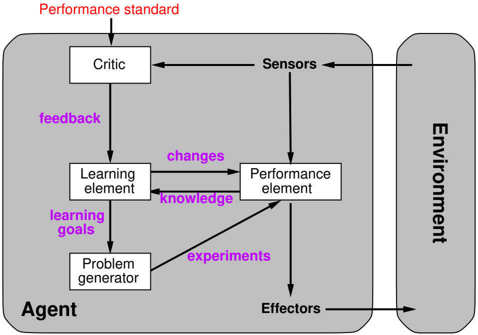
CS 316: Learning from observations

Stefan D. Bruda

Winter 2023

LEARNING AGENTS

- Learning is essential for unknown environments/lazy designers
  - 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 agent = performance element + learning element



LEARNING ELEMENT

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

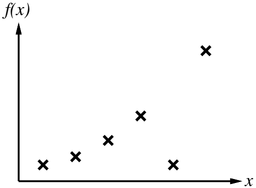
- Design of learning element is dictated by
  - what type of performance element is used
  - which functional component is to be learned
  - how that functional compoent is represented
  - what kind of feedback is available
- Example scenarios:

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

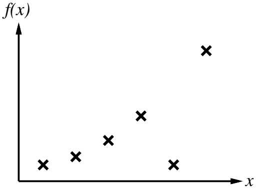
- Supervised learning: correct answers for each instance
- Reinforcement learning: occasional rewards

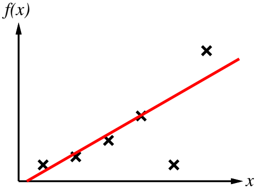
- Aka the method of natural science
- Simplest form: learn a function from examples (tabula rasa)
  - f is the target function
  - An example is a pair $x, f(x)$, e.g., $\begin{array}{c|c|c} O & O & X \\ \hline & X & \\ \hline X & & \end{array}$, $+1$
  - Problem: find a hypothesis h such that $h \approx f$ 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?)
- The aim of supervised learning is to find a simple hypothesis that is approximately consistent with the training examples

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

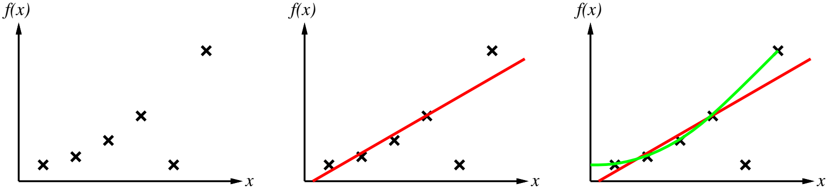


- Construct/adjust *h* to agree with *f* on training set
  - *h* is consistent if it agrees with *f* on all examples

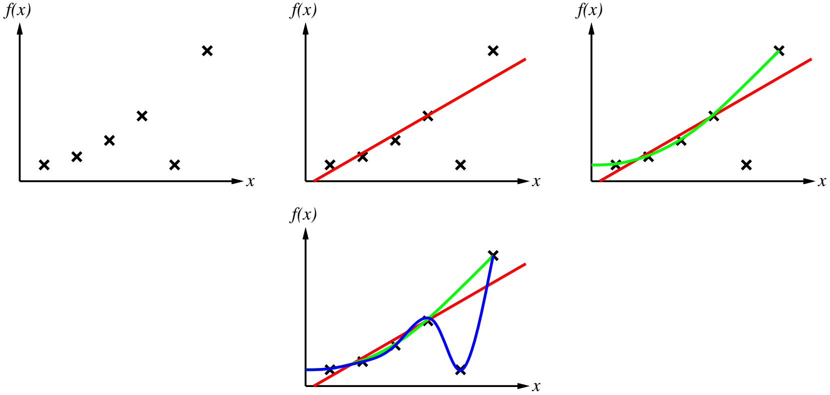




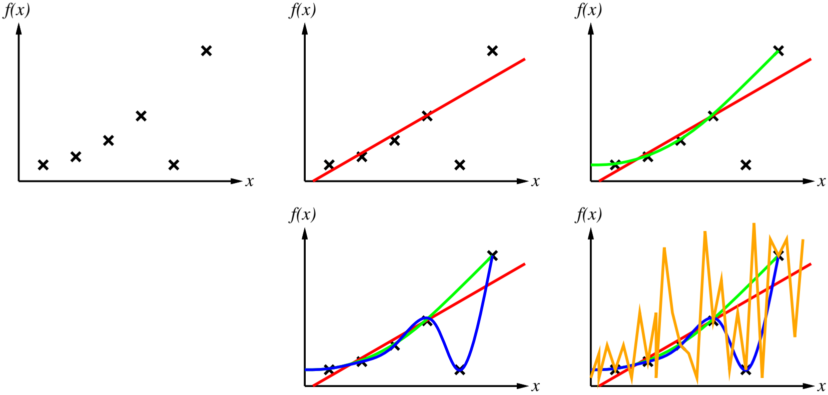
- Construct/adjust h to agree with f on training set
  - h is consistent if it agrees with f on all examples



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

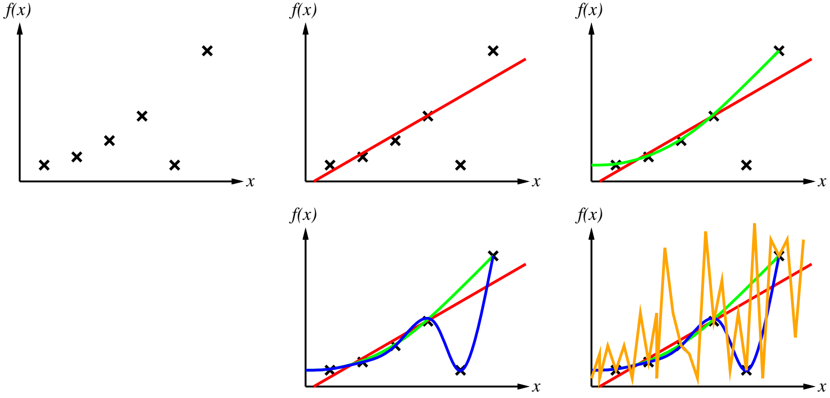


- Construct/adjust *h* to agree with *f* on training set
  - *h* is consistent if it agrees with *f* on all examples



The Inductive Learning Method

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



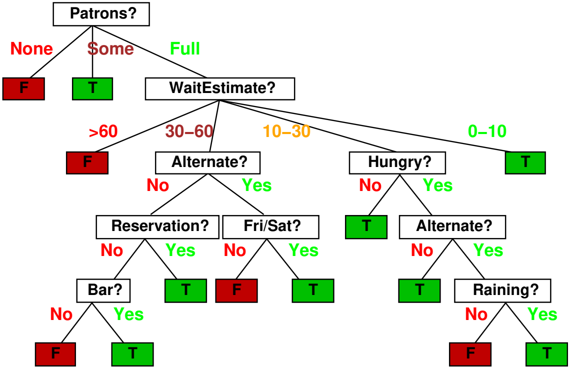
- Occam's razor: maximize a combination of consistency and simplicity

- 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)

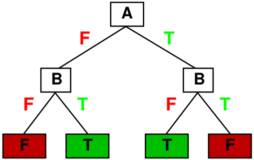
- One possible representation for hypotheses
  - E.g., here is the “true” tree for deciding whether to wait:



EXPRESSIVENESS

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

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



- Trivially, there is a consistent decision tree for any training set
  - One path to leaf for each example (unless *f* is nondeterministic in *x*)
  - But it probably won't generalize to new examples
- Prefer to find more compact decision trees

HYPOTHESIS SPACE

- How many distinct decision trees with *n* Boolean attributes?

- How many distinct decision trees with n Boolean attributes?
  - = number of Boolean functions

- 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., $Hungry \wedge \neg Rain$)?

Hypothesis space

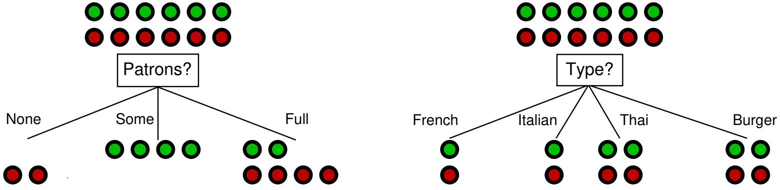
- How many distinct decision trees with n Boolean attributes?
  - = number of Boolean functions
  - = number of distinct truth tables with 2^n rows = 2^{2^n}
  - E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees
- How many purely conjunctive hypotheses (e.g., $Hungry \wedge \neg Rain$)?
  - 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, but also:
  - Increases number of hypotheses consistent with a training set $\Rightarrow$ may get worse predictions

- Aim: find a small tree consistent with the training examples
- Idea: recursively choose "most significant" attribute as the root of the (sub)tree

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

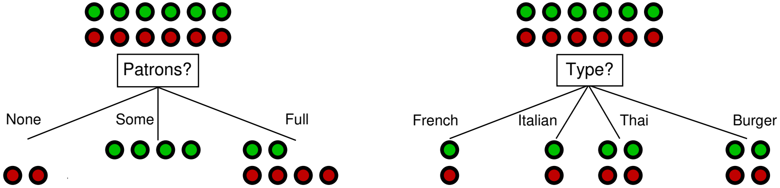
if *examples* is empty **then return** *default*
else if all *examples* have the same classification **then**
  return the classification
else if *attributes* is empty **then return** MODE(*examples*)
else
  best ← CHOOSE-ATTRIBUTE(*attributes*, *examples*)
  tree ← a new decision tree with root test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

- 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 gain)

- A good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



  - *Patrons*? is a better choice $\rightarrow$ gives information about the classification (information gain)
- 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 $\langle P_1, \ldots, P_n \rangle$:

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

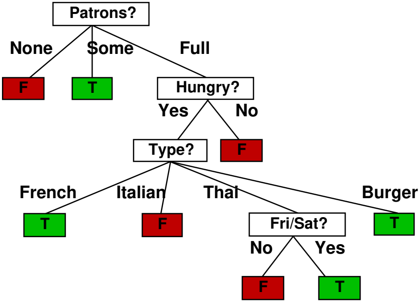
  (also called the entropy of the prior)

- 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

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

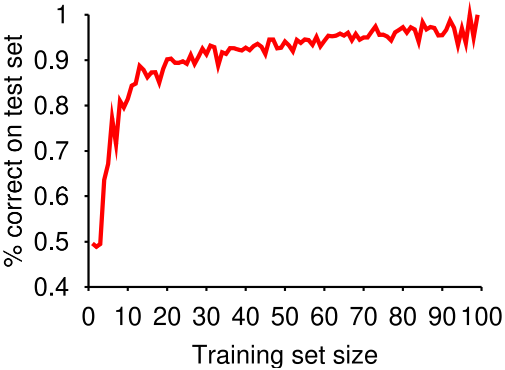
  - For *Patrons*?, this is 0.459 bits, for *Type* this is (still) 1 bit
  - $\Rightarrow$ choose the attribute that minimizes the remaining information needed

- Decision tree learned from the 12 examples:

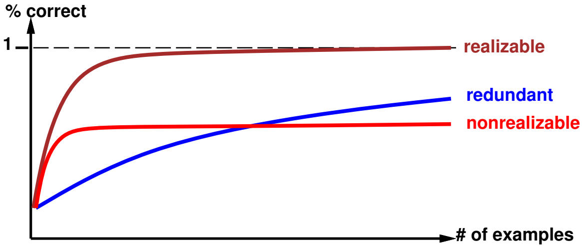


- Substantially simpler than "true" tree; a more complex hypothesis is not justified by small amount of data

- 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



- Learning curve parameters:
  - 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)



- Learning performance = prediction accuracy measured on test set