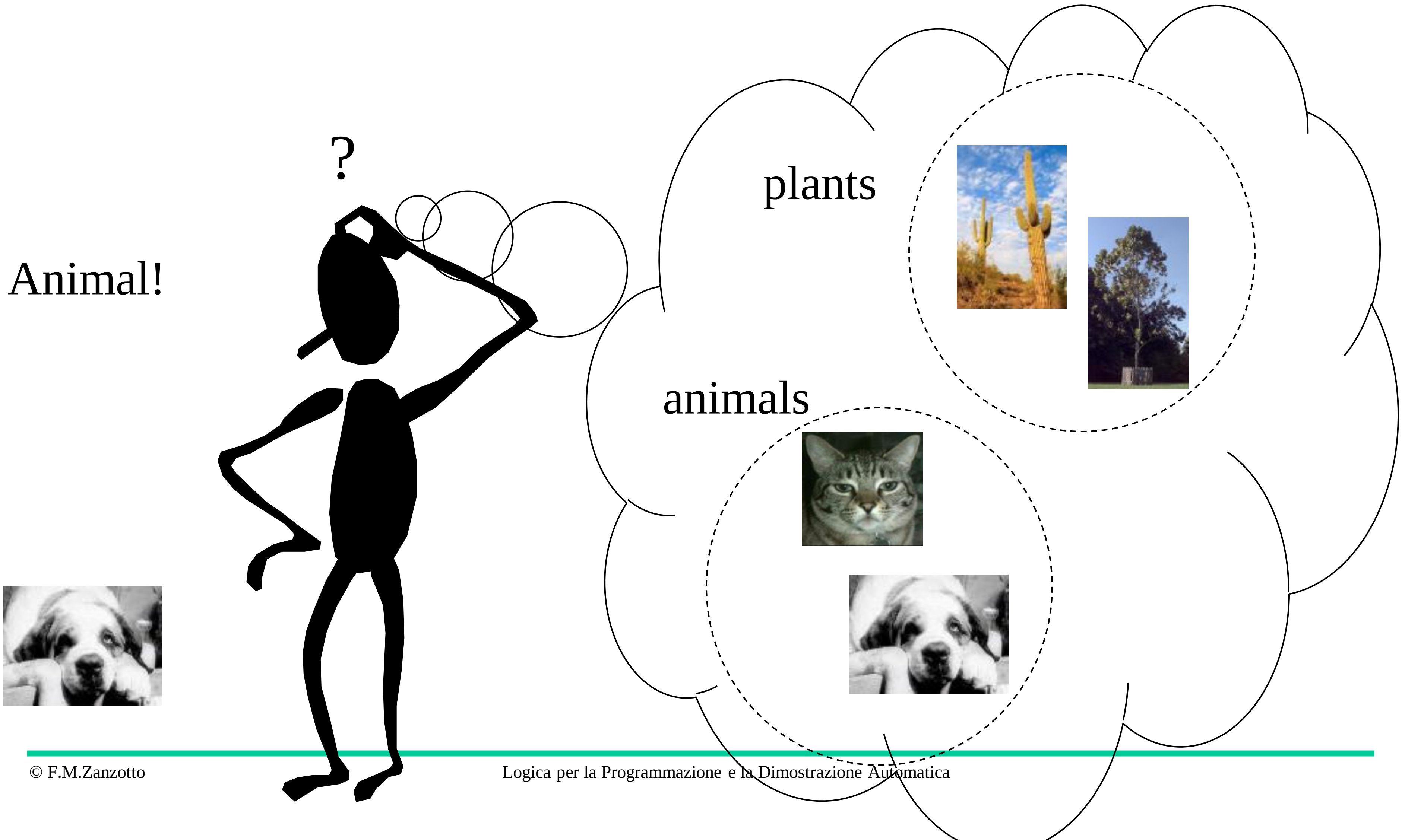
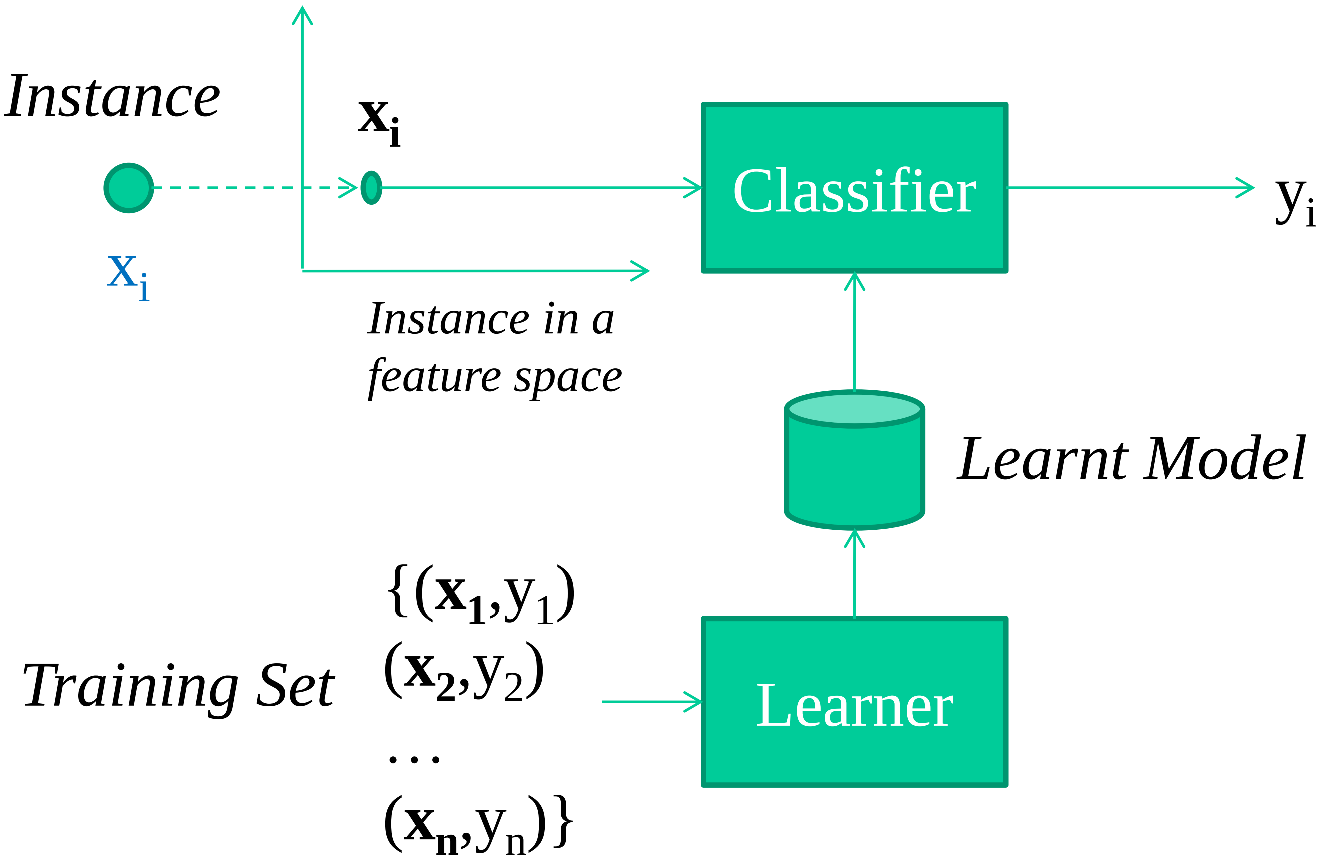
Cenni di Machine Learning: Decision Tree Learning

Fabio Massimo Zanzotto

Concept Learning… what?



Quick background on Supervised Machine Learning



Learning…

What?

- Concepts/Classes/Sets

How?

- using similarities among objects/instances

Where these similarities may be found?

Observation Space: Feature Space

A Feature Space (FS) is a vector space defined as:

$$FS = F_1 \times \ldots \times F_n$$

an instance *i* is a point in the feature space:

$$i = (f_1, \ldots, f_n) \in FS$$

A classification function is:

$$Tagger : FS \rightarrow T$$

where T is the set of the target classes

Observation Space: Feature Space

- Pixel matrix?
- Ditribution of the colors
- Features as:
  - has leaves?
  - is planted in the ground?
  - ...

Note: in deciding the feature space some prior knowledge is used

Definition of the classification problem

- Learning requires positive and negative examples
- Verifying learning algorithms require test examples

- Given a set of instances *I* and a desired target function tagger *Tagger*, we need annotators to define training and testing examples

Annotation is an hard work

- Annotation procedure:
  - definition of the scope: the target classes *T*
  - definition of the set of the instances *I*
  - definition of the annotation procedure
  - actual annotation

Annotation problems

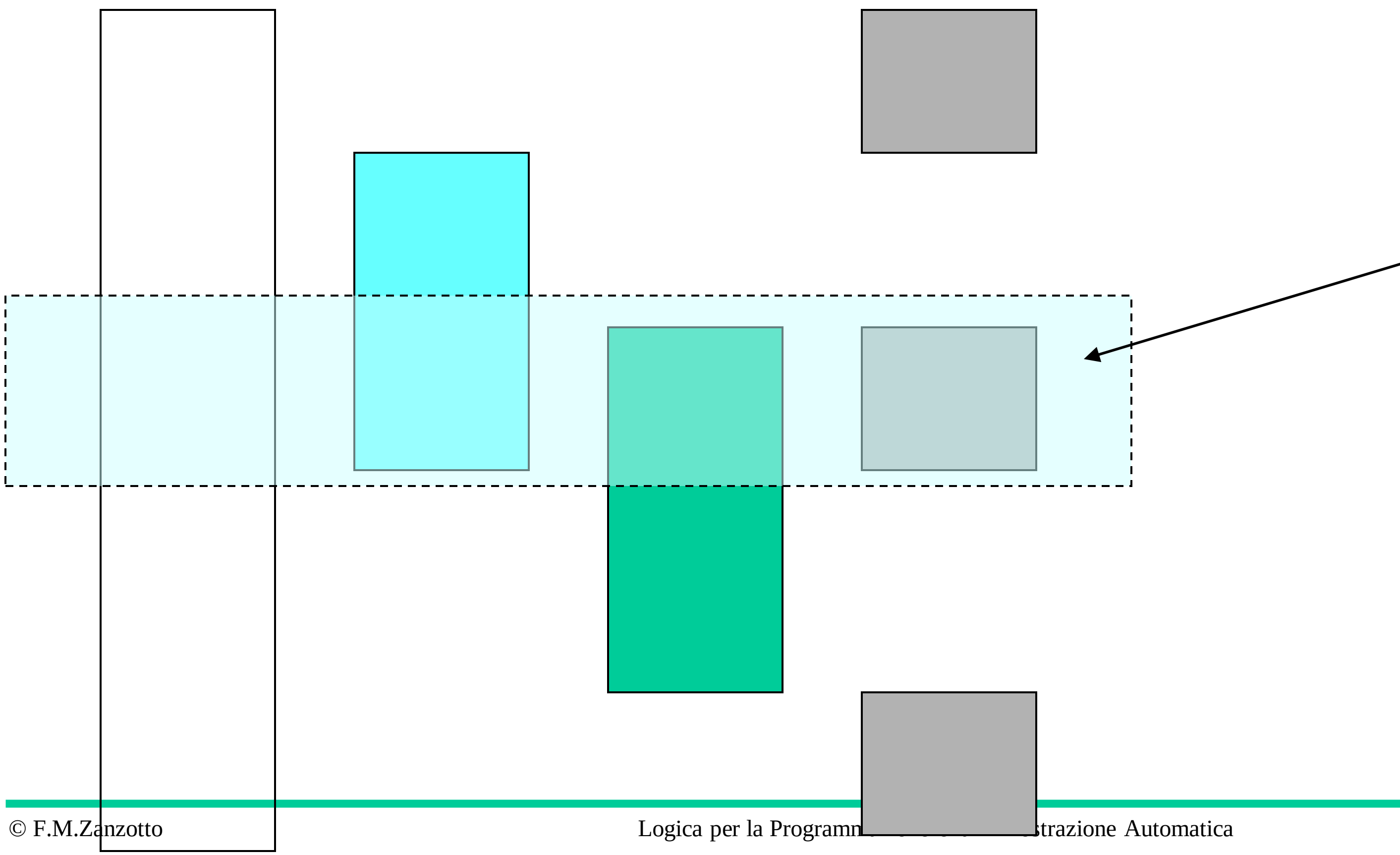
- Are the target classes well defined (and shared among people)?
- How is possible to measure this?
  - ➔ inter-annotator agreement

instances should be annotated

by more than one annotators

Speeding up the annotation...

... and controlling the results



section for the inter-annotator agreement

Annotator 1

Annotator 2

Annotator 3

Instances in *I*

After the annotation

- We have:
  - the behaviour of a tagger

$$\text{Tagger}_{\text{oracle}}: I \rightarrow T$$

for all the examples in *I*

Using the annotated material

- Supervised learning:
  - I is divided in two halves:
    - $I_{training}$: used to train the Tagger
    - $I_{testing}$: used to test the Tagger

  or

  - I is divided in n parts, and the training-testing is done n times (n-fold cross validation), in each iteration:
    - n-1 parts are used for training
    - 1 part is used for testing

Evaluation Metrics

Given the oracle: $Tagger_{oracle}: I \rightarrow T$

Accuracy of the tagger: $Tagger : FS \rightarrow T$

where $$Accuracy = \frac{1}{|I_{testing}|} \sum_{i \in I_{testing}} \delta(Tagger(i), Tagger_{oracle}(i))$$

$$\delta(Tagger(i), Tagger_{oracle}(i)) = \begin{cases} 0 & \text{if } Tagger(i) \neq Tagger_{oracle}(i) \\ 1 & \text{if } Tagger(i) = Tagger_{oracle}(i) \end{cases}$$

Precision/Recall/F-measure

needed for taggers that assign more than one category, i.e.,

$$Tagger : FS \rightarrow 2^T$$

defining:

$System=\{(i,t)|i \in I_{testing} \text{ and } t \in Tagger(i)\}$

$Oracle=\{(i,t)|i \in I_{testing} \text{ and } t=Tagger_{oracle}(i)\}$

precision and recall of the system are defined as:

$Precision = |System \cap Oracle| / |System|$

$Recall = |System \cap Oracle| / |Oracle|$

Categorizzazione: come avviene?

- *Attributo definitorio*
  - Studiata e provata sulle reti di concetti (più il concetto richiesto è distante dell'attributo definitorio più il tempo di risposta è alto)
- *Basata su esempi (recenti)*
- *Prototipo*

Categorizzazione: in apprendimento automatico

- *Attributo definitorio*
  - Decision trees and decision tree learning
- *Basata su esempi (recenti)*
  - *K-neighbours and lazy learning*
- *Prototipo*
  - *Class Centroids and Rocchio Formula*

Categorizzazione: in apprendimento automatico

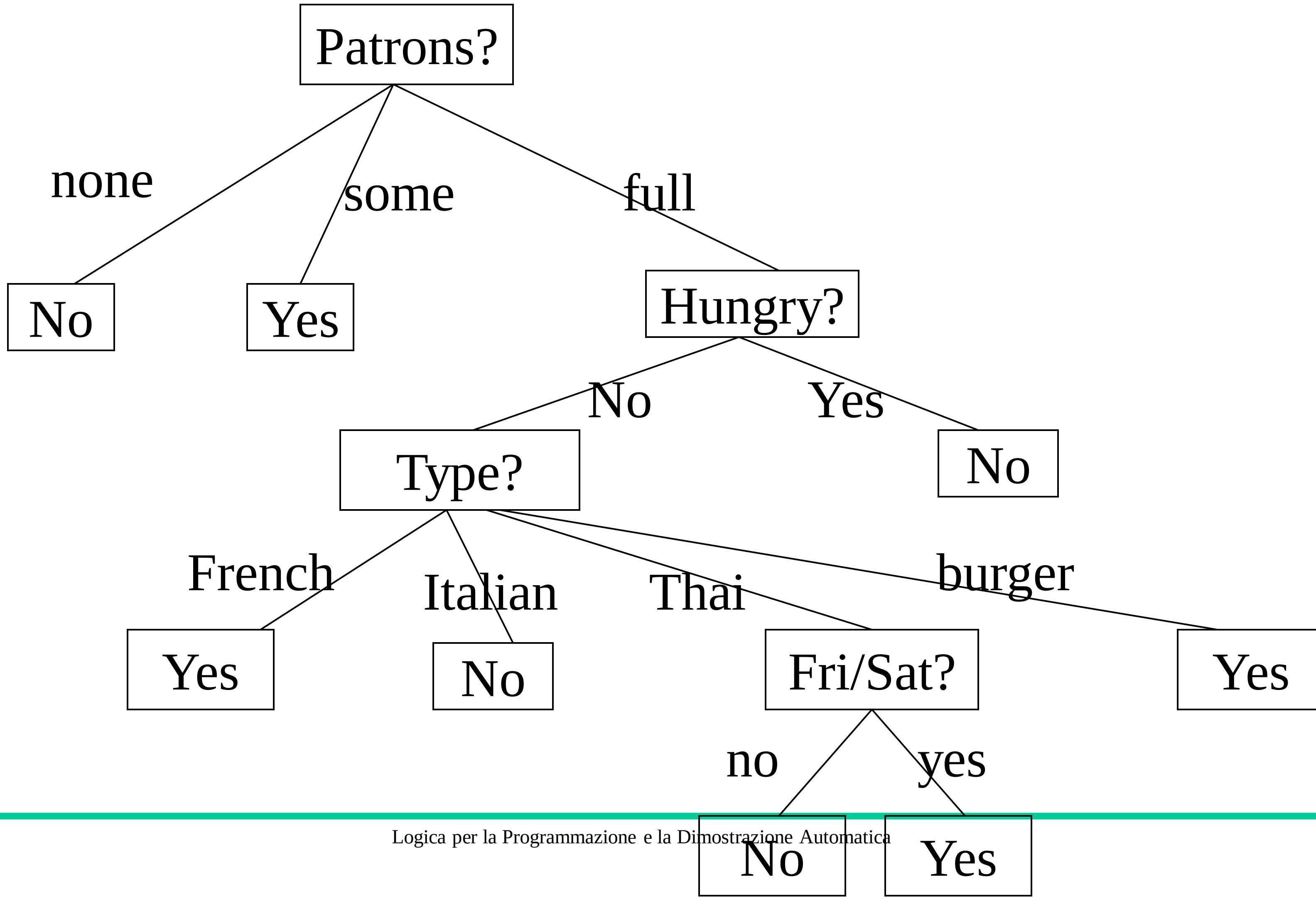
- *Attributo definitorio*
  - Decision trees and decision tree learning

Decision Tree: example

Waiting at the restaurant.

- Given a set of values of the initial attributes, foresee whether or not you are going to wait.

Decision Tree: how is it?



The Restaurant Domain

	Attributes								Goal
Example	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X_1	No	Yes	Some	$$$	No	Yes	French	0-10	Yes
X_2	No	Yes	Full	$	No	No	Thai	30-60	No
X_3	No	No	Some	$	No	No	Burger	0-10	Yes
X_4	Yes	Yes	Full	$	No	No	Thai	10-30	Yes
X_5	Yes	No	Full	$$$	No	Yes	French	>60	No
X_6	No	Yes	Some	$$	Yes	Yes	Italian	0-10	Yes
X_7	No	No	None	$	Yes	No	Burger	0-10	No
X_8	No	Yes	Some	$$	Yes	Yes	Thai	0-10	Yes
X_9	Yes	No	Full	$	Yes	No	Burger	>60	No
X_{10}	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	No
X_{11}	No	No	None	$	No	No	Thai	0-10	No
X_{12}	Yes	Yes	Full	$	No	No	Burger	30-60	Yes

Will we wait, or not?

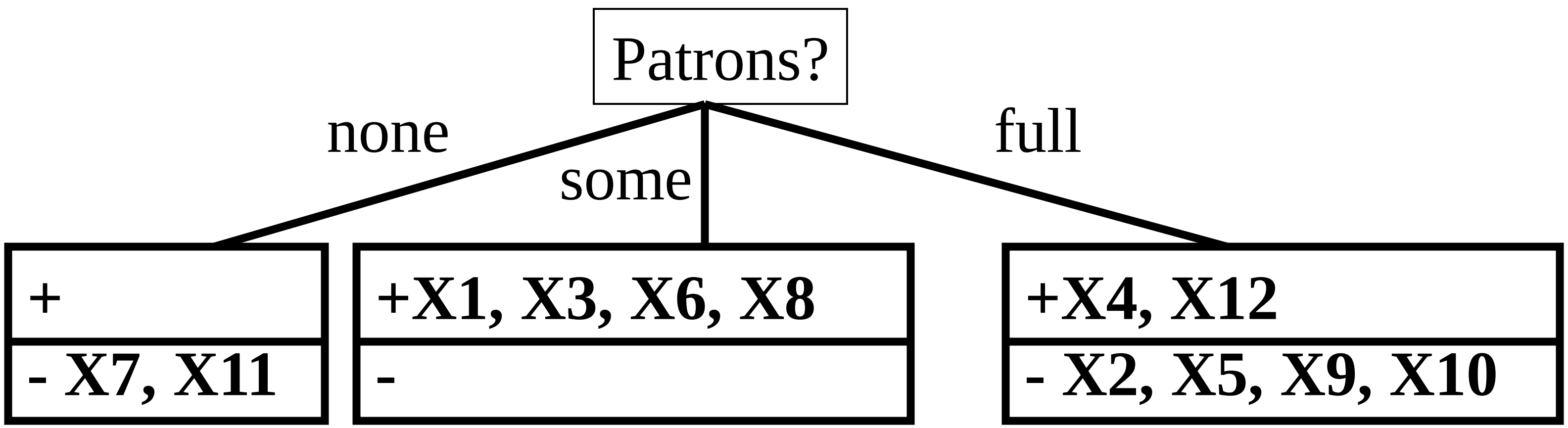
http://www.cs.washington.edu/education/courses/473/99wi/

Splitting Examples by Testing on Attributes

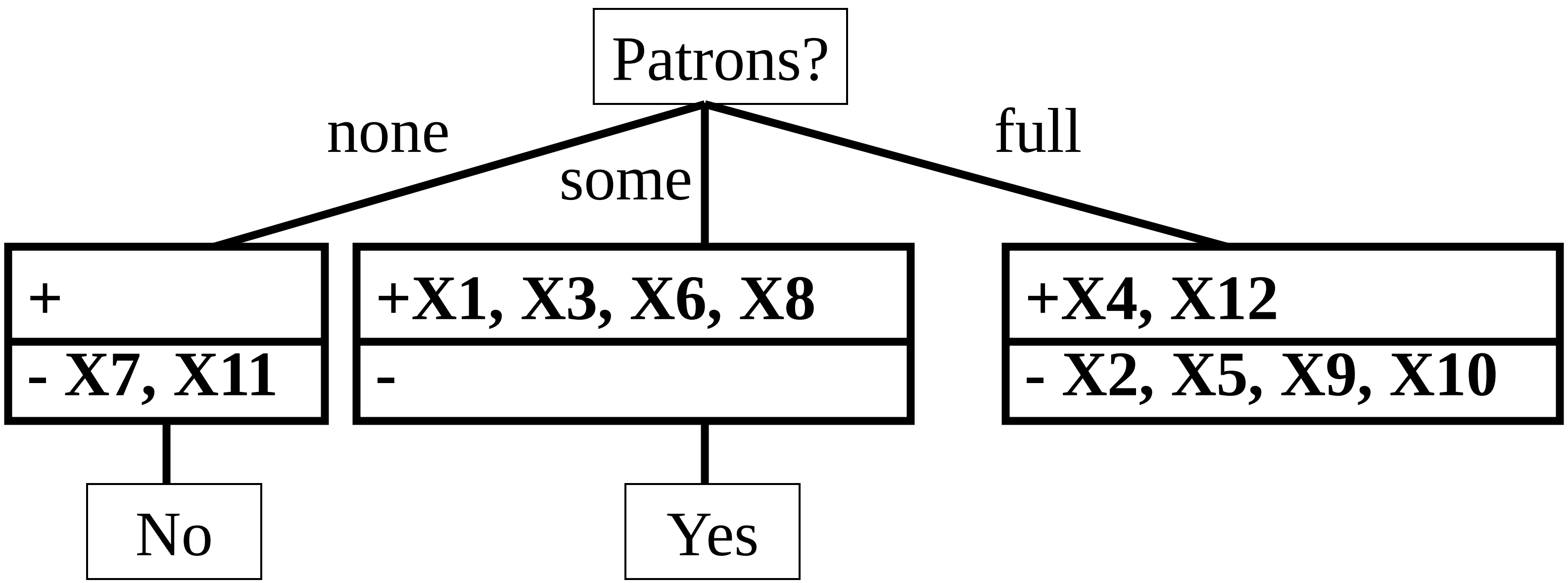
+ X1, X3, X4, X6, X8, X12 (Positive examples)
- X2, X5, X7, X9, X10, X11 (Negative examples)

Splitting Examples by Testing on Attributes (con't)

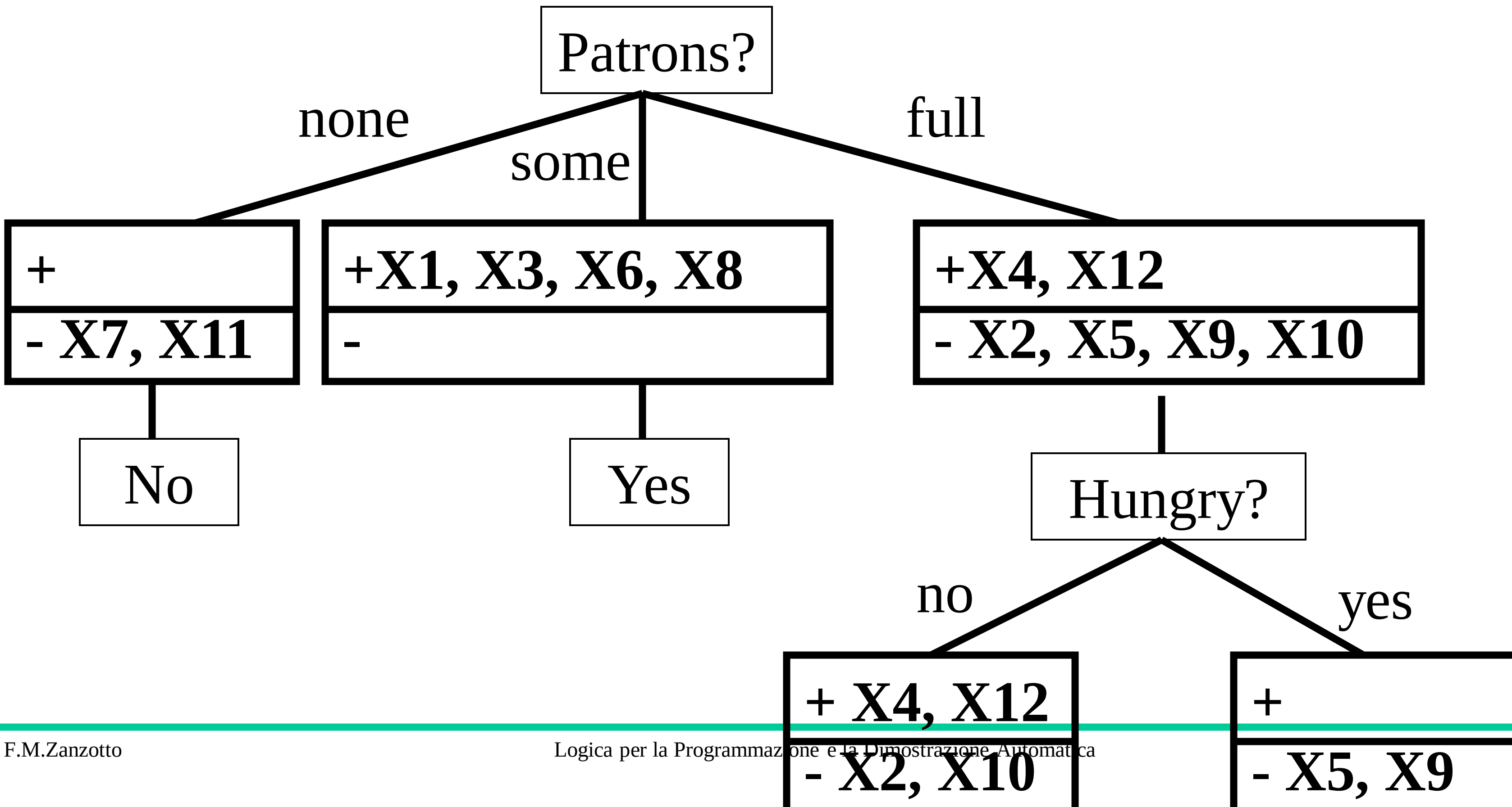
+ X1, X3, X4, X6, X8, X12 (Positive examples)
- X2, X5, X7, X9, X10, X11 (Negative examples)



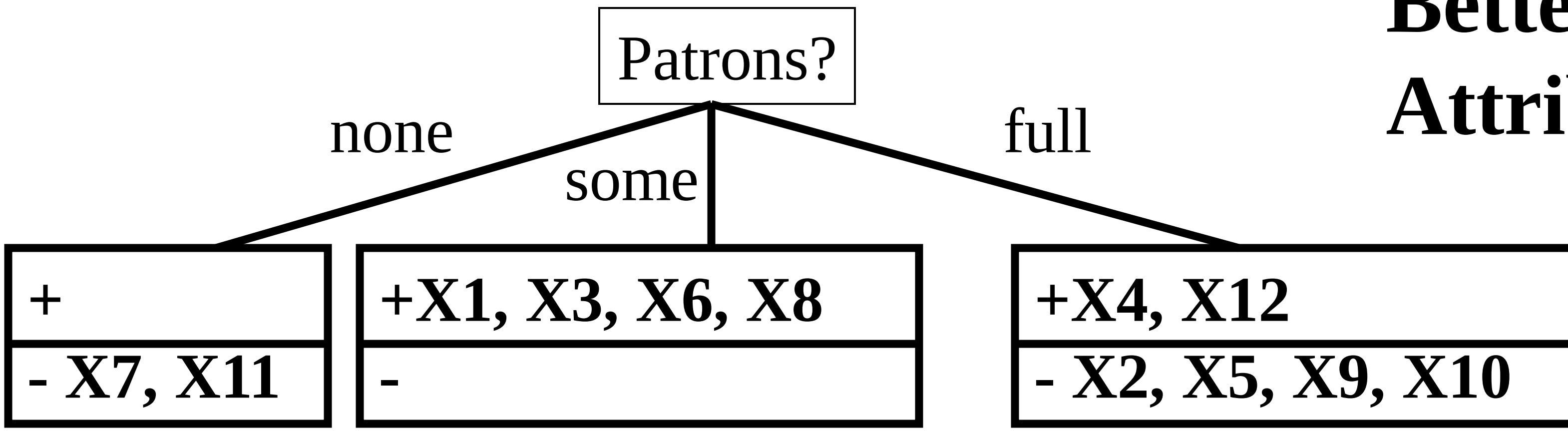
Splitting Examples by Testing on Attributes (con't)



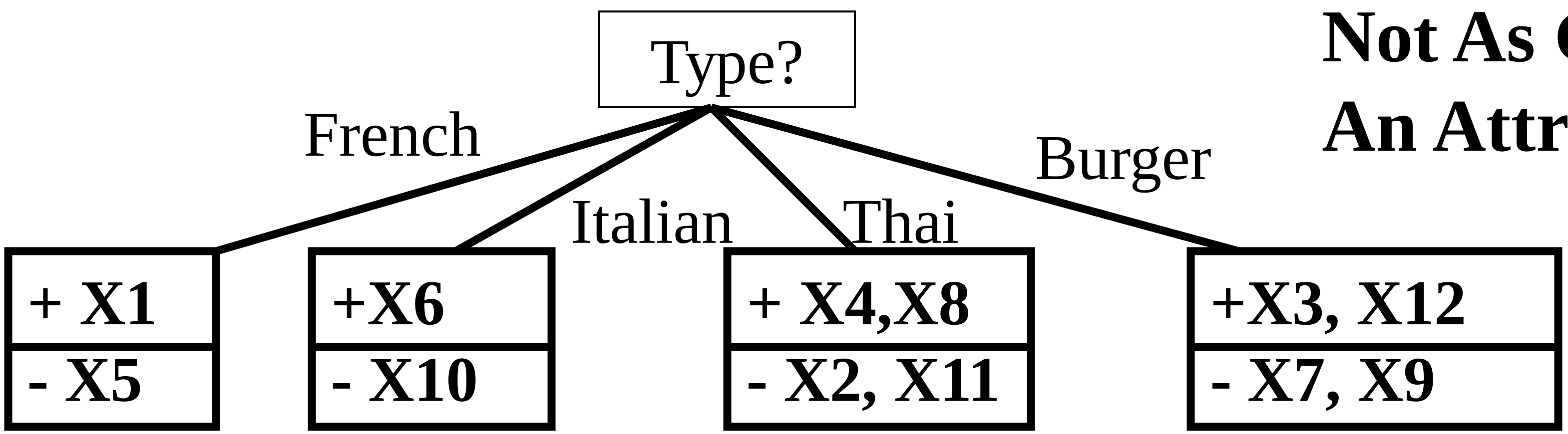
Splitting Examples by Testing on Attributes (con't)



What Makes a Good Attribute?



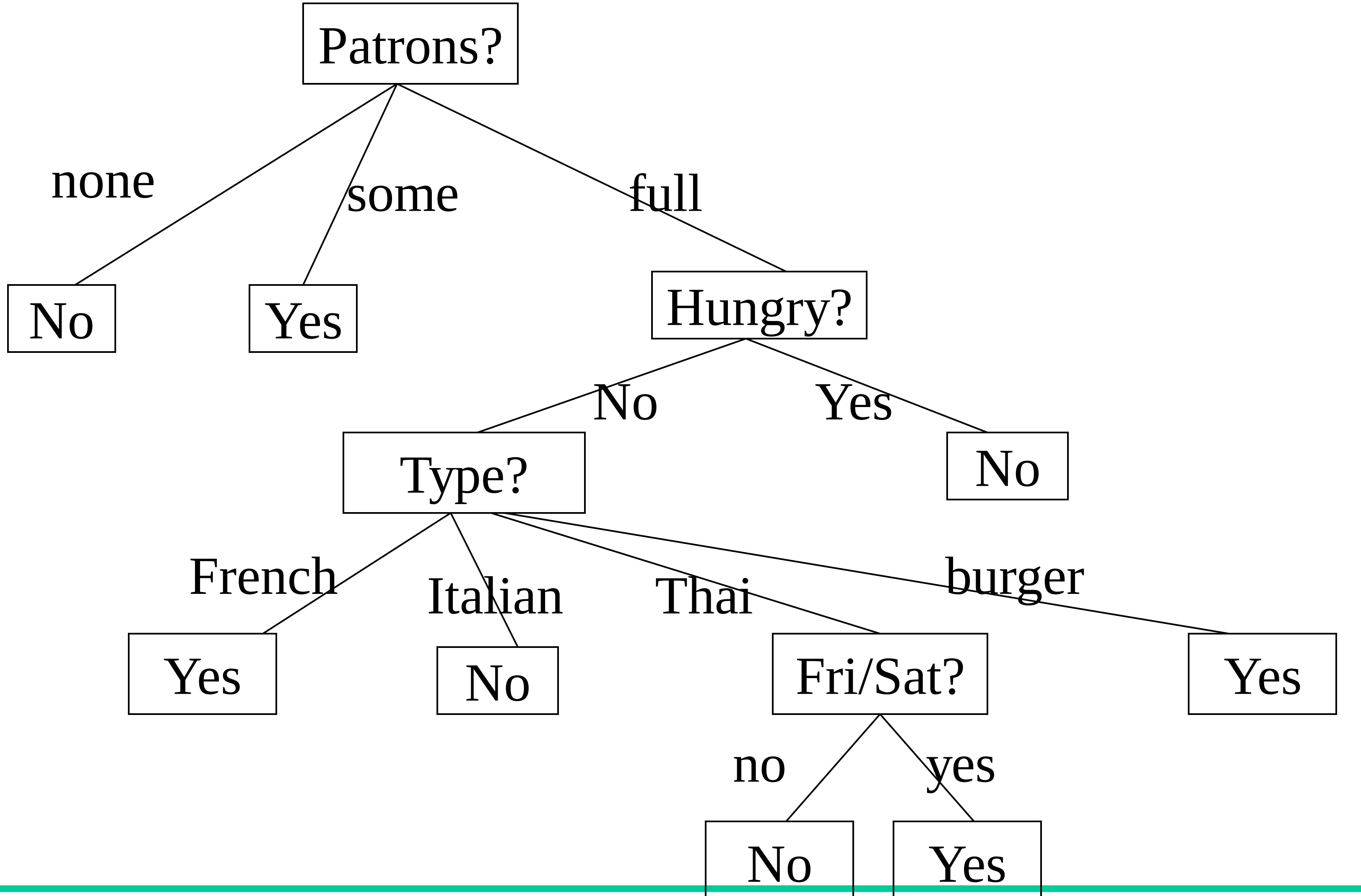
Better Attribute



Not As Good An Attribute

http://www.cs.washington.edu/education/courses/473/99wi/

Final Decision Tree



http://www.cs.washington.edu/education/courses/473/99wi/

Decision Tree Learning: ID3 algorithm

function ID3 (**R:** attributes, **S:** a training set, *default_value*) returns a **decision tree**;

begin

if S=∅, **then return** *default_value*;

else if S consists of records all with the value *v* **then return** value *v*;

else f R=∅, **then return** the most frequent value *v* for records of S;

else

let *A* be the attribute with largest Gain(*A*,*S*) among attributes in R;

let $\{a_j |\ j=1,2, .., m\}$ be the values of attribute *A*;

let $\{S_j |\ j=1,2, .., m\}$ be the subsets of *S* consisting respectively of records with value a_j for A;

return a tree with root labeled *A* and arcs labeled $a_1, a_2, .., a_m$ going respectively to the trees (ID3(R-{A}, S_1), ID3(R-{A}, S_2), ...,ID3(R-{A}, S_m);

end

Attribute selection function: Gain(*A*,*S*)

- Information Theory
  - each set of messages has its intrinsic information related to the probability of a given message in the text
- It is demonstrated that:
  - M is the set of messages
  - p is a probability distribution of the messages
  - the information carried by the set of messages is:

$$I(M) = \sum_{m \in M} p(m) log_2(\frac{1}{p(m)})$$

Attribute selection function: Gain(*A*,*S*)

- Given:
  - a set of target classes **T**
  - *A*: an attribute with values *a* in **A**
  - *S*: a set of training instances
- Using the notion of information I(M) it is possible to define:

$$I_S(T) = \sum_{t \in T} p_S(t) log_2(\frac{1}{p_S(t)})$$

$$Remaining(A) = \sum_{a \in A} p_S(a) I_{S_{A=a}}(T)$$

$$Gain(A,S)) = I_S(T) - Remaining(A)$$

Example: Gain(A,S) with a binary classification

Abusing the notation and using the maximun likelihood probability estimation model:

$$I(\frac{p}{p+n},\frac{n}{p+n}) = -\frac{p}{p+n}log_2\frac{p}{p+n} - \frac{n}{p+n}log_2\frac{n}{p+n}$$

$$Remaining(A) = \sum_{a\in A} -\frac{p_a+n_a}{p+n} I(\frac{p_a}{p_a+n_a},\frac{n_a}{p_a+n_a})$$

where:

- p is the # of positive examples in S
- n is the # of negative examples in S
- p_a is the # of positive examples in S fixing the value of A as a
- n_a is the # of negative examples in S fixing the value of A as a

Prolog Coding

```
attribute(fri,[yes,no]).
attribute(hun,[yes,no]).
attribute(pat,[none,some,full]).
```

```
example(yes, [fri = no, hun = yes, pat = some, …]).
```

```
induce_tree( Tree)  :-
	findall( example( Class, Obj), example( Class, Obj), Examples),
	findall( Att, attribute( Att, _ ), Attributes),
	induce_tree( Attributes, Examples, Tree).
```

```
induce_tree( Attributes, Examples, tree( Attribute, SubTrees))  :-
 choose_attribute( Attributes, Examples, Attribute), !,
 del( Attribute, Attributes, RestAtts),                 % Delete Attribute
 attribute( Attribute, Values),
 induce_trees( Attribute, Values, RestAtts, Examples, SubTrees).

induce_tree( _, Examples, leaf( ExClasses))  :-     % No (useful) attribute,
leaf with class distr.
 findall( Class, member( example( Class, _), Examples), ExClasses).
```

```
induce_tree( _, [], null)  :-  !.

induce_tree( _, [example( Class,_ ) | Examples], leaf( Class))  :-
 not ( member( example( ClassX, _), Examples),          % No other
example
                 ClassX \== Class), !.                          % of different class
```