Machine Learning: Decision Trees

Chapter 19.3

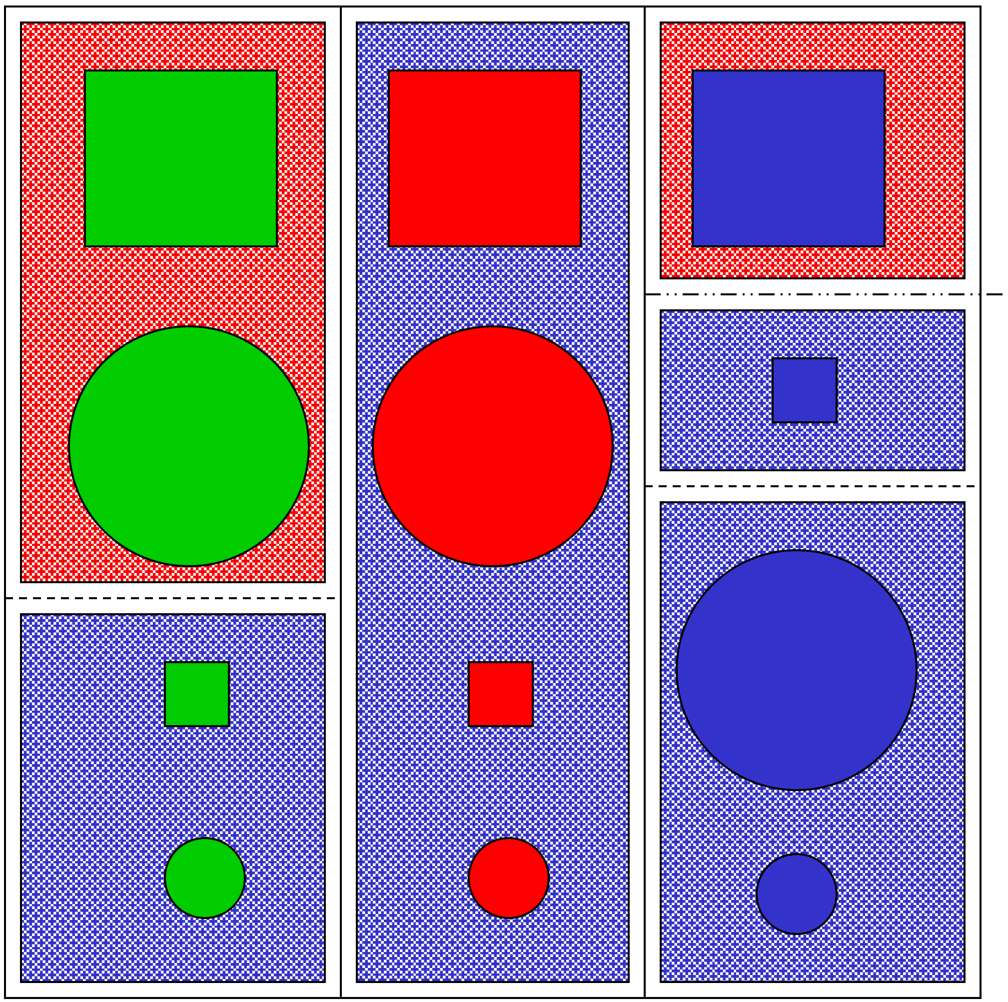
Some material adopted from notes by Chuck Dyer

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

Task

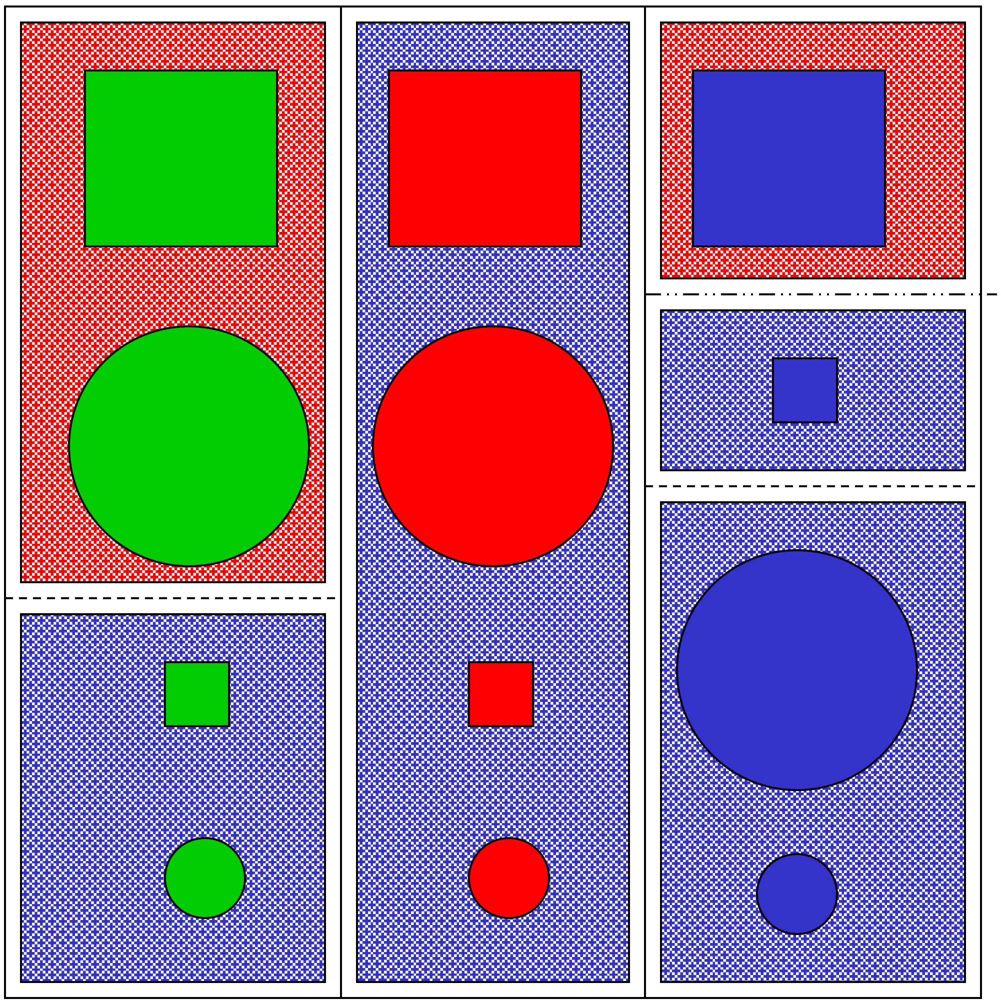
Classify new object with a size, color & shape as positive or negative

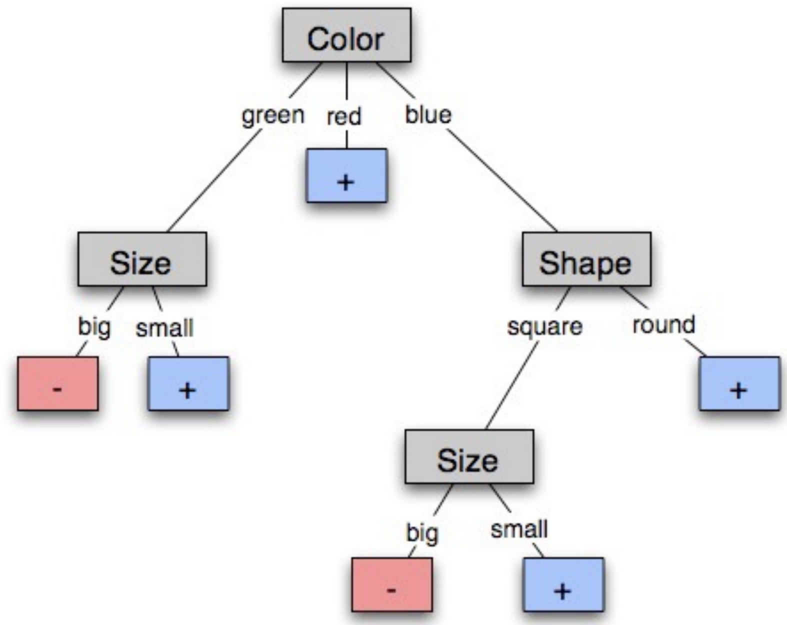
Training data

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

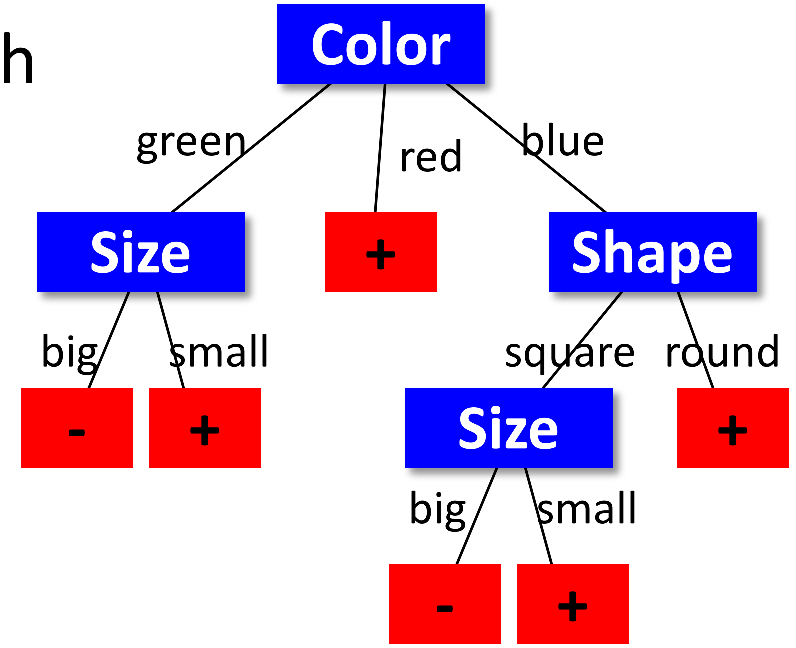




Negative things are big, green shapes and big, blue squares

Learning decision trees

- Goal: Build **decision tree** to classify examples as positive or negative instances of concept using supervised learning from training data
- A **decision tree** is a tree in which
  - **non-leaf nodes** have an attribute (feature)
  - **leaf nodes** have a classification (+ or -)
  - **arcs** have a possible value of its attribute

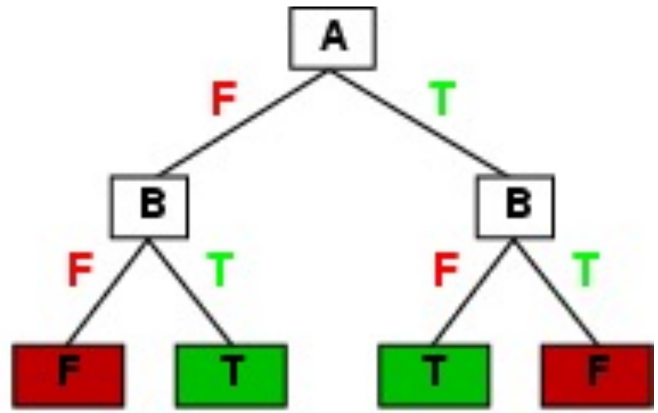


- Generalization: allow for >2 classes
  - e.g., classify stocks as {sell, hold, buy}

Expressiveness of Decision Trees

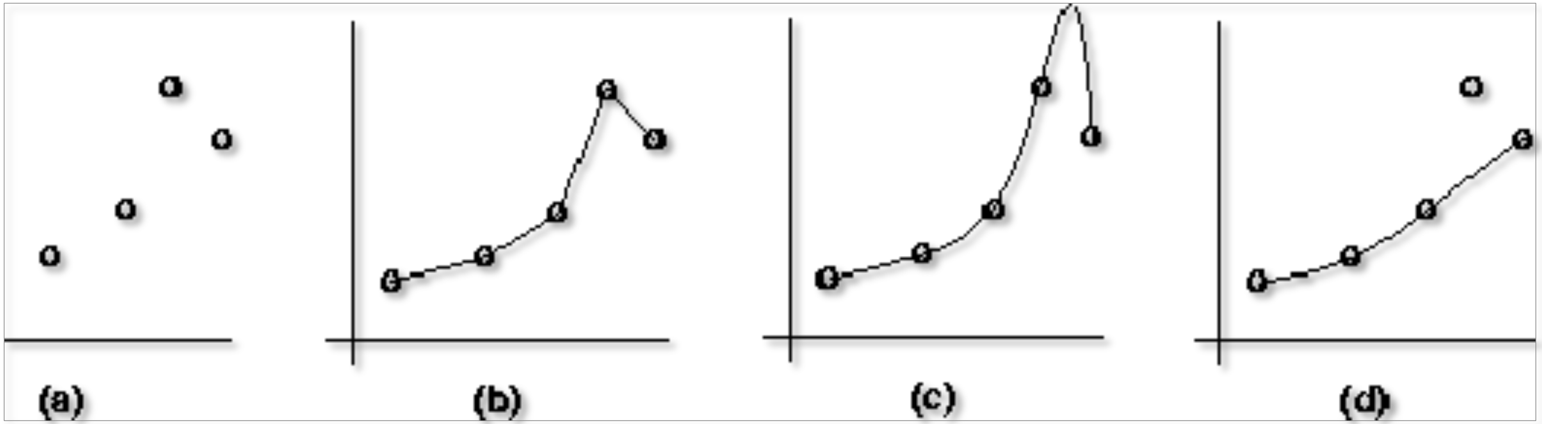
- Can express any function of input attributes, e.g., for Boolean functions, truth table row → path to leaf:

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, but it probably won't **generalize** to new examples
- 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)
- Can make several hypotheses about f, e.g.: (b), (c) & (d)
- Preference reveals learning technique **bias**, e.g.:
  - prefer piece-wise functions (b)
  - prefer a smooth function (c)
  - prefer a simple function and treat outliers as noise (d)

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

R&N's restaurant domain

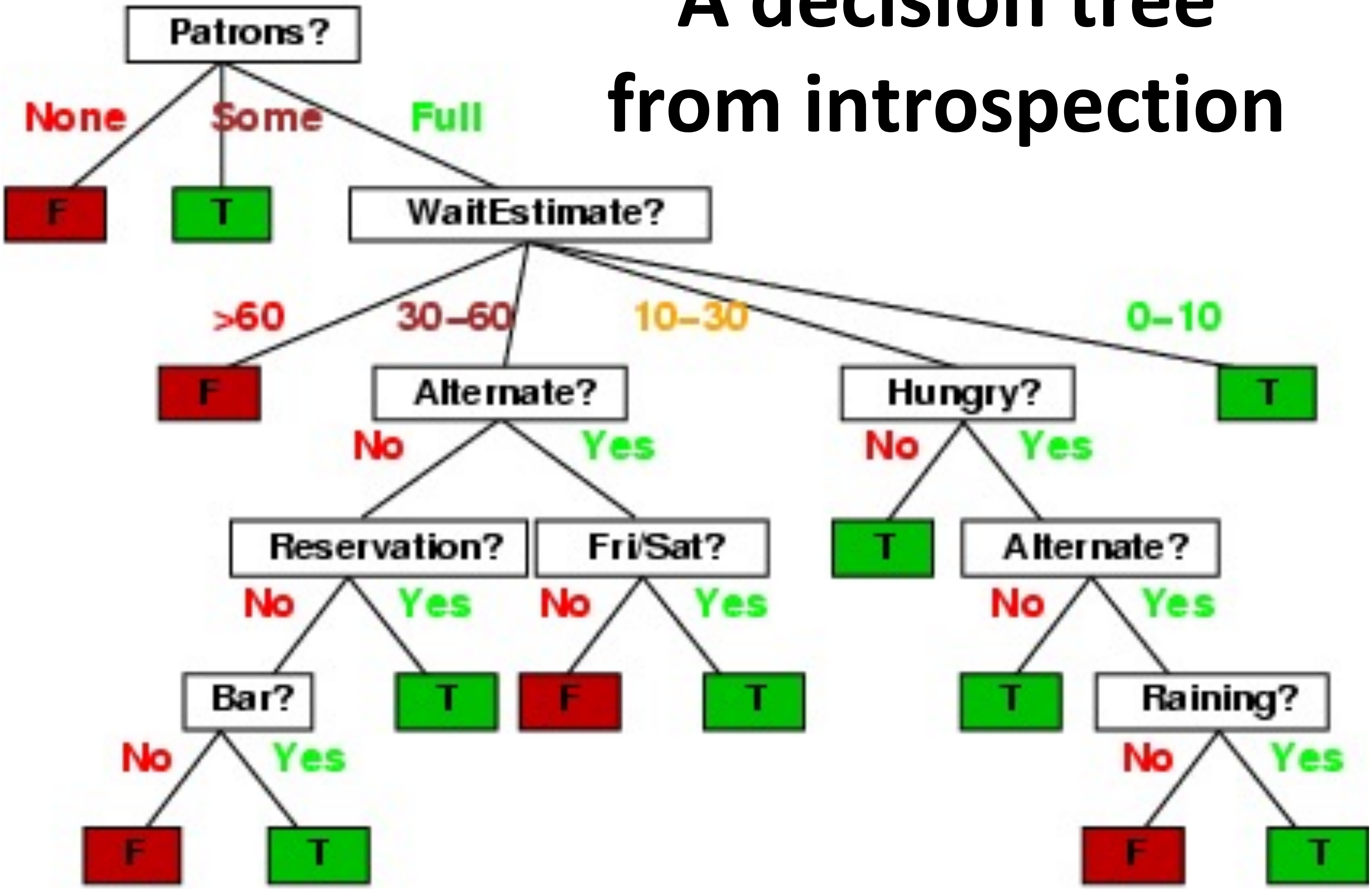
- Develop decision tree that customers make when deciding whether to wait for a table or leave
- **Two classes**: wait, leave
- **Ten attributes**: Alternative available? Bar in restaurant? Is it Friday? Are we hungry? How full is restaurant? How expensive? Is it raining? Do we have reservation? What type of restaurant is it? Estimated waiting time?
- Set of **12 training examples**
- ~7,000 possible cases (i.e., combinations of values)

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 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 attribute value
  - Partition examples using values of attribute, & assign these subsets of examples to the child nodes
  - Repeat for each child node until examples associated with a node are all positive or negative

Choosing best attribute

- Key problem: choose attribute to split 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***
  - **Gini impurity:** one with smallest gini impurity value
- The last two measure the **homogeneity** of the target variable within the subsets
- The ID3 algorithm uses **max-gain**

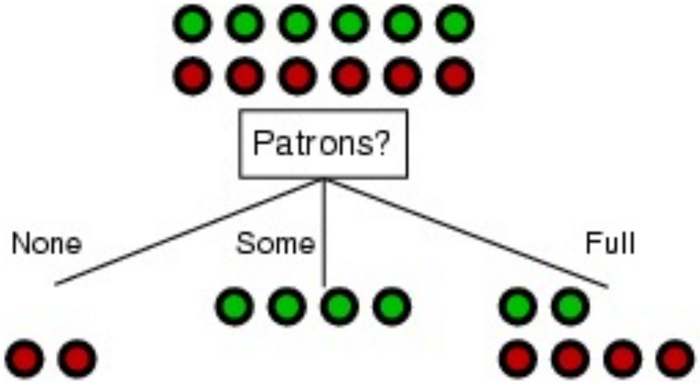
A Simple Example

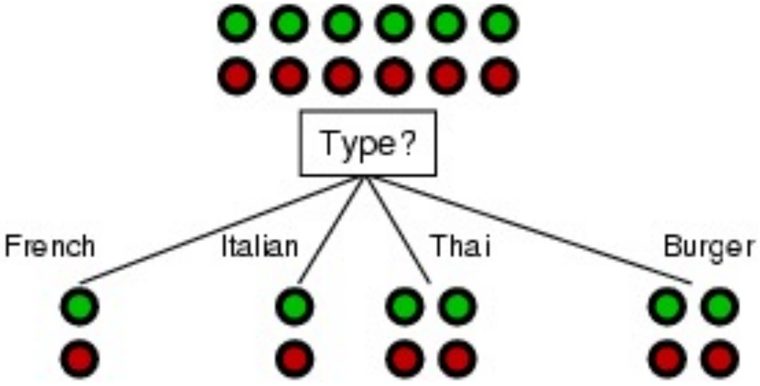
For this data, is it better to start the tree by asking about the restaurant **type** or its current **number of patrons**?

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

Choosing an attribute

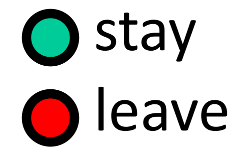
Idea: good attribute splits examples into subsets that are (ideally) *all positive* or *all negative*



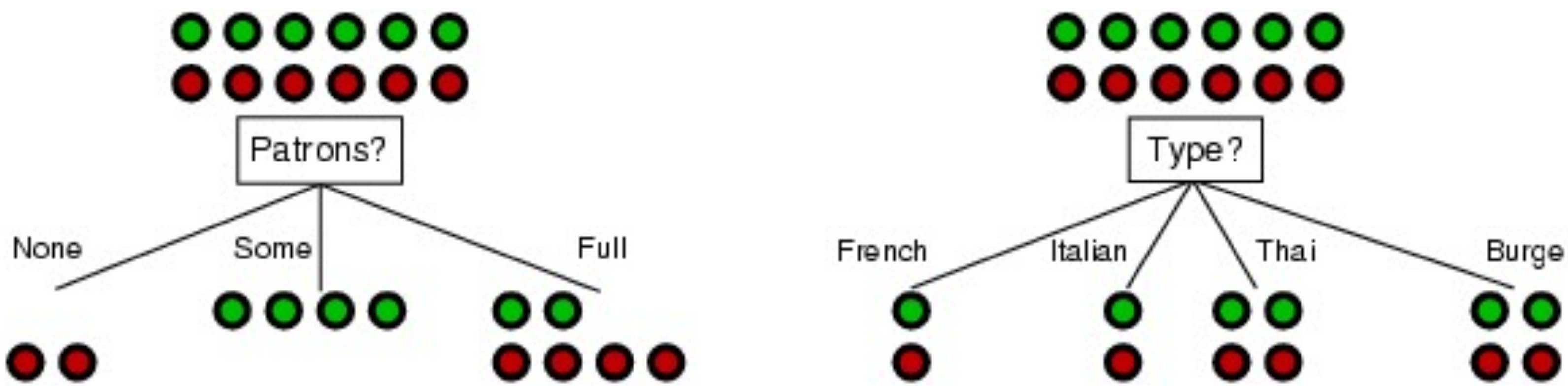


Which is better: *Patrons?* or *Type?*

Choosing an attribute

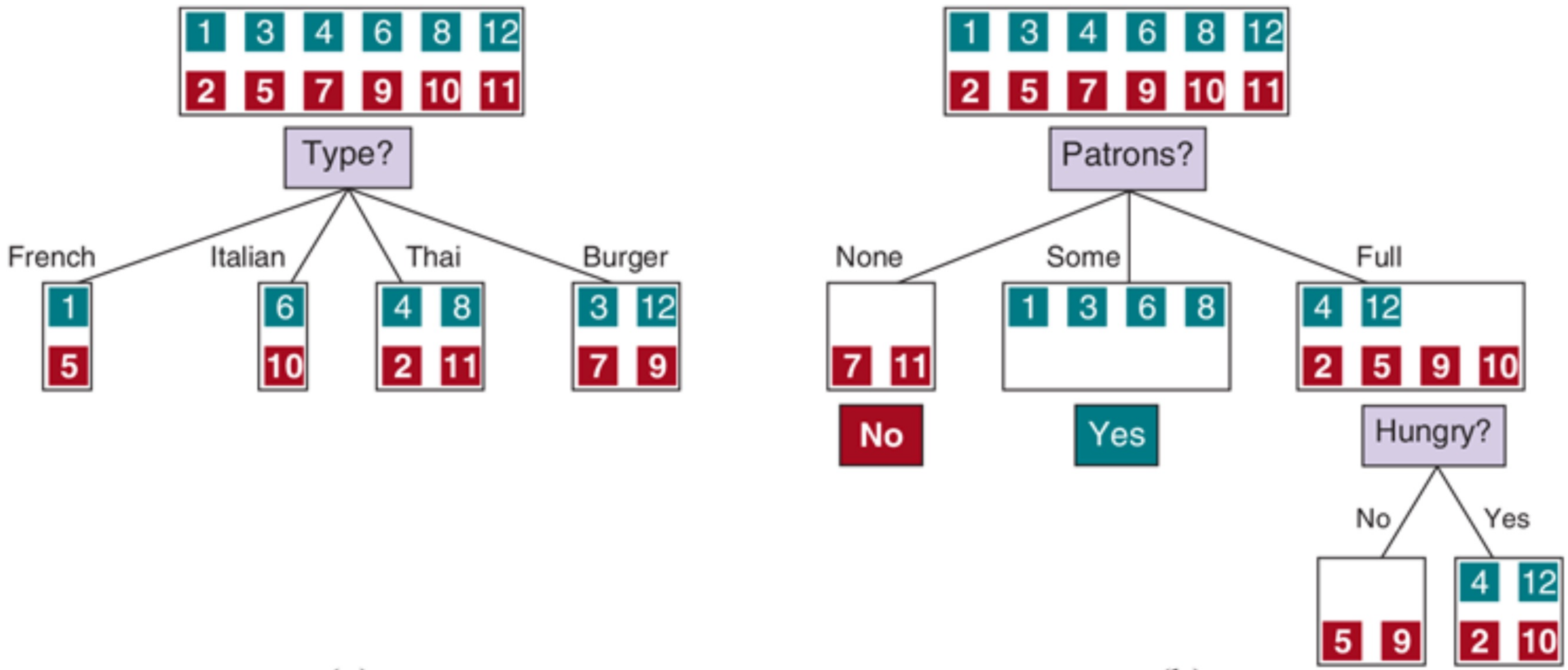


Idea: good attribute splits examples into subsets that are (ideally) *all positive* or *all negative*



- **Patrons:** for six examples we know the answer and for six we can predict with probability 0.67
- **Type:** our prediction is no better than chance (0.50)

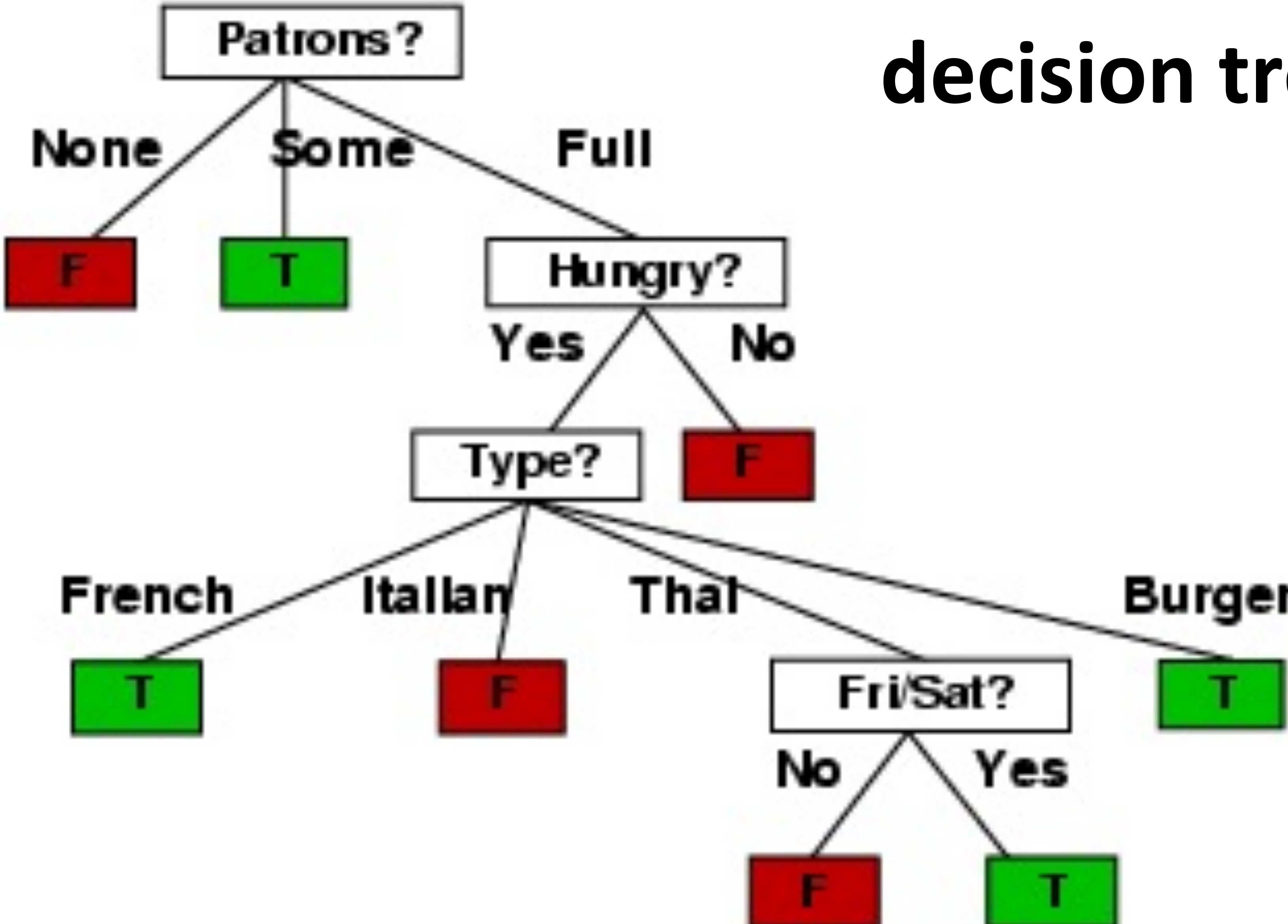
Choosing Patrons yields more information



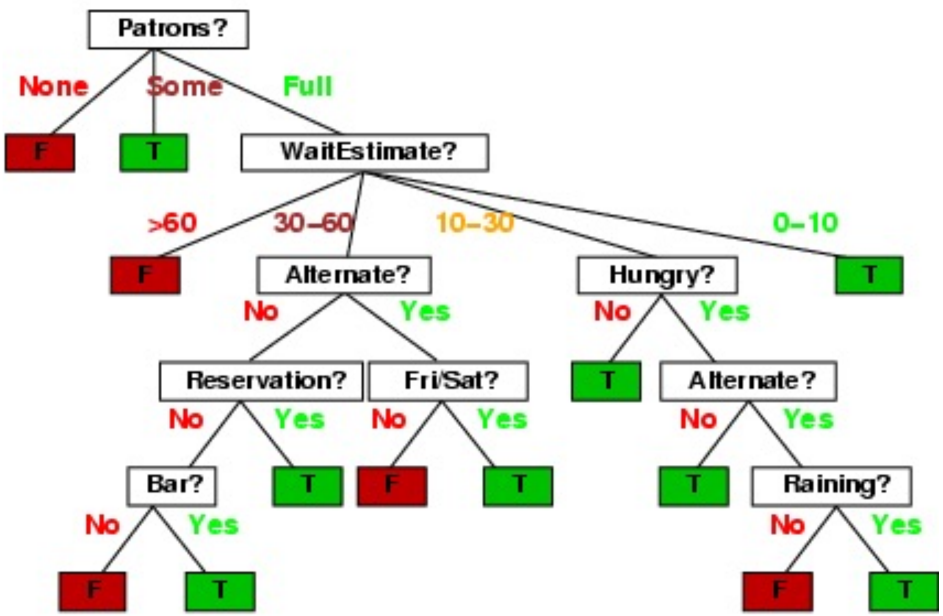
(a) (b)

The ID3 algorithm used this to decide what attribute to ask bout next when building a decision tree

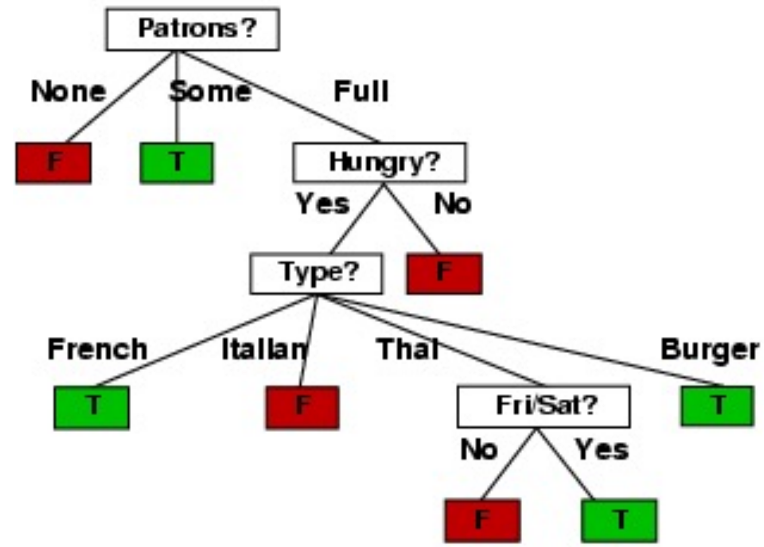
ID3-induced decision tree



Compare the two Decision Trees



Human-generated decision tree



ID3-generated decision tree

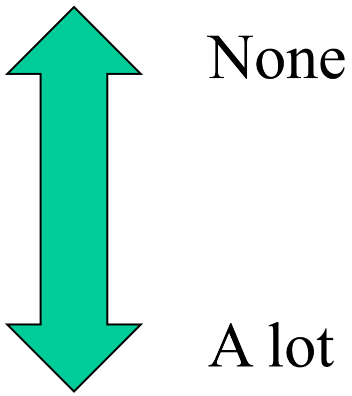
- Intuitively, ID3 tree looks better: it's shallower and has fewer nodes
- ID3 uses **information theory** to decide which question is best to ask next

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 the number of bits needed to store/send using an optimal encoding

Information theory 101

- Information entropy ... tells how much information there is in an event or message.
- More uncertain it is, 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**
- Def: Information of a message is $-\log_2(p) = \log_2(n)$

  e.g., with 16 messages, then log(16) = 4 and we need **4 bits** to identify/send each message
- What if the messages are not equally likely?
- For **probability distribution P** $(p_1, p_2 \ldots p_n)$ for n messages, its information (*H* or *information entropy*) is:

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

Information 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

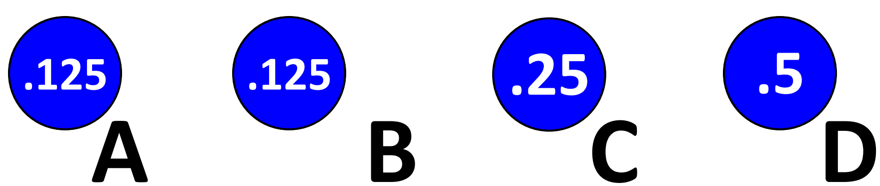
Example: Huffman code

- 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 as followings:
  - Rank symbols in order of probability of occurrence
  - Successively combine 2 symbols of lowest probability to form new symbol; eventually we get binary tree where each node is probability of nodes below
  - Trace path to each leaf, noting direction at each node

Huffman code example

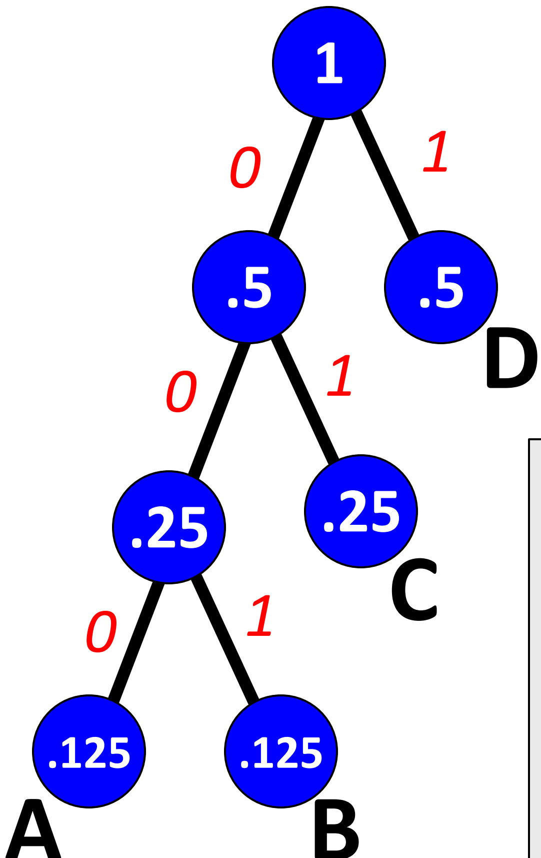
M	P
A	.125
B	.125
C	.25
D	.5

- Four possible messages (A, B, C, D) each with a probability of being sent
- We could encode them using 2 bits per message: A=00, B=01, C=10, D=11
- Sending 1,000 messages will require 2,000 bits
- We can do better with a Huffman code!



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

- Using this code for many messages (A,B,C or D), average bits/message should approach 1.75
- Sending 1000 messages will need ~1750 bits not 2000 bits

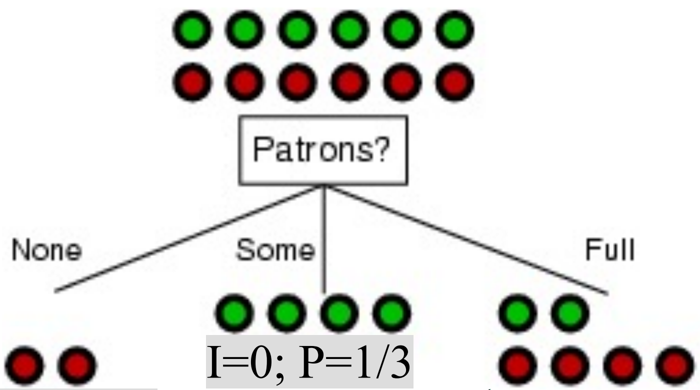
Information gain

- **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
- goal: **create small DTs** to minimize questions

Information Gain

stay
leave

$I = .5*\log_2(.5) + .5*\log_2(.5) = 0.5+0.5 = 1$

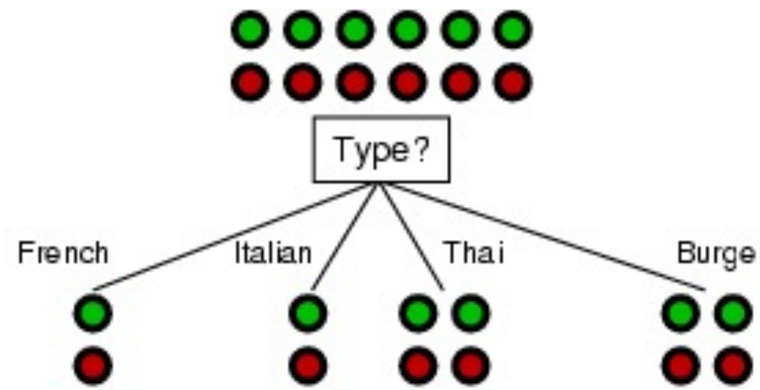


I=0; P=1/6

I=0; P=1/3

$I=(1/3*\log_2(1/3)+2/3*\log_2(2/3); P=1/2 = 0.46$

Information gain = 1 - 0.46 = 0.54



I=1;P=1/6 I=1; P=1/6 I=1; P=2/6 I=1; P=2/6

$I = 6/6*1 = 1$

Information gain = 1 - 1 = 0

- **Information gain** for asking **Patrons = 0.56**, for asking **Type = 0**
- Note: If only one of the N categories has any instances, the information entropy is always 0

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 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 other selection metric gain ratios, e.g., gini impurity metric
- Handle real-valued data
- Noisy data and overfitting
- Generation of rules
- Setting parameters
- Cross-validation for experimental validation of performance
- **C4.5:** extension of ID3 accounting for unavailable values, continuous attribute value ranges, pruning of decision trees, rule derivation, etc.

Real-valued data?

- Many ML systems work only on **nominal data**
- We often classify data into one of 4 basic types:
  - **Nominal data** is named, e.g., representing restaurant type as Thai, French, Italian, Burger
  - **Ordinal data** has a well-defined sequence: small, medium, large
  - **Discrete data** is easily represented by integers
  - **Continuous data** is captured by **real** numbers
- There are others, like intervals: age 0-3, 3-5, …
- Handling some types may need new techniques

Techniques for real-valued data

For ML systems that expect nominal 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 ☹?

ML systems must deal with *noise* in training data

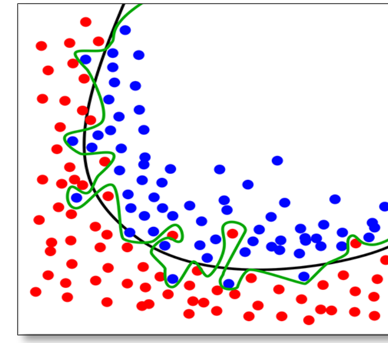
- Two examples have same attribute/value pairs, but different classifications
- Some attribute values wrong due to errors in the data acquisition or preprocessing phase
- Classification is wrong (e.g., + instead of -) because of some error
- Some attributes irrelevant to decision-making, e.g., color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

Bias in the training data is a related problem

Bias: If it's cloudy, it's a tank

- You may hear about a ML system designed to classify images into those with & without tanks
  - It was trained on N images with tanks and M images with no tanks
  - But the positive examples were all taken on a cloudy day; the negative on a sunny one
- System worked well, but had learned to detect the weather ☹
- The story is too good to be true; see Neural Net Tank Urban Legend
- But avoiding bias when training an AI or ML system is a real problem!

Overfitting ☹



- *Overfitting* occurs when a statistical model describes random error or noise instead of underlying relationship
- If hypothesis space has many dimensions (many attributes) we may find **meaningless regularity** in data irrelevant to true distinguishing features

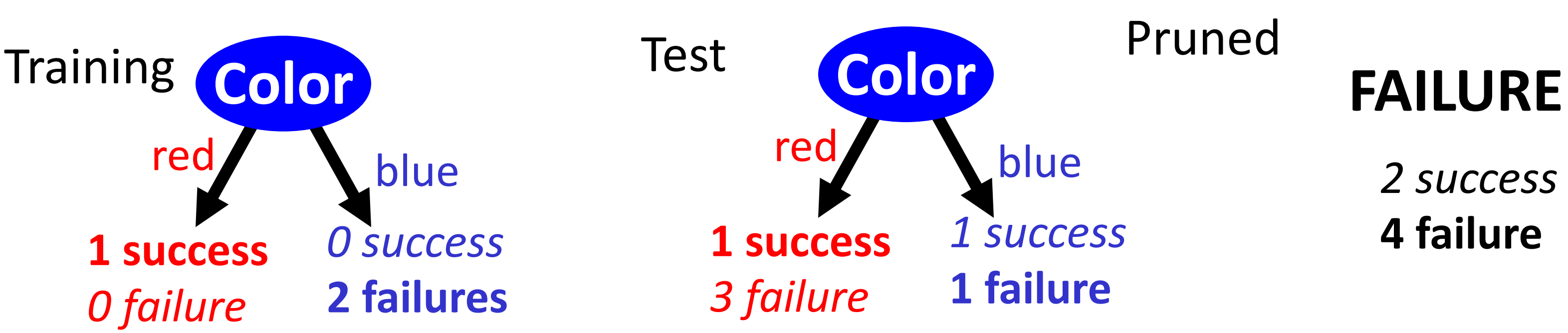
  Students with an *m* in first name, born in July, & whose SSN digits sum to a prime number get better grades in AI
- If we have **too little training data**, even a reasonable hypothesis space can overfit

Avoiding Overfitting

- Remove obviously irrelevant features
  - E.g., remove 'year observed', 'month observed', 'day observed', 'observer name' from the attributes used
- Get more training data
- Pruning lower nodes in a 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 4



Converting decision trees to rules

- Easy to get rules from decision tree: write rule for each path from the root to leaf
- Rule’s left-hand side built from the label of the nodes & labels of arcs
- Resulting rules set can be simplified:
  - Let LHS be the rule’s left hand side (condition part)
  - 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 easy** to implement
  - **Simple model:** translate to a set of rules
  - **Useful: e**mpirically valid in many commercial products
  - **Robust:** handles noisy data
  - **Explainable:** easy for people to understand
- Weaknesses
  - Large decision trees may be hard to understand
  - Requires fixed-length feature vectors
  - Non-incremental, adding one new feature requires rebuilding entire tree