Learning Sets of Rules

Computer Science 760
Patricia J Riddle
Motivation

Set of if-then rules that jointly define the target function

Rules are easy (?) for people to understand and edit

Rules we’ve seen

- Translate a decision tree into a set of rules
- Use a genetic algorithm that encodes a rule set

But also first-order rules or partial or overlapping models
Decision tree

- **Outlook**
  - Sunny
  - Overcast
  - Rain
    - Yes
  - Humidity
    - High
      - No
      - Yes
    - Normal
      - Yes
    - Wind
      - Strong
      - Weak
      - No
      - Yes
Sequential Covering

Learn one-rule, remove the data it covers, then iterate

Our rule must have high accuracy but not necessarily high-coverage

(what does this do to the overfitting or oversearching problem??)

Only throw out positive examples covered

Final rules sorted by accuracy over the *whole* training set

Widely used
Issues with Sequential Covering

Greedy search, so no guarantees about smallest set or best set of rules

So each rule is learned on a different distribution of the training set.....isn’t this a problem???

Definitely skewed to best “set of rules” not best “rules”
Sequential Covering Algorithm

Sequential-covering(Target-attribute, Attributes, Examples, Threshold)

Learned_rules←{}

Rule←LEARN-ONE-RULE(Target-attribute, Attributes, Examples)

While PERFORMANCE(Rule, Examples) > Threshold, do

    Learned-rules←Learned-rules + Rule
    Examples←Examples-{examples correctly classified by Rule}
    Rule←LEARN-ONE-RULE(Target-attribute, Attributes, Examples)

Learned-rules ←sort Learned-rules according to PERFORMANCE over Examples

Return Learned-rules
How to Learn-One-Rule

General-to-specific search through the space of possible rules in search of a rule with high accuracy

Many ways to evaluate best descendant (same as decision trees) - like entropy

(greedy, no-backtracking) can extend to beam-search - CN2

Search continues until it reaches a maximally specific hypothesis that contains all available attributes

Postcondition is determined last
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 0
Initialise Candidate-hypothesis to the set \{Best-hypothesis\}

While Candidate-hypothesis is not empty. Do

Generate the next move specific candidate-hypotheses

All-constraints <- the set of all constraints of the form (a=v), where a is a member of Attributes, and v is a value of a that occurs in the current set of Examples

New-candidate-hypotheses <-

for each h in Candidate-hypotheses

for each c in All-constraints,

  Create a specialization of h by adding the constraint c

Remove from New-candidate-hypotheses any hypotheses that are duplicates, inconsistent or not maximally specific

Update Best-hypothesis

For all h in New-candidate-hypotheses do

  If (PERFORMANCE(h,Examples,Target-attribute) > PERFORMANCE(Best-hypothesis,Examples,Target-attribute))

    Then Best-hypothesis <- h

Update Candidate-hypotheses

Candidate-hypotheses <- the k best members of New-candidate-hypotheses, according to the PERFORMANCE measure

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
PERFORMANCE(h, Examples, Target-attribute)

H-examples <- the subset of Examples that match h

Return -Entropy(h-examples), where entropy is with respect to Target-attribute
Variation

Learning rules for only a single class - negation as failure -
“pregnant women who are likely to have twins”

Must change “performance” to fractions of positives covered – AQ
Skewed sample size encourages this also!

AQ uses single positive seed example to focus search in
Learn-One-Rule

Only considers attributes satisfied by that positive instance

A new seed example is chosen from those positive examples
not yet covered
Design Choices: Sequential versus Simultaneous

Sequential Covering Algorithms learn one rule at a time, remove the covered examples, and repeat.

Decision trees can be seen as Simultaneous Covering Algorithms
- Sequential covering algorithms perform $n \times k$ primitive search steps to learn $n$ rules each containing $k$ attribute-value tests.
- If the decision trees is a complete binary tree, it makes $(n-1)$ primitive search steps where $n$ is the number of paths (i.e., rules).

So Sequential Covering Algorithms must be supported by additional data, but have the advantage of allowing rules with different tests.
General-to-Specific versus Specific-to-General

General to specific starts at the one maximally general hypothesis

In specific to general there are many maximally specific hypothesis (the training data).

Golem chooses several randomly and picks the best learned hypothesis.
Generate-then-test or Example-driven

GTT hypothesis performance is based on many training examples

the effect of noisy data is minimized
Post-pruning

In either system post-pruning can be used to increase the effectiveness of rules on a validation set
Rule Performance Measures

Relative frequency - AQ – \( \frac{n_c}{n} \)

\( n \) – instances that match the if part of the rule
\( n_c \) – instances that match the if and the then parts of the rule
M-estimate of Accuracy

M-estimate of accuracy - CN2 - \( \frac{N_c + mp}{n + m} \)

\( p \) – prior probability

\( m \) – the weight or equivalent number of instances for weighing this prior
Entropy

- Entropy – CN2 –

\[-Entropy(S) = \sum_{i=1}^{c} p_i \log_2 p_i\]

c – the number of classes

\(p_i\) is the proportion of instances in the set \(S\) for which the class takes on the \(i^{th}\) value
Exhaustive Rule Learning

Greedy search can miss good rules
  What about over-searching???
  Really multiple comparison problem

Disallowing overlapping rules can cause problems

Solution: look at every rule and keep it if it is good
Brute

Exhaustive depth bounded search

When evaluating single rules, coverage is important
  Chi-squared statistic

Multiple comparisons more of a problem!!
  Validation sets difficult for rules
  We use randomization testing

Presenting multiple rules are difficult
  Also a problem with similar rules and additional conjuncts

“equivalent to” (sort of) association rules
Brute Rules vs Association Rules

• Classification rules versus Association rules

• Apriori runs the data past the hypothesis space (held in memory)

• Brute runs the hypothesis space past the data (held in memory)
Brute Run

> brute -T iopus -d 4 -S chi -F simnum -F simparent -r 100 dataset3 b
Setting up tests...
Doing search...
  1: MinPos = 1, Tests = 1899 .... Rules = 1,899, Seconds = 1.
  2: MinPos = 1, Tests = 1899 .... Rules = 168,025, Seconds = 1.
  3: MinPos = 1, Tests = 1899 .... Rules = 7,673,351, Seconds = 21.
  4: MinPos = 1, Tests = 1899 .... Rules = 161,432,100, Seconds = 464.
done.

Data positive coverage = 69.2%.
Test positive coverage = 25.0%.

Search time = 486 seconds.
Rules examined = 169,275,375.
Search speed = 348,303 rules per second.
>
<table>
<thead>
<tr>
<th>Rule</th>
<th>IF</th>
<th>THEN</th>
<th>Data Acc</th>
<th>Data Cov</th>
<th>Data Chi</th>
<th>Test Acc</th>
<th>Test Cov</th>
<th>Test Chi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>attr6 = a &amp;&amp; attr11 &lt;&gt; e &amp;&amp; attr31 &gt;= 21</strong></td>
<td><strong>attr1 = b</strong></td>
<td>100.0</td>
<td>34.6</td>
<td>116.7</td>
<td>50.0</td>
<td>12.5</td>
<td>8.3</td>
</tr>
<tr>
<td>2</td>
<td><strong>attr6 &lt;&gt; c &amp;&amp; attr6 &lt;&gt; b &amp;&amp; attr11 &lt;&gt; e &amp;&amp; attr31 &gt;= 24</strong></td>
<td><strong>attr1 = b</strong></td>
<td>100.0</td>
<td>34.6</td>
<td>116.7</td>
<td>50.0</td>
<td>12.5</td>
<td>8.3</td>
</tr>
<tr>
<td>3</td>
<td><strong>attr6 &lt;&gt; c &amp;&amp; attr7 &gt;= 27 &amp;&amp; attr11 &lt;&gt; e &amp;&amp; attr31 &gt;= 24</strong></td>
<td><strong>attr1 = b</strong></td>
<td>100.0</td>
<td>34.6</td>
<td>116.7</td>
<td>50.0</td>
<td>12.5</td>
<td>8.3</td>
</tr>
<tr>
<td>4</td>
<td><strong>attr2 = c &amp;&amp; attr3 &lt;&gt; c &amp;&amp; attr7 &gt;= 27 &amp;&amp; attr35 &lt; 1029</strong></td>
<td><strong>attr1 = b</strong></td>
<td><strong>78.6</strong></td>
<td><strong>42.3</strong></td>
<td><strong>107.4</strong></td>
<td><strong>0.0</strong></td>
<td><strong>0.0</strong></td>
<td><strong>0.1</strong></td>
</tr>
<tr>
<td>5</td>
<td><strong>attr3 &lt;&gt; c &amp;&amp; attr7 &gt;= 24 &amp;&amp; attr31 &gt;= 21 &amp;&amp; attr39 &gt;= 7</strong></td>
<td><strong>attr1 = b</strong></td>
<td><strong>83.3</strong></td>
<td><strong>38.5</strong></td>
<td><strong>104.7</strong></td>
<td><strong>20.0</strong></td>
<td><strong>12.5</strong></td>
<td><strong>2.3</strong></td>
</tr>
<tr>
<td>6</td>
<td><strong>attr6 &lt;&gt; c &amp;&amp; attr7 &gt;= 24 &amp;&amp; attr31 &gt;= 27 &amp;&amp; attr39 &gt;= 7</strong></td>
<td><strong>attr1 = b</strong></td>
<td><strong>100.0</strong></td>
<td><strong>30.8</strong></td>
<td><strong>103.7</strong></td>
<td><strong>20.0</strong></td>
<td><strong>12.5</strong></td>
<td><strong>2.3</strong></td>
</tr>
</tbody>
</table>
## Brute Bottom Rules

<table>
<thead>
<tr>
<th>Data</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc</td>
<td>Cov</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>IF attr6 &lt;&gt; c &amp;&amp; attr6 &lt;&gt; b &amp;&amp; attr25 &gt;= 9 &amp;&amp; attr31 &lt; 139 THEN attr1 = b</td>
<td>100.0</td>
</tr>
<tr>
<td>IF attr6 &lt;&gt; c &amp;&amp; attr6 &lt;&gt; b &amp;&amp; attr25 &gt;= 9 &amp;&amp; attr31 &gt;= 16 THEN attr1 = b</td>
<td>100.0</td>
</tr>
<tr>
<td>IF attr6 &lt;&gt; c &amp;&amp; attr6 &lt;&gt; b &amp;&amp; attr25 &gt;= 9 &amp;&amp; attr35 &lt; 1029 THEN attr1 = b</td>
<td>100.0</td>
</tr>
<tr>
<td>IF attr6 &lt;&gt; c &amp;&amp; attr6 &lt;&gt; b &amp;&amp; attr25 &gt;= 9 &amp;&amp; attr35 &gt;= 64 THEN attr1 = b</td>
<td>100.0</td>
</tr>
<tr>
<td>IF attr6 &lt;&gt; c &amp;&amp; attr6 &lt;&gt; b &amp;&amp; attr25 &gt;= 9 &amp;&amp; attr36 = i THEN attr1 = b</td>
<td>100.0</td>
</tr>
<tr>
<td>IF attr6 &lt;&gt; c &amp;&amp; attr6 &lt;&gt; b &amp;&amp; attr25 &gt;= 9 &amp;&amp; attr30 = u THEN attr1 = b</td>
<td>100.0</td>
</tr>
</tbody>
</table>
Other Brute Features

Only does one class at a time

Chi-square allows negative rules to be found

Can use beam-search

Can make decision list
## Confusion Matrix

<table>
<thead>
<tr>
<th></th>
<th>Then True</th>
<th>Then False</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IF True</td>
<td>a</td>
<td>b</td>
<td>$N_{IT}$</td>
</tr>
<tr>
<td>IF False</td>
<td>c</td>
<td>d</td>
<td>$N_{IF}$</td>
</tr>
<tr>
<td></td>
<td>$N_{TT}$</td>
<td>$N_{TF}$</td>
<td>$N$</td>
</tr>
</tbody>
</table>
Let us look at Accuracy & Error

• Acc = a/N_{IT}

• Error = b/N_{IT}
Chi Square Formula

\[ X^2 = \frac{N(ad - bc)^2}{N_{IT}N_{IF}N_{TT}N_{TF}} \]

- Uses the WHOLE table!
Learning First Order Rules

Inductive logic programming (ILP)

Automatically inferring Prolog programs from examples
Why Not Propositional Rules?

Name1=Sharon, Mother1=Louise,
Father1=Bob, Male1=False,
Female1=True, Name2=Bob,
Mother2=Nora, Father2=Victor,
Male2=True, Female2=False

If (Father1=Bob) ^ (Name2=Bob) ^ Female1=True
then Daughter1-2=True

Can’t describe relations between attributes!
First Order Horn Clauses

If Father(y,x) ^ Female(y) then Daughter(x,y)

Can also have variables in the preconditions which are not used in the postconditions - such variables are assumed to be existentially quantified

If Father(y,z) ^ Mother(z,x) ^ Female(y) then GrandDaughter(x,y)

Can also represent (and learn!) recursive functions

If Parents(x,z) ^ Ancestor(z,y) then Ancestor(x,y)
Terminology

Every well-formed expression is composed of

- **constants** (e.g., Mary, 23, or Joe),
- **variables** (e.g., x),
- **predicates** (e.g., Female, as in Female(Mary)), and
- **functions** (e.g., age is in age(Mary)).

A **term** is any constant, any variable, or any function applied to any term. Examples include Mary, x, age(Mary), age(x).
A **literal** is any predicate (or its negation) applied to any set of terms. Examples include Female(Mary), ¬Female(x), Greater_than(age(Mary), 20)).

A **ground literal** is a literal that does not contain any variables (e.g., ¬Female(Joe)).

A **negative literal** is a literal containing a negated predicate (e.g., ¬Female(Joe)).

A **positive literal** is a literal with no negation sign (e.g., Female(Mary)).
Clauses

A **clause** is any disjunction of literals $M_1 v \ldots M_n$ whose variables are universally quantified.

A **Horn clause** is an expression of the form $H \leftarrow (L_1 ^ \ldots ^ {^L_n})$ where $H, L_1 \ldots L_n$ are positive literals. $H$ is called the head or consequent of the Horn clause. The conjunction of literals $L_1 ^ L_2 ^ \ldots ^ {^L_n}$ is called the body or antecedents of the Horn clause.

For any literals $A$ and $B$, the expression $(A \leftarrow B)$ is equivalent to $(A v \neg B)$, and the expression $\neg (A ^ B)$ is equivalent to $(\neg A v \neg B)$. Therefore a Horn clause can equivalently be written as the disjunction $H v \neg L_1 v \ldots v \neg L_n$. 
A substitution is any function that replaces variables by terms. For example, the substitution \( \{x/3, y/z\} \) replaces the variable \( x \) by the term 3 and replaces the variable \( y \) by the term \( z \). Given a substitution \( \theta \) and a literal \( L \) we write \( L\theta \) to denote the result of applying substitution \( \theta \) to \( L \).

A unifying substitution for two literals \( L_1 \) and \( L_2 \) is any substitution \( \theta \) such that \( L_1\theta = L_2\theta \).
FOIL

Extension of Sequential Covering to first order representations

Learns Horn clauses with 2 exceptions

1. More restrictive - literals are not permitted to contain function symbols - reduces complexity of hypothesis space
2. More expressive - literals appearing in the body may be negated

Learn recursive Quicksort & legal from illegal chess positions
FOIL Algorithm

FOIL(Target-predicate, Predicates, Examples)
Pos ← those Examples for which the Target-predicate is True
Neg ← those Examples for which the Target-predicate is False
Learned-rules ← {}  
While Pos, do
    Learn a NewRule
    NewRule ← the rule that predicts Target-predicate with no preconditions
    NewRuleNeg ← Neg
    While NewRuleNeg, do
        Add a new literal to specialize NewRule
        Candidate_literals ← generate candidate new literals for NewRule, based on Predicates
        Best_literal ← \arg\max_{L \in \text{Candidate-literals}} \text{Foil-Gain}(L, \text{NewRule})
        Add Best-literal to preconditions of NewRule
        NewRuleNeg ← subset of NewRuleNeg that satisfies NewRule preconditions
        Learned-rules ← Learned-rules + NewRule
    Pos ← Pos - {members of Pos covered by NewRule}
Return Learned-rules
Differences between FOIL & Sequential Covering

Seeks only rules where target literal is True

Performs simple hill-climbing search rather than beam search

Adding each new rule generalizes the disjunctive hypothesis so viewed at this level the search is specific-to-general

Adding new conjuncts to each rule is a general-to-specific hill-climbing search
Issues for FOIL

1. How to generate candidate specializations of a rule - need to accommodate variables

2. What performance measure to use - need to distinguish between different bindings of the rules variables
Generating Candidate Specializations

1. $Q(v_1,\ldots,v_r)$ - where $Q$ is any predicate occurring in Predicates and the $v_i$ are either new variables or variables already present in the rule. At least one $v_i$ in the created literal must already exist in the rule.

2. $\text{Equal}(x_j,x_k) = \text{where } x_j \text{ and } x_k \text{ are variables already present in the rule}$

3. The negation of either of the above
FOIL Example

GrandDaughter(x,y) where Predicates contains Father and Female

Candidate Literals: Equal(x,y), Female(x), Female(y), Father(x,y), Father(y,x), Father(x,z), Father(z,x), Father(y,z), Father(z,y) and the negation of each

Let us assume FOIL greedily selects GrandDaughter(x,y) ← Father(y,z)
FOIL Example II

FOIL now considers all those before and Female(z), Equal(z,x), Equal(z,y), Father(z,w), Father(w,z) and their negations

Continues until it covers only positive examples, then remove all positive examples covered and start search for next rule
Guiding Search in FOIL

Performance of the rule over the training data

Must consider all possible bindings of each variable

Use the closed world assumption - any literal involving these predicates and these constants that is not listed is assumed false
Evaluation Function

Target literal GrandDaughter(x, y)

Assertions - GrandDaughter(Victor, Sharon),
Father(Sharon, Bob), Father(Tom, Bob), Female(Sharon),
Father(Bob, Victor)

Given the 4 constants there are 16 possible variable bindings
for the initial rule - 1 positive x/Victor, y/Sharon and 15
negative

Evaluation function let R´ be the rule created by adding a new
literal L to the old rule R
Foil-Gain

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

Where

- \( p_0 \) is the number of positive bindings of rule \( R \) and \( n_0 \) is the number of negative bindings,
- \( p_1 \) is the number of positive bindings of rule \( R' \),
- \( n_1 \) is the number of negative bindings of rule \( R' \), and
- \( t \) is the number of positive bindings of rule \( R \) which are still covered by \( R' \)

Reduction due to \( L \) in the total bits needed to encode the classification of all positive bindings of \( R \).
Learning Recursive Rule Sets

Just include the target in the list of Predicates

Need test to avoid learning rules sets that produce infinite recursion
Summary of FOIL

FOIL extension of CN2

General-to-specific search adding new literals

Literals may introduce new variables

Foil-Gains used as evaluation function

FOIL has been shown to successfully learn recursive rule sets

To handle noisy data, some tradeoff between accuracy, coverage, and complexity tells it when to stop adding new literals

FOIL also performs post-pruning
Induction as Inverted Deduction

Induction is the inverse of deduction

Given some data D and some partial background theory B, learning generates a hypothesis h that together with B explains D

More precisely, if the training data is a set of examples of the form \(<x_i, f(x_i)>\) where \(x_i\) denotes the ith training example and \(f(x_i)\) denotes its target value.

Then learning is the problem of discovering h such that

\[(\forall < x_i, f(x_i) > \in D)(B \land h \land x_i) \text{ entails } f(x_i)\]
Inverted Deduction Example I

Target concept is Child(u, v)

Single positive example Child(Bob, Sharon) where instance is described by Male(Bob), Female(Sharon), and Father(Sharon, Bob)

General background knowledge of Parent(u, v) ← Father(u, v)

Two of the many hypothesis that satisfy

\((B \land h \land x_i)\) entails \(f(x_i)\) are:

h1: Child(u, v) ← Father(v, u) and
h2: Child(u, v) ← Parent(v, u)
Inverted Deduction Example II

New predicates which were not present in the initial description can be introduced into the hypothesis - constructive induction

Well understood algorithms for automated deduction

Inverses of these procedures can automate inductive generalization
Inverse Entailment Operators

\[ O(B,D) = h \text{ such that } (\forall < x_i, f(x_i) > \in D)(B \land h \land x_i) \text{ entails } f(x_i) \]

Usually many hs so use Minimum Description Length

Incorporating background knowledge allows a more rich definition of when the hypothesis is said to fit the data

Several practical difficulties:

- Noisy training data
- First order logic is so expressive that the search is intractable - restricted forms of expression or additional second-order knowledge
- The complexity of the hypothesis space search increases as background knowledge is increased
Inverting Resolution

- Resolution rule - Robinson 65 - sound and complete
- This operator used in Cigol
- $C = A \lor B$ and $C_2 = B \lor D$
- Any literal present in $C$ but not in $C_1$ must be present in $C_2$
- The literal that occurs in $C_1$ but not in $C$ must be the literal removed by the resolution rule and therefore its negation must occur in $C_2$
- $C_2 = A \lor \neg D$ or $C_2 = A \lor \neg D \lor B$
- Not deterministic! - so prefer shorter clauses
- Cigol uses inverse resolution with sequential covering but with 1st order representations
Resolution

\[ P \lor L \]
\[ \neg L \lor R \]
\[ \text{-------} \]
\[ P \lor R \]

\[ C_1: \text{Pass} \lor \neg \text{KnowMaterial} \]
\[ C_2: \text{KnowMaterial} \lor \neg \text{Study} \]

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

\[ C_1: \text{Pass} \lor \neg \text{KnowMaterial} \]
\[ C_2: \text{KnowMaterial} \lor \neg \text{Study} \]
\[ C: \text{PassExam} \lor \neg \text{Study} \]
First Order Resolution

Substitutions

\[ \theta = x/\text{Bob}, y/z, \ L = \text{Father}(x, \text{Bill}) \]

\[ L\theta = \text{Father}(\text{Bob}, \text{Bill}) \]

Unifying substitutions

\[ L_1 = \text{Father}(x, y), L_2 = \text{Father}(\text{Bill}, z), \theta = x/\text{Bill}, z/y \]

\[ L_1\theta = L_2\theta = \text{Father}(\text{Bill}, y) \]

\[ C = (C_1 - L_1)\theta \cup (C_2 - L_2)\theta \]
Example

\[ C_1 = \text{White}(x) \leftarrow \text{Swan}(x) \text{ and } \]
\[ C_2 = \text{Swan}(\text{Fred}) \]

\[ L_1 = \neg \text{Swan}(x) \]
\[ L_2 = \text{Swan}(\text{Fred}) \]

\[ \theta = x / \text{Fred} \]

\[ L_1 \theta = \neg L_2 \theta = \neg \text{Swan}(\text{Fred}) \]

\[ C = \text{White}(\text{Fred}) \]
1st Order Resolution Rule

Find a literal $L_1$ from clause $C_1$, literal $L_2$ from clause $C_2$, and substitution $\theta$ such that $L_1\theta = \neg L_2\theta$

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

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

Nondeterministic because of \( C_1, \theta_1, \theta_2 \)

\[ \text{Grandchild}(y, x) \leftarrow \text{Father}(x, z) \land \text{Father}(z, y) \]
Inverse Example

Father(Tom, Bob)  GrandChild(y, x) v ¬Father(x, z) v ¬Father(z, y)

Father(Shannon, Tom)  GrandChild(Bob, x) v ¬Father(x, Tom)

GrandChild(Bob, Shannon)
Summary Inverse Resolution

Only generates “good” hypothesis as opposed to generate and test

So we would expect it to be more focused and efficient

But hobbled because can only consider a small fraction of the data when generating a hypothesis at each step
Generalization, Subsumption, Entailment

More general than - given two boolean functions $h_j(x)$ and $h_k(x)$ we say that $h_j \succeq h_k$ if and only if
$$(\forall x) h_k(x) \to h_j(x)$$

$\theta$-subsumption - Clause $C_j$ is said to $\theta$-subsume clause $C_k$ if and only if there exists a substitution $\theta$ such that $C_j \theta \subseteq C_k$

Entailment $C_j$ entails $C_k$ if and only if $C_k$ follows deductively from $C_j$
Inverse Examples

A $\theta$-subsumes B but A is not more general than B
- A: $\text{Mother}(x, y) \leftarrow \text{Father}(x, z) \land \text{Spouse}(z, y)$
- B: $\text{Mother}(x, \text{Louise}) \leftarrow \text{Father}(x, \text{Bob}) \land \text{Spouse}(\text{Bob}, y) \land \text{Female}(x)$

A entails B but A does not $\theta$-subsume B
- A: $\text{Elephant}(\text{father_of}(x)) \leftarrow \text{Elephant}(x)$
- B: $\text{Elephant}(\text{father_of}(\text{father_of}(y))) \leftarrow \text{Elephant}(y)$
Progol

Inverse entailment to generate the single most specific hypothesis

Then general to specific search using this bound

1. Restricted language
2. Sequential Covering
   1. Inverse entail most specific hypothesis
   2. General to specific search
Summary

Sequential covering learns disjunctive set of rules

First-order rules with FOIL

Inverse entailment
Questions you should be able to answer?

• What is the difference between sequential and simultaneous covering algorithms?

• Why are overlapping models important?

• How to handle unbalanced datasets?