Learning from Observations

Chapter 18, Sections 1–4

Outline

♢ Learning agents

♢ Inductive learning

♢ Decision tree learning

(Next lecture covers neural networks)

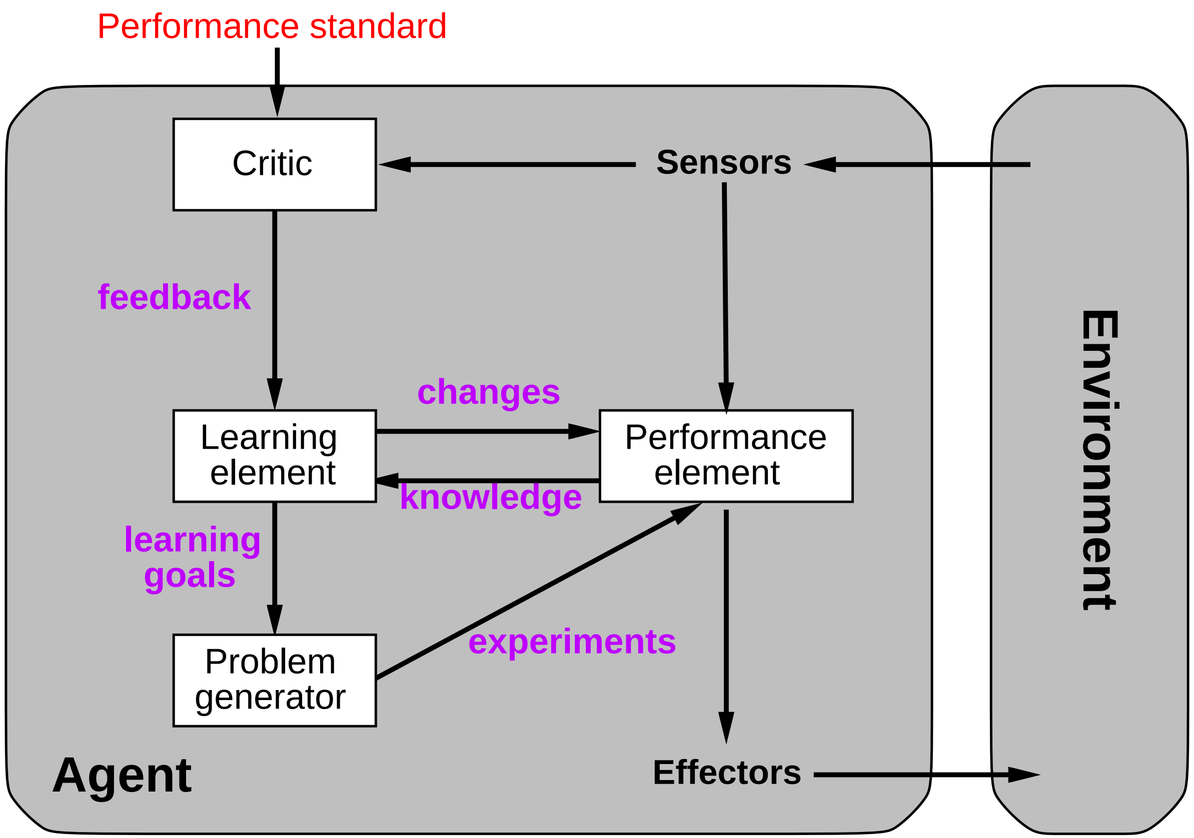
Learning

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

Learning agents



Performance -vs- Learning Element

- Performance element is what we have called *agent* up to now
- Critic/LearningElement/ProblemGenerator handle **improvement**
- Performance standard is fixed;
  - can't adjust performance standard to flatter own behavior
  - no standard *in the environment*: ordinary chess and suicide chess LOOK identical. Essentially, certain kinds of percepts are "hardwired" as good/bad (e.g., pain, hunger)
- Learning element may use knowledge already acquired in the perf. element
- Learning may require experimentation - actions an agent might not normally consider such as dropping rocks for the Tower of Pisa

Learning element

Design of learning element is dictated by

- ◊ type of performance element
- ◊ functional component to be learned
- ◊ representation of that functional component
- ◊ type of available feedback

Example scenarios:

Performance element	Component	Representation	Feedback
Alpha−beta search	Eval. fn.	Weighted linear function	Win/loss
Logical agent	Transition model	Successor−state axioms	Outcome
Utility−based agent	Transition model	Dynamic Bayes net	Outcome
Simple reflex agent	Percept−action fn	Neural net	Correct action

Supervised learning: correct answers for each instance are provided

Reinforcement learning: occasional rewards of type *good/bad*

Inductive learning (a.k.a. Science)

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

f is the target function

An example is a pair x, $f(x)$, e.g., $\begin{array}{c|c|c} O & O & X \\ \hline & X & \\ \hline X & & \end{array}\ ,\quad +1$

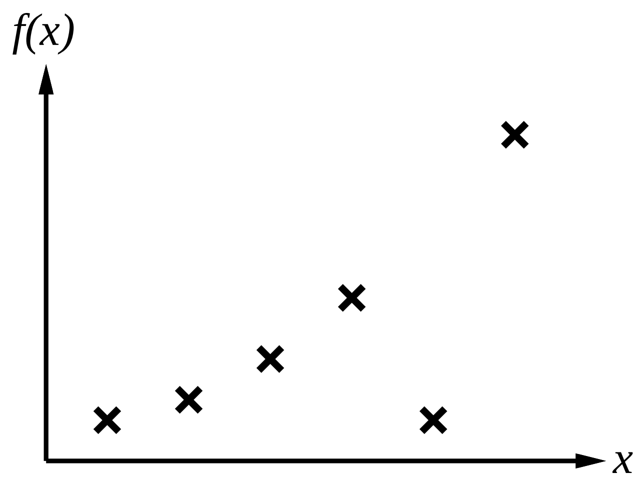
Problem: find a(n) hypothesis h
such that $h \approx f$
given a training set of examples

(**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?**)

Inductive learning method

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

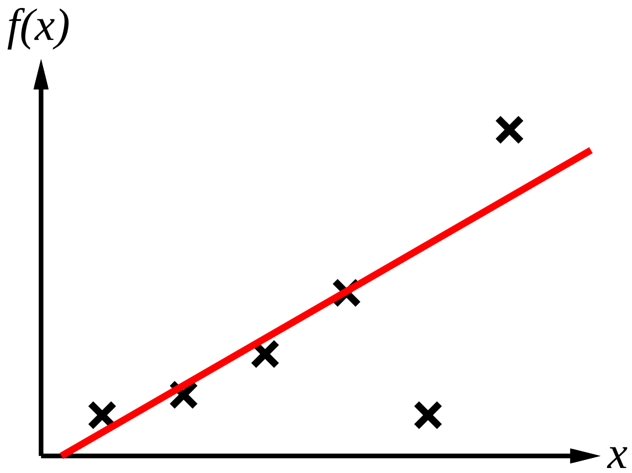
E.g., curve fitting:



Inductive learning method

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

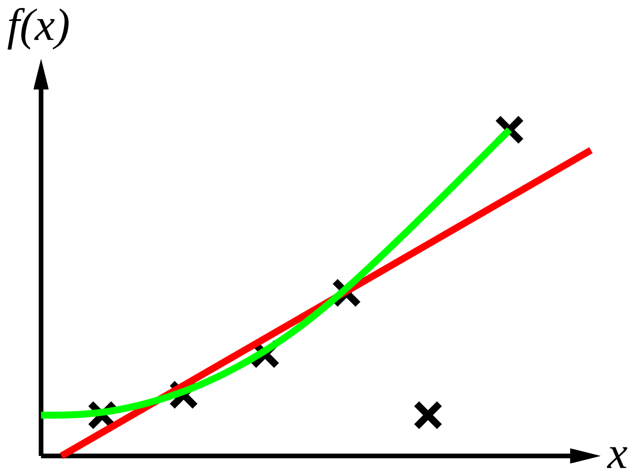
E.g., curve fitting:



Inductive learning method

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

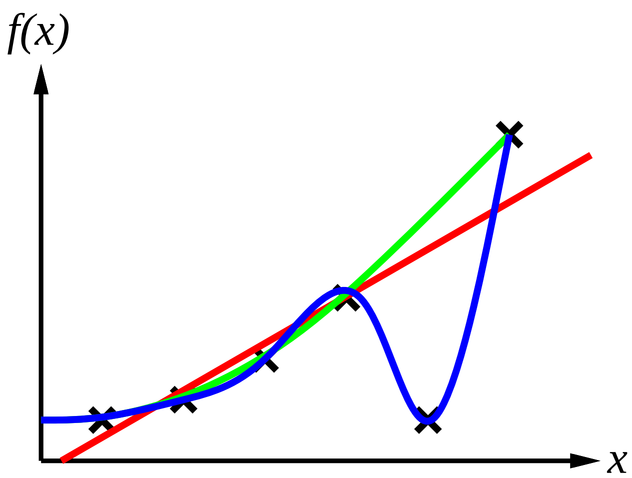
E.g., curve fitting:



Inductive learning method

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

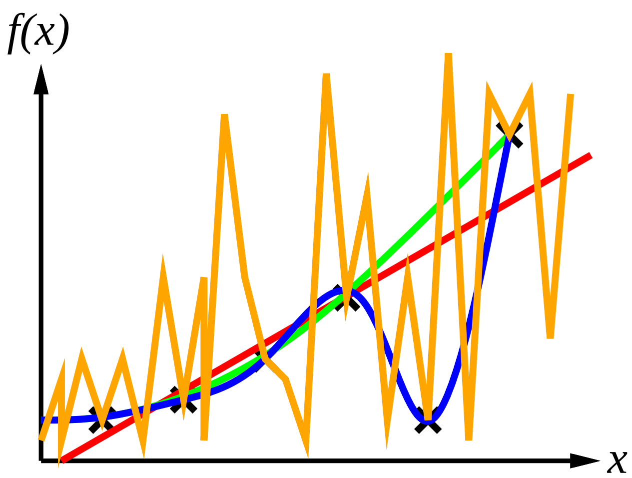
E.g., curve fitting:



Inductive learning method

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

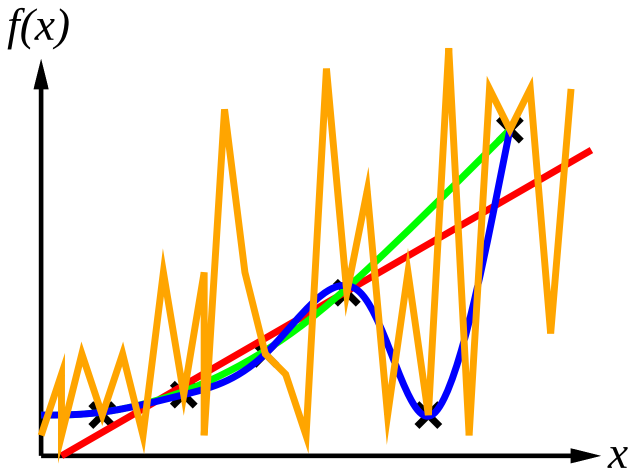
E.g., curve fitting:



Inductive learning method

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:



Ockham's razor: maximize a combination of consistency and simplicity

Attribute-based representations

Examples described by attribute values (Boolean, discrete, continuous, etc.)
E.g., situations where I will/won't wait for a table:

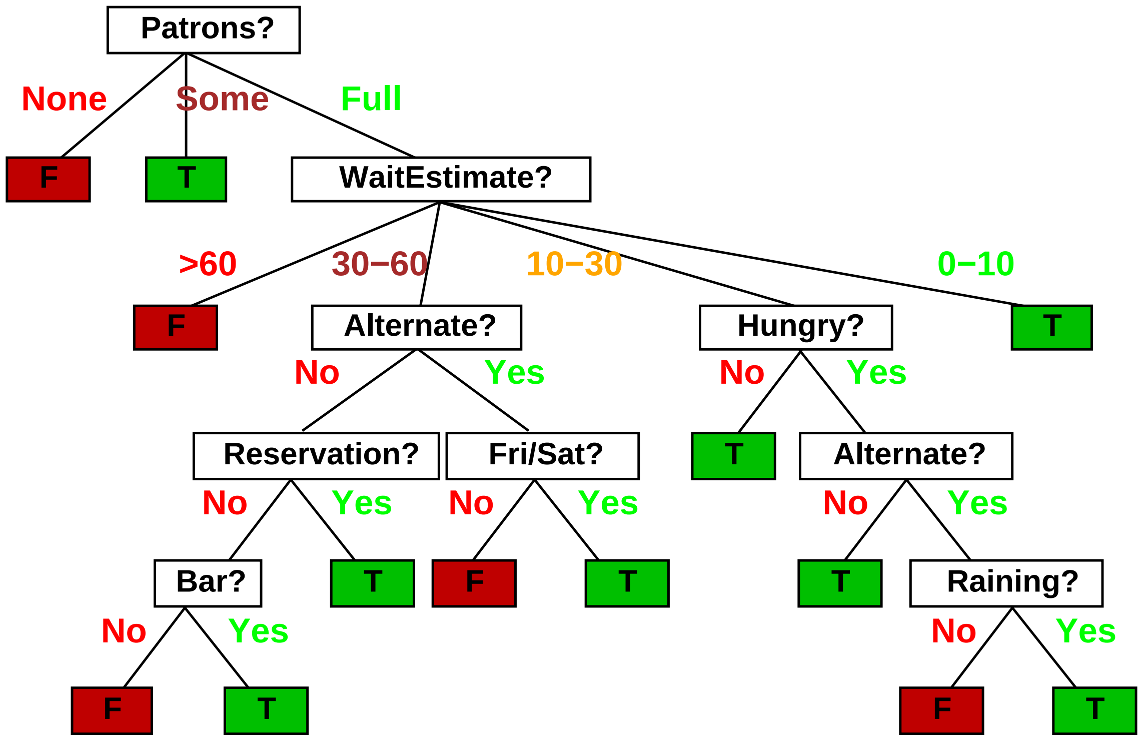
Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*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

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

Decision trees

One possible representation for hypotheses

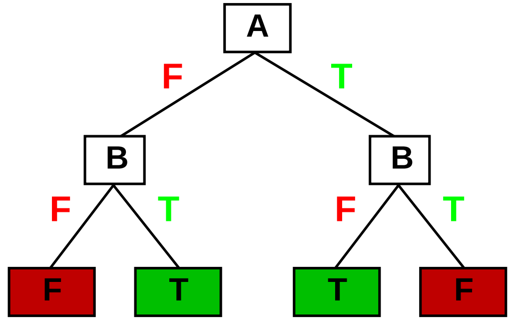
E.g., here is the "true" tree for deciding whether to wait:



Expressiveness

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

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Trivially, $\exists$ 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: Ask the fewest number of questions (i.e. query the fewest attributes) to reach a decision about a classification or an action to take.

Hypothesis Space Size

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}
  - Each row represents a situation.
  - So there are $S = 2^n$ situations.
  - Each situation can be classified as True or False.
  - So there are $2^S = 2^{2^n}$ different ways to classify the situations.
  - E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees
- This is a general result for any decision-making system with:
  - A fixed number of situations.
  - A fixed set of actions/classifications for each situation.
- The **strategy** in a decision table $=$ its 2^n actions.

Hypothesis Space Size (2)

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

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

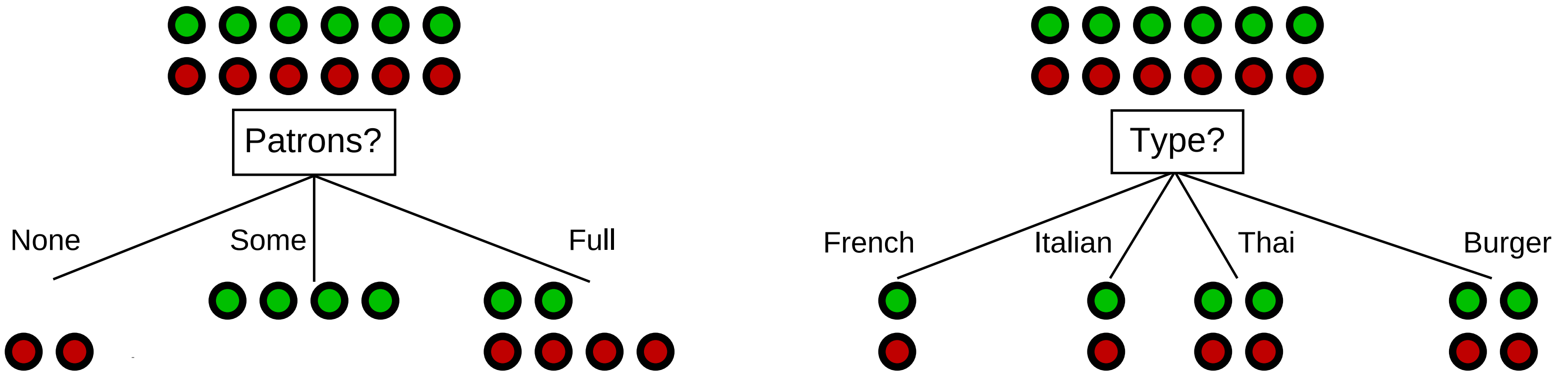
Aim: find a small tree consistent with the training examples

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_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Choosing an attribute

Key Point: a good attribute splits the examples into subsets that are as close as possible to **all positive** and **all negative**.
If the split is perfect, then no further questions need to be asked.



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

Information and Entropy

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 $\langle 0.5, 0.5 \rangle$

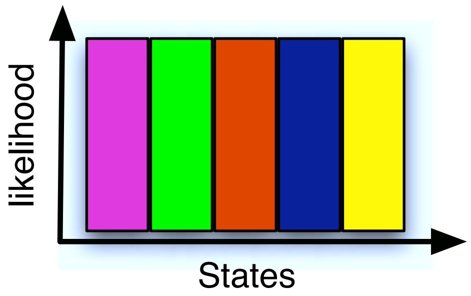
Information in an answer when prior is $\langle P_1, \ldots, P_n \rangle$ is

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

(also called entropy of the prior)

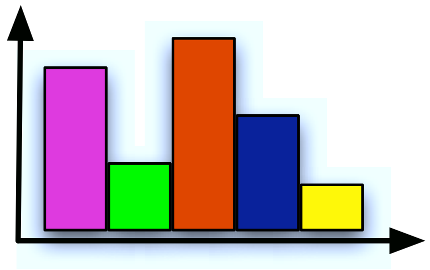
- Maximum entropy occurs when $P_i = P_j \; \forall i, j$
- In such a perfectly even distribution, $H(\langle P_1, \ldots, P_n \rangle) = \log_2 n$
- Conversely, if $P_k = 1$ and $P_i = 0 \; \forall i \neq k$, $H(\langle P_1, \ldots, P_n \rangle) = 0$

Information and Entropy (2)



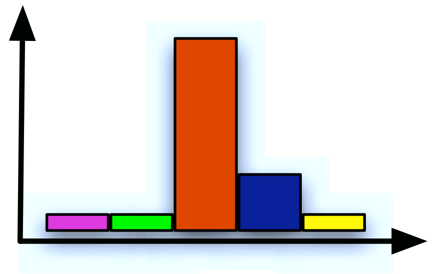
High Entropy

Maximum Disorder
All states equally likely.
A lot of info is still needed
to discriminate among
alternatives.



Medium Entropy

Clear separation
into high- and
low-probability
alternatives



Low Entropy

Minimum Disorder
One state
clearly dominates.
Only a little additional
info is needed to decide
among the alternatives.

Information and Entropy (3)

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

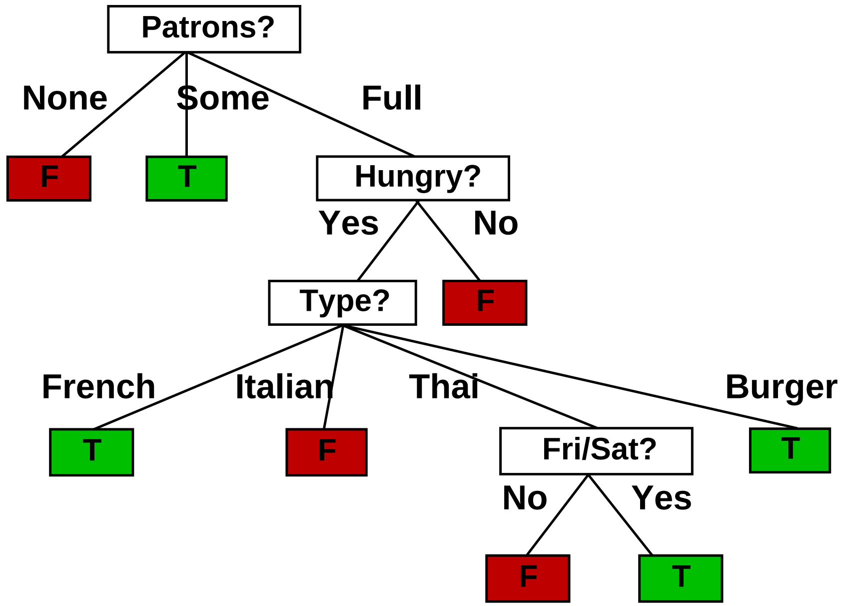
$$\Sigma_i \; \frac{p_i + n_i}{p + n} H(\langle p_i/(p_i + n_i), n_i/(p_i + n_i)\rangle)$$

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

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

Restaurant Example Revisited

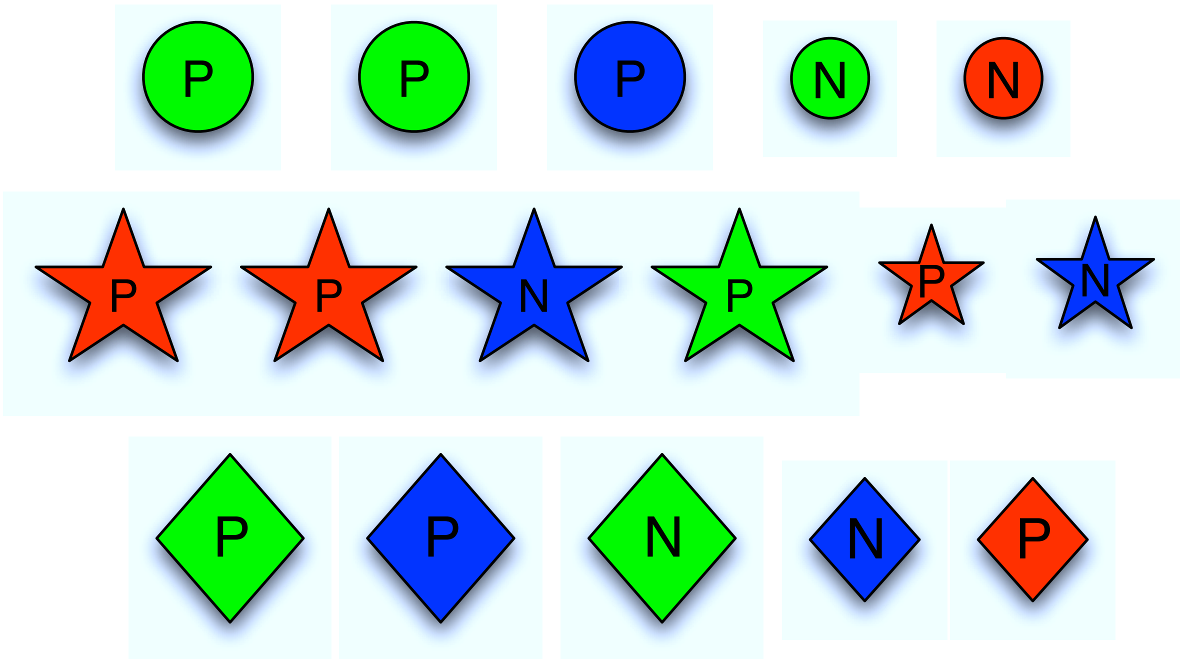
Decision tree learned from the 12 examples:



Substantially simpler than "true" tree—a more complex hypothesis isn't justified by small amount of data

Colored Shapes

Build an efficient decision tree for sorting the positive and negative examples. I.e., Minimize # questions needed to correctly classify any of the 16.

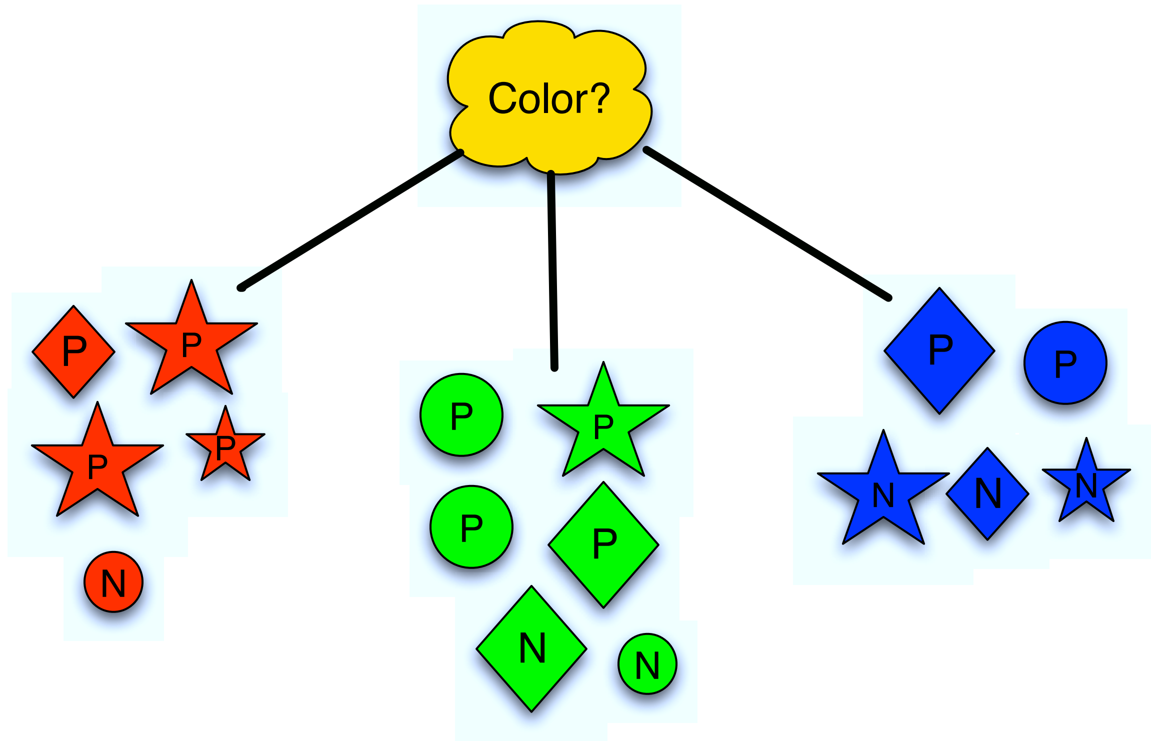


Attributes: **Color** (Red, Blue, Green), **Size** (Big, Small), **Shape** (Circle, Star, Diamond)

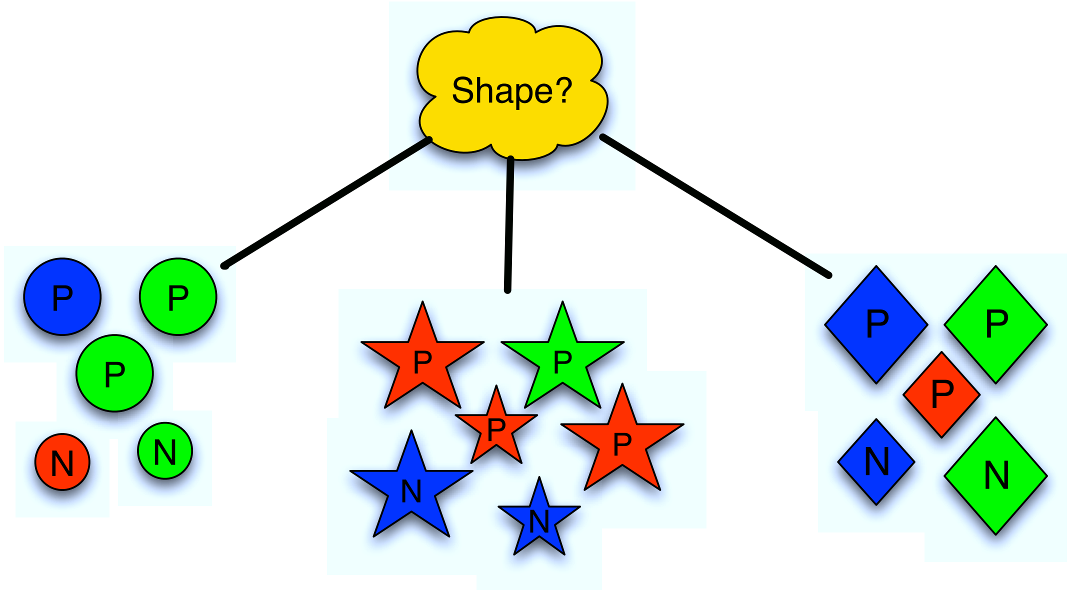
Counts: Red: 5, Blue: 5, Green: 6 Big: 10, Small: 6 Circle: 5, Star: 6, Diamond: 5

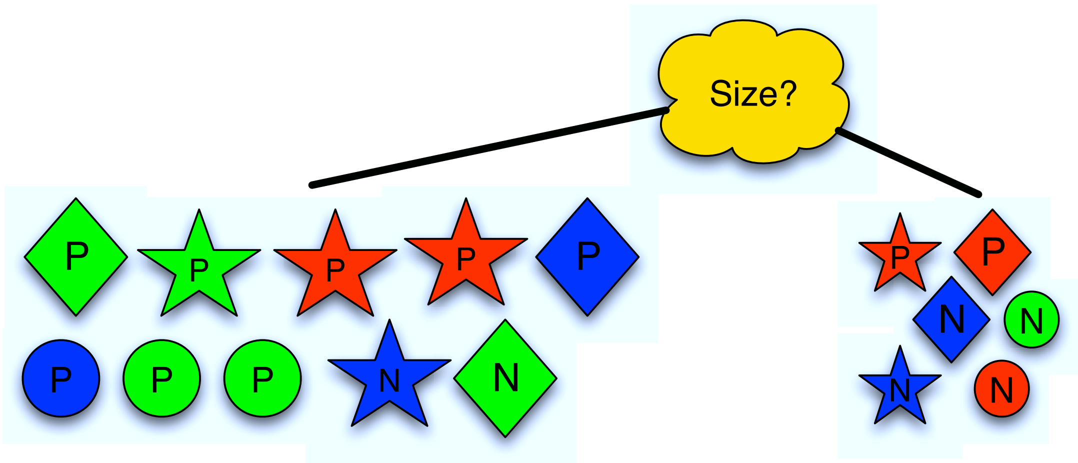
Best First Question?

- What is the best attribute to ask about, first?
- Try each one, and see how the P and N examples get partitioned by each question.



Trying Other Questions





Expected Remaining Information Needs

Color?

- Red: $\frac{5}{16}(-0.8\log_2 0.8 + -0.2\log_2 0.2) = .226$
- Green: $\frac{6}{16}(-0.666\log_2 0.666 + -0.333\log_2 0.333) = .344$
- Blue: $\frac{5}{16}(-0.4\log_2 0.4 + -0.6\log_2 0.6) = .303$
- Total: .873

Shape?

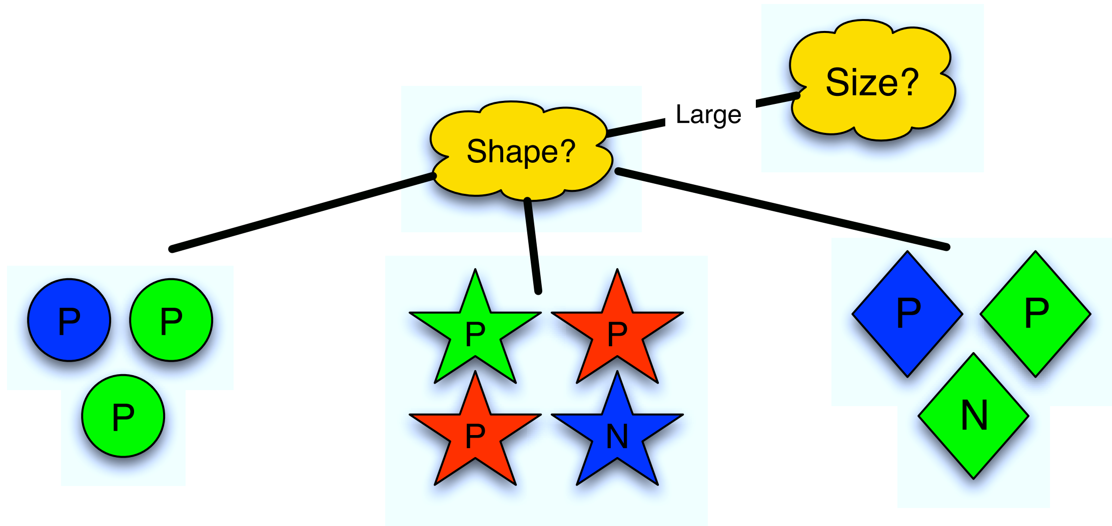
- Circle: $\frac{5}{16}(-0.6\log_2 0.6 + -0.4\log_2 0.4) = .303$
- Star: $\frac{6}{16}(-0.666\log_2 0.666 + -0.333\log_2 0.333) = .344$
- Diamond: $\frac{5}{16}(-0.6\log_2 0.6 + -0.4\log_2 0.4) = .303$
- Total: .950

Size?

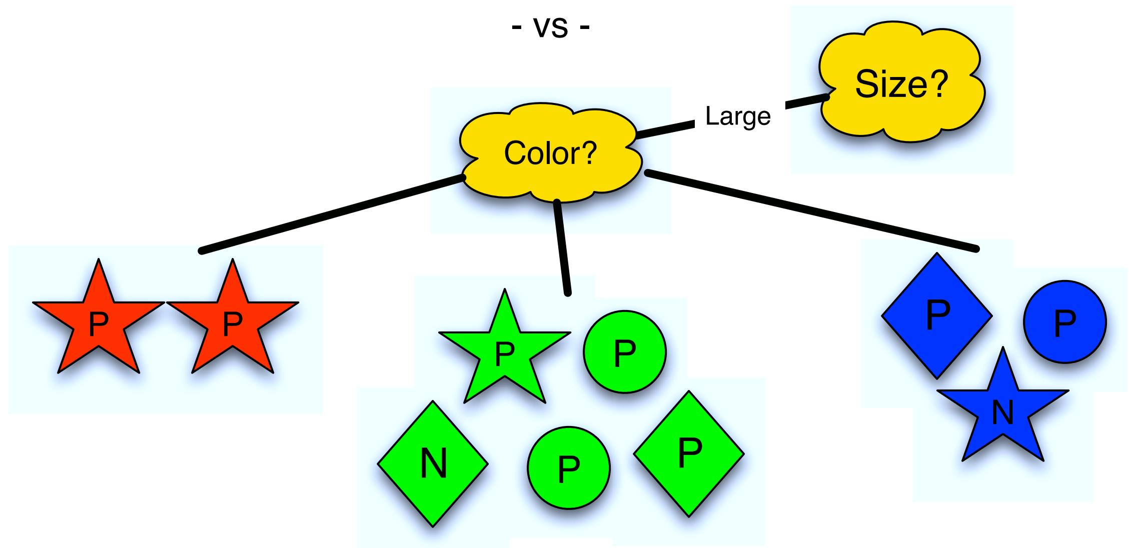
- Large: $\frac{10}{16}(-0.8\log_2 0.8 + -0.2\log_2 0.2) = .451$
- Small: $\frac{6}{16}(-0.333\log_2 0.333 + -0.666\log_2 0.666) = .344$
- **Total: .795** Yields partitions with best separation of P and N.

Recursion!!

Same analysis on each branch, using instance subset + remaining questions.



- vs -



Expected Remaining Info Needs

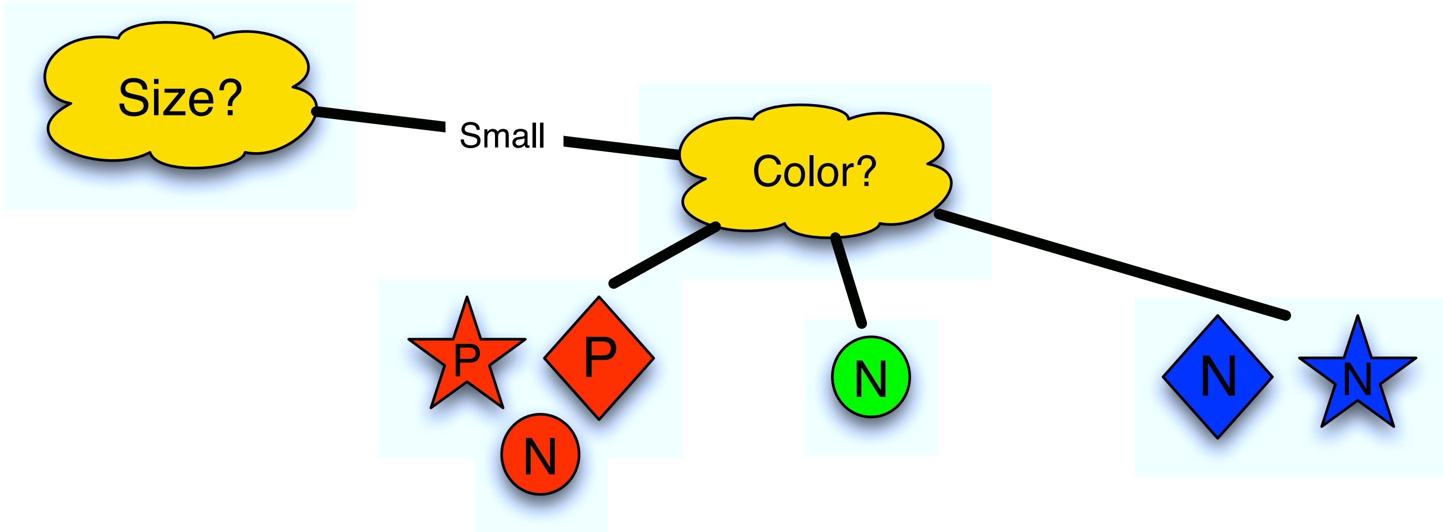
Size = Large, then Shape?

- Circle: $\frac{3}{10}(-1.0 \log_2 1.0 + -0.0 \log_2 0.0) = 0$
- Star: $\frac{4}{10}(-0.75 \log_2 0.75 + -0.25 \log_2 0.25) = .325$
- Diamond: $\frac{3}{10}(-0.666 \log_2 0.666 + -0.333 \log_2 0.333) = .275$
- **Total: .600** So on this branch, ask **Shape?** next.

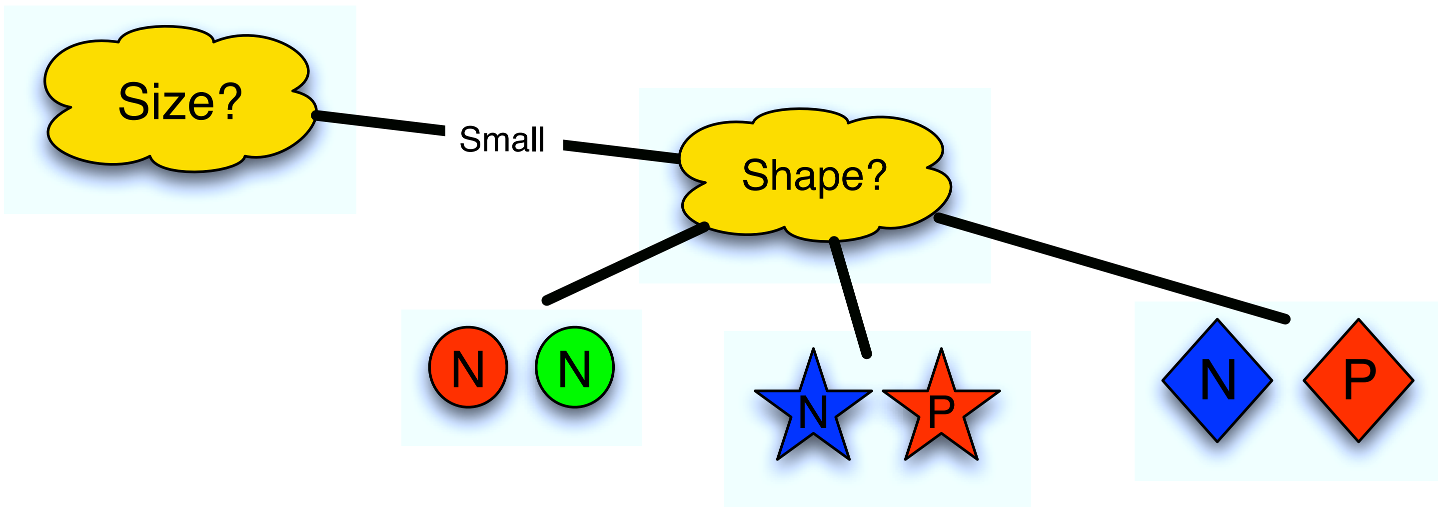
Size = Large, then Color?

- Red: $\frac{2}{10}(-1.0 \log_2 1.0 + -0.0 \log_2 0.0) = 0$
- Green: $\frac{5}{10}(-0.8 \log_2 0.8 + -0.2 \log_2 0.2) = .361$
- Blue: $\frac{3}{10}(-0.666 \log_2 0.666 + -0.333 \log_2 0.333) = .275$
- Total: .636

Recursion For Size = Small



- vs -



Expected Remaining Info Needs

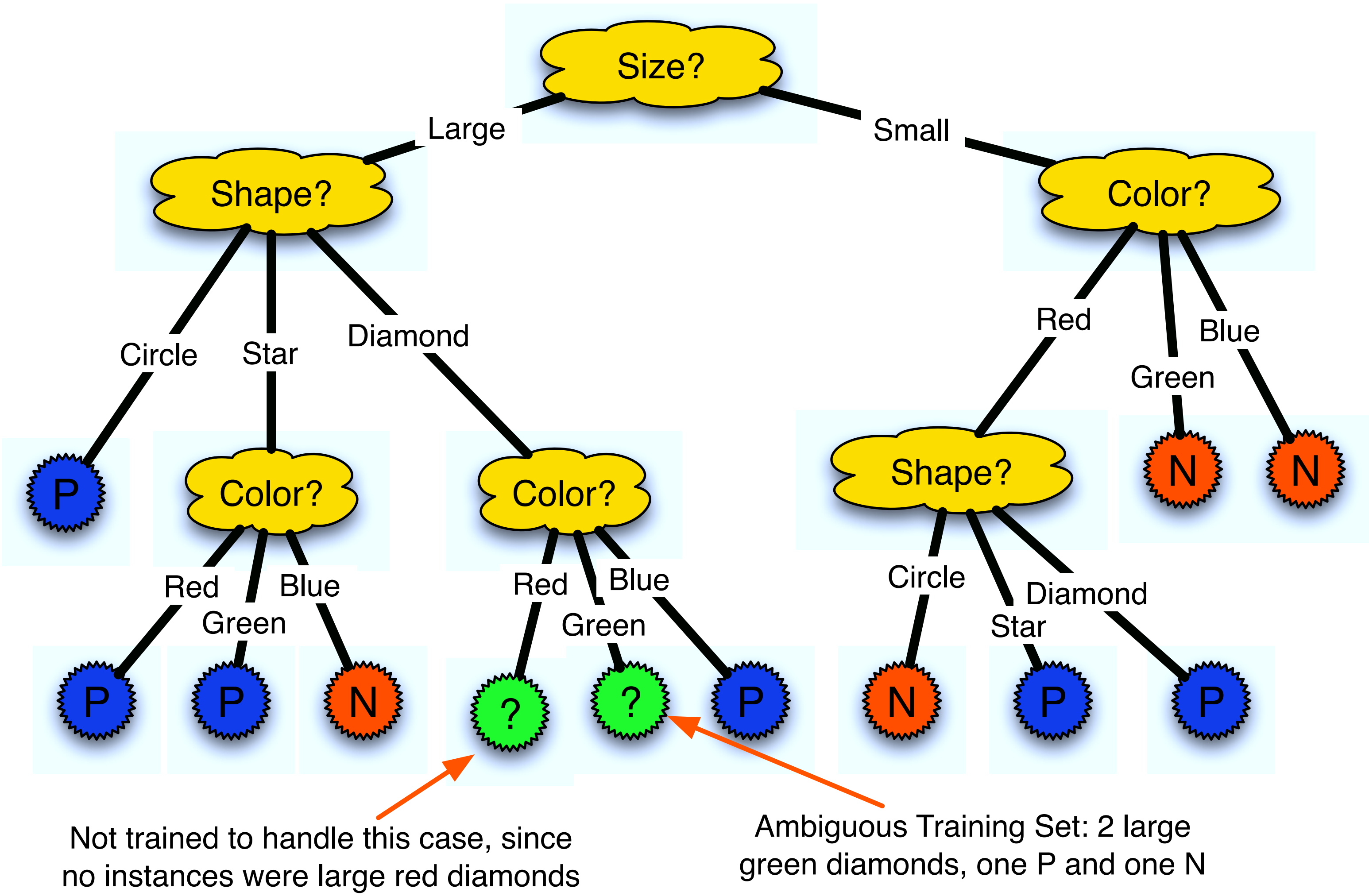
Size $=$ Small, then Shape?

- Circle: $\frac{2}{6}(-0.0\log_2 0.0 + -1.0\log_2 1.0) = 0$
- Star: $\frac{2}{6}(-0.5\log_2 0.5 + -0.5\log_2 0.5) = .333$
- Diamond: $\frac{2}{6}(-0.5\log_2 0.5 + -0.5\log_2 0.5) = .333$
- Total: .666

Size $=$ Small, then Color?

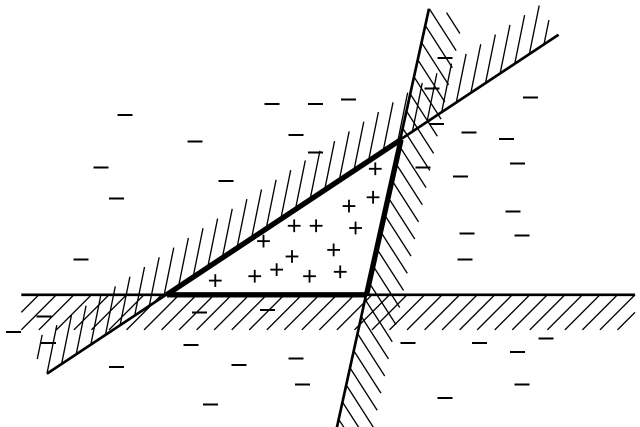
- Red: $\frac{3}{6}(-0.666\log_2 0.666 + -0.333\log_2 0.333) = .459$
- Green: $\frac{1}{6}(-0.0\log_2 0.0 + -1.0\log_2 1.0) = 0$
- Blue: $\frac{2}{6}(-0.0\log_2 0.0 + -1.0\log_2 1.0) = 0$
- **Total: .459** On this branch, ask **Color?** next.

The Complete Decision Tree



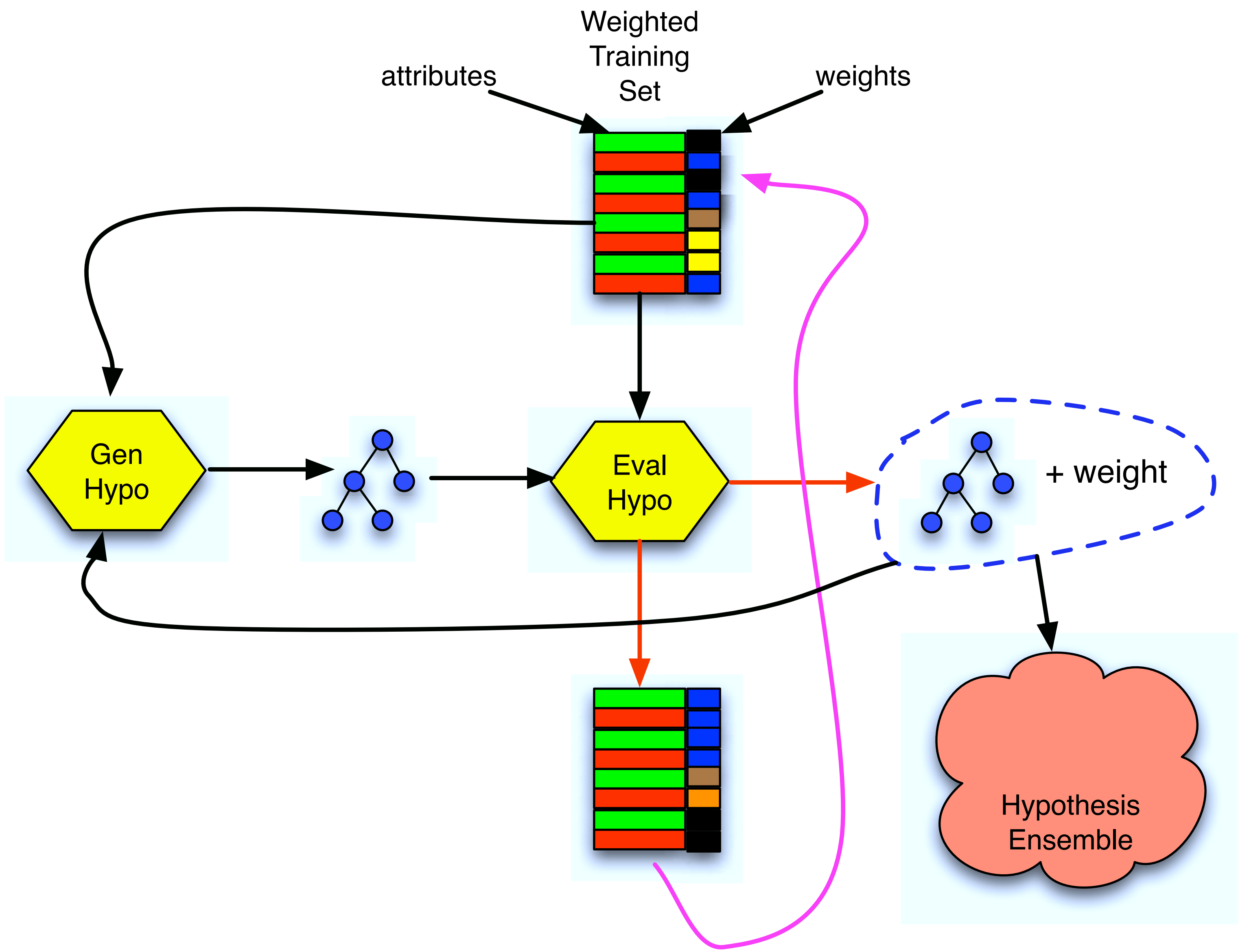
Boosting

- Many learning problems are very complex.
- A single perfect hypothesis is hard (or impossible) to create.
- So create a population of different hypotheses, where:
  - Each is generated from the same training set.
  - But the examples are weighted, and weights change between hypothesis-creation rounds.
- During testing, each hypothesis gets to **vote** on the correct answer.
- Votes are weighted by the **credibility** of the hypothesis (which is derived from its accuracy on the training data)



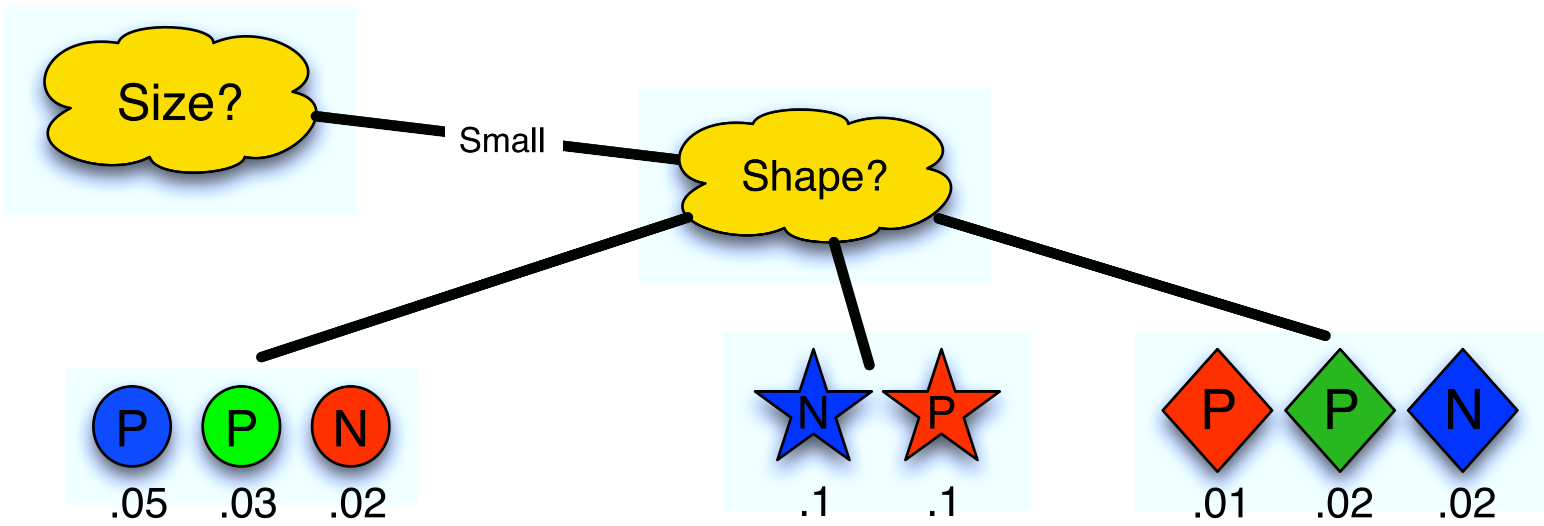
Ch

Basic Boosting Process



Ch

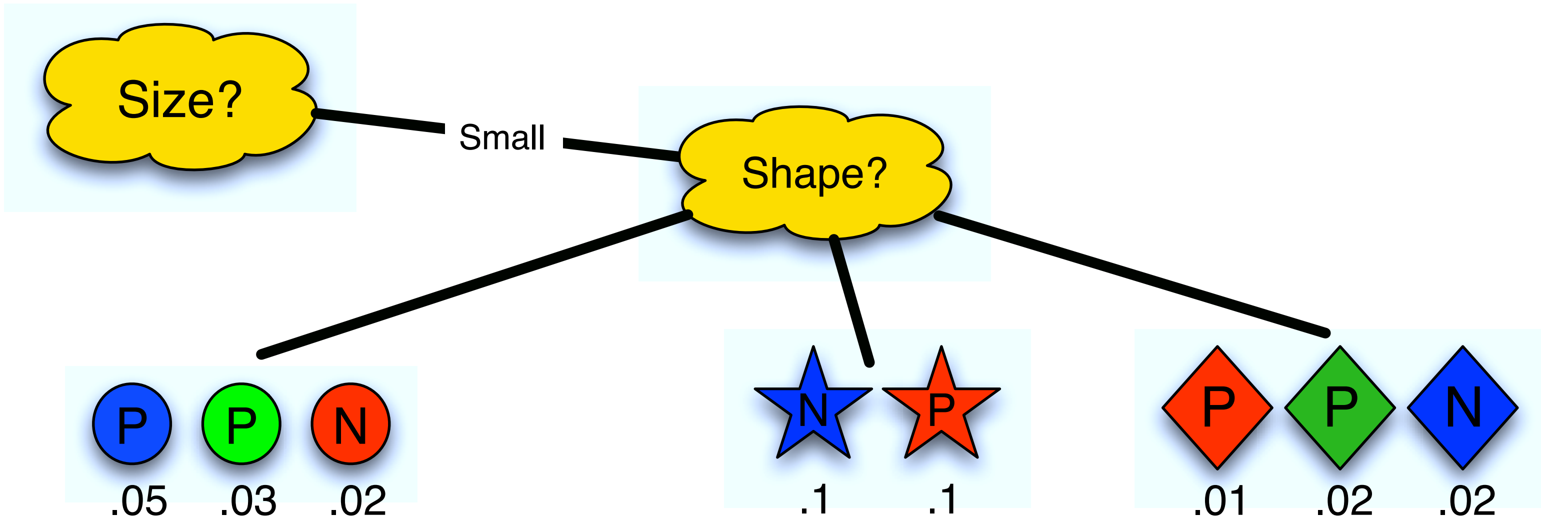
Weighted Examples in Boosting



Total weight of subtree examples = 0.35
Weighted Expected Information Needs for **Shape?**

- Circle: $\frac{0.1}{0.35}(-0.666 \log_2 0.666 + -0.333 \log_2 0.333) = .262$
- Star: $\frac{0.2}{0.35}(-0.5 \log_2 0.5 + -0.5 \log_2 0.5) = .571$
- Diamond: $\frac{0.05}{0.35}(-0.666 \log_2 0.666 + -0.333 \log_2 0.333) = .131$
- Total: 0.964

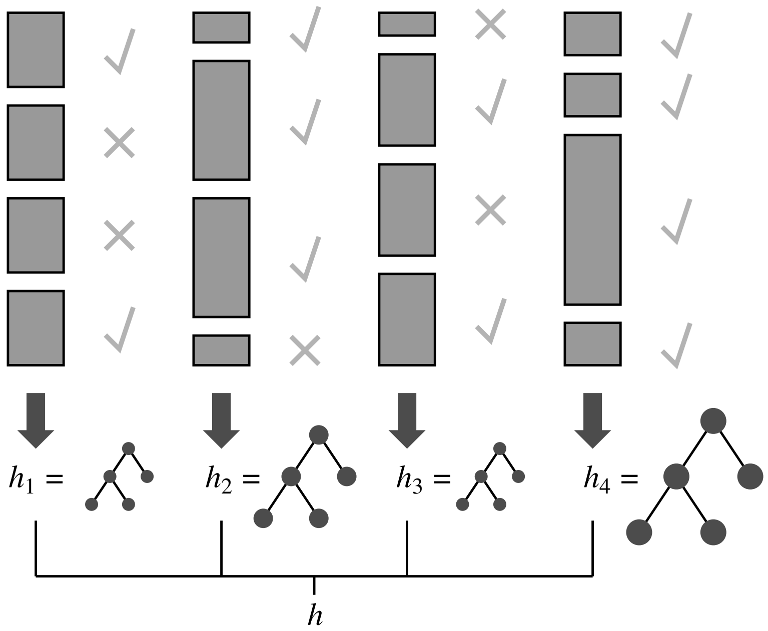
Weighted Examples in Boosting: Alternative 2



Total weight of subtree examples = 0.35
Here, we use weights in entropy calculations also.

- Circle: $\frac{0.1}{0.35}(-0.8\log_2 0.8 + -0.2\log_2 0.2) = .206$
- Star: $\frac{0.2}{0.35}(-0.5\log_2 0.5 + -0.5\log_2 0.5) = .571$
- Diamond: $\frac{0.05}{0.35}(-0.6\log_2 0.6 + -0.4\log_2 0.4) = .139$
- Total: 0.916

Evolving Example Weights in Boosting



Given hypothesis H and training examples $\{...(x_i, y_i)...\}$ with weights w_i.

$error \leftarrow 0$

$\forall i$: If $H(x_i) \neq y_i$ then $error \leftarrow error + w_i$

$\forall i$: If $H(x_i) = y_i$ then $w_i \leftarrow w_i(\frac{error}{1-error})$

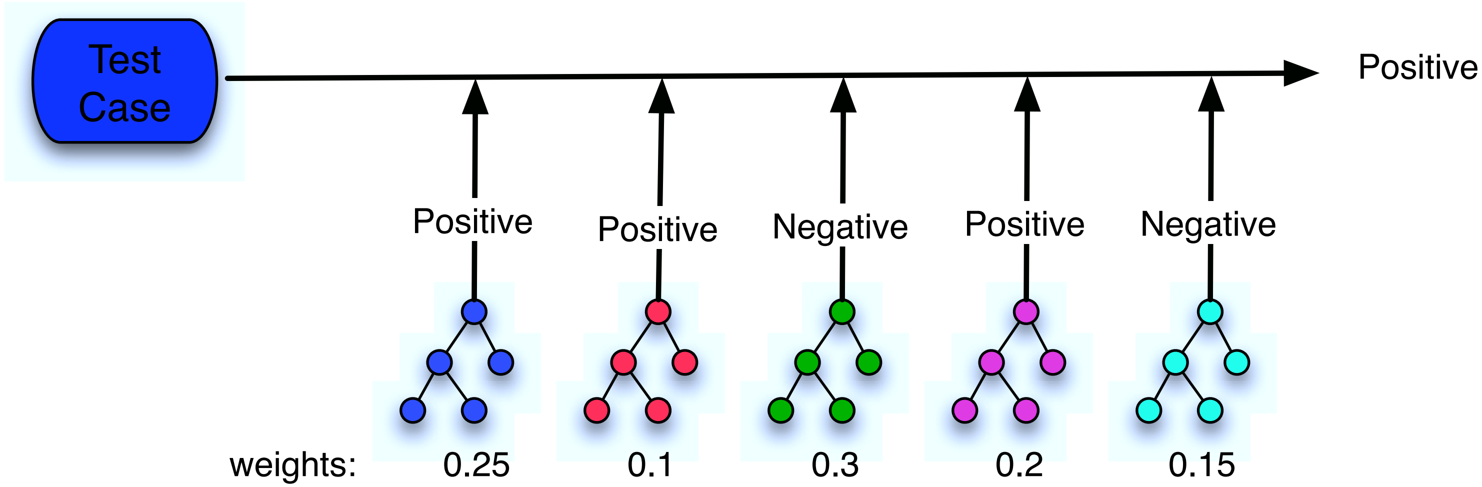
Total Error and Weight Updates

- The updated weights of the training examples are normalized after each hypothesis evaluation.
- So they always sum to 1.
- Using this update function, $w_i \leftarrow w_i(\frac{error}{1-error})$ for instances that are correctly solved by hypothesis H:
  - If error $= 0.5$, then w_i does not change.
  - If error < 0.5, then w_i decreases. Hence, after normalization, weights of unsolved examples will increase, thus increasing their importance.
  - If error > 0.5, then w_i increases. Hence, after normalization, weights of unsolved examples will decrease. Here, there is so much error that the solved examples need to be emphasized.

The Ensemble Classifier

Voting Result:

Positive: 0.25 + 0.1 + 0.2 = 0.55
Negative: 0.3 + 0.15 = 0.45



Training and Test Sets

- Given a data set, S, consisting of many instances.
- Each instance has attributes plus an answer.
  - Robotics: attributes = sensor readings, answer = correct action.
  - Medicine: attributes = symptoms, answer = disease.
  - Classification: attributes = object features, answer = object class.
- Divide S into S_{train} and S_{test}. Often with 75% or more of S in S_{train}.
- Use S_{train} as input to the hypothesis generator.
- Each $s \in S_{train}$ may be processed MANY times during **training**, i.e., the formation and refinement of an h.
- To test h, find $h(s) \forall s \in S_{test}$. Hopefully, h will get most of these correct, even though it has not seen them before. I.e., h should **generalize** from the training examples.
- **Overtraining:** h becomes overly specialized for $s \in S_{train}$ so that it cannot handle much in S_{test}.

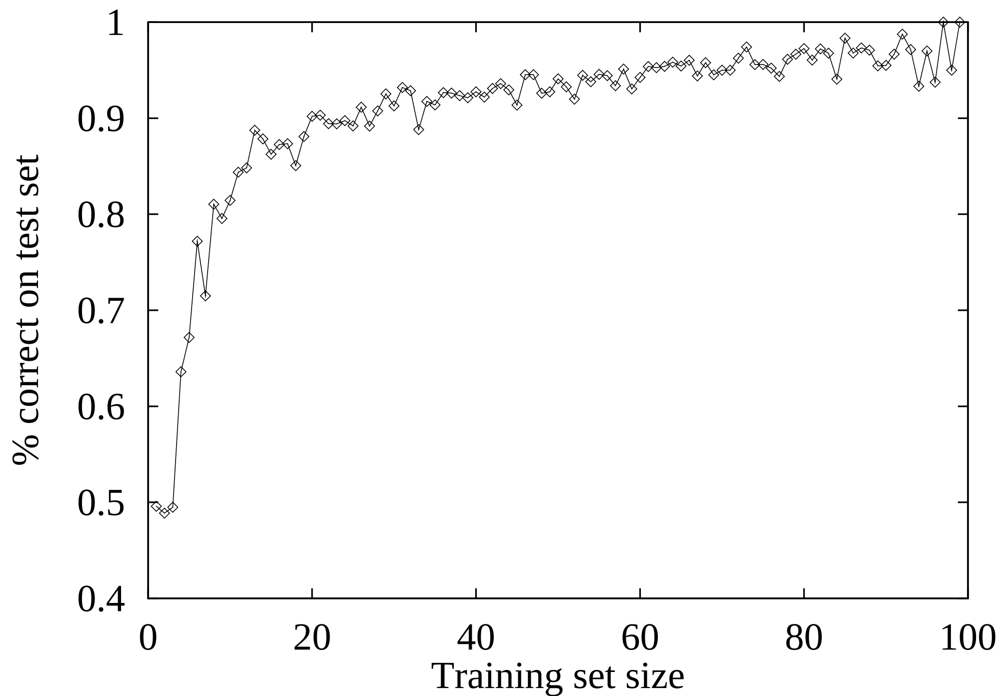
Performance measurement

How do we know that $h \approx f$? (Hume's **Problem of Induction**)

1) Use theorems of computational/statistical learning theory

2) Try h on a new test set of examples
   (use **same distribution over example space** as training set)

Learning curve $=$ % correct on test set as a function of training set size

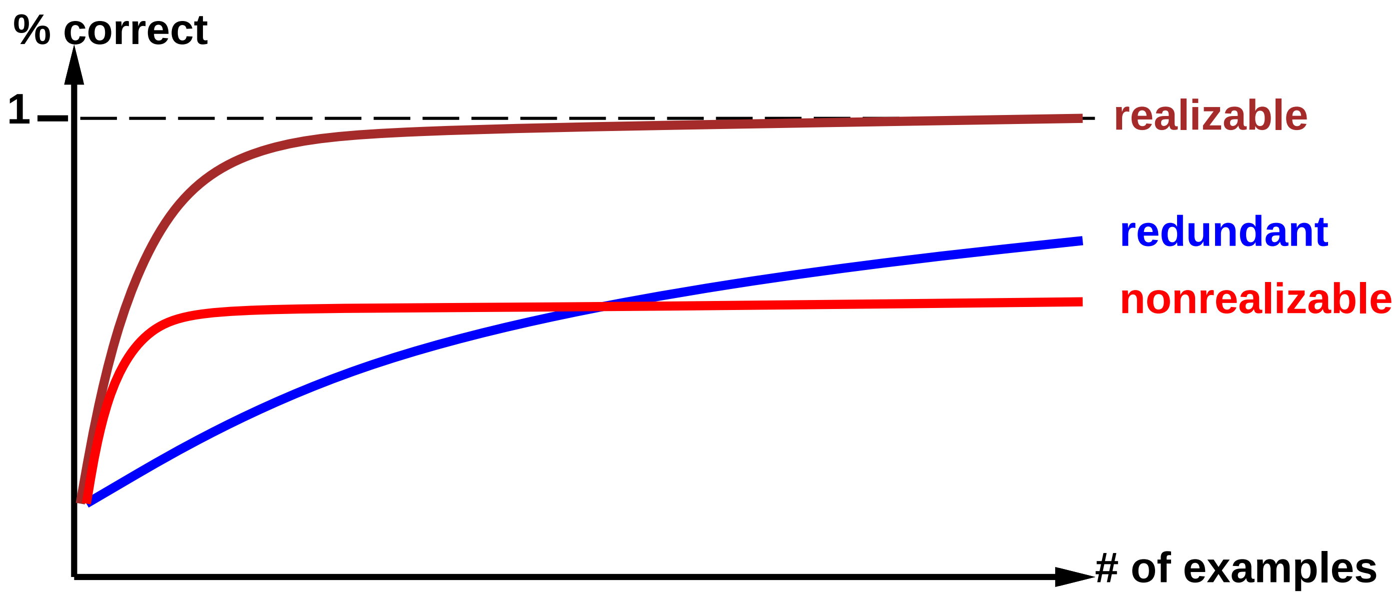


Performance measurement (2)

Learning curve depends on

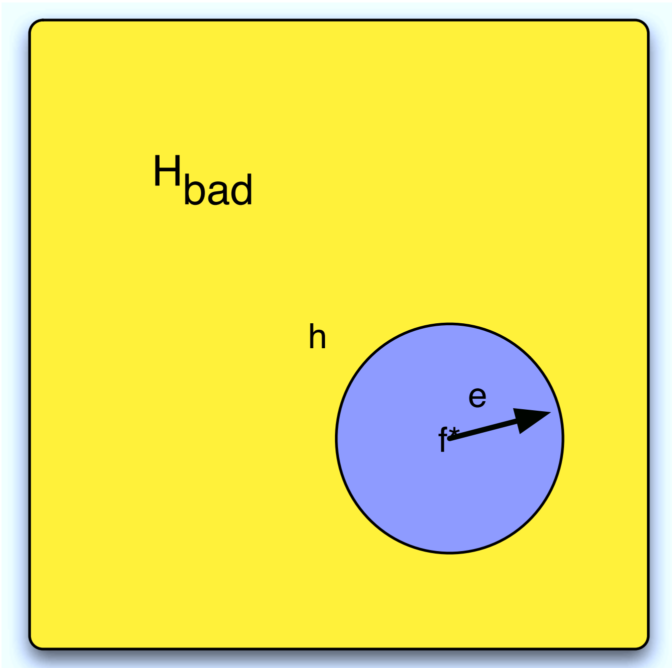
– realizable (can express target function) vs. non-realizable
  non-realizability can be due to missing attributes
  or restricted hypothesis class (e.g., thresholded linear function)

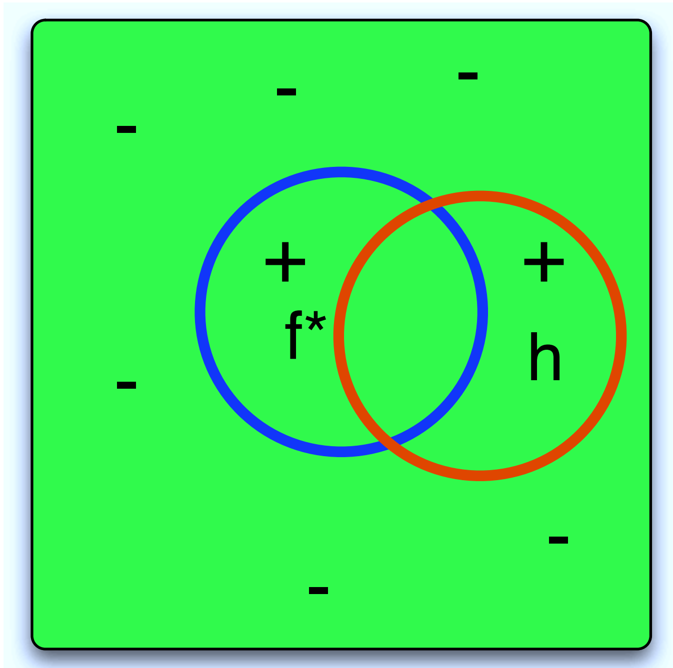
– redundant expressiveness (e.g., loads of irrelevant attributes)



Computational Learning Theory

- Although a learning system L may appear magical at times, it is not.
- In many cases, we can carefully analyze the situations in which L can and **should** perform well, along with those where it will probably fail.
- Computational Learning Theory (CLT) helps formalize these chances of success by doing general combinatorial analyses of hypothesis spaces and instance spaces.





PAC Hypothesis

- **Probably Approximately Correct (PAC)**: Correct on enough **training** instances that we have sufficient statistical confidence that it will also perform correctly on the **test** instances.
- **Stationary Assumption:** Training and test sets drawn from the same distribution over the example space. I.e., no deception (where L is trained on things irrelevant to testing).

Notation:

- X = set of all examples.
- D = distribution from which examples are drawn.
- H = set of possible hypotheses.
- N = number of examples in training set.
- f(x) = the **true** function that L tries to learn.

$\forall h \in H : error(h) = P(h(x) \neq f(x) \mid \text{x drawn from D})$

Avoiding Getting Fooled

- Find the probability p that a bad hypothesis, h, can actually perform perfectly on a set of N training instances.
- Set N sufficiently high so as to reduce p and insure that every consistent hypothesis (i.e. one that correctly handles all training cases) is a PAC hypothesis.

- Assume $error(h) > \epsilon$. So $h \in H_{bad}$.
- Then $P($h is consistent with a particular training instance$) \leq (1 - \epsilon)$.
- Then $P($h is consistent with N instances$) \leq (1 - \epsilon)^N$.
- So $P(H_{bad}$ contains a consistent hypothesis$) \leq \mid H_{bad} \mid (1 - \epsilon)^N$.
- And $\mid H_{bad} \mid (1 - \epsilon)^N \leq \mid H \mid (1 - \epsilon)^N$
- We want to pick an error threshold, δ and insure that $\mid H \mid (1 - \epsilon)^N \leq \delta$.

Avoiding Getting Fooled (2)

- Since $1 - \epsilon \leq e^{-\epsilon}$, $\mid H \mid (1-\epsilon)^N \leq \mid H \mid e^{-\epsilon N}$
- Now solve $\mid H \mid e^{-\epsilon N} \leq \delta$ for N:
  - Taking logs of both sides: $\ln \mid H \mid - \epsilon N (\ln e) \leq \ln \delta$
  - Rearranging: $-\epsilon N \leq \ln \delta - \ln \mid H \mid$
  - Dividing by $-\epsilon$, we get: $N \geq \frac{1}{\epsilon}(\ln \mid H \mid - \ln \delta)$
  - So: $N \geq \frac{1}{\epsilon}(\ln \mid H \mid + \ln \frac{1}{\delta})$
- So when N $\geq$ this threshold, $\mid H \mid (1-\epsilon)^N \leq \delta$.
- Or: 1 - $\mid H \mid (1-\epsilon)^N \geq 1 - \delta$.
- In other words: If L returns a hypothesis (h) that is consistent with N $\geq$ threshold instances, then there is a (1 - δ) probability that $error(h) \leq \epsilon$ (i.e. that h is within a pre-specified error bound).
- So for any H, we can select a desired ϵ and δ, and then compute the N that will give us that level of assurance.
- Note that for higher ϵ, there is less chance of being fooled, so $N \downarrow$.

CLT with Boolean Literals

- Given a set of n variables.
- The space of possible conjunctive hypotheses involving those literals has 3^n possible hypotheses.
- Since, for any variable, v, a hypothesis either contains v or not(v), or it makes no reference to v.

Insuring a PAC hypothesis in Boolean-Literal Space.

- Assume a space of with conjunctions of up to 10 variables, so $\mid H \mid = 3^{10}$.
- Assume that we want a 95% certainty that L will produce a consistent hypothesis whose error is no more than 0.1.
- Thus, $\epsilon = 0.1$ and $\delta = 1 - 0.95 = 0.05$
- So: $N \geq \frac{1}{0.1}(\ln 3^{10} + \ln \frac{1}{.05}) = 10(10 \ln 3 + \ln 20) \approx 140.$
- Conclusion, by using 140 random instances, we have 95% certainty that our final hypothesis has no more than a 10% chance of misclassifying an example. With 280 instances, $error(h) \leq 0.05$.

Summary

Learning needed for unknown environments, lazy designers

Learning agent = performance element + learning element

Learning method depends on type of performance element, available feedback, type of component to be improved, and its representation

For supervised learning, the aim is to find a simple hypothesis approximately consistent with training examples

Decision tree learning using information gain

Learning performance = prediction accuracy measured on test set