Learning Sets of Rules

[Read Ch. 10]
[Recommended exercises 10.1, 10.2, 10.5, 10.7, 10.8]

- Sequential covering algorithms
- FOIL
- Induction as 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\_attribute, Attributes, Examples, Threshold)

- $Learned\_rules \leftarrow \{\}$
- $Rule \leftarrow \text{LEARN-ONE-RULE}(Target\_attribute, Attributes, Examples)$
- while PERFORMANCE($Rule$, Examples) $> Threshold$, do
  - $Learned\_rules \leftarrow Learned\_rules + Rule$
  - $Examples \leftarrow Examples - \{\text{examples correctly classified by } Rule\}$
  - $Rule \leftarrow \text{LEARN-ONE-RULE}(Target\_attribute, Attributes, Examples)$
- $Learned\_rules \leftarrow \text{sort } Learned\_rules \text{ accord to PERFORMANCE over } Examples$
- return $Learned\_rules$
Learn-One-Rule

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

...
**Learn-One-Rule**

- Pos $\leftarrow$ positive Examples
- Neg $\leftarrow$ negative Examples
- while Pos, do
  - Learn a NewRule
    - NewRule $\leftarrow$ most general rule possible
    - NewRuleNeg $\leftarrow$ Neg
    - while NewRuleNeg, do
      - Add a new literal to specialize NewRule
        1. Candidate_literals $\leftarrow$ generate candidates
        2. Best_literal $\leftarrow$ argmax$_{L \in \text{Candidate_literals}}$ Performance(SpecializeRule(NewRule, L))
        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
Subtleties: Learn One Rule

1. May use beam search
2. Easily generalizes 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 rule predictions, \( n = \) all predictions
   - \( m \) estimate: \[ \frac{n_c + mp}{n + m} \]
Variants of Rule Learning Programs

- *Sequential* or *simultaneous* covering of data?
- General $\rightarrow$ specific, or specific $\rightarrow$ 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

\[ \text{Ancestor}(x, y) \leftarrow \text{Parent}(x, y) \]
\[ \text{Ancestor}(x, y) \leftarrow \text{Parent}(x, z) \land \text{Ancestor}(z, y) \]

- General purpose programming language
  PROLOG: programs are sets of such rules
First Order Rule for Classifying Web Pages

[Slattery, 1997]

course(A) ←
    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
FOIL($Target\_predicate$, $Predicates$, $Examples$)

- $Pos$ $\leftarrow$ positive $Examples$
- $Neg$ $\leftarrow$ negative $Examples$
- while $Pos$, do
  
  Learn a NewRule
  - $NewRule$ $\leftarrow$ most general rule possible
  - $NewRuleNeg$ $\leftarrow$ $Neg$
  - while $NewRuleNeg$, do
    
    Add a new literal to specialize $NewRule$
    1. $Candidate\_literals$ $\leftarrow$ generate candidates
    2. $Best\_literal$ $\leftarrow$
      $\arg\max_{L \in Candidate\_literals} Foil\_Gain(L, NewRule)$
    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 $Learned\_rules$
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

\[ \text{Foil Gain}(L, R) \equiv t \left( \log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right) \]

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

$$(\forall (x_i, f(x_i)) \in D) \ B \land h \land x_i \vdash f(x_i)$$

where

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

So let’s design inductive algorithm by inverting operators for automated deduction!
Induction as Inverted Deduction

“pairs of people, \( \langle u, v \rangle \) such that child of \( u \) is \( v \),”

\[
f(x_i) : \quad \text{Child}(Bob, Sharon) \\
\quad x_i : \text{Male}(Bob), \text{Female}(Sharon), \text{Father}(Sharon, Bob) \\
\quad B : \quad \text{Parent}(u, v) \leftarrow \text{Father}(u, v)
\]

What satisfies \((\forall (x_i, f(x_i)) \in D) \ B \land h \land x_i \vdash f(x_i)\)?

\[
h_1 : \text{Child}(u, v) \leftarrow \text{Father}(v, u) \\
h_2 : \text{Child}(u, v) \leftarrow \text{Parent}(v, u)
\]
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 questions of deduction....

(Jevons 1874)
Induction as Inverted Deduction

We have mechanical *deductive* operators
\[ F(A, B) = C, \text{ where } A \land B \vdash C \]

need *inductive* operators
\[ O(B, D) = h \text{ where } (\forall \langle x_i, f(x_i) \rangle \in D) (B \land h \land x_i) \vdash f(x_i) \]
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

\[
B \land h \land x_i \vdash f(x_i)
\]

• Suggests algorithms that search \( H \) guided by \( B \)
Induction as Inverted Deduction

Negatives:

• Doesn’t allow for noisy data. Consider

\[(\forall (x_i, f(x_i)) \in D) \ (B \land h \land x_i) \vdash f(x_i)\]

• First order logic gives a huge hypothesis space \( H \)

→ overfitting...
→ intractability of calculating all acceptable \( h \)’s
Deduction: Resolution Rule

\[ \begin{align*}
P \lor L \\
\neg L \lor R \\
P \lor R
\end{align*} \]

1. Given initial clauses \( C_1 \) and \( C_2 \), find a literal \( L \) from clause \( C_1 \) such that \( \neg L \) occurs in clause \( C_2 \).

2. Form the resolvent \( C \) by including all literals from \( C_1 \) and \( C_2 \), except for \( L \) and \( \neg L \). More precisely, the set of literals occurring in the conclusion \( C \) is

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

where \( \cup \) denotes set union, and “−” denotes set difference.
Inverting Resolution

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

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

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

First order resolution:

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 $\neg L_2 \theta$. More precisely, the set of literals occurring in the conclusion $C$ is

$$C = (C_1 - \{L_1\}) \theta \cup (C_2 - \{L_2\}) \theta$$
Inverting First order resolution

\[ C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1\theta_1\theta_2^{-1}\} \]
Cigol

\[
\text{Father} (\text{Tom, Bob})
\]

\[
\text{GrandChild} (y, x) \lor \neg \text{Father} (x, z) \lor \neg \text{Father} (z, y)
\]

{Bob/y, Tom/z}

\[
\text{Father} (\text{Shannon, Tom})
\]

\[
\text{GrandChild} (\text{Bob}, x) \lor \neg \text{Father} (x, \text{Tom})
\]

{Shannon/x}

\[
\text{GrandChild} (\text{Bob, Shannon})
\]
**Progol**

**PROGOL:** Reduce comb 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 $\langle x_i, f(x_i) \rangle$
   - Find most specific hypothesis $h_i$ s.t.
     \[ B \land h_i \land x_i \vdash f(x_i) \]
     – actually, considers only $k$-step entailment

3. Conduct general-to-specific search bounded by specific hypothesis $h_i$, choosing hypothesis with minimum description length