Lecture 6: Inductive Logic Programming

Cognitive Systems II - Machine Learning

SS 2005

Part II: Special Aspects of Concept Learning

FOIL, Inverted Resolution, Sequential Covering
Motivation

- it is useful to learn the target function as a set of if-then-rules
- one of the most expressive and human readable representations
- e.g. decision trees

Inductive Logic Programming (ILP):
- rules are learned directly
- designed to learn first-order rules (i.e. including variables)
- sequential covering to incrementally grow the final set of rules

- PROLOG programs are sets of first-order rules

⇒ a general-purpose method capable of learning such rule sets may be viewed as an algorithm for automatically inferring PROLOG programs
Sequential Covering

**basic strategy:** learn one rule, remove the data it covers, then iterate this process

one of the most widespread approaches to learn a disjunctive set of rules (each rule itself is conjunctive)

**subroutine** LEARN-ONE-RULE

- accepts a set of positive and negative examples as input and outputs a single rule that covers many of the positive and few of the negative examples
- **high accuracy:** predictions should be correct
- **low coverage:** not necessarily predictions for each example

performs a greedy search without backtracking

⇒ no guarantee to find the smallest or best set of rules
Sequential Covering Algorithm

\[\text{SEQUENTIAL-COVERING}(\text{Target\_attribute}, \text{Attributes}, \text{Examples}, \text{Threshold})\]

\begin{itemize}
  \item \(\text{Learned\_Rules} \leftarrow \{\}\)
  \item \(\text{Rule} \leftarrow \text{LEARN-ONE-RULE}(\text{Target\_attribute}, \text{Attributes}, \text{Examples})\)
  \item While \(\text{PERFORMANCE}(\text{Rule}, \text{Examples}) > \text{Threshold}\), Do
    \begin{itemize}
      \item \(\text{Learned\_rules} \leftarrow \text{Learned\_rules} + \text{Rule}\)
      \item \(\text{Examples} \leftarrow \text{Examples} - \{ \text{examples correctly classified by Rule} \}\)
      \item \(\text{Rule} \leftarrow \text{LEARN-ONE-RULE}(\text{Target\_attribute}, \text{Attributes}, \text{Examples})\)
    \end{itemize}
  \item \(\text{Learned\_rules} \leftarrow \text{sort Learned\_rules accord to PERFORMANCE over Examples}\)
  \item return \(\text{Learned\_rules}\)
\end{itemize}
question: How shall LEARN-ONE-RULE be designed to meet the needs of the sequential covering algorithm?

organize the search through $H$ analogous to ID3

but follow only the most promising branch in the tree at each step

begin by considering the most general rule precondition (i.e. empty test)

then greedily add the attribute test that most improves rule performance over the training examples

unlike to ID3, this implementation follows only a single descendant at each search step rather than growing a subtree that covers all possible values of the selected attribute
General to Specific Beam Search

IF Wind=weak THEN PlayTennis=yes

IF Wind=strong THEN PlayTennis=no

IF Humidity=normal THEN PlayTennis=yes

IF Humidity=high THEN PlayTennis=no

IF Humidity=normal Wind=weak THEN PlayTennis=yes

IF Humidity=normal Wind=strong THEN PlayTennis=yes

IF Humidity=normal Outlook=sunny THEN PlayTennis=yes

IF Humidity=normal Outlook=rain THEN PlayTennis=yes
so far a local greedy search (analogous to hill-climbing) is employed

- danger of suboptimal results
- susceptible to the typical hill-climbing problems

⇒ extension to beam search

- algorithm maintains a list of the $k$ best candidates at each step
- at each step, descendants are generated for each of the $k$ candidates and the resulting set is again reduced to the $k$ most promising candidates
**LEARN-ONE-RULE**

**LEARN-ONE-RULE**(*Target_attribute*, *Attributes*, *Examples*, *k*)

Returns a single rule that covers some of the Examples. Conducts a general to specific greedy beam search for the best rule, guided by the PERFORMANCE metric.

- Initialize *Best_hypothesis* to the most general hypothesis $\emptyset$
- Initialize *Candidate_hypotheses* to the set \{*Best_hypothesis*\}

While *Candidate_hypotheses* is not empty, Do

1. Generate the next more specific *candidate_hypotheses*
   - *New_Candidate_hypotheses* ← new generated and specialized candidates
2. Update *Best_hypotheses*
   - *Best_hypothesis* ← $h$ with best PERFORMANCE
3. Update *Candidate_hypotheses*
   - *Candidate_hypotheses* ← the $k$ best members of *New_Candidate_hypotheses*

Return a rule of the form

“IF *Best_hypothesis* THEN prediction”

where *prediction* is the most frequent value of *Target_attribute* among those *Examples* that match *Best_hypothesis*. 
**Sequential vs. Simultaneous Covering**

- **Sequential Covering:**
  - Learn just one rule at a time, remove the covered examples and repeat the process on the remaining examples.
  - Many search steps, making independent decisions to select each precondition for each rule.

- **Simultaneous Covering:**
  - ID3 learns the entire set of disjunct rules simultaneously as part of a single search for a decision tree.
  - Fewer search steps, because each choice influences the preconditions of all rules.

⇒ Choice depends on how much data is available.
- Plentiful → Sequential covering (more steps supported)
- Scarce → Simultaneous covering (decision sharing effective)
Differences in Search

**generate-then-test:**
- search through all syntactically legal hypotheses
- generation of the successor hypotheses is only based on the syntax of the hypothesis representation
- training data is considered after generation to choose among the candidate hypotheses
- each training example is considered several times
  ⇒ impact of noisy data is minimized

**example driven:**
- individual training examples constrain the generation of hypotheses
- e.g. FIND-S, CANDIDATE ELIMINATION
  ⇒ search can easily be misled
Learning First-Order Rules

- Propositional expressions do not contain variables and are therefore less expressive than first-order expressions.

- No general way to describe essential relations among the values of attributes.

- Now we consider learning first-order rules (Horn Theories).
  - A Horn clause is a clause containing at most one positive literal.

- Expression of the form:
  \[ H \lor \neg L_1 \lor \neg L_2 \lor \ldots \lor \neg L_n \]
  \[ \iff H \leftarrow (L_1 \land L_2 \land \ldots \land L_n) \]
  \[ \iff \text{IF} (L_1 \land L_2 \land \ldots \land L_n) \text{ THEN } H \]

- FOL terminology see CogSysI.
FOIL (Quinlan, 1990)

- natural extension of SEQUENTIAL-COVERING and LEARN-ONE-RULE

- outputs sets of first-order rules similar to Horn Clauses with two exceptions
  1. more restricted, because literals are not permitted to contain function symbols
  2. more expressive, because literals in the body can be negated

- differences between FOIL and earlier algorithms:
  - seeks only rules that predict when the target literal is True
  - conducts a simple hill-climbing search instead of beam search
FOIL algorithm

\[
\text{FOIL}(\text{Target\_predicate}, \text{Predicate}, \text{Examples})
\]

\[
\begin{align*}
\text{Pos} & \leftarrow \text{those Examples for which the Target\_predicate is True} \\
\text{Neg} & \leftarrow \text{those Examples for which the Target\_predicate is False} \\
\text{Learned\_rules} & \leftarrow \{\} \\
\text{while Pos, Do} & \\
& \text{NewRule} \leftarrow \text{the rule that predicts Target\_predicate with no precondition} \\
& \text{NewRuleNeg} \leftarrow \text{Neg} \\
& \text{while NewRuleNeg, Do} \\
& \text{Candidate\_literals} \leftarrow \text{generate new literals for NewRule, based on Predicates} \\
& \text{Best\_literal} \leftarrow \max_{L \in \text{Candidate\_literals}} \text{FoilGain}(L, \text{NewRule}) \\
& \text{add Best\_literal to preconditions of Rule} \\
& \text{NewRuleNeg} \leftarrow \text{subset of NewRuleNeg that satisfies NewRule preconditions} \\
& \text{Learned\_rules} \leftarrow \text{Learned\_rules} + \text{NewRule} \\
& \text{Pos} \leftarrow \text{Pos} - \{\text{members of Pos covered by NewRule}\} \\
& \text{Return Learned\_Rules}
\end{align*}
\]
FOIL Hypothesis Space

outer loop (set of rules):
- specific-to-general search
- initially, there are no rules, so that each example will be classified negative (most specific)
- each new rule raises the number of examples classified as positive (more general)
- disjunctive connection of rules

inner loop (preconditions for one rule):
- general-to-specific search
- initially, there are no preconditions, so that each example satisfied the rule (most general)
- each new precondition raises the number of examples classified as negative (more specific)
- conjunctive connection of preconditions
Generating Candidate Specializations

current rule:

\[ P(x_1, x_2, ..., x_k) \leftarrow L_1...L_n \]

where

\( L_1...L_n \) are the preconditions and

\( P(x_1, x_2, ..., x_k) \) is the head of the rule

FOIL generates candidate specializations by considering new literals \( L_{n+1} \) that fit one of the following forms:

- \( Q(v_1, ..., v_r) \) where \( Q \in Predicates \) and the \( v_i \) are new or already present variables (at least one \( v_i \) must already be present)

- \( Equal(x_j, x_k) \) where \( x_j \) and \( x_k \) are already present in the rule

- the negation of either of the above forms
Induction as Inverted Deduction

**Observation:** induction is just the inverse of deduction

In general, machine learning involves building theories that explain the observed data.

Given some data $D$ and some background knowledge $B$, learning can be described as generating a hypothesis $h$ that, together with $B$, explains $D$.

$$\forall <x_i, f(x_i)> \in D)(B \land h \land x_i) \models f(x_i)$$

The above equation casts the learning problem in the framework of deductive inference and formal logic.
Induction as Inverted Deduction

- **features of inverted deduction:**
  - subsumes the common definition of learning as finding some general concept
  - background knowledge allows a more rich definition of when a hypothesis $h$ is said to “fit” the data

- **practical difficulties:**
  - noisy data makes the logical framework completely lose the ability to distinguish between truth and falsehood
  - search is intractable
  - background knowledge often increases the complexity of $H$
Inverting Resolution

resolution is a general method for automated deduction

complete and sound method for deductive inference

see CogSys1

Inverse Resolution Operator (propositional form):

1. Given initial clause $C_1$ and $C$, find a literal $L$ that occurs in $C_1$ but not in clause $C$.

2. Form the second clause $C_2$ by including the following literals

$$C_2 = (C - (C_1 - \{L\})) \cup \{L\}$$

inverse resolution is not deterministic
Inverting Resolution

\[ C_2: \text{KnowMaterial} \lor \neg \text{Study} \]

\[ C_1: \text{PassExam} \lor \neg \text{KnowMaterial} \]

\[ C: \text{PassExam} \lor \neg \text{Study} \]
Inverting Resolution

**Inverse Resolution Operator (first-order form):**

- resolution rule:
  1. Find a literal $L_1$ from clause $C_1$, literal $L_2$ from clause $C_2$, and substitution $\theta$ such that $L_1 \theta = \neg L_2 \theta$
  2. Form the resolvent $C$ by including all literals from $C_1 \theta$ and $C_2 \theta$, except for $L_1 \theta$ and $\neg L_2 \theta$. That is,

$$C = (C_1 - \{L_1\}) \theta \cap (C_2 - \{L_2\}) \theta$$

- analytical derivation of the inverse resolution rule:

$$C = (C_1 - \{L_1\}) \theta_1 \cap (C_2 - \{L_2\}) \theta_2 \text{ where } \theta = \theta_1 \theta_2$$

$$C - (C_1 - \{L_1\}) \theta_1 = (C_2 - \{L_2\}) \theta_2 \text{ where } L_2 = \neg L_1 \theta_1 \theta_2^{-1}$$

$$\Rightarrow \quad C_2 = (C - (C_1 - \{L_1\}) \theta_1) \theta_2^{-1} \cap \{\neg L_1 \theta_1 \theta_2^{-1}\}$$
Inverting Resolution

\[ D = \{ \text{GrandChild}(Bob, Shannon) \} \]

\[ B = \{ \text{Father}(Shannon, Tom), \text{Father}(Tom, Bob) \} \]
Generalization, $\theta$-Subsumption, Entailment

- interesting to consider the relationship between the more_general_than relation and inverse entailment

- more_general_than: $h_j \geq h_k$ iff $(\forall x \in X)[h_k(x) \rightarrow h_j(x)]$. A hypothesis can also be expressed as $c(x) \leftarrow h(x)$.

- $\theta$ – subsumption: Consider two clauses $C_j$ and $C_k$, both of the form $H \lor L_1 \lor ... \lor L_n$, where $H$ is a positive literal and the $L_i$ are arbitrary literals. Clause $C_j$ is said to $\theta$ – subsume clause $C_k$ iff $(\exists \theta)[C_j \theta \subseteq C_k]$.

- Entailment: Consider two clauses $C_j$ and $C_k$. Clause $C_j$ is said to entail clause $C_k$ (written $C_j \vdash C_k$) iff $C_j$ follows deductively from $C_k$. 

Lecture 6: Inductive Logic Programming – p. 22
Generalization, $\theta$-Subsumption, Entailment

- if $h_1 \geq g h_2$ then $C_1 : c(x) \leftarrow h_1(x)$ $\theta$-subsumes $C_2 : c(x) \leftarrow h_2(x)$

- furthermore, $\theta$-subsumption can hold even when the clauses have different heads

  $A : \text{Mother}(x, y) \leftarrow \text{Father}(x, z) \land \text{Spouse}(z, y)$
  $B : \text{Mother}(x, L) \leftarrow \text{Father}(x, B) \land \text{Spouse}(B, y) \land \text{Female}(x)$

  $A\theta \subseteq B$ if we choose $\theta = \{y/L, z/B\}$

- $\theta$-subsumption is a special case of entailment

  $A : \text{Elephant}(\text{father}_\text{of}(x)) \leftarrow \text{Elephant}(x)$
  $B : \text{Elephant}(\text{father}_\text{of}(\text{father}_\text{of}(y))) \leftarrow \text{Elephant}(y)$

  $A \vdash B$, but $\neg \exists \theta[A\theta \subseteq B]$
Summary

- learns sets of first-order rules directly
- sequential covering algorithms learn just one rule at a time and perform many search steps
- hence, applicable if data is plentiful
- **FOIL** is a sequential covering algorithm
  - a specific-to-general search is performed to form the result set
  - a general-to-specific search is performed to form each new rule
- Induction can be viewed as the inverse of deduction
- hence, an inverse resolution operator can be found