CS 188: Artificial Intelligence

Learning II: Decision Tree & Logistic Regression

Instructors: Stuart Russell and Dawn Song

University of California, Berkeley

Recap: Supervised learning

- To learn an unknown ***target function*** f
- Input: a ***training set*** of ***labeled examples*** (x_j, y_j) where $y_j = f(x_j)$
    - E.g., x_j is an image, $f(x_j)$ is the label “giraffe”
    - E.g., x_j is a seismic signal, $f(x_j)$ is the label “explosion”
- Output: ***hypothesis*** h that is “close” to f, i.e., predicts well on unseen examples (“***test set***”)
- Many possible hypothesis families for h
    - Linear models, logistic regression, neural networks, decision trees, examples (nearest-neighbor), grammars, kernelized separators, etc etc
- ***Classification*** = learning f with discrete output value
- ***Regression*** = learning f with real-valued output value

Training data

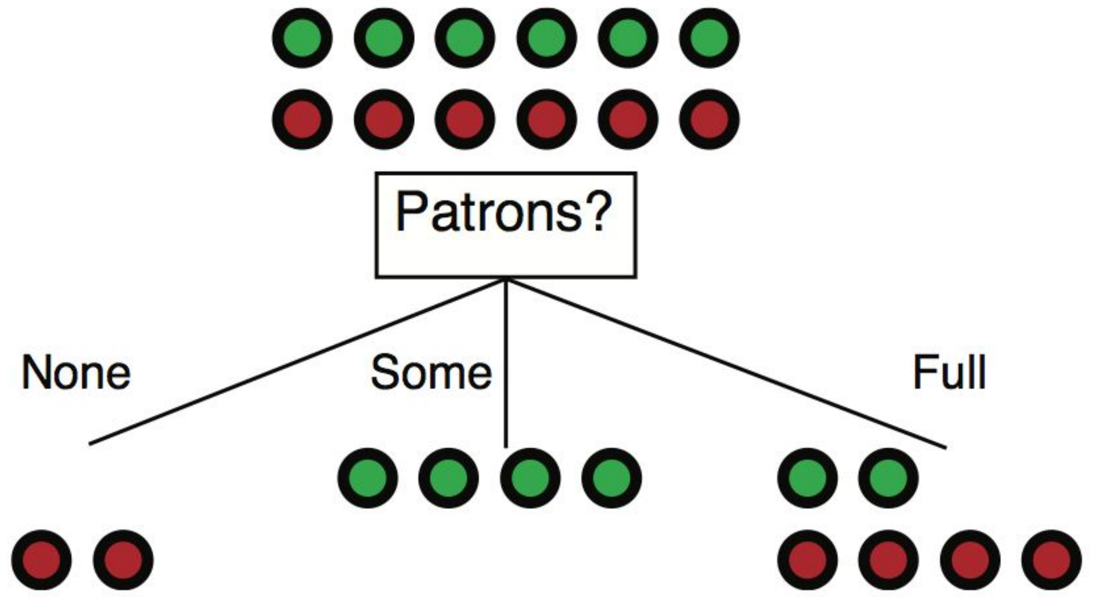
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 =$ *Yes*
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 =$ *No*
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 =$ *Yes*
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 =$ *Yes*
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 =$ *No*
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 =$ *Yes*
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 =$ *No*
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 =$ *Yes*
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 =$ *No*
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} =$ *No*
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} =$ *No*
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} =$ *Yes*

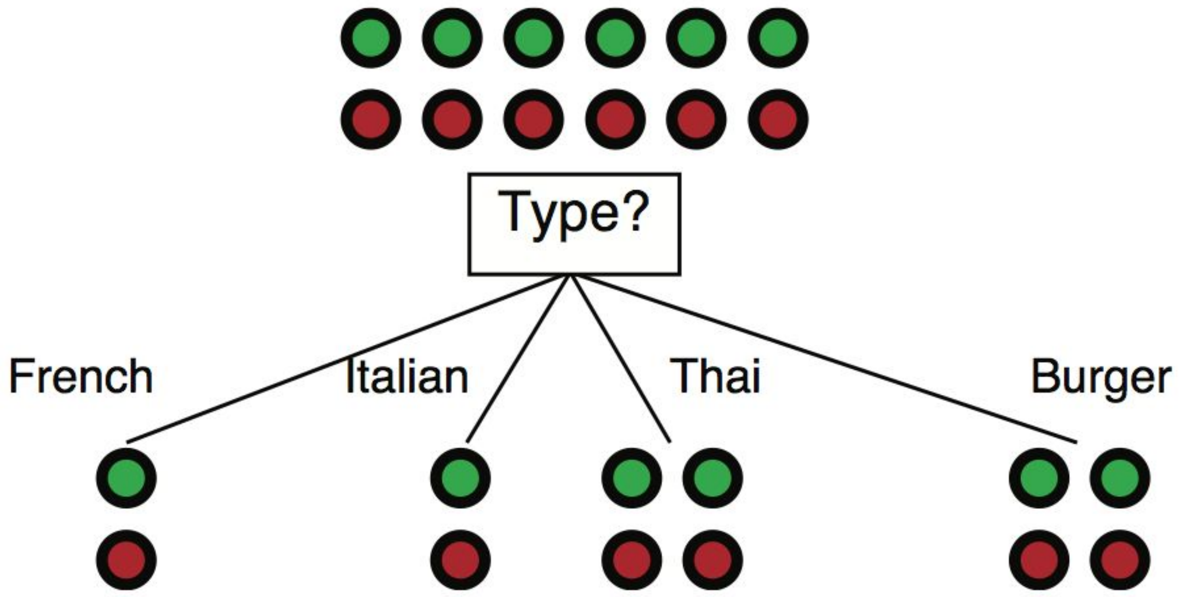
Learning a Decision Tree

- Goal
  - Given training data of a list of attributes & label
  - Hypothesis family H: decision trees
  - Find (approximately) the smallest decision tree that fits the training data
- Iterative process to choose the next attribute to split on

Choosing an attribute: Information gain

- Idea: measure contribution of attribute to increasing "purity" of labels in each subset of examples; find most distinguishing feature

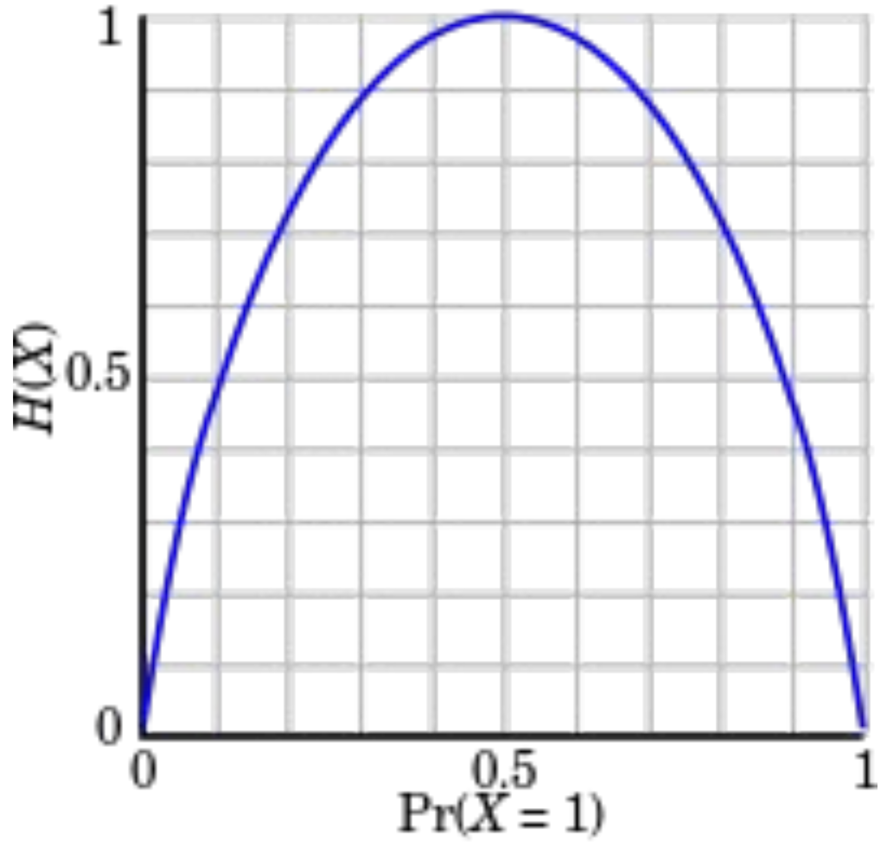




- Patrons is a better choice: gives ***information*** about classification; more distinguishing feature

Information

- Information answers questions
- The more clueless I am about the answer initially, the more information is contained in the answer
- Scale: 1 bit = answer to Boolean question with prior $\langle 0.5, 0.5\rangle$
- For a coin with probability p heads, entropy $H(\langle p,1-p\rangle) = -p\log p - (1-p)\log(1-p)$
- Convenient notation: $B(p) = H(\langle p,1-p\rangle)$


1
0.5
0
H(X)
0
0.5
1
Pr(X = 1)

Information

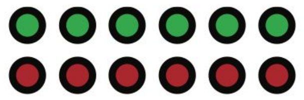
- Information in an answer when prior is $\langle p_1,\ldots,p_n \rangle$ is

  $$H(\langle p_1,\ldots,p_n \rangle) = \Sigma_i -p_i \log p_i$$

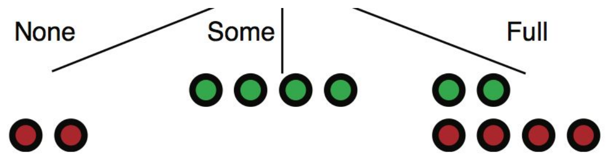
  - This is the ***entropy*** of the prior
- Entropy was initially proposed in physics (thermodynamics)
- Shannon developed information theory, using entropy to measure information

Information gain from splitting on an attribute

- Suppose we have p positive and n negative examples at the root
  - => ______________ bits needed to classify a new example
  - E.g., for 12 restaurant examples, p = n = 6 so we need _____ bit



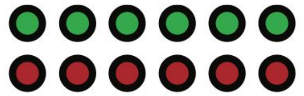
- An attribute splits the examples E into subsets E_k, each of which (we hope) needs less information to complete the classification
  - For an example in E_k we expect to need _____________ more bits
  - Probability a new example goes into E_k is ________________



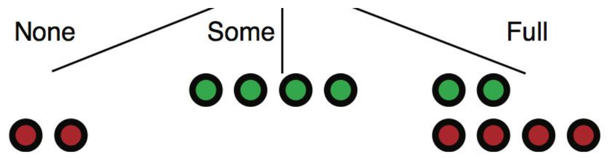
  - *Expected* number of bits needed after split is ________________________
  - Information gain =________________________

Information gain from splitting on an attribute

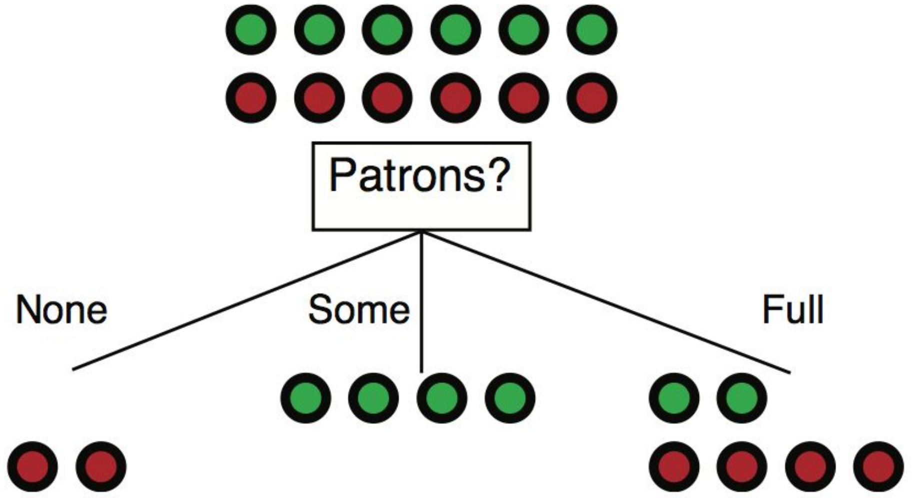
- Suppose we have p positive and n negative examples at the root
  - => $B(p/(p+n))$ bits needed to classify a new example
  - E.g., for 12 restaurant examples, $p = n = 6$ so we need 1 bit

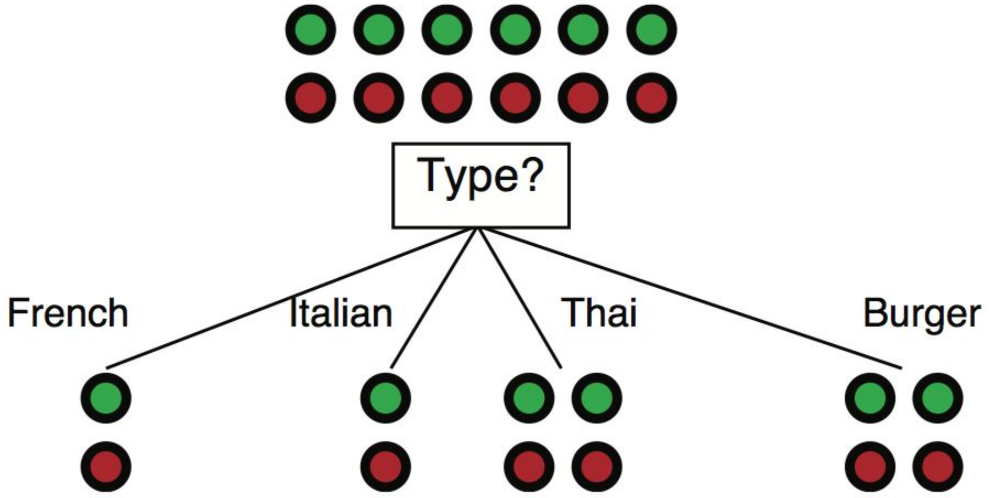


- An attribute splits the examples E into subsets E_k, each of which (we hope) needs less information to complete the classification
  - For an example in E_k we expect to need $B(p_k/(p_k+n_k))$ more bits
  - Probability a new example goes into E_k is $(p_k+n_k)/(p+n)$
  - *Expected* number of bits needed after split is $\sum_k (p_k+n_k)/(p+n)\, B(p_k/(p_k+n_k))$
  - Information gain = $B(p/(p+n)) - \Sigma_k (p_k+n_k)/(p+n)\, B(p_k/(p_k+n_k))$



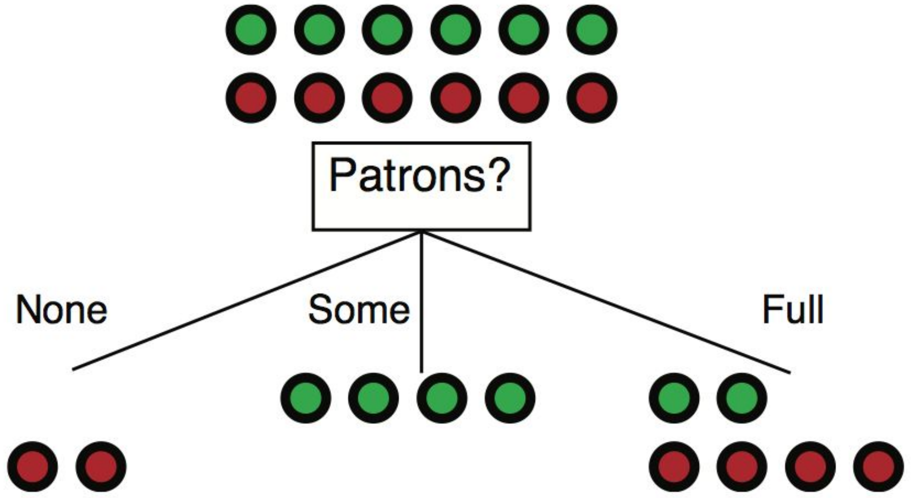
Example

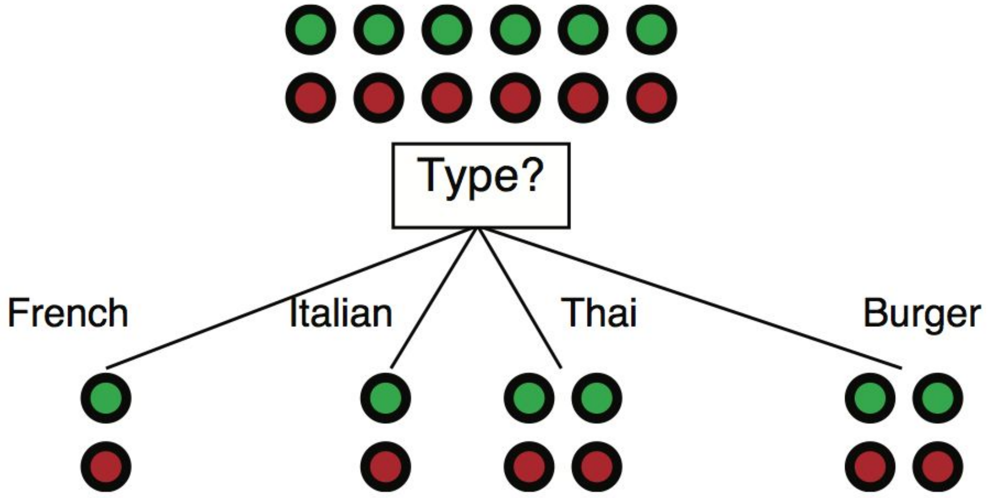




Information gain?

Example





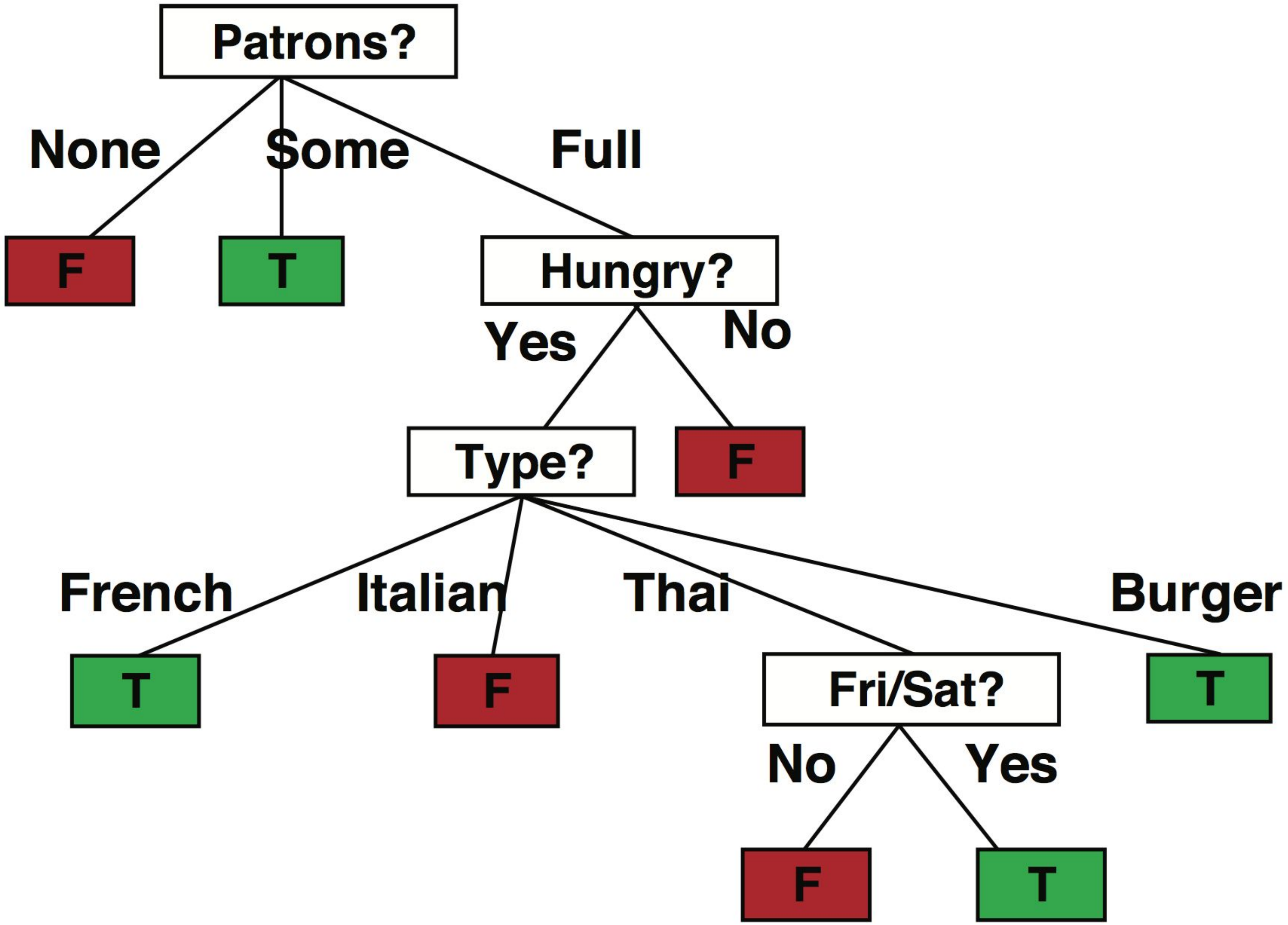
1 – [(2/12)B(0) + (4/12)B(1) + (6/12)B(2/6)]
= 0.541 bits

1 – [(2/12)B(1/2) + (2/12)B(1/2) +
(4/12)B(2/6) + (4/12)B(1/2)]
= 0 bits

Results for restaurant data

Decision tree learned from the 12 examples:

- Simpler than "true" tree!

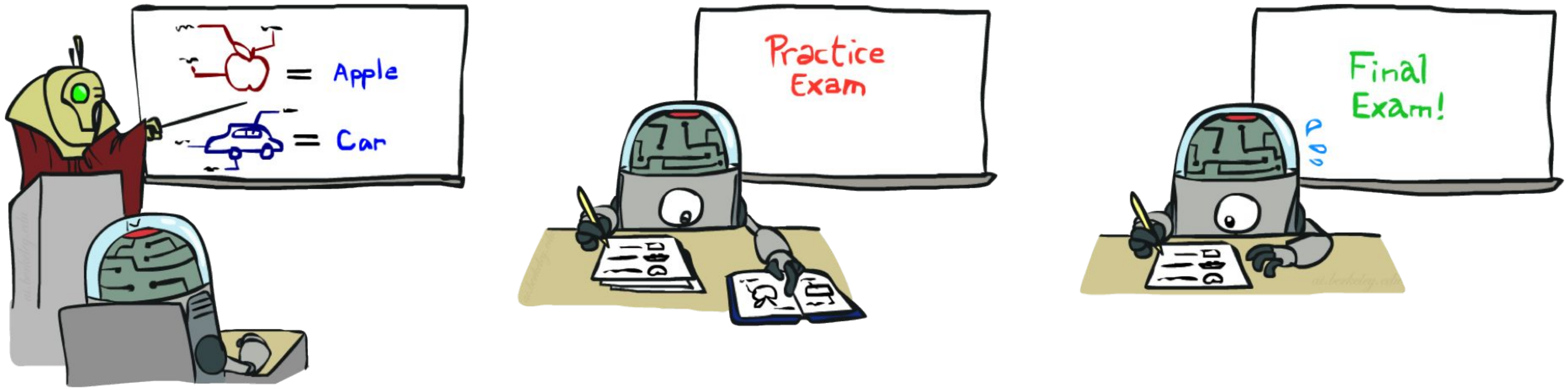


Decision tree learning

```
function Decision-Tree-Learning(examples,attributes,parent_examples) returns a tree
if examples is empty then return Plurality-Value(parent_examples)
else if all examples have the same classification then return the classification
else if attributes is empty then return Plurality-Value(examples)
else A ← argmax_{a ∈ attributes} Importance(a, examples)
     tree ← a new decision tree with root test A
     for each value v of A do
          exs ← the subset of examples with value v for attribute A
          subtree ← Decision-Tree-Learning(exs, attributes − A, examples)
          add a branch to tree with label (A = v_k) and subtree subtree
return tree
```

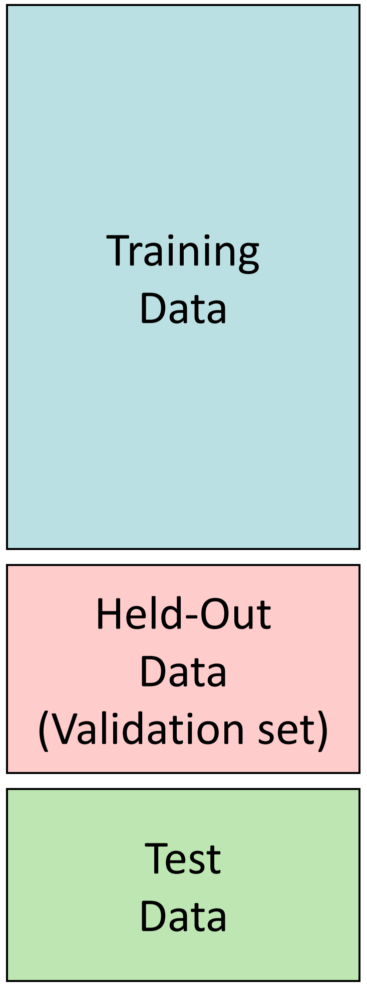
Plurality-Value selects the most common output value among a set of examples

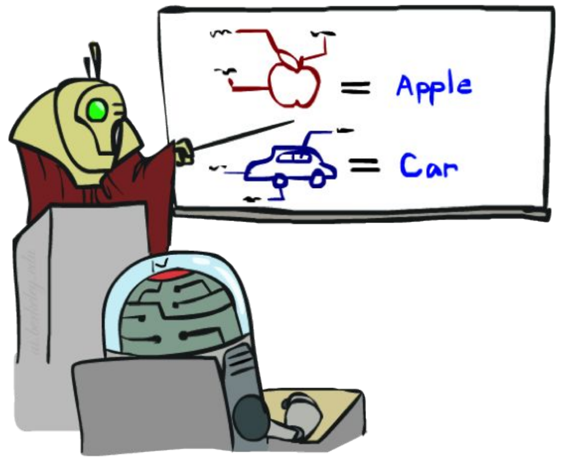
Training and Testing



A few important points about learning

- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held out set
  - Test set
- Features: attribute-value pairs which characterize each x
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - (Tune hyperparameters on held-out set)
  - Compute accuracy of test set
  - Very important: never “peek” at the test set!
- Evaluation
  - Accuracy: fraction of instances predicted correctly
- Overfitting and generalization
  - Want a classifier which does well on *test* data
  - Overfitting: fitting the training data very closely, but not generalizing well
  - Underfitting: fits the training set poorly

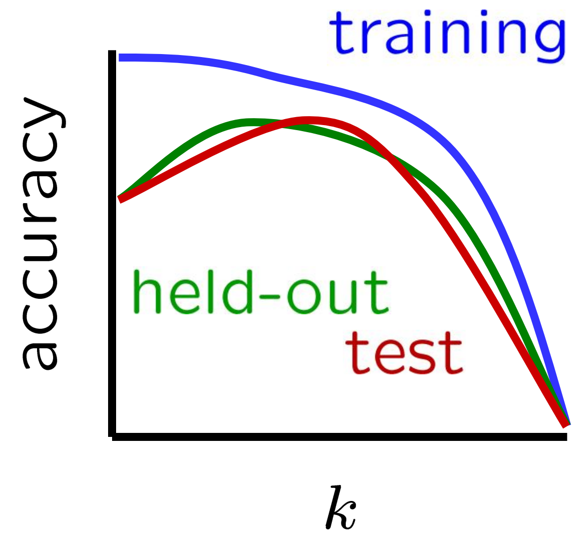


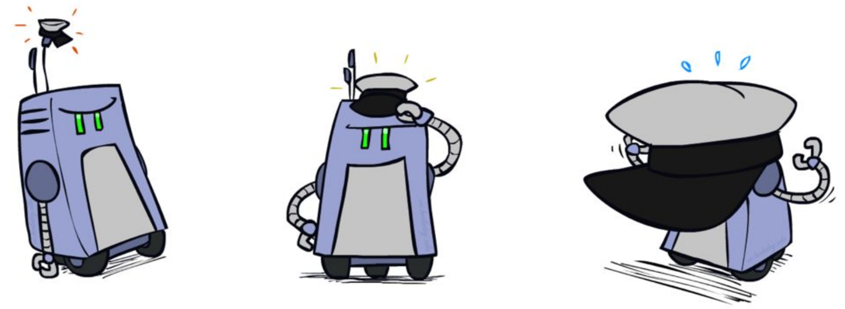


A few important points about learning

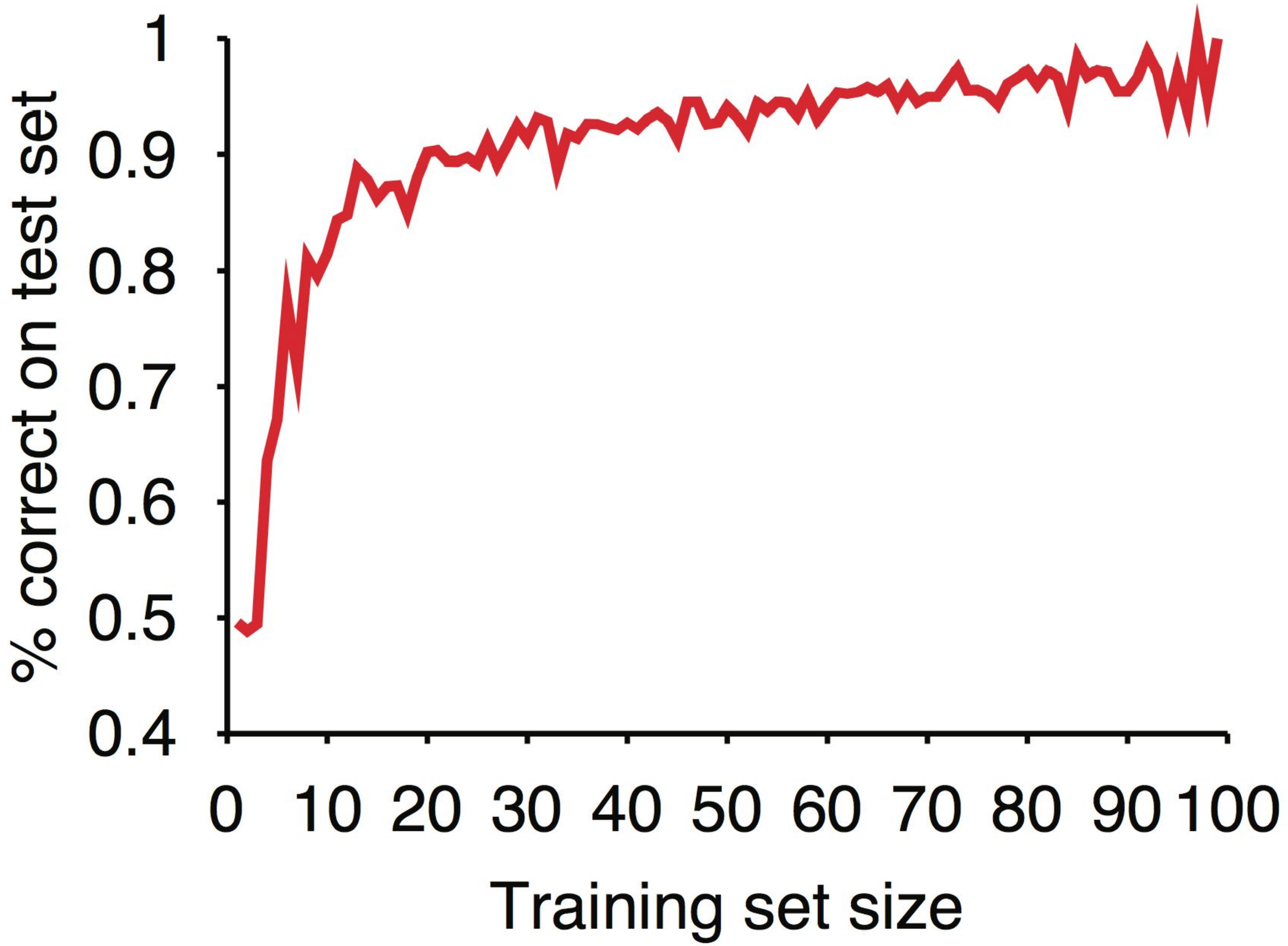
- What should we learn where?
  - Learn parameters from training data
  - Tune hyperparameters on different data
    - Why?
  - For each value of the hyperparameters, train and test on the held-out data
  - Choose the best value and do a final test on the test data

- What are examples of hyperparameters?

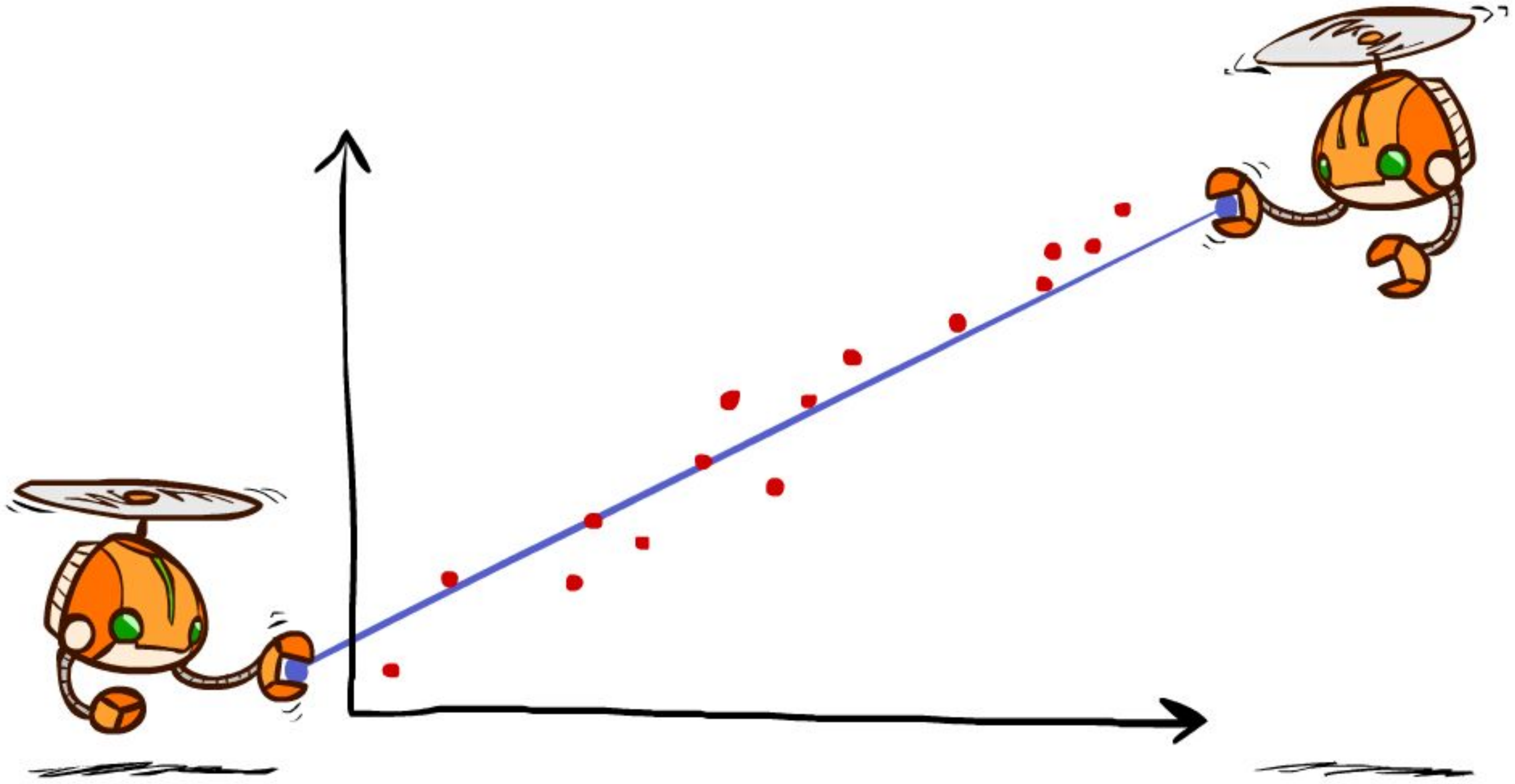




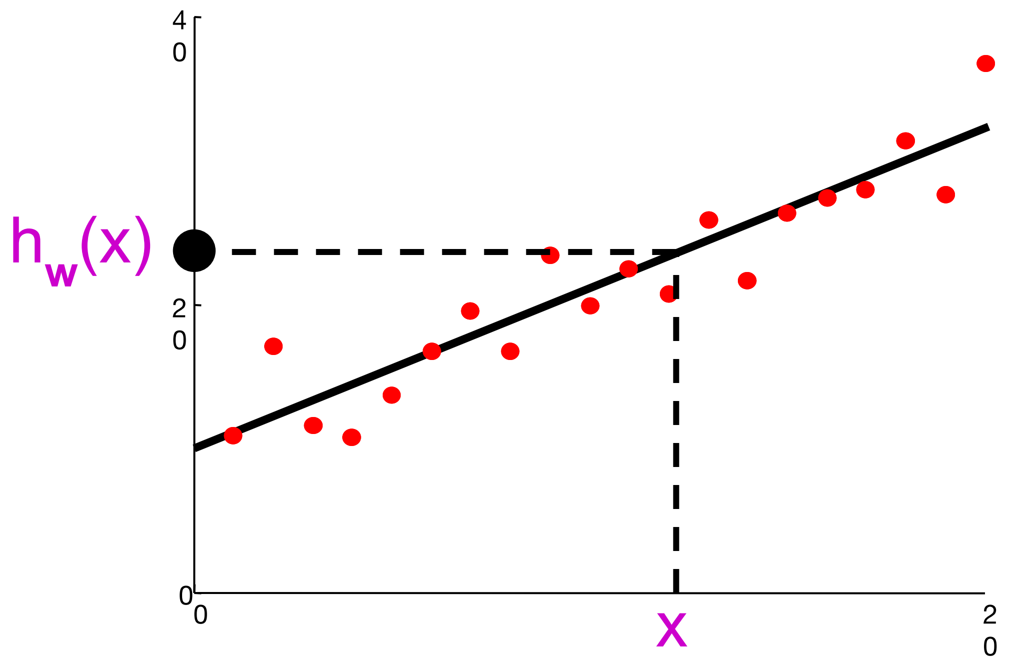
Results for restaurant data



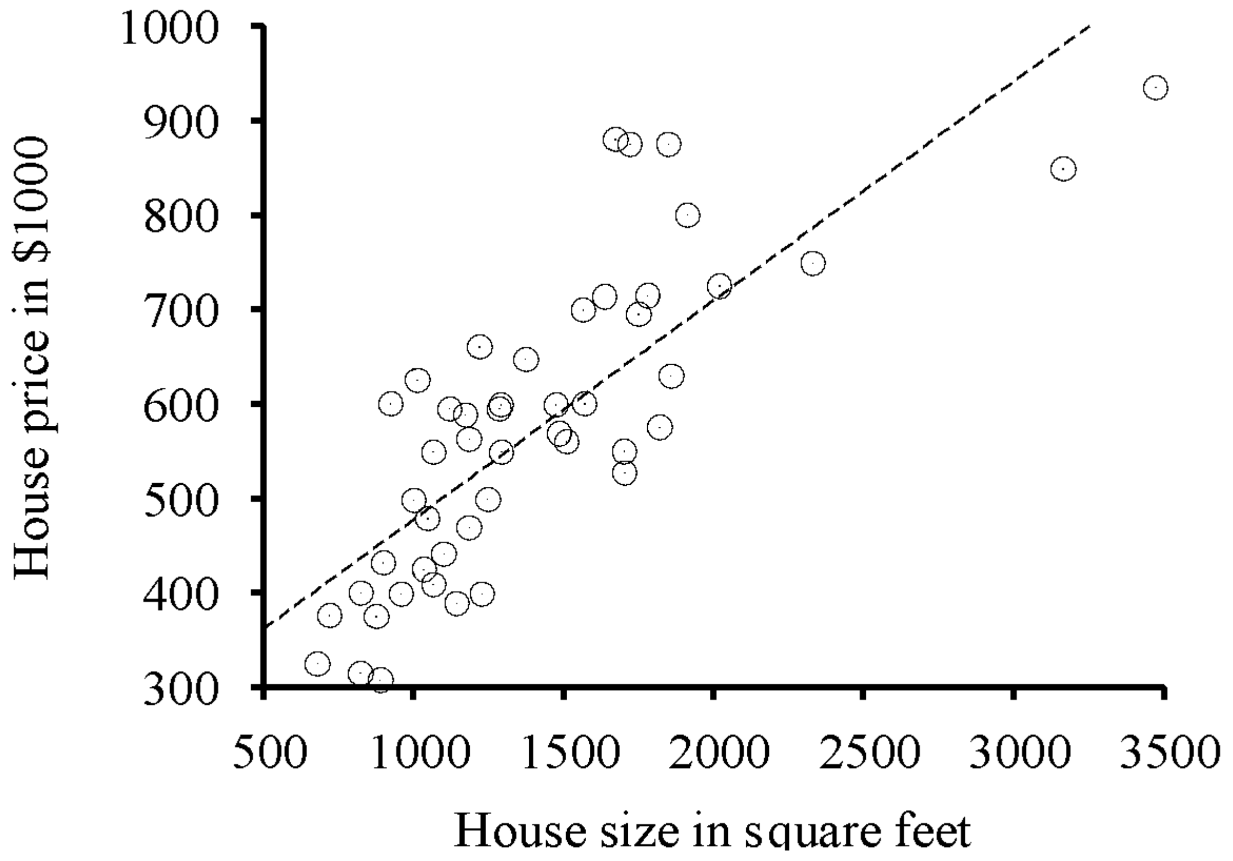
Linear Regression



Linear regression = fitting a straight line/hyperplane



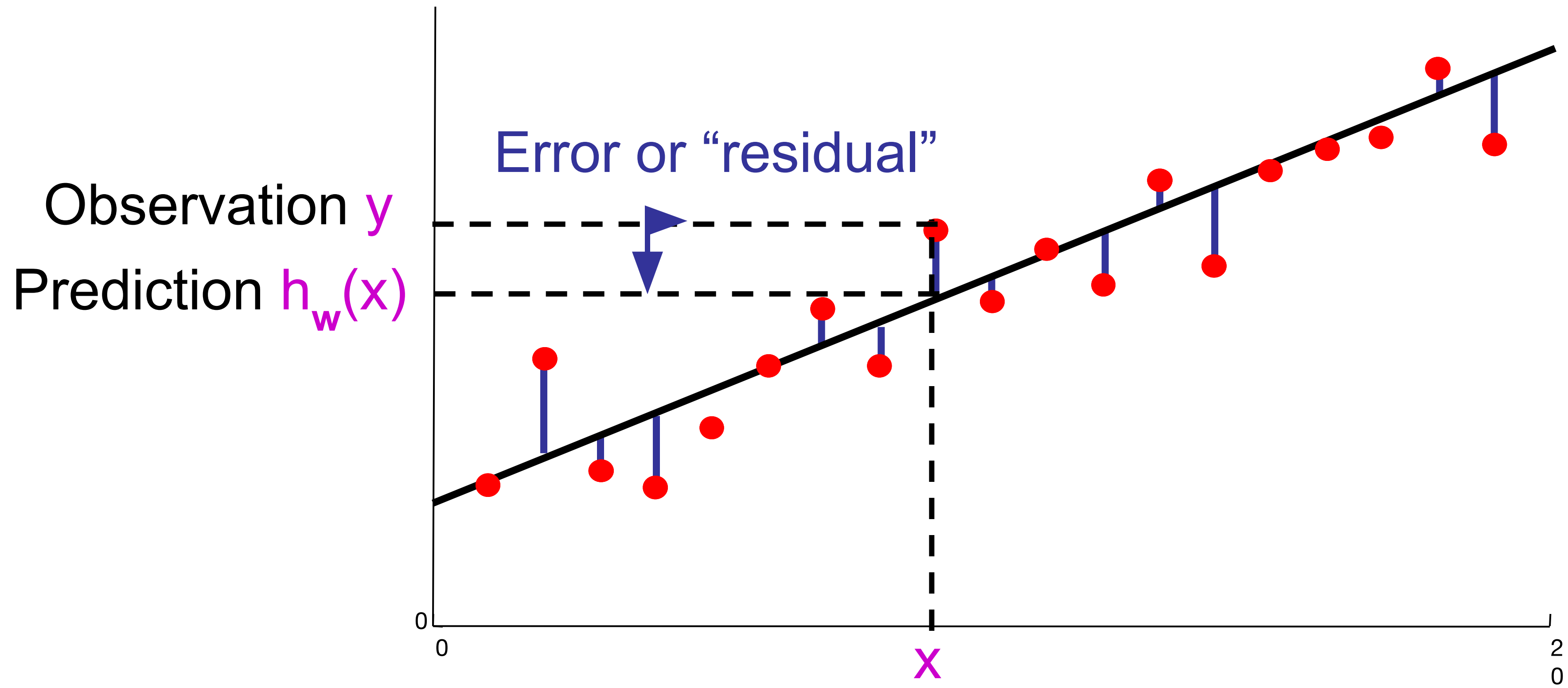
Prediction: $h_w(x) = w_0 + w_1x$



Berkeley house prices, 2009

Prediction error

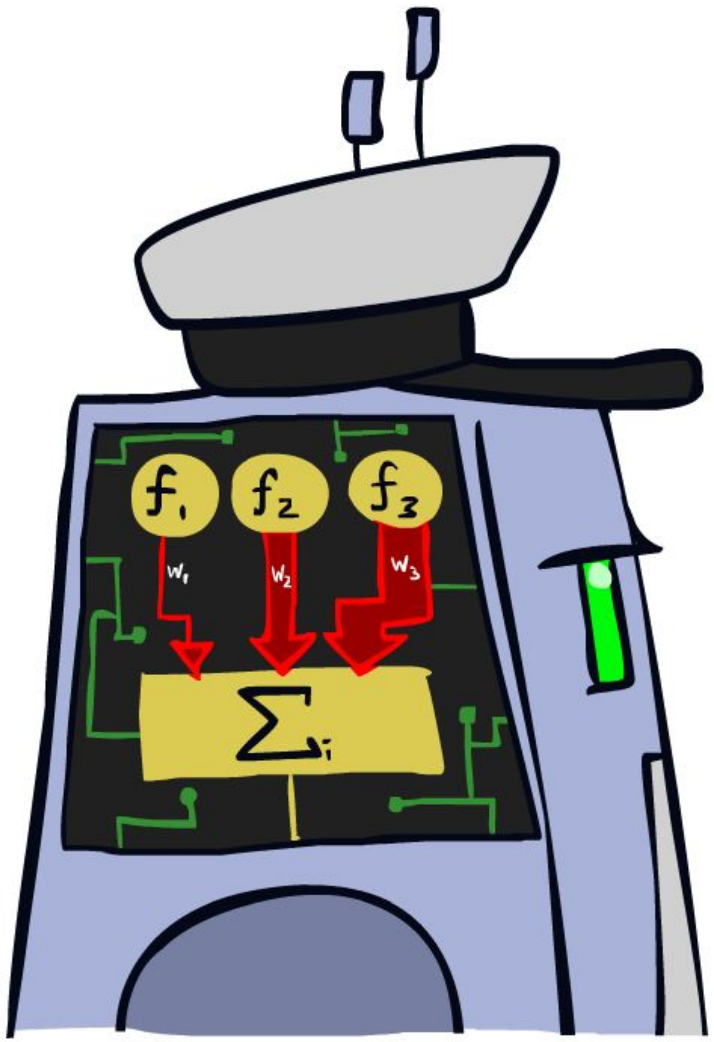
Error on one instance: $y - h_w(x)$



Least squares: Minimizing squared error

- L2 loss function: sum of squared errors over all examples
  - $Loss = \Sigma_j (y_j - h_{\mathbf{w}}(x_j))^2 = \Sigma_j (y_j - (w_0 + w_1 x_j))^2$
- We want the weights w^* that minimize loss
- At w^* the derivatives of loss w.r.t. each weight are zero:
  - $\partial Loss/\partial w_0 = -2 \ \Sigma_j (y_j - (w_0 + w_1 x_j)) = 0$
  - $\partial Loss/\partial w_1 = -2 \ \Sigma_j (y_j - (w_0 + w_1 x_j)) \, x_j = 0$
- Exact solutions for N examples:
  - $w_1 = [N\Sigma_j x_j y_j - (\Sigma_j x_j)(\Sigma_j y_j)]/[N\Sigma_j x_j^2 - (\Sigma_j x_j)^2]$ and $w_0 = 1/N \, [\Sigma_j y_j - w_1 \Sigma_j x_j]$
- For the general case where x is an n-dimensional vector
  - $\mathbf{X}$ is the data matrix (all the data, one example per row); $\mathbf{y}$ is the column of labels
  - $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

Linear Classifiers



Feature Vectors

x $\rightarrow$ $f(x)$ $\rightarrow$ y

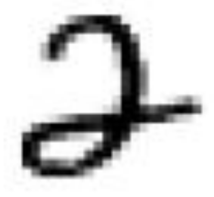
```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```

$\Rightarrow$

```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

$\Rightarrow$

SPAM
or
+



$\Rightarrow$

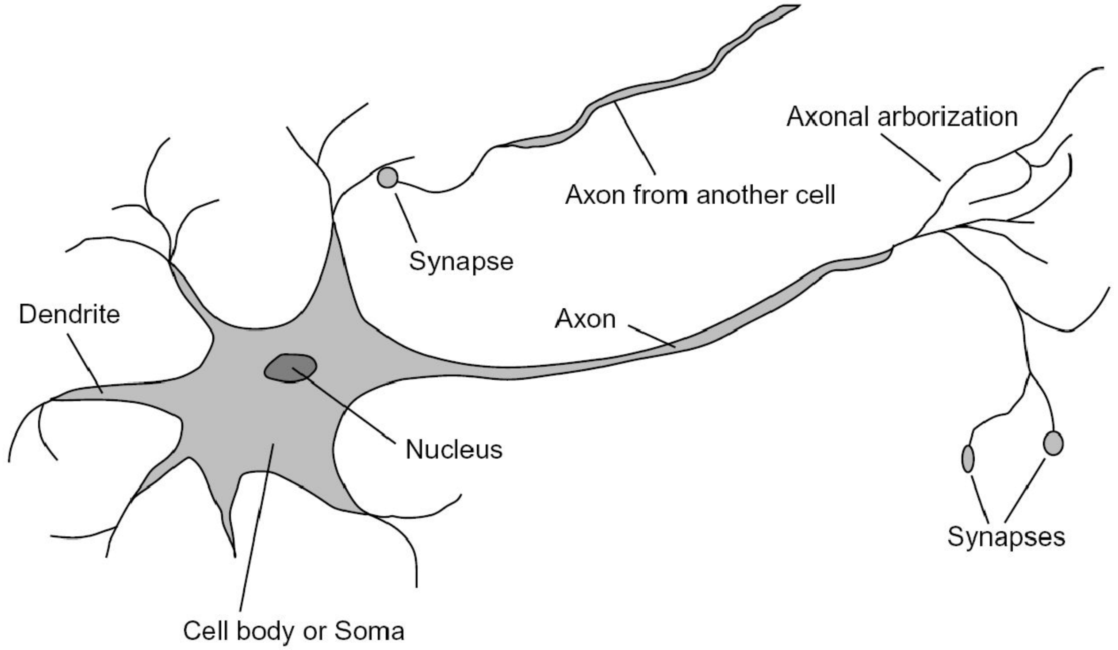
```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

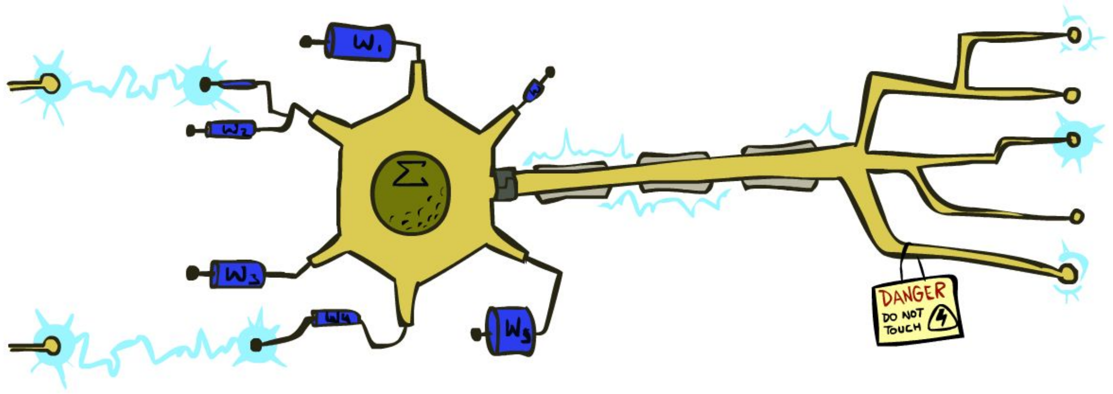
$\Rightarrow$

“2”

Some (Simplified) Biology

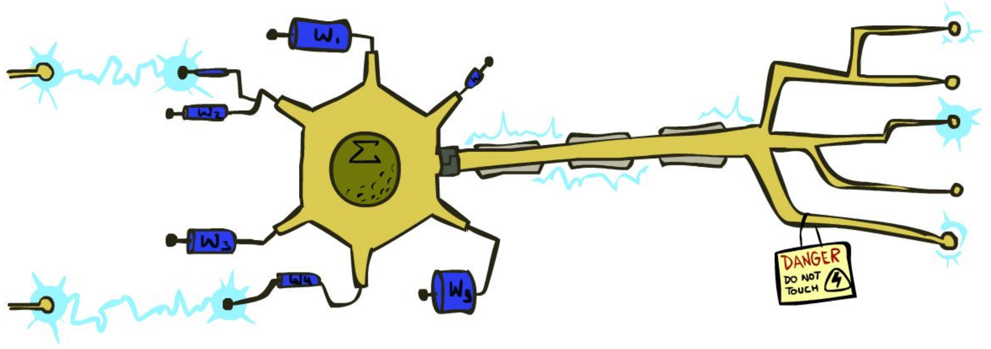
- Very loose inspiration: human neurons





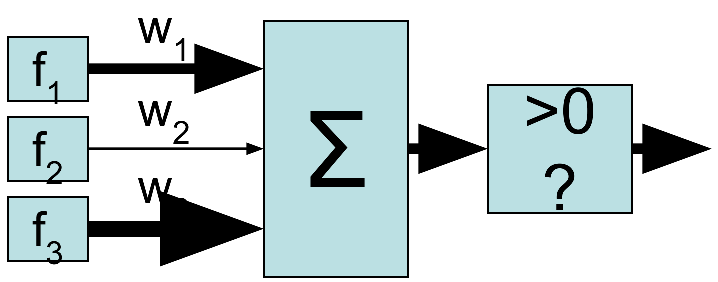
Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

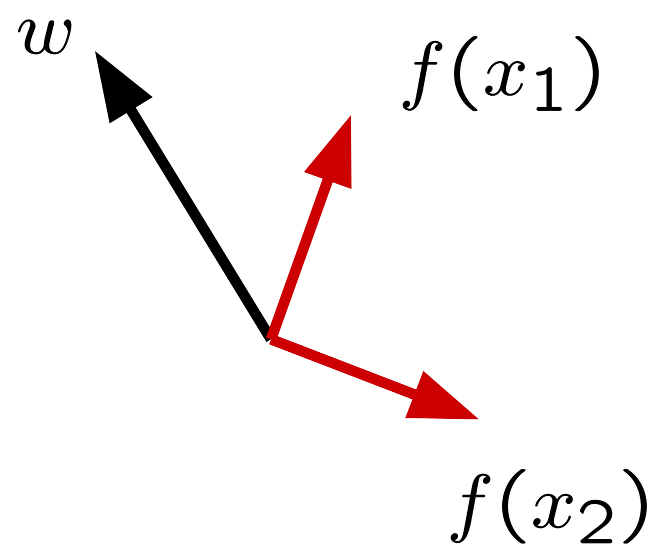
- If the activation is:
  - Positive, output +1
  - Negative, output -1



Weights

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples

```
# free      : 4
YOUR_NAME   :-1
MISSPELLED  : 1
FROM_FRIEND :-3
...
```

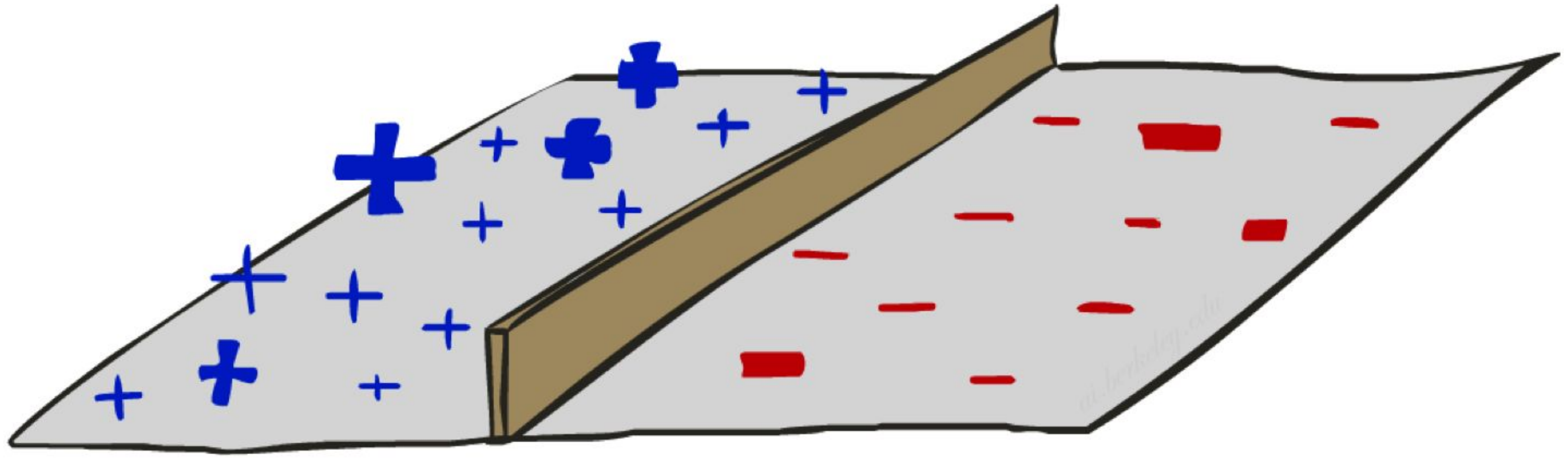


```
# free      : 2
YOUR_NAME   : 0
MISSPELLED  : 2
FROM_FRIEND : 0
...
```

```
# free      : 0
YOUR_NAME   : 1
MISSPELLED  : 1
FROM_FRIEND : 1
...
```

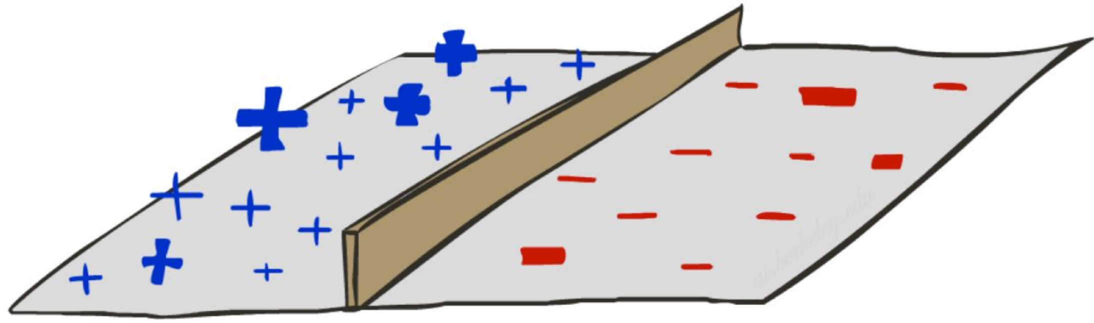
Dot product $w \cdot f$ positive means the positive class

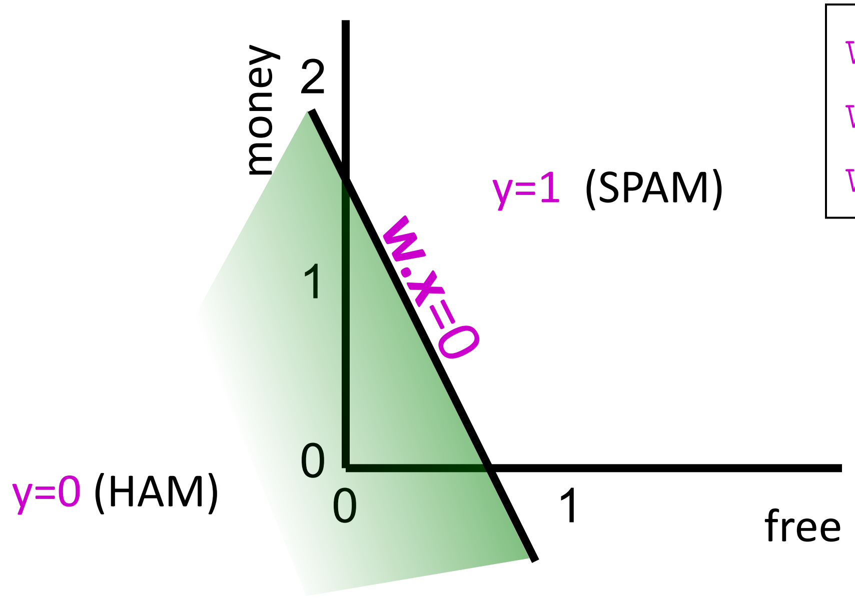
Threshold perceptron as linear classifier



Binary Decision Rule

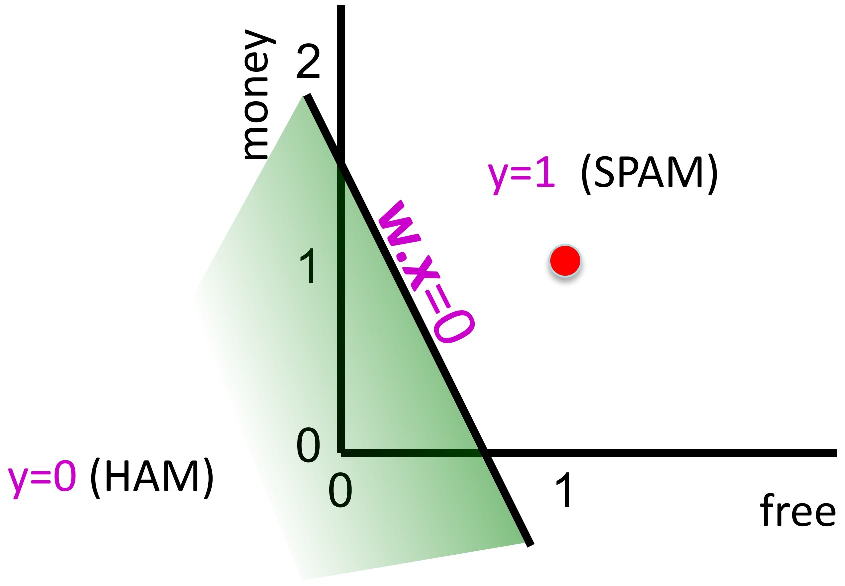
- A ***threshold perceptron*** is a single unit that outputs
  - $y = h_{\mathbf{w}}(\mathbf{x}) = 1$ when $\mathbf{w}.\mathbf{x} \geq 0$
    $= 0$ when $\mathbf{w}.\mathbf{x} < 0$
- In the input vector space
  - Examples are points $\mathbf{x}$
  - The equation $\mathbf{w}.\mathbf{x}=0$ defines a ***hyperplane***
  - One side corresponds to $y=1$
  - Other corresponds to $y=0$





w_0	:	-3
w_{free}	:	4
w_{money}	:	2

Example



w_0	:	-3
w_{free}	:	4
w_{money}	:	2

x_0	:	1
x_{free}	:	1
x_{money}	:	1

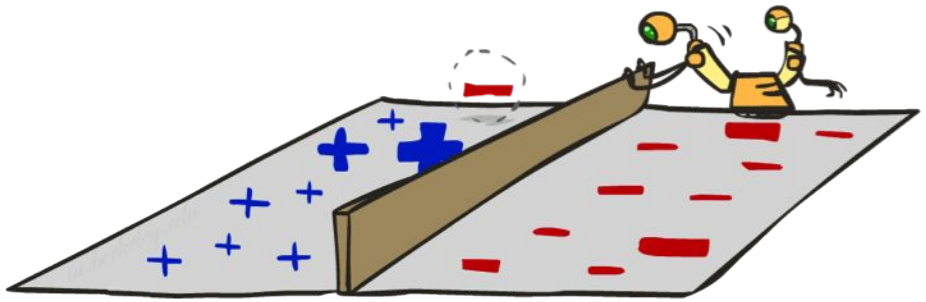
```
Dear Stuart, I'm leaving Macrosoft
to return to academia. The **money** is
is great here but I prefer to be **free**
to do my own research; and I *really*
love teaching undergrads!
Do I need to finish
my BA first before applying?
Best wishes
Bill
```

$$\mathbf{w.x} = -3 \times 1 + 4 \times 1 + 2 \times 1 = 3$$

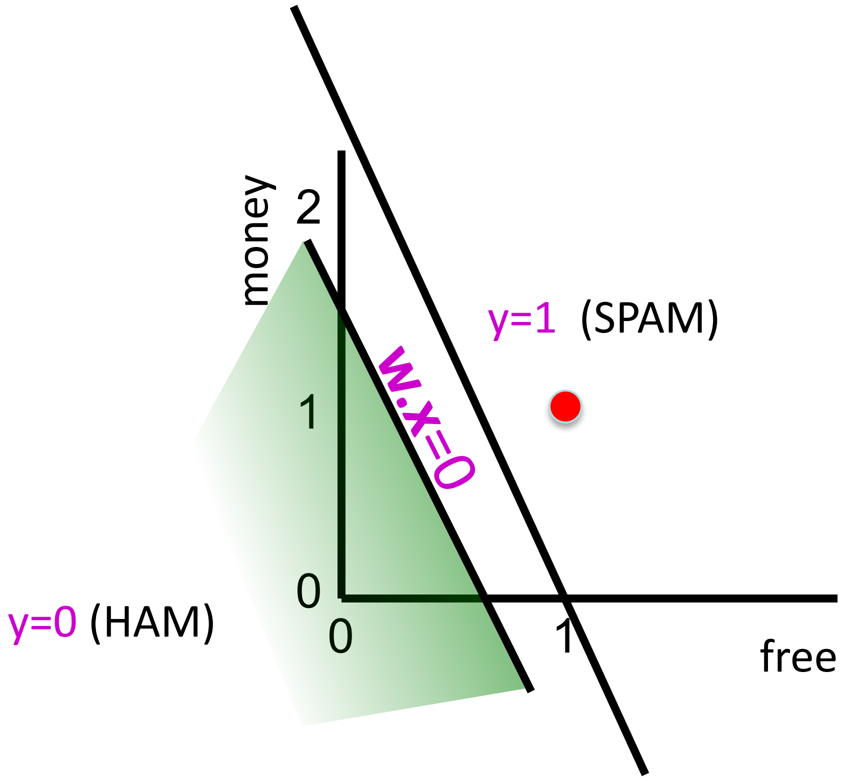
Weight Updates

Perceptron learning rule

- If true $y \neq h_{\mathbf{w}}(\mathbf{x})$ (an error), adjust the weights
- If $\mathbf{w.x} < 0$ but the output should be $y=1$
  - This is called a ***false negative***
  - Should ***increase*** weights on ***positive*** inputs
  - Should ***decrease*** weights on ***negative*** inputs
- If $\mathbf{w.x} > 0$ but the output should be $y=0$
  - This is called a ***false positive***
  - Should ***decrease*** weights on ***positive*** inputs
  - Should ***increase*** weights on ***negative*** inputs
- The ***perceptron learning rule*** does this:
  - $\mathbf{w} \leftarrow \mathbf{w} + \underbrace{\alpha}_{\text{learning rate}} \underbrace{(y - h_{\mathbf{w}}(\mathbf{x}))}_{\text{+1, -1, or 0 (no error)}} \mathbf{x}$



Example



w_0	:	-3
w_{free}	:	4
w_{money}	:	2

x_0	:	1
x_{free}	:	1
x_{money}	:	1

```
Dear Stuart, I wanted to let you know
that I have decided to leave Macrosoft
and return to academia. The money is
is great here but I prefer to be free
to pursue more interesting research
and I really love teaching
undergraduates! Do I need to finish
my BA first before applying?
Best wishes
Bill
```

$$\mathbf{w.x} = -3\times1 + 4\times1 + 2\times1 = 3$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x}))\,\mathbf{x}$$

$\alpha = 0.5$

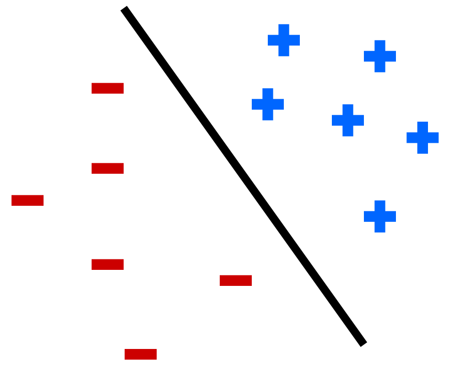
$$\mathbf{w} \leftarrow (-3,4,2) + 0.5\,(0-1)\,(1,1,1)$$
$$= (-3.5,3.5,1.5)$$

Perceptron convergence theorem

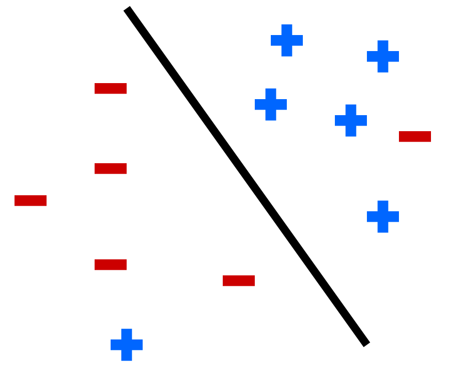
- A learning problem is ***linearly separable*** iff there is some hyperplane exactly separating +ve from –ve examples

- Convergence: if the training data are ***separable***, perceptron learning applied repeatedly to the training set will eventually converge to a perfect separator

Separable



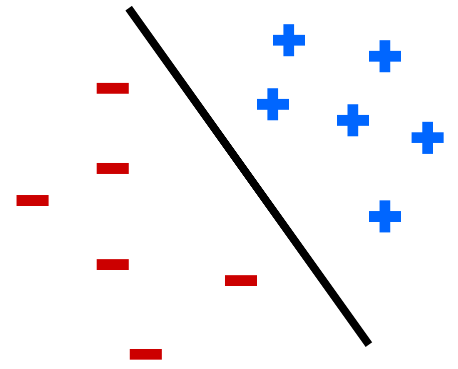
Non-Separable



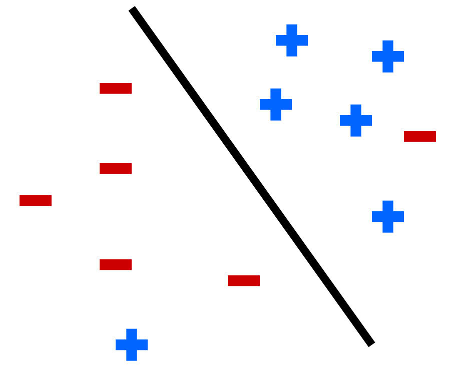
Perceptron convergence theorem

- A learning problem is ***linearly separable*** iff there is some hyperplane exactly separating +ve from –ve examples
- Convergence: if the training data are separable, perceptron learning applied repeatedly to the training set will eventually converge to a perfect separator
- Convergence: if the training data are ***non-separable***, perceptron learning will converge to a minimum-error solution provided the learning rate α is decayed appropriately (e.g., $\alpha=1/t$)

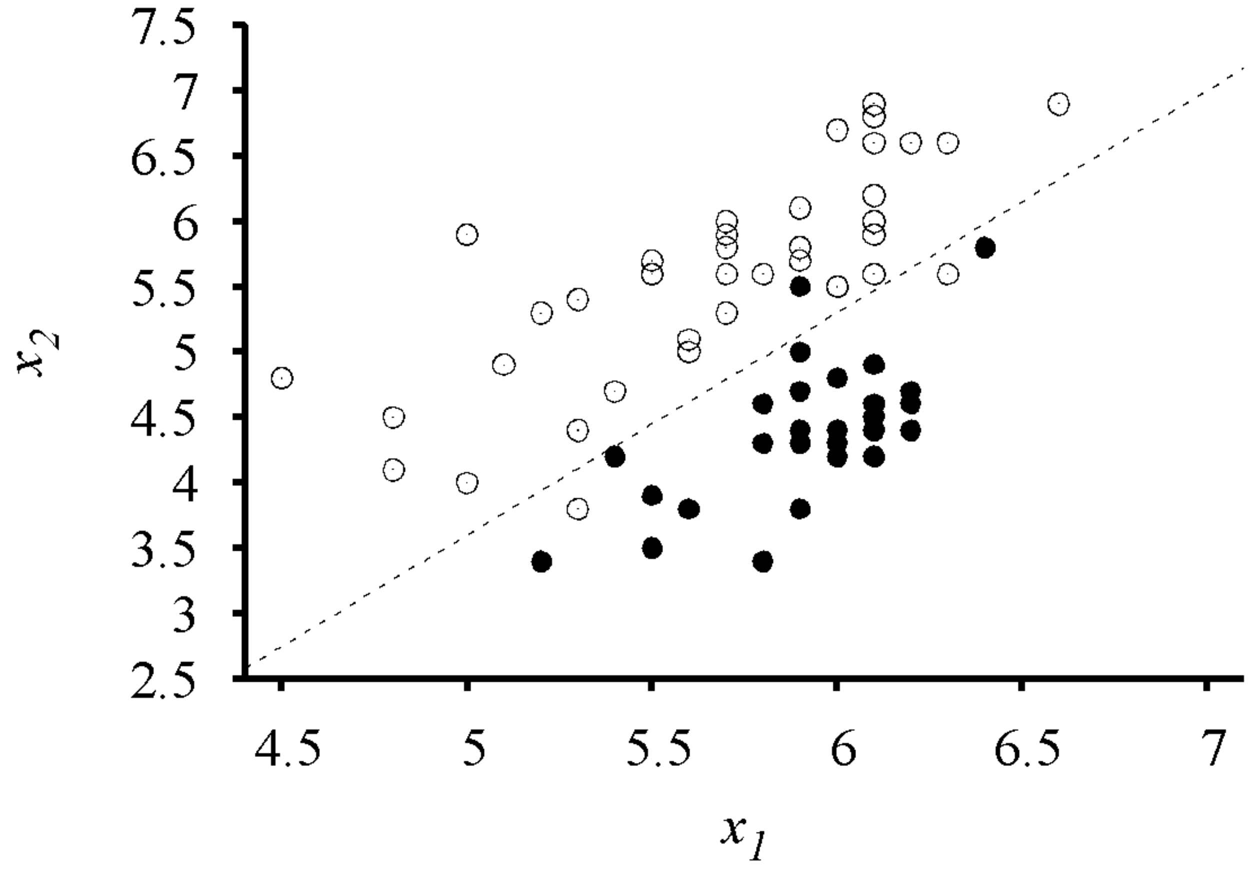
Separable

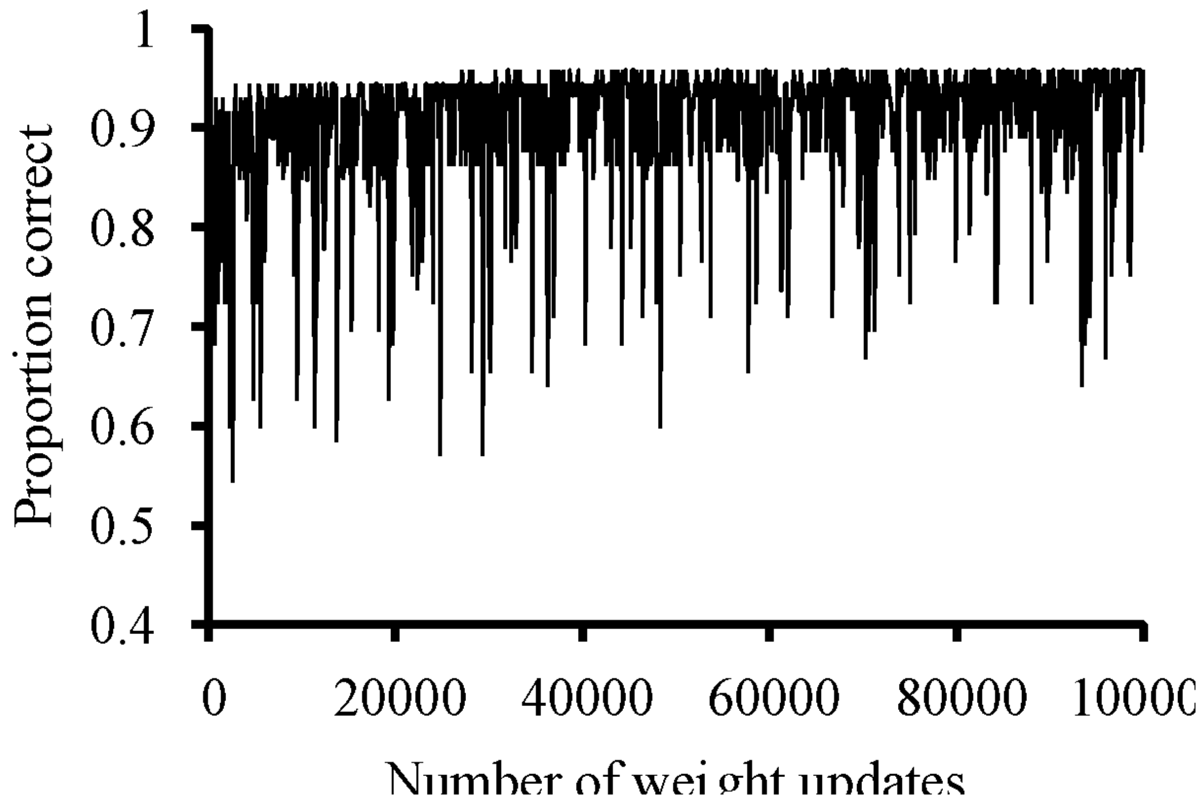


Non-Separable



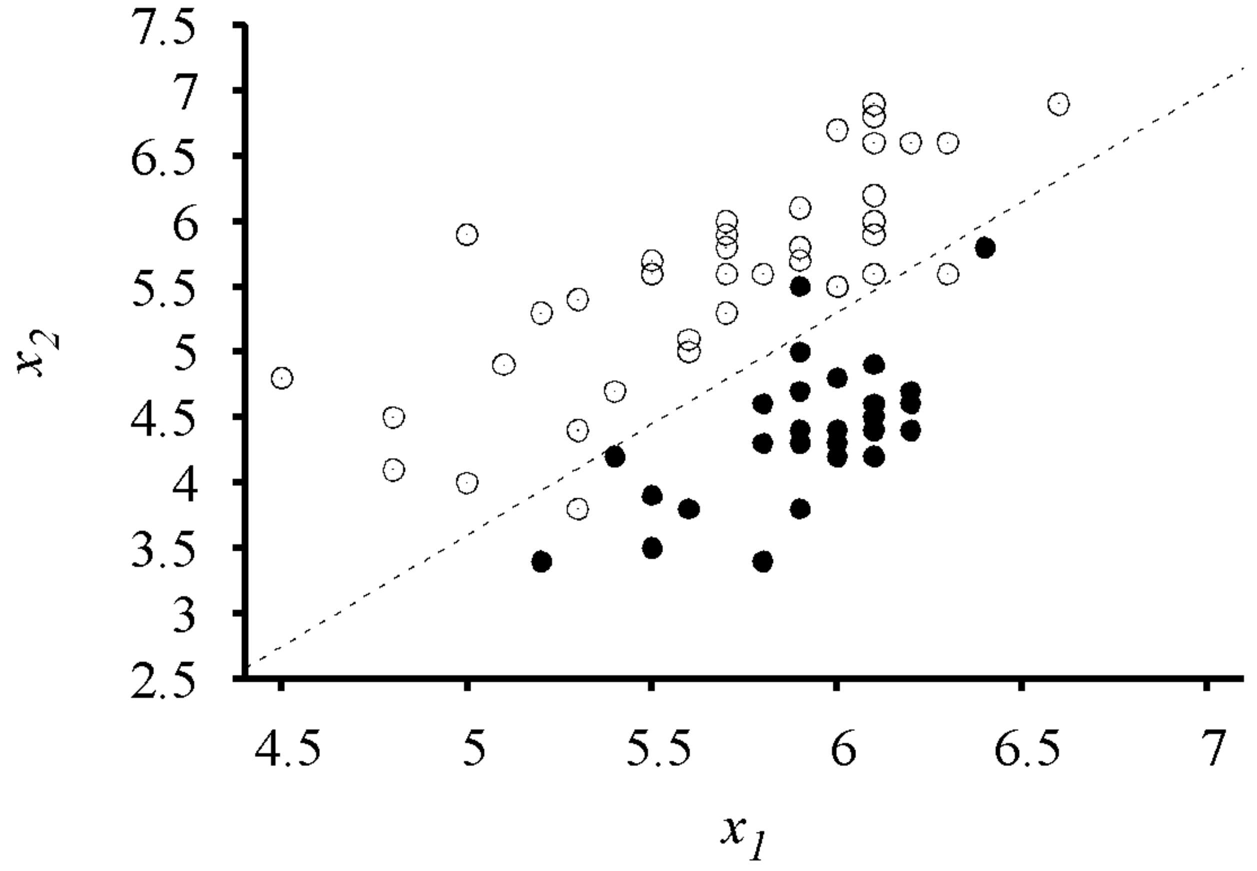
Perceptron learning with fixed α

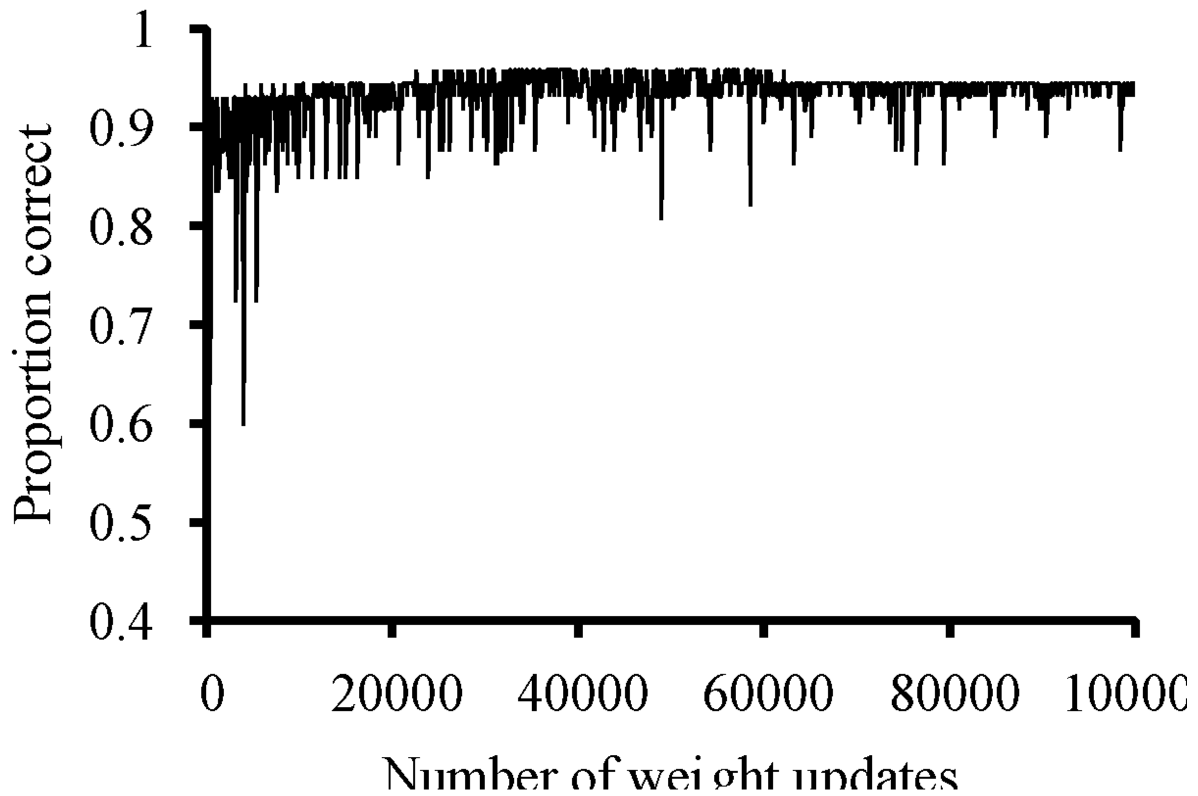




71 examples, 100,000 updates
fixed $\alpha = 0.2$, no convergence

Perceptron learning with decaying α

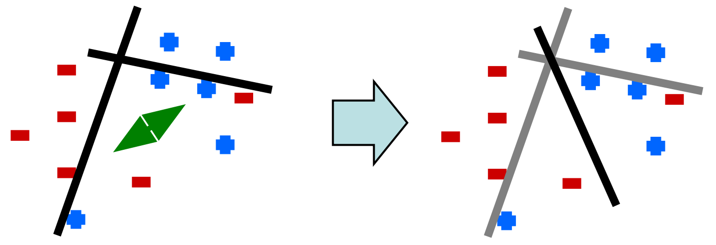


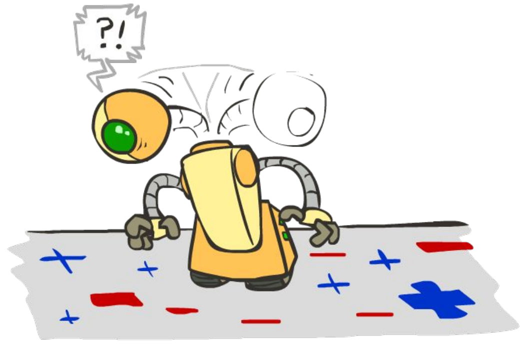


71 examples, 100,000 updates
decaying $\alpha = 1000/(1000 + t)$, near-convergence

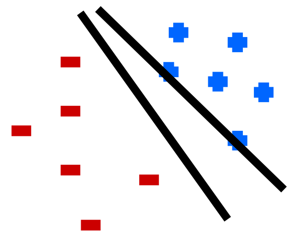
Problems with the Perceptron

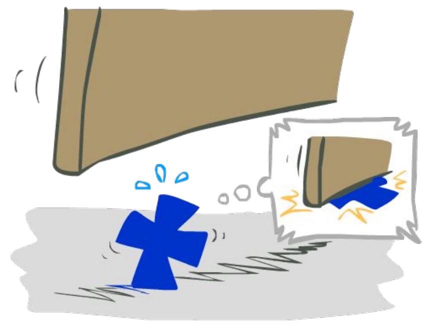
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



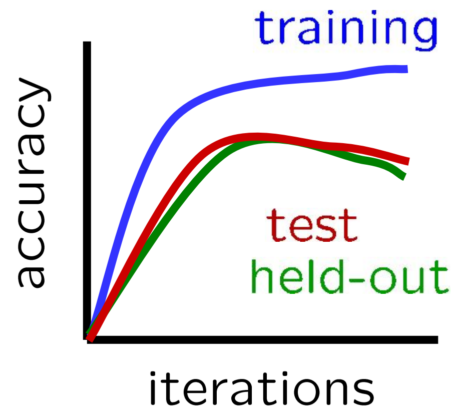


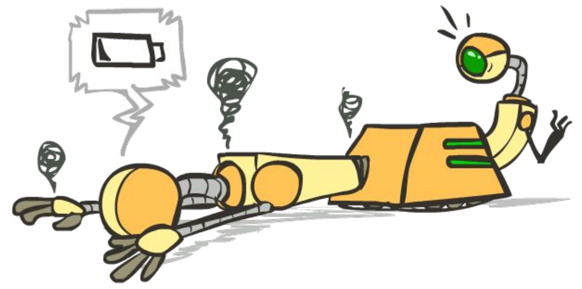
- Mediocre generalization: finds a "barely" separating solution



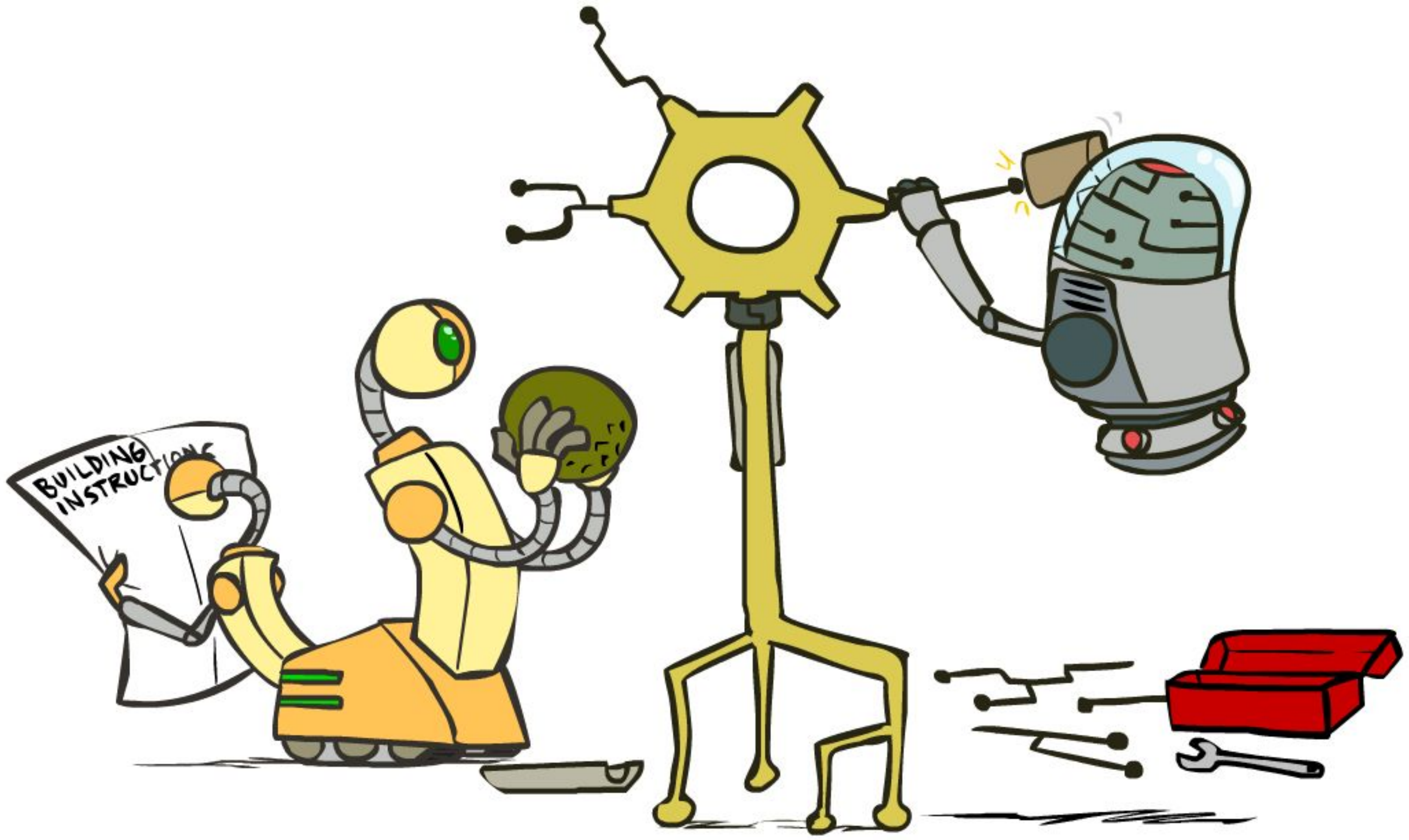


- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



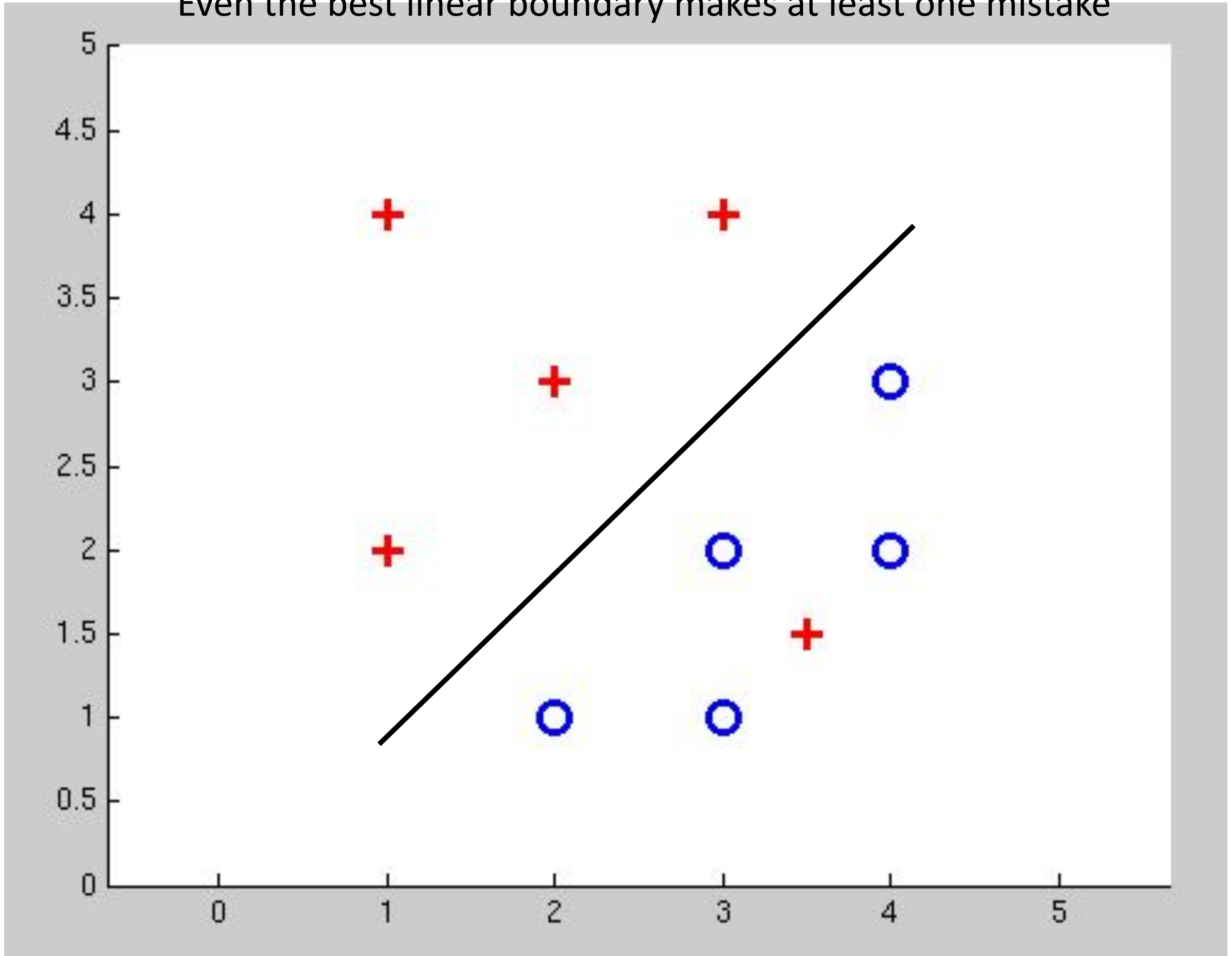


Improving the Perceptron

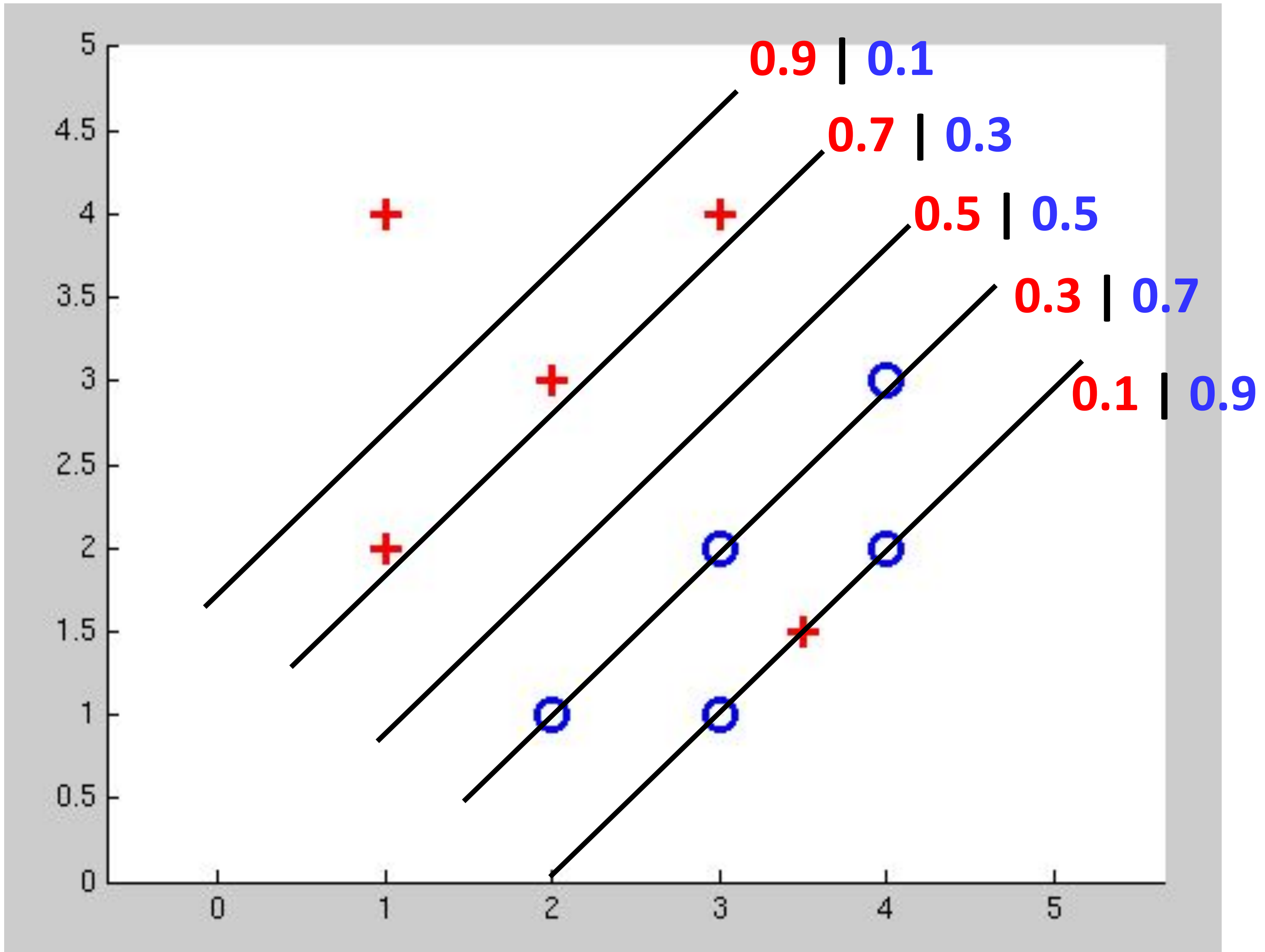


Non-Separable Case: Deterministic Decision

Even the best linear boundary makes at least one mistake



Non-Separable Case: Probabilistic Decision

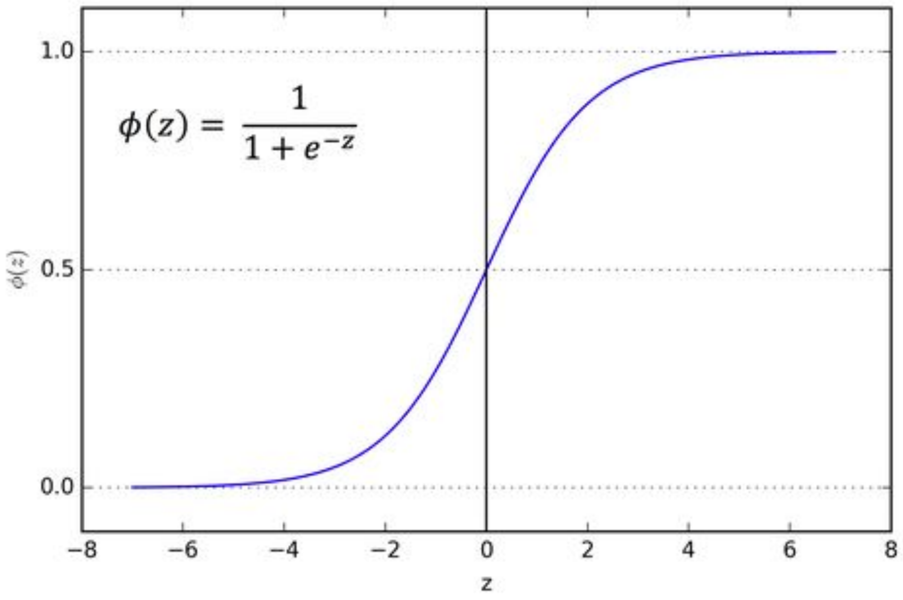


How to get probabilistic decisions?

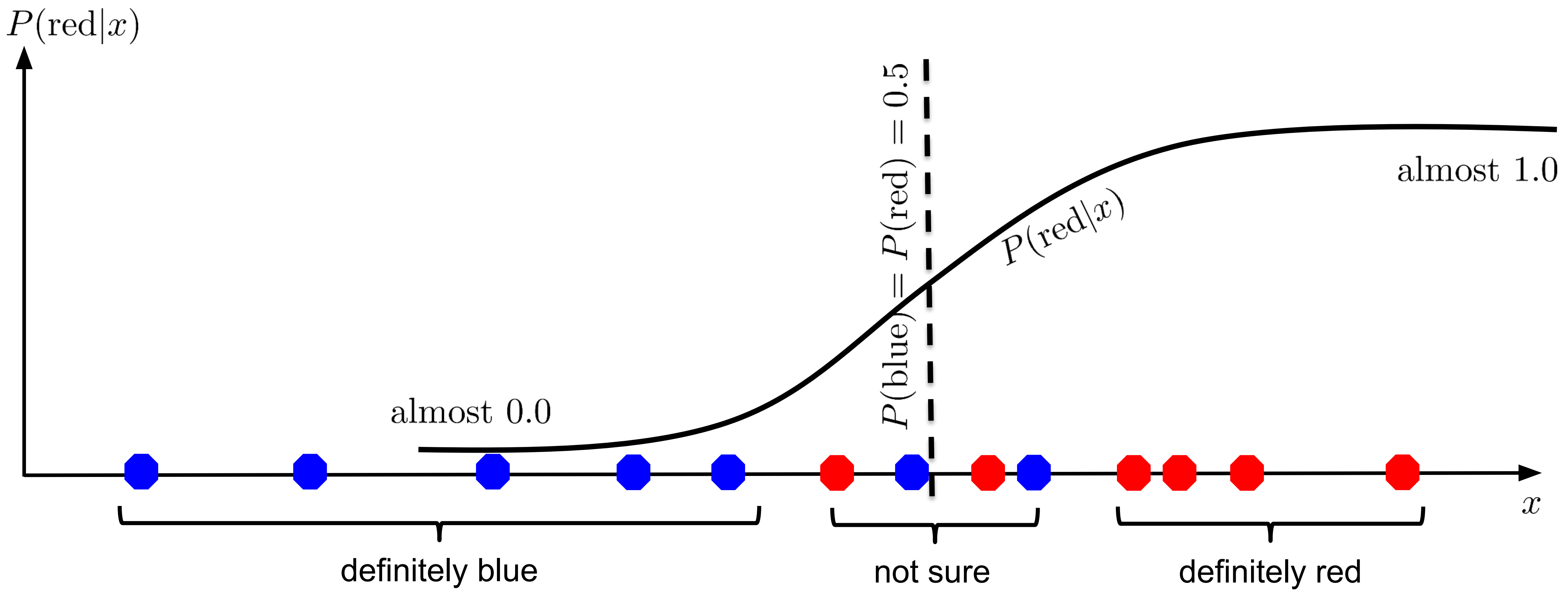
- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive □ want probability going to 1
- If $z = w \cdot f(x)$ very negative □ want probability going to 0

- Sigmoid function

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



A 1D Example

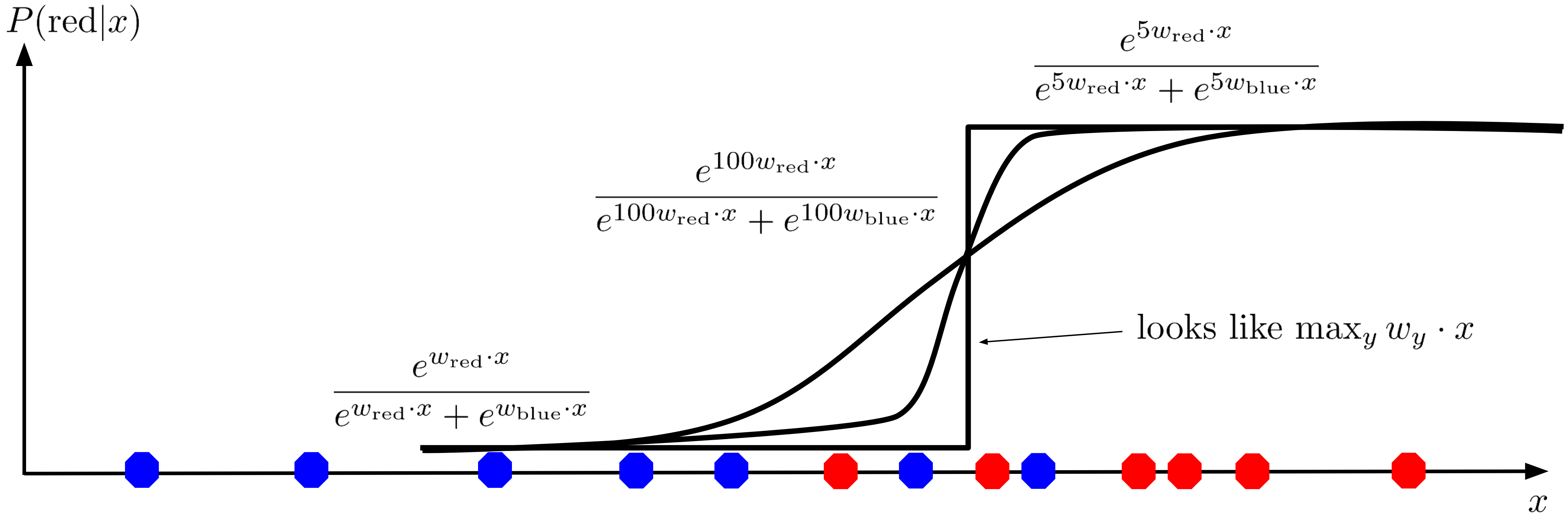


$$P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}$$

probability increases exponentially as we move away from boundary

normalizer

The *Soft* Max



$$P(\text{red}|x) = \frac{e^{w_{\text{red}}\cdot x}}{e^{w_{\text{red}}\cdot x}+e^{w_{\text{blue}}\cdot x}}$$

Best w?

- Maximum likelihood estimation:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

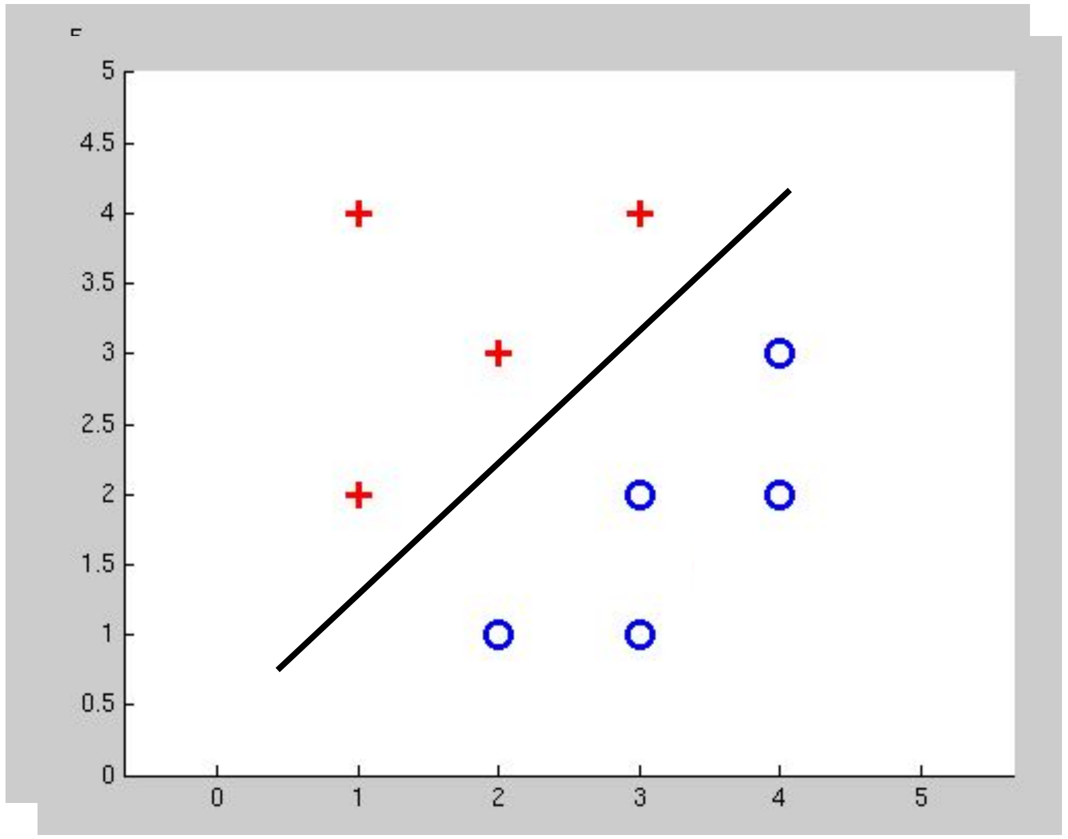
with:

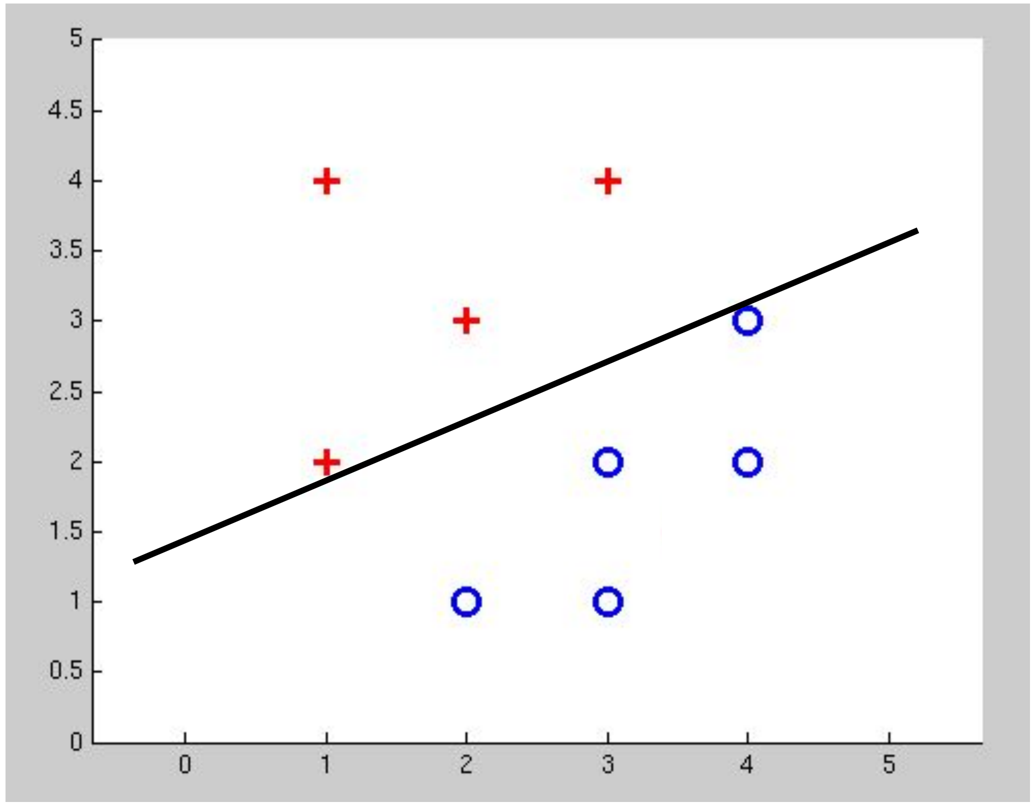
$$P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

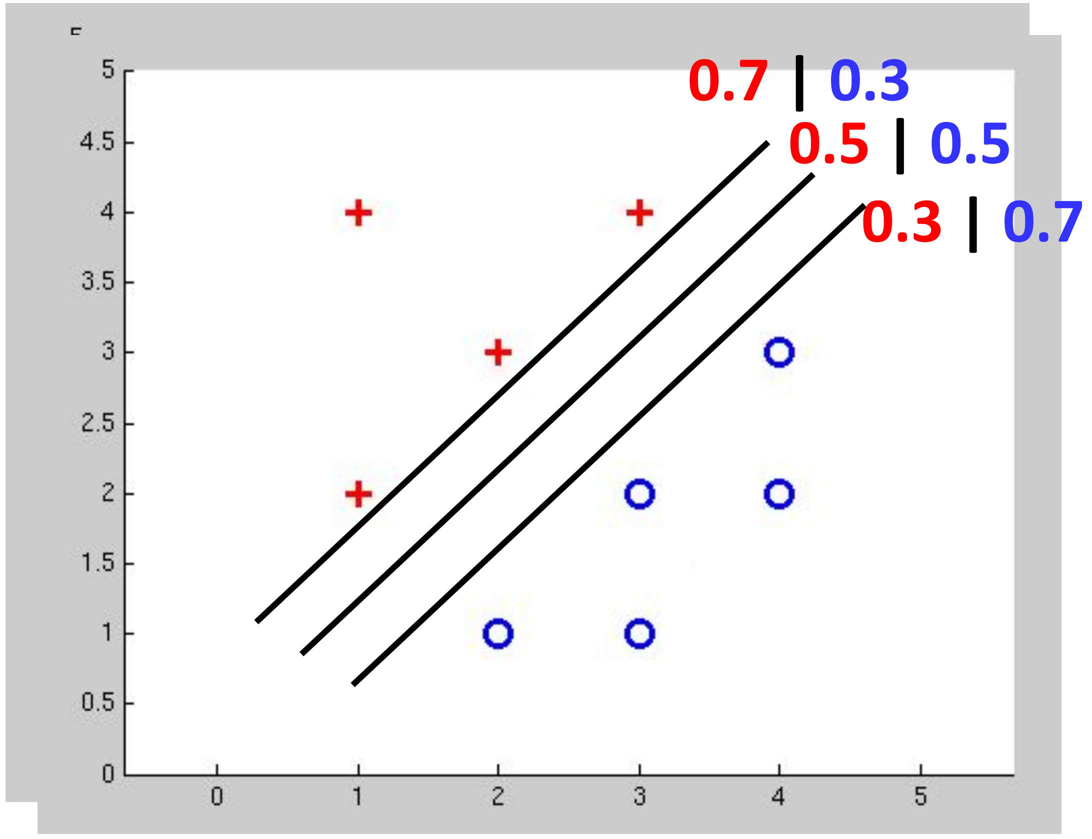
= Logistic Regression

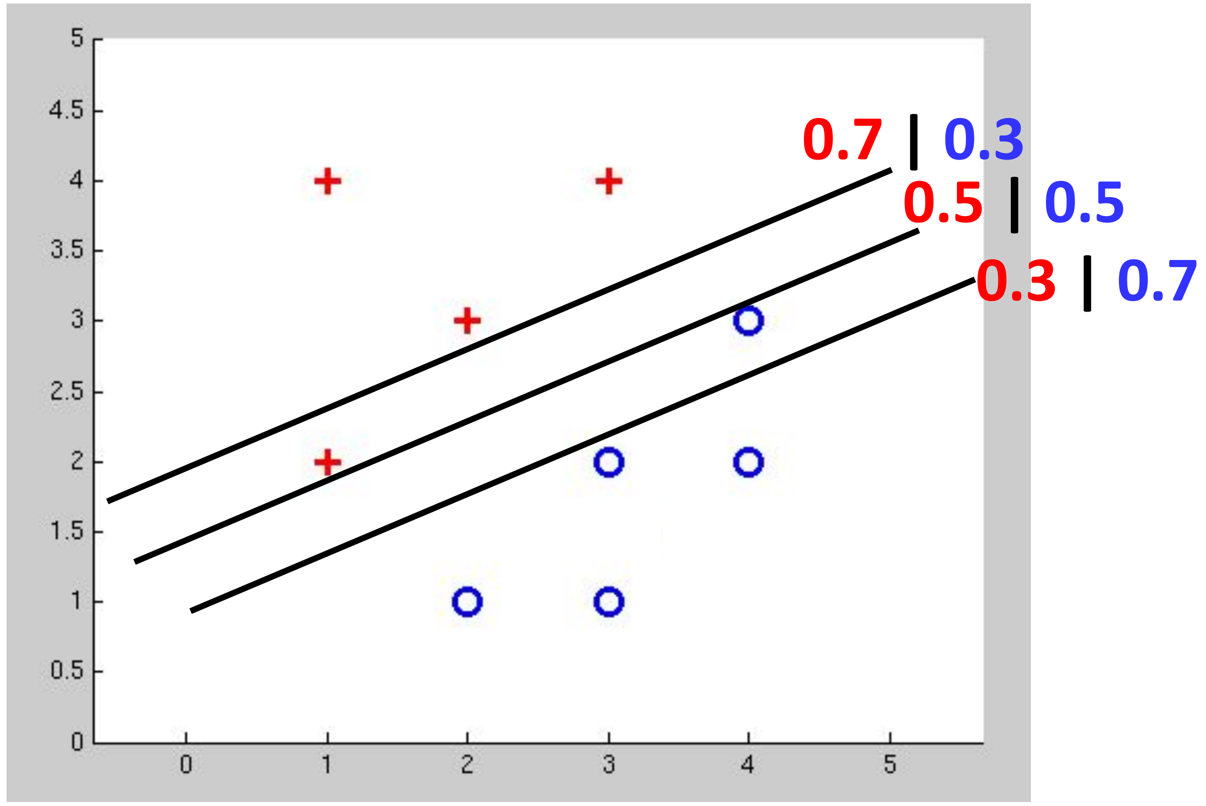
Separable Case: Deterministic Decision – Many Options





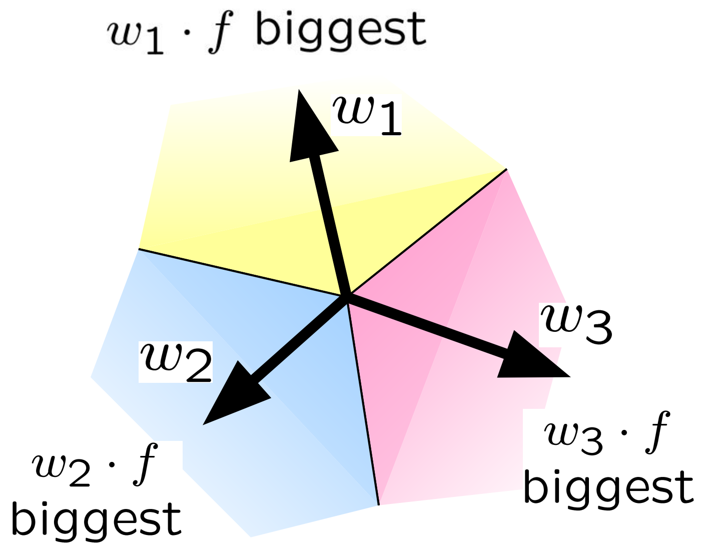
Separable Case: Probabilistic Decision – Clear Preference





Multiclass Logistic Regression

- Recall Perceptron:
  - A weight vector for each class: w_y
  - Score (activation) of a class y: $w_y \cdot f(x)$
  - Prediction highest score wins $y = \arg\max_y \; w_y \cdot f(x)$



- How to make the scores into probabilities?

$$\underbrace{z_1, z_2, z_3}_{\text{original activations}} \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}}_{\text{softmax activations}}$$

Best w?

- Maximum likelihood estimation:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

with:

$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

= Multi-Class Logistic Regression