“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.

An 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*

Our agent's ideal goal is to find
the most efficient, correct decision tree.

Since most efficient is too hard, it will have to settle
for as efficient a tree as can be found in reasonable time.

---

input	*examples*	a training set
	attributes	a set of attributes
	default	the default goal predicate value
output		a decision tree

The Induction Algorithm

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

if all *examples* have the same classification,
    then return that classification

if *attributes* is empty,
    then return the most common classification
        of the remaining *examples*

choose the attribute *a* that **best discriminates** among the remaining *examples*

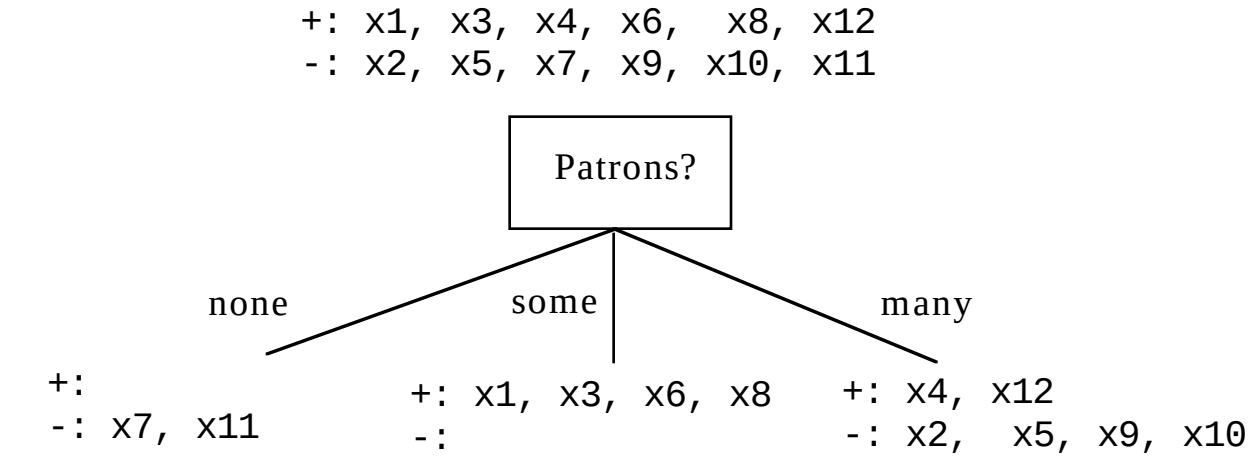
create a tree *t* with *a* as its root

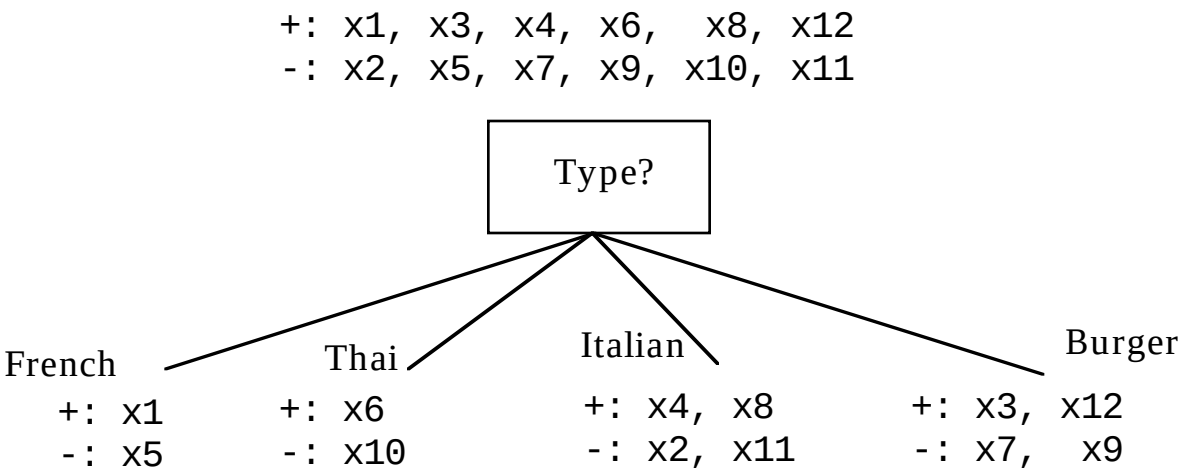
for each possible value *v* of *a*
    select the subset of examples *ex* having value(*a*) = *v*
    let subtree sub be the result of recursively calling
        the induction algorithm with *ex*, (*attributes* - *a*),
        and the most common classification of *ex*
    add a branch to *t* with label *v* and subtree *sub*

return *t*

[Assume, for now that "best discriminates" means "creates subsets of roughly equal size but with some subsets having members with a common answer".]

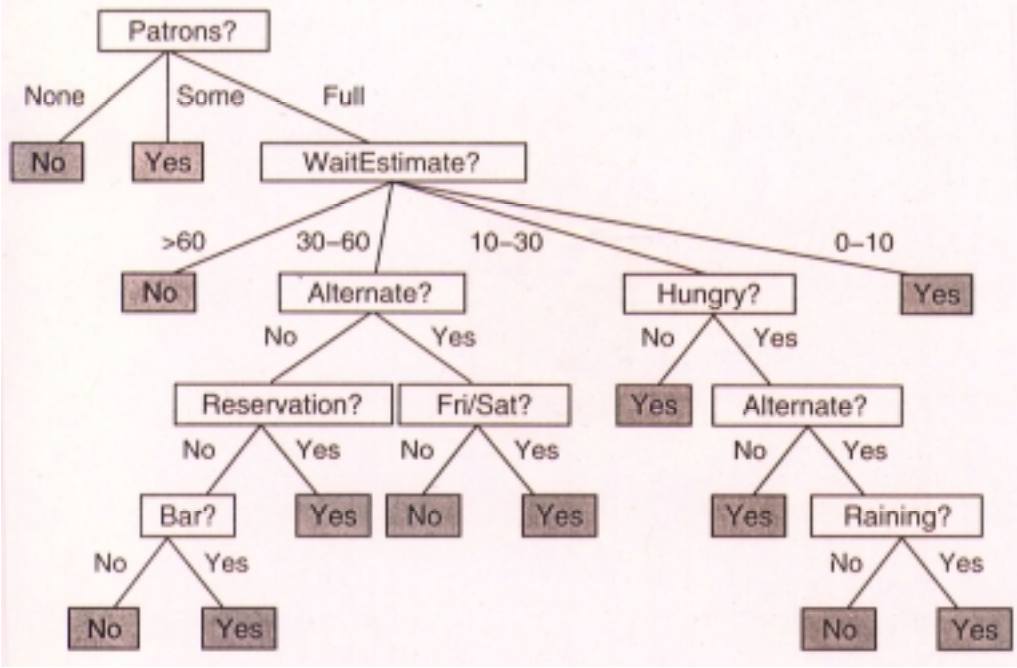
What Does “Best Discriminates” Mean?





Using information theory, we can compute a “right answer” to what discriminates best. Nilsson gives a simple approximation rule in Section 17.5.

The Answer...


Patrons?
None
Some
Full
No
Yes
WaitEstimate?
>60
30–60
10–30
0–10
No
Alternate?
Hungry?
Yes
No
Yes
No
Yes
Reservation?
Fri/Sat?
Yes
Alternate?
No
Yes
No
Yes
No
Yes
Bar?
Yes
No
Yes
Yes
Raining?
No
Yes
No
Yes
No
Yes
No
Yes

Evaluating a Learning Algorithm

A learning algorithm is good if it produces hypotheses that do a good job predicting the values of unseen cases.

One technique for evaluating a learning algorithm:

- Partition the set of cases into two sets: a training set and a test set.
- Run the algorithm on the training set to induce a decision tree.
- Evaluate the decision tree's performance when applied to the test set.

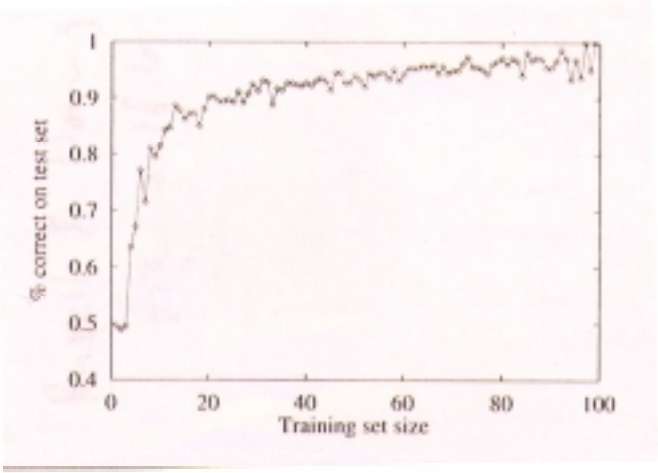
Experimental questions

- How do we split the case set? Size? Make-up?
- How good is good enough? Partial credit?

Evaluating the Induction Algorithm

Russell and Norvig ran an experiment on our table from the restaurant domain. They generated random sets of cases using the problem and target attributes. Then they ran 20 trials each for training set sizes of 1-100, with each training set chosen randomly from the set of all cases. On each trial, any case not in the training set was placed in the test set.

Here are the results:



This is called a ***happy graph***. There was a pattern, and the algorithm found it.

Questions:

- What would an unhappy graph look like?
- Can a learning agent learn too much?

Exercise: Build a Decision Tree

A number of patients have shown up at the local hospital emergency room complaining of certain symptoms. Our crack staff has identified the problem as an uncommon allergic reaction to a certain food. The patients all know each other, but some of their other friends have not had this reaction.

The doctors know how to treat the reaction, but they would also like to be able to suggest some dining guidelines to this group of people so that they can avoid the reaction if they choose.

Here is a set of case data on some members of the group of friends. Use our induction algorithm to build a decision tree for dining options...

Case #	Restaurant	Meal	Day	Cost	Reaction?
1	Sam's	breakfast	Saturday	cheap	yes
2	Lobdell	lunch	Saturday	expensive	no
3	Sam's	lunch	Sunday	cheap	yes
4	FooBarBaz	breakfast	Monday	cheap	no
5	Sam's	breakfast	Sunday	expensive	no

Toward a Solution

Reinforcement Learning

Induction is much different than the sort of learning that neural networks and genetic algorithms do. A program can do induction in batch from problem/solution pairs. Neural nets and GAs rely on interleaving learning with problem solving in order to get *feedback*.

Basic Statement:

An agent is given a sequence of trials for which it knows:

- the states it visited for each trial
- the payoff it received at the end of the trial

The agent has no knowledge of:

- the domain (full effects of actions)
- the payoff system

The agent is to learn

- the domain
- the expected value of payoff for each action
- a problem-solving policy

Types of Reinforcement Learning

Passive learning

> The agent has no real control over its actions. It wants to learn the expected values of states.

Active learning

> The agent can choose actions on its own. It wants to learn not only the expected values of states but also an optimal policy.

Model-based learning

> The agent learns the expected payoffs of each action and then tries to learn an optimal policy.

“Q learning”

> The agent tries to learn the optimal policy without knowing the payoffs or probabilities directly.
