

Classification in Text Processing

Technique: Decision Tree Learning with ID3

Problem: Any classification problem

Presented by: Anna Rumshisky

Outline

- Classification vs. Clustering
- Classification problems in Text Processing
 - Training Corpus vs. Test Corpus
 - Labeled data
- Feature selection
- Decision Tree construction with ID3
 - Selecting “best attribute”
 - Information Gain
- What is Entropy?

Classification

Goal: Classify 'objects' from a universe, i.e. identify it as a member of one of the pre-set *categories* or *classes*

- assign a label to each object ...
- "target attribute" of each instance

Examples:

Problem

POS Tagging

Sense Disambiguation

Information retrieval

Author identification

Spam filtering

Text categorization

Object

Word

Word

Document

Document

Document

Document

Categories

POS

The word's senses

Relevant/Irrelevant

Authors

Spam/Not Spam

Topics

Word Sense Disambiguation

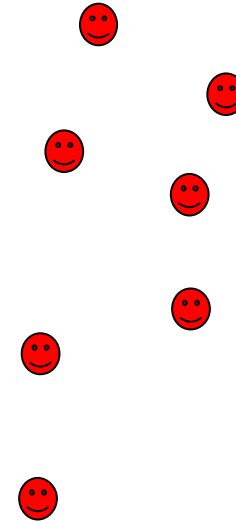
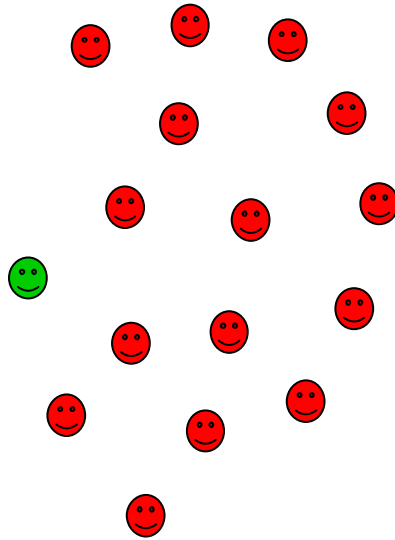
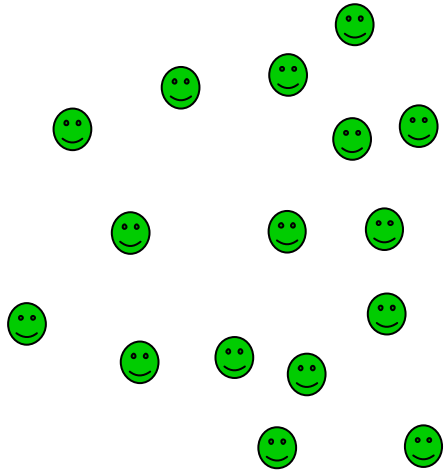
e.g. **Interest** (noun)


- Curiosity
- Percentage of borrowed amount
- Advantage
- ...


Classification vs. Clustering

- **Classification** assumes labeled data: we know how many classes there are and we have examples for each class (labeled data).
- Classification is supervised
- In **Clustering** we don't have labeled data; we just assume that there is a natural division in the data and we may not know how many divisions (clusters) there are
- Clustering is unsupervised

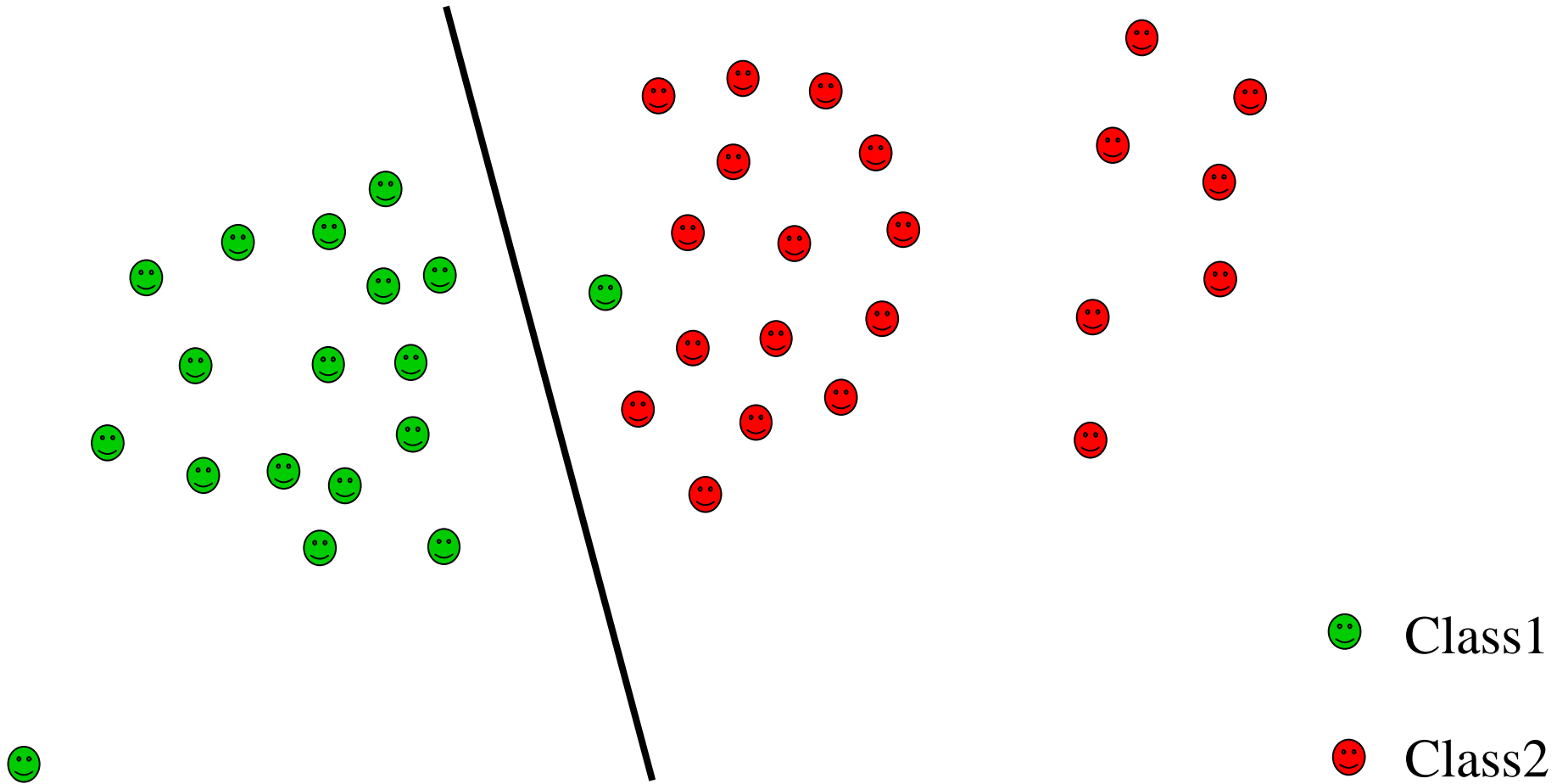
Classification



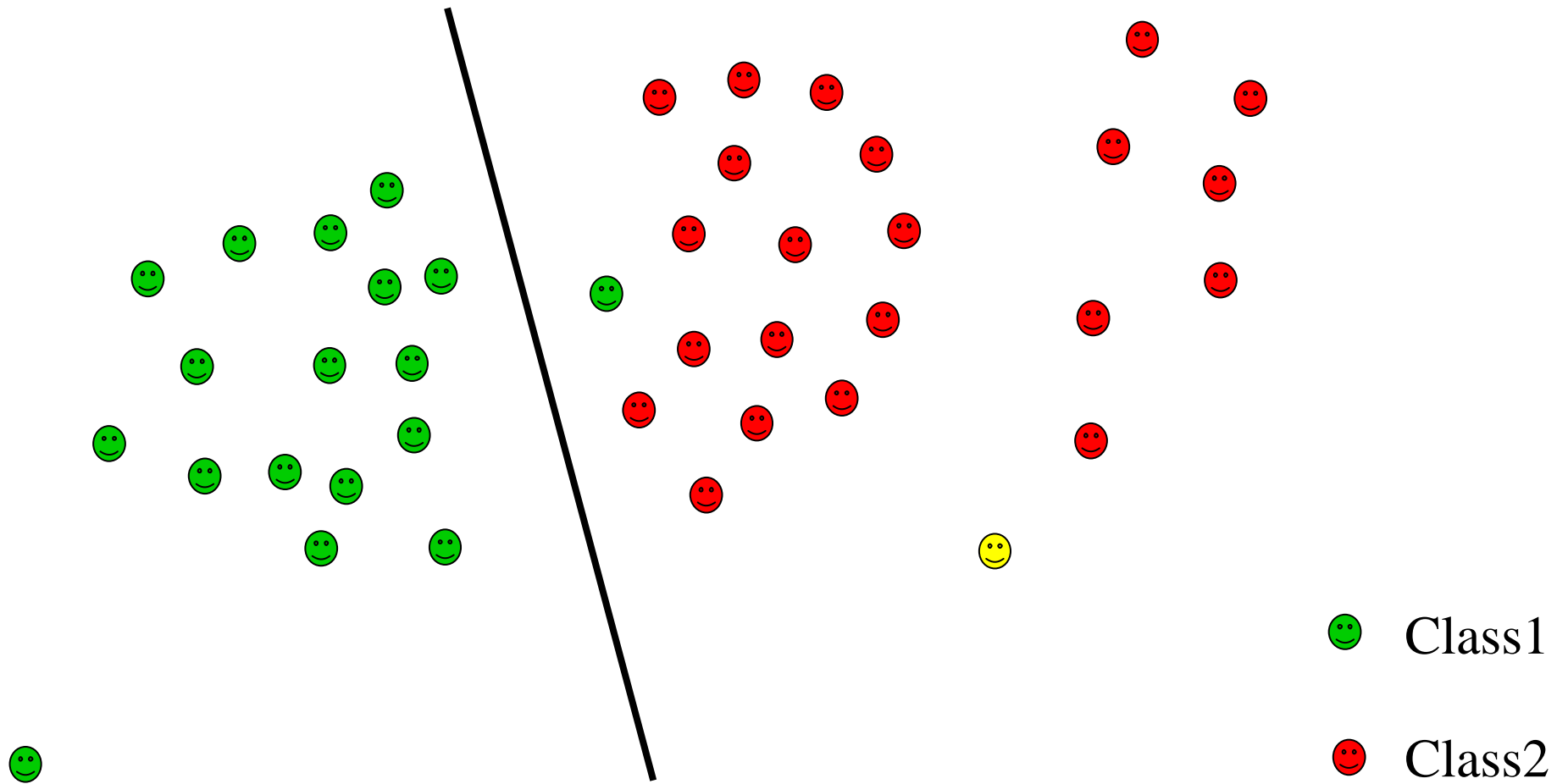
 Class1 = Sense 1

 Class2 = Sense 2

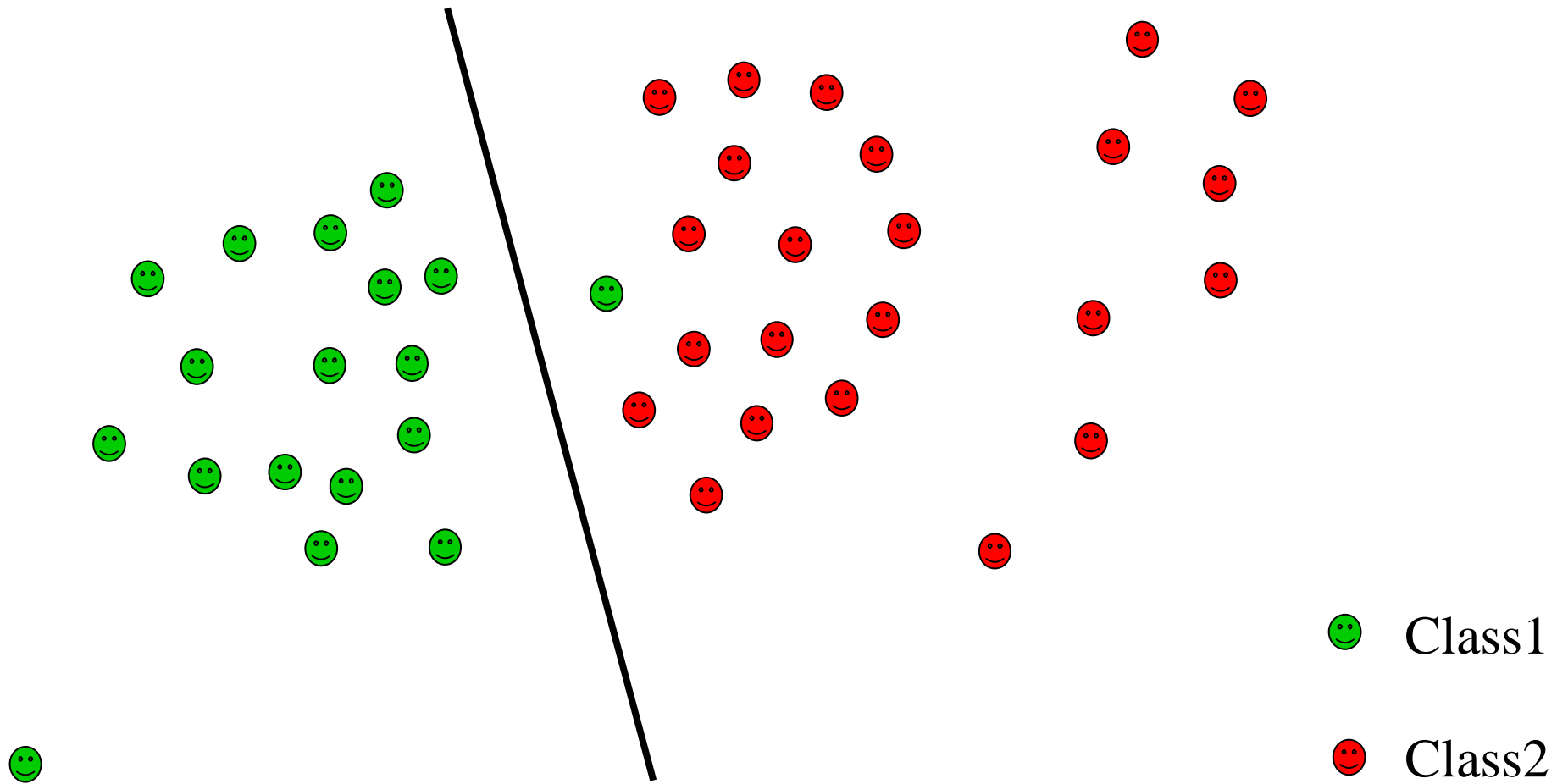
Classification



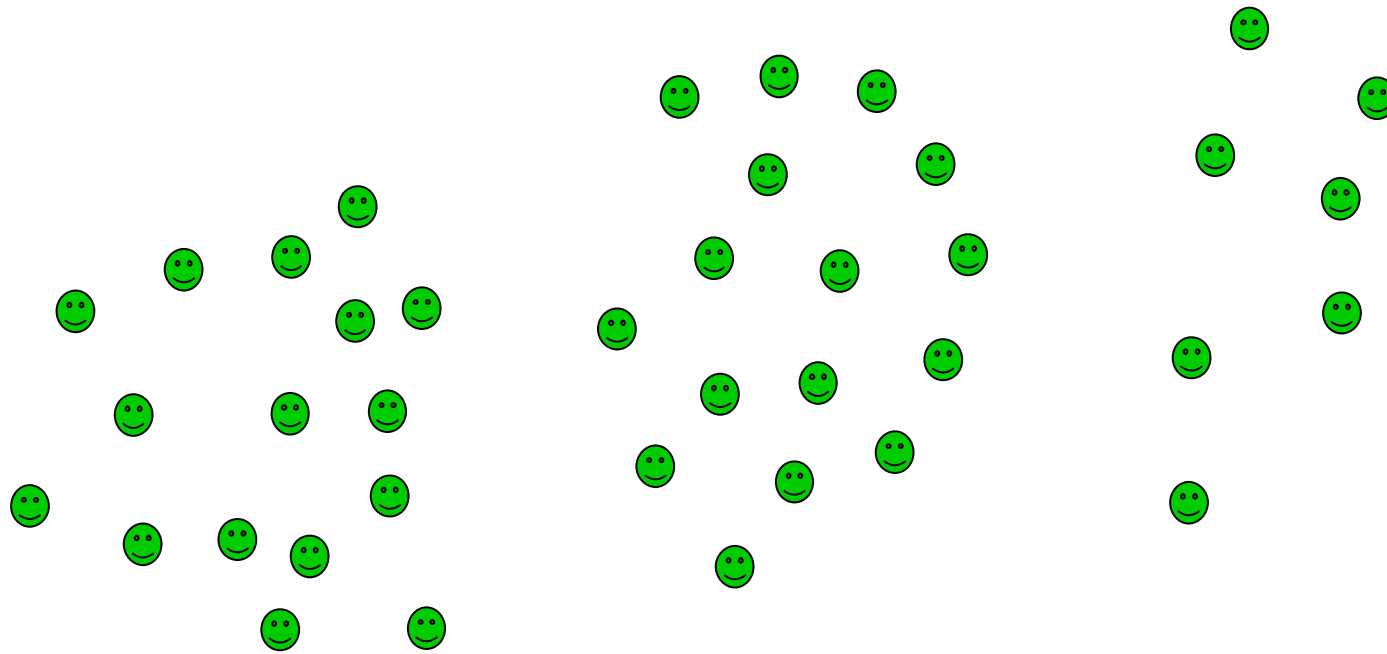
Classification



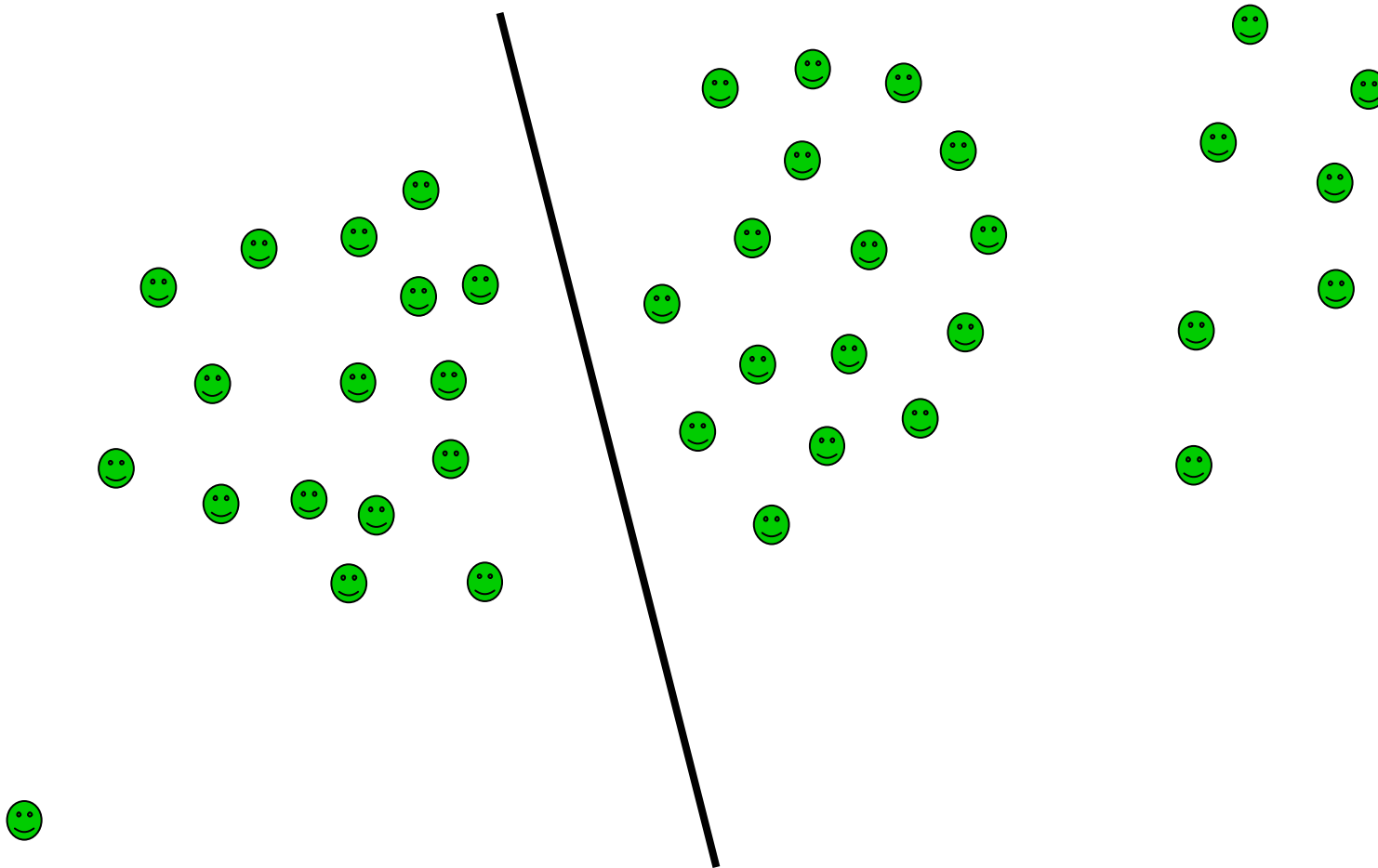
Classification



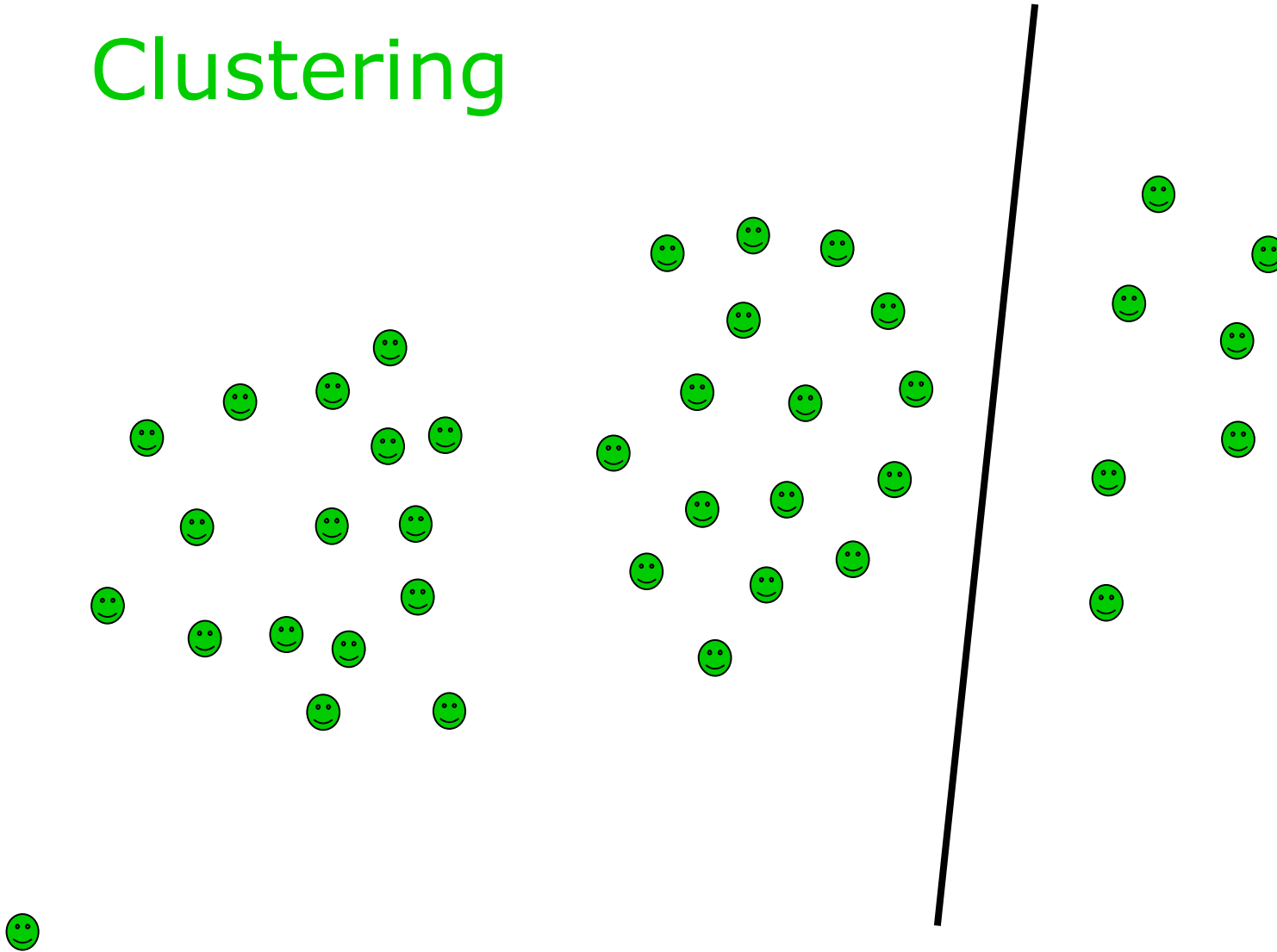
Clustering



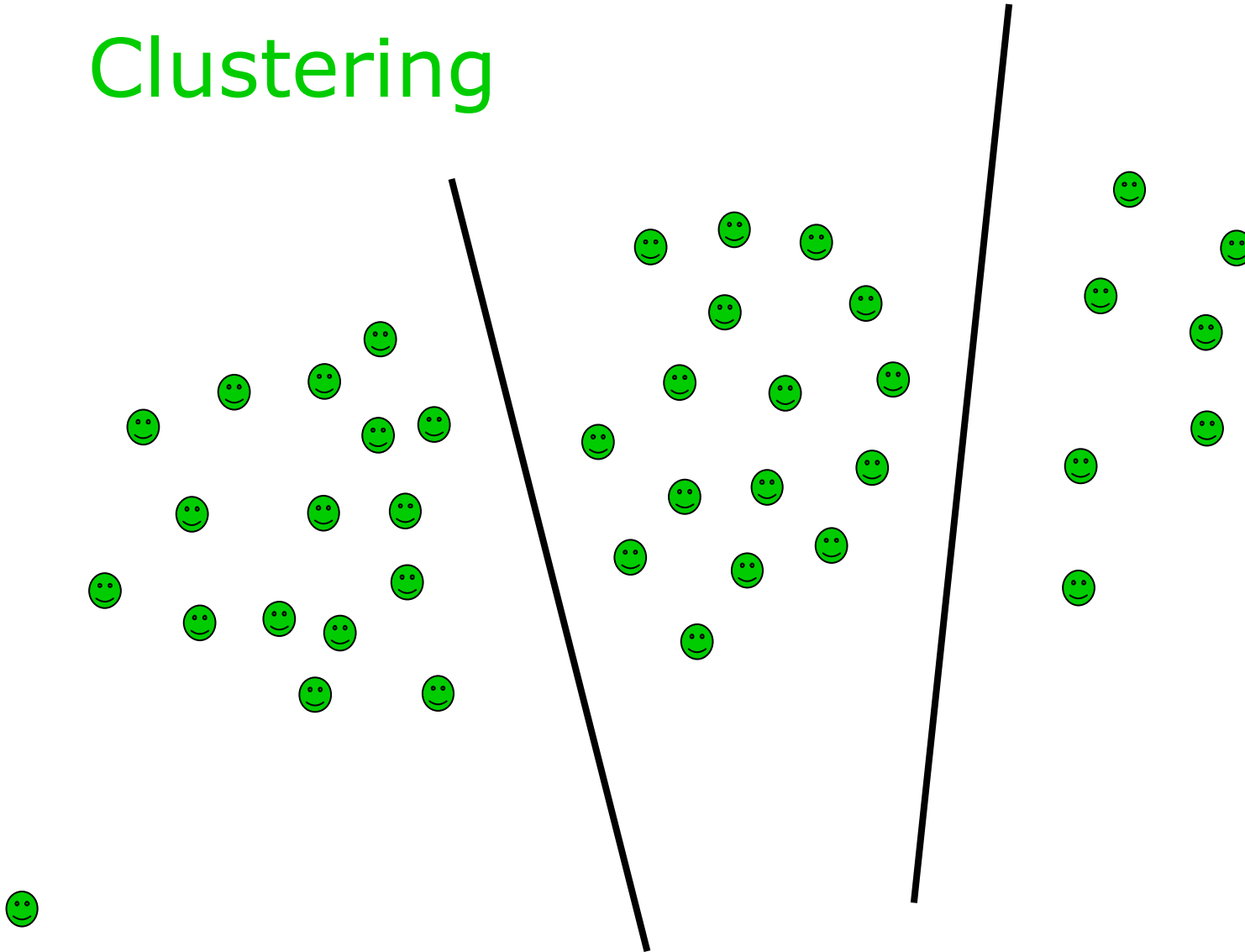
Clustering



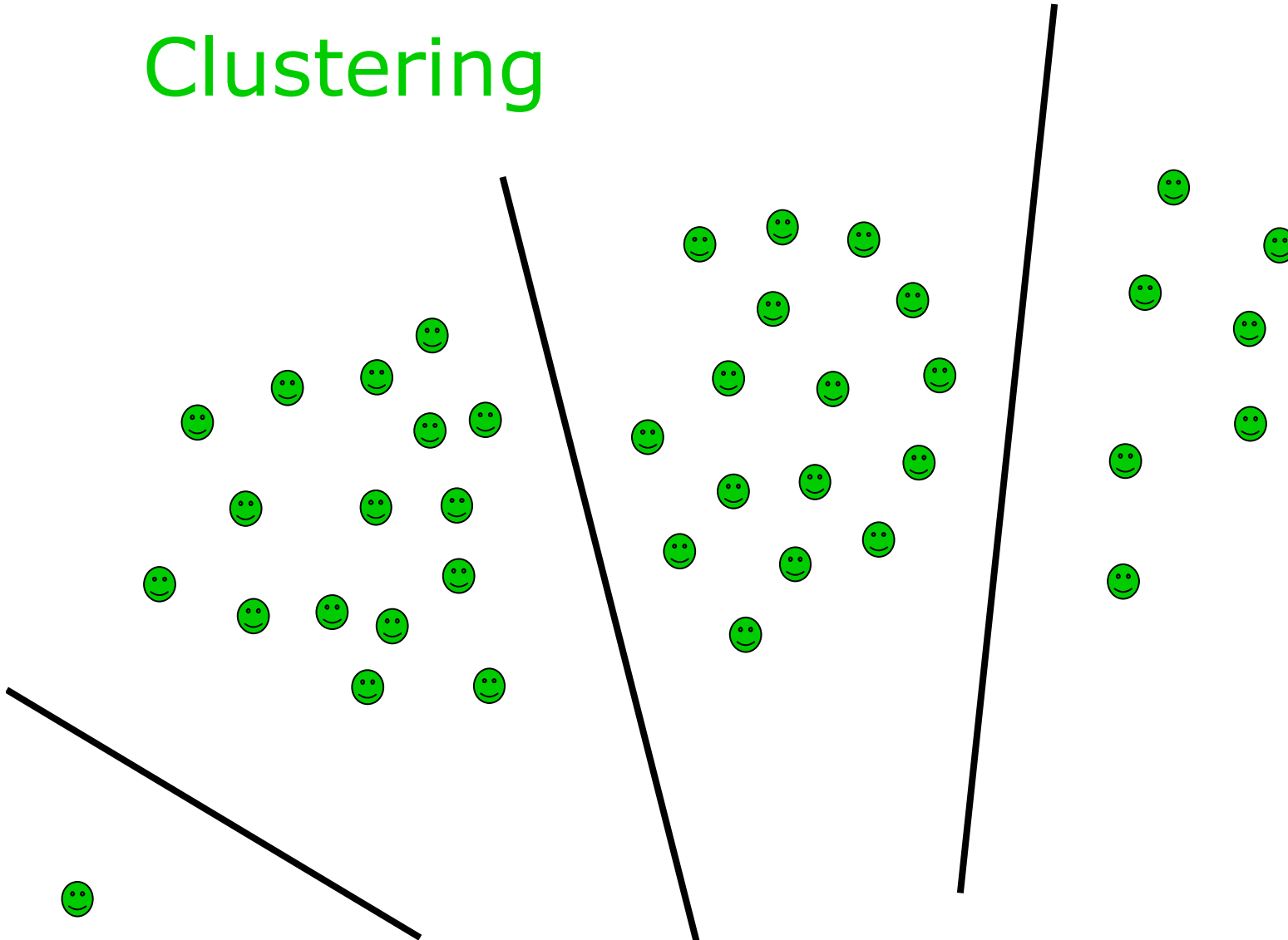
Clustering



Clustering



Clustering



Training, Testing, and Using a Classifier

- Create a **training corpus** (manual tagging)
- Create a **test corpus** (manual tagging)
- Train the model on the training corpus
- Test its performance on the test corpus
- If satisfactory, use the model to process unseen text

Features: parameters used to classify, i.e. to assign target attribute value

- Features: each instance to be classified has parameters
- **Spam Filtering**: spam vs. not spam
- **Features**:
 - Words present in text : viagra, profits... (boolean)
 - Frequency of word: viagra(4), profits(2)... (integer)
 - Subject Capitalization? Yes/no
 - ...
- **Target attribute**: number of values?
- **Features** for your task?

Classification

- Define classes
- Label text
- Extract Features
 - each instance represented as a vector (multi-valued tuple)
 <f1, f2, f3, f4>: <10, 2, 1, 0>
- Choose a classifier
 - `>>> my_classifier.classify(token)`
 - The Naive Bayes Classifier
 - Decision trees
 - NN (perceptron)
 - SVM
- Train it (and test it)
- Use it to classify new examples

Evaluating classifiers

- Contingency table for the evaluation of a binary classifier

	GREEN is correct	RED is correct
GREEN was assigned	a	b
RED was assigned	c	d

- Accuracy = $(a+d)/(a+b+c+d)$
- Precision: $P_GREEN = a/(a+b)$, $P_RED = d/(c+d)$
- Recall: $R_GREEN = a/(a+c)$, $R_RED = d/(b+d)$

Training size

- The more the better! (usually)
- Results for text classification*

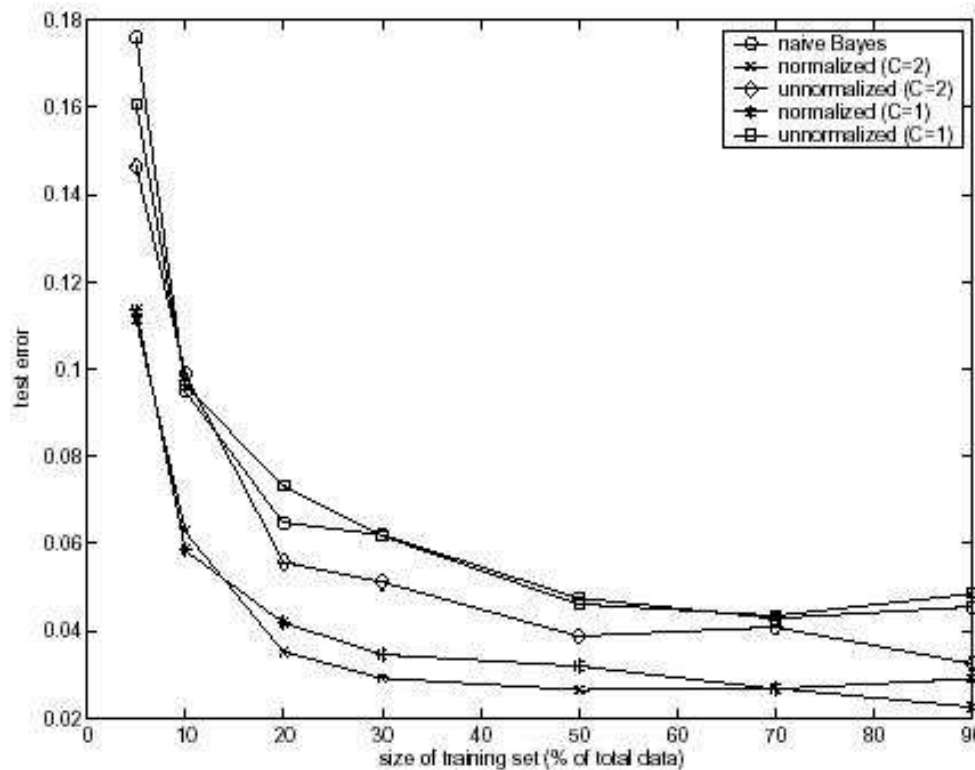


Figure 1: Test error vs training size on the newsgroups rec.sport.baseball and rec.sport.hockey

Decision Trees

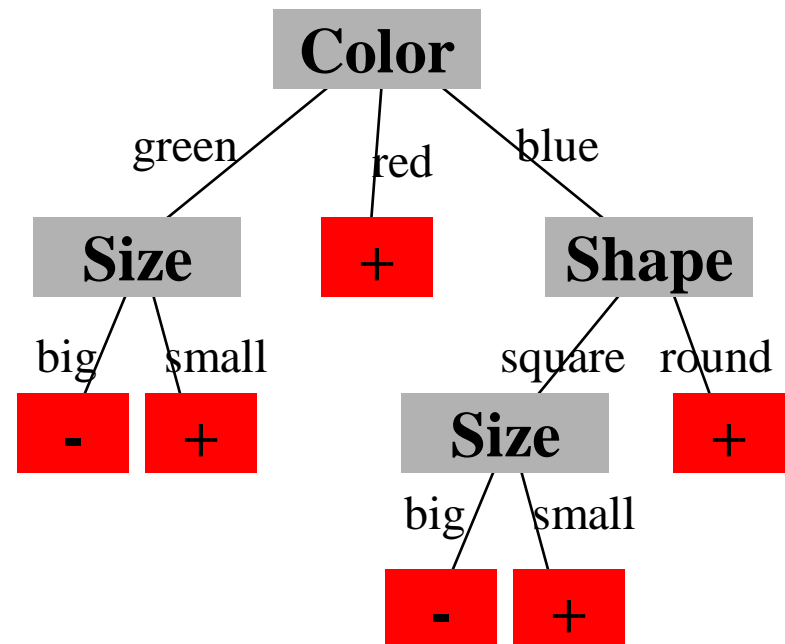
Decision Tree classifier and how
to build it with ID3

What you need to build a decision tree

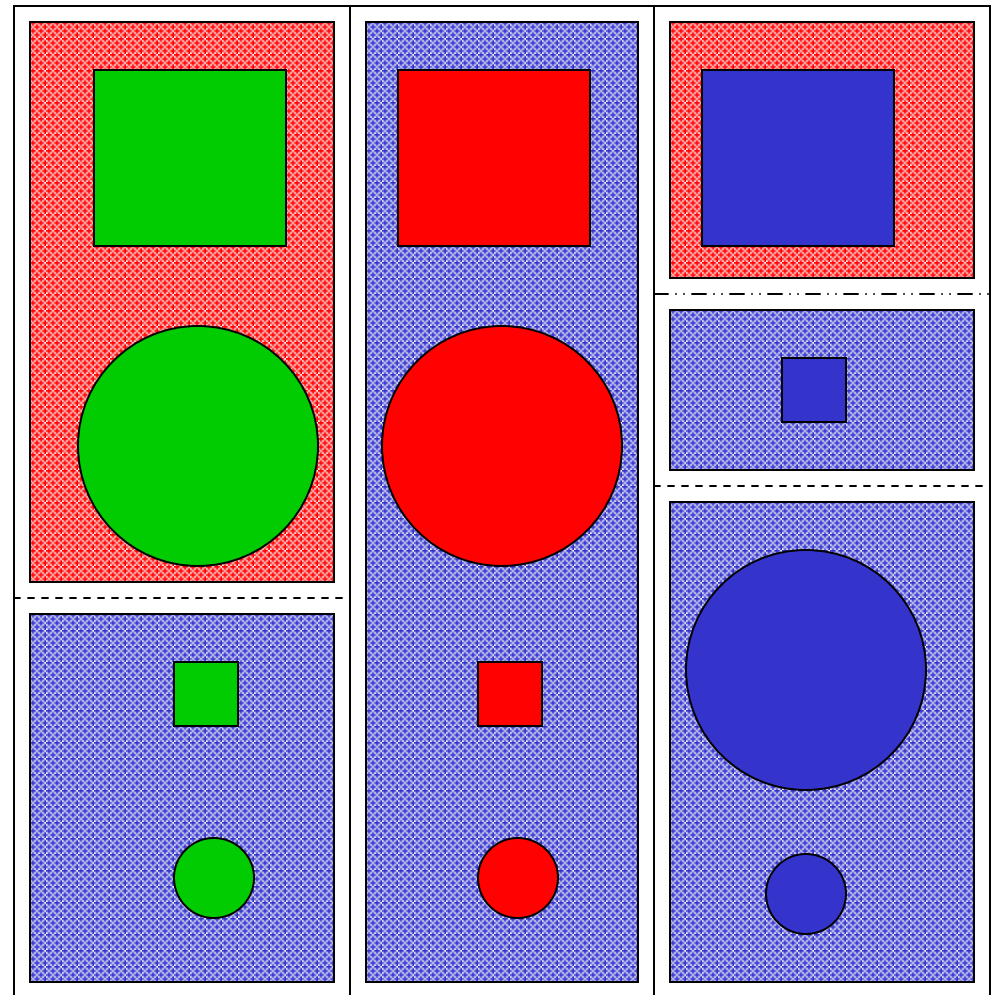
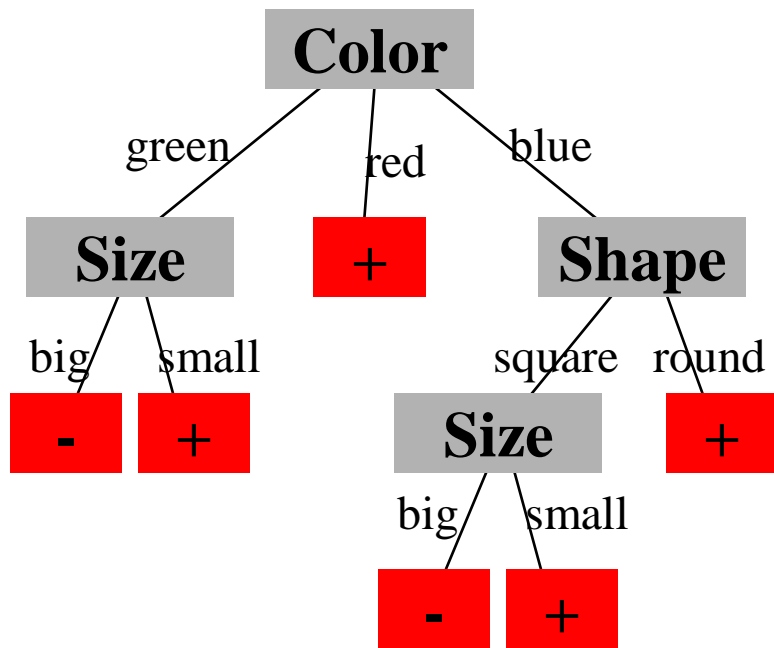
- Target attribute, with several values (= categories)
- Training data, manually labeled with those values
 - Sufficient number of instances
- Set of features associated with each instance

Learning decision trees

- Goal: Build a **decision tree** to classify the instances as positive or negative
- 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 target attribute value (+ or -)
 - a set of similarly labeled training instances
 - each arc has associated with it one of the possible values of the attribute at the node from which the arc is directed
 - at each node you have a decision attribute that splits the training set according to the values of that attribute
- Generalization: allow for >2 classes
 - e.g., {sell, hold, buy}



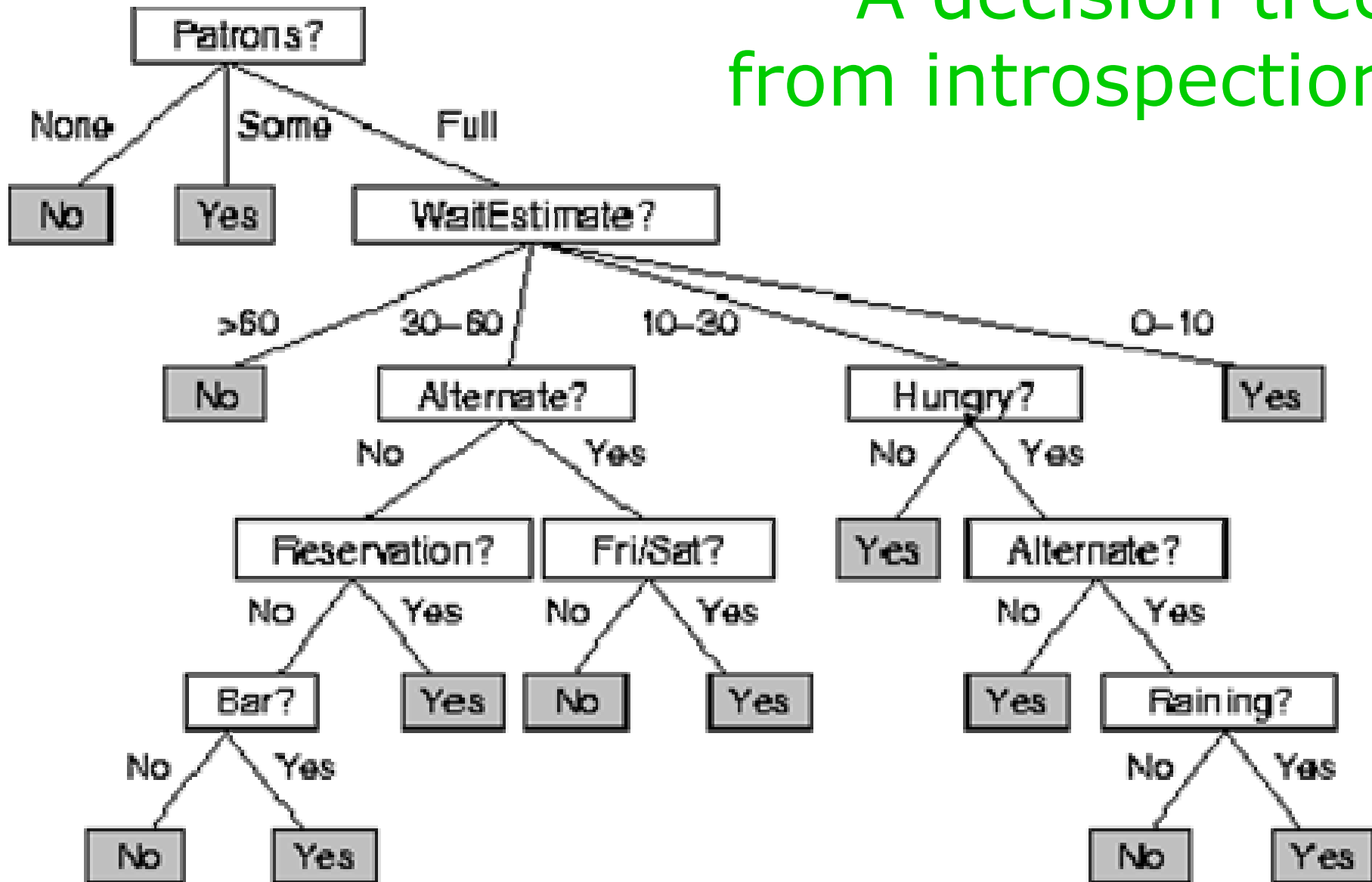
Decision tree-induced partition – example



Russel & Norvig's restaurant example

- 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



A training set

Example	Attributes										Goal
	Aff	Beef	Fish	Ham	Pork	Price	Rain	Res	Type	Est	Will Wait
X_1	Yes	No	No	Yes	Source	\$\$\$	No	Yes	French	0-10	Yes
X_2	Yes	No	No	Yes	Full	\$	No	No	Thai	30-60	No
X_3	No	Yes	No	No	Source	\$	No	No	Burger	0-10	Yes
X_4	Yes	No	Yes	Yes	Full	\$	No	No	Thai	10-30	Yes
X_5	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	No
X_6	No	Yes	No	Yes	Source	\$\$	Yes	Yes	Italian	0-10	Yes
X_7	No	Yes	No	No	None	\$	Yes	No	Burger	0-10	No
X_8	No	No	No	Yes	Source	\$\$	Yes	Yes	Thai	0-10	Yes
X_9	No	Yes	Yes	No	Full	\$	Yes	No	Burger	>60	No
X_{10}	Yes	Yes	Yes	Yes	Full	\$\$\$	No	Yes	Italian	10-30	No
X_{11}	No	No	No	No	None	\$	No	No	Thai	0-10	No
X_{12}	Yes	Yes	Yes	Yes	Full	\$	No	No	Burger	30-60	Yes

Constructing a Decision Tree given a training set

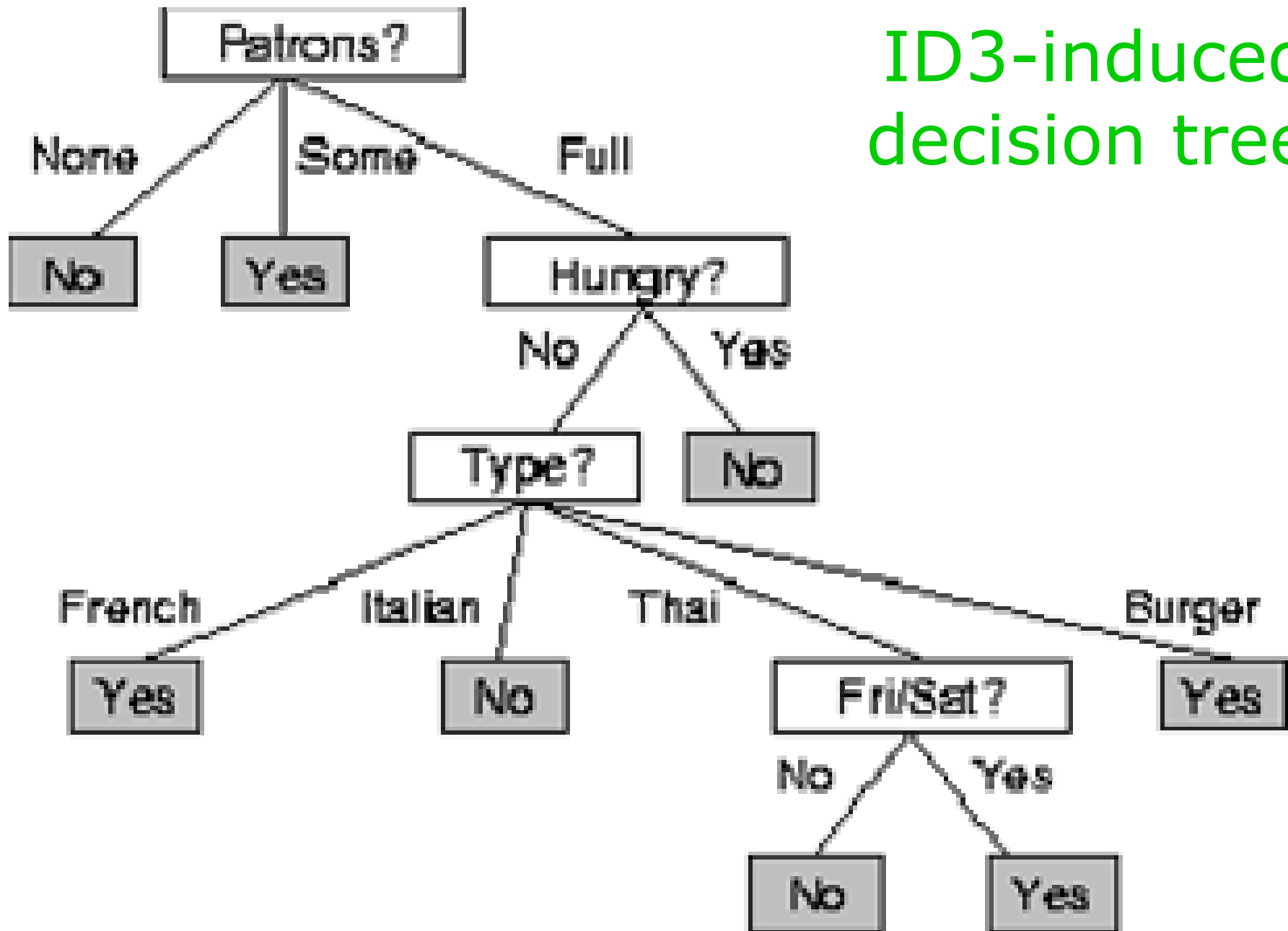
ID3

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

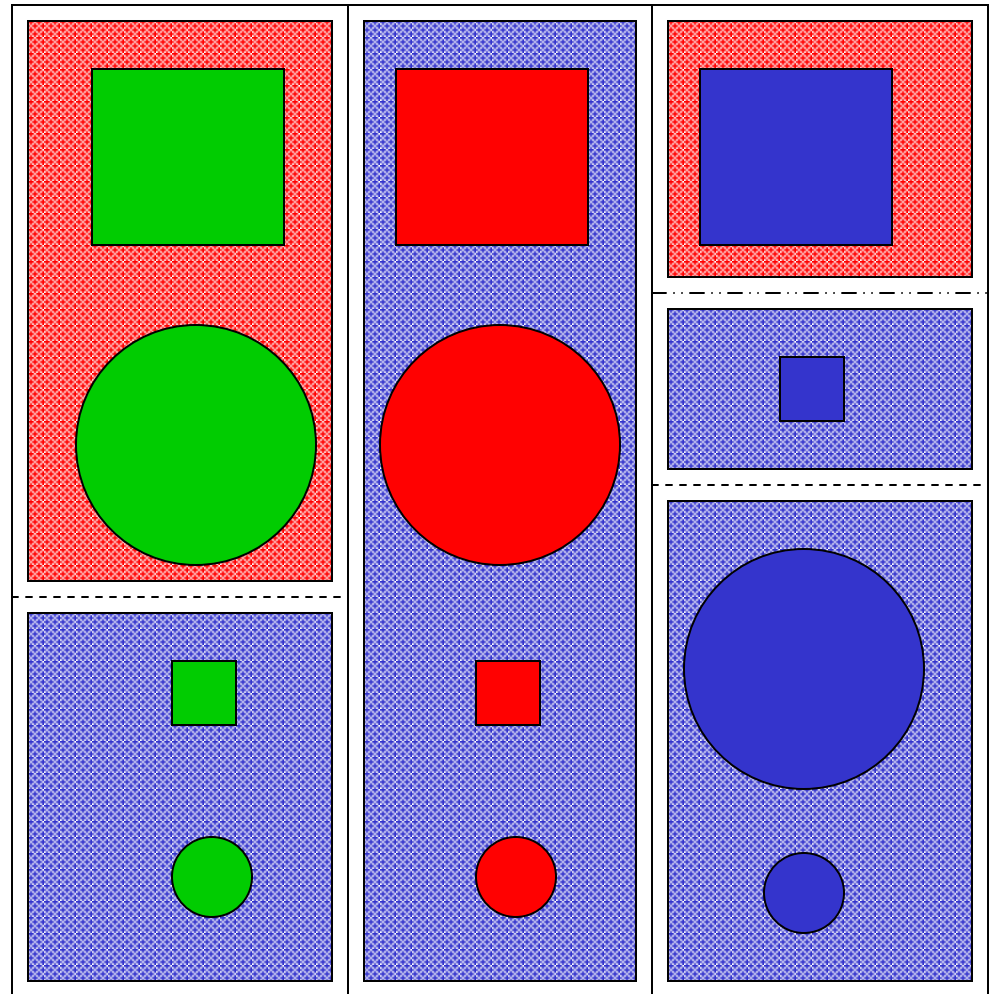
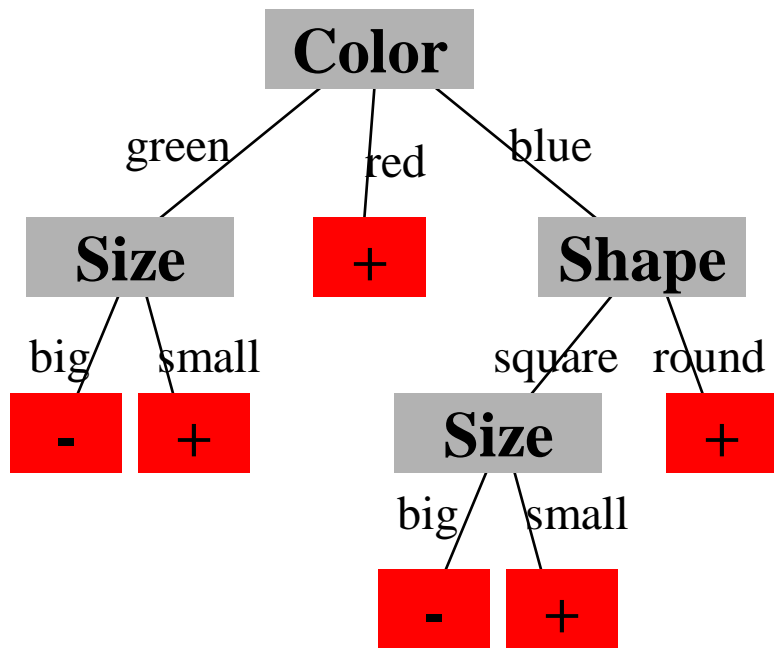
ID3-induced decision tree



Choosing the best decision attribute

- The decision attribute at each node should give the maximum information about the target attribute.
- How do we model that?
- Our training set gives us information about the probability distribution of the target attribute values.

Decision tree-induced partition – example



Entropy and the information gain

- Information contained in an outcome of a random variable is modeled as the number of bits necessary to encode that outcome.
- How much more do we know after a coin is tossed? We know 1 bit!

Number of bits necessary to encode N outcomes?

Consider a variable with n equiprobable outcomes:

	<u>Outcomes</u>	<u>Binary representation</u>	<u>Length in bits</u>
2^3 {	2^2 {	2^1 {	0 = 0 1
			1 = 1 1
	2 = 10 2		
	3 = 11 2		
	4 = 100 3		
	5 = 101 3		
	6 = 110 3		
	7 = 111 3		
	8 = 1000 4		
	9 = 1001 4		
...	

Entropy (Information theory)

- If there are n equally probable possible messages, then the probability p of each is $1/n$
- Information conveyed by a message is
$$-\log(p) = -\log(1/n) = -\log(n^{-1}) = \log(n)$$
- E.g., if there are 16 messages, then $\log(16) = 4$ and we need 4 bits to identify/send each message
- In general, if we are given a variable distributed according to a probability distribution
$$X \sim P = (p_1, p_2, \dots, p_n)$$
- Then **entropy of X** is:
$$H(X) = -(p_1 * \log(p_1) + p_2 * \log(p_2) + \dots + p_n * \log(p_n))$$

Entropy (Information theory) II

- Entropy of $X \sim P$:

$$H(X) = -(p_1 * \log(p_1) + p_2 * \log(p_2) + \dots + p_n * \log(p_n))$$

- Examples:

- If P is $(0.5, 0.5)$ then $H(X)$ is 1
- If P is $(0.67, 0.33)$ then $H(X)$ is 0.92
- If P is $(1, 0)$ then $H(X)$ is 0

Entropy as expected number of bits

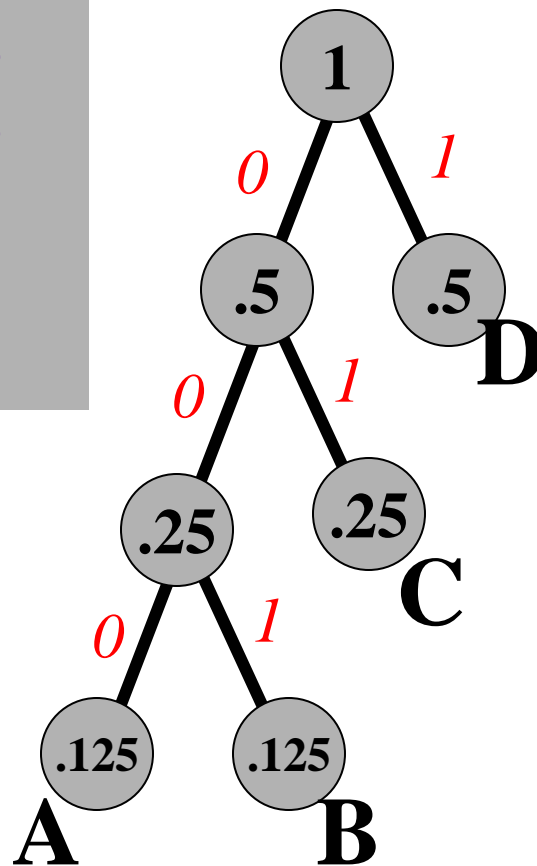
- When the outcomes of a random variable are equiprobable, its *entropy* is exactly equal to the number of bits necessary to encode the number of outcomes
- This is the idea behind the definition of **entropy** as **the number of bits, necessary, on the average, to encode an outcome of a random variable**
- What happens when the outcomes are not equiprobable?
- This is also the idea behind Huffman codes
 - Less frequent outcomes are encoded with longer sequences of bits

Huffman codes

- In 1952 MIT student David Huffman devised, in the course of doing a homework assignment, an elegant coding scheme 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 has the probability of all nodes beneath it
 - Trace a path to each leaf, noticing the direction at each node
- A Huffman code essentially builds a decision tree to identify the message.
- Easy to prove that Huffman code is optimal when all probabilities are integer powers of $1/2$.

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**

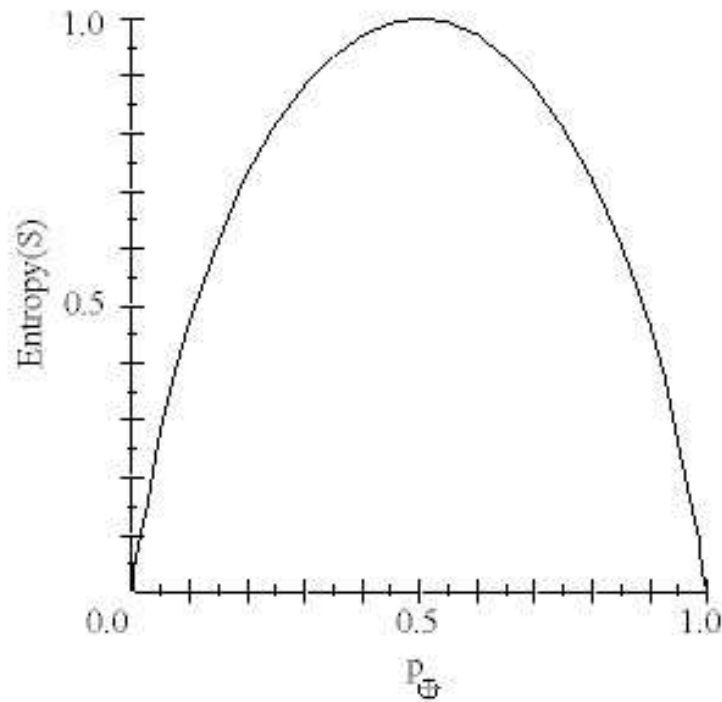
Entropy of $X \sim (p_1, p_2)$

Information theory: optimal length code assigns
 $-\log_2 p$ bits to message having probability p .

So, expected number of bits to encode \oplus or \ominus of
random member of S :

$$p_{\oplus}(-\log_2 p_{\oplus}) + p_{\ominus}(-\log_2 p_{\ominus})$$

Entropy in Bernoulli trials



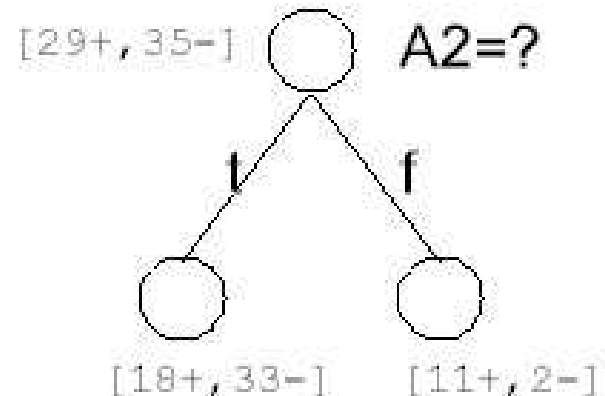
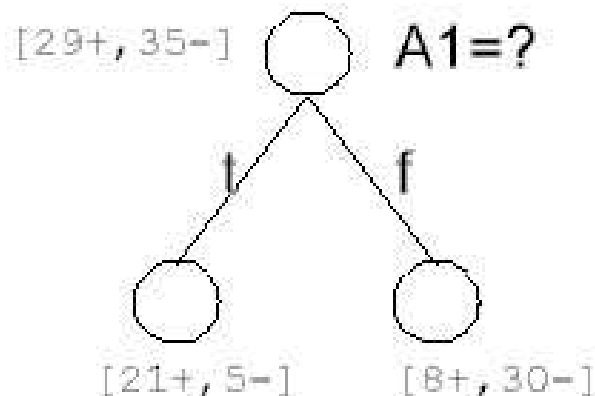
- S is a sample of training examples
- p_{\oplus} is the proportion of positive examples in S
- p_{\ominus} is the proportion of negative examples in S
- Entropy measures the impurity of S

$$\text{Entropy}(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

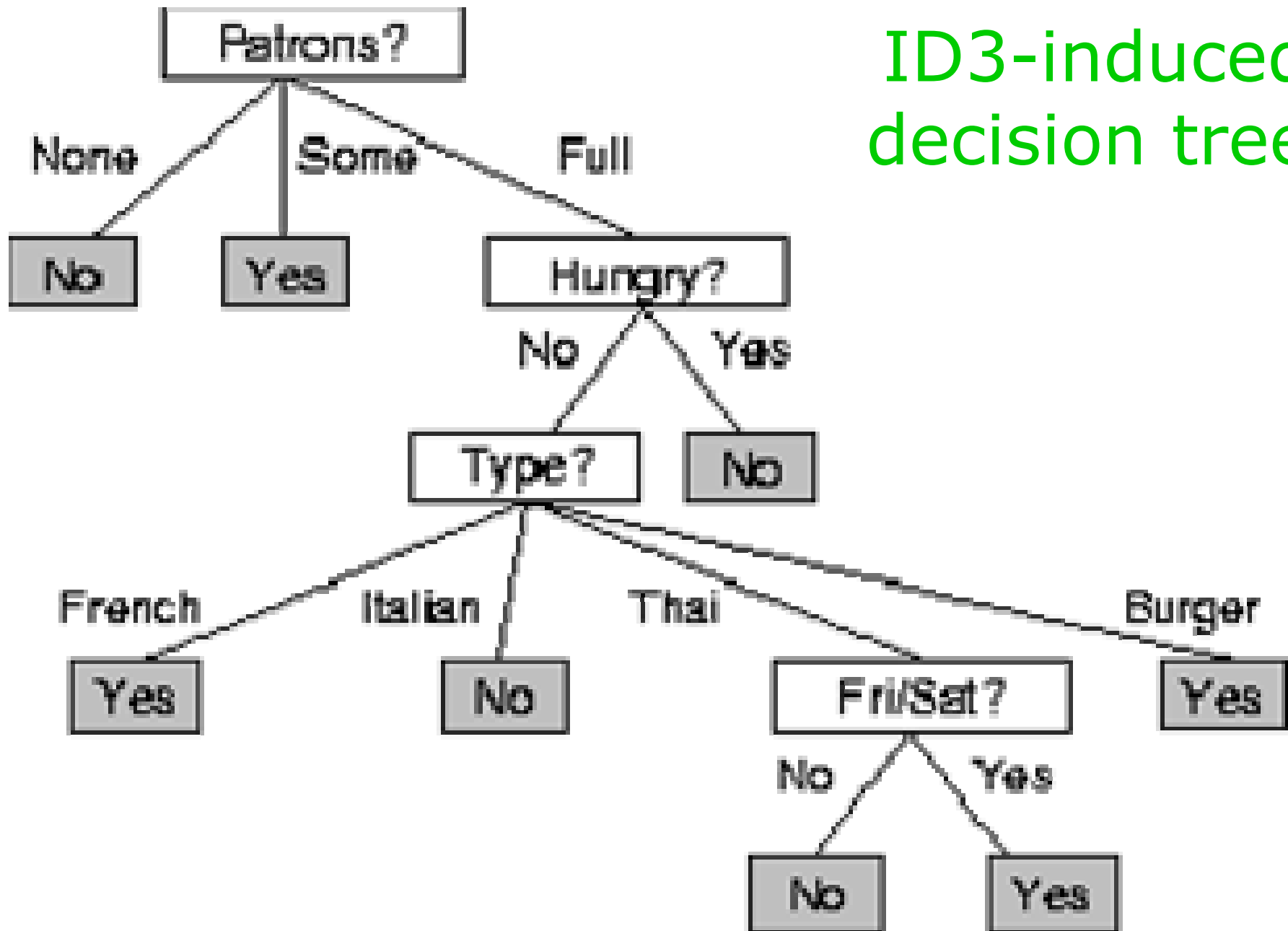
Information Gain

$Gain(S, A) =$ expected reduction in entropy due to sorting on A

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$



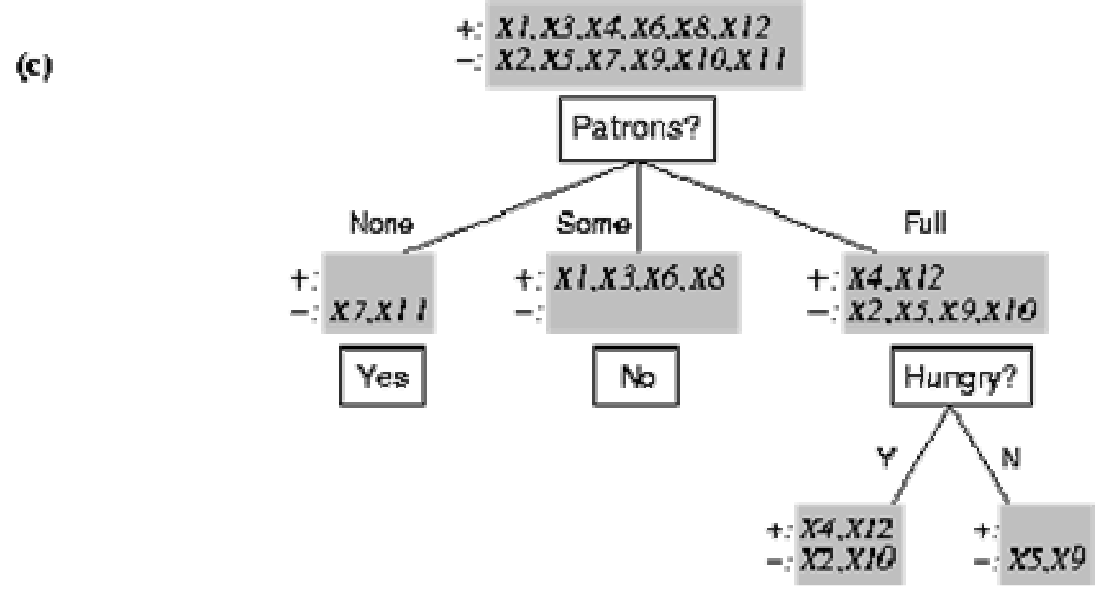
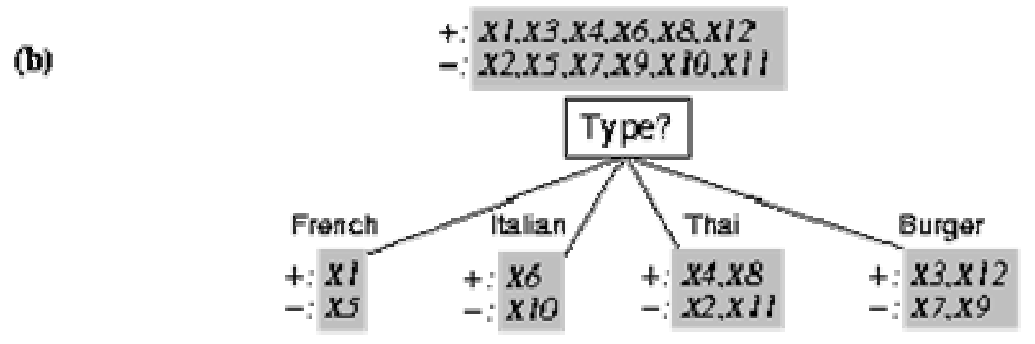
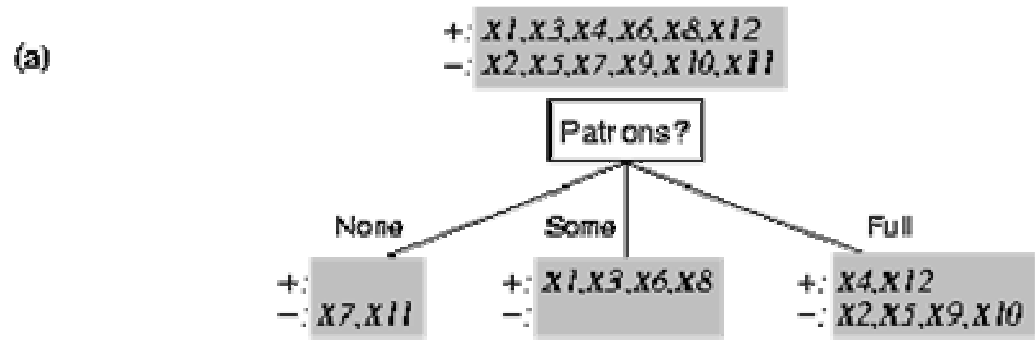
ID3-induced decision tree



Maximum Information Gain's Use of Entropy

- We choose the *decision attribute* at each node so as to maximize the “information gain” at each node of the tree.
- That is, choose the decision attribute so as to minimize the total weighted sum of entropies of **the target attribute distribution** at *each child node*.
- In the best case, the *decision attribute* splits the data exactly as the *target attribute* does, so that the target attribute entropy at each node after the split is 0.
- In the worst case, the decision attribute does not reduce the entropy of the *target attribute* at all.

Maximum information gain:
 “Patrons?”
 or “Type?”



Computing the information gain

Decision attribute: "Patrons?"

- Target attribute entropy before the split

$$\{+: 6, -: 6\} \quad X \sim (1/2, 1/2)$$

$$H(X) = -1/2 \lg 1/2 - 1/2 \lg 1/2 = -1 * \lg 1/2 = -1 * -1 = 1$$

(one bit required to encode the outcome)

- Target attribute entropy after the split

None {+:0, -:2}, **Some** {+:4, -:0}, **Full** {+:2, -:4}

$$X \sim (0, 1)$$

$$X \sim (1, 0)$$

$$X \sim (1/3, 2/3)$$

$$H(X_{\text{None}}) = 0$$

$$H(X_{\text{Some}}) = 0$$

$$H(X_{\text{Full}}) = -1/3 \lg 1/3 - 2/3 \lg 2/3 \approx .39 + .53 = .92$$

- Information Gain

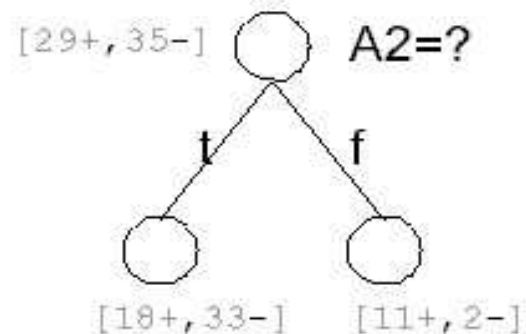
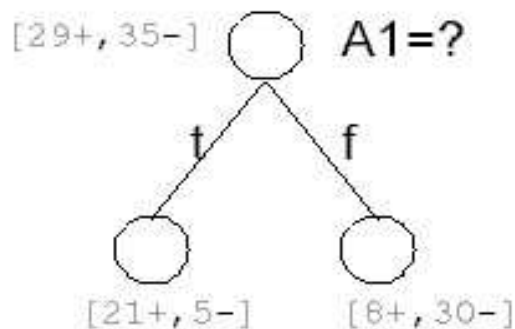
$$H(X) - (H(X_{\text{None}}) * 2/12 + H(X_{\text{Some}}) * 4/12 + H(X_{\text{Full}}) * 6/12) \\ = 1 - 0.46 = .54$$

Top-Down Induction of Decision Trees

Main loop:

1. $A \leftarrow$ the “best” decision attribute for next *node*
2. Assign A as decision attribute for *node*
3. For each value of A , create new descendant of *node*
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

Which attribute is best?



Which attribute is better?

```
>>> log(35.0/64,2)
-0.87071698305503364
>>> 35.0/64 * log(35.0/64,2)
-0.47617335010822154
>>> 35.0/64 * log(35.0/64,2) + 29.0/64 * log(29.0/64,2)
-0.99365071169104047
>>> - (35.0/64 * log(35.0/64,2) + 29.0/64 * log(29.0/64,2) )
0.99365071169104047
>>> 21.0/26 * log(21.0/26,2) + 5.0/26 * log(5.0/26,2)
-0.70627408918760071
>>> - (21.0/26 * log(21.0/26,2) + 5.0/26 * log(5.0/26,2)) * 26/64.0
0.28692384873246279
>>> - (8.0/38 * log(8.0/38,2) + 30.0/38 * log(30/38.0,2)) * 38/64.0
0.44085199441563588
>>> - (21.0/26 * log(21.0/26,2) + 5.0/26 * log(5.0/26,2)) * 26/64.0 - (8.0/38 * log(8.0/38,2) + 30.0/38 *
    log(30/38.0,2)) * 38/64.0
0.72777584314809873
>>> - (18.0/51 * log(18.0/51,2) + 33.0/51 * log(33.0/51,2)) * 51/64.0 - (11.0/13 * log(11.0/13,2) +
    2.0/13 * log(2/13.0,2)) * 13/64.0
0.87221882822780661
```

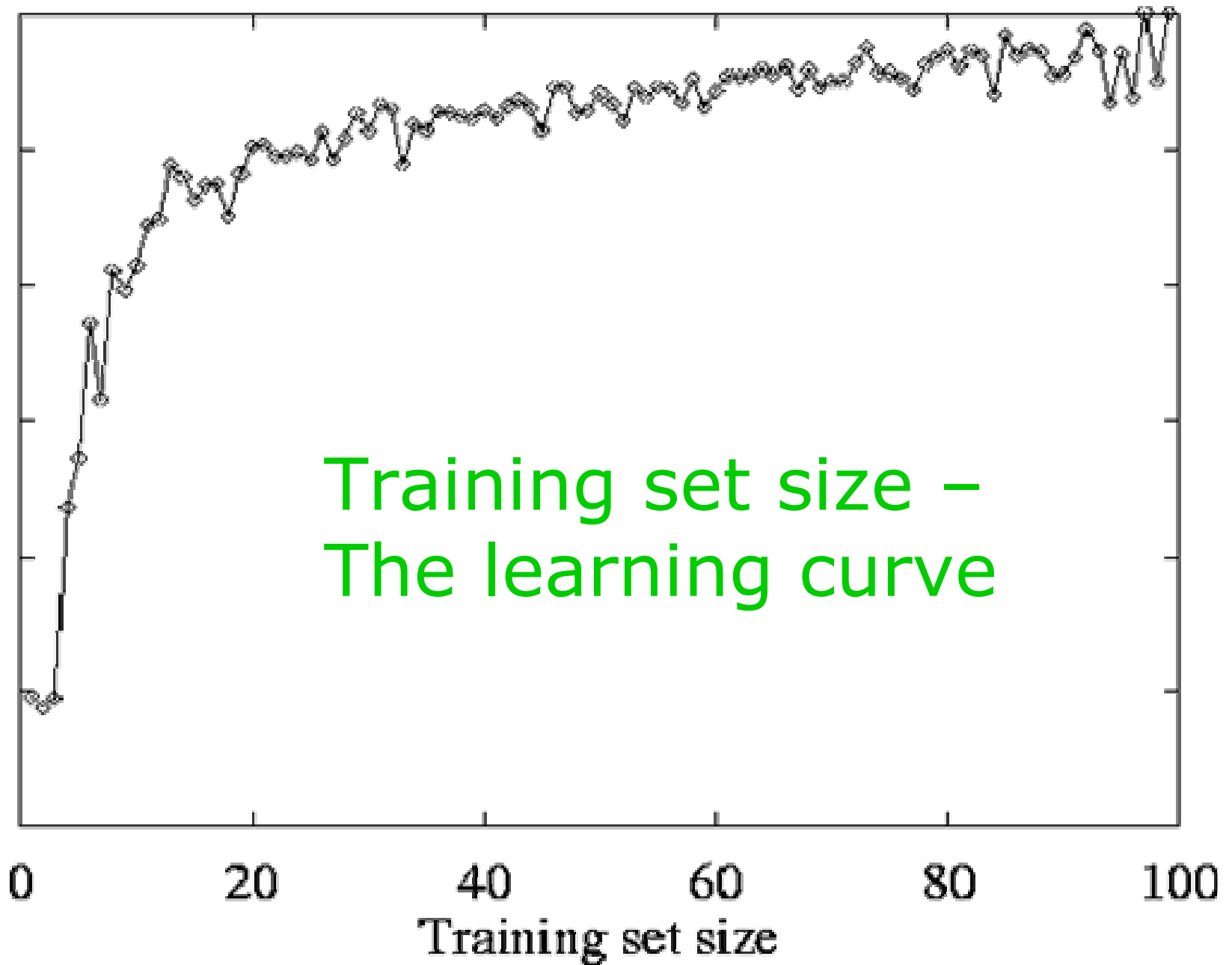
Decision Tree Algorithm

Remarks

- Preference for short trees and for those with high information gain near the root
- The algorithm is greedy
- Local maxima are possible

Other considerations

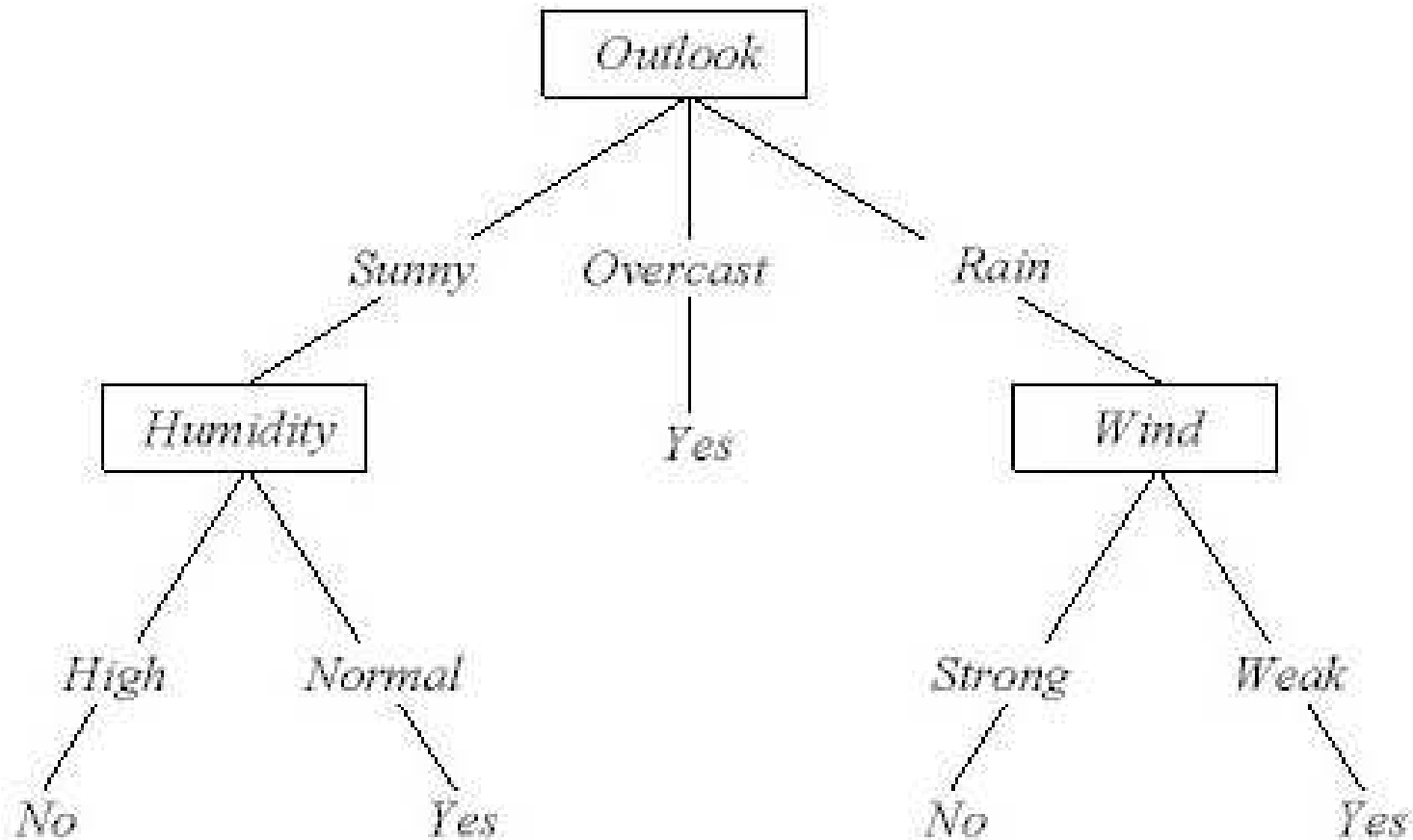
- Effects of training set size
- Converting a decision tree to a set of rules
- Overfitting the data and tree pruning



Converting decision trees to rules

- 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

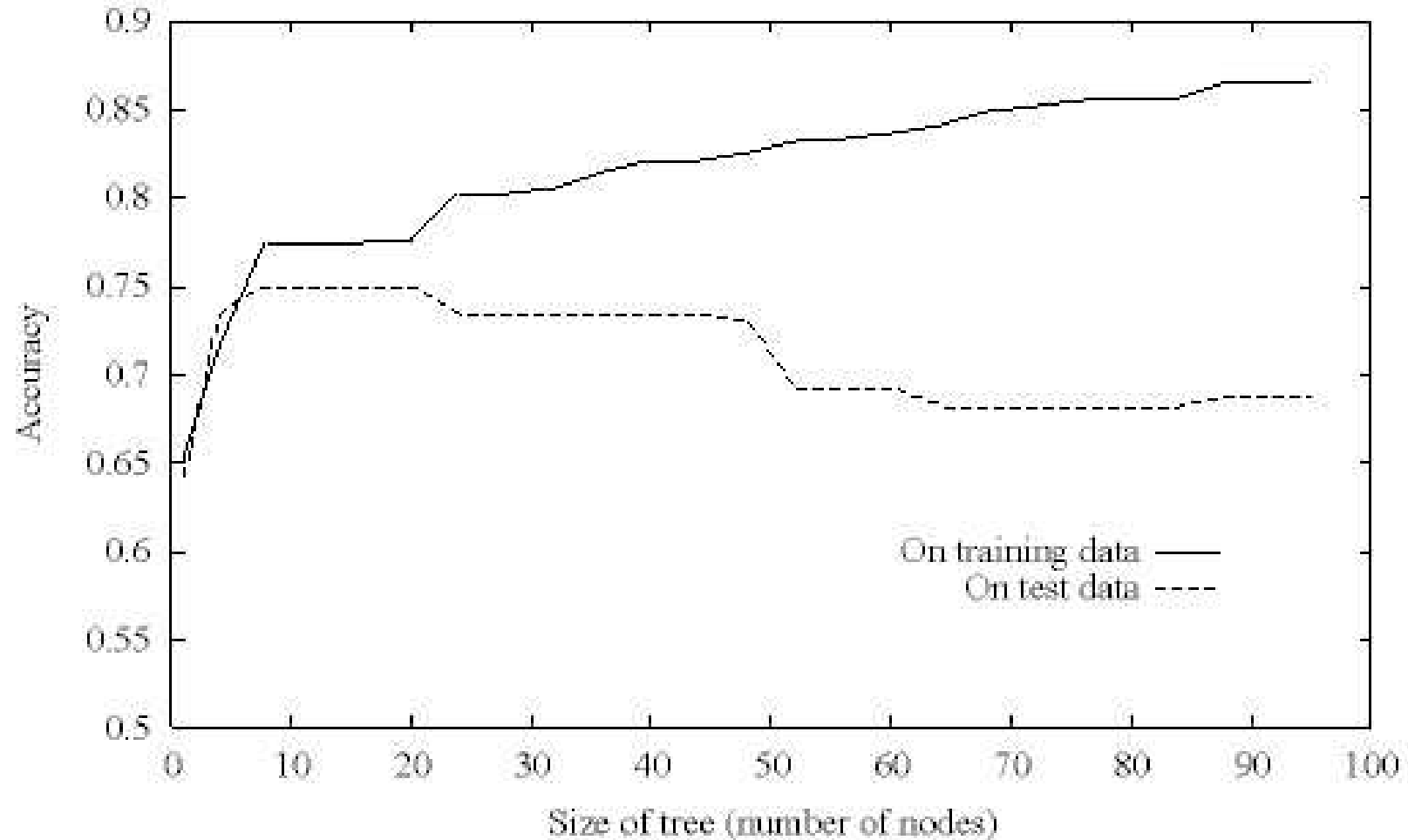
Converting a Tree to Rules



IF $(Outlook = Sunny) \wedge (Humidity = High)$
THEN $PlayTennis = No$

IF $(Outlook = Sunny) \wedge (Humidity = Normal)$
THEN $PlayTennis = Yes$

Overfitting when Decision Tree grows too large



Too many nodes, decision path too detailed

Overfitting in Decision Trees

Consider error of hypothesis h over

- training data: $error_{train}(h)$
- entire distribution \mathcal{D} of data: $error_{\mathcal{D}}(h)$

Hypothesis $h \in H$ **overfits** training data if there is an alternative hypothesis $h' \in H$ such that

$$error_{train}(h) < error_{train}(h')$$

and

$$error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$

Overfitting the training data

- Algorithm finds *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 *information gain* of the best attribute at a node is below a threshold, stop and make this node a leaf rather than generating children nodes.

How can we avoid overfitting?

- stop growing when data split not statistically significant
- grow full tree, then post-prune

How to select “best” tree:

- Measure performance over training data
- Measure performance over separate validation data set
- MDL: minimize (Minimum Description Length)
 $size(tree) + size(misclassifications(tree))$

Reduced-Error Pruning

Split data into *training* and *validation* set

Do until further pruning is harmful:

1. Evaluate impact on *validation* set of pruning each possible node (plus those below it)
 2. Greedily remove the one that most improves *validation* set accuracy
- produces smallest version of most accurate subtree
 - What if data is limited?

Rule Post-Pruning

1. Convert tree to equivalent set of rules
2. Prune each rule independently of others
3. Sort final rules into desired sequence for use

Perhaps most frequently used method (e.g., C4.5)

Further remarks

- Real-valued data
 - Select a set of thresholds defining intervals
 - Each interval becomes a discrete value of the attribute
- 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