Artificial Intelligence Programming

Decision Tree Learning

Cindi Thompson

Department of Computer Science

University of San Francisco

Recap: Learning

- Today, we'll talk about how to acquire knowledge from observation.
- Focus on learning knowledge that can be thought of or represented as propositional rules
  - $sunny \wedge warm \rightarrow PlayTennis$
  - $cool \wedge (rain \vee strongWind) \rightarrow \neg PlayTennis$

Deduction vs. Induction

- Up to now, we've looked at cases where our agent is given general knowledge and uses this to solve a particular problem.
  - Suck always cleans a room
- This general-to-specific reasoning is known as *deduction*.
- Advantage: deduction is sound, assuming your knowledge is correct.
- Sometimes, you may not have general information about a problem.
- Instead, you might have *data* about particular instances of a problem.
- the problem then is to figure out a general rule from specific data.

Example

- Recall the example of an agent deciding whether we should play tennis on a given day.
- There are four observable percepts (aka attributes):
  - Outlook (sunny, rainy, overcast)
  - Temperature (hot, mild, cool)
  - Humidity (high, low)
  - Wind (strong, weak)
- We don't have a model, but we do have some data about past decsions.
- Can we induce a general rule for when to play tennis?

Supervised Learning

- Recall that one way to characterize learning problems is by the type of feedback that is given to the learning agent.
  - In supervised learning, an external source (often called a teacher) provides the agent with *labeled examples*
  - Agent sees specific actions/cases, along with their classification.
- D2 was Sunny, mild, high humidity and weak wind. We played tennis.

Classification

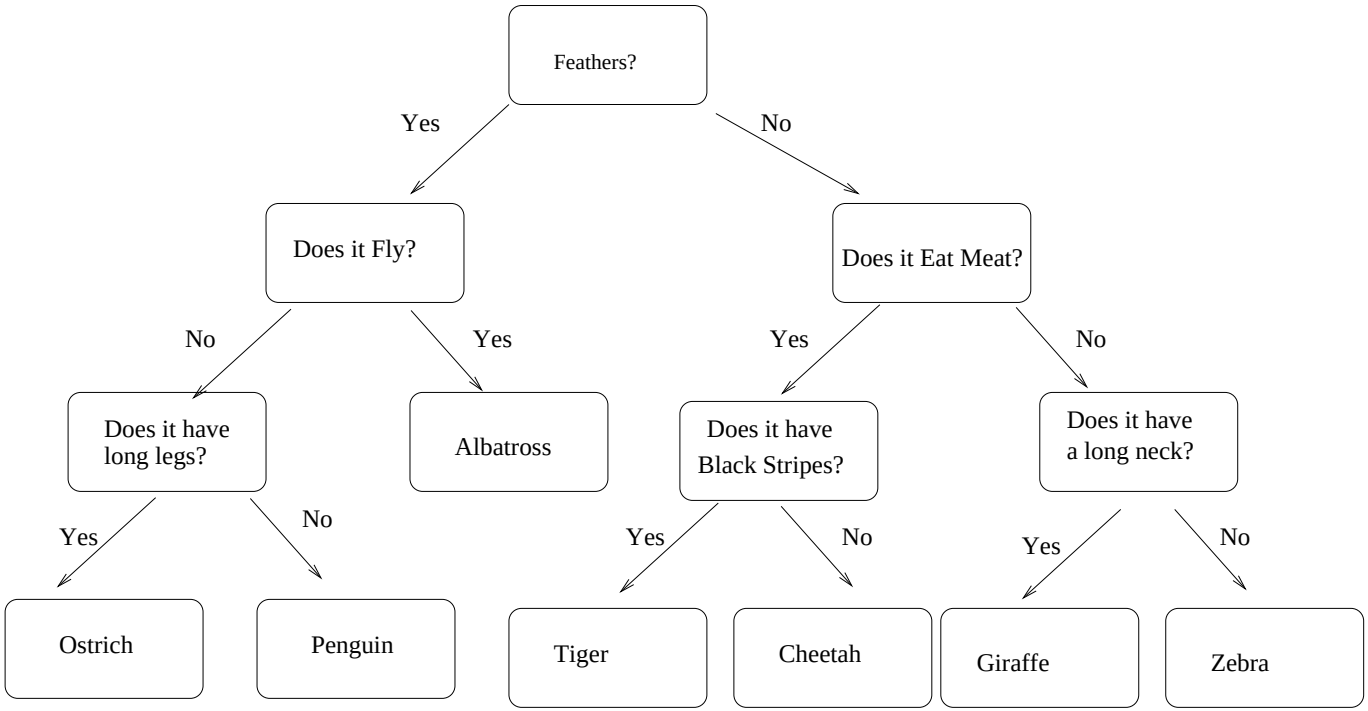
- the particular learning problem we are focusing on is sometimes known as *classification*
  - For a given input, determine which class it belongs to.
- Programs that can perform this task are referred to as *classifiers*

Defining the Learning Problem

- We can phrase the learning problem as that of estimating a function f that tells us how to classify a set of inputs.
- An example is a list of inputs $\mathbf{x}$ and the corresponding $f(\mathbf{x})$ - the class that $\mathbf{x}$ belongs to.
  - $<< Overcast, Cool, Low, Weak >, playTennis >$
- We can define the learning task as follows:
  - Given a collection of examples of f, find a function H that approximates f for our examples.
  - H is called a *hypothesis*.

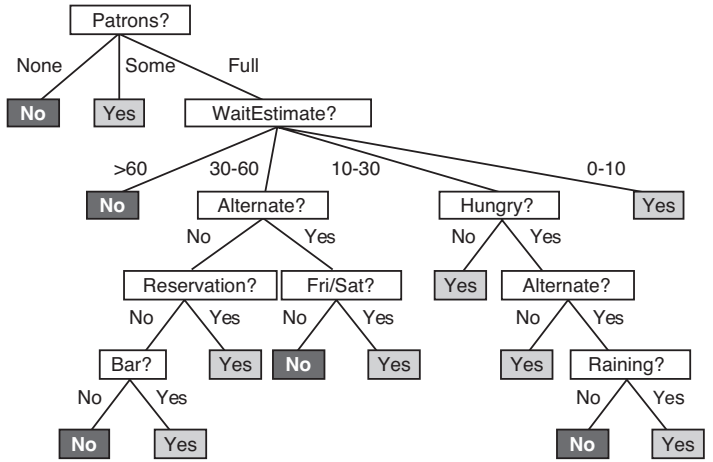
Learning Decision Trees

- Decision trees are data structures that provide an agent with a means of classifying examples.
- At each node in the tree, an attribute is tested.



Another Example

- R & N show a decision tree for determining whether to wait at a busy restaurant.
- The problem has the following inputs/attributes:
  - Alternative nearby
  - Has a bar
  - Day of week
  - Hungriness
  - Crowd
  - Price
  - Raining?
  - Reservation
  - Type of restaurant
  - Wait estimate



- Note that not all atributes are used.

Trees as rules

- A decision tree is just a compiled set of rules.
- We can rewrite the tree as a clause in which the path to the leaf is on the left, and the leaf is on the right.
  - $Wait30 \wedge reservation \rightarrow Stay$
  - $Wait10-30 \wedge Hungry \wedge \neg Alternate \rightarrow Stay$
  - $NoPatrons \rightarrow \neg Stay$
- The tree gives us a more efficient way of determining what to do - we're able to check several rules at once.

An example training set

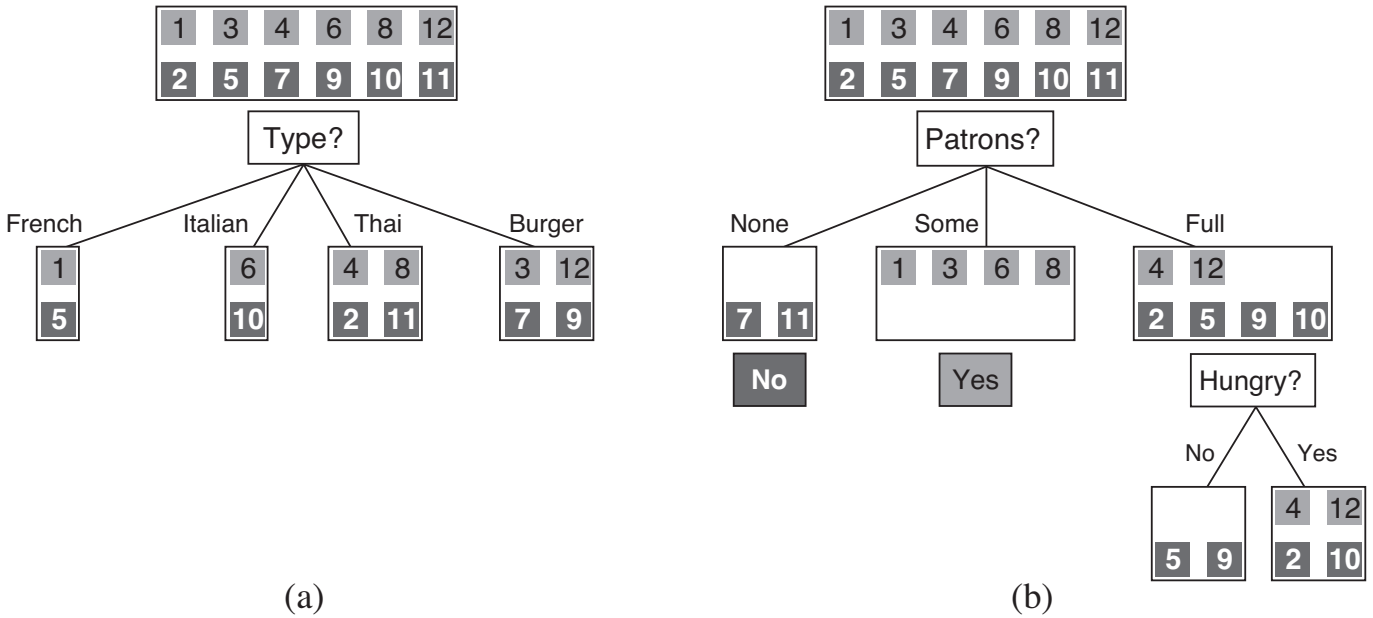
Ex	*Attributes*										*Target*
	Alt	Bar	Fri	Hun	Pat	\$	$Rain$	Res	$Type$	Est	Wait?
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0-10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30-60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0-10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10-30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	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

Inducing a decision tree

- An example is a specific set of attributes, along with a classification (Wait or not)
- Examples where Wait is true are called positive examples
- Examples where Wait is false are called negative examples
- The set of labeled examples is called the *training set*.
- We want to have a tree that:
  - Classifies the training set correctly
  - Accurately predicts unseen examples
  - is as small as possible (Occam's razor)
- What if we construct a tree with one leaf for each example?

Choosing useful attributes

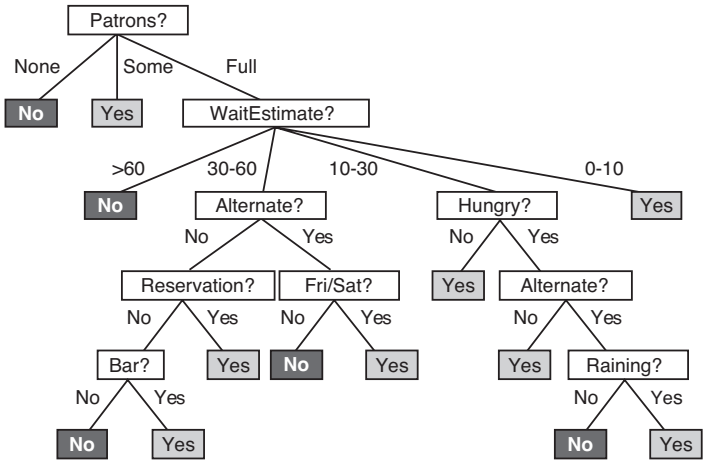
- Intuitively, we would like to test attributes that 'split' the training set into examples of all the same class.
  - Unbalanced number of 'yes' and 'no' answers
- Splitting on restaurant type is not useful - positive and negative examples are still clustered together.
  - Equal number of 'yes' and 'no' answers
- Splitting on crowdedness is more effective.



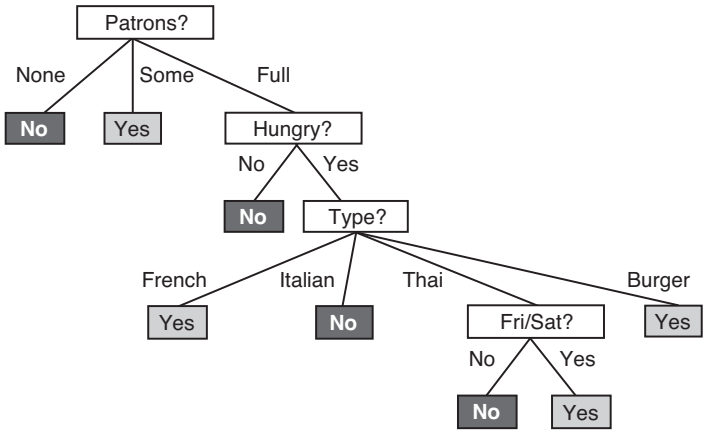
Constructing a decision tree

1. **Base case:** If all examples are positive or negative, we are done.
2. If there are no examples left, then we haven't seen an instance of this classification, so we use the majority classification of the parent.
3. If there are no attributes left to test, then we have instances with the same description, but different classifications.
   - Insufficient description
   - Noisy data, nondeterministic domain
   - Use majority vote
4. **Recursive step** Else, pick the best attribute to split on and recursively construct subtrees.

An example tree



- The hand-constructed tree.



- The induced tree. Notice that it's simpler and discovers a relationship between Thai food and waiting.

Using the tree to make decisions

- We can now use this tree to make decisions, or to *classify* new instances.
- Suppose we have a new instance:$Alt = F$, $Bar = F$,$Fri = T$,$Hungry = T$, $Patrons = Full$, $Rain = F$,$Reservations = F$,$Type = Burger$, $EstimatedTime = 10 - 30$.
- We traverse the tree from the root, following the appropriate 'Full', 'Hungry', and 'Type' branches.
- According to the tree, we should wait.

Choosing an Attribute

- The key to constructing a compact and efficient decision tree is to effectively choose attributes to test.
- Intuition: we want to choose tests that will separate our data set into positive and negative examples.
- We want to measure the amount of *information* provided by a test.
- This is a mathematical concept that characterizes the number of bits needed to answer a question or provide a fact.

Information

- Example:
  - In the vacuum world, rooms can be either clean or dirty. This requires one bit to represent.
  - What if a room could take on four states? Eight?
  - What if a room could only be in one state? How many bits would we need to represent this?

Information Theory

- More formally, let's say there are n possible answers $v_1, v_2, ..., v_n$ to a question, and each answer has probability $P(v_n)$ of occuring.
- The *information content* of the answer to the question is: $I = \sum_{i=1}^{n} -P(v_i) log_2 P(v_i)$
- For a coin, this is: $-\frac{1}{2} log_2 \frac{1}{2} + -\frac{1}{2} log_2 \frac{1}{2} = 1.$
- Questions with one highly likely anwer will have low information content. (if the coin comes up heads 99/100 of the time, I = 0.08)
- Information content is also sometimes called entropy.
  - This is often used in compression and data transfer algorithms

Using Information Theory

- For decision trees, we want to know how valuable each possible test is, or how much information it yields.
- We can estimate the probabilities of possible answers from the training set.
- Usually, a single test will not be enough to completely separate positive and negative examples.
- Instead, we need to think about how much better we'll be after asking a question.
- This is called the *information gain*.

Information Gain

- If a training set has p positive examples and n negative examples, the information in a correct answer is: $I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}$
- We want to find the attribute that will come the closest to separating the positive and negative examples.
- We begin by computing the remainder - this is the information still in the data after we test attribute A.
- Say attribute A can take on v possible values. This test will create v new subsets of data, labeled $E_1, E_2, ..., E_v$.
- The remainder is the sum of the information in each of these subsets.
- $Remainder(A) = \sum_{i=1}^{v} \frac{p_i+n_i}{p+n} * I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$

Information Gain

- Information gain can then be quantified as the difference between the original information (before the test) and the new information (after the test).
- $Gain(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - Remainder(A)$
- Heuristic: Always choose the attribute with the largest information gain.
  - Question: What kind of search is this?

Decision tree pseudocode

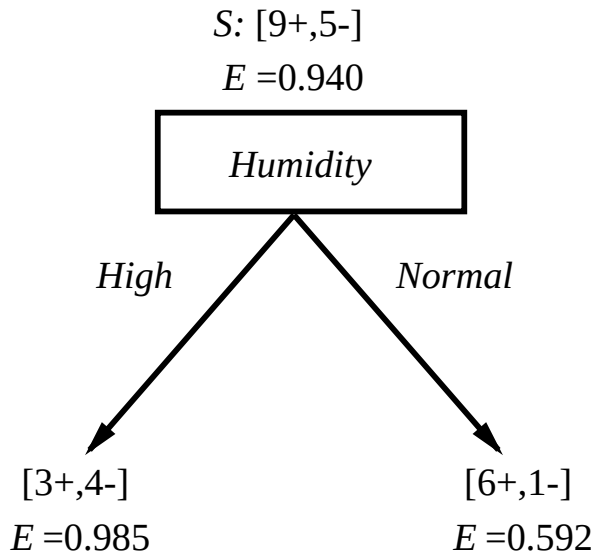
```
def makeTree(dataset) :
   if all data are in the same class :
       return a single Node with that classification
   if there are no attributes left to test:
       return a single Node with majority classification
   else :
        select the attribute that produces the largest information
        gain
        split the dataset according to the values of this attribute to
        create v smaller datasets.
        create a new Node - each child will be created by calling
        makeTree with one on the v subsets.
```

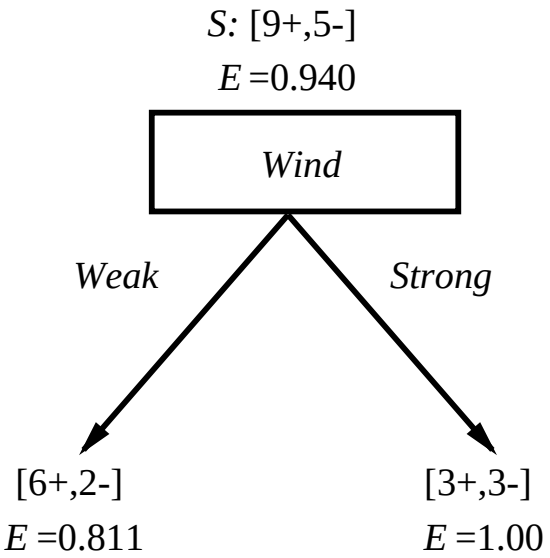
Example

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example

hich attribute is the best classifier





Gain (S, Humidity)

= .940 - (7/14).985 - (7/14).592
= .151

Gain (S, Wind)

= .940 - (8/14).811 - (6/14)1.0
= .048

Many classes

- The entropy calculation extends just fine to data with lots of classes.
- Suppose we have three classes: yes, no, maybe
- Data: yes, yes, maybe, no, yes, yes, no, maybe, yes, no.
- Entropy: $-\frac{5}{10}lg(\frac{5}{10}) * -\frac{3}{10}lg(\frac{3}{10}) * -\frac{2}{10}lg(\frac{2}{10})$

Issues: Noise

- If there are two examples that have the same attributes but different values, a decision tree will be unable to classify them separately.
- We say that this data is *noisy*.
- Solution: Use majority rule at the parent node.

Issues: Overfitting

- A common problem is learning algorithms occurs when there are random or anomalous patterns in the data.
  - For example, in the tennis problem, by chance it might turn out that we always play on Tuesdays.
- The phenomenon of learning quirks in the data is called *overfitting*.
- In decision trees, overfitting is dealt with through pruning.
- Once the tree is generated, we evaluate the significance of each node.

Simplifying the tree

Two approaches

- Stop growing the tree early, before perfect classification or pure nodes are reached
- Allow overfitting, then post-prune

First is much more difficult!

Pruning

Which (misclassified) examples should you use to decide what to prune?

- Better to not use the same set that you used to build the tree (why?)
- A *validation set* is a subset of the training examples that is held out for post-processing steps such as pruning

Cost-complexity pruning

- The cost complexity pruning algorithm used in CART is an example of the postpruning approach.
- Cost complexity of a tree is a function of the number of leaves in the tree and the error rate of the tree (where the error rate is the percentage of examples misclassified by the tree).
- It starts from the bottom of the tree. For each internal node, N, it computes the cost complexity of the subtree at N, and the cost complexity of the subtree at N if it were to be pruned.
- The two values are compared. If pruning the subtree at node N would result in a smaller cost complexity, then the subtree is pruned. Otherwise, it is kept.

Issues: Missing Data

- Decision trees assume that no data is missing.
  - Needed for entropy calculation
- Can use most common value
- Can probabilistically assign a value based on frequency.
- Can have a user-supplied default.

Issues: Continuous-valued inputs

- Decision trees can also be extended to work with integer and continuous-valued inputs.
- Discretize the range into, for example, < 70 and > 70.
- Challenge: What value yields the highest information gain?
- Can use a hill-climbing search to find this value.

Evaluation

- Typically, in evaluating the performance of a learning algorithm, we'll be interested in the following sorts of questions:
  - Does performance improve as the number of training examples increases?
  - How do precision and accuracy trade off as the number of training examples changes?
  - How does performance change as the problem gets easier/harder?
- So what does 'performance' mean?

Evaluation

- Recall that supervised algorithms start with a set of labeled data.
- Divide this data into two subsets:
  - Training set: used to train the classifier.
  - Test set: used to evaluate the classifier's performance.
  - These sets are disjoint.
- Procedure:
  - Train the algorithm with the classifier.
  - Run each element of the test set through the classifier. Count the number of incorrectly classified examples.
    - If the classification is binary, you can also measure precision and accuracy.

Evaluation

- How do we know we have a representative training and test set?
- Try it multiple times.
- N-fold cross-validation:
  - Do this N times:
    - Select 1/N documents at random as the test set.
    - Remainder is the training set.
    - Test as usual.
    - Average results.

Features of Decision Trees

- Work well with symbolic data
- Use information gain to determine what questions are most effective at classifying.
  - Greedy search
- Produce a human-understandable hypothesis
- Fixes needed to deal with noise or missing data
- Can represent all Boolean functions

Uses of Decision Trees

- Management
- Automated help systems
- Microsoft's online help
- Data mining