5 Inductive Logic Programming and Knowledge Discovery in Databases

Sašo Džeroski
Jožef Stefan Institute

Abstract

Inductive logic programming (ILP) can be viewed as machine learning in a first-order language, where relations are present in the context of deductive databases. It is relevant for knowledge discovery in relational and deductive databases, as it can describe patterns involving more than one relation. The chapter introduces ILP and gives a survey of basic ILP techniques, as well as other ILP topics relevant to KDD. KDD applications of ILP are discussed, two of which are described in some detail: biological classification of river water quality and predicting protein secondary structure.

5.1 KDD and Machine Learning

Knowledge discovery in databases (KDD) is concerned with identifying interesting patterns and describing them in a concise and meaningful manner (Frawley et al. 1991). Frawley et al. (1991) also recognize that machine learning can be applied to KDD problems. If we interpret patterns as classes or concepts, clustering (unsupervised learning) can be used to identify patterns and concept learning (supervised) to describe them.

Attribute-value learning systems, such as C4.5 (Quinlan 1993), ASSISTANT (Cestnik et al. 1987), and CN2 (Clark and Boswell 1991), can and have been applied to KDD problems. However, these approaches have serious limits. The patterns discovered by the above learning systems are expressed in attribute-value languages which have the expressive power of propositional logic. These languages are limited and do not allow for representing complex structured objects and relations among objects or their components. The domain (background) knowledge that can be used in the learning process is also of a very restricted form and other relations from the database cannot be used in the learning process.

Recent developments in inductive learning focus on the problem of constructing a logical (intensional) definition of a relation (Quinlan 1990) from example tuples known to belong or not to belong to it, where other relations (background knowledge) may be used in the induced definition.

Table 5.1
A Relational Database with Two Relations (tables): Potential-Customer and Married-To.

Potential-Customer

Person	Age	Sex	Income	Customer
Ann Smith	32	F	10 000	yes
Joan Gray	53	F	1 000 000	yes
Mary Blythe	27	F	20 000	no
Jane Brown	55	F	20 000	yes
Bob Smith	30	M	100 000	yes
Jack Brown	50	M	200 000	yes

Married-To

Husband	Wife
Bob Smith	Ann Smith
Jack Brown	Jane Brown

In this way, new relations can be specified by a small number of example tuples, which are then generalized to induce a logical definition. Alternatively, existing relations can be compressed into their corresponding logical definitions.

As the intensional definitions induced can be recursive (Quinlan 1990), we may say that they are expressed in the formalism of *deductive databases* (Ullman 1988). The logic programming school in deductive databases (Lloyd 1987) argues that deductive databases can be effectively represented and implemented using logic and logic programming. Within this framework, the induction of logical definitions of relations can be considered logic program synthesis and has been recently named *Inductive Logic Programming (ILP)* (Muggleton 1992a; De Raedt 1992; Lavrač and Džeroski 1994).

Early ILP systems, e.g., MIS (Shapiro 1983) and CIGOL (Muggleton and Buntine 1988), did not address the problem of handling noisy data. On the other hand, recent ILP systems, such as FOIL (Quinlan 1990) and LINUS (Lavrač et al. 1991), are capable of noise-handling, as well as handling continuous-valued attributes. In this way, they come closer towards applicability for KDD problems.

Before proceeding further, let us illustrate the use of attribute-value learning and ILP for KDD. Consider the relational database shown in Table 5.1, containing information about the potential customers of an imaginary enterprise. Suppose we are interested in distinguishing between persons who are potential customers and those who are not (field Customer in the table Potential-Customer).

We could use an attribute-value learning system on the table Potential-Customer, where the field Customer would be the class and the fields Age, Sex and Income the attributes. In this way, the following two patterns could be discovered:

IF Income(Person) $\geq$ 100 000 **THEN** Potential-Customer(Person)
IF Sex(Person) = F **AND** Age(Person) $\geq$ 32 **THEN** Potential-Customer(Person).

If we use an ILP system to define the relation Potential-Customer in terms of itself and the relation Married-To, we can find the following patterns:
IF Married(Person,Spouse) **AND** Income(Person) $\geq$ 100 000
THEN Potential-Customer(Spouse)
IF Married(Person,Spouse) **AND** Potential-Customer(Person)
THEN Potential-Customer(Spouse).

The latter is written as $potential\_customer(Spouse, SAge, SSex, SIncome) \leftarrow married(Person, Spouse), potential\_customer(Person, Age, Sex, Income)$ in logic programming notation.

The above example illustrates that attribute-value learning systems can be used to discover patterns in an isolated file of database records, i.e., a single relation in a relational database, where records (tuples) can be viewed as training instances. ILP, on the other hand, can be used to discover patterns involving several relations in a database. If we want to learn concepts that involve several relations in a database using propositional approaches, we have to resort to creating a single universal relation. In the above example, the universal relation would include the fields Spouse-Age, Spouse-Sex, Spouse-Income, and Spouse-Customer, in addition to the fields of the relation Potential-Customer. This may be inconvenient and impractical (the universal relation can be very large) and suggests the use of ILP approaches for KDD.

The chapter first introduces the problem of inductive logic program-

Table 5.2
A Simple ILP problem: learning the *daughter* relationship.

Training examples	Background knowledge			
⊕ daughter(sue,eve).	mother(eve,sue).	parent(X,Y) ←	female(ann).	male(pat).
⊕ daughter(ann,pat).	mother(ann,tom).	mother(X,Y).	female(sue).	male(tom).
⊖ daughter(tom,ann).	father(pat,ann).	parent(X,Y) ←	female(eve).	
⊖ daughter(eve,ann).	father(tom,sue).	father(X,Y).		

ming by giving some terminology, a definition of the problem and a simple example. It then gives a survey of basic ILP techniques, including relative least general generalization, inverse resolution, searching refinement graphs, using rule models, and transforming ILP problems to propositional form. Recent advances in ILP relevant to KDD are outlined next. Finally, KDD applications of ILP are discussed: applications of ILP to biological classification of river water quality (Džeroski et al. 1994), and prediction of the secondary structure of a protein from its amino-acid sequence (Muggleton et al. 1992).

5.2 Inductive Logic Programming

This section first gives a simple example that will be used later to illustrate how ILP techniques work. It proceeds to introduce the necessary database and logic programming terminology. It then discusses several dimensions of ILP and finally defines the problem of empirical inductive logic programming.

5.2.1 An Example ILP Problem

Let us illustrate the ILP task on a simple problem of learning family relations. The task is to define the relation $daughter(X, Y)$, which states that person X is a daughter of person Y, in terms of the background knowledge relations $female$ and $parent$. These relations are given in Table 5.2. There are two positive and two negative examples of the *target* relation *daughter*.

In the hypothesis language of logic programs, it is possible to formulate the following definition of the target relation:

$$daughter(X, Y) \leftarrow female(X), parent(Y, X).$$

This definition is a view that defines the target relation intensionally in terms of the background knowledge relations.

Table 5.3
Database and logic programming terms.

DB terminology	LP terminology
relation name p	predicate symbol p
attribute of relation p	argument of predicate p
tuple $\langle a_1, \ldots, a_n \rangle$	ground fact $p(a_1, \ldots, a_n)$
relation p -	predicate p -
a set of tuples	defined extensionally by a set of ground facts
relation q	predicate q
defined as a view	defined intensionally by a set of rules (clauses)

5.2.2 Deductive Databases and Logic Programming

Before discussing ILP further, let us first introduce some basic terminology from the areas of logic programming and deductive databases. An n-ary *relation* p is a set of tuples, i.e., a subset of the Cartesian product of n domains $D_1 \times D_2 \times \ldots \times D_n$, where a *domain* (or a *type*) is a set of values. We assume that a relation is finite unless stated otherwise. A set of relations forms a *relational database* (RDB) (Ullman 1988).

A deductive database (DDB) consists of a set of *database clauses*. A database clause is a *typed program clause* of the form:

$$p(X_1, \ldots, X_n) \leftarrow L_1, \ldots, L_m.$$

where the body of a clause is a conjunction of positive literals $q_i(Y_1, \ldots, Y_{n_i})$ and/or negative literals *not* $q_j(Y_1, \ldots, Y_{n_j})$. The basic difference between program clauses and database clauses is in the use of types. In typed clauses, a type is associated with each variable appearing in a clause. The type of a variable specifies the range of values which the variable can take. For example, in the relation $lives\_in(X, Y)$, we may want to specify that X is of type *person* and Y is of type *city*.

A set of program clauses with the same predicate symbol p in the head forms a *predicate definition*. A predicate can be defined *extensionally* as a set of ground facts or *intensionally* as a set of database clauses (Ullman 1988). It is assumed that each predicate is either defined extensionally or intensionally. A relation in a database is essentially the same as a predicate definition in a logic program. Table 5.3 relates the database (Ullman 1988) and logic programming (Lloyd 1987) terms that will be used throughout the chapter.

Database clauses use variables and function symbols in predicate argu-

ments. Recursive types and recursive predicate definitions are allowed. *Deductive hierarchical databases* (DHDB) (Lloyd 1987) are deductive databases restricted to nonrecursive predicate definitions and to nonrecursive types. The latter determine finite sets of values which are constants or structured terms with constant arguments.

5.2.3 Dimensions of ILP

Inductive logic programming systems (as well as other inductive learning systems) can be divided along several dimensions. First of all, they can learn either a *single concept* or *multiple concepts* (predicates). Second, they may require all the training examples to be given before the learning process (*batch learners*) or may accept examples one by one (*incremental learners*). Third, during the learning process, a learner may rely on an oracle to verify the validity of generalizations and/or classify examples generated by the learner. The learner is called *interactive* in this case and *non-interactive* otherwise. Fourth, a learner may invent new terms (new predicates). This results in extending the learner's vocabulary which may facilitate the learning task. Finally, a learner may try to learn a concept from scratch or can accept an initial hypothesis (theory) which is then revised in the learning process. The latter systems are called *theory revisors.*

Although these dimensions are in principle independent, existing ILP systems are situated at two ends of the spectrum. At one end are batch non-interactive systems that learn single predicates from scratch, while at the other are interactive and incremental theory revisors that learn multiple predicates. Following (De Raedt 1992) we call the first type of ILP systems *empirical ILP systems* and the second type *interactive ILP systems.*

MIS (Shapiro 1983) and CLINT (De Raedt 1992) are typical interactive ILP systems: they learn definitions of multiple predicates from a small set of examples and queries to the user. On the other hand, empirical ILP systems typically learn a definition of a single predicate from a large collection of examples. This class of ILP systems includes FOIL (Quinlan 1990), GOLEM (Muggleton and Feng 1990), and LINUS (Lavrač et al. 1991). At present, it seems that empirical ILP systems are more relevant to practical applications. Thus, the applications described in this chapter involve empirical ILP systems, an empirical ILP problem will be used to illustrate the basic ILP techniques, and the discussion will

concern mainly empirical ILP. However, the basic techniques described are used in both interactive and empirical ILP systems.

5.2.4 Empirical Inductive Logic Programming

The task of empirical single predicate learning in ILP can be formulated as follows:

Given:

- a set of training examples $\mathcal{E}$, consisting of true $\mathcal{E}^+$ and false $\mathcal{E}^-$ ground facts of an unknown predicate p,
- a concept description language $\mathcal{L}$, specifying syntactic restrictions on the definition of predicate p,
- background knowledge $\mathcal{B}$, defining predicates q_i (other than p) which may be used in the definition of p

Find:

- a definition $\mathcal{H}$ for p, expressed in $\mathcal{L}$, such that $\mathcal{H}$ is complete, i.e., $\forall e \in \mathcal{E}^+ : \mathcal{B} \wedge \mathcal{H} \vdash e$, and consistent, i.e., $\forall e \in \mathcal{E}^- : \mathcal{B} \wedge \mathcal{H} \nvdash e$, with respect to $\mathcal{E}$ and $\mathcal{B}$.

One usually refers to the true facts $\mathcal{E}^+$ as *positive ($\oplus$) examples*, the false facts $\mathcal{E}^-$ as *negative ($\ominus$) examples* and the definition of p as the definition of the *target* relation. Positive examples are tuples known to belong to the target relation, while negative examples are tuples known not to belong to the target relation. A definition of the target predicate is sought that will characterize the examples precisely (i.e., will explain all of the positive and none of the negative examples). When learning from noisy examples, the completeness and consistency criteria need to be relaxed, i.e., an approximate characterization of the examples is considered sufficient.

The symbol $\vdash$ denotes the relation of derivability under the assumption of a particular deductive inference rule, such as resolution: $T \vdash e$ means that e can be deductively derived from T. In database terms, $T \vdash e$ would mean that the query e succeeds when posed to the database T. The task of empirical ILP is to construct an intensional definition (view) for the target relation p in terms of other relations from a given

database. One possible use of the constructed view is to replace the facts (examples) about p, thus saving storage space.

The language of concept descriptions $\mathcal{L}$ is usually called the *hypothesis language*. It is typically some subset of the language of program clauses. The complexity of learning grows with the expressiveness of the hypothesis language $\mathcal{L}$. Therefore, to limit the complexity of learning, $\mathcal{L}$ has to be limited by imposing restrictions on hypothesized clauses. Typical restrictions on the concept description language include the restriction to DHDB clauses and the restriction to constrained clauses (clauses that do not introduce new variables in the body).

5.3 Basic ILP Techniques

ILP techniques search for clauses in the hypothesis language that are consistent with the training examples. The space of clauses is structured by the generality relation between clauses, called θ-subsumption. A clause c is more general than clause d if c can be matched with a subset of d. For example, the clause $daughter(X, Y) \leftarrow female(X)$ is more general than the clause $daughter(ann, Y) \leftarrow female(ann), parent(Y, ann)$.

The basic ILP techniques considered here have been all used in empirical ILP systems and are thus relevant to KDD. Some of the techniques have been also used or even first introduced in interactive ILP systems. The techniques summarized below are: relative least general generalization, inverse resolution, searching refinement graphs, using rule models, and transforming ILP problems to propositional form.

5.3.1 Relative Least General Generalization

Plotkin's notion of least general generalization (lgg) (Plotkin 1969) is important for ILP since it forms the basis of cautious generalization. Cautious generalization assumes that if two clauses c_1 and c_2 are true, it is very likely that their most specific generalization $lgg(c_1, c_2)$ will also be true. The lgg of two clauses is computed by computing the lgg of each pair of literals in the heads and bodies, respectively. The lgg of two literals is computed by matching them and substituting the parts that don't match by a variable. We will not give the definition of lgg here, but will rather illustrate it on an example. If $c_1 = daughter(mary, ann) \leftarrow female(mary), parent(ann, mary)$ and $c_2 =$

$daughter(eve, tom) \leftarrow female(eve), parent(tom, eve)$, then $lgg(c_1, c_2) = daughter(X, Y) \leftarrow female(X), parent(Y, X)$ (X stands for $lgg(mary, eve)$ and Y for $lgg(ann, tom)$).

The *relative least general generalization* (*rlgg*) of two clauses c_1 and c_2 is the least general clause which is more general than both c_1 and c_2 with respect (*relative*) to background knowledge $\mathcal{B}$. Relative least general generalization is the basic technique used in the ILP system GOLEM (Muggleton and Feng 1990), where the background knowledge $\mathcal{B}$ is restricted to ground facts. If K denotes the conjunction of all these facts, the *rlgg* of two ground atoms A_1 and A_2 (positive examples), relative to $\mathcal{B}$ is defined as:

$$rlgg(A_1, A_2) = lgg((A_1 \leftarrow K), (A_2 \leftarrow K))$$

Given the positive examples $e_1 = daughter(mary, ann)$ and $e_2 = daughter(eve, tom)$ and the background knowledge $\mathcal{B}$ for the family example in Section 5.2.1, the least general generalization of e_1 and e_2 relative to $\mathcal{B}$ is computed as:

$$rlgg(e_1, e_2) = lgg((e_1 \leftarrow K), (e_2 \leftarrow K))$$

where K denotes the conjunction of the literals $parent(ann, mary)$, $parent(ann, tom)$, $parent(tom, eve)$, $parent(tom, ian)$, $female(ann)$, $female(mary)$, and $female(eve)$.

For notational convenience, the following abbreviations are used: *d-daughter, p-parent, f-female, a-ann, e-eve, m-mary, t-tom, i-ian.* The conjunction of facts from the background knowledge (comma stands for conjunction) is

$$K = p(a, m), p(a, t), p(t, e), p(t, i), f(a), f(m), f(e).$$

The computation of $rlgg(e_1, e_2)$ produces the following clause

$$\begin{aligned} d(V_{m,e}, V_{a,t}) \leftarrow\ & p(a, m), p(a, t), p(t, e), p(t, i), f(a), f(m), f(e), \\ & p(a, V_{m,t}), p(V_{a,t}, V_{m,e}), p(V_{a,t}, V_{m,i}), p(V_{a,t}, V_{t,e}), \\ & p(V_{a,t}, V_{t,i}), p(t, V_{e,i}), f(V_{a,m}), f(V_{a,e}), f(V_{m,e}). \end{aligned}$$

where $V_{x,y}$ stands for $rlgg(x, y)$, for each x and y. In general, a *rlgg* of a set of training examples can contain infinitely many literals or at least grow exponentially with the number of examples. Since such a

clause can be intractably large, GOLEM uses a determinacy constraint on introducing new variables into the body of the *rlgg*. Even under this constraint, *rlggs* are usually long clauses with many irrelevant literals.

Eliminating irrelevant literals yields the clause

$$d(V_{m,e}, V_{a,t}) \leftarrow p(V_{a,t}, V_{m,e}), f(V_{m,e}),$$

which stands for $daughter(X,Y) \leftarrow female(X), parent(Y,X)$.

The ILP system GOLEM based on *rlgg* has been applied to several practical problems, including protein secondary structure prediction (predicting the shape of a protein from its sequence of amino acids) (Muggleton et al. 1992) and learning structure-activity relationships (relating the effectiveness of a drug, e.g., the degree to which it slows bacterial growth, to its chemical structure) (King et al. 1992).

5.3.2 Inverse Resolution

The basic idea of *inverse resolution* (introduced as a generalization technique to ILP by Muggleton and Buntine (1988)) is to invert the *resolution* rule of deductive inference (Robinson 1965). The basic resolution step in propositional logic derives the proposition $p \vee r$ given the premises $p \vee \bar{q}$ and $q \vee r$. In first-order logic, resolution is more complicated, involving substitutions. The conclusion reached from clauses c and d by a resolution inference step is denoted by $res(c,d)$ and is called the resolvent of c and d.

To illustrate resolution in first-order logic, we use the example from Section 5.2.1. Suppose that background knowledge $\mathcal{B}$ consists of the clauses $b_1 = female(mary)$ and $b_2 = parent(ann, mary)$ and $\mathcal{H} = \{c\}$ $= \{daughter\ (X,Y) \leftarrow female(X),\ parent(Y,X)\}$. Let $\mathcal{T} = \mathcal{H} \cup \mathcal{B}$. Suppose we want to derive the fact $daughter(mary, ann)$ from $\mathcal{T}$. To this end, we proceed as follows:

- First, the resolvent $c_1 = res(c, b_1)$ is computed under substitution $\theta_1 = \{X/mary\}$. This means the substitution θ_1 is first applied to c to obtain $daughter(mary, Y) \leftarrow female(mary)$, $parent(Y, mary)$, which is then resolved with b_1 as in the propositional case. The resolvent of $daughter(X,Y) \leftarrow female(X), parent(Y,X)$ and $female(mary)$ is thus $c_1 = res(c, b_1) = daughter(mary, Y) \leftarrow parent\ (Y, mary)$.

- The next resolvent $c_2 = res(c_1, b_2)$ is computed under the substitution $\theta_2 = \{Y/ann\}$. The clauses $daughter(mary, Y) \leftarrow parent(Y, mary)$ and $parent(ann, mary)$ resolve in $c_2 = res(c_1, b_2) = daughter(mary, ann)$.

Inverse resolution, as implemented in CIGOL (Muggleton and Buntine 1988), uses a generalization operator based on inverting substitution (Buntine 1988). Given a clause W, an *inverse substitution* θ^{-1} of a substitution θ is a function that maps terms in $W\theta$ to variables, such that $W\theta\theta^{-1} = W$. Let $c = daughter(X, Y) \leftarrow female(X), parent(Y, X)$ and the substitution $\theta = \{X/mary, Y/ann\}$:

$$c' = c\theta = daughter(mary, ann) \leftarrow female(mary), parent(ann, mary).$$

By applying the inverse substitution $\theta^{-1} = \{mary/X, ann/Y\}$ the original clause c is obtained:

$$c = c'\theta^{-1} = daughter(X, Y) \leftarrow female(Y), parent(Y, X).$$

In the general case, inverse substitution is substantially more complex. It involves the *places* of terms in order to ensure that the variables in the initial clause W are appropriately restored in $W\theta\theta^{-1}$.

Let the background knowledge $\mathcal{B}$ contain the two facts (clauses) $b_1 = female(mary)$ and $b_2 = parent(ann, mary)$. Let the current hypothesis $\mathcal{H} = \emptyset$. Suppose that the learner encounters the positive example $e_1 = daughter(mary, ann)$. The inverse resolution process might then proceed as follows:

- In the first step, inverse resolution attempts to find a clause c_1 which will, together with b_2, entail e_1 and can be added to the current hypothesis $\mathcal{H}$ instead of e_1. Using the inverse substitution $\theta_2^{-1} = \{ann/Y\}$, an inverse resolution step generates the clause $c_1 = ires(b_2, e_1) = daughter(mary, Y) \leftarrow parent(Y, mary)$. Clause c_1 becomes the current hypothesis $\mathcal{H}$, such that $\{b_2\} \cup \mathcal{H} \vdash e_1$.
- Inverse resolution then takes $b_1 = female(mary)$ and the current hypothesis $\mathcal{H} = \{c_1\} = \{daughter(mary, Y) \leftarrow parent(Y, mary)\}$. By computing $c' = ires(b_1, c_1)$, using the inverse substitution $\theta_1^{-1} = \{mary/X\}$, it generalizes the clause c_1 with respect to background knowledge $\mathcal{B}$, yielding $c' = daughter(X, Y) \leftarrow female(X)$,

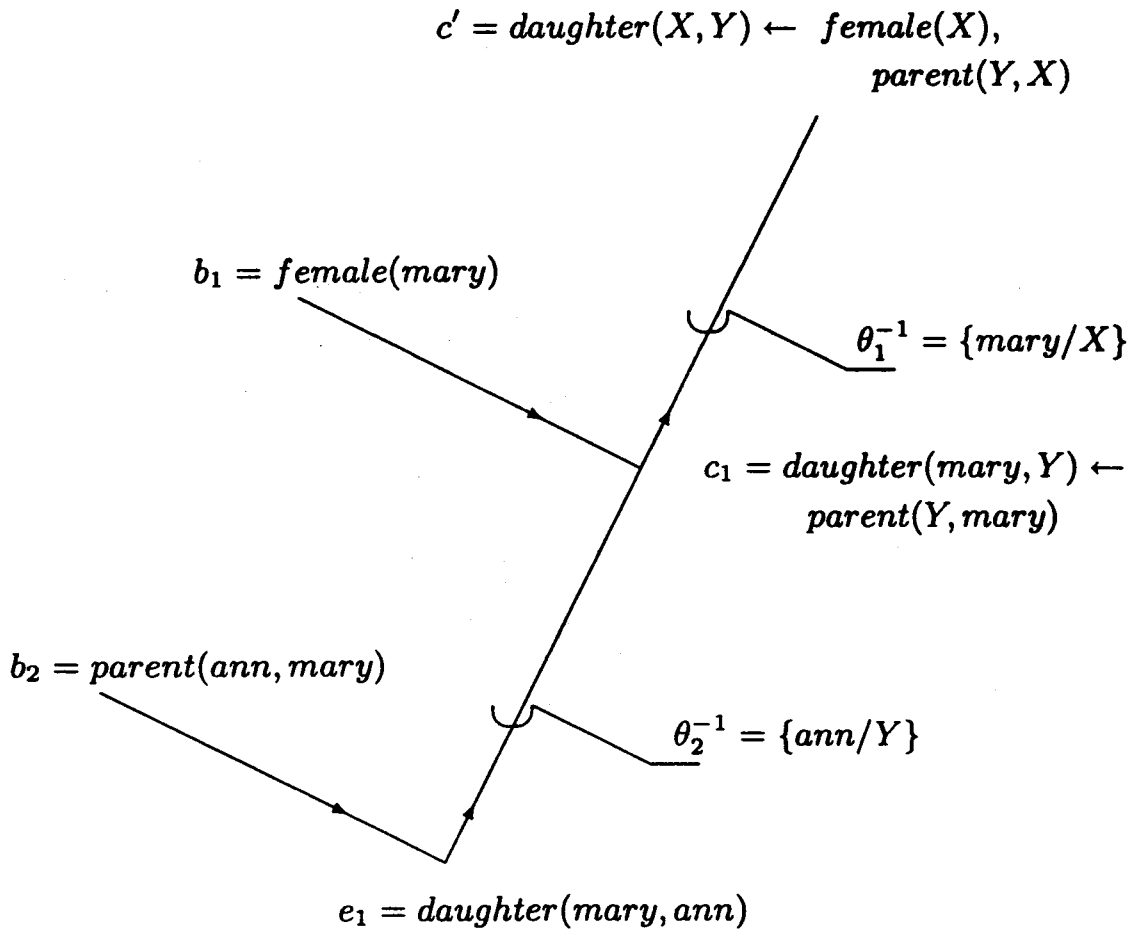

Figure 5.1
An inverse linear derivation tree.

$parent(Y,X)$. In the current hypothesis $\mathcal{H}$, clause c_1 can now be replaced by the more general clause c' which together with $\mathcal{B}$ entails the example e_1. The induced hypothesis is $\mathcal{H} = \{daughter(X,Y) \leftarrow female(X), parent(Y,X)\}$.

The corresponding inverse linear derivation tree is illustrated in Figure 5.1.

The generalization operator illustrated in the above example is called the *absorption* operator (the **V** operator), and is used in the interactive ILP systems MARVIN (Sammut and Banerji 1986) and CIGOL (Muggleton and Buntine 1988) to generate hypotheses. Several other inverse resolution operators are used in CIGOL. An important operator is *intra-construction*, also called the **W** operator, which combines two **V** operators. This operator generates clauses using predicates which are not available in the initial learner's vocabulary. The ability to introduce new predicates is referred to as *predicate invention*.

While most systems based on inverting resolution are interactive, an empirical system based on mode-directed inverse resolution (PROGOL [Muggleton 1995]) has been recently developed. It has been applied to the problem of predicting the mutagenicity of chemical compounds from their structure (Srinivasan et al. 1994).

5.3.3 Searching Refinement Graphs

The basic specialization ILP technique is *top-down search of refinement graphs*. Top-down learners start from the most general clause and repeatedly refine (specialize) it until it no longer covers negative examples. During the search they ensure that the clause considered covers at least one positive example. The ILP technique of top-down search of a refinement graph was first used in the interactive ILP system MIS (Model Inference System (Shapiro 1983)). It is also used in the empirical ILP system FOIL (Quinlan 1990) and its derivatives, e.g., mFOIL (Džeroski and Bratko 1992) and FOCL (Pazzani and Kibler 1992).

For a selected hypothesis language $\mathcal{L}$ and given background knowledge $\mathcal{B}$, the hypothesis space of program clauses is a lattice, structured by the θ-subsumption generality ordering. In this lattice, a *refinement graph* can be defined as a directed, acyclic graph in which *nodes* are program clauses and *arcs* correspond to the basic refinement operations: substituting a variable with a term and adding a literal to the body of a clause.

Part of the refinement graph for the family relations problem is depicted in Figure 5.2. The search for a clause starts with the clause $daughter(X, Y) \leftarrow$, which covers two negative examples. The refinements of this clause are then considered, of which $daughter(X, Y) \leftarrow female(X)$ and $daughter(X, Y) \leftarrow parent(Y, X)$ cover only one negative example each. These two clauses have a common refinement

$$daughter(X, Y) \leftarrow female(X), parent(Y, X)$$

which covers no negative examples. The search of the refinement lattice is usually heuristic, as for instance in FOIL (Quinlan 1990).

5.3.4 Using Rule Models

The system MOBAL (Morik et al. 1993) contains a learning module based on the use of *rule models*. Rule models are a form of declarative bias, more specifically declarative language bias. They explicitly specify the

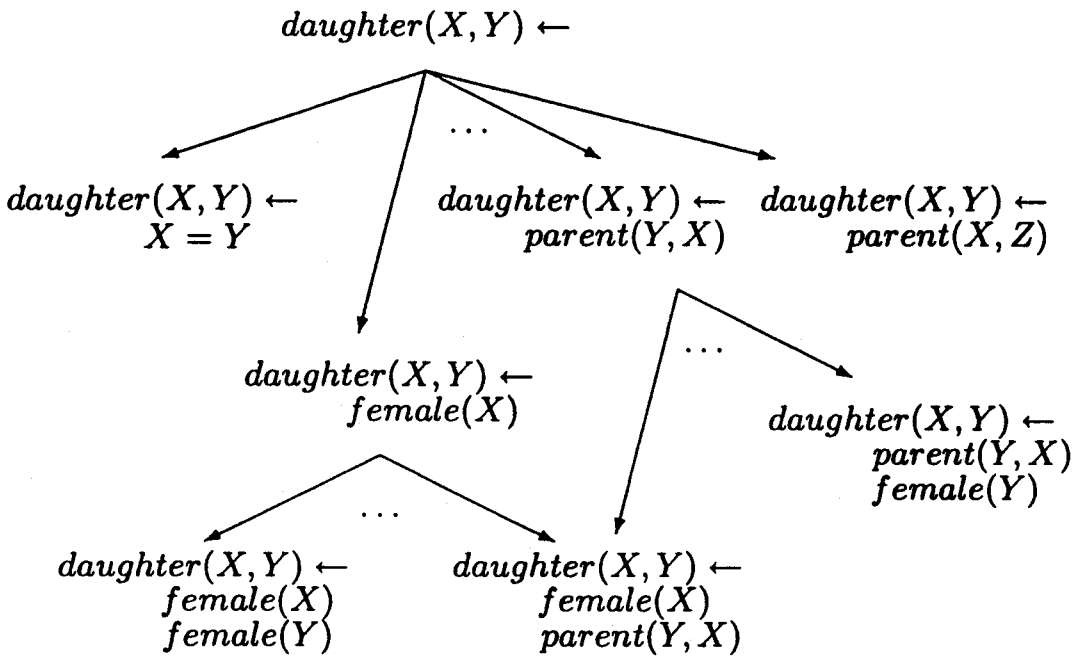

Figure 5.2
Part of the refinement graph for the family relations problem.

form of clauses that can appear in the hypothesis and are used as templates for constructing hypotheses. They are provided by the user or can be derived from previously learned rules by the model acquisition tool within MOBAL.

A *rule model* R has the form $T \leftarrow L_1, \ldots, L_m$ where T and L_i are literal schemas. Each *literal schema* L_i has the form $Q_i(Y_1, \ldots, Y_{n_i})$ or *not* $Q_j(Y_1, \ldots, Y_{n_j})$ where all non-predicate variables Y_k are implicitly universally quantified. The *predicate variable* Q_i can be instantiated only to a predicate symbol of the specified arity n_i.

The search for hypotheses in MOBAL is guided by the rule models. The hypothesis space that is actually searched is the set of all predicate ground instantiations of rule models with predicates from the background knowledge $\mathcal{B}$. An *instantiation* Θ is a second-order substitution, i.e., a finite set of pairs P/p where P is a predicate variable and p is a predicate symbol. An instantiation of a rule model R (denoted by $R\Theta$) is obtained by replacing each predicate variable in R by the corresponding predicate symbol from the set Θ. A second-order rule model R is *predicate ground* if all of its predicate variables are instantiated; in this case, R becomes a first-order rule (clause).

For learning the *daughter* relation from the family relations problem

of Section 5.2.1, the user needs to define the following model:

$$P(X,Y) \leftarrow R(X), Q(Y,X).$$

In this case, the correct clause $daughter(X,Y) \leftarrow female(X), parent(Y,X)$ can be induced. The above model could also be instantiated to the clause $son(X,Y) \leftarrow male(X)$, $parent(Y,X)$. We can thus see that one rule model can be useful for solving several learning problems.

The space of clauses specified by the rule models is still very large. For each rule model, all possible instantiations of predicate variables are tried systematically and the resulting clauses are tested against the examples and background knowledge. A hierarchical order of rule models, based on extended θ-subsumption, allows MOBAL to prune parts of the hypothesis space.

MOBAL has been applied to the practical problem of learning access rules for telecommunications equipment in a security context (Morik et al. 1993).

5.3.5 Transforming ILP Problems to Propositional Form

The ILP system LINUS (Lavrač et al. 1991; Lavrač and Džeroski 1994) is based on the following idea: background knowledge can give rise to new attributes for propositional learning. An ILP problem is transformed from relational to attribute-value form and solved by an attribute-value learner. This approach is feasible only for a restricted class of ILP problems. Thus, the hypothesis language is restricted to function-free program clauses which are *typed* (each variable is associated with a predetermined set of values), *constrained* (all variables in the body of a clause also appear in the head), and *nonrecursive* (the predicate symbol from the head does not appear in any of the literals in the body), i.e., to function-free constrained DHDB clauses.

The LINUS algorithm consists of the following three steps:

- The learning problem is transformed from relational to attribute-value form.
- The transformed learning problem is solved by an attribute-value learner.
- The induced hypothesis is transformed back into relational form.

Table 5.4
Propositional form of the *daughter* relation problem.

Class	Variables		Propositional features							
	X	Y	f(X)	f(Y)	m(X)	m(Y)	p(X,X)	p(X,Y)	p(Y,X)	p(Y,Y)
⊕	mary	ann	true	true	false	false	false	false	true	false
⊕	eve	tom	true	false	false	true	false	false	true	false
⊖	tom	ann	false	true	true	false	false	false	true	false
⊖	eve	ann	true	true	false	false	false	false	false	false

The above algorithm allows for a variety of approaches developed for propositional problems, including noise-handling techniques in attribute-value algorithms, such as ASSISTANT (Cestnik et al. 1987) or CN2 (Clark and Niblett 1989; Clark and Boswell 1991), to be used for learning relations. It is illustrated on the simple ILP problem of learning family relations from Section 5.2.1. The task is to define the relation $daughter(X, Y)$, which states that person X is a daughter of person Y, in terms of the background knowledge relations *female*, *male* and *parent*. All the variables are of type *person*, which is defined as $person = \{ann, eve, ian, mary, tom\}$. There are two positive and two negative examples of the target relation. The training examples and the relations from the background knowledge are given in Table 5.2.

The first step of the algorithm, i.e., the transformation of the ILP problem into attribute-value form, is performed as follows. The possible applications of the background predicates on the arguments of the target relation are determined, taking into account argument types. Each such application introduces a new attribute. In our example, all variables are of the same type *person*. The corresponding attribute-value learning problem is given in Table 5.4, where f stands for *female*, m for *male* and p for *parent*. The attribute-value tuples are generalizations (relative to the given background knowledge) of the individual facts about the target relation.

In Table 5.4, *variables* stand for the arguments of the target relation, and *propositional features* denote the newly constructed attributes of the propositional learning task. When learning function-free clauses, only the new attributes are considered for learning.

In the second step, an attribute-value learning program induces the following if-then rule from the tuples in Table 5.4:

$$Class = \oplus \quad \textbf{if} \quad [female(X) = true] \quad \wedge \quad [parent(Y, X) = true]$$

In the last step, the induced if-then rules are transformed into DHDB

clauses. In our example, we get the clause

$$daughter(X,Y) \leftarrow female(X), parent(Y,X).$$

LINUS has been applied to the problem of learning diagnostic rules for early rheumatic diseases from a medical database (Lavrač et al. 1993). DINUS (Džeroski et al. 1992; Lavrač and Džeroski 1994) is an extension of LINUS that learns determinate clauses, the same class of clauses learned by GOLEM (Muggleton and Feng 1990). It has been applied to the problem of behavioral cloning, i.e., learning control rules for dynamic systems (Džeroski et al. 1995).

Note that several different techniques may be employed within a single ILP system. For example, rule models in MOBAL are used in combination with top-down search. Bidirectional search, combining top-down and bottom-up search for clauses, may also be used in an ILP system (Siou 1994).

5.4 Recent Developments in ILP Relevant to KDD

This section outlines three recent advances in ILP relevant to KDD: multiple predicate learning (De Raedt et al. 1993), inductive data engineering (Flach 1993), and clausal discovery (De Raedt and Bruynooghe 1993).

5.4.1 Multiple Predicate Learning

The task of empirical multiple predicate learning is to learn the logical definitions of m different predicates $p_1, \ldots, p_m$, from a set of training examples $\mathcal{E}$ of these predicates and a background theory $\mathcal{B}$, containing predicate definitions for $q_1, \ldots, q_l$ (De Raedt et al. 1993). It is similar to empirical single predicate learning, but involves additional problems that arise when learning mutually dependent predicates. These include the consideration of clauses only at a local level, determining the order of learning the predicates, overgeneralization and mutual recursion.

An example task of multiple predicate learning would be to learn the definitions of the target predicates $male\_ancestor(X,Y)$ and $female\_ancestor(X,Y)$ in terms of the background knowledge predicates $mother(X,Y)$ and $father(X,Y)$. To illustrate the problems encountered when learning multiple predicates, consider the following definitions induced by FOIL (Quinlan 1990).

$$male\_ancestor(X,Y) \leftarrow$$
$$father(X,W), mother(M,W), female\_ancestor(M,Y)$$

$$female\_ancestor(X,Y) \leftarrow$$
$$mother(X,W), father(F,W), male\_ancestor(F,Y)$$

The definitions are correct, but not operational: they would cause an endless recursion if executed. No ground facts about the target relations can be derived from these definitions.

The ILP system MPL (De Raedt et al. 1993) has been designed specifically for learning multiple predicates. It is based on a top-down search and considers clauses both locally and globally, backtracking when inconsistencies are detected. It induces the following correct definition of the predicates $male\_ancestor(X,Y)$ and $female\_ancestor(X,Y)$.

$$female\_ancestor(X,Y) \leftarrow mother(X,Y).$$
$$male\_ancestor(X,Y) \leftarrow father(X,Y).$$
$$female\_ancestor(X,Y) \leftarrow mother(X,Z), female\_ancestor(Z,Y).$$
$$male\_ancestor(X,Y) \leftarrow father(X,Z), female\_ancestor(Z,Y).$$
$$male\_ancestor(X,Y) \leftarrow father(X,Z), male\_ancestor(Z,Y).$$
$$female\_ancestor(X,Y) \leftarrow$$
$$female\_ancestor(X,Z), male\_ancestor(Z,Y).$$

5.4.2 Inductive Data Engineering

The term inductive data engineering (IDE) is used to denote the interactive restructuring of databases by means of induction (Flach 1993). The main idea is to use induction to determine integrity constraints that are valid (or almost valid) in a database and then use the constraints to decompose (restructure) the database. The integrity constraints used in the IDE system INDEX (Flach 1993) include functional and multivalued dependencies.

An unstructured database typically contains redundant information. INDEX decomposes such a relation into more primitive relations by using attribute dependencies. It defines the original relation as an intensional relation in terms of the primitive relations. In ILP terms, INDEX invents new predicates. It interacts with the user during the restructuring process, e.g., when choosing which dependency to decompose the relation upon.

If a dependency is satisfied on a given relation, it suggests a horizontal decomposition, where the original relation is reconstructed by a join operation. If a dependency is not satisfied, but is considered interesting (e.g., it is almost satisfied), it may suggest a vertical decomposition, where the original relation is reconstructed by a set union operation. The dependency is typically satisfied in the relations created by vertical decomposition, thus suggesting further horizontal decompositions.

To illustrate the IDE process, consider a train schedule represented by a single unstructured relation

$$train(Direction, Hour, Minutes, FirstStop)$$

(Flach 1993). The multivalued dependency $[] \rightarrow\rightarrow Hour$ (trains run every hour) is almost satisfied, suggesting a vertical decomposition. The relation *train* is then split into two relations: *regular_train* (trains that run every hour) and *irregular_train*: $train(A, B, C, D) \leftarrow regular\_train(A, B, C, D) \vee irregular\_train(A, B, C, D)$. The dependency $[] \rightarrow\rightarrow Hour$ is satisfied for the relation *regular_train*. Consequently, we can perform a horizontal decomposition: $regular\_train(D, H, M, FS) \leftarrow hour(H), reg\_train(D, M, FS)$.

5.4.3 Clausal Discovery

The ILP setting described in Section 5.2, including single and multiple predicate learning, interactive and empirical ILP, is the normal ILP setting (Muggleton and De Raedt 1994). Classification rules are induced in the normal ILP setting; if the induced rules are sufficiently accurate, they may replace the examples; namely, the examples can be derived from the rules. A radically different approach is taken in the nonmonotonic ILP setting: given a database, a set of constraints is to be found that hold for the database. As the constraints can have the form of first-order clauses, the approach is known as clausal discovery (De Raedt and Bruynooghe 1993).

To illustrate clausal discovery in the nonmonotonic ILP setting, consider a database containing facts about family relations: $mother(X, Y)$, $father(X, Y)$, and $parent(X, Y)$. The following integrity constraints, among others, have been discovered by the ILP system CLAUDIEN (De Raedt and Bruynooghe 1993):

$$mother(X, Y) \vee father(X, Y) \leftarrow parent(X, Y)$$

$$parent(X,Y) \leftarrow father(X,Y)$$
$$\leftarrow mother(X,Y), father(X,Y)$$
$$\leftarrow parent(X,X)$$

The third and fourth constraint state that it is impossible that X is at the same time both the mother and the father of Y, and that it is impossible for a person to be his own parent. Unlike in the normal ILP setting, if we throw away the database, the learned theory may not be able to reconstruct it.

CLAUDIEN searches a refinement graph for full clauses. The language of the clauses is specified through a declarative bias specification language, thus focusing the search to relevant clauses. Potential applications of CLAUDIEN are described in (Dehaspe et al. 1994), including the discovery of attribute dependencies and integrity constraints. As the nonmonotonic setting allows for more efficient induction than the normal setting (De Raedt and Džeroski 1994), we are likely to see KDD applications of clausal discovery. Section 5.5 describes the application of CLAUDIEN to discovering rules for biological classification of river water quality.

5.5 KDD Applications of ILP

Several ILP applications have emerged recently. They vary considerably in the degree of sophistication and significance with respect to the area of application. Most of these are laboratory demonstrations (some of them using real data) rather than fielded real-world applications. Surveys of ILP applications can be found in (Bratko and King 1994; Bratko and Džeroski 1995), and (Lavrač and Džeroski 1994). Lavrač and Džeroski (1994) identify knowledge discovery in databases as one of the main application areas of ILP.

Applications of ILP to different problems of knowledge discovery in databases include learning rules for early diagnosis of rheumatic diseases from a medical database (Lavrač et al. 1993), and biological classification of river water quality from an environmental database (Džeroski et al. 1994). ILP has also been applied to several tasks of knowledge discovery in biomolecular databases, where the results achieved have come close to practical significance. These applications are: modeling the structure-activity relationships for two series of drugs (King et al. 1992;

Hirst et al. 1995a; Hirst et al. 1995b), prediction of the local secondary structure (shape) of a protein from its amino-acid sequence (Muggleton et al. 1992), and prediction of the mutagenicity of compounds using only their chemical structure (Srinivasan et al. 1994). In all three cases, ILP approaches have generated rules that perform as well or better than conventional approaches and provide insight into the relevant stereochemistry.

In the remainder of this section, we give a summary of the applications of ILP to biological classification of river water quality (Džeroski et al. 1994), and prediction of the secondary structure of a protein from its amino-acid sequence (Muggleton et al. 1992). While the datasets in these applications are not tremendously large by today's standards, they nonetheless represent real-world datasets used in specific scientific studies.

5.5.1 Biological Classification of River Water Quality

Increasing importance is placed on the use of riverine ecology as a means of monitoring and classifying river quality, both in terms of water quality and its broader environmental quality (Ruck et al. 1993). The various biological flora and fauna, such as attached algae, macrophytes and benthic (or river bed) macro-invertebrates, are seen as continuous monitors of the rivers' "health," and field data on these are used to classify the river. In the case of water quality monitoring and classification, the biological methods are used to complement the more traditional chemical methods. An introduction to biological monitoring theory and techniques can be found in (De Pauw and Hawkes 1993).

At present, the most suitable single group for monitoring purposes is considered to be the benthic macro-invertebrates. These animals form part of the community associated with the river bed and are relatively immobile. They are present in all rivers, except in cases of extreme pollution, and cover a range of life modes and trophic levels. Taxonomic classification of these organisms to family or genus level is not too demanding, and qualitative identification is often carried out at the river bank. Also the different species are known to have different sensitivities to pollutants, thus the structure of the benthic macro-invertebrate community is affected by both degradable organic matter (sewage) and toxic pollutants (pesticides and heavy metals).

The task addressed here is to interpret benthic samples of macro-

invertebrates in water quality terms. In other words, given are samples of the river beds at different sites and their classification into one of five quality classes. The task is to learn general rules that will be able to classify new samples (Džeroski et al. 1994). The study used 292 field samples of benthic communities collected from British Midlands rivers, classified by an acknowledged expert river ecologist (H. A. Hawkes). The samples come from the database maintained by the National River Authority of the United Kingdom, where the results of monitoring the environmental quality of British rivers are stored.

Two ILP systems were applied to the problem of classification of biological samples. GOLEM (Muggleton and Feng 1990) works in the normal ILP setting, where the task is to find classification rules that explain the training examples, while CLAUDIEN (De Raedt and Bruynooghe 1993) works in the nonmonotonic ILP setting, where the task is to find valid rules that are confirmed by the training examples. A propositional learning system, CN2 (Clark and Boswell 1991; Džeroski et al. 1993) was also applied to learn classification rules.

For the normal ILP setting, the problem was formulated as follows: for each class a separate ILP problem was created, where the positive examples are the samples classified in that class, and all the other samples are negative examples. Thus, the target predicates were $b1a(X)$, $b1b(X)$, $b2(X)$, $b3(X)$, and $b4(X)$. The background knowledge consisted of eighty predicates of the form $family(X, A)$, each denoting that $family$ is present in sample X at abundance level A. (In several cases, identification was carried to levels other than family, e.g., species or genera level. For simplicity, we will use the term family throughout, regardless of the taxonomic identification level.) Predicates of this kind include $tipulidae(X, A)$, $asellidae(X, A)$, etc. In addition, the background predicate $greater\_than(A, B)$ was available, stating that abundance level A is greater than abundance level B.

The default settings of GOLEM were used, except for the fact that rules were allowed to cover up to five negative examples. GOLEM produced three rules for class B1a, fourteen rules for B1b, sixteen for B2, two for B3, and none for B4. For example, the rule $b1a(X) \leftarrow leuctridae(X, A)$ states that a sample belongs to the best water quality class if Leuctridae are present. This rule covers forty-three positive and four negative examples, and agrees with expert knowledge; the family Leuctridae is an indicator of good water quality. Another good

rule is the following: $b1b(X) \leftarrow ancylidae(X, A), gammaridae(X, B), hydropsychidae(X, C), rhyacophilidae(X, D), greater\_than(B, A), greater\_than(B, D)$. Gammaridae in abundance is a good indicator of class B1b, along with the other families present.

Out of the thirty-five rules, twenty-five are considered good or acceptable by the expert, but some of them are judged to be too specific. Note that GOLEM cannot use the absence of particular families in the rules within the representation adopted here. Together with GOLEM's strategy of looking for the most specific rules consistent with the examples, this may have influenced the generality (specificity) of the generated rules. This may also be the reason that no rules for class B4 were induced. Namely, the absence of most families is characteristic for this class.

Unlike GOLEM, CLAUDIEN (De Raedt and Bruynooghe 1993) checks all possible rules, within a specified language, for consistency with the given examples. Only tests of the presence of different families were allowed in the antecedents (no absence tests and no abundance level tests), while one or more quality classes were allowed in the consequents. Rules were required to cover at least thirty examples.

Altogether, seventy-nine rules were generated. Of these, twenty-eight involved the presence of a single family. These rules in fact specify the range of quality classes in which a certain family is present. The rule $b1a(X) \vee b1b(X) \leftarrow perlodidae(X, A)$ specifies that Perlodidae are found in good quality water (classes B1a and B1b). If Rhyacophilidae are also present, then the water is almost certainly of class B1a: $b1a(X) \leftarrow perlodidae(X, A), rhyacophilidae(X, B)$. Both rules are judged by the expert to be good.

Only nine rules had a single class (B1a) in the conclusion. The others had a number of possible classes in the conclusion, which was considered natural and understandable by the expert. This indicates that class B1a is easy to characterize in terms of the families present, while for the other classes references to the abundance levels or absence of certain families is required in order to find significant rules (that cover at least thirty samples). Overall, two thirds of the rules were considered to be good. The rest were not outright wrong, but still unacceptable to the expert for a number of reasons, such as mentioning families which are irrelevant (according to the expert opinion) for a particular water quality class.

In summary, the rules induced by the ILP systems GOLEM and CLAUDIEN were found to be consistent with the expert knowledge to a great

extent. Compared to other methods for automating the classification process, such as neural networks, ILP produces symbolic rules that can be used as a knowledge base for an expert system. The rules generated by CLAUDIEN were judged to be the most intuitive and promising. This is due to the coverage of more than one class by a single rule; this is important as the classification problem is based on a discretization of a continuous space. The fact that a single family is used in some rules also contributes to the overall intuitiveness of the CLAUDIEN rules.

Rules generated by GOLEM were considered too specific. The CN2 rules were quite general, but relied too heavily on the absence of certain families. According to the expert, rules should rely on the presence rather than absence of families as the latter may be due to reasons other than water quality (e.g., seasonal variations). However, the absence of some families seems to be necessary for rules classifying into poorer water quality classes.

The best direction for further work seems the selective use of absence information, which can be easily achieved using the declarative bias facilities of CLAUDIEN, together with the use of diversity information. The performance of the rules obtained in this way will be tested on an independent data set and compared to the corresponding performance of neural networks and Bayesian systems (Ruck et al. 1993; Walley et al. 1992). If their performance so merits the rules will then be used as a knowledge base of an expert system for biological classification of river water quality.

5.5.2 Predicting Protein Secondary Structure

We now briefly review the problem of learning rules for predicting the secondary structure of proteins, to which GOLEM has been applied (Muggleton et al. 1992). We first describe the problem, then the relevant background knowledge, and finally the results.

A protein is basically a string of amino acids (or *residues*). Predicting the three-dimensional shape of proteins from their amino acid sequence is widely believed to be one of the hardest unsolved problems in molecular biology. It is also of considerable interest to the pharmaceutical industry since the shape of a protein generally determines its function.

The sequence of amino acids is called the *primary structure* of the protein. Spatially, the amino acids are arranged in different patterns (spirals, turns, flat sections, etc.). The three-dimensional spatial shape

Table 5.5
Rules for predicting α-helix secondary structure.

Level 0 rule

$$\begin{aligned} alpha0(A,B) \leftarrow\ & octf(D,E,F,G,B,H,I,J,K), \\ & position(A,D,Q), not\_p(Q), not\_k(Q), \\ & position(A,E,O), not\_aromatic(O), small\_or\_polar(O), \\ & position(A,F,R), \\ & position(A,G,P), not\_aromatic(P), \\ & position(A,B,C), very\_hydrophobic(C), not\_aromatic(C), \\ & position(A,H,M), large(M), not\_aromatic(M), \\ & position(A,I,L), hydrophobic(L), \\ & position(A,K,N), large(N), ltv(N,R). \end{aligned}$$

Level 1 rule

$$\begin{aligned} alpha1(A,B) \leftarrow\ & octf(D,E,F,G,B,H,I,J,K), \\ & alpha0(A,F), alpha0(A,G). \end{aligned}$$

Level 2 rule

$$\begin{aligned} alpha2(A,B) \leftarrow\ & octf(C,D,E,F,B,G,H,I,J), \\ & alpha1(A,B), alpha1(A,G), alpha1(A,H). \end{aligned}$$

of a protein is called the *secondary structure*. When trying to predict the shape (*secondary structure*) it is easier to consider only one particular shape (pattern), instead of the multitude of possibilities; the α-helix (spiral) shape was considered by Muggleton et al. (1992).

The target relation $alpha(Protein, Position)$ specifies that the residue at position *Position* in protein *Protein* belongs to an α-helix. Negative examples state all residue positions in particular proteins which are not in an α-helix secondary structure. For instance, the positive example $alpha(1HMQ, 104)$ means that the residue at position 104 in protein 1HMQ (hemerythrin met) is in an α-helix. The negative example $alpha(1HMQ, 105)$ states that the residue at position 105 in the same protein is not in an α-helix (it actually belongs to a β-coil secondary structure).

To learn rules to identify whether a position in a protein is in an α-helix, Muggleton et al. (1992) use the following information as background knowledge.

- The relation $position(A, B, C)$ states that the residue of protein

A at position B is the amino-acid C. Each residue can be any one of the 20 possible amino acids and is denoted by a lower case character. For example, the fact $position(1HMQ, 111, g)$ says that the residue at position 111 in protein $1HMQ$ is glycine (using the standard one-character amino acid coding).

- The following arithmetic relations allow indexing of the protein sequence, relative to the residue for which secondary structure is being predicted. They have to be provided explicitly, as GOLEM has no built-in arithmetics. The relation $octf(A, B, C, D, E, F, G, H, I)$ specifies nine adjacent positions in a protein, i.e., it says that positions A to I occur in sequence. One fact in this relation may be, for instance, $octf(19, 20, 21, 22, 23, 24, 25, 26, 27)$. The $alpha\_triplet(A, B, C)$, $alpha\_pair(A, B)$ and $alpha\_pair4(A, B)$ relations allow a similar kind of indexing, which specifies residues that are likely to appear on the same face of an α-helix. They might be defined declaratively by the following three facts: $alpha\_triplet(n, n+1, n+4)$, $alpha\_pair(n, n+3)$, $alpha\_pair4(n, n+4)$.
- Physical and chemical properties of individual residues are described by unary predicates. These properties include hydrophobicity, hydrophilicity, charge, size, polarity, whether a residue is aliphatic or aromatic, whether it is a hydrogen donor or acceptor, etc. Sizes, hydrophobicities, polarities, etc., are represented by constants, such as $polar0$ and $hydro0$. The use of different constant names for polarity zero $polar0$ and hydrophobicity zero $hydro0$ is necessary to prevent unjustified generalizations in GOLEM (these could be prevented by explicit statement of types instead). Relations between the constants, such as $less\_than(polar0, polar1)$, are also provided as background knowledge.

Let us now proceed with a very short description of the experimental setup and the results (Muggleton et al. 1992). Sixteen proteins from the Brookhaven database (Bernstein et al. 1977) were used, twelve for training and four for testing. Without addressing in detail how GOLEM was run to induce rules from the training data, let us mention that each of the induced rules was allowed to misclassify up to 10 instances and some subjective statistical criteria were used to judge its significance

before allowing it into the set of rules used for classification. The induced rules covered around 60% of the instances.

To improve the coverage of these preliminary rules, the learning process was iterated. The predicted secondary structure positions found using the initial rules (called level 0 rules) were added to the background information. GOLEM was then run again to produce new (level 1) rules. This was necessary as the level 0 predictions were speckled, i.e., only short α-helix sequences were predicted. The level 1 rules in effect filtered the speckled predictions and joined together short sequences of α-helix predictions. The learning process was iterated once more with level 1 predictions added to the background knowledge and level 2 rules were induced. Finally, the symmetric variants of the rules induced at level 1 and level 2 were added to the rule set (e.g., $alpha1(A,B) \leftarrow octf(D,E,F,G,B,H,I,J,K)$, $alpha0(A,F)$, $alpha0(A,G)$): this was suggested by a domain expert that inspected the rules.

Applying GOLEM to the training set produced twenty-one level 0 rules, five symmetric level 1 rules and two symmetric level 2 rules. For illustration, Table 5.5 gives one rule from each level. The induced rules achieved accuracies of 78% on the training and 81% on the testing set. For comparison, the best previously reported result is 76%, achieved by using a neural network approach (Kneller et al. 1990). The rules induced by GOLEM also have the advantage over the neural network method of being more understandable. The propositional learner PROMIS (King and Sternberg 1990) achieved 73% accuracy using the same data set as GOLEM. However, as the representation power of PROMIS is limited, it was not able to find some of the important relationships between residues that GOLEM found to be involved in α-helix formation.

5.6 Discussion

We have given a brief introduction to inductive logic programming, as well as the basic techniques currently in use in ILP: relative least general generalization, inverse resolution, top-down search of refinement graphs, the use of rule models, and the transformation of ILP problems to propositional form. We have also outlined some recent developments in ILP relevant to KDD: multiple predicate learning, inductive data engineering and clausal discovery. Finally, we have described two applications

of ILP to KDD problems: biological classification of river water quality and predicting protein secondary structure. The remainder of the chapter gives a discussion on where to find out more about ILP, what ILP approaches to use for different kinds of KDD problems, and current hot topics in ILP research.

5.6.1 More Information on ILP

As the field of inductive logic programming is relatively young, most of the publications in the area appear in conference and workshop proceedings. There have been four international workshops on inductive logic programming, with proceedings published as technical reports (Muggleton 1991; Muggleton 1992b; Muggleton 1993; Wrobel 1994). The proceedings contain papers describing theoretical advances in inductive logic programming, practical implementations of ILP systems and applications of inductive logic programming.

A special section of the *SIGART Bulletin*, 5(1), January 1994, is devoted to ILP. It includes an introductory paper, as well as papers dealing with major topics within ILP, such as top-down search for logic program clauses and applications of ILP. The paper (Muggleton 1991b) introduced the term inductive logic programming. The most complete survey paper so far is (Muggleton and De Raedt 1994).

The collection of papers "Inductive Logic Programming" (Muggleton 1992a) contains many revised and extended papers from The Proceedings of the First International Workshop on Inductive Logic Programming (Muggleton 1991b), as well as many other papers relevant to the field. The book "Interactive Theory Revision: An Inductive Logic Programming Approach" (De Raedt 1992) gives a thorough account on interactive inductive logic programming, focusing on the ILP system CLINT. The book "Knowledge Acquisition and Machine Learning: Theory, Methods and Applications" (Morik et al. 1993) describes the ILP toolbox MOBAL and its applications.

An extensive overview of empirical ILP techniques, including descriptions of several empirical ILP systems is given in the book "Inductive Logic Programming: Techniques and Applications" (Lavrač and Džeroski 1994). The book gives a general introduction to the field, which we have summarized in Sections 5.2 and 5.3. It also gives a detailed account of handling imperfect (noisy) data in ILP, an important topic for practical applications. Part IV of the book concentrates on applications

Table 5.6
ILP resources on the internet.

Resource	How to reach it
ILPNET WWW	`http://www-ai.ijs.si/ilpnet.html`
ILP Newsletter	Send email to `ilpnet@ijs.si`
ILP Data sets	`ftp://ftp.gmd.de:/MachineLearning/ILP/public/data`
ML Repository	`ftp://ftp.ics.uci.edu:/pub/machine-learning-databases`
ILP Publications	`ftp://ftp.gmd.de:/MachineLearning/ILP/public/papers`
ILP References	`ftp://ftp.gmd.de:/MachineLearning/ILP/public/bib`
ILP Systems	`ftp://ftp.gmd.de:/MachineLearning/ILP/public/software`

of inductive logic programming. It describes several applications in detail and gives and overview of several other ILP applications, including protein secondary structure prediction as described in Section 5.5.2.

For the enthusiastic reader, Table 5.6 gives a list of ILP resources available on the Internet. These include ILP publications, ILP datasets, and ILP systems. The main sources are the ILPNET World-Wide-Web server in Ljubljana and the Machine Learning Archive in Bonn. The former provides general information on ILP and ILP research, as well as pointers to other sources of information on ILP.

5.6.2 When to Use What

Giving definite recommendations as to which of the variety of ILP approaches is suitable for which problem is a difficult task. This is not easy even for attribute-value systems which have been around for much longer and have reached maturity. Nevertheless, we will attempt to lay out a few guidelines.

If interested in learning classification rules for a given relation which would rely on other relations, one should use an ILP system operating within the normal ILP setting. In addition to revealing regularities in the data, such a set of classification rules can be used to replace the table it was derived from (provided it is accurate enough). If one is only interested in regularities in his database (possibly clausal integrity constraints) that need not be used for classification, the CLAUDIEN ILP system can be recommended. CLAUDIEN (De Raedt and Bruynooghe 1993) operates in the nonmonotonic ILP setting.

In the normal setting, interactive ILP systems, such as CLINT (De Raedt 1992), should be used when interaction with the user is desired or the database studied is at an early stage of construction. However,

empirical ILP systems are presently more appropriate for use in KDD, especially in the context of a database that is past the initial stages of development.

MPL (De Raedt et al. 1993) can be recommended for learning classification type logical definitions for several mutually dependent predicates. Other ILP systems, e.g., MOBAL (Morik et al. 1993), can also be used for this task. The use of MOBAL is especially recommended (both for single and multiple predicate learning) if the user has knowledge of the form of patterns to be discovered: this knowledge can be provided to MOBAL in the form of rule models.

Regarding the other empirical ILP systems for single predicate learning, the following recommendations can be made. If constrained or determinate clauses suffice for the application, LINUS and DINUS (Lavrač and Džeroski 1994) are a good choice. They rely on attribute-value systems to do the learning and are thus capable of handling noisy data and real-valued attributes. Most ILP systems do not have noise-handling capabilities. However, there are several exceptions, e.g., FOSSIL (Furnkranz 1994) and MILP (Kovačič 1994). Also, FOIL (Quinlan 1990) and GOLEM (Muggleton and Feng 1990) have some rudimentary noise-handling capabilities.

Top-down systems, such as FOIL, are better for finding simpler patterns, even in the presence of noise (in this case use mFOIL (Džeroski and Bratko 1992)). Bottom-up systems, such as GOLEM, are better at finding complicated patterns from data without imperfections, because they start with the most specific hypothesis in the hypothesis language that explains the examples. To conclude this recommendation section, we recommend that the reader gain access to the ILP systems from the Machine Learning Archive in Bonn (see Table 5.6) and try them out on their favorite KDD problem.

5.6.3 Hot Topics in ILP Research

Having started from a sound theoretical basis, current ILP research is increasingly turning to topics that are relevant for practical applications. One such topic is the computational complexity of ILP algorithms. Although theoretic by nature, this topic is of immediate practical interest: provably efficient ILP algorithms are obviously desirable. Research in this area has identified several classes of patterns (logic programs) that can be efficiently induced by current ILP systems, e.g., determinate logic

programs (Džeroski et al. 1992). It has also shown that the nonmonotonic ILP setting allows for more efficient clausal discovery than the normal ILP setting (De Raedt and Džeroski 1994).

The study of the computational complexity of ILP problems/algorithms has shown that current ILP algorithms would scale relatively well with the increasing number of examples or facts in the background knowledge database. However, they would not scale well with the number of arguments of the predicates (relations) involved, and in some cases with the complexity of the patterns searched. There are also indications that the use of declarative bias can help to resolve these problems.

Declarative bias is an active area of ILP research. Formalisms for specifying declarative bias include rule models (Morik et al. 1993), clause sets (Bergadano and Gunetti 1994), and antecedent description grammars (Cohen 1994). The bias specification mechanism of (Ade et al. 1995) is a generalization of these approaches and is used in the ILP system CLAUDIEN (De Raedt and Bruynooghe 1993).

Finally, handling imperfect (noisy) data and numerical constraints in ILP are also among the main topics of ILP research. This includes upgrading propositional techniques to the first-order case (Lavrač and Džeroski 1994), as well as using MDL-based heuristics (Kovačič 1994) for noise handling. Relational regression is used (Džeroski et al. 1995) to learn numerical constraints in the normal setting, and a similar regression-based approach can be used in the nonmonotonic setting.

5.7 Conclusion

ILP is concerned with the induction of first-order rules in the form of clausal theories or logic programs. The induction process happens in the context of a deductive database, i.e., a set of intensional and extensional relations. ILP can thus be used for discovering patterns that involve several relations. The expressive power of the pattern language is a strong motivation for the use of ILP in a KDD context. While efficiency and scaling problems may be expected when using ILP for KDD, recent ILP systems provide powerful tools to cope with these problems, e.g., facilities for specifying a strong declarative bias. Handling imperfect data and real-valued attributes is also increasingly present in ILP systems. In their efforts to prove the practical value of their systems,

ILP researchers are sure to address KDD applications: we hope that their work will be complemented with work by KDD practitioners, thus helping to discover new and useful knowledge.

Acknowledgments

This chapter is based on an invited talk at the AAAI'93 Workshop on Knowledge Discovery in Databases. Section 5.2, Section 5.3, and Section 5.5.2, are based on Chapter 2, Chapter 3, and Chapter 14.1 of the book "Inductive Logic Programming: Techniques and Applications" by Nada Lavrač and Sašo Džeroski, published by Ellis Horwood in 1994. The author acknowledges the support of the Slovenian Ministry of Science and Technology and the ESPRIT III Project No. 6020 "Inductive Logic Programming."

References

Adé, H.; De Raedt, L.; and Bruynooghe, M. 1995. Declarative Bias for Bottom-up ILP Systems. *Machine Learning,* 20: 119–154.

Bergadano, F.; and Gunetti, D. 1994. Learning Clauses by Tracing Derivations. In Proceedings of the Fourth International Workshop on Inductive Logic Programming, 11-30. Bonn, Germany: GMD.

Bernstein, F.; Koetzle, T.; Williams, G.; Meyer, E.; Brice, M.; Rodgers, J.; Kennard, O.; Shimanouchi, T.; and Tasumi, M. 1977. The Protein Data Bank: A Computer-based Archival File for Macromolecular Structures. *Journal of Molecular Biology,* 112: 535–542.

Bratko, I.; and Džeroski, S. 1995. Engineering Applications of Inductive Logic Programming. *New Generation Computing (Special Issue on Inductive Logic Programming),* 13: 313–333.

Bratko, I.; and King, R. 1994. Applications of Inductive Logic Programming. *SIGART Bulletin (Special Issue on Inductive Logic Programming),* 5(1): 43–49.

Buntine, W. 1988. Generalized Subsumption and Its Applications to Induction and Redundancy. *Artificial Intelligence,* 36(2): 149–176.

Cestnik, B.; Kononenko, I.; and Bratko, I. 1987. ASSISTANT 86: A Knowledge Elicitation Tool for Sophisticated Users. In *Progress in Machine Learning,* eds. I. Bratko and N. Lavrač, 31–45. Wilmslow, UK: Sigma Press.

Clark, P.; and Boswell, R. 1991. Rule Induction with CN2: Some Recent Improvements. In Proceedings of the Fifth European Working Session on Learning, 151–163. Berlin: Springer-Verlag.

Clark, P.; and Niblett, T. 1989. The CN2 Induction Algorithm. *Machine Learning,* 3(4): 261–283.

Cohen, W. 1994. Grammatically Biased Learning: Learning Logic Programs Using an Explicit Antecedent Description Language. *Artificial Intelligence,* 68: 303–366.

De Pauw, N.; and Hawkes, H. 1993. Biological Monitoring of River Water Quality. In Proceedings of the Freshwater Europe Symposium on River Water Quality Monitoring and Control, 87–111. Birmingham, UK: Aston University.

De Raedt, L. 1992. *Interactive Theory Revision: An Inductive Logic Programming Approach.* London: Academic Press.

De Raedt, L.; and Bruynooghe, M. 1993. A Theory of Clausal Discovery. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 1058–1063. San Mateo, Calif.: Morgan Kaufmann.

De Raedt, L.; and Džeroski, S. 1994. First Order jk-clausal Theories Are PAC-Learnable. *Artificial Intelligence,* 70: 375–392.

De Raedt, L.; Lavrač, N.; and Džeroski, S. 1993. Multiple Predicate Learning. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 1037–1042. San Mateo, Calif.: Morgan Kaufmann.

Dehaspe, L.; Van Laer, W.; and De Raedt, L. 1994. Applications of a Logical Discovery Engine. In Proceedings of the Fourth International Workshop on Inductive Logic Programming, 291–304. Bonn, Germany: GMD.

Džeroski, S.; and Bratko, I. 1992. Handling Noise in Inductive Logic Programming. In Proceedings of the Second International Workshop on Inductive Logic Programming. Tokyo: ICOT.

Džeroski, S.; Todorovski, L.; and Urbančic, T. 1995. Handling Real Numbers in ILP: A Step Towards Successful Behavioral Cloning. In Proceedings of the Eighth European Conference on Machine Learning, 283–286. Berlin: Springer-Verlag.

Džeroski, S.; Dehaspe, L.; Ruck, B.; and Walley, W. 1994. Classification of River Water Quality Using Machine Learning. In Proceedings of the Fifth International Conference on the Development and Application of Computer Techniques to Environmental Studies, volume I, 129–137. Southampton, UK: Computational Mechanics Publications.

Džeroski, S.; Cestnik, B.; and Petrovski, I. 1993. Using the m-Estimate in Rule Induction. *Journal of Computing and Information Technology,* 1(1): 37–46.

Džeroski, S.; Muggleton, S.; and Russell, S. 1992. PAC-Learnability of Determinate Logic Programs. In Proceedings of the Fifth ACM Workshop on Computational Learning Theory, 128–135. New York: ACM Press.

Flach, P. 1993. Predicate Invention in Inductive Data Engineering. In Proceedings of the Sixth European Conference on Machine Learning, 83–94. Berlin: Springer-Verlag.

Frawley, W.; Piatetsky-Shapiro, G.; and Matheus, C. 1991. Knowledge Discovery in Databases: An Overview. In *Knowledge Discovery in Databases,* eds. G. Piatetsky-Shapiro and W. Frawley, 1–27. Menlo Park, Calif.: The AAAI Press.

Furnkranz, J. 1994. FOSSIL: A Robust Relational Learner. In Proceedings of the Seventh European Conference on Machine Learning, 122–137. Berlin: Springer-Verlag.

Hirst, J.; King, R.; and Sternberg, M. 1995a. Quantitative Structure-Activity Relationships: Neural Networks and Inductive Logic Programming Compared Against Statistical Methods. I. The Inhibition of Dihydrofolate Reductase by Pyrimidines. *Journal of Medical Chemistry.* Forthcoming.

Hirst, J.; King, R.; and Sternberg, M. 1995b. Quantitative Structure-Activity Relationships: Neural Networks and Inductive Logic Programming Compared Against Statistical Methods. II. The Inhibition of Dihydrofolate Reductase by Triazines. *Journal of Medical Chemistry.* Forthcoming.

King, R.; Muggleton, S.; Lewis, R.; and Sternberg, M. 1992. Drug Design by Machine Learning: The Use of Inductive Logic Programming to Model the Structure-Activity Relationships of Trimethoprim Analogues Binding to Dihydrofolate Reductase. *Proceedings of the National Academy of Sciences,* 89: 11322–11326.

King, R.; and Sternberg, M. 1990. Machine Learning Approach for the Prediction of Protein Secondary Structure. *Journal of Molecular Biology,* 216: 441–457.

Kneller, D.; Cohen, F.; and Langridge, R. 1990. Improvements in Protein Secondary Structure Prediction by an Enhanced Neural Network. *Journal of Molecular Biology,* 214: 171–182.

Kovačič, M. 1994. MILP-a Stochastic Approach to Inductive Logic Programming. In Proceedings of the Fourth International Workshop on Inductive Logic Programming, 123–138. Bonn, Germany: GMD.

Lavrač, N.; and Džeroski, S. 1994. *Inductive Logic Programming: Techniques and Applications.* Chichester, UK: Ellis Horwood.

Lavrač, N.; Džeroski, S.; Pirnat, V.; and Križman, V. 1993. The Utility of Background Knowledge in Learning Medical Diagnostic Rules. *Applied Artificial Intelligence,* 7: 273–293.

Lavrač, N.; Džeroski, S.; and Grobelnik, M. 1991. Learning Nonrecursive Definitions of Relations with LINUS. In Proceedings of the Fifth European Working Session on Learning, 265–281. Berlin: Springer-Verlag.

Lloyd, J. 1987. *Foundations of Logic Programming,* Second Edition. Berlin: Springer-Verlag.

Morik, K.; Wrobel, S.; Kietz, J.-U.; and W. Emde 1993. *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications.* London: Academic Press.

Muggleton, S., ed. 1993. Proceedings of the Third International Workshop on Inductive Logic Programming. Ljubljana, Slovenia: Jožef Stefan Institute.

Muggleton, S., ed. 1992a. *Inductive Logic Programming.* London: Academic Press.

Muggleton, S., ed. 1992b. Proceedings of the Second International Workshop on Inductive Logic Programming. Tokyo: ICOT.

Muggleton, S., ed. 1991a. Proceedings of the International Workshop on Inductive Logic Programming. Porto, Portugal: University of Porto.

Muggleton, S. 1991b. Inductive Logic Programming. *New Generation Computing,* 8(4): 295–318.

Muggleton, S.; and Buntine, W. 1988. Machine Invention of First-Order Predicates by Inverting Resolution. In Proceedings of the Fifth International Conference on Machine Learning, 339–352. San Mateo, Calif.: Morgan Kaufmann.

Muggleton, S.; and De Raedt, L. 1994. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming,* 19/20: 629–679.

Muggleton, S.; and Feng, C. 1990. Efficient Induction of Logic Programs. In Proceedings of the First Conference on Algorithmic Learning Theory, 368–381. Tokyo: Ohmsha.

Muggleton, S.; King, R.; and Sternberg, M. 1992. Protein Secondary Structure Prediction Using Logic. *Protein Engineering,* 5: 647–657.

Muggleton, S. 1995. Inverse Entailment and Progol. *New Generation Computing (Special Issue on Inductive Logic Programming),* 13: 245–286.

Pazzani, M.; and Kibler, D. 1992. The Utility of Knowledge in Inductive Learning. *Machine Learning,* 9(1): 57–94.

Plotkin, G. 1969. A Note on Inductive Generalization. In *Machine Intelligence 5,* eds. B. Meltzer and D. Michie, 153–163. Edinburgh, UK: Edinburgh University Press.

Quinlan, J. 1993. *C4.5: Programs for Machine Learning.* San Mateo, Calif.: Morgan Kaufmann.

Quinlan, J. 1990. Learning Logical Definitions from Relations. *Machine Learning,* 5(3): 239–266.

Robinson, J. 1965. A Machine-oriented Logic Based on the Resolution Principle. *Journal of the ACM,* 12(1): 23–41.

Ruck, B.; Walley, W.; and Hawkes, H. 1993. Biological Classification of River Water Quality Using Neural Networks. In Proceedings of the Eight International Conference on Artificial Intelligence in Engineering, 361–372. Amsterdam, The Netherlands: Elsevier.

Sammut, C.; and Banerji, R. 1986. Learning Concepts by Asking Questions. In *Machine Learning: An Artificial Intelligence Approach,* Volume II, 167–191. eds. R. Michalski, J. Carbonell, and T. Mitchell. San Mateo, Calif.: Morgan Kaufmann.

Shapiro, E. 1983. *Algorithmic Program Debugging.* Cambridge, Mass.: The MIT Press.

Siou, E. 1994. A Bidirectional Search for Clauses. In Proceedings of the Fourth International Workshop on Inductive Logic Programming, 365–376. Bonn, Germany: GMD.

Srinivasan, A.; Muggleton, S.: King, R.; and Sternberg, M. 1994. Mutagenesis: ILP Experiments in a Non-determinate Biological Domain. In Proceedings of the Fourth International Workshop on Inductive Logic Programming, 217–232. Bonn, Germany: GMD.

Ullman, J. 1988. *Principles of Database and Knowledge Base Systems,* Volume I. Rockville, Mass.: Computer Science Press.

Walley, W.; Boyd, M.; and Hawkes, H. 1992. An Expert System for the Biological Monitoring of River Pollution. In Proceedings of the Fourth International Conference on the Development and Application of Computer Techniques to Environmental Studies, 1030–1047. Amsterdam, The Netherlands: Elsevier.

Wrobel, S., ed. 1994. Proceedings of the Fourth International Workshop on Inductive Logic Programming. Bonn, Germany: GMD.