Online Learnability of s-CNF

An ***s*-clause** is a clause with at most s literals. An ***s*-CNF** is a conjunction of s-clauses.

The class s-CNF is efficiently learnable online from $X = \{0,1\}^n$.

The proof is analogical to that of s-DNF learnability (the $c_1, c_2, \ldots c_{n'}$ are then all s-clauses).

(Learning an s-CNF from conjunctive observations is the subject of Project 1.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

s-term DNF and *s*-clause CNF

An ***s*-term DNF** is a disjunction of at most *s*conjunctions. An ***s*-clause CNF** is a conjunction of at most *s* clauses. The two names are also used to refer to the respective classes of formulas.

From the De Morgan's laws, it follows that each *s*-term DNF can be written as a tautologically equivalent *s*-CNF, by "multiplying out". Example with 2-term DNF:

$$(p_1 \wedge p_2 \wedge p_3) \vee (p_4 \wedge p_5) =$$
$$(p_1 \vee p_4) \wedge (p_1 \vee p_5) \wedge (p_2 \vee p_4) \wedge (p_2 \vee p_5) \wedge (p_3 \vee p_4) \wedge (p_3 \vee p_5)$$

Therefore

$$\mathcal{C}(s\text{-term DNF}) \subseteq \mathcal{C}(s\text{-CNF}) \tag{1}$$

Analogically, $\mathcal{C}(s\text{-clause CNF}) \subseteq \mathcal{C}(s\text{-DNF})$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Learnability of s-term DNF and s-clause CNF

Since s-DNF is efficiently learnable online from $X = \{0,1\}^n$, so is s-clause CNF.

This is because due to (1), for each target s-clause CNF, the agent can learn an equivalent s-DNF.

For an analogical reason, s-clause CNF is also efficiently learnable online from $X = \{0,1\}^n$, with the agent using s-DNF.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Clausal Observations and Hypothesis

Consider a conjunction $h = h_{k+1}$ learned from contingent conjunctions $x_{\leq k}$, i.e.,

$$h = \mathrm{lgg}(\{\, x_{k_i} \,\}) \tag{2}$$

where $\{\, x_{k_i} \,\}$ are exactly the positive examples from $x_{\leq k}$. (2) can be equivalently written as

$$\neg h = \mathrm{lgg}(\{\, \neg x_{k_i} \,\})$$

where $\neg h$ and all of $\neg x_{k_i}$ are *clauses* and the policy (??) is

$$h(x) = 1 \text{ iff } \neg h \models \neg x \tag{3}$$

because $\neg h \models \neg x$ is equivalent to $x \models h$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Disjunctive Observations and Hypothesis (cont'd)

For clausal hypotheses h and clausal observations x we define the decision policy

$$h(x) = 1 \text{ iff } h \models x \tag{4}$$

which is consistent with (3) where $\neg h$ and $\neg x$ are clauses.

Analogically to Theorem (??) we have the following evident principle

Theorem 1

Let h, h' be clauses. If $h \subseteq h'$ then $h \models h'$. Let furthermore h not be tautologically false. Then $h \subseteq h'$ if and only if $h \models h'$.

Thus if h is not tautologically false, (4) is equivalent to (??).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Disjunctive Observations and Hypothesis (cont'd)

The policy (??) is the same for for disjuctions and conjunctions, despite the difference btw. (4) and (??).

The mistake bound of the generalization algorithm asserted by Theorem (??) remains true even when X is assumed to consist of clauses.
(exercise problem.)

So clauses can be learned from clauses online by the generalization algorithm with no modification of it.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Clausal Observations and Hypothesis

Clauses can be written as implications, e.g. $\neg \mathrm{p} \vee \mathrm{q} \models\!\!\models \mathrm{p} \rightarrow \mathrm{q}$, and interpreted as *rules*. Consider two positive clausal examples

$$x_1 = \mathrm{man} \wedge \mathrm{adult} \wedge \mathrm{young} \rightarrow \mathrm{married} \vee \mathrm{bachelor}$$
$$x_2 = \mathrm{man} \wedge \mathrm{adult} \rightarrow \mathrm{married} \vee \mathrm{bachelor} \vee \mathrm{nerd}$$

$h = \mathsf{lgg}(x_1, x_2) = \mathsf{Lits}(x_1) \cap \mathsf{Lits}(x_2) =$

$$\mathrm{man} \wedge \mathrm{adult} \rightarrow \mathrm{married} \vee \mathrm{bachelor}$$

The correct rule is thus learned from observed rules which are true but insufficiently general.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

First-Order Logic Observations

We will study the case when observations X are *first-order logic* (**FOL**, for short) conjunctions or clauses, i.e., conjunctions or disjunctions of FOL literals made using

- a finite set $\mathcal{P}$ of *predicates*
- a finite set $\mathcal{F}$ of *functions* (including constants)

Symbols x, y, z (typed in italics, possibly with indexes) will denote FOL *variables*.

Note: confusion of a FOL variable *x or y with an* observation *x resp.* action *y is excluded: FOL variables occur only as terms in predicates.*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

First-Order Logic Conjunctions and Clauses

If a FOL conjunction or clause is shown *without quantifiers* then all its variables are implicitly quantified

- *existentially*, for a *conjunction*
- *universally*, for a *clause*

So the negation of a FOL conjunction is a FOL clause and vice versa.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

FOL Subsumption

We now extend subsumption to FOL.

Let h, h' be two FOL conjunctions or two FOL clauses. We say that h **subsumes** h' (written $h \subseteq_\theta h'$) if there is a substitution θ such that $\text{Lits}(h\theta) \subseteq \text{Lits}(h')$. We say that h **strictly subsumes** h' (written $h \subset_\theta h'$) if $h \subseteq_\theta h'$ and $h' \not\subseteq_\theta h$.

Propositional formulas are special cases of FOL formulas with propositional variables corresponding to zero-arity predicates. So subsumption as defined above is identical to this definition for propositional conjunctions or clauses.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Subsumption

"There exist two vertices not connected by an edge":

$$h = \mathrm{vertex}(x) \wedge \mathrm{vertex}(y) \wedge \neg \mathrm{edge}(x, y)$$

"There exists a vertex not connected by an edge to vertex a":

$$h' = \mathrm{vertex}(x) \wedge \mathrm{vertex}(\mathsf{a}) \wedge \neg \mathrm{edge}(x, \mathsf{a})$$

$h \subseteq_\theta h'$ because for $\theta = \{ y \mapsto \mathsf{a} \}$, $\mathrm{Lits}(h) \subseteq \mathrm{Lits}(h')$.

$h' \not\subseteq_\theta h$ because the literal $\mathrm{vertex}(\mathsf{a})$ is not in h.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Subsumption vs. Consequence in FOL

We call a FOL conjunction or FOL clause **self-resolving** if it contains a positive literal and a negative literal both with the same predicate.

The relationship between $\subseteq_\theta$ and $\models$ for conjunctions (Theorem (??)) and clauses (Theorem (1)) is extended to FOL as follows:

Theorem 2

Let h, h' be FOL conjunctions. If $h \subseteq_\theta h'$ then $h' \models h$. Let furthermore h' not be self-resolving. Then $h \subseteq_\theta h'$ if and only if $h' \models h$.
Let h, h' be FOL clauses. If $h \subseteq_\theta h'$ then $h \models h'$. Let furthermore h not be self-resolving. Then $h \subseteq_\theta h'$ if and only if $h \models h'$.

Simple corollary: the theorem holds when $\subseteq_\theta$ is replaced by $\approx_\theta$ and $\models$ is replaced by $\models\!\!\models$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Example: Subsumption vs. Consequence for Conjunctions

- Both $h' \models h$ and $h \subseteq_\theta h'$:

$$h = \mathrm{vertex}(x) \wedge \mathrm{vertex}(y) \wedge \neg\mathrm{edge}(x, y)$$
$$h' = \mathrm{vertex}(x) \wedge \mathrm{vertex}(\mathrm{a}) \wedge \neg\mathrm{edge}(x, \mathrm{a})$$

- $h' \models h$ but $h \not\subseteq_\theta h'$ because h is self-resolving (exercise problem):

$$h = \mathrm{p}(x, y) \wedge \mathrm{p}(y, z) \wedge \neg\mathrm{p}(x, z)$$
$$h' = \mathrm{p}(\mathrm{a}, \mathrm{b}) \wedge \mathrm{p}(\mathrm{b}, \mathrm{c}) \wedge \mathrm{p}(\mathrm{c}, \mathrm{d}) \wedge \neg\mathrm{p}(\mathrm{a}, \mathrm{d})$$

- Similarly for clauses (just negating the h, h' above): $h \models h'$ but $h \not\subseteq_\theta h'$ because h is self-resolving:

$$h = \mathrm{p}(x, y) \wedge \mathrm{p}(y, z) \rightarrow \mathrm{p}(x, z)$$
$$h' = \mathrm{p}(\mathrm{a}, \mathrm{b}) \wedge \mathrm{p}(\mathrm{b}, \mathrm{c}) \wedge \mathrm{p}(\mathrm{c}, \mathrm{d}) \rightarrow \mathrm{p}(\mathrm{a}, \mathrm{d})$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Least General Generalization in FOL

The definition of propositional least general generalization is extended to FOL simply by replacing the respective symbols $\subseteq, \subset$ by $\subseteq_\theta, \subset_\theta$.

For a pair of FOL conjunctions (FOL clauses, respectively) h, h', we say that g is a **least general generalization** of h and h' if $g \subseteq_\theta h$, $g \subseteq_\theta h'$, and there is no FOL conjunction (FOL clause) g' such that $g' \subseteq_\theta h$, $g' \subseteq_\theta h'$ and $g \subset_\theta g'$.

Example: $\mathrm{p}(x)$ is a least general generalization of clauses $\mathrm{p}(\mathrm{a})$ and $\mathrm{p}(\mathrm{b}) \vee \mathrm{p}(\mathrm{c})$. Other least general generalizations of the same literals include e.g. $\mathrm{p}(y)$ or $\mathrm{p}(x) \vee \mathrm{p}(y)$.

Let g, g' be two different least general generalizations of h and h'. Then $g \approx_\theta g'$. (*Exercise problem*)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

lgg in FOL

Unlike in propositional logic, where the least general generalization is just set intersection, in FOL the computation is more involved.

We will develop the lgg operator which yields a least general generalization for FOL conjunctions or FOL clauses and reduces to set intersection in the propositional case.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

lgg of FOL terms

For FOL terms t_1, t_2:

$$\mathrm{lgg}(t_1, t_2) = \begin{cases} s\left(\mathrm{lgg}(u_1, v_1), \mathrm{lgg}(u_2, v_2), \ldots, \mathrm{lgg}(u_a, v_a)\right) \\ \quad \text{if } t_1 = s(u_1, u_2, \ldots u_n) \text{ and } t_2 = s(v_1, v_2, \ldots v_n), \\ \quad \text{where } n \in \{0\} \cup \mathbb{N} \text{ and } s \text{ is a function symbol.} \\ v \text{ if } t_1, t_2 \text{ have already been matched to variable } v \\ \text{new variable otherwise} \end{cases}$$

Note that the definition above applies also to FOL constants since a constant is a function of arity 0.

Examples:

$$\mathrm{lgg}(\mathrm{john}, \mathrm{mary}) = x_1$$
$$\mathrm{lgg}\left(\mathrm{father\_of}(\mathrm{john}), \mathrm{father\_of}(\mathrm{mary})\right) = \mathrm{father\_of}(x_1)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

lgg of FOL Literals

For FOL literals $\mathcal{L}_1, \mathcal{L}_2$:

$$\text{lgg}(\mathcal{L}_1, \mathcal{L}_2) = \begin{cases} p\left(\underline{\text{lgg}(u_1, v_1)}, \underline{\text{lgg}(u_2, v_2)}, \ldots, \underline{\text{lgg}(u_a, v_a)}\right) \\ \quad \text{if } \mathcal{L}_1 = p(u_1, u_2, \ldots u_n) \text{ and } \mathcal{L}_2 = p(v_1, v_2, \ldots v_n), \\ \quad \text{where } n \in \{0\} \cup \mathbb{N} \text{ and } p \text{ is a signed (negated or not) predicate symbol.} \\ \\ \text{undefined otherwise} \end{cases}$$

Examples:

$$\text{lgg}(\neg\text{p}, \neg\text{p}) = \neg\text{p}$$

$$\text{lgg}(\text{p}, \text{q}) \text{ undefined}$$

$$\text{lgg}\left(\begin{array}{c} \text{met}\left(\text{father\_of}(\text{john}), \text{mother\_of}(\text{john})\right), \\ \text{met}\left(\text{father\_of}(\text{mary}), \text{mother\_of}(\text{mary})\right) \end{array}\right) = \text{met}\left(\text{father\_of}(\text{x}), \text{mother\_of}(\text{x})\right)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

lgg of FOL Literals in Pseudo-Code

The lgg of two literals is computed by the Anti-unification algorithm, which rewrites both of the input literals a, b into $\text{lgg}(a, b)$.

Require: Literals a, b with same sign, symbol and arity, and not sharing a variable (if they share a variable, replace all occurrences of it in a with a new variable)

$i = 0$; $\theta := \emptyset$; $\sigma := \emptyset$ ▷ *a counter and two substitutions*
$v_1, v_2, \ldots$: variables not appearing in a or b
while $a \neq b$ **do**
    Let p be the leftmost position where a and b differ and s and t be the terms at this position in a and b, respectively.
    if for some j $(1 \leq j \leq i)$, $v_j\theta = s$ and $v_j\sigma = t$ **then** ▷ *variable already assigned*
        put v_j to position p in both a and b ▷ *replace the terms with that variable*
    else
        $i := i + 1$
        put v_i to position p in both a and b ▷ *replace the terms with a new variable*
        $\theta := \theta \cup \{ v_i \mapsto s \}$, $\sigma := \sigma \cup \{ v_i \mapsto t \}$ ▷ *store assignment of* v_i
    end if
end while
return a

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

lgg of FOL Conjunctions or Clauses

For FOL conjunctions (clauses, respectively) h, h', define $\text{lgg}(h, h')$ as the conjunction (disjunction) of

$$\text{lgg}(\mathcal{L}_i, \mathcal{L}_j) \tag{5}$$

for all $\mathcal{L}_i \in h$, $\mathcal{L}_j \in h'$ such that (5) is defined.

Theorem 3

Let h, h' be two FOL clauses or two FOL conjunctions. Then $\text{lgg}(h, h')$ is a least general generalization of h and h'.

(proof - deals with clauses but the case for conjunctions is analogical.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE