Machine Learning: Decision Trees

Chapter 18.1-18.3

Some material adopted from notes
by Chuck Dyer

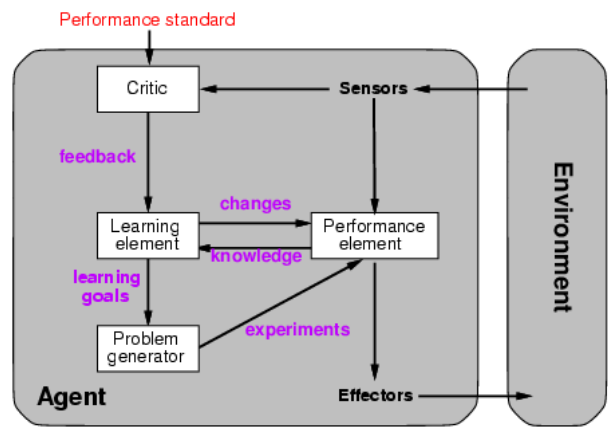
What is learning?

- "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 derived 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

5

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 given example is an instance of the concept or not
  - If it is an instance, we call it a positive example
  - If it is not, it is called a negative example
  - Or we can make a probabilistic prediction (e.g., using a Bayes net)

6

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 +/-

7

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 can be interpreted as a point in an n-dimensional **feature space**, where n is the number of attributes

8

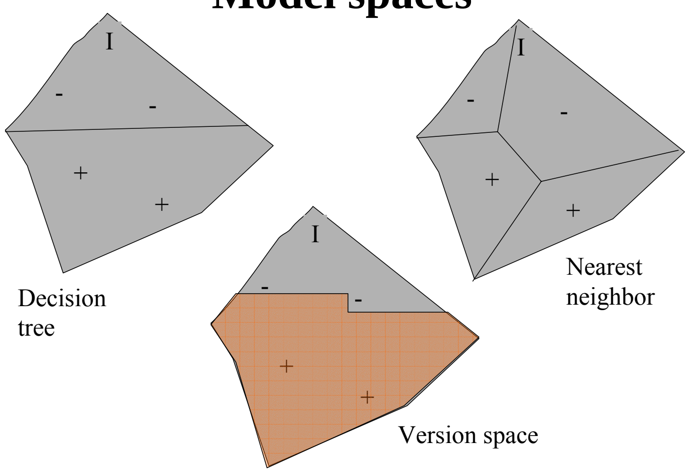
Inductive learning as search

- Instance space I defines the language for the training and test instances
  - Typically, but not always, each instance $i \in I$ 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
  - $M: I \rightarrow C$, $M = \{m1, \ldots mn\}$ (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

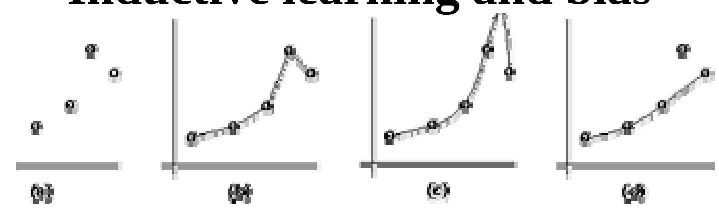
Model spaces

- **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 a member of 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



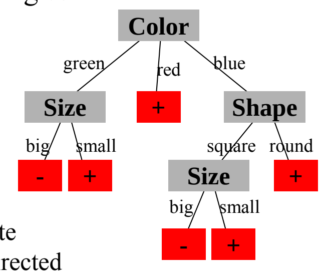
- Suppose that we want to learn a function f(x) = y and we are given some sample (x,y) pairs, as in figure (a)
- There are several hypotheses we could make about this function, e.g.: (b), (c) and (d)
- A preference for one over the others reveals the **bias** of our learning technique, e.g.:
  - prefer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

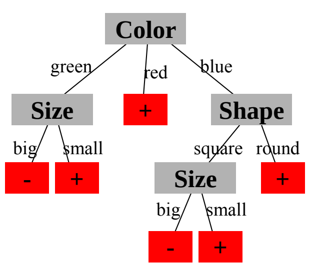
Learning decision trees

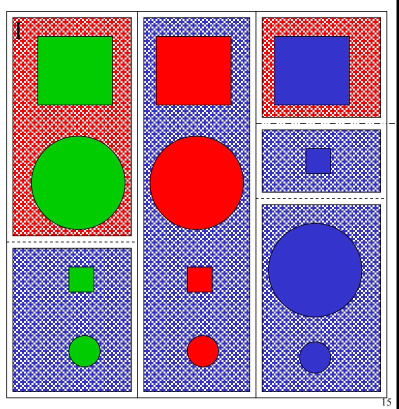
- Goal: Build a **decision tree** to classify examples as positive or negative instances of a concept using supervised learning from a training set
- A **decision tree** is a tree where
  - each non-leaf node has associated with it an attribute (feature)
  - each leaf node has associated with it a classification (+ 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

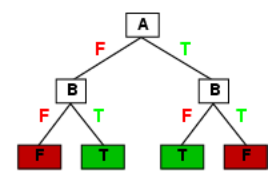




Expressiveness

- 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

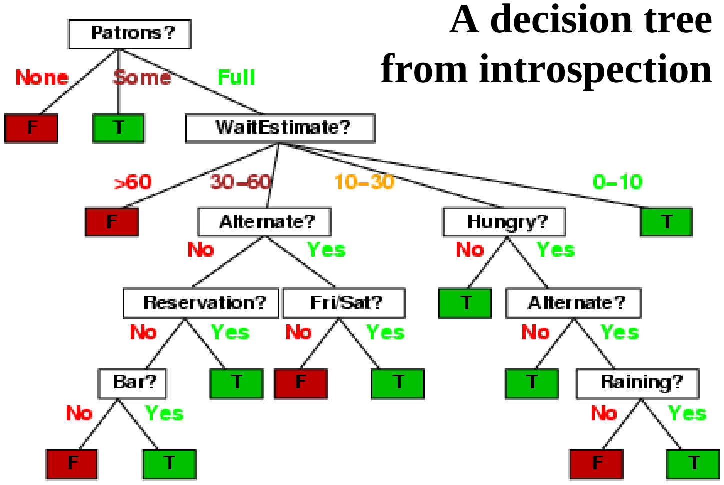
Hypothesis spaces

- **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* $\wedge \neg$*Rain*)?**
  - Each attribute can be in (positive), in (negative), or out
    $\Rightarrow 3^n$ 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
    $\Rightarrow$ may get worse predictions in practice

R&N's restaurant domain

- Develop a decision tree to model the 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

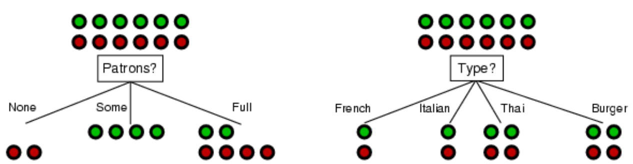
- A greedy algorithm for decision tree construction developed by Ross Quinlan circa 1987
- Top-down construction of decision tree by recursively selecting “best attribute” to use at the current node in tree
  – Once attribute is selected for current node, generate child nodes, one for each possible value 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

- The key problem is 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., the attribute that will result in the smallest expected size of the subtrees rooted at its children
- The ID3 algorithm uses the Max-Gain method of selecting the best attribute

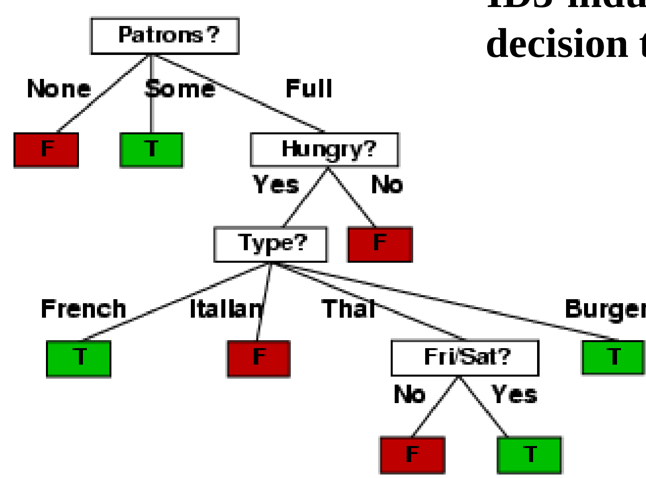
Choosing an attribute

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



Patrons? is a better choice than *type?*

ID3-induced decision tree



Information theory 101

- Information is measured in bits
- If there are n equally probable possible messages, then the probability p of each is 1/n
- Information conveyed by a 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
- In general, given a probability distribution for the n messages $P = (p_1, p_2, .., p_n)$
- Then the information conveyed by the distribution (aka *entropy* of P) is:

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

25

Information theory 101

- Information theory sprang almost fully formed from the seminal work of Claude E. Shannon at Bell Labs
  - classic paper "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: he measure of data, known as information entropy, is usually expressed by the average number of bits needed for storage or communication.

26

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) is 1
  - If P is (0.67, 0.33) then I(P) is 0.92
  - If P is (1, 0) then I(P) is 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

27

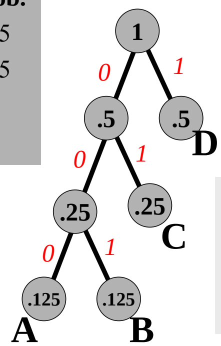
Huffman code

- 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

28

Huffman code example

Msg.	Prob.
A	.125
B	.125
C	.25
D	.5



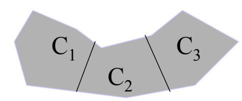
M	code	length	prob	
A	000	3	0.125	0.375
B	001	3	0.125	0.375
C	01	2	0.250	0.500
D	1	1	0.500	0.500
average message length				1.750

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

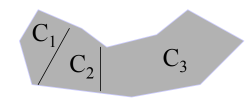
Information for classification

- 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 the information needed to identify the class of an element of T is
  Info(T) = I(P)

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



High information

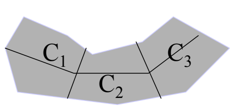


Low information

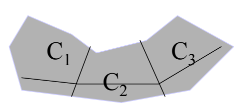
Information for classification II

- 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
  - information needed to identify an element of T and
  - information needed to identify an element of T after the value of attribute X has been obtained

That is, this is the **gain in information due to attribute X**

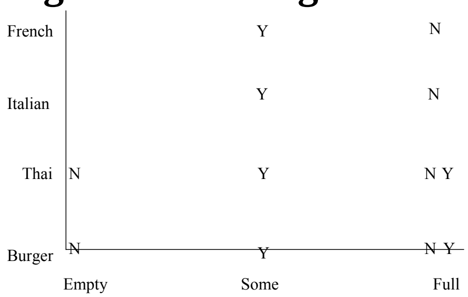
- We can use this to rank attributes and to build decision trees where at each node is located the attribute with greatest gain among the attributes not yet considered in the path from the root
- The intent of this ordering is:
  - To create small decision trees so that records can be identified after only a few questions
  - To 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) =
1/6 (0) + 1/3 (0) +
1/2 (- (2/3 log 2/3 +
1/3 log 1/3))
= 1/2 (2/3*.6 +
1/3*1.6)
= .47

•I (Type, T) =
1/6 (1) + 1/6 (1) +
1/3 (1) + 1/3 (1) = 1



Gain (Pat, T) = 1 - .47 = .53
Gain (Type, T) = 1 – 1 = 0

The ID3 algorithm is used to build 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: a set of input attributes,
                 C: the class attribute,
                 S: a training set) returns a decision tree;
   begin
     If S is empty, return a single node with value Failure;
     If every example in S has the 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;
       [note: there will be errors, i.e., improperly classified
       records];
     Let D be attribute with largest Gain(D,S) among attributes in R;
     Let {dj| j=1,2, .., m} be the values of attribute D;
     Let {Sj| j=1,2, .., m} be the subsets of S consisting
        respectively of records with value dj for attribute D;
     Return a tree with root labeled D and arcs labeled
        d1, d2, .., dm going respectively to the trees
        ID3(R-{D},C,S1), ID3(R-{D},C,S2) ,.., ID3(R-{D},C,Sm);
   end ID3;
```


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 the decision tree learning algorithm

- 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

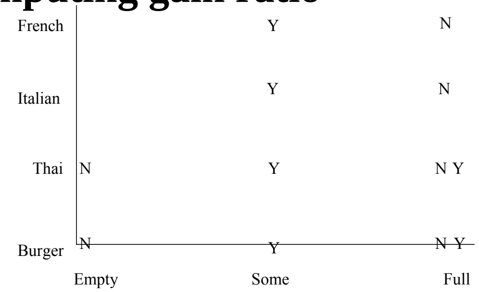
  SplitInfo(D,T) = I(|T1|/|T|, |T2|/|T|, .., |Tm|/|T|)

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

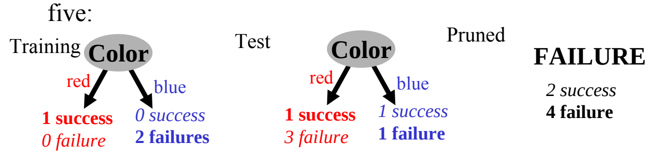
- Select a set of thresholds defining intervals
- Each interval becomes a discrete value of the attribute
- Use some simple heuristics…
  - always divide into quartiles
- Use domain knowledge…
  - divide age into infant (0-2), toddler (3 - 5), school-aged (5-8)
- Or treat this as another learning problem
  - 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 and overfitting

- 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
- The last problem, irrelevant attributes, can result in overfitting the training example data.
  - If the hypothesis space has many dimensions because of a large number of attributes, we may find **meaningless regularity** in the data that is irrelevant to the true, important, distinguishing features
  - Fix by pruning lower nodes in the decision tree
  - For example, 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 a rule set from a decision tree: write a rule for each path in the decision tree from the root to a leaf
- In that rule the left-hand side is easily 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
  - Let LHS' be obtained from LHS by eliminating some conditions
  - We can certainly replace LHS by LHS' in this rule if the subsets of the training set that satisfy respectively LHS and LHS' are equal
  - A rule may be eliminated by using metaconditions such as “if no other rule applies”

Evaluation methodology

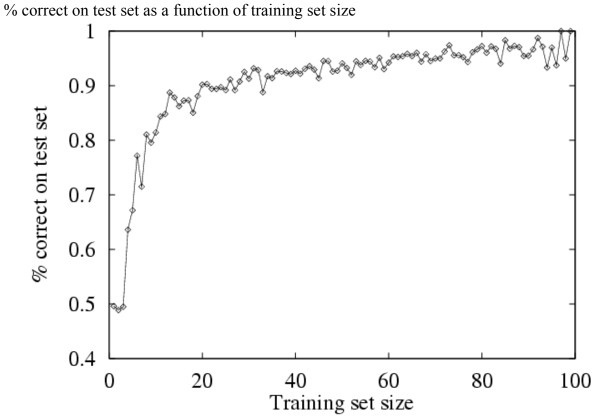
- Standard methodology:
  1. Collect a large set of examples (all with correct classifications)
  2. Randomly divide collection into two disjoint sets: training and test
  3. Apply learning algorithm to training set giving hypothesis H
  4. Measure performance of H w.r.t. test set
- Important: keep the training and test sets disjoint!
- To study the efficiency and robustness of an algorithm, repeat steps 2-4 for different training sets and sizes of training sets
- If you improve your algorithm, start again with step 1 to avoid evolving the algorithm to work well on just this collection

Performance measurement

How do we know that $h \approx f$?

- Use theorems of computational/statistical learning theory
- Try h on a new test set of examples

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



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)