Learning Rules

- If-Then Rules are a standard knowledge representation that has proven useful in building expert systems

if (Outlook = overcast) then Play_Tennis = YES
if (Outlook = sunny) $\wedge$ (Humidity = high) then Play_Tennis = No

- Relatively easy for people to understand
- Useful in providing insight and understanding of the regularities in the data

Learning Rules

- If-Then Rules are a standard knowledge representation that has proven useful in building expert systems

if (Outlook = overcast) then Play_Tennis = YES
if (Outlook = sunny) $\wedge$ (Humidity = high) then Play_Tennis = No

- Relatively easy for people to understand
- Useful in providing insight and understanding of the regularities in the data
- There are a number of methods for inducing sets of rules from data

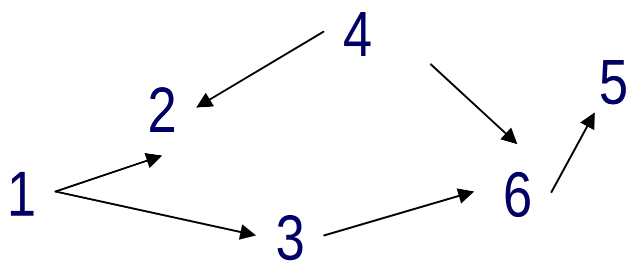
- Rule learning methods can be extended to handle relational representations (first-order-representations; inductive logic programming)

if Parent(x,y) then Ancestor(x,y)
if Parent(x,z) $\wedge$ Ancestor(z,y) then Ancestor(x,y)

Grandfather(x,y) = father(x,z) & father (z,y)

Example: Relational Learning
Inductive Logic Programming

- Finding a path in a directed acyclic graph



- What is the definition of a path?
- Definition in terms of what?
- If you want to learn this definition, what will the input be?
    - How will it be applied later?

- Today:
    - Some Background
    - The difficulties in Learning Rules
        - Learning Sets of Rules
    - Rule Learning Algorithm(s)
    - Generalization to relational Learning

Knowledge Representation

- Set of Rules: $X_1 \wedge X_2 \wedge \ldots \wedge X_m \rightarrow C_1$
  or..

  $Y_1 \wedge Y_2 \wedge \ldots \wedge Y_k \rightarrow C_2$

- Disjunctive Rules:

  DNF: Disjunction of all rules with YES as a consequent

- Ordered set of Rules:

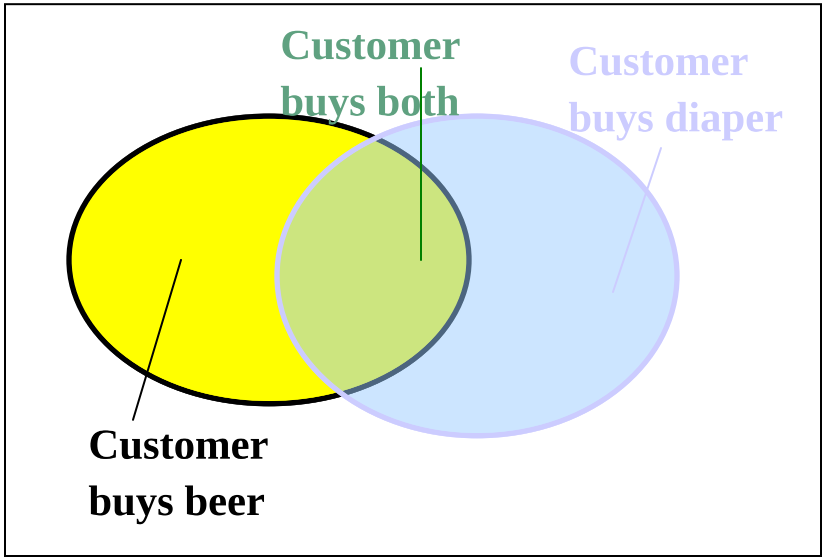
  Decision Lists: If (Condition-1) then C
  Else if (Condition-2) then D
  ……
  Else 0

Association Rules

- In the context of Data Mining the search is for rules that represent regularities in the data
- Frequent pattern: pattern that occurs frequently in a database
- Motivation: finding regularities in data
  - What products are often purchased together? Beer & diapers?!
  - What are the subsequent purchases after buying a PC?
- The goal is not to learn a classifier
  - Consequently, very simple conceptually (but tricky to scale up)

Basic Concepts: Frequent Patterns and Association Rules

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B, E, F



- Itemset $X=\{x_1, \ldots, x_k\}$
- Find all the rules $X \rightarrow Y$ with min confidence and support
  - support, *s*, fraction of examples that contain both X and Y
  - confidence, *c*, fraction of examples that contain X that also contain Y.

Let min_support = 50%,
min_conf = 50%:

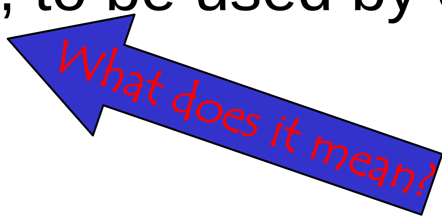
$A \rightarrow C$ (s,c) = (50%, 66.7%)

$C \rightarrow A$ (s,c) = (50%, 100%)

Learning Rules

- We will view Rule Learning in the context of Classification. The goal is to represent a function (Boolean function; multi-value function) as a collection of rules.

- As the example of Data Mining shows, rules can be useful for other things. For example, it is possible to view them as features, to be used by other learning algorithms.



Learning Rules

- Translate decision trees into rules (C4.5)
- Sequential (set) covering algorithms

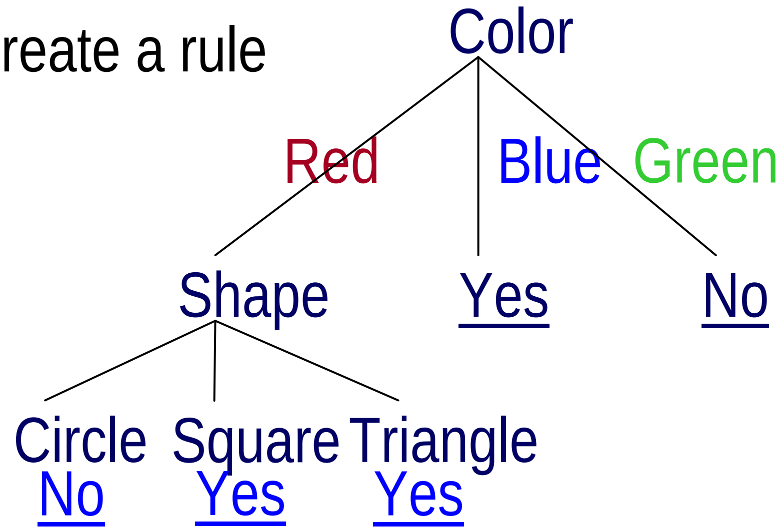
  - General to Specific (top down) (CN2, FOIL)
  - Specific to General (bottom up) (GOLEM)
  - Hybrid search (AQ, Progol)

But other algorithms may be viewed as learning (generalized) rules
(E.g., linear separators)

All the discussion today is algorithmic – given a collection of points, find a set of rules that is consistent with it. The hope is that this set of rules will also be okay in the future…

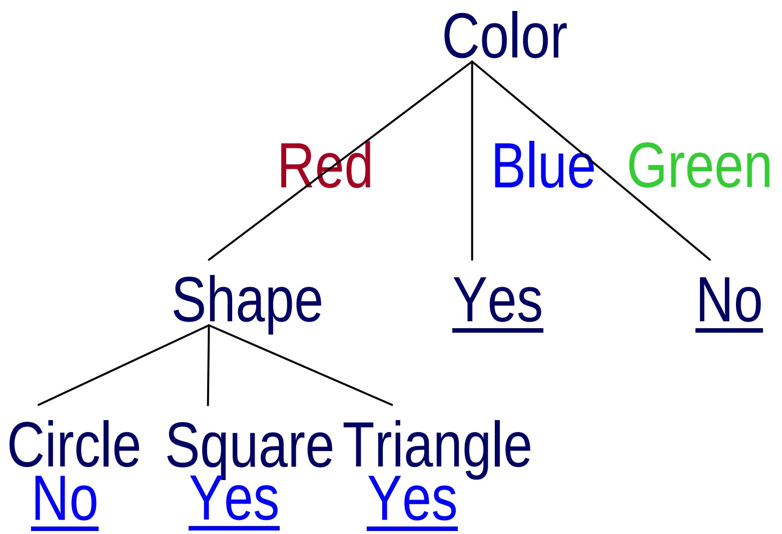
1. Decision Trees to Rules

For each path in the decision tree create a rule



Decision Trees to Rules

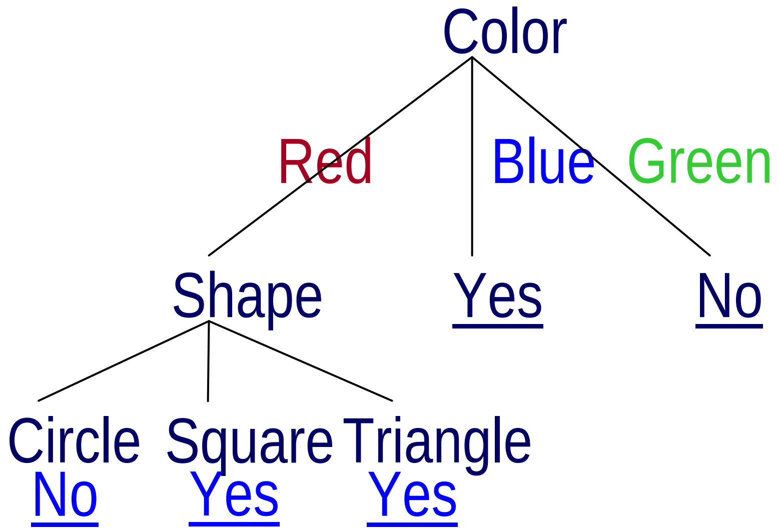
$Red \wedge Circle \rightarrow No$
$Red \wedge Square \rightarrow Yes$
$Red \wedge Triangle \rightarrow Yes$
$Blue \rightarrow Yes$
$Green \rightarrow No$



Decision Trees to Rules

In case of a Boolean Function:

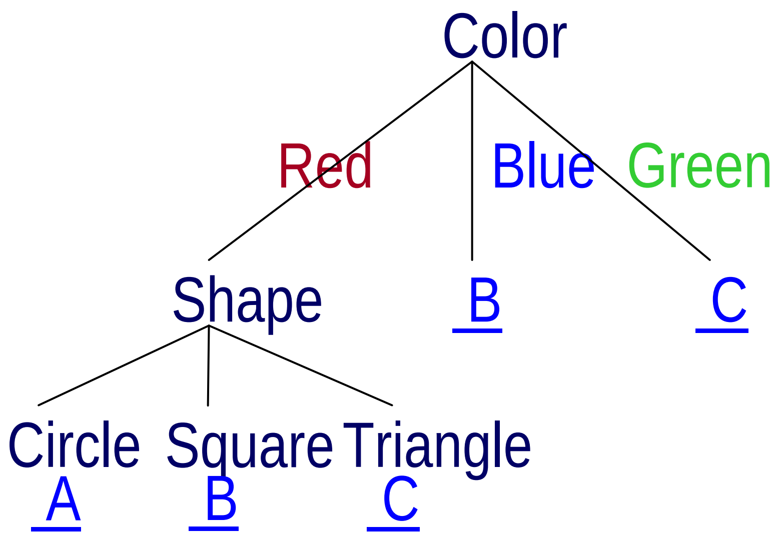
Red* $\wedge$ *Square* $\rightarrow$ *Yes
Red* $\wedge$ *Triangle* $\rightarrow$ *Yes
Blue* $\rightarrow$ *Yes



Decision Trees to Rules

In the general case:

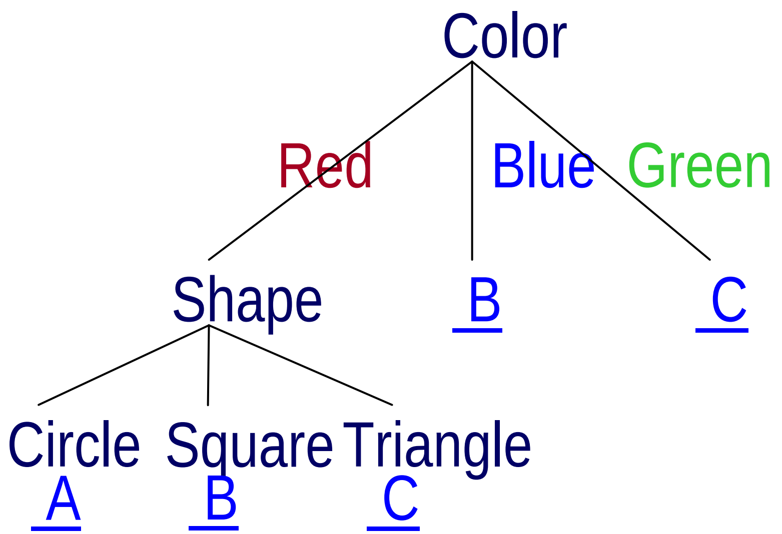
$$Red \wedge Circle \rightarrow A$$
$$Red \wedge Square \rightarrow B$$
$$Red \wedge Triangle \rightarrow C$$
$$Blue \rightarrow B$$
$$Green \rightarrow C$$



Decision Trees to Rules

In the general case:

$Red \wedge Circle \rightarrow A$
$Red \wedge Square \rightarrow B$
$Red \wedge Triangle \rightarrow C$
$Blue \rightarrow B$
$Green \rightarrow C$

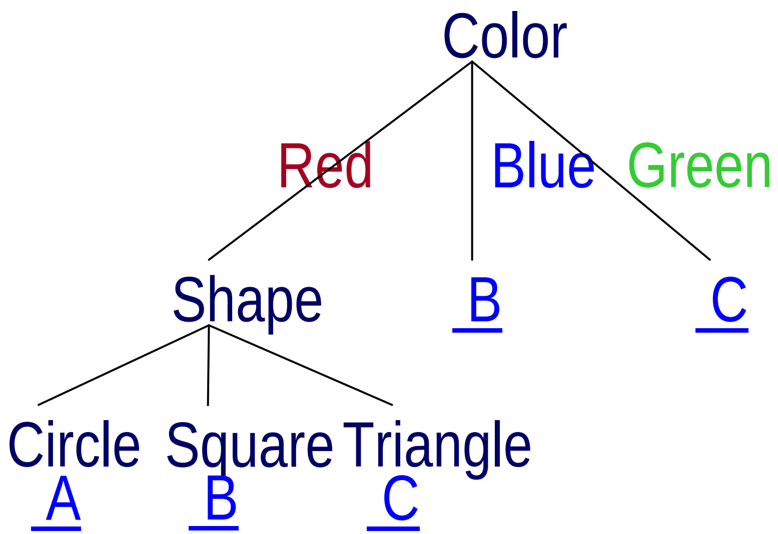


• Resulting rules may contain unnecessary antecedents that are not needed to eliminate negative examples or that result in overfitting the data (same as in Decision Trees)
• Post-prune the rules using MDL, cross-validations or related methods
• After Pruning, rules may conflict (fire together and assign different categories to a single novel test instances). (unlike Decision Trees)

Decision Trees to Rules

In the general case:

$$\mathit{Red} \wedge \mathit{Circle} \rightarrow A$$
$$\mathit{Red} \wedge \mathit{Square} \rightarrow B$$
$$\mathit{Red} \wedge \mathit{Triangle} \rightarrow C$$
$$\mathit{Blue} \rightarrow B$$
$$\mathit{Green} \rightarrow C$$

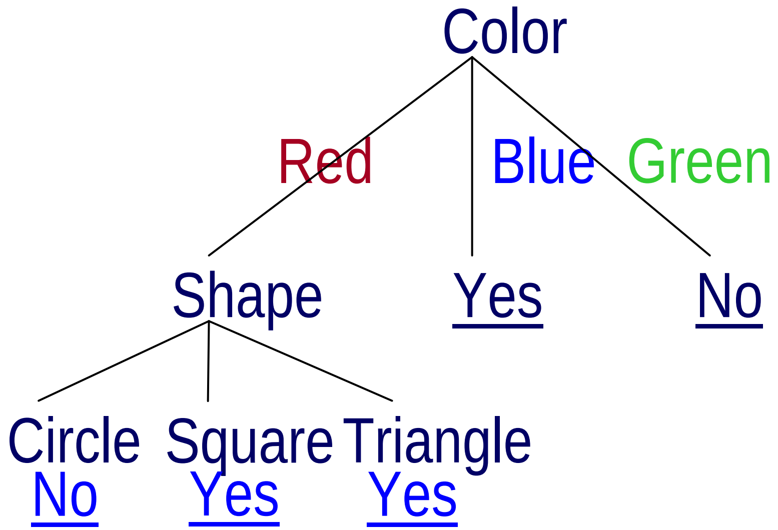


- Resulting rules may contain unnecessary antecedents that are not needed to eliminate negative examples or that result in overfitting the data.
- Post-prune the rules using and MDL, cross-validations or related methods
- After Pruning, rules may conflict (fire together and assign different categories to a single novel test instances). (unlike Decision Trees)

$$\mathit{Red} \wedge \mathit{Circle} \rightarrow A \qquad \mathit{Red} \wedge \mathit{Big} \rightarrow B$$

Test Case: (big, red, circle)

Decision Trees to Rules



Red $\wedge$ ***Square*** $\rightarrow$ ***Yes***
Red $\wedge$ ***Triangle*** $\rightarrow$ ***Yes***
Blue $\rightarrow$ ***Yes***

Solution:

- Sort rules by observed accuracy on the training data; treat the rules as an ordered set.

E.g: Decision list: If, Then, else

2. Why isn't it trivial?

The Current Best Learning Algorithm

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

The Current Best Learning Algorithm

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes

H=rain,mild,high,weak$\rightarrow$yes

H=rain, * , * ,weak$\rightarrow$yes

H=rain, * , * ,weak$\rightarrow$yes; (overcast,cool,normal,strong) $\rightarrow$ Yes

The Current Best Learning Algorithm

- H: Any hypothesis consistent with the first example in Examples

- For each remaining example e in Examples
  - If e is false positive for H (it is negative, H says it's positive)
    - H : a specialization of H that is consistent with Examples
  - Else if e is false negative for H (it is positive, H says it's negative)
    - H : a generalization of H that is consistent with Examples
  - If no consistent specialization/generalization can be found
    - Fail;
- return H

- The Algorithm needs to choose generalizations and specializations (there may be several). If it gets into trouble it has to backtrack to an earlier decision or otherwise it fails.

Difficulties

The Current Best Learning Algorithm

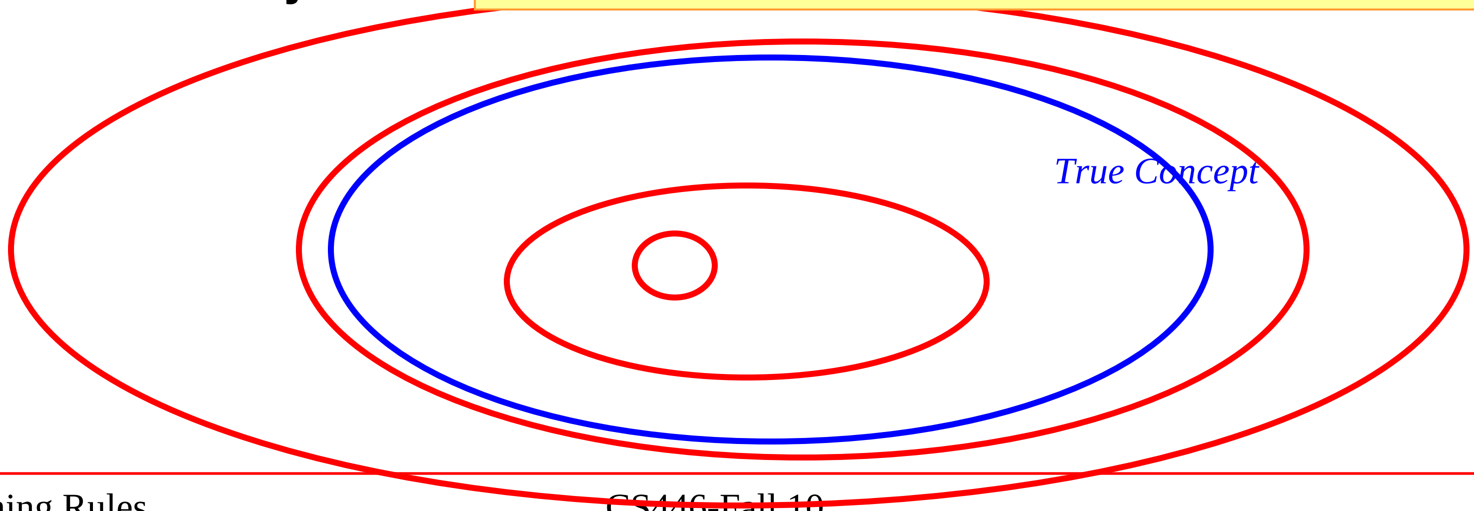
Learn the rule structure and the set of rules simultaneously, greedily.

- Generalization:
  - Remove a conjunct (sunny and normal to sunny)
  - Add a disjunct (sunny to sunny or cool)

When to add and when to remove?
Credit Assignment problem

- Specialization:
  - Add a conjunct
  - Remove a disjunct

Specifically: Rule construction; Set selection

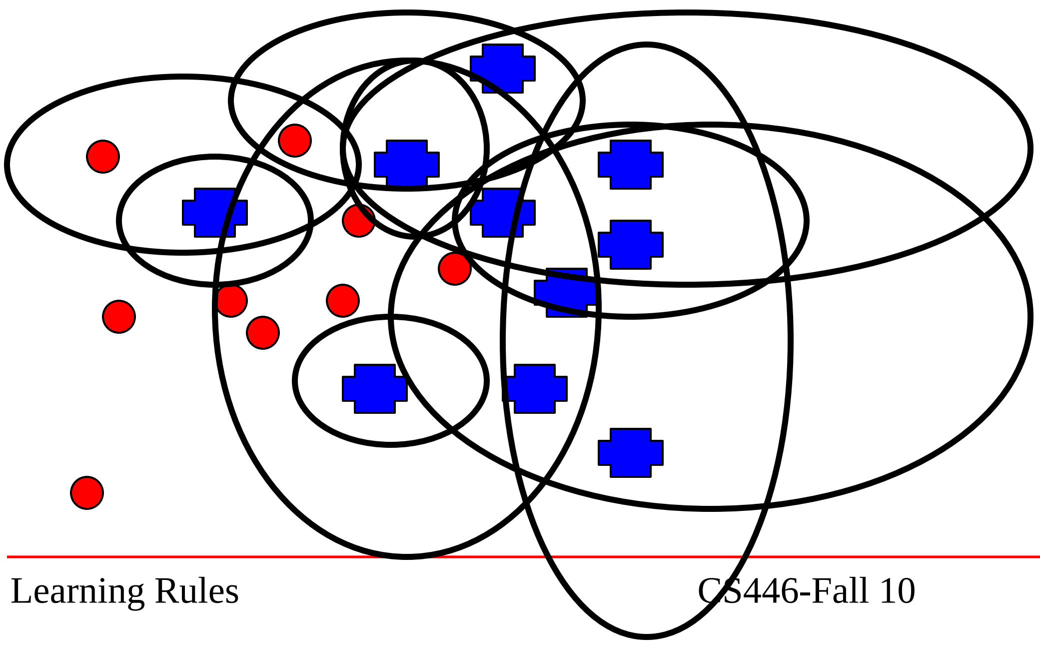


3. Learning Rules as Set Cover

- Assume you are given a set of rules, and only needs to find a list that classifies correctly all the examples.

- Set Cover Problem: X - a set of elements
  F: a family of subsets of X, such that $X = \bigcup_{S \in F} S$

  - X - set of positive examples
  - F - Collection of rules that cover only positive examples

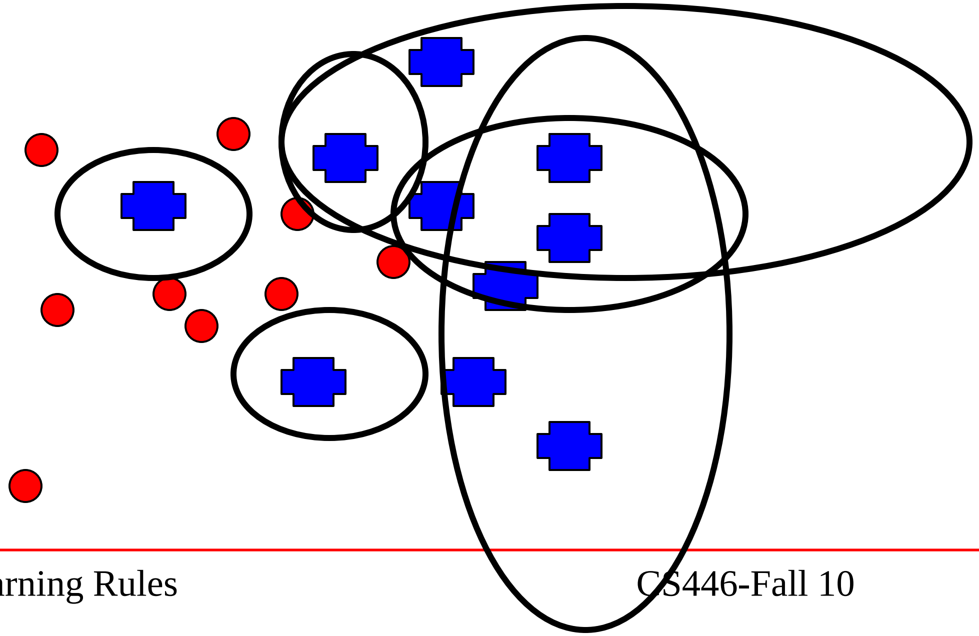


Set of Rules

Learning Rules as Set Cover

- Assume you are given a set of rules, and only needs to find a list that classifies correctly all the examples.

- Set Cover Problem: X - a set of elements
  F: a family of subsets of X, such that $X = \bigcup_{S \in F} S$
  - X - set of positive examples
  - F - Collection of rules that cover only positive examples

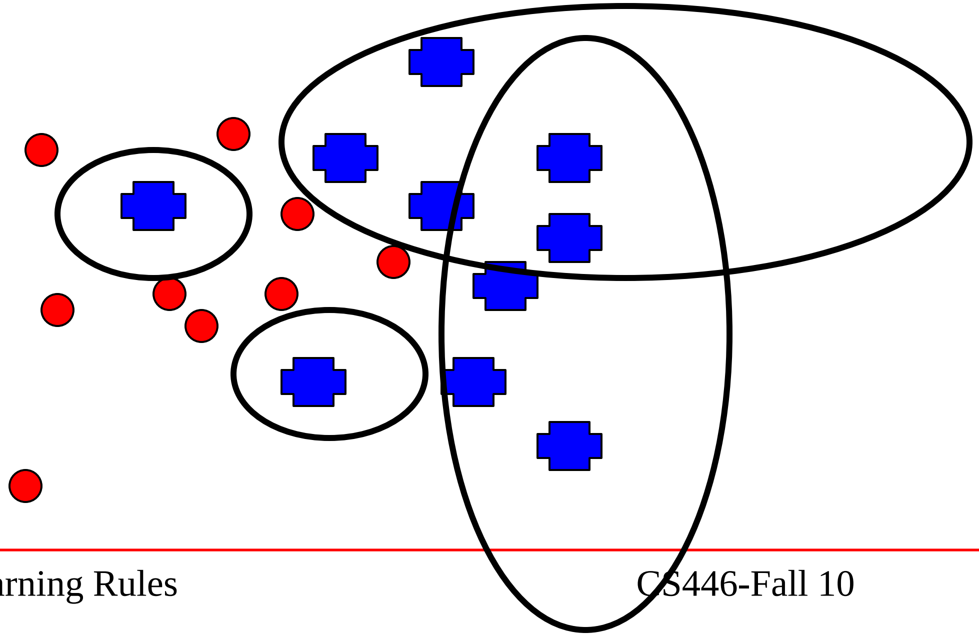


Learning Rules as Set Cover

- Assume you are given a set of rules, and only needs to find a list that classifies correctly all the examples.

- Set Cover Problem: X - a set of elements
  F: a family of subsets of X, such that $X = \bigcup_{S \in F} S$

  - X - set of positive examples
  - F - Collection of rules that cover only positive examples



- The problem of finding a minimal set cover is NP-Complete
- Good greedy approximation algorithm
- Can we find F?

Learning Rules with Sequential Covering

- A set of rules is learned one at a time
- Each time: use best rule:

  Rule that covers a large number of positives examples without covering any negatives; then, go on with the remaining positive examples.

- Let *P* be the set of positive examples.
- Until *P* is empty do:
  - Choose a rule *R* that covers a large number of positives w/o covering any negatives.
  - Add R to the list of the learned rules
  - Remove positives covered by *R* and from *P*
- What is the interpretation of this set of rules (I.e., how to use it) ?
- Minimum set cover is NP-Hard. The greedy algorithm is a good approximation.

- Remaining problem: How to learn a single rule ?

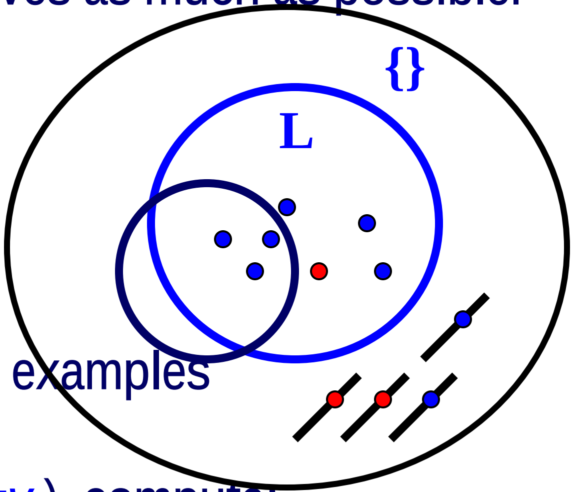
Single Rule

4. Learning A Single Rule Top-Down

$$\forall X_i \in A,\ X_1 \wedge X_2 \wedge \ldots \wedge X_k . \rightarrow YES$$

Different from homework?

- A Top-Down (general to specific) approach starts with an empty rule and greedily adds antecedents, one at a time, that eliminate negative examples while maintaining coverage of positives as much as possible.



- Algorithms based on *FOIL* (Quinlan, 1990)

- Let *A={}*
- Let *N* be the set of all negative examples
- Let *P* be the current set of uncovered positive examples
- Until *N* is empty do
  - For every feature-value pair (literal) L = (*f*=*v*) compute:

    Gain(f=v, P, N)
  - Pick a literal, L = (*f*=*v*) with highest *Gain*
  - Add L to *A*
  - Remove from *P* examples that do not satisfy L (will not be covered)
  - Remove from *N* examples that do not satisfy L (already rejected)

Return the conjunction of all literals in *A*

The Gain Metric

- Want to achieve two goals:
  - Decrease coverage of negative examples
    Measure increase in percentage of positive examples covered when making the proposed specialization to the current rule
  - Maintain coverage of as many positives as possible
    Count number of positive examples covered

Gain(L, P, N) :

- Let *N** *be a subset of N* that satisfy the literal *L*
- Let *P** *be a subset of P* that satisfy the literal *L* (still covered)
- return:

$$|P^*| \left(\log \frac{|P^*|}{|P^*| + |N^*|} - \log \frac{|P|}{|P| + |N|} \right)$$

Example: Top Down Rule Learning

(000 -) (001 -) (010 -) (011 -) (100 +) (101 +) (110 -) (111 -)

$$|P^*|\left(\frac{\log |P^*|}{|P^*| + |N^*|} - \frac{\log |P|}{|P| + |N|}\right)$$

N be a subset of N* that satisfy the literal *L*
P be a subset of P* that satisfy the literal *L* (still covered)

Example: Top Down Rule Learning

(000 -) (001 -) (010 -) (011 -) (100 +) (101 +) (110 -) (111 -)

P=2, N=6

L=x1: P =2, N* = 2 Gain = 2 log2 /4 - log2 /8 = 3/8*

$$|P^*|\left(\frac{\log|P^*|}{|P^*|+|N^*|} - \frac{\log|P|}{|P|+|N|}\right)$$

N be a subset of N* that satisfy the literal *L*

P be a subset of P* that satisfy the literal *L* (still covered)

Example: Top Down Rule Learning

(000 -) (001 -) (010 -) (011 -) (100 +) (101 +) (110 -) (111 -)

P=2, N=6

L=x1: P =2, N* = 2 Gain = 2 log2 /4 - log2 /8 = 3/8*

L=x2: P =0, N* = 4 Gain = 0 - log2 /8 = -1/8*

L=x3: P =1, N* = 3 Gain = 1 log1 /4 - log2 /8 = -1/8*

$$|P^*| \frac{\log |P^*|}{|P^*| + |N^*|} - \frac{\log |P|}{|P| + |N|}$$

N be a subset of N* that satisfy the literal *L*

P be a subset of P* that satisfy the literal *L* (still covered)

Example: Top Down Rule Learning

(000 -) (001 -) (010 -) (011 -) (100 +) (101 +) (110 -) (111 -)

P=2, N=6

L=x1: P =2, N* = 2 Gain = 2 log2 /4 - log2 /8 = 3/8*

L=x2: P =0, N* = 4 Gain = 0 - log2 /8 = -1/8*

L=x3: P =1, N* = 3 Gain = 1 log1 /4 - log2 /8 = -1/8*

First literal chosen is x1

$$|P^*| \left(\log \frac{|P^*|}{|P^*| + |N^*|} - \log \frac{|P|}{|P| + |N|} \right)$$

N be a subset of N* that satisfy the literal *L* (already rejected)

P be a subset of P* that satisfy the literal *L* (still covered)

Example: Top Down Rule Learning

(000 -) (001 -) (010 -) (011 -) (100 +) (101 +) (110 -) (111 -)

P=2, N=6

L=x1: P =2, N* = 2 Gain = 2 log2 /4 - log2 /8 = 3/8*

L=x2: P =0, N* = 4 Gain = 0 - log2 /8 = -1/8*

L=x3: P =1, N* = 3 Gain = 1 log1 /4 - log2 /8 = -1/8*

$$|P^*| \frac{\log |P^*|}{|P^*| + |N^*|} - \frac{\log |P|}{|P| + |N|}$$

First literal chosen is x1

(100 +) (101 +) (110 -) (111 -)

P=2, N=2

N be a subset of N* that satisfy the literal *L* (already rejected)

P be a subset of P* that satisfy the literal *L* (still covered)

Example: Top Down Rule Learning

(000 -) (001 -) (010 -) (011 -) (100 +) (101 +) (110 -) (111 -)

P=2, N=6

L=x1: P =2, N* = 2 Gain = 2 log2 /4 - log2 /8 = 3/8*

L=x2: P =0, N* = 4 Gain = 0 - log2 /8 = -1/8*

L=x3: P =1, N* = 3 Gain = 1 log1 /4 - log2 /8 = -1/8*

$$|P^*| \frac{\log |P^*|}{|P^*| + |N^*|} - \frac{\log |P|}{|P| + |N|}$$

First literal chosen is x1

(100 +) (101 +) (110 -) (111 -)

P=2, N=2

L=x2: P =0, N* = 2 Gain = 0 - log2 /4 = -1/4*

L=x3: P =1, N* = 1 Gain = 1 log1 /2 - log2 /4 = -1/4*

L=not(x2) P =2, N* = 0 Gain = 2 log2 /2 - log2 / 4=1-1/4*

N be a subset of N* that satisfy the literal *L* (already rejected)

P be a subset of P* that satisfy the literal *L* (still covered)

Example: Top Down Rule Learning

(000 -) (001 -) (010 -) (011 -) (100 +) (101 +) (110 -) (111 -)

P=2, N=6

L=x1: P =2, N* = 2 Gain = 2 log2 /4 - log2 /8 = 3/8*

L=x2: P =0, N* = 4 Gain = 0 - log2 /8 = -1/8*

L=x3: P =1, N* = 3 Gain = 1 log1 /4 - log2 /8 = -1/8*

$$|P^*|\left(\log\frac{|P^*|}{|P^*|+|N^*|} - \log\frac{|P|}{|P|+|N|}\right)$$

First literal chosen is x1

(100 +) (101 +) (110 -) (111 -)

P=2, N=2

L=x2: P =0, N* = 2 Gain = 0 - log2 /4 = -1/4*

L=x3: P =1, N* = 1 Gain = 1 log1 /2 - log2 /4 = -1/4*

L=not(x2) P =2, N* = 0 Gain = 2 log2 /2 - log2 / 4=1-1/4*

we have learned: x1 and not(x2)

N be a subset of N* that satisfy the literal *L* (already rejected)

P be a subset of P* that satisfy the literal *L* (still covered)

Example: Top Down Rule Learning

(000 -) (001 -) (010 -) (011 -) (100 +) (101 +) (110 -) (111 -)

P=2, N=6

L=x1: P =2, N* = 2 Gain = 2 log2 /4 - log2 /8 = 3/8*

L=x2: P =0, N* = 4 Gain = 0 - log2 /8 = -1/8*

L=x3: P =1, N* = 3 Gain = 1 log1 /4 - log2 /8 = -1/8*

$$|P^*| \left(\log \frac{|P^*|}{|P^*| + |N^*|} - \log \frac{|P|}{|P| + |N|} \right)$$

First literal chosen is x1

(100 +) (101 +) (110 -) (111 -)

P=2, N=2

L=x2: P =0, N* = 2 Gain = 0 - log2 /4 = -1/4*

L=x3: P =1, N* = 1 Gain = 1 log1 /2 - log2 /4 = -1/4*

L=not(x2) P =2, N* = 0 Gain = 2 log2 /2 - log2 / 4=1-1/4*

we have learned: x1 and not(x2)

What if the examples were generated from a DNF?

N be a subset of N* that satisfy the literal *L* (already rejected)

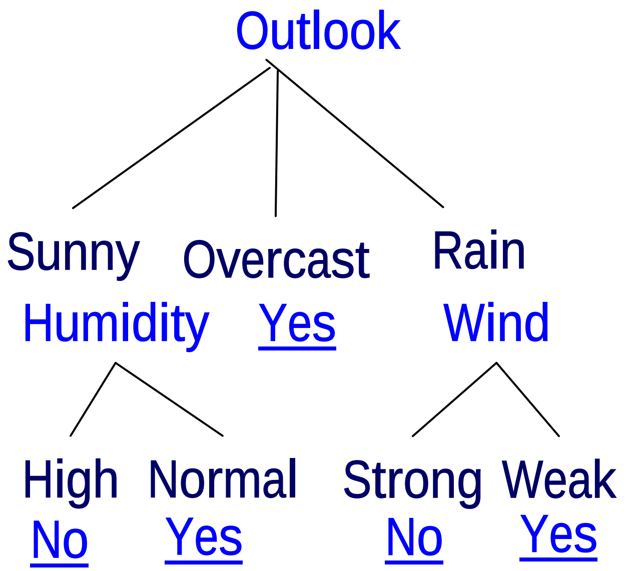
P be a subset of P* that satisfy the literal *L* (still covered)

Other General-to-Specific Methods

- As In ID3
  - Follow only the most promising branch at every step.
    - Choose best attribute to split on
      for each value, choose one of the splits and go on.
    - At some point, determine the consequent of the rule
    - Go back to search for the best attribute, but on a different set of examples

Other General-to-Specific Methods

- As In ID3
  - Follow only the most promising branch at every step.
    - Choose best attribute to split on
      for each value, choose one of the splits and go on.
    - At some point, determine the consequent of the rule
    - Go back to search for the best attribute, but on a different set of examples



Other General-to-Specific Methods

- As In ID3
  - Follow only the most promising branch at every step.
    - Choose best attribute to split on
      for each value, choose one of the splits and go on.
    - At some point, determine the consequent of the rule
    - Go back to search for the best attribute, but on a different set of examples

- This is a greedy depth-first-search, with no backtracking.
    - no guarantee that it will make optimal decision

- Beam search: maintain a list of the k best candidates at each step.
    - At each step, generate descendants for each of the k best candidates, and reduce the resulting set again to the best k

Summary: Incremental Reduced Error Pruning

```
procedure IREP(Pos,Neg)
begin
    Ruleset := ∅
    while Pos≠ ∅ do
        /* grow and prune a new rule */
        split (Pos,Neg) into (GrowPos,GrowNeg)
          and (PrunePos,PruneNeg)
        Rule := GrowRule(GrowPos,GrowNeg)
        Rule := PruneRule(Rule,PrunePos,PruneNeg)
        if the error rate of Rule on
          (PrunePos,PruneNeg) exceeds 50% then
            return Ruleset
        else
            add Rule to Ruleset
            remove examples covered by Rule
              from (Pos,Neg)
        endif
    endwhile
    return Ruleset
end
```

IREP

- Integrates Reduced Error Pruning with a Separate and Conquer (Sequential Covering) rule learning algorithm.
- A rule is a conjunction of features; a rule set is a DNF formula.
- Builds up a rule set in a greedy fashion, one rule at a time.
- After each rule is found, all exemplas covered by it (both P and N) are deleted.
- This process is repeated until there are no more positive examples, or until the only rule found has unacceptably large error rate.

Summary: Incremental Reduced Error Pruning

To Relational Learning

```
procedure IREP(Pos,Neg)
begin
    Ruleset := ∅
    while Pos≠ ∅ do
        /* grow and prune a new rule */
        split (Pos,Neg) into (GrowPos,GrowNeg)
          and (PrunePos,PruneNeg)
        Rule := GrowRule(GrowPos,GrowNeg)
        Rule := PruneRule(Rule,PrunePos,PruneNeg)
        if the error rate of Rule on
          (PrunePos,PruneNeg) exceeds 50% then
            return Ruleset
        else
            add Rule to Rulese
            remove examples c
              from (Pos,Neg)
        endif
    endwhile
    return Ruleset
end
```

Two classes

Grow Set, Prune Set: Rand(2/3, 1/3)

Repeatedly add the feature that maximizes FOIL's information gain criterion

Rule is immediately pruned after being grown. Every final sequence of conditions is considered; chooses a deletion that maximizes $(p^* + (N-N^*))/(P+N)$

Better (Ripper): $(P^*-N^*)/(P^* + N^*)$

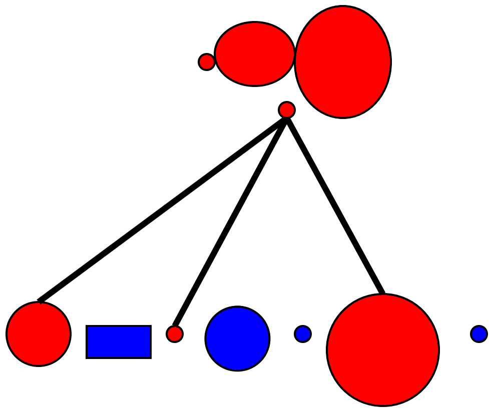
Better (Ripper): MDL based stopping criterion – stops when the last rule adds too much to the description length.

Done with this Rule. Add a new one.

Optimization (Ripper): Global REP:

1. revise (grow/prune) considering reduced error on the whole set.
2. If uncovered positives, add rules.

Learning Rules Bottom-Up (2)



- Let P be the current set of uncovered positive examples
- Let R be a random sample of s pairs (a, b) from P

- LGGs = {LGG(a,b) | all pairs from R}
- Remove from LGGs ones that cover negative examples

- Let g be the LGG with the greatest positive cover
- Remove from P the examples covered by g (already covered)
- Do while g increases its positive coverage
- Let E be a random sample of s examples from P
- Let LGGs = { LGG(g, e) | e in E} (all candidates cover more positives than g)
- Remove from LGGs ones that cover negative examples
- Let g be the LGG with the greatest positive coverage
- Remove from P the examples covered by g

- Return rule If g then YES

Example: Bottom-UP Rule Learning

(0000 -) (0010 -) (0100 -) (0110 -) (1000 +) (1010 +) (1100 -) (1110 -)
(0001 -) (0011 -) (0101 -) (0111 -) (1001 +) (1011 +) (1101 -) (1111 -)

Example: Bottom-UP Rule Learning

(0000 -) (0010 -) (0100 -) (0110 -) (1000 +) (1010 +) (1100 -) (1110 -)
(0001 -) (0011 -) (0101 -) (0111 -) (1001 +) (1011 +) (1101 -) (1111 -)

g = LGG(1010,1011)= x1 and not(x2) and x3

Example: Bottom-UP Rule Learning

(0000 -) (0010 -) (0100 -) (0110 -) (1000 +) (1010 +) (1100 -) (1110 -)
(0001 -) (0011 -) (0101 -) (0111 -) (1001 +) (1011 +) (1101 -) (1111 -)

g = LGG(1010,1011)= x1 and not(x2) and x3

Does not cover negatives
Cover some of the positives

Example: Bottom-UP Rule Learning

(0000 -) (0010 -) (0100 -) (0110 -) (1000 +) (1010 +) (1100 -) (1110 -)
(0001 -) (0011 -) (0101 -) (0111 -) (1001 +) (1011 +) (1101 -) (1111 -)

g = LGG(1010,1011)= x1 and not(x2) and x3

Does not cover negatives
Cover some of the positives

LGG(g, 1001) = x1 and not(x2)

Example: Bottom-UP Rule Learning

(0000 -) (0010 -) (0100 -) (0110 -) (1000 +) (1010 +) (1100 -) (1110 -)
(0001 -) (0011 -) (0101 -) (0111 -) (1001 +) (1011 +) (1101 -) (1111 -)

g = LGG(1010,1011)= x1 and not(x2) and x3

Does not cover negatives
Cover some of the positives

LGG(g, 1001) = x1 and not(x2)

What if the examples were generated from a DNF

Other Options for Guiding the search of Rules

- Sequential covering is one alternative; can be used when a set of rules is given or interleaved with a rule search algorithm.

- Relative Frequency: $\frac{n_c}{n}$

  n - the number of examples the rule matches

  n_c - the number of these it classifies correctly

- Entropy:

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

  S - the set of examples that match the rule precondition

  c - the number of values taken by the target function

  p_i - the proportion of examples from S for which the function takes on the ith value

- Mistake Driven:

  reduction in # of mistakes made by the current hypothesis

Does it Work ?

Rule Learning vs. Knowledge Engineering

An influential experiment with AQ (Michalsky & Chilausky, 1980) demonstrated that rule induction from examples can be more efficient and effective than knowledge engineering (acquiring rules by interviewing experts)

Data:

Examples of 15 diseases described using 35 feature; 630 total examples

290 most diverse examples were used for training

Performance:

A few minutes training vs 45 hours consultation with expe

(97.6% first rule correct, 100% one rule correct (vs. 72%

What happens in Larger Domains ?

Many variables? Many rules? Longer rules?

A lot of successful works on "generalized rules" learning (Linear functions)

Ripper is currently one of the best Rule Learning Algorithms, and in some contexts, competitive with linear threshold functions.

Relational Learning

- Target concept: Daughter(x,y) (x is a daughter of y)
- examples: (names are unique identifiers)

(name, mother, father, m/f; name, mother, father, m/f ; label)

E.g,: (Sharon, Louise, Bob,f; Bob, Nora, Victor,m; True)

- Propositional Rule Learning may result in very specific rules:

If (father(1) = Bob) and (Name(2)= Bob) and (m/f(1)=f)

then True

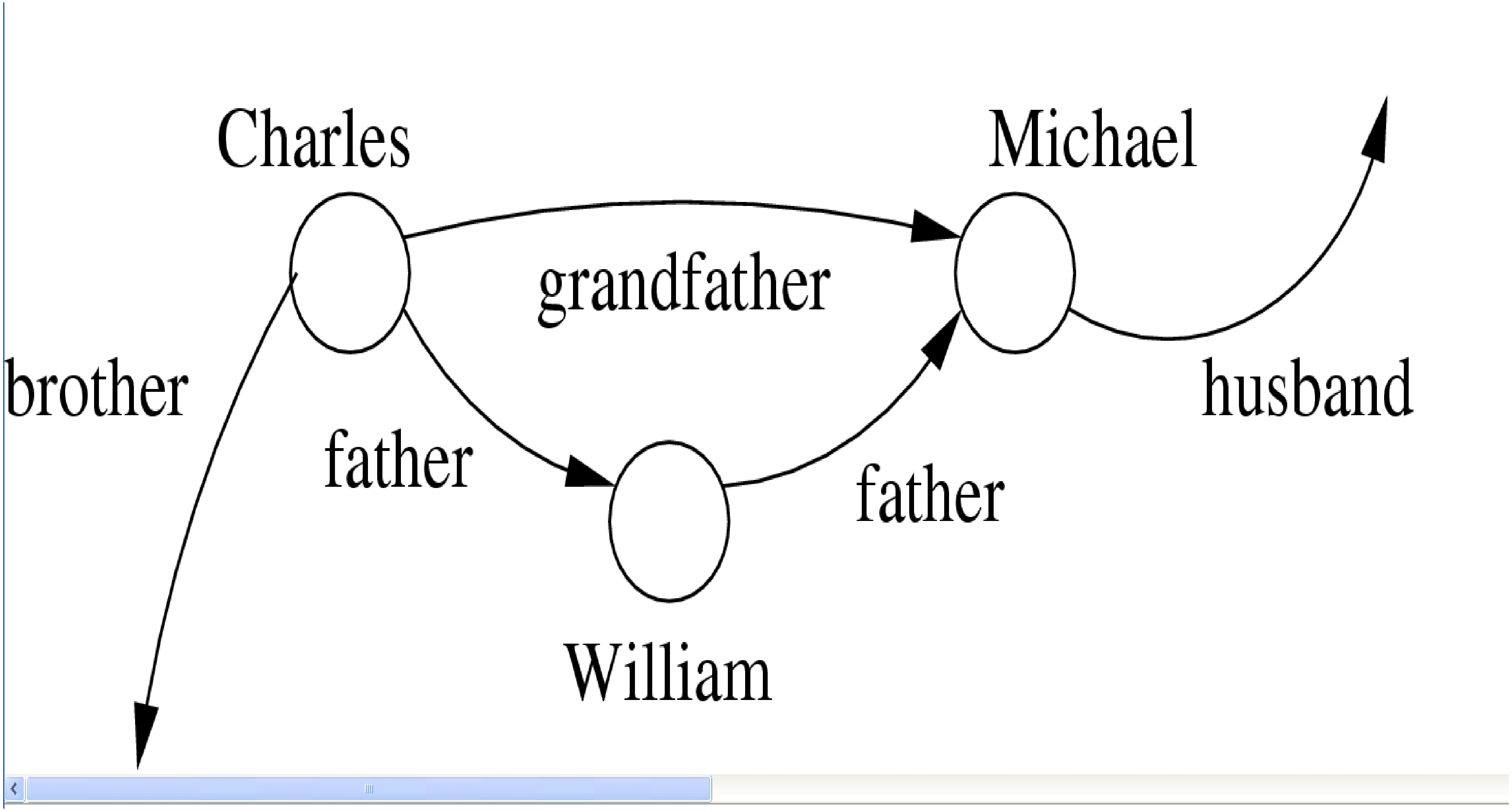
- Too specific to be useful
- We want something like:

If father(y,x) and female(x)

then daughter (x,y)

where x, y are variables that can be bound to any person

Grandfather(x,y) = father(x,z) & father (z,y)



Relational Learning - cont.

- More generally:

  If father(y,z) and mother(z,x) and female(x)

  then granddaughter (x,y)

where x, y,z are variables; z appears in the precondition, but not in the postcondition (z is existentially quantified)

- We may even want to use the same predicates in the precondition and in the postcondition

  if Parent(x,y) then Ancestor(x,y)

  if Parent(x,z) and Ancestor(z,y) then Ancestor(x,y)

yielding a recursive definition

More powerful representation language; how about learning?

Work on Relational Learning

- Traditionally, this work was done in a sub-field of Machine Learning called Inductive Logic Programming (ILP) and focused on trying to learn Logical Definitions (Prolog Programs)

- More recently, work in this area is called Statistical Relational Learning, although this term is loaded and is used for more than just dealing with "relational domains".
- Key idea: often you want to, or have to abstract over feature values.
  - In some problems this is necessary; in some impossible

- We will:
  - Show a few examples to illustrate the need [Some NLP examples at the end]
  - Exemplify one ILP algorithm
  - Comment on when/why these learning techniques are needed.
- Possible area for a class project (next time)

Relational Learning - cont.

- More generally:

    If father(y,z) and mother(z,x) and female(x)
    then granddaughter (x,y)

where x, y,z are variables; z appears in the precondition, but not in the postcondition
(z is existentially quantified)

- We may even want to use the same predicates in the precondition and in the postcondition

    if Parent(x,y) then Ancestor(x,y)
    if Parent(x,z) and Ancestor(z,y) then Ancestor(x,y)

yielding a recursive definition

Relational Learning and ILP

- Examples may be represented using relations
- Concepts may be relational

- Basic building blocks: literals - predicates applied to terms
  father(Bob,Sharon), not-married(x), greater_than(age(Sharon),20)

- Inductive Logic Programming:
  Induce a disjunction of (Horn) clauses (If-then rules) definitions for some target predicate P

$$P \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_k$$

- Given background predicates

Relational Learning and ILP

- Inductive Logic Programming:
  Induce a Horn-clause definition for some target predicate P
  given definition of background predicates

Goal: Find syntactically simple definition D for P such that
given background definitions B
For every positive example *p* : D together with B imply *p*
For every negative example *n* : D together with B do not imply *n*

Background Definitions can be provided
- Extensionally: List of ground literals
- Intensionally: Horn definition of the predicate

- Usually there is no distinction between examples and background knowledge, and everything is given extensionally. (List of facts)

FOIL

- Top down sequential covering algorithm,
  adapted for Prolog clauses without functions

- Learn-one-Rule: General to specific search, extended to accommodate first order rules

- Rules are extensions of Horn; allow negative literals in the antecedent

- Background (examples) provided extensionally

  This is how we learn what predicates are available

  father(Bob,Sharon), mother(Louisa, Sharon), female(Sharon)

Positive examples are those literals in which the target predicate is True

Negative examples are provided using the closed world assumption

FOIL - Algorithm

Let *P* be the set of positive examples.

- Until *P* is empty do:
  - Learn a new rule *R* that covers a large number of positives w/o covering any negatives.
    - Let *A={}* be a set of preconditions (predicts Target with no precondition)
    - Let *N* be the set of all negative examples
    - Until *N* is empty do
      (Add a new literal to specialize *R)*
      * Generate candidate literals for *R*
        L= Best Literal = argmax Gain(Lit, P, N)
      * Add L to *A*
      * Remove from *P* examples that do not satisfy L (will not be covered)
      * Remove from *N* examples that do not satisfy L (already rejected)
  - Add R to the list of the learned rules
  - Update the set *P* : Remove positives covered by *R* and from *P*
- Return the list of learned rules

Search in FOIL

- Background provided extensionally
  This is how we learn what predicates are available
  father(Bob,Sharon), mother(Louisa, Sharon), female(Sharon),
- Initialization:
  Most general target predicate
  granddaughter (x,y) <------
- Possible specializations of a clause:
  consider literals that fit one of the following forms:
  Q(x,y,z…), not-Q(x,y,z…), (x=y), not(x=y)
  where Q is a predicate (known from the background information)
  x,y,x,… are variables. All but one must already exist in the clause

Candidate additions to the rule precondition:
father(x,y), mother(x,y), father(x,z), female(y), equal(x,y), (and negations)

Search in FOIL (2)

- At every step FOIL considers all known literals plus additional literals that are generated with a new variable
  If we have considered:
  father(x,y), mother(x,y), father(x,z), female(y), equal(x,y), (and negations)
  we will consider now also:
  father(x,w), mother(x,w), father(w,z), father(z,w)…

At some point in the search we will generate the rule
granddaughter(x,y) <---- father(y,z) and mother(z,x) and female(x)

which covers all the positive examples and none of the negatives.
If there are remaining positive examples to be covered, then we begin at this point a search for a new rule.

Search in FOIL (2)

- At every step FOIL considers all known literals plus additional literal that are generated with a new variable
  If we have considered:
  father(x,y), mother(x,y), father(x,z), female(y), equal(x,y), (and negations)
  we will consider now also:
  father(x,w), mother(x,w), father(w,z), father(z,w)…

At some point in the search we will generate the rule
granddaughter(x,y) <---- father(y,z) and mother(z,x) and female(x)

which covers all the positive examples and none of the negatives.
If there are remaining positive examples to be covered, then we begin at this point a search for a new rule.

Works since: The relational rule holds in the data.
We search exhaustively.

Note that (Search in FOIL (2))

At some point in the search we will generate the rule
granddaughter(x,y) <---- father(y,z) and mother(z,x) and female(x)
which covers all the positive examples and none of the negatives.
If there are remaining positive examples to be covered, then we begin at this point a search for a new rule.

Works since: The relational rule holds in the data.
We search exhaustively.

- In some sense, this is very similar to propositional learning

$y \leftarrow$ A and B and C

In an example (A= ,B= ,C= ,D= ,E= ,……;y) a proposition is either T or F

father(x,y) is also either T or F in an example but, possibly, several things could make it T. (E.g., father(Bob, Sharon),….)

- Problems are introduced when evaluating existential expressions.

Search in FOIL (3)

- All possible bindings are considered when generating candidate literals

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

Closed World Assumption: Any literal involving the predicate GrandDaughter, Father, or Female and contains the constants above is FALSE unless in the list

Starting with: granddaughter(x,y) ←
we need to consider any substitution binding x,y to the constants

Some are positive: x/Sharon; y/Victor (since GrandDaughter(Sharon,Victor))
and some negative: x/Bob; y/Victor

- Here we have 15 Negative bindings and 1 positive
- New variables -- more bindings --- (|V|**|constants|)

Search in FOIL (4): Choosing Literals

•Consider a rule R and a new literal L

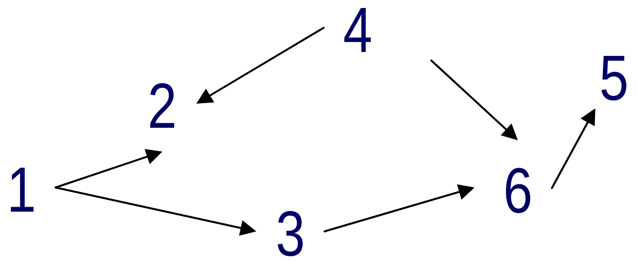
Gain(L, R) :

- Let *N* *be the number of negative bindings of R*
- Let *N * be the number of negative bindings of R with the addition of L*
- Let *P* *be the number of positive bindings of R*
- Let *P* be the number of positive bindings of R with the addition of L*
- Let *P+ be the number of positive examples of R that are still covered when adding L*

$$|P^{+}| \log \frac{|P^{*}|}{|P^{*}| + |N^{*}|} - \log \frac{|P|}{|P| + |N|}$$

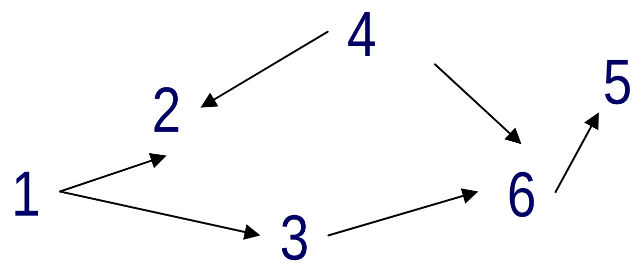
Example

- Finding a path in a directed acyclic graph



Example-2

- Finding a path in a directed acyclic graph
- path(x,y):-edge(x,y)
- path(x,y):-edge(x,z),path(z,y)



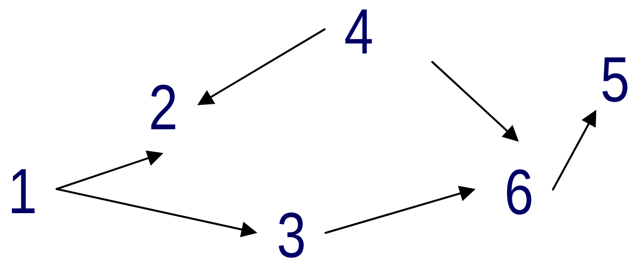
- edge(1,2), edge(1,3),edge(3,6),edge(4,2),edge(4,6),edge(6,5)

- path(1,2),path(1,3),path(1,6),path(1,5),path(3,6), path(3,5), path(4,2),path(4,6),path(4,5),path(6,5)

Negative examples can be provided directly or with the closed world assumption

Example-3

Positive Examples: (written as bindings (x,y))
(1,2), (1,3), (1,6), (1,5), (3,6),
(3,5), (4,2), (4,6), (4,5), (6,5)

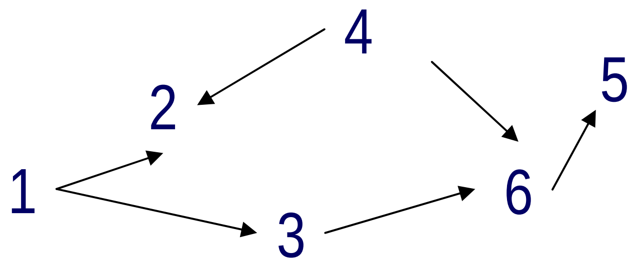


Negative examples;

Start with empty rule:
path(x,y):-
Consider adding literal edge(x,y)
(also consider edge(y,x), edge(x,z),edge(z,x),path(y,x),path(x,z),path(z,x),
x=y and negations)

Example-4

Positive Examples:
(1,2), (1,3), (1,6), (1,5), (3,6),
(3,5), (4,2), (4,6), (4,5), (6,5)



Negative examples;

The rule:

path(x,y):- edge(x,y)

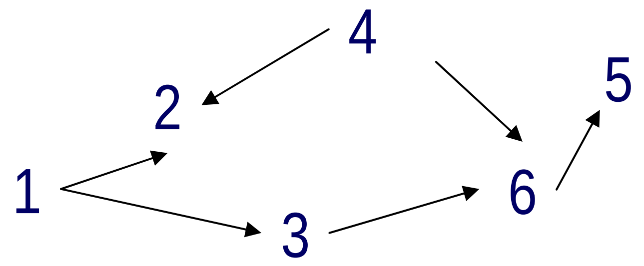
Covers 6 positive examples and no negative example

(We know that since we have a list of bindings for edge(x,y)

Example-5

Positive Examples:

(1,2), (1,3), (1,6), (1,5), (3,6),
(3,5), (4,2), (4,6), (4,5), (6,5)



$$|P^+| \log \frac{|P^*|}{|P^*| + |N^*|} - \log \frac{|P|}{|P| + |N|}$$

Negative examples;

The rule:

path(x,y):- edge(x,y)

Empty Rule: (P,N)= (10,20)
edge(x,y): (P,N) = (6,0)
edge(y,x): (P,N) = (0,6)

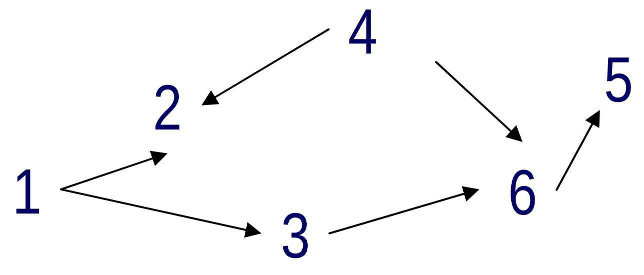
Covers 6 positive examples and no negative example.
Done with the internal process -- found a good rule.
We start with this rule and remove covered examples

Example-6

Positive Examples:

(1,6), (1,5)

(3,5), (4,5),



Negative examples;

(1,4),(2,1),(2,3),(2,4),(2,5)

(2,6),(3,1),(3,2),(3,4),(4,1)

(4,3),(5,1),(5,2),(5,3),(5,4)

(5,6),(6,1),(6,2),(6,3),(6,4)

Start with a new empty rule:

path(x,y)

Consider literal edge(x,z) (among others)

Example-7

Positive Examples:

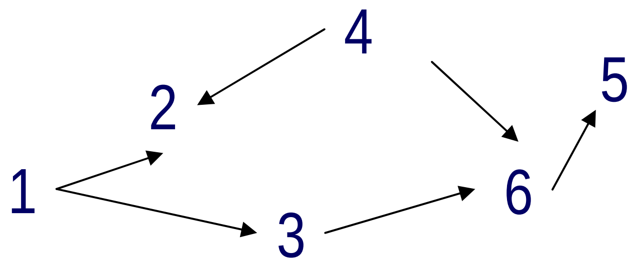
(1,6), (1,5)
(3,5), (4,5),

Negative examples;

(1,4),(2,1),(2,3),(2,4),(2,5)
(2,6),(3,1),(3,2),(3,4),(4,1)
(4,3),(5,1),(5,2),(5,3),(5,4)
(5,6),(6,1),(6,2),(6,3),(6,4)

Start with a new empty rule:

path(x,y)



$$|P^{+}| \log \frac{|P^{*}|}{|P^{*}| + |N^{*}|} - \log \frac{|P|}{|P| + |N|}$$

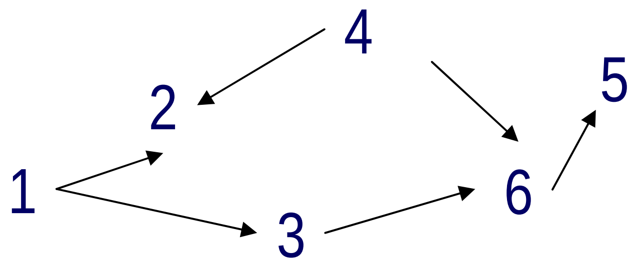
Empty Rule: (P,N)= (4,20)
edge(x,y): (P,N) = (0,0)
edge(x,z): (P,N) = ?

Consider literal edge(x,z) (among others)

Example-8

Positive Examples:

(1, 6, z), (1, 5, z),
(3, 5, z), (4, 5, z),



$$|P^+| \log \frac{|P^*|}{|P^*| + |N^*|} - \log \frac{|P|}{|P| + |N|}$$

Negative examples;

(1,4,z), (2,1,z),(2,3,z),(2,4,z),(2,5,z)
(2,6,z), (3,1,z),(3,2,z),(3,4,z),(4,1,z)
(4,3,z), (5,1,z),(5,2,z),(5,3,z),(5,4,z)
(5,6,z), (6,1,z),(6,2,z),(6,3,z),(6,4,z)

Empty Rule: (P,N)= (4,20)
edge(x,y): (P,N) = (0,0)
edge(x,z): (P,N) = ?

path(x,y):-edge(x,z)

New rule covers all the 4 remaining positives
but also 10 of the 20 negatives

Example-9

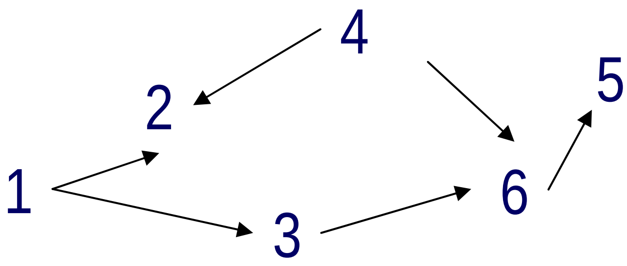
Generate expanded tuples (bindings) (x,y,z)

Positive: (1,6,2), (1,6,3),(1,5,2),(1,5,3)
(3,5,6), (4,5,2),(4,5,6)

Negative:

(1,4,2), (1,4,3)
(3,1,6),(3,2,6),(3,4,6),
(4,1,2),(4,1,6),(4,3,2),(4,3,6),
(6,1,5),(6,2,5),(6,3,5),(6,4,5)

path(x,y):-edge(x,z)



$$|P^{+}| \log \frac{|P^{*}|}{|P^{*}| + |N^{*}|} - \log \frac{|P|}{|P| + |N|}$$

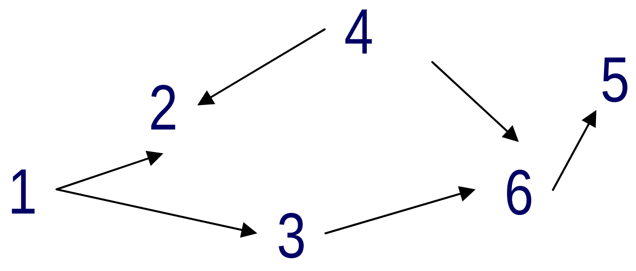
Empty Rule: (P,N)= (4,26)
edge(x,y): (P,N) = (0,0)
edge(x,z): (P,N) = (7,13)
P+ = 4

(note P+≠P)

Example-10

Positive Examples:

(1,6), (1,5),
(3,5), (4,5),



Negative examples;

(1,4), (2,1),(2,3),(2,4),(2,5)
(2,6), (3,1),(3,2),(3,4),(4,1)
(4,3), (5,1),(5,2),(5,3),(5,4)
(5,6), (6,1),(6,2),(6,3),(6,4)

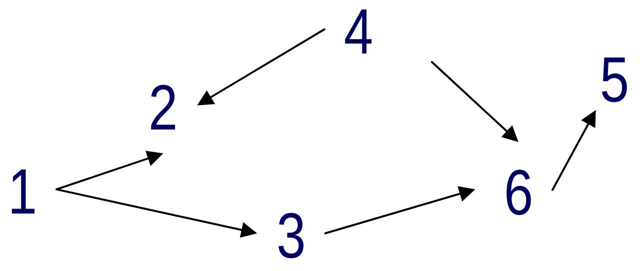
path(x,y):-edge(x,z)

New rule covers all the 4 remaining positives but also 10 of the 20 negatives

Example-10

Positive Examples:
(1,6), (1,5),
(3,5), (4,5),

Negative examples;
(1,4), (2,1),(2,3),(2,4),(2,5)
(2,6), (3,1),(3,2),(3,4),(4,1)
(4,3), (5,1),(5,2),(5,3),(5,4)
(5,6), (6,1),(6,2),(6,3),(6,4)



path(x,y):-edge(x,z)

New rule covers all the 4 remaining positives but also 10 of the 20 negatives

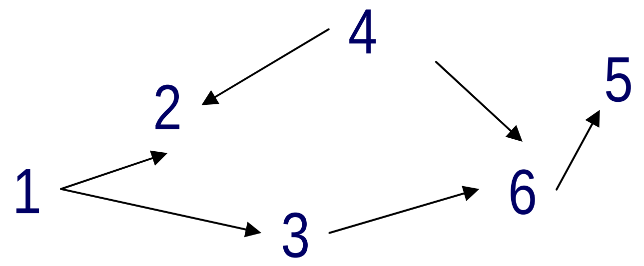
Try to specialize the rule

Example-11

Generate expanded tuples (bindings) (x,y,z)

Positive: (1,6,2), (1,6,3),(1,5,2),(1,5,3)
(3,5,6), (4,5,2),(4,5,6)

Negative:



(1,4,2), (1,4,3)
(3,1,6),(3,2,6),(3,4,6),
(4,1,2),(4,1,6),(4,3,2),(4,3,6),
(6,1,5),(6,2,5),(6,3,5),(6,4,5)

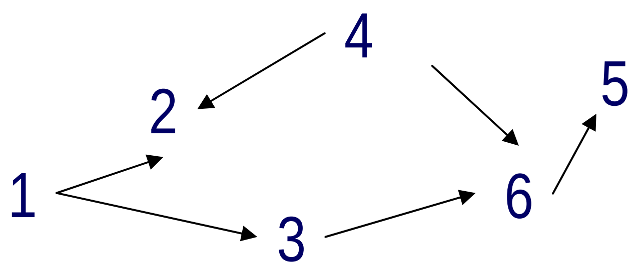
Current Rule:
path(x,y):-edge(x,z)

Consider literal path(z,y)
(as well as edge(x,y),edge(y,z)edge(x,z),path(z,x) etc.)

Example-12

Generate expanded tuples (bindings) (x,y,z)

Positive: (1,6,2), (1,6,3),(1,5,2),(1,5,3)
(3,5,6), (4,5,2),(4,5,6)



Negative:

(1,4,2), (1,4,3)
(3,1,6),(3,2,6),(3,4,6),
(4,1,2),(4,1,6),(4,3,2),(4,3,6),
(6,1,5),(6,2,5),(6,3,5),(6,4,5)

Current rule: path(x,y) : - edge(x,z), path(z,y)

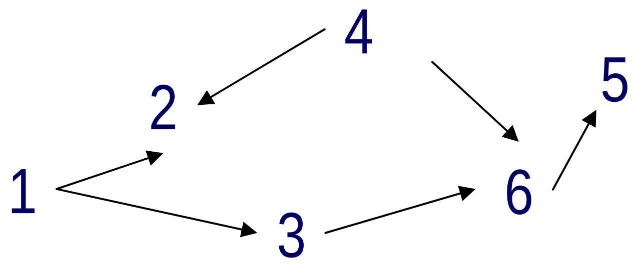
No negative covered. Complete clause.

Example-12

Generate expanded tuples (bindings) (x,y,z)

Positive: (1,6,2), (1,6,3),(1,5,2),(1,5,3)
(3,5,6), (4,5,2),(4,5,6)

Negative:



(1,4,2), (1,4,3)
(3,1,6),(3,2,6),(3,4,6),
(4,1,2),(4,1,6),(4,3,2),(4,3,6),
(6,1,5),(6,2,5),(6,3,5),(6,4,5)

Current rule: path(x,y) : - edge(x,z), path(z,y)

Not all the bindings are satisfied now, but all positive examples are.

Since we cover all positive examples, the definition (using two rules) is complete

More FOIL

- Limitations:
  - Search space for literals can become intractable
  - Hill climbing search
  - Background literals must be sufficient (methods for predicate inventions)
  - In principle: evaluating the body of the rule is intractable (subsumption)
  - In some applications there is a need for a mix of relational and ground literals.

- Applications:
  - Learning Family relations (comparison with Neural Networks)
  - Text categorization based on words and their ordering relations
  - Classifying web pages based on the link structure
  - Learning to take actions
  - Significant success in computational chemistry

Note that (Search in FOIL (2))

At some point in the search we will generate the rule
granddaughter(x,y) ← father(y,z) and mother(z,x) and female(x)
which covers all the positive examples and none of the negatives.
If there are remaining positive examples to be covered, then we begin at this point a search for a new rule.

Works since: The relational rule holds in the data.
We search exhaustively.

- In some sense, this is very similar to propositional learning

y ← A and B and C

In an example (A= ,B= ,C= ,D= ,E= ,……;y) a proposition is either T or F

father(x,y) is also either T of F in an example but, possibly, several things could make it T. (E.g., father(Bob, Sharon),….)

- Problems are introduced when evaluating existential expressions.

Propositionalization

- aunt(x,z) =
  wife(x,y) ^ uncle(y,z) or sister(x,y) ^ father(y,z)

Can we make this a propositional learning problem?

Notes

- Relational Learning
  The learning process is essentially propositional --
  the ground literals are used in the learning process.

- Generalization:
  Done on the relational level as well as the functional level

path(x,y)

path(1,y)   path (x,3)   path(3,y)   ……..

path(1,2), path(1,3), path(1,6), path(1,5), path(3,6),path (3,5),path(4,2)…

- Scaling up:
  Is a major issue

Propositionalization

1. Instead of a rule representation

$$\mathbf{R} = [\forall \mathbf{x},(\exists \mathbf{y},\Phi_1(\mathbf{x},\mathbf{y}) \wedge \Phi_2(\mathbf{x},\mathbf{y})) \rightarrow \mathbf{f(x)}]$$

We use generalized rules:

~~$\mathbf{R} = [\forall \mathbf{x},(\exists \mathbf{y},[\mathbf{w}_1\Phi_1(\mathbf{x},\mathbf{y}) + \mathbf{w}_2\Phi_2(\mathbf{x},\mathbf{y})] \geq 1) \rightarrow \mathbf{f(x)}]$~~

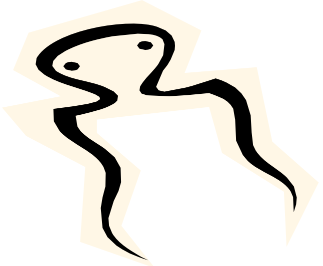
- More expressive; Easier to learn

2. Restrict to Quantified Propositions

Single predicate in scope

$$\mathbf{R'} = [\forall \mathbf{x},[\mathbf{w}_1 \cdot \underline{(\exists \mathbf{y}_1,\mathbf{c}_1(\mathbf{x},\mathbf{y}_1))} + \mathbf{w}_2 \cdot \underline{(\exists \mathbf{y}_2,\mathbf{c}_2(\mathbf{x},\mathbf{y}_2))} > 1] \rightarrow \mathbf{f(x)}]$$

- Allows use of Propositional Algorithms; but more predicates are required to maintain expressivity



Expressivity

$$R = [\forall x, (\exists y, c_1(x,y) \wedge c_2(x,y)) \rightarrow f(x)]$$

Restricting to using quantified proposition

$$R' = [\forall x, ((\exists y_1, c_1(x, y_1)) \wedge (\exists y_2, c_2(x, y_2))) \rightarrow f(x)] \quad \neq R$$

can be overcome using new predicates (features)

$$R'' = [\forall x, y, (c_1(x,y) \wedge c_2(x,y)) \rightarrow f'(x,y)]$$

$$R = [\forall x, (\exists y, f'(x,y)) \rightarrow f(x)]$$

Why Quantified Propositions?

Allow different parts of the program's conditions to be evaluated separately from others.

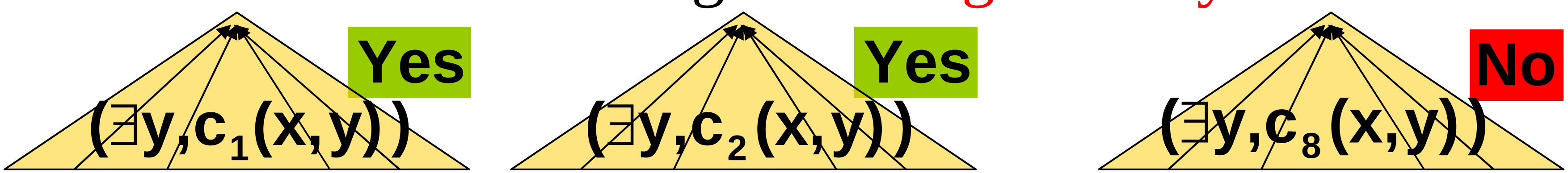
$$R' = [\forall x, ((\exists y_1, c_1(x, y_1)) \wedge (\exists y_2, c_2(x, y_2))) \rightarrow f(x)]$$

Given a sentence -

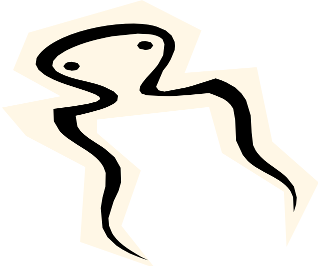
binding of x determines the example

Given a binding -

$(\exists y, c(x,y))$ is assigned a single binary value



$(\text{Sentence}, x = \text{this})$

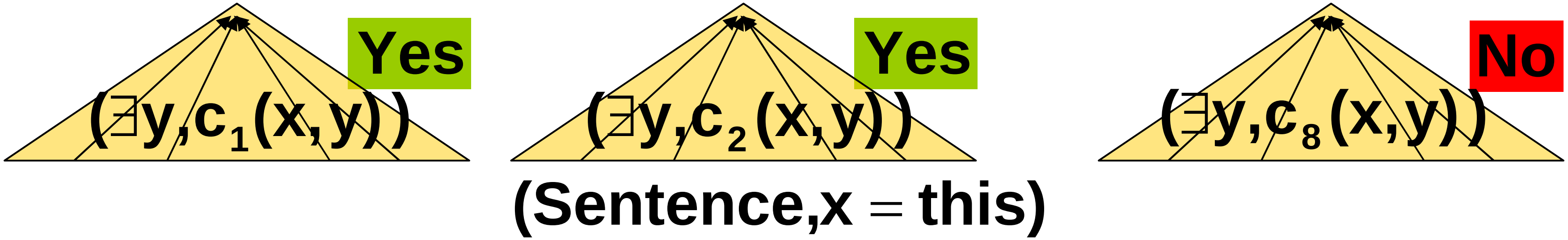


Why Quantified Propositions?

Allow different parts of the program's condition to be evaluated separately from others.

$$R' = [\forall x, ((\exists y_1, c_1(x, y_1)) \wedge (\exists y_2, c_2(x, y_2))) \rightarrow f(x)]$$

For each x: the sentence is mapped into a collection of binary features in the relational space



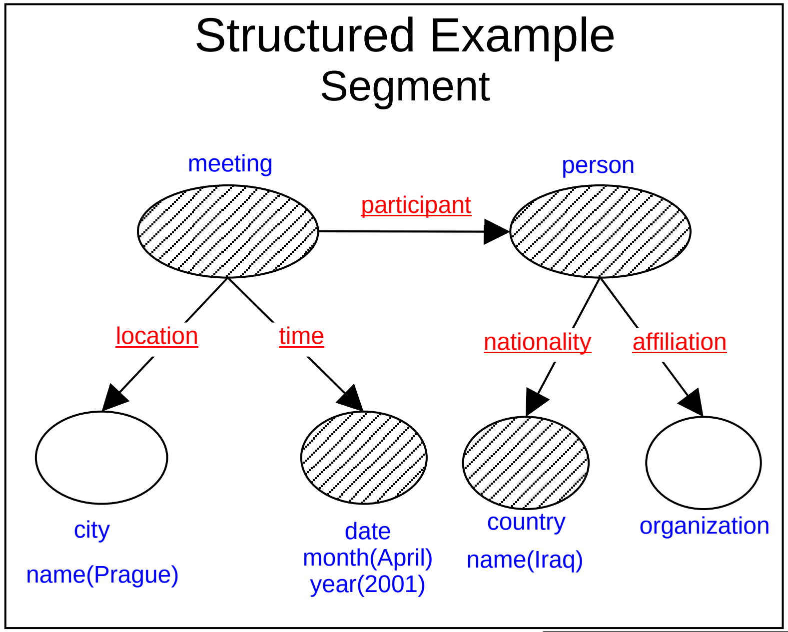
Note that (Search in FOIL (2))

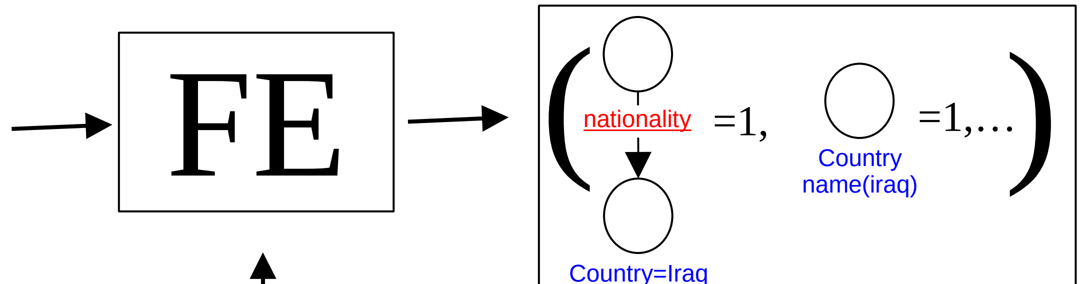
At some point in the search we will generate the rule
granddaughter(x,y) ← father(y,z) and mother(z,x) and female(x)
which covers all the positive examples and none of the negatives.

This can be achieved using a propositional learning algorithm if the Features are FUNCTIONS of the primitive predicates.
The feature space may become very large – but these features are touched anyhow by the relational learning algorithm
Details: [Cumby&Roth, 99, 01; Roth&Yih'01;other propositionalization papers]

Feature Extraction

Features of the type listed below are extracted from the example segment on the left; the binding of the left most feature type is emphasized on the example segment.

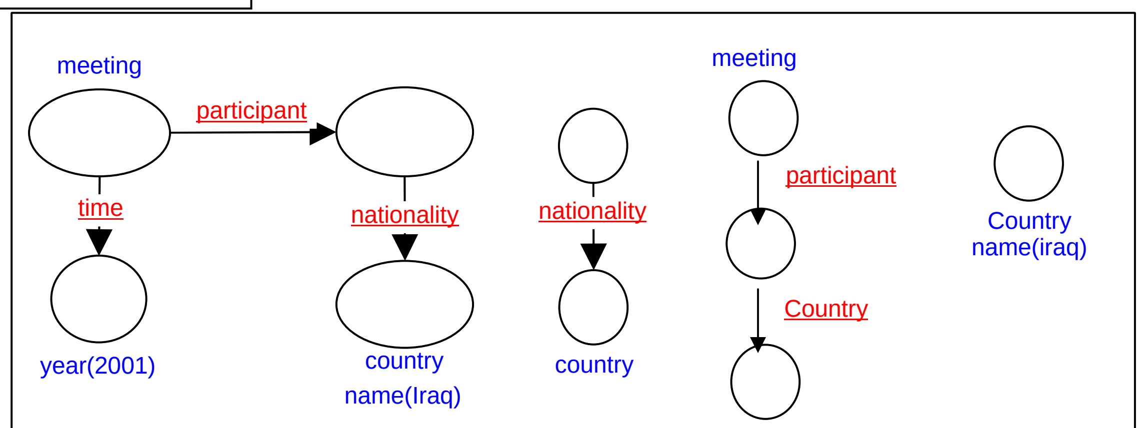




feature vector = list of active substructures(descriptions)

Attributes (node labels)

Roles (edge labels)



Summary: Learning Rules and ILP

- A sequential covering algorithm learns a disjunctive set of rules
  - A greedy algorithm for learning rule sets
  - (different from the "simultaneous" covering of ID3)

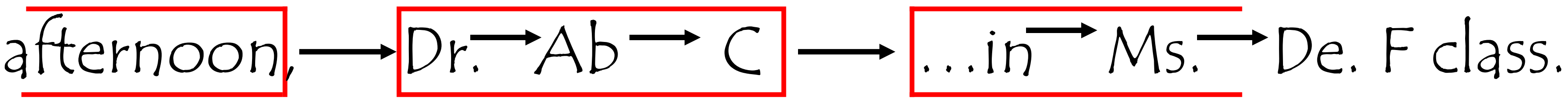
- A variety of methods can be used to learn a single rule:
  - General to specific search
  - Specific to general (LGG) search
  - Various statistical measures may guide the search
- Sets of First Order Rules:
  - Highly expressive representation
  - Extend search techniques from propositional to first-order (FOIL)
  - A few systems exist both for propositional and first order learning
- Active research area: mostly via propositialization

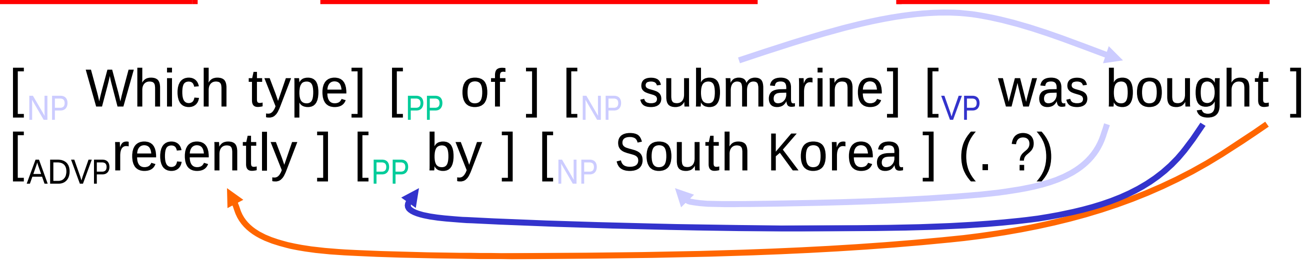
When is ILP usefull?

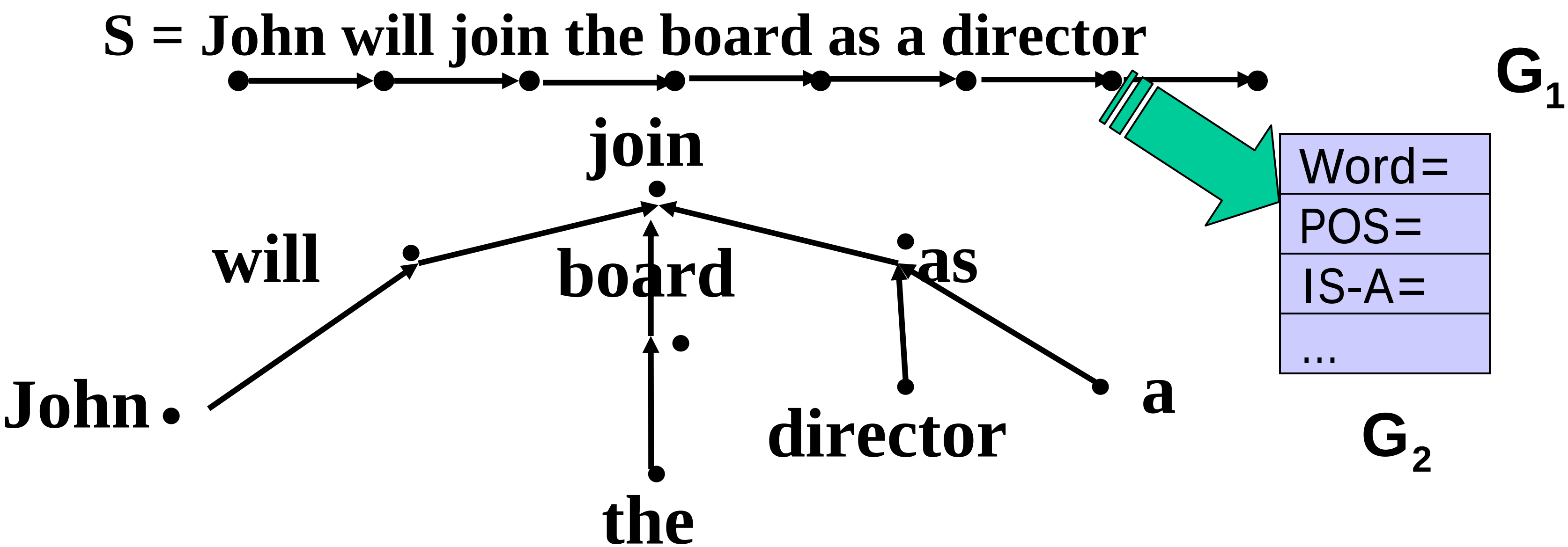
- **ILP is a good choice whenever**
  - relation among considered objects have to be taken into account
  - the training data have no uniform structure (some objects are described extensively, other are mentioned in several facts only)
  - there is extensive background knowledge which should be used for construction of hypothesis

- Key: Good when concise descriptions are good enough
  - No need for a lot of propositional (lexical) information
  - Has been successful in some domains: Bioinformatics, medicine, ecology
  - Needs work: better algorithms

Structured Domain







The boy ran away quickly

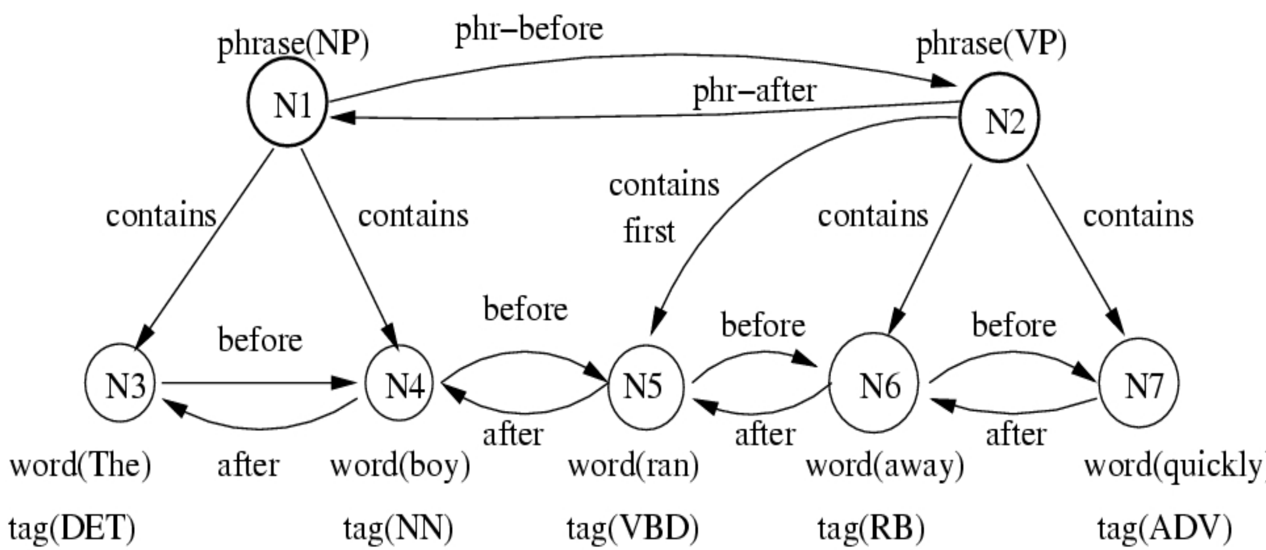


Figure 1: A concept graph for a partial parse of a sentence.

Relational Learning

The theory presented claims that the algorithm runs...

[The theory presented claims] that [the algorithm runs]

Subject(x) = F(after(x,verb),before(x,determiner), noun(x).....)

- Real world data is stored in relational form:

  P is a faculty in department D

  S is a student in Department D

  P is an advisor of S

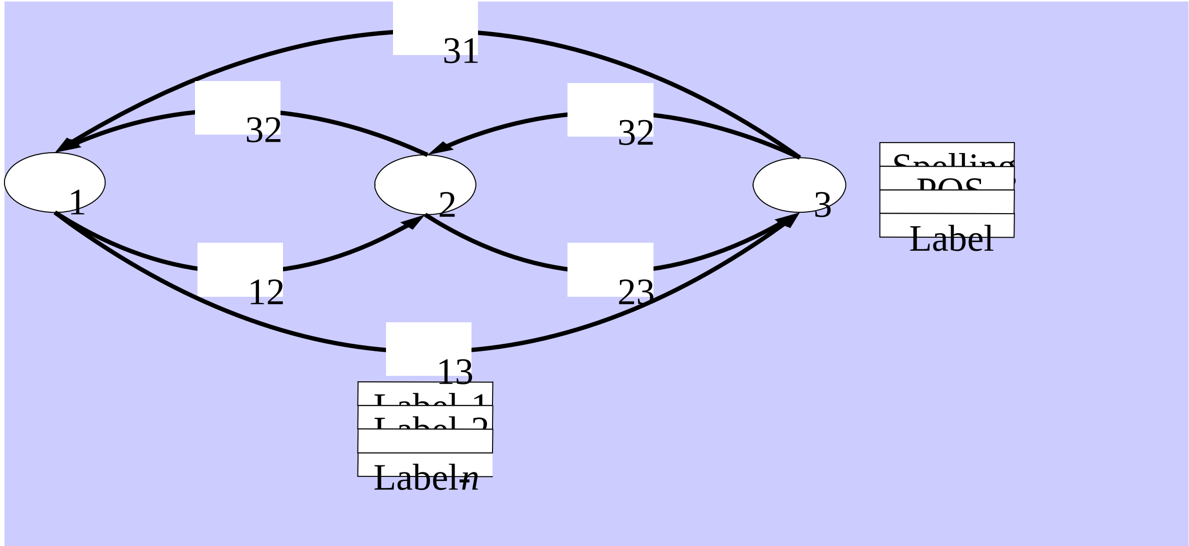
- Is there are need to know the names of the people to say something useful?

> Want to exploit relational information when learning

- Web page classification, e.g, classify Professors pages
  - Assume that you learn on Computer Science web pages?
  - Will it work on Physics web pages?

Structured Domain

- Learn labels on nodes and edges
- Have hypotheses that depends on the structure



Structured Data: Concept Graph Representation

Text: Mohammed Atta met with an Iraqi intelligence agent in Prague in April 2001.