Rule Induction

- **Given:** Set of positive and negative examples of some concept
- **Example:** \((x_1, x_2, \ldots, x_n, y)\)
- **\(y\):** concept (Boolean)
- **\(x_1, x_2, \ldots, x_n\):** attributes (assume Boolean)
- **Goal:** Induce a set of rules that cover all positive examples and no negative ones
  - **Rule:** \(x_a \land x_b \land \ldots \Rightarrow y\) (\(x_i\): Literal, i.e., \(x_i\) or its negation)
  - Same as *Horn clause*: Body \(\Rightarrow\) Head
  - Rule \(r\) covers example \(x\) iff \(x\) satisfies body of \(r\)
- **\(\text{Eval}(r)\):** Accuracy, info gain, coverage, support, etc.

---

**Learning a Single Rule**

\[
\begin{align*}
\text{head} & \leftarrow y \\
\text{body} & \leftarrow \emptyset \\
\text{repeat} & \\
& \text{for each} \ \text{literal} \ x \\
& \quad r_x \leftarrow r \text{ with } x \text{ added to body} \\
& \quad \text{Eval}(r_x) \\
& \quad \text{body} \leftarrow \text{body} \land \text{best } x \\
& \text{until no } x \text{ improves } \text{Eval}(r) \\
\text{return } r
\end{align*}
\]

**Learning a Set of Rules**

\[
\begin{align*}
R & \leftarrow \emptyset \\
S & \leftarrow \text{examples} \\
\text{repeat} & \\
& \text{learn a single rule } r \\
& \quad R \leftarrow R \cup \{r\} \\
& \quad S \leftarrow S - \text{positive examples covered by } r \\
& \text{until } S = \emptyset \\
\text{return } R
\end{align*}
\]
First-Order Rule Induction (a.k.a. Inductive Logic Programming)

- $y$ and $x_i$ are now predicates with arguments
  - E.g.: $y$ is $\text{Ancestor}(x,y)$, $x_i$ is $\text{Parent}(x,y)$
- Literal to add are predicates or their negations
- Literal to add must include at least one variable already appearing in rule
- Adding a literal changes # groundings of rule
  - E.g.: $\text{Ancestor}(x,z) \land \text{Parent}(z,y) \Rightarrow \text{Ancestor}(x,y)$
- Eval($r$) must take this into account
  - E.g.: Multiply by # positive groundings of rule still covered after adding literal

MLN Structure Learning

- Generalizes feature induction in Markov nets
- Any inductive logic programming approach can be used, but . . .
- Goal is to induce any clauses, not just Horn
- Evaluation function should be likelihood
- Requires learning weights for each candidate
- Turns out not to be bottleneck
- Bottleneck is counting clause groundings
- Solution: Subsampling

MLN Structure Learning

- Initial state: Unit clauses or hand-coded KB
- Operators: Add/remove literal, flip sign
- Evaluation function:
  - Pseudo-likelihood + Structure prior
- Search: Beam search, shortest-first search