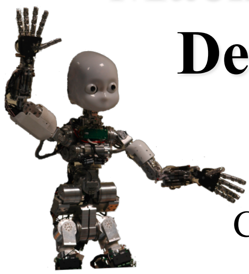
Machine Learning: Decision Trees



Chapter 18.1-18.3

Some material adopted from notes by Chuck Dyer

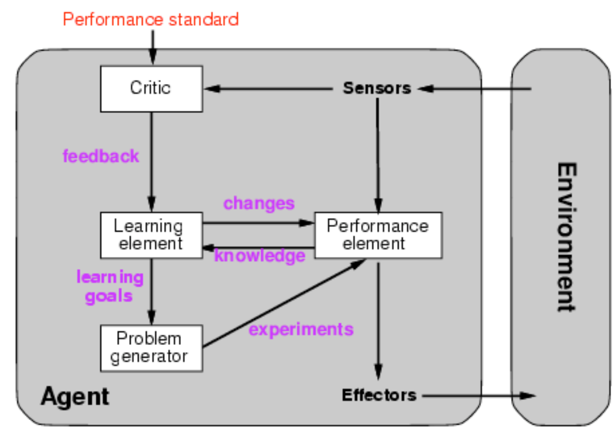
What is learning?

- “Learning denotes changes in a system that ... enable a system to do the same task more efficiently the next time.” –Herbert Simon
- “Learning is constructing or modifying representations of what is being experienced.” –Ryszard Michalski
- “Learning is making useful changes in our minds.” –Marvin Minsky

Why study learning?

- Understand and improve efficiency of human learning
  - Use to improve methods for teaching and tutoring people (e.g., better computer-aided instruction)
- Discover new things or structure previously unknown
  - Examples: data mining, scientific discovery
- Fill in skeletal or incomplete specifications about a domain
  - Large, complex AI systems can’t be completely built by hand and require dynamic updating to incorporate new information
  - Learning new characteristics expands the domain or expertise and lessens the “brittleness” of the system
- Build agents that can adapt to users, other agents, and their environment

A general model of learning agents



Major paradigms of machine learning

- **Rote learning** – One-to-one mapping from inputs to stored representation. "Learning by memorization." Association-based storage and retrieval.
- **Induction** – Use specific examples to reach general conclusions
- **Clustering** – Unsupervised identification of natural groups in data
- **Analogy** – Determine correspondence between two different representations
- **Discovery** – Unsupervised, specific goal not given
- **Genetic algorithms** – "Evolutionary" search techniques, based on an analogy to "survival of the fittest"
- **Reinforcement** – Feedback (positive or negative reward) given at the end of a sequence of steps

The inductive learning problem

- Extrapolate from a given set of examples to make accurate predictions about future examples
- **Supervised versus unsupervised learning**
  - Learn an unknown function f(X) = Y, where X is an input example and Y is the desired output.
  - **Supervised learning** implies we are given a **training set** of (X, Y) pairs by a "teacher"
  - **Unsupervised learning** means we are only given the Xs and some (ultimate) feedback function on our performance.
- **Concept learning or classification**
  - Given a set of examples of some concept/class/category, determine if a new one is an instance of the concept or not
  - Instances are *positive examples,* non-instances *negative examples*
  - Or we can make a probabilistic prediction (e.g., using a Bayes net)

Supervised concept learning

- Given a training set of positive and negative examples of a concept
- Construct a description that will accurately classify whether future examples are positive or negative
- That is, learn some good estimate of function f given a training set $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ where each y_i is either + (positive) or - (negative), or a probability distribution over +/-

Inductive learning framework

- Raw input data from sensors are typically preprocessed to obtain a **feature vector**, X, that adequately describes all of the relevant features for classifying examples
- Each x is a list of (attribute, value) pairs. For example,
  X = [Person:Sue, EyeColor:Brown, Age:Young, Sex:Female]
- The number of attributes (a.k.a. features) is fixed (positive, finite)
- Each attribute has a fixed, finite number of possible values (or could be continuous)
- Each example is interpreted as a point in an n-dimensional **feature space**, where n is the number of attributes

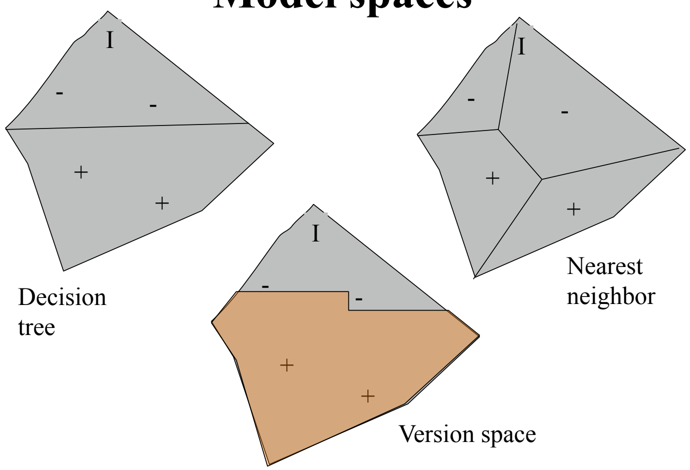
Inductive learning as search

- Instance space I defines the language for the training and test instances
  - Typically, but not always, each instance i∈I is a feature vector
  - Features are sometimes called attributes or variables
  - I: $V_1 \times V_2 \times \ldots \times V_k$, $i = (v_1, v_2, \ldots, v_k)$
- Class variable C gives an instance's class (to be predicted)
- Model space M defines the possible classifiers
  - M: I → C, M = {m1, … mn} (possibly infinite)
  - Model space is sometimes, but not always, defined in terms of the same features as the instance space
- Training data can be used to direct the search for a good (consistent, complete, simple) hypothesis in the model space

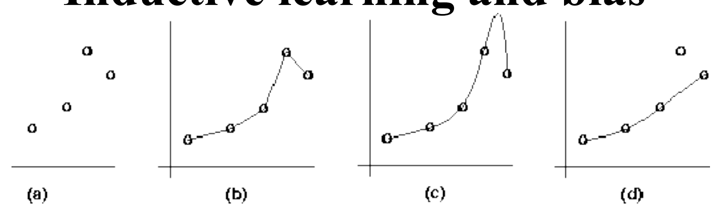
Model spaces

- **Decision trees**
  - Partition the instance space into axis-parallel regions, labeled with class value
- **Version spaces**
  - Search for necessary (lower-bound) and sufficient (upper-bound) partial instance descriptions for an instance to be in the class
- Nearest-neighbor classifiers
  - Partition the instance space into regions defined by the centroid instances (or cluster of k instances)
- Associative rules (feature values → class)
- First-order logical rules
- Bayesian networks (probabilistic dependencies of class on attributes)
- Neural networks

Model spaces



Inductive learning and bias



- Suppose that we want to learn a function f(x) = y and we are given some sample (x,y) pairs, as in figure (a)
- There are several hypotheses we could make about this function, e.g.: (b), (c) and (d)
- A preference for one over the others reveals the **bias** of our learning technique, e.g.:
  - prefer piece-wise functions
  - prefer a smooth function
  - prefer a simple function and treat outliers as noise

Preference bias: Ockham's Razor

- A.k.a. Occam's Razor, Law of Economy, or Law of Parsimony
- Principle stated by William of Ockham (1285-1347/49), a scholastic, that
  - "*non sunt multiplicanda entia praeter necessitatem*"
  - or, entities are not to be multiplied beyond necessity
- The simplest consistent explanation is the best
- Therefore, the smallest decision tree that correctly classifies all of the training examples is best.
- Finding the provably smallest decision tree is NP-hard, so instead of constructing the absolute smallest tree consistent with the training examples, construct one that is pretty small

Learning decision trees

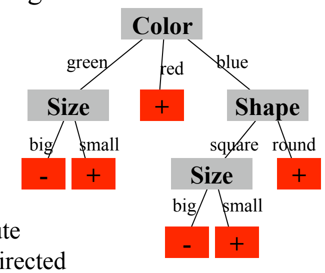
•Goal: Build a **decision tree** to classify examples as positive or negative instances of a concept using supervised learning from a training set

•A **decision tree** is a tree where

– each non-leaf node has associated with it an attribute (feature)

–each leaf node has associated with it a classification (+ or -)

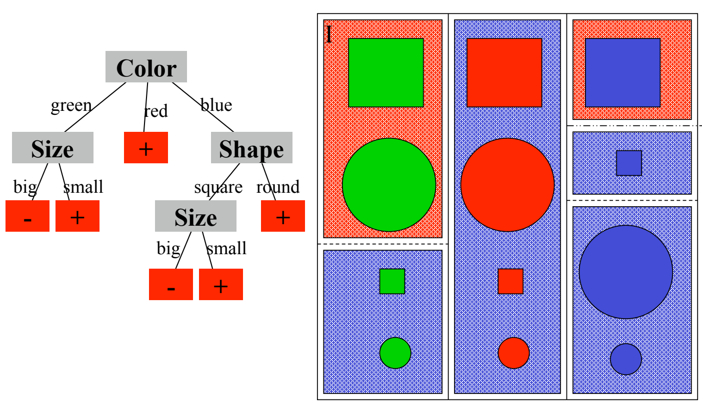
–each arc has associated with it one of the possible values of the attribute at the node from which the arc is directed



•Generalization: allow for >2 classes

–e.g., for stocks, classify into {sell, hold, buy}

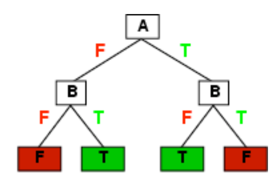
Decision tree-induced partition – example



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 with one path to leaf for each example (unless *f* nondeterministic in *x*) but it probably won't generalize to new examples
- We prefer to find more compact decision trees

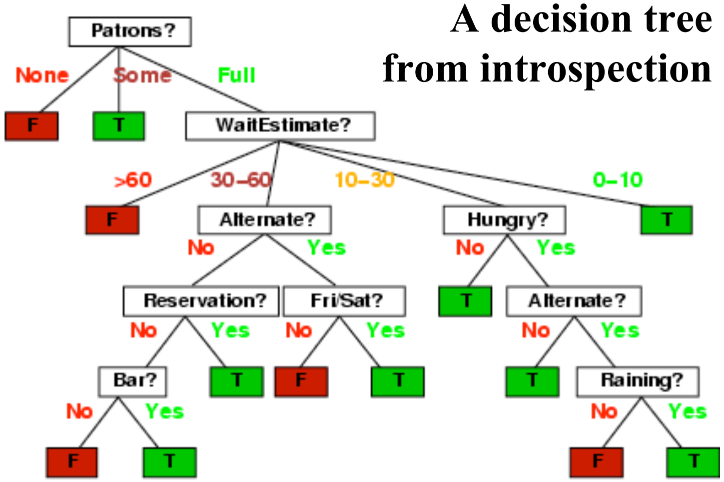
Hypothesis spaces

- **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, 18,446,744,073,709,551,616 trees
- **How many conjunctive hypotheses (e.g., *Hungry* ∧ ¬*Rain*)?**
  - Each attribute can be in (positive), in (negative), or out
    ⇒ 3^n distinct conjunctive hypotheses
  - e.g., with 6 Boolean attributes, 729 trees
- **A more expressive hypothesis space**
  - increases chance that target function can be expressed
  - increases number of hypotheses consistent with training set
    ⇒ may get worse predictions in practice

R&N's restaurant domain

- Develop a decision tree to model decision a patron makes when deciding whether or not to wait for a table at a restaurant
- Two classes: wait, leave
- Ten attributes: Alternative available? Bar in restaurant? Is it Friday? Are we hungry? How full is the restaurant? How expensive? Is it raining? Do we have a reservation? What type of restaurant is it? What's the purported waiting time?
- Training set of 12 examples
- ~ 7000 possible cases

A decision tree from introspection



Attribute-based representations

Example	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

- Examples described by attribute values (Boolean, discrete, continuous)
  - E.g., situations where I will/won't wait for a table
- Classification of examples is positive (T) or negative (F)
- Serves as a training set

ID3 Algorithm

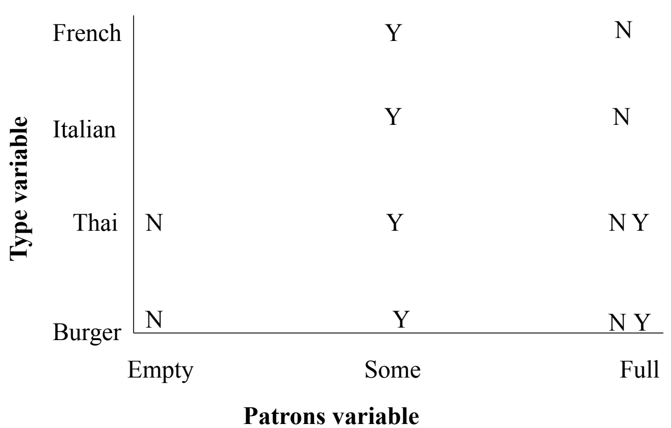
- A greedy algorithm for decision tree construction developed by Ross Quinlan circa 1987
- Top-down construction of decision tree by recursively selecting "best attribute" to use at the current node in tree
  - Once attribute is selected for current node, generate child nodes, one for each possible value of selected attribute
  - Partition examples using the possible values of this attribute, and assign these subsets of the examples to the appropriate child node
  - Repeat for each child node until all examples associated with a node are either all positive or all negative

Choosing the best attribute

- Key problem: choosing which attribute to split a given set of examples
- Some possibilities are:
  - **Random:** Select any attribute at random
  - **Least-Values:** Choose the attribute with the smallest number of possible values
  - **Most-Values:** Choose the attribute with the largest number of possible values
  - **Max-Gain:** Choose the attribute that has the largest expected *information gain*–i.e., attribute that results in smallest expected size of subtrees rooted at its children
- The ID3 algorithm uses the Max-Gain method of selecting the best attribute

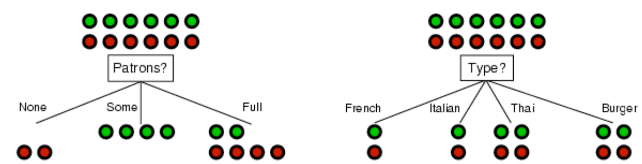
Restaurant example

Random: Patrons or Wait-time; **Least-values**: Patrons; **Most-values**: Type; **Max-gain**: ???



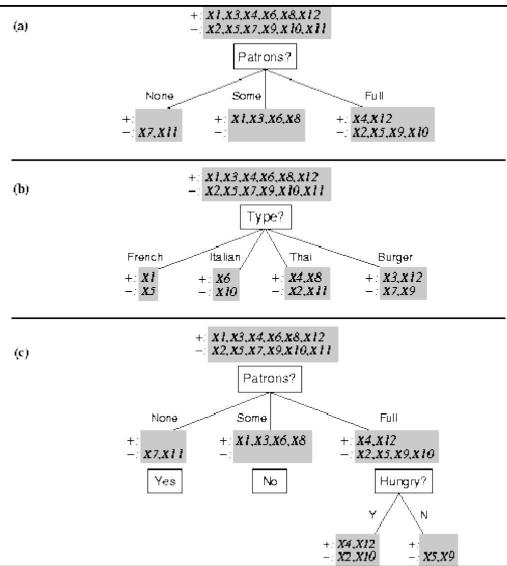
Choosing an attribute

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

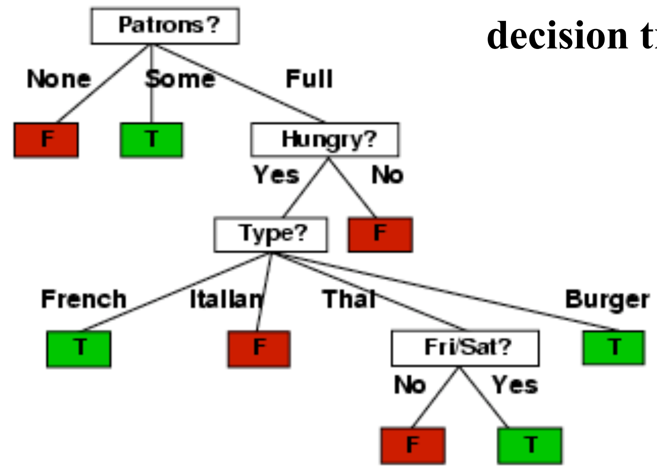


Which is better: *Patrons?* or *Type?*

Splitting examples by testing attributes



ID3-induced decision tree



Information theory 101

- Information theory sprang almost fully formed from the seminal work of Claude E. Shannon at Bell Labs
  - A Mathematical Theory of Communication, *Bell System Technical Journal*, 1948.
- Intuitions
  - Common words (a, the, dog) are shorter than less common ones (parlimentarian, foreshadowing)
  - In Morse code, common (probable) letters have shorter encodings
- Information is measured in minimum number of bits needed to store or send some information
- Wikipedia: The measure of data, known as information entropy, is usually expressed by the average number of bits needed for storage or communication.

Information theory 101

- Information is measured in bits
- Information conveyed by message depends on its probability
- With n equally probable possible *messages*, the probability p of each is *1/n*
- Information conveyed by message is -log(p) = log(n)
  - e.g., with 16 messages, then log(16) = 4 and we need 4 bits to identify/send each message
- Given probability distribution for n messages $P = (p_1, p_2 \ldots p_n)$, the information conveyed by distribution (aka *entropy* of P) is:

$$I(P) = -(p_1 * \log(p_1) + p_2 * \log(p_2) + .. + p_n * \log(p_n))$$

probability of msg 2

info in msg 2

Information theory II

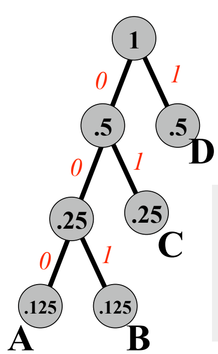
- Information conveyed by distribution (a.k.a. *entropy* of P):
  $I(P) = -(p_1*\log(p_1) + p_2*\log(p_2) + .. + p_n*\log(p_n))$
- Examples:
  - If P is (0.5, 0.5) then I(P) = .5*1 + 0.5*1 = 1
  - If P is (0.67, 0.33) then I(P) = -(2/3*log(2/3) + 1/3*log(1/3)) = 0.92
  - If P is (1, 0) then I(P) = 1*1 + 0*log(0) = 0
- The more uniform the probability distribution, the greater its information: More information is conveyed by a message telling you which event actually occurred
- Entropy is the average number of bits/message needed to represent a stream of messages

Huffman code

- In 1952 MIT student David Huffman devised, in the course of doing a homework assignment, an elegant coding scheme which is optimal in the case where all symbols' probabilities are integral powers of 1/2.
- A Huffman code can be built in the following manner:
  - Rank all symbols in order of probability of occurrence
  - Successively combine the two symbols of the lowest probability to form a new composite symbol; eventually we will build a binary tree where each node is the probability of all nodes beneath it
  - Trace a path to each leaf, noticing the direction at each node

Huffman code example

M	P
A	.125
B	.125
C	.25
D	.5



M	code	length	prob	
A	000	3	0.125	0.375
B	001	3	0.125	0.375
C	01	2	0.250	0.500
D	1	1	0.500	0.500
average message length				1.750

If we use this code to many messages (A,B,C or D) with this probability distribution, then, over time, the average bits/message should approach 1.75

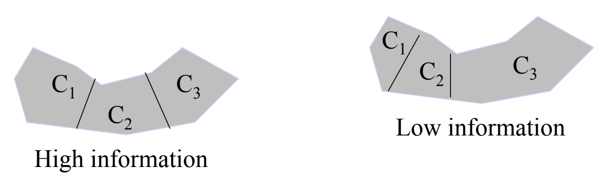
Information for classification

If a set T of records is partitioned into disjoint exhaustive classes $(C_1,C_2,..,C_k)$ on the basis of the value of the class attribute, then information needed to identify class of an element of T is:

Info(T) = I(P)

where P is the probability distribution of partition $(C_1,C_2,..,C_k)$:

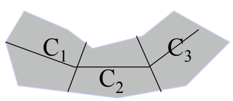
$P = (|C_1|/|T|, |C_2|/|T|, ..., |C_k|/|T|)$



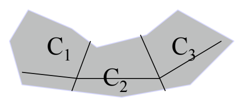
Information for classification II

If we partition T w.r.t attribute X into sets $\{T_1, T_2, .., T_n\}$ then the information needed to identify the class of an element of T becomes the weighted average of the information needed to identify the class of an element of T_i, i.e. the weighted average of $Info(T_i)$:

$$Info(X,T) = \sum |T_i|/|T| * Info(T_i)$$



High information



Low information

Information gain

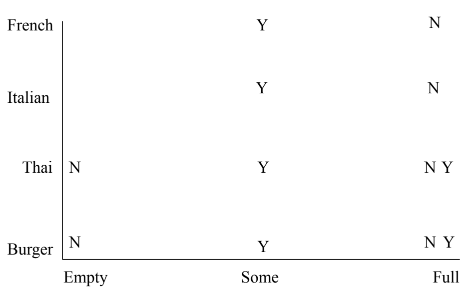
- Consider the quantity Gain(X,T) defined as
  Gain(X,T) = Info(T) - Info(X,T)
- This represents the difference between
  - info needed to identify element of T and
  - info needed to identify element of T after value of attribute X known
- This is the **gain in information due to attribute X**
- Use to rank attributes and build DT where each node uses attribute with greatest gain of those not yet considered (in path from root)
- The intent of this ordering is to:
  - Create small DTs so records can be identified with few questions
  - Match a hoped-for minimality of the process represented by the records being considered (Occam's Razor)

Computing information gain

I(T) =
- (.5 log .5 + .5 log .5)
= .5 + .5 = 1

I (Pat, T) =
2/12 (0) + 4/12 (0) +
6/12 (- (4/6 log 4/6 +
2/6 log 2/6))
= 1/2 (2/3*.6 +
1/3*1.6)
= .47

I (Type, T) =
2/12 (1) + 2/12 (1) +
4/12 (1) + 4/12 (1) = 1



Gain (Pat, T) = 1 - .47 = .53

Gain (Type, T) = 1 – 1 = 0

The ID3 algorithm builds a decision tree, given a set of non-categorical attributes C1, C2, .., Cn, the class attribute C, and a training set T of records

```
function ID3(R:input attributes, C:class attribute,
  S:training set) returns decision tree;
    If S is empty, return single node with value Failure;
    If every example in S has same value for C, return
      single node with that value;
    If R is empty, then return a single node with most
      frequent of the values of C found in examples S;
      # causes errors -- improperly classified record
    Let D be attribute with largest Gain(D,S) among R;
    Let {dj| j=1,2, .., m} be values of attribute D;
    Let {Sj| j=1,2, .., m} be subsets of S consisting of
              records with value dj for attribute D;
    Return tree with root labeled D and arcs labeled
       d1..dm going to the trees ID3(R-{D},C,S1). . .
       ID3(R-{D},C,Sm);
```

How well does it work?

Many case studies have shown that decision trees are at least as accurate as human experts.

- A study for diagnosing breast cancer had humans correctly classifying the examples 65% of the time; the decision tree classified 72% correct
- British Petroleum designed a decision tree for gas-oil separation for offshore oil platforms that replaced an earlier rule-based expert system
- Cessna designed an airplane flight controller using 90,000 examples and 20 attributes per example

Extensions of ID3

- Using gain ratios
- Real-valued data
- Noisy data and overfitting
- Generation of rules
- Setting parameters
- Cross-validation for experimental validation of performance
- C4.5 is an extension of ID3 that accounts for unavailable values, continuous attribute value ranges, pruning of decision trees, rule derivation, and so on

Using gain ratios

- The information gain criterion favors attributes that have a large number of values
  - If we have an attribute D that has a distinct value for each record, then Info(D,T) is 0, thus Gain(D,T) is maximal
- To compensate for this Quinlan suggests using the following ratio instead of Gain:

  GainRatio(D,T) = Gain(D,T) / SplitInfo(D,T)
- SplitInfo(D,T) is the information due to the split of T on the basis of value of categorical attribute D

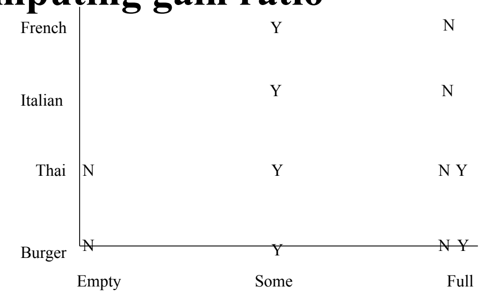
  SplitInfo(D,T) = I(|T1|/|T|, |T2|/|T|, .., |Tm|/|T|)

where {T1, T2, .. Tm} is the partition of T induced by value of D

Computing gain ratio

- I(T) = 1
- I (Pat, T) = .47
- I (Type, T) = 1

Gain (Pat, T) =.53
Gain (Type, T) = 0



SplitInfo (Pat, T) = - (1/6 log 1/6 + 1/3 log 1/3 + 1/2 log 1/2) = 1/6*2.6 + 1/3*1.6 + 1/2*1 = 1.47

SplitInfo (Type, T) = 1/6 log 1/6 + 1/6 log 1/6 + 1/3 log 1/3 + 1/3 log 1/3 = 1/6*2.6 + 1/6*2.6 + 1/3*1.6 + 1/3*1.6 = 1.93

GainRatio (Pat, T) = Gain (Pat, T) / SplitInfo(Pat, T) = .53 / 1.47 = .36

GainRatio (Type, T) = Gain (Type, T) / SplitInfo (Type, T) = 0 / 1.93 = 0

Real-valued data

- Select a set of thresholds defining intervals
- Each interval becomes a discrete value of the attribute
- Use some simple heuristics…
  - always divide into quartiles
- Use domain knowledge…
  - divide age into infant (0-2), toddler (3 - 5), school-aged (5-8)
- Or treat this as another learning problem
  - Try a range of ways to discretize the continuous variable and see which yield "better results" w.r.t. some metric
  - E.g., try midpoint between every pair of values

Noisy data

- Many kinds of "noise" can occur in the examples:
- Two examples have same attribute/value pairs, but different classifications
- Some values of attributes are incorrect because of errors in the data acquisition process or the preprocessing phase
- The classification is wrong (e.g., + instead of -) because of some error
- Some attributes are irrelevant to the decision-making process, e.g., color of a die is irrelevant to its outcome

Overfitting

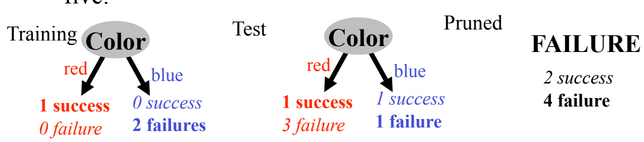
- Irrelevant attributes, can result in *overfitting* the training example data
- If hypothesis space has many dimensions (large number of attributes), we may find **meaningless regularity** in the data that is irrelevant to the true, important, distinguishing features
- If we have too little training data, even a reasonable hypothesis space will 'overfit'

Overfitting

- Fix by by removing irrelevant features
  - E.g., remove 'year observed', 'month observed', 'day observed', 'observer name' from feature vector
- Fix by getting more training data
- Fix by pruning lower nodes in the decision tree
  - E.g., if gain of the best attribute at a node is below a threshold, stop and make this node a leaf rather than generating children nodes

Pruning decision trees

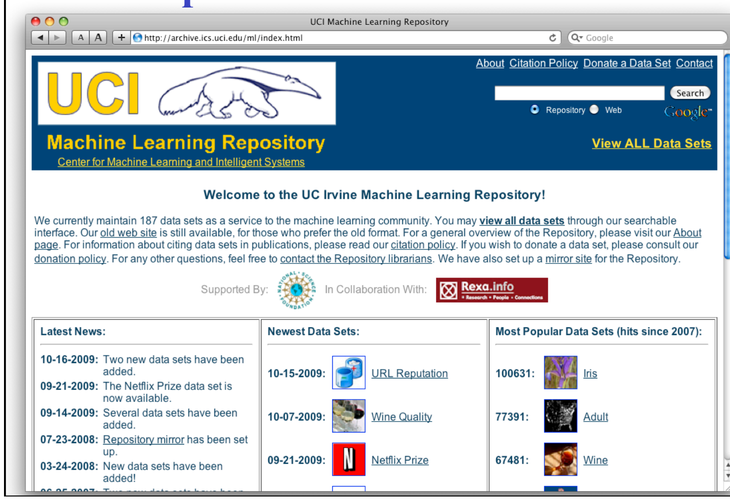
- Pruning of the decision tree is done by replacing a whole subtree by a leaf node
- The replacement takes place if a decision rule establishes that the expected error rate in the subtree is greater than in the single leaf. E.g.,
  - Training: one training red success and two training blue failures
  - Test: three red failures and one blue success
  - Consider replacing this subtree by a single Failure node.
- After replacement we will have only two errors instead of five:



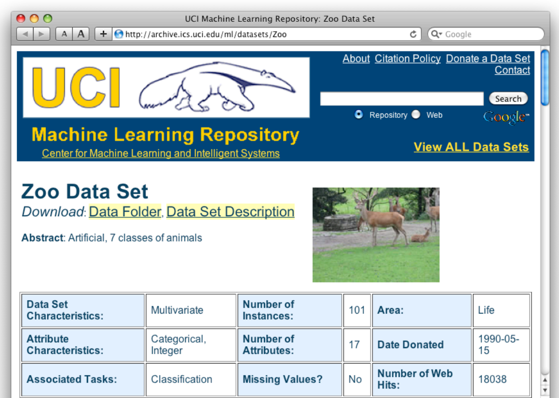
Converting decision trees to rules

- It is easy to derive rules from a decision tree: write a rule for each path from the root to a leaf
- In that rule the left-hand side is built from the label of the nodes and the labels of the arcs
- The resulting rules set can be simplified:
  - Let LHS be the left hand side of a rule
  - LHS' obtained from LHS by eliminating some conditions
  - Replace LHS by LHS' in this rule if the subsets of the training set satisfying LHS and LHS' are equal
  - A rule may be eliminated by using meta-conditions such as "if no other rule applies"

http://archive.ics.uci.edu/ml



Example: zoo data



http://archive.ics.uci.edu/ml/datasets/Zoo

Zoo data

animal name: string
hair: Boolean
feathers: Boolean
eggs: Boolean
milk: Boolean
airborne: Boolean
aquatic: Boolean
predator: Boolean
toothed: Boolean
backbone: Boolean
breathes: Boolean
venomous: Boolean
fins: Boolean
legs: {0,2,4,5,6,8}
tail: Boolean
domestic: Boolean
catsize: Boolean
type: {mammal, fish, bird, shellfish, insect, reptile, amphibian}

101 examples

```
aardvark,1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,mammal
antelope,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,mammal
bass,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,fish
bear,1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,mammal
boar,1,0,0,1,0,0,1,1,1,1,0,0,4,1,0,1,mammal
buffalo,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,mammal
calf,1,0,0,1,0,0,0,1,1,1,0,0,4,1,1,1,mammal
carp,0,0,1,0,0,1,0,1,1,0,0,1,0,1,1,0,fish
catfish,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,fish
cavy,1,0,0,1,0,0,0,1,1,1,0,0,4,0,1,0,mammal
cheetah,1,0,0,1,0,0,1,1,1,1,0,0,4,1,0,1,mammal
chicken,0,1,1,0,1,0,0,0,1,1,0,0,2,1,1,0,bird
chub,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,fish
clam,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,shellfish
crab,0,0,1,0,0,1,1,0,0,0,0,0,4,0,0,0,shellfish
…
```

Zoo example

```
aima-python> python
>>> from learning import *
>>> zoo
<DataSet(zoo): 101 examples, 18 attributes>
>>> dt = DecisionTreeLearner()
>>> dt.train(zoo)
>>> dt.predict(['shark',0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0])
'fish'
>>> dt.predict(['shark',0,0,0,0,0,1,1,1,1,0,0,1,0,1,0,0])
'mammal’
```

Zoo example

```
>> dt.dt
DecisionTree(13, 'legs', {0: DecisionTree(12, 'fins', {0: DecisionTree(8, 'toothed', {0: 'shellfish', 1: 'reptile'}), 1: DecisionTree(3, 'eggs', {0: 'mammal', 1: 'fish'})}), 2: DecisionTree(1, 'hair', {0: 'bird', 1: 'mammal'}), 4: DecisionTree(1, 'hair', {0: DecisionTree(6, 'aquatic', {0: 'reptile', 1: DecisionTree(8, 'toothed', {0: 'shellfish', 1: 'amphibian'})}), 1: 'mammal'}), 5: 'shellfish', 6: DecisionTree(6, 'aquatic', {0: 'insect', 1: 'shellfish'}), 8: 'shellfish'})
```

Zoo example

```
>>> dt.dt.display()
Test legs
 legs = 0 ==> Test fins
    fins = 0 ==> Test toothed
       toothed = 0 ==> RESULT =  shellfish
       toothed = 1 ==> RESULT =  reptile
    fins = 1 ==> Test eggs
       eggs = 0 ==> RESULT =  mammal
       eggs = 1 ==> RESULT =  fish
 legs = 2 ==> Test hair
    hair = 0 ==> RESULT =  bird
    hair = 1 ==> RESULT =  mammal
 legs = 4 ==> Test hair
    hair = 0 ==> Test aquatic
       aquatic = 0 ==> RESULT =  reptile
       aquatic = 1 ==> Test toothed
          toothed = 0 ==> RESULT =  shellfish
          toothed = 1 ==> RESULT =  amphibian
    hair = 1 ==> RESULT =  mammal
 legs = 5 ==> RESULT =  shellfish
 legs = 6 ==> Test aquatic
    aquatic = 0 ==> RESULT =  insect
    aquatic = 1 ==> RESULT =  shellfish
 legs = 8 ==> RESULT =  shellfish
```

Zoo example

```
>>> dt.dt.display()
Test legs
 legs = 0 ==> Test fins
    fins = 0 ==> Test toothed
       toothed = 0 ==> RESULT = shellfish
       toothed = 1 ==> RESULT = reptile
    fins = 1 ==> Test milk
       milk = 0 ==> RESULT = fish
       milk = 1 ==> RESULT = mammal
 legs = 2 ==> Test hair
    hair = 0 ==> RESULT = bird
    hair = 1 ==> RESULT = mammal
 legs = 4 ==> Test hair
    hair = 0 ==> Test aquatic
       aquatic = 0 ==> RESULT = reptile
       aquatic = 1 ==> Test toothed
          toothed = 0 ==> RESULT = shellfish
          toothed = 1 ==> RESULT = amphibian
    hair = 1 ==> RESULT = mammal
 legs = 5 ==> RESULT = shellfish
 legs = 6 ==> Test aquatic
    aquatic = 0 ==> RESULT = insect
    aquatic = 1 ==> RESULT = shellfish
 legs = 8 ==> RESULT = shellfish
```

Add the shark example to the training set and retrain

Evaluation methodology

- Standard methodology:
  1. Collect large set of examples with correct classifications
  2. Randomly divide collection into two disjoint sets: *training* and *test*
  3. Apply learning algorithm to training set giving hypothesis H
  4. Measure performance of H w.r.t. test set
- Important: keep the training and test sets disjoint!
- Study efficiency and robustness of algorithm: repeat steps 2-4 for different training sets and sizes of training sets
- On modifying algorithm, restart with step 1 to avoid evolving algorithm to work well on just this collection

K-fold Cross Validation

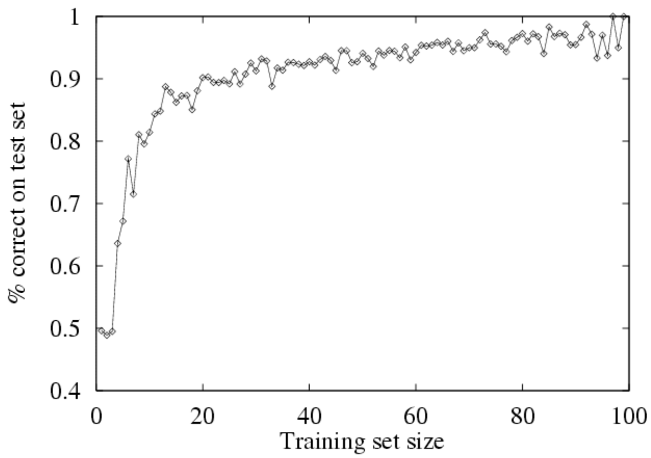
- Problem: getting “ground truth” data can be expensive
- Problem: ideally need different test data each time we test
- Problem: experimentation is needed to find right “feature space” and parameters for ML algorithm
- Goal: minimize amount of training+test data needed
- Idea: split training data into K subsets, use K-1 for *training*, and one for *development testing*
- Common K values are 5 and 10

Zoo evaluation

```
>>> train_and_test(DecisionTreeLearner(), zoo, 0, 10)
1.0
>>> train_and_test(DecisionTreeLearner(), zoo, 90, 100)
0.80000000000000004
>>> train_and_test(DecisionTreeLearner(), zoo, 90, 101)
0.81818181818181823
>>> train_and_test(DecisionTreeLearner(), zoo, 80, 90)
0.90000000000000002
>>> cross_validation(DecisionTreeLearner(), zoo, 10, 20)
0.95500000000000007
>>> leave1out(DecisionTreeLearner(), zoo)
0.97029702970297027
```

Learning curve

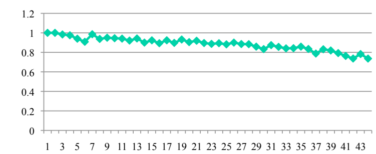
Learning curve = % correct on test set as a function of training set size



Zoo

>>> learningcurve(DecisionTreeLearner(), zoo)

[(2, 1.0), (4, 1.0), (6, 0.98333333333333339), (8, 0.97499999999999998), (10, 0.94000000000000006), (12, 0.90833333333333321), (14, 0.98571428571428577), (16, 0.9375), (18, 0.94999999999999996), (20, 0.94499999999999995), … (86, 0.78255813953488373), (88, 0.73636363636363644), (90, 0.70777777777777795)]



Summary: Decision tree learning

- Inducing decision trees is one of the most widely used learning methods in practice
- Can out-perform human experts in many problems
- Strengths include
  - Fast
  - Simple to implement
  - Can convert result to a set of easily interpretable rules
  - Empirically valid in many commercial products
  - Handles noisy data
- Weaknesses include:
  - Univariate splits/partitioning using only one attribute at a time so limits types of possible trees
  - Large decision trees may be hard to understand
  - Requires fixed-length feature vectors
  - Non-incremental (i.e., batch method)