Generalization Process

Symbolic learning

❒ Generalization is a main concept

➢ Dictionary definition

- A general statement or concept obtained by inference from specific cases

➢ Main objectives in ML

- To learn the common properties of a set of examples belonging/expressing the same target concept
- We are mainly interested here by the *symbolic representation* (Lh = logic) Inductive Logic Programming domain (ILP)

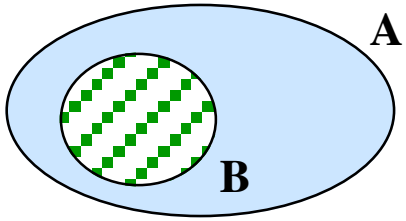
❒ Example: learning of the “sunrise” concept

Generality relation

❒ **What's the meaning of « to be more general » ?**

➢ "Set theory" definition

• We say that definition A is more general than definition B if the set of all the instances covered by B are included in the set of the instances of A.



"Mammalian" concept is more general than "Elephant" concept

➢ Logic definition
(Notice that "is more general than" ⇔ "subsume")

• Let F and G two conjunctive formula:

G « θ-subsume » F is and only if $\exists\theta$, $\theta G \subseteq F$ (Plotkin 70)

❒ **An example:**

G :- animal (X) $\wedge$ have (X, trunk)

F :- animal (jumbo) $\wedge$ ear (jumbo, large) $\wedge$ have (jumbo, trunk)

G **θ-subsume** F indeed for the substitution θ={X/ jumbo} within G:

θG = animal (jumbo) $\wedge$ have (jumbo, trunk) $\subseteq F$

Generalization problem

(**Plotkin 70**, Michalski 83, Kodratoff et al. 86, Muggleton et al. 92, Rouveirol 92, ...)

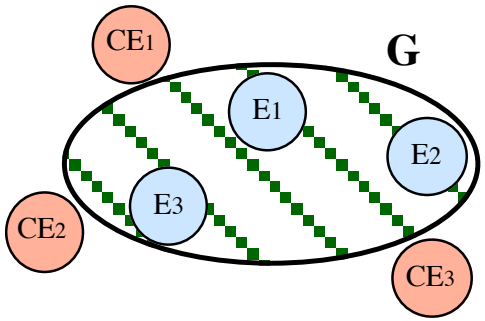
❒ **Let …**

- A set of instances (or facts) in X containing
  - Positive examples E_1, ... , E_n of a concept C to learn
  - Negative examples (counter-example) CE_1, ... , CE_m of this concept
- A « Domain theory / Background knowledge» T (i.e. a set of inference rules)
- A set of bias B (or constraints) to prune the search space $\mathcal{H}$
- A function M (*i.e. unification*) to map F on an instance I, such that : $M(F, I) \rightarrow \theta$

➔ Generalization is the search of a formula G verifying constraints B such that:

❶ G covers all the E_i : $\exists\theta,\ \theta(G) \subseteq T \wedge E_i$ (G is said **complete**)

❷ G covers no CE_j : $\forall\theta,\ \theta(G) \not\subset T \wedge CE_j$ (G is said **consistent**)

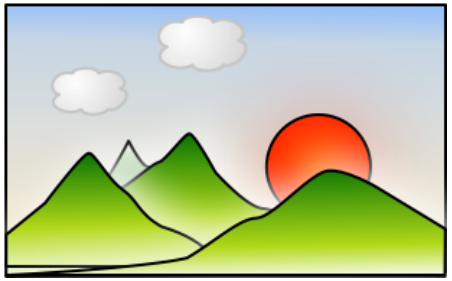


- Compact the knowledge
  (we go from N examples to 1 description)
- Rule : If G Then C
- Clause : C :- G.

Generalisation of a set of examples

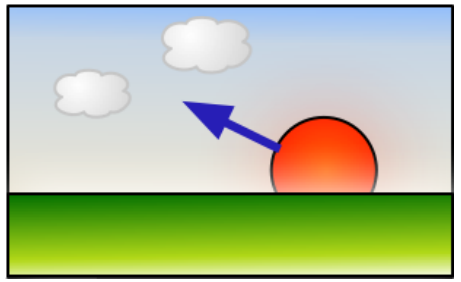
Learning of the « sunrise» concept

- We use here a representation language base on the « predicative logic»
- We could imagine the predicates can be provided by a picture analysis tool



E1 :- mountain (a), sun (b),
go_up (b,a), cloud (c),
above (c,a)

E2 :- town (c), sun (d),
go_up (d,c), cloud (e),
above (e,c)



G :- sun (X),
go_up (X,Y),
cloud (Z),
above (Z,Y).

Substitution list

Variable	θ_1 for E1	θ_2 for E2
X /	b	d
Y /	a	c
Z /	c	e

How to compute the generalization ?

❒ The complexity of this task depends on Lh

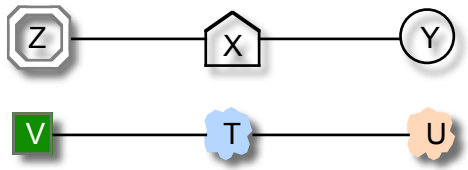
- Trivial when Lh is using propositional (conjunctive) representation

Ex1 : temperature (40°)	∧ lymphocyte (5000)	∧ no_background ∧ ...
Ex2 : temperature (39°)	∧	∧ no_background ∧ ...
Ex3 : temperature (39.2°)	∧	∧ no_background ∧ ...
Gen: temperature *[39°,40°]*	∧	∧ no_background ∧ *...*

- Far more complex with the predicative logic

Several substitution … with a “one to one” mapping respecting the topology:

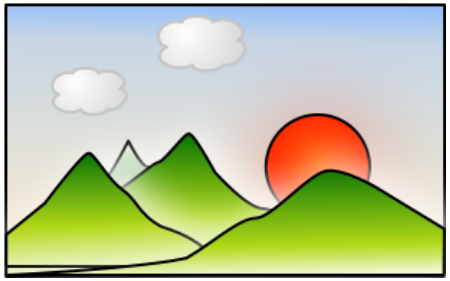
- Choice 1: (X/A:E, Y/B:F, Z/D:G) →
- Choice 2 : (T/A:F, U/B:E, V/C:H) →



The search space with N examples having V variables is V^N

A (naive) heuristic approach

❒ **For a "one to one" mapping**



E1 :- mountain (a), sun (b),
go_up (b,a), cloud (c), above (c,a)

E2 :- town (c), sun (d),
go_up (d,c), cloud (e), above (e,c)

1) To create the "signature" of the objects (constants)

- E1 : a={(mountain0), (go_up 1), (above 1)}, b={(sun 0), (go_up 0)}, c={(cloud 0), (above 0)}
- E2 : c={(town 0), (go-up 1), (above 1)}, d={(sun 0), (go_up 0)}, e={(cloud 0), (above 0)}

2) Mapping between the *most similar objects* and creation of the variables

d	b	$\rightarrow$ X
c	a	$\rightarrow$ Y
e	c	$\rightarrow$ Z

3) Substitution of the constants by the variables in the examples

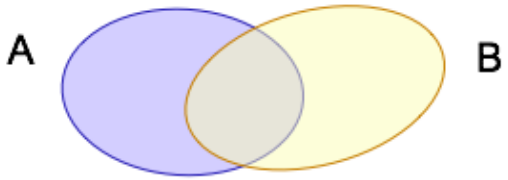
4) Computing G just corresponds to evaluate E1 $\cap$ E2

What's the meaning of "most similar"

- **"Contrast model"** (Tversky 77)
  - When we have two representations defined by a set of properties

$$SIM(x,y) = \frac{f(p_x \cap p_y)}{f(p_x \cup p_y) + \alpha f(p_x - p_y) + \beta f(p_y - p_x)}$$

with:
- p_x = properties of x
- p_y = properties of y



  - If we want a symmetrical measure ($\alpha = \beta = 0$)

- **Application to the predicative logic**
  - Signature (constants) = list of occurrences: (… (prédicat$_i$, position ...) …)
  - Then we search for the *maximum weighted mapping in a bipartite graph*

    (A very classical allocation problem in computer science)
  - Very simple approach (most complete exist)
    - Doesn't take into account the notion of value
    - Doesn't take into account the notion of relation between constants

Complete approach : Least General Generalisation

❒ Definition of LGG (plotkin 70)

G is a LGG of formulas A and B if

- G is a generalization of A and B
- All the other generalization of A and B are a generalization of G

❒ How to compute the LGG

- For two terms u and v

If u=v then LGG(u, v) = u

If u = $f(s_1, \ldots, s_i)$ and v = $f(t_1, \ldots, t_i)$

Then LGG(u, v) = $f(LGG(s_1, t_1), \ldots, LGG(s_i, t_i)$

Else LGG(u, v) is a variable X *(the naming process must be consistent)*

- For two litterals K and L

If the predicates K and L have the same sign, symbol and arity

Then lgg ($p(u_1, \ldots, u_n)$ and $p(v_1, \ldots, v_n)$) = $p(lgg(u_1,v_1), \ldots, lgg(u_n,v_n))$.

Else lgg(K, L) = Ø

- For two clauses C1 et C2

1) To compute LGG (T1, T2) between the heads of clauses to build the new head

2) To compute LGG by doing the Cartesian Product between the bodies B1 of C1 and B2 of C2

LGG(B1, B2) = LGG ($B1_1$, $B2_1$), LGG ($B1_1$, $B2_2$), …, LGG ($B1_p$, $B2_q$)

LGG (2)

❒ Example of LGG

Let the two clauses

```
q(g(a)) :-  p(g(a), b), r(h(b), c), r(b, d).
q(x)    :-  p(x, y), r(y, z), r(h(y), z), s(g(x), h(y)).
```

Here is the LGG :

```
q(X) :- p(X, Y), r(Z, U), r(V, W), r(h(Y), U), r(Y, W).
```

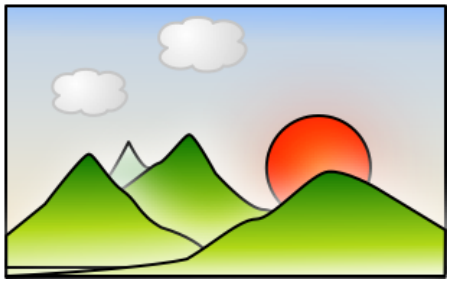
with the substitutions

```
{X/{g(a),x},Y/{b,y},Z/{h(b),y},U/{c,z},V/{b,h(y)},W/{d,z}}
```

❒ Some remarks about LGG

- In ILP, we don't take into account the functional terms (flattening)
- LGG result is independent of the order (LGG is transitive)
- LGG becomes huge is some predicates have multiple occurrences (i.e. 2^N)
- LGG contains (many) redundant literals *(in red in the previous example)* !!!
  - Reduction algorithms in $O(n^4)$
  - When we deal with "real problem" one-one mapping are more realistic !

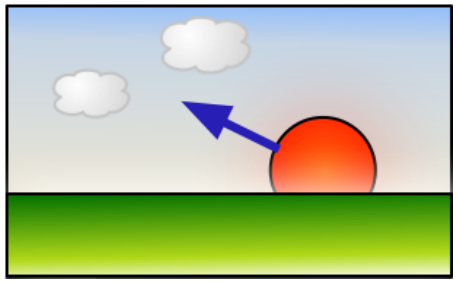
Generalisation using counter examples



E1 :- mountain (a), sun (b),
go_up (b,a), cloud (c), above (c,a)

E2 :- town (c), sun (d),
go_up (d,c), cloud (e), above (e,c)

CE :- town (f), mountain (g), sun (i)
go_up (h,f), cloud (h), above (h,f)



G :- sun (X), go_up (X,Y),
cloud (Z), above (Z,Y)

Matching between G and CE

Using the mapping function M

❶ sun (X) ⇒ X / i

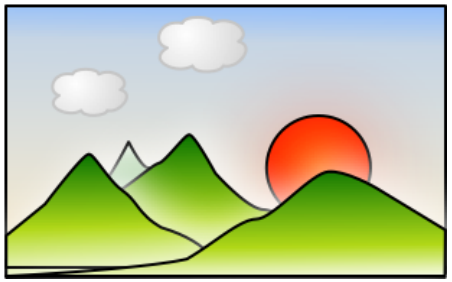
❷ go_up (i, Y) ⇒ Failure

G' :- sun (X), go_up (X,Y),
~~cloud (Z), above (Z,Y)~~

☞ G' is complete et consistent

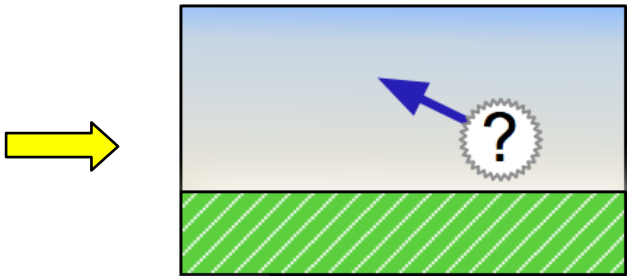
☞ G' is minimal

Interest of the Background Knowldge



E1 :- star (b), type (b, « G2 »),
mountain (a), go_up (b,a)

E2 :- sun (c),
town (d), go_up (c,d)



G :- go_up (X,Y) !!!

❒ **Let the following background knowledge**

- R1: IF montain (X) THEN skyline(X)
- R2: IF town (X) THEN skyline(X)
- R3: IF star (X), type (X, « G2 ») THEN sun (X)

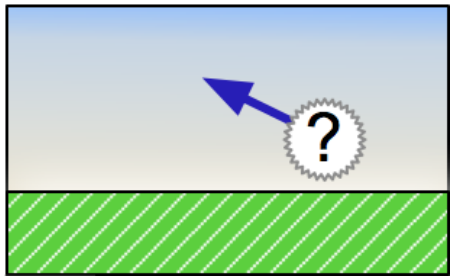
We can deduce new fact by forward chaining:

➢ Using R1 et R3 with E1

E1:- sun (b), skyline (a), go_up (b,a)

➢ Using R2 with E2

E2:- sun (c), skyline (d), go_up (c,d)

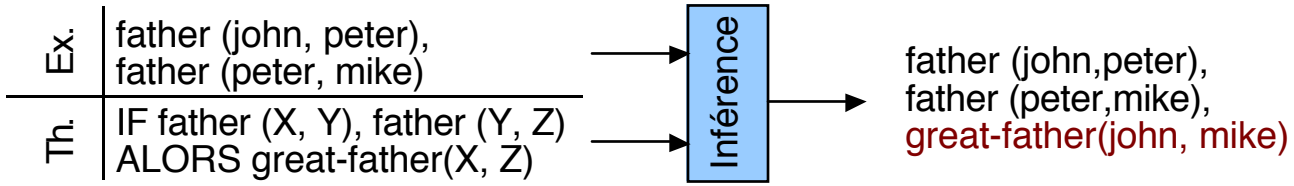


G' :- sun (X), go_up (X,Y), skyline (Y)

The background knowledge problem

❒ When and how to use it ?

- ➢ When it is needed, but ...
  - Syntactical criteria are too weak
  - We need some meta-knowledge (wel known problem) ...
- ➢ So in practice : saturation process (RLGG) …



❒ Advantages

- ➢ Simple forward chaining
- ➢ All the facts can be used by the learning system
- ➢ Strategy independent of the learning algorithms

❒ Drawbacks

- ➢ What to do with a large BK ?
- ➢ We increase the number of the attributes (not so good)
- ➢ In practice the problem is open and no more studied !

Several way to generalize

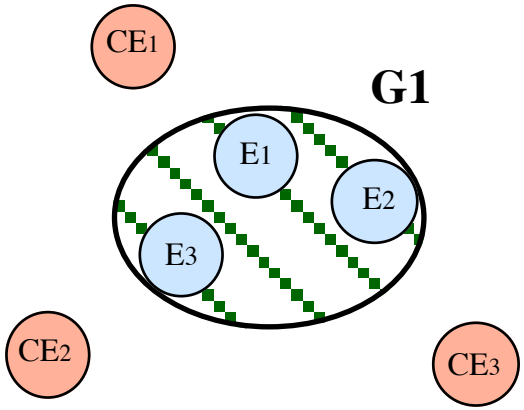
E1 :- mountain (a), sun (b), go_up (b,a), cloud (c), above (c,a)
E2 :- town (c), sun (d), go_up (d,c), cloud (e), above (e,c)
CE :- town (f), mountain (g), sun (i)go_up (h,f), cloud (h), above (h,f)

❒ Least General Generalization (LGG)

G = sun (X), go_up (X,Y), cloud (Z), above (Z,Y)

- ➢ We keep all the shared knowledge
- ➢ Simple to compute, at least when
  - We have conjunctive formulas
  - The hypothesis language is not too complex

☛ Huge size : bad readability and performances

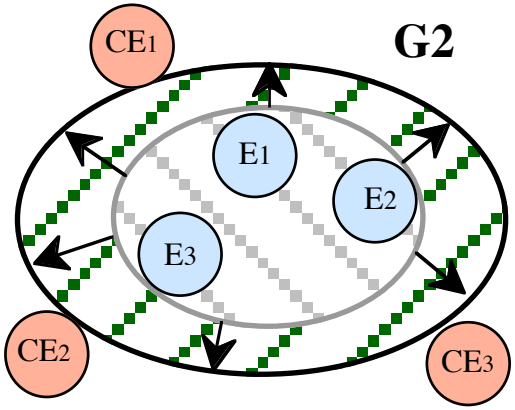


❒ Most General Generalization (MGG)

G' = sun (X), go_up (X,Y)

- ➢ We just keep the useful knowledge
- ➢ Operational definition of the concept

☛ Many possibilities : how to explore ?



Two classical "Generate and Test" strategies

❶ **Iterative specialization** (Foil-like algorithm)

- Let generalization G maximally specific (LGG)
- Let F = ∅

While F is not consistent

- To select an atom A_i of G not yet used (multiple choice !!)
- To add A_i to the formula F
- To test the consistency of F with the counter-examples

❷ **Iterative generalization** (Golem-like algorithm)

- Let generalization G maximally specific (LGG)
- Let F = G

While F is consistent

- To selection an atom A_i of F (multiple choice !!)
- To suppress A_i from the formula F
- To test the consistency of F with the counter-examples

Is it possible to have a better, non heuristical, approach ? Yes

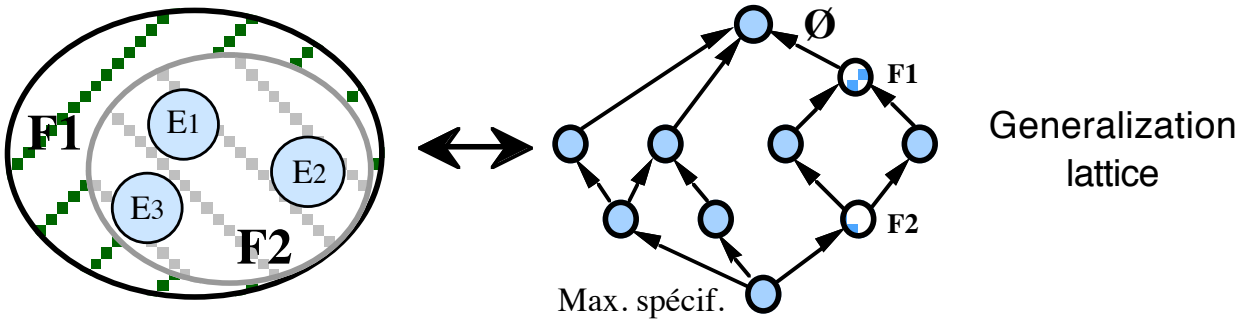
Generalization lattice

❒ Generality relation induces a partial order of the elements of $L_{\mathcal{H}}$

- If *F1 "subsume" F2*

  Then

  F2 ⇒ F1

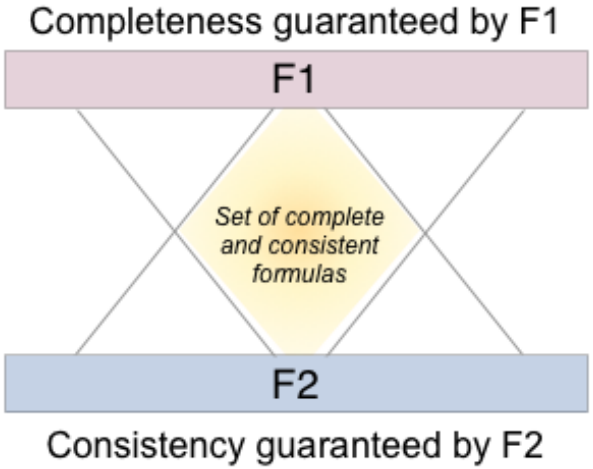


Generalization lattice

❒ Some interesting properties when F2 ⇒ F1

- If F1 is **complete** all the more general formula are complete too
- If F1 is **consistent** then F2 is consistent too

- If F2 is **consistent** all the more specific formula are consistent too
- If F2 is **complete** then F1 is complete too

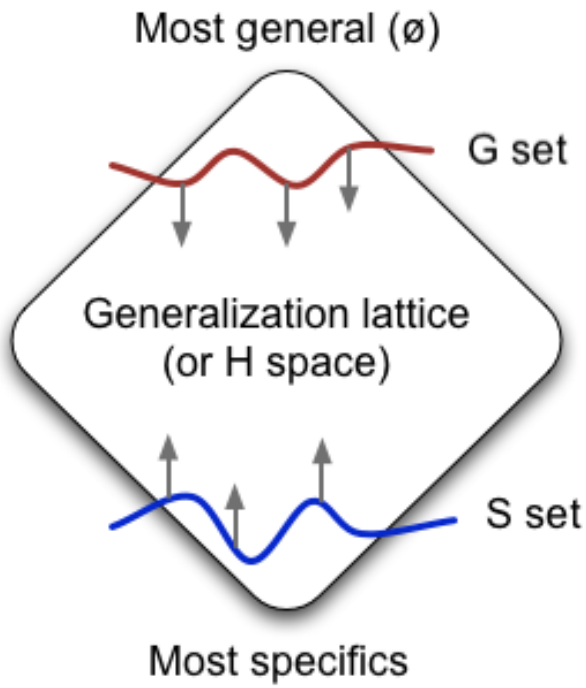


Version Space Algorithm

(Mitchell 82)

❒ **All the possible generalizations can be described with just two frontiers**

- G : Set of the most *general generalizations* that are complete and consistent according to the already seen instances
- S : Set of the most *specific generalizations* that are complete and consistent according to the already seen instances



An incremental process (on-line learning)

Examples and counter-examples arrive one by one in the system

- When Examples:
  → we generalize formulas of S if needed.
- When Counter-examples:
  → we specialize formulas of G if needed.

Generalization result doesn't depend on the presentation ordering!

General algorithm

While there are some instances « I » to generalize, do:

IF I is an example

❶ **Generalization** of all the models of S rejecting I
- We must do the minimal generalization (smallest generalization step)
- Each model must be a specialization of a model of G
- No model of S is a generalization of another one (most specific models)

❷ To suppress from G the models not mapping with I

IF I is a counter-example

❶ **Specialization** of all the models of G covering I
- We must do the minimal specializations (smallest specialization step)
- Each model must be a generalization of a model of S
- No model of G is a specialization of another one (most general models)

❷ To suppress from S the models mapping with I

IF S and G ≠ Ø

If S = G Then we get a complete and consistent solution

Else All the solution between S and G are possible solutions

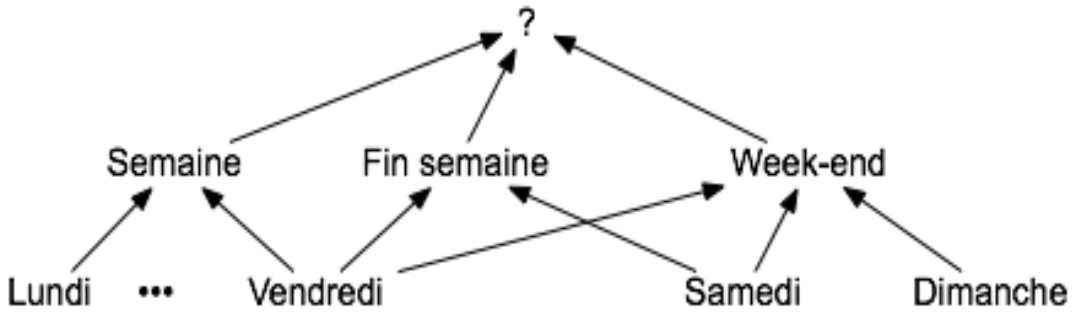
Else Failure: there is an incoherence in the training set → impossible to learn with $L_{\mathcal{H}}$

Example EV (1)

❒ **We search for the conditions of a “food poisoning”**

	Restaurant	Repas	Jour	Coût	Réaction
1	Pierrot	déjeuner	vendredi	pas cher	oui
2	A la frite	diner	vendredi	normal	non
3	Pierrot	diner	samedi	pas cher	oui
4	Quick	déjeuner	dimanche	pas cher	non
5	Pierrot	déjeuner	dimanche	normal	non
6	Pierrot	diner	jeudi	pas cher	oui

➢ We have Background Knowledge T providing a generalization on the day



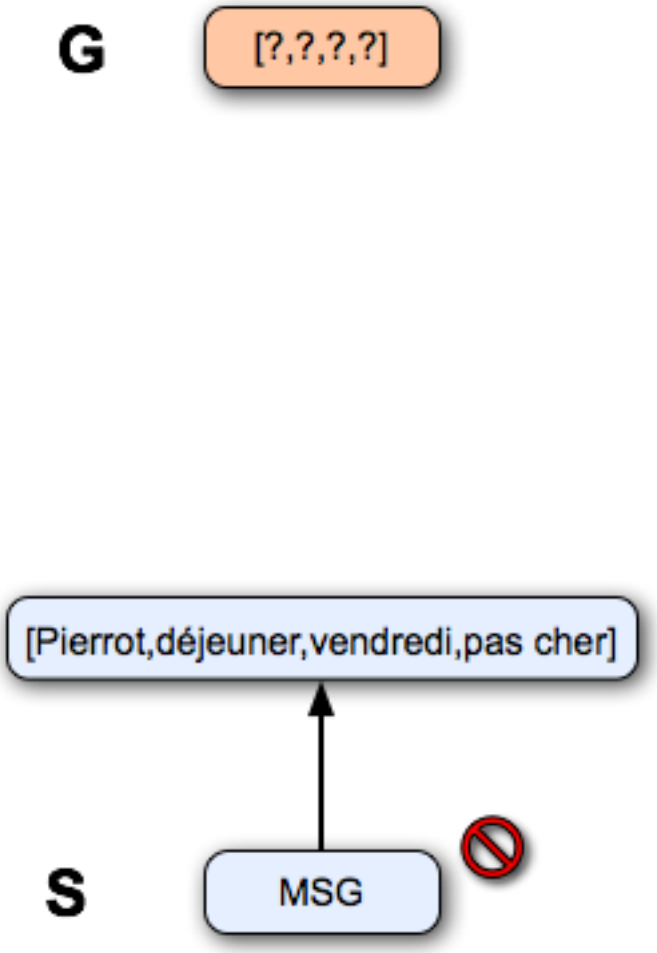
➢ Initialization of the set G et S :

- G = Ø we will write [?, ?, ?, ?]
- S = MSG (Most Specific Generalization)

Example EV (2)

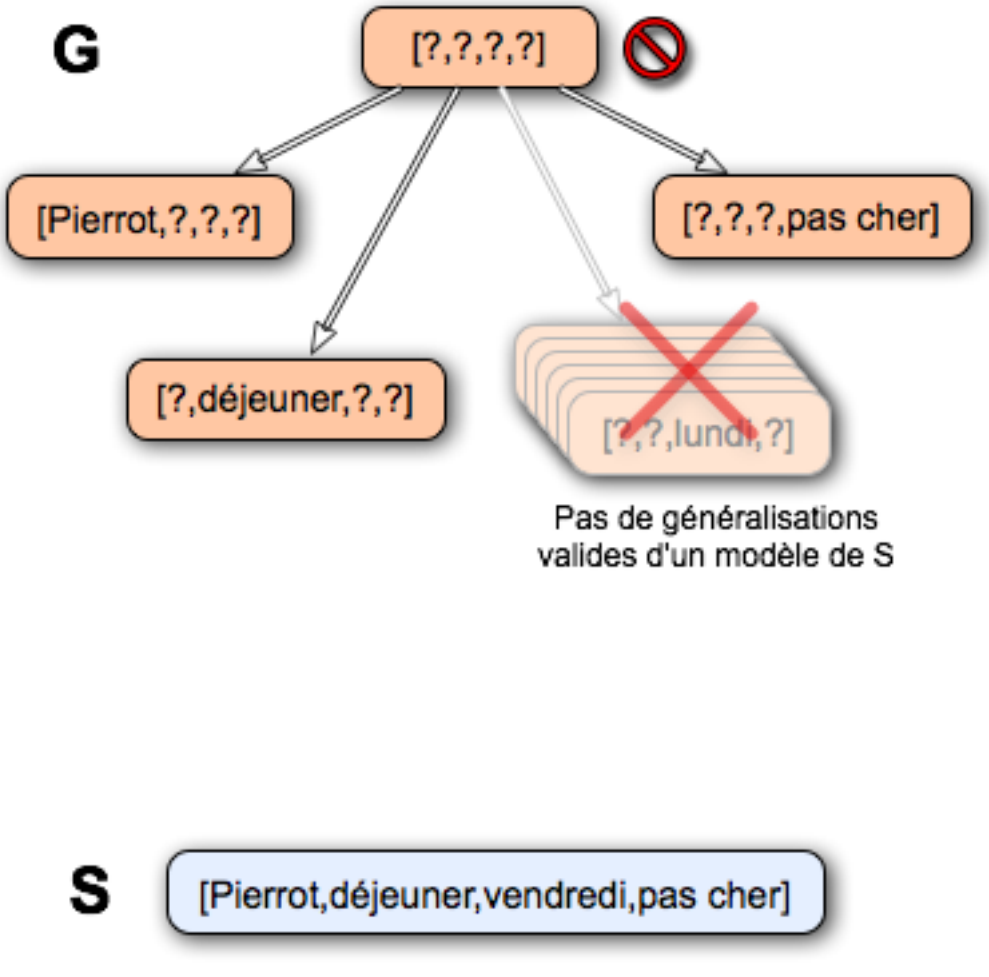
❶ Adds the first example :

[Pierrot, déjeuner, vendredi, pas cher]



❷ Adds the first counter-example :

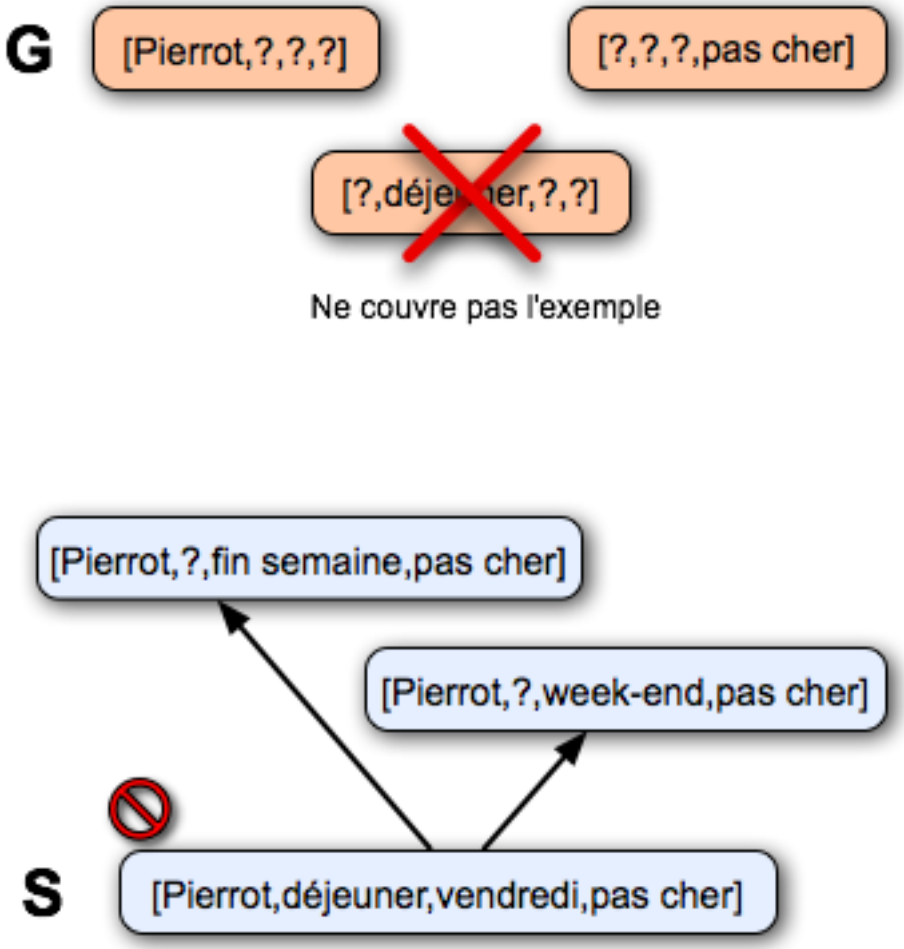
[A la frite, diner, vendredi, normal]



Example EV (3)

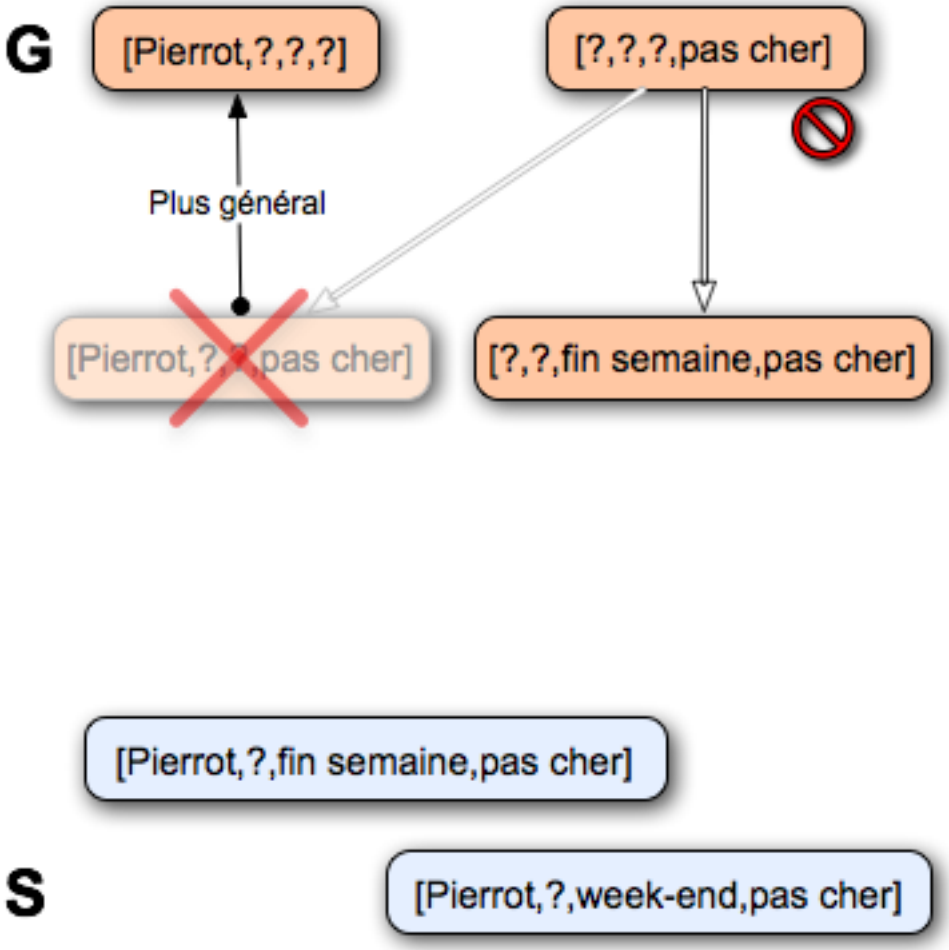
❸ Next example :

[Pierrot, diner, samedi, pas cher]



❹ Next counter-example :

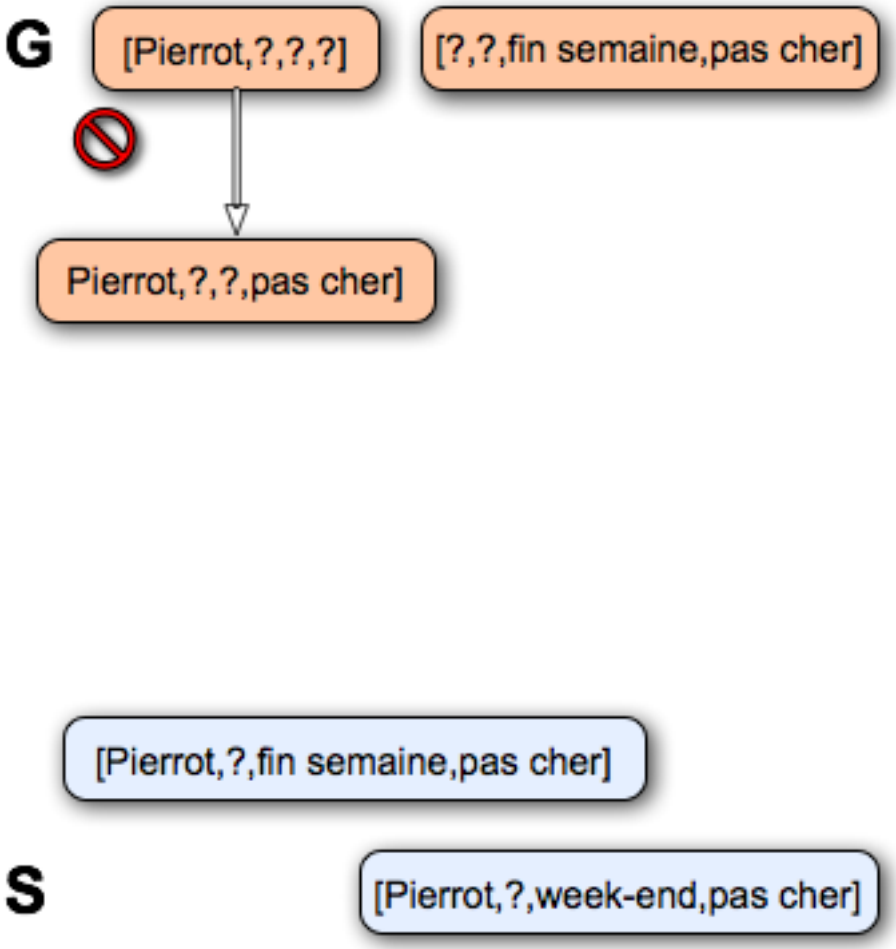
[Quick, déjeuner, dimanche, pas cher]



Example EV (4)

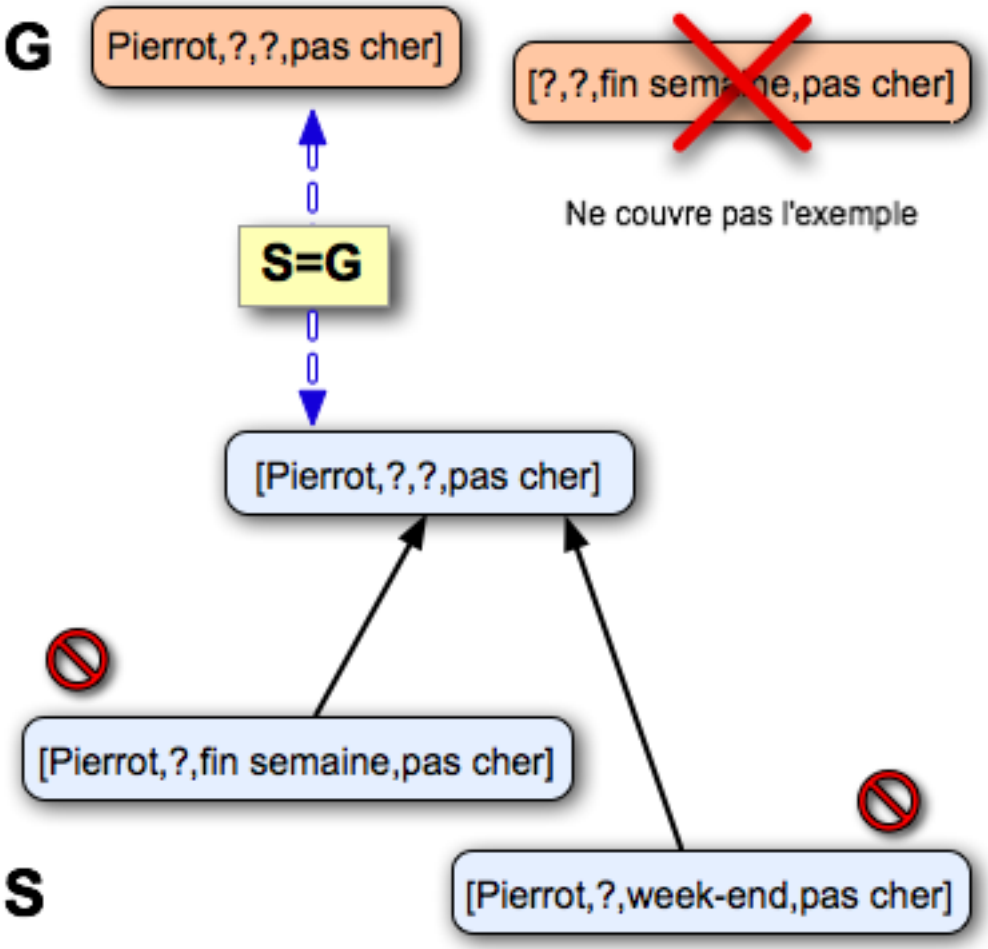
❺ Next counter-example :

[Pierrot, déjeuner, dimanche, normal]



❻ Last example :

[Pierrot, diner, jeudi, pas cher]



Version space in practice

- **Theoritical advantages**
  - Efficient way to perform a complete search in the generalization lattice
  - Incremental approach independent of the examples ordering
- **On a real data set**
  - Only rather limited representation language can be used
    - Boolean representation
      - But in some (rare) pathological cases G set can increase exponentially with the number of attributes
    - Very difficult to learn a disjunctive concept
    - Impossible to use predicative logic
      - To many ways to generalize/specialize the formulas
  - Impossible to deal with noisy data (error on a value or about the class)
    - Erroneous modifications of G and S cannot be canceled
  - Limited applicability to very well-defined problems

Supervised learning algorithms

TDIDT approaches : ID3, C4.5, … (Quinlan 86-96)

(Top Down Induction of Decision Tree)

❒ Principles

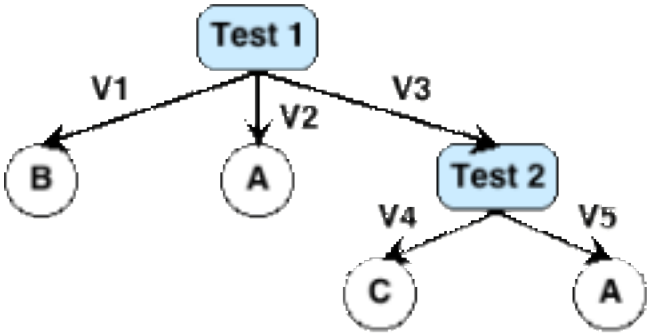
- ➢ To build a tree allowing to guess the class of an example
- ➢ Old approach but still active
- ➢ Very good ratio quality/complexity (ID3, CART, ID5, Assistant 86, C4.5, C5.0, OC1, ...)

❒ Input

- ➢ A set of training examples with 2 classes or more …
- ➢ Representation language is propositional logic

❒ Output

- ➢ Decision tree (For instance : flora)
  - Nodes : Attributes to test
  - Edges : Values of the attributes
  - Leaves : Classes to guess



- ➢ Allows to predict the class of a new example

How to build the tree ?

❒ What are the attributes to select ?

➢ Goal : to optimize the compactness of the tree in order …

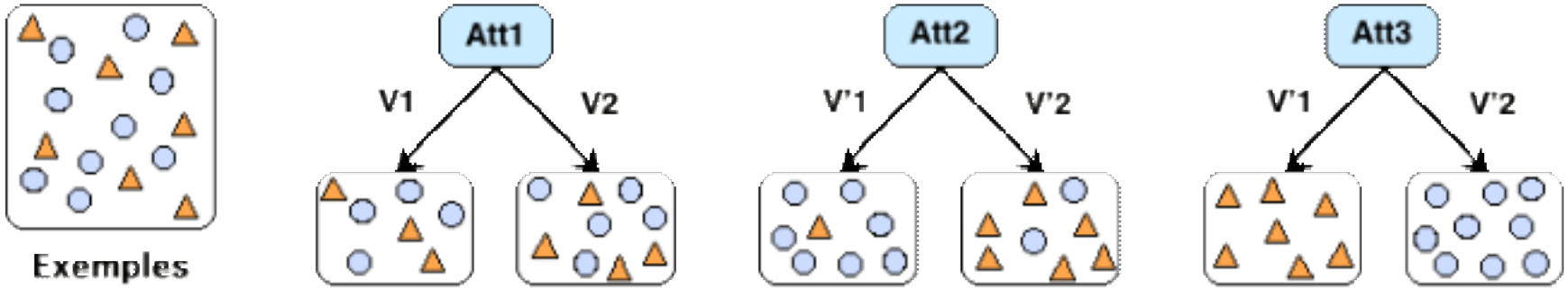
- to improve readability (semantical evaluation)
- to decrease the number of questions to ask
- to have a most general tree (cover more cases)

☞ Searching for the best tree is a NP complete problem

#tree (level, #values, #attributes) = $V^P A_D^P$ → #tree (5, 2, 50) = 8 10^9 trees

❒ Selecting the most discriminative attributes ...

➢ What is an interesting attribute ?



☞ How to measure the disorder level of a set ?

Shannon's Entropy (Shannon 48)

❒ Measure of the average information content

- ➢ It's a measure of the uncertainty associated with a random variable E
  - Strong entropy : E = {3, 1, 4, 1, 5, 9, 2, 6, 3, ...}
  - Small entropy : E = {2, 2, 2, 2, 2, 2, 2, 2, ...}

$$Entropie(E) = \sum_{i=1}^{n} -P_i \times Log_2(P_i)$$

P_i : propability that an item of E belongs to the category C_i. We approximate this value with the frequency of C_i in the training set.

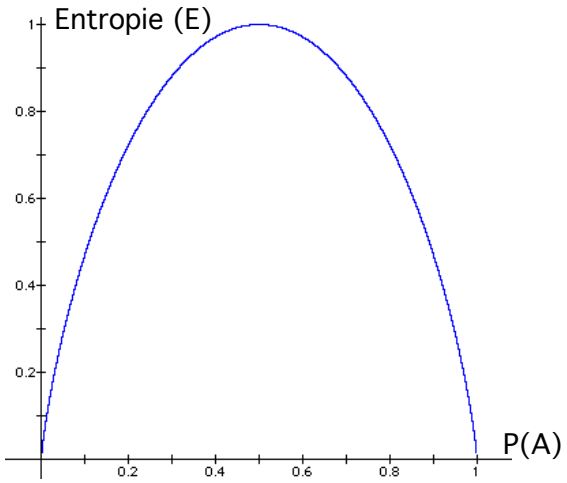
- ➢ Entropy measure the disorder level
- ➢ In practice: number of bits allowing to code efficiently E

❒ in our problem ...

- ➢ A node in the tree ↔ Set of E examples (i.e. root)
- ➢ Let a set E containing to classes A and B

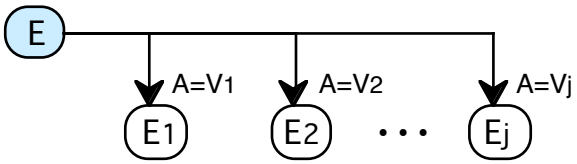
$$Entropy\ (E) = - P_A\ Log_2\ P_A - P_B\ Log_2\ P_B$$

- E1 : P(A) = 1 & P(B) = 0 ⇒ Entropy (E1) = 0
- E2 : P(A) = 0,90 & P(B) = 0,10 ⇒ Entropy (E2) = 0,47
- E3 : P(A) = 0,75 & P(B) = 0,25 ⇒ Entropy (E3) = 0,81
- E4 : P(A) = 0,50 & P(B) = 0,50 ⇒ Entropy (E4) = 1



Entropy minimization

- **The goal is to minimize the desorder of E**
  - When we select the attribute A with J possible values
    - We split E into J sub-set E_j



  - Average entropy EM associated to the use of A on the set E :

$$EM(E,A) = \sum_{j=1}^{V} P(V_j) \times Entropy(E_j)$$

$P(V_j)$: probability to observe the value V_j in the domain. We approximate this value with the frequency of V_i in the training set.

  - Gain of entropy associated to the use of A:

$$Gain(E,A) = Entropy(E) - EM(E,A)$$

    - The idea is to select the attribute having the best gain

Algorithm

❒ **A typical gradient search approach**

➢ Initialization: the root contains all the training examples.

- **While** *Some leaves* L_k *in the tree contain a mixure of several classes* **Do**

  Foreach L_k

  ❶ Foreach attribute A_i not already used in the current path

  - Measure the entropy gain associated to the use of A_i

  ❷ Select the attribute $\mathcal{A}$ maximizing the gain.

  ❸ Add to the node L_k as many children that $\mathcal{A}$ has different values

  ❹ Allocate the examples contain in L_k to its different children according to their value for $\mathcal{A}$

Example (1)

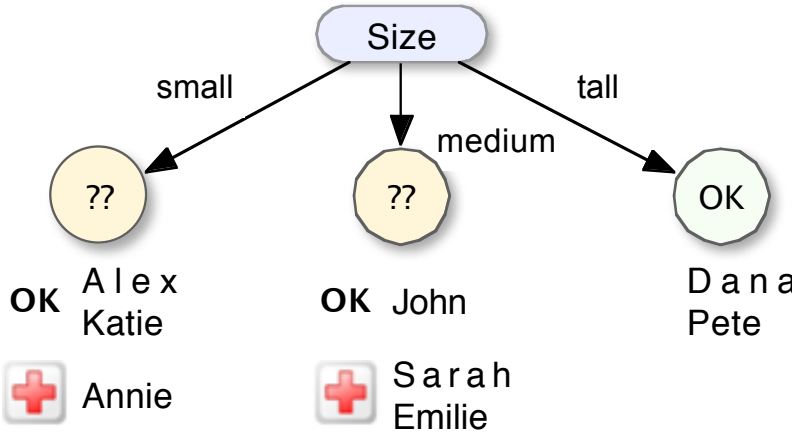
❒ **Learning the "Sunburn" concept**

	Size	Weight	Hair	Lotion	Concept
Sarah	medium	light	blond	no	sunburn
Dana	tall	normal	blond	yes	no sunburn
Alex	small	normal	brown	yes	no sunburn
Annie	small	normal	blond	no	sunburn
Emily	medium	heavy	red	no	sunburn
Pete	tall	heavy	brown	no	no sunburn
John	medium	heavy	brown	no	no sunburn
Katie	small	light	blond	yes	no sunburn

$$Initial\ \ Entropy = \underbrace{\left(-\frac{3}{8}\log_2\frac{3}{8}\right)}_{sunburn} + \underbrace{\left(-\frac{5}{8}\log_2\frac{5}{8}\right)}_{no\ sunburn} = 0.95$$

Example (2)

❒ Entropy associated to the attribute Size



Entropy associated to a set :

$$Entropie(E) = \sum_{i=1}^{n} -P_i \times Log_2(P_i)$$

Average entropy for the partition :

$$EM(E,A) = \sum_{j=1}^{V} \Pr ob(V_j) \times Entropie(E_j)$$

➢ Entropy for the three subsets

	Sunburn	Not sunburn	Σ
Entropy (small)	-1/3 . $\log_2$ (1/3)	-2/3 . $\log_2$ (2/3))	0,92
Entropy (medium)	-2/3 . $\log_2$ (2/3)	-1/3 . $\log_2$ (1/3)	0,92
Entropy (tall)	0 (empty)	-2/2 . $\log_2$ (2/2)	0

➢ Average entropy

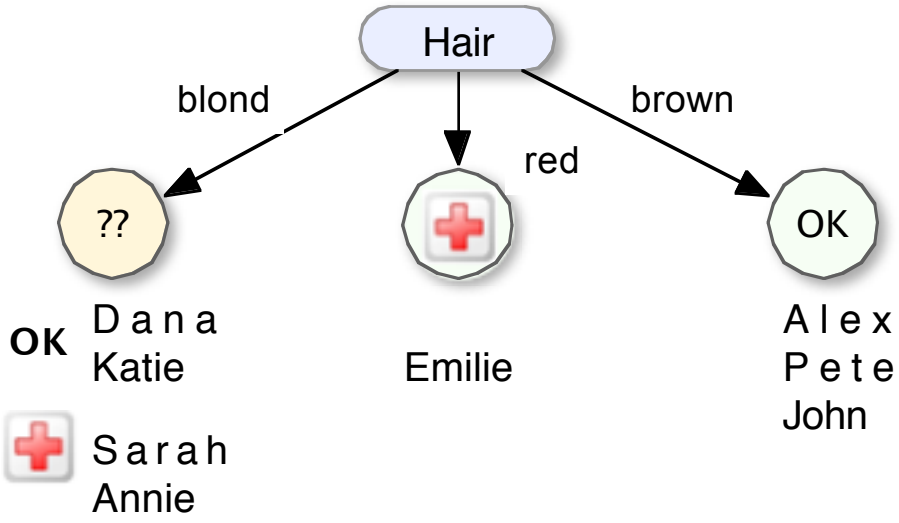
	Small	Medium	Tall	Σ
EM (E, Size)	3/8 . (0,92)	3/8 . (0,92)	2/8 . (0)	**0,69**

Example (3)

❒ **In practice four features can be used: Size, Weight, Hair, Lotion**

$$Gain(E,A) = Entropy(E) - EM(E,A)$$

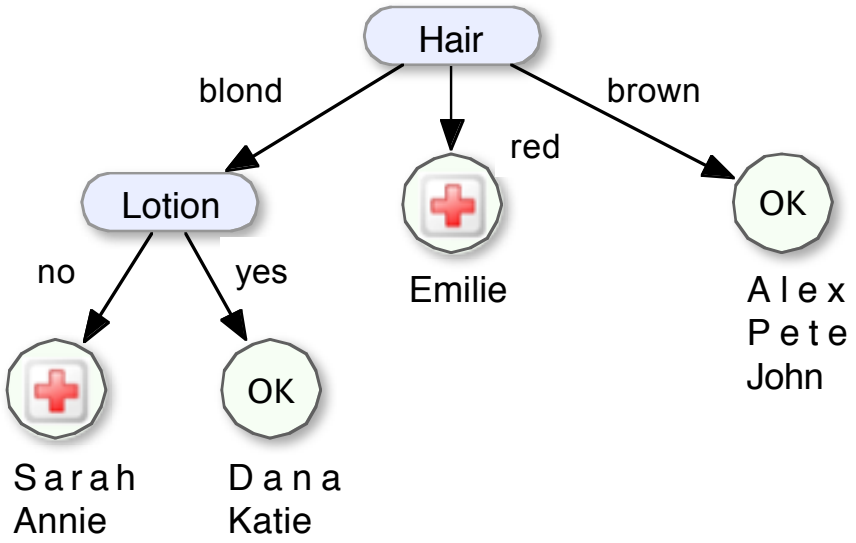
	Entropy (E)	EM (E,A)	Gain (E,A)
Attribute Size	0,95	0,69	0,26
Attribute Weight	0,95	0,94	0,01
Attribute Hair	0,95	0,5	**0,45**
Attribute Lotion	0,95	0,61	0,34



Example (4)

❒ **Now only three possibilities: Size, Weight, Lotion**

	Entropy (hair=blond)	EM (E,A)	Gain (E,A)
Attribute Size	1	0,5	0,5
Attribute Weight	1	1	0
Attribute Lotion	1	0	**1**



Corresponding set of rules:

R1 : If hair = brown
Then no sunburn

R2 : If hair= red
Then sunburn

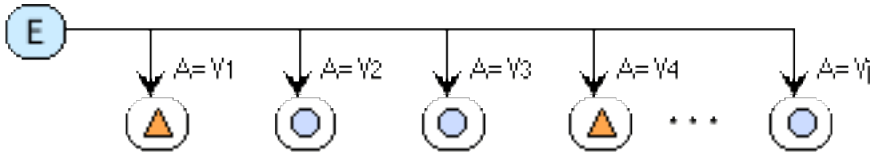
R3 : If hair = blond & lotion = yes
Then no sunburn

R4 : If hair = blonde & lotion = no
Then sunburn

Multiple values problem

❒ **How to manage attributes having a different number of values ?**

➢ Gain (E, A) tends to favored attributes having multiple values V= {v_1, …v_j)



➢ Idea : to penalize by the domain size and to favor the well-balanced repartitions

• $\text{Gain}_N (E, A) = \dfrac{Gain(E,A)}{f(Log_2|V|, Entropie(V))}$

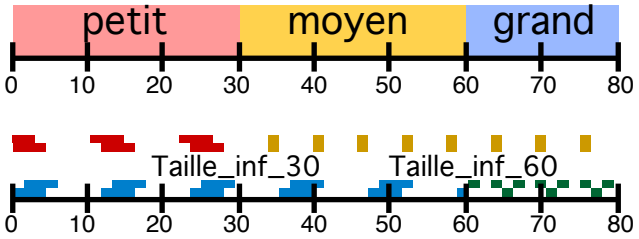
$$\text{Entropie (V)} = -\sum_{j=1}^{|V|} P(v_j) \times Log_2 P(v_j)$$

❒ **How to manage the continuous values?**

➢ A classical approach: discretization … but there are two ways:

• Ordered set of values

• Many Booleans attributes



Missing data

	A1	A2	A3	A4	•••	Ak
ex1						
ex2		???				
ex3						
•••						???
exn			???			

❒ Don't care values

➢ Some attributes cannot be simultaneously valued

Examples : Fin_nb et Leg_nb ; Melting_point et Sublimation_point ; ...

➥ To do as if the attribute has all the possible values (null hypothesis)

❒ Unknown values

➢ Some attributes have missing values

Example : Experimental domain, transcription error, …

➢ Several strategies can be used for a given node N of the decision tree

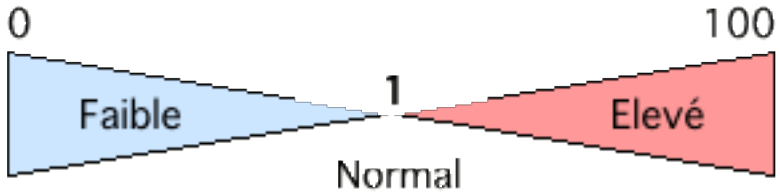
① Uses the most frequent value of the feature in this node
② Uses the most frequent value of the examples belonging to the same class
③ Duplicates the example in the node children with a weight = $P(V_i)$

Using the background knowledge

❒ **Sometime using an attribute has a well-known cost:**

- ➢ Some attributes can be “expansive” to acquire … (ex : scanner VS blood analysis)
- ➢ … or are painfull (ex : an electromyography checkup)
- ➢ … or are long to acquire (ex : display on the screen VS phone call or email)
- ➢ … or will be difficult to understand by the end-user (ex : “flower color” VS “dicotyldeon”)

☞ We need to take into account this information : Cost (A)



- $\text{Gain}_C(E, A) = \dfrac{Gain^2(E,A)}{Cost(A)}$

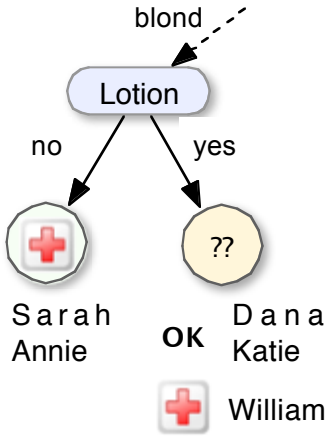
How to deal with the noise in the data

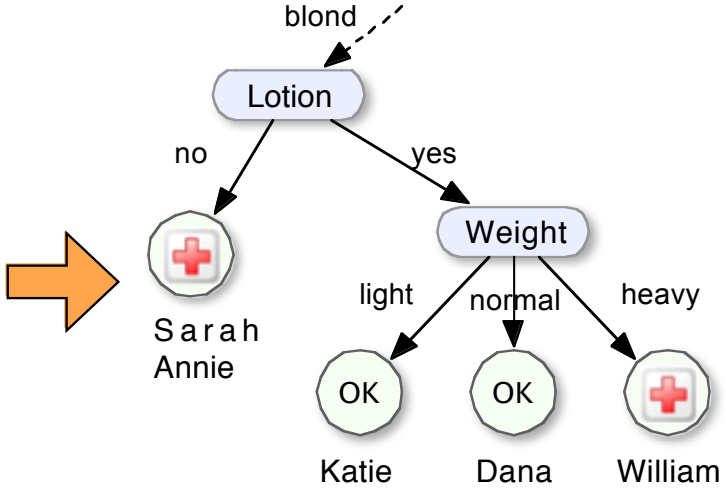
❒ Two kinds of errors

- Bad acquisition of the value of some attributes
- Wrong label (class) for some examples

❒ This leads to an overfitting problem

- William : blond, small, heavy, *lotion,* sunburn





➢ Worsens the readability of the tree
➢ Decreases the generalization capacity
➢ The model draws some wrong conclusions

So what to do?

Pruning technics

❒ During the creation of the tree ...

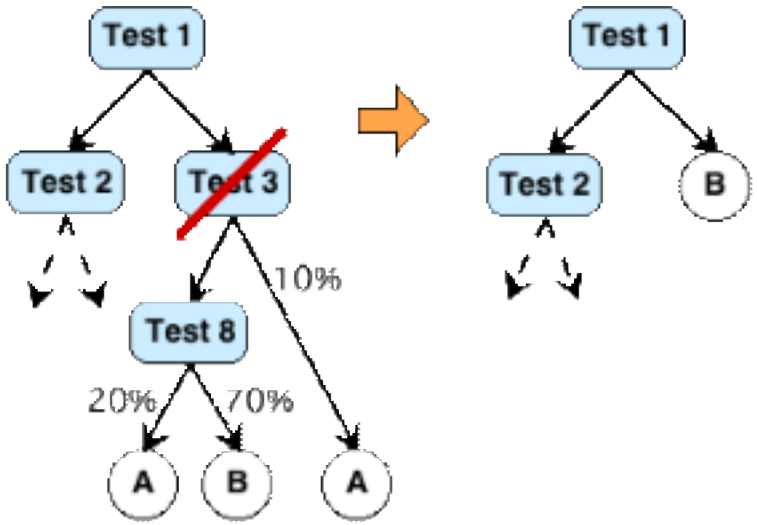
➢ We stop the building of a branch as soon as either:

- the information gain is under a given threshold, or
- the size of the example set becomes statistically unrelevant

❒ After the creation (Pruning) ...

➢ The tree is simplified using a *validation set* (≠ from training set)

Let currentErrorRate = initialError
  Foreach nodes N_i of the tree (depth first search)
    - **Transform** N_i into a leaf L
    - **Label** L with the majority class in the subtree
    - **Apply** the validation test on the tree
  If error(N_i) ≤ currentErrorRate
    Then currentErrorRate := error(N_i)
    Else remove the modification
  Return the simplified tree



Decision trees limits

❒ They cannot learn any concept

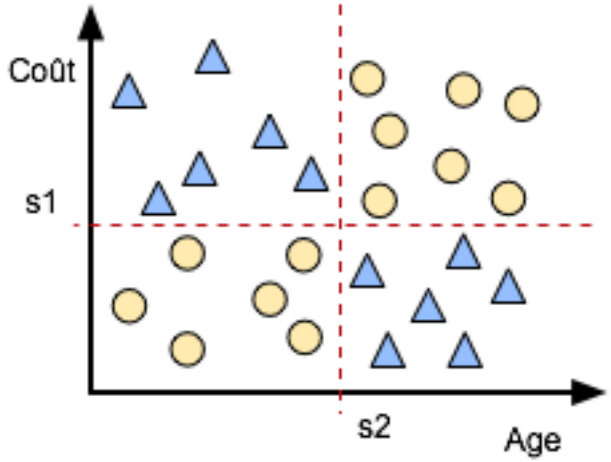
➢ Some simple concept are not writable as a tree

R1 : If a=true Then C1
R2 : If b=true Then C2

R1 : If a=true Then C1
R2 : If a=false & b=true Then C2

➢ It works under the assumption that the attributes are not correlated !

- We use a gradient search (“look ahead” of one step) ...



Simple set of rles (XOR-like)

If Cost < s1 & Age < s2 Then ○
If Cost > s1 & Age < s2 Then △
If Cost < s1 & Age > s2 Then △
If Cost > s1 & Age > s2 Then ○

But these txo attributes cannot be selected ...

- Gain (E, Cost) = Entropy(E) - EM (E, Coût) → 0
- Gain (E, Age) = Entropy(E) - EM (E, Age) → 0

❒ A partial solution

➢ To turn the tree into a set of rules then pruning of the premisses

Conclusion

❒ A very efficient approaches

➢ Very fast algorithm

➢ Low complexity

- If we use all the attributes (worst case): Θ (#Example.#Attribute^2)
- For a binary tree with P levels : Θ (#Example.#Attribute.Log$_2$(P))

➢ Many extensions of the basic algorithm :

- Incremental : ID5, ...
- Statistical pruning : **C4.5, C5.0**, ASSISTANT 86, ...
- Direct use of real numbers : NewId, ...

➢ Many commercial realeases:

- C5 : http://www.rulequest.com/
- CART 5 : http://www.salford-systems.com/products-cart.html
- CLEMENTINE 7 : http://www.spss.com/spssbi/clementine/whats_new.htm
- ...

How to experiment ML tools ?

❒ **Several "Open source" systems exist!**

- ➢ Weka: (http://www.cs.waikato.ac.nz/ml/weka/)
  - ○ A large set of tool implemented in JAVA (portability)
  - ○ Complete support for Data Mining : collect / processing / validation
  - ○ GUIs to explore interactively the dataset or to manage data flows
- ➢ Orange: (http://orange.biolab.si/)
  - ○ A growing collection of tools written in Python
  - ○ Very modern and clean user interfaces, more polished than those of Weka
  - ○ Visual programming or scripting interface in Python
- ➢ R: (http://www.r-project.org/)
  - ○ A very huge collection of methods covering Statistics and Machine Learning
  - ○ Script based interface with visualization tools
  - ○ Very good documentation with working examples

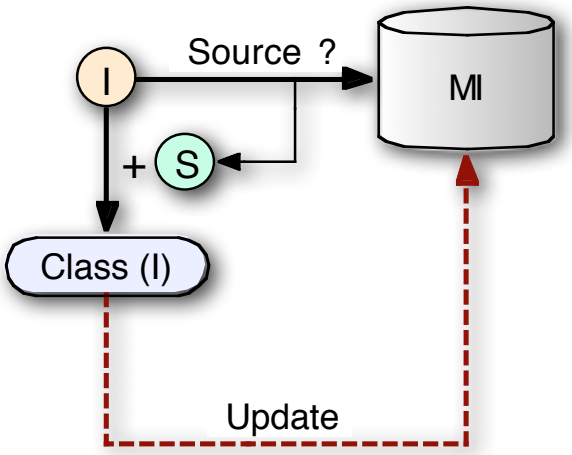
Instance Based Learning

❒ Main principles

- ➢ Learning is simply based on the memorization of the examples/instances
- ➢ This kind of approaches is known under several different names:
  - Case Based Reasoning (CBR)
  - k-Nearest Neighbors (k-NN with k = 1 here)
- ➢ Typicallly an incremental/online learning task, with 2 steps ...
  - Update of the instances database (MI)
  - Use of the learning tool to predict the concept of a new instance

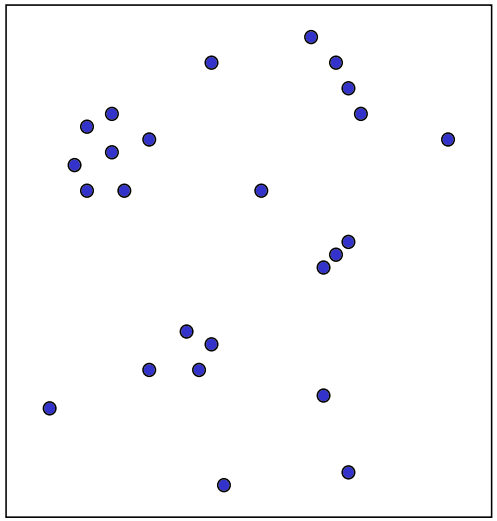
❒ Learning is based on the definition of 3 functions:

- ➢ “Similarity” measure between 2 instances S and I
- ➢ A classification criteria based on the similarity
- ➢ Update of the instance database

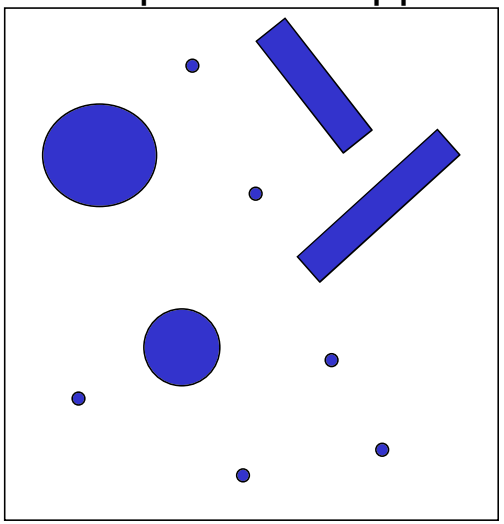


Examples VS Knowledge ...

IBL : Instances or data



Concept based approaches



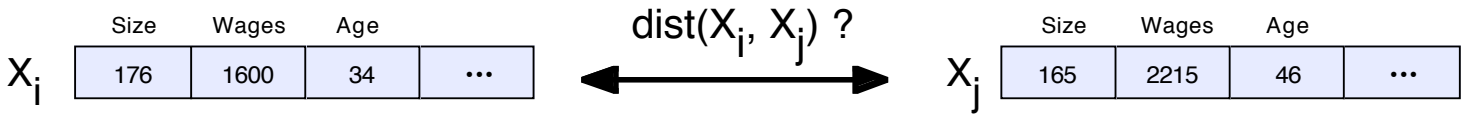
- We just work with examples
- Readibility of the database is weak
- Similarity based approach
+ Updating step is trivial
+ No information loss (robustness)

+ We built an abstract definition (rules, …)
+ Easy to read in the symbolic approach
- Generalization based approach
- Updating through a “revision process”
- We forget some information

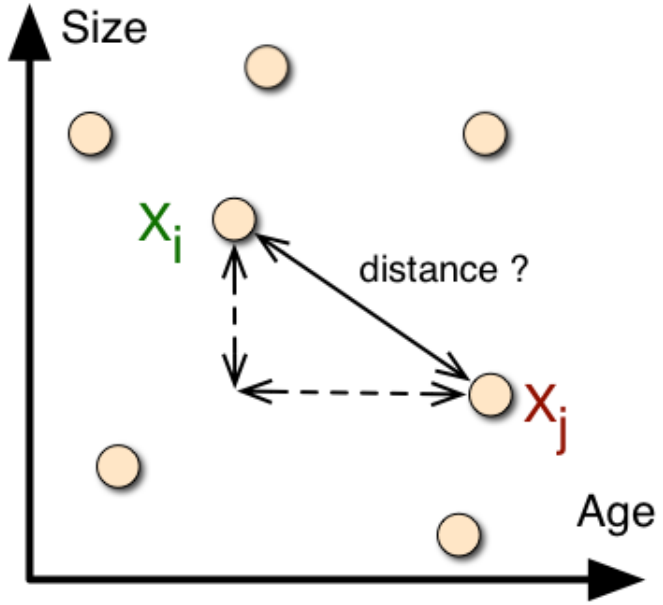
Similarity or Distance

❒ What's about ?

➢ Goal: quantifying the difference between features of two examples/instances



❒ Geometrical interpretation and properties



➢ Classical distances

Manhattan : $dist(x_i, x_j) = |age_i - age_j| + |size_i, size_j|$

Euclidian : $dist(x_i, x_j) = \sqrt{(age_i - age_j)^2 + (size_i - size_j)^2}$

Many others (see clustering lecture) …

➢ General properties of the distances (or metrics)

$\forall$ a, b, c $\in \Omega$

1) $\delta(a, a) = 0$ (minimality)
2) $\delta(a, b) = d(b, a)$ (symmetry)
3) $\delta(a, b) = 0 \Rightarrow a = b$ (identity)
4) $\delta(a, c) \leq \delta(a, b) + \delta(b, c)$ (triangular inequality)

How to compare two instances ?

❒ In practice

- ➢ Features are not always homogeneous (types : numbers, ordered, symbols, ...)
  - The distances must be defined according to the data type T : $dist_T()$
- ➢ Distances are often normalized in [0..1] to allow feature comparisons
- ➢ Similarity is sometime easier to understand : SIM (x,y) = (1 - DIST (x,y))

$$DIST(x,y) = \frac{\sum_{i}^{Attributs} dist_T(x_i, y_i)}{Card(Features)}$$

Features = union of the features of x and y

❒ Some classical definition for the « $dist_T$ » functions

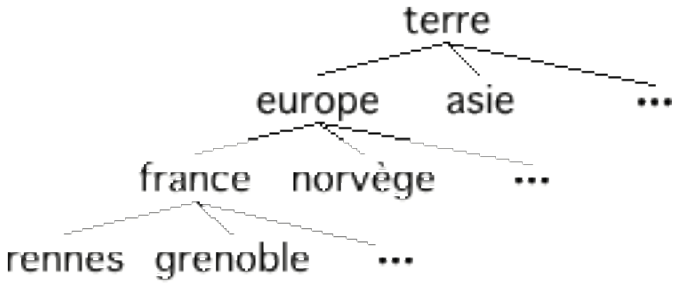
- **Boolean** : $dist_T(x_i, y_i) = if(x_i = y_i)\ then\ 0\ else\ 1$
- **Set** : $dist_T(x_i, y_i) = card(x_i \cap y_i) / card(x_i \cup y_i)$
- **Numbers** : $dist_T(x_i, y_i) = |x_i - y_i| / card_i$
- **Intervals** : $dist_T(x_i, y_i) = \left(|\min x_i - \min y_i| + |\max x_i - \max y_i|\right) / 2.card_i$
- **Taxonomy** : $dist_T(x_i, y_i) = \#\ path(x_i, y_i) / \#\ PathMax$

An example

❒ Let the following features

- Sex : boolean (male, female)
- Age : number [1..100]
- Weight : number [10..120]
- Size : ordered (XS, M, L, XL)
- Hair : ordered (blond, brown, red)
- Time : intervalle [0 .. 24]
- Home : taxonomy [(terre / continent / pays / ville)]

Taxonomy of the countries



❒ Distance between two people

	Sex	**Age**	**Weight**	**Size**	**Hair**	**Time**	**Home**
Yves	male	22	-	L	brown	[6..16]	Rennes
Joan	female	32	56	L	-	[10..24]	Norvège
$dist_T$	1	0,1	1	0	1	0,25	0,5
sim_T	0	0,9	0	1	0	0,75	0,5

- $DIST_{manhattan}$ (Yves, Joan) = 3,85 / 7 = 0,55 (and the similarity is 0,45)

What is important in a distance is the "ranking", not the "meaning"

Method IB 1

❒ Learning algorithm

Let I a new instance to classify using the database MI

Procedure IB1 (MI, I)

Foreach S ∈ MI:

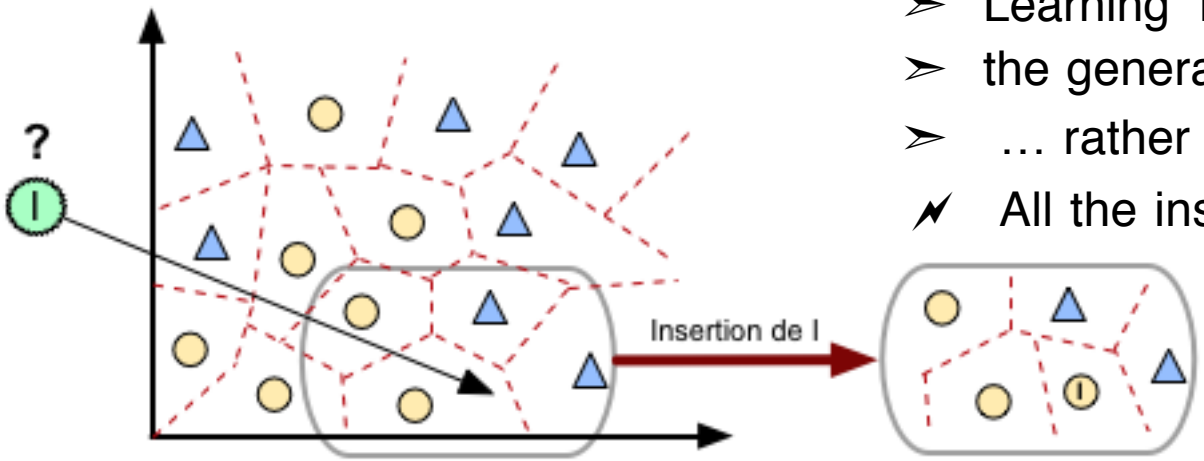
Evaluate Sim (I,S) (similarity function)

Let S_{max} the most similar case with I

Class(I) = Class (S_{max}) (classification function)

Correction of Class(I) if needed ; MI ← I (update function)

❒ IB1 analysis



- ➢ Learning “by heart” but …
- ➢ the generalization is done by SIM(,)
- ➢ … rather good prediction rates
- ✍ All the instances are stored ☹

Method IB 2

❒ Learning algorithm

Let I a new instance to classify using the database MI

Procedure IB2 (MI, I)

Foreach S ∈ MI:

Evaluate Sim (I,S) (similarity function)

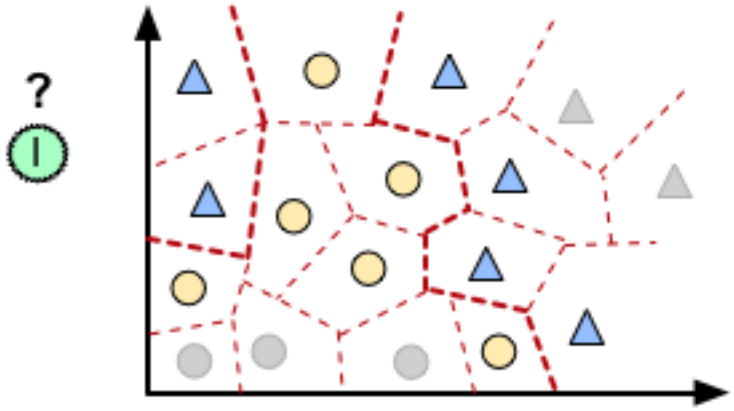
Let S_{max} the most similar case with I

Class(I) = Class (S_{max}) (classification function)

When Class(I) is wrong: (update function)

Correction of Class(I) if needed ; MI ← I

❒ IB2 analysis



➢ Only the concepts' frontiers are stored

➢ More efficient use of the memory

↯ Results are worst than those of IB1:

- Learner becomes "order sensitive"
- Learner becomes "noise sensitive"

IB 2+ with "forget"

❒ **How to deal with order/noise sensitivity**

➢ We associate 2 features to each instance : Relevance(I)=0, Usefulness (I)=0

❒ **Learning algorithm**

Let I a new instance to classify using the database MI

Procedure IB2 (MI, I)

Foreach S ∈ MI:

Evaluate Sim (I,S) (similarity function)

Usefulness (S) = Usefulness (S) - μ

Let S_{max} the most similar case with I

Class(I) = Class (S_{max}) (classification function)

Usefulness (S) = Usefulness (S) + k.μ

If Class(I) is wrong: (update function)

Correction of Class(I) if needed ; MI ← I

Relevance (S_{max}) = Relevance (S_{max}) - ε

Else: Relevance (S_{max}) = Relevance (S_{max}) + ε

Procedure Upkeep (MI)

Foreach S ∈ MI:

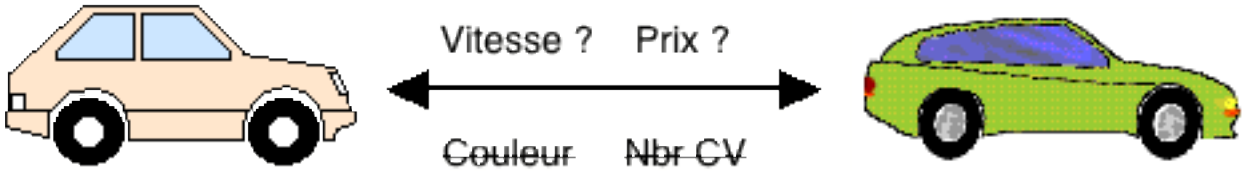
If Relevance (S) < t1 || Usefulness < t2 then MI = MI – S

Weighting of the features

❒ All the features are not relevant !

➢ Relevance is a ”goal dependant” notion

- Example : we would like to guess the type of a car (Urban, Sport) from its features



➢ Use of a weigthed distance measure

$$DIST(x,y) = \frac{\sum_{i}^{Attributs} W_i . dist_T(x_i, y_i)}{\sum_{j}^{Attributs} W_i}$$

➢ Ok, but how to choose the “right” values for the weights ? By hand ?

- Could seem easy with few values and features …
- In practice that’s a real mess
- How about learning them ?

How to learn the weights ?

❒ **Find the weight during the learning step** (RELIEF : Kira & Rendell 92, …, Sun & Wu 2007))

- Let S_{max} the most similar case with I ; class(I) = class (S_{max})
- During the update step

  Foreach of the W_A ,we compute:

$$Influence_A = \frac{W_A \ \ sim_T(S_A, I_A)}{\sum_{j}^{Attributs} W_j \ \ sim_T(S_j, I_j)}$$

- If class(I) is right, W_i is "rewarded":

$$W_A = W_A + K \times (Influence_A)^2$$

- If class(I) is wrong, W_i is "penalized":

$$W_A = W_A - K \times (Influence_A)^2$$

➢ Example (all domains are [0..100] and K = 0,5)

feature	S_{max}	I	sim_T	W_A	$Influence_A$	Class(I) is right	Class(I) is wrong
size	30	50	0,8	2	(2x0,8/2,8) =57 %	W_i=2,16	W_i=1,84
age	20	80	0,4	1	(1x0,4/2,8) =14 %	W_i=1,01	W_i=0,99
weight	45	65	0,8	1	(1x0,8/2,8) =29 %	W_i=1,04	W_i=0,96

❒ **Many other possible improvements on this basis**

➢ Weighting of the features can be class (or even instance) dependant

➢ The weights can be learned by another learning system, …