Part I

Extension of Inductive Logic Programming

Chapter 3

Inductive Learning of Logic Programs

3.1 Inductive Concept Learning

As a field of machine learning, when positive examples and negative examples are given, the goal of inductive concept learning is to construct concept representation that covers all of the positive examples and none of the negative examples. The term *cover* should be defined in each problem setting. Suppose that positive examples (+) and negative examples (-) are given on a two-dimensional field (Fig. 3.1).

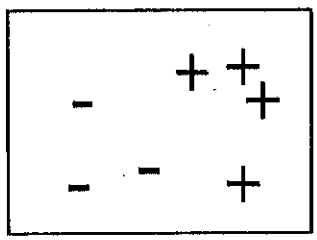

Figure 3.1: Sample Data

Suppose also that the concept representation is a rectangle. Examples are said to be *covered* iff the examples are inside the rectangle.

When all of the positive examples are covered, the concept is called *complete*; otherwise it is called *incomplete*. When none of the negative examples are covered, the concept is called *consistent*; otherwise it is called *inconsistent*. Consequently, there can be four types of coverage based on completeness and consistency (Fig 3.2).

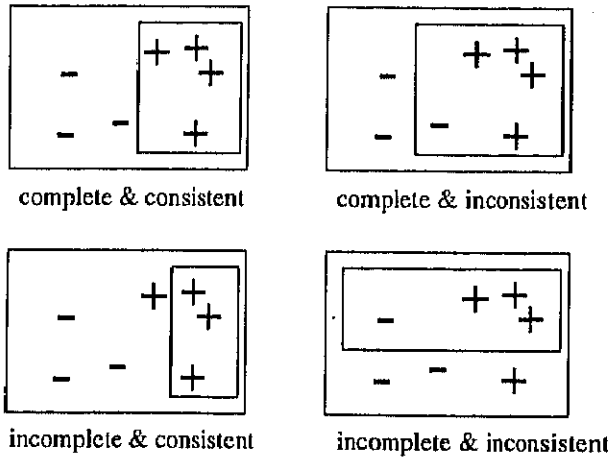

Figure 3.2: Four Types of Coverage

The goal of inductive concept learning is to find complete and consistent concept representation from examples. In a practical sense, since data usually have some noise, the goal is to find a concept representation that covers almost all of the positives but few of the negatives.

3.2 Inductive Logic Programming (ILP)

3.2.1 Framework

Inductive Logic Programming (ILP) [36] is a part of inductive concept learning, and the concept representation is definite clauses, in other word, logic programs [32]).

Formally, the framework of ILP is defined as follows. Let a signature $\Sigma = (\mathcal{F}, \mathcal{P}, \mathcal{C})$ where $\mathcal{F}$ is a finite set of function symbols, $\mathcal{P}$ is a finite

set of predicate symbols, $\mathcal{C}$ is a finite set of constant symbols, and an observation predicate $p \in \mathcal{P}$. Example language L_E, background knowledge language L_B, and hypothesis language L_H are defined as follows.

- L_E: the set of all ground atomic formulae whose predicate symbols is an observation predicate symbol p.
- L_B: the set of all ground unit clauses[1] .
- L_H: the set of all definite clauses whose heads are atomic formulae with p and whose bodies consist of literals with predicates symbols in background knowledge language L_B.

Let a finite set of positive examples be $E^+ \subseteq L_E$, a finite set of negative examples be $E^- \subseteq L_E$, and a finite set of background knowledge be $B \subseteq L_B$, where the following conditions are satisfied:

$$\forall e \in E^+ \; B \not\vdash e$$

$$\forall e \in E^- \; B \not\vdash e$$

ILP is defined as φ that satisfies the following conditions with respect to hypotheses H.

$$H \subseteq L_H \;\; s.t. \;\; H = \varphi(L_E, L_B, L_H, B, E^+, E^-).$$

$$B \cup H \textit{ is consistent.}$$

$$\forall e \in E^+ \; B \cup H \vdash e.$$

$$\forall e \in E^- \; B \cup H \not\vdash e.$$

As with inductive concept learning, when learning from real-world data with some noise, the last two conditions are relaxed to "for most of $e \in E^+$ and for most of $e \in E^-$". Various kinds of heuristic evaluation measures have been proposed to estimate the appropriateness of H (*c.f.*, [28]).

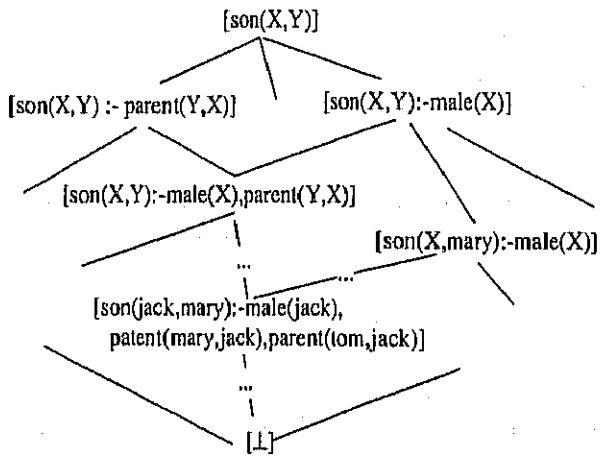

Figure 3.3: Sample Hypothesis Space

3.2.2 Hypothesis Space

In ILP, the hypothesis space is formed by the general-specific relation defined by ordering $\leq$ of clauses.

Let L_H be the set of all definite clauses whose heads are atomic formulae with p and whose bodies consist of literals with predicates symbols in background knowledge language L_B. A hypothesis space is the lattice on equivalence classes of the clauses in L_B, based on ordering $\leq$ of clauses defined in Section 2.2.

Example 4 *Let a signature* $\Sigma = (\mathcal{F}, \mathcal{P}, \mathcal{C})$, *where* $\mathcal{F} = \emptyset$, $\mathcal{P}$=$\{son^{(2)}$, $male^{(1)}$, $parent^{(2)}\}$, *and* $\mathcal{C}$=$\{mary,tom,jack\}$. *Let* $\mathcal{V} = \{X, Y\}$ *and the observation predicate symbol be* $son^{(2)}$. *The hypothesis space formed by ordering* $\leq$ *of clauses in* L_H *is shown in Fig 3.3.*

3.2.3 Search Bias

There are two methods to control hypothesis space in ILP: *search bias* and *language bias*. The search bias controls the search space by restrict-

[1] Some ILP frameworks allow a set of horn clauses.

ing the manner of searching. The top-down approach constructs hypotheses from general to specific by searching in the hypothesis space. A typical example of learners that employ the top-down approach is FOIL [46]. On the other hand, the bottom-up approach constructs hypothesis from specific to general by searching in the hypothesis space. That is, starting from positive examples, the hypothesis is gradually generalized. Typical examples of the learners that employ bottom-up approach are Golem [38] and Progol [40]. CHILL [65] employs a combination of the two approaches.

3.2.4 Language Bias

Example language L_E, background knowledge language L_B, and hypothesis language L_H determine the search space itself; language bias directly controls the size of part of the hypothesis space.

L_B provides a choice of whether the learning framework allows recursive definition in hypothesis clauses. That is, if observation predicate symbol p is included in L_B, H could be recursive definitions. Otherwise, recursive definitions are not allowed in H. In this thesis, L_B does not include p. Other restrictions to hypothesis language includes a limit on the number of literals in the body and variable depth of variables [38] in a hypothesis.

Let us consider restrictions on the hypothesis language introduced in [38].

Definition 65 (Ordered Horn Clauses) *Let $C = A{:-}B_1,...,B_n$ be a Horn clause. A Horn clause C is called an ordered Horn clause iff $B_1, ..., B_n$ are totally ordered by order $\succ$ as $B_1 \succ B_2 \succ ... \succ B_n$.*

Definition 66 (Determinate Terms) *[38]. Let K be a logic program, the examples E be a set of ground atoms, and $M(K)$ be the least Herbrand model of K. Let $A{:-}B_1,...,B_m,B_{m+1},...,B_n$ be an ordered Horn clause and t be a term in B_{m+1}. t is a determinate term w.r.t. B_{m+1} iff for every substitution θ such that $A\theta \in E$ and $\{B_1,...,B_m\}\theta \subseteq M(K)$, there is a unique atom $B_{m+1}\theta\sigma$ in $M(K)$.*

Intuitively, this states that a term t is determinate iff each of its variables that do not appear in preceding literals in the clause has only

one possible bindings given the bindings of its variables that appear in preceding literals [28].

Definition 67 (Variable Depth) *[28] Let $A :- B_1, ..., B_m, B_{m+1}, ..., B_n$ be an ordered Horn clause. Variables that appear in the head of the clause have depth zero. Let a variable V appear first in literal B_{m+1}. Let d be the maximum depth of the other variables in B_{m+1} that appear in the clause $A :- B_1, ..., B_m$. Then, the depth of variable V is $d + 1$.*

The number of literals in the body of the hypothesis, determinate terms and variable depth are widely used to make the hypothesis space tractable.

3.2.5 Previous ILP Systems

Previous ILP systems include FOIL [46], Golem [38], Progol [40], CHILL [65], mFOIL [28], LINUS [27], MOBAL [34], FOCL [41], CHAMP [24].

FOIL is a typical example of the top-down learners. FOIL learns from positive and negative examples[2] of function-free extended logic programs in the following form:

$$p(X_1, ..., X_n) :- L_1, ..., L_m,$$

where p is the predicate of examples and L_i are literals.

FOIL is such a covering algorithm that creates clauses one-by-one and removes covered positive examples. The learning process stops when all the positives are covered or constraints to hypothesis encoding length is violated. In the clause construction step, FOIL first creates the head $p(X_1, ..., X_n)$ and then specializes it by adding literals to the body. Literal selection is based on an entropy-based search heuristics called *weighted information gain* [28].

FOIL is followed by other ILP systems including mFOIL, FOCL, and CHAMP. mFOIL applied m-estimate [11] as a search heuristic to FOIL. FOCL combined FOIL and *explanation based learning (EBL)*. CHAMP incorporated the *predicate invention* algorithm DBC into a FOIL-like learning algorithm.

[2] If negative examples are not given, FOIL generates negatives based on the closed-world assumption.

Golem [38] and Progol [40] are typical examples of bottom-up learners. Golem uses Plotkin's rlgg [43, 44]. Plotkin gave a deterministic procedure that computes a least general generalization of terms and clauses with respect to ordering $\leq$. Plotkin's rlgg is an extension of the least general generalization (lgg).

As with FOIL, Golem is also a covering algorithms. A hypothesis clause is constructed as follows. Golem selects several pairs of positive examples e_1 and e_2 and computes rlggs $lgg(e_1 :- B_1, ..., B_n, e_2 :- B_1, ..., B_n)$, where $B_1, ..., B_n$ are the literals in background knowledge B. Then, it selects the rlgg that covers the greatest number of positive examples. Finally, it removes redundant literals from the body of selected clauses.

Progol uses an A*-like search and inverse-entailment. It develops a most specific clause, called $\perp_i$, and conducts an A*-like search of clauses that subsume $\perp_i$. CHILL employs a combination of top-down and bottom-up approaches by constructing the head of clauses in a bottom-up manner using the lgg and the body in a top-down manner similar to FOIL. MOBAL is a knowledge acquisition tool that learns a fixed form of clauses predefined by a *rule schema*.

LINUS takes a very different approach. LINUS transforms examples and background knowledge to attribute-value tuples. Then, it generates rules or a decision tree by using an attribute-value learner. Finally, LINUS transforms the results back to clauses. Table 3.1 shows a summary of comparison in previous ILP systems.

Above all, FOIL, Golem, Progol are the most well-known, high performance, and publicly available ILP systems that researchers in the ILP community use to compare performance of their ILP systems.

Table 3.1: Previous ILP Systems

system	creator	learning approach
FOIL	Quinlan	top-down
Golem	Muggleton et al.	bottom-up based on the rlgg
Progol	Muggleton	bottom-up based on inverse-entailment
CHILL	Zelle et al.	combination of bottom-up and top-down
mFOIL	Lavrač et al.	top-down with m-estimate
LINUS	Lavrač et al.	conversion to attribute-value learning
MOBAL	Morik et al.	rule schema based
FOCL	Pazzani et al.	combination of top-down and EBL
CHAMP	Kijsirikul et al.	top-down with predicate invention