

# Rule Based Learning

- 1 Rules
  - Predicate
  - Horn-clauses
  
- 2 Learning of Predicates
  - Decision Trees
  - Sequential Covering
  
- 3 Inductive Logic Programing
  - Top-Down
  - Bottom-Up

- 1 Rules
  - Predicate
  - Horn-clauses
- 2 Learning of Predicates
  - Decision Trees
  - Sequential Covering
- 3 Inductive Logic Programing
  - Top-Down
  - Bottom-Up

# Predicate

Predicate — Logical expressions

# Predicate

**Predicate** — Logical expressions

Depends in the values of attributes

# Predicate

**Predicate** — Logical expressions

Depends in the values of attributes

$\text{Rainy} \wedge \text{Cold}$

# Horn Clauses

## Horn Clauses — First Order Logic

# Horn Clauses

Horn Clauses — First Order Logic

Can include **variables**

# Horn Clauses

## Horn Clauses — First Order Logic

Can include **variables**

$$\text{sibling}(x, y) : \text{parent}(z, x) \wedge \text{parent}(z, y)$$

# Horn Clauses

## Horn Clauses — First Order Logic

Can include **variables**

$$\text{sibling}(x, y) : \text{parent}(z, x) \wedge \text{parent}(z, y)$$

More expressional power than predicates

Foundation for e.g. *Prolog*

Why would one like to learn rules?

Why would one like to learn rules?

- Can be transformed into "natural language"

Why would one like to learn rules?

- Can be transformed into "natural language"
- Explaining

Why would one like to learn rules?

- Can be transformed into "natural language"
- Explaining

Drawbacks?

Why would one like to learn rules?

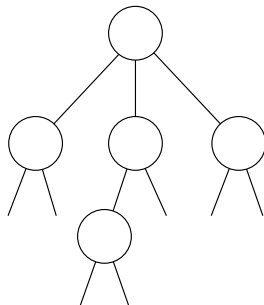
- Can be transformed into "natural language"
- Explaining

Drawbacks?

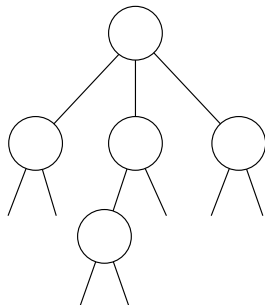
- Hard to handle noise and uncertainty

- ① Rules
  - Predicate
  - Horn-clauses
  
- ② Learning of Predicates
  - Decision Trees
  - Sequential Covering
  
- ③ Inductive Logic Programing
  - Top-Down
  - Bottom-Up

## Decision Trees



## Decision Trees



Disjunction of conjunctions

## Sequential Covering (Sekvensiell täckning)

## Sequential Covering (Sekvensiell täckning)

Idéa underlying *Sequential Covering*

Do not attempt to explain everything at once!

## Sequential Covering (Sekvensiell täckning)

Idéa underlying *Sequential Covering*

Do not attempt to explain everything at once!

- 1 Construct **one** rule

## Sequential Covering (Sekvensiell täckning)

### Idéa underlying *Sequential Covering*

Do not attempt to explain everything at once!

- 1 Construct **one** rule which
  - Matches many positive examples
  - Does not match any negative examples

## Sequential Covering (Sekvensiell täckning)

### Idéa underlying *Sequential Covering*

Do not attempt to explain everything at once!

- 1 Construct **one** rule which
  - Matches many positive examples
  - Does not match any negative examples
- 2 Remove the positive examples which matched

## Sequential Covering (Sekvensiell täckning)

### Idéa underlying *Sequential Covering*

Do not attempt to explain everything at once!

- 1 Construct **one** rule which
  - Matches many positive examples
  - Does not match any negative examples
- 2 Remove the positive examples which matched
- 3 Repeat until all positive examples are removed

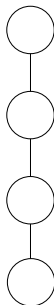
How does one find a good rule?

How does one find a good rule?

**Greedy Search** — Select attributes which matches as many positive examples as possible

How does one find a good rule?

**Greedy Search** — Select attributes which matches as many positive examples as possible



# Beam search

## Beam search

Alternative to greedy search

## Beam search

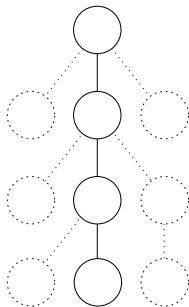
Alternative to greedy search

A few alternative search paths are kept

## Beam search

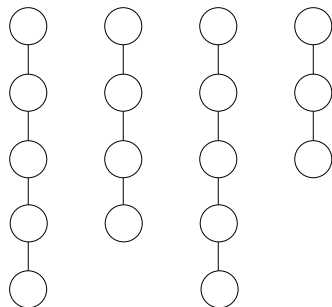
Alternative to greedy search

A few alternative search paths are kept



Result from the search

Result from the search: several rules



- 1 Rules
  - Predicate
  - Horn-clauses
- 2 Learning of Predicates
  - Decision Trees
  - Sequential Covering
- 3 Inductive Logic Programing
  - Top-Down
  - Bottom-Up

# Inductive Logic Programing

## ILP — Inductive Logic Programming

# Inductive Logic Programing

## ILP — Inductive Logic Programming

Automatic construction of Horn Clauses from training data

# Inductive Logic Programing

## ILP — Inductive Logic Programming

Automatic construction of Horn Clauses from training data

### Horn Clauses

- More expressional power than predicates
- Variables in the definitions

# Inductive Logic Programming

## ILP — Inductive Logic Programming

Automatic construction of Horn Clauses from training data

### Horn Clauses

- More expressional power than predicates
- Variables in the definitions

Example

$$\begin{aligned} \text{grandfather}(x, y) : & \text{parent}(x, z) \\ & \wedge \text{male}(x) \\ & \wedge \text{parent}(z, y) \end{aligned}$$

# Inductive Logic Programing

How can a program automatically find the rule

$$\text{grandfather}(x, y) : \text{parent}(x, z) \wedge \text{male}(x) \wedge \text{parent}(z, y)$$

from examples like this

$\text{grandfather}(\textit{Sven}, \textit{Pär}) \mapsto \mathbf{False}$   
 $\text{grandfather}(\textit{Pär}, \textit{Sven}) \mapsto \mathbf{True}$   
 $\text{grandfather}(\textit{Lisa}, \textit{Sven}) \mapsto \mathbf{False}$

- Top-down
- Bottom-up

- Top-down
  - Use Sequential Covering
  - Start from a general rule and specialize
- Bottom-up

- **Top-down**  
Use Sequential Covering  
Start from a general rule and specialize
- **Bottom-up**  
Start from one example and construct explaining rules

Top-Down technique to find **one rule**

Top-Down technique to find **one rule**

- Start being maximally general

Top-Down technique to find **one rule**

- Start being maximally general

`grandfather(x, y) : True`

Top-Down technique to find **one rule**

- Start being maximally general

$\text{grandfather}(x, y) : \mathbf{True}$

- Generate possible specializations

## Top-Down technique to find **one rule**

- Start being maximally general

$\text{grandfather}(x, y) : \mathbf{True}$

- Generate possible specializations

$\text{grandfather}(x, y) : \text{parent}(x, y)$

$\text{grandfather}(x, y) : \text{parent}(x, z)$

$\text{grandfather}(x, y) : \text{female}(x)$

$\text{grandfather}(x, y) : \text{male}(x)$

## Top-Down technique to find **one rule**

- Start being maximally general

$\text{grandfather}(x, y) : \mathbf{True}$

- Generate possible specializations

$\text{grandfather}(x, y) : \text{parent}(x, y)$

$\text{grandfather}(x, y) : \text{parent}(x, z)$

$\text{grandfather}(x, y) : \text{female}(x)$

$\text{grandfather}(x, y) : \text{male}(x)$

- Keep the one which best describes data

## Top-Down technique to find **one rule**

- Start being maximally general

grandfather( $x, y$ ) : **True**

- Generate possible specializations

grandfather( $x, y$ ) : parent( $x, y$ )

grandfather( $x, y$ ) : parent( $x, z$ )

grandfather( $x, y$ ) : female( $x$ )

grandfather( $x, y$ ) : male( $x$ )

- Keep the one which best describes data
- If the rule matches negative examples: **specialize it**

How does one specialize a rule?

How does one specialize a rule?

- Append a conjunction factor

How does one specialize a rule?

- Append a conjunction factor

$\text{grandfather}(x, y) : \text{parent}(x, z) \wedge \text{male}(x)$

$\text{grandfather}(x, y) : \text{parent}(x, z) \wedge \text{male}(z)$

How does one specialize a rule?

- Append a conjunction factor

$\text{grandfather}(x, y) : \text{parent}(x, z) \wedge \text{male}(x)$

$\text{grandfather}(x, y) : \text{parent}(x, z) \wedge \text{male}(z)$

- Choose the best

How does one specialize a rule?

- Append a conjunction factor

$$\text{grandfather}(x, y) : \text{parent}(x, z) \wedge \text{male}(x)$$
$$\text{grandfather}(x, y) : \text{parent}(x, z) \wedge \text{male}(z)$$

- Choose the best

Repeat this until no negative examples match

How are new possible factors generated?

How are new possible factors generated?

- Test **all known rules** with "old" variables

How are new possible factors generated?

- Test **all known rules** with "old" variables

```
grandfather(x, y) : parent(x, y)
                   parent(y, x)
                   male(x)
                   male(y)
```

How are new possible factors generated?

- Test **all known rules** with "old" variables
- Introduce **new variables**, in all but one position

How are new possible factors generated?

- Test **all known rules** with "old" variables
- Introduce **new variables**, in all but one position

```
grandfather(x, y) : parent(x, z)
                   parent(z, x)
                   parent(y, z)
                   parent(z, y)
```

How are new possible factors generated?

- Test **all known rules** with "old" variables
- Introduce **new variables**, in all but one position
- Use **equality** and inequalities

How are new possible factors generated?

- Test **all known rules** with "old" variables
- Introduce **new variables**, in all but one position
- Use **equality** and inequalities

grandfather( $x, y$ ) :  $x = y$   
 $x \neq y$   
 $x < y$   
 $x > y$

How are new possible factors generated?

- Test **all known rules** with "old" variables
- Introduce **new variables**, in all but one position
- Use **equality** and inequalities
- Use **type information** for pruning

## Bottom Up techniques

Bottom Up techniques

Inverse Resolution

Bottom Up techniques

## Inverse Resolution

Use methods from automatic deduction "backwards"

## Bottom Up techniques

### Inverse Resolution

Use methods from automatic deduction "backwards"

- Start from **one example**

## Bottom Up techniques

### Inverse Resolution

Use methods from automatic deduction "backwards"

- Start from **one example**
- Build an explaining rule

## Bottom Up techniques

### Inverse Resolution

Use methods from automatic deduction "backwards"

- Start from **one example**
- Build an explaining rule
- Remove data that is explained

## Bottom Up techniques

### Inverse Resolution

Use methods from automatic deduction "backwards"

- Start from **one example**
- Build an explaining rule
- Remove data that is explained
- Repeat