Machine Learning and Data Analysis
Lecture 12: Learning in Predicate Logic

Filip Železný

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics
Intelligent Data Analysis lab
http://ida.felk.cvut.cz

January 11, 2011

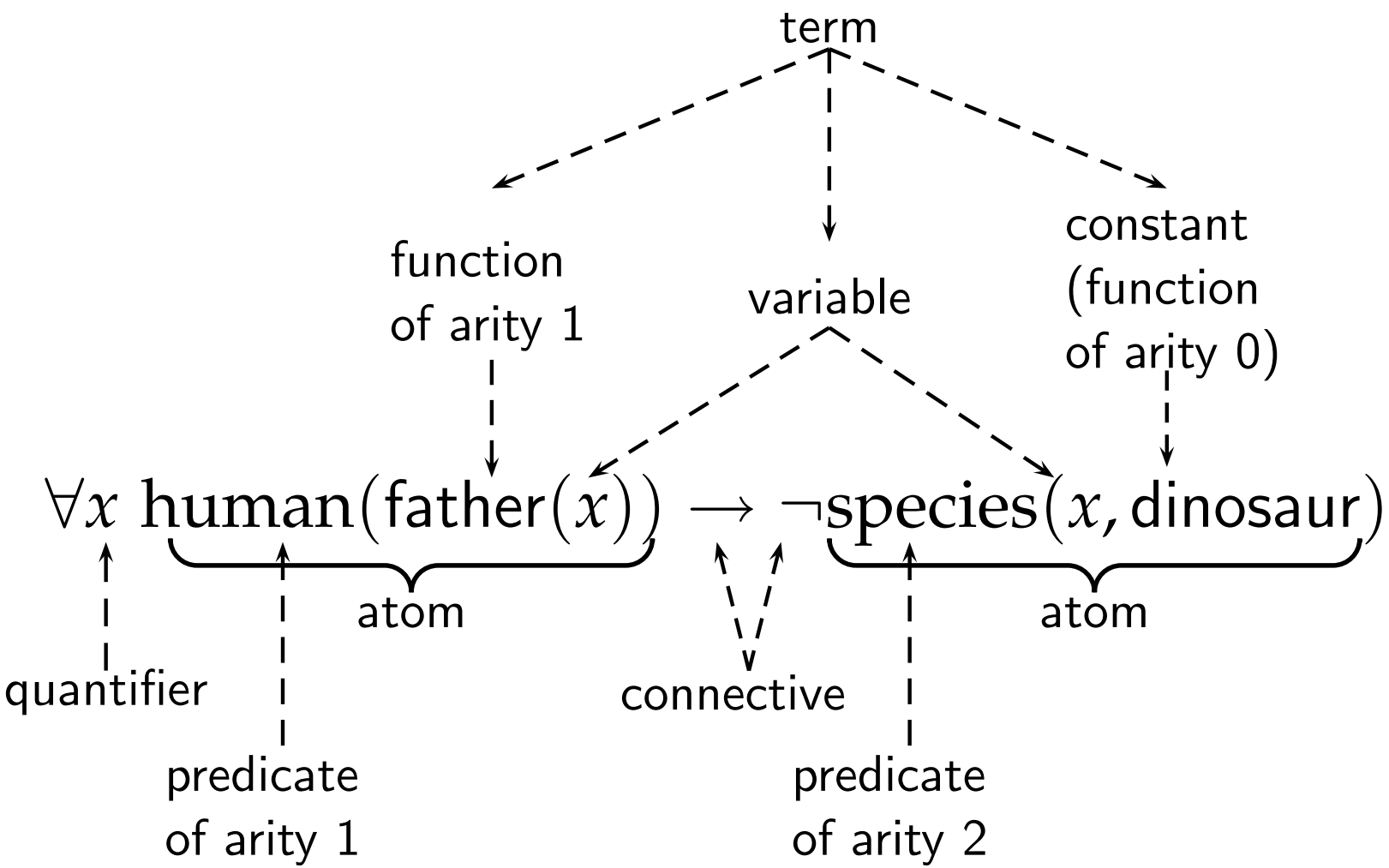
Review of PAC-learning in Propositional Logic

In propositional logic, we have proved the following classes PAC learnable:

- Conjunctions and disjunctions on n propositional symbols
- k-conjunctions and k-disjunctions
  - contain at most k literals
  - $|\mathcal{Q}^{k\text{-conj}}| = |\mathcal{Q}^{k\text{-disj}}| = \mathcal{O}(n^k)$
- k-CNF and k-DNF
  - Conjunctions of clauses (k-disjunctions), resp. disjunctions of terms (k-conjunctions)
  - Poly number of $\mathcal{O}(n^k)$ different possible clauses (terms), so k-CNF and k-DNF as easy as conjunctions (disjunctions)
- k-term DNF and k-clause CNF
  - by k-CNF, resp. k-DNF, not by themselves!
- k-Decision lists

What about learnability in *predicate* logic?

Review of First-Order Predicate Logic Syntax



Functions can be nested, e.g. $\text{father}(\text{father}(\ldots\text{father}(x)\ldots))$.

Operator precedence

To avoid too many parentheses, we set the following operator precedence

1. negation $\neg$
2. conjunction $\wedge$
3. disjunction $\vee$
4. implication $\rightarrow$
5. quantifiers $\exists$, $\forall$

CNF Representation

Any first-order formula can be converted into a CNF, which is a conjunction of universally quantified clauses = clausal theory

Example:

$$\forall x \ \text{person}(x) \rightarrow \exists y \ \text{parent}(y, x) \wedge \neg \left(\text{mother}(y, x) \wedge \text{father}(y, x)\right)$$

rewrite (note precedence of $\wedge$ over $\vee$):

$$\forall x \ \neg\text{person}(x) \vee \exists y \ \text{parent}(y, x) \wedge \neg \left(\text{mother}(y, x) \wedge \text{father}(y, x)\right)$$

move negation straight to atoms:

$$\forall x \ \neg\text{person}(x) \vee \exists y \ \text{parent}(y, x) \wedge \left(\neg\text{mother}(y, x) \vee \neg\text{father}(y, x)\right)$$

CNF Representation (cont'd)

skolemize:

$$\forall x \ \neg\text{person}(x) \vee \text{parent}(\text{par}(x), x) \wedge$$
$$(\neg\text{mother}(\text{par}(x), x) \vee \neg\text{father}(\text{par}(x), x))$$

use the distribution principle $a \vee (b \wedge c) \equiv (a \vee b) \wedge (a \vee c)$

$$\forall x \ \ (\neg\text{person}(x) \vee \text{parent}(\text{par}(x), x)) \wedge$$
$$(\neg\text{person}(x) \vee \neg\text{mother}(\text{par}(x), x) \vee \neg\text{father}(\text{par}(x), x))$$

Drop universal quantifiers since *all* variables are now quantified universally.

$$(\neg\text{person}(x) \vee \text{parent}(\text{par}(x), x)) \wedge$$
$$(\neg\text{person}(x) \vee \neg\text{mother}(\text{par}(x), x) \vee \neg\text{father}(\text{par}(x), x))$$

CNF Representation (cont'd)

Rewrite as a *set* of clauses (implicitly connected by $\wedge$).

For readability, use the principle $\neg a \vee b \equiv a \rightarrow b$

$$\text{person}(x) \rightarrow \text{parent}(\text{par}(x), x)$$
$$\text{person}(x) \wedge \text{mother}(\text{par}(x), x) \wedge \text{father}(\text{par}(x), x) \rightarrow \bot$$

$\bot$ is logical contradiction ('false').

Positive and Negative Literals

For any clause C with literals l_i $(1 \leq i \leq j+k)$

$$C = \underbrace{\neg l_1 \vee \neg l_2 \vee \ldots \vee \neg l_j}_{\text{negative literals}} \vee \underbrace{l_{j+1} \vee l_{j+2} \vee \ldots \vee l_{j+k}}_{\text{positive literals}}$$

$$= \underbrace{l_1 \wedge l_2 \wedge \ldots \wedge l_j}_{\text{negative literals}} \rightarrow \underbrace{l_{j+1} \vee l_{j+2} \vee \ldots \vee l_{j+k}}_{\text{positive literals}}$$

denote

- $C^- = \{l_1, \ldots l_j\}$ (the set of atoms appearing as negative literals in C)
- $C^+ = \{l_{j+1}, \ldots l_{j+k}\}$ (those appearing as positive literals in C)

Substitution

A *substitution* is a mapping from a set of variables to terms.

Example

$$\vartheta = \{x \mapsto x, y \mapsto x, z \mapsto \mathrm{f}(x, \mathrm{a})\}$$

Example of applying a substitution:

$$C = \mathrm{p}(w, x, y) \rightarrow \mathrm{q}(z, \mathrm{a})$$

$$C\vartheta = \mathrm{p}(w, x, x) \rightarrow \mathrm{q}(\mathrm{f}(x, \mathrm{a}), \mathrm{a})$$

$$C^{-}\vartheta = \{\mathrm{p}(w, x, x)\}$$

A *ground* formula (or a set of atoms) is a formula (set of atoms) containing no variables.

A *grounding substitution* for a formula (or a set of atoms) ϕ is a substitution ϑ such that $\phi\vartheta$ is ground.

Herbrand Interpretation and Model

Generalization of *truth-value assignment* in propositional logic.

Let P be a finite set of predicates and F be finite set of functions (including constants).

Herbrand base $\mathcal{H}$: set of all atoms that can be constructed with P and F (may be infinite)

Herbrand interpretation I: a subset of $\mathcal{H}$. (We will omit 'Herbrand'.)

A clause C is *true* in I if for any grounding substitution ϑ for C

$$C^+\vartheta \cap I \neq \{\} \text{ whenever } C^-\vartheta \subseteq I$$

An interpretation I is *model* of a clausal theory T if all clauses of T are true in I; we write

$$I \models T$$

Interpretations and Models: Examples

Consider the clausal theory:

$$\text{edge}(x, y) \rightarrow \text{path}(x, y)$$
$$\text{edge}(x, z) \wedge \text{path}(z, y) \rightarrow \text{path}(x, y)$$

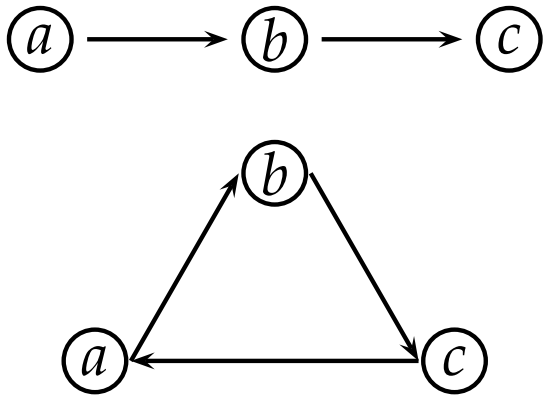
The following interpretations are two of its models

Herbrand:

$$\{\text{edge}(\text{a}, \text{b}), \text{edge}(\text{b}, \text{c}),$$
$$\text{path}(\text{a}, \text{b}), \text{path}(\text{b}, \text{c}), \text{path}(\text{a}, \text{c})\}$$

$$\{\text{edge}(\text{a}, \text{b}), \text{edge}(\text{b}, \text{c}), \text{edge}(\text{c}, \text{a})$$
$$\text{path}(\text{a}, \text{a}), \text{path}(\text{a}, \text{b}), \text{path}(\text{a}, \text{c}),$$
$$\text{path}(\text{b}, \text{a}), \text{path}(\text{b}, \text{b}), \text{path}(\text{b}, \text{c})$$
$$\text{path}(\text{c}, \text{a}), \text{path}(\text{c}, \text{b}), \text{path}(\text{c}, \text{c})\}$$

Non-Herbrand (informal):



Range-Restricted jk-theories

Clause C is

- *range-restricted* if all variables in C^+ appear also in C^-.
- a *jk-clause* if it contains at most k literals and each of them contains at most j occurences of predicate, variable and function symbols

Example:

$$\text{son}(x, y) \wedge \text{brother}(y, z) \rightarrow \text{uncle}(z, x)$$

is a range-restricted $3, 3$-clause, but not e.g. a range-restricted $3, 2$-clause.

A range-restricted jk-theory is a set of range-restricted jk-clauses.

Learning Range-Restricted jk-Theories

Given finite sets P, F of predicate/function symbols, observation space X contains finite interpretations x on the Herbrand base constructed with P and F. The *size* of examples is given by the triple

$$|P|, |F|, n = \max\{|x| \mid x \in X\}$$

Hypothesis space $\mathcal{Q}^{jk} =$

$$\{q_T \mid T \text{ is a range restricted } jk\text{-theory using only symbols from } P \text{ and } F\}$$

$$q_T(x) = 1 \text{ iff } x \models T$$

To see if $\mathcal{Q}^{jk}$ is efficiently PAC-learnable, we will determine if

- $\ln |\mathcal{Q}^{jk}|$ is polynomial.
- a consistent q_T can be produced for any sample $S = \{x_1, \ldots x_m\}$ in polynomial time.

polynomial: in $1/\delta$, $1/\epsilon$, $|P|$, $|F|$, n

Cardinality of $\mathcal{Q}^{jk}$

With l different literals

$$c = \sum_{i=1}^{k} \left(\begin{array}{c} l \\ i \end{array} \right) = \mathcal{O}(l^k)$$

different clauses containing at most k literals can be constructed.

Atoms can be constructed with $|P|$ different predicate symbols.

With maximum atom size j, an atom has at most $j-1$ places for function or variable symbols (exactly one place occupied by the predicate symbol).

There are $|F|$ different function symbols. Since any jk-theory contains at most jk variables, there are jk different variable symbols.

So $|P|(|F|+jk)^{j-1}$ different atoms, i.e. $l = 2|P|(|F|+jk)^{j-1}$ different literals can be used to form clauses.

Therefore c is polynomial in $|P|$ and $|F|$ and so is $\ln|\mathcal{Q}^{jk}| = \ln|2^c|$.

Consistent q_T

A q_T consistent with a sample S is produced by a generalization algorithm.

$\phi = \wedge_{i=1}^{c'} C_i$ {conjunction of all range-restricted jk-clauses formed using P, F and jk different variables}
for each example $(x, 1) \in S$ **do**
  for $i = 1 \ldots c'$ **do**
    if $x \not\models C_i$ **then**
      delete C_i from ϕ
return ϕ

Number c from the previous slide considered all jk-clauses, not only those range-restricted, so $c' \leq c$.

Similar to the generalization algorithm for learning propositional conjunctions. Also here only *positive* examples (interpretations) are used.

Efficiency of the Generalization Algorithm

The algorithm makes mc' steps ($m = |S|$) where c' is polynomial, i.e. polynomial number of steps.

It needs polynomial time if each step takes polynomial time, i.e. if checking $x \models C$ for interpretation x and a range-restricted jk-clause C takes polynomial time.

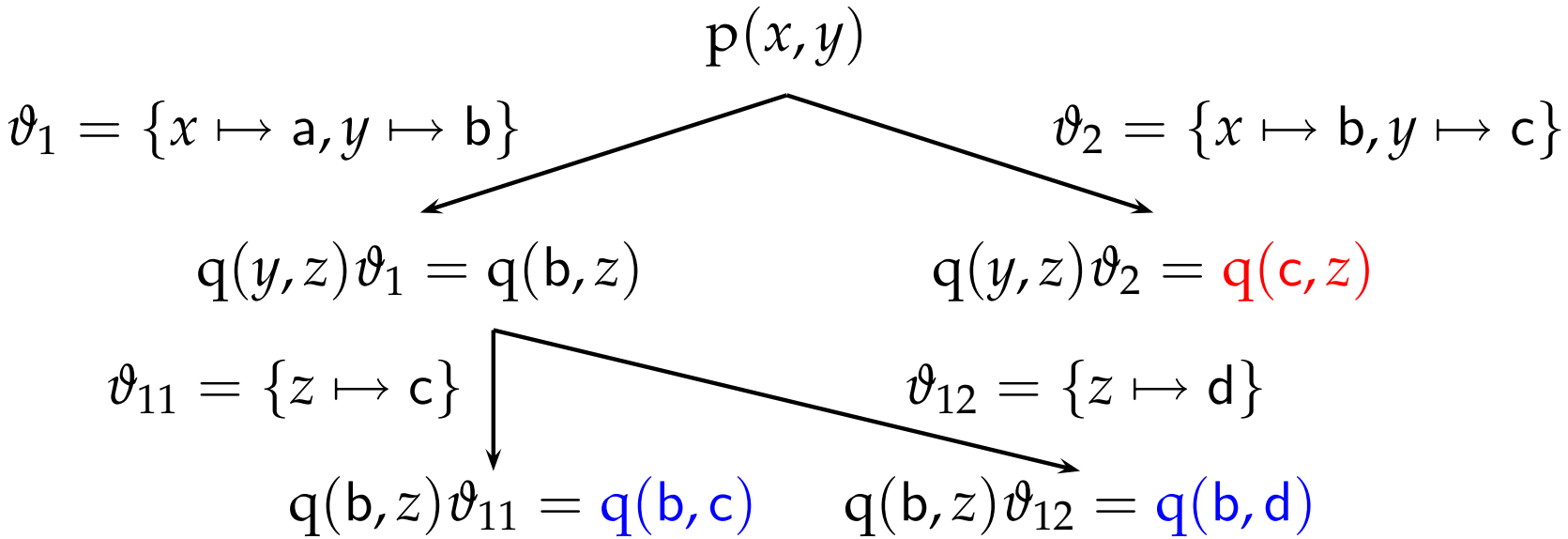
Checking $x \models C_i$ requires

1. finding all substitutions ϑ so that $C^- \vartheta \subseteq x$
2. checking if $C^+ \vartheta \cap x \neq \{\}$ for each such ϑ

Finding ϑ so that $C^-\vartheta \subseteq x$

A tree-search

Example: $C^- = \{\mathrm{p}(x,y), \mathrm{q}(y,z)\}$, $x = \{\mathrm{p}(\mathrm{a},\mathrm{b}), \mathrm{p}(\mathrm{b},\mathrm{c}), \mathrm{q}(\mathrm{b},\mathrm{c}), \mathrm{q}(\mathrm{b},\mathrm{d})\}$



Solutions:

- $\vartheta = \vartheta_1 \cup \vartheta_{11} = \{x \mapsto \mathrm{a}, y \mapsto \mathrm{b}, z \mapsto \mathrm{c}\}$
- $\vartheta = \vartheta_1 \cup \vartheta_{11} = \{x \mapsto \mathrm{a}, y \mapsto \mathrm{b}, z \mapsto \mathrm{d}\}$

PAC-learnability of $\mathcal{Q}^{jk}$

The tree has depth at most k and branching factor at most n (maximum size of interpretation x).

Thus it has at most n^k vertices.

The atom in each vertex has at most j arguments so the tree can be searched in $\mathcal{O}((jn)^k)$ units of time.

For each resulting ϑ, $C^+\vartheta \cap x \neq \{\}$ can be checked in $\mathcal{O}(jn)$ units of time since C is range-restricted and thus $C^+\vartheta$ is ground.

Therefore, checking $x \models C_i$ takes polynomial time.

In summary, the generalization algorithm runs in time polynomial in $|P|, |F|, n$.

Since also $\ln|\mathcal{Q}^{jk}|$ is polynomial in $|P|$ and $|F|$, $\mathcal{Q}^{jk}$ is efficiently PAC-learnable.

Expressivness of $\mathcal{Q}^{jk}$

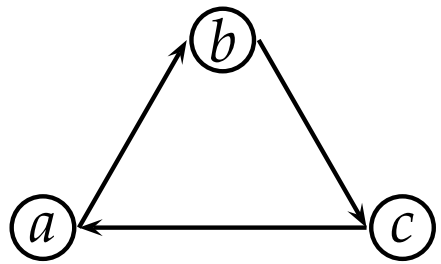
$\mathcal{Q}^{1k} = \mathcal{Q}^{k\text{-CNF}}$ (propositional k-CNF), but for $j > 1$, $\mathcal{Q}^{jk}$ is much more expressive. For example, the concept of a *directed path* in a graph can be (recursively) expressed in $\mathcal{Q}^{3,3}$:

$$\text{edge}(x, y) \rightarrow \text{path}(x, y)$$
$$\text{edge}(x, z) \wedge \text{path}(z, y) \rightarrow \text{path}(x, y)$$

the concept be efficiently PAC-learned from example interpretations such as

$$\{\text{edge}(\text{a}, \text{b}), \text{edge}(\text{b}, \text{c}), \text{path}(\text{a}, \text{b}), \text{path}(\text{b}, \text{c}), \text{path}(\text{a}, \text{c})\}$$

$$\{\text{edge}(\text{a}, \text{b}), \text{edge}(\text{b}, \text{c}), \text{edge}(\text{c}, \text{a}) \text{path}(\text{a}, \text{a}), \text{path}(\text{a}, \text{b}), \text{path}(\text{a}, \text{c}), \text{path}(\text{b}, \text{a}), \text{path}(\text{b}, \text{b}), \text{path}(\text{b}, \text{c}) \text{path}(\text{c}, \text{a}), \text{path}(\text{c}, \text{b}), \text{path}(\text{c}, \text{c})\}$$



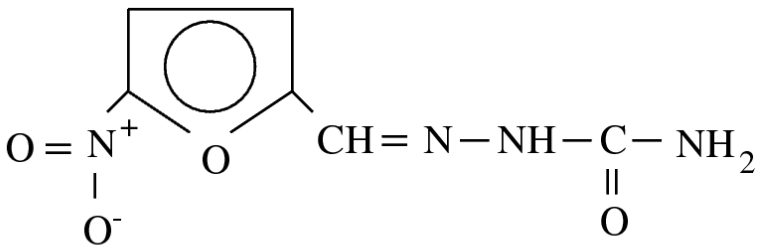
Inductive Logic Programming

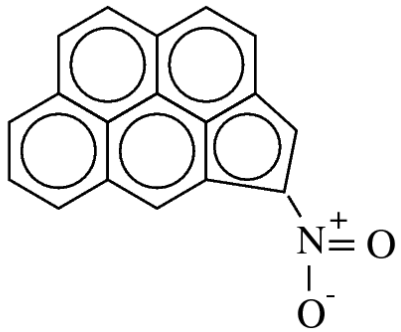
The concept can be written as a Prolog program:

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).
```

Thus algorithms for learning in predicate logic allow to *induce* (= learn) some Prolog *programs* from examples.

Therefore they are called algorithms of inductive logic programming (ILP). ILP has applications in problems where hypotheses are learned from structured data such as in biochemistry





Learning from Intepretations

The learning principles explained so far are referred to as the ILP setting of learning from (finite) interpretations.

Some concepts have only infinite models. E.g.

$$\mathrm{even}(\mathrm{s}(\mathrm{s}(0)))$$
$$\mathrm{even}(x) \rightarrow \mathrm{even}(\mathrm{s}(\mathrm{s}(x)))$$

(the unary function s produces the successor of its argument)

so they cannot be learned from finite interpretations.

Learning from Clauses

An alternative setting of ILP is known as learning from clauses or learning from entailment.

Here $x \in X$ are first-order predicate clauses and $\mathcal{Q}$ contains hypotheses q_T in the form of first-order clausal theories T such that

$$q_T(x) = 1 \text{ iff } T \vdash x$$

where the $\vdash$ is the *logical entailment* relation. $T \vdash x$ holds iff all models of T are also models of x. For example:

$$\begin{array}{l} \text{human}(\text{Sokrates}) \\ \text{human}(x) \rightarrow \text{mortal}(x) \end{array} \vdash \text{mortal}(\text{Sokrates})$$

Samples S from X are assumed non-contradictory, i.e. no positive (negative) example entails a negative (positive) example.

Learning from Clauses: Example

Consider sample S containing the following positive examples

$$x_1 = \text{even}(\text{s}(\text{s}(\text{s}(\text{s}(\text{s}(\text{s}(0)))))))$$
$$x_2 = \text{even}(\text{s}(\text{s}(x))) \rightarrow \text{even}(\text{s}(\text{s}(\text{s}(\text{s}(x)))))$$

and the negative example

$$x_3 = \text{even}(\text{s}(\text{s}(x)))$$

Then the intended hypothesis q_T

$$\text{even}(\text{s}(\text{s}(0)))$$
$$\text{even}(x) \rightarrow \text{even}(\text{s}(\text{s}(x)))$$

is consistent with S ($T \vdash x_1$, $T \vdash x_2$, $T \nvdash x_3$), which can be shown e.g. by resolution. So can we learn from clauses what we cannot learn from interpretations?

Learning from Clauses: Problems

Main problems causing the inability to PAC-learn general clausal theories from clauses.

1. **Infinite VC-dimension.** General clausal theories have the expressive power of a Turing machine. Any (finite) sample S can be shattered e.g. by the trivial theory

$$S = \wedge_i x_i$$

where x_i are clauses taken from the positive examples of S. This problem can be avoided by making $\mathcal{Q}$ finite, for example $\mathcal{Q} = \mathcal{Q}^{jk}$.

2. **Undecidability of entailment**

$$T \vdash x$$

is generally unecidable (even if T is a single clause)
This makes it impossible to *verify* if a hypothesis is consistent with a sample, therefore *finding* a consistent hypothesis is also generally impossible.

ϑ-Subsumption

If T is a single clause, $T \vdash x$ can be *approximated* by the *ϑ-subsumption* relation $\preceq_\vartheta$.

$T \preceq_\vartheta x$ holds iff there is a substitution ϑ such that all literals of $T\vartheta$ are also in x.

Checking ϑ-subsumption decidable but NP-complete. Maximum size of the two clauses must be fixed for efficient checking.

Example:

$$T = \text{path}(x,z) \wedge \text{edge}(z,y) \rightarrow \text{path}(x,y)$$
$$x = \text{path}(x,\text{a}) \wedge \text{edge}(\text{a},y) \wedge \text{edge}(\text{b},y) \rightarrow \text{path}(x,y)$$

Here both $T \vdash x$ and $T \preceq_\vartheta x$ (for $\vartheta = \{z \mapsto \text{a}\}$)

ϑ-Subsumption (cont'd)

However, in:

$$T = \mathrm{p}(x,y,z) \rightarrow \mathrm{p}(y,z,x)$$
$$x = \mathrm{p}(x,y,z) \rightarrow \mathrm{p}(z,x,y)$$

$T \vdash x$ (can be checked by resolving T with T) but $T \not\preceq_\vartheta x$.

ϑ-Subsumption is an approximation to entailment in that

- $T \preceq_\vartheta x$ implies $T \vdash x$
- $T \vdash x$ implies $T \preceq_\vartheta x$ only if T and x are not *self-resolving*