

# Lecture 11: Learning 1

[http://cs.nju.edu.cn/yuy/course\\_ai18.ashx](http://cs.nju.edu.cn/yuy/course_ai18.ashx)



# Previously...



## Search

Path-based search

Iterative improvement search

## Knowledge

Propositional Logic

First Order Logic (FOL)

## Uncertainty

Bayesian network

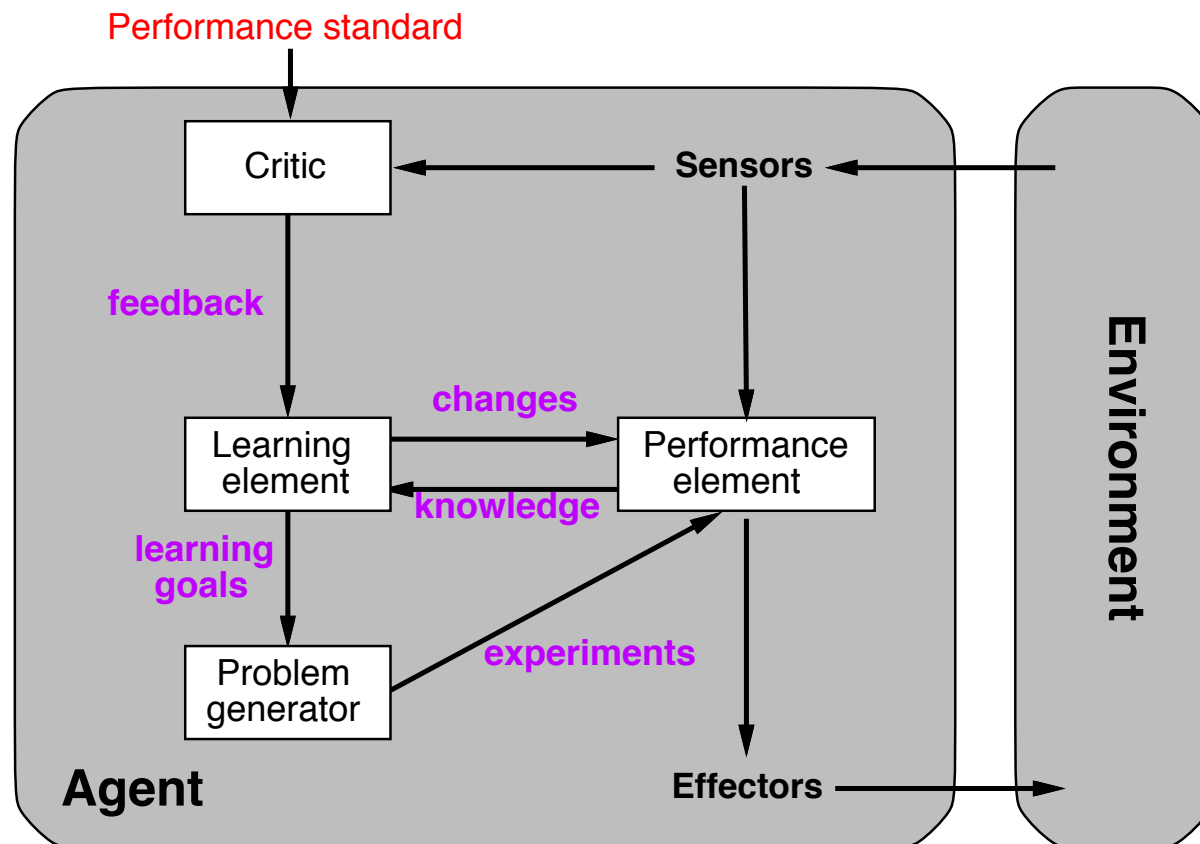
# Learning



Learning is essential for unknown environments,  
i.e., when designer lacks omniscience

Learning is useful as a system construction method,  
i.e., expose the agent to reality rather than trying to write it down

Learning modifies the agent's decision mechanisms to improve performance



# Inductive Learning



Simplest form: learn a function from examples (**tabula rasa**)

$f$  is the target function

An example is a pair  $x, f(x)$ , e.g., 

$O$	$O$	$X$
	$X$	
$X$		

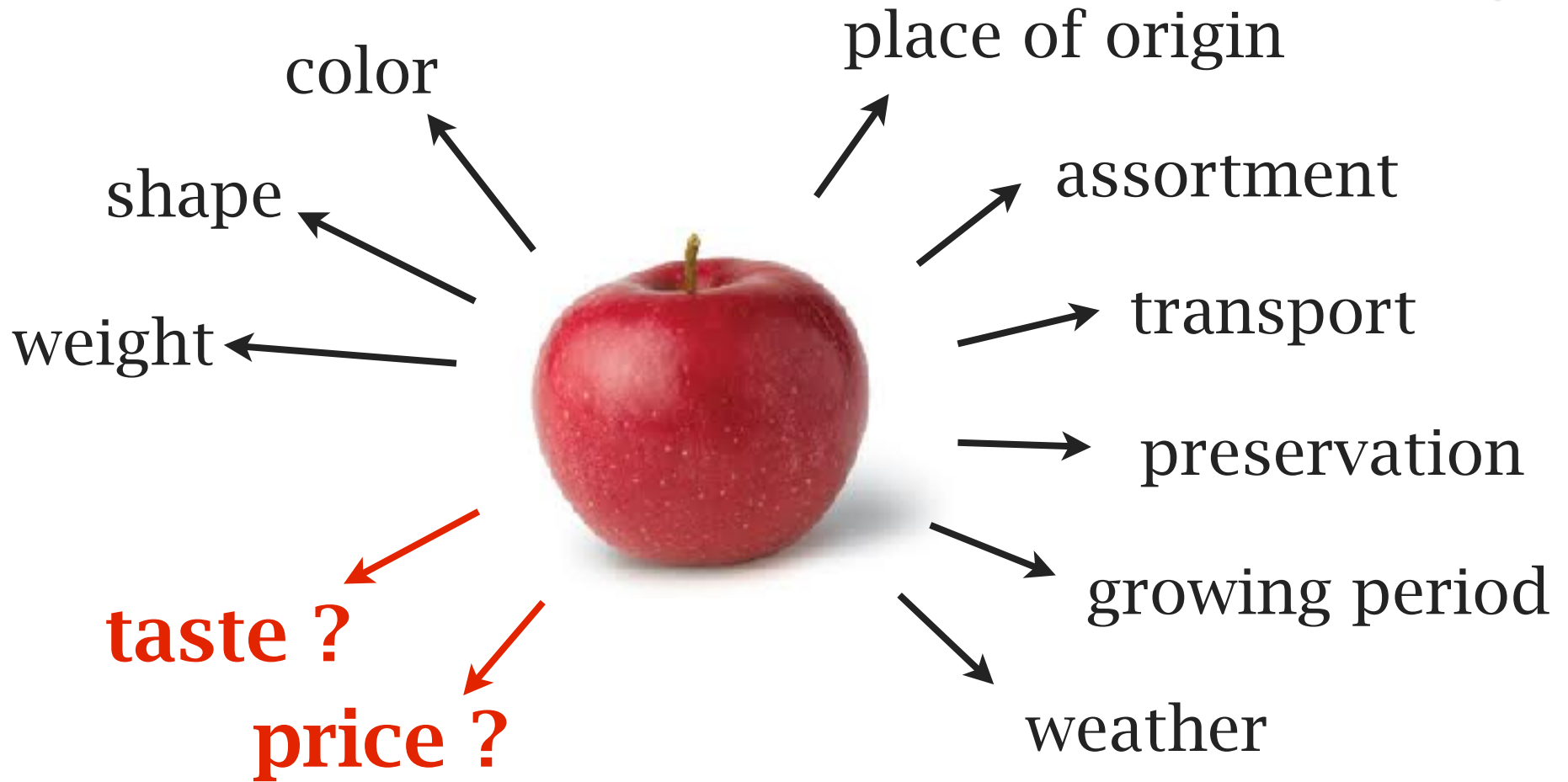
,  $+1$

Problem: find a(n) hypothesis  $h$   
such that  $h \approx f$   
given a training set of examples

(This is a highly simplified model of real learning:

- Ignores prior knowledge
- Assumes a deterministic, observable “environment”
- Assumes examples are given
- Assumes that the agent wants to learn  $f$ —why?)

# Attribute-based representations



# Attribute-based representations



Examples described by **attribute values** (Boolean, discrete, continuous, etc.)

E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>WillWait</i>
$X_1$	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>0-10</i>	<i>T</i>
$X_2$	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>30-60</i>	<i>F</i>
$X_3$	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>Some</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>0-10</i>	<i>T</i>
$X_4$	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>10-30</i>	<i>T</i>
$X_5$	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>&gt;60</i>	<i>F</i>
$X_6$	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Italian</i>	<i>0-10</i>	<i>T</i>
$X_7$	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>0-10</i>	<i>F</i>
$X_8$	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Thai</i>	<i>0-10</i>	<i>T</i>
$X_9$	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>&gt;60</i>	<i>F</i>
$X_{10}$	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>Italian</i>	<i>10-30</i>	<i>F</i>
$X_{11}$	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>0-10</i>	<i>F</i>
$X_{12}$	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>30-60</i>	<i>T</i>

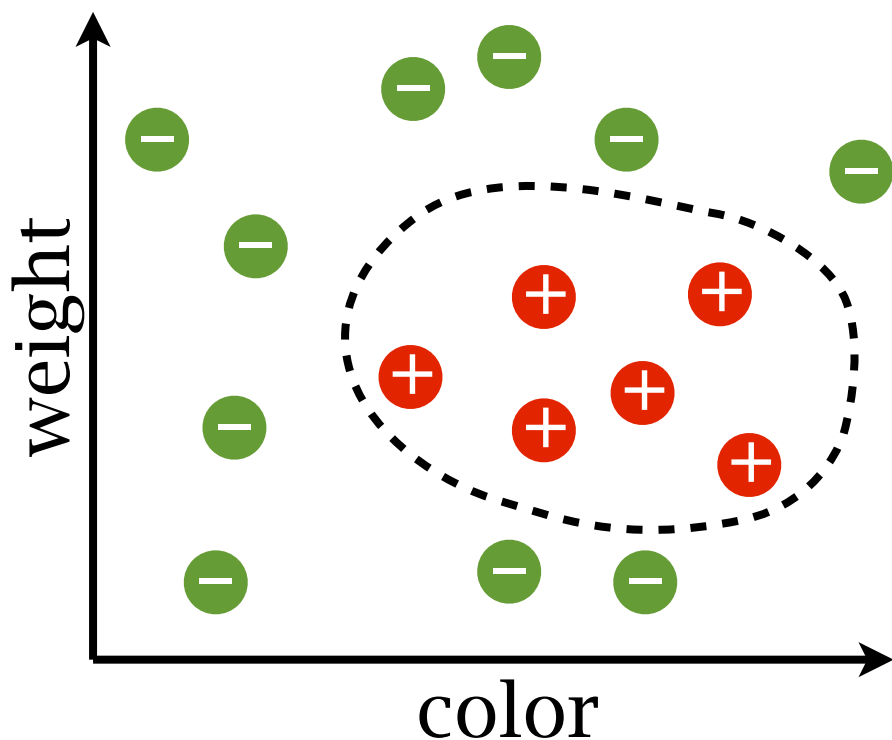
Classification of examples is **positive** (T) or **negative** (F)

# Learning task: Classification



**Features:** color, weight

**Label:** taste is sweet (positive/+) or not (negative/-)



(color, weight)  $\rightarrow$  sweet ?

$$\mathcal{X} \rightarrow \{-1, +1\}$$

ground-truth function  $f$

examples/training data:

$$\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$$

$$y_i = f(\mathbf{x}_i)$$

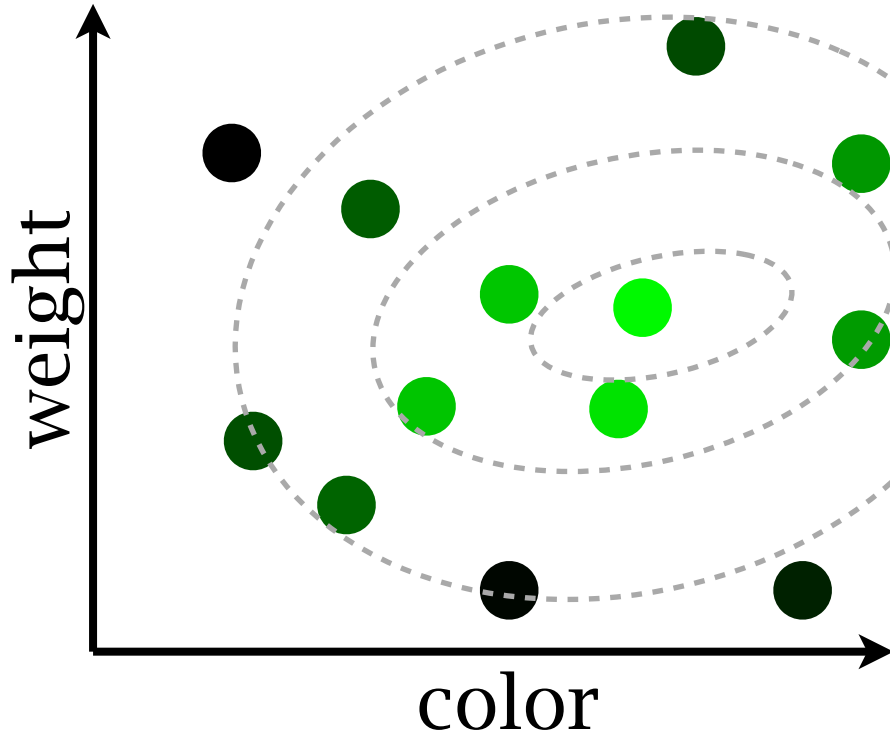
learning: find an  $f'$  that is close to  $f$



# Learning task: Regression

**Features:** color, weight

**Label:** price [0,1]



(color, weight)  $\rightarrow$  price

$\mathcal{X} \rightarrow [0, +1]$

ground-truth function  $f$

examples/training data:

$\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$

$y_i = f(\mathbf{x}_i)$

learning: find an  $f'$  that is close to  $f$

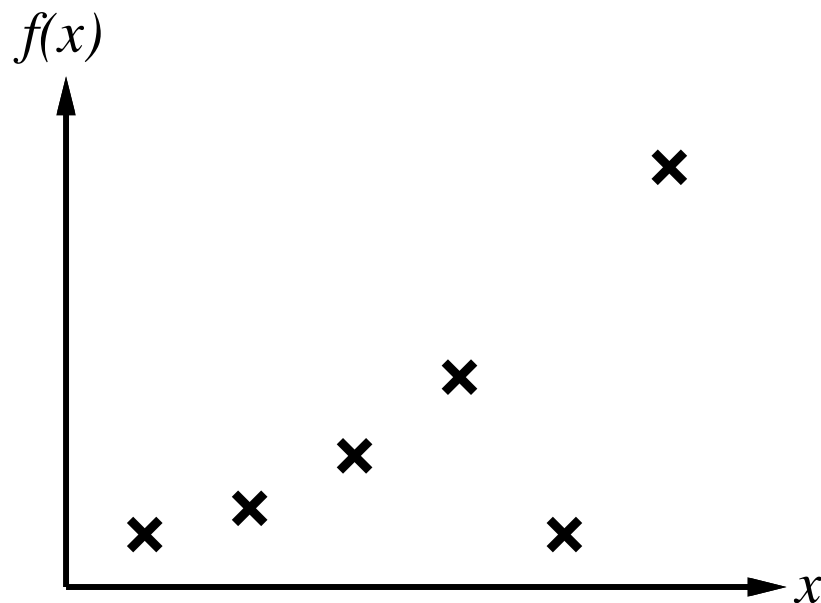


# Learning task: Regression



Construct/adjust  $h$  to agree with  $f$  on training set  
( $h$  is consistent if it agrees with  $f$  on all examples)

E.g., curve fitting:

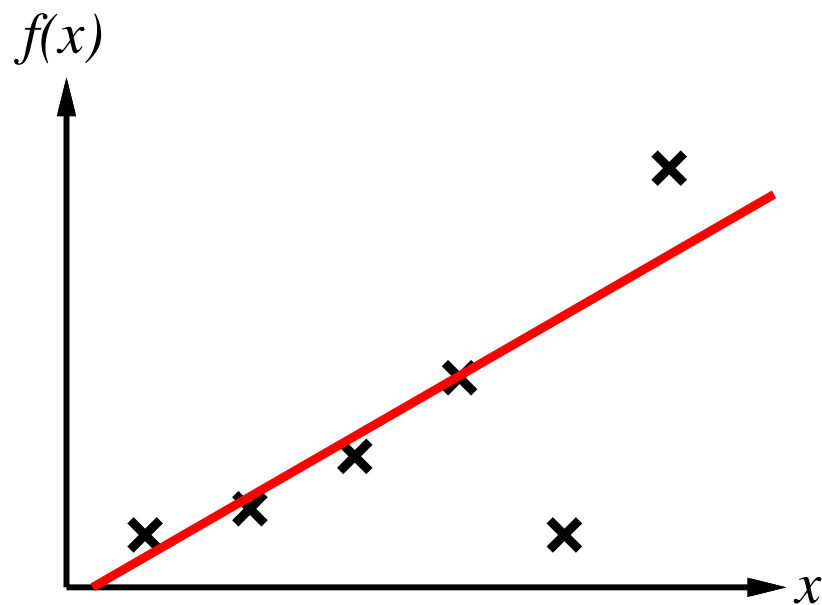


# Learning task: Regression



Construct/adjust  $h$  to agree with  $f$  on training set  
( $h$  is consistent if it agrees with  $f$  on all examples)

E.g., curve fitting:

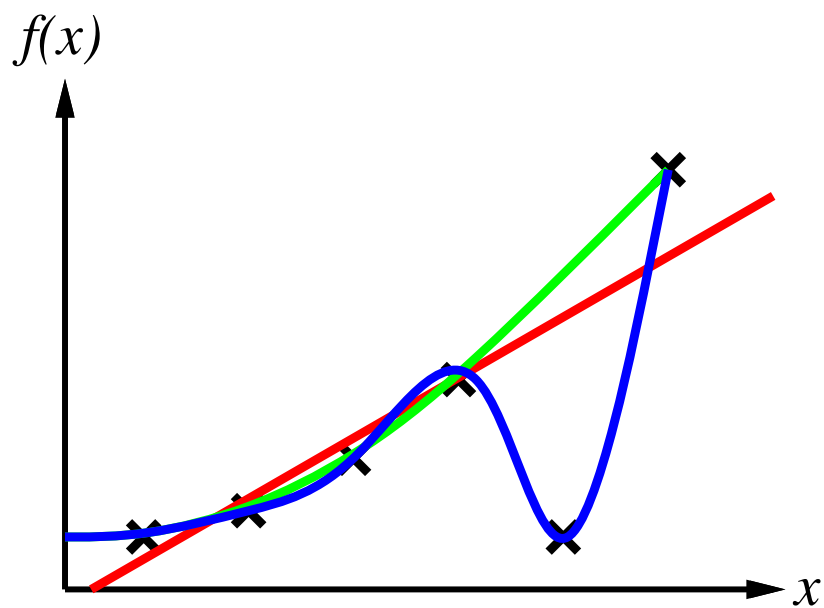


# Learning task: Regression



Construct/adjust  $h$  to agree with  $f$  on training set  
( $h$  is consistent if it agrees with  $f$  on all examples)

E.g., curve fitting:

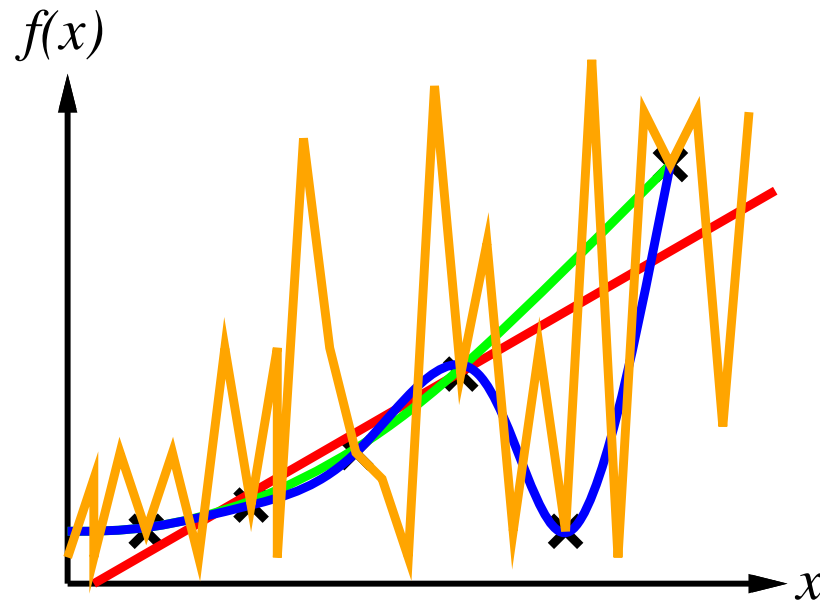


# Learning task: Regression



Construct/adjust  $h$  to agree with  $f$  on training set  
( $h$  is consistent if it agrees with  $f$  on all examples)

E.g., curve fitting:



*how to learn? why it can learn?*

# Learning algorithms



Decision tree

Neural networks

Linear classifiers

Bayesian classifiers

Lazy classifiers

...

Why different classifiers?

heuristics

viewpoint

performance

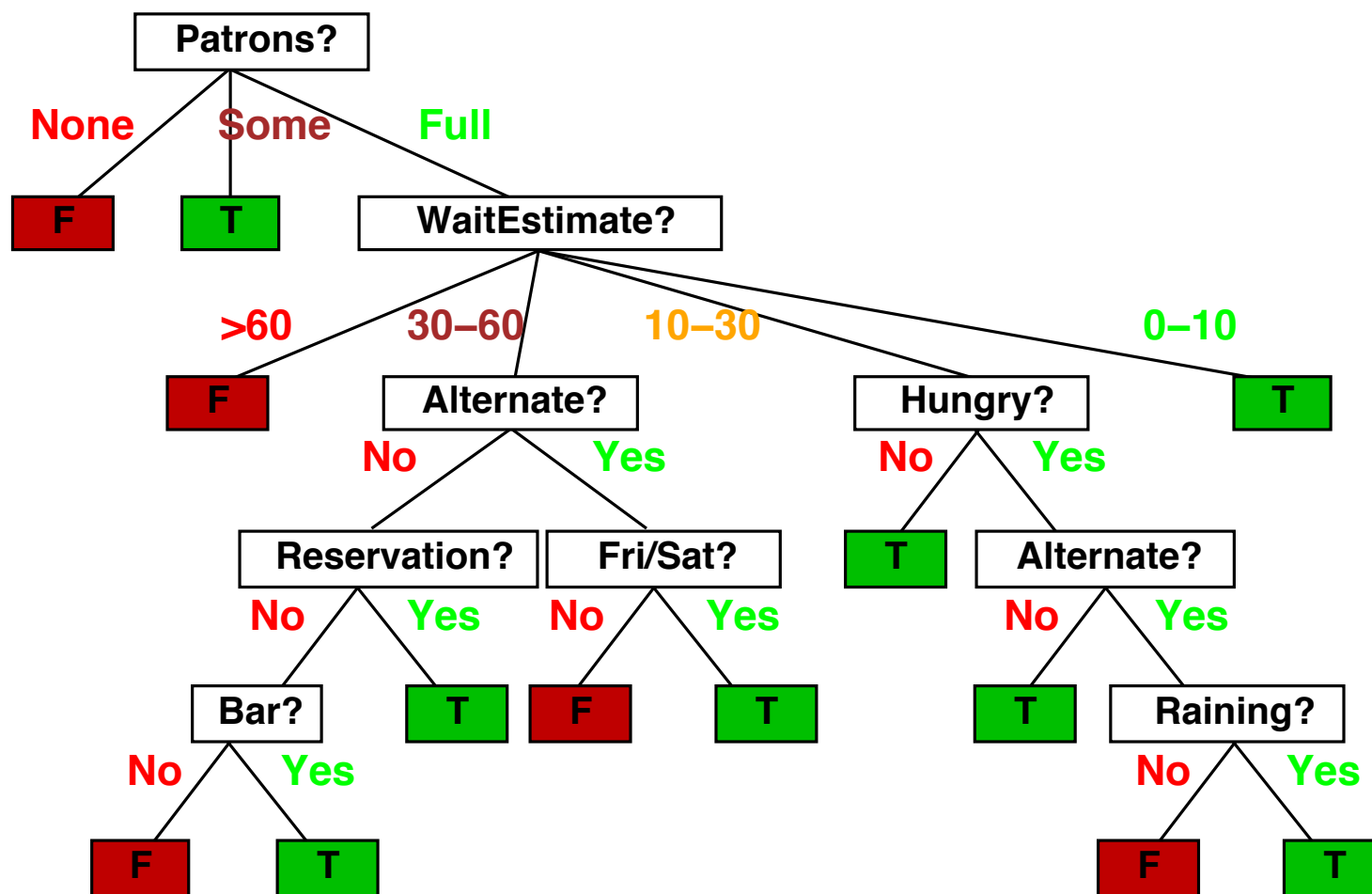
# Decision tree learning



## what is a decision tree

One possible representation for hypotheses

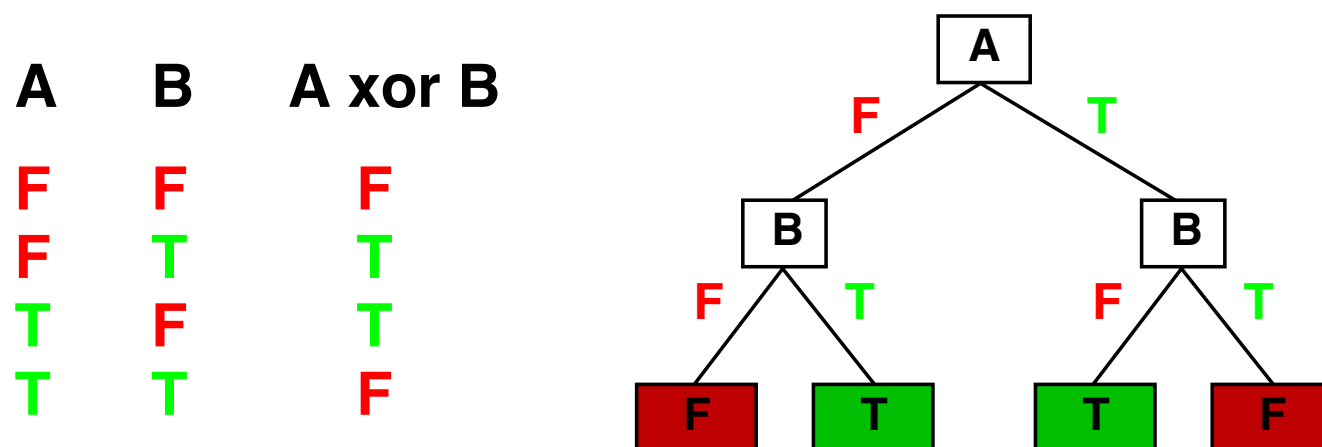
E.g., here is the “true” tree for deciding whether to wait:



# Expressiveness



Decision trees can express any function of the input attributes.  
E.g., for Boolean functions, truth table row  $\rightarrow$  path to leaf:



Trivially, there is a consistent decision tree for any training set  
w/ one path to leaf for each example (unless  $f$  nondeterministic in  $x$ )  
but it probably won't generalize to new examples

Prefer to find more **compact** decision trees

# Hypothesis spaces (all possible trees)



How many distinct decision trees with  $n$  Boolean attributes??

= number of Boolean functions

= number of distinct truth tables with  $2^n$  rows =  $2^{2^n}$


E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

How many purely conjunctive hypotheses (e.g.,  $Hungry \wedge \neg Rain$ )??

Each attribute can be in (positive), in (negative), or out

$\Rightarrow 3^n$  distinct conjunctive hypotheses

More expressive hypothesis space

– increases chance that target function can be expressed 

– increases number of hypotheses consistent w/ training set

$\Rightarrow$  may get worse predictions 



# Decision tree learning algorithm



Aim: find a small tree consistent with the training examples

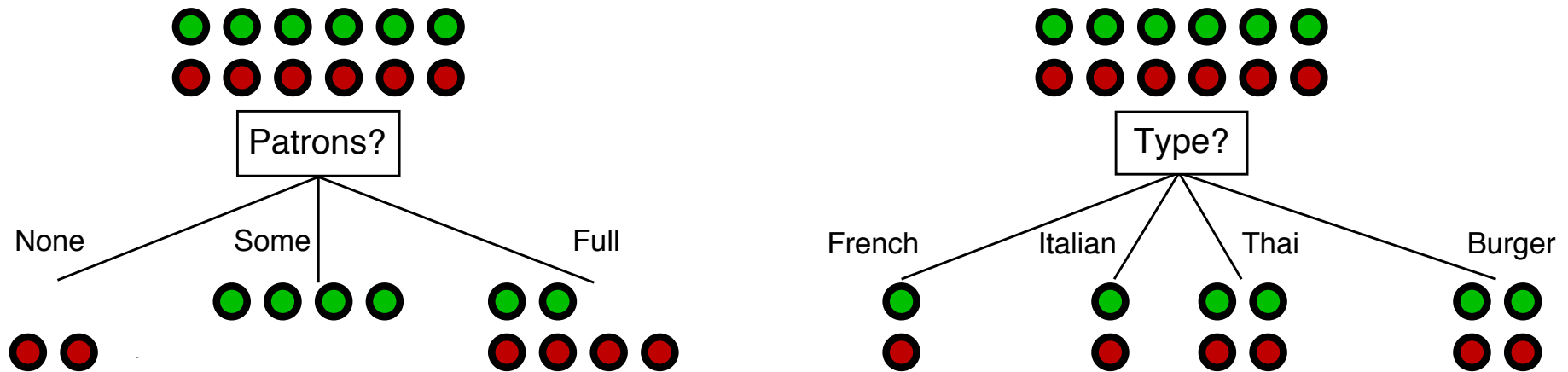
Idea: (recursively) choose “most significant” attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
    best ← CHOOSE-ATTRIBUTE(attributes, examples)
    tree ← a new decision tree with root test best
    for each value  $v_i$  of best do
      examplesi ← {elements of examples with  $best = v_i$ }
      subtree ← DTL(examplesi, attributes – best, MODE(examples))
      add a branch to tree with label  $v_i$  and subtree subtree
  return tree
```

# Choosing an attribute



Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”



*Patrons?* is a better choice—gives **information** about the classification

# Information



Information answers questions

The more clueless I am about the answer initially, the more information is contained in the answer

Scale: 1 bit = answer to Boolean question with prior  $\langle 0.5, 0.5 \rangle$

Information in an answer when prior is  $\langle P_1, \dots, P_n \rangle$  is

$$H(\langle P_1, \dots, P_n \rangle) = \sum_{i=1}^n -P_i \log_2 P_i$$

(also called **entropy** of the prior)

# Information



Suppose we have  $p$  positive and  $n$  negative examples at the root

$\Rightarrow H(\langle p/(p+n), n/(p+n) \rangle)$  bits needed to classify a new example

E.g., for 12 restaurant examples,  $p = n = 6$  so we need 1 bit

An attribute splits the examples  $E$  into subsets  $E_i$ , each of which (we hope) needs less information to complete the classification

Let  $E_i$  have  $p_i$  positive and  $n_i$  negative examples

$\Rightarrow H(\langle p_i/(p_i+n_i), n_i/(p_i+n_i) \rangle)$  bits needed to classify a new example

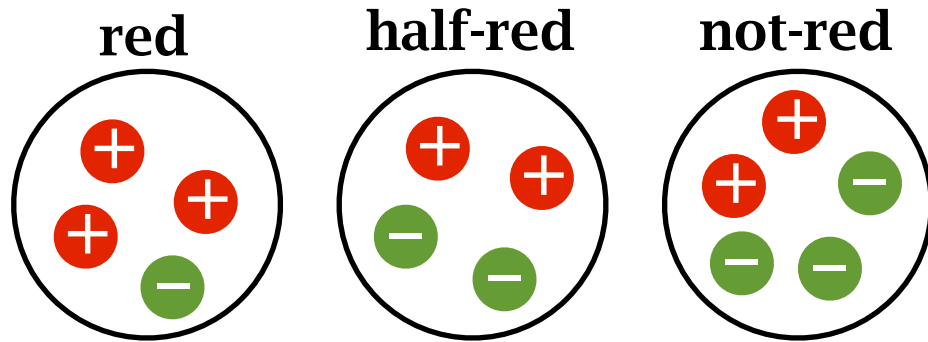
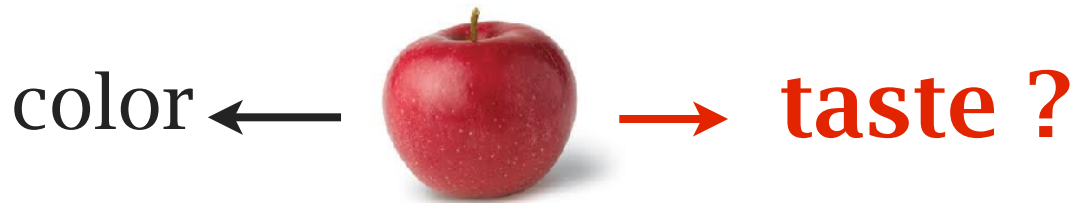
$\Rightarrow$  **expected** number of bits per example over all branches is

$$\sum_i \frac{p_i + n_i}{p + n} H(\langle p_i/(p_i + n_i), n_i/(p_i + n_i) \rangle)$$

For *Patrons?*, this is 0.459 bits, for *Type* this is (still) 1 bit

$\Rightarrow$  choose the attribute that minimizes the remaining information needed

# Example



id	color	taste
1	red	sweet
2	red	sweet
3	half-red	sweet
4	not-red	sweet
5	not-red	not-sweet
6	half-red	sweet
7	red	not-sweet
8	not-red	not-sweet
9	not-red	sweet
10	half-red	not-sweet
11	red	sweet
12	half-red	not-sweet
13	not-red	not-sweet

information gain:

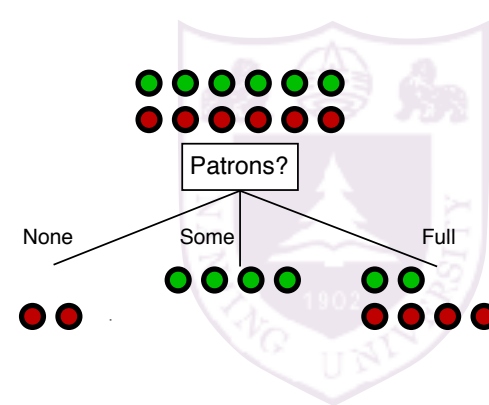
entropy before split:  $H(X) = - \sum_i \text{ratio}(\text{class}_i) \ln \text{ratio}(\text{class}_i) = 0.6902$

entropy after split:  $I(X; \text{split}) = \sum_i \text{ratio}(\text{split}_i) H(\text{split}_i)$

information gain:  $= \frac{4}{13} 0.5623 + \frac{4}{13} 0.6931 + \frac{5}{13} 0.6730 = 0.6452$

$\text{Gain}(X; \text{split}) = H(X) - I(X; \text{split}) = 0.045$

# Decision tree learning algorithm



Aim: find a small tree consistent with the training examples

Idea: (recursively) choose “most significant” attribute as root of (sub)tree

**function** **DTL**(*examples*, *attributes*, *default*) **returns** a decision tree

**if** *examples* is empty **then return** *default*

**else if** all *examples* have the same classification **then return** the classification

**else if** *attributes* is empty **then return** **MODE**(*examples*)

**else**

*best* ← **CHOOSE-ATTRIBUTE**(*attributes*, *examples*)

*tree* ← a new decision tree with root test *best*

**for each** value  $v_i$  of *best* **do**

*examples<sub>i</sub>* ← {elements of *examples* with *best* =  $v_i$ }

*subtree* ← **DTL**(*examples<sub>i</sub>*, *attributes* – *best*, **MODE**(*examples*))

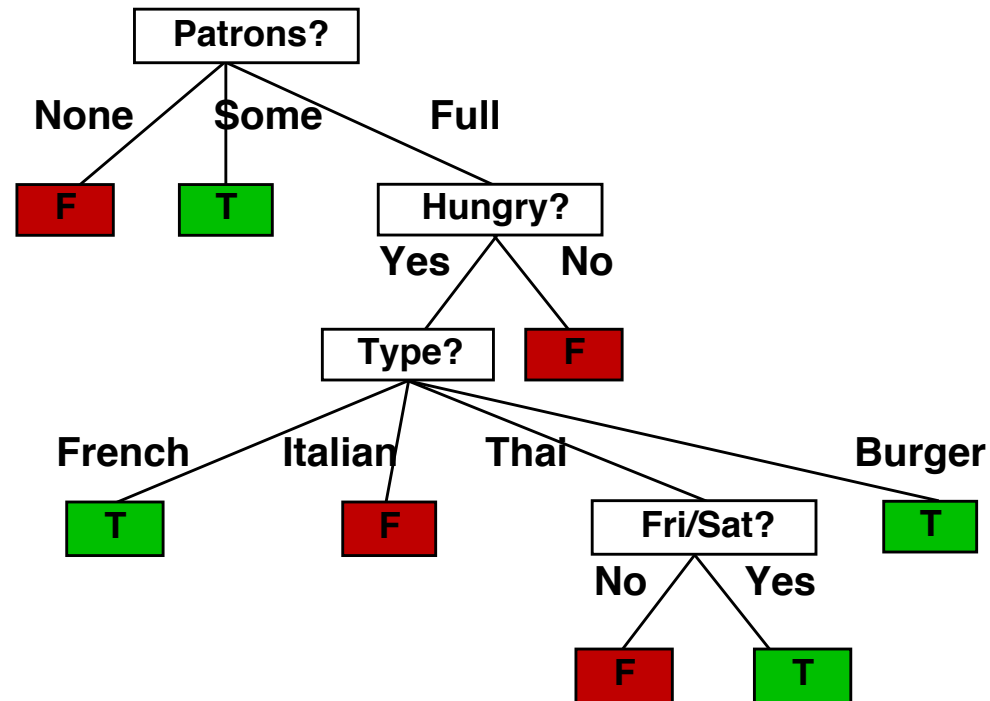
add a branch to *tree* with label  $v_i$  and subtree *subtree*

**return** *tree*

# Example of learned tree



Decision tree learned from the 12 examples:

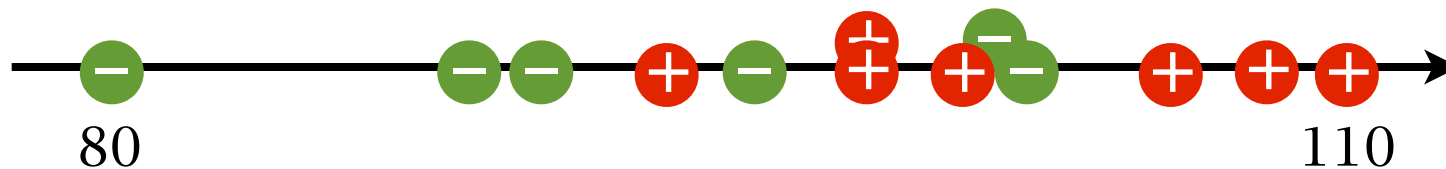


Substantially simpler than “true” tree—a more complex hypothesis isn’t justified by small amount of data

# Continuous attribute

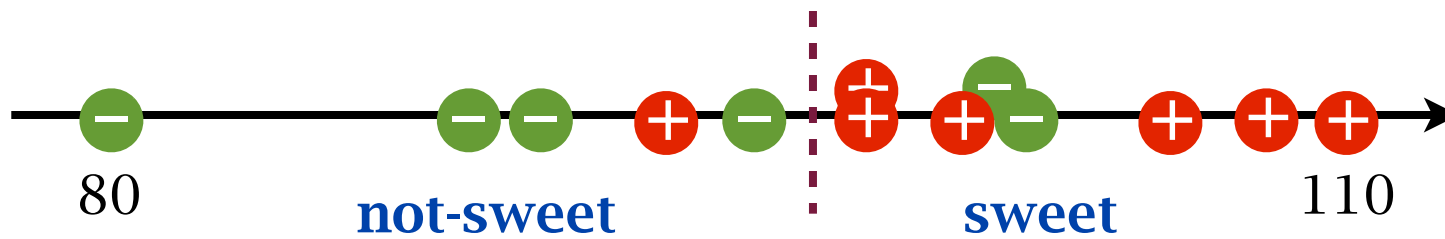


id	weight	taste
1	110	sweet
2	105	sweet
3	100	sweet
4	93	sweet
5	80	not-sweet
6	98	sweet
7	95	not-sweet
8	102	not-sweet
9	98	sweet
10	90	not-sweet
11	108	sweet
12	101	not-sweet
13	89	not-sweet





# Continuous attribute



for every split point

information gain:

entropy before split:  $H(X) = - \sum_i ratio(class_i) \ln ratio(class_i) = 0.6902$

entropy after split:  $I(X; split) = \sum_i ratio(split_i) H(split_i)$   
 $= \frac{5}{13} 0.5004 + \frac{8}{13} 0.5623 = 0.5385$

information gain:

$$Gain(X; split) = H(X) - I(X; split) = 0.1517$$

# Non-generalizable feature



id	color	weight	taste
1	red	110	sweet
2	red	105	sweet
3	half-red	100	sweet
4	not-red	93	sweet
5	not-red	80	not-sweet
6	half-red	98	sweet
7	red	95	not-sweet
8	not-red	102	not-sweet
9	not-red	98	sweet
10	half-red	90	not-sweet
11	red	108	sweet
12	half-red	101	not-sweet
13	not-red	89	not-sweet

the system may not know non-generalizable features

$$IG = H(X) - 0$$

Gain ratio as a correction:

$$\text{Gain ratio}(X) = \frac{H(X) - I(X; \text{split})}{IV(\text{split})}$$

$$IV(\text{split}) = H(\text{split})$$

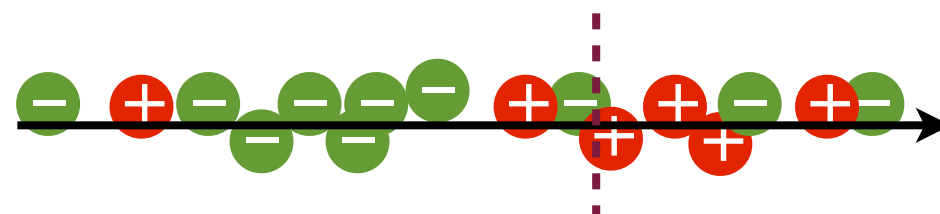
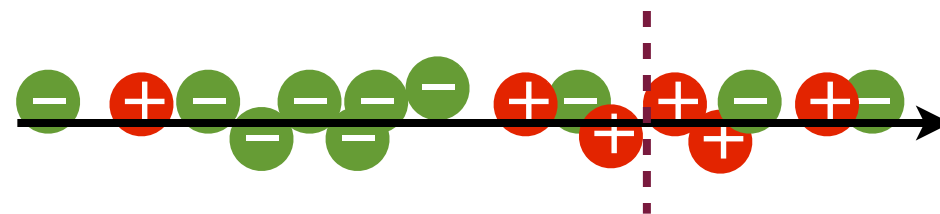
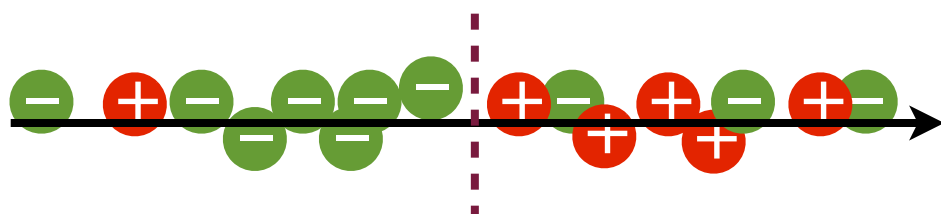
# Alternative to information: Gini index



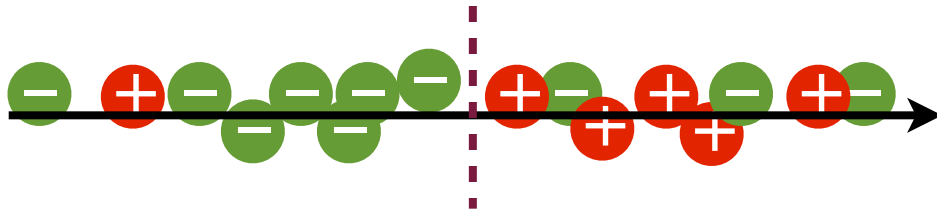
## Gini index (CART):

$$\text{Gini: } Gini(X) = 1 - \sum_i p_i^2$$

$$\text{Gini after split: } \frac{\#\text{left}}{\#\text{all}} Gini(\text{left}) + \frac{\#\text{right}}{\#\text{all}} Gini(\text{right})$$

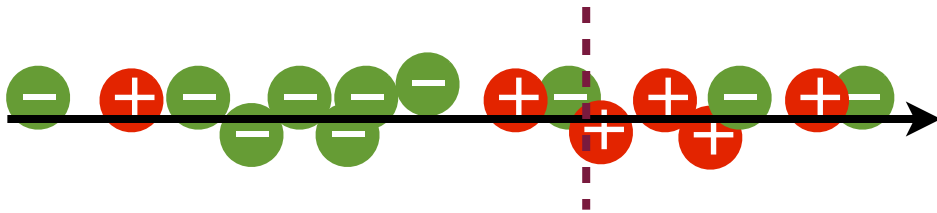


# Training error v.s. Information gain



training error: 4

information gain:  $IG = H(X) - 0.5192$



training error: 4

information gain:  $IG = H(X) - 0.5514$

*training error is less smooth*

# Decision tree learning algorithms



**ID3: information gain**

**C4.5: gain ratio, handling missing values**



Ross Quinlan

**CART: gini index**



Leo Breiman 1928-2005



Jerome H. Friedman

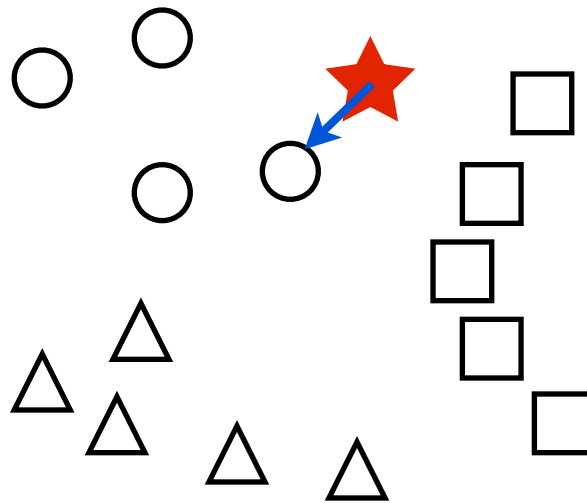


# Nearest Neighbor Classifier

# Nearest neighbor



what looks similar are similar

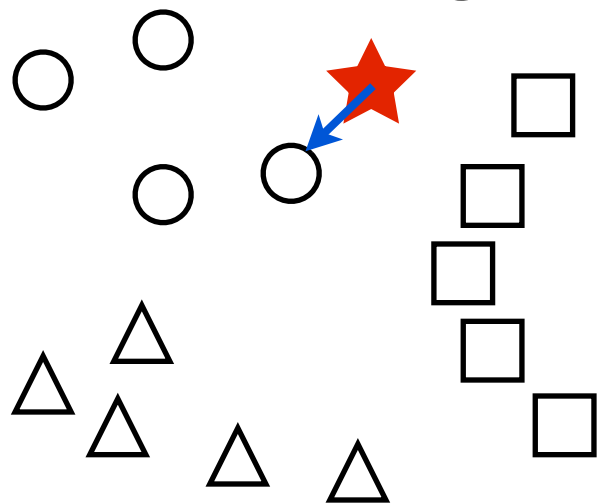


# Nearest neighbor

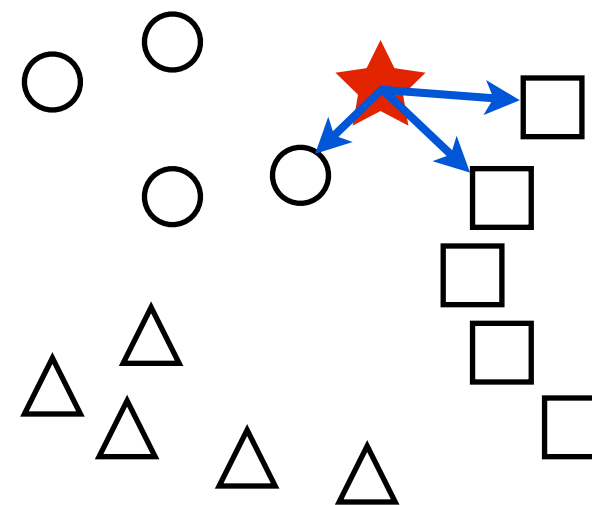


for classification:

1-nearest neighbor:



$k$ -nearest neighbor:



Predict the label as that of the NN  
or the (weighted) majority of the  $k$ -NN

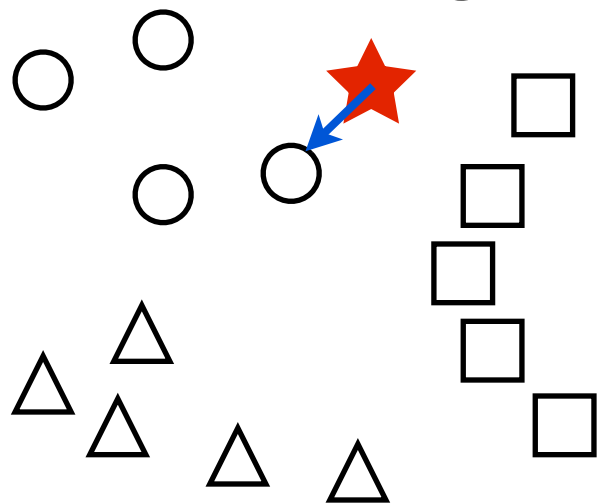


# Nearest neighbor

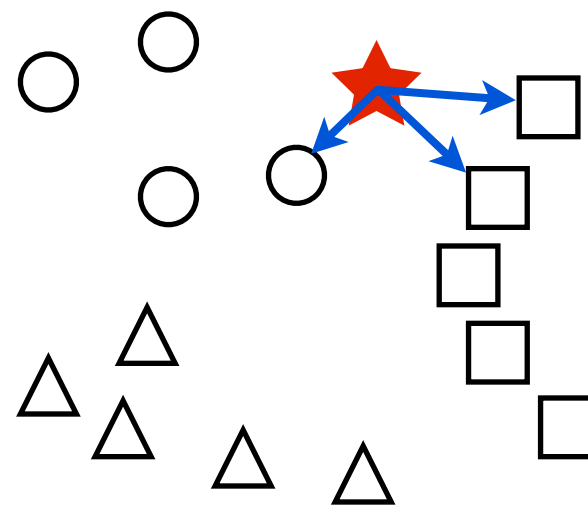


for regression:

1-nearest neighbor:



$k$ -nearest neighbor:

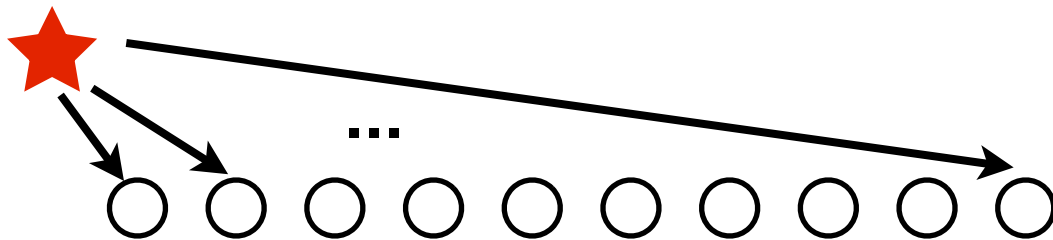


Predict the label as that of the NN  
or the (weighted) *average* of the  $k$ -NN

# Search for the nearest neighbor



## Linear search



$n$  times of distance calculations

$$O(dn \ln k)$$

$d$  is the dimension,  $n$  is the number of samples

# Nearest neighbor classifier



- ▶ as classifier, asymptotically less than 2 times of the optimal Bayes error
- ▶ naturally handle multi-class
- ▶ no training time
- ▶ nonlinear decision boundary
  
- ▶ slow testing speed for a large training data set
- ▶ have to store the training data
- ▶ sensitive to similarity function

nonparametric method



# Naive Bayes Classifier

# Bayes rule



classification using posterior probability

for binary classification

$$f(\mathbf{x}) = \begin{cases} +1, & P(y = +1 | \mathbf{x}) > P(y = -1 | \mathbf{x}) \\ -1, & P(y = +1 | \mathbf{x}) < P(y = -1 | \mathbf{x}) \\ \text{random,} & \textit{otherwise} \end{cases}$$

in general

$$\begin{aligned} f(\mathbf{x}) &= \arg \max_y P(y | \mathbf{x}) \\ &= \arg \max_y P(\mathbf{x} | y)P(y)/P(\mathbf{x}) \\ &= \arg \max_y P(\mathbf{x} | y)P(y) \end{aligned}$$

how the  
probabilities be  
estimated

# Naive Bayes

$$f(x) = \arg \max_y P(\mathbf{x} | y)P(y)$$

estimation the a priori by frequency:

$$P(y) \leftarrow \tilde{P}(y) = \frac{1}{m} \sum_i I(y_i = y)$$



# Consider a very simple case



color ←



→ taste ?

id	color	taste
1	red	sweet
2	red	sweet
3	half-red	not-sweet
4	not-red	not-sweet
5	not-red	not-sweet
6	half-red	not-sweet
7	red	sweet
8	not-red	not-sweet
9	not-red	not-sweet
10	half-red	not-sweet
11	red	sweet
12	half-red	not-sweet
13	not-red	not-sweet

$$P(\text{red} \mid \text{sweet}) = 1$$

$$P(\text{half-red} \mid \text{sweet}) = 0$$

$$P(\text{not-red} \mid \text{sweet}) = 0$$

$$P(\text{sweet}) = 4/13$$

$$P(\text{red} \mid \text{not-sweet}) = 0$$

$$P(\text{half-red} \mid \text{not-sweet}) = 4/9$$

$$P(\text{not-red} \mid \text{not-sweet}) = 5/9$$

$$P(\text{not-sweet}) = 9/13$$

# Consider a very simple case



id	color	taste
1	red	sweet
2	red	sweet
3	half-red	not-sweet
4	not-red	not-sweet
5	not-red	not-sweet
6	half-red	not-sweet
7	red	sweet
8	not-red	not-sweet
9	not-red	not-sweet
10	half-red	not-sweet
11	red	sweet
12	half-red	not-sweet
13	not-red	not-sweet

what the  $f'$  would be?

$$f(x) = \arg \max_y P(x | y)P(y)$$

$$P(\text{red} | \text{sweet})P(\text{sweet}) = 4/13$$

$$P(\text{red} | \text{not-sweet})P(\text{not-sweet}) = 0$$

$$P(\text{half-red} | \text{sweet})P(\text{sweet}) = 0$$

$$P(\text{half-red} | \text{not-sweet})P(\text{not-sweet}) = \frac{4}{9} \times \frac{9}{13} = \frac{4}{13}$$

*perfect  
but not realistic*





# Naive Bayes

$$f(x) = \arg \max_y P(\mathbf{x} | y)P(y)$$

estimation the a priori by frequency:

$$P(y) \leftarrow \tilde{P}(y) = \frac{1}{m} \sum_i I(y_i = y)$$

assume features are conditional independence given the class (**naive assumption**):

$$\begin{aligned} P(\mathbf{x} | y) &= P(x_1, x_2, \dots, x_n | y) \\ &= P(x_1 | y) \cdot P(x_2 | y) \cdot \dots \cdot P(x_n | y) \end{aligned}$$

decision function:

$$f(x) = \arg \max_y \tilde{P}(y) \prod_i \tilde{P}(x_i | y)$$

# Naive Bayes



color={0,1,2,3} weight={0,1,2,3,4}

color	weight	sweet?
3	4	yes
2	3	yes
0	3	no
3	2	no
1	4	no

$$P(y = \text{yes}) = 2/5$$

$$P(y = \text{no}) = 3/5$$

$$P(\text{color} = 3 \mid y = \text{yes}) = 1/2$$

...

$$f(y \mid \text{color} = 3, \text{weight} = 3) \rightarrow$$

$$P(\text{color} = 3 \mid y = \text{yes})P(\text{weight} = 3 \mid y = \text{yes})P(y = \text{yes}) = 0.5 \times 0.5 \times 0.4 = 0.1$$

$$P(\text{color} = 3 \mid y = \text{no})P(\text{weight} = 3 \mid y = \text{no})P(y = \text{no}) = 0.33 \times 0.33 \times 0.6 = 0.06$$

$$f(y \mid \text{color} = 0, \text{weight} = 1) \rightarrow$$

$$P(\text{color} = 0 \mid y = \text{yes})P(\text{weight} = 1 \mid y = \text{yes})P(y = \text{yes}) = 0$$

$$P(\text{color} = 0 \mid y = \text{no})P(\text{weight} = 1 \mid y = \text{no})P(y = \text{no}) = 0$$

# Naive Bayes



color={0,1,2,3} weight={0,1,2,3,4}

color	weight	sweet?
3	4	yes
2	3	yes
0	3	no
3	2	no
1	4	no

+

color	sweet?
0	yes
1	yes
2	yes
3	yes

**smoothed (Laplacian correction) probabilities:**

$$P(\text{color} = 0 \mid y = \text{yes}) = (0 + 1) / (2 + 4)$$

$$P(y = \text{yes}) = (2 + 1) / (5 + 2)$$

for counting frequency,  
assume every event  
has happened once.

$$f(y \mid \text{color} = 0, \text{weight} = 1) \rightarrow$$

$$P(\text{color} = 0 \mid y = \text{yes})P(\text{weight} = 1 \mid y = \text{yes})P(y = \text{yes}) = \frac{1}{6} \times \frac{1}{7} \times \frac{3}{7} = 0.01$$

$$P(\text{color} = 0 \mid y = \text{no})P(\text{weight} = 1 \mid y = \text{no})P(y = \text{no}) = \frac{2}{7} \times \frac{1}{8} \times \frac{4}{7} = 0.02$$

# Naive Bayes



advantages:

very fast:

scan the data once, just count:  $O(mn)$

store class-conditional probabilities:  $O(n)$

test an instance:  $O(cn)$  ( $c$  the number of classes)

good accuracy in many cases

parameter free

output a probability

naturally handle multi-class

disadvantages:

the strong assumption may harm the accuracy

does not handle numerical features naturally