I. Introduction

1. Logic Programming
2. Machine Learning
3. Inductive Learning
4. Inductive Logic Programming

Logic Programming Terminology

Constant: a symbol representing an object (e.g. `john`, `table`, `1`, `pi`, `true` etc.)

Variable: a symbol representing an unspecified object (e.g. `X`, `Human`, `Number`, etc.)

Function: a symbol representing a mapping from n objects to 1 object (e.g. `father`, `successor`, `[|]`). Constants may be viewed as functions with 0 arguments.

Term: a constant, a variable, or a function with n argument terms. (e.g. `father(john)`, `succ(succ(X))`, `[1,2,3]`).

Predicate: a symbol representing a relationship between n objects (e.g. `father`, `<`, `=`).

Atom(ic Formula): a predicate with n argument terms (e.g. `succ(1) > 1`, `father(X,father(X))`).

Logic Programming Terminology (Ctd.)

Literal: An atom (*positive* literal) or a negated atom (*negative* literal, e.g. $\neg$ `mother(X,father(X))`). Each literal evaluates to *true* or *false*.

Clause: A set (disjunction) of literals (e.g. $\{$`father(X,Y)` $\vee$ `mother(X,Y)` $\vee$ $\neg$ `parent(X,Y)`$\}$). Notational convention: `father(X,Y), mother(X,Y) :- parent(X,Y).` In this case positive literals are called *head* of the clause, negative literals are called *body* of the clause.

Horn Clause: A clause with at most one positive literal.

Definite Clause: A horn clause with exactly one positive literal (e.g. `mother(X,Y) :- parent(X,Y), female(X).`).

(Definite) Logic Program: a set of definite clauses

```
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

Procedural Semantics of PROLOG

Program Execution: PROLOG proves a query by looking for clauses with a head that *unifies* with the query. It then tries to prove each literal in the body of the clause (left to right).

Backtracking: When it cannot prove the body of a clause, PROLOG backtracks and tries the next clause with a matching head.

Closed World Assumption: PROLOG assumes that it knows everything, therefore if it cannot prove a fact it assumes it is *false*.

Cut (!): prevents backtracking beyond this point.

Negation as Failure: a special operator `not` (or `\+`) that evaluates to `true` when its argument cannot be proven.

```
not(X) :- X, !, fail.
not(X).
```

```
| ?- trace,quicksort([2,1,4,3],X).
 1  1  Call: quicksort([2,1,4,3],_82) ?
 2  2  Call: partition([1,4,3],2,_710,_711) ?
 3  3  Call: 1=<2 ?
 3  3  Exit: 1=<2 ?
 3  3  Call: partition([4,3],2,_933,_711) ?
 4  4  Call: 4=<2 ?
 4  4  Fail: 4=<2 ?
 4  4  Call: 4>2 ?
 4  4  Exit: 4>2 ?
 4  4  Call: partition([3],2,_933,_1537) ?
 5  5  Call: 3=<2 ?
 5  5  Fail: 3=<2 ?
 5  5  Call: 3>2 ?
 5  5  Exit: 3>2 ?
 5  5  Call: partition([],2,_933,_2140) ?
 5  5  Exit: partition([],2,[],[]) ?
 4  4  Exit: partition([3],2,[],[3]) ?
 3  3  Exit: partition([4,3],2,[],[4,3]) ?
 2  2  Exit: partition([1,4,3],2,[1],[4,3]) ?
 6  2  Call: quicksort([1],_703) ?
 7  3  Call: partition([],1,_3643,_3644) ?
 7  3  Exit: partition([],1,[],[]) ?
 8  3  Call: quicksort([],_3636) ?
 8  3  Exit: quicksort([],[]) ?
 9  3  Call: quicksort([],_3630) ?
 9  3  Exit: quicksort([],[]) ?
 10  3  Call: append([],[1],_703) ?
 10  3  Exit: append([],[1],[1]) ?
 6  2  Exit: quicksort([1],[1]) ?
 11  2  Call: quicksort([4,3],_697) ?
 12  3  Call: partition([3],4,_5514,_5515) ? s
 12  3  Exit: partition([3],4,[3],[]) ?
 14  3  Call: quicksort([3],_5507) ? s
 14  3  Exit: quicksort([3],[3]) ?
 19  3  Call: quicksort([],_5501) ?
 19  3  Exit: quicksort([],[]) ?
 20  3  Call: append([3],[4],_697) ? s
 20  3  Exit: append([3],[4],[3,4]) ?
 11  2  Exit: quicksort([4,3],[3,4]) ?
 22  2  Call: append([1],[2,3,4],_82) ?
 23  3  Call: append([],[2,3,4],_7647) ?
 23  3  Exit: append([],[2,3,4],[2,3,4]) ?
 22  2  Exit: append([1],[2,3,4],[1,2,3,4]) ?
 1  1  Exit: quicksort([2,1,4,3],[1,2,3,4]) ?

X = [1,2,3,4] ?

yes
{trace}
| ?-
```

Learning

...is hard to define.

> *"Lernen ist der Sammelname für Vorgänge, Prozesse oder nicht unmittelbar beobachtbare Veränderungen im Organismus, die durch Erfahrungen entstehen und zu Veränderungen des Verhaltens führen."* [Bergius,1971]

> *"Lernen bedeutet Veränderungen in der Wahrscheinlichkeit, mit der Verhaltensweisen in bestimmten Situationen auftreten."* [Hilgard,1973]

> *"Lernen ist eine Verhaltensänderung, die* nicht *durch Reifungsvorgänge, Verletzungen oder Erkrankungen, Ermüdungsprozesse oder durch Anlagen erklärt werden kann."* [Joerger,1976]

Learning

...in Artificial Intelligence

> *"Learning denotes changes in the system that [...] enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time."* [Simon,1983]

> *"Learning is making useful changes in our minds."* [Minsky,1985]

> *"Learning is constructing or modifying representations of what is being experienced."* [Michalski,1986]

Machine Learning Techniques

- decision tree learning
- conceptual clustering
- explanation-based learning
- case-based learning
- reinforcement learning
- temporal difference learning
- neural networks
- genetic algorithms

and ...

- *inductive logic programming*

In(tro)**duction**

Induction is the reasoning from observations to general principles that underly these observations.

Deductive reasoning can be formalized in *First Order Predicate Calculus* and has a sound and complete inference rule (*resolution*).

Problem: How can inductive reasoning be formalized?

ILP being based on first-order logic hopes to provide a suitable framework for studying induction.

Deductive Reasoning

$$T \quad \cup \quad B \quad \models \quad E$$

T		B		E
parent(X,Y) :- mother(X,Y). parent(X,Y) :- father(X,Y).	$\cup$	mother(penelope,victoria). mother(penelope,arthur). father(christopher,victoria). father(christopher,arthur).	$\models$	parent(penelope,victoria). parent(penelope,arthur). parent(christopher,victoria). parent(christopher,arthur).

Inductive Reasoning

$$E \quad \cup \quad B \quad \models \quad T$$

```
parent(penelope,victoria).
parent(penelope,arthur).
parent(christopher,victoria).
parent(christopher,arthur).
```

$\bigcup$

```
mother(penelope,victoria).
mother(penelope,arthur).
father(christopher,victoria).
father(christopher,arthur).
```

$\models$

```
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

Abductive Reasoning

$$E \quad \cup \quad T \quad \models \quad B$$

```
parent(penelope,victoria).
parent(penelope,arthur).
parent(christopher,victoria).
parent(christopher,arthur).
```

$\cup$

```
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

$\models$

```
mother(penelope,victoria).
mother(penelope,arthur).
father(christopher,victoria).
father(christopher,arthur).
```

Induction of classifiers

Most intensely studied task in Machine Learning.

Given:

- a set of training examples $E = \{e_i\}$
- a classification c_i for each example (often $\oplus$ and $\ominus$)

Find: A function h (*hypothesis*) such that

$$h(e_i) = c_i$$

for as many e_i as possible.

Often h should also predict the classification for examples e_i that have not been available at learning time, i.e. it should form a *generalization* of the training examples.

Learning as Search

Given:

- Example Description Language E
- Hypothesis Description Language H
- positive examples P
- negative examples N
- a matching predicate $\texttt{covers}(h,e)$

Find: A hypothesis that covers all positive (*completeness*) and no negative examples (*consistency*)

$$h \in H : \forall p \in P : \texttt{covers}(h, p) \land \forall n \in N : \neg\texttt{covers}(h, n).$$

Simple Example

Instance Space: Attribute-value representation

No.	*Attributes*			*Class*
	Shape	Color	Size	
1	triangle	black	big	$\oplus$
2	square	white	big	$\ominus$
3	triangle	black	small	$\oplus$
4	circle	white	small	$\ominus$

Hypothesis Space: conjunction of attribute-value tests.

Examples for correct (*complete* and *consistent*) theories:

$$(\text{Shape} = \textit{triangle}) \text{ or } <\textit{triangle}, ?, ?>$$

and

$$(\text{Shape} = \textit{triangle} \wedge \text{Color} = \textit{black}) \text{ or } <\textit{triangle}, \textit{black}, ?>$$

Another Example

Instance Space: 2 real-valued attributes (e.g. *temperature* and *blood pressure*)

Hypothesis Space: rectangles

Figure 18.14 from (Russell & Norvig, 1995)

Ordered Hypothesis Space

A hypothesis g is called *more general* than a hypothesis s if g covers a superset of the instances covered by s. g is a *generalization* of s. s is *more specific* than g. It is a *specialization* of g.

Example:
$<$*triangle*, ?, ?$>$ is more general than $<$*triangle*, *black*, ?$>$.

Using the *generality* relation the hypothesis space can (sometimes) be ordered into a lattice with a *most general hypothesis* (e.g. always predict $\oplus$) and a *most specific hypothesis* (e.g. the disjunction of all positive examples).

Example

Figure 18.10 from (Russell & Norvig, 1995)

x and y axis	real-valued attributes
$+$	positive examples
$-$	negative examples
$\square$	hypotheses of the type $(a \leq x \leq b) \wedge (c \leq y \leq d)$

Instance Space and Hypothesis Space

Here goes Tom Mitchell's slide with same title

by Tom Mitchell, 1995

A General Incremental Induction Algorithm

```
procedure INCREMENTAL-INDUCTION()

Hypothesis = always predict ⊕
PositiveExamples = NegativeExamples = ∅
Example = READNEXTEXAMPLE()
repeat
    if NEGATIVE(Example)
       if COVERS(Hypothesis,Example)
          ;; Hypothesis too general, i.e. inconsistent
          Hypothesis = SPECIALIZE(Hypothesis)
          ;; so that it no longer covers Example, but still covers all PositiveExamples
       NegativeExamples = NegativeExamples ∪ Example
    if POSITIVE(Example)
       if not COVERS(Hypothesis,Example)
          ;; Hypothesis too specific, i.e. incomplete
          Hypothesis = GENERALIZE(Hypothesis)
          ;; so that it covers Example, but still does not cover any NegativeExamples
       PositiveExamples = PositiveExamples ∪ Example
    Example = READNEXTEXAMPLE()
until Example = ∅
return(Hypothesis)
```

The Version Space

Figure 18.13 from (Russell & Norvig, 1995)

Assumptions

1. Training examples contain no errors.

   Example: If example 3 is an erroneously classified negative example, a correct hypothesis would be

   $$<?, black, big>$$

   which is not a generalization of the given positive examples.

2. It is possible to formulate a correct theory within H.

   Example: Add the positive example

   $$< circle, grey, ? >$$

   and a formulation of a correct theory would have to include *disjunction* or *negation*

   $$<?, black \vee grey, ?> \text{ or } <?, \neg white, ?>$$

   which are not part of the hypothesis space.

Assumptions

3. The generalizations and specializations of two objects must be finite and computable without resorting to checking the (possibly infinite) number of instances covered by them.

   Example 1: Allowing unrestricted disjunction and conjunction will produce the disjunction of the positive instances as the most specific theory and the conjunction of the negated negative instances as the most general theory.

   Example 2: Implication between clauses is undecidable.[a]

[a] Implication forms a natural generality criterion for general first-order clauses. If $c_1 \models c_2$ all instances derivable from c_2 are also derivable from c_1 ($c_2 \models c \Rightarrow c_1 \models c_2 \models c$), thus c_1 is more general than c_2.

Bias

"Bias refers to any criterion for choosing one generalization over another other than strict consistency with the observed training instances." *(Mitchell, 1980)*

Language Bias: The hypothesis language usually does not allow to define all possible subsets of the instances, i. e. all possible concepts.

Search Bias: Many search algorithms use heuristics that prefer certain types of hypothesis. An often used heuristic is a preference for simple hypothesis (*Ockham's Razor*).

Search Algorithms

Search Method:

- systematic search
  - depth-first search
  - breadth-first search
  - best-first search
- heuristic search
  - hill-climbing
  - beam search

Search Direction:

- top-down search (general-to-specific)
- bottom-up search (specific-to-general)
- bi-directional search

Hypothesis Spaces

- Neural Networks
- Decision Trees
- Production Rules
- Probabilistic Functions
- Propositional Classification Rules
- PROLOG Programs
- ...

Logic programs are the most expressive among these hypothesis languages. They are equivalent to Turing machines.

A Generic ILP Algorithm

procedure ILP(*Examples*)

$\textsc{Initialize}(\mathit{Theories}, \mathit{Examples})$
repeat
    $T = \textsc{Select}(\mathit{Theories}, \mathit{Examples})$
    $\{T_i\}_{i=1}^{n} = \textsc{Refine}(T, Examples)$
    $\mathit{Theories} = \textsc{Reduce}(\mathit{Theories} \cup \bigcup_{i=1}^{n} T_i, \mathit{Examples})$
until $\textsc{StoppingCriterion}(\mathit{Theories}, \mathit{Examples})$
return(*Theories*)

Procedures for a Generic ILP Algorithm

INITIALIZE: initialize a set of theories (e.g. $\mathit{Theories} = \{\mathtt{true}\}$ or $\mathit{Theories} = \mathit{Examples}$)

SELECT: select the most promising candidate theory

REFINE: apply *refinement operators* that generate new theories (*specialization*, *generalization*, ...).

REDUCE: discard unpromising theories

STOPPINGCRITERION: determine whether the current set of theories is already good enough (e.g. when it contains a *complete* and *consistent* theory)

SELECT and REDUCE together implement the search strategy.

e.g. *hill-climbing*: REDUCE = only keep the best theory.