Learning Sets of Rules

- Sequential covering algorithms
- FOIL
- Induction as the inverse of deduction
- Inductive Logic Programming

Learning Disjunctive Sets of Rules

Method 1: Learn decision tree, convert to rules

Method 2: Sequential covering algorithm

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

Sequential Covering Algorithm

SEQUENTIAL-COVERING(*Target_attr,Attrs,Examples,Thresh*)

Learned_rules ← {}

Rule ← LEARN-ONE-RULE(*Target_attr,Attrs,Examples*)

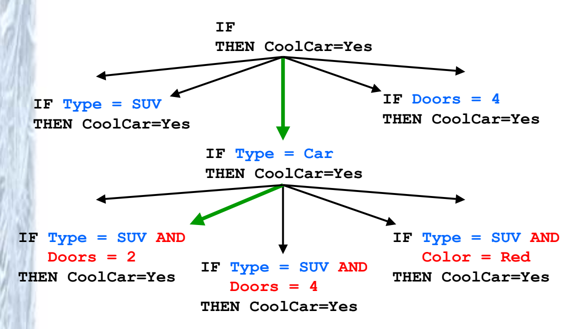
while PERFORMANCE(*Rule,Examples*) > *Thresh* do

- *Learned_rules* ← *Learned_rules* + *Rule*
- *Examples* ← *Examples* - {examples correctly classified by *Rule*}
- *Rule* ← LEARN-ONE-RULE(*Target_attr,Attrs,Examples*)

Learned_rules ← sort *Learned_rules* according to PERFORMANCE over *Examples*

return *Learned_rules*

Learn-One-Rule



Covering Rules

Pos ← positive *Examples*

Neg ← negative *Examples*

while *Pos* do (*Learn a New Rule*)

NewRule ← most general rule possible

NegExamplesCovered ← *Neg*

while *NegExamplesCovered* do

Add a new literal to specialize *NewRule*

1. *Candidate_literals* ← generate candidates
2. *Best_literal* ← $\text{argmax}_{L \in candidate\_literals}$ PERFORMANCE(SPECIALIZE-RULE(*NewRule,L*))
3. Add *Best_literal* to *NewRule* preconditions
4. *NegExamplesCovered* ← subset of *NegExamplesCovered* that satistifies *NewRule* preconditions

Learned_rules ← *Learned_rules* + *NewRule*

Pos ← *Pos* - {members of *Pos* covered by *NewRule*}

Return *Learned_rules*

Subtleties: Learning One Rule

1. May use *beam search*
2. Easily generalize to multi-valued target functions
3. Choose evaluation function to guide search:
   - Entropy (i.e., information gain)
   - Sample accuracy: $\frac{n_c}{n}$

     where n_c = correct predictions,

     n = all predictions
   - m estimate: $\frac{n_c + mp}{n + m}$

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 functions?

Learning First Order Rules

Why do that?

- Can learn sets of rules such as

  Ancestor(x,y) ← Parent(x,y)

  Ancestor(x,y) ← Parent(x,z) ∧ Ancestor(z,y)

- General purpose programming language

  PROLOG: programs are sets of such rules

First Order Rule for Classifying Web Pages

From (Slattery, 1997)

course(A) ←
    has-word(A,instructor),
    NOT has-word(A,good),
    link-from(A,B)
    has-word(B,assignment),
    NOT link-from(B,C)

Train: 31/31, Test 31/34

FOIL

FOIL(*Target_predicate,Predicates,Examples*)
Pos ← positive *Examples*
Neg ← negative *Examples*
while *Pos* do (*Learn a New Rule*)
    NewRule ← most general rule possible
    NegExamplesCovered ← *Neg*
    while *NegExamplesCovered* do
        Add a new literal to specialize *NewRule*
        1. *Candidate_literals* ← generate candidates
        2. *Best_literal* ← $\text{argmax}_{L \in candidate\_literal}$ FOIL_GAIN(*L,NewRule*)
        3. Add *Best_literal* to *NewRule* preconditions
        4. *NegExamplesCovered* ← subset of *NegExamplesCovered* that satistifies *NewRule* preconditions
    Learned_rules ← *Learned_rules* + *NewRule*
    Pos ← *Pos* - {members of *Pos* covered by *NewRule*}
Return *Learned_rules*

Specializing Rules in FOIL

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

Candidate specializations add 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

Information Gain in FOIL

$$FOIL\_GAIN(L,R) \equiv t\left(\log_2 \frac{p_1}{p_1+n_1} - \log_2 \frac{p_0}{p_0+n_0}\right)$$

Where

- L is the candidate literal to add to rule R
- p_0 = number of positive bindings of R
- n_0 = number of negative bindings of R
- p_1 = number of positive bindings of $R+L$
- n_1 = number of negative bindings of $R+L$
- t is the number of positive bindings of R also covered by $R+L$

Note

- $-\log_2 \frac{p_0}{p_0+n_0}$ is optimal number of bits to indicate the class of a positive binding covered by R

Induction as Inverted Deduction

Induction is finding h such that

$$(\forall \langle x_i, f(x_i) \rangle \in D)\ B \wedge h \wedge x_i \vdash f(x_i)$$

where

- x_i is the ith training instance
- $f(x_i)$ is the target function value for x_i
- B is other background knowledge

So let's design inductive algorithms by inverting operators for automated deduction!

Induction as Inverted Deduction

"pairs of people, <u,v> such that child of u is v,"

$f(x_i)$: *Child(Bob,Sharon)*

x_i : *Male(Bob),Female(Sharon),Father(Sharon,Bob)*

B : *Parent(u,v)* $\leftarrow$ *Father(u,v)*

What satisfies $(\forall \langle x_i, f(x_i) \rangle \in D)\ B \wedge h \wedge x_i \vdash f(x_i)$?

h_1 : *Child(u,v)* $\leftarrow$ *Father(v,u)*

h_2 : *Child(u,v)* $\leftarrow$ *Parent(v,u)*

Induction and Deduction

Induction is, in fact, the inverse operation of deduction, and cannot be conceived to exist without the corresponding operation, so that the question of relative importance cannot arise. Who thinks of asking whether addition or subtraction is the more important process in arithmetic? But at the same time much difference in difficulty may exist between a direct and inverse operation; … it must be allowed that inductive investigations are of a far higher degree of difficulty and complexity than any question of deduction … (Jevons, 1874)

Induction as Inverted Deduction

We have mechanical deductive operators

$$F(A,B) = C, \text{ where } A \wedge B \vdash C$$

need *inductive* operators

$O(B,D) = h$ where

$$(\forall \langle x_i, f(x_i) \rangle \in D)\ B \wedge h \wedge x_i \vdash f(x_i)$$

Induction as Inverted Deduction

Positives:

- Subsumes earlier idea of finding h that "fits" training data
- Domain theory B helps define meaning of "fit" the data

$$B \wedge h \wedge x_i \vdash f(x_i)$$

- Suggests algorithms that search H guided by B

Negatives:

- Doesn't allow for noisy data. Consider

$$(\forall \langle x_i, f(x_i) \rangle \in D)\ B \wedge h \wedge x_i \vdash f(x_i)$$

- First order logic gives a huge hypothesis space H
  - overfitting…
  - intractability of calculating all acceptable h's

Deduction: Resolution Rule

$$\frac{\begin{array}{c} P \vee L \\ \neg L \vee R \end{array}}{P \vee R}$$

1. Given initial clauses $C1$ and $C2$, find a literal L from clause $C1$ such that $\neg L$ occurs in clause $C2$.
2. Form the resolvent C by including all literals from $C1$ and $C2$, except for L and $\neg L$. More precisely, the set of literals occurring in the conclusion C is

$$C = (C1 - \{L\}) \cup (C2 - \{\neg L\})$$

where $\cup$ denotes set union, and "-" set difference.

Inverting Resolution

C_1: *PassExam* $\vee$ $\neg$*KnowMaterial* C_2: *KnowMaterial* $\vee$ $\neg$*Study*

$\boxed{C\text{: } PassExam \vee \neg Study}$

C_1: *PassExam* $\vee$ $\neg$*KnowMaterial* $\boxed{C_2\text{: } KnowMaterial \vee \neg Study}$

C: *PassExam* $\vee$ $\neg$*Study*

Inverted Resolution (Propositional)

1. Given initial clauses C_1 and C, find a literal L that occurs in clause C_1, but not in clause C.
2. Form the second clause C_2 by including the following literals

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

First Order Resolution

1. Find a literal L_1 from clause C_1 , literal L_2 from clause C_2, and substitution θ 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 $\neg L_2\theta$. More precisely, the set of literals occuring in the conclusion is

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

Inverting:

$$C_2 = (C - (C_1 - \{L_1\})\,\theta_1)\,\theta_2^{-1} \cup \{\neg L_1\,\theta_1\,\theta_2^{-1}\}$$

Cigol

Father(Tom,Bob) $\boxed{GrandChild(y,x) \vee \neg Father(x,z) \vee \neg Father(z,y))}$

{*Bob/y,Tom/z*}

Father(Shannon,Tom) $\boxed{GrandChild(Bob,x) \vee \neg Father(x,Tom))}$

{*Shannon/x*}

GrandChild(Bob,Shannon)

Progol

PROGOL: Reduce combinatorial explosion by generating the most specific acceptable *h*

1. User specifies *H* by stating predicates, functions, and forms of arguments allowed for each
2. PROGOL uses sequential covering algorithm.

   For each $<x_i, f(x_i)>$

   – Find most specific hypothesis h_i s.t.

   $B \wedge h_i \wedge x_i \vdash f(x_i)$

   actually, only considers k-step entailment
3. Conduct general-to-specific search bounded by specific hypothesis h_i, choosing hypothesis with minimum description length

Learning Rules Summary

- Rules: easy to understand
  - Sequential covering algorithm
  - generate one rule at a time
  - general to specific - add antecedents
  - specific to general - delete antecedents
  - Q: how to evaluate/stop?
- First order logic and covering
  - how to connect variables
  - FOIL

Learning Rules Summary (cont)

- Induction as inverted deduction
  - what background rule would allow deduction?
  - resolution
  - inverting resolution
  - and first order logic
    - Cigol, Progol