Learning Sets of Rules

- Sequential covering algorithms
- FOIL
- Inductive Logic Programming

Slide CS478–1

Propositional vs. First-order Rules

Propositional (logic) rules do not contain any variables.
First-order (logic) rules can contain variables.

Name1:	Chelsea	*Name2:*	Bill	
Mother1:	Hillary	*Mother2:*	Virginia	
Father1:	Bill	*Father2:*	Bruno	$\Rightarrow$
Male1: :	False	*Male2:*	True	
Female1:	True	*Female2:*	False	

$Daughter_{1,2} = TRUE$

Slide CS478–2

A propositional representation could only learn the rule:

IF ($Father1$=Bill) $\wedge$ ($Name2$=Bill) $\wedge$ ($Female1$=True)
THEN $Daughter_{1,2} = TRUE$

A first-order representation could learn the rule:

IF $Father(x, y) \wedge Female(y)$ THEN $Daughter(y, x)$

Slide CS478–3

Sequential Covering Algorithms

The basic algorithm:

1. *Learn one rule*
2. Remove the data it covers
3. Repeat

More specific version:

1. *Learn one rule* with high accuracy, any coverage
2. Remove positive examples covered by this rule
3. Repeat

Slide CS478–4

Generic Covering Algorithm

COVER($Target\_attr, Attrs, Examples, Threshold$)

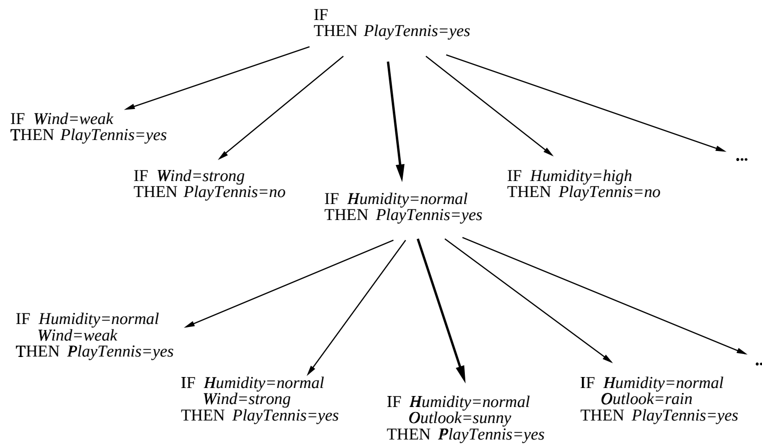
- $Learned\_rules \leftarrow \{\}$
- $Rule \leftarrow$ LEARN-ONE-RULE($Target\_attr, Attrs, Examples$)
- WHILE PERFORMANCE($Rule, Examples$) $> Threshold$, DO
  - $Learned\_rules \leftarrow Learned\_rules + Rule$
  - $Examples \leftarrow Examples - \{$EXAMPLES CORRECTLY CLASSIFIED BY $Rule\}$
  - $Rule \leftarrow$ LEARN-ONE-RULE($Target\_attr, Attrs, Examples$)
- $Learned\_rules \leftarrow$ SORT $Learned\_rules$ ACCORD TO PERFORMANCE OVER $Examples$
- RETURN $Learned\_rules$

Slide CS478–5

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*Ski?*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Slide CS478–6

Learn-One-Rule Search Space



Slide CS478–7

LEARN-ONE-RULE($Target\_attr, Attrs, Examples$)

- $Pos \leftarrow$ positive $Examples$; $Neg \leftarrow$ negative $Examples$
- If Pos

$NewRule \leftarrow$ most general rule possible; $NewRuleNeg \leftarrow Neg$

While $NewRuleNeg$

1. $Candidate\_literals(CLs) \leftarrow$ generate candidates
2. $Best\_literal \leftarrow argmax_{L \in CLs}$ $PERFORMANCE(Specialize(NewRule, L))$
3. add $Best\_literal$ to $NewRule$ preconditions
4. $NewRuleNeg \leftarrow$ subset of $NewRuleNeg$ that satisfies $NewRule$ preconditions

- Return $NewRule$

Slide CS478–8

Common Performance Metrics

Entropy: S = examples that match the rule's preconditions.

$$-Entropy(S) \equiv \sum_{i=1}^{c} x_i \log_2 x_i$$

Relative Frequency:

$$\frac{n_c}{n}$$

n = # examples the rule matches
n_c = # examples the rule matches and classifies correctly

---

m estimate:

$$\frac{n_c + mp}{n + m}$$

p = prior probability of the class assigned by the rule
m = # examples needed to override the prior

Learn-One-Rule Search Space

- general-to-specific search
- searches for a rule with high accuracy, but possibly low coverage
- measure to select the "best" descendant: one whose covered examples have the lowest entropy
- greedy
- can extend to perform a *beam search*

Variants of Rule Learning Programs

- *Sequential* or *simultaneous* covering of data?
- General → specific, or specific → general?
- Generate-and-test, or example-driven?
- Whether and how to post-prune?
- What statistical evaluation function?

Learning First Order Rules

Why do that?

- Can learn sets of rules such as
  $Ancestor(x, y) \leftarrow Parent(x, y)$
  $Ancestor(x, y) \leftarrow Parent(x, z) \wedge Ancestor(z, y)$
- General purpose programming language Prolog: programs are sets of such rules

First Order Rule for Classifying Web Pages

[Slattery, 1997]

course(A) $\leftarrow$

has-word(A, instructor),
Not has-word(A, good),
link-from(A, B),
has-word(B, assign),
Not link-from(B, C)

Train: 31/31, Test: 31/34

Slide CS478–13

Learning First-Order Rules

- Inductive learning of first-order rules is often called **inductive logic programming (ILP)**, because it can be used to learn PROLOG programs.
- ILP methods usually learn first-order **Horn Clauses**. A Horn clause is a disjunction of literals that has at most one positive literal (see book for details), such as:

$$C \vee \neg X_1 \vee \ldots \vee \neg X_n$$

which can conveniently be rewritten as:

$$X_1 \wedge \ldots \wedge X_n \rightarrow C$$

Slide CS478–14

FOIL($Target\_predicate, Predicates, Examples$)

- $Pos \leftarrow$ positive $Examples$; $Neg \leftarrow$ negative $Examples$
- While Pos

$NewRule \leftarrow$ most general rule possible; $NewRuleNeg \leftarrow Neg$

While $NewRuleNeg$

1. $Candidate\_literals(CLs) \leftarrow$ generate candidates
2. $Best\_literal \leftarrow argmax_{L \in CLs}\ Foil\_Gain(L, NewRule)$
3. add $Best\_literal$ to $NewRule$ preconditions
4. $NewRuleNeg \leftarrow$ subset of $NewRuleNeg$ that satisfies $NewRule$ preconditions

$Learned\_rules \leftarrow Learned\_rules + NewRule$

$Pos \leftarrow Pos - \{$members of Pos covered by $NewRule\}$

- Return $Learned\_rules$

Slide CS478–15

Specializing Rules in FOIL

Given a rule:

$$P(x_1, x_2, \ldots, x_k) \leftarrow L_1 \ldots L_n$$

Candidate specializations can add a new literal of form:

- $Q(v_1, \ldots, v_r)$, where at least one of the v_i in the created literal must already exist as a variable in the rule.
- $Equal(x_j, x_k)$, where x_j and x_k are variables already present in the rule
- The negation of either of the above forms of literals

Slide CS478–16

FOIL Gain Metric

Two Goals:

1. Decrease coverage of negative examples.
2. Maintain coverage of as many positive examples as possible.

$$FOIL\_Gain(L,R) \equiv t\left[log_2(\frac{P_{R+L}}{P_{R+L}+N_{R+L}}) - log_2(\frac{P_R}{P_R+N_R})\right]$$

where

- L is a literal and R is a rule
- P_R is the number of positive bindings for R
- N_R is the number of negative bindings for R
- P_{R+L} is the number of positive bindings for $R+L$
- N_{R+L} is the number of negative bindings for $R+L$
- t is the number of positive bindings of R and $R+L$

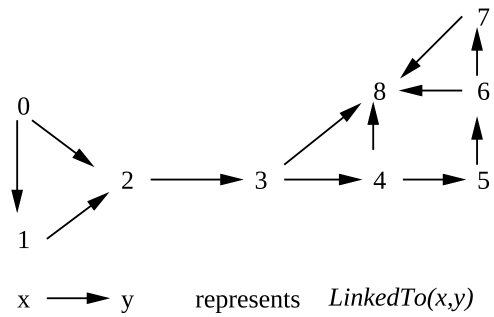
Learning Recursive Rules

- FOIL can learn recursive rules, such as:

  $Ancestor(x,y) \leftarrow Parent(x,y)$
  $Ancestor(x,y) \leftarrow Parent(x,z) \wedge Ancestor(z,y)$

- To learn recursive rules, the target predicate can be added to the list of candidate predicates used during rule learning.
- Special tricks are needed to avoid learning infinitely recursive rules.

FOIL Example



Instances:

- pairs of nodes, e.g. $\langle 1, 5 \rangle$, with graph described by literals *LinkedTo(0,1)*, $\neg$ *LinkedTo(0,8)* etc.

Slide CS478–19

Target function:

- *CanReach(x,y)* true iff directed path from x to y

Hypothesis space:

- Each $h \in H$ is a set of horn clauses using predicates $LinkedTo$ (and $CanReach$)

Slide CS478–20

Summary

- Rule learning systems have achieved good results and have produced rules that perform at least as well as manually engineered rules.
- Rule learning approaches can consider one attribute value independent of the others.
- To deal with overfitting, rules can be post-pruned.
- To handle noise, the criteria for adding literals must be loosened up.
- But the search can become intractable if the space of literals gets too large.
- Hill-climbing search can get stuck on local maxima.
- Closed-world assumption required for negative examples.

Slide CS478–21