

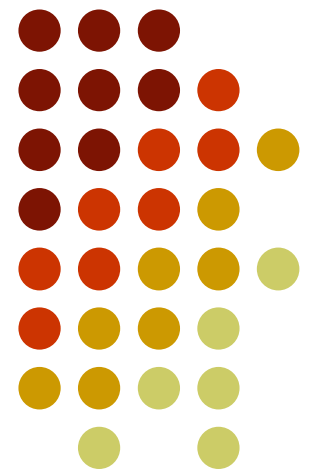
Inductive Logic Programming

Ganesh Ramakrishnan

Adjunct Professor, CSE Dept

&

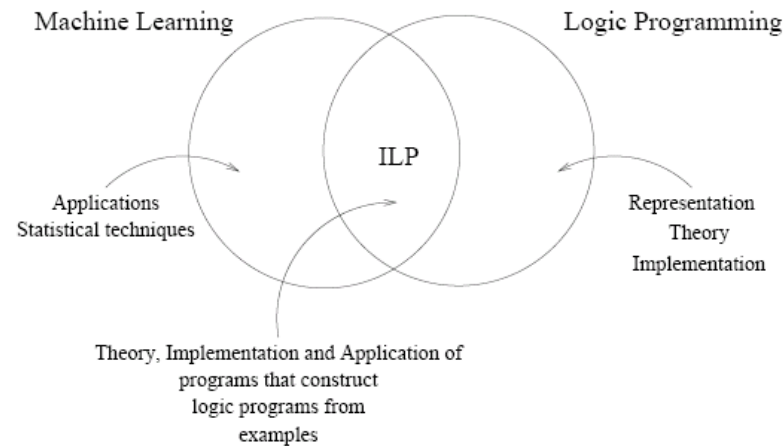
IBM India Research Labs, Delhi





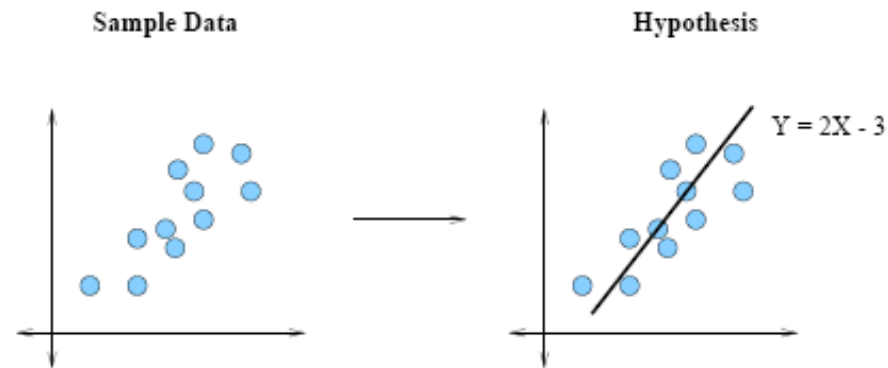
What is ILP

- Inductive Logic Programming?
- OR
- Inductive Logic Programming?

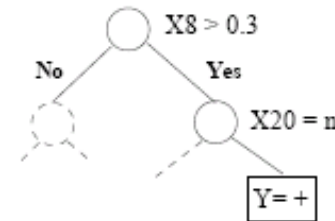


Machine Learning

Programs that hypothesize general descriptions from sample data



X1	X3	Y
		+
		+
	...	-
	...	-
	...	-
		-



Instances of some sorted/unordered lists of numbers
...

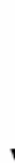
A general program for sorting lists of numbers



Logic Programming

- Study of using symbolic logic as a programming language
 - Specification = Programming

Logic program:
 $\forall X, Y \text{ grandfather}(X, Y) \leftarrow \exists Z (\text{father}(X, Z), \text{parent}(Z, Y))$
father(henry,jane) ←
father(henry,joe) ←
parent(jane,john) ←
parent(joe,robert) ←



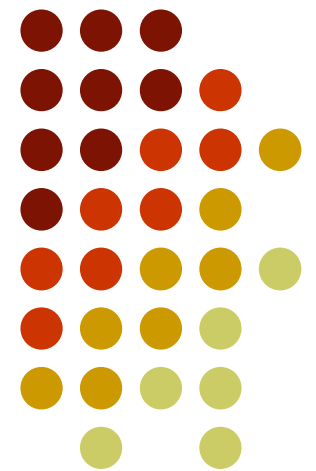
Derived facts:
grandfather(henry,john) ←
grandfather(henry,robert) ←



Logical Reasoning: 3 types

- Given preconditions α , post-conditions β and the rule $R1: \alpha \therefore \beta$ (α therefore β).
 - **Deduction** means determining β . It is using the rule and its preconditions to make a conclusion
 - **Induction** means determining $R1$. It is learning $R1$ after numerous examples of β and α .
 - **Abduction** means determining α . It is using the post-condition and the rule to assume that the precondition could explain the postcondition ($\beta \wedge R1 \Rightarrow \alpha$).

First Order Logic: Primer





First-Order Logic: Primer

- Constant: Objects in the domain
E.g.: Anna,
- Variable: Ranges over objects
E.g.: x
- Functions: Takes a tuple of objects and returns an object
E.g.: $\text{MotherOf}(x)$, $\text{Friends}(x, y)$
- Predicates: Represents either the property of an object or relationship between objects
E.g.: $\text{IsTall}(x)$, $\text{Friends}(x, y)$



First-Order Logic

- **Terms:** A constant, variable or functional expression (a function applied to a tuple of terms)

<i>Expression</i>	<i>Term?</i>
<i>peter</i>	✓
<i>X</i>	✓
<i>log(X)</i>	✓
<i>son(peter, peter)</i>	×
<i>log(son(peter, peter))</i>	×
<i>sin(log(cos(X/2)))</i>	✓

- **Atoms:** Predicate symbol applied to a tuple of terms
 - *son(spock, sarek)*
- **Clause:** Statements of the form $p_1 \vee p_2 \dots \leftarrow q_1 \wedge q_2$
 - **Definite clause:** Head has 1 atom without a negation and body has no negation.
- *Datalog* = First Order Logic – Function symbols
 - Term Datalog was coined in the mid 1980's by a group of researchers interested in database theory.

First-Order Logic



- **World/Domain** (Herbrand Interpretation):
Assignment of truth values to all ground predicates
 - If we take all predicates and replace variables with constants, we will get a large number of Boolean variables.
 - Construction of world is concerned with truth assignments to these Boolean variables.
 - Later, we will be interested in probability distributions over these assignments.
- **Propositionalization:**
Create all ground atoms and clauses. The resultant set is called the *Herbrand Universe*

Model Theory



- **(Herbrand) Model**

- An interpretation that gives the value *TRUE* for a formula is called a *model* for that formula

p	q	$p \leftarrow q$	Model for $p \leftarrow q$?
f	f	t	✓
f	t	f	×
t	f	t	✓
t	t	t	✓

- **Valid Formulae**

- Formulae for which *every* interpretation is a model are said to be *valid*

p	q	$(p \leftarrow q) \wedge q$	$p \leftarrow (p \leftarrow q) \wedge q$
f	f	f	t
f	t	f	t
t	f	f	t
t	t	t	t

- **Model Theory**

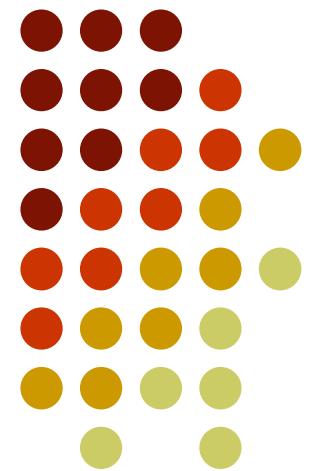
- Concerned with attributing meaning to logical sentences



Model Theory (contd)

- Satisfiability
 - A formula is said to be *satisfiable* if it has at least 1 model. Otherwise it is said to be *unsatisfiable*
- The set of all *Herbrand* models for a definite-clause program P is partially ordered by \subseteq and forms a lattice.

Reasoning: Primer





Deduction Theorem

Let $P = \{s_1, \dots, s_n\}$ be a set of clauses
and s be a sentence (not necessarily
ground)

Theorem. $P \models s$ iff $P - \{s_i\} \models (s \leftarrow s_i)$

Implication is preserved if we remove any sentence/formula from
the right and make it a condition



Proof Procedures In Logic

- Concern
 - Searching spaces efficiently, keeping in mind soundness and completeness
- Soundness
 - Anything that is derived *should* be a logical consequence
- Completeness
 - *Any* logical consequence should be derivable

Computation and Search Rules



- Typical logic problem: Solve queries of the form l_1, l_2, \dots, l_n ? where the l_i are literals
- Two issues:
 - Which literal of the l_i should be solved first?
The rule governing this is called the *computation* rule
(determines a tree of choices)
 - Which clause should be selected first, when more than one can be used to solve the literal selected?
 - The rule governing this is called the *search* rule
(determines the order in which this tree is searched)

Logic program:
 $\forall X, Y \text{ grandfather}(X, Y) \leftarrow \exists Z (\text{father}(X, Z), \text{parent}(Z, Y))$
father(henry,jane) ←
father(henry,joe) ←
parent(jane,john) ←
parent(joe,robert) ←



Derived facts:
grandfather(henry,john) ←
grandfather(henry,robert) ←

Resolution and Unification



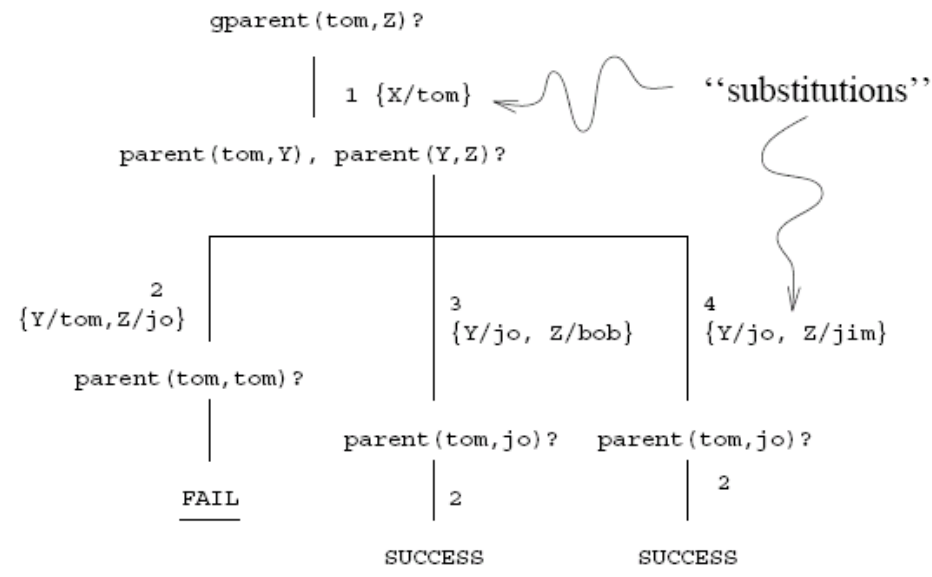
- Given: Program P and a query: $l_1, l_2, \dots, l_{j-1}, l_j, \dots, l_n$?
 1. Use the computation rule to select l_j
 2. Use the search rule to select a clause: $l_j \leftarrow b_1, b_2, \dots, b_k$ in P that can solve l_j . If none found, *STOP*.
 3. Solve the query: $l_1, l_2, \dots, l_{j-1}, b_1, b_2, \dots, b_k, \dots, l_n$?
 - The step of replacing the literal selected with the literals comprising the body of the clause is an application of the rule of inference known as **resolution**
- The head of the clause selected does not have to match *exactly* the literal selected. It will be enough if the two can *unify*
- **Unification:** There is some substitution of variables for terms in the two literals that makes them the same
 - Unification is “join” with respect to a **specialisation order**
 - E.g: $f(g(A), A) = f(B, xyz)$: Unifies A with the atom xyz and B with the term $g(xyz)$



Example for Datalog

1. $gparent(X, Z) \leftarrow parent(X, Y), parent(Y, Z)$
2. $parent(tom, jo) \leftarrow$
3. $parent(jo, bob) \leftarrow$
4. $parent(jo, jim) \leftarrow$

Rightmost literal first computation rule:



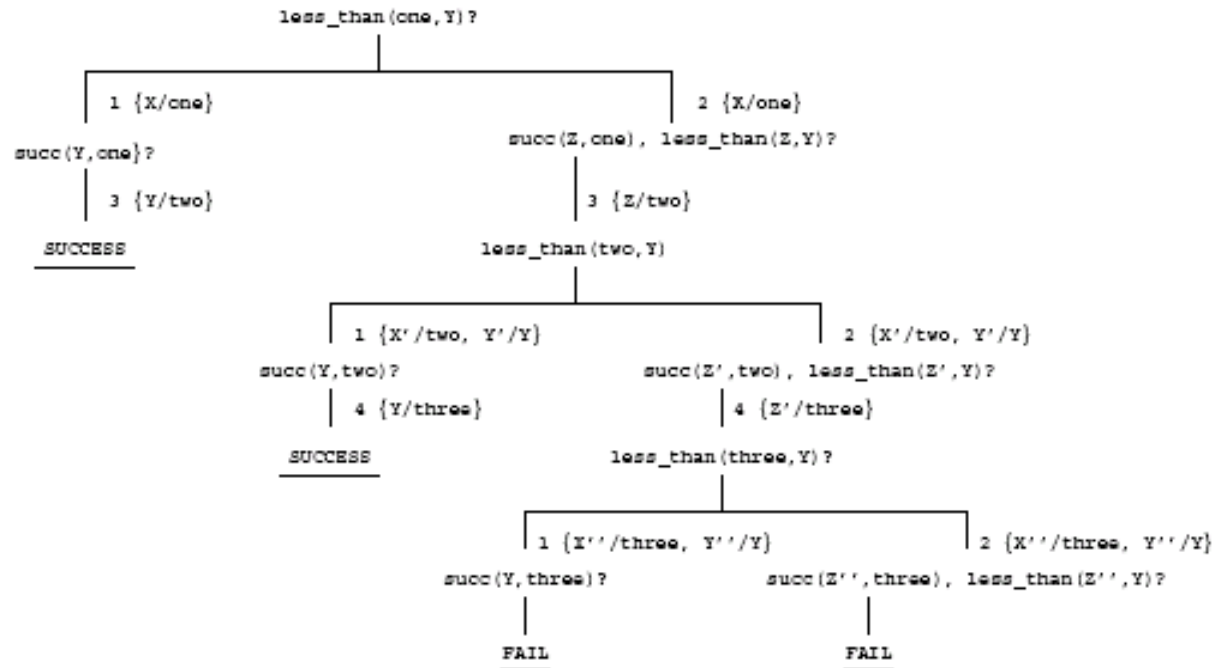
Example for Datalog



1. $less_than(X, Y) \leftarrow succ(Y, X)$
2. $less_than(X, Y) \leftarrow succ(Z, Y), less_than(Z, Y)$
3. $succ(two, one) \leftarrow$
4. $succ(three, two) \leftarrow$

Leftmost literal rule for the query

$less_than(one, Y)$



Computation and search rules: completeness



- One way to search the trees obtained so far is depth-first, left-to-right
 - Since clauses that appear first (reading top to bottom) in the program have been drawn on the left, this search rule selects clauses in order of appearance in the program
- Most logic programs are executed using the following:
 - Computation rule. Leftmost literal first
 - Search rule. Depth first search for clauses in order of appearance

Inference in First-Order Logic



- Traditionally done by theorem proving
 - E.g.: Prolog uses resolution
- More recently.....
 - *Propositionalization* followed by *model checking* turns out to be much faster
 - Comes as a surprise, because expected to be inefficient, since not “*lifted*”
 - However, in many domains, it is very fast.
- **Model checking: Satisfiability testing**
 - Two main families of satisfiability solvers:
 - **Backtracking** (Typical example is DPLL)
 - **Stochastic local search** (Typical example is WalkSAT)

Satisfiability



- **Input:**
 - Set of clauses
 - Convert KB to conjunctive normal form (CNF) after propositionalization
 - Every propositional formula can be converted into an equivalent formula that is in CNF using rules about logical equivalences:
 - The Double Negative Law
 - The De Morgan's laws
 - The distributive Law.
- **Output:**
 - Truth assignment that satisfies all clauses, OR
 - Failure (if no truth assignment exists)
- The paradigmatic NP-complete problem
- **Solution:** Search

Satisfiability



- **Parameters:**
 - #Clauses: More clauses give more constraints
 - #Variables: More variables give more freedom
- **Key point:**
 - Most SAT problems are under-constrained and actually easy
 - In many cases, any random solution satisfies all clauses
 - Though exponentially hard in worst case
- **Hard region: Over-constrained**
 - Small region of parameter space
 - Narrow range of #Clauses / #Variables
 - For random 3-sat problems, the hard region is approx for #Clauses / #Variables > 4 (*area of research*)



Backtracking

- Basic Idea:
 - Assign truth values by depth-first search
 1. Start off with no truth values assigned
- Assigning a variable deletes false literals and satisfied clauses
- Empty set of clauses: Success
- Empty clause: Failure
- Additional improvements:
 - **Unit propagation** (unit clause forces truth value)
 - **Pure literals** (same truth value everywhere)

DPLL example

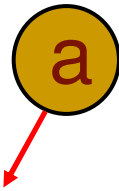


- C1:(a \vee b)
- C2:(\neg a \vee \neg b)
- C3:(a \vee \neg c)
- C4:(c \vee d \vee e)
- C5:(d \vee \neg e)
- C6:(\neg d \vee \neg f)
- C7:(f \vee e)
- C8:(\neg f \vee \neg e)

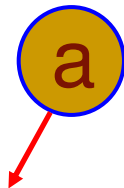
Legend

 false

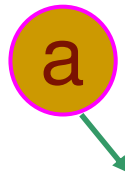
 true



a=false by branching

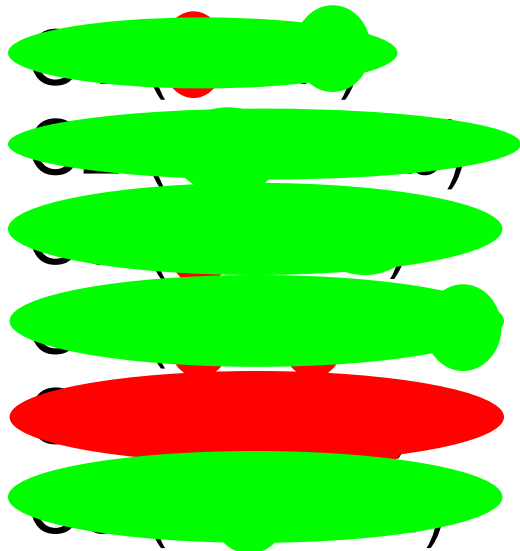


a=false by pure symbol




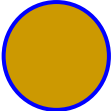
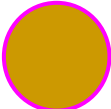
a=true by an unit clause

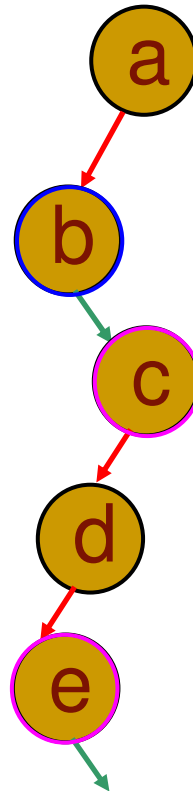
DPLL example



C7: $(f \vee e)$

C8: $(\neg f \vee \neg e)$

-  branching
-  pure symbol
-  unit clause





Unit Clause?

Q4: is a unit clause
 Yes: pure symbol
 No: pure clause

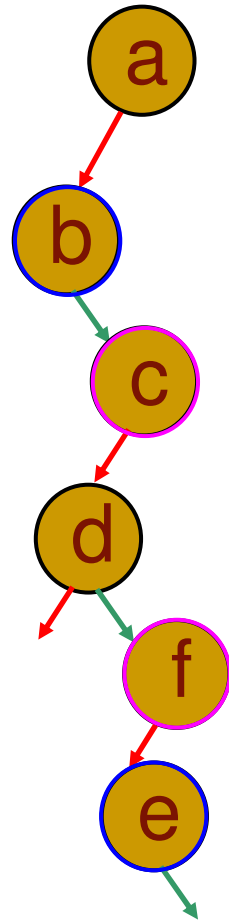
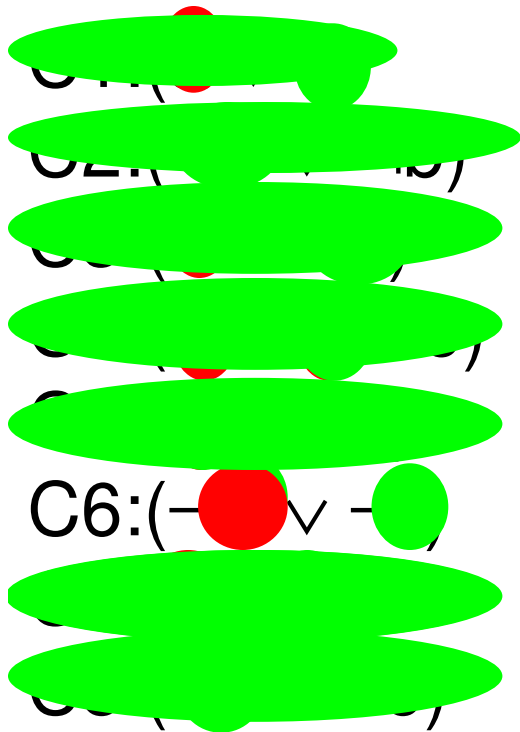
C5 is unsatisfied,
 Early termination

Backtrack upto the
 last branching:

$d = \text{false}$


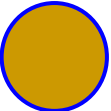
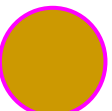
-  false
-  true



DPLL example



Formula Satisfied!

C6 is a unit clause

-  branching
-  pure symbol
-  unit clause

-  false
-  true



Exercise

- Find a satisfying assignment using DPLL

$$(\neg a \vee b) (\neg a \vee \neg b \vee c)$$

$$(\neg c \vee d \vee \neg e) (a \vee c)$$

$$(\neg d \vee \neg f) (a \vee c)$$

$$(e \vee \neg f)$$



The DPLL Algorithm

```
if  $CNF$  is empty then
  return true
else if  $CNF$  contains an empty clause then
  return false
else if  $CNF$  contains a pure literal  $x$  then
  return  $DPLL(CNF(x))$ 
else if  $CNF$  contains a unit clause  $\{u\}$  then
  return  $DPLL(CNF(u))$ 
else
  choose a variable  $x$  that appears in  $CNF$ 
  if  $DPLL(CNF(x)) = true$  then return true
  else return  $DPLL(CNF(\neg x))$ 
```

Stochastic Local Search



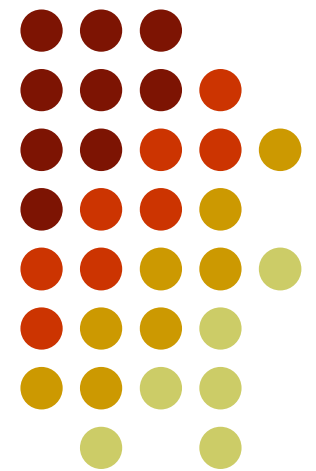
- Uses complete assignments instead of partial
- Start with random state
- Flip variables in unsatisfied clauses
- Hill-climbing: Minimize # unsatisfied clauses
- Avoid local minima: Random flips
- Multiple restarts

The WalkSAT Algorithm



```
for  $i \leftarrow 1$  to max-tries do  
  solution = random truth assignment  
  for  $j \leftarrow 1$  to max-flips do  
    if all clauses satisfied then  
      return solution  
     $c \leftarrow$  random unsatisfied clause  
    with probability  $p$   
      flip a random variable in  $c$   
    else  
      flip variable in  $c$  that maximizes  
        number of satisfied clauses  
return failure
```

Rule Induction





Rule Induction

- **Given:** Set of positive and negative examples of some concept
 - **Example:** $(x_1, x_2, \dots, x_n, y)$
 - y : **concept** (Boolean)
 - x_1, x_2, \dots, x_n : **attributes** (assume Boolean)
- **Goal:** Induce a set of rules that cover all positive examples and no negative ones
 - **Rule:** $x_a \wedge x_b \wedge \dots \Rightarrow y$ (x_a : Literal, i.e., x_i or its negation)
 - Same as **Horn clause:** $Body \Rightarrow Head$
 - Rule r **covers** example x iff x satisfies body of r
- **Eval(r):** Accuracy, info. gain, coverage, support, etc.



Learning a Single Rule

```
head ← y  
body ← ∅  
repeat  
  for each literal x  
     $r_x \leftarrow r$  with x added to body  
    Eval( $r_x$ )  
    body ← body ^ best x  
until no x improves Eval(r)  
return r
```

Learning a Set of Rules



$R \leftarrow \emptyset$

$S \leftarrow \text{examples}$

repeat

learn a single rule r

$R \leftarrow R \cup \{ r \}$

$S \leftarrow S - \text{positive examples covered by } r$

until $S = \emptyset$

return R



First-Order Rule Induction

- y and x_i are now predicates with arguments
E.g.: y is **Ancestor(x,y)**, x_i is **Parent(x,y)**
- Literals to add are predicates or their negations
- Literal to add must include at least one variable already appearing in rule
- Adding a literal changes # groundings of rule
E.g.: **Ancestor(x,z) ^ Parent(z,y) \Rightarrow Ancestor(x,y)**
- $Eval(r)$ must take this into account
E.g.: Multiply by # positive groundings of rule still covered after adding literal

“Inductive” Logic Programming



(Sample data)

Examples:

```
grandfather(henry, john) ←  
grandfather(henry, robert) ←
```

+

Background:

```
father(henry, jane) ←  
father(henry, joe) ←  
parent(jane, john) ←  
parent(joe, robert) ←
```



Hypothesis:

```
 $\forall X, Y \text{ grandfather}(X, Y) \leftarrow \exists Z (\text{father}(X, Z), \text{parent}(Z, Y))$ 
```

(A logic program)

More “Interesting” ILP



Examples:

Some carcinogenic chemicals
Some non-carcinogenic chemicals

1000's

+

Background:

Molecular structure of chemicals
General chemical knowledge

10,000's



Hypothesis:

$\forall X \text{ carcinogenic}(X) \leftarrow \dots$

...

...

10's

Induction by inverting deduction



- First investigated in depth mathematically by the 19th century political economist and philosopher of science Stanley Jevons
- From Jevons' book on inductive inference:
 - *Induction is, in fact, the inverse operation of deduction, and cannot be conceived to exist without the corresponding operation, so that the question of relative importance cannot arise. Who thinks of asking whether addition or subtraction is the more important process in arithmetic? But at the same time much difference in difficulty may exist between a direct and inverse operation; the integral calculus, for instance, is infinitely more difficult than the differential calculus of which it is the inverse. Similarly, it must be allowed that inductive investigations are of a far higher degree of difficulty and complexity than any questions of deduction; ...*

Hypothesis formation and justification



- **Abduction.** Process of hypothesis formation.
- **Justification.** The degree of belief assigned to a hypothesis given a certain amount of evidence.



Specific logical setting for abduction

B	$= C_1 \wedge C_2 \wedge \dots$	Background
E	$= E^+ \wedge E^-$	Examples
E^+	$= e_1 \wedge e_2 \wedge \dots$	Positive Examples
E^-	$= \overline{f_1} \wedge \overline{f_2} \wedge \dots$	Negative Examples
H	$= D_1 \wedge D_2 \wedge \dots$	Hypothesis

Prior Satisfiability. $B \wedge E^- \not\models \square$

Posterior Satisfiability. $B \wedge H \wedge E^- \not\models \square$

Prior Necessity. $B \not\models E^+$

Posterior Sufficiency. $B \wedge H \models E^+$,
 $B \wedge D_i \models e_1 \vee e_2 \vee \dots$

Acknowledgement



Some Slides borrowed from

- *Ashwin Srinivasan*
- *Pedro Domingos tutorial on Statistical Relational Learning at ICML'07*