Learning Checkers as Finding a Function

What should a checker player learn?

At the base level, it needs to know:

- how to search a game tree
- how to evaluate states in the tree

At the meta level, it needs to know:

- how to extend its search at the right states
- how to manage its time

Representing Checkers Knowledge

The program must assign values to states, say:

• 100	=	“I know I will win”
• 0	=	“I expect to draw”
• —100	=	“I know I will lose”

It might try to learn a function such as:

$$\begin{aligned} V(b) = w0 &+ w_1bp(b) + w_2rp(b) \\ &+ w_3bk(b) + w_4rk(b) \\ &+ w_5bt(b) + w_6bt(b) \end{aligned}$$

where:

- bp(b) is the number of black pieces on board b
- wp(b) is the number of white pieces on board b
- bk(b) is the number of black kings on board b
- wk(b) is the number of white kings on board b
- bt(b) is the number of black threats on board b
- rt(b) is the number of white threats on board b

and the w_i’s are weights assigned to the features

Some Fundamental Assumptions of Research on Machine Learning

Learning is about understanding the world.

Search and inference are about acting in the world. They assume that the agent already has a reasonable model of the world.

Learning is about finding patterns.

Search and inference are about choosing actions based on known patterns.

An Exercise

Consider the problem faced by an infant learning to speak and understand language. Explain how this process fits into our general model of learning, and identify the form and content of each of the components of the model for this task.

---

insert figure from Russell and Norvig

Don’t try to solve the problem. Just express it in the terms of our model of a learning agent.

On the Exercise

The first step you face is to appreciate the variety of state and behavior implicit in the task:

- recognize spoken words
- generate spoken words
- learn the vocabulary
- learn idioms
- learn the grammar of sentences
- learn the semantics of sentences
- learn the pragmatics of communication
- learn disambiguation strategies for pronouns
- …

These issues are still active areas of research into human understanding of language. They raise questions of

- “nature versus nurture”
- “meaning versus reference”

Not too surprisingly, implementing a program that can do any of them is still quite difficult.

Inductive Learning

Basic Statement:

Given a set of data vectors

$$d_i = [\ d_{i1},\ d_{i2},\ d_{i3},\ \ldots,\ d_{in},\ v_i\ ]$$

learn that

$$f(d_i) = v$$

Because many potential functions f_j satisfy this definition, we will settle for any function f' that closely predicts the input vectors.

Assumptions:

- The process is done in “batch mode”, not incrementally.
- The input vectors are “perfect”—all values are present and correct.

Questions that must be answered:

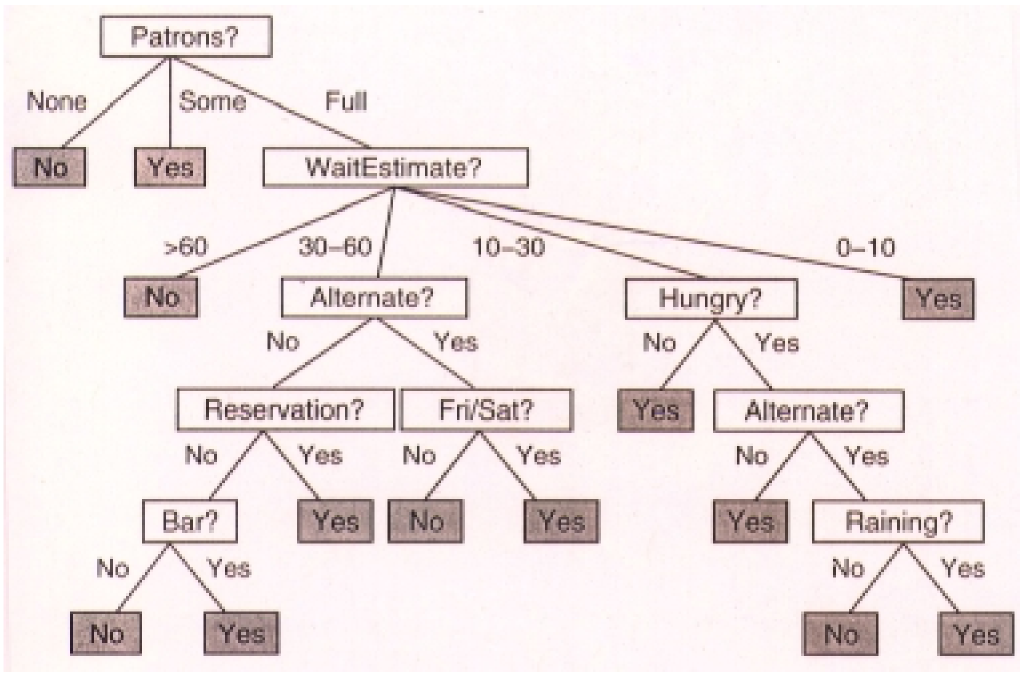
- What kinds of values can be in an input vector?
- What kinds of functions are we willing to learn?

A Context for Induction: Decision Trees

Input: an *n*-ary vector of **problem attributes**

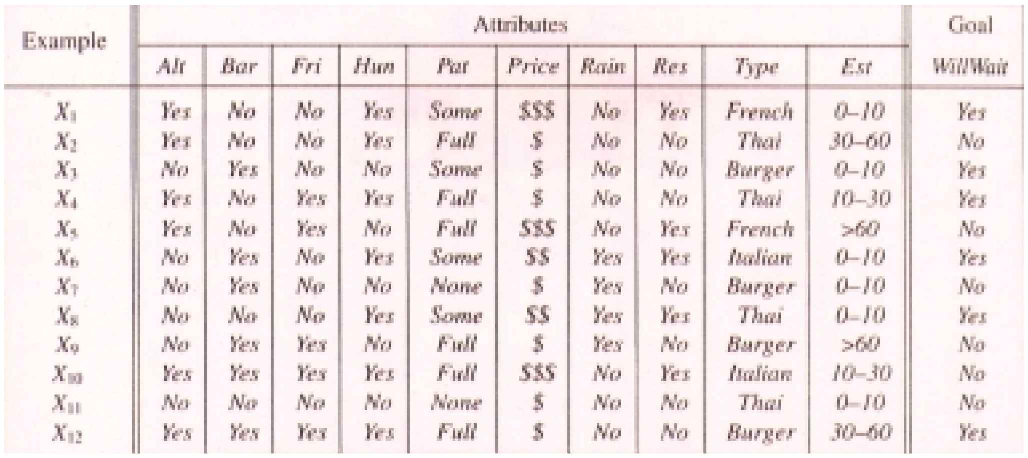
Output: the value of the **target attribute**

Here is an example of a decision tree, for the task of deciding whether or not to wait for a seat at a restaurant, given the features of a visit:



Input: A Set of Feature Vectors

Here is a set of feature vectors, from which a learning agent might try to build the “Will wait?” decision tree in our restaurant example:



Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

The features in our vector are:

- Do we have an <u>alternative</u> restaurant?
- Does this restaurant have a <u>bar</u>?
- Is today a <u>Friday</u> or Saturday night?
- Are we quite <u>hungry</u>?
- How many <u>patrons</u> are at this restaurant?
- What is the <u>price</u> range for meals at this restaurant?
- Is it <u>raining</u> outside?
- Do we have <u>reservations</u>?
- What <u>type</u> of food do they serve at this restaurant?
- What is the <u>estimated</u> wait before we will be seated?

Characteristics of Decision Trees

---

essentially propositional

---

work best when many combinations mean the same

---

cannot represent many functions efficiently

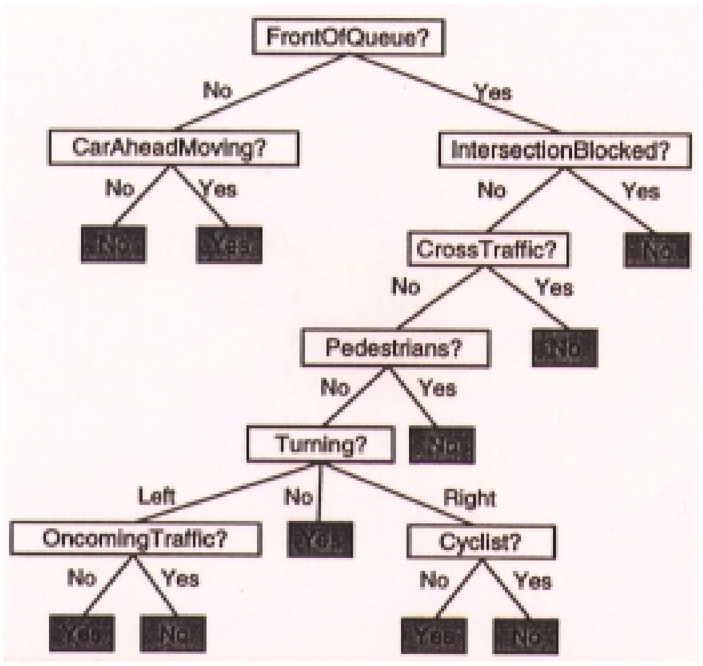
Example of when decision trees don’t work well:

Input:	string of ones and zeros
Output:	Have we seen more ones than zeros?

An Exercise

Construct a decision tree that can decide whether or not to drive forward at an intersection with a stoplight.

A Possible Solution



This problem is deceptively simple—unless we get caught up in all of the possible exceptions that can occur.

Real-world situations can be full of exceptions. Decision trees handle exceptions badly. They result in trees that are unbalanced and very deep on the main branches.

You can imagine many more exceptions than this tree handles, but would we want to include them all in the tree?

This is (another) form of the **qualification problem**.

“Inducing” a Decision Tree

In order to learn a decision tree, our agent will need to have some information to learn from:

a **training set** of examples

each example is described by its **values** for the problem’s attributes

each example is described by its output **value**, from the possible values of the target attribute

---

In the restaurant example, our problem attributes are “What is the estimated time?”, “What kind of food do they serve?”, and the like.

The target attribute is “Will we wait?” It is a boolean attribute: its value is either yes or no.

Problems with boolean target attributes are called **classification** problems. The learning agent is learning to recognize whether a situation is a positive example of some concept or a negative example.

Biases in Learning

Because there are, in general, an infinite number of functions f_i that satisfy the learner's goal, we need to give our agent a **bias** toward a particular class of functions.

Even if we think that we aren't giving our agent a bias, it will have one by default, by virtue of how we implement the program.

In the absence of any domain knowledge that suggests a good bias, we can give our agent a simple yet surprisingly good domain-independent bias:

Occam's Razor

The most likely solution is the simplest one that is consistent with all observations.

This principle biases our agent toward simplicity.

One good feature of simplicity: when the agent is wrong, it is generally **less wrong** than it would be with a more complex hypothesis.

(And it will be wrong.)