SMU ILP: Learning from Entailment Cont', Propositionalization

This tutorial consists of two parts. Firstly, an extension of lgg to the setting with background knowledge is presented. Secondly, some ideas of connecting ILP with attribute-value machine learning methods are given.

Relative Least General Generalization

In some cases, the agent may posses some additional information about the world, a background theory B. The learning task then changes to searching a hypothesis h which entails just positive observations with the background theory, i.e.

$$B \cup \{h\} \vdash o \; \forall o \in O^+,$$
$$B \cup \{h\} \nvdash o \; \forall o \in O^-.$$

We will restrict ourselves to a background theory B such that it is a finite set of ground facts, e.g. $B = \{human(adam), human(eva)\}$. In general, the theory can contain non-ground first-order rules, but this out of the scope of this course.

Before jumping to the lgg enriched by a domain theory, we have do define few things[1], but most of them are analogical to what you have seen in the last tutorial. Firstly, note the fact that if $\gamma_1 \subseteq_\theta \gamma_2$, then $\gamma_1\theta \implies \gamma_2$ is a tautology. The core idea here is to use θ-subsumption with respect to a background theory B, that is, $\gamma_1 \subseteq_\theta^B \gamma_2$ if there exists a substitution θ such that $B \vdash (\gamma_1\theta \implies \gamma_2)$. While having defined $\subseteq_\theta^B$, the definitions of θ-subsume equivalence with respect to B, $\approx_\theta^B$, strict θ-substumption w.r.t. B, etc., are analogical to θ-subsumption ones.

Now, we will define the *relative least general generalization* with respect to a background theory B of two clauses γ_1 and γ_2, which is given by

[1] You've seen it at the lecture [1].

$$
\begin{aligned}
RLGG_B(\gamma_1, \gamma_2) = \gamma \\
s.t. \quad & \gamma \subseteq_\theta^B \gamma_1 \\
& \gamma \subseteq_\theta^B \gamma_2 \\
& \forall \gamma' s.t. \gamma' \subseteq_\theta^B \gamma_1 \wedge \gamma' \subseteq_\theta^B \gamma_2 : \gamma' \subseteq_\theta^B \gamma
\end{aligned}
$$

Note that the only difference with lgg is replacing $\subseteq_\theta$ by $\subseteq_\theta^B$. The last step here is to formally define the computation of rlgg, which can be processed as follows:

$$RLGG_B = LGG(\gamma_1 \vee_{l \in B} \neg l, \gamma_2 \vee_{l \in B} \neg l).$$

Consider $\gamma_1 = p(X)$, $\gamma_2 = q(X)$ and $B = \{r(a,b), r(a,a)\}$, rlgg of these two is

$$\neg r(a,b) \vee \neg r(a,a) \vee \neg r(a, X_{a,b}) \vee \neg r(a, X_{b,a}).$$

One can easily see that γ_1 and γ_2 do not have any lgg, yet rlgg products a non-empty disjunction of literals. This simply follows from the fact that the ground literals are redundant w.r.t. B, they follow from the B, thus they can be deleted from the resulted lgg. So, we may rewrite the definition of rlgg as follows:

$$RLGG_B = LGG(\gamma_1 \impliedby B, \gamma_2 \impliedby B) \setminus \{\neg l : l \in B\}.$$

The final step of rlgg is to compute the reduced clause of the produced lgg after removing B.

Example: Compute rlgg of γ_1 and γ_2 w.r.t. B:

- $\gamma_1 = moveable(a)$
- $\gamma_2 = moveable(b)$
- $B = \{on(a,c), on(c,d), on(b,d), shape(a,box), shape(b,box), shape(c,box), shape(d,floor)\}$

Propositionalization

In this part of the tutorial, we take a huge step aside from the generalizing agent, but we will still use the expressivity of FOL. One of the possibilities of using FOL with connection to the attribute-value machine learning approaches is propositionalization. Its idea is to transform (multi-)relational data to attribute-value data, which can be thereafter used to feed standard machine learning algorithsm. In spirit, bag-of-words representation, as used in NLP, is a special case of it. So, given a list of first order queries and an interpretation, it products a boolean vector in which each single position i corresponds to whether a query i

from the list covers the interpretation; this is called *existential propositionalization*. There are multiple types of propositionalization, e.g. *counting* one which produces number of grounding for a query in which the query holds on a given interpretation.

You may try to play with this using TreeLiker. Briefly, there are those definitions and constraints on the language bias:

1. + / - marks input and output argument respectively
2. # marks a constant
3. ! marks an ignored variable
4. In order a template to be valid, any variable must appear exactly once as an output and at least once as an input.
5. Any literal in template can contain at most one input argument.
6. A valid template does not contain a cycle among types.

References

[1] Filip Železný and Jiří Kléma. *SMU textbook*. 2017.