# Learning

## Lecture 10

# Outline

# Outline

# The Performance Element

- So far, we have only looked at the so-called performance element of an agent.
- The performance element maps percepts to actions.
  - In so doing, the agent's choice of action may be goal-directed or utility-based.

# Learning Agents

- A general learning agent consists of the following components:
  1. A performance element.
  2. A critic.
     - Evaluates the choice of the performance element based on some *impartial* performance measure.
  3. A learning element.
     - Collects information from the critic and knowledge about the performance element to suggest modifications to the performance element.
  4. A problem generator.
     - Suggests exploratory actions (every now and then) that might lead to the discovery of better performance policies.

# Components of the Performance Element

- The performance element may consist of several components:
  - A direct mapping from conditions on current state to actions.
  - A mapping from percept sequence to relevant properties of the world.
  - Information about how the world evolves.
  - Information about the effects of actions.
  - . . .

# The Learning Problem

- Each of the components of the performance element may be thought of as a function.
- The learning problem is to construct accurate representations of these functions.
- The agent always has a representation of the function; learning makes it more accurate.
- Available feedback and prior knowledge give rise to classes of learning problems.

# Available Feedback

- Learning problems may be classified according to the feedback available:
  1. Some correct input-output pairs are known: supervised learning.
  2. An evaluation of the output of the function is available: reinforcement learning.
  3. No evaluation of the output is available: unsupervised learning.
- We shall concentrate on supervised learning.

# Outline

# Induction

- Inductive learning is similar to curve fitting.
  - Could be identical to curve fitting if the function to be learned is continuous.
- We need to learn some unary function $f$.
- We are a given a training set $\{(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n))\}$.
- Each pair is called an example.
- Induction should produce a function $h$ that approximates $f$. $h$ is called a hypothesis.

# Example: Reflex Agent

- $f$ is a mapping form percepts to actions.
- An example is a percept-action pair provided by a teacher.
- $h$ is a function that agrees with $f$ on all examples.

# Example: Reflex Agent

global *examples* $\longleftarrow$ {}

function REFLEX-PERFORMANCE-ELEMENT(*percept*) returns action
    If (*percept*, *a*) $\in$ *examples* then return *a*
    *h* $\longleftarrow$ INDUCE(*examples*)
    return *h*(*percept*)

procedure REFLEX-LEARNING-ELEMENT(*percept*, *action*)
    *examples* $\longleftarrow$ *examples* $\cup$ {(*percept*, *action*)}

# Outline

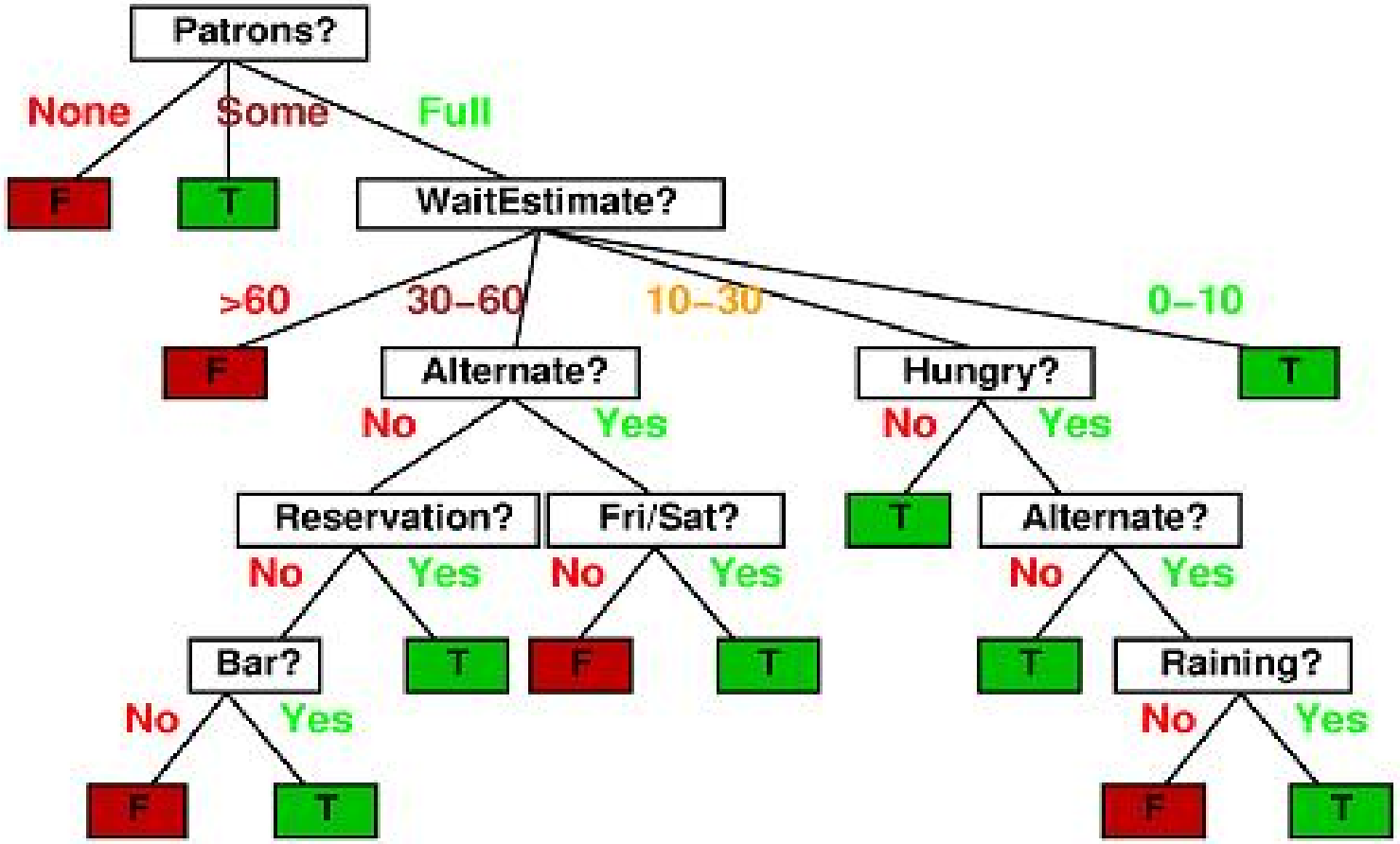
# What Is a Decision Tree?

- A decision tree is a special kind of performance element.
- Essentially, a Boolean function.
- Given the values of some attributes, is a certain goal predicate true or false?
- For example, this can model whether a certain action is to be performed.

# More Precisely

- Suppose we have a set $\mathcal{A}$ of attributes $A_1, A_2, \ldots A_m$.
- Attribute $A_i$ can assume values from a set $V_i$.
- The decision tree represents a Boolean function $f$ of the attribute values.
- How many such functions are possible?

# Sample Decision Tree



# Decision Trees as Sets of Sentences

- We can think of a decision tree as a set of implications:
  - $Patrons(None) \Rightarrow \neg Wait$
  - $Patrons(Some) \Rightarrow Wait$
  - $Patrons(Full) \wedge WaitEstimate(30-60) \wedge Alternate(N) \wedge Reservation(Y) \Rightarrow Wait$
  - $\ldots$

# Decision Tree Algorithm

global *Tree*

function DECISION-TREE($V$, *root*) returns Boolean
    If LEAF(*root*, *Tree*) then return LABEL(*root*)
    $e \longleftarrow V$[LABEL(*root*)]
    $n \longleftarrow$ CHILDOF(*root*, $e$, *Tree*)
    return DECISION-TREE($V$, $n$)

# Outline

# Decision Tree Examples

- An example is a pair $((v_1, v_2, \ldots, v_m), f(v_1, v_2, \ldots, v_m))$, where $v_i \in V_i$ is the value of $A_i$.
- An example is a positive example if $f(v_1, v_2, \ldots, v_m)$ is *true*. Otherwise it is a negative example.

# Sample Training Set

| E.g. | Attributes | | | | | | | | | | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Alt* | *Bar* | *Fri* | *Hun* | *Pat* | *Price* | *Rain* | *Res* | *Type* | *Est* | *Wait* |
| $X_1$ | *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* |

GUC German University in Cairo

# Trivial Tree

- One path from the root to a leaf for each example.
- This tree is consistent with the training set.
- Unfortunately, it is not very helpful with new cases.
- It just memorizes the entire training set, without representing any possible patterns therein.

# Ockham's Razor

## Ockham's Razor

The most likely hypothesis is the simplest one consistent with the training set.

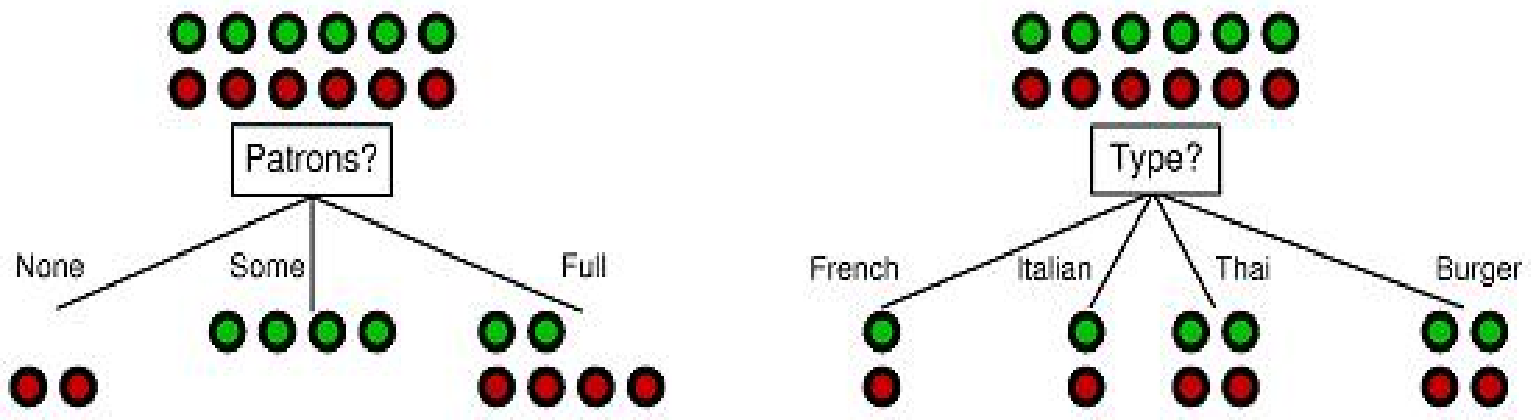
- Not only should we look for a consistent tree, but also for a small one.
- Finding a small tree implies finding a general pattern.
- Unfortunately, the problem of finding the smallest tree is intractable.

# Algorithm Idea

- Need to construct a small tree.
- Depends on the order in which we test for attribute values.
- Be greedy and start with the attribute that makes the *most difference* to the classification of an example.
- Making the most difference roughly means reducing the number of subsequent tests.

# Good Attributes

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




Clearly, *Patrons*? is a better attribute.

# Decision Tree Learning

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* $\longleftarrow$ CHOOSE-ATTRIBUTE(*attributes*, *examples*)
    *tree* $\longleftarrow$ a new decision tree with root label *best*
    for each value $v_i$ of *best* do
      $examples_i \longleftarrow$ {elements of *examples* with $best = v_i$}
      *subtree* $\longleftarrow$ DTL($examples_i$, $attributes - best$, MODE(*examples*))
      add a branch to *tree* with label $v_i$ and subtree *subtree*
    return *tree*

# Attribute Choice

- How do we choose a good attribute?
- Well, it is the attribute that provides more information.
- *Patrons*? provides more information than any other attribute.
- Need to be more precise about "information".

# Information Theory Basics

- Think of information as answering a question.
- The more I know about the expected answer, the less informative the actual answer is.

## Example

- If I have no clue about whether my paper will be accepted, a letter from the reviewers is very informative.
- If I already know that the paper will be accepted, a letter from the reviewers carries no information.

# Entropy

- Information content, or entropy, is measured in bits.
- Scale: 1 bit represents the entropy of an answer to a yes/no question about which one is clueless.
- In general, if a question has a set $V = \{v_1, v_2, \ldots, v_k\}$ of possible answers, then the entropy of the actual answer is

$$H(P_1, P_2, \ldots, P_k) = -\sum_{i=1}^{k} P_i \log_2 P_i$$

  where $P_i = P(v_i)$
- $P_i \log_2 P_i = 0$ when $P_i = 0$.
- Note that if $k = 2$, then the maximum entropy is 1 bit.

# Entropy and Decision Trees (I)

- For a given tuple of attribute values, the question is whether the query predicate is *true* or *false*.
- Before any of the attributes are tested, the only estimate we have for the probabilities of the two answers is the proportions of positive and negative examples in the training set.
- Thus, an estimate of the entropy of a correct answer is

$$H(\frac{p}{p+n}, \frac{n}{p+n})$$

where $p$ is the number of positive examples and $n$ is the number of negative examples.

- This number represents how much information we initially need.

# Entropy and Decision Trees (II)

- Testing for an attribute will give us some of this information.
  - All of it if the attribute is perfect.
- Each attribute divides the training set into $k$ subsets, one corresponding to each possible value $v_i$.
- Subset $E_i$ has $p_i$ positive examples and $n_i$ negative examples.
- Thus, along the $v_i$ branch, we need

$$H\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

bits of information to answer the question.

# Entropy and Decision Trees (III)

- An arbitrary example has value $v_i$ of an attribute with probability

$$\frac{p_i + n_i}{p + n}$$

- Thus, on average, after testing attribute $A$ we will need

$$Remainder(A) = \sum_{i=1}^{k} \frac{p_i + n_i}{p + n} H(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i})$$

bits of information.

# Entropy and Decision Trees (IV)

- The information we gain by testing attribute $A$ is given by

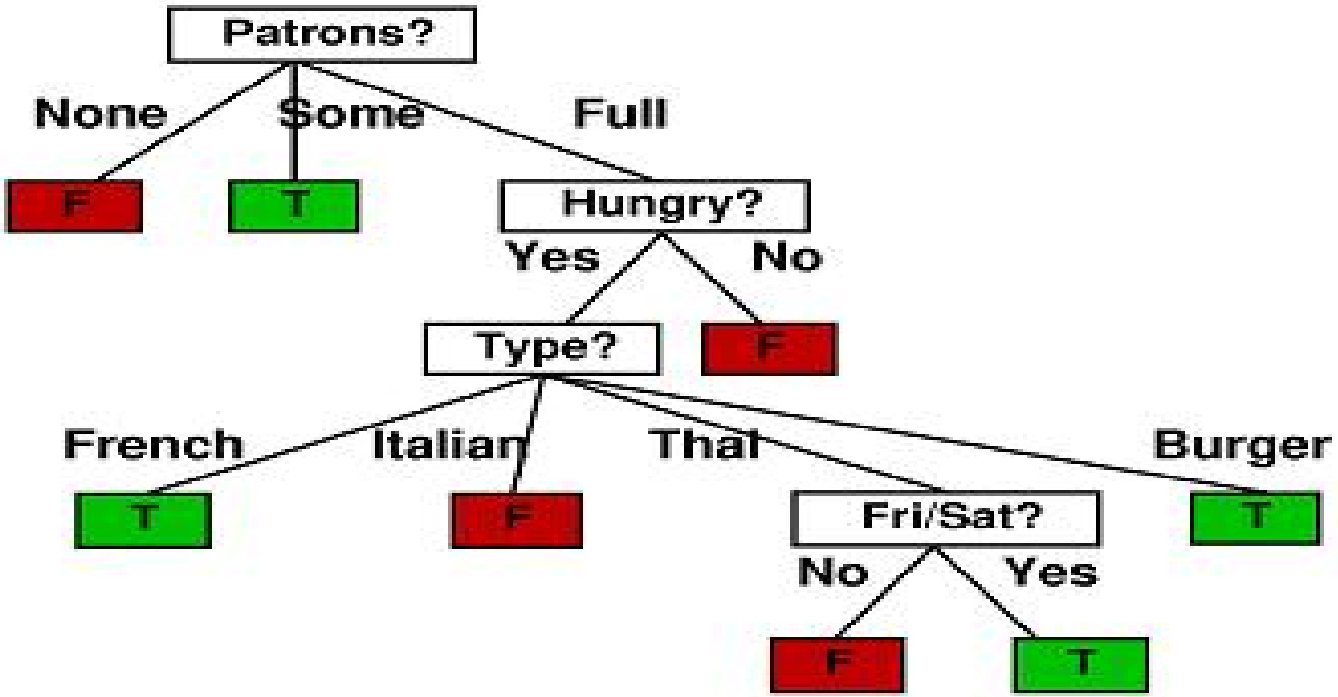
$$Gain(A) = H(\frac{p}{p+n}, \frac{n}{p+n}) - Remainder(A)$$

- Algorithm CHOOSE-ATTRIBUTE just picks the attribute with the highest gain.

## In the *Wait* example

- $Gain(Patrons) \approx 0.541$ bits
- $Gain(Type) = 0$ bits

# Induced Restaurant Tree




Much simpler than the true tree, but appropriate for the given training set.