Machine Learning: Decision Trees

Chapter 18.1-18.3

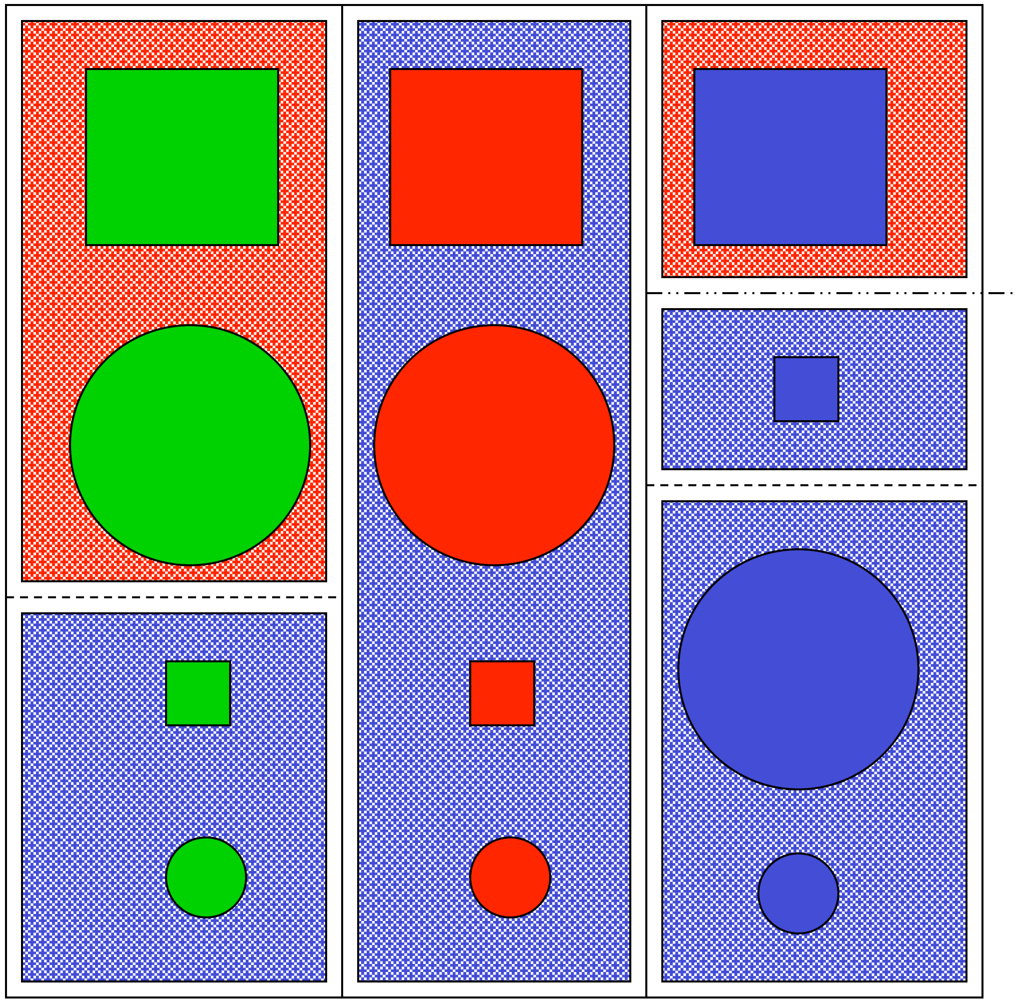
Some material adopted from notes by Chuck Dyer

Decision Trees (DTs)

- A supervised learning method used for classification and regression
- Given a set of training tuples, learn model to predict one value from the others
  - Learned value typically a class (e.g. goodRisk)
- Resulting model is simple to understand, interpret, visualize and apply

Learning a Concept

The red groups are negative examples, blue positive



Attributes

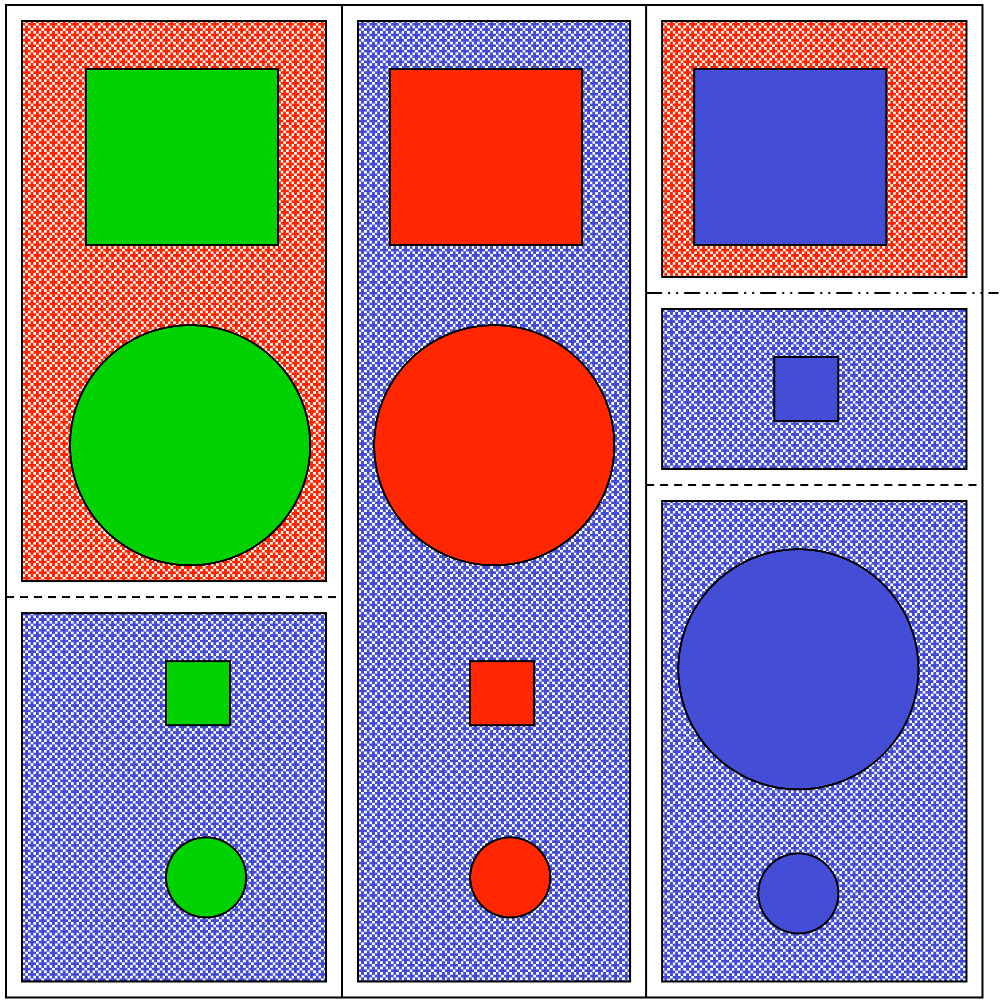
- Size: large, small
- Color: red, green, blue
- Shape: square, circle

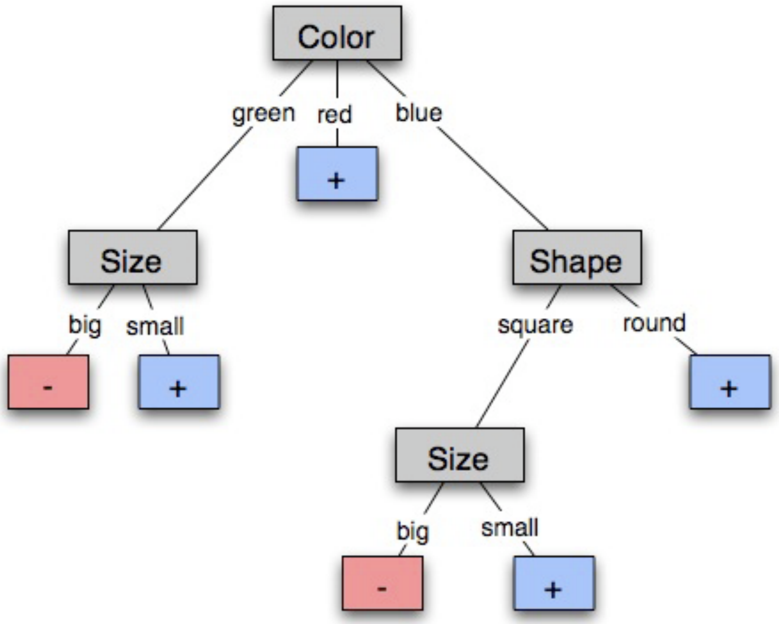
Training data

Size	Color	Shape	class
Large	Green	Square	Negative
Large	Green	Circle	Negative
Small	Green	Square	Positive
Small	Green	Circle	positive
Large	Red	Square	Positive
Large	Red	Circle	Positive
Small	Red	Square	Positive
Small	Red	Circle	Positive
Large	Blue	Square	Negative
Small	Blue	Square	Positive
Large	Blue	Circle	Positive
Small	Blue	Circle	Positive

A decision tree-induced partition

The red groups are negative examples, blue positive

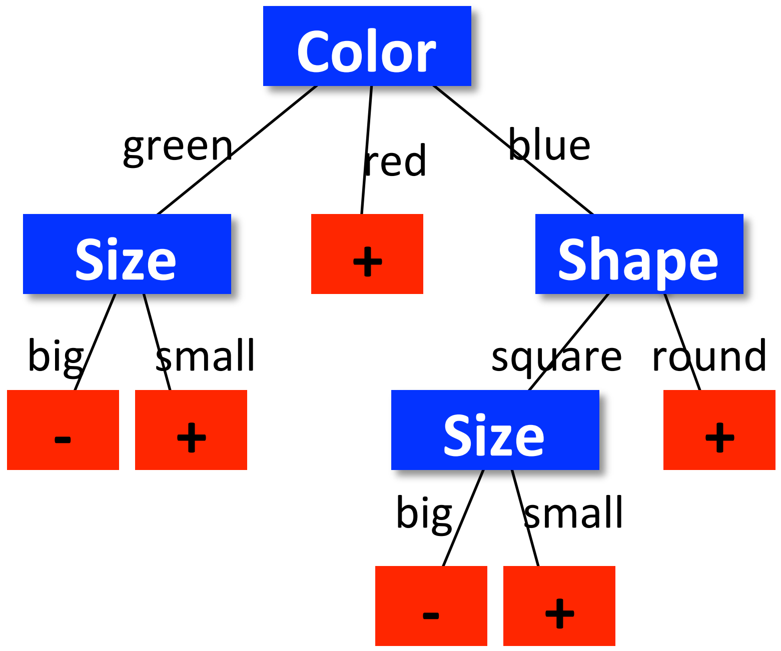




Negative things are big, green shapes and big, blue squares

Learning decision trees

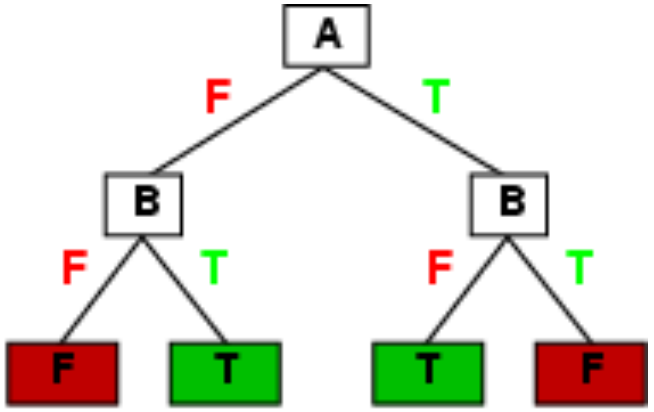
- Goal: Build a **decision tree** to classify examples as positive or negative instances of a concept using supervised learning from a training set
- A **decision tree** is a tree where
  - each non-leaf node has an attribute (feature)
  - each leaf node has a classification (+ or -)
  - each arc has a possible value of its attribute
- Generalization: allow for >2 classes
  - e.g., for stocks, classify into {sell, hold, buy}



Expressiveness of Decision Trees

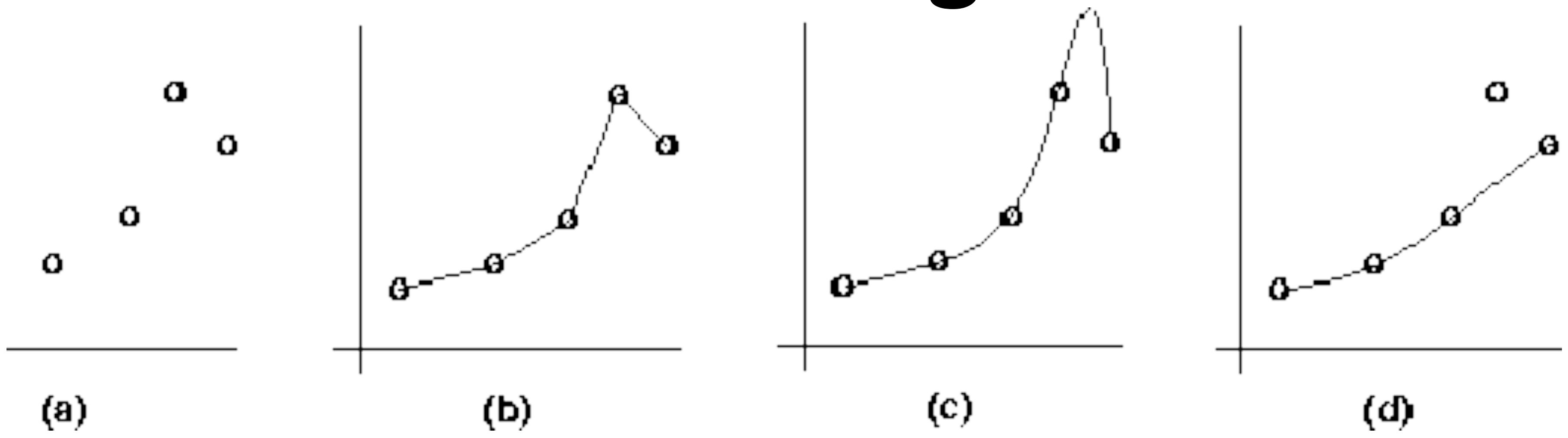
- Can express any function of the input attributes, e.g. for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- There's a consistent decision tree for any training set with one path to leaf for each example (assuming deterministic), but it probably won't generalize to new examples
- We prefer more compact decision trees

Inductive learning and bias



- Suppose that we want to learn a function **f(x) = y** and we're given 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: Occam's Razor

- William of Ockham (1285-1347)
  - "*non sunt multiplicanda entia praeter necessitatem*"
  - entities are not to be multiplied beyond necessity
- **Simplest** consistent explanation is the best
- **Smaller** decision trees correctly classifying training examples preferred over larger ones
- Finding **the** smallest decision tree is NP-hard, so we use algorithms that find reasonably small ones

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* $\wedge$ ¬*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

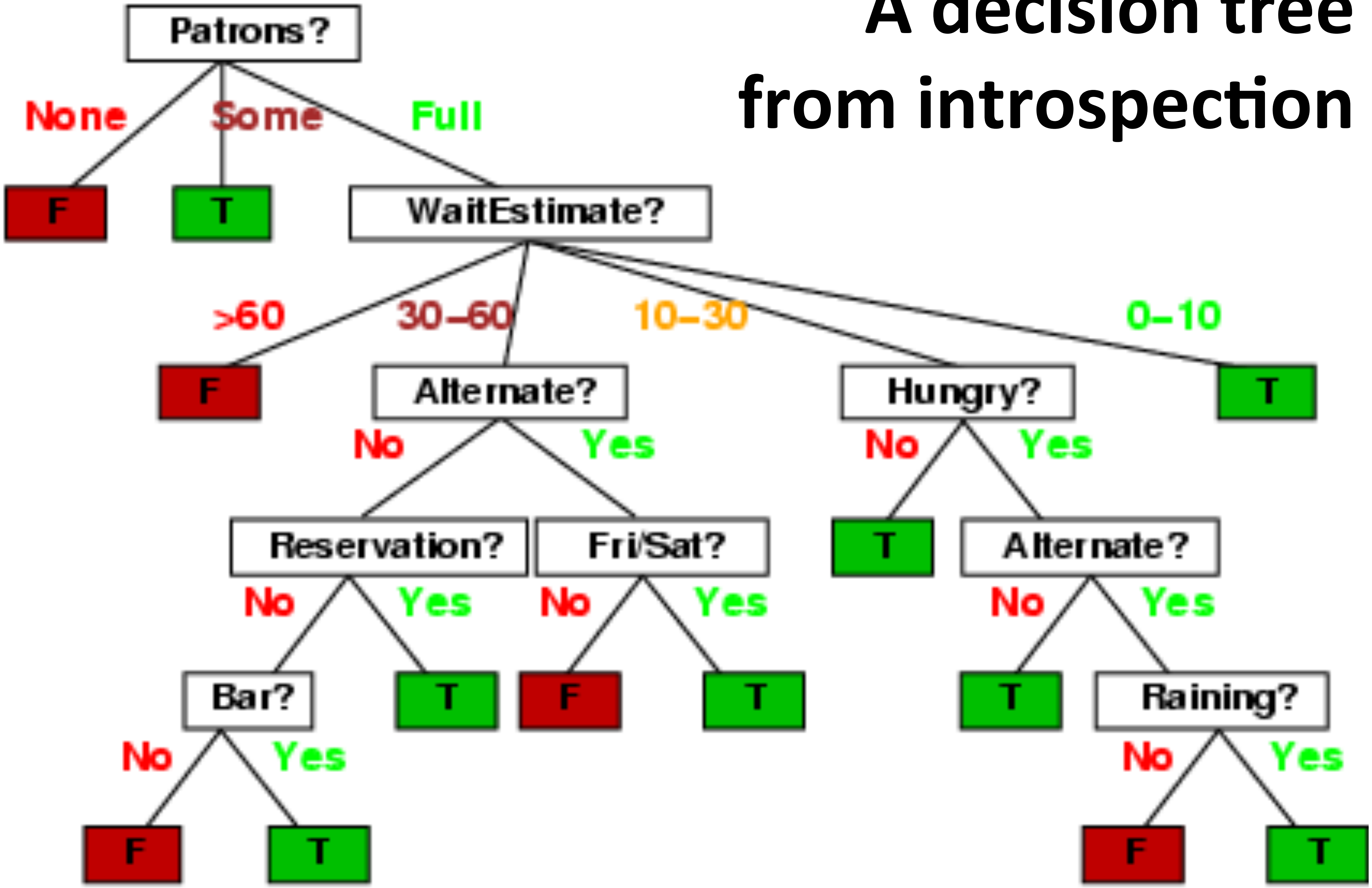
- Develop decision tree for decision patron makes when deciding whether or not to wait for a table
- Two classes: wait, leave
- Ten attributes: Alternative available? Bar in restaurant? Is it Friday? Are we hungry? How full is the restaurant? How expensive? Is it raining? Do we have reservation? What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

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

A decision tree from introspection



Issues

- It’s like 20 questions
- We can generate many decision trees depending on what attributes we ask about and in what order
- How do we decide?
- What makes one decision tree better than another: number of nodes? number of leaves? maximum depth?

ID3 / C4.5 / J48 Algorithm

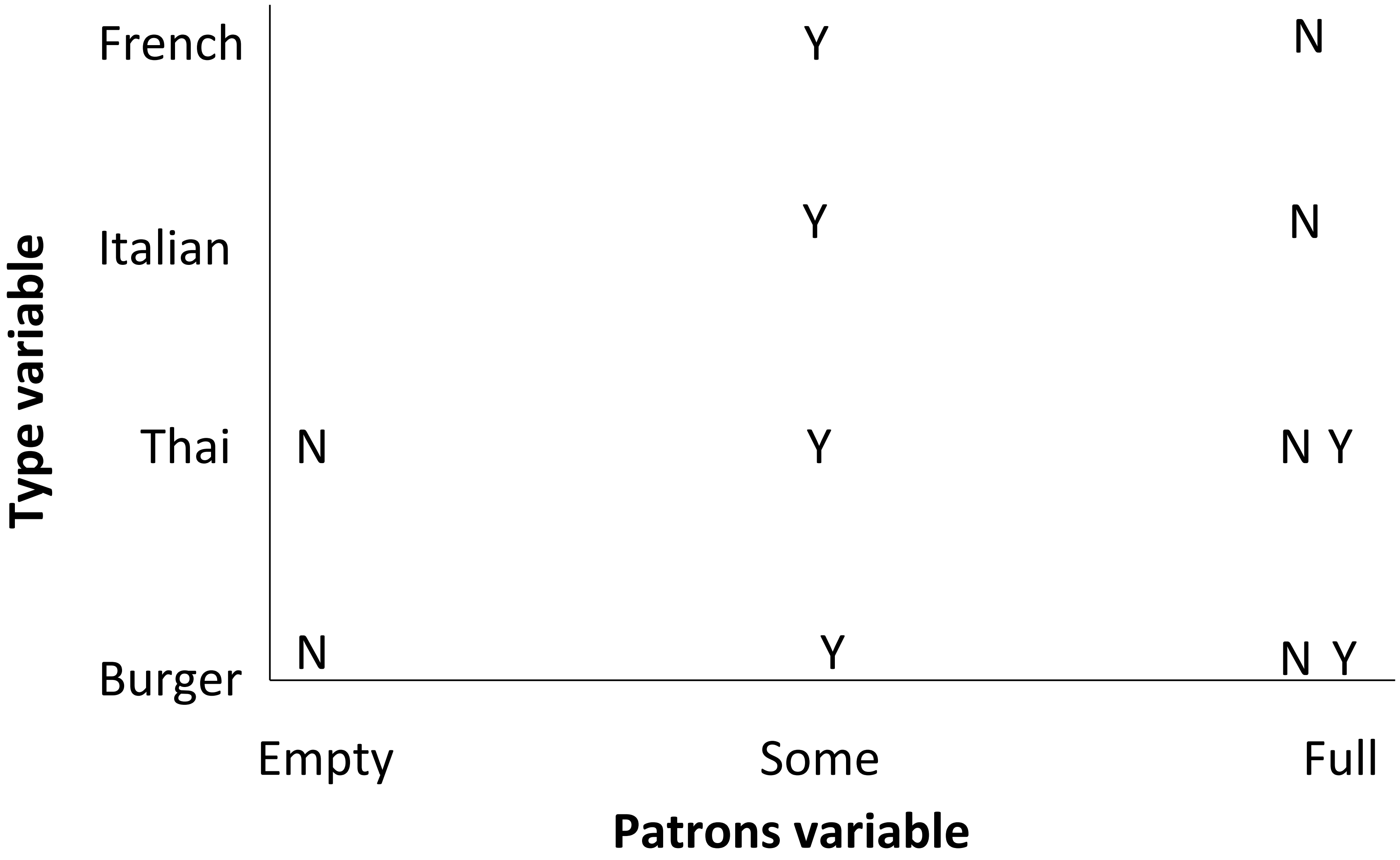
- Greedy algorithm for decision tree construction developed by Ross Quinlan circa 1987
- Top-down construction of tree by recursively selecting *best attribute* to use at current node
  - Once attribute selected for current node, generate child nodes, one for each possible value of attribute
  - Partition examples using values of attribute, & assign these subsets of examples to appropriate child node
  - Repeat for each child node until all examples associated with node are all positive or negative

Choosing the best attribute

- Key problem: choose attribute to split a given set of examples
- Possibilities for choosing attribute:
  - **Random:** Select one at random
  - **Least-Values:** one with smallest # of possible values
  - **Most-Values:** one with largest # of possible values
  - **Max-Gain:** one with largest expected ***information gain*** – i.e., 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

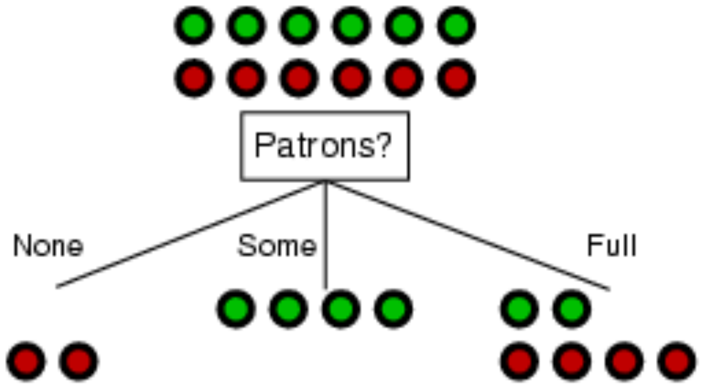
Random: Patrons or Wait-time; **Least-values**: Patrons; **Most-values**: Type; **Max-gain**: ???

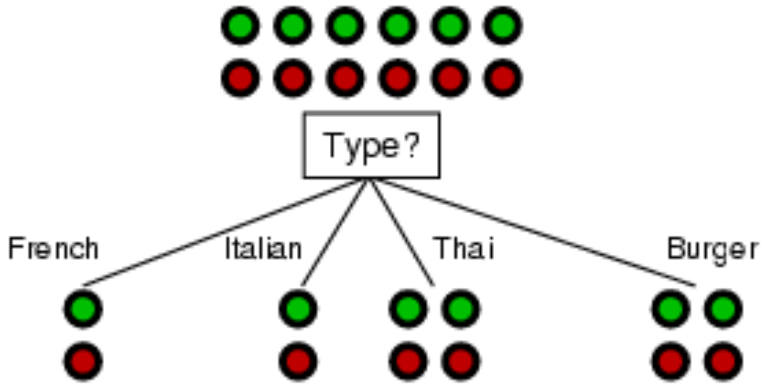


Choosing an attribute

● stay
● leave

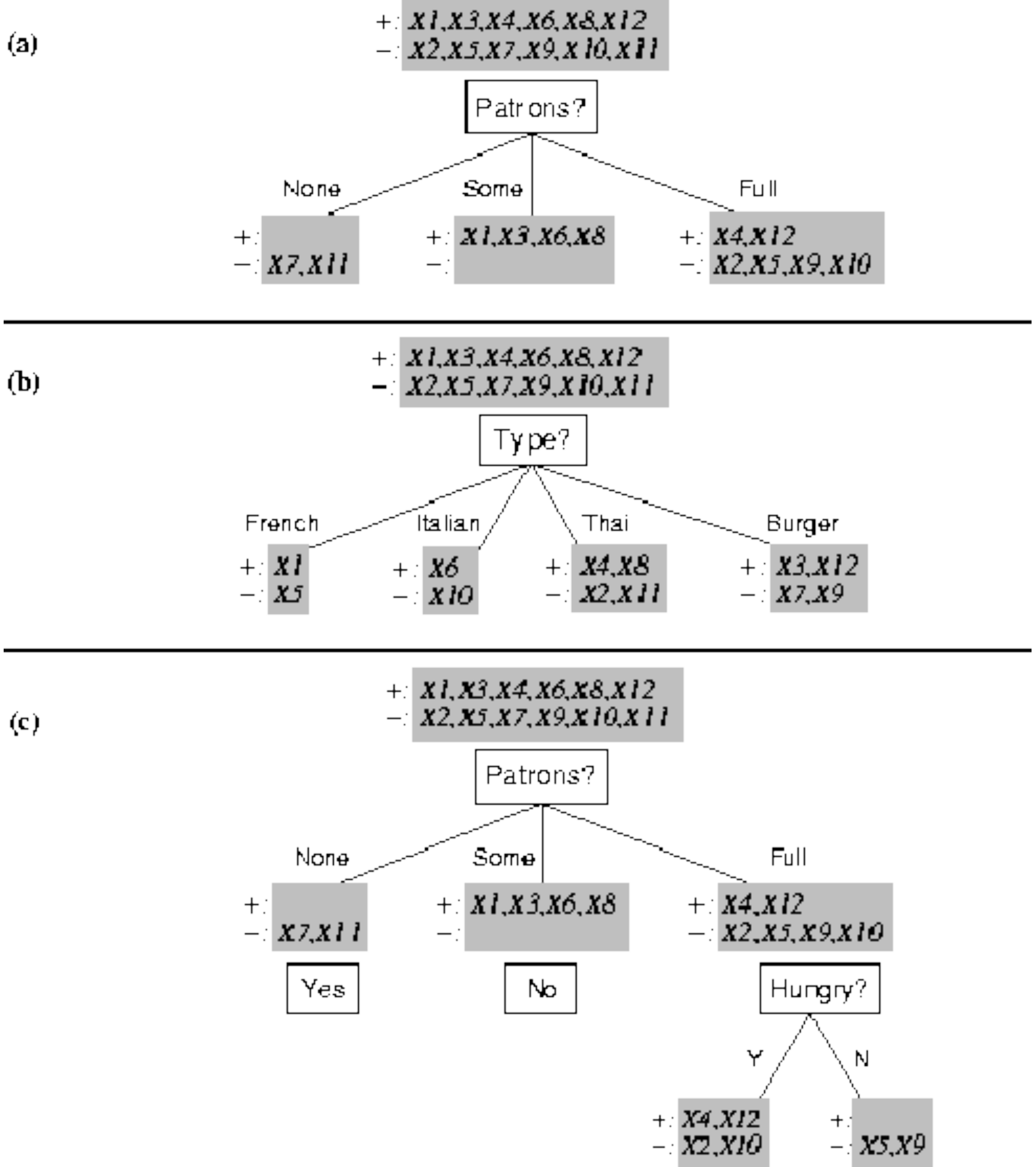
Idea: good attribute splits examples into subsets that are (ideally) *all positive* or *all negative*



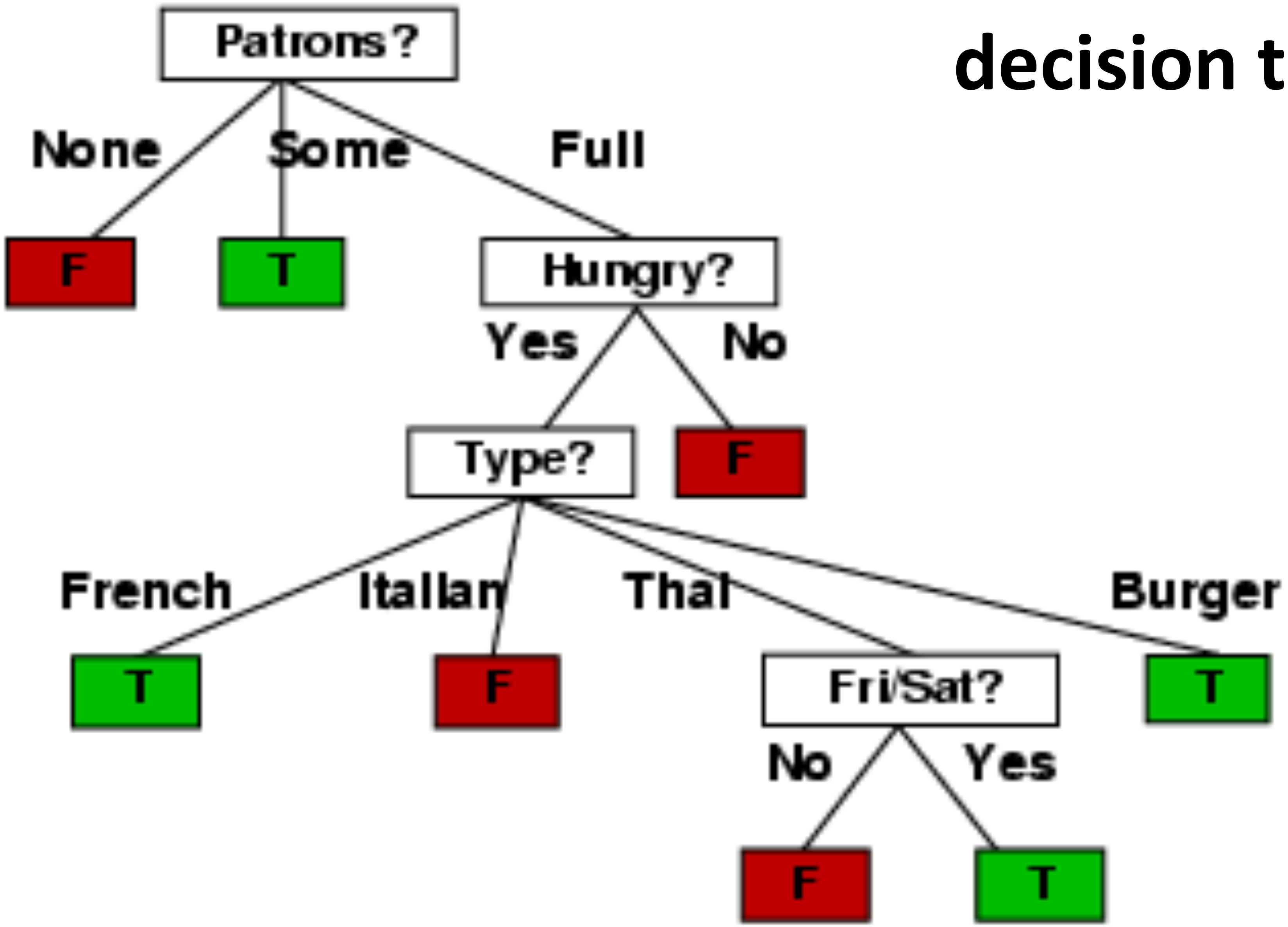


Which is better: *Patrons?* or *Type?*

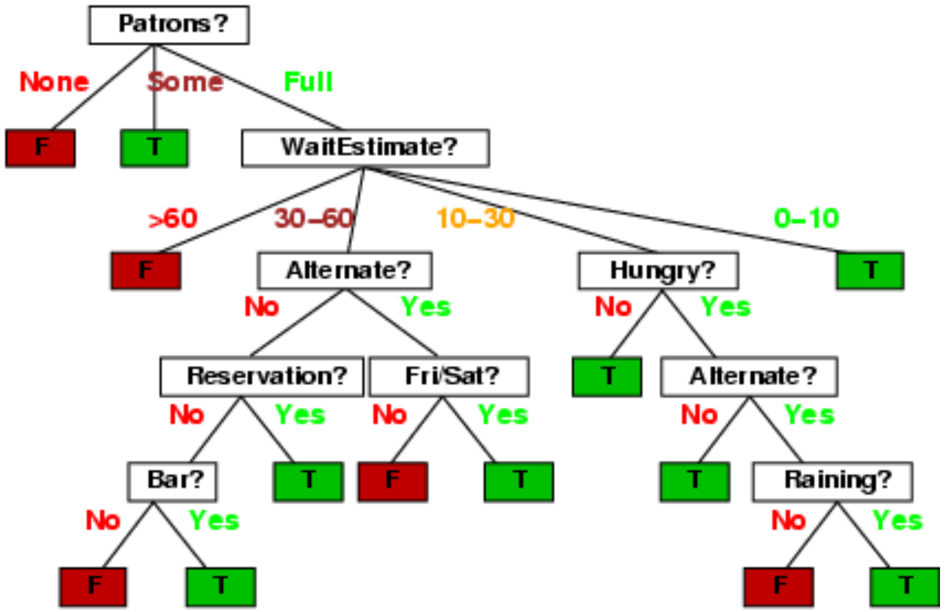
Splitting examples by testing attributes

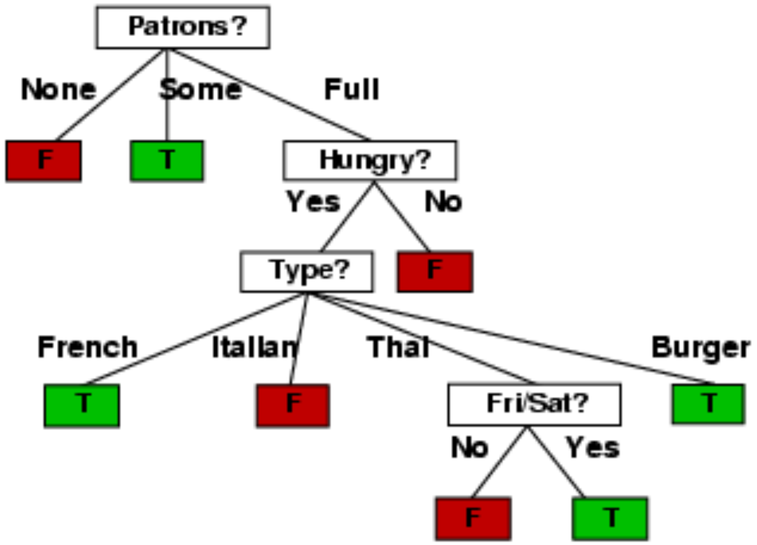


ID3-induced decision tree



Compare the two Decision Trees



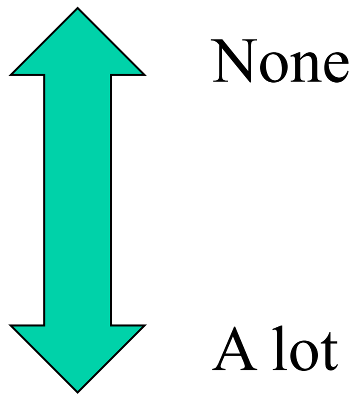


Information theory 101

- Sprang fully formed from Claude Shannon's seminal work: Mathematical Theory of Communication in 1948
- Intuitions
  - Common words (a, the, dog) shorter than less common ones (parlimentarian, foreshadowing)
  - morse code: common letters have shorter encodings
- Information inherent in data/message (inform-ation entropy) measured in minimum number of bits needed to store/send using a good encoding

Information theory 101

- Information entropy ... tells how much information there is in an event. The more uncertain or random the event is, the more information it contains.
- Receiving a message is an event
- How much information is in these messages
  - – The sun rose today!
  - – It's sunny today in Honolulu!
  - – The coin toss is heads!
  - – It's sunny today in Seattle!
  - – Life discovered on Mars!



Information theory 101

- For **n equally probable** possible messages or data values, each has probability **1/n**
- Information of 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
- If probability distribution P (p_1,p_2 … p_n) for n messages, its information (aka H or *entropy*) is:

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

Entropy of a distribution

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

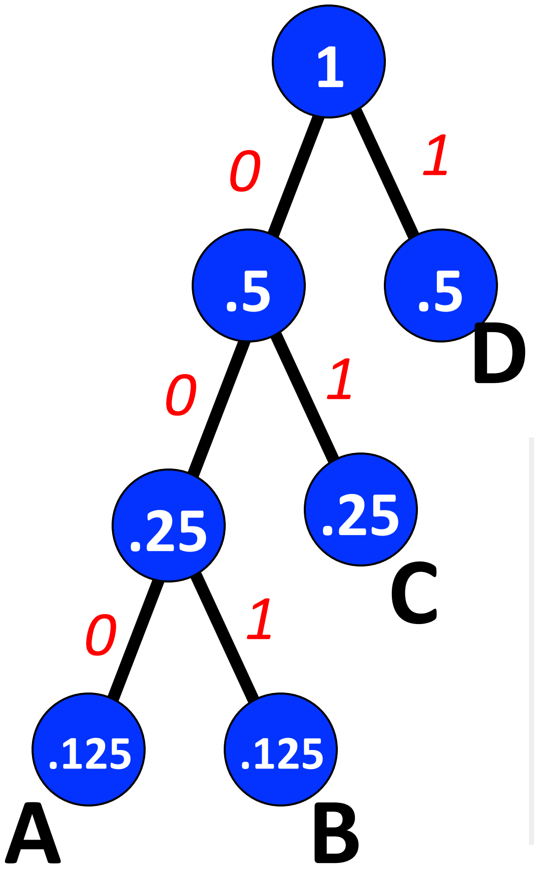
- Examples:
  - If P is (0.5, 0.5) then I(P) = 0.5*1 + 0.5*1 = 1
  - If P is (0.67, 0.33) then I(P) = -(2/3*log(2/3) + 1/3*log(1/3)) = 0.92
  - If P is (1, 0) then I(P) = 1*1 + 0*log(0) = 0
- More uniform probability distribution, greater its information: more information is conveyed by a message telling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

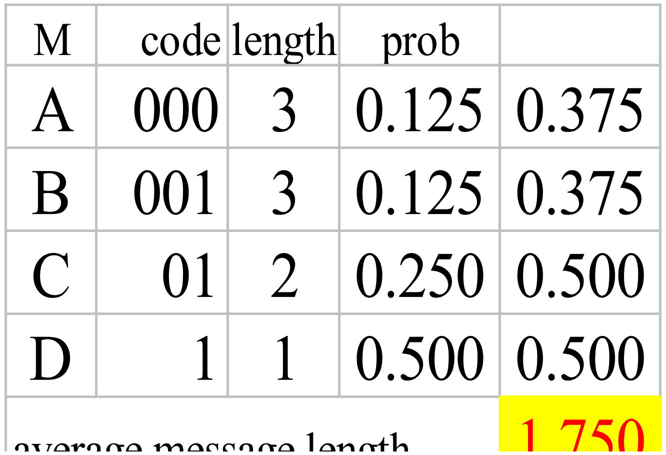
Example: Huffman code

- In 1952 MIT student David Huffman devised (for a homework assignment!) a coding scheme that's optimal when all data probabilities are powers of 1/2
- A Huffman code can be built in the following manner:
  - Rank symbols in order of probability of occurrence
  - Successively combine two symbols of lowest probability to form a new composite symbol; eventually we will build a binary tree where each node is probability of all nodes beneath it
  - Trace path to each leaf, noticing 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

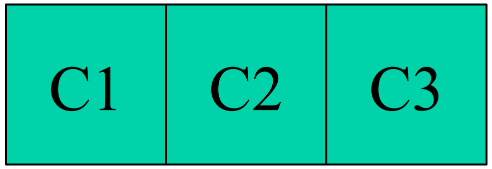
Information for classification

If set T of records is divided into disjoint exhaustive classes (C_1,C_2,..,C_k) by value of class attribute, then information needed to identify class of an element of T is:

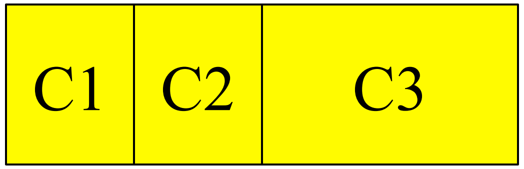
$$\text{Info}(T) = I(P)$$

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

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



High information

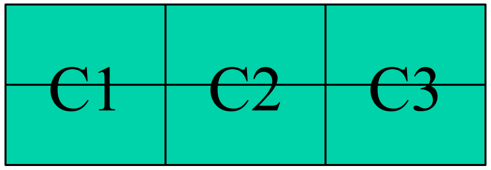


Lower information

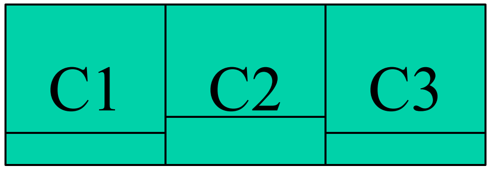
Information for classification II

If we further divide T wrt attribute X into sets $\{T_1, T_2, .., T_n\}$, the information needed to identify 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

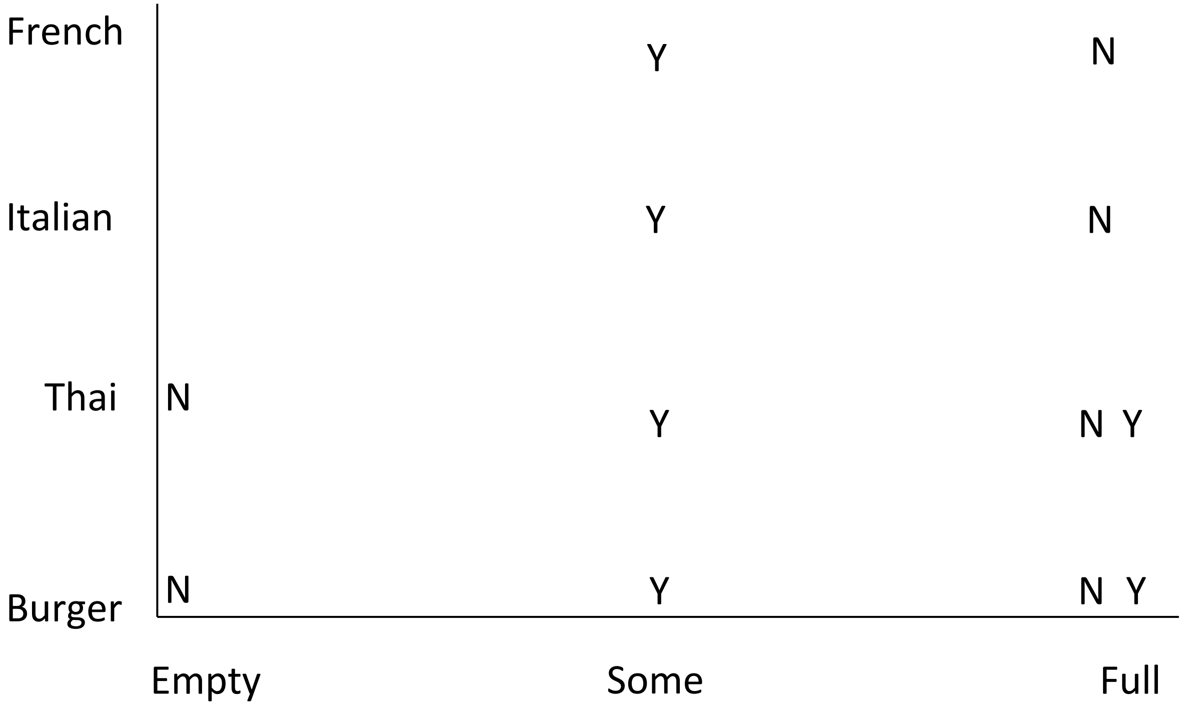
Information gain

- **Gain(X,T) = Info(T) - Info(X,T)** is difference of
  - info needed to identify element of T and
  - info needed to identify element of T after value of attribute X known
- This is gain in information due to attribute X
- Used to rank attributes and build DT where each node uses attribute with greatest gain of those not yet considered in path from root
- Intent: create small DTs to minimize questions

Computing Information Gain

Should we ask about restaurant type or how many patrons there are?

- I(T) = ?
- I (Pat, T) = ?
- I (Type, T) = ?



Gain (Patrons, T) = ?

Gain (Type, T) = ?

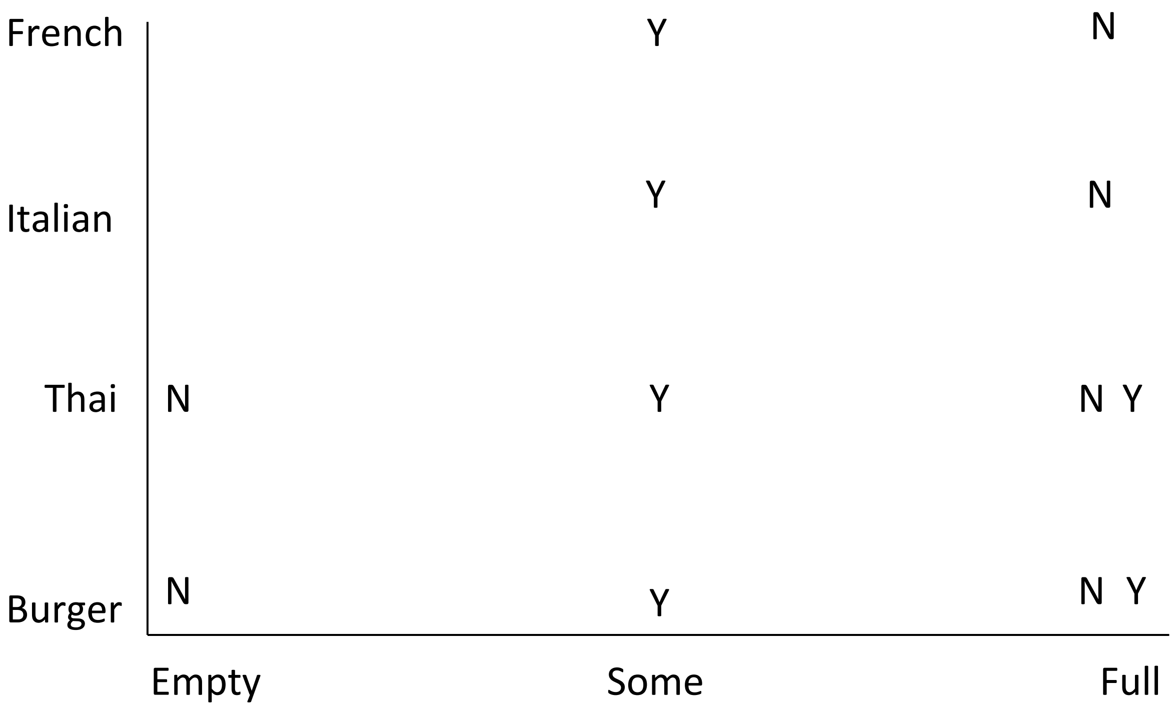
$$I(P) = -(p_1*\log(p_1) + p_2*\log(p_2) + .. + p_n*\log(p_n))$$

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

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

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?

Case studies show that decision trees often at least as accurate as human experts

- Study for diagnosing breast cancer had humans correctly classifying the examples 65% of the time; DT classified 72% correct
- British Petroleum designed DT 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

- Information gain criterion favors attributes that have a large number of values
  - An attribute D with a distinct value for each record has Info(D,T) 0, thus Gain(D,T) is maximal
- To compensate, use GainRatio instead of Gain:

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

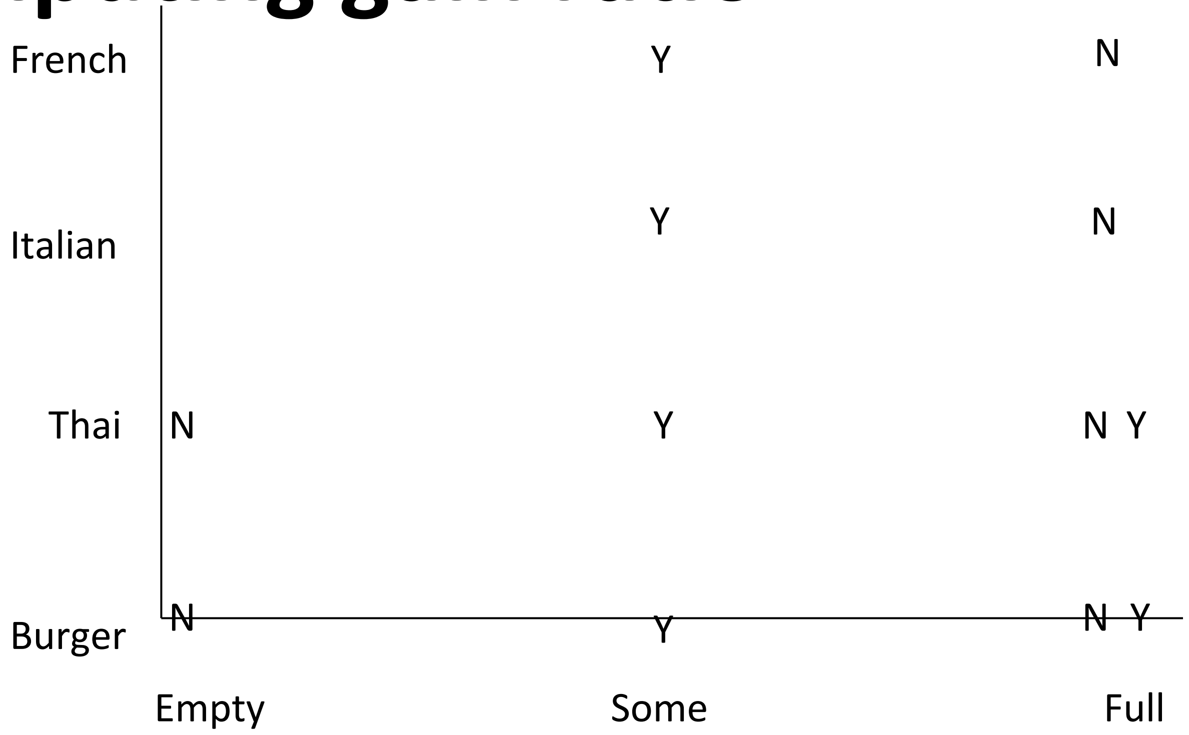
  SplitInfo(D,T) = I(|T1|/|T|, |T2|/|T|, .., |Tm|/|T|)

where {T1, … Tm} is 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 thresholds defining intervals so each becomes a discrete value of attribute
- Use heuristics, e.g. always divide into quartiles
- Use domain knowledge, e.g. divide age into infant (0-2), toddler (3-5), school-aged (5-8)
- Or treat this as another learning problem
  - Try different ways to discretize continuous variable; 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 training data

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

Overfitting

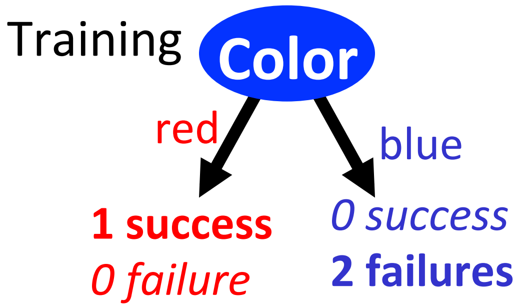
- *Overfitting* occurs when a statistical model describes random error or noise instead of underlying relationship
- If hypothesis space has many dimensions (large number of attributes) we may find **meaningless regularity** in the data that is irrelevant to the true, important, distinguishing features
- If we have too little training data, even a reasonable hypothesis space can overfit

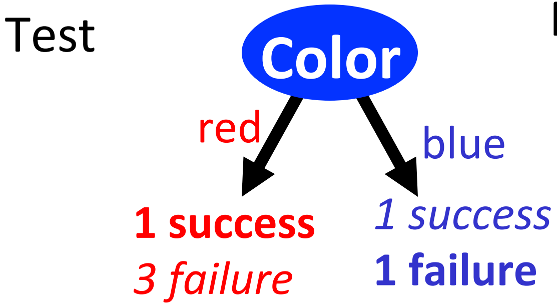
Overfitting

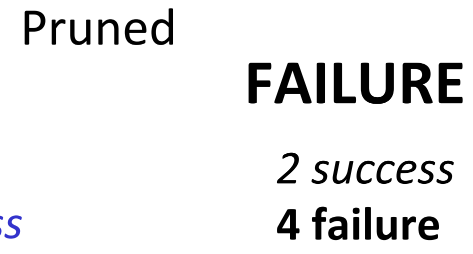
- Fix by by removing irrelevant features
  - E.g., remove 'year observed', 'month observed', 'day observed', 'observer name' from feature vector
- Fix by getting more training data
- Fix by pruning lower nodes in the decision tree
  - E.g., if gain of best attribute at a node is below a threshold, stop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

Pruning decision trees

- Pruning a decision tree is done by replacing a whole subtree by a leaf node
- Replacement takes place if the expected error rate in the subtree is greater than in the single leaf, e.g.,
  - Training: 1 training red success and 2 training blue failures
  - Test: 3 red failures and one blue success
  - Consider replacing this subtree by a single Failure node.
- After replacement, only 2 errors instead of 5







Converting decision trees to rules

- It’s 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”

Summary: decision tree learning

- Widely used learning methods in practice for problems with relatively few features
- Strengths
  - Fast and simple to implement
  - Can convert result to a set of easily interpretable rules
  - Empirically valid in many commercial products
  - Handles noisy data
  - Easy for people to understand
- Weaknesses
  - 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)