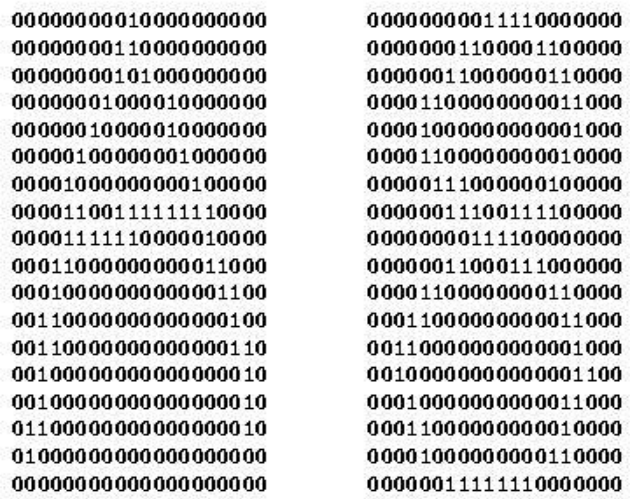
Supervised Learning (contd)
Linear Separation

Mausam
(based on slides by UW-AI faculty)

Images as Vectors

Binary handwritten characters



Greyscale images

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

.....

Treat an image as a high-dimensional vector
(e.g., by reading pixel values left to right, top to bottom row)

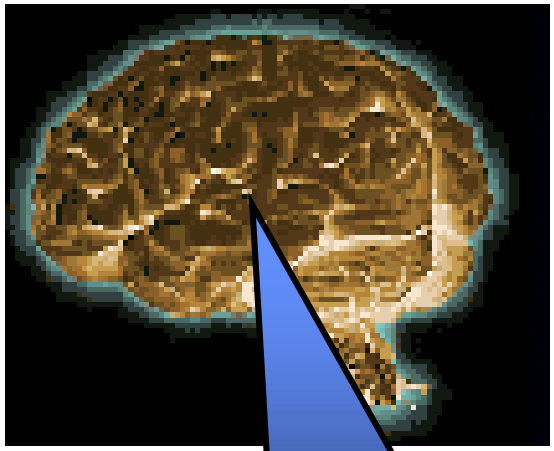
$$\mathbf{I} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{N-2} \\ p_N \end{bmatrix}$$

Pixel value p_i can be 0 or 1 (binary image) or 0 to 255 (greyscale)

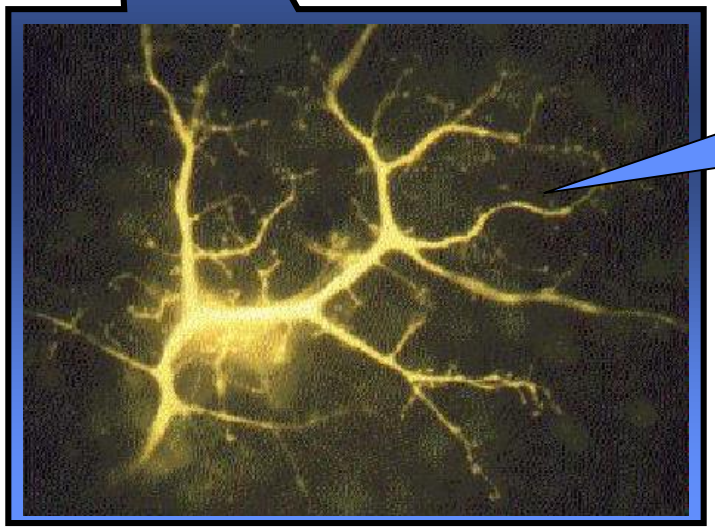
The human brain is extremely good at classifying images

Can we develop classification methods by emulating the brain?

Brain Computer: What is it?



Human brain contains a massively interconnected net of 10^{10}-10^{11} (10 billion) neurons (cortical cells)

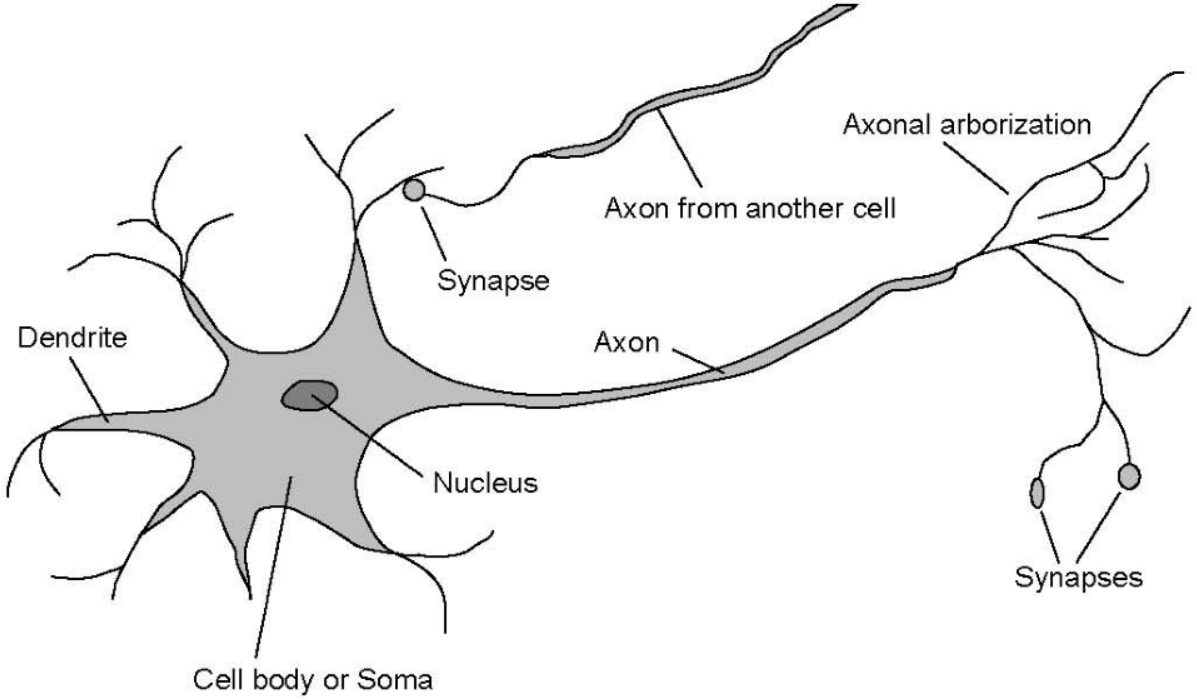


Biological Neuron

- The simple "arithmetic computing" element

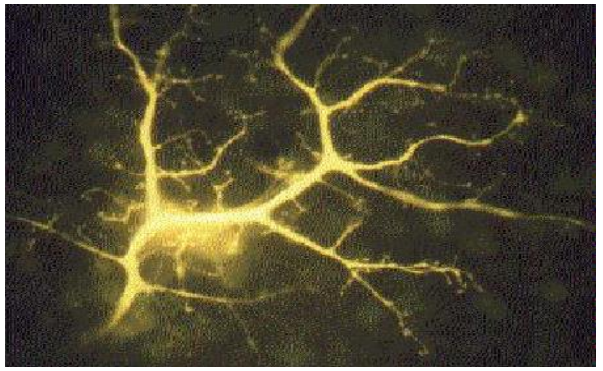
Brains

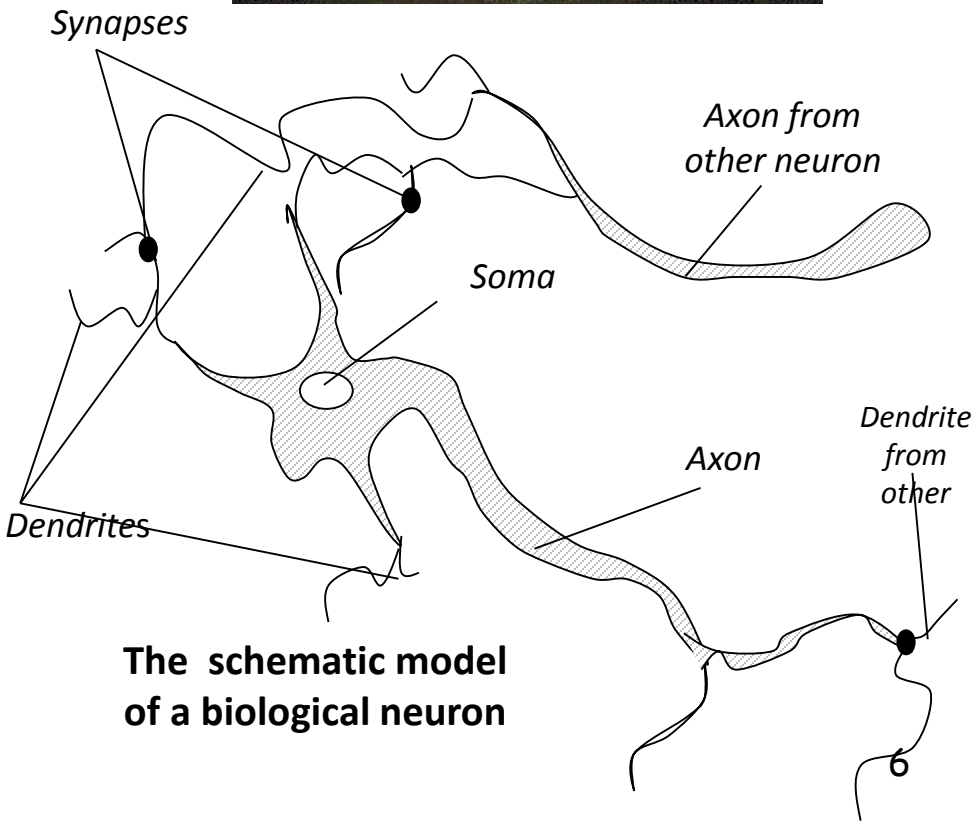
10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential



Biological Neurons

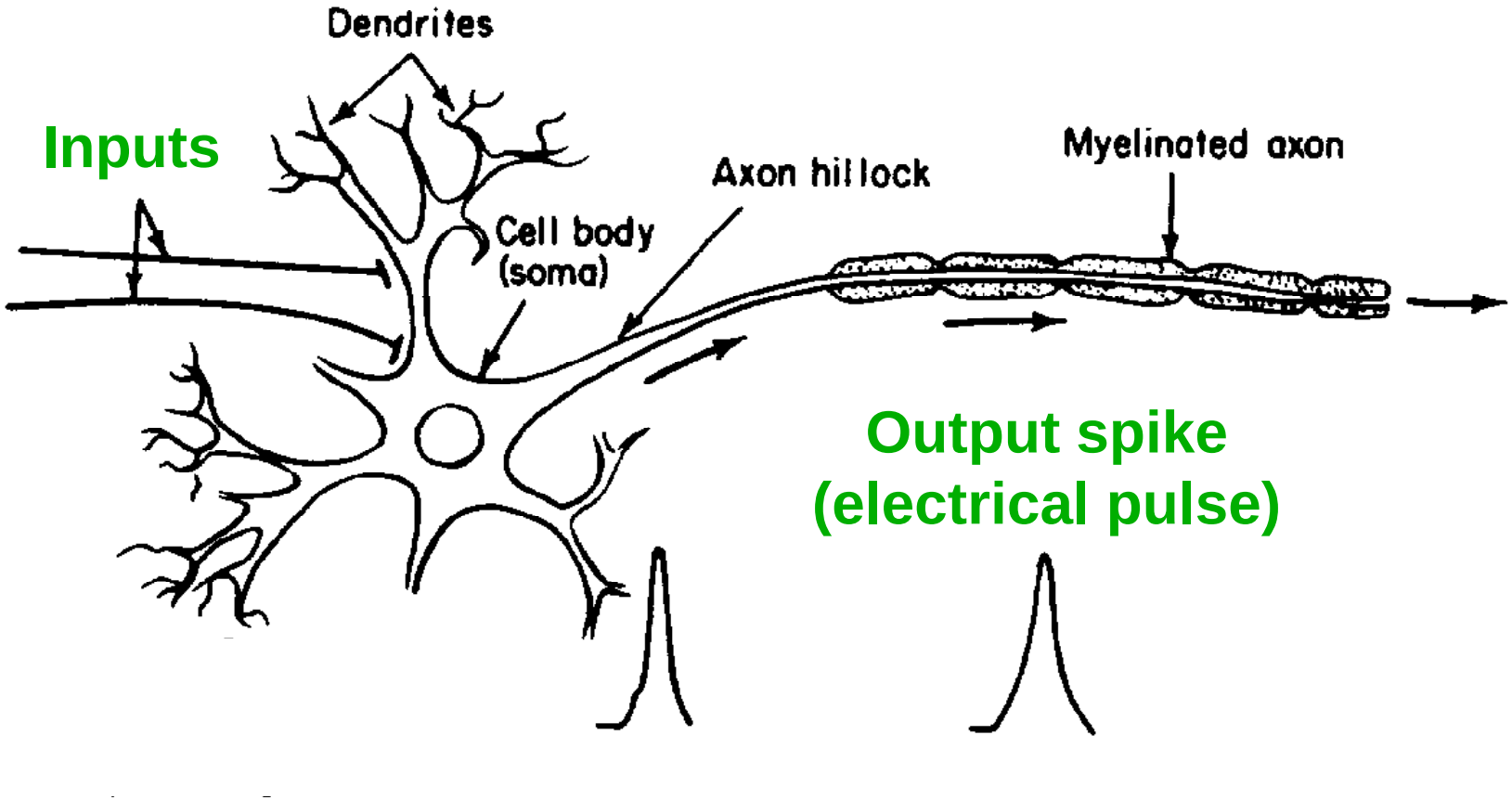
1. ***Soma* or *body cell* -** is a large, round central body in which almost all the logical functions of the neuron are realized.
2. ***The axon* (output)**, is a nerve fibre attached to the soma which can serve as a final output channel of the neuron. An axon is usually highly branched.
3. ***The dendrites* (inputs)-** represent a highly branching tree of fibres. These long irregularly shaped nerve fibres (processes) are attached to the soma.
4. ***Synapses*** are specialized contacts on a neuron which are the termination points for the axons from other neurons.





The schematic model of a biological neuron

Neurons communicate via spikes

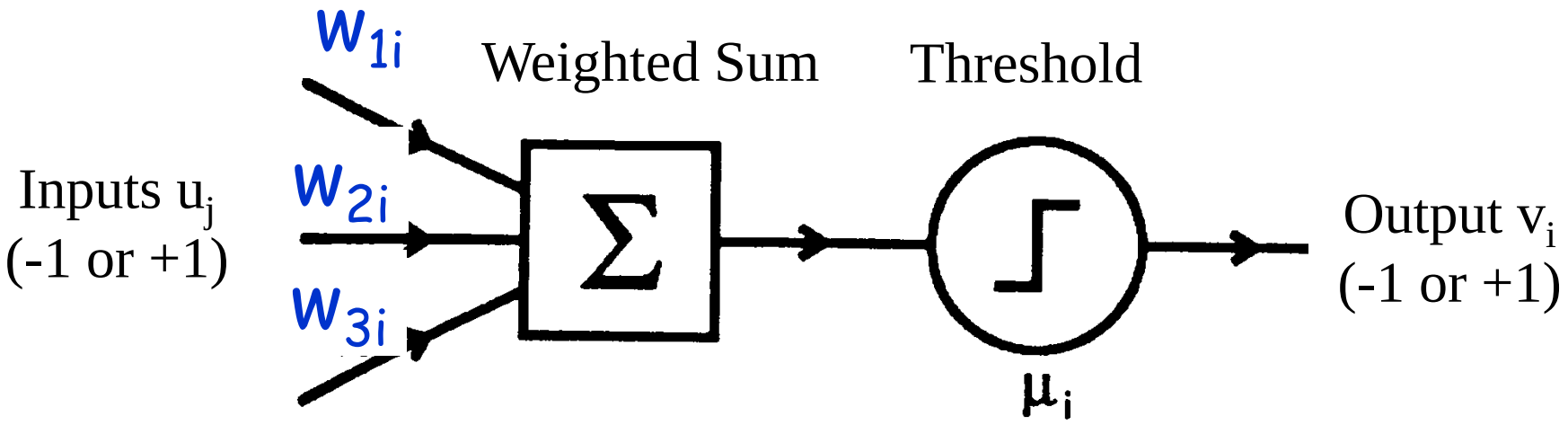


Output spike roughly dependent on whether sum of all inputs reaches a threshold

Neurons as "Threshold Units"

Artificial neuron:

- m binary inputs (-1 or 1), 1 output (-1 or 1)
- Synaptic weights w_{ji}
- Threshold μ_i



$$v_i = \Theta(\sum_j w_{ji} u_j - \mu_i)$$

$\Theta(x) = 1$ if $x > 0$ and -1 if $x \leq 0$

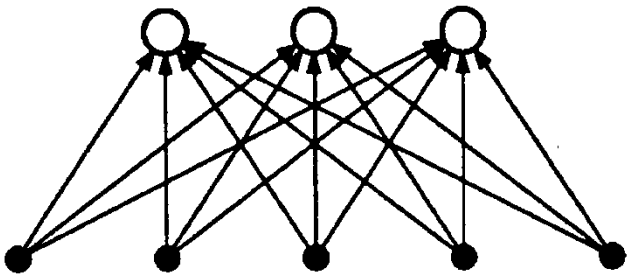
"Perceptrons" for Classification

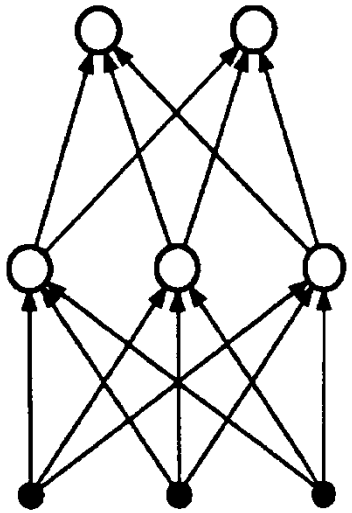
Fancy name for a type of layered "feed-forward" networks (no loops)

Uses artificial neurons ("units") with binary inputs and outputs

Multilayer

Single-layer





Perceptrons and Classification

Consider a single-layer perceptron

- Weighted sum forms a *linear hyperplane*

$$\sum_j w_{ji} u_j - \mu_i = 0$$

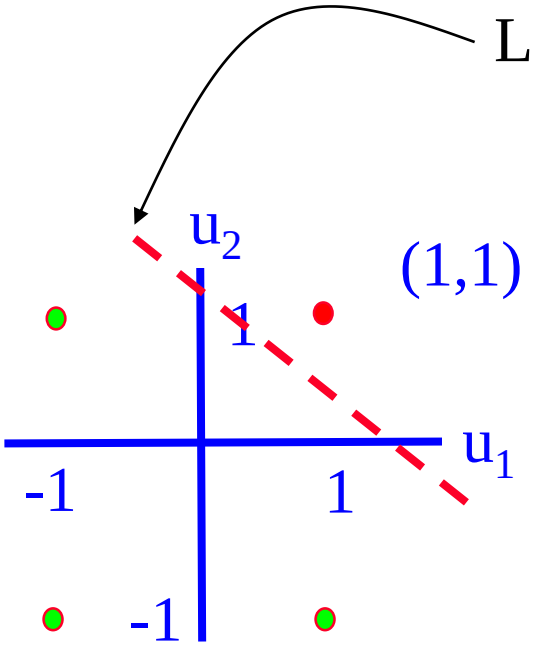
- Everything *on one side* of this hyperplane is in class 1 (output = +1) and everything *on other side* is class 2 (output = -1)

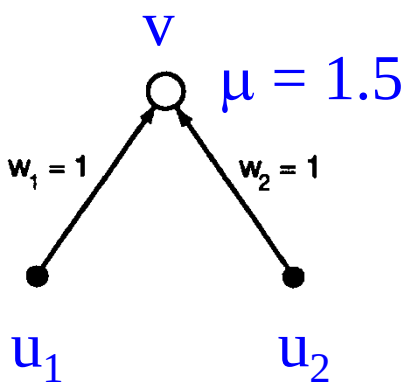
Any function that is linearly separable can be computed by a perceptron

Linear Separability

Example: AND is linearly separable

u_1	u_2	AND
-1	-1	-1
1	-1	-1
-1	1	-1
1	1	1





$v = 1$ iff $u_1 + u_2 - 1.5 > 0$

Similarly for OR and NOT

How do we *learn* the appropriate weights given only examples of (input,output)?

Idea: Change the weights to decrease the error in output

Perceptron Training Rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

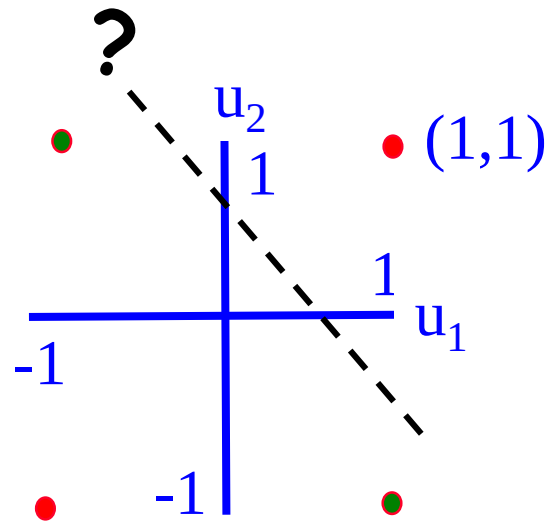
$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called *learning rate*

What about the XOR function?

u_1	u_2	XOR
-1	-1	1
1	-1	-1
-1	1	-1
1	1	1



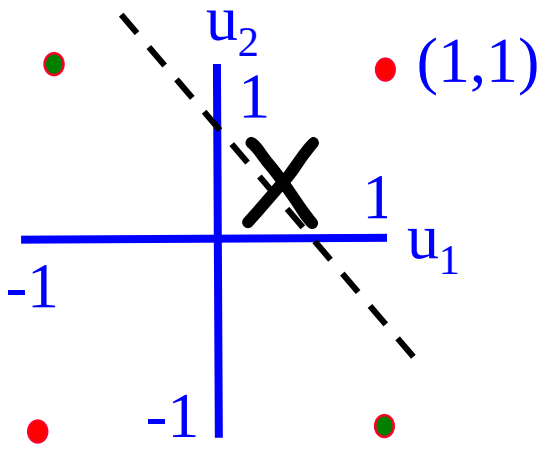
Can a perceptron separate the +1 outputs from the -1 outputs?

Linear Inseparability

Perceptron with threshold units fails if classification task is not linearly separable

- Example: XOR
- No single line can separate the "yes" (+1) outputs from the "no" (−1) outputs!

Minsky and Papert's book showing such negative results put a damper on neural networks research for over a decade!



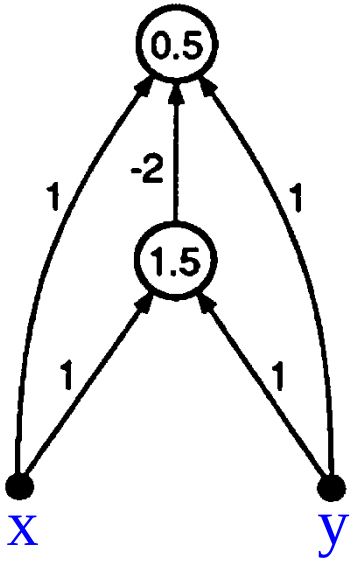
How do we deal with linear inseparability?

Idea 1: Multilayer Perceptrons

Removes limitations of single-layer networks

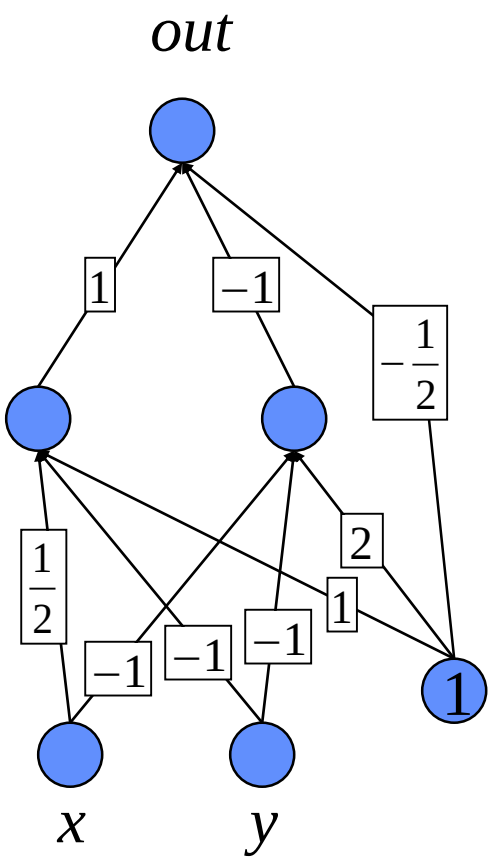
- Can solve XOR

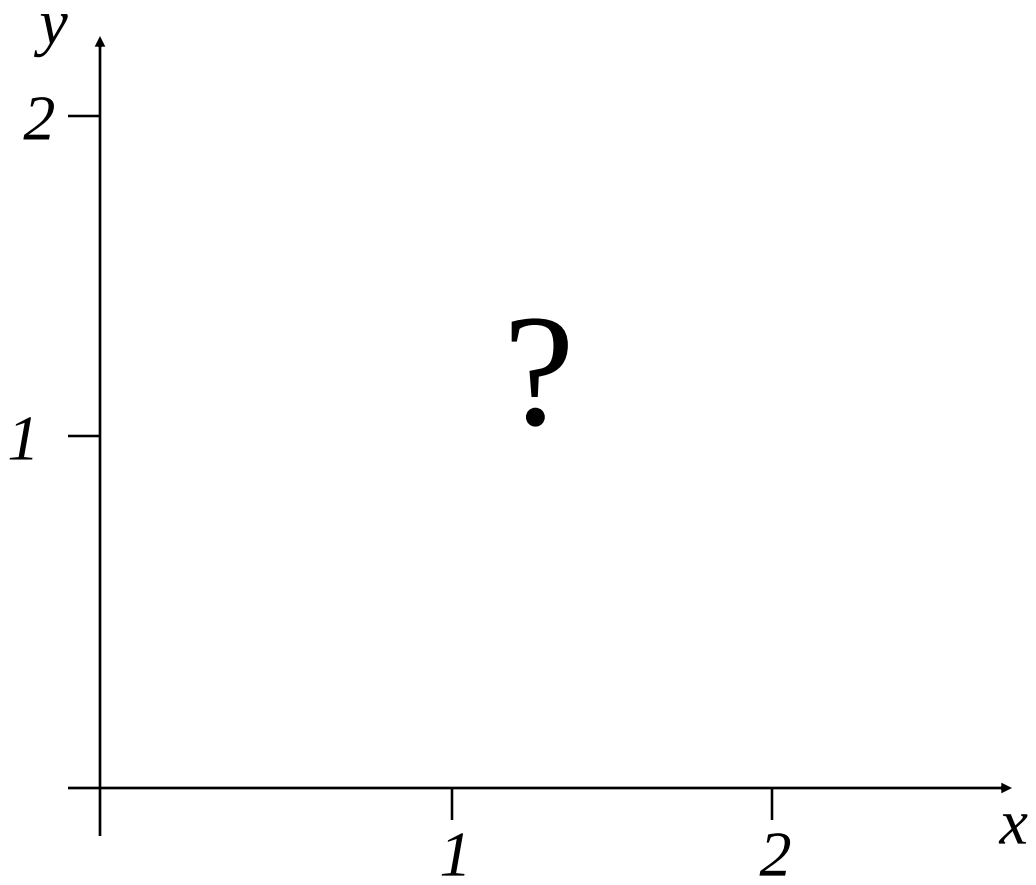
Example: Two-layer perceptron that computes XOR



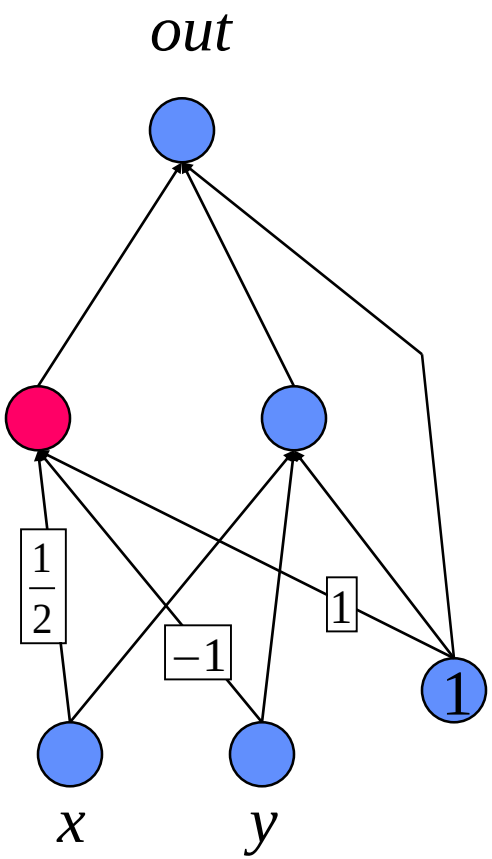
Output is +1 if and only if $x + y - 2\Theta(x + y - 1.5) - 0.5 > 0$

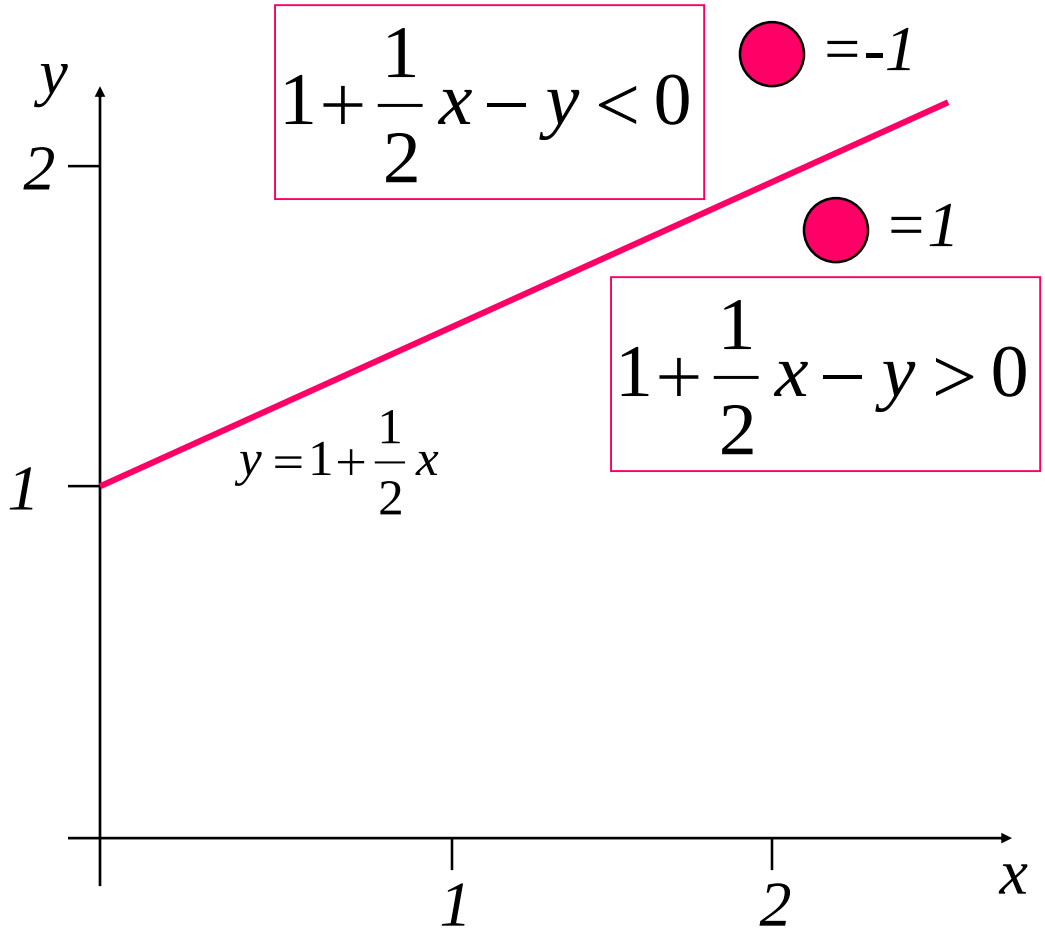
Multilayer Perceptron: What does it do?



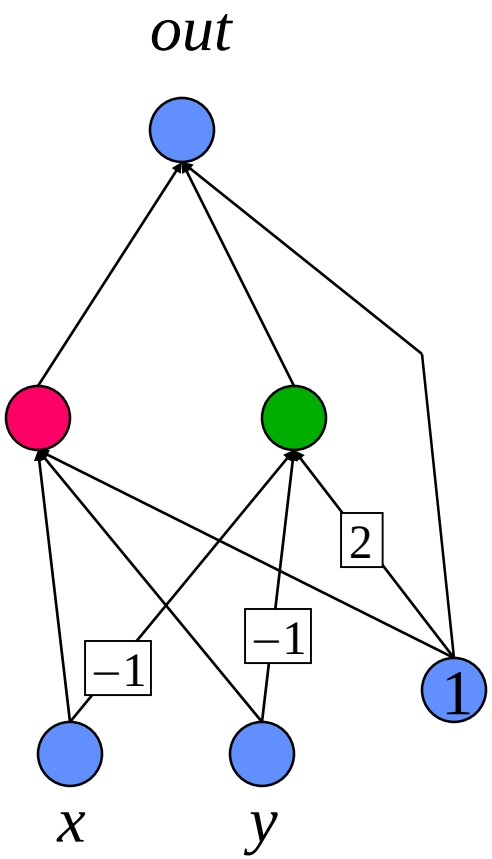


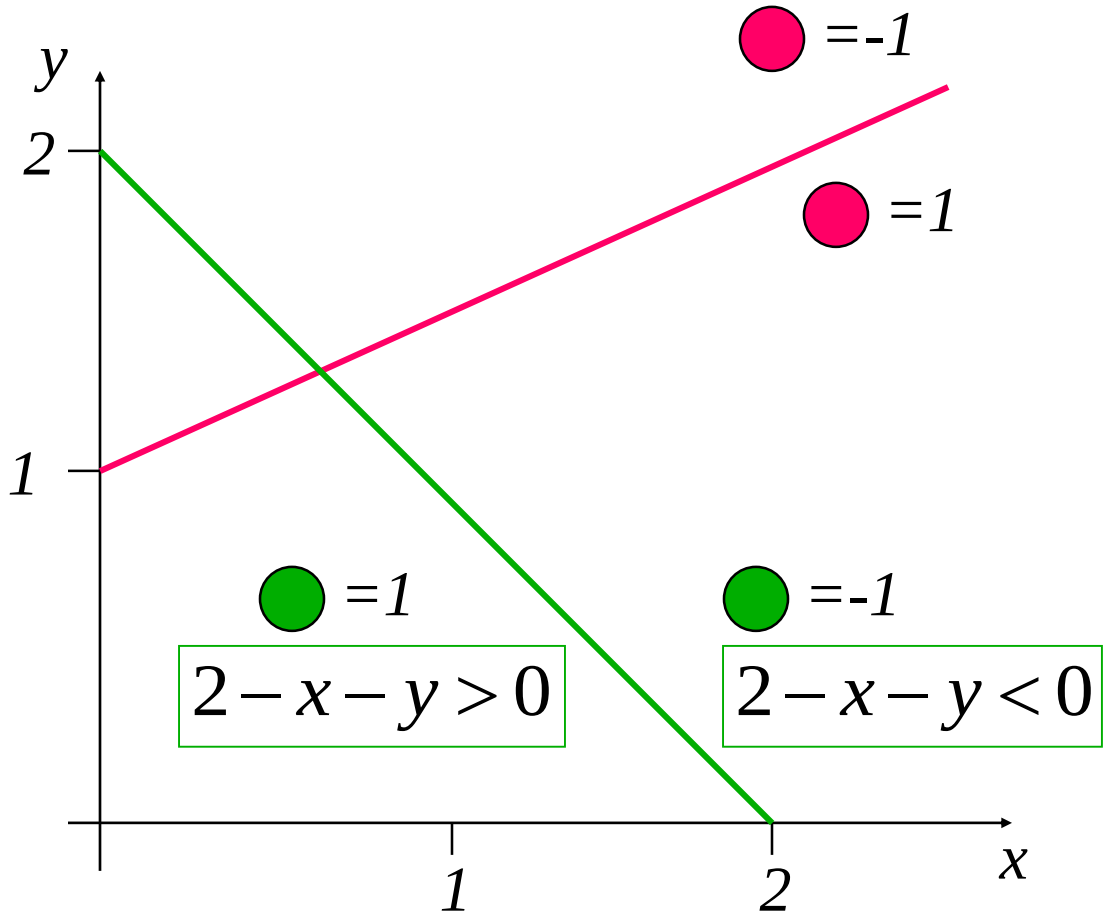
Multilayer Perceptron: What does it do?



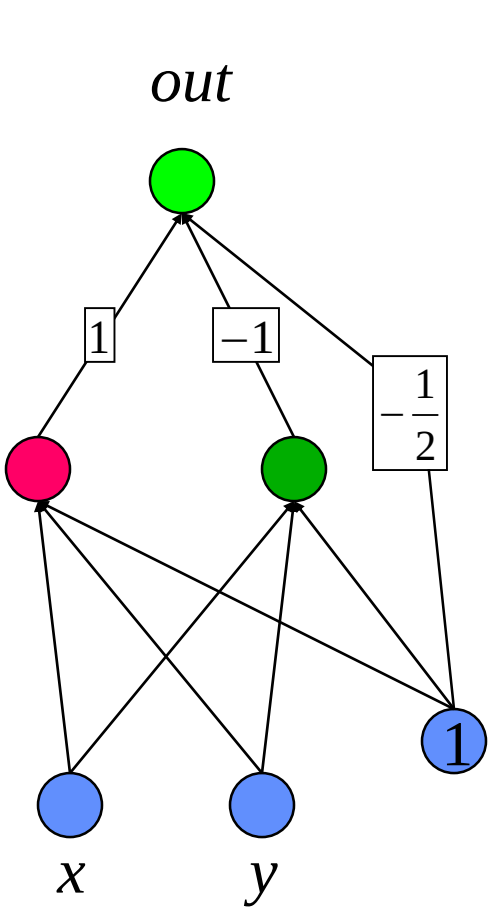


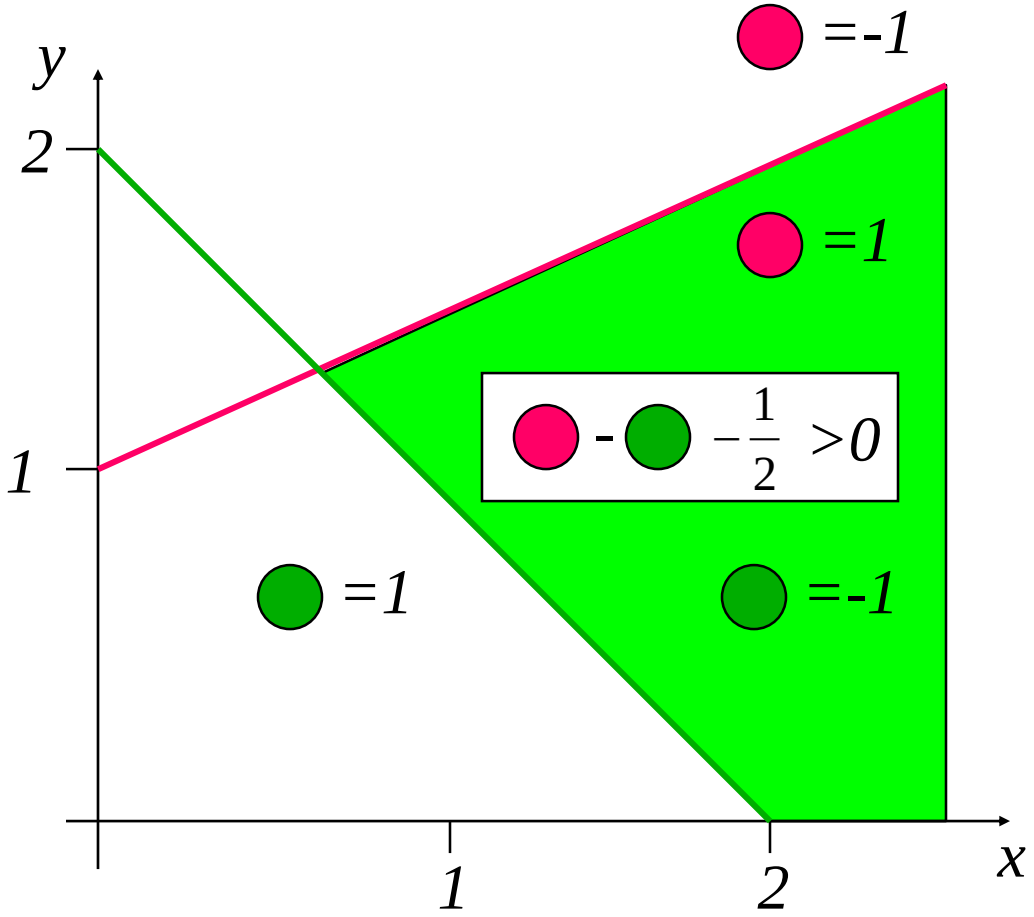
Multilayer Perceptron: What does it do?



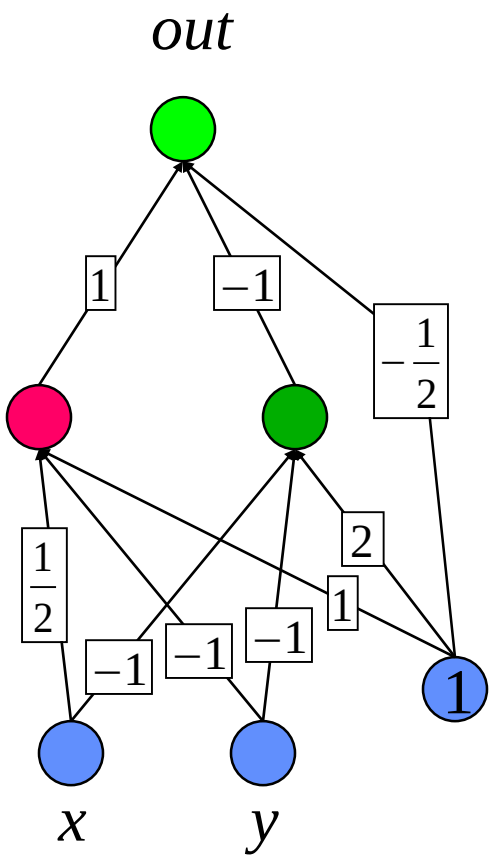


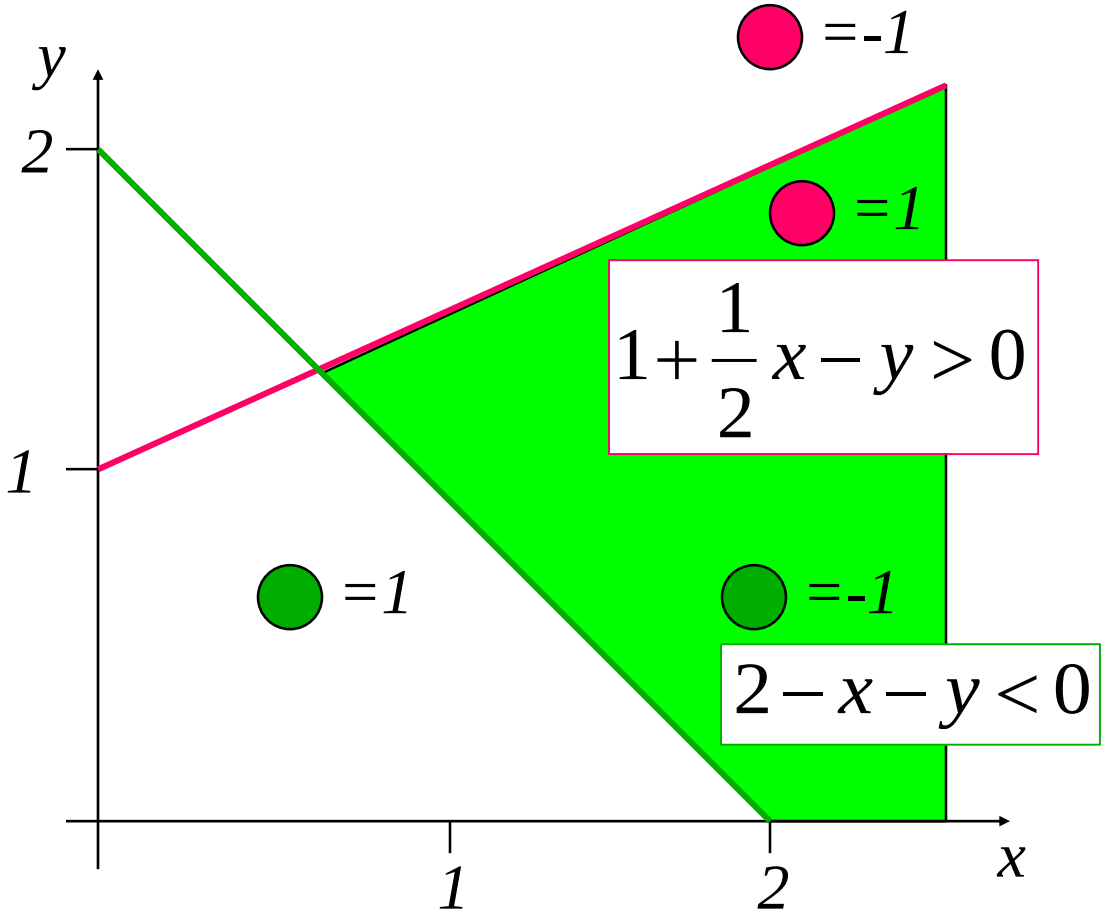
Multilayer Perceptron: What does it do?



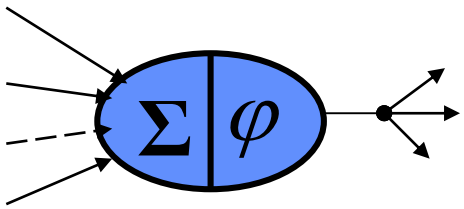


Perceptrons as Constraint Satisfaction Networks



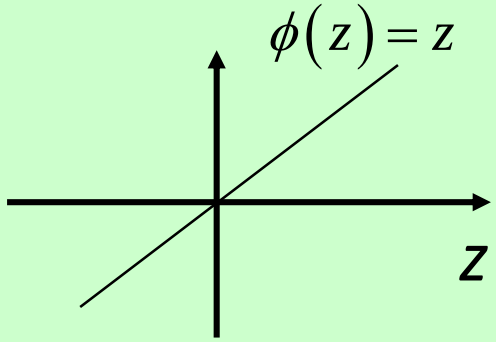


Artificial Neuron: Most Popular Activation Functions



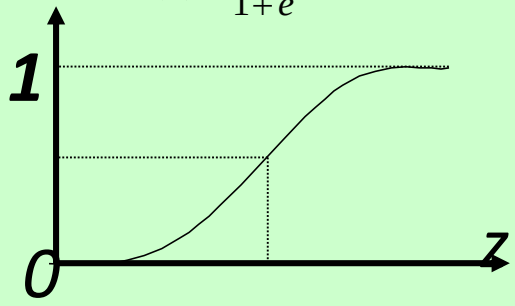
Linear activation

$$\phi(z) = z$$



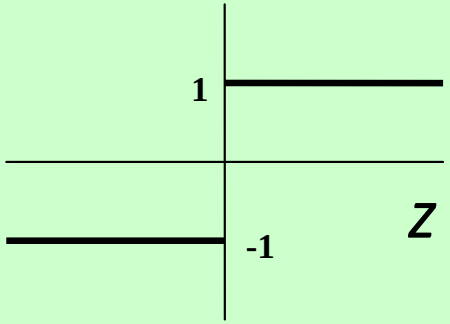
Logistic activation

$$\phi(z) = \frac{1}{1+e^{-\alpha z}}$$



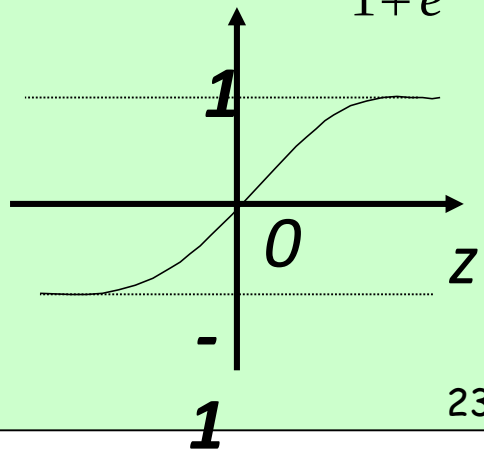
Threshold activation

$$\phi(z) = \text{sign}(z) = \begin{cases} 1, & if \quad z \geq 0, \\ -1, & if \quad z < 0. \end{cases}$$



Hyperbolic tangent activation

$$\varphi(u) = tanh(\gamma u) = \frac{1-e^{-2\gamma u}}{1+e^{-2\gamma u}}$$



Neural Network Issues

- Multi-layer perceptrons can represent any function
- Training multi-layer perceptrons hard
  - Backpropagation

- Early successes
  - Keeping the car on the road

- Difficult to debug
  - Opaque

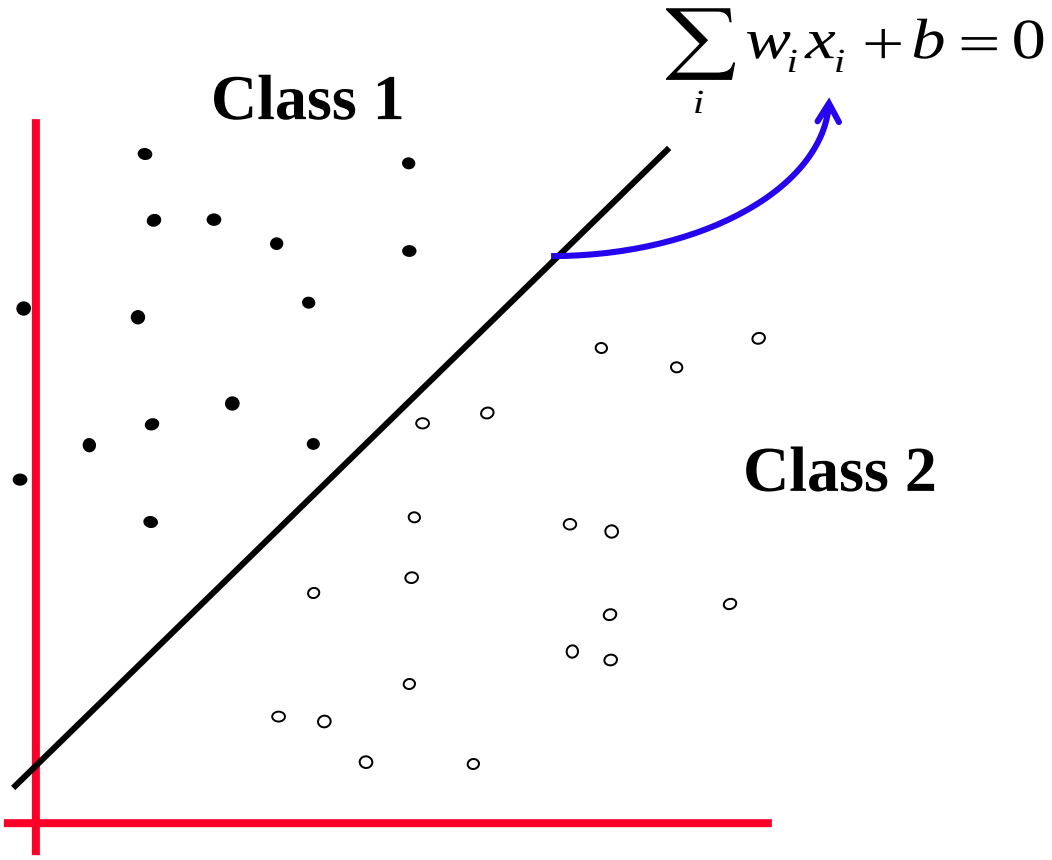
Back to Linear Separability

- **Recall: Weighted sum in perceptron forms a *linear hyperplane***

$$\sum_i w_i x_i + b = 0$$

- Due to threshold function, everything *on one side* of this hyperplane is labeled as class 1 (output = +1) and everything *on other side* is labeled as class 2 (output = -1)

Separating Hyperplane

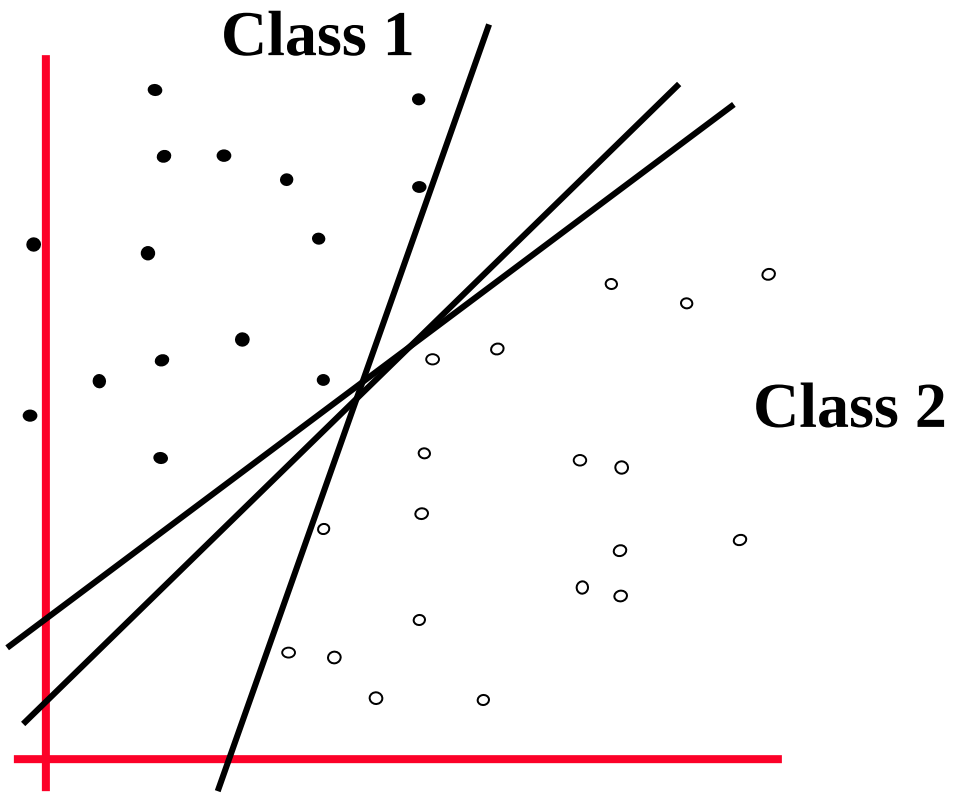


- denotes +1 output
- denotes -1 output

Need to choose **w** and *b* based on training data

Separating Hyperplanes

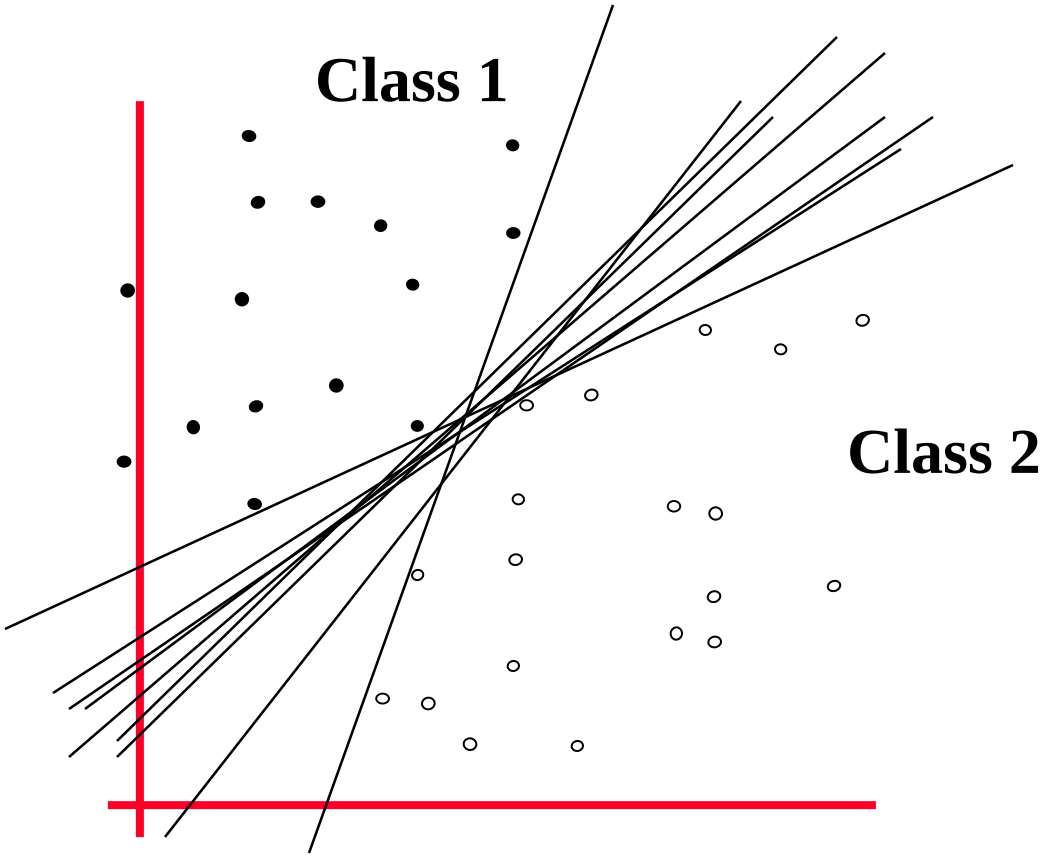
Different choices of **w** and *b* give different hyperplanes



- denotes +1 output
- denotes -1 output

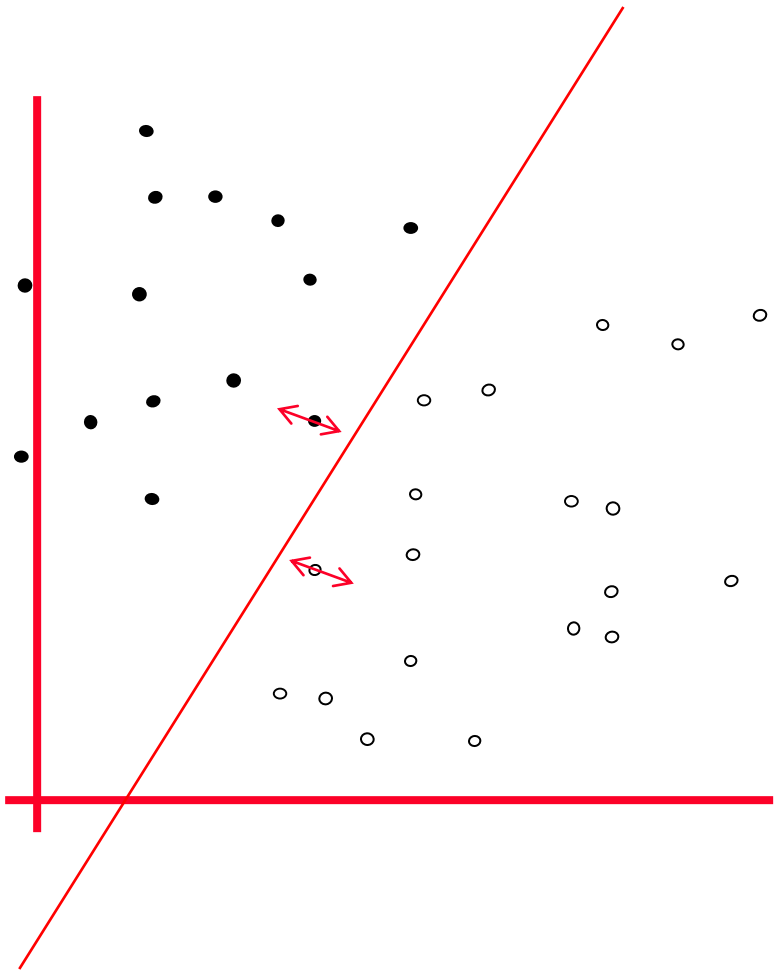
(This and next few slides adapted from Andrew Moore's)

Which hyperplane is best?



- denotes +1 output
- denotes -1 output

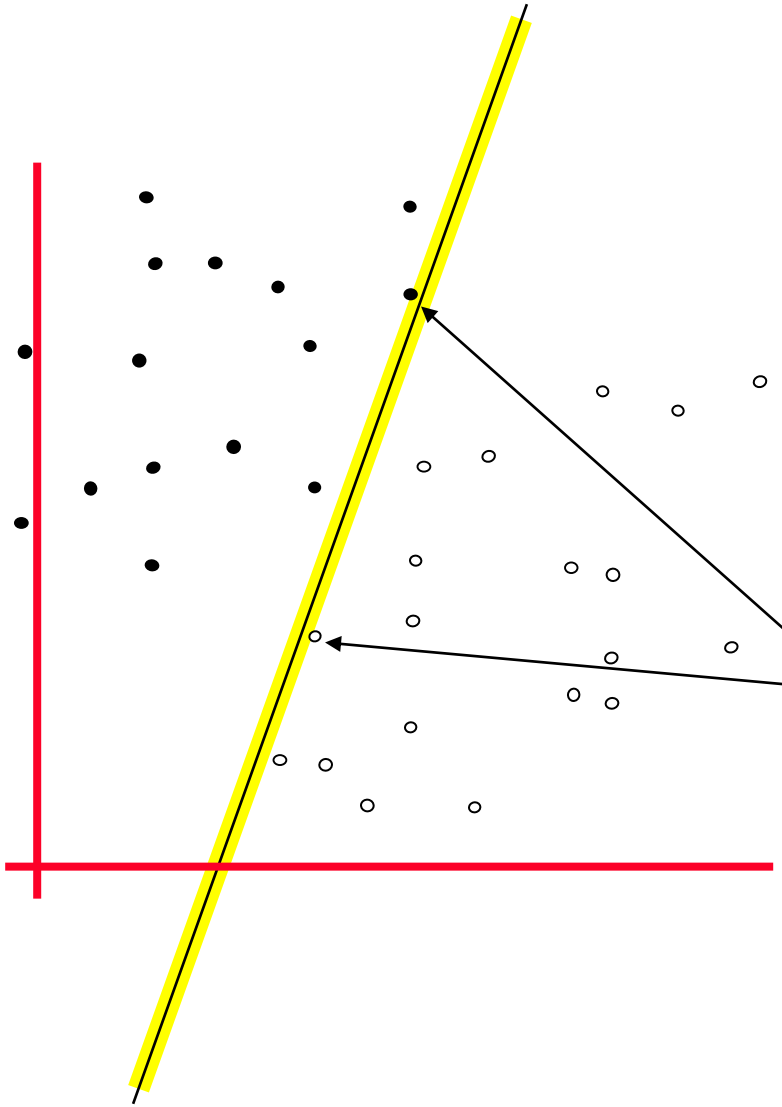
How about the one right in the middle?



Intuitively, this boundary seems good

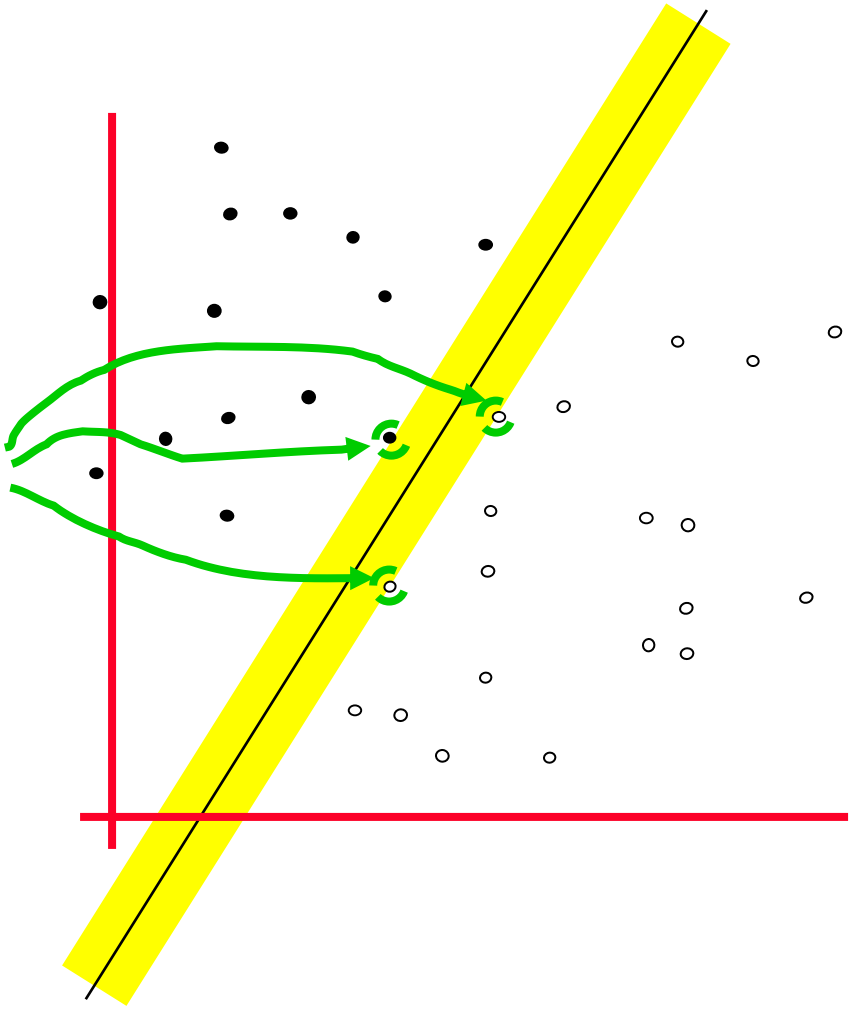
Avoids misclassification of new test points if they are generated from the same distribution as training points

Margin



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

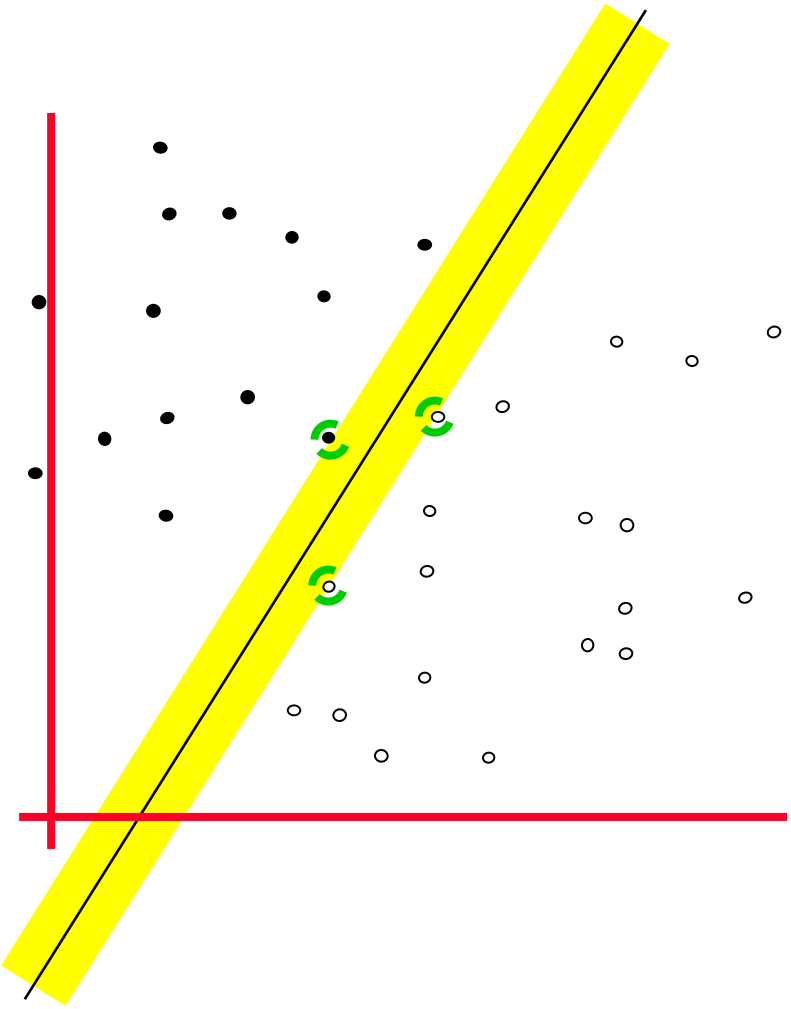
Maximum Margin and Support Vector Machine



Support Vectors are those datapoints that the margin pushes up against

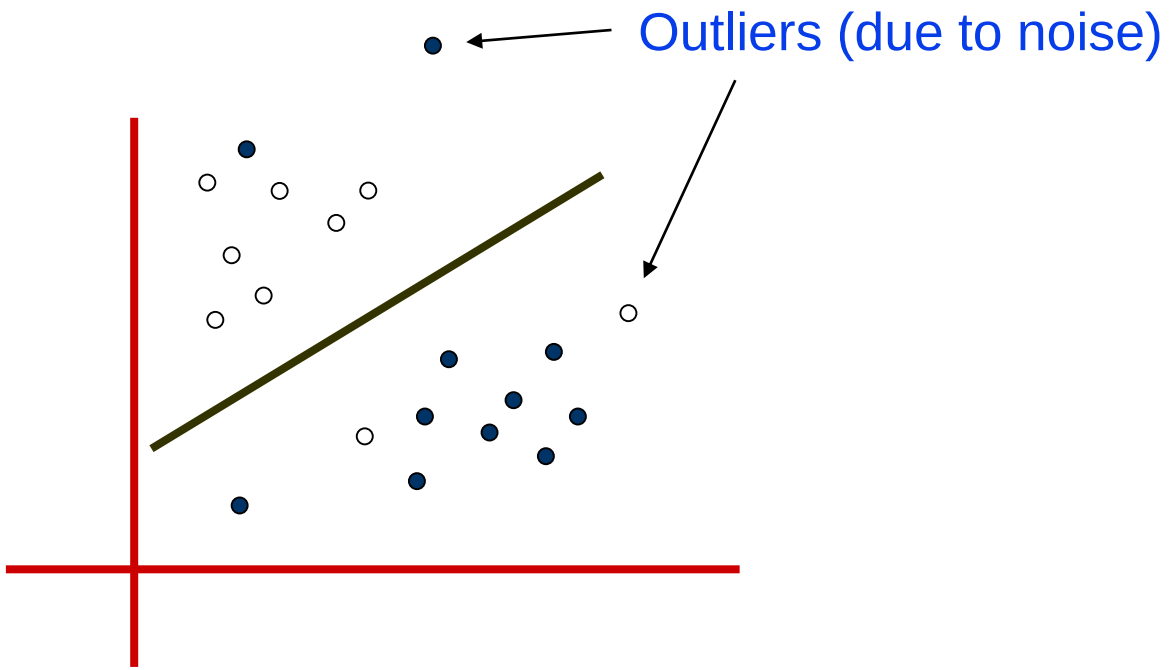
The maximum margin classifier is called a Support Vector Machine (in this case, a Linear SVM or LSVM)

Why Maximum Margin?

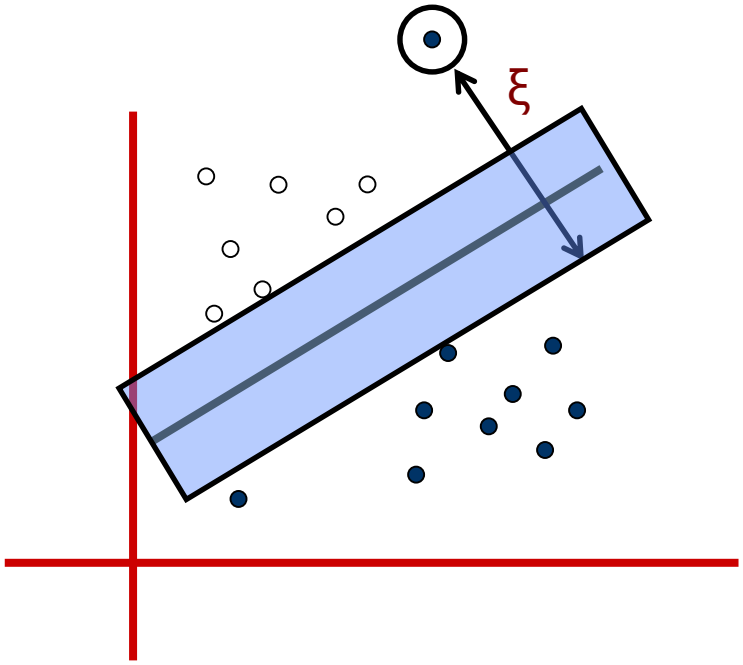


- Robust to small perturbations of data points near boundary
- There exists theory showing this is best for generalization to new points
- Empirically works great

What if data is not linearly separable?



Approach 1: Soft Margin SVMs

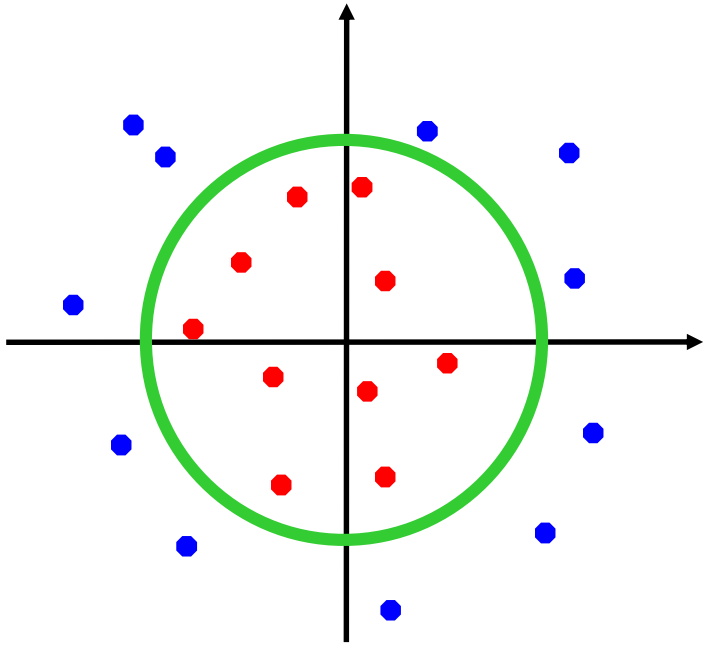


Allow *errors* ξ_i (deviations from margin)

Trade off margin with errors.

Minimize: margin + error-penalty

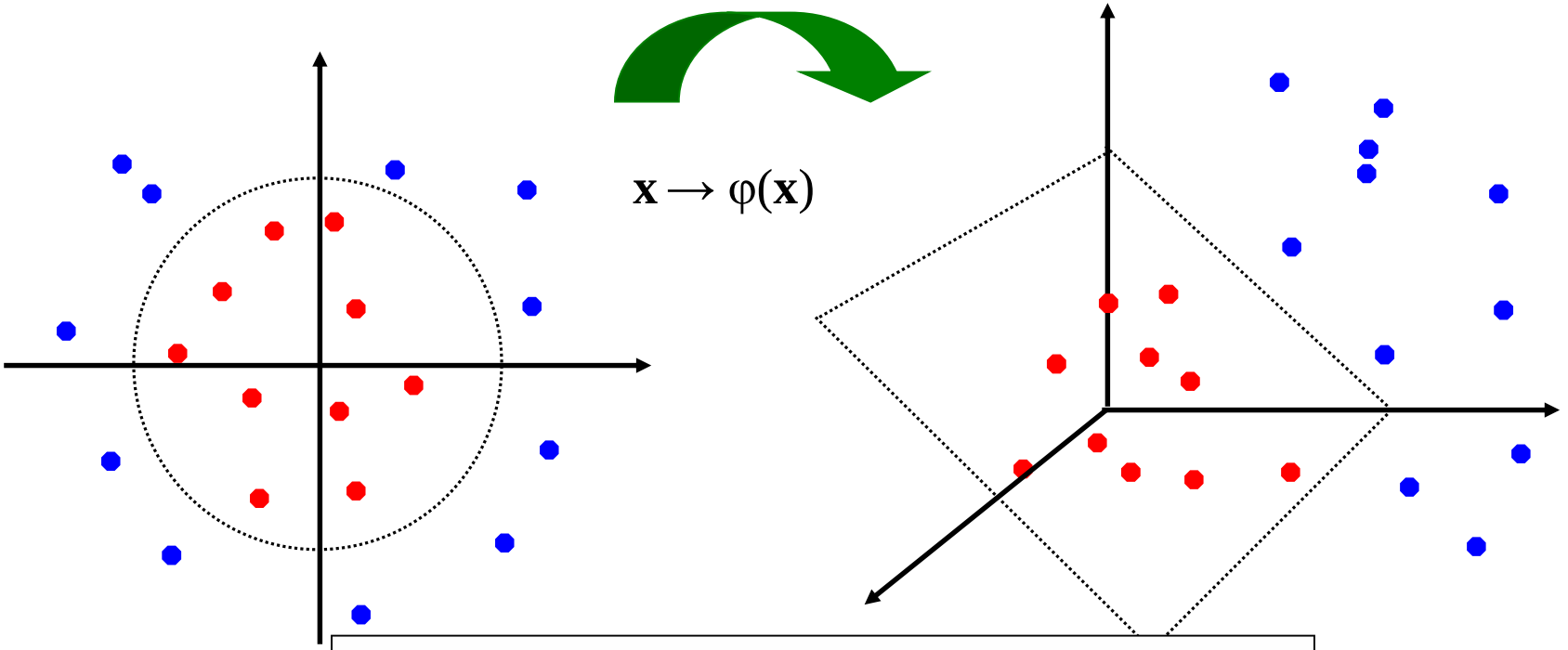
What if data is not linearly separable: Other ideas?



Not linearly separable

What if data is not linearly separable?

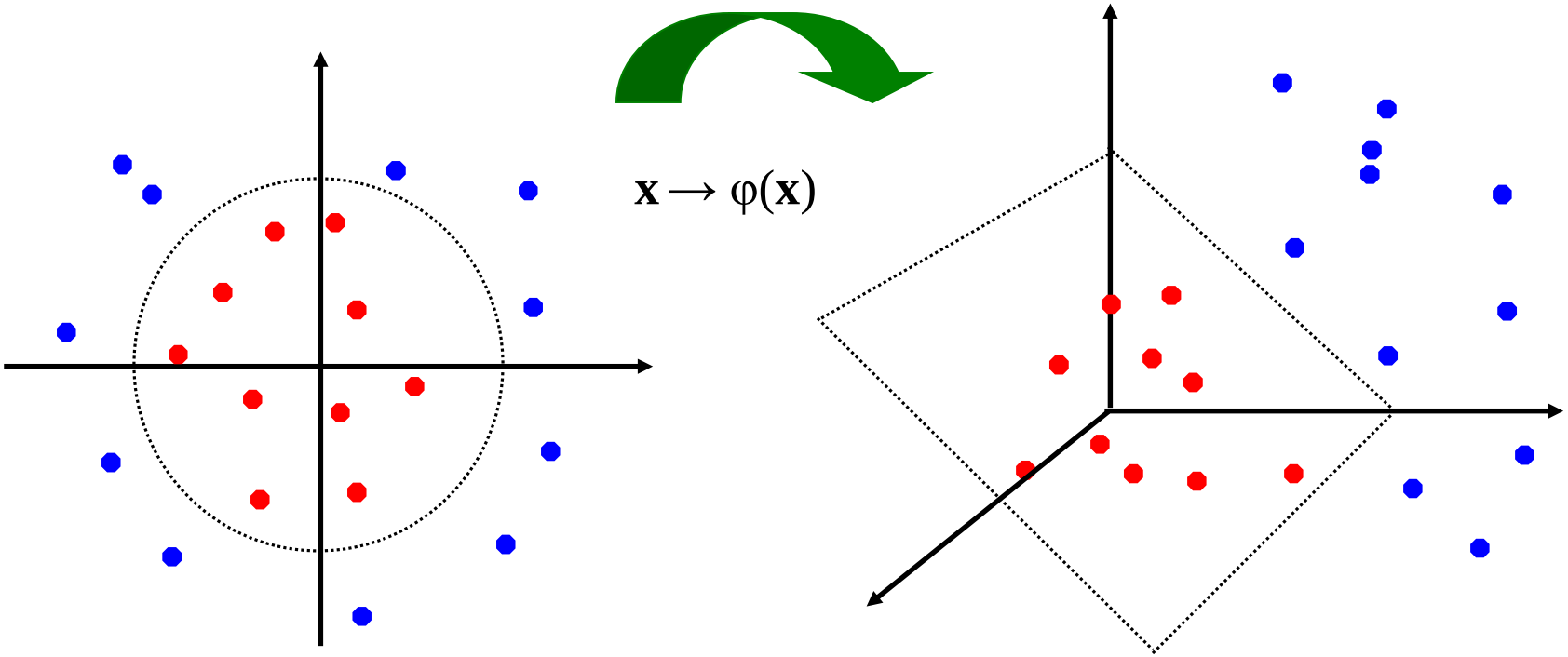
Approach 2: Map original input space to higher-dimensional feature space; use linear classifier in higher-dim. space



$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

Kernel: additional bias to convert into high d space

Problem with high dimensional spaces



Computation in high-dimensional feature space can be costly

The high dimensional projection function $\varphi(x)$ may be too complicated to compute

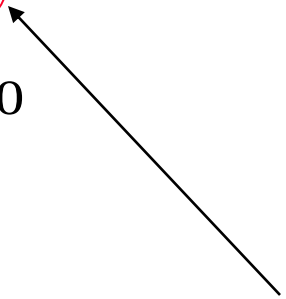
Kernel trick to the rescue!

The Kernel Trick

Dual Formulation: SVM maximizes the quadratic function:

$$\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{subject to } \alpha_i \geq 0 \text{ and } \sum_i \alpha_i y_i = 0$$



Insight:

The data points only appear as inner product

- No need to compute high-dimensional $\varphi(x)$ explicitly! Just replace inner product $x_i \cdot x_j$ with a kernel function $K(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$
- E.g., Gaussian kernel

$$K(x_i, x_j) = \exp(-||x_i - x_j||^2 / 2\sigma^2)$$

- E.g., Polynomial kernel

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^d$$

K-Nearest Neighbors

A simple *non-parametric* classification algorithm

Idea:

- Look around you to see how your neighbors classify data
- Classify a new data-point according to a *majority vote* of your k nearest neighbors

Distance Metric

How do we measure what it means to be a neighbor (what is "close")?

Appropriate distance metric depends on the problem

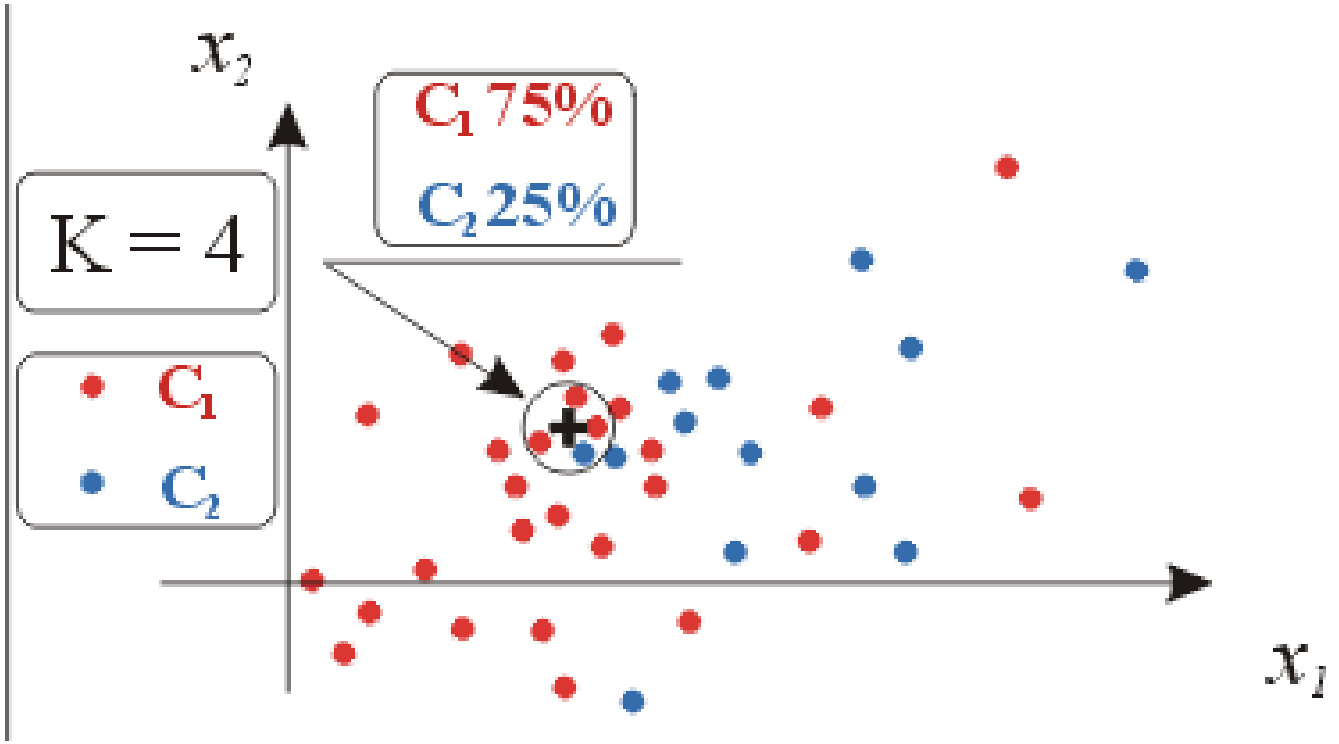
Examples:

- x discrete (e.g., strings): Hamming distance
  - $d(x_1,x_2)$ = # features on which x_1 and x_2 differ
- x continuous (e.g., vectors over reals): Euclidean distance
  - $d(x_1,x_2) = || x_1-x_2 ||$ = square root of sum of squared differences between corresponding elements of data vectors

Example

Input Data: 2-D points (x_1,x_2)

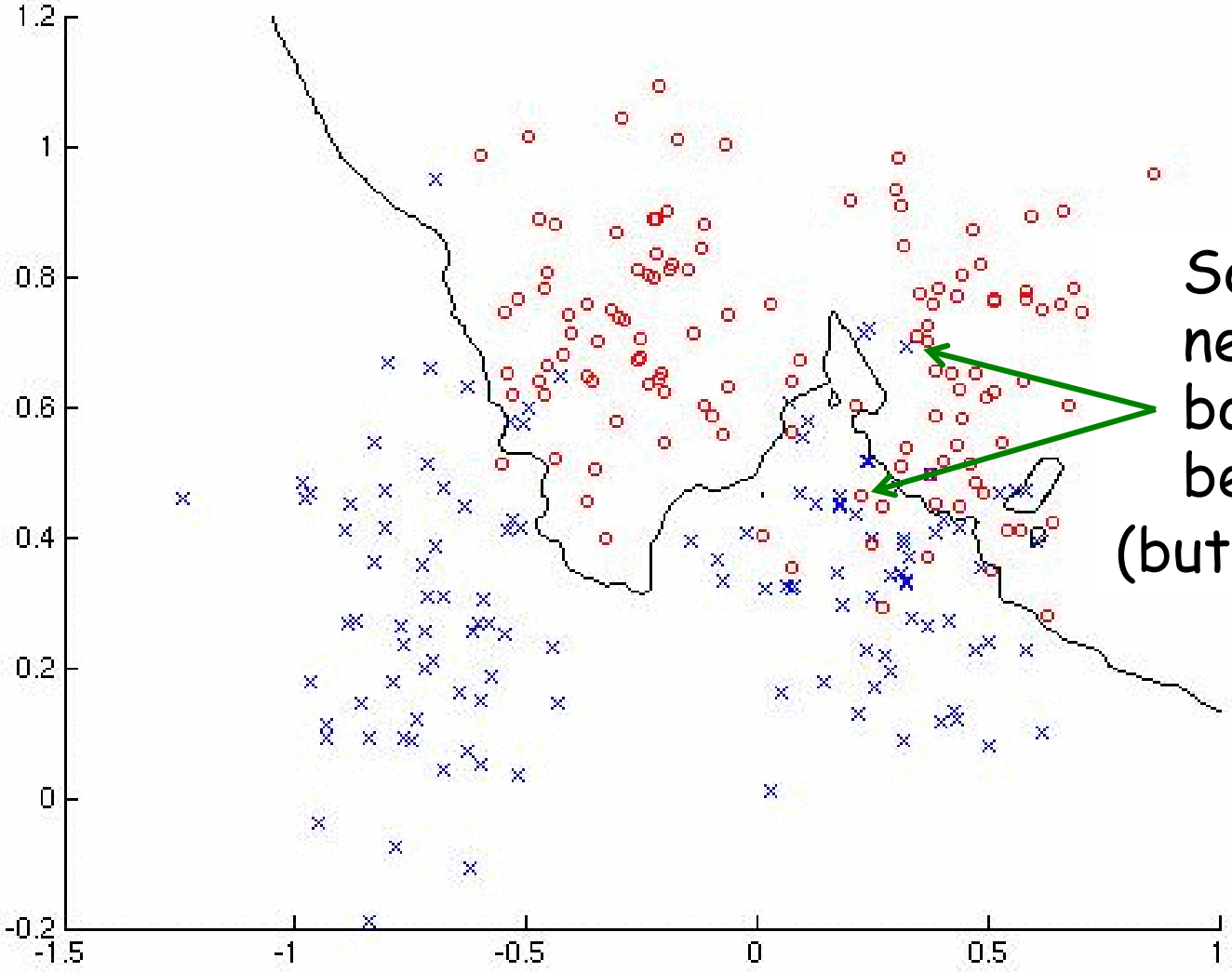
Two classes: C_1 and C_2. New Data Point +



K = 4: Look at 4 nearest neighbors of +

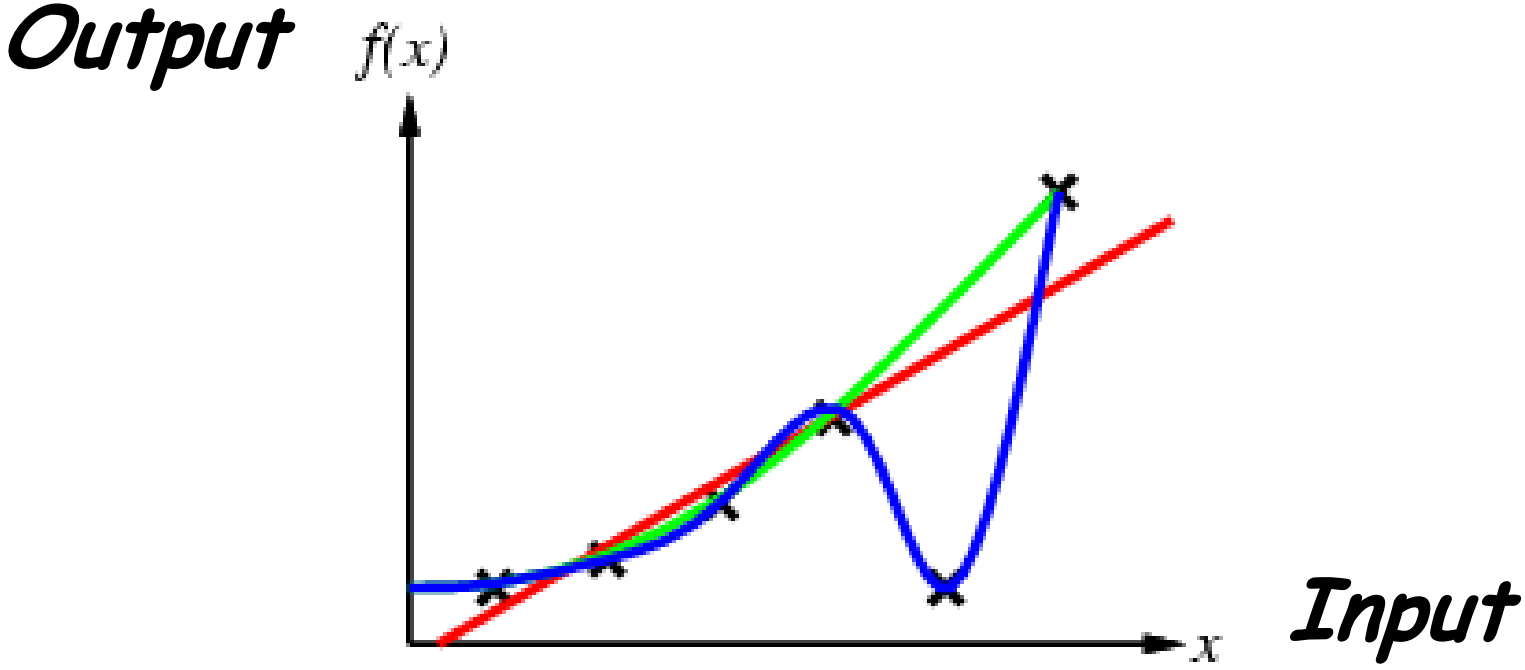
3 are in C_1, so classify + as C_1

Decision Boundary using K-NN



Some points near the boundary may be misclassified (but maybe noise)

What if we want to learn continuous-valued functions?



Regression

K-Nearest neighbor

take the average of k-close by points

Linear/Non-linear Regression

fit parameters (gradient descent)
minimizing the regression error/loss

Neural Networks

remove the threshold function

Large Feature Spaces

Easy to overfit

Regularization

add penalty for large weights

prefer weights that are zero or close to zero

minimize

regression error + C.regularization penalty

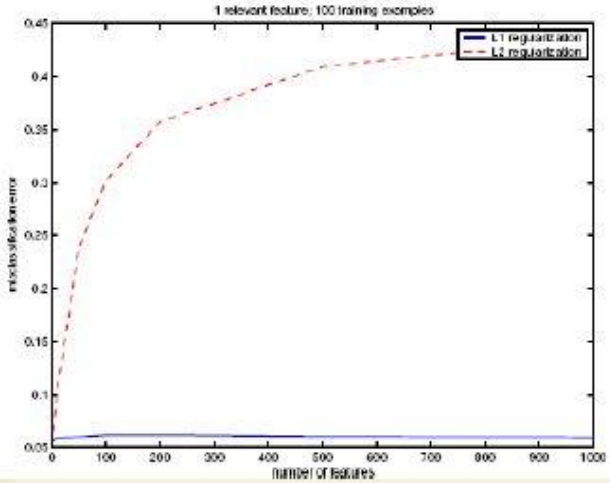
Regularizations

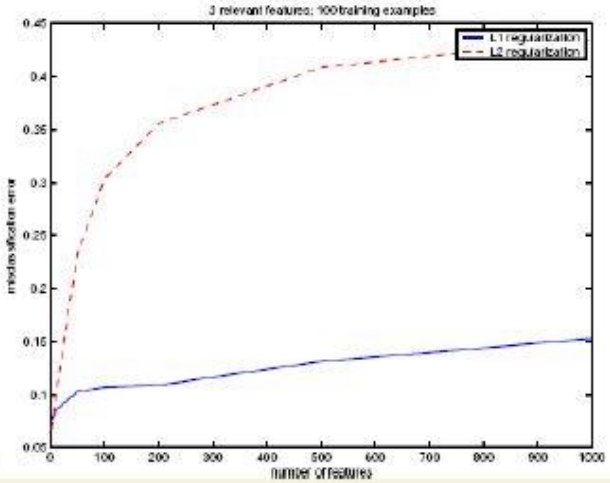
L1 : diamond
L2 : circle

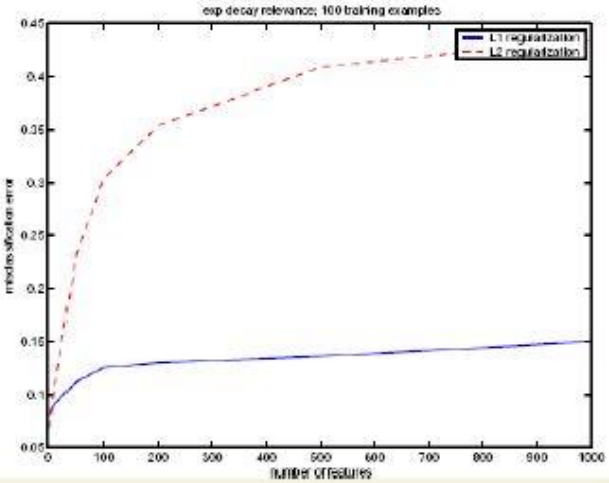
Derivatives
L1 : constant
L2 : high for large weights

L1 harder to optimize, but not too hard.
- discontinuous but convex

L1 vs. L2







1 relevant feature, 100 training examples

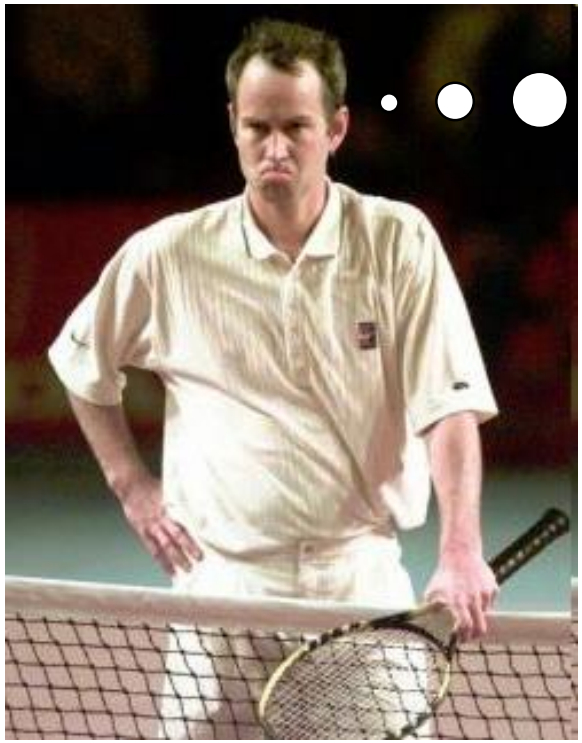
3 relevant feature, 100 training examples

exp decay in relevance, 100 training examples

Supervised Learning (contd)
Decision Trees

Mausam
(based on slides by UW-AI faculty)

Decision Trees



To play or not to play?

http://www.sfgate.com/blogs/images/sfgate/sgreen/2007/09/05/2240773250x321.jpg

Example data for learning the concept "Good day for tennis"

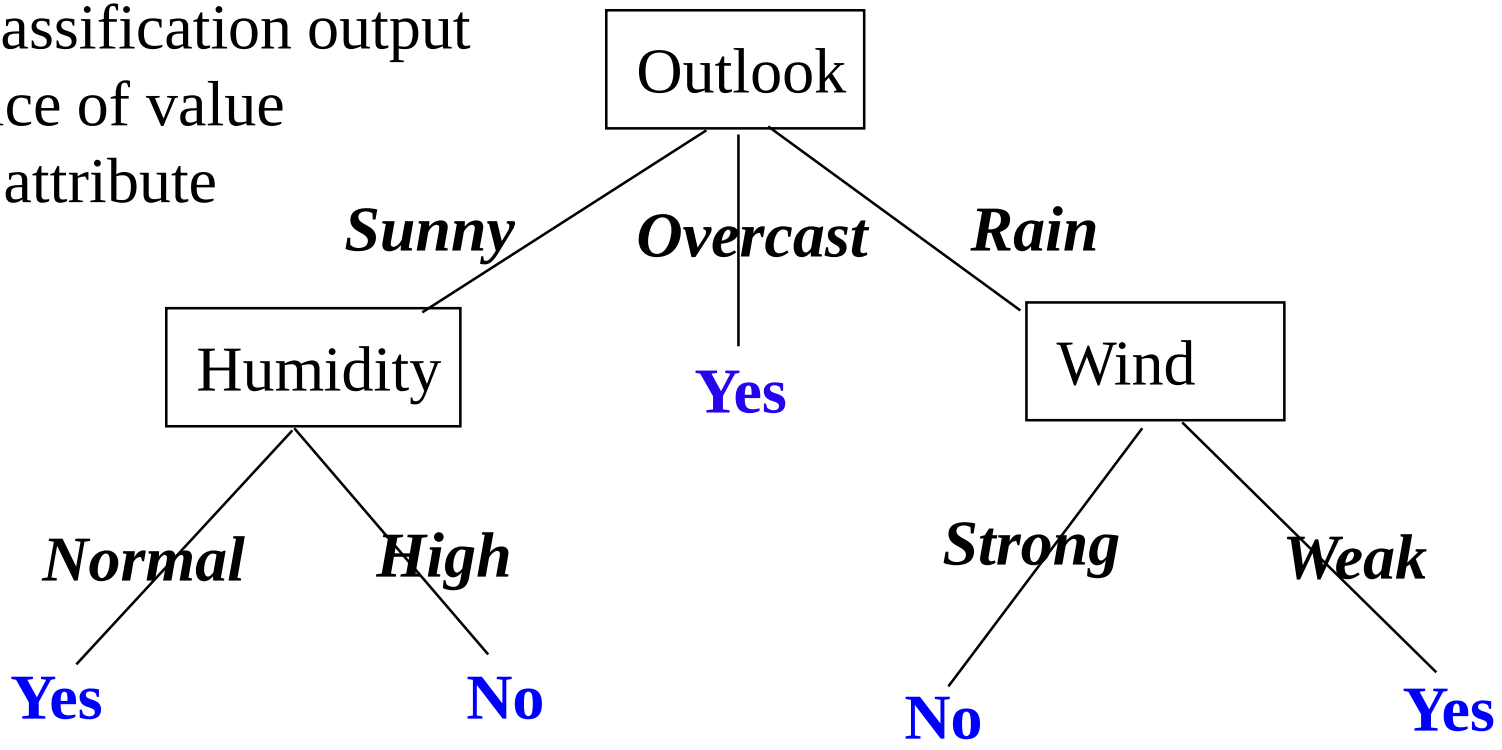
Day	Outlook	Humid	Wind	PlayTennis?
d1	s	h	w	n
d2	s	h	s	n
d3	o	h	w	y
d4	r	h	w	y
d5	r	n	w	y
d6	r	n	s	y
d7	o	n	s	y
d8	s	h	w	n
d9	s	n	w	y
d10	r	n	w	y
d11	s	n	s	y
d12	o	h	s	y
d13	o	n	w	y
d14	r	h	s	n

- Outlook = sunny, overcast, rain
- Humidity = high, normal
- Wind = weak, strong

A Decision Tree for the Same Data

Decision Tree for "PlayTennis?"

Leaves = classification output
Arcs = choice of value for parent attribute

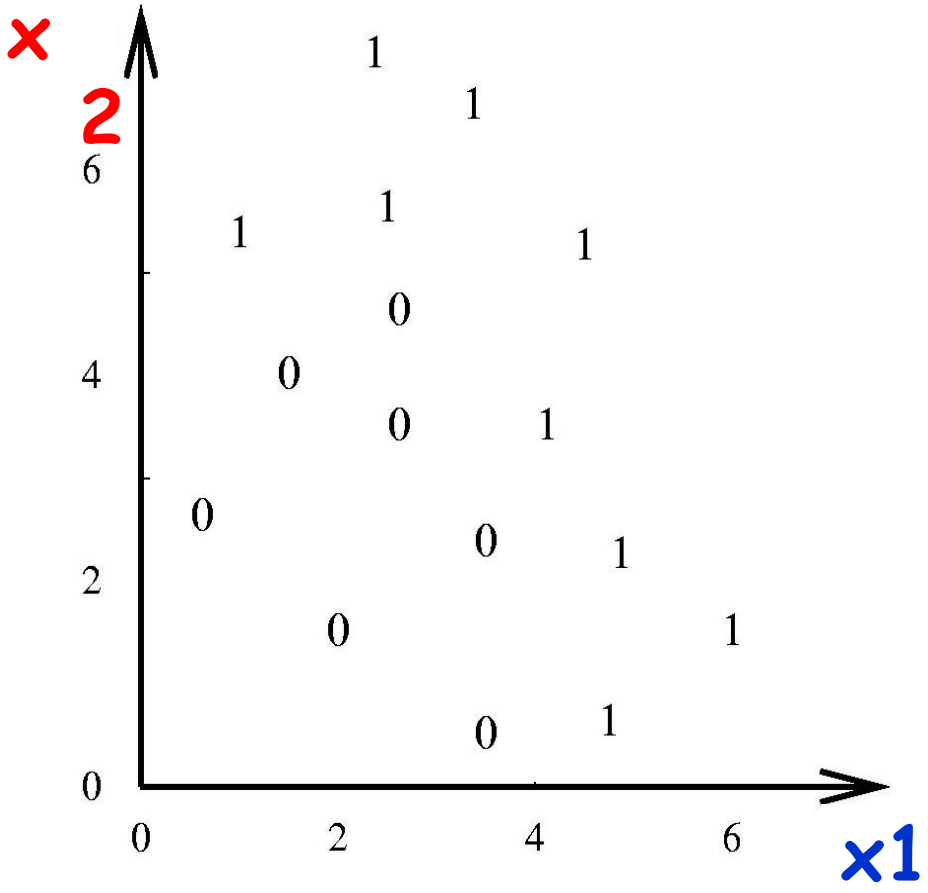


Decision tree is equivalent to logic in disjunctive normal form

PlayTennis $\Leftrightarrow$ (Sunny $\wedge$ Normal) $\vee$ Overcast $\vee$ (Rain $\wedge$ Weak)

Example: Decision Tree for Continuous Valued Features and Discrete Output

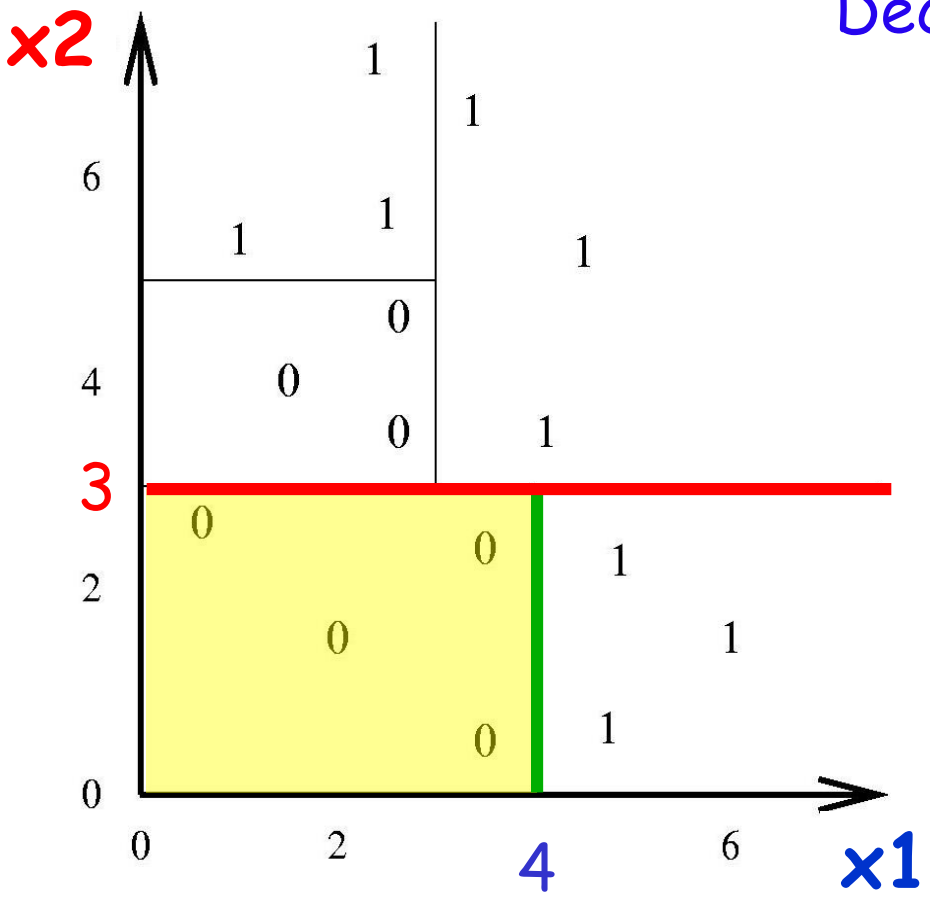
Input real number attributes (x1,x2), Classification output: 0 or 1



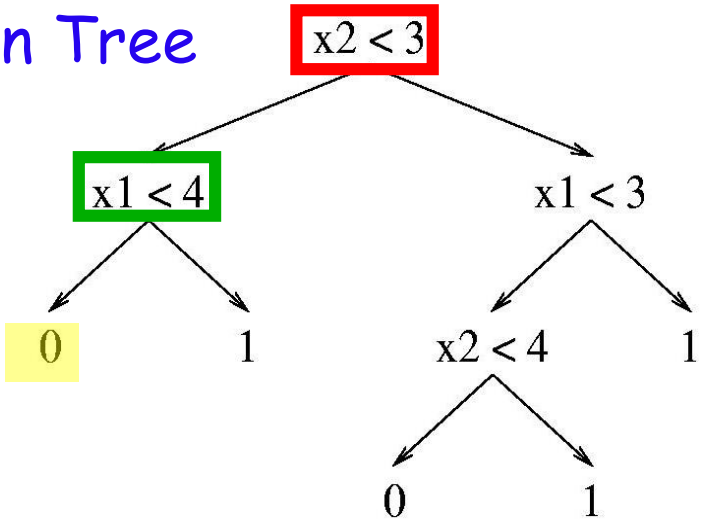
How do we branch using attribute values x1 and x2 to partition the space correctly?

Example: Classification of Continuous Valued Inputs

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the K classes.



Decision Tree

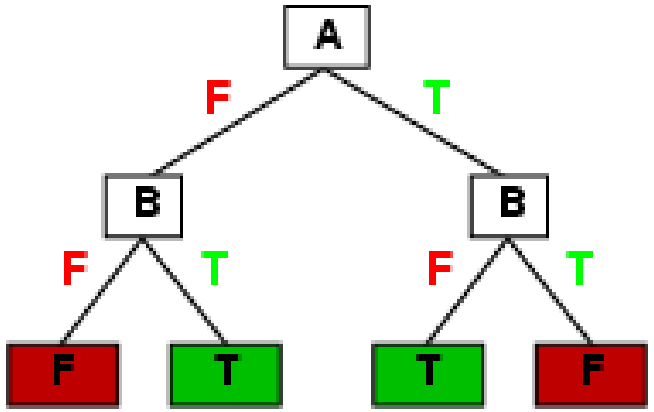


Expressiveness of Decision Trees

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

- But most likely won't generalize to new examples

Prefer to find more compact decision trees

Learning Decision Trees

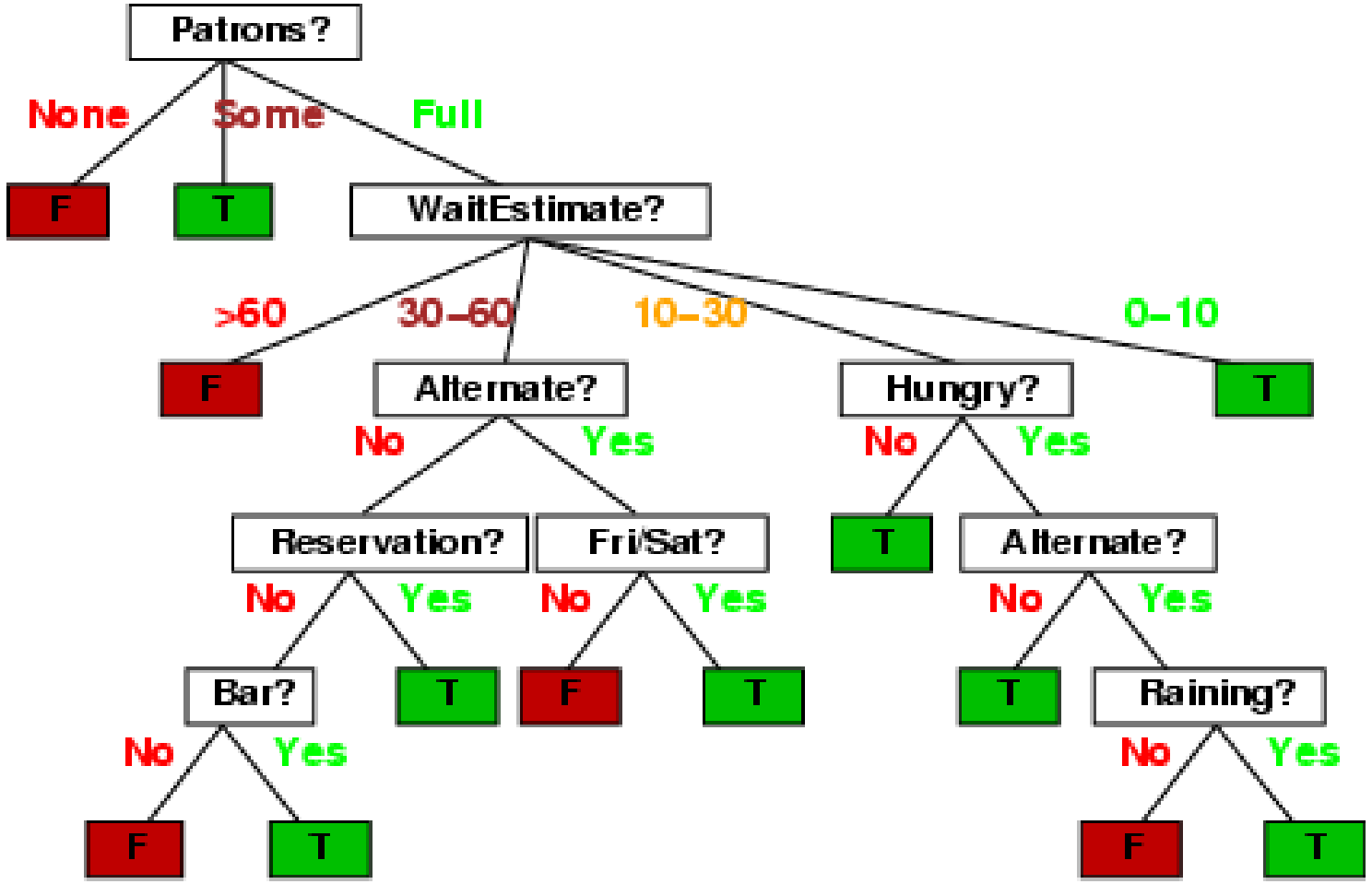
Example: When should I wait for a table at a restaurant?

Attributes (features) relevant to *Wait?* decision:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Example Decision tree

A decision tree for *Wait?* based on personal "rules of thumb":



Input Data for Learning

Past examples when I did/did not wait for a table:

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

Classification of examples is positive (T) or negative (F)

Decision Tree Learning

Aim: find a small tree consistent with training examples

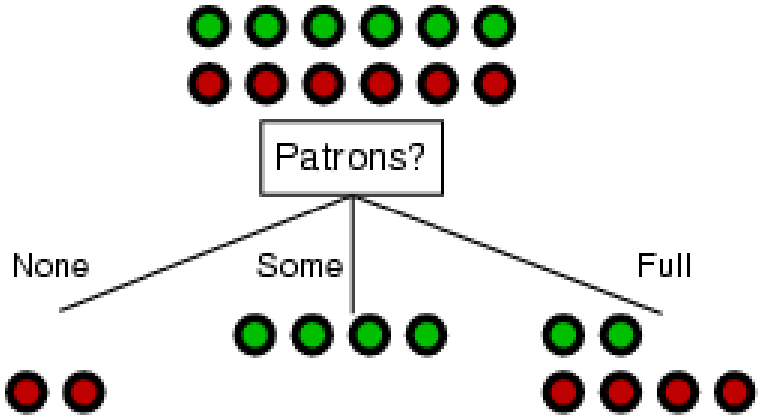
Idea: (recursively) choose "most significant" attribute as root of (sub)tree

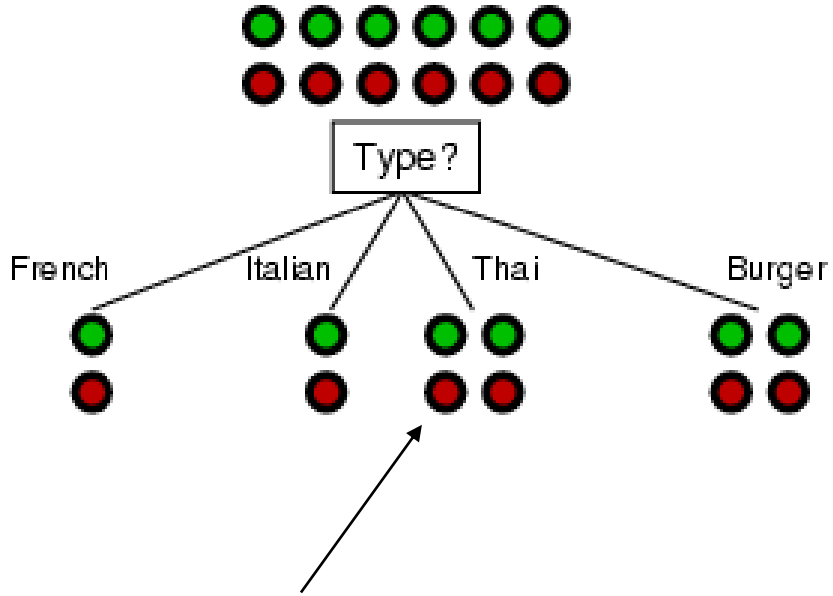
```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Choosing an attribute to split on

Idea: a good attribute should reduce uncertainty

- E.g., splits the examples into subsets that are (ideally) "all positive" or "all negative"

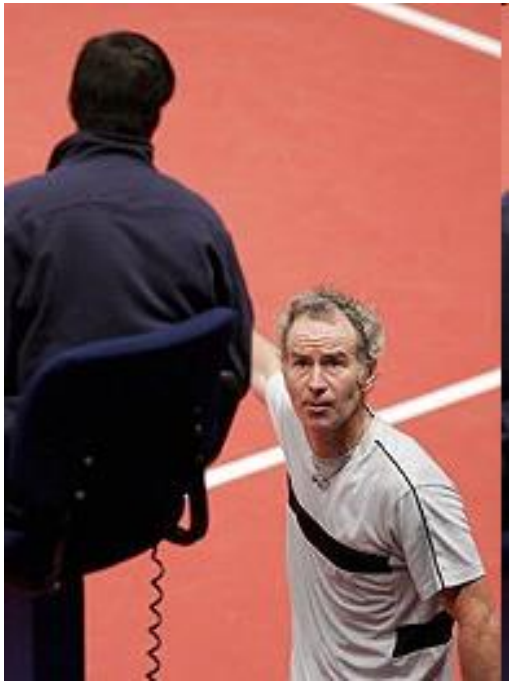




Patrons? is a better choice

For *Type?*, to wait or not to wait is still at 50%

How do we quantify uncertainty?



http://a.espncdn.com/media/ten/2006/0306/photo/g_mcenroe_195.jpg

Using information theory to quantify uncertainty

Entropy measures the amount of uncertainty in a probability distribution

Entropy (or Information Content) of an answer to a question with possible answers $v_1, \ldots, v_n$:

$$I(P(v_1), \ldots, P(v_n)) = \Sigma_{i=1} -P(v_i) \log_2 P(v_i)$$

Using information theory

Imagine we have p examples with Wait = True (positive) and n examples with Wait = false (negative).

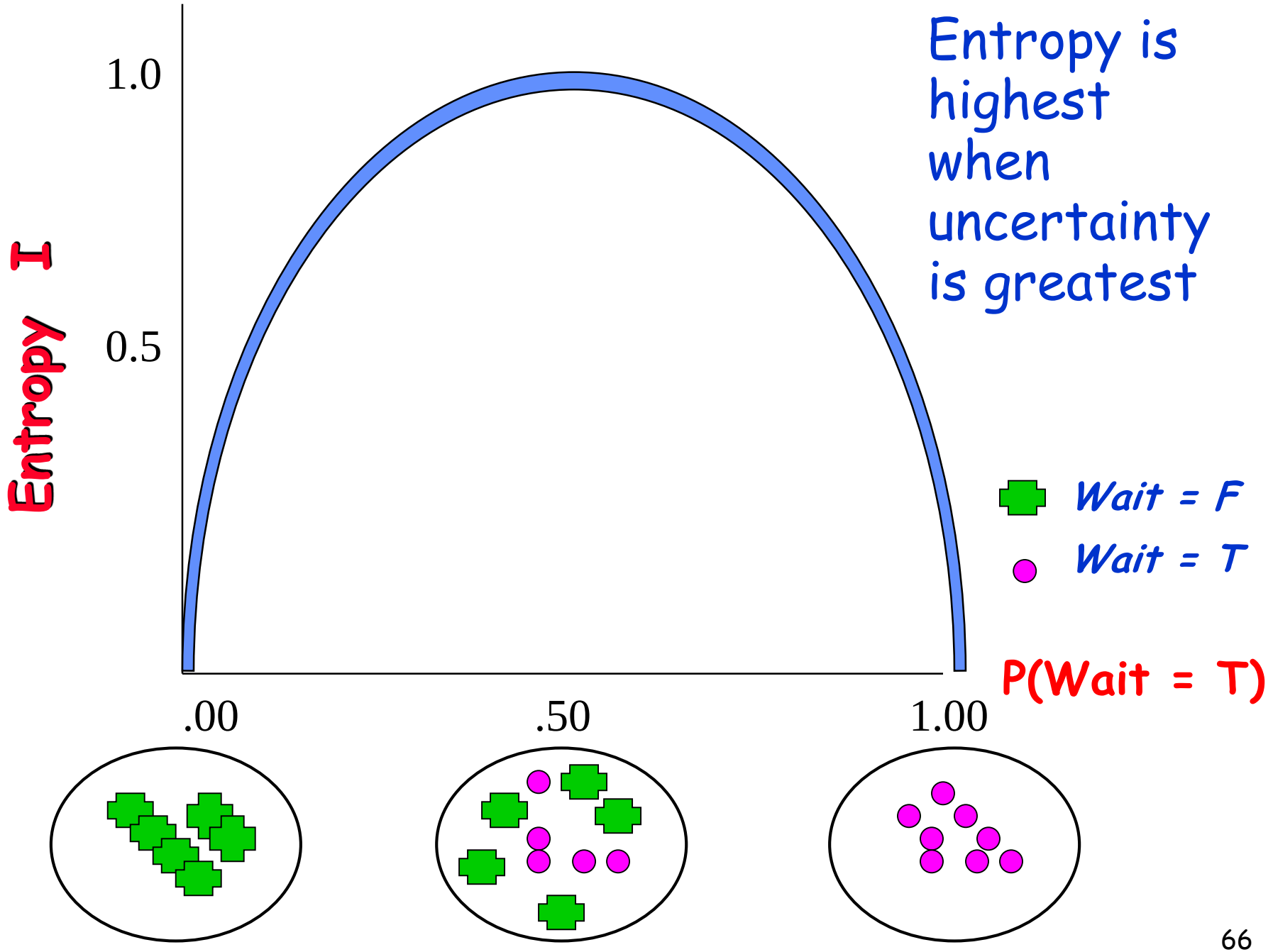
Our best estimate of the probabilities of Wait = true or false is given by:

$$P(true) \approx p / p+n$$

$$p(false) \approx n / p+n$$

Hence the entropy of Wait is given by:

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n}\log_2 \frac{p}{p+n} - \frac{n}{p+n}\log_2 \frac{n}{p+n}$$


Entropy I
1.0
0.5
Entropy is highest when uncertainty is greatest
Wait = F
Wait = T
P(Wait = T)
.00
.50
1.00

Choosing an attribute to split on

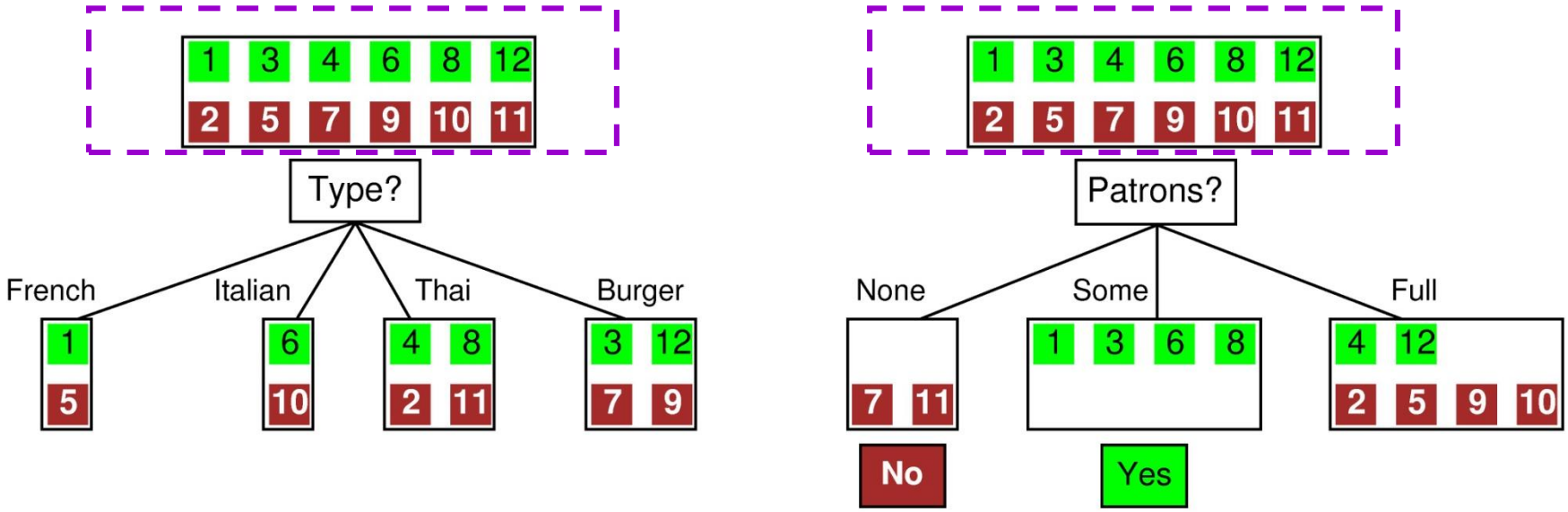
Idea: a good attribute should reduce uncertainty and result in "gain in information"

How much information do we gain if we disclose the value of some attribute?

Answer:

uncertainty before – uncertainty after

Back at the Restaurant



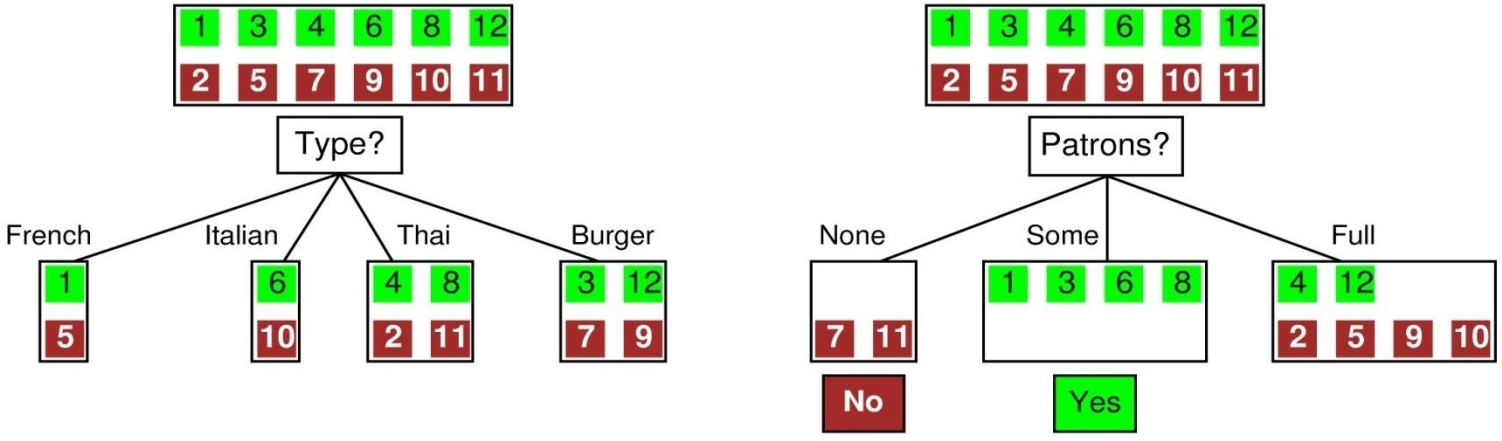
Before choosing an attribute:

Entropy = - 6/12 log(6/12) – 6/12 log(6/12)

= - log(1/2) = log(2) = 1 bit

There is “1 bit of information to be discovered”

Back at the Restaurant



If we choose Type: Go along branch "French": we have entropy = 1 bit; similarly for the others.

Information gain = 1-1 = 0 along any branch

If we choose Patrons:

In branch "None" and "Some", entropy = 0

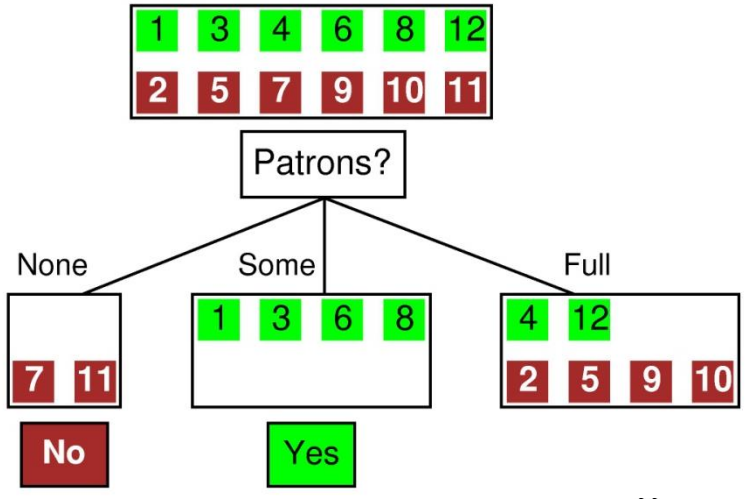
For "Full", entropy = -2/6 log(2/6)-4/6 log(4/6) = 0.92

Info gain = (1-0) or (1-0.92) bits > 0 in both cases

So choosing Patrons gains more information!

Entropy across branches

- How do we combine entropy of different branches?
- Answer: Compute average entropy
- Weight entropies according to probabilities of branches
  - 2/12 times we enter "None", so weight for "None" = 1/6
  - "Some" has weight: 4/12 = 1/3
  - "Full" has weight 6/12 = $\frac{1}{2}$



$$AvgEntropy = \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} Entropy\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

weight for each branch

entropy for each branch

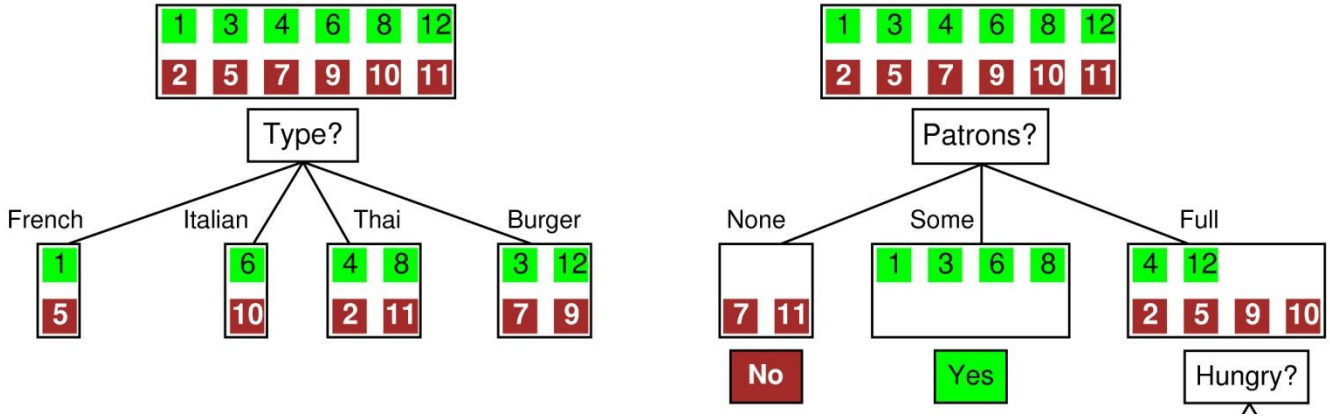
Information gain

Information Gain (IG) or reduction in entropy from using attribute A:

IG(A) = Entropy before – AvgEntropy after choosing A

Choose the attribute with the largest IG

Information gain in our example



$$IG(Patrons) = 1 - [\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})] = .541 \text{ bits}$$

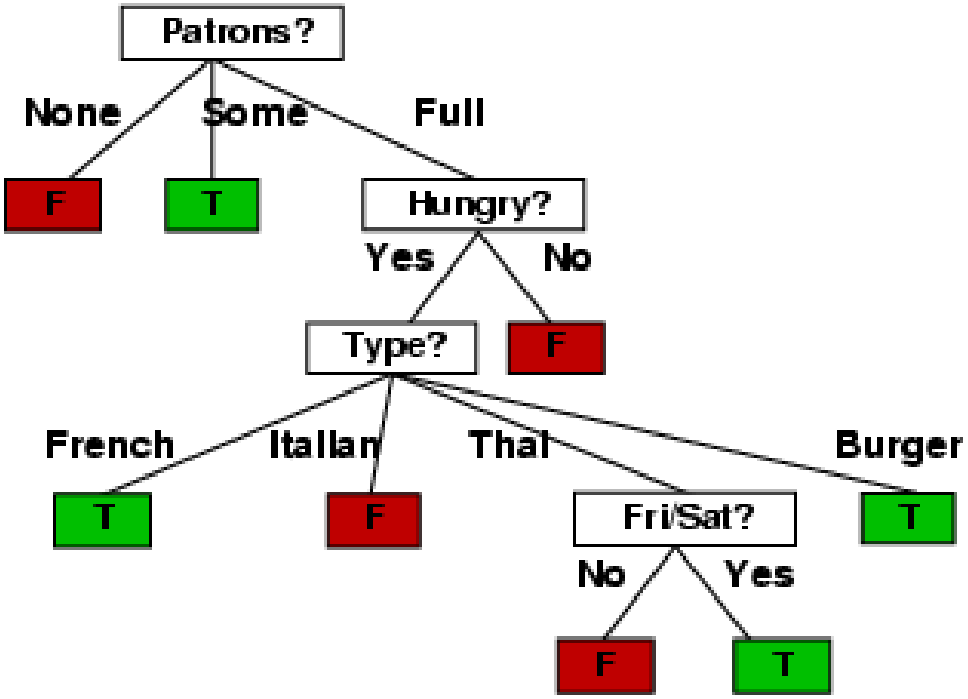
$$IG(Type) = 1 - [\frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4})] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes

$\Rightarrow$ DTL algorithm chooses *Patrons* as the root

Should I stay or should I go?
Learned Decision Tree

Decision tree learned from the 12 examples:



Substantially simpler than "rules-of-thumb" tree

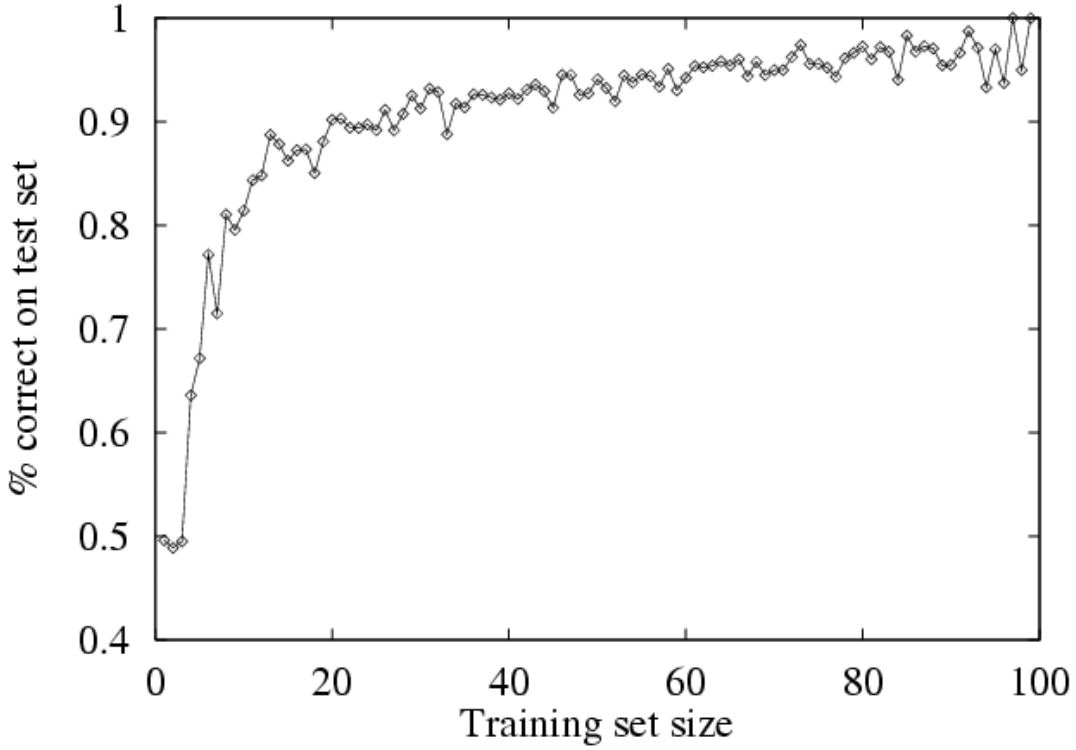
- more complex hypothesis not justified by small amount of data

Performance Evaluation

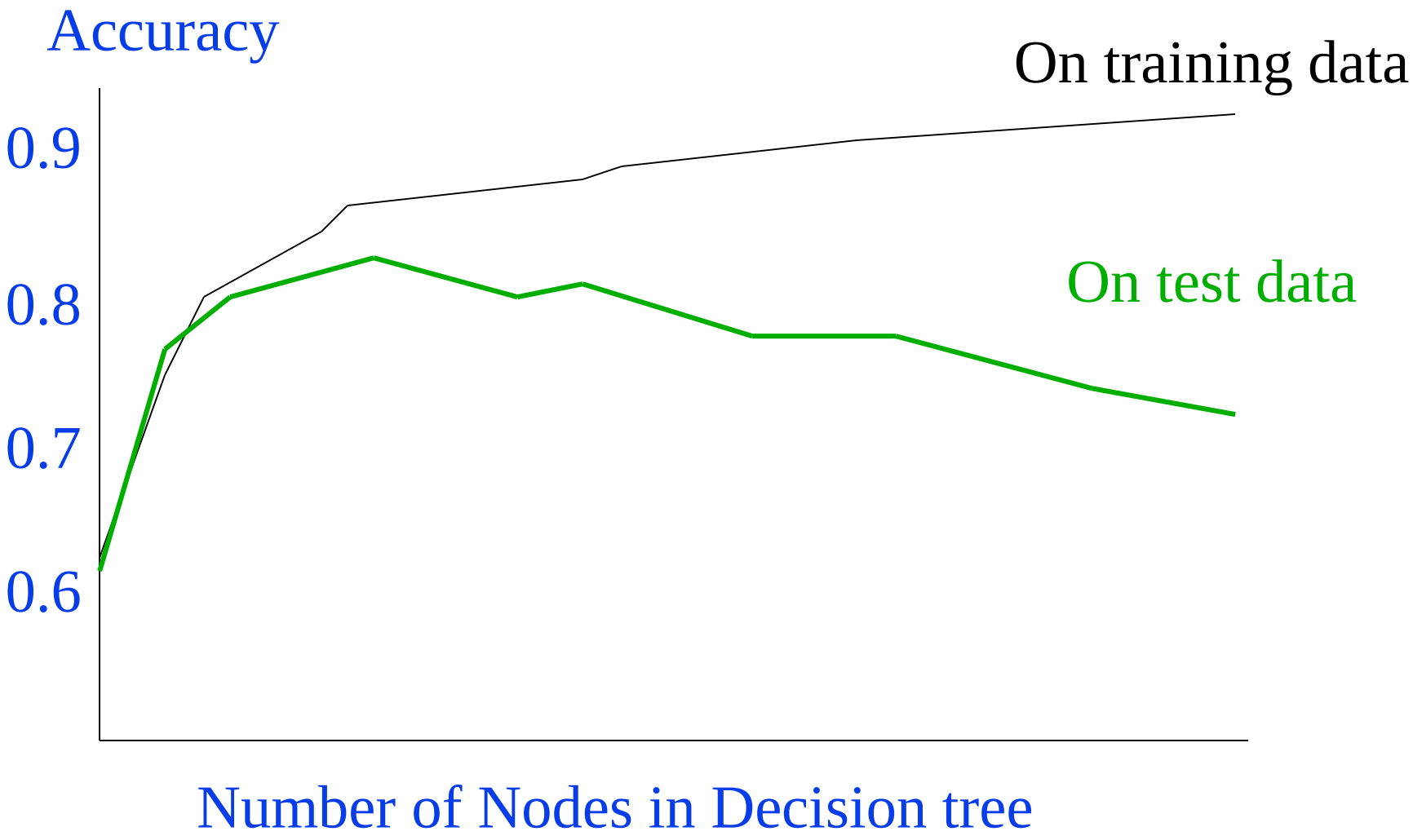
How do we know that the learned tree $h \approx f$?

Answer: Try h on a new test set of examples

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



Overfitting



Overfitting

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

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

Rule #2 of Machine Learning

The *best* hypothesis almost never achieves 100% accuracy on the training data.

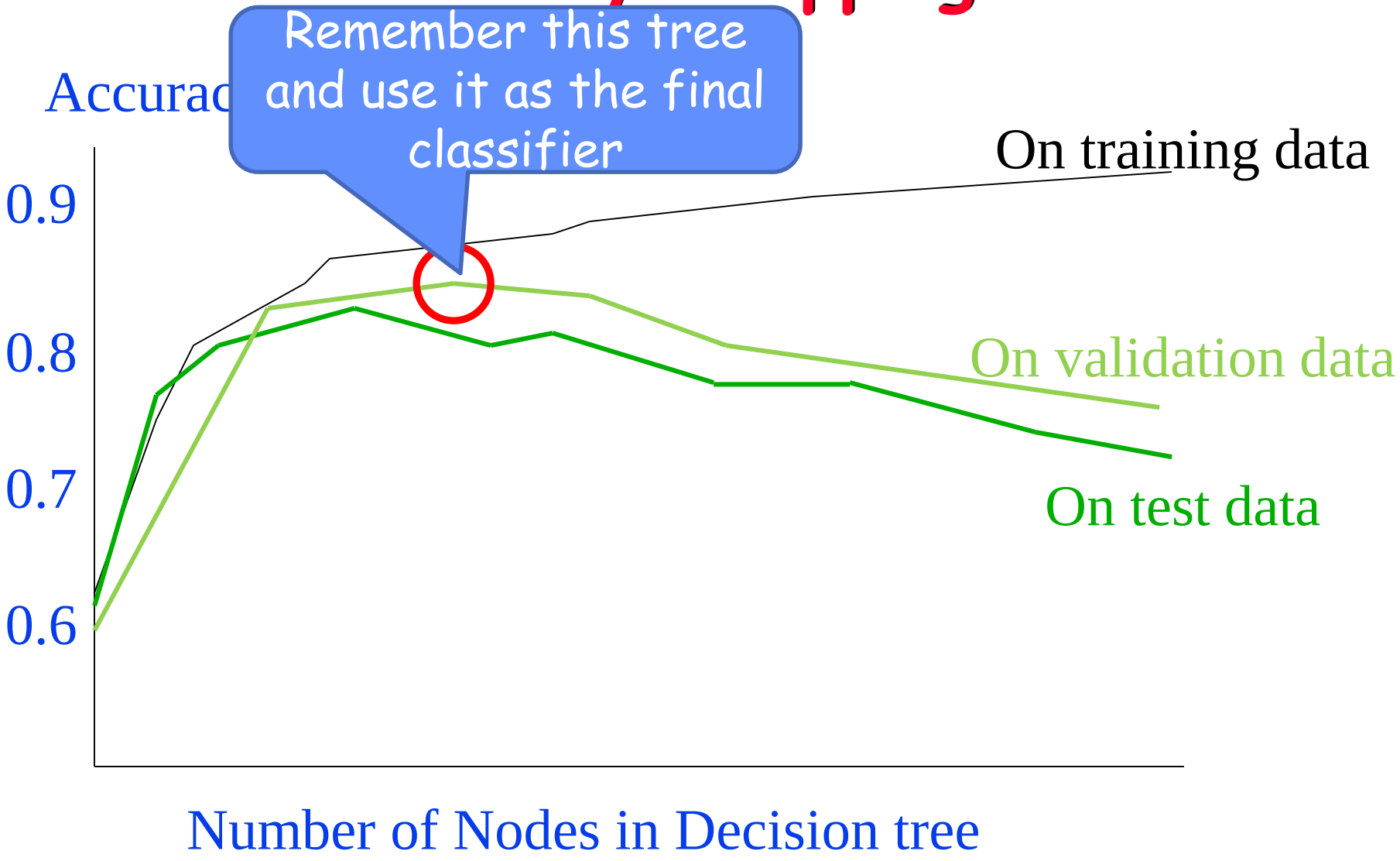
(Rule #1 was: you can't learn anything without inductive bias)

Avoiding overfitting

- Stop growing when data split not statistically significant
- Grow full tree and then prune

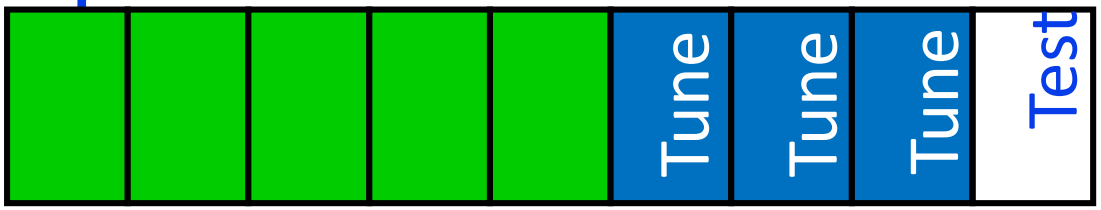
- How to select best tree?
  - Measure performance over the training data
  - Measure performance over separate validation set
  - Add complexity penalty to performance measure

Early Stopping



Reduced Error Pruning

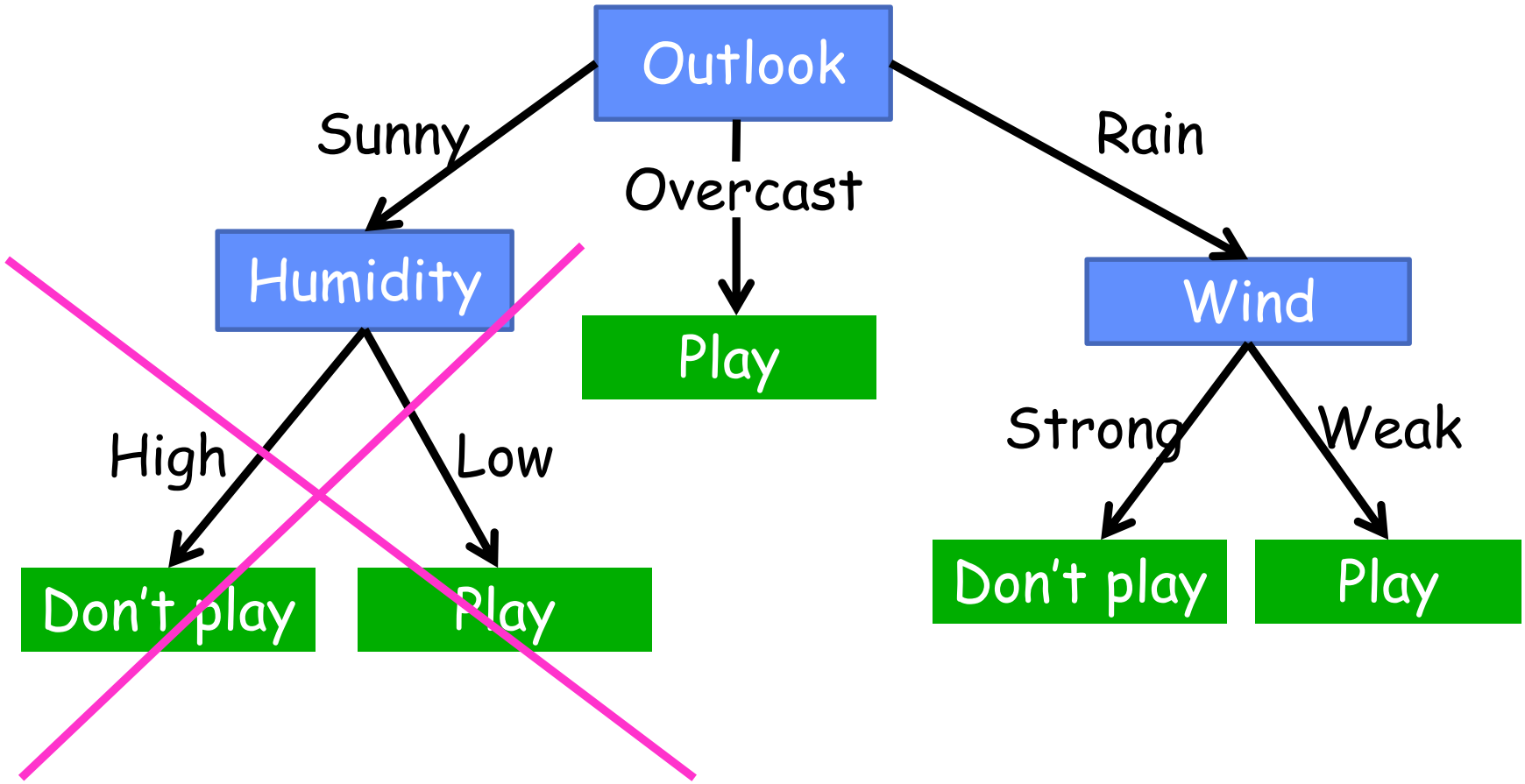
Split data into train and validation set



Repeat until pruning is harmful

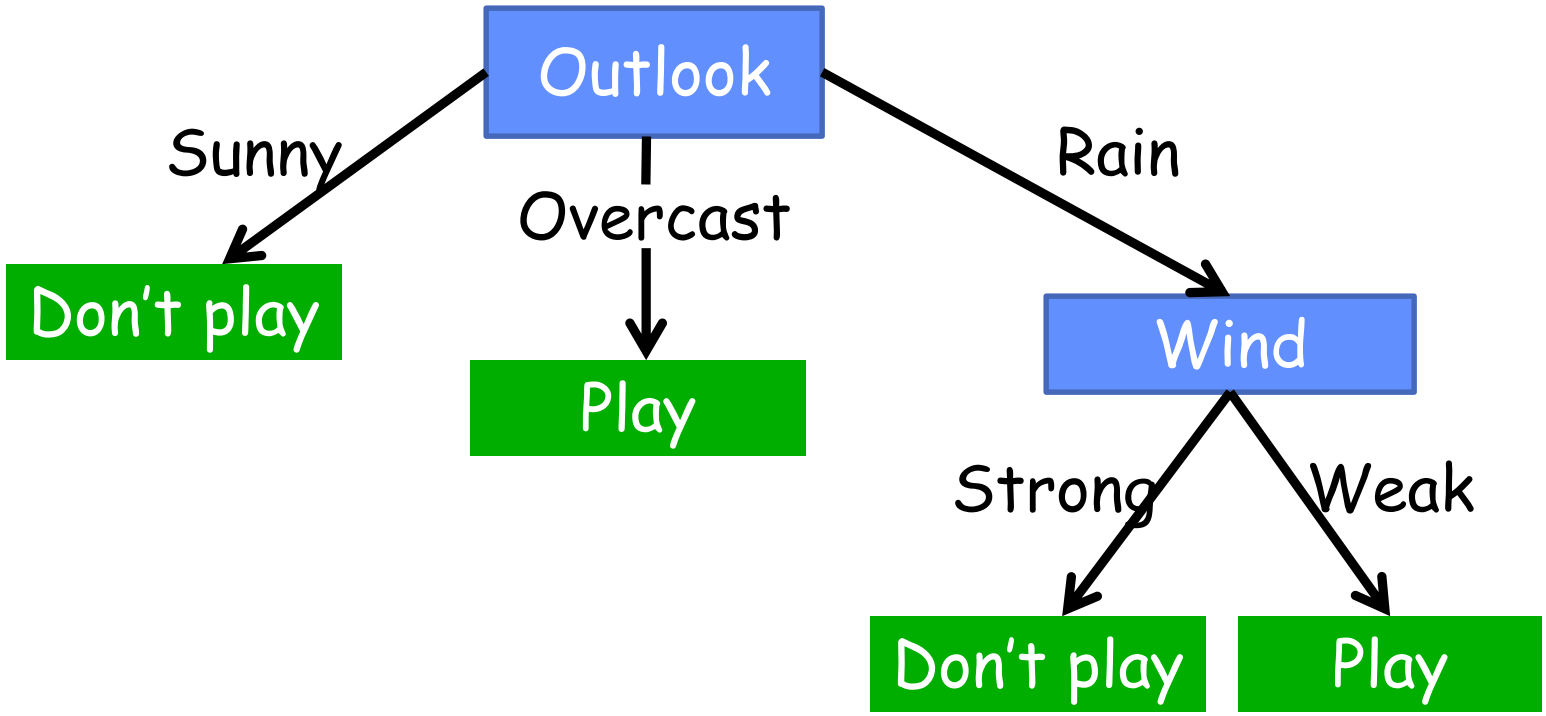
- Remove each subtree and replace it with majority class and evaluate on validation set
- Remove subtree that leads to largest gain in accuracy

Reduced Error Pruning Example



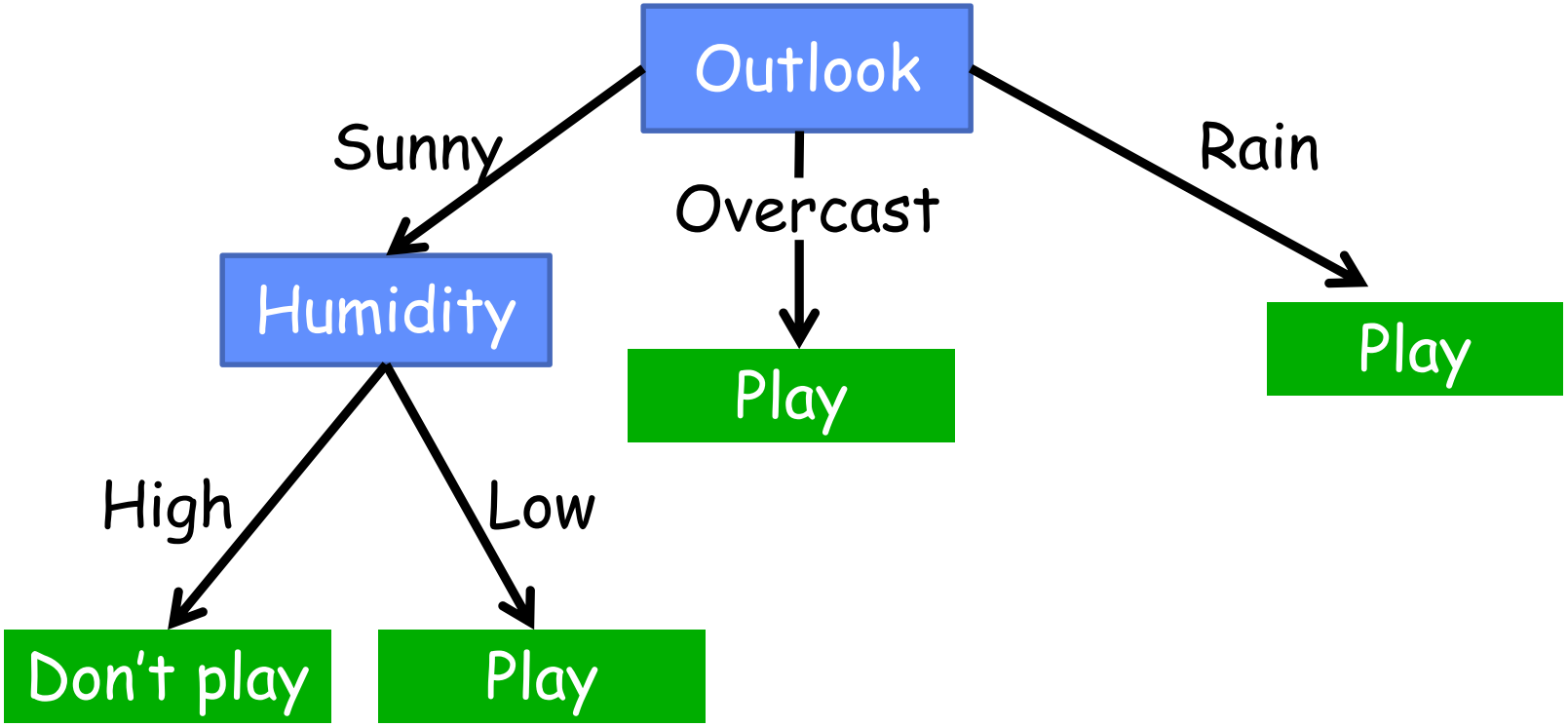
Validation set accuracy = 0.75

Reduced Error Pruning Example



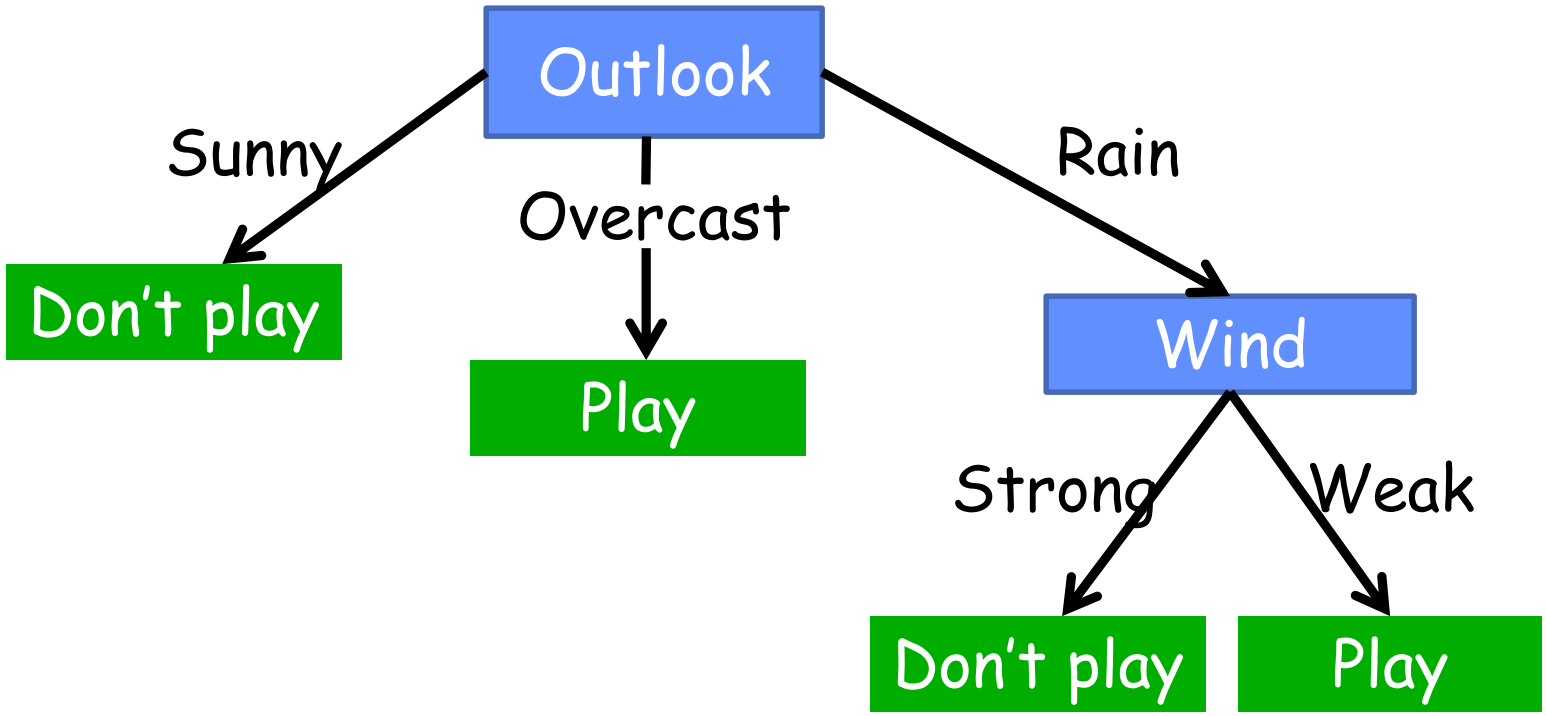
Validation set accuracy = 0.80

Reduced Error Pruning Example



Validation set accuracy = 0.70

Reduced Error Pruning Example

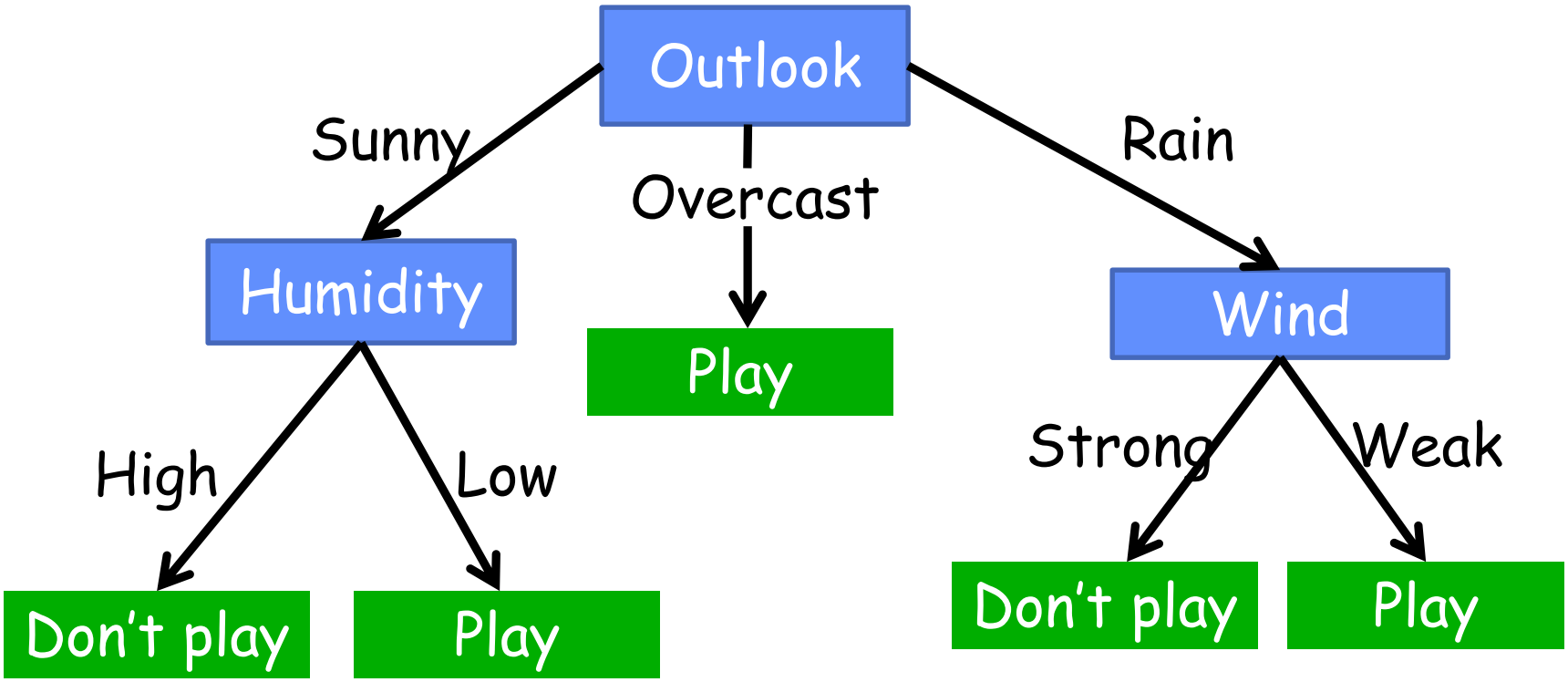


Use this as final tree

Post Rule Pruning

- Split data into train and validation set
- Prune each rule independently
  - Remove each pre-condition and evaluate accuracy
  - Pick pre-condition that leads to largest improvement in accuracy
- Note: ways to do this using training data and statistical tests

Conversion to Rule



Outlook = Sunny ∧ Humidity = High ⇒ Don't play

Outlook = Sunny ∧ Humidity = Low ⇒ Play

Outlook = Overcast ⇒ Play

Scaling Up

- ID3 and C4.5 assume data fits in main memory

  (ok for 100,000s examples)
- SPRINT, SLIQ: multiple sequential scans of data

  (ok for millions of examples)
- VFDT: at most one sequential scan

  (ok for billions of examples)

Decision Trees – Strengths

Very Popular Technique

Fast

Useful when

- Target Function is discrete
- Concepts are likely to be disjunctions
- Attributes may be noisy

Decision Trees – Weaknesses

Less useful for continuous outputs

Can have difficulty with continuous input features as well…

- E.g., what if your target concept is a circle in the x1, x2 plane?
  - Hard to represent with decision trees…
  - Very simple with instance-based methods we'll discuss later…