EMPOWERED NEGATIVE SPECIALIZATION IN INDUCTIVE LOGIC PROGRAMMING

Stefano Ferilli, Andrea Pazienza, and Floriana Esposito

Friday 25th September 2015
Inductive Logic Programming aims at learning concepts from examples.

Two refinement operators:

- **generalization**: refines hypothesis that does not account for a positive example,
- **specialization**: refines hypothesis that erroneously accounts for a negative example.

The addition of **negative information** may allow to learn a broader range of concepts.
Specialization Operator

**Specializing**: adding proper literals to a clause that is **inconsistent** w. r. t. a negative example, in order to avoid its covering that example.

**Residual** $\Delta_j(E, C)$ of an Example $E$ w. r. t. a Clause $C$: all the literals in the example that are not involved in the $\theta_{OI}$-subsumption test, after the antisubstitution phase.

The space of single **consistent negative downward refinements** does not ensure **completeness** w. r. t. the previous positive examples.
Example: Edible Mushrooms

A MUSHROOM m is described by the following features: a stem s, a cap c, spores p, gills g, dots d.

- **Positive examples:** \( P_1 = m \leftarrow s, c, p, g. \) \( P_2 = m \leftarrow s, c, d. \)
- **Least General Generalization:** \( C_1 = m \leftarrow s, c. \)
- **Negative example:** \( N_1 = m \leftarrow s, c, p, g, d. \)
- **Residuals:**
  \[ \Delta_1(P_1, C_1) = \{p, g\} \] \[ \Delta_2(P_2, C_1) = \{d\} \] \[ \Delta_3(N_1, C_1) = \{p, g, d\}. \]

**Correct refinements** of \( C_1 \) could be:

\[ C'_2 = m \leftarrow s, c, \neg(p, d). \] \( C''_2 = m \leftarrow s, c, \neg(g, d). \)

So, we might invent a new predicate \( n \), defined as

\[ n \leftarrow p, d. \] \( n \leftarrow g, d. \]

and specialize \( C_1 \) in \( C'_1 = m \leftarrow s, c, \neg n. \)

I.e., an edible mushroom must not have both spores and dots.
**Challenge:** determine a **minimal** subset of the **negative** residual.

The search space is represented as a **binary tree**. To restrict the search space:

- Each vertex is a **candidate definition**.
- The number of literals decreases as the depth of the vertex increases.
- Derive two subsets from the whole negative residual, based on a **pair of literals** in it.
- The **tree levels** are explored until the 2-literal level is reached.
- If any of the vertexes is able to restore consistency, the level immediately above is scanned, and so on until a suitable set of literals is found, or the specialization fails.
Consider a hypothesis: \( C = h :: a, b \)., four positive examples:

\[
P_1 = h :: a, b, c, d, e.
\]

\[
P_2 = h :: a, b, e, f, g.
\]

\[
P_3 = h :: a, b, c, e, f.
\]

\[
P_4 = h :: a, b, c, d, f, g.
\]

and a negative one: \( N = h :: a, b, c, d, e, f, g. \)

**No two-literal solutions exist**

---

**Invented predicate:** \( n :: c, f, g. \)

**Specialize** \( C \) in \( C' = h :: a, b, \neg n. \)
Evaluation

Figure: Runtime (y-axis) by number of literals in the residuals and number of examples for each setting (x-axis).