Methods for Intelligent Systems

Lecture Notes on Machine Learning

Matteo Mattecci

matteucci@elet.polimi.it

Department of Electronics and Information

Politecnico di Milano

Classification Rules

What are Classification Rules?

Classification Rules or Decision Rules are classical IF-THEN rules:

- The IF part states a condition over the data
- The THEN part includes a class label

```
IF (lotion = no) THEN sunburn
```

Depending on the conditions we can have different kind of knowledge representation:

- Propositional, with attribute-value comparisons

$$\texttt{(Overcast = sunny)} \wedge \texttt{(Temperature > 30)} \rightarrow \texttt{PlayGolf}$$

- First order Horn clauses, with variables: $L1 \wedge L2 \wedge L3 \wedge \ldots \wedge Ln \rightarrow H$
  - $H, L1, \ldots, Ln$ are positive literals (predicates applied to terms)
  - H is called head or consequent
  - $L1 \wedge L2 \wedge L3 \wedge \ldots \wedge Ln$ is called body or antecedents

$$\texttt{father(y,x)} \wedge \texttt{female(x)} \rightarrow \texttt{daughter(x,y)}$$

Why shoud we use Classification Rules?

Inductive Learning Hypothesis: any hypothesis (h) found to approximate the target function (t) over a sufficiently large set of training examples (e) will also approximate the target function (t) well over other unobserved examples.

One of the most **expressive** and most **human readable** representation for hypotheses (i.e., models) is sets of IF-THEN rules.
They are also easy to use in Expert Systems.

Learning can be seen as exploring the Hypothesis Space

- **General to Specific:** Start with the most general hypothesis and then go on through specialization steps
- **Specific to General:** Start with the set of the most specific hypothesis and then go on through generalization steps

We'll come back to this soon! ;-)

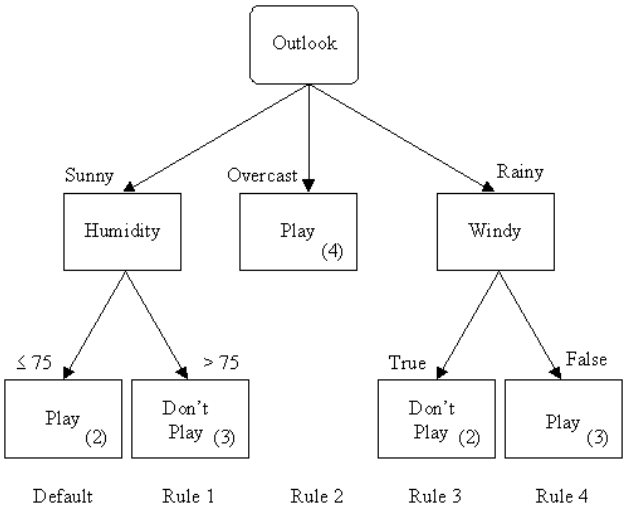
An Example: “Is this a nice day to play golf?”

Outlook	Temp	Humid.	Windy	Play
sunny	85	85	false	No
sunny	80	90	true	No
overcast	83	78	false	Yes
rain	70	96	false	Yes
rain	68	80	false	Yes
rain	65	70	true	No
overcast	64	65	true	Yes
sunny	72	95	false	No
sunny	69	70	false	Yes
rain	75	80	false	Yes
sunny	75	70	true	Yes
overcast	72	90	true	Yes
overcast	81	75	false	Yes
rain	71	80	true	No

How can we learn a set of rule from this data?

- Learn a Decision Tree, from the data then convert it into rules
- Use a Sequential Covering Algorithm

Extracting Rules from the Decision Tree



1. `if (outlook = overcast) then play`
2. `if (outlook = rain) and (windy = false) then play`
3. `if (outlook = sunny) and (humidity = high) then don't play`
4. `if (outlook = rain) and (windy = true) then don't play`
5. `Default class: play`

This is old stuff! Let's try something new!

Sequential Covering Algorithm

In order for a rule to be useful there are two pieces of information to be considered:

- **Accuracy:** How often is the rule correct?
- **Coverage:** How often does the rule apply?

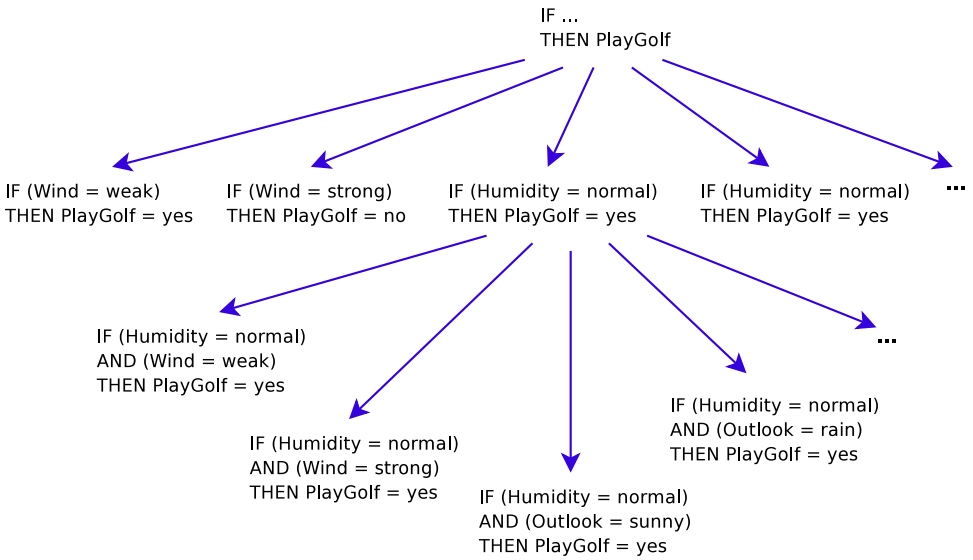
Considering this informations we can use a simple **Sequential Covering** algorithm:

- Consider the set E of positive and negative examples
- Repeat
  - Learn one rule with high accuracy, any coverage
  - Remove positive examples covered by this rule

  Until all the examples are covered

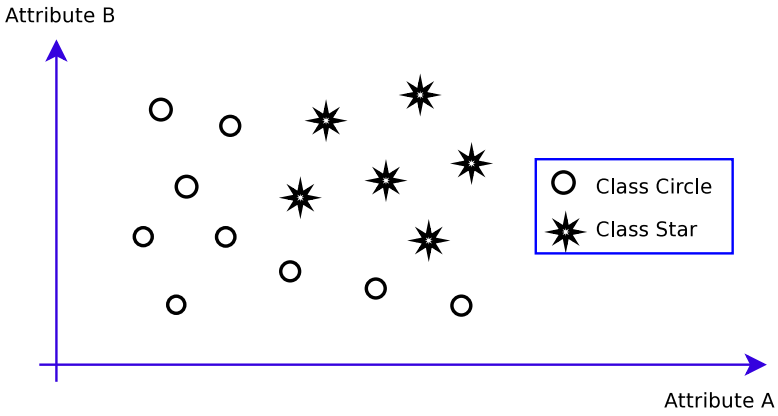
Let's figure out the algorithm!

Learning One Rule



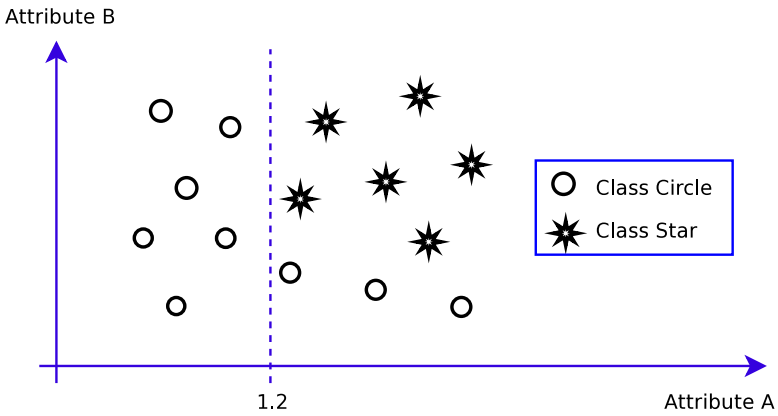
- Start from the most general rule, consisting of an empty condition
- Add tests on single attributes until the accuracy improves . . .

Learning One Rule Example: The `Star` Class



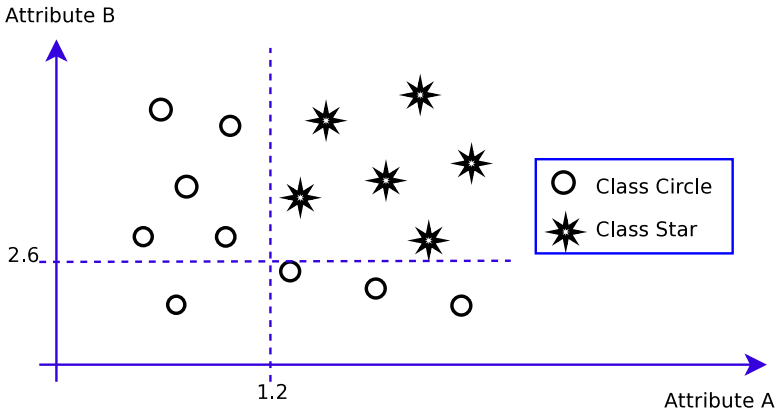

```
IF True
THEN Star
```


Learning One Rule Example: The `Star` Class




```
IF A>1.2
THEN Star
```

Learning One Rule Example: The `Star` Class




```
IF A>1.2
AND B>2.6
THEN Star
```

- Rule for `Star` class:
  - `IF A > 1.2 AND B > 2.6 THEN Star`
- Possible rules for `Circle` class:
  - `IF A < 1.2 THEN Circle`
  - `IF A > 1.2 AND B < 2.6 THEN Circle`

Contact Lens Example (I)

age	prescription	astigmatism	Tear production	lenses
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
middle-aged	myope	no	reduced	none
middle-aged	myope	no	normal	soft
middle-aged	myope	yes	reduced	none
middle-aged	myope	yes	normal	hard
middle-aged	hypermetrope	no	reduced	none
middle-aged	hypermetrope	no	normal	soft
middle-aged	hypermetrope	yes	reduced	none
middle-aged	hypermetrope	yes	normal	none
old	myope	no	reduced	none
old	myope	no	normal	none
old	myope	yes	reduced	none
old	myope	yes	normal	hard
old	hypermetrope	no	reduced	none
old	hypermetrope	no	normal	soft
old	hypermetrope	yes	reduced	none
old	hypermetrope	yes	normal	none

- The rule we seek:
  `IF ?`
  `THEN Recommendation = hard`
- Possible test:

`Age = Young`	2/8	
`Age = middle-aged`	1/8	
`Age = old`	1/8	
`Prescr. = myope`	3/12	
`Prescr. = hypermetrope`	1/12	
`Astigmatism = no`	0/12	
`Astigmatism = yes`	4/12	←
`Tear prod. = reduced`	0/12	
`Tear prod. = normal`	4/12	←

- With ties pick the one with higher coverage or random . . .

Contact Lens Example (II)

Actual rule: `IF Astigmatism = Yes THEN Recommendation = hard`

age	prescription	astigm atism	Tear production	lenses
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
middle-aged	myope	yes	reduced	none
middle-aged	myope	yes	normal	hard
middle-aged	hypermetrope	yes	reduced	none
middle-aged	hypermetrope	yes	normal	none
old	myope	yes	reduced	none
old	myope	yes	normal	hard
old	hypermetrope	yes	reduced	none
old	hypermetrope	yes	normal	none

- Try to get something more accurate:
  `IF Astigmatism = Yes`
  `AND ?`
  `THEN Recommendation = hard`
- Possible test:

`Age = young`	2/4
`Age = middle-aged`	1/4
`Age = old`	1/4
`Prescr. = myope`	3/6
`Prescr. = hypermetrope`	1/6
`Tear prod. = reduced`	0/6
`Tear prod. = normal`	4/6 ←

Contact Lens Example (III)

```
IF Astigmatism = Yes AND Tear_Production = normal
THEN Recommendation = hard
```

age	prescription	astigm atism	Tear production	lenses
young	myope	yes	normal	hard
young	hypermetrope	yes	normal	hard
middle-aged	myope	yes	normal	hard
middle-aged	hypermetrope	yes	normal	none
old	myope	yes	normal	hard
old	hypermetrope	yes	normal	none

- Try to get something even more accurate:

`Age = young`	2/2
`Age = middle-aged`	1/2
`Age = old`	1/2
`Prescr. = myope`	3/3 ←
`Prescr. = hypermetrope`	1/3

With ties prefer the attribute with higher coverage:

```
IF Astigmatism = Yes
AND Tear_Production = normal
AND Prescription = myope
THEN Recommendation = hard
```

age	prescription	astigm atism	Tear production	lenses
young	myope	yes	normal	hard
young	hypermetrope	yes	normal	hard
middle-aged	myope	yes	normal	hard
old	myope	yes	normal	hard

Contact Lens Example (IV)

We can now check the coverage of the rule we have found

age	Spectacle prescription	astigmatism	Tear production rate	Recommended lenses
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
middle-aged	myope	no	reduced	none
middle-aged	myope	no	normal	soft
middle-aged	myope	yes	reduced	none
middle-aged	myope	yes	normal	hard
middle-aged	hypermetrope	no	reduced	none
middle-aged	hypermetrope	no	normal	soft
middle-aged	hypermetrope	yes	reduced	none
middle-aged	hypermetrope	yes	normal	none
old	myope	no	reduced	none
old	myope	no	normal	none
old	myope	yes	reduced	none
old	myope	yes	normal	hard
old	hypermetrope	no	reduced	none
old	hypermetrope	no	normal	soft
old	hypermetrope	yes	reduced	none
old	hypermetrope	yes	normal	none

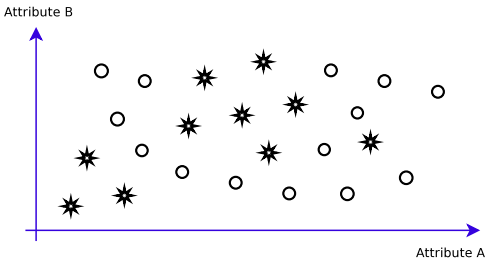
```
IF Astigmatism = Yes
AND Tear_Production = normal
AND Prescription = myope
THEN Recommendation = hard
```

Remove these cases from the dataset and a pply the learning process to the new dataset

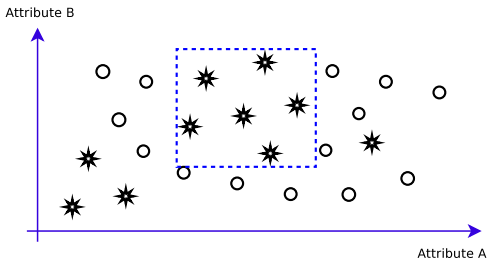
```
IF Age = young
AND Astigmatism = Yes
AND Tear_Production = normal
THEN Recommendation = hard
```

Can guess any flow of this algorithm?
Overfitting is behind the corner!

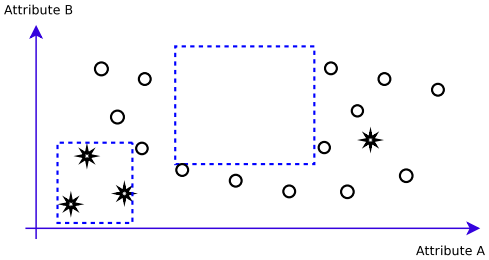
Overfitting the `Star` Class



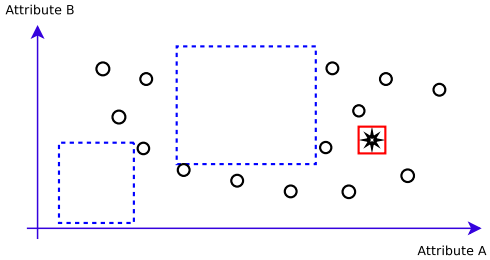
Suppose we have these data



Learn a rule



Remove covered and learn a new rule



Remove covered and learn a new rule

Watch out for too specific rules with low coverage!

Containing Overfitting

The learn-one-rule algorithm on the previous slides tries to learn a rule as accurate as possible. Such a rule may:

- be very complicated (containing many conjuncts)
- exhibit low predictive accuracy on unseen examples
- fit into noise

Use approximated, but simpler rules:

- Pre-pruning: Stop refinement of rules although still not accurate
  - **Minimum Purity Criterion:** Requires a certain percentage of the examples covered by the rules is positive
  - **Significance Testing:** Verify significant differences between the distribution of instances covered by a rule and its direct predecessor.
- Post-pruning: Learn “pure” rules than remove some attributes

The latter used to work better, let’s focus on it!

Post-Pruning of Classification Rules

The general idea of Post-Pruning is to learn a “pure” rule first then remove some attribute value pairs from the rule to make it more general:

- Method 1: Separate the training data into training set and validation set; learn rules from the training set and test the accuracy of the pruned rule on a validation set.
- Method 2: Use a rule quality measure, test the quality of the pruned rule on the same training set from which is rule was learned:
  - Compute a rule quality value: Q(R)
  - Check each attribute-value pair L in R to see if removal of L decreases Q(R)
  - If not, L is removed (i.e., R is generalized)
  - Repeat the process until no further improvement is possible

For instance, ELEM2 uses rule quality formula: $Q(r) = \log \frac{P(r|c)(1-P(r|\bar{c})}{P(r|\bar{c})(1-P(r|c)}$.

Using Classification Rules

When matching a new example with a set of rules:

1. Single-match: only one rule is matched with the example then classify the example into the class indicated by the rule.
2. Multiple-match: multiple rules are matched with the example:
   - If rules indicate the same class then classify into that class.
   - When matched rules indicate different classes:
     - Method 1: Rank the rules, use the first matched rule to classify
     - Method 2: Compute a decision score for each of the involved classes: $DS(C) = \sum_i Q(r_i)$ where $Q(r_i)$ is a quality measure of rule r_i choose the one with the highest decision score.
3. No-match: there is no matching rule
   - Method 1: Use a default (majority class) to classify
   - Method 2: Partial matching is performed. Calculate a matching score for each partially matched rule and a decision score for each involved class. Choose the class with the highest decision score.

Classification Rules vs. Decision Trees

They can represent the same kind of knowledge:

- Decision rules are easier to understand (human readable)
- Rules are more flexible than decision trees
  - No overlapping among branches in a decision tree so they do not suffer from replicated subtrees
  - Branches in a decision tree share at least one attribute (you can always translate a tree into a rule but not the viceversa)
- In multi-class situations, covering algorithm concentrates on one class at a time whereas decision tree learner takes all classes into account
- Sequential covering maximizes single rule's accuracy reducing its coverage while decision tree maximizes overall purity
- If-then rules can be used with expert systems

Still remains for both of them the issue of selecting the *best attribute* and dealing with *missing values* or *continuous attributes*.

Selecting Attribute-Value Pairs

Various methods for selecting attributes have been proposed with the goal of improve rule accuracy

- Training accuracy of the rule on the training example (AQ15): p/t
  - t instances covered by rule, p number of these that are positive
  - Produce rules for positive instances as quickly as possible
  - May produce rules with very small coverage
- Information gain (CN2, PRISM): $\log_2 p/t - \log_2 P/T$
  - P and T are the positive and total number of examples that satisfied the previous rule
  - Equivalent to p/t
- Information gain with coverage (FOIL): $p(\log_2 p/t - \log_2 P/T)$
  - P and T are the positive and total number of examples that satisfied the previous rule
  - *Coverage* is also considered in the evaluation.

Dealing with Missing Values

In certain cases, the available data may be missing values for some attributes, these records will fail any test.

There a few strategies for dealing with the missing attribute value:

- Treat “missing” as a separate value
- Assign it the value that is most common among training examples
- Assign it the most common value among examples that have the same classification
- Assign a probability to each of the possible values rather than simply assigning the most common one. These probabilities can be estimated based on the observed frequencies among the examples.

Given a Boolean attribute A, if we have 6 known examples with A = 1 and 4 with A = 0, then we would say the probability that A(x) = 1 is 0.6, and the probability that A(x) = 0 is 0.4. A fractional 0.6 of records can be assumed to have A = 1 and the rest A = 0. (C4.5)

How can I use the classifiers learned this way?

Dealing with Continuous Values (I)

Discretize numeric attributes by dividing each of them into intervals:

- Sort instances according to attribute's values
- Place breakpoints where the class changes (the majority class)
- This minimizes the total error

Example: Temperature from weather data

Outlook	Temp.	Humid.	Windy	Play
sunny	85	85	false	No
sunny	80	90	true	No
overcast	83	86	false	Yes
rain	75	80	false	Yes
...	...	...	...	...

- Very sensitive to noise
- Unique ID attributes will have zero errors
- Incorrect exmaple classisi-cations induce splits

64 | 65 | 68 69 70 | 71 72 72 | 75 75 | 80 | 81 83 | 85
Yes | No | Yes Yes Yes | No No Yes | Yes Yes | No | Yes Yes | No

Dealing with Continuous Values (II)

Example: Temperature from weather data

Outlook	Temp.	Humid.	Windy	Play
sunny	85	85	false	No
sunny	80	90	true	No
overcast	83	86	false	Yes
rain	75	80	false	Yes
...	...	...	...	...

- Very sensitive to noise
- Unique ID attributes will have zero errors
- Incorrect exmaple classisi-cations induce splits

64 | 65 | 68 69 70 | 71 72 72 | 75 75 | 80 | 81 83 | 85
Yes | No | Yes Yes Yes | No No Yes | Yes Yes | No | Yes Yes | No

Simple solution: enforce a minimum number of majority class instances per each interval; for instance, with min = 3) we get:

64 65 68 69 70 | 71 72 72 75 75 | 80 81 83 85
Yes No Yes Yes Yes | No No Yes Yes Yes | No Yes Yes No

Rule Learning Alternatives

The Sequential Learning Algorithm is non the only learning algorithm we can use to learn rules from a dataset of examples:

- The **1R** algorithm learns a 1-level decision tree, i.e., rules that all test one particular attribute
  - One branch for each value, each branch assigns most frequent class
  - Choose attribute with lowest error rate (i.e., proportion of instances that do not belong to the majority class of their corresponding branch)
- The **RISE** algorithm (Rule Induction from a Set of Exemplars) works in a specific-to-general approach:
  - Initially, it creates one rule for each training example
  - Then it goes on through elementary generalization steps until the overall accuracy does not decrease

Check for their pseudo-code!

Learning First Order Rules: the Monk 1 dataset

Let's Play a little the simple `Monk 1` dataset:

A1	A2	A3	A4	A5	A6	C
1	1	1	1	3	1	1
1	1	1	1	3	2	1
1	1	1	3	2	1	1
1	1	1	3	3	2	1
1	1	2	3	1	2	1
1	2	1	1	1	2	1
1	2	1	1	2	1	0
1	2	1	1	3	1	0
1	2	1	1	4	2	0
1	2	1	2	1	1	1
1	2	1	2	3	1	0
1	2	1	2	3	2	0

```
attribute #5 = 1: 1 (29.0/1.4)
attribute #5 = 2: 0 (31.0/13.4)
attribute #5 = 3:
|   attribute #6 = 1: 0 (13.0/4.7)
|   attribute #6 = 2:
|   |   attribute #3 = 1: 1 (7.0/3.4)
|   |   attribute #3 = 2: 0 (10.0/4.6)
attribute #5 = 4:
|   attribute #1 = 1: 0 (14.0/2.5)
|   attribute #1 = 2:
|   |   attribute #2 = 1: 0 (6.0/1.2)
|   |   attribute \#2 = 2: 1 (4.0/1.2)
|   |   attribute #2 = 3: 0 (1.0/0.8)
|   attribute #1 = 3:
|   |   attribute #2 = 1: 1 (0.0)
|   |   attribute #2 = 2: 0 (3.0/1.1)
|   |   attribute #2 = 3: 1 (6.0/1.2)
```

Learning First Order Rules Can Help!

The previous Decision Tree translates into the following rules:

- Rule 1: `IF attribute #5=1 THEN 1`
- Rule 7: `IF attribute #1=1 AND attribute #2=1 THEN 1`
- Rule 14: `IF attribute #1=1 AND attribute #5=4 THEN 0`
- Rule 17: `IF attribute #1=2 AND attribute #2=2 THEN 1`
- Rule 20: `IF attribute #1=3 AND attribute #2=3 THEN 1`
- Rule 16: `IF attribute #1=2 AND attribute #2=1 AND attribute #5=4 THEN 0`
- Default class: `0`

Decision trees and classification rules do not include variables, but only propositions, if we could use variables then

`(A1 = A2) OR (A5=1)`

We need an algorithm to learn Horn clauses!

Learning First Order Rules with FOIL

The idea: extend typical sequential covering algorithms to learn first order rules, or Horn clauses:

$$L1 \wedge L2 \wedge L3 \wedge \ldots \wedge Ln \rightarrow H$$

- $H, L1, \ldots, Ln$ are positive literals (predicates applied to terms)
- H is called head or consequent
- $L1 \wedge L2 \wedge L3 \wedge \ldots \wedge Ln$ is called body or antecedents

Example: `father(y,x)` $\wedge$ `female(x)` $\rightarrow$ `daughter(x,y)`

The most known algorithm is FOIL that employs an approach very similar to the sequential covering algorithms

- FOIL rules are more restricted than general Horn clauses (literals cannot contain functions)
- FOIL rules are more expressive than Horn clauses because the literals appearing in the body may be negated