Machine Learning

Recap of main issues/phases

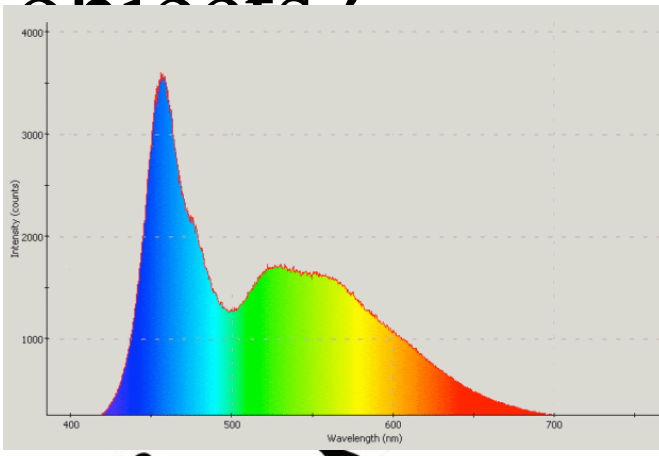
Recap: Phases/problems in designing a ML algorithm

1. Modeling the domain objects
2. Choosing a learning experience
3. Modeling the target function
4. Defining a Learning Algorithm
5. Performance Evaluation

Example : recognizing lions and frogs

1. Representation: How do we represent (model) our objects?

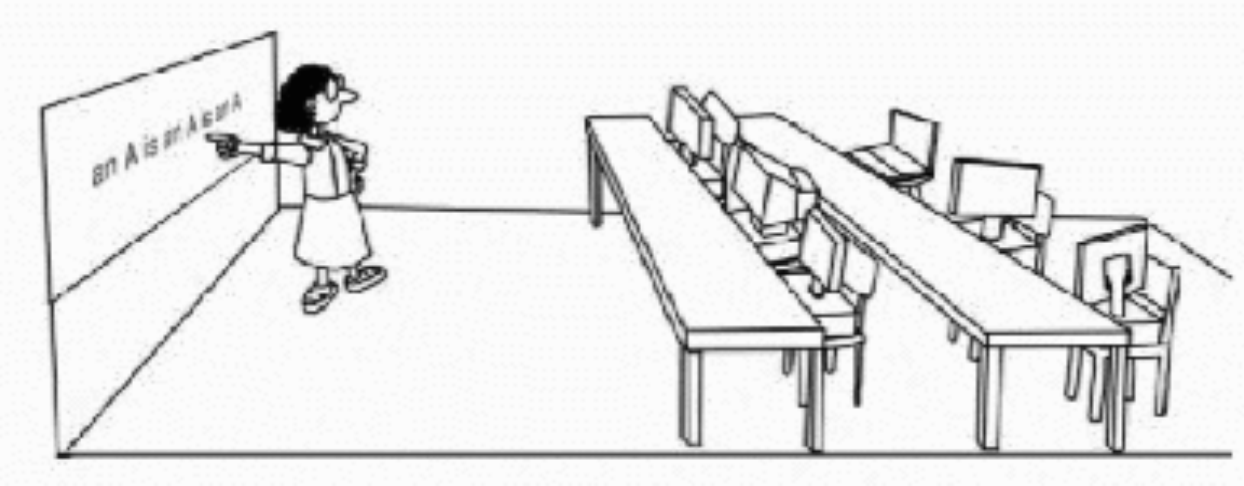
- Simple: color! (e.g. a bitmap)
- Less simple: silhouette



2. From what experience?

Types of learning:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

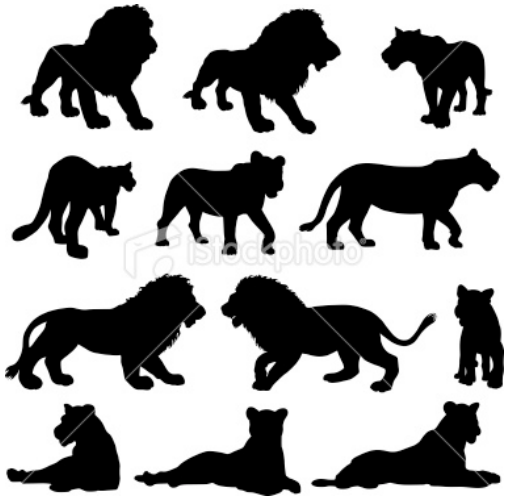


Learning paradigms

- Supervised learning: someone tells you what to do for a number of example cases. For example, the system is provided with a dataset of images already classified (cats, dogs, frogs, lions..) an must learn which features can help to distinguish between types;
- Unsupervised: There is no “teacher”. System must learn “regularities” in the data. For example, being able to group users according to their tastes, in a restaurant recommendation task
- Reinforcement learning: learning by doing. No teacher, but some target objective. System must learn the best strategy towards the target. For example, a robot moving in a hostile environment may learn from errors and achievements.

Supervised learning

- Either an "expert" (e.g. ask someone to manually classified examples) or some available database of already classified examples

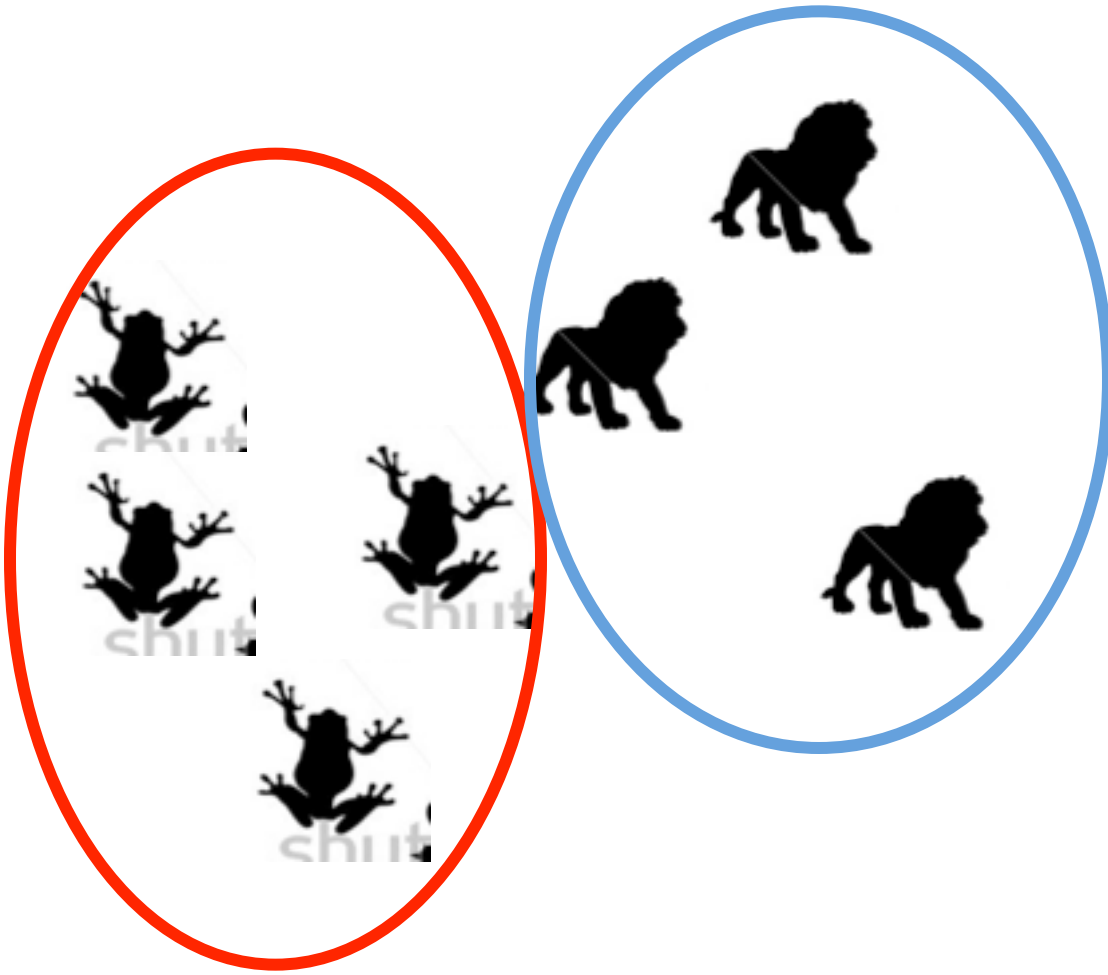


➔ lion

➔ frog

Unsupervised learning

- No examples are available. The learner must be able to identify distinguishing features that differentiate the various classes
- Note: system does not learn who is a frog and who is a lion, but only to assign an image to cluster 1 or cluster 2



Reinforcement learning

- No examples are available, but so function is provi to associate a reward (or punishment) to a good (bad) move

Machine learning types of tasks

- **Classification/categorization**: given an object, learn to assign this object to a category (chosen from a pre-defined set) or to "similarity" classes (not known a priori)
  - Image recognition: given a set of images and a set of categories (e.g. dogs, cats, lions, frogs..) assign images to the appropriate category
  - Grass grubs danger: given a set of climatic condistions etc. determine wether it is advisable or not to use pesticides (categories are "advisable" "not advisable")
  - Medical diagnosis: given a set of possible illnesses..
- **Problem solving**: given a task, learn a strategy (or adaptive strategy) to perform it
  - Given an unknown environment, learn to explore it (robot on Mars)
  - Given a physical architecture of a robot, learn a strategy (set of moves) to fly (run, swim..)
  - Self- driving car: given an environment with obstacles, drive from X to Y stopping and turning as appropriate

Summary

1. Machine learning "general" **tasks**: classification, problem solving
2. **Learning paradigms**: supervised, unsupervised, reinforcement
3. **Sub-problems**:
   - representation: how to represent domain objects and the target function
   - algorithm selection: how to learn the target function
   - evaluation: how to test the performance of the learner

Let's start!!

Inductive Classification

Machine learning tasks

- Classification
- Problem Solving

  - Classification/categorization: the set of categories is given (e.g. lion, frog)
  - Classification/clustering: the set of categories is not known (we need to "cluster" instances by similarity)
  - First case is TRAINED or SUPERVISED
  - Second is UNSUPERVISED

Supervised categorization: definition

- Given:
  - A description of an instance (=the entities we want to classify), $x \in X$, where X is the ***instance language*** or ***instance space (e.g. a way of representing instances)***.
  - A fixed (known) set of categories: $C=\{c_1, c_2,\ldots c_n\}$
- Determine:
  - The category of x: $c(x) \in C$, where $c(x)$ is a **classification function c: X→C** whose domain is X and whose range is C.
  - If $c(x)$ is a binary function $C=\{0,1\}$ ({true,false}, {positive, negative}) then it is called a ***concept*** (and we talk about CONCEPT LEARNING, or INDUCTIVE LEARNING)
  - **In inductive learning, the system tries to induce a GENERAL CLASSIFICATION RULE from a set of available classified examples**

Definiton of the supervised classification task:

- A **training example** is an instance $x \in X$, paired with its correct category $c(x)$: $<x, c(x)>$ for an unknown **categorization function**, $c(x)$.
- Usually, x is represented by a number of **features** (more precisely, a **feature vector** x:<x1,x2..xn>)
- **Given** a set of training examples, D (named *training set or learning set)*
- **Find** a hypothesized categorization function, $h(x)$, such that: $\forall < x, c(x) > \in D : h(x) = c(x)$

Consistency: the hypothesis function must be consistent with the learning set

Don’t get confused!

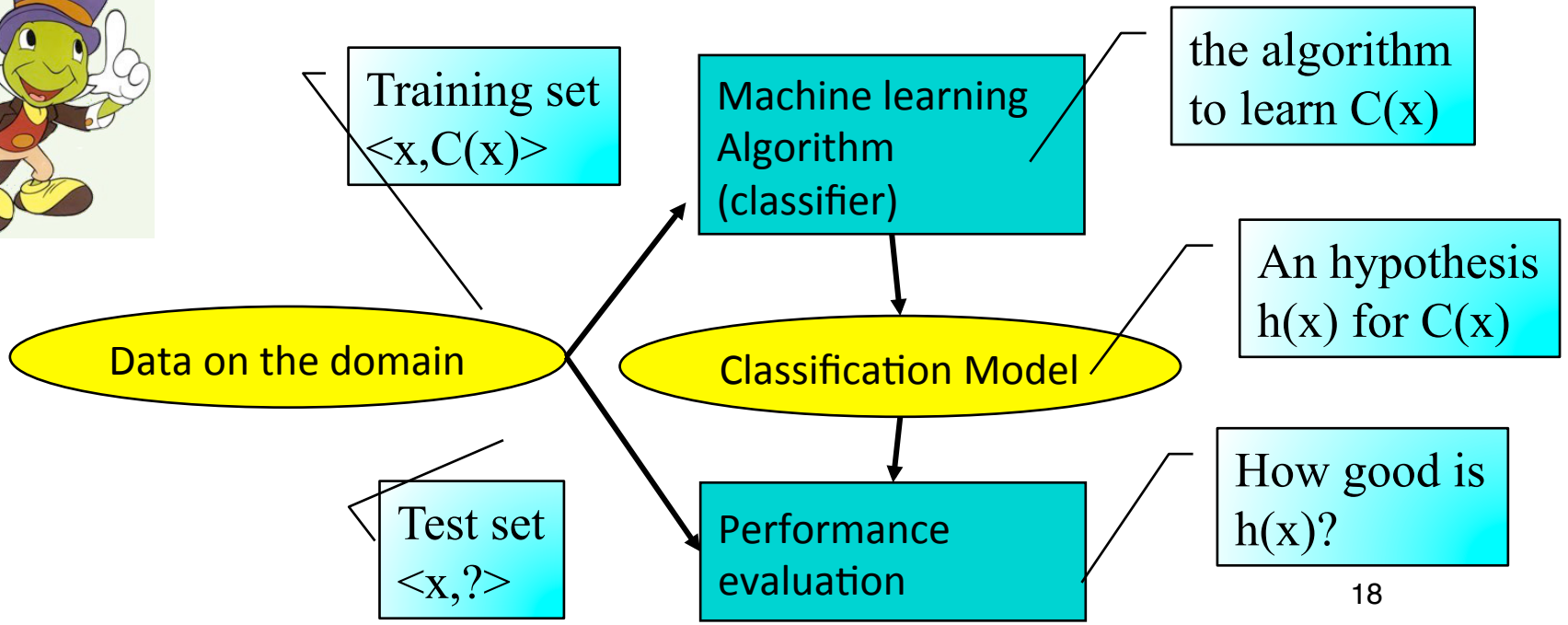
- C(x) is a classification function, that we aim to learn with a ML algorithm
- When given an object x, C(x) always returns the right classification
- Unfortunately, **perfect learning is impossible** in the vast majority of cases! We cannot learn C(x)
- That’s why we talk about hypotheses h(x): the target is to learn a function h(x) which approximates at best the real (unknown) function
- How do we know how good is a specific h(x)?? We actually don’t.. We can only ESTIMATE the goodness, using a fragment of our available classified data, named the **test set T**.
- However, in selecting our data for learning and testing, we must be careful..
- So we are ready for a more “formal” statement:

Inductive Learning Hypothesis

- *Any function that is found to approximate the target concept well on a sufficiently large set of training examples will also approximate the target function well on unobserved examples.*
- Assumes that the training and test examples are drawn **independently from the same underlying distribution** (IID).
- What we are saying here is that 1) we need "enough" data; 2)data must be representative of the domain
- However this is very vague (what is "well"? What is "sufficiently large"?)
- Additional assumptions are necessary about the target concept and the notion of "approximating the target function well on unobserved examples" should be defined appropriately (cf. *computational learning theory*).

Workflow of a Supervised Classifier

Available classified data *D* are split in learning set **L** and test set **T**. **L** is used to train the classifier (=ML algorithm). The output of learning is an hypothesis function **h(*x*)**, i.e., the *Classification Model.* To verify how good is *h(x)* we use it to classify examples in the Test set **T**. The error rate is estimated by the number of cases in which *c(x)≠h(x)* for *x in* **T**



18

A Sample Conc Problem

LANGUAGE: The name and values of features used to represent domain objects

- Instance **language**: <size, color, shape>
  - size ∈ {small, medium, large}
  - color ∈ {red, blue, green}
  - shape ∈ {square, circle, triangle
- C = {positive, negative}
- D:

This means that every instance is represented by a set of attributes, or **features**, each taking values in a finite set

Instances	Size	Color	Shape	C(x)
x1	small	red	circle	positive
x2	large	red	circle	positive
x3	small	red	triangle	negative
x4	large	blue	circle	negative

Training set

Hypothesis representation: which set of functions can we use to represent C(x)?

- As we said, may representations are possible for C(x).
- For example, here we can represent an hypothesis h(x) for C(x) e.g. with a **boolean expression**, or a rule, e.g.
- If **(color=red)&(shape=circle)** THEN C=**positive**
- Or equivalently: **red&circle (if boolean expr. is true, then c(x)=1)**

Example	Size	Color	Shape	Category
1	small	red	circle	**positive**
2	large	red	circle	**positive**
3	small	red	triangle	negative
4	large	blue	circle	negative

Do we have any possible choice for c(x)?

- Of course NO!!
- In general, we can choose boolean, algebraic or probabilistic functions, BUT possible choices depend on the domain objects and their complexity of representation (e.g. if features are dependent or independent, if they are boolean – or can approximate with a boolean- discrete, or continuous, etc.)
- **In inductive learning we restrict to boolean or discrete feature representation, and boolean functions for c(x)**

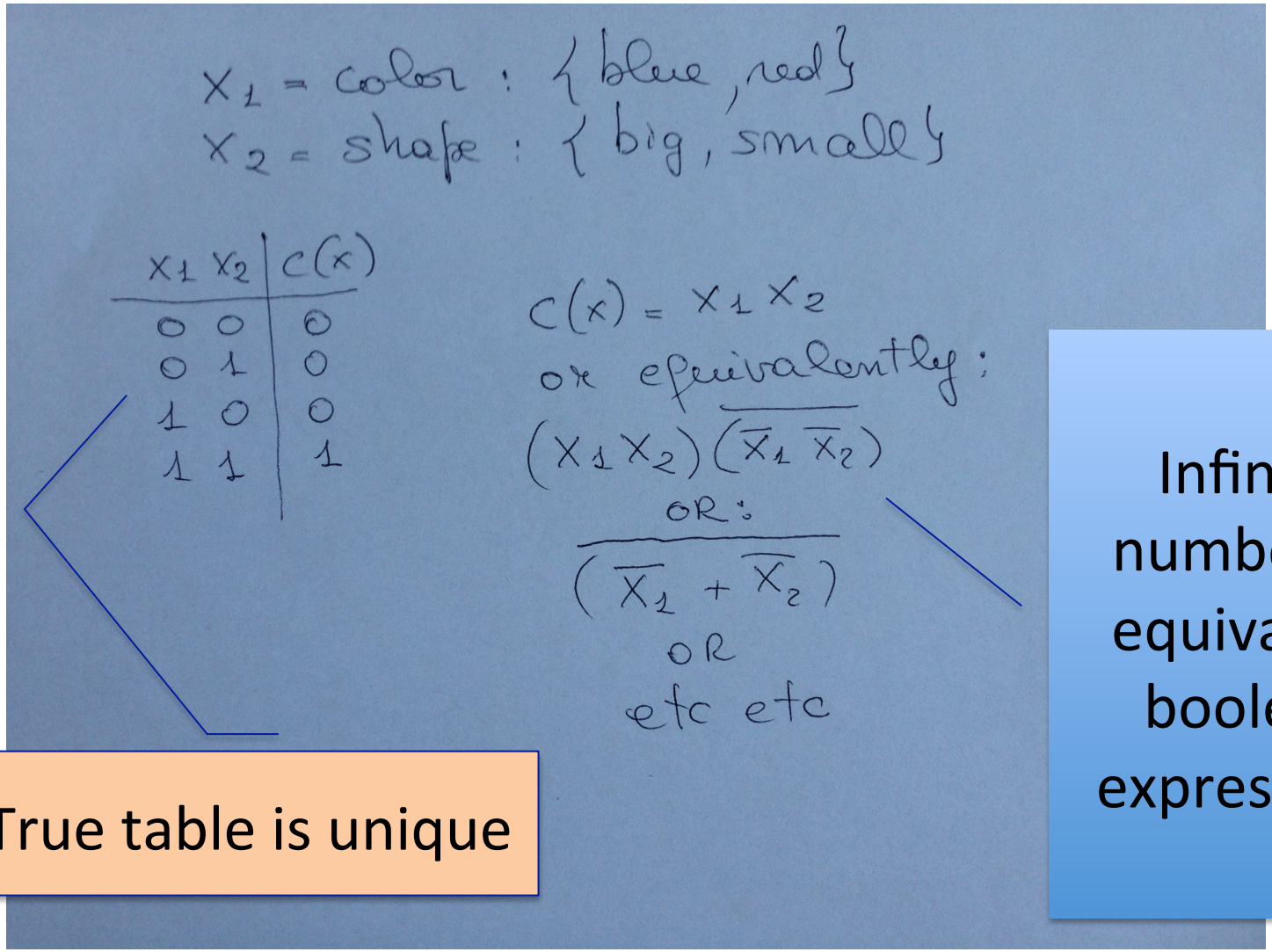
Hypothesis Selection

- **Many hypotheses are usually consistent with the training data (number of equivalent boolean expressions i**

*Notice that the first is the conjunctive normal form **CNF***

  - red & circle
  - (small & circle) or (large & red)
  - (small & red & circle) or (large & red & circle)
  - not [(red & triangle) or (blue & circle)]
  - not [(small & red & triangle) or (large & blue & circle)]

You should know (but just in case..)



True table is unique

Infinite number of equivalent boolean expressions

So how to choose c(x)?

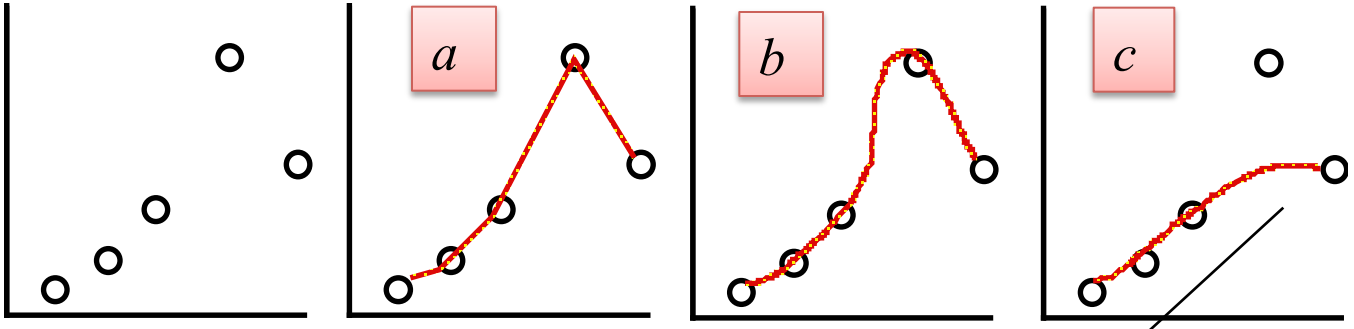
- **Bias**
  - Bias is any criterion other than "consistency with the training data" that is used to select alternative hypotheses.
  - E.g: "*preferring a conjunctive form*" is an example of bias. For example, we decide to learn a boolean function, but among the possible boolean functions, we select conjunctive forms.

Inductive Bias

- A hypothesis space that does not include all possible classification functions on the instance space is said to **incorporates a bias** in the type of classifiers it can learn (e.g. restricting to conjunctive functions is a bias in boolean concept learning)
- Any means that a learning system uses to choose between two functions that are both consistent with the training data is called ***inductive bias***.
- Inductive bias can take two forms:
  - ***Language bias***: The language for representing concepts defines a hypothesis space that does not include all possible functions (e.g. linear vrs boolean functions).
  - ***Search bias***: The language is expressive enough to represent all possible functions (e.g. disjunctive normal form) but the search algorithm embodies a preference for certain functions over others (e.g. conjunctive functions, or unconsistent functions) This is called **syntactic simplicity**.

BIAS

- More in general, bias is a criterion for preferring a set of hypoteses over another



Here for example we relax the consistency criterion

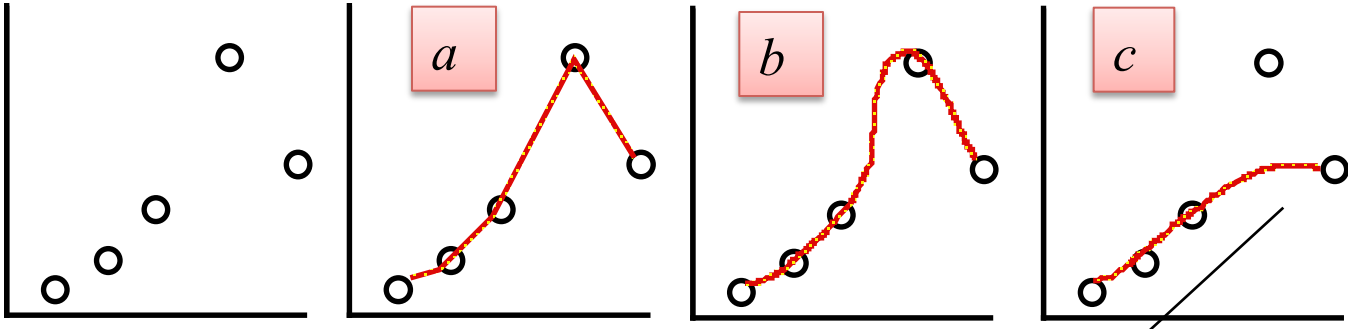
- **a** an vrs. non-linear functions), **b** and **c** have a different **search** bias, since **c** relaxes consistency in favor of simplicity

Ockham (Occam)'s Razor

- William of Ockham (1295-1349) was a Franciscan friar who applied the criteria to theology:
  - "Entities should not be multiplied beyond necessity" (Classical version but not an actual quote, which is: *entia non sunt multiplicanda praeter necessitatem*)
  - "The supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience." (Einstein)
- Requires a precise definition of "simplicity".
- Acts as a bias which assumes that nature itself is simple.
- Role of Occam's razor in machine learning remains controversial (more on CLT course).

BIAS

- More in general, bias is a criterion for preferring a set of hypoteses over another



Here for example we relax the consistency criterion

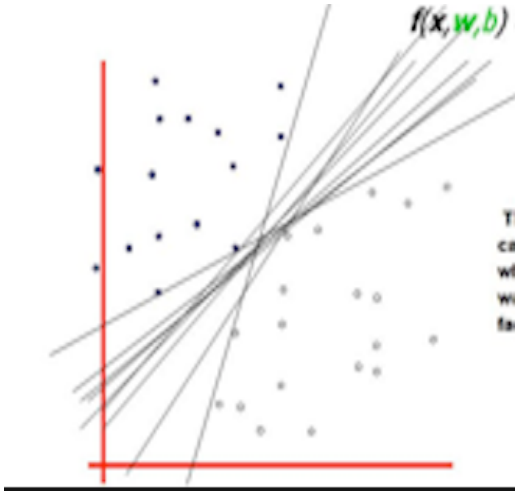
- **a** an vrs. non-linear functions), **b** and **c** have a different **search** bias, since **c** relaxes consistency in favor of simplicity

Ockham (Occam)'s Razor

- William of Ockham (1295-1349) was a Franciscan friar who applied the criteria to theology:
  - "Entities should not be multiplied beyond necessity" (Classical version but not an actual quote, which is: *entia non sunt multiplicanda praeter necessitatem*)
  - "The supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience." (Einstein)
- Requires a precise definition of "simplicity".
- Acts as a bias which assumes that nature itself is simple.
- Role of Occam's razor in machine learning remains controversial (more on CLT course).

Hypothesis Space (1)

- How difficult is learning an hypothesis? It depends upon the number of alternatives! In other terms, it depends on the DIMENSION OF THE HYPOTHESIS SPACE
- Learned functions a priori restrict to a given ***hypothesis space***, H, of functions $h(x)$ that can be considered as possible hypotheses for $c(x)$.
- Depending upon the chosen class of hypotheses (conjunctive forms, boolean expressions, algebraic functions, probabilities..) the hypothesis space can be small, large, or infinite!! (E.g. if c(x) is linear function, like $c(x)=w_1x_1+w_2x_2+..w_nx_n$ – with x_i real-valued or discrete features and w_i *real-valued coefficients*, there might be infinite linear functions that correctly classify the examples in an available dataset!)
- In inductive learning, the hypothesis space is **FINITE**. But, how big?



Hypothesis Space of boolean functions (1)

- So how many hypotheses for a boolean function with *n* features, each of which can assume the values 0, 1 or "?" (don't care)?
- =number of distinct truth tables with 2^n rows is 2^{2^n}

  (remember: there is <u>one</u> truth table for any boolean function, but <u>infinite</u> number of equivalent logic expressions)
- Example n=2 (boolean features are x,y) An hypothesis is a specific truth table, with two boolean variables. So, |H|= $2^{2^2} = 2^4 = 16$

Hypothesis Space of boolean functions (1)

- So how many hypotheses for a boolean function with *n* features, each of which can assume the values 0, 1 or "?" (don't care)?
- =number of distinct truth tables with 2^n rows is 2^{2^n}

  (remember: there is <u>one</u> truth table for any boolean function, but <u>infinite</u> number of equivalent logic expressions)
- Example n=2 (boolean features are x,y) An hypothesis is a specific truth table, with two boolean variables. So, |H|= $2^{2^2} = 2^4 = 16$

Example

xy	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10	h11	h12……				
00	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
10	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
11	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Hypothesis Space of conjunctive functions (2)

- Let's compute |H| for **conjunctive functions** like *small&red*
- To learn concepts on instances described by *n* discrete-valued features, consider the **space of conjunctive hypotheses** represented by a vector of *n* features

  $<f_1, f_2, \ldots f_n>$ where each f_i is either:
  - ?, a *wild card* indicating **no constraint on the *i*th feature** (= the feature is irrelevant)
  - A **specific value** from the domain of the *i*th feature (ex: color=red)
  - Ø indicating **no value is acceptable** (=there are NO objects belonging to the target class in the dataset)
- Sample conjunctive hypotheses have the following shape (wrt previous example):
  - <big, red, ?> (*equivalent to big&red, or [size=big]&[color=red]*)
  - <?, ?, ?> (most general hypothesis, $\forall x,\ c(x) = 1$)
  - < Ø, Ø, Ø> (most specific hypothesis, $\forall x,\ c(x) = 0$)

- **Notation**: I can represent both **instances** and **hypotheses** as VECTORS, e.g.
- x: < small,red,circle>➔ (size=small)&(color=red)&(shape=circle)
- h: <small,?,circle> ➔ IF (size=small)&(color=don't care)&(shape=circle) THEN True

Hypothesis Space of conjunctive functions (3)

- How many conjunctive functions? Any feature can be 0, 1, always false (indicated with $\varnothing$) or always true (indicated with ?) therefore 4^n
- However all hypotheses with at least one feature equal to $\varnothing$ are equivalent (they are all false) therefore $|H| = 3^n+1$
- Example n=2 $H=\{\varnothing, ?, xy, x\bar{y}, \bar{x}y, \bar{x}\bar{y}, x?, ?y, \bar{x}?, ?\bar{y}\}$
- So 10 possible hypotheses for C(x)

Criteria to select best hypothesis: Generalization

- Even with a bias, a ML algorithm may output several different hypotheses, all consistent with the learning set. How do we choose?
- Hypotheses must **generalize** to correctly classify instances **not in the training data**.
- *Simply memorizing training examples is a consistent hypothesis that does not generalize*:

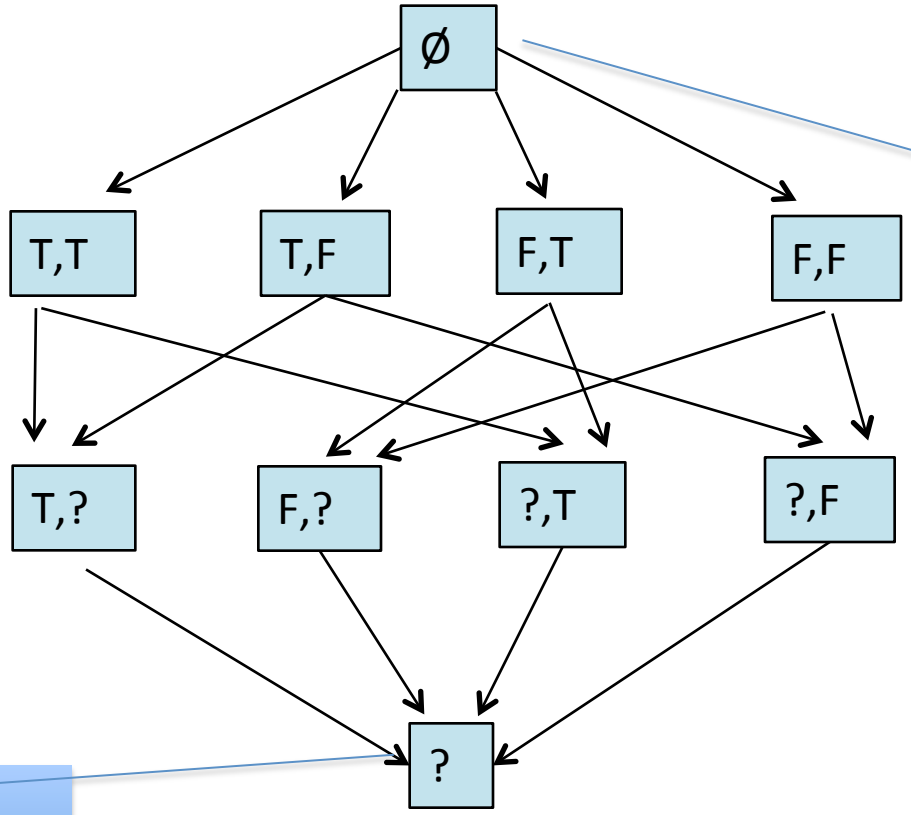
((small&red&circle)or(large&red&circle))&(not((small & red & triangle) or (large & blue & circle)or…))

- *Occam's razor*:
  - Finding a *simple* hypothesis helps to ensure generalization.
- **BUT: how do we know that one hyp. is more general than others?**

Using the Generality Structure

- Given two hypotheses h_1 and h_2, h_1 is ***more general than or equal to*** h_2 ($h_1 \geq h_2$) iff every instance that satisfies h_2 also satisfies h_1.
- Given two hypotheses h_1 and h_2, h_1 is **(*strictly*) *more general than*** h_2 ($h_1 > h_2$) iff $h_1 \geq h_2$ and it is not the case that $h_2 \geq h_1$.
- **Generality defines a partial order on hypotheses.**

Example hypothesis space for conjunctive functions (two binary features) ordered by generality

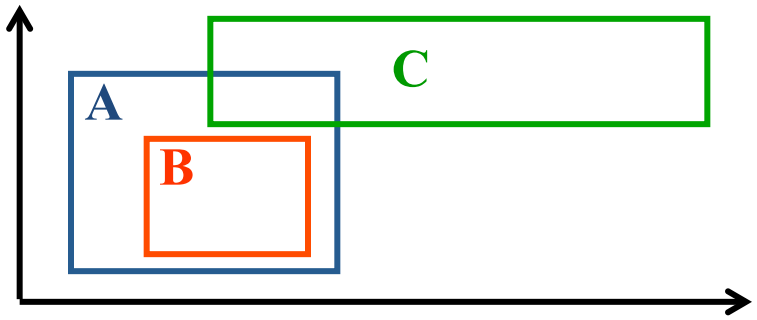


Here all hypotheses with at least one "Ø"

Here the hypothesis with all "?"

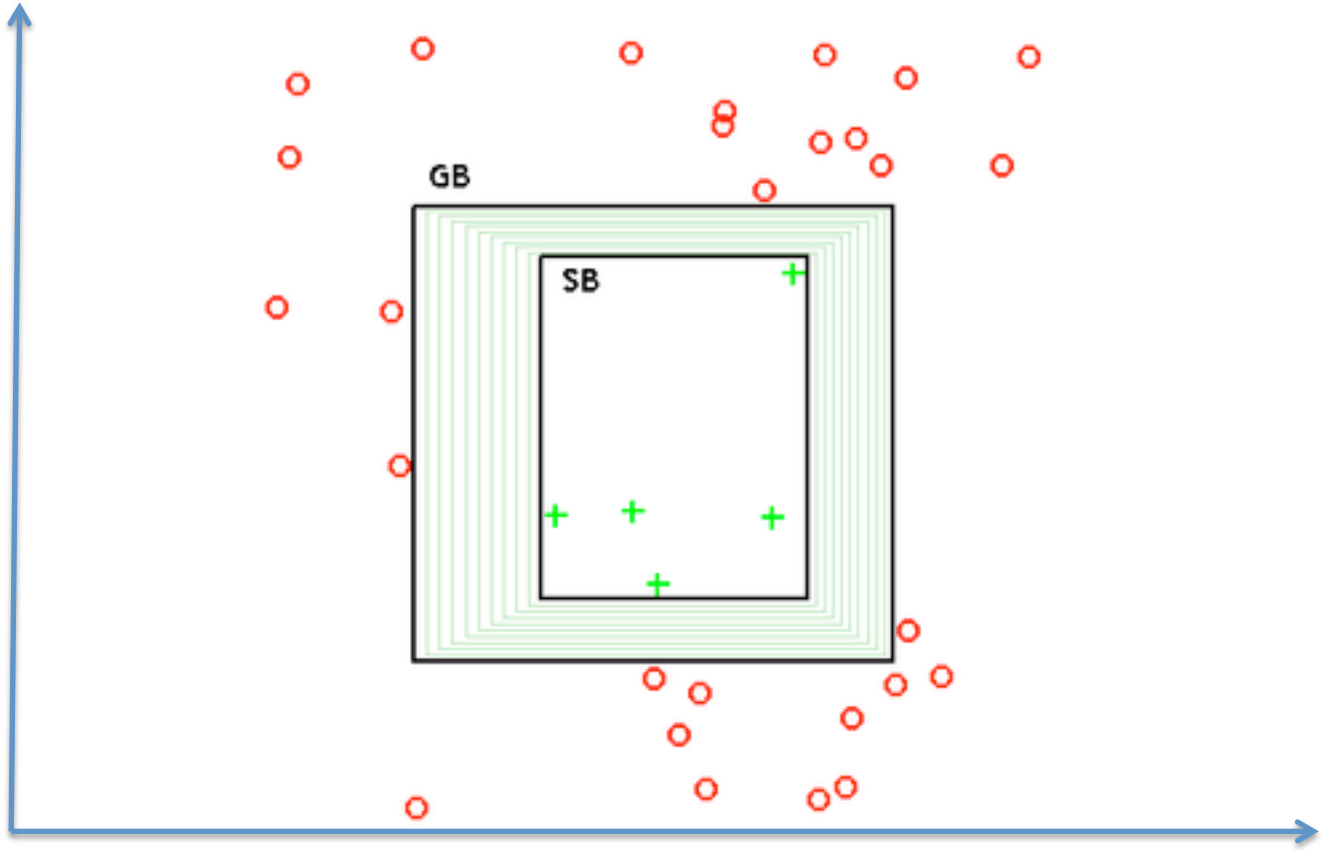
Other examples of Generality

- Conjunctive feature vectors
  - <?, red, ?> is more general than <?, red, circle> (remember ?="any value is ok")
  - Neither of <?, red, ?> and <?, ?, circle> is more general than the other.
- Example: Axis-parallel rectangles in 2-d space



  - A is more general than B
  - Neither of A and C are more general than the other.

Digression: What are these rectangles?



Suppose X axis is cholesterol and Y is age. Green points are positive for illness M and each rectangle represents the rule: ***if A<chol<B AND C<age<D then M***

Therefore (hyper)rectangles are a class of hypotheses represented by ranges.

Question

- Consider h1: <big, red, circle> and
  h2: <?, blue, ?>
- Is h2>h1?
- Given two hypotheses h_1 and h_2, h_1 is ***more general than or equal to*** h_2 ($h_1 \geq h_2$) iff every instance that satisfies h_2 also satisfies h_1.
- So, is h2>h1?

Evaluation of hypotheses

We need to evaluate how the selected hypothesis "approximates" the real (unknown) classification function

How? We use the training set (a fraction of the original classified dataset D that we DID NOT USED during the learning phase)

We need to measure:

- Classification accuracy (% of instances classified correctly).
  - Measured on an **independent test data**.
- Training time (efficiency of training algorithm).
- Testing time (efficiency of subsequent classifications, when the system is "operative").
- **We will devote a lesson to ML systems evaluation**

Summary so far

- We learned the workflow of a ML learning system
- We have seen that **complexity of learning** depends (also) upon the "shape" of the classification function to be learned (boolean,algebraic, probabilistic) and on the complexity of representation of the objects to be classified (how many features? Boolean, discrete or continuous? Are the features related or independent?)
- We measured the (a priori) **dimension of the hypothesis space** for the case of boolean functions and, within boolean functions, of conjunctive functions
- We also defined the notion of **generality of hypotheses** and we have shown that, for conjunctive functions, we can build the hypothesis space as a tree structure
- Generality is important, but the **goodness** of an hypothesis needs to be **evaluated** against an (independent) test set

But how do we learn an hypothesis?

- We know that hypotheses can have different shapes, can be finite of infinite, can be ordered according to generality, can be evaluated
- BUT: what about LEARNING a “good” hypothesis?
- Let’s start with algorithms (simple algorithms for conjunctive learning, to begin..)

Concept (or Inductive) learning

Algorithms to learn boolean classifications

Concept learning (summary so far)

- Objective: learn a boolean classification c(x) for objects $x \in X$
- Concept learning is a form of **supervised learning**: we are given a set *D* of pairs <x,c(x)> for which the classification is known.
- Every object *x* is described by a set of features (also called attributes): $x : \langle f_1, .. f_n \rangle$
- Features are either boolean or discrete-valued

Boolean functions

- Concept learning implies learning an hypothesis *h(x)* for *c(x)*
- Perfect learning is usually impossible, the objective is to learn a "good" approximation
- Consistent learning is when:

$$\forall x \in D \; h(x) = c(x)$$

Conjunctive Rule Learning

- Conjunctive functions are easily learned by finding all commonalities shared by all positive examples.

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative

Learned rule: red & circle → positive

- Must check **consistency** with negative examples. If inconsistent, **no** conjunctive rule exists.

Limitations of Conjunctive Rules

- If a concept does not have **a single set of necessary and sufficient conditions**, conjunctive learning fails.

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative
5	medium	red	circle	negative

Learned rule: red & circle → positive

Inconsistent with negative example #5!

Disjunctive Concepts

- Concept may be disjunctive (in this case a conjunctive hypthesis cannot be found!)

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative
5	medium	red	circle	negative

Learned rules:

small & circle → positive

large & red → positive

h(x)= (small & circle) or (large & red)

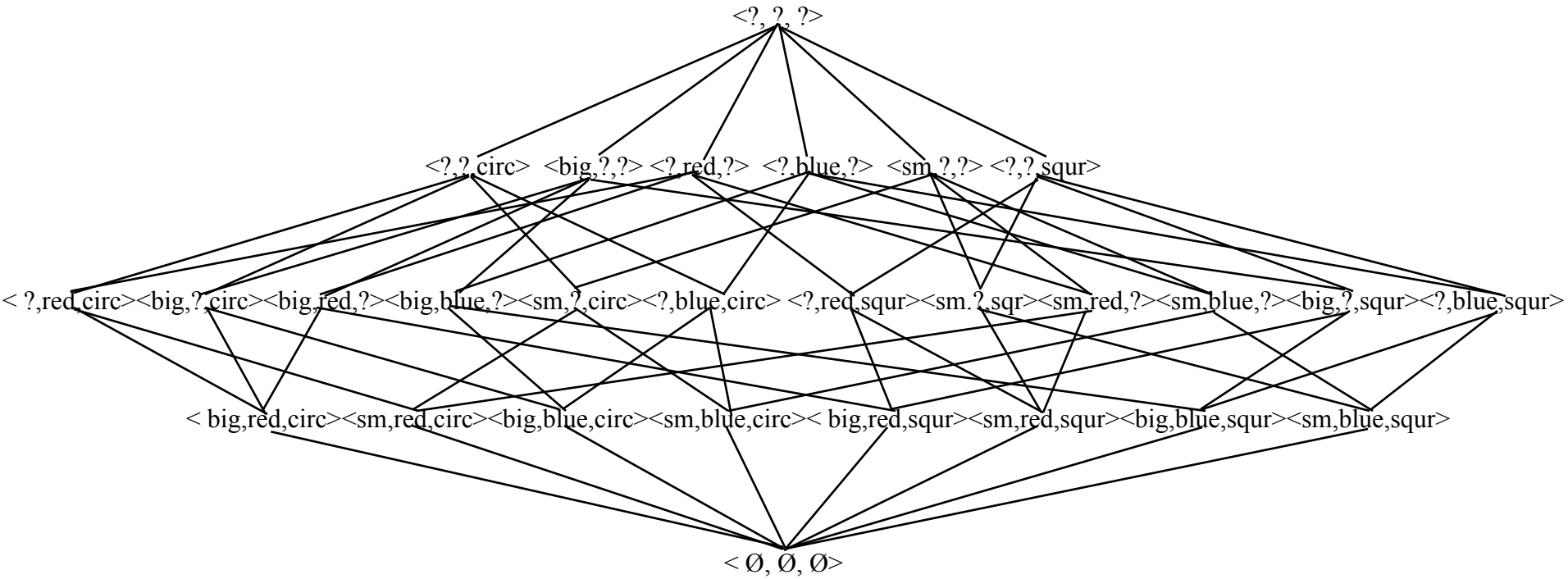
Concept Learning as Search

- Conjunctive hypotheses are a small subset of the space of possible boolean functions
- We can see conjunctive learning as the task of searching the best hypothesis while travelling in the search space

Generalization Lattice

Let's consider a simpler case wrt previous example, now features are all boolean

Size: {small, big} Color: {red, blue} Shape: {circle, square}



Number of hypotheses = $3^3 + 1 = 28$

Algorithm 1: Most Specific Learner (Find-S)

- **Find the most-specific hypothesis (least-general generalization, LGG) that is consistent with the training data.**
- Incrementally update hypothesis h(x) after every positive example, generalizing it "just enough" **to be consistent with** the new example (we also say that h(x) "does not satisfies x").
- For conjunctive feature vectors, this is easy:

Initialize $h = \langle \emptyset, \emptyset, \ldots \emptyset \rangle$

For each **positive** training instance x in D

  For **each feature** f_i

    If the constraint on f_i in h is **not** satisfied by x

      If f_i in h is $\emptyset$

        then set f_i in h to the value of f_i in x

        else set f_i in h to "?"

If h is consistent with **all negative training instances** in D

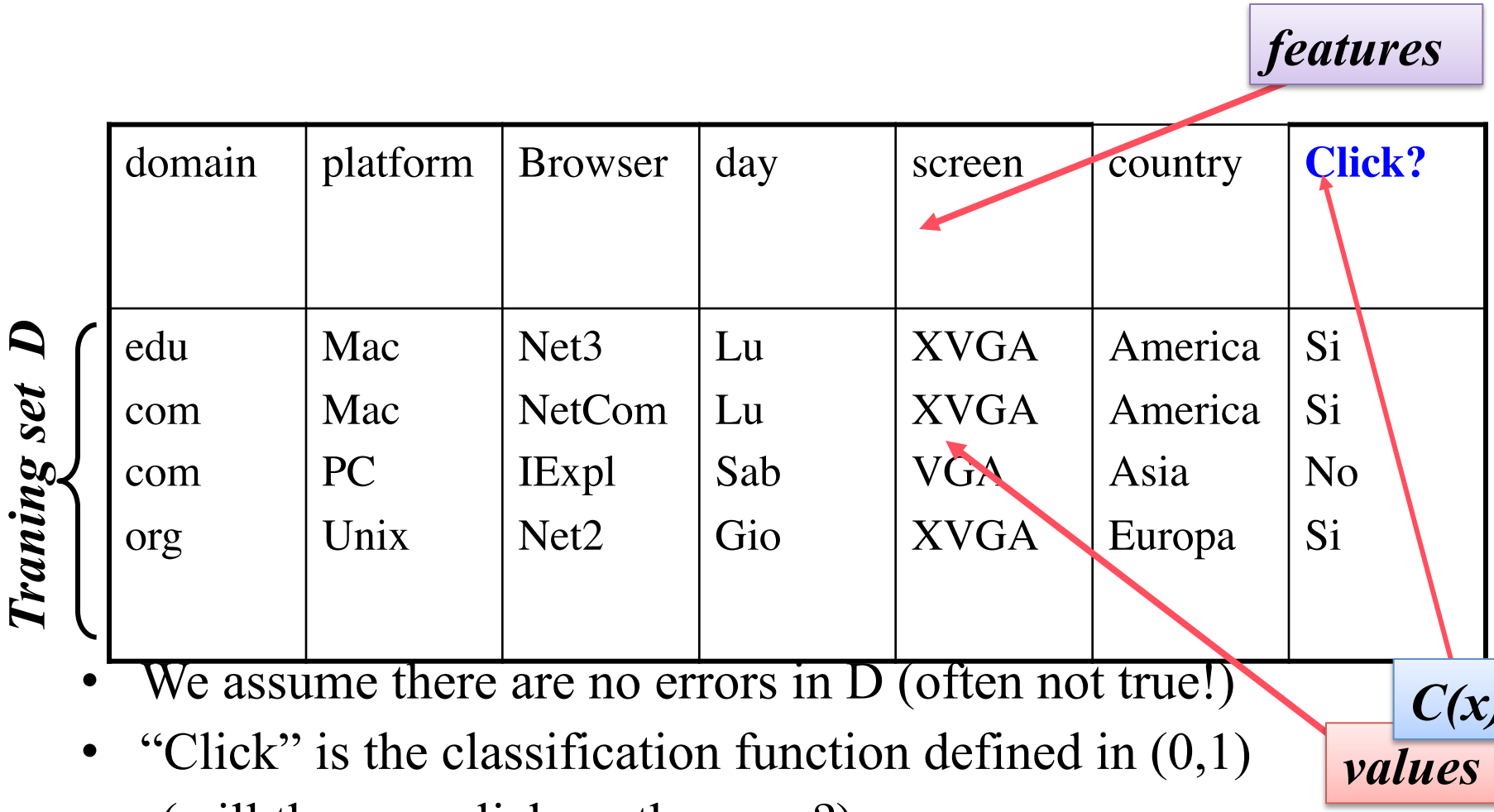
  then return h

  else no consistent hypothesis exists

Time complexity: $O(|D|\, n)$ if n is the number of features

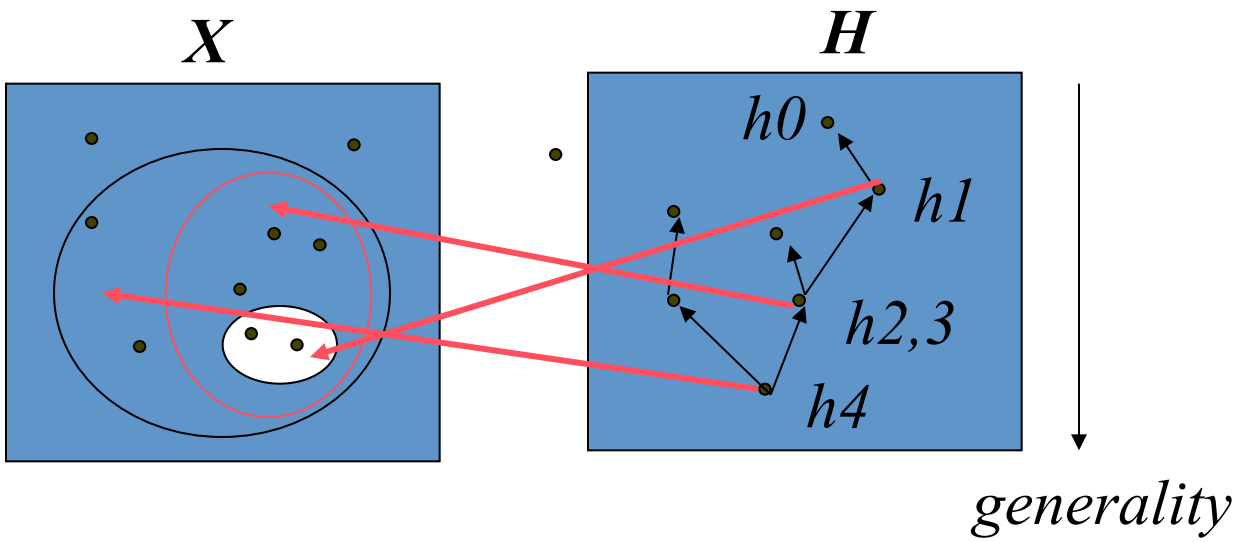
$$f_i^h \equiv f_i^x$$

(an odd) Example: learning a user profile



- We assume there are no errors in D (often not true!)
- "Click" is the classification function defined in (0,1) (will the user click on the page?)

Find-S



Training set D

Hypohesis space H

X1= (<edu,mac,Net3,Lun,XVGA,America>,1)

X2=(<com,mac,Net3,Mar,XVGA,America>1)

X3= (<com,PC,IE,Sab,VGA,Eur>,0)

X4= (<org,Unix,Net2,Mer,XVGA,America>,1)

h0 = <∅, ∅, ∅, ∅, ∅, ∅>

h1=<edu,mac,Net3,Lun,XVGA,America>

h2=<?,mac,Net3,?,XVGA,America>

h3=<?,mac,Net3,?,XVGA,America>

h4=<?,?,?,?,XVGA,America>

1= positive, c(x)=true; 0=negative, c(x)=false

Properties of Find-S

- For conjunctive feature vectors, the most-specific hypothesis **is unique** and found by Find-S (if enough examples are provided).
- If the most specific hypothesis **is not consistent with the negative examples**, then there is no consistent function in the hypothesis space, since, by definition, it cannot be made more specific and retain consistency with the positive examples.
- Notice however that FIND S does not consider negative examples! (consistency is checked for at the end)
- For conjunctive feature vectors, if the most-specific hypothesis is inconsistent, **then the target concept must be disjunctive**.

Issues with Find-S

- Given sufficient training examples, does Find-S converge to a correct definition of the target concept (assuming it is in the hypothesis space)?
- How do we know when the hypothesis has converged to a correct definition?
- Why prefer the most-specific hypothesis? Are more general hypotheses consistent? What about the most-general hypothesis? What about the simplest hypothesis?
- If the least general generalization LGG is not unique
  - Which LGG should be chosen?
  - How can a single consistent LGG be efficiently computed or determined not to exist?
- What if there is noise in the training data and some training examples are incorrectly labeled?

Effect of Noise in Training Data

- Frequently realistic training data is corrupted by errors (noise) in the features or class values.
- Such noise can result in missing valid generalizations.
  - For example, imagine there are many positive examples like #1 and #2, but out of many negative examples, only one like #5 that actually resulted from a error in labeling.

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative
5	medium	red	circle	negative

Algorithm 2: Version Space

- Given an hypothesis space, *H*, and training data, *D*, the ***version space*** is the *complete subset of H that is consistent (compatible)with D*.
- The version space can be naively generated for any finite *H* by enumerating **all hypotheses and** eliminating the inconsistent ones.
- Can one compute the version space more efficiently than using enumeration (considering all hypothesis space and ordering h_i?)

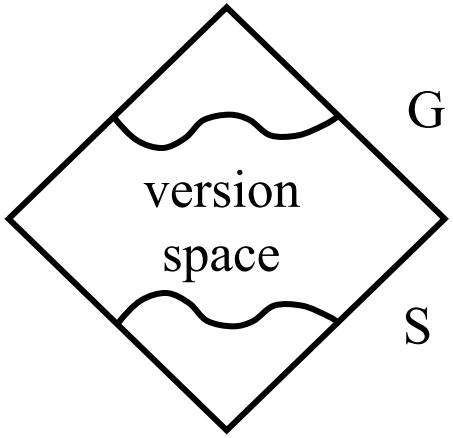
Version Space with S and G

- The version space can be represented in a more compact way by maintaining **two boundary sets** of hypotheses, *S*, the set of **most specific consistent hypotheses**, and *G*, the set of **most general consistent hypotheses**:

$$S = \{s \in H \mid Consistent(s, D) \wedge \neg \exists s' \in H[s > s' \wedge Consistent(s', D)]\}$$

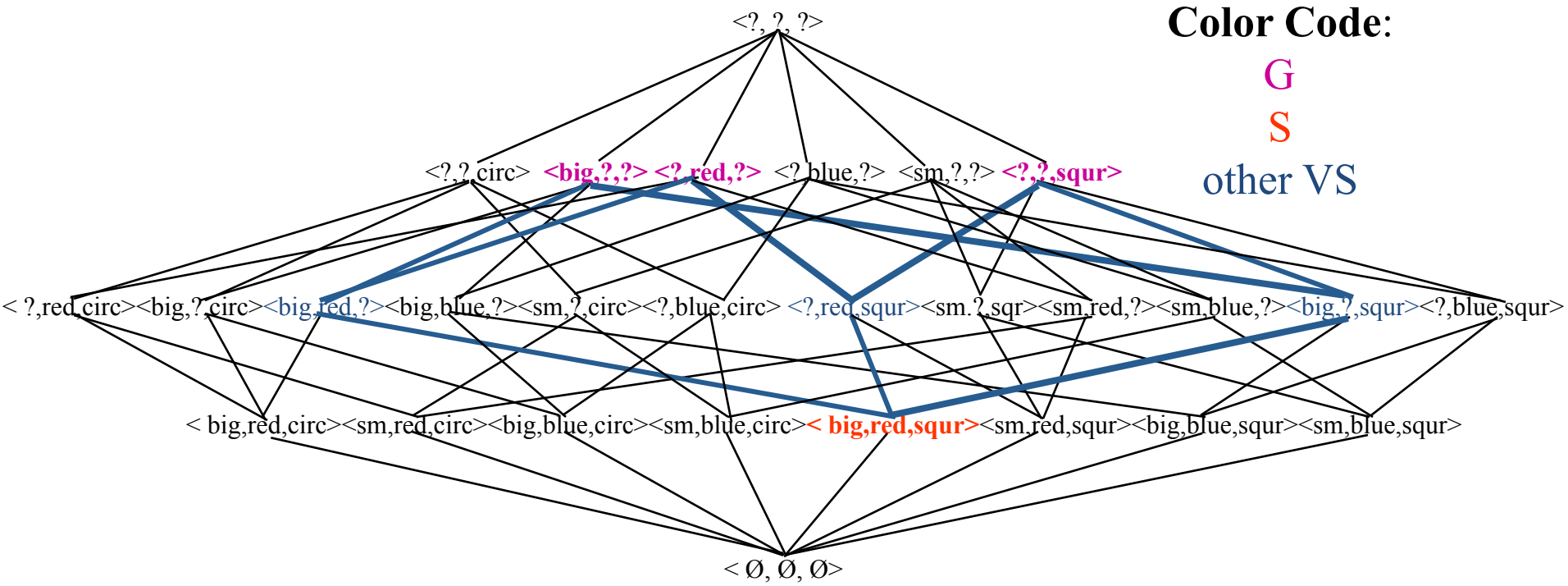
$$G = \{g \in H \mid Consistent(g, D) \wedge \neg \exists g' \in H[g' > g \wedge Consistent(s', D)]\}$$

- S and G represent the entire version space via its boundaries in the generalization lattice:



Version Space Lattice

Size: {sm, big} Color: {red, blue} Shape: {circ, squr}



<<big, red, squr>, 1>
<<sm, blue, circ>, 0>

Candidate Elimination (Version Space) Algorithm

Initialize G to the set of most-general hypotheses in H
Initialize S to the set of most-specific hypotheses in H
For each training example, d, do:
    If d is a positive example then:
        Remove from G any hypotheses that do not match d
        For each hypothesis s in S that does not match d
            Remove s from S
            Add to S all minimal generalizations, h, of s such that:
                1) h matches d
                2) some member of G is more general than h
        Remove from S any h that is more general than another hypothesis in S
    If d is a negative example then:
        Remove from S any hypotheses that match d
        For each hypothesis g in G that matches d
            Remove g from G
            Add to G all minimal specializations, h, of g such that:
                1) h does not match d
                2) some member of S is more specific than h
        Remove from G any h that is more specific than another hypothesis in G

Sample VS Trace

S= {< Ø, Ø, Ø>}; *G*= {<?, ?, ?>}
SIZE: (big,small) COLOR: (red, blue) SHAPE: (circ, square, triangle)

Positive: X: <big, red, circle>
Nothing to remove from *G (X is compatible with G, G would "accept" X)*
Minimal generalization of only *S* element is <big, red, circle> which is more specific than *G*.
S={<big, red, circle>}; *G*={<?, ?, ?>}

Negative: Y: <small, red, triangle>
Nothing to remove from *S*. (*Y is compatible with S={<big, red, circle>}, S would reject Y*)
Minimal specializations of <?, ?, ?> that would reject the negative example are: <big, ?, ?>, <?, blue, ?>, <?, ?, circle>, <?, ?, square> but some are **not more general than** some element of *S* hence **the final set G is <big, ?, ?>, <?, ?, circle>**
***S*={<big, red, circle>}; *G*={<big, ?, ?>, <?, ?, circle>}**

Sample VS Trace (cont)

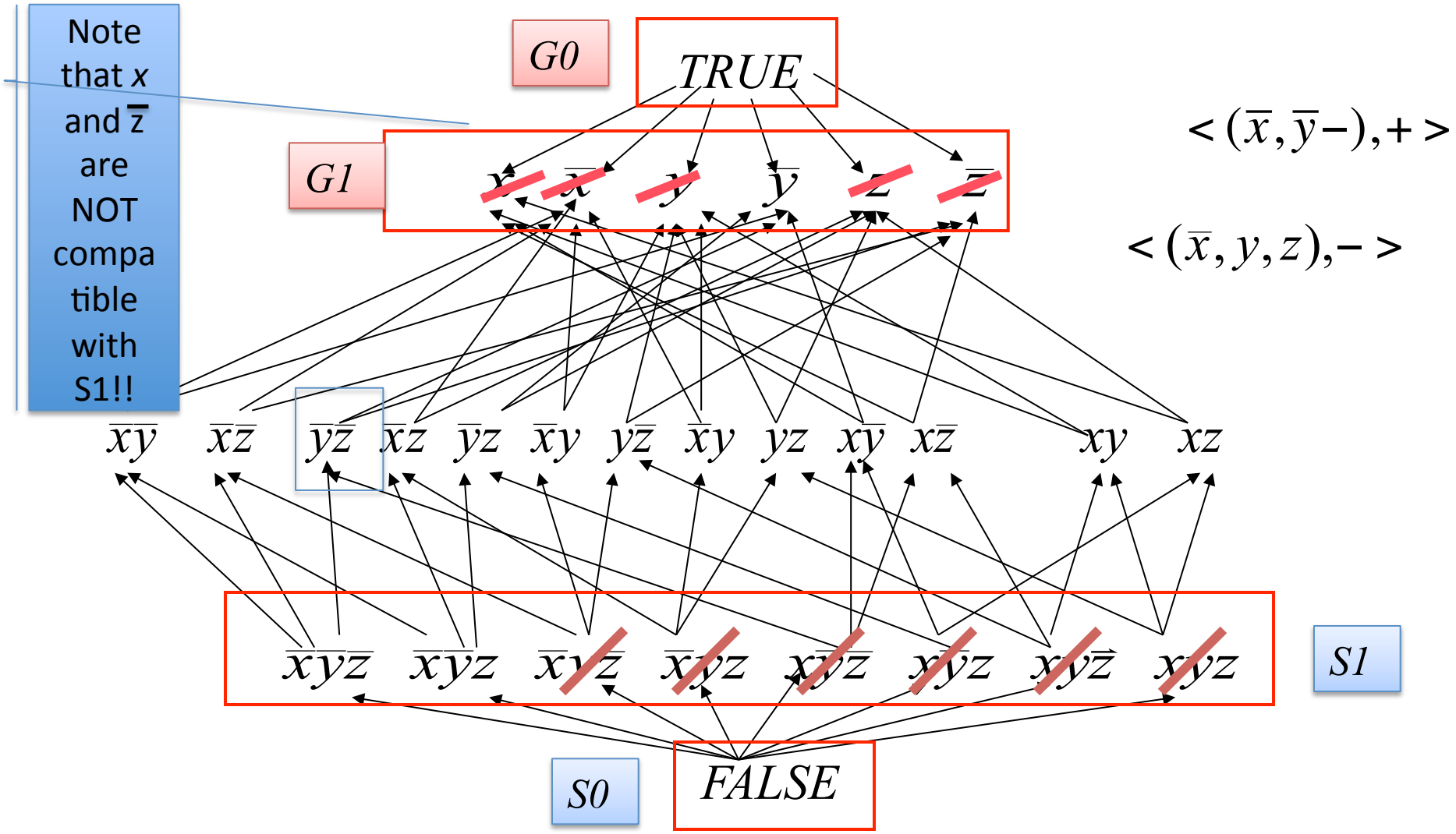
***S*={<big, red, circle>}; *G*={<big, ?, ?>, <?, ?, circle>}**

Positive: Z: <small, red, circle>
Remove <big, ?, ?> from *G (it would erroneously reject the example)*
Minimal generalization of S: <big, red, circle> that would accept the positive example is <?, red, circle>
***S*={<?, red, circle>}; *G*={<?, ?, circle>}**

Negative: N: <big, blue, circle>
Nothing to remove from *S (S would correctly reject the example)*
G: Minimal specializations of <?, ?, circle> that would reject the example are: <small, ? circle>, <?, red, circle>, but one is not more general than some element of *S*.
***S*={<?, red, circle>}; *G*={<?, red, circle>}**

S=*G*; Converged!

Example 2 (a generic 3-monomial conjunctive function)



Properties of VS Algorithm

- *S* summarizes the relevant information in the positive examples (relative to *H*) so that positive examples do not need to be retained.
- *G* summarizes the relevant information in the negative examples, so that negative examples do not need to be retained.
- Result **is not affected by the order in which examples are processes** but computational efficiency may.
- Positive examples move the *S* boundary up; Negative examples move the *G* boundary down.
- If *S* and *G* converge to the same hypothesis, then **it is the only one in *H* that is consistent with the data.**
- If *S* and *G* become empty (if one does the other must also) then **there is no hypothesis in *H* consistent with the data.**

Correctness of Learning

- Since the entire version space is maintained, given a continuous stream of noise-free training examples, the VS algorithm will eventually converge to the correct target concept if it is in the hypothesis space, H, or eventually correctly determine that it is not in H.
- Convergence is correctly indicated when S=G.

Computational Complexity of VS

- Computing the *S* set for conjunctive feature vectors **is linear in the number of features** and the number of training examples.
- Computing the *G* set for conjunctive feature vectors **is exponential in the number of training examples in the worst case**.
- In more expressive languages (than conjunctive rules), both *S* and *G* can grow exponentially.
- The order in which examples are processed can significantly affect computational complexity.

Before we start presenting new (and more practical) ML algorithms..

A number of relevant issues that apply to any ML problem/algorithm

1. Feature selection and object representatzion
2. Best order to present training examples
3. Multiple categories

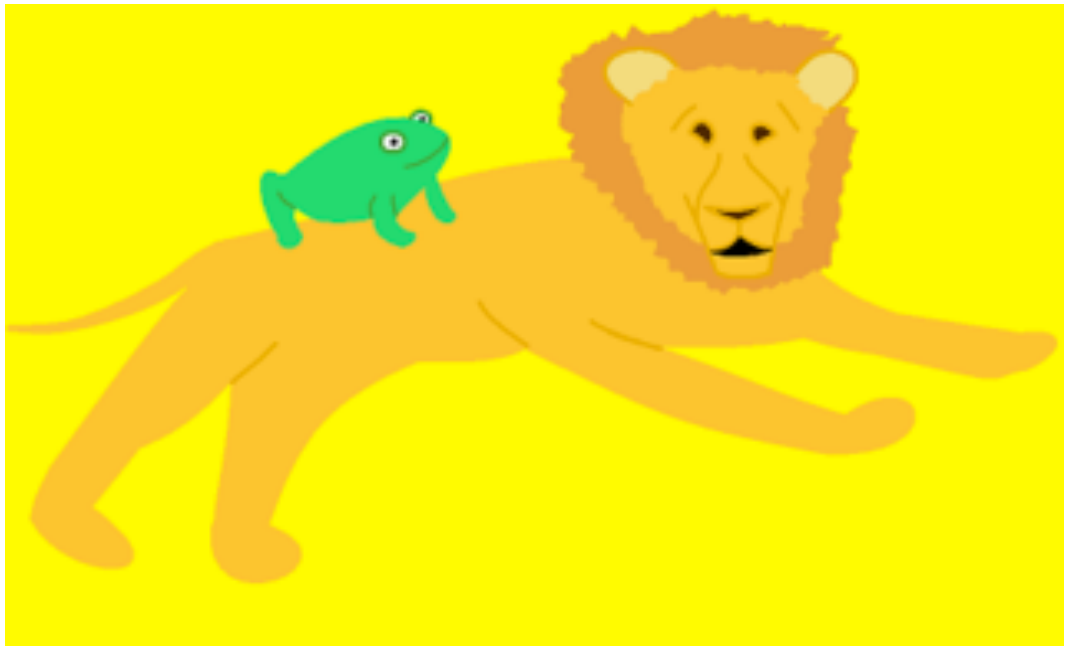
1. Feature Selection

- Many factors affect the success of machine learning on a given task.
- **The representation and quality of the example data is first and foremost**.
- In theory, having more features should result in more discriminating power.
- However, practical experience with machine learning algorithms has shown that this is not always the case.

The importance of features (attributes) selection

- Reduce the cost of learning by reducing the number of attributes.
- Provide better learning performance compared to using full attribute set.

Feature selection is **task dependent**



To classify lions and frogs in the appropriate category, a simple color histogram could perform very well:
f=color

Feature selection is task dependent

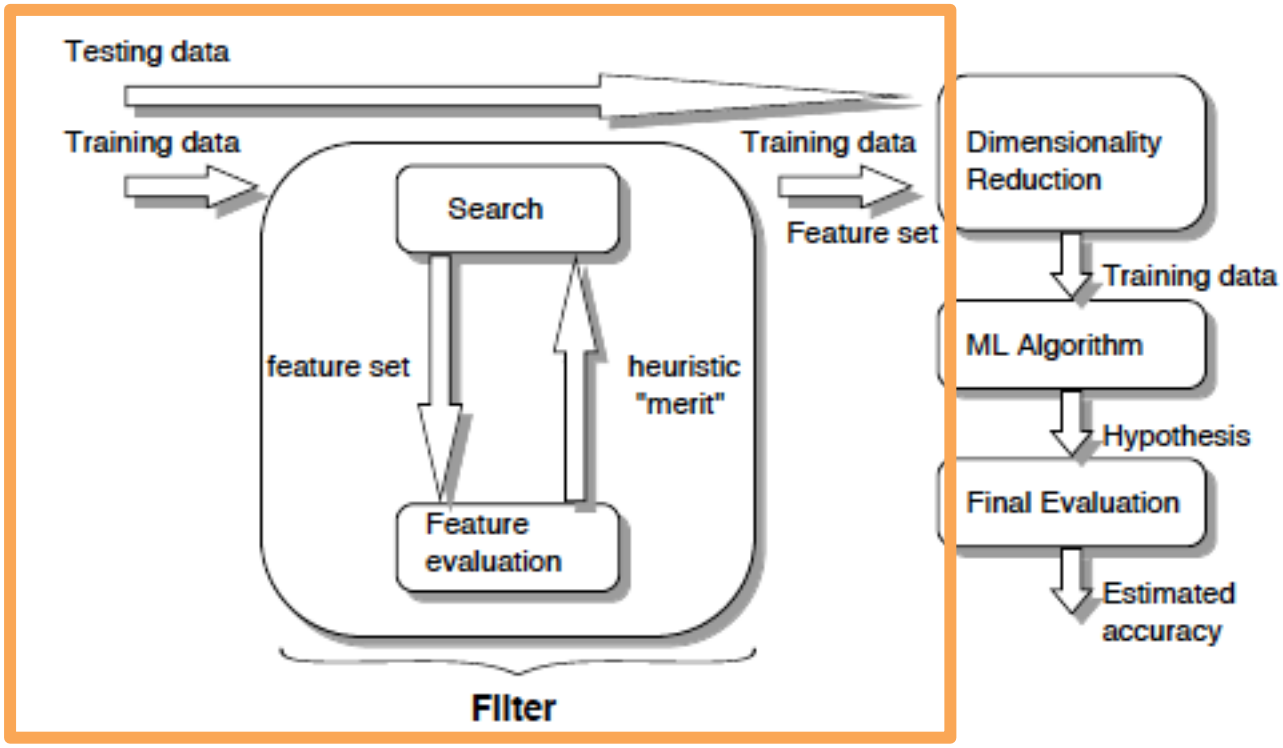
To classify horses and lions, more features are needed. Some would be useless, e.g Color, Number-of-legs..since they would not help to differentiate the 2 categories

Feature selection methods

There are two approach for attribute selection.

- Filter approach attempt to assess the merits of attributes from the data, ignoring learning algorithm.
- Wrapper approach the selection of a subset of attributes is done using the learning algorithm as a black box.

Filtering


Testing data
Training data
Search
feature set
heuristic
"merit"
Feature
evaluation
Filter
Training data
Feature set
Dimensionality
Reduction
Training data
ML Algorithm
Hypothesis
Final Evaluation
Estimated
accuracy

Filtering

- A feature $\mathbf{f}_i$ is said to be strongly relevant to the target concept(s) **if the probability distribution of the class values**, given the full feature set, changes when $\mathbf{f}_i$ is removed.

- A feature $\mathbf{f}_i$ is said to be weakly relevant if it is not strongly relevant and the probability distribution of the class values, given some subset **S** (containing $\mathbf{f}_i$) of the full feature set, does not changes when $\mathbf{f}_i$ is removed.

Example: want to learn *chair(x)* (chair(x)=true if x=chair)

x=(color, has-back,4-legs) (3 boolean features to describe instances)

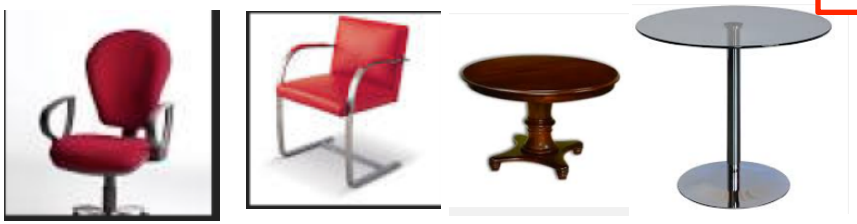
Prior probability of instances in D
P(chair)=8/16=0,5
P(table)=P(not(chair))=8/16=0,5
So we have the maximum uncertainty.
If we classify at random, our probability of error is 50%

Consider the feature:
4-legs (boolean)

Example

- If we group instances according to the 4-legs feature, we have :

4-legs=true



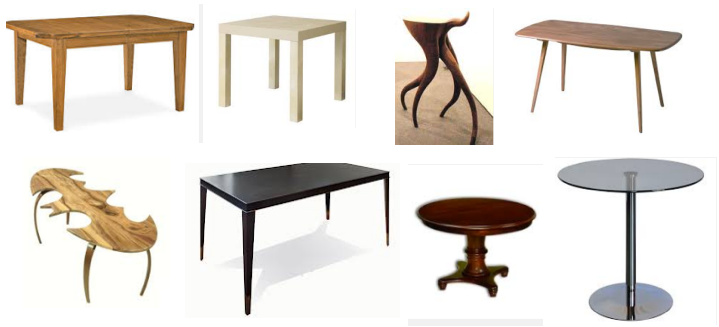
4-legs=false

Grouping instances according to 4-legs does not vary the probability distribution of the 2 categories: it remains 0,5!!

What about other features?

- *"4-legs"* is not a good feature to correctly classify tables and chairs!! It leaves you with the same uncertainty (0.5)
- Instead "*has-back*" would be enough to perfectly separate the 2 categories (given the trainig set of instances of the example)
- In fact *has-back=true* would group only chairs, *has-back=false* would group only tables)

Has_back=yes



Has_back=no

Are we sure?

- So, given our data, we could use the simple rule: **IF has-back = yes THEN chair(x)=YES**
- **But what about "unseen" instances?**

This was simple example..

- In previous example, one feature was useless, the other was 100% useful (could use that single feature to decide the correct class)
- In general, the problem is to automatically analyze all features and order them according to the REDUCTION OF UNCERTAINTY we get when grouping our dataset according to the values of each instance

Measuring the "probability distribution": Entropy filtering

- Ranking according to entropy gain of attributes.
- Entropy for given set of data with 2 classes can be defined as

$$Entropy = -\sum_{1}^{2} p(j) log_2 p(j)$$

- p(j) is the probability of class j, estimated by the relativefrequency of elements classified as "j" in the training set

Example

- Learning set D includes 5 istances, 2 classified as positives 3 negatives
- Probability of positive (estimate) p+=2/5
- Probability of negative (estimate) p-=3/5
- Entropy of the training set is:
- $E(D)=-(2/5)\log_2(2/5)-(3/5)\log_2(3/5)$ =
  (0.4)x1.32 + (0.6)x0.737= 0.528+0.44=0.97
- Note: if instances in D are equally distributed between positive and negative, Entropy is 1
- E(D)= -0.5log(0.5)-0.5log(0.5)=0.5+0.5=1
- If all instances are classified THE SAME 8all are positive or all are negative) Entropy is 0
- E(D)= -1log(1) =0

Entropy Filtering

After classification using one specific feature j, we can calculate the **gain of feature j:**

$$GAIN = Entropy - (\sum_{1}^{2} \frac{n_i}{n} Entropy(i))$$

- where n_i/n is an estimate of the probability that the considered feature has value i. n_i is the number of instances in the learning set D having the value of feature *j* equal to the value i (e.g. for the chairs case, there are 8/16 instances with feature *4-legs=yes*, and 8/16 with *4-legs=no*)
- Entropy(i) is the entropy of the subset Di with f_j=i
- Larger value of gain better attribute.

Wrt previous example of tables and chairs

$$GAIN = Entropy - (\sum_{1}^{2} \frac{n_i}{n} Entropy(i))$$

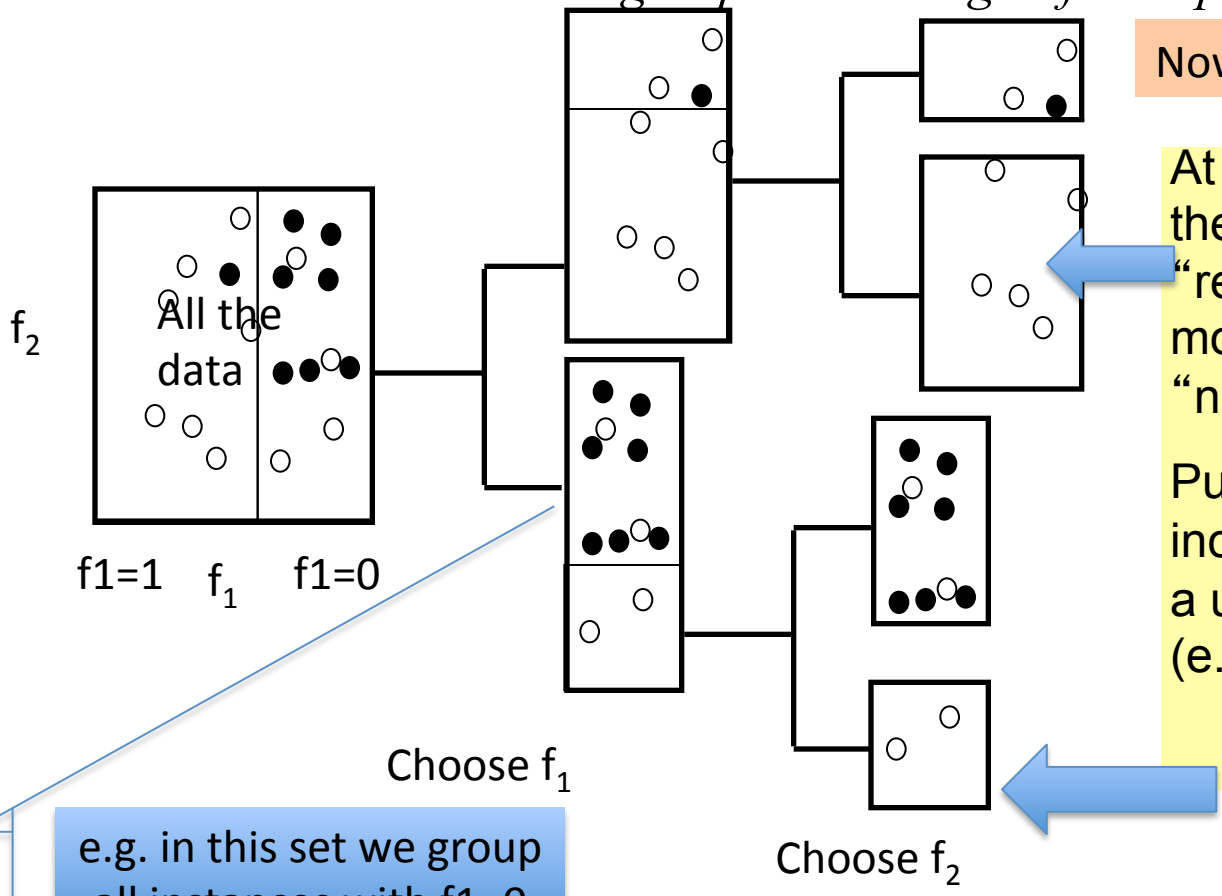
- E=-0.5log$_2$(0.5) -0.5log$_2$(0.5)= **1** (initial entropy, since there are 50% tables, 50% chairs)
- If we classify according to feature *4-legs*:
  - E(4-legs=true)=1 since out of 12 objects with 4 legs, we have 6 chairs and 6 tables
  - E(4-legs=false)=1 since out of 4 objects without 4 legs, we have 2 chairs and 2 tables
  - GAIN=1-((12/16) E(4-legs=true) +(4/16) E(4-legs=false))=**1**-1=0 **NO GAIN!!**
- If we instead classify according to has-back:
  - E(has-back=true)=0 (they are all chairs)
  - E(has-back=false)=0 (they are all tables)
  - Hence GAIN=**1**-0=1

In previous example we had two extremes: perfectly useless (4-legs) and perfectly useful (has-back) features

Common case is that entropy changes, but not from 1 to 0!!

Ex. black instances positive, white are negative

Partition the data set in two groups according to f1=1: positive and negative



Now we partition according to f2

At each step, choose the feature that "reduces entropy" most. Work towards "node purity".

Pure nodes are those including instances with a unique classification (e.g. all tables)

e.g. in this set we group all instances with f1=0

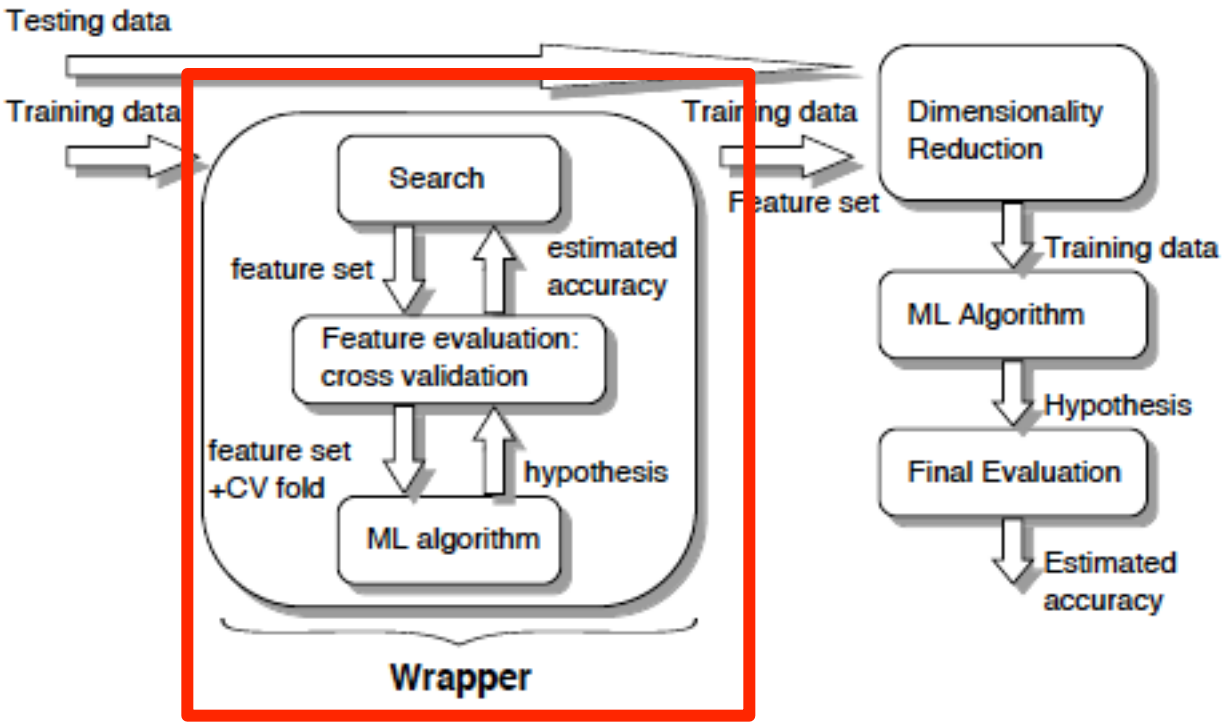
Summary on filtering

- Filtering is a method to order features according to relevance
- The Entropy filtering order the features according to information gain
- Perfect features are those with gain 1, useless features are those with gain 0, most features are somewhere in between
- Note that filtering, as we said initially, only looks at the distribution of feature values in the dataset, NOT at the machine learning algorithm

Wrappers

- Employs the target learning algorithm to evaluate feature sets
- Uses an induction algorithm along with a statistical re-sampling technique such as cross-validation to estimate the final accuracy of feature subsets

Wrappers


Testing data
Training data
Search
feature set
estimated accuracy
Feature evaluation: cross validation
feature set +CV fold
hypothesis
ML algorithm
Wrapper
Training data
Feature set
Dimensionality Reduction
Training data
ML Algorithm
Hypothesis
Final Evaluation
Estimated accuracy

Wrappers

Say we have features A, B, C and classifier M. We want to predict C(X) given the smallest possible subset of {A,B,C}, while achieving maximal performance (accuracy)

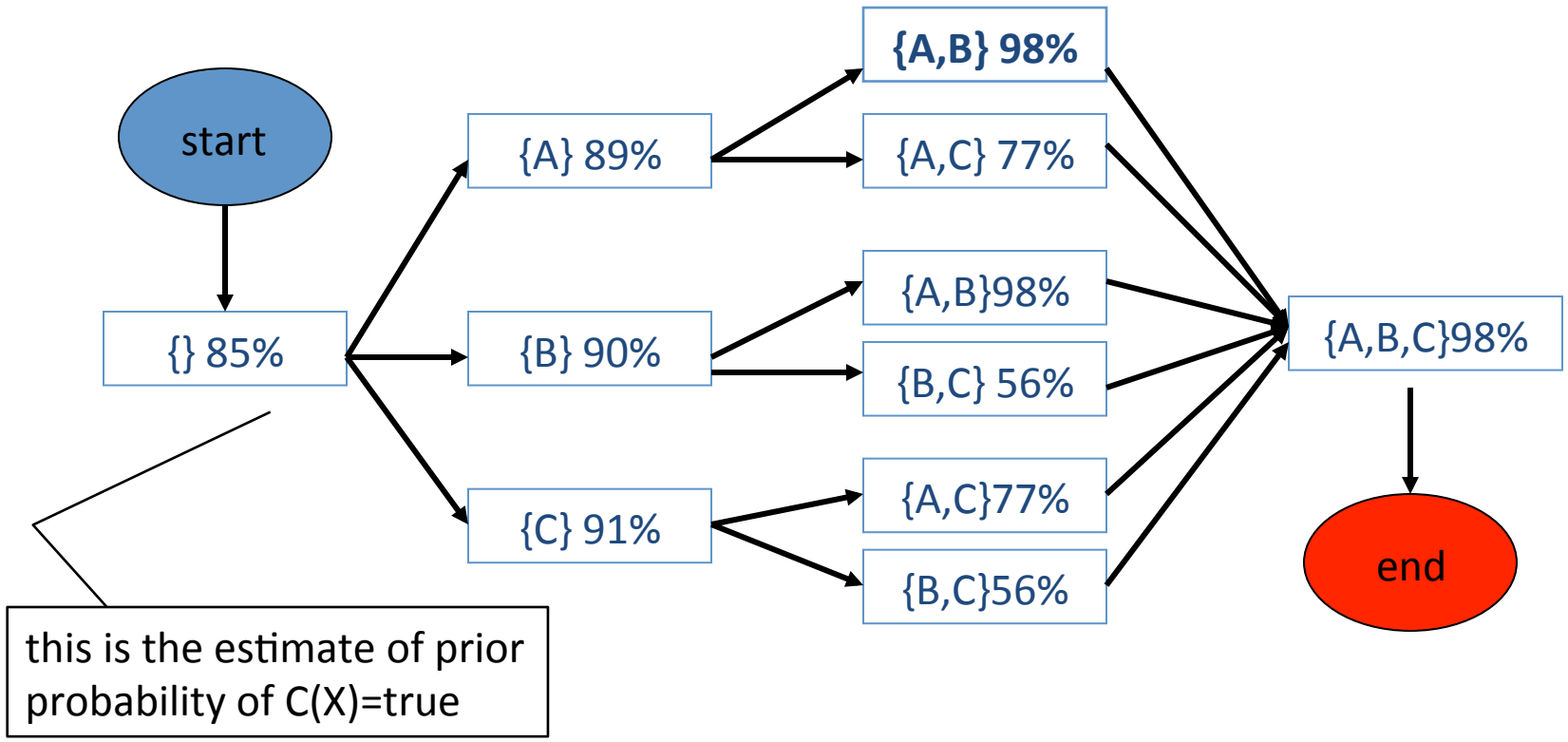
FEATURE SET	CLASSIFIER	PERFORMANCE
{A,B,C}	M	**98%**
{A,B}	M	**98%**
{A,C}	M	77%
{B,C}	M	56%
{A}	M	89%
{B}	M	90%
{C}	M	91%
{.}	M	85%

The third column is the number of correctly classified istances by the classifier M when using the set of features in column1

When zero features are used, the classifier performs a random guess. In this case, P=85% means that the prior probability of C(X)=true is 0,85

Wrappers (2)

The set of all subsets is **the power set and its size is $2^{|V|}$** (V= number of features). Hence for large V **we cannot do this procedure exhaustively**; instead we rely on **heuristic search** of the space of all possible feature subsets.



Wrappers: Hill climbing

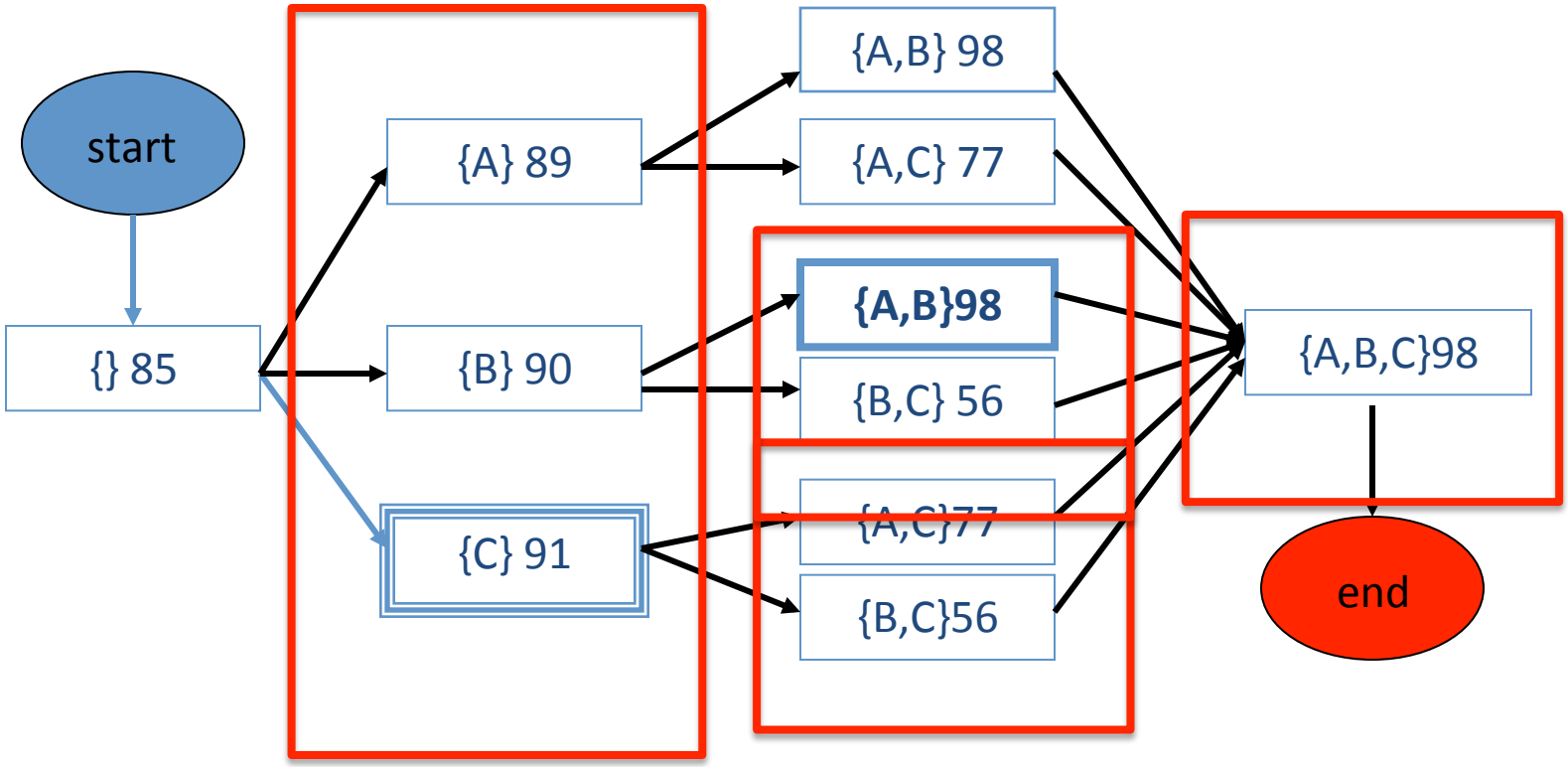
A common example of heuristic search is hill climbing: keep adding features one at a time until no further improvement can be achieved.

1. Let $v \leftarrow$ initial state of attributes.
2. Expand v: find v's children.
3. Apply the evaluation function f to each child w of v.
4. Let $v' =$ the child w with highest evaluation $f(w)$
5. If $f(v') > f(v)$ then $v \leftarrow v'$; go to 2
6. Return v

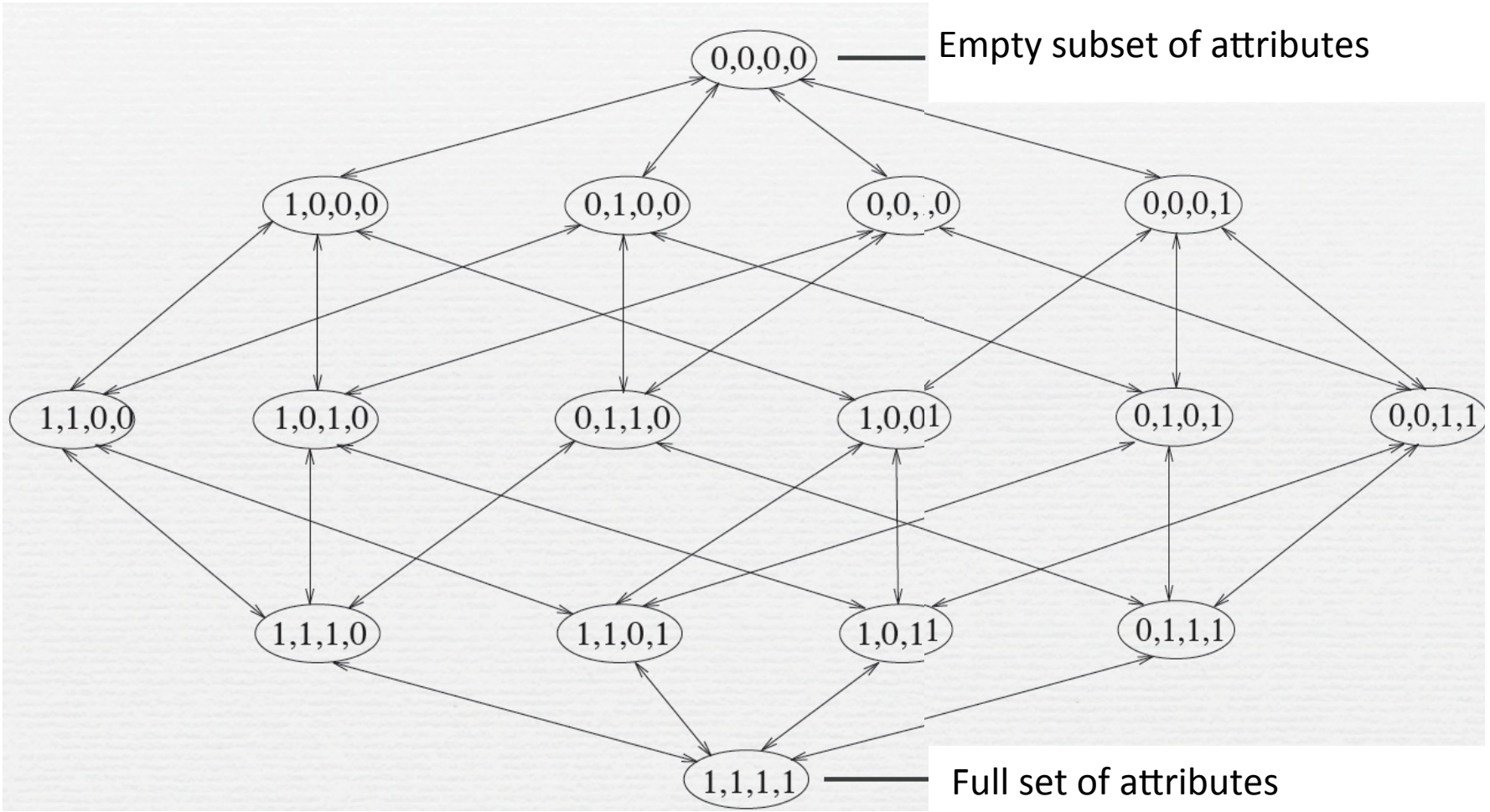
We start with an empty set of attributes. At each step k, we consider all possible $\binom{v}{k}$ combinations of k attributes. *f(v)* is any evaluation function (e.g., precision)

Hill Climbing

Start with empty set of attributes
Step 1: with first expansion (add one attribute in each node) f(v')>f(v) for all nodes
Step 2: node C is expanded first (since had highest perf), but condition is not met, so backtrack to second best (B)
Step 3: B is expanded and nodes A,B and B,C are generated; A,B is the best
~~Step 4: A,B is expanded but condition is not met; OUTPUT is~~ ***~~{A,B}~~***



Ex: Attribute tree with 4 binary attributes (= features)



NOTE: terms “ATTRIBUTE” and “FEATURE” can be used interchangeably. Same meaning!

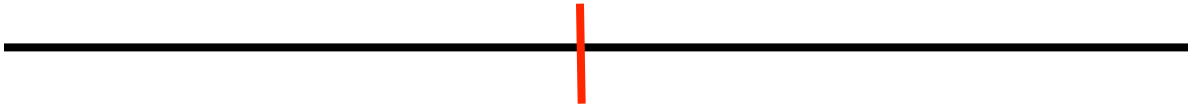
A number of relevant issues that apply to any ML problem/algorithm

1. Feature selection and object representation √
2. **Best order to present training examples**
3. Multiple categories

Best way to select examples: Active Learning

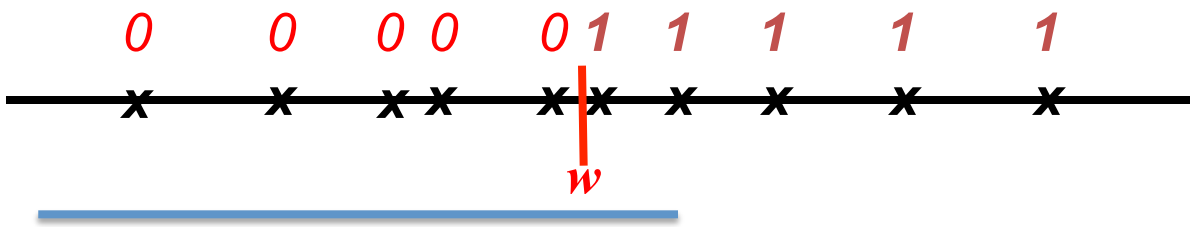
- In ***active learning***, the system is responsible for selecting good training examples and asking a teacher (oracle) to provide a class label.
- Goal is to **minimize the number of examples required to learn an accurate concept description**.
- The general idea is to select the training examples in a "smart" way
- Note: only applicable if we have lots of examples, target is to avoid loosing time with examples which are less useful (e.g. too similar to already analyzed ones), OR if we need the help of an expert to label examples in the appropriate class (experts have a cost!)

Toy Example: 1D classifier



- Objective: learning a threshold function (*if x>w then true else false*). The threshold *w* is unknown .. It is what we want to learn
- Example: we want to learn the body mass index above which there is a high risk of diabetes
- To train the system we need to "label" a training set (we need to identify people with and without diabetes, and then measure their BMI).
- The number of examples N depends on the number of possible values of x (possibly discretized). For the BMI, values go from 10 to 40 (N=30).

Toy Example: 1D classifier



Goal: given the training set, find transition between 0 and 1 labels (e.g. find w) in minimum steps

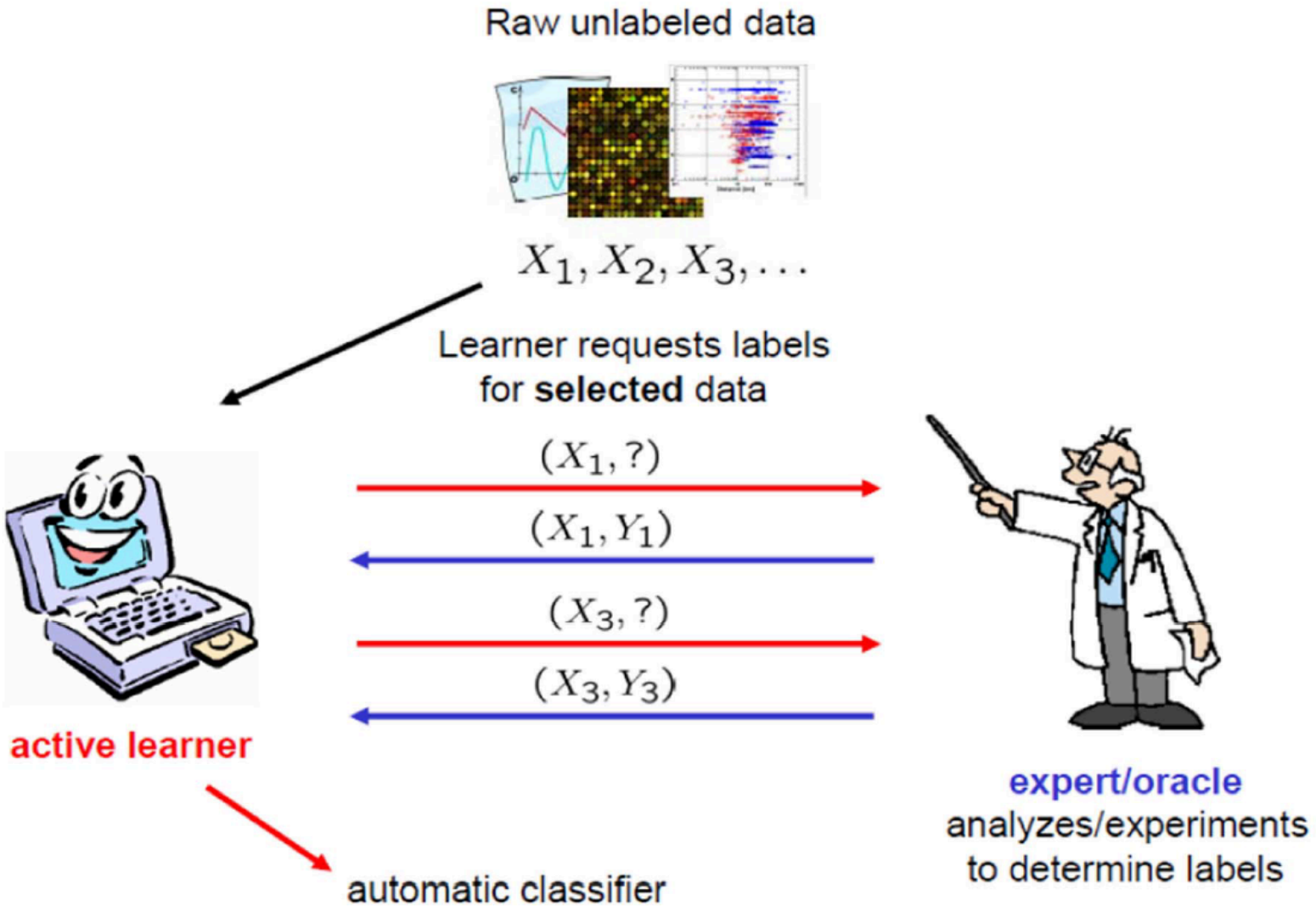
Naïve method: choose points to label at random on line (choose patients from a record database at random)

- *Requires O(N) training data to find underlying classifier*

Better method: binary search for transition between 0 and 1

- *Requires O(log N) training data to find underlying classifier*
- *Exponential reduction in training data size!*

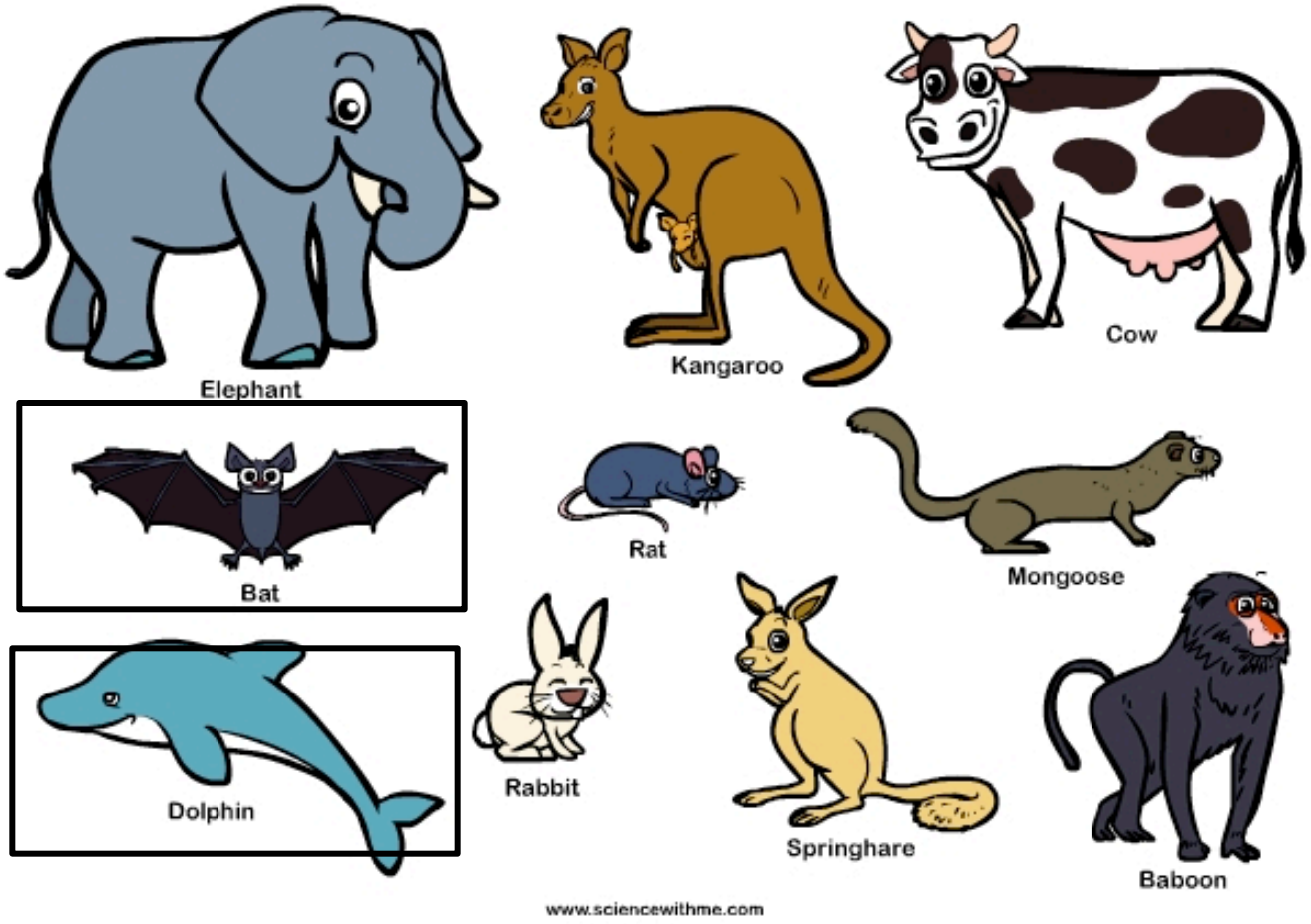
Active learning: choose labeled examples



Active learning strategies

- Uncertainty sampling: label those points for which the current model is **least certain** as to what the correct output should be (if model is probabilistic)
- Query by committee: a variety of models are trained on the current labeled data, and vote on the output for unlabeled data; **label those points for which the "committee" disagrees the most (most complex cases)**
- Expected model change: label those points that would **most change the current model**
- Expected error reduction: label those points that would **most reduce the model's generalization error**
- Variance reduction: label those points that **would minimize output variance**, which is one of the components of error

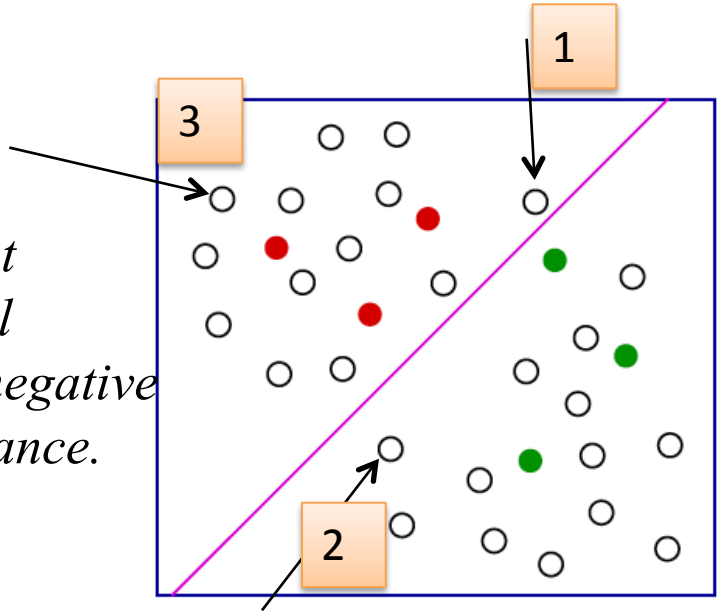
A common method is: take the example that “mostly differs” from those seen so far



Typical heuristics for active learning

- Start with a pool of unlabeled data
- Pick a few points at random and get their labels
- Repeat
  - Pick the unlabeled point that is closest to the boundary

  (or most uncertain, or most likely to decrease overall uncertainty,...)



Which of the following instances would be an interesting example?

Knowing the label of the first two instances is more helpful at determining the positive/negative boundary than the third instance.

A number of relevant issues that apply to any ML problem/algorithm

1. Feature selection and object representatzion√
2. Best order to present training examples √
3. **Multiple categories**

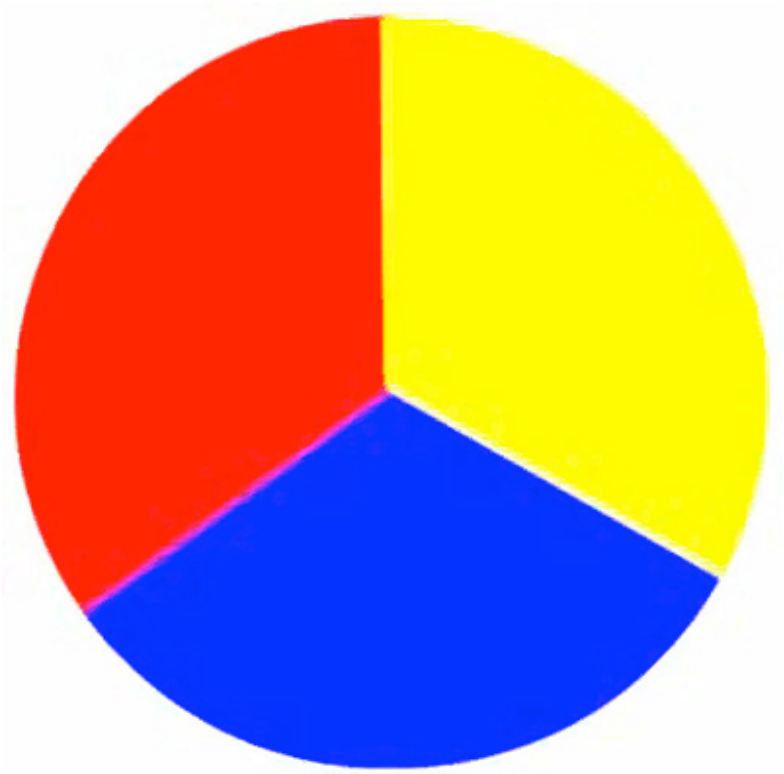
Learning for Multiple Categories

- What if the classification problem is not concept learning and involves **more than two categories** (e.g. select among several possible illnesses, given the symptoms)?
- Can treat as a **series of concept learning problems (e.g. we use n independent classifiers)**, where for each classifier C_i, all

$$x \in D, \ s.t. \ c_i(x) = 1$$

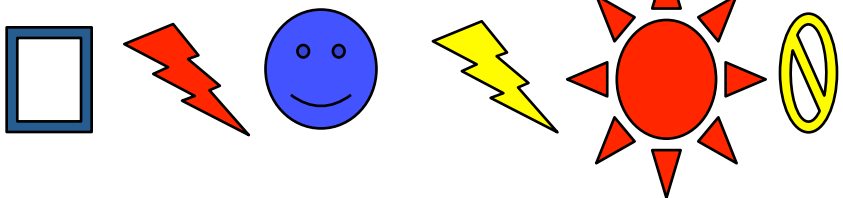
  are treated as positive and all other instances in categories $C_{j,}\ j \neq i$ are treated as negative (***one-versus-all***).
- This will assign a unique category to each training instance but may assign a novel instance to zero or multiple categories.
- If the binary classifier produces confidence estimates (e.g. based on **voting**), then a novel instance can be assigned to the category with the highest confidence.

Example



- Classifier 1: red or not-red
- Classifier 2: blue or not-blue
- Classifier 3: yellow or not-yellow

D



T