First-Order Logic and θ-Subsumption

B4M36SMU

In this tutorial, we will start with a light refresh of the first-order logic basics required for this course. Further, we will talk about θ-subsumption, which lies in the heart of an upcoming Inductive Logic Programming (ILP) algorithm that fills this part of this course. Finally, a clause reduction algorithm based on θ-subsumption is presented.

Motivation

Before diving into technical details, let us take a step back and look at what the motivation for combining first-order logic (FOL) with machine learning is. Specifically, is finite vector representation expressive enough to represent all kinds of problems? Consider the following domains:

- graphs with the variable number of nodes, e.g. molecules
- relationship representation, e.g. taxonomies
- structured data, e.g. parse tree of a sentence, databases, ...
- model's interpretability

Most likely, many of you have already met with *attribute-value* realm of machine learning and corresponding algorithms, e.g. SVM, NN, ... These approaches assume a fixed-size input vector. In our scenario, however, we are given learning examples that may vary in their sizes. For example, the task is to classify the explosiveness of a molecule represented by a graph.

Yet, one may say that nothing is lost and an attribute-value machine learning algorithm may be used using feature engineering. That is transforming a graph (molecule) into a corresponding (feature) vector representation; for example, f_1 is the number of carbons in the molecule, f_2 the number of double bonds within the molecule, and finally, f_3 is the number of benzene cores. This approach, however, is incapable of learning structural patterns that go beyond the fixed features. Another way of using classical attribute-value methods requires learning of embedding of samples firstly.

Despite these possible approaches, the task may be different from graph classification, e.g. including hypergraphs and structured output (knowledge graph completion). Therefore we will study part of ILP in this part of the course.

Last but not least, having a rule-based machine learning method brings additional points for interpretability, which is an active field of researches' interests. However, how many first-order clauses are you able to manually inspect to interpret a model?

First-Order Logic (A Very Informal Refresher)

This section contains basic definitions of elements of first-order predicate logic needed for this tutorial.

Syntax

A term is either a constant, variable, or a compound term. A constant starts with an upper case, e.g. *Adam*. All other elements of the first-order logic language start with lower case, e.g. a variable x. A compound term is a function symbol, e.g. $fatherOf/1$, applied on an a-tuple of terms where a is the arity of the function symbol, e.g. $fatherOf(Adam)$. Predicate symbols express relations among a terms, e.g. $sibling/2$ is a predicaty of arity 2. An atom is a predicate symbol of arity a applied to an a-tuple of terms, e.g. $sibling(x, Adam)$. A literal is an atom or its negation, e.g. $\neg sibling(x, Adam)$. Formulae are constructed from literals using logical connectives ($\wedge$, $\vee$, $\implies$,...) and quantifiers ($\forall$, $\exists$). A *ground* term, *ground* literal, *ground* clause and *ground* do not contain any variables. A clause is a

universally quantified disjunction of literals, e.g. $\forall x, y : \neg sibling(Adam, x) \vee sibling(x, y)$. For brevity, the quantifier part of a clause is sometimes not written explicitly when it is clear from the context. A formula is said to be in *conjunctive normal form* (*CNF*) if it is a conjunction of disjunctions. For instance $(\forall x, y : sibling(x, y) \vee \neg human(Eva) \vee \neg human(Adam)) \wedge (\forall x : \neg human(x) \vee mortal(x))$ is in CNF. A quantifier bounds scope of a variable. If a quantifier is left out (because it is clear from the context), then a variable's scope is limited only to the clause within which it is appearing.

Substitution is an assignment of **terms to variables**, e.g. $\theta = \{x \mapsto A, y \mapsto f(A, z)\}$. By applying a substitution to a clause, literal, or term, variables in the respective clause, literal, or term are replaced by their images in the substitution. For instance, consider the previous θ and $L = l(x, y, w)$. Then we have $L\theta = l(A, f(A, z), w)$. If $L\theta$ is ground, we say that the substitution θ is a *grounding substitution*.

Given two terms or atoms L_1 and L_2, the unification problem is to find a substitution θ such that $L_1\theta = L_2\theta$. Note that unification of two expressions does not always exist, e.g. there is no unification of $L_1 = p(x)$ and $L_2 = q(A)$.

Clauses can be split into several categories according to some properties, e.g. number of positive literals. Here, we recall the basic ones. Horn clause is a clause with at most one positive literal. Definite clause is a horn clause with exactly one positive literal, e.g. $\neg l_1 \vee \neg l_2 \vee \neg l_3 \vee l_4$. We call positive literals a clause *head*, i.e. l_4; rest of the clause (negative literals) are called *body*, i.e. $\neg l_1 \vee \neg l_2 \vee \neg l_3$. A clause is *range-restricted* if variables occurring in head also occur in body.

A note on the used notation. We use the notation in which constants start with uppercase letters while the rest (e.g. variables) starts with lowercase letters.

Semantics

An interpretation (also sometimes called *possible world*) defines which atoms are true and which are false. An interpretation o is a model of clause γ ($o \models \gamma$) iff γ is true in the interpretation. The previous definition of an interpretation is rather an informal one. In order to do it formally, we would have to define a universe $\mathcal{U}$ and a mapping which maps each constant to an element of $\mathcal{U}$; for each predicate of arity a, it defines upon which a-tuple of elements $\mathcal{U}$ it holds, etc. However, we do not need such formal formulation, because it is sufficient for us to use *Herbrand interpretations* only. In Herbrand interpretations, each symbol maps to itself, i.e. it uses *Herbrand universe* and part of the interpretation's mapping is given, e.g. constant *Adam* maps to *Adam*. *Herbrand universe* consists of all ground terms which can be composed constant and function symbols. *Herbrand base* is a set of all ground atoms which can be formed out of predicate symbols and terms from Herbrand universe.

For more information see [1, 2].

Exercise

- What is the difference between a term and an atom, e.g. $fatherOf(Adam)$ and $human(Adam)$?
- Given theory $\Phi = \{\forall x, y : \neg edge(x, y) \vee edge(y, x), \forall x : \neg edge(x, x)\}$ find two Herbrand interpretations, from which the first one is a model of Φ and the second is not.
- Unify $\gamma_1 = edge(x, y)$ with $\gamma_2 = edge(C, D)$.
- Unify $\gamma_1 = edge(x, y)$ with $\gamma_2 = edge(y, x)$.
- Unify $\gamma_1 = edge(x, C)$ with $\gamma_2 = edge(A, B)$.
- Unify $\gamma_1 = path(x, f(x))$ with $\gamma_2 = path(g(y), y)$.
- Given $\gamma = r(x, y, z, A)$ and Herbrand universe $\{A, B, f(A, B)\}$, how many grounding substitutions there are?

Solution:

- In an interpretation, a term maps to an element of the universe while an atom maps to a truth value.
- Firstly, let us start with rewriting of the clauses within Φ. The first clause can be rewritten as $edge(x, y) \implies edge(y, x)$, so it basically forces symmetry of the *edge* relation. The second clause is a constraint forbidding the relation to be reflexive. Interpreting the $edge/2$ relation as a directed edge from the first node to the second, the clauses tell that only graphs with bi-directly bounded nodes, having no loops, can be models of the theory Φ. So, $\{node(1), node(2), edge(1, 2), edge(2, 1)\} \models \Phi$ because all clauses are satisfied, while $\{node(1), edge(1, 1)\} \not\models \Phi$ because the loop-forbidding clause is violated. Sure, $\{\} \models \Phi$ as all Φ's clauses are satisfied.
- $\theta = \{x \mapsto C, y \mapsto D\}$
- Recall that scope of a variable is restricted to a clause, therefore we may rewrite the clauses as $\gamma_1 = edge(x, y)$ and $\gamma_2 = edge(w, z)$. Finally, there are many of unifying substitutions, for example, the following ones: $\theta_1 = \{x \mapsto w, y \mapsto z\}$, $\theta_2 = \{w \mapsto x, z \mapsto y\}$, $\theta_2 = \{x \mapsto w, z \mapsto y\}$, $\theta_3 = \{w \mapsto z, y \mapsto z\}$, or even $\theta_4 = \{x \mapsto x_1, y \mapsto x_1, w \mapsto x_1, z \mapsto x_1\}$.
- There is none.
- There is none.
- There are 3^3 grounding substitutions.

θ-subsumption

One of the core concepts of ILP is θ-subsumption. We say that γ_1 *θ-subsumes* γ_2, $\gamma_1 \subseteq_\theta \gamma_2$, iff there exists substitution θ such that $\gamma_1\theta \subseteq \gamma_2$ in the sense of literals. For example, for $\gamma_1 = p(x, y)$ and $\gamma_2 = p(A, z)$ it holds that $\gamma_1 \subseteq_\theta \gamma_2$. To check this by the definition, firstly find a substitution, e.g. $\theta = \{x \mapsto A, y \mapsto z\}$, and check the sets of literals, e.g. $\gamma_1\theta = \{p(A, z)\} \subseteq \{p(A, z)\} = \gamma_2$. Note here that the $=$ relation is a little bit overloaded, meaning that a clause is the same as a set of literals it contains. On the contrary, $\gamma_2 \not\subseteq_\theta \gamma_1$. In general, however, it may happen that two clauses are *subsume-equivalent*, i.e. $\gamma_i \approx_\theta \gamma_j$ iff $\gamma_i \subseteq_\theta \gamma_j$ and $\gamma_j \subseteq_\theta \gamma_i$.

Exercise

List all pairs s.t. $\gamma_1 \subseteq_\theta \gamma_2$ from the following clauses:

(a) $p(x)$

(b) $q(y) \vee p(y)$

(c) $p(A) \vee p(B) \vee p(z)$

(d) $q(A)$

(e) $q(x) \vee p(z) \vee q(z)$

Solution:

- $(a) \subseteq_\theta (b) : p(x) \subseteq_\theta q(y) \vee p(y)$
- $(a) \subseteq_\theta (c) : p(x) \subseteq_\theta p(A) \vee p(B) \vee p(z)$, three possible ways $\theta_1 = \{x \mapsto A\}$, $\theta_2 = \{x \mapsto B\}$, $\theta_3 = \{x \mapsto z\}$
- $(a) \subseteq_\theta (e) : p(x) \subseteq_\theta q(x) \vee p(z) \vee q(z)$
- $(b) \subseteq_\theta (e) : q(y) \vee p(y) \subseteq_\theta q(x) \vee p(z) \vee q(z)$
- $(e) \subseteq_\theta (b) : q(x) \vee p(z) \vee q(z) \subseteq_\theta q(y) \vee p(y)$

Clause Reduction

We will conclude this tutorial by learning an algorithm based on θ-subsumption. From the previous section we know *θ-subsumption* and *subsume-equivalence* relations. We also define *strict θ-subsumption* which expresses that one given clause θ-subsumes another given clause, but not the other way around; i.e. we say that γ_1 *strictly θ-subsumes* γ_2, $\gamma_1 \subset_\theta \gamma_2$, if $\gamma_1 \subseteq_\theta \gamma_2 \wedge \gamma_2 \not\subseteq_\theta \gamma_1$.

Using these relations we can define equivalence classes of clauses: all subsume equivalent clauses are in one class for which we select a shortest one to be their representative (all of these shortest clauses are the same up to renaming of variables – i.e. up to *isomorphism*). This, in turn, allows us to traverse the space of clauses more efficiently. We say that a clause γ is reduced if there is no γ', $\gamma' \subsetneq \gamma$ such that $\gamma' \approx_\theta \gamma$. In other words, the reduced clause of a clause γ' is in the same subsume-equivalence class as γ, but has the smallest number of literals.

If, after deleting one of its literals, a clause γ is in the same subsume-equivalence class, then the shorter clause can be used instead of the original one. We can compute a reduced clause by applying this strategy iteratively on every literal of the original clause. See the following pseudocode of the algorithm which expects a clause γ as input and returns its reduction:

$\gamma' \leftarrow \gamma$
forall $l \in \gamma$ **do**
  if $\gamma \subseteq_\theta (\gamma' \setminus \{l\})$ **then**
    $\gamma' \leftarrow \gamma' \setminus \{l\}$
  end
end
return γ'

Exercise

- Find a reduction of the clause: $\forall x, y, z\colon p(x, y) \vee p(A, B) \vee m(z) \vee m(y)$
- Find a reduction of the clause: $\forall x, y, z\colon p(x, y) \vee p(A, z) \vee m(z) \vee m(y)$

Solution:
The reduction of $\forall x, y, z\colon p(x, y) \vee p(A, B) \vee m(z) \vee m(y)$ is $p(x, y) \vee p(A, B) \vee m(y)$.

- Literal $p(x, y)$ can't be removed since $p(x, y) \vee p(A, B) \vee m(z) \vee m(y) \not\subseteq_\theta p(A, B) \vee m(z) \vee m(y)$. $p(x, y)$ would have to map to $p(A, B)$, i.e. using $\theta = \{x \mapsto A, y \mapsto B\}$, but the subsumption relation does not hold after applying the substitution, i.e. $\{p(A, B), m(z), m(B)\} \not\subseteq \{p(A, B), m(z), m(y)\}$.
- Literal $p(A, B)$ can't be removed from the clause. You may have already noticed that removing ground literals is not possible during clause reduction.
- Literal $m(z)$ can be removed. The clause obtained by applying the substitution $\theta = \{z \mapsto y\}$ on the original clause subsumes the shorter one. Therefore we store the shorter clause to γ'.
- Literal $m(y)$ can't be removed. After removing the literal from the previously obtained clause, i.e. $\gamma' \setminus \{m(y)\}$, there is no literal with $m/1$ predicate, thus it can't be subsumed by the original clause.

The reduction of $\forall x, y, z\colon p(x, y) \vee p(A, z) \vee m(z) \vee m(y)$ is $p(A, z) \vee m(z)$.

- Literal $p(x, y)$ can be removed. After applying the substitution $\theta = \{x \mapsto A, y \mapsto z\}$ on the original clause, we obtain $p(A, z) \vee m(z)$ which subsumes the shorter one ($p(A, z) \vee m(z) \vee m(y)$). We update γ', $\gamma' = p(A, z) \vee m(z) \vee m(y)$.
- Literal $p(A, z)$ can't be removed.
- Literal $m(z)$ can't be removed. The only possible substitution is $\theta = \{z \mapsto y\}$, but applying it on the original clause does not produce a subset of the γ'.
- Literal $m(y)$ can be removed. We could actually do it during the removal of $p(x, y)$ using the same substitution.

References

[1] Luc De Raedt. *Logical and relational learning.* Springer Science & Business Media, 2008.

[2] Shan-Hwei Nienhuys-Cheng and Ronald De Wolf. *Foundations of inductive logic programming.* Vol. 1228. Springer Science & Business Media, 1997.