Inductive logic programming

Simon Jacquet

Faculty of Computer Science
Unamur

December 20th 2021

Settings

$$B \wedge H \models E \tag{1}$$
$$B \wedge \overline{E} \models \overline{H} \tag{2}$$
$$B \wedge \overline{E} \models \overline{\bot} \tag{3}$$
$$\overline{\bot} \models \overline{H} \tag{4}$$
$$H \models \bot \tag{5}$$

- B: background knowledge
- H: hypothesis
- E: examples
- $\overline{\bot}$: set of all true literals (wrt. $B \wedge \overline{E}$)
- $\bot$: most specific clause

Covering algorithm

Algorithm 1: Cover set algorithm

input: h, i, B, M, E

1 If $E = \emptyset$ return B

2 Let e be the first example in E

3 Construct clause $\perp_i$ for e ; `// Algorithm 2`

4 Construct clause H from $\perp_i$; `// Algorithm 3`

5 Let $B = B \cup H$

6 Let $E' = \{e : e \in E \text{ and } B \models e\}$

7 Let $E = E - E'$

8 Goto 1

Algorithm for constructing $\perp_i$

Algorithm 2: Constructing $\perp_i$

1. Add $\overline{e}$ to the background knowledge
2. $InTerms = \emptyset$, $\perp = \emptyset$
3. Find the first head mode declaration h such that h subsumes a with substitution θ
   For each v/t in θ,
      if v corresponds to a #type, replace v in h by t
      if v corresponds to a +type or -type, replace v in h by v_k
         where v_k is the variable such that $k = hash(t)$
      If v corresponds to a +type, add t to the set InTerms.
   Add h to $\perp$.
4. For each body mode declaration b
      For every possible substitution θ of variables corresponding to +type
            by terms from the set InTerms
         Repeat recall times
            If Prolog succeeds on goal b with answer substitution θ'
               For each v/t in θ and θ'
                  If v corresponds to #type, replace v in b by t
                  otherwise replace v in b by v_k where $k = hash(t)$
                  If v corresponds to a -type, add t to the set InTerms
               Add $\overline{b}$ to $\perp$
5. Increment the variable depth
6. Goto step 4 if the maximum variable depth has been achieved.

Algorithm for searching the lattice

Algorithm 3: Lattice search algorithm

input: h, i, B, M, E

1 $Open = \{\square\}$, $Closed = \emptyset$

2 $s = best(Open)$, $Open = Open - s$, $Closed = Closed \cup \{s\}$; `// Node selection`

3 **if** $prune(s)$ **then** goto 5 ; `// Node pruning`

4 $Open = (Open \cup \rho(s)) - Closed$; `// Clause refinement`

5 **if** $terminated(Closed, Open)$ **then return** $best(Closed)$; `// End of search`

6 **if** $Open = \emptyset$ **then return** e; `// No generalisation`

7 Goto 2

Algorithm 3: Lattice search algorithm

input: h, i, B, M, E

1 $Open = \{\square\}$, $Closed = \emptyset$

2 $s = best(Open)$, $Open = Open - s$, $Closed = Closed \cup \{s\}$; `// Node selection`

3 **if** $prune(s)$ **then** goto 5 ; `// Node pruning`

4 $Open = (Open \cup \rho(s)) - Closed$; `// Clause refinement`

5 **if** $terminated(Closed, Open)$ **then return** $best(Closed)$; `// End of search`

6 **if** $Open = \emptyset$ **then return** e; `// No generalisation`

7 Goto 2

- To do so, we must be able to:
  - Compare solutions: $best(\cdot)$
  - Build new hypotheses: $\rho(\cdot)$
  - Discard wrong solutions: $prune(\cdot)$
  - End the search: $terminated(\cdot)$

Using numbers in the search

- p_s: # true positives
- n_s: # false positives
- c_s: # atoms in body
- h_s: # optimistic estimate of literals needed
- $g_s = p_s - c_s - h_s$
- $f_s = p_s - n_s - c_s - h_s$

- $best(Open)$: returns state s in set $Open$
  - $c_s \leq c$
  - with maximum f_s
- $prune(s)$: returns *true* iff either:
  - $n_s = 0$ and $f_s > 0$
  - $g_s \leq 0$
  - $c_s > c$
- $terminated(Closed, Open)$: return *true* iff:
  - $s = best(Closed)$, $n_s = 0$, $f_s > 0$
  - $f_s \geq g_{best(Open)}$

Learning with only positive examples

- Progol is able to learn from only positive examples
  - n_s, in the previous slide, would always be 0
  - An empty hypothesis would maximize f_s
- It searches for a good compromise between the size of an hypothesis and its generality
- Using probability distributions, it considers the hypothesis maximizing its log-probability:

$$\log(P(H|E)) = d_m - m\log(g(H)) - sz(H)$$

- Work is still necessary to better understand how to compute the generality $g(H)$ and size $sz(H)$ of an hypothesis

Refinement operator $\rho(\cdot)$

- $s_0 = \langle \Box, \emptyset, 0 \rangle$
- $\langle C', \theta', k' \rangle \in \rho(\langle C, \theta, k \rangle)$ iff either
  1. $C' = C$, $k' = k + 1$, $\theta' = \theta$ $(k < n)$
  2. $C' = C \cup \{l\}$, $k' = k$, $\langle l, \theta' \rangle \in \delta(\theta, k)$ and $C' \in \mathcal{L}_i(M)$

Splittable variable

A variable is splittable if it corresponds to a $+$type, -type in a modeh or a -type in a modeb.

$\langle l, \theta' \rangle \in \delta(\theta, k)$

Let $l_k = p(u_1, \ldots, u_m)$ k-th literal of $\perp_i$

Let $l = p(v_1, \ldots, v_m)$

- If u_j is not splittable: $v_j / u_j \in \theta$
- If u_j is splittable, either:
  - $v_j / u_j \in \theta'$
  - $v_j \notin dom(\theta)$, a new variable, and $\theta' = \theta \cup \{v_j / u_j\}$