiting background knowledge. We now discuss the third contribution: Datalog++ enables tight integration between mining and querying, between induction and deduction. The following deductive query, for example, asks for pairs of Greek and Italian restaurants in the outcome of the preceding Datalog++ data mining program:

$$\begin{aligned} \mathit{ItaloGreek}(r_1, r_2) \leftarrow \mathit{ans}(r_1, r_2), \mathit{Serves}(r_1, \mathrm{Italian}, c_1), \\ \mathit{Serves}(r_2, \mathrm{Greek}, c_2) \ . \end{aligned}$$

Intuitively, we can distinguish between inductive and deductive queries. By an *inductive query*, we mean a clause $Head \leftarrow Body$, where *Head* represents a pattern in $\mathcal{L}$ and *Body* represents the knowledge sources on which patterns depend. Evaluating such a clause corresponds to exploring the search space according to given search criteria. In the above example, the clauses defining the *ans* predicate can be considered an inductive query.

In [28,29], inductive queries are formally modeled in a structured way by means of clauses containing specific user-defined aggregates. We illustrate this by an example. The *Patterns* aggregate defined in [31] can be used in inductive queries to extract frequent itemsets. Let us call a restaurant "light" if it serves only light dishes, and let us call an eater "pro-Greek" if he has visited at least one restaurant serving Greek food. We are interested in the following task: find sets s of light restaurants such that at least three pro-Greek eaters have visited each restaurant in s. In association rule terminology, the problem involves finding frequent ("at least three") itemsets subject to constraints (restaurants must be light, eaters must be pro-Greek). The following Datalog++ program performs this task:

$$\begin{aligned} \mathit{ResByEater}(e, \langle r \rangle) &\leftarrow \mathit{Visits}(e, r, d) \\ \mathit{Itemset}(\mathit{Patterns}\langle(3, s)\rangle) &\leftarrow \mathit{ResByEater}(e, s), r \in s, \\ &\quad\ \mathit{Serves}(r, \text{Greek}, c) \\ \mathit{ans}(s, n) &\leftarrow \mathit{Itemset}(s, n), \neg \mathit{NotLight}(s) \\ \mathit{NotLight}(s) &\leftarrow r \in s, \mathit{Serves}(r, f, c), \mathit{Dishes}(f, x), \\ &\quad\ x \neq \text{Light} \ . \end{aligned}$$

The first clause groups tuples in the *Visits* relation on *Eater*. The second clause is an inductive query: the $\mathit{Patterns}\langle(3, s)\rangle$ aggregate extracts frequent itemsets from the grouped restaurants; the details of its operation can be found in [31]. After evaluating the second clause, $\mathit{Itemset}(s, n)$ holds if s is a set of restaurants visited by $n \geq 3$ pro-Greek eaters. Finally, the third clause restricts the result to sets of light restaurants. This example shows three important issues:

- Primary knowledge is specified by means of Datalog++ clauses. In the above example, the extension of the *ResByEater* predicate constitutes the primary knowledge source. The use of Datalog++ for specifying primary knowledge allows complex data manipulations, including recursion and stratified negation.
- The *Patterns* aggregate is then evaluated relative to the primary knowledge source. More precisely, the $\mathit{Patterns}\langle(3, s)\rangle$ aggregate acts as an interface to a specific data mining algorithm by explicitly defining the support threshold (the value 3) and the input transactions (the set of all possible s resulting from the evaluation of the query). The evaluation of such a data mining aggregate may benefit from the use of background

knowledge, as described in Sect. 3.3. Moreover, user-defined aggregates allow defining search criteria in an ad hoc way.
- The results of the mining phase can be queried and combined with further background knowledge, thus allowing refinement of the results for specific application needs. The adoption of a rule-based language in this phase allows integration between induced and deduced knowledge and supports interoperability between different data mining packages.

The approach described can be generalized to a variety of mining tasks, provided that the patterns to be discovered can be conveniently encoded in Datalog++. For certain tasks, like decision tree learning, such encoding may be intricate, thus impeding the interoperation between mining and querying. The Datalog++ approach was found suitable for frequent pattern discovery, Bayesian classification, supervised discretization, and clustering. We refer the interested reader to [50] for an in-depth treatment of inductive queries, as well as for examples of more complicated data mining tasks accomplished in Datalog++.

This concludes the database perspective; in the next two sections, we discuss the role of logic in data mining from a machine learning perspective. Finally, Sect. 6 tries to combine both points of view.

4 Induction of Logic Programs

Initially, inductive logic programming (ILP) focused on using a collection of relations to infer a clausal theory. This initial focus of ILP is explained in the current section. After an introductory example, we formalize the data mining task traditionally addressed in ILP. We then look at the fundamentals of ILP systems for solving this task.

In recent years, ILP has broadened its scope to cover standard data mining tasks such as classification, regression, clustering, and association rules. These extensions are discussed in Sect. 5.

4.1 Introductory Example

We use a database perspective to illustrate the induction of logic programs. The example uses the eaters database **DB** of Fig. 1. In addition to a database, ILP tasks contain *examples* to predict. The disjoint relations *Likes* and *Dislikes* of Fig. 3 contain *positive* and *negative* examples, respectively. We are interested in the following task: write a query Q such that $Likes \subseteq Q(\mathbf{DB})$ and $Dislikes \cap Q(\mathbf{DB}) = \{\}$. The first requirement, called *completeness*, says that the target query Q must yield all tuples of *Likes*. The second requirement, *consistency*, says that Q must yield no tuple of *Dislikes*.

Likes	*Eater*	*Food*
	Ron	Italian
	Ron	Greek
	Sam	Greek
	Tim	Italian

Dislikes	*Eater*	*Food*
	Ron	Mexican
	Sam	Italian
	Tim	Greek
	Tim	Junk

Fig. 3. Positive and negative examples

Consider the Datalog program P that uses the name *ans* to refer to the name of the target relation:

$$\begin{cases} ans(\text{Ron}, \text{Italian}) \leftarrow \\ ans(\text{Ron}, \text{Greek}) \leftarrow \\ ans(\text{Sam}, \text{Greek}) \leftarrow \\ ans(\text{Tim}, \text{Italian}) \leftarrow \end{cases} .$$

The query result $P(\mathbf{DB})$ is exactly the relation *Likes*, but, obviously, the program P is of no interest. The problem becomes more interesting if we restrict the query language to rule-based conjunctive queries. The Datalog program P itself is not conjunctive but is composed of four rule-based conjunctive queries. Any one rule from P in isolation is too specific, as each rule yields only one tuple of *Likes*. By generalizing these four rules of P into one, we obtain the following rule:

$$ans(e, f) \leftarrow \ ,$$

where e and f are variables. Note that this is not quite a rule-based conjunctive query, as the variables e and f appear in the head but not in the body of the rule. We can reasonably assume that e is an eater and f a dish:

$$ans(e, f) \leftarrow Visits(e, r, d), Dishes(f, x) \ .$$

This rule, which states that all eaters like all dishes, is complete, as it allows deriving every tuple of *Likes*, but is inconsistent, as it also yields all tuples of *Dislikes*. The rule is too general, so we are looking for a query that is properly contained in the above one. A rule can be restricted by substituting a constant for a variable. Alternatively, the body of the rule can be extended by adding new predicates. For example,

$$ans(e, f) \leftarrow Visits(e, r, d), Serves(r', f', c), Dishes(f, x), r' = r, f' = f \ ,$$

which can be simplified by unifying variables and by using the second integrity constraint in Fig. 1, giving

$$ans(e, f) \leftarrow Visits(e, r, d), Serves(r, f, c) \ .$$

The latter rule expresses that eaters like the food served in the restaurants they visit. The output of this query on input **DB** contains each tuple of *Likes* as well as $\langle \text{Tim}, \text{Mexican} \rangle$; it contains no tuple of *Dislikes*. Thus, we have found a solution to the initial problem.

4.2 Problem Statement

We now formalize the data mining task addressed in ILP, preceded by a brief recall of clausal languages.

We assume a first-order alphabet $\mathcal{A}$, containing predicate symbols, function symbols, constants, and variables. An *atom* has the form $P(\bar{x})$ where P is an n-ary predicate symbol and $\bar{x}$ is an n-tuple of variables, constants, and function terms. A *fact* is an atom without variables. A *positive literal* is an atom, a *negative literal* the negation of an atom. A *clause* is a finite set of literals. The clause $\{A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_n\}$, where $A_1, \ldots, A_m, B_1, \ldots, B_n$ are atoms, is treated as the universal closure of $A_1 \vee \ldots \vee A_m \vee \neg B_1 \vee \ldots \vee \neg B_n$, which is equivalently represented as $A_1 \vee \ldots \vee A_m \leftarrow B_1 \wedge \ldots \wedge B_n$ and abbreviated as $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$. The empty clause is denoted $\square$. A clause is *Horn* if it contains at most one positive literal.

Languages are defined relative to an alphabet $\mathcal{A}$. The *clausal language* $\mathcal{C}$ is the set of all clauses that can be constructed from the symbols in $\mathcal{A}$. The *Horn language* $\mathcal{H}$ is the set of all Horn clauses that can be constructed from the symbols in $\mathcal{A}$.

ILP systems construct logic programs from examples and background knowledge. Given background knowledge, positive and negative examples, the challenge is to find a hypothesis that is *consistent* and *complete*: the hypothesis in combination with the background knowledge must entail all positive examples (completeness) and none of the negative examples (consistency). The background knowledge and the hypothesis are expressed in two (not necessarily distinct) languages that are fixed in advance.

A usual setting is as follows. Positive and negative examples are given by two disjoint sets P and N of facts, respectively. The language for expressing background knowledge and the hypothesis is restricted to a Horn language $\mathcal{H}$. Then, given background knowledge $\Sigma \subseteq \mathcal{H}$, the task is to find a hypothesis $H \in \mathcal{H}$ such that $\Sigma \cup \{H\}$ logically implies each example of P (completeness) but no example of N (consistency). A complete and consistent hypothesis may not exist in the restricted hypothesis language. In that case, *recall*, *precision*, and *accuracy* measures can be used to assess the effectiveness of a hypothesis.

We can relate the above data mining task to the dimensions discussed in Sect. 2.1. We note that primary knowledge is considered background knowledge in ILP. The hypothesis language corresponds to the pattern search space, and the search criterion is captured by completeness and consistency with respect to a set of positive and negative examples.

4.3 Three Common Generality Orders

The two main operations in ILP for modifying an initial hypothesis are generalization and specialization. An initial hypothesis that is consistent (i.e. entails no negative examples) but incomplete (i.e. does not entail all positive

examples) can be generalized so as to capture more positive examples. *Over-generalization* has occurred when the generalized hypothesis also captures negative examples and hence has become inconsistent. Dually, a complete but inconsistent hypothesis can be specialized to make it consistent.

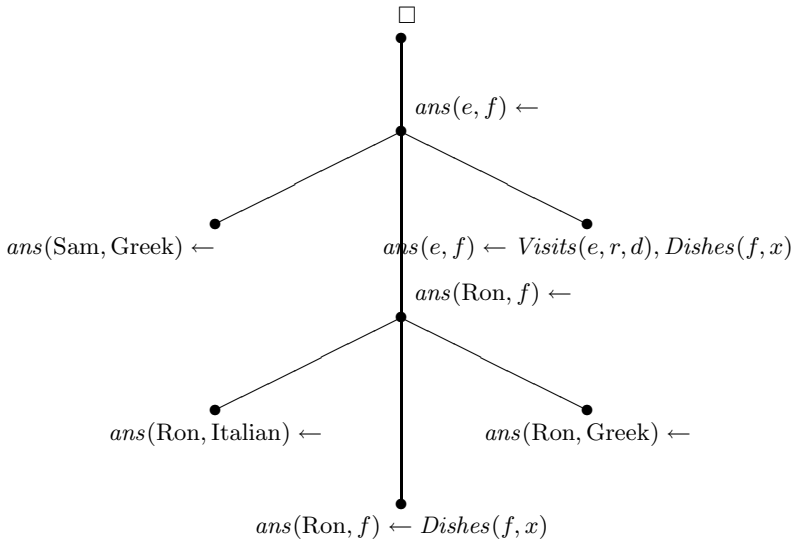

Fig. 4. Part of the quasi-order $\succeq$ for clauses about the eaters database

Generalization and specialization make sense only if there is an underlying *generality order*. The three most important generality orders used in ILP are as follows:

- *θ-subsumption.* A clause $C = \{L_1, \ldots, L_n\}$ θ-subsumes a clause $D = \{M_1, \ldots, M_n\}$, denoted by $C \succeq D$, if there is a substitution θ such that $C\theta \subseteq D$. For example, let $C = \{P(x, y), P(y, x)\}$ and $D = \{P(z, z)\}$. Let $\theta = \{x/z, y/z\}$. Then $C\theta = \{P(z, z)\} \subseteq D$, hence $C \succeq D$. Figure 4 shows a number of clauses ordered by $\succeq$.
- *Logical implication.* Let C and D be clauses. $C \models D$ denotes that $\{C\}$ logically implies D, where logical implication captures its natural semantics.[1]

[1] More precisely, the logical implication is between the universally quantified disjunctions represented by C and D, respectively.

- *Logical implication relative to background knowledge.* Let Σ be a first-order theory (i.e. a set of sentences) representing *background knowledge*. We write $C \models_\Sigma D$ if and only if $\Sigma \cup \{C\} \models D$.

It is easy to see that $\mathcal{C}$ or $\mathcal{H}$, equipped with any one of these orders, is a quasi-order.[2] The existence of least upper bounds and greatest lower bounds in these quasi-orders has been studied in [60].

Among the three orders, logical implication relative to background knowledge is semantically most meaningful and desirable. Despite this, ILP most often uses θ-subsumption for practical reasons: for Horn clauses with function symbols, the implication problem is undecidable [53], whereas deciding θ-subsumption is **NP**-complete [27, problem LO18][46].

The relative strengths of $\succeq$, $\models$, and $\models_\Sigma$ can be summarized as follows. It is easy to see that $C \models D$ implies $C \models_\Sigma D$, but the inverse does not hold. Next, $C \succeq D$ implies $C \models D$, but the inverse does not hold. θ-subsumption is strictly weaker than logical implication, even for function-free clauses. Finally, the Subsumption Theorem [61] states that if a set Σ of clauses logically implies a clause C, then either C is a tautology, or some clause D with $D \succeq C$ can be derived from Σ by resolution.[3]

4.4 Traversing the Search Space of Hypotheses

ILP aims at constructing a clausal hypothesis from examples and background knowledge. *Top-down* ILP systems such as FOIL [63,64] traverse the search space $\mathcal{C}$ or $\mathcal{H}$, ordered by θ-subsumption, from general to specific hypotheses. The most general hypothesis is the empty clause $\square$, which represents a contradiction. Clearly, the empty clause θ-subsumes any other clause, including all positive and negative examples, and hence constitutes a complete but inconsistent hypothesis. We can say that the hypothesis $\square$ is too strong because it allows deriving too many examples. The following basic syntactic operations allow weakening an initial hypothesis that is too strong:

- Adding a literal to a clause. For example,

  $$ans(e, f) \leftarrow Visits(e, r, d), Dishes(f, x) \ ,$$

  is weakened to

  $$ans(e, f) \leftarrow Visits(e, r, d), Dishes(f, x), Serves(r', f', c) \ .$$

- Substituting a term for a variable. For example, by substituting f for f', and r for r', the latter clause is further weakened to

  $$ans(e, f) \leftarrow Visits(e, r, d), Dishes(f, x), Serves(r, f, c) \ .$$

[2] A quasi-order is a binary relation that is reflexive and transitive.

[3] Roughly, resolution [68] is a sound inference rule saying that the clauses $C \cup \{L\}$ and $D \cup \{\neg L\}$ logically imply $C \cup D$.

Using these operations, one can restrict, step-by-step, an initial complete but inconsistent hypothesis, until one arrives at a most general hypothesis that is both complete and consistent.

In general, elementary specialization steps are executed by applying a so-called *downward refinement operator*, i.e. a mapping ρ from $\mathcal{C}$ to subsets of $\mathcal{C}$ such that for every $C \in \mathcal{C}$ and $D \in \rho(C)$, $C \succeq D$. On the input of an *individual* clause C, the downward refinement operator computes a set of clauses each of which is θ-subsumed by C. Of course, ρ can be restricted to Horn clauses. Starting from an initial hypothesis (for example, $\square$), the search space is traversed from general to specific hypotheses by recursively applying ρ until a satisfactory hypothesis is found. For this search to be effective, the downward refinement operator ρ must satisfy a number of properties, as investigated in [79]. A practical refinement operator will be steered by language bias, which will be discussed in Sect. 4.5.

Alternatively, *bottom-up* ILP systems such as GOLEM [56] traverse the search space from specific to general hypotheses. The most specific hypothesis can be represented by an artificial clause (True) that represents a tautology. This tautological clause is obviously consistent but incomplete, as it cannot derive any positive example. We can say that the tautological hypothesis is too weak. We can strengthen, step-by-step, a consistent but incomplete hypothesis until we arrive at a most specific hypothesis that is both consistent and complete.

4.5 Language Bias

In general, the hypothesis language allows expressing a very large or even infinite number of hypotheses. As already mentioned in Sect. 4.4, practical applications often require, for obvious reasons, reducing the search space of candidate hypotheses to be considered. This is mostly done by imposing further syntactic restrictions on hypotheses. Such restrictions are called *language bias* and can take very different forms. In the discovery of clauses, language bias could put an upper bound on the number of literals or the number of variables that can appear in a clause. A powerful language for specifying language bias is DLAB [20]. We now discuss three frequently encountered techniques for biasing a clausal language: type declarations, mode declarations, and metapatterns.

Typing. It is often natural to associate each n-ary predicate symbol P with a fixed n-tuple of *types*, denoted *type*(P). If *type*$(P) = \langle \mathtt{T_1}, \ldots, \mathtt{T_n} \rangle$ and a clause contains the atom $P(x_1, \ldots, x_n)$, then the variable x_i is said to be *of type* $\mathtt{T_i}$ ($i \in \{1, \ldots, n\}$). The syntactic restriction then requires that no variable be of two distinct types. In this way, the set of clauses to be considered is strongly reduced.

For example, we can introduce types R, F, C, E, and D for restaurants, food, cities, eaters, and dates, respectively. The type declarations are $type(Serves) = \langle \mathtt{R}, \mathtt{F}, \mathtt{C} \rangle$, $type(Visits) = \langle \mathtt{E}, \mathtt{R}, \mathtt{D} \rangle$, $type(ans) = \langle \mathtt{E}, \mathtt{F} \rangle$. Then,

$$ans(e, f) \leftarrow Visits(e, r, x), Serves(r, f, x)$$

would be illegal, as the variable x should be of both type D and C.

Modes. In a similar way, one can associate a *mode* with each occurrence of a variable in a clause, which specifies whether the variable is input (mode in) or output (mode out). As for typing, mode declarations are specified at the predicate level. Each n-ary predicate symbol P can be associated with two fixed n-tuples of modes: $modeh(P)$ applies if P occurs in the head of a clause, and $modeb(P)$ applies if P occurs in the body. The syntactic restrictions require that a variable that occurs with mode in in the body must also occur with mode in in the head or must occur in a preceding literal with mode out. The intuition is that a variable that occurs in an input position in the body but not in the head of a clause must be bound before being used. For example, the mode declarations

$$\begin{aligned} modeh(Fac) &= \langle \mathtt{in}, \mathtt{out} \rangle \\ modeb(Fac) &= \langle \mathtt{in}, \mathtt{out} \rangle \\ modeb(Decr) &= \langle \mathtt{in}, \mathtt{out} \rangle \\ modeb(Mult) &= \langle \mathtt{in}, \mathtt{in}, \mathtt{out} \rangle \ , \end{aligned}$$

allow the clause encoding the equation $x! = (x-1)! \times x$:

$$Fac(x, y) \leftarrow Decr(x, w), Fac(w, z), Mult(z, x, y) \ .$$

The second occurrence of w in this rule is of mode in and, as w does not occur in the head of the rule, must be preceded by an occurrence of w of mode out. The first occurrence of w is of mode out.

Metapatterns. A metapattern is a template or higher order expression that describes a type of pattern to be discovered. For example [72],

$$\begin{aligned} &\alpha(x, y) \leftarrow \beta(x, y) \\ &\alpha(x, y) \leftarrow \beta(x, z), \alpha(z, y) \ . \end{aligned}$$

In the above metaquery, α and β are syntactic placeholders for predicate symbols. The search space is limited to patterns that can be obtained from the above metaquery by substituting predicate symbols for α and β. The above metaquery thus expresses that the predicate symbol substituted for α must correspond to the transitive closure of some other predicate.

5 Multirelational Data Mining

In recent years, ILP has broadened its scope to cover standard data mining tasks such as classification, regression, clustering, and association rules. This research is triggered by the need to pass from single-relational to multirelational data mining, as explained next. Sections 5.2–5.4 discuss recent contributions of ILP to classification, clustering, and association rules.

5.1 First-Order Examples

In attribute-value (or propositional) learning, every example (case or instance) corresponds to a fact or, equivalently, to a tuple. Classification tasks additionally require labeling every example by a class. This representation falls short in domains where examples are structured objects. For example, in the chemical domain, examples may be molecules. It is impractical or even infeasible to capture, in a single tuple, the relative positioning of atoms to each other in a molecule.

In relational (or first-order) learning, every example is a set of facts or, equivalently, a (small) relational database. An identifier is used to distinguish examples. Figure 5 shows three examples, each one corresponding to an eater and containing up to three facts. The examples are classified as *Good* or *Bad*. The background knowledge Σ consists of the relations *Serves* and *Dishes* of Fig. 1. Note that because the relationship between eaters and restaurants is many-to-many, we cannot represent each example entirely by a single tuple. More illustrations of the advantages of the relational over the attribute-value representation can be found in [84].

For a long time, learning algorithms have been developed to deal with propositional representations. These algorithms do not apply directly in domains with structured objects, unless one first translates data from relational to propositional representations. This translation, also known as propositionalization [49], often generates a very large number of attributes and is not generally applicable. For this reason, a better solution is to "upgrade" attribute-value learning algorithms so that they can work directly with relational representations. This "upgrade approach" has been taken in several recent systems: TILDE [9] upgrades C4.5 [65], WARMR [21] upgrades propositional association rules, RDBC [47] upgrades a bottom-up agglomerative clustering algorithm, and RIBL [24] upgrades the distance-weighted k-nearest neighbor algorithm. In the following sections, we discuss, in more detail, first-order extensions of classification, clustering, and association rules. A general recipe for the upgrade approach has been proposed in [80].

5.2 First-Order Classification

TILDE [9] extends standard decision trees to capture first-order examples. A *first-order decision tree* takes the form of a propositional binary decision

Identifier	Facts	Class
Ron	$HasVisited(\text{Trevi})$	$Class(Good)$
	$HasVisited(\text{Pronto})$	
	$HasVisited(\text{Naxos})$	
Sam	$HasVisited(\text{Naxos})$	$Class(Bad)$
Tim	$HasVisited(\text{Pronto})$	$Class(Good)$
	$HasVisited(\text{Trevi})$	
	$HasVisited(\text{Taco})$	

Fig. 5. Three preclassified first-order examples

tree, where attribute-value tests are replaced by conjunctions of function-free atoms; an example is shown in Fig. 6. For each node N, let $CONJ(N)$ be the conjunction associated with N. To avoid semantic ambiguity, a syntactic restriction is imposed requiring that a variable x occurring in $CONJ(N)$ cannot appear in the right-hand tree emanating from the node N.

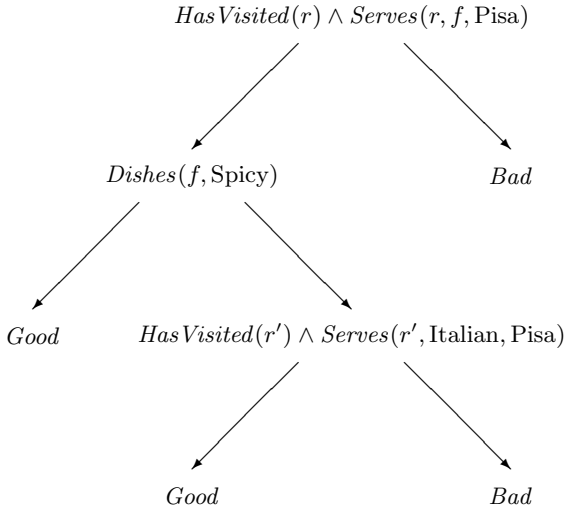

Fig. 6. First-order decision tree

As with propositional binary decision trees, classifying an example e means routing e down the tree until a terminal node or leaf is reached. To route an example e down the tree of Fig. 6, we first check whether $e \models_\Sigma \exists r, f(HasVisited(r) \wedge Serves(r, f, \text{Pisa}))$, i.e. has the eater ever visited a restaurant in Pisa. If a test succeeds at a node, the example descends along the left-hand branch; if a test fails, the right-hand branch is taken. Suppose that the test succeeds and the example is routed to the left. We then ask whether $e \models_\Sigma \exists r, f(HasVisited(r) \wedge Serves(r, f, \text{Pisa}) \wedge Dishes(f, \text{Spicy}))$, i.e.

has the eater ever visited a Pisa restaurant serving spicy food. Suppose that the test fails and the example is routed to the right child. The next question is whether $e \models_\Sigma \exists r'(\mathit{HasVisited}(r') \wedge \mathit{Serves}(r', \mathrm{Italian}, \mathrm{Pisa}))$, i.e. has the eater ever visited an Italian restaurant in Pisa. If so, the eater is classified as having good taste; otherwise the example is classified as bad.

Blockeel and De Raedt [9] provide operational semantics for logical decision trees in terms of an equivalent logic program. Next, we give a pure first-order logic semantics. The precise meaning of the tree can be described by associating, with each node N, a first-order formula, denoted $SAT(N)$, characterizing all examples that can reach that node:

1. If N is the root of the tree, then $SAT(N) := \texttt{true}$, expressing that all classifications start at the root of the tree.
2. If N is a nonleaf node with left child L and right child R, then $SAT(L) := SAT(N) \wedge CONJ(N)$ and $SAT(R) := SAT(N) \wedge \neg\exists^*(SAT(L))$.

If an example e reaches a node N, then it must satisfy the existential closure of $SAT(N)$ relative to background knowledge Σ, i.e. $e \models_\Sigma \exists^*(SAT(N))$. Practically, if an example e reaches a node N with left child L, we test whether $e \models_\Sigma \exists^*(SAT(L))$. If so, the example is routed to the left child L; otherwise, it is routed to the right. It can be verified that the tree of Fig. 6 classifies eaters as good if they have visited a restaurant in Pisa that serves Italian or spicy food.

The TILDE system [9] builds first-order decision trees from preclassified first-order examples. The distinguishing feature between attribute-value and first-order decision trees is the language that is used for the tests to be placed in the nodes. On the other hand, TILDE borrows its split, prune, and stop criteria from the tree-building algorithm C4.5, which works out because these criteria are language-independent. A task where C4.5 is of no help concerns the effective generation of candidate tests to split on, which becomes considerably more complex if tests can contain variables. This task is accomplished in TILDE by a refinement operator based on θ-subsumption that can be guided by different types of language bias.

There is an apparent correspondence between first-order decision trees and the traditional ILP view of inducing logic programs from examples. Good and bad eaters can be regarded as positive and negative examples, respectively. The first-order decision tree can be translated into a logic program that distinguishes positive from negative examples.

The k-nearest neighbor algorithm is another well-known classification method. This method relies on a distance metric between cases but is independent of the actual representation of cases. This means that the k-nearest neighbor algorithm can be applied to first-order examples as soon as we can compute the distance between two first-order examples. First-order distance metrics have been proposed, among others, in [8,11,59,67]. Reference [24] first defines a distance measure between relational examples and then uses it to upgrade the k-nearest neighbor algorithm.

Defining semantically meaningful distance measures between first-order examples is a difficult problem. Predicates that are present in one example may not even appear in the description of another example. Moreover, the relevance of different predicates and attributes has to be evaluated relative to background knowledge. For example, the distance between two eaters may rely on the actual restaurants visited, but just as well on the city, the type, and even the food aspects of these restaurants.

5.3 First-Order Clustering

Recall from Sect. 2.4 that the aim of clustering is to divide a given set of examples into several subsets (called clusters) such that examples of the same cluster are similar, whereas examples belonging to different clusters are not. Given a distance metric between examples, this amounts to minimizing the distance between examples of the same cluster (intracluster distance) and maximizing the distance between examples of different clusters (intercluster distance). The term conceptual clustering is used if extensionally defined clusters are characterized by intentional concept descriptions that are often organized in concept hierarchies [77].

Most standard clustering algorithms rely on a distance metric between examples. Some clustering algorithms, such as k-means clustering, additionally require the computation of the center of a cluster. If we want to upgrade these algorithms to a first-order setting, we need to extend the concepts of distance and center to first-order examples. RDBC [47] uses the first-order distance metric proposed in [11] to upgrade a bottom-up agglomerative clustering algorithm. Definitions of cluster center in a first-order setting are proposed in [48].

In line with conceptual clustering, TIC [10] also provides a first-order characterization of clusters. A first-order sentence ϕ splits a set S of (first-order) examples into two sets: $P := \{e \mid e \models_\Sigma \phi\}$ and $N := \{e \mid e \not\models_\Sigma \phi\}$. Given a distance measure that computes the distance between two clusters of examples, a first-order clustering problem can then be defined as finding the formula ϕ that maximizes the (intercluster) distance between P and N. By applying the split recursively, we can construct a hierarchy of clusters, each of which is characterized by a first-order sentence. The TIC system restricts the language of ϕ to the tests that can be used in the first-order decision trees discussed in Sect. 5.2. Under this restriction, a *clustering tree* is just a first-order decision tree, except that the leaves of the tree do not contain class labels but store sets of training cases. Despite this resemblance, the algorithms for building clustering trees and decision trees are quite different: whereas TILDE aims at discriminating examples that belong to different classes, splitting in TIC aims at separating examples that are at a great distance from each other.

The quality of a clustering tree can be measured in different ways. One can verify whether the tree yields cohesive clusters in a set of unseen examples

or whether the tree accurately predicts the values of attributes missing from test cases. In the latter case, new examples are sorted down the tree, and missing values are inferred from the clusters stored in the leaves of the tree.

5.4 First-Order Association Rules

The association rules introduced in Sect. 2.4 can be called *propositional* because they can be represented using propositional logic. WARMR [21,22] was the first system for the discovery of "multirelational" association rules. We now present a generalized description of the problem in terms of database queries, which first appeared in [34].

If Q_1 and Q_2 are queries such that $Q_2 \subseteq Q_1$, then $Q_1 \Rightarrow Q_2$ is called a *first-order association rule*.[4] For a given database $\mathbf{DB}$, the *confidence* of this rule is defined as $\frac{|Q_2(\mathbf{DB})|}{|Q_1(\mathbf{DB})|}$, a rational number between zero and one. The *support* is defined as a fraction proportional to $|Q_2(\mathbf{DB})|$. For example,

$$Q_1 = ans(e) \leftarrow Visits(e, \text{Naxos}, d)$$
$$Q_2 = ans(e) \leftarrow Visits(e, \text{Naxos}, d), Visits(e, r, d'), Serves(r, \text{Italian}, c) \ .$$

The first query asks which eaters visit the Naxos restaurant; the second query asks which eaters visit the Naxos restaurant and, moreover, visit a restaurant serving Italian food. Clearly, $Q_2 \subseteq Q_1$. Then the confidence of $Q_1 \Rightarrow Q_2$ is the conditional probability that an eater visits a restaurant serving Italian food provided that he visits Naxos. For the eaters database, Q_1 yields the eaters Ron and Sam, and Q_2 yields only Ron; so the confidence of the above rule is 0.5.

The data mining problem of interest is to discover first-order association rules that exceed given support and confidence thresholds. For semantical or performance reasons, restrictions can be put on the query language used for expressing Q_1 and Q_2. For rule-based conjunctive queries, further syntactic restrictions can be put on the use of constants, variables, and predicate symbols. In particular, a rule is called *unirelational* if it contains only one predicate symbol.

Unsurprisingly, propositional association rules can be obtained by imposing straightforward syntactic restrictions on first-order association rules. The propositional association rules discussed in Sect. 2.4 are unirelational. In particular, the rule "$City = \text{Pisa} \Rightarrow Food = \text{Italian}$" corresponds to $Q_1 \Rightarrow Q_2$, where

$$Q_1 = ans(r) \leftarrow Serves(r, f, \text{Pisa})$$
$$Q_2 = ans(r) \leftarrow Serves(r, \text{Italian}, \text{Pisa}) \ .$$

[4] $Q_2 \subseteq Q_1$ means that for any database $\mathbf{DB}$, $Q_2(\mathbf{DB}) \subseteq Q_1(\mathbf{DB})$. We refer to [1] for a detailed treatment of query containment.

The Boolean association rule $\{\text{Pronto}, \text{Naxos}\} \Rightarrow \{\text{Taco}\}$ corresponds to $Q_1 \Rightarrow Q_2$, where

$$\begin{aligned} Q_1 &= ans(e) \leftarrow Visits(e, \text{Pronto}, d),\ Visits(e, \text{Naxos}, d') \\ Q_2 &= ans(e) \leftarrow Visits(e, \text{Pronto}, d),\ Visits(e, \text{Naxos}, d'), \\ &\qquad\quad Visits(e, \text{Taco}, d") \ . \end{aligned}$$

If the visits in the antecedent have to precede those in the consequent, it suffices to add the inequalities $d \leq d"$ and $d' \leq d"$ to the body of Q_2. This allows capturing the `MINE RULE` query described in Sect. 3.2, exhibiting a point of convergence between different perspectives. Such points of convergence are discussed next.

6 A Vision of Convergence

So far, we have presented two main perspectives on the contribution of logic to data mining. The inductive database perspective presented in Sect. 3 aims to amalgamate mining and querying. Mined knowledge is stored in the database along with primary knowledge and thus becomes available for further querying. Queries then serve two goals: inducing new knowledge and inquiring into previously induced knowledge. We showed how Datalog++ allows specifying data mining problems as well as computing solutions. Sections 4–5 describe an ILP perspective, in which the main objective is to enhance the expressive power of standard data mining techniques to model and solve more complex problems. The ILP perspective has generally assumed loose coupling between data mining engines and databases.

The inductive database and ILP perspectives have been studied rather independently in the literature. An interesting and important task is to combine both perspectives in an overall framework capable of covering the different dimensions of a data mining task (see Sect. 2.1): primary knowledge sources, pattern search space, search criteria, and background knowledge. In this way, multirelational data mining can meet the potentials offered by integrating mining and querying present in deductive databases.

Interesting in this respect is the logical language RDM proposed by De Raedt [19]. A feature of the approach is the use of terms for conjunctive queries. More precisely, a constant can be a conjunctive query, and a variable can be a placeholder for a conjunctive query. RDM queries can be regarded as higher order queries that seek for standard conjunctive queries by means of certain constraints. In the remainder of this section, we present RDM using the database jargon.

In the technical treatment, we assume a fixed relational database schema $\mathbf{S}$, and we denote by $\mathbf{Q}$ the class of all conjunctive queries over $\mathbf{S}$. We assume denumerably many *query variables* $q_1, q_2, \ldots$ It is implicitly understood that query variables range over the class $\mathbf{Q}$. Three possible constraints involving query variables are presented next.

Subsumption Constraints are introduced to further restrict the range of query variables; they are of the form $Q \subseteq q_1$, $q_1 \subseteq Q$, or $q_1 \subseteq q_2$, where q_1, q_2 are query variables and $Q \in \mathbf{Q}$. Satisfaction of a subsumption constraint is relative to a query variable assignment, i.e. a mapping from query variables to $\mathbf{Q}$; semantics can be defined in the natural way (not elaborated here). The constraint $Q \subseteq q_1$, for example, restricts the range of q_1 to conjunctive queries that contain Q. It is important to note that in this context the symbol $\subseteq$ stands for standard query containment, i.e. $Q_1 \subseteq Q_2$ if and only if for any database $\mathbf{DB}$, $Q_1(\mathbf{DB}) \subseteq Q_2(\mathbf{DB})$. It is well known [1] that containment $Q_1 \subseteq Q_2$ of conjunctive queries coincides with the existence of a substitution, called homomorphism, from Q_2 to Q_1. Hence, a subsumption constraint imposes syntactic restrictions on the queries in the range of a query variable. For example, let *AllResto* and *ProGreek* denote the following conjunctive queries in what follows:

$$\begin{aligned} \mathit{AllResto} := \mathit{ans}(e) \leftarrow\ & \mathit{Visits}(e, \text{Pronto}, d_1),\ \mathit{Visits}(e, \text{Trevi}, d_2), \\ & \mathit{Visits}(e, \text{Naxos}, d_3),\ \mathit{Visits}(e, \text{Taco}, d_4) \\ \mathit{ProGreek} := \mathit{ans}(e) \leftarrow\ & \mathit{Visits}(e, \text{Naxos}, d)\ . \end{aligned}$$

The subsumption constraint $\mathit{AllResto} \subseteq q_1$ then implies, among others, that the query assigned to q_1 must not contain constants other than Pronto, Trevi, Naxos, or Taco. Also, the head of that query must be of the form $\mathit{ans}(v)$, or even $\mathit{ans}(e)$ after variable renaming v/e. The body cannot contain predicates other than *Visits*. The constraint $q_1 \subseteq \mathit{ProGreek}$ means that q_1 is restricted to conjunctive queries that can yield only eaters who have once visited Naxos.

Frequency Constraints. In data mining, we are typically interested in the strength of a rule relative to some given database $\mathbf{DB}$. To this extent, we introduce *frequency constraints* which use $\mathit{cnt}(q_1)$, where q_1 is a query variable, to refer to the cardinality of the answer set $q_1(\mathbf{DB})$ (up to the substitution of a query for the variable q_1, of course). Frequency constraints can then be defined as equalities and inequalities involving counts. For example, the constraint,

$$\mathit{AllResto} \subseteq q_1, c = \mathit{cnt}(q_1), c \geq 3\ ,$$

is satisfied by the queries $Q \in \mathbf{Q}$ ranging over the search space defined by *AllResto*, having support greater than three. The extension to other aggregate functions is obvious.

Coverage Constraints are used to state that a target query must yield a specified answer from a given database. A *positive* coverage constraint is of the form $t \in q_1$, where t is a ground fact and q_1 is a query variable. The constraint is satisfied relative to a database $\mathbf{DB}$ if and only if $t \in q_1(\mathbf{DB})$ (again up to an assignment of a query to the variable q_1). A *negated* coverage

constraint $t \notin q_1$ can be used to state that a given negative example must not be covered. For example, the constraint,

$$\mathit{AllResto} \subseteq q_1, \mathit{ans}(\text{Tim}) \in q_1, \mathit{ans}(\text{Ed}) \notin q_1 \ ,$$

restricts q_1 to the queries whose answers include Tim but not Ed in the database under consideration. Coverage constraints allow capturing the problems dealt with in standard ILP. Recall in this respect that ILP was introduced in Sect. 4 as the problem of finding a conjunctive query that covers all positive examples but no negative example.

RDM Queries are defined by extending standard rule-based conjunctive queries with query variables and the above subsumption, frequency, and coverage constraints. An example is the RDM query (we use the predicate *out* for the answer to an RDM query):

$$\mathit{out}(q) \leftarrow \mathit{AllResto} \subseteq q, q \subseteq \mathit{ProGreek}, \mathit{cnt}(q) \geq 1 \ ,$$

which restricts the search space to conjunctive queries between (w.r.t. $\subseteq$) *AllResto* and *ProGreek*. An answer is $\mathit{out}(Q)$, where Q is the following conjunctive query:

$$Q := \mathit{ans}(e) \leftarrow \mathit{Visits}(e, \text{Pronto}, d_1), \mathit{Visits}(e, \text{Naxos}, d) \ .$$

RDM is powerful enough to ask for complex patterns. The predicate *Rule* allows capturing the first-order association rules introduced in Sect. 5.4:

$$\mathit{Rule}(q_1, q_2) \leftarrow \mathit{AllResto} \subseteq q_2, q_2 \subseteq q_1 \ .$$

We have, for example, $\mathit{Rule}(Q_1, Q_2)$ with

$$\begin{aligned} Q_1 := \mathit{ans}(e) \leftarrow\ & \mathit{Visits}(e, \text{Pronto}, d_1), \mathit{Visits}(e, \text{Trevi}, d_2) \\ Q_2 := \mathit{ans}(e) \leftarrow\ & \mathit{Visits}(e, \text{Pronto}, d_1), \mathit{Visits}(e, \text{Trevi}, d_2), \\ & \mathit{Visits}(e, \text{Taco}, d_3) \ . \end{aligned}$$

As explained before, $\mathit{Rule}(Q_1, Q_2)$ can be treated as an association rule saying that visitors of Pronto and Trevi also visit Taco. Note incidentally that *Rule* also yields answers that do not correspond to standard association rules. For example, $\mathit{Rule}(Q_1, Q_2')$ with

$$\begin{aligned} Q_2' := \mathit{ans}(e) \leftarrow\ & \mathit{Visits}(e, \text{Pronto}, d_1), \mathit{Visits}(e, \text{Trevi}, d_2), \\ & \mathit{Visits}(e', \text{Taco}, d_3) \ . \end{aligned}$$

$\mathit{Rule}(Q_1, Q_2')$ states that if an eater visits both Pronto and Trevi, then some (possibly other) eater visits Taco.

Closure means that predicates defined by RDM queries can be used everywhere a base predicate is allowed. For example, the predicate *Vain* defined in terms of *Rule*:

$$Vain(q_1, q_2) \leftarrow Rule(q_1, q_2), Rule(q_2, q_1) \ .$$

Vain retains the worthless rules whose left- and right-hand sides are equivalent. Finally, the RDM query,

$$out(q_1, q_2, c) \leftarrow Rule(q_1, q_2), c = \frac{cnt(q_2)}{cnt(q_1)}, c \geq 0.7, \neg Vain(q_1, q_2) \ ,$$

returns nonvain association rules with confidence greater than 0.7.

De Raedt [19] points out a relationship between RDM and constraint logic programming: each conjunct in the body of an RDM query imposes certain constraints on the conjunctive queries (or rules) to be discovered. Data mining tasks can thus be specified in RDM by imposing appropriate constraints on the search space and the search criteria. In particular, subsumption constraints are useful for specifying the search space, and frequency and coverage constraints can be used to specify search criteria.

7 Concluding Remarks

Data mining includes such tasks as classification, clustering, discovery of clauses, and discovery of association rules. Much research has been spent on the development of efficient algorithms for these tasks. Illustrative of this are the numerous algorithms for mining Boolean association rules that have appeared in the database literature.

So far, relatively little attention has been paid to supporting the interactive and exploratory nature of data mining tasks. At the beginning of a data mining session, a user has typically no clear idea about where to head. He will proceed in a "trial and error" fashion, by executing the same data mining task multiple times with different search criteria (for example, different threshold values), by filtering and refining the output of a data mining task through queries, and by "chaining" mining and querying. A coherent formalism capable of dealing uniformly with induced, deduced, and background knowledge would represent a major breakthrough in the design and development of data mining applications.

In the ILP community, the concern for the expressiveness of data mining methods has resulted in a shift from the traditional attribute-value setting, which requires that each example be a single tuple, to a multirelational setting, where each example is a (small) relational database. This research has resulted in an upgrade of standard data mining techniques from a propositional to a first-order logic framework, thus allowing more expressive patterns and models.

An orthogonal, database-oriented approach assumes that discovered objects (classifiers, clustering, rules) are treated as first-class citizens of the database (known as closure) and that data mining is amalgamated with database querying. These assumptions are commonly captured by the term "inductive database." The ultimate goal is to integrate and transcend the different dimensions of a data mining task: the primary knowledge sources, the search space of patterns to be discovered, the search criteria, and the background knowledge. The inductive database paradigm thus asks for extending database languages with data mining primitives and for ensuring the principle of closure. SQL has been extended in several ways to meet these needs. An alternative route is to construct deductive databases in Datalog extended with user-defined aggregates. The latter approach facilitates the representation and use of background knowledge.

An important problem is to find elegant ways of encoding discovered knowledge in (relational) databases. We showed that first-order association rules can be captured by (pairs of) conjunctive queries. These rules can be stored in a natural way if the database provides an abstract data type (ADT) for conjunctive queries. The operations for this ADT should include, among others, a test for query containment or, equivalently, for θ-subsumption. In this way, the task of association rule mining can be accomplished by inductive database querying. However, it is less clear how other data mining tasks can be treated along the same lines.

Finally, we stress that KDD is larger than data mining stricto sensu, including also data preparation and actionability issues. Logic has turned out to be important in data preparation tasks like data cleaning and data integration.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading, MA, 1995.
2. R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 207–216, 1993.
3. R. Agrawal and K. Shim. Developing tightly-coupled data mining applications on a relational database system. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pp. 287–290, 1996.
4. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. Int. Conf. Very Large Data Bases*, pp. 487–499, 1994.
5. D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey data reduction report. *Data Engineering Bulletin*, 20(4):3–45, 1997.
6. R. J. Bayardo Jr., R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. In *Proc. 15th Int. Conf. on Data Engineering (ICDE'99)*, pp. 188–197, 1999.

7. M. J. A. Berry and G. Linoff. *Data Mining Techniques for Marketing, Sales, and Customer Support*. Wiley, New York, 1997.
8. G. Bisson. Learning in FOL with a similarity measure. In *Proc. 10th National Conf. on Artificial Intelligence (AAAI'92)*, pp. 82–87, 1992.
9. H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1–2):285–297, 1998.
10. H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *Proc. 15th Int. Conf. on Machine Learning (ICML'98)*, pp. 55–63, 1998.
11. U. Bohnebeck, T. Horváth, and S. Wrobel. Term comparisons in first-order similarity measures. In *Proc. 8th Int. Workshop on Inductive Logic Programming (ILP'98)*, LNAI 1446, pp. 65–79, 1998.
12. J.-F. Boulicaut, M. Klemettinen, and H. Mannila. Querying inductive databases: A case study on the MINE RULE operator. In *Proc. 2nd European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98)*, LNCS 1510, pp. 194–202, 1998.
13. J.-F. Boulicaut, P. Marcel, and C. Rigotti. Query driven knowledge discovery in multidimensional data. In *Proc. of the ACM 2nd Int. Workshop on Data Warehousing and OLAP (DOLAP'99)*, pp. 87–93, 1999.
14. T. Calders, R. T. Ng, and J. Wijsen. Searching for dependencies at multiple abstraction levels. *ACM Trans. on Database Systems*, 27(3):229–260, 2002.
15. T. Calders and J. Wijsen. On monotone data mining languages. In *Proc. 8th Int. Workshop on Database Programming Languages (DBPL'01)*, LNCS 2397, pp. 119–132, Springer, 2002.
16. S. Chaudhuri, U. M. Fayyad, and J. Bernhardt. Scalable classification over SQL databases. In *Proc. 15th Int. Conf. on Data Engineering (ICDE'99)*, pp. 470–479, 1999.
17. 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.
18. S. Choenni and A. Siebes. Query optimization to support data mining. In *Proc. Int. Workshop on Database and Expert Systems Applications (DEXA'97)*, pp. 658–663, 1997.
19. L. De Raedt. A logical database mining query language. In *Proc. 10th Int. Conf. on Inductive Logic Programming (ILP'00)*, LNAI 1866, pp. 78–92, 2000.
20. L. Dehaspe and L. De Raedt. DLAB: A declarative language bias formalism. In *Proc. Int. Symposium on Foundations of Intelligent Systems (ISMIS'96)*, LNCS 1079, pp. 613–622, 1996.
21. L. Dehaspe and H. Toivonen. Discovery of frequent DATALOG patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
22. L. Dehaspe and H. Toivonen. Discovery of relational association rules. In A. Džeroski and N. Lavrač, editors, *Relational Data Mining*, Chap. 8, pp. 189–212, Springer, 2001.
23. A. Džeroski and N. Lavrač, editors. *Relational Data Mining*. Springer, Berlin, 2001.
24. W. Emde and D. Wettschereck. Relational instance-based learning. In *Proc. 13th Int. Conf. on Machine Learning (ICML'96)*, pp. 122–130, 1996.
25. A. Famili, W.-M. Shen, R. Weber, and E. Simoudis. Data preprocessing for intelligent data analysis. *Intelligent Data Analysis*, 1(1), 1997.

26. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, Cambridge, MA, 1996.
27. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979.
28. F. Giannotti and G. Manco. Querying inductive databases via logic-based user-defined aggregates. In *Proc. 3rd European Conf. on Principles of Data Mining and Knowledge Discovery (PKDD'99)*, LNAI 1704, pp. 125–135, 1999.
29. F. Giannotti and G. Manco. Making knowledge extraction and reasoning closer. In *Proc. 4th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'00)*, LNAI 1805, pp. 360–371, 2000.
30. 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.
31. F. Giannotti, G. Manco, and F. Turini. Specifying mining algorithms with iterative user-defined aggregates: A case study. In *Proc. 5th European Conf. on Principles of Data Mining and Knowledge Discovery (PKDD'01)*, LNAI 2168, pp. 128–139, 2001.
32. F. Giannotti, D. Pedreschi, and C. Zaniolo. Semantics and expressive power of nondeterministic constructs in deductive databases. *Journal of Computer and System Sciences*, 62(1):15–42, 2001.
33. B. Goethals and J. Van den Bussche. On supporting interactive association rule mining. In *Proc. of the 2nd Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK'00)*, LNCS 1874, pp. 307–316, 2000.
34. B. Goethals and J. Van den Bussche. Relational association rules: Getting WARMeR. In *Proc. of the ESF Exploratory Workshop on Pattern Detection and Discovery*, LNCS 2447, pp. 125–139, 2002.
35. G. Graefe, U. M. 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 (KDD'98)*, pp. 204–208, 1998.
36. J. Han. Towards on-line analytical mining in large databases. *SIGMOD Record*, 27(1):97–107, 1998.
37. J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A data mining query language for relational databases. In *Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'96)*, 1996.
38. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, 2000.
39. J. Han, L. Lakshmanan, and R. T. Ng. Constraint-based multidimensional data mining. *IEEE Computer*, 32(8):46–50, 1999.
40. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 1–12, 2000.
41. D. J. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, Cambridge, MA, 2001.
42. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Commun. of the ACM*, 39(11):58–64, 1996.
43. T. Imielinski and A. Virmani. MSQL: A query language for database mining. *Data Mining and Knowledge Discovery*, 3(4):373–408, 1999.

44. T. Imielinski, A. Virmani, and A. Abdulghani. DMajor-Application programming interface for database mining. *Data Mining and Knowledge Discovery*, 3(4):347–372, 1999.
45. A. K. Jain, M. N. Murthy, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.
46. D. Kapur and P. Narendran. NP-completeness of the set unification and matching problems. In *Proc. 8th Int. Conf. on Automated Deduction*, LNCS 230, pp. 489–495, 1986.
47. M. Kirsten and S. Wrobel. Relational distance-based clustering. In *Proc. 8th Int. Workshop on Inductive Logic Programming (ILP'98)*, LNAI 1446, pp. 261–270, 1998.
48. M. Kirsten and S. Wrobel. Extending k-means clustering to first-order representations. In *Proc. 10th. Int. Conf. on Inductive Logic Programming (ILP'00)*, LNCS 1866, pp. 112–119, 2000.
49. S. Kramer, N. Lavrač, and P. Flach. Propositionalization approaches to relational data mining. In A. Džeroski and N. Lavrač, editors, *Relational Data Mining*, Chap. 11, pp. 262–291, Springer, 2001.
50. G. Manco. Foundations of a Logic-Based Framework for Intelligent Data Analysis. Ph.D. Thesis, Department of Computer Science, University of Pisa, 2001.
51. H. Mannila. Inductive databases and condensed representations for data mining. In *Proc. Int. Symposium on Logic Programming (ILPS'97)*, pp. 21–30, 1997.
52. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
53. J. Marcinkowski and L. Pacholski. Undecidability of the Horn-clause implication problem. In *Proc. of 33rd Annual IEEE Symposium on the Foundations of Computer Science*, pp. 354–362, 1992.
54. R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery*, 2(2):195–224, 1998.
55. T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.
56. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proc. of the 1st International Workshop on Algorithmic Learning Theory (ALT'90)*, pp. 368–381, 1990.
57. A. Netz, S. Chaudhuri, U. M. Fayyad, and J. Bernhardt. Integrating data mining with SQL databases: OLE DB for data mining. In *Proc. 17th Int. Conf. on Data Engineering (ICDE'01)*, pp. 379–387, 2001.
58. R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 13–24, 1998.
59. S.-H. Nienhuys-Cheng. Distance between Herbrand interpretations: A measure for approximations to a target concept. In *Proc. 7th Int. Workshop on Inductive Logic Programming (ILP'97)*, LNAI 1297, pp. 213–226, 1997.
60. S.-H. Nienhuys-Cheng and R. de Wolf. Least generalizations and greatest specializations of sets of clauses. *Journal of Artificial Intelligence Research*, 4:341–363, 1996.
61. S.-H. Nienhuys-Cheng and R. de Wolf. The subsumption theorem in inductive logic programming: Facts and fallacies. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pp. 265–276, IOS Press, 1996.

62. G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pp. 229–248, AAAI/MIT Press, 1991.
63. J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
64. J. R. Quinlan and R. M. Cameron-Jones. Induction of logic programs: FOIL and related systems. *New Generation Computing*, 13(3&4):287–312, 1995.
65. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA, 1993.
66. E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.
67. J. Ramon and M. Bruynooghe. A framework for defining distances between first-order logic objects. In *Proc. 8th Int. Workshop on Inductive Logic Programming (ILP'98)*, LNCS 1446, pp. 271–280, 1998.
68. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of ACM*, 12(1):23–41, 1965.
69. R. Sadri, C. Zaniolo, A. M. Zarkesh, and J. Adibi. A sequential pattern query language for supporting instant data mining for e-services. In *Proc. 27th Int. Conf. on Very Large Data Bases (VLDB'01)*, pp. 653–656, 2001.
70. S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. *Data Mining and Knowledge Discovery*, 4(2/3):89–125, 2000.
71. A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 21th Int. Conf. on Very Large Data Bases (VLDB'95)*, pp. 432–444, 1995.
72. W.-M. Shen, K. Ong, B. G. Mitbander, and C. Zaniolo. Metaqueries for data mining. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pp. 375–398, AAAI/MIT Press, 1996.
73. A. Siebes and M. L. Kersten. KESO: Minimizing database interaction. In *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, pp. 247–250, 1997.
74. C. Silverstein, S. Brin, and R. Motwani. Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2(1):39–68, 1998.
75. R. Srikant and R. Agrawal. Mining generalized association rules. *Future Generation Computer Systems*, 13(2/3):161–180, 1997.
76. R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, pp. 67–73, 1997.
77. K. Thompson and P. Langley. Concept formation in structured domains. In D. H. Fisher, M. J. Pazzani, and P. Langley, editors, *Concept Formation: Knowledge and Experience in Unsupervised Learning*, pp. 127–161. Morgan Kaufmann, 1991.
78. S. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: A generalization of association-rule mining. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 1–12, 1998.
79. P. R. J. van der Laag and S.-H. Nienhuys-Cheng. Completeness and properness of refinement operators in inductive logic programming. *Journal of Logic Programming*, 34(3):201–225, 1998.

80. W. Van Laer and L. De Raedt. How to upgrade propositional learners to first order logic: A case study. In A. Džeroski and N. Lavrač, editors, *Relational Data Mining*, Chap. 10, pp. 235–261, Springer, 2001.
81. H. Wang and C. Zaniolo. Using SQL to build new aggregates and extenders for object-relational systems. In *Proc. 26th Int. Conf. on Very Large Data Bases (VLDB'00)*, pp. 166–175, 2000.
82. S. M. Weiss and N. Indurkhya. *Predictive Data Mining: A Practical Guide*. Morgan Kaufmann, San Francisco, CA, 1997.
83. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, CA, 1999.
84. S. Wrobel. Inductive logic programming for knowledge discovery in databases. In A. Džeroski and N. Lavrač, editors, *Relational Data Mining*, Chap. 4, pp. 74–101, Springer, 2001.
85. 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 (DOOD'93)*, LNCS 760, pp. 204–221, 1993.
86. C. Zaniolo and H. Wang. Logic-based user-defined aggregates for the next generation of database systems. In K. R. Apt, V. W. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Perspective*, pp. 401–426, Springer, 1999.

Index