Learning Sets of Rules

- 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:
  ▶ Learn one rule with high accuracy, any coverage.
  ▶ Remove positive examples covered by this rule.
  ▶ Repeat.
Sequential Covering Algorithm

\[ \text{\textbf{Sequential-covering}(Target\_attribute, Attributes, Examples, Threshold)} \]

\[ \text{Learned\_rules} \leftarrow \{\} \]
\[ \text{Rule} \leftarrow \text{learn-one-rule}(Target\_attribute, Attributes, Examples) \]
\[ \text{while performance(Rule, Examples)} > \text{Threshold}, \text{do:} \]
  \[ \bullet \text{Learned\_rules} \leftarrow \text{Learned\_rules} + \text{Rule} \]
  \[ \bullet \text{Examples} \leftarrow \text{Examples} - \{\text{examples correctly classified by Rule}\} \]
  \[ \bullet \text{Rule} \leftarrow \text{learn-one-rule}(Target\_attribute, Attributes, Examples) \]
\[ \text{Learned\_rules} \leftarrow \text{sort Learned\_rules accord to performance over Examples} \]
\[ \text{return Learned\_rules} \]
Learn-One-Rule

10 Learning Sets of Rules
Learn-One-Rule

- $Pos \leftarrow$ positive Examples, $Neg \leftarrow$ negative Examples

- while $Pos$, do

  Learn a $NewRule$:
  - $NewRule \leftarrow$ most general rule possible, $NewNeg \leftarrow Neg$
  - while $NewNeg$, do

    Add a new literal to specialize $NewRule$:
    - $Candidate\_literals \leftarrow$ generate candidates
    - $Best\_literal \leftarrow \text{argmax}_{L \in Candidate\_literals} \text{Performance}(\text{SpecializeRule}(NewRule, L))$
    - $R \leftarrow \text{SpecializeRule}(NewRule, Best\_literal)$
    - $NewNeg \leftarrow$ subset of $NewNeg$ that satisfies preconditions of $R$.

  - $Learned\_rules \leftarrow Learned\_rules + R$
  - $Pos \leftarrow Pos - \{\text{members of } Pos \text{ covered by } R\}$

- Return $Learned\_rules$
Subtleties: Learn One Rule

- Gradient descent, which may lead to local minima.
  - **Beam search**: better, but local minima may still be reached.

- Easily generalizes to multi-valued target functions.

- 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-to-specific or specific-to-general?

▶ Generate-and-test or example-driven?

▶ Whether and how to post-prune?

▶ What statistical evaluation function?
Learning First Order Rules

- 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.

- Warning: Only constants, but no function symbols allowed.
  
  ▶ Still propositional in nature!
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 $NewRule$:
    
    - $Candidate\_literals \leftarrow$ generate candidates
    - $Best\_literal \leftarrow \text{argmax}_{L \in Candidate\_literals} \text{Foil}\_\text{Gain}(L, NewRule)$
    - add $Best\_literal$ to $NewRule$ preconditions
    - $NewRuleNeg \leftarrow$ subset of $NewRuleNeg$ that satisfies $NewRule$ preconditions
    
    - $Learned\_rules \leftarrow Learned\_rules + NewRule$
    
    - $Pos \leftarrow Pos - \{\text{members of } Pos \text{ covered by } NewRule\}$

- 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 \( L_i \) 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 atoms.
Learning of the target  \textit{GrandDaughter}

\begin{itemize}
\item Target\_Predicat = \textit{GrandDaughter}, Predicates = \{\textit{Female, Father}\}
\item Most general rule:
\begin{align*}
\text{GrandDaughter}(x, y) \leftarrow \\
\text{Candidate\_literals} = \text{Equal}(x, y), \text{Female}(x), \text{Female}(y), \\
\text{Father}(x, y), \text{Father}(y, x), \text{Father}(x, z), \\
\text{Father}(z, x), \text{Father}(y, z), \text{Father}(z, y), \\
\text{and negations thereof}.
\end{align*}
\item More specific rule:
\begin{align*}
\text{GrandDaughter}(x, y) \leftarrow \text{Father}(y, z) \\
\text{Candidate\_literals} = \text{Equal}(z, x), \text{Equal}(z, y), \text{Female}(z), \\
\text{Father}(z, w), \text{Father}(w, z), \text{Father}(x, w), \\
\text{Father}(w, x), \text{Father}(y, w), \text{Father}(w, y), \\
\text{and negations thereof plus the abovementioned ones}.
\end{align*}
\item Eventually we reach:
\begin{align*}
\text{GrandDaughter}(x, y) \leftarrow \text{Father}(y, z), \text{Father}(z, x), \text{Female}(y)
\end{align*}
\end{itemize}
Positive and Negative Bindings

- **Examples** = GrandDaughter(Victor, Sharon), Female(Sharon), Father(Sharon, Bob), Father(Tom, Bob), Father(Bob, Victor)

- We are making the closed world assumption!

- Reconsider
  
  \[ \text{GrandDaughter}(x, y) \leftarrow \]

- There are \(2^4 = 16\) different bindings with domain \{x, y\} and constants Victor, Sharon, Bob, Tom.

- Only \{x/Victor, y/Shannon\} yields a positive example, the others yield negative examples.

- Hence, \{x/Victor, y/Shannon\} is called positive binding, all others are called negative bindings.

- FOIL prefers rules with more positive bindings.
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 minimum number of bits needed to encode a positive binding covered by \( R \).
- \( -\log_2 \frac{p_1}{p_1+n_1} \) is minimum number of bits needed to encode a positive binding covered by \( R + L \).
- \( \text{Foil\_Gain}(L, R) \) is the reduction due to \( L \).
FOIL Example

Instances:
- pairs of nodes describing graph using predicate LinkedTo.

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 LinkedTo and CanReach.
Induction as Inverted Deduction

- Induction is finding \( h \) such that

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

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!
Example

“pairs of people, \langle u, v \rangle \text{ such that child of } u \text{ is } v,”

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

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

Among others:

\[ h_1 : \text{ Child}(u, v) \leftarrow \text{ Father}(v, u) \]
\[ h_2 : \text{ Child}(u, v) \leftarrow \text{ Parent}(v, u) \]
Induction

[Jevons 1874] 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. ...
The Search for Inductive Operators

- We have mechanical deductive operators $F(A, B) = C$, where $A \land B \vdash C$

- We need inductive operators

$$O(B, D) = h \text{ where } (\forall \langle x_i, f(x_i) \rangle \in D) (B \land h \land x_i) \vdash f(x_i)$$
Features

▶ 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 \( B \).

▶ Negatives:

▷ Doesn’t allow for noisy data. Consider

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

▷ First order logic gives a huge hypothesis space \( H \).
    → Overfitting...
    → Intractability of calculating all acceptable \( h \)’s.
Deduction: Resolution Rule (Propositional)

\[
\begin{array}{c}
P \lor L \\
\neg L \lor R \\
P \lor R
\end{array}
\]

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

- Find a literal \( L_1 \) from clause \( C_1 \), literal \( L_2 \) from clause \( C_2 \) and substitution \( \theta \) such that \( \theta \) is a most general unifier for \( L_1 \) and \( \neg L_2 \).

- Form the resolvent \( C \) by including all literals from \( C_1\theta \) and \( C_2\theta \), except for \( L_1\theta \) and \( L_2\theta \), i.e.,

\[
C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta.
\]

- Clauses are standardized apart:
  - \( \text{Var}(C_1) \cap \text{Var}(C_2) = \emptyset \),
  - \( \text{Dom}(\theta) = \text{Var}(C_1) \cup \text{Var}(C_2) \),
  - \( \theta_1 = \theta|_{\text{Var}(C_1)} \),
  - \( \theta_2 = \theta|_{\text{Var}(C_2)} \),
  - \( \theta = \theta_1 \cup \theta_2 \),
  - \( C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2 \).
Inverse Substitution

Given atom $A$. An **inverse substitution** $\theta^{-1}$ of a substitution $\theta$ is a mapping from terms occurring in $A\theta$ to variables such that $A\theta\theta^{-1} = A$.

**Examples**

1. $A = Daughter(x, y)$, $\theta = \{x/Mary, y/Ann\}$:
   
   $\theta^{-1} = \{Mary/x, Ann/y\}$.

2. $A = Loves(x, y)$, $\theta = \{x/Ann, y/Ann\}$:
   
   $\theta^{-1} = ?$.

**Solution** Add positions to distinguish different occurrences of the same term:

2’. $A = Loves(x, y)$, $\theta = \{x/Ann, y/Ann\}$:

   $\theta^{-1} = \{(Ann, 1)/x, (Ann, 2)/y\}$.

Such substitutions are called **inverse substitutions with positions**.
Inverse First-Order Resolution

- Given $C$ and $C_1$. Find $C_2$ assuming that $C_1$ and $C_2$ have no literals in common.
- Recall
  \[ C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2. \]
- But then
  \[ C - (C_1 - \{L_1\})\theta_1 = (C_2 - \{L_2\})\theta_2. \]
- Hence,
  \[ C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1\theta_1\theta_2^{-1}\}. \]
  is called inverse resolvent of $C$ and $C_1$.

- Non-determinism: Given $C$ as observed positive example.
  - Which clause from background knowledge $B$ shall we select as $C_1$?
  - Which literal from $C_1$ shall we select to resolve upon?
  - Which inverse substitution $\theta_2^{-1}$ shall we choose?

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

- Let $B = \{\{Female(Mary)\}, \{Parent(Ann, Mary)\}\}$.
  - Let $C = \{Daughter(Mary, Ann)\}$ and $C_1 = \{Parent(Ann, Mary)\}$.
    - $L_1 = Parent(Ann, Mary)$.
    - $\theta_1 = \varepsilon$.
    - Choose $\theta_2^{-1} = \{Ann/y\}$.
    - We obtain $C_2 = \{Daughter(Mary, y), \neg Parent(y, Mary)\}$.
    - Observe, $B \land C_2 \vdash \{Daughter(Mary, Ann)\}$.

- Let $C = \{Daughter(Mary, y), \neg Parent(y, Mary)\}$ and $C_1 = \{Female(Mary)\}$.
  - $L_1 = Female(Mary)$.
  - $\theta_1 = \varepsilon$.
  - Choose $\theta_2^{-1} = \{Mary/x\}$.
  - We obtain $C_2 = \{Daughter(x, y), \neg Parent(y, x), \neg Female(x)\}$.
  - Observe, $B \land C_2 \vdash \{Daughter(Mary, Ann)\}$.
Most Specific Inverse Resolution

- **Idea** Use the most specific inverse substitution.

- Let $B = \{ \{ \text{Female}(\text{Mary}) \} \}, \{ \text{Parent}(\text{Ann}, \text{Mary}) \} \}$.

  - Let $C = \{ \text{Daughter}(\text{Mary}, \text{Ann}) \}$ and $C_1 = \{ \text{Parent}(\text{Ann}, \text{Mary}) \}$.
    - $L_1 = \text{Parent}(\text{Ann}, \text{Mary})$.
    - $\theta_1 = \varepsilon$.
    - Choose $\theta_2^{-1} = \varepsilon$.
    - We obtain $C_2 = \{ \text{Daughter}(\text{Mary}, \text{Ann}), \neg \text{Parent}(\text{Ann}, \text{Mary}) \}$.
    - Observe, $B \land C_2 \vdash \{ \text{Daughter}(\text{Mary}, \text{Ann}) \}$.

  - Let $C = \{ \text{Daughter}(\text{Mary}, \text{Ann}), \neg \text{Parent}(\text{Ann}, \text{Mary}) \}$ and $C_1 = \{ \text{Female}(\text{Mary}) \}$.
    - $L_1 = \text{Female}(\text{Mary})$.
    - $\theta_1 = \varepsilon$.
    - Choose $\theta_2^{-1} = \varepsilon$.
    - We obtain $C_2 = \{ \text{Daughter}(\text{Mary}, \text{Ann}), \neg \text{Parent}(\text{Ann}, \text{Mary}), \neg \text{Female}(\text{Mary}) \}$.
    - Observe, $B \land C_2 \vdash \{ \text{Daughter}(\text{Mary}, \text{Ann}) \}$.
A Unifying Framework for Generalization

Let \( B = \{ \{ \text{Female}(Mary) \}, \{ \text{Parent}(Ann, Mary) \}, \{ \text{Female}(Eve) \}, \{ \text{Parent}(Tom, Eve) \} \} \).

Training Data: \{ \text{Daughter}(Mary, Ann) \}, \{ \text{Daughter}(Eve, Tom) \} \).

Using most specific inverse resolution we obtain:

\[
\{ \text{Daughter}(Mary, Ann), \neg \text{Parent}(Ann, Mary), \neg \text{Female}(Mary) \},
\{ \text{Daughter}(Eve, Tom), \neg \text{Parent}(Tom, Eve), \neg \text{Female}(Eve) \}. 
\]

Compute the so-called least general generalization:

\[
\{ \text{Daughter}(x, y), \neg \text{Parent}(y, x), \neg \text{Female}(x) \}. 
\]

What is the least general generalization?
\[ \theta \text{-Subsumption} \]

- Let \( C_1 \) and \( C_2 \) be two clauses.
  - \( C_1 \) \( \theta \)-subsumes \( C_2 \) if there exists a substitution \( \theta \) such that \( C_1 \theta \subseteq C_2 \).
  - A clause is \textit{reduced} if it is not \( \theta \)-subsumption equivalent to any proper subset of itself.

- **Example:** Consider

\[
C_1 = \{ \text{Daughter}(x, y), \neg \text{Parent}(y, x) \} \\
C_2 = \{ \text{Daughter}(\text{Mary}, y), \neg \text{Female}(\text{Mary}), \neg \text{Parent}(y, \text{Mary}) \}
\]

\( C_1 \) \( \theta \)-subsumes \( C_2 \) under \( \theta = \{ x / \text{Mary} \} \).

- \( C_1 \) is \textit{at least as general as} \( C_2 \), in symbols \( C_1 \leq C_2 \), if \( C_1 \) \( \theta \)-subsumes \( C_2 \).
- \( C_1 \) is \textit{more general than} \( C_2 \), in symbols \( C_1 < C_2 \), if \( C_1 \leq C_2 \) and \( C_2 \not\leq C_1 \).
  - \( C_1 \) is a \textit{generalization} of \( C_2 \).
  - \( C_2 \) is a \textit{specialization} of \( C_1 \).
**θ-Subsumption: Properties**

- If $C_1 \leq C_2$ then $C_1 \models C_2$.
- $\leq$ induces a lattice on the set of reduced clauses.
  - $lub$ and $glb$ always exist,
  - are unique modulo variable renaming
- The least general generalization of two reduced clauses $C_1$ and $C_2$, in symbols $lgg(C_1, C_2)$, is the lub of $C_1$ and $C_2$ in the $\theta$-subsumption lattice.
- $\theta$-subsumption provides a generality ordering for hypothesis.
- It can be used to prune the search space.
Computing Least General Generalizations

▶ Terms

▷ \( lgg(t, t) = t. \)
▷ \( lgg(F(s_1, \ldots, s_n), F(t_1, \ldots, t_n)) = F(lgg(s_1, t_1), \ldots, lgg(s_n, t_n)). \)
▷ \( lgg(F(s_1, \ldots, s_n), G(t_1, \ldots, t_m)) = v, \) where \( F \neq G \)
   and \( v \) is a variable which represents \( lgg(f(s_1, \ldots, s_n), g(t_1, \ldots, t_m)). \)
▷ \( lgg(s, t) = v, \) where \( s \neq t, \) at least one of \( s \) and \( t \) is a variable
   and \( v \) represents \( lgg(s, t). \)

▶ Atoms

▷ \( lgg(P(s_1, \ldots, s_n), P(t_1, \ldots, t_n)) = P(lgg(s_1, t_1), \ldots, lgg(s_n, t_n)). \)
▷ \( lgg(P(s_1, \ldots, s_n), Q(t_1, \ldots, t_m)) \) is undefined if \( P \neq Q. \)

▶ Negated atoms

▷ \( lgg(\neg P(s_1, \ldots, s_n), \neg P(t_1, \ldots, t_n)) = \neg P(lgg(s_1, t_1), \ldots, lgg(s_n, t_n)). \)
▷ \( lgg(\neg P(s_1, \ldots, s_n), Q(t_1, \ldots, t_m)) \) is undefined.

▶ Clauses

▷ Let \( C_1 = \{L_1, \ldots, L_n\} \) and \( C_2 = \{K_1, \ldots, K_m\} \) be two clauses.
   \( lgg(C_1, C_2) = \{ lgg(L_i, K_j) \mid L_i \in C_1, K_j \in C_2, \ lgg(L_i, K_j) \text{ defined} \}. \)