Finding Explanations: Rules!

- Propositional Rules:
    - Rules from Decision Trees
    - Rule Sets and Lists
    - Geometrical Rules
    - CN2
- First Order Rules:
    - Inductive Logic Programming
    - FOIL

Propositional Rules

Propositional Rules

Propositional[1] Rules are simple:

[1] proposition = *truthbearer* or *statement*.
Either true or false (under an interpretation).

Propositional Rules

Propositional[1] Rules are simple:

- only atomic facts

[1] proposition = *truthbearer* or *statement*.
Either true or false (under an interpretation).

Propositional Rules

Propositional[1] Rules are simple:

- only atomic facts
- combinations of facts using only logical operators

[1] proposition = *truthbearer* or *statement*.
Either true or false (under an interpretation).

Propositional Rules

Propositional[1] Rules are simple:

- only atomic facts
- combinations of facts using only logical operators
- no variables (in contrast to first order rules)

[1] proposition = *truthbearer* or *statement*.
Either true or false (under an interpretation).

Propositional Rules

Propositional[1] Rules are simple:

- only atomic facts
- combinations of facts using only logical operators
- <u>no</u> variables (in contrast to first order rules)

Example:

$$\text{IF} \;\; x_1 \leq 10 \;\; \text{AND} \;\; x_3 = red \;\; \text{THEN class A}$$

[1] proposition = *truthbearer* or *statement*.
Either true or false (under an interpretation).

Propositional Rules

Propositional[1] Rules are simple:

- only atomic facts
- combinations of facts using only logical operators
- no variables (in contrast to first order rules)

Example:

$$\text{IF} \quad \underbrace{x_1 \leq 10 \;\; \text{AND} \;\; x_3 = red}_{\text{antecedent}} \quad \text{THEN} \quad \underbrace{\text{class A}}_{\text{consequent}}$$

[1] proposition = *truthbearer* or *statement*.
Either true or false (under an interpretation).

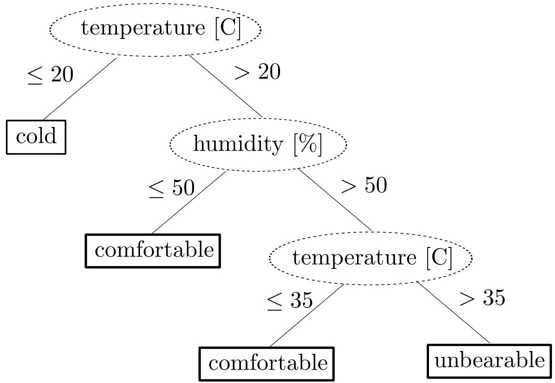
Propositional Rules

Atomic facts of propositional rules:

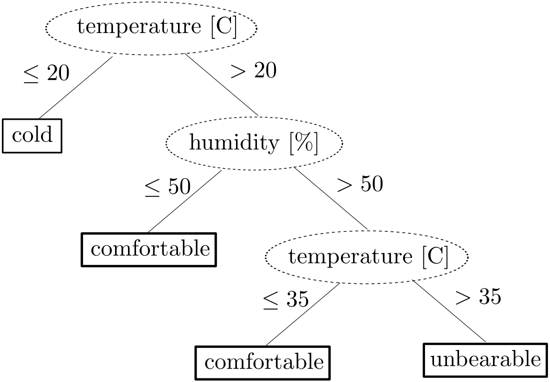
- constraints on numerical attributes, e.g. $<$, $\leq$, $=$, $\ldots$
- constraints on nominal attributes, e.g. $=$, $\in$ set
- constraints on ordinal attributes, e.g. $<$, $\in$ set, $\in$ range

Finding Propositional Rules in Data

Extracting Rules from Trees

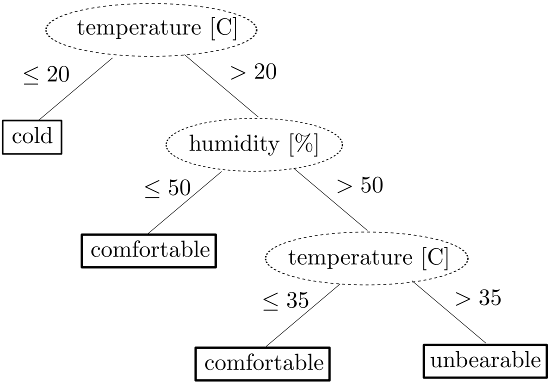


Extracting Rules from Trees



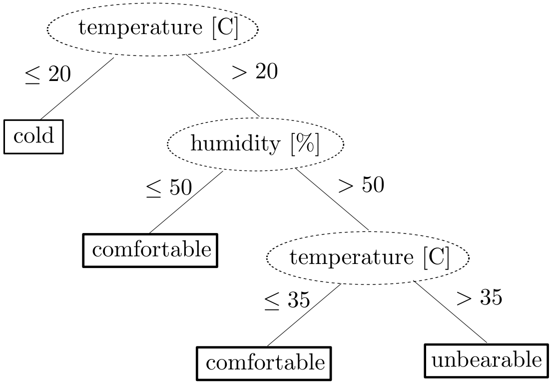
R_a : IF temperature ≤ 20 THEN class "cold"

Extracting Rules from Trees



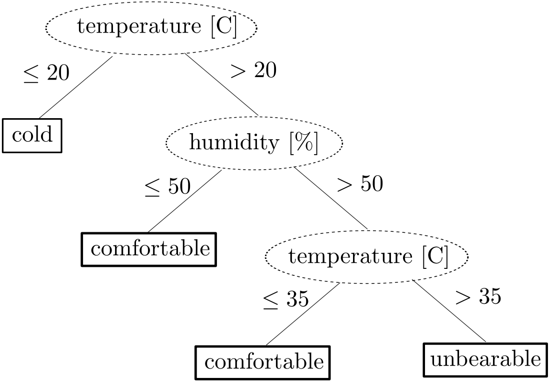
R_a : IF temperature ≤ 20 THEN class "cold"
R_b : IF temperature > 20 AND humidity ≤ 50 THEN class "comf."

Extracting Rules from Trees



R_a : IF temperature ≤ 20 THEN class "cold"
R_b : IF temperature > 20 AND humidity ≤ 50 THEN class "comf."
R_c : IF temperature $\in (20, 35]$ AND humidity > 50 THEN class "comf."

Extracting Rules from Trees



R_a : IF temperature ≤ 20 THEN class "cold"
R_b : IF temperature > 20 AND humidity ≤ 50 THEN class "comf."
R_c : IF temperature $\in (20, 35]$ AND humidity > 50 THEN class "comf."
R_d : IF temperature > 35 AND humidity > 50 THEN class "unbearable"

Extracting Rules from Trees

Rules from Decision Tree are:

Extracting Rules from Trees

Rules from Decision Tree are:

- mutually exclusive (and therefore also conflict free)

Extracting Rules from Trees

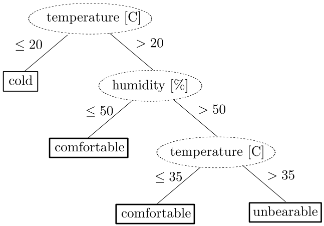
Rules from Decision Tree are:

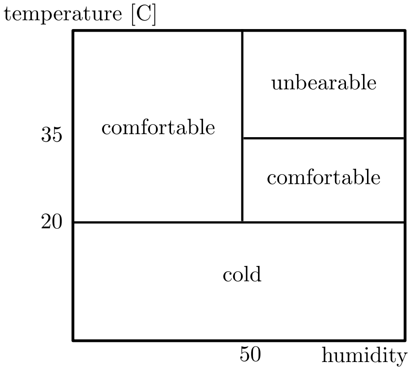
- mutually exclusive (and therefore also conflict free)
- unordered

Extracting Rules from Trees

Rules from Decision Tree are:

- mutually exclusive (and therefore also conflict free)
- unordered
- complete





Extracting Rules from Trees

Rules from Decision Tree have the usual problems
(inherited from the heuristic decision tree induction algorithms):

Extracting Rules from Trees

Rules from Decision Tree have the usual problems
(inherited from the heuristic decision tree induction algorithms):

- instability

Extracting Rules from Trees

Rules from Decision Tree have the usual problems
(inherited from the heuristic decision tree induction algorithms):

- instability

and (due to the recursive nature of the trees):

Extracting Rules from Trees

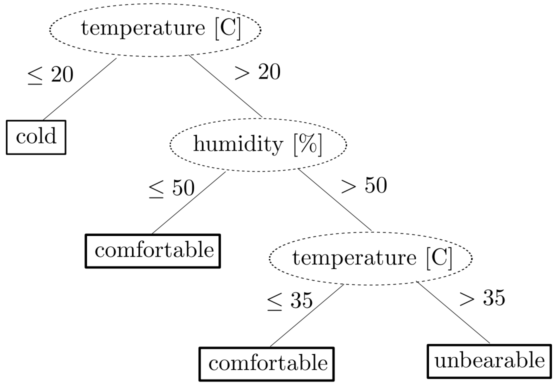
Rules from Decision Tree have the usual problems
(inherited from the heuristic decision tree induction algorithms):

- instability

and (due to the recursive nature of the trees):

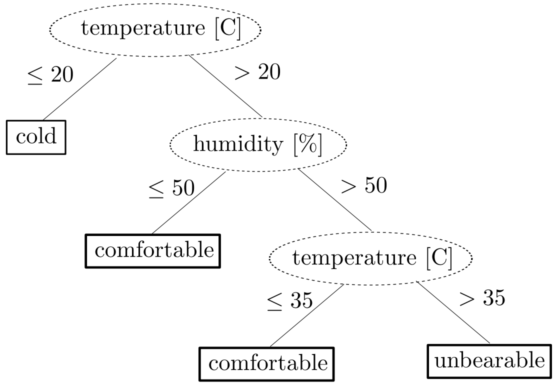
- redundancy
  (constraints on splits appear in several rules.)

Extracting Rules from Trees



Non-redundant rule set:

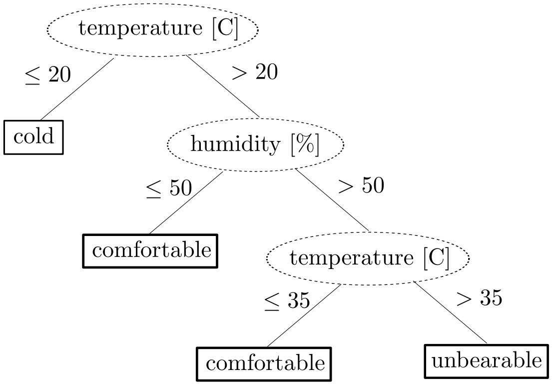
Extracting Rules from Trees



Non-redundant rule set:

R_1 : IF temperature ≤ 20 THEN class "cold"

Extracting Rules from Trees

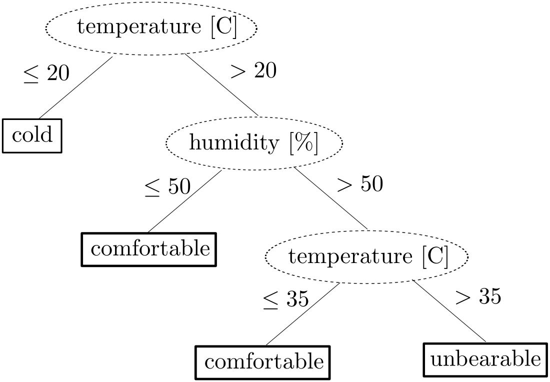


Non-redundant rule set:

R_1 : IF temperature ≤ 20 THEN class "cold"

R_2 : IF humidity ≤ 50 THEN class "comfortable"

Extracting Rules from Trees



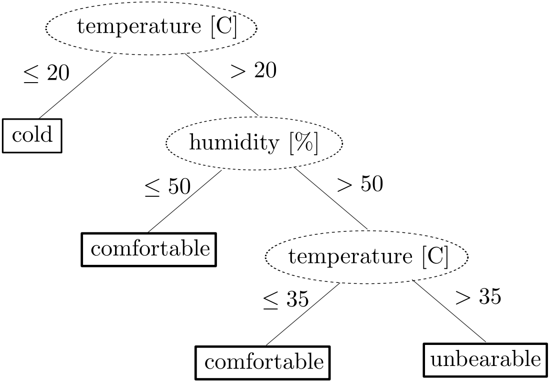
Non-redundant rule set:

R_1 : IF temperature ≤ 20 THEN class "cold"

R_2 : IF humidity ≤ 50 THEN class "comfortable"

R_3 : IF temperature ≤ 35 THEN class "comfortable"

Extracting Rules from Trees



Non-redundant rule set:

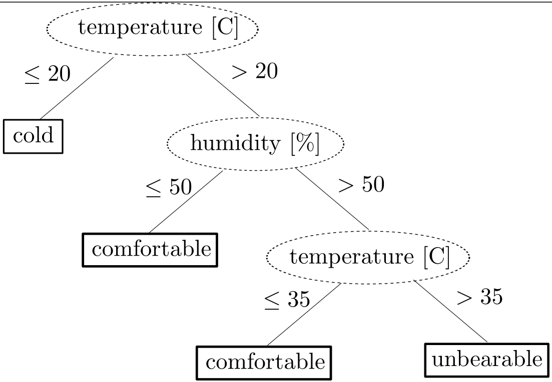
R_1 : IF temperature ≤ 20 THEN class "cold"

R_2 : IF humidity ≤ 50 THEN class "comfortable"

R_3 : IF temperature ≤ 35 THEN class "comfortable"

R_4 : class $=$ "unbearable"

Extracting Ordered Rules from Trees



Non-redundant rule set:

R_1 : IF temperature ≤ 20 THEN class "cold"

R_2 : IF humidity ≤ 50 THEN class "comfortable"

R_3 : IF temperature ≤ 35 THEN class "comfortable"

R_4 : class = "unbearable"

↑ order of rules matters!

Extracting Ordered Rules from Trees

- previous example was heavily imbalanced tree.
- more balanced tree results in nested ordering of rules.
- in normal trees ordered rule extraction is considerably more complex.

Learning Propositional Rules

Categorization of more general Propositional Rule Learners:

Learning Propositional Rules

Categorization of more general Propositional Rule Learners:

- Supported Attribute Types

Learning Propositional Rules

Categorization of more general Propositional Rule Learners:

- Supported Attribute Types
  - nominal only $\Rightarrow$ relatively small hypothesis space

Learning Propositional Rules

Categorization of more general Propositional Rule Learners:

- Supported Attribute Types
  - nominal only ⇒ relatively small hypothesis space
  - numerical only ⇒ "geometrical" rule learners

Learning Propositional Rules

Categorization of more general Propositional Rule Learners:

- Supported Attribute Types
  - nominal only $\Rightarrow$ relatively small hypothesis space
  - numerical only $\Rightarrow$ "geometrical" rule learners
  - mixed attributes $\Rightarrow$ more complex heuristics needed

Learning Propositional Rules

Categorization of more general Propositional Rule Learners:

- Supported Attribute Types
  - nominal only $\Rightarrow$ relatively small hypothesis space
  - numerical only $\Rightarrow$ "geometrical" rule learners
  - mixed attributes $\Rightarrow$ more complex heuristics needed
- Learning Strategy

Learning Propositional Rules

Categorization of more general Propositional Rule Learners:

- Supported Attribute Types
  - nominal only ⇒ relatively small hypothesis space
  - numerical only ⇒ "geometrical" rule learners
  - mixed attributes ⇒ more complex heuristics needed
- Learning Strategy
  - specializing

Learning Propositional Rules

Categorization of more general Propositional Rule Learners:

- Supported Attribute Types
  - nominal only $\Rightarrow$ relatively small hypothesis space
  - numerical only $\Rightarrow$ "geometrical" rule learners
  - mixed attributes $\Rightarrow$ more complex heuristics needed
- Learning Strategy
  - specializing
  - generalizing

Learning Propositional Rules: Generalizing

An example for a rule generalization:

Learning Propositional Rules: Generalizing

An example for a rule generalization:

Given a training instance $(\vec{x}_1, k)$ with

$$\vec{x}_1 = (12, 3.5, \mathrm{red})$$

Learning Propositional Rules: Generalizing

An example for a rule generalization:

Given a training instance $(\vec{x}_1, k)$ with

$$\vec{x}_1 = (12, 3.5, \mathrm{red})$$

an initial, special rule could look like:

$$\text{IF} \;\; x_1 = 12 \text{ AND } x_2 = 3.5 \text{ AND } x_3 = '\mathrm{red}' \;\; \text{THEN class } k$$

Learning Propositional Rules: Generalizing

An example for a rule generalization:

Given a training instance $(\vec{x}_1, k)$ with

$$\vec{x}_1 = (12, 3.5, \text{red})$$

an initial, special rule could look like:

$$\text{IF} \quad x_1 = 12 \text{ AND } x_2 = 3.5 \text{ AND } x_3 = '\text{red}' \quad \text{THEN class } k$$

For a second example $(\vec{x}_2, k)$ with

$$\vec{x}_2 = (12.3, 3.5, \text{blue})$$

Learning Propositional Rules: Generalizing

An example for a rule generalization:

Given a training instance $(\vec{x}_1, k)$ with

$$\vec{x}_1 = (12, 3.5, \text{red})$$

an initial, special rule could look like:

$$\text{IF} \quad x_1 = 12 \text{ AND } x_2 = 3.5 \text{ AND } x_3 = '\text{red}' \quad \text{THEN class } k$$

For a second example $(\vec{x}_2, k)$ with

$$\vec{x}_2 = (12.3, 3.5, \text{blue})$$

the rule is generalized:

$$\text{IF } x_1 \in [12, 12.3] \text{ AND } x_2 = 3.5 \text{ AND } x_3 \in \{'\text{red}', '\text{blue}'\} \text{ THEN class } k$$

Learning Propositional Rules: Generalizing

Two main options for Generalization exist:

Learning Propositional Rules: Generalizing

Two main options for Generalization exist:

- Generalize existing rule to cover one more pattern

Learning Propositional Rules: Generalizing

Two main options for Generalization exist:

- Generalize existing rule to cover one more pattern
- Merge two existing rules (the more general case).

Learning Propositional Rules: Generalizing

Two main options for Generalization exist:

- Generalize existing rule to cover one more pattern
- Merge two existing rules (the more general case).

The resulting training algorithms generally are:

Learning Propositional Rules: Generalizing

Two main options for Generalization exist:

- Generalize existing rule to cover one more pattern
- Merge two existing rules (the more general case).

The resulting training algorithms generally are:

- greedy (complete search of merge tree infeasible).
  (note difference to Find-S which explores entire space!)

Learning Propositional Rules: Generalizing

Two main options for Generalization exist:

- Generalize existing rule to cover one more pattern
- Merge two existing rules (the more general case).

The resulting training algorithms generally are:

- greedy (complete search of merge tree infeasible).
  (note difference to Find-S which explores entire space!)
- differ in
  - the choice of rules/patterns to merge

Learning Propositional Rules: Generalizing

Two main options for Generalization exist:

- Generalize existing rule to cover one more pattern
- Merge two existing rules (the more general case).

The resulting training algorithms generally are:

- greedy (complete search of merge tree infeasible).
  (note difference to Find-S which explores entire space!)
- differ in
  - the choice of rules/patterns to merge
  - the used stopping criteria.

Remark: Specializing Propositional Rules

Specializing Rule Learners follow the same principle.

Remark: Specializing Propositional Rules

Specializing Rule Learners follow the same principle.

- they start with very general rules

$$\text{IF true THEN class } k.$$

Remark: Specializing Propositional Rules

Specializing Rule Learners follow the same principle.

- they start with very general rules

$$\text{IF} \;\; \text{true} \;\; \text{THEN class } k.$$

- and iteratively specialize the rule.

Finding Sets of Rules

So far we only generalized/specialized one rule.

Finding Sets of Rules

So far we only generalized/specialized <u>one</u> rule.

- Most real world data sets are too complex to be explained by one rule only.

Finding Sets of Rules

So far we only generalized/specialized one rule.

- Most real world data sets are too complex to be explained by one rule only.
- Many rule learning algorithms wrap the learning of one rule into an outer loop based on set covering strategy (sequential covering):

Finding Sets of Rules

So far we only generalized/specialized one rule.

- Most real world data sets are too complex to be explained by one rule only.
- Many rule learning algorithms wrap the learning of one rule into an outer loop based on set covering strategy (sequential covering):
  - attempts to build most important rules first

Finding Sets of Rules

So far we only generalized/specialized one rule.

- Most real world data sets are too complex to be explained by one rule only.
- Many rule learning algorithms wrap the learning of one rule into an outer loop based on set covering strategy (sequential covering):
  - attempts to build most important rules first
  - iteratively adds smaller / less important rules

Geometrical Rule Learners

Geometrical Rule Learners

- limited to numerical attributes (comparable ranges help, too)
- Goal:
  - Find rectangular (axes parallel) area(s) one by one that are occupied only by patterns of one class.
  - each such area represents a rule:

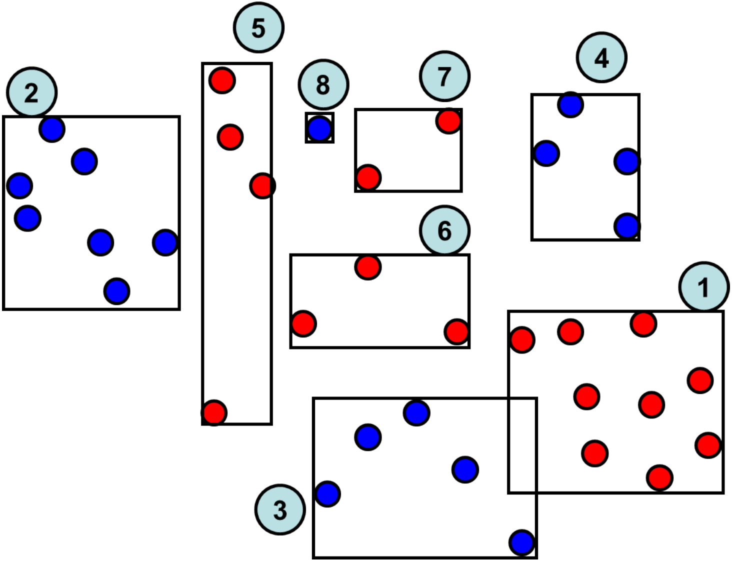
$$\text{IF } x_1 \in [a_1, b_1] \wedge \cdots \wedge x_n \in [a_n, b_n] \text{ THEN class } k$$

- Creates one rule after the other until no more useful rules can be built.

  To find one rule:
  - draw random starting point
  - form most specific rectangular hypothesis covering this point
  - While possible
    - find nearest neighbor of same class
    - generalize hypothesis (rectangle) to include this point
      (the latter may not be possible for all neighbors of the same class)

Geometrical Rule Learners

An example:



Geometrical Rule Learners: RecBF Learner

- Goal: Finding specific and general rules
  - Allows to estimate classification certainty
- RecBF algorithm: motivated by Neural Network Learning method
- Finds most specific within each (locally) most general rule
- Iterative algorithm
- (Much) faster than rule-by-rule approach.

BuildRuleSet

Algorithm RecBF(T)

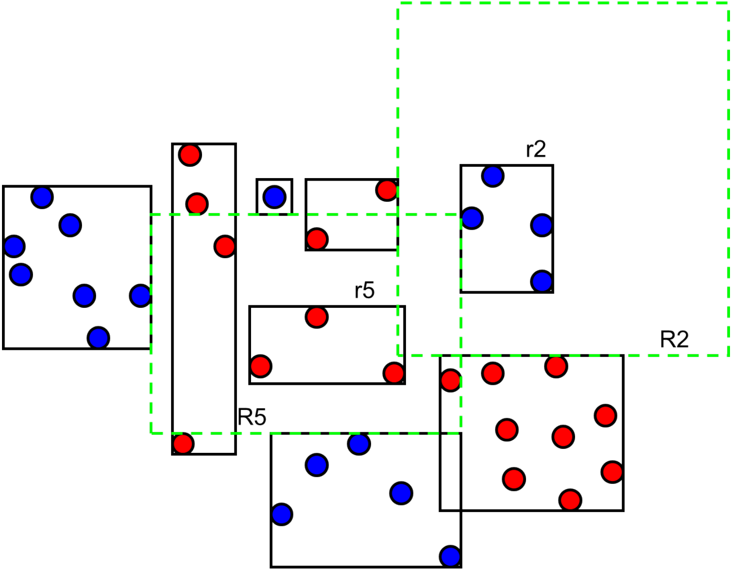
input: training data T

output: a general and specific rule set R matching T

$R = \emptyset$
while rules R are not stable
    $\forall(\vec{x}, k) \in T$
    if $\exists R^k \in R : \mathrm{covers}(R^k, \vec{x})$ **then**
        // Covered:
        $R^k.w++$ // increase weight
        $\mathrm{cover}(R^k, \vec{x})$
    else
        // Commit:
        $R = R \cup \mathrm{newRule}(\vec{x})$
    endif
    $\forall R^{k'} \in R : k \neq k' \wedge \mathrm{covers}(R^k, \vec{x})$:
        $\mathrm{shrink}(R^{k'}, \vec{x})$

RecBF Learner

An example:

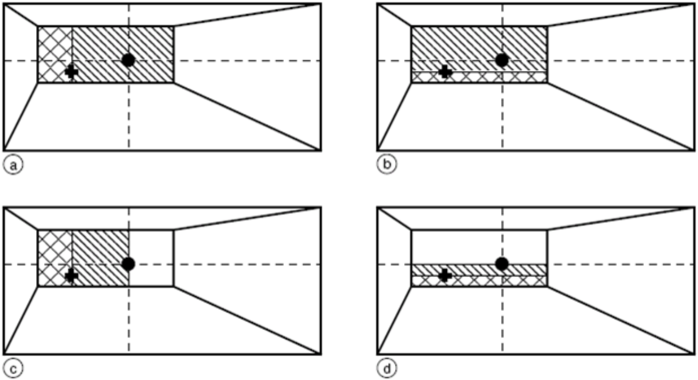


RecBF Learner: Operations

- Rule stability:
  - Algorithm converges guaranteed if training data is conflict free (Maximum: n iterations for n training patterns)
- Covered:
  - simple boundary test
  - Make sure specific rule R covers new $\vec{x}$: boundary expansion
- Commit:
  - Simple insert of most general rule R with corresponding most specific rule, centered on new training pattern
- Shrink:
  - Reducing rectangle to exclude conflicting pattern: n choices.
  - Heuristics:
    - maximize remaining volume
    - Avoid shrinkage of specific rule of R

RecBF Learner: Shrink

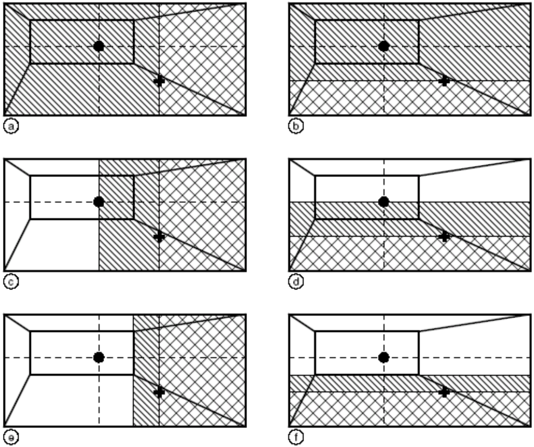
Options for Specialization of specialized rules:



(a,b): entire region considered,
(c,d): regions to anchor considered.

RecBF Learner: Shrink

Options for Specialization of general rules:



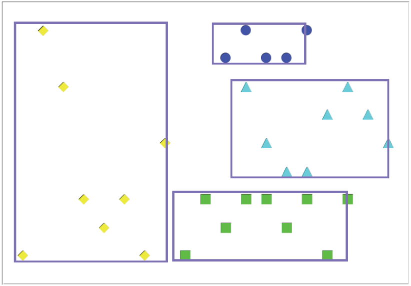
(a,b): entire region considered,
(c,d): regions to anchor considered,
(e,f): regions to “core” considered.

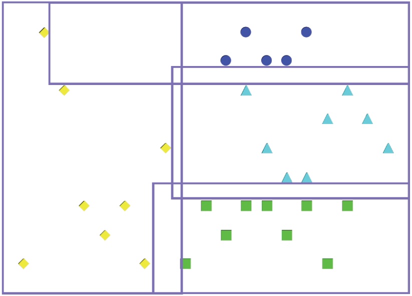
RecBF Learner: Operations

- Problems:
  - Depends on order of training examples
  - Shrink-procedure based on heuristics
- Properties
  - Explains all training pattern correctly
  - Most general rules depend on few attributes only
  - Most specific rules depend on all attributes

Note: Geometrical Sets of Rules

Examples[2] of Specialized and Generalized Rules:

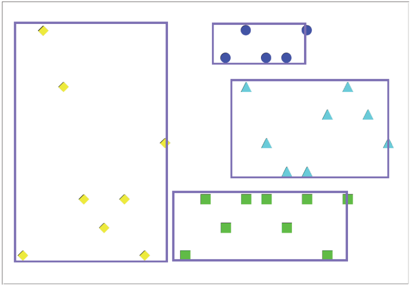


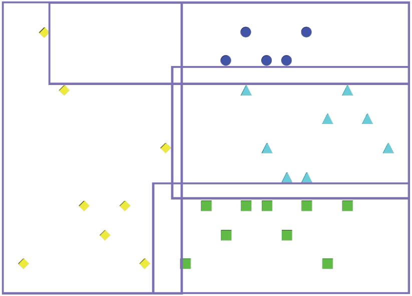


[2]images from Peter Flach

Note: Geometrical Sets of Rules

Examples[2] of Specialized and Generalized Rules:





Are there more general (or special) rule sets?

[2] images from Peter Flach

Back to Propositional Rules:
CN2

The CN2 Rule Learning Algorithm

Peter Clark and Tim Niblett: **The CN2 Induction Algorithm**,
Machine Learning Journal, 3(4):261-283, 1989.

- prominent, early example of rule learning algorithm
- set covering approach
- greedy algorithm rule specialization
- simple heuristic for "most important" rule selection.

BuildRuleSet

Algorithm BuildRuleSet$(D, p_{\min})$

input:	training data D
parameter:	performance threshold $p_{\min}$
output:	a rule set R matching D with performance $\geq p_{\min}$

$R = \emptyset$
$D_{\text{rest}} = D$
while $(\text{Performance}(R, D_{\text{rest}}) < p_{\min})$
    $r = \text{FindOneGoodRule}(D_{\text{rest}})$
    $R = R \cup \{r\}$
    $D_{\text{rest}} = D_{\text{rest}} - \text{covered}(r, D_{\text{rest}})$
endwhile
return R

Alternative implementations by varying:

BuildRuleSet

Algorithm BuildRuleSet($D, p_{\min}$)

input:	training data D
parameter:	performance threshold $p_{\min}$
output:	a rule set R matching D with performance $\geq p_{\min}$

$R = \emptyset$
$D_{\text{rest}} = D$
while $(\text{Performance}(R, D_{\text{rest}}) < p_{\min})$
    $r = \text{FindOneGoodRule}(D_{\text{rest}})$
    $R = R \cup \{r\}$
    $D_{\text{rest}} = D_{\text{rest}} - \text{covered}(r, D_{\text{rest}})$
endwhile
return R

Alternative implementations by varying:

- performance measure?

BuildRuleSet

Algorithm BuildRuleSet($D, p_{\min}$)

input:	training data D
parameter:	performance threshold $p_{\min}$
output:	a rule set R matching D with performance $\geq p_{\min}$

$R = \emptyset$
$D_{\text{rest}} = D$
while $(\text{Performance}(R, D_{\text{rest}}) < p_{\min})$
  $r = \text{FindOneGoodRule}(D_{\text{rest}})$
  $R = R \cup \{r\}$
  $D_{\text{rest}} = D_{\text{rest}} - \text{covered}(r, D_{\text{rest}})$
endwhile
return R

Alternative implementations by varying:

- performance measure?
- strategy for rule finding?

BuildRuleSet

Algorithm BuildRuleSet$(D, p_{\min})$

input:	training data D
parameter:	performance threshold $p_{\min}$
output:	a rule set R matching D with performance $\geq p_{\min}$

```
R = ∅
D_rest = D
while (Performance(R, D_rest) < p_min)
    r = FindOneGoodRule(D_rest)
    R = R ∪ {r}
    D_rest = D_rest − covered(r, D_rest)
endwhile
return R
```

Alternative implementations by varying:

- performance measure?
- strategy for rule finding?

  (We could just use a decision tree learner and build only the 'best branch...)

Remark: Decision Tree Based Good Rule Finding

- Following (more than) one branch resembles a general-to-specific beam search:
  - Hypothesis (rule) is increasingly specified
  - At each branching point all possible specializations are considered
  - Choice of best branch can be made based on information gain, coverage, or mix of both
  - Class (rule post condition) depends on majority class at each point
- Greedy algorithm can produce sub-optimal solution
  - Beam search produces k candidate solutions (at each level only the k best branches are kept)
- Sometimes only learning rules of one class is sufficient (e.g. to model a minority class), default class = majority class.

FindOneGoodRule

Algorithm FindOneGoodRule(D_{rest})

input: (subset of) training data D_{rest}

output: one good rule r explaining some instances of the training data

```
h_best = true   // most general hypothesis
H_candidates = {h_best}
while H_candidates ≠ ∅
    H_candidates = specialize(H_candidates)
    h_best = arg max_{h ∈ H_candidates ∪ {h_best}} {Performance(h, D_rest)}
    update(H_candidates)   // clean up
endwhile
return 'IF h_best THEN arg max_k{|covered_k(h_best, D_rest)|}'
```

$h_{\text{best}} = \text{true}$ // most general hypothesis

$H_{\text{candidates}} = \{h_{\text{best}}\}$

while $H_{\text{candidates}} \neq \emptyset$

$\quad H_{\text{candidates}} = \text{specialize}\,(H_{\text{candidates}})$

$\quad h_{\text{best}} = \arg\max_{h \in H_{\text{candidates}} \cup \{h_{\text{best}}\}} \{\text{Performance}(h, D_{\text{rest}})\}$

$\quad \text{update}(H_{\text{candidates}})$ // clean up

endwhile

return $'\text{IF } h_{\text{best}} \text{ THEN } \arg\max_k\{|\text{covered}_k(h_{\text{best}}, D_{\text{rest}})|\}'$

Heuristics for FindOneGoodRule

How do we evaluate the accuracy A of a rule?

Heuristics for FindOneGoodRule

How do we evaluate the accuracy A of a rule?

- Base Assumption:

$$A('\text{IF Conditions THEN class } k') = p(k|\text{Conditions})$$

Heuristics for FindOneGoodRule

How do we evaluate the accuracy A of a rule?

- Base Assumption:

$$A('\text{IF Conditions THEN class } k') = p(k|\text{Conditions})$$

- Estimating the probability using relative frequencies:

$$p(k|\text{Conditions}) = \frac{\#\text{covered correct}}{\#\text{covered total}}$$

Probability Estimates

Probability Estimates

- Relative frequency of covered correctly:

$$p(k|R) = \frac{\#\text{covered correct}}{\#\text{covered total}}$$

Problems with small samples.

Probability Estimates

- Relative frequency of covered correctly:

$$p(k|R) = \frac{\#\text{covered correct}}{\#\text{covered total}}$$

Problems with small samples.

- Laplace estimate:

$$p(k|R) = \frac{\#\text{covered correct} + 1}{\#\text{covered total} + \#\text{classes}}$$

Assumes uniform prior distribution of classes.

Probability Estimates

- Relative frequency of covered correctly:

$$p(k|R) = \frac{\#\text{covered correct}}{\#\text{covered total}}$$

  Problems with small samples.

- Laplace estimate:

$$p(k|R) = \frac{\#\text{covered correct} + 1}{\#\text{covered total} + \#\text{classes}}$$

  Assumes uniform prior distribution of classes.

- m-estimate:

$$p(k|R) = \frac{\#\text{covered correct} + m \cdot p(k)}{\#\text{covered total} + m}$$

  ...

Probability Estimates (cont.)

...

- m-estimate:

$$p(k|R) = \frac{\#\text{covered correct} + m \cdot p(k)}{\#\text{covered total} + m}$$

  - special case: $p(k) = 1/\#\text{classes}$, $m = \#\text{classes}$
  - takes into account prior class probabilities
  - independent of number of classes
  - m is domain dependent (more noise, larger m)

Other Search Heuristics

- Expected accuracy on positives:

$$A(R) = p(k|R)$$

Other Search Heuristics

- Expected accuracy on positives:

$$A(R) = p(k|R)$$

- Entropy (#bits needed to specify that example is covered by R):

$$I(R) = -\log_2 p(k|R)$$

Other Search Heuristics

- Expected accuracy on positives:

$$A(R) = p(k|R)$$

- Entropy (#bits needed to specify that example is covered by R):

$$I(R) = -\log_2 p(k|R)$$

- Accuracy gain (increase in expected accuracy):

$$AG(R', R) = p(k|R') - p(k|R)$$

Other Search Heuristics

- Expected accuracy on positives:

$$A(R) = p(k|R)$$

- Entropy (#bits needed to specify that example is covered by R):

$$I(R) = -\log_2 p(k|R)$$

- Accuracy gain (increase in expected accuracy):

$$AG(R', R) = p(k|R') - p(k|R)$$

- Information gain (decrease in information needed):

$$IG(R', R) = \log_2 p(k|R') + \log_2 p(k|R)$$

Other Search Heuristics

- Expected accuracy on positives:

$$A(R) = p(k|R)$$

- Entropy (#bits needed to specify that example is covered by R):

$$I(R) = -\log_2 p(k|R)$$

- Accuracy gain (increase in expected accuracy):

$$AG(R', R) = p(k|R') - p(k|R)$$

- Information gain (decrease in information needed):

$$IG(R', R) = \log_2 p(k|R') + \log_2 p(k|R)$$

- Weighted measures in order to favor more general rules:

$$W\_G(R', R) = \frac{\#\text{correct covered by } R'}{\#\text{correct covered by } R} \cdot \_G(R', R)$$

Propositional Rule Induction

Issues with most propositional rule learners:

Issues with most propositional rule learners:

- heuristics for real world data often fail

Propositional Rule Induction

Issues with most propositional rule learners:

- heuristics for real world data often fail
  - too many dimensions make greedy constraint picking difficult

Propositional Rule Induction

Issues with most propositional rule learners:

- heuristics for real world data often fail
  - too many dimensions make greedy constraint picking difficult
  - in real world feature spaces often patterns of same class not "axes parallel"

Propositional Rule Induction

Issues with most propositional rule learners:

- heuristics for real world data often fail
  - too many dimensions make greedy constraint picking difficult
  - in real world feature spaces often patterns of same class not “axes parallel”
- sharp boundaries for noisy data unsuited

  $\Rightarrow$ Fuzzy Rules

  $\Rightarrow$ Hierarchical Rule Systems

Propositional Rule Induction

Issues with most propositional rule learners:

- heuristics for real world data often fail
  - too many dimensions make greedy constraint picking difficult
  - in real world feature spaces often patterns of same class not "axes parallel"
- sharp boundaries for noisy data unsuited
  - $\Rightarrow$ Fuzzy Rules
  - $\Rightarrow$ Hierarchical Rule Systems
- Propositional Rules not very expressive...

Limitations of Propositional Rules

Propositional rule learners can not express rules such as:

$$\text{IF } x \text{ is Father of } y \text{ AND } y \text{ is female THEN } y \text{ is Daughter of } x$$

Limitations of Propositional Rules

Propositional rule learners can not express rules such as:

$$\text{IF } \ x \ \text{ is Father of } \ y \ \text{ AND } \ y \ \text{ is female } \ \text{THEN } \ y \ \text{ is Daughter of } \ x$$

They would need to "cover" training examples for all possible (x, y) combinations.

Limitations of Propositional Rules

Propositional rule learners can not express rules such as:

$$\text{IF} \;\; x \;\; \text{is Father of} \;\; y \;\; \text{AND} \;\; y \;\; \text{is female} \;\; \text{THEN} \;\; y \;\; \text{is Daughter of} \;\; x$$

They would need to “cover” training examples for all possible (x, y) combinations.

For this, other types of rules are more approriate.

First Order Rules

First Order Rules

First Order Rules differ from propositional logic by their use of variables. FOL is also based on a number of base constructs:

First Order Rules

First Order Rules differ from propositional logic by their use of variables. FOL is also based on a number of base constructs:

- constants, e.g. Bob, Luise, red, green,

First Order Rules

First Order Rules differ from propositional logic by their use of variables. FOL is also based on a number of base constructs:

- constants, e.g. Bob, Luise, red, green,
- variables, e.g. x and y in the example above,

First Order Rules

First Order Rules differ from propositional logic by their use of variables. FOL is also based on a number of base constructs:

- constants, e.g. Bob, Luise, red, green,
- variables, e.g. x and y in the example above,
- predicates, e.g. $\text{is\_father}(x, y)$, which produce truth values as a result

First Order Rules

First Order Rules differ from propositional logic by their use of variables. FOL is also based on a number of base constructs:

- constants, e.g. Bob, Luise, red, green,
- variables, e.g. x and y in the example above,
- predicates, e.g. $\text{is\_father}(x, y)$, which produce truth values as a result
- functions, e.g. $\text{age}(x)$, which produce constants.

First Order Rules

Using constants, variables, predicates, and functions we can construct:

First Order Rules

Using constants, variables, predicates, and functions we can construct:

- terms: constants, variables, or functions applied to a term,

First Order Rules

Using constants, variables, predicates, and functions we can construct:

- terms: constants, variables, or functions applied to a term,
- literals: predicates (or negations of predicates) applied to any set of terms,

First Order Rules

Using constants, variables, predicates, and functions we can construct:

- terms: constants, variables, or functions applied to a term,
- literals: predicates (or negations of predicates) applied to any set of terms,
- ground literals: literals that do not contain a variable,

First Order Rules

Using constants, variables, predicates, and functions we can construct:

- terms: constants, variables, or functions applied to a term,
- literals: predicates (or negations of predicates) applied to any set of terms,
- ground literals: literals that do not contain a variable,
- clauses: disjunctions of literals whose variables are universally quantified.

First Order Rules

Using constants, variables, predicates, and functions we can construct:

- terms: constants, variables, or functions applied to a term,
- literals: predicates (or negations of predicates) applied to any set of terms,
- ground literals: literals that do not contain a variable,
- clauses: disjunctions of literals whose variables are universally quantified.

  (Reminder:

  - $\forall$: the universal quantifier, e.g. $\forall x, y : \mathrm{Daughter}(x, y) \rightarrow \mathrm{female(x)}$
  - $\exists$: existential quantifier, e.g. $\forall x : \exists y : \mathrm{female(x)} \rightarrow \mathrm{Daughter}(x, y)$

)

First Order Rules

Using constants, variables, predicates, and functions we can construct:

- terms: constants, variables, or functions applied to a term,
- literals: predicates (or negations of predicates) applied to any set of terms,
- ground literals: literals that do not contain a variable,
- clauses: disjunctions of literals whose variables are universally quantified.

  (Reminder:

  - $\forall$: the universal quantifier, e.g. $\forall x, y : \mathrm{Daughter}(x, y) \rightarrow \mathrm{female(x)}$
  - $\exists$: existential quantifier, e.g. $\forall x : \exists y : \mathrm{female(x)} \rightarrow \mathrm{Daughter}(x, y)$

)
- horn clauses: clauses with at most one positive literal.

Horn Clauses

Horn clauses are especially interesting because any disjunction with at most one positive literal can be written as

$$H \vee \neg L_1 \vee \cdots \vee \neg L_n$$

Horn Clauses

Horn clauses are especially interesting because any disjunction with at most one positive literal can be written as

$$\begin{array}{rl} & H \vee \neg L_1 \vee \cdots \vee \neg L_n \\ \hat{=} & H \Rightarrow (L_1 \wedge \cdots \wedge L_n) \end{array}$$

Horn Clauses

Horn clauses are especially interesting because any disjunction with at most one positive literal can be written as

$$\begin{aligned} & H \vee \neg L_1 \vee \cdots \vee \neg L_n \\ \hat{=} \quad & H \Rightarrow (L_1 \wedge \cdots \wedge L_n) \\ \hat{=} \quad & \text{IF} \quad L_1 \quad \text{AND ...AND} \quad L_2 \quad \text{THEN} \quad H. \end{aligned}$$

Horn Clauses

Horn clauses are especially interesting because any disjunction with at most one positive literal can be written as

$$\begin{aligned} & H \vee \neg L_1 \vee \cdots \vee \neg L_n \\ \hat{=} \quad & H \Rightarrow (L_1 \wedge \cdots \wedge L_n) \\ \hat{=} \quad & \text{IF} \;\; L_1 \;\; \text{AND} \ldots\text{AND} \;\; L_2 \;\; \text{THEN} \;\; H. \end{aligned}$$

Horn Clauses express Rules

H, the head of the rule is the consequent,
all L_i together form the body or antecedent.

Horn Clauses

Horn clauses are especially interesting because any disjunction with at most one positive literal can be written as

$$\begin{aligned} & H \vee \neg L_1 \vee \cdots \vee \neg L_n \\ \hat{=} \quad & H \Rightarrow (L_1 \wedge \cdots \wedge L_n) \\ \hat{=} \quad & \text{IF} \quad L_1 \quad \text{AND} \ \ldots \text{AND} \quad L_2 \quad \text{THEN} \quad H. \end{aligned}$$

Horn Clauses express Rules

H, the <u>head</u> of the rule is the consequent,
all L_i together form the <u>body</u> or antecedent.

Horn Clauses

Horn clauses are especially interesting because any disjunction with at most one positive literal can be written as

$$
\begin{aligned}
& H \vee \neg L_1 \vee \cdots \vee \neg L_n \\
\hat{=}\quad & H \Rightarrow (L_1 \wedge \cdots \wedge L_n) \\
\hat{=}\quad & \text{IF} \quad L_1 \quad \text{AND} \ldots \text{AND} \quad L_2 \quad \text{THEN} \quad H.
\end{aligned}
$$

Horn Clauses express Rules

H, the head of the rule is the consequent,
all L_i together form the body or antecedent.

Horn clauses are also used to express Prolog programs ⇒ Learning First Order Rules is often called "Inductive Logic Programming" (ILP).

First Order Rule Types

A rule body is satisfied if at least one binding exists that satisfies all literals.

Binding (of variables) = substitution of variables by appropriate constants.

First Order Rule Types

A rule body is satisfied if at least one binding exists that satisfies all literals.

Binding (of variables) = substitution of variables by appropriate constants.

A few examples:

- simple rules:
  IF x is Parent of y AND y is male
  THEN x is Father of y

First Order Rule Types

A rule body is satisfied if at least one binding exists that satisfies all literals.

Binding (of variables) = substitution of variables by appropriate constants.

A few examples:

- simple rules:
  IF x is Parent of y AND y is male
  THEN x is Father of y
- existentially qualified variables (z in this case):
  IF y is Parent of z AND z is Parent of x
  THEN x is Grandchild of y

First Order Rule Types

A rule body is satisfied if at least one binding exists that satisfies all literals.

Binding (of variables) = substitution of variables by appropriate constants.

A few examples:

- simple rules:
  IF x is Parent of y AND y is male
  THEN x is Father of y
- existentially qualified variables (z in this case):
  IF y is Parent of z AND z is Parent of x
  THEN x is Grandchild of y
- Recursive rules:
  IF x is Parent of z AND z is Anchestor of y
  THEN x is Anchestor of y
  IF x is Parent of y THEN x is Anchestor of y

Learning First Order Rules: FOIL

J.R. Quinlan and R.M. Cameron-Jones: **FOIL: A Midterm Report**, *Proceedings of the European Conference on Machine Learning*, 3-20, 1993.

- FOIL (First Order Inductive Learning method) was one of the first algorithms published.

Learning First Order Rules: FOIL

J.R. Quinlan and R.M. Cameron-Jones: **FOIL: A Midterm Report**, *Proceedings of the European Conference on Machine Learning*, 3-20, 1993.

- FOIL (First Order Inductive Learning method) was one of the first algorithms published.
- one (dramatic) restriction: no functions!

Learning First Order Rules: FOIL

J.R. Quinlan and R.M. Cameron-Jones: **FOIL: A Midterm Report**, *Proceedings of the European Conference on Machine Learning*, 3-20, 1993.

- FOIL (First Order Inductive Learning method) was one of the first algorithms published.
- one (dramatic) restriction: no functions!
- one (easy) extension: literals in body can be negated.

Learning First Order Rules: FOIL

- **Algorithm** BuildRuleSet($\cdot$): remains the same (sequential set covering)

Learning First Order Rules: FOIL

- **Algorithm** BuildRuleSet(·): remains the same (sequential set covering)
- **Algorithm** FindOneGoodRule(·): slightly changed.
  Rule specialization adds literals one by one by doing one of following:

- **Algorithm** BuildRuleSet(·): remains the same (sequential set covering)
- **Algorithm** FindOneGoodRule(·): slightly changed.
  Rule specialization adds literals one by one by doing one of following:
  - adding `P(v1,...,vr)`, where P is any predicate name occurring in the set of available predicates. At least one of the variables must already be present in the original rule, the others can be either new or existing;

Learning First Order Rules: FOIL

- **Algorithm** BuildRuleSet(·): remains the same (sequential set covering)
- **Algorithm** FindOneGoodRule(·): slightly changed.
  Rule specialization adds literals one by one by doing one of following:
  - adding `P(v1,...,vr)`, where P is any predicate name occurring in the set of available predicates. At least one of the variables must already be present in the original rule, the others can be either new or existing;
  - adding `Equal(x,y)`, where x and y are variables already present in the rule; or

Learning First Order Rules: FOIL

- **Algorithm** BuildRuleSet($\cdot$): remains the same (sequential set covering)
- **Algorithm** FindOneGoodRule($\cdot$): slightly changed.
  Rule specialization adds literals one by one by doing one of following:
  - adding `P(v1,...,vr)`, where P is any predicate name occurring in the set of available predicates. At least one of the variables must already be present in the original rule, the others can be either new or existing;
  - adding `Equal(x,y)`, where x and y are variables already present in the rule; or
  - negating of either of the above forms of literals.

Learning First Order Rules: FOIL

- **Algorithm** BuildRuleSet(·): remains the same (sequential set covering)
- **Algorithm** FindOneGoodRule(·): slightly changed.
  Rule specialization adds literals one by one by doing one of following:
  - adding `P(v1,...,vr)`, where P is any predicate name occurring in the set of available predicates. At least one of the variables must already be present in the original rule, the others can be either new or existing;
  - adding `Equal(x,y)`, where x and y are variables already present in the rule; or
  - negating of either of the above forms of literals.
- How does FOIL pick the “best” specialization?

FOIL: Guiding the Search

FoilGain used to evaluation contribution of new literal L to rule R:

$$\text{FoilGain}(L, R) = t\left(-\log_2 \frac{p'}{p'+n'} - \log_2 \frac{p}{p+n}\right)$$

- R: rule
- p: number of positive bindings of R
- n: number of negative bindings of R
- L: new literal
- R': new rule (R with added L)
- p': number of positive bindings of R'
- n': number of negative bindings of R'

If L introduces a new variable, any original binding is considered to be covered if at least some of the (extended) binding of R is present in the bindings of R'.

FOIL: Example

- Target literal: GrandDaughter(x, y)
  (the grand daughter of x is y)
- Training "data" (assertions):
  - GrandDaughter(Victor, Sharon)
  - Father(Sharon, Bob)
  - Father(Tom, Bob)
  - Female(Sharon)
  - Father(Bob, Victor)
- Closed world assumption:
  all other bindings of predicates above are implicitly false, e.g.
  - ¬Father(Bob, Tom)
  - ¬Female(Victor)
  - ¬GrandDaugher(Bob, Sharon), ...
- Available predicates:
  - Father(x, y)
  - Female(x)

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:
  - Father(x,y), Father(y,x)

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:
  - Father(x,y), Father(y,x)
  - Female(x), Female(y)

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:
  - Father(x,y), Father(y,x)
  - Female(x), Female(y)
  - Father(x,z), Father(z,x), Father(y,z), Father(z,y)

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:
  - Father(x,y), Father(y,x)
  - Female(x), Female(y)
  - Father(x,z), Father(z,x), Father(y,z), Father(z,y)
  - Equals(x,y)

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:
  - Father(x,y), Father(y,x)
  - Female(x), Female(y)
  - Father(x,z), Father(z,x), Father(y,z), Father(z,y)
  - Equals(x,y)
  - and all negations

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:
  - Father(x,y), Father(y,x)
  - Female(x), Female(y)
  - Father(x,z), Father(z,x), Father(y,z), Father(z,y)
  - Equals(x,y)
  - and all negations
- first selected literal:

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:
  - Father(x,y), Father(y,x)
  - Female(x), Female(y)
  - Father(x,z), Father(z,x), Father(y,z), Father(z,y)
  - Equals(x,y)
  - and all negations
- first selected literal:
  - Father(y,z)
    resulting in $4 * 4 * 4$ bindings with another free variable z

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:
  - Father(x,y), Father(y,x)
  - Female(x), Female(y)
  - Father(x,z), Father(z,x), Father(y,z), Father(z,y)
  - Equals(x,y)
  - and all negations
- first selected literal:
  - Father(y,z)
    resulting in $4 * 4 * 4$ bindings with another free variable z
- second literal: Father(z,x)

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:
  - Father(x,y), Father(y,x)
  - Female(x), Female(y)
  - Father(x,z), Father(z,x), Father(y,z), Father(z,y)
  - Equals(x,y)
  - and all negations
- first selected literal:
  - Father(y,z)
    resulting in $4 * 4 * 4$ bindings with another free variable z
- second literal: Father(z,x)
- third literal: Female(y)

FOIL: Example

- Starting Point: Most general rule: GrandDaughter(x,y) :- .
  With $4 * 4$ possible bindings for x and y.
  (1 positive binding, 15 negative bindings)
- Possible extensions:
  - Father(x,y), Father(y,x)
  - Female(x), Female(y)
  - Father(x,z), Father(z,x), Father(y,z), Father(z,y)
  - Equals(x,y)
  - and all negations
- first selected literal:
  - Father(y,z)
    resulting in $4 * 4 * 4$ bindings with another free variable z
- second literal: Father(z,x)
- third literal: Female(y)
- Final result:

  GrandDaughter(x,y) :- Father(y,z), Father(z,x), Female(y)

FOIL: Summary

- FOIL combines

FOIL: Summary

- FOIL combines
  - outer specific-to-general search (additional rules add positive coverage)

FOIL: Summary

- FOIL combines
  - outer specific-to-general search (additional rules add positive coverage)
  - inner general-to-specific search (rules are specialized by adding literals)

FOIL: Summary

- FOIL combines
  - outer specific-to-general search (additional rules add positive coverage)
  - inner general-to-specific search (rules are specialized by adding literals)
- FOIL can also learn recursive concepts (ignored in example)

FOIL: Summary

- FOIL combines
  - outer specific-to-general search (additional rules add positive coverage)
  - inner general-to-specific search (rules are specialized by adding literals)
- FOIL can also learn recursive concepts (ignored in example)
  - somewhat tricky since infinite recursion needs to be avoided

FOIL: Summary

- FOIL combines
  - outer specific-to-general search (additional rules add positive coverage)
  - inner general-to-specific search (rules are specialized by adding literals)
- FOIL can also learn recursive concepts (ignored in example)
  - somewhat tricky since infinite recursion needs to be avoided
- Extensions of FOIL also handle noisy data

FOIL: Summary

- FOIL combines
  - outer specific-to-general search (additional rules add positive coverage)
  - inner general-to-specific search (rules are specialized by adding literals)
- FOIL can also learn recursive concepts (ignored in example)
  - somewhat tricky since infinite recursion needs to be avoided
- Extensions of FOIL also handle noisy data
  - Stopping criteria considers description-length principle to avoid adding lengthy rules to explain few examples.

Learning First Order Rules: Summary

- Inductive Logic Programming allows to learn concepts from multi relational databases (no prior integration necessary).

Learning First Order Rules: Summary

- Inductive Logic Programming allows to learn concepts from multi relational databases (no prior integration necessary).
- most existing algorithms scale poorly (FOIL is a good example)

Learning First Order Rules: Summary

- Inductive Logic Programming allows to learn concepts from multi relational databases (no prior integration necessary).
- most existing algorithms scale poorly (FOIL is a good example)
- Applicable for special types of data (e.g. structured or mainly nominal values)