Inductive logic programming

What I learned

Simon Jacquet

Faculty of Computer Science
Unamur

October 15th 2021

Logic

- With,

Statement A	All men are mortal
Statement B	Socrates is a man

- One can deduce,

Statement C	Socrates is mortal

- Deduction goes from general to specific
  - ☞ Statement A is more 'general' than statement C

Generality with subsumption (i)

Formula substitution

Let θ be a substitution of the form $\{v_1/t_1, ..., v_n/t_n\}$.
Let F be an arbitrary formula.
$F\theta$ is the formula F where each of its variable v_i have been replaced by t_i.

Atom subsumption

Let θ be a substitution of the form $\{v_1/t_1, ..., v_n/t_n\}$.
Let F be an arbitrary atom.
$F\theta$ is the formula F where each of its variable v_i have been replaced by t_i.

Clause subsumption

Let C and D be clauses.
$C \preceq D$, i.e. C subsumes D iff $\exists\theta, C\theta \subseteq D$.

Generality with subsumption (ii)

A	All men are mortal	mortal(X)
C	Socrates is mortal	mortal(socrates)

- Let $\theta = \{X/\text{socrates}\}$

$$\begin{array}{rccc} & A\theta & = & C \\ \Leftrightarrow & A & \preceq & C \end{array}$$

☞ Statement A is indeed more general than statement C

Generality with entailment

- Clauses of interest:

A	mortal(X) $\leftarrow$ man(X)
C	mortal(socrates)

- Background knowledge:

B	man(socrates)

- $A, B \models C$

  ☞ With B as background knowledge, A entails C

Inductive logic programming (i)

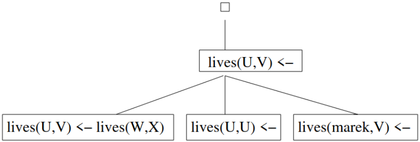
B	Background knowledge	Logic program
E	Examples	Set of ground unit clauses
H	Hypothesis	Logic program

Given B and E, find H, such that:

$$B, H \models E$$

Inductive logic programming (ii)

Shapiro's refinement graph:



- Search unbounded
- Consider many solution that do not entail all examples
- Prolog uses this type of search

Inductive logic programming (iii)

There exists another learning approach:

Given B and E, find H, such that:

$$
\begin{aligned}
B, H &\models E \\
B, \overline{E} &\models \overline{H} \\
B, \overline{E} &\models \overline{\bot} \models \overline{H} \\
H &\models \bot
\end{aligned}
$$

- $\overline{\bot}$ is the conjunction of literals that are true in all models $B \wedge \overline{E}$
- $\bot$ is the most specific clause for B and E
- The algorithm explores hypotheses more general than $\bot$
- The algorithm uses a metric to rank hypotheses
  - ☞ hypothesis chosen to be general but with few literals
- Progol uses this type of search

Limitations

- There are two main limitations to this approach:
  1. Cannot invent predicates
  2. Does not handle recursion

- Those two limitations are handled by meta interpretative learning (MIL)

Context free grammar

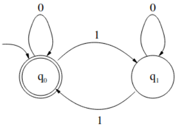
- A context free grammar consists of production rules of the following form:

$$
\begin{aligned}
S &\rightarrow \lambda \\
S &\rightarrow aS \\
S &\rightarrow Sb \\
S &\rightarrow S_1 S_2
\end{aligned}
$$

where S, S_1 and S_2 are non-terminal symbols, a, b are terminal symbols and λ marks the end of the string

Meta interpretative learning (i)

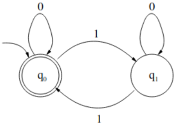
- Parity example:

Finite acceptor	Definite Clause Grammar (DCG)		Positive examples
	$q_0([],[])$	$\leftarrow$	λ
	$q_0([0\|A],B)$	$\leftarrow q_0(A,B)$	0
	$q_0([1\|A],B)$	$\leftarrow q_1(A,B)$	11
	$q_1([0\|A],B)$	$\leftarrow q_1(A,B)$	00
	$q_1([1\|A],B)$	$\leftarrow q_0(A,B)$	101

- The example above allow only strings containing an even number of ones
- How do we learn such a model from examples?

Meta interpretative learning (ii)

- Parity example:

Finite acceptor	**Definite Clause Grammar (DCG)**	**Positive examples**
	$q_0([],[]) \leftarrow$	λ
	$q_0([0\|A],B) \leftarrow q_0(A,B)$	0
	$q_0([1\|A],B) \leftarrow q_1(A,B)$	11
	$q_1([0\|A],B) \leftarrow q_1(A,B)$	00
	$q_1([1\|A],B) \leftarrow q_0(A,B)$	101

- Observe that clauses take the form

$$
\begin{aligned}
Q([],[]) &\leftarrow \\
Q([C|x],y) &\leftarrow Q(x,y)
\end{aligned}
$$

Meta interpretative learning (iii)

$$\begin{aligned} Q([],[]) &\leftarrow \\ Q([C|x],y) &\leftarrow Q(x,y) \end{aligned}$$

Meta-Interpreter (Regular)	**Ground facts**
$parse(S) \leftarrow parse(q0, S, [])$.	$acceptor(q0) \leftarrow$
$parse(Q, [], []) \leftarrow acceptor(Q)$.	$delta1(q0, 0, q0) \leftarrow$
$parse(Q, [C\|X], Y) \leftarrow$	$delta1(q0, 1, q1) \leftarrow$
$delta1(Q, C, P)$,	$delta1(q1, 0, q1) \leftarrow$
$parse(P, X, Y)$.	$delta1(q1, 1, q0) \leftarrow$

- Using the new formulation, we can build an interpreter
- We still want to learn the ground facts from examples

Meta interpretative learning (iv)

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P), abduce(delta1(Q,C,P),G1,G3),
↪  parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).

% Examples for parity example
parse([],[],G1), parse([0],G1,G2), parse([0,0],G2,G3), parse([1,1],G3,G4),
↪  parse([0,0,0],G4,G5), parse([0,1,1],G5,G6), parse([1,0,1],G6,G),
↪  not(parse([1],G,G)), not(parse([0,1],G,G).

% Output
G = [delta1(s(1),0,s(1)), delta1(s(0),1,s(1)), delta1(s(1),1,s(0)),
↪  delta1(s(0),0,s(0)), acceptor(s(0))]
```

- In a few lines of Prolog, we can quickly learn the parity example
- This model is called Metagol_R

Learning in action

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
? parse([],[],G1).
H = []
```

Learning in action

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
? parse(s(0),[],[],[],G1).
H = []
```

Learning in action

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
? abduce(acceptor(s(0)),[],G1).
H = []
```

Learning in action

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
G1 = [acceptor(s(0))])
H = G1 = [acceptor(s(0))]
```

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
? parse([0],[acceptor(s(0))],G2).
H = [acceptor(s(0))]
```

Learning in action

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
? parse(s(0),[0],[],[acceptor(s(0))],G2).
H = [acceptor(s(0))]
```

Learning in action

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
? skolem(P), abduce(delta1(s(0),0,P),[acceptor(s(0))],T),
| parse(P,[],[],T,G2).
H = [acceptor(s(0))]
```

Learning in action

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
? abduce(delta1(s(0),0,s(0)),[acceptor(s(0))],T),
| parse(s(0),[],[],T,G2).
H = [acceptor(s(0))]
```

Learning in action

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
? parse(s(0),[],[],[delta1(s(0),0,s(0)), acceptor(s(0))],G2).
H = [acceptor(s(0))]
```

Learning in action

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
? abduce(acceptor(s(0)),
| [delta1(s(0),0,s(0)), acceptor(s(0))],G2).
H = [acceptor(s(0))]
```

Learning in action

```
% Meta interpretor for regular grammar
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- skolem(P),
 ↪  abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

skolem(s(0)). skolem(s(1)).
```

```
G2 = [delta1(s(0),0,s(0)), acceptor(s(0))]
H = G2 = [delta1(s(0),0,s(0)), acceptor(s(0))]
```

Meta interpretative learning (v)

- MIL has gone even further to consider other metarules

Name	**Meta-Rule**	**Order**
Instance	$P(X, Y) \leftarrow$	$True$
Base	$P(x, y) \leftarrow Q(x, y)$	$P \succ Q$
Chain	$P(x, y) \leftarrow Q(x, z), R(z, y)$	$P \succ Q, P \succ R$
TailRec	$P(x, y) \leftarrow Q(x, z), P(z, y)$	$P \succ Q,$ $x \succ z \succ y$

- It considers datalog logic programs in $\mathcal{H}_2^2$
  - arity ≤ 2 for predicates
  - #atoms ≤ 2 in the body of clauses
- $\mathcal{H}_2^2$ has Universal Turing Machine expressivity

ILP for texts of law

- There are at least three ways to study texts of law with ILP
  - ☞ Identify logical rules in texts of law and generalize them with ILP (slides 6-8)
  - ☞ Suppose that words in texts of law form a context free grammar and learn it with MIL (slides 11-14)
  - ☞ Believe in the expressivity of $\mathcal{H}_2^2$ (slide 15)