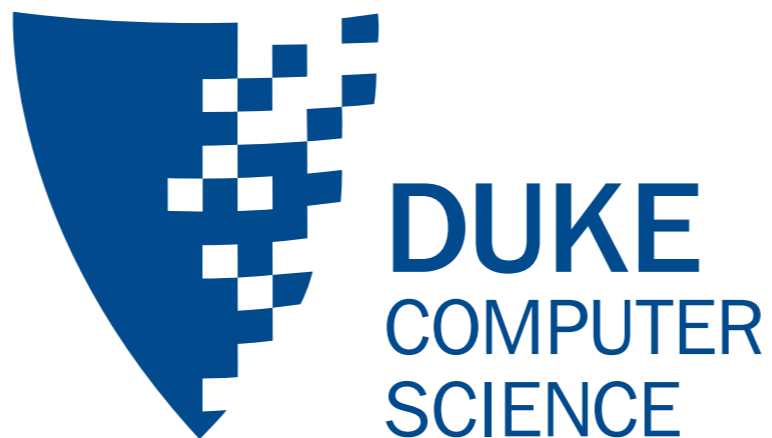


Machine Learning

George Konidaris
gdk@cs.duke.edu



Spring 2016

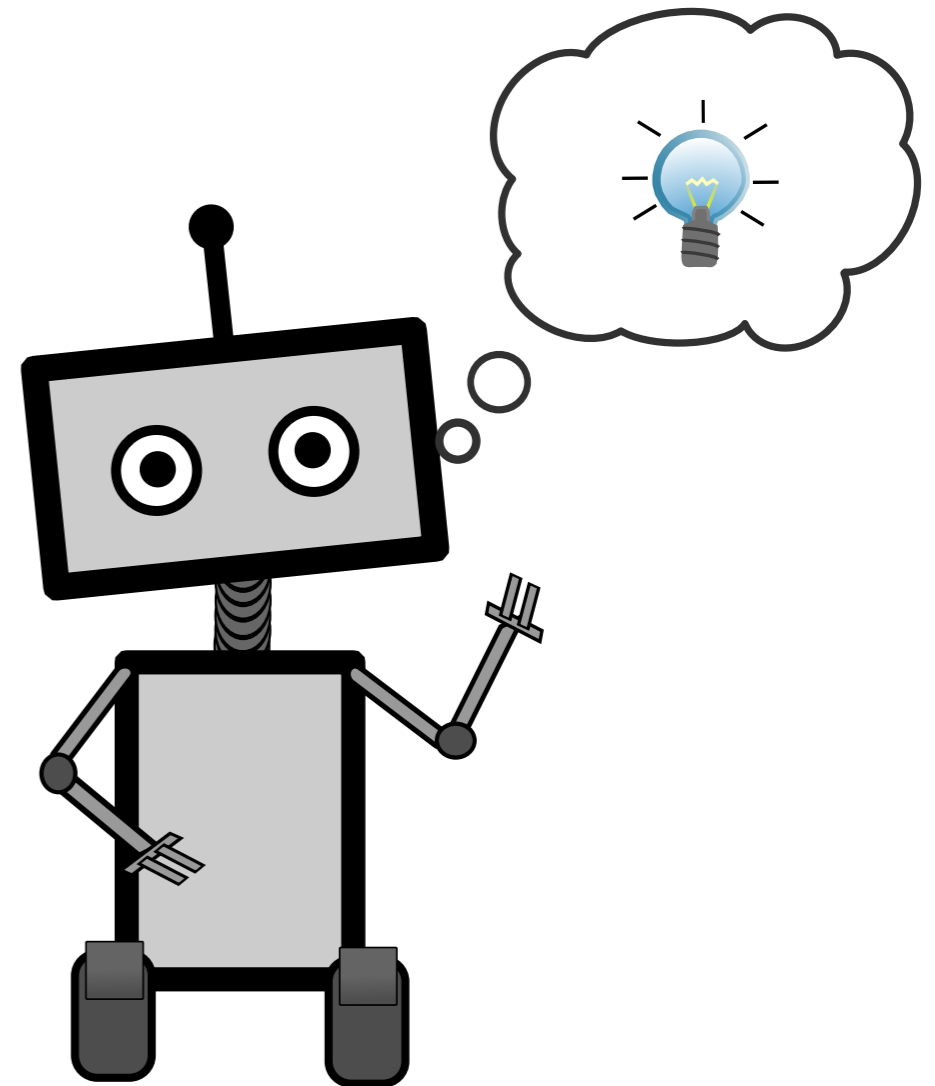
Machine Learning

Subfield of AI concerned with *learning from data*.

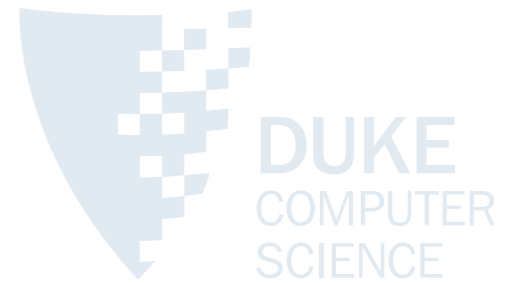
Broadly, using:

- *Experience*
- To Improve *Performance*
- On Some *Task*

(Tom Mitchell, 1997)



Recall: Supervised Learning



Formal definition:

Given training data:

$X = \{x_1, \dots, x_n\}$ **inputs**

$Y = \{y_1, \dots, y_n\}$ **labels - if discrete: *classification***

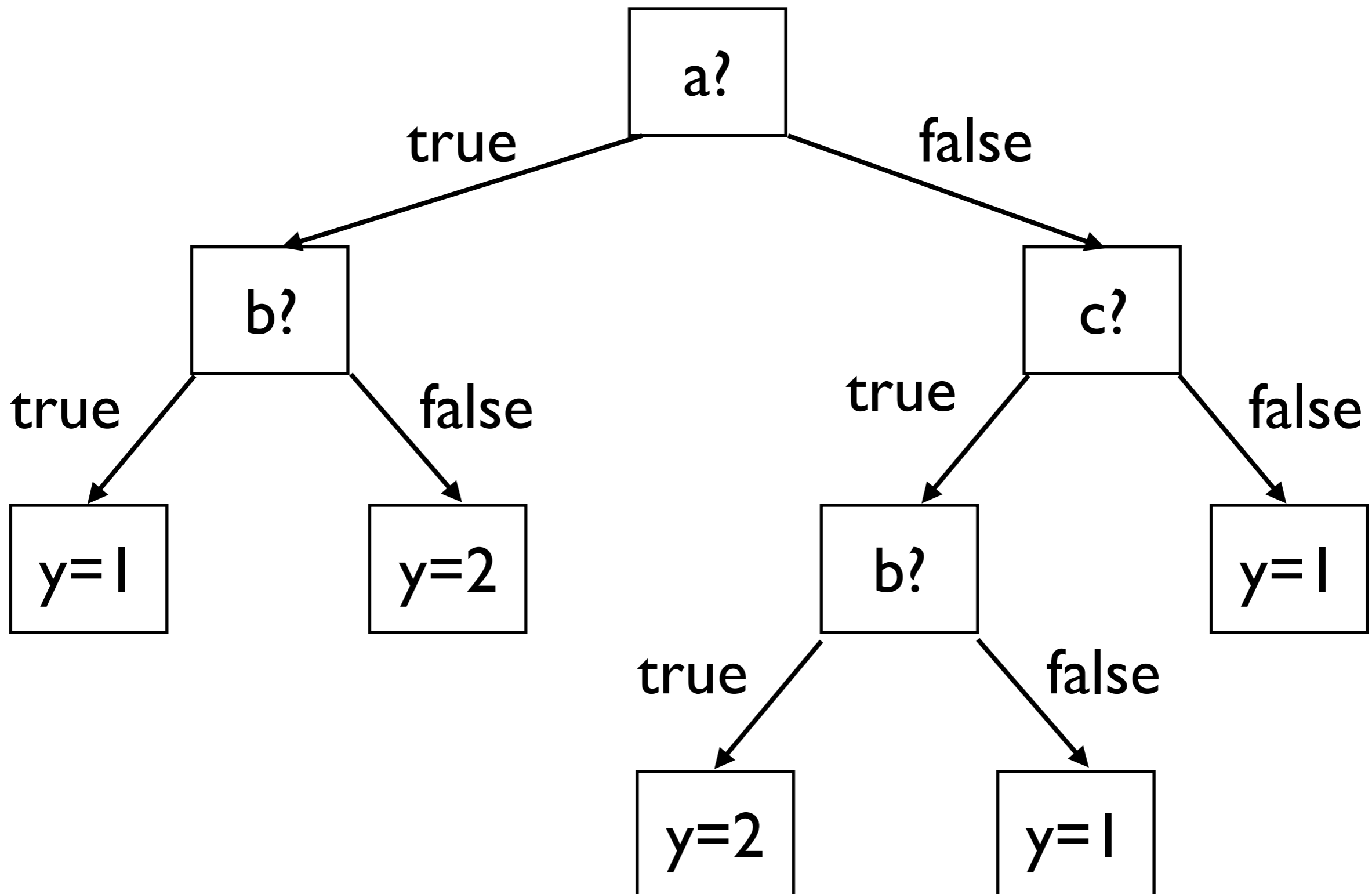
Produce:

Decision function $f : X \rightarrow Y$

That minimizes error:

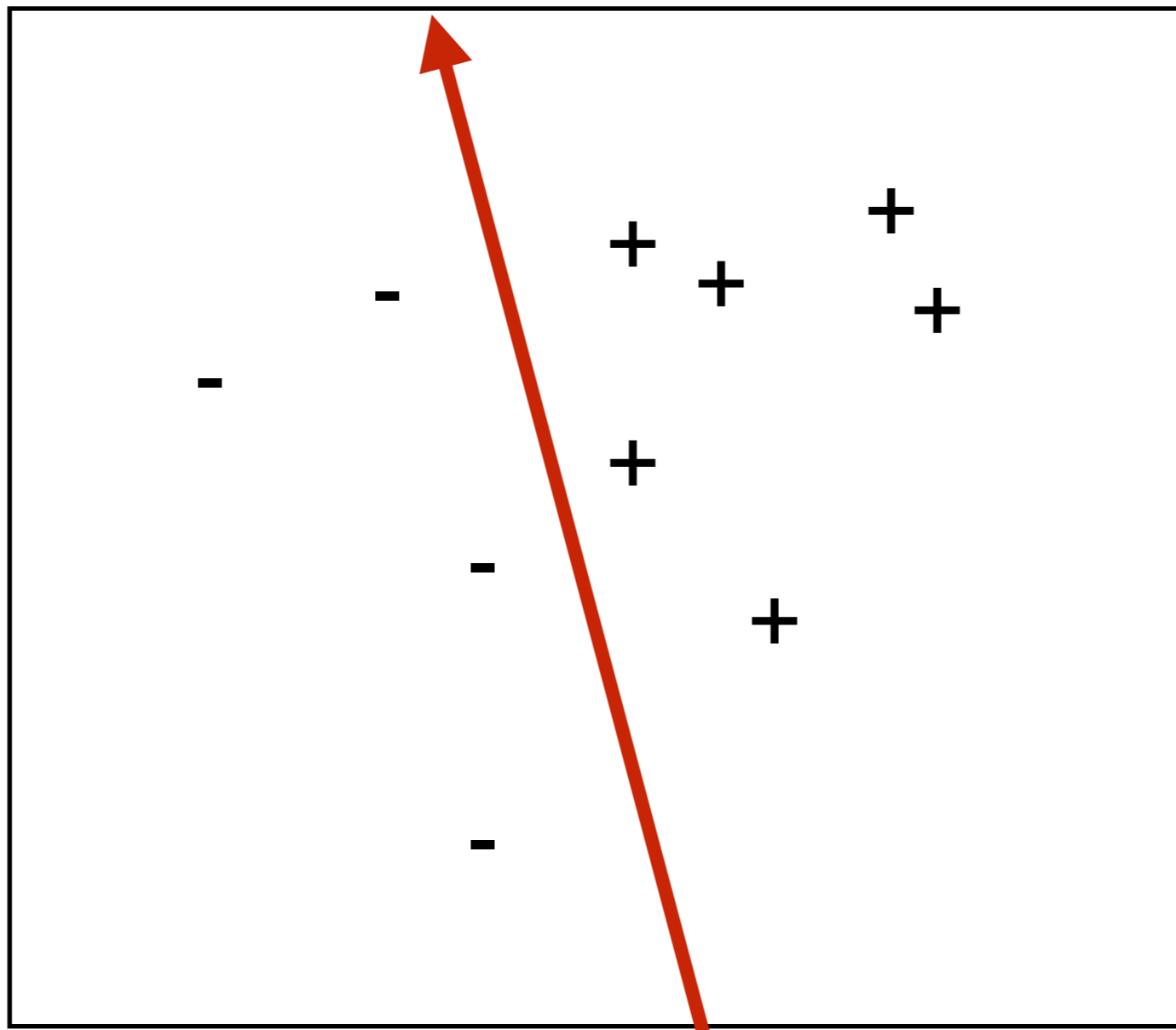
$$\sum_i err(f(x_i), y_i)$$

Decision Trees



The Perceptron

If your input (x_i) is real-valued ... *explicit decision boundary?*



The Perceptron

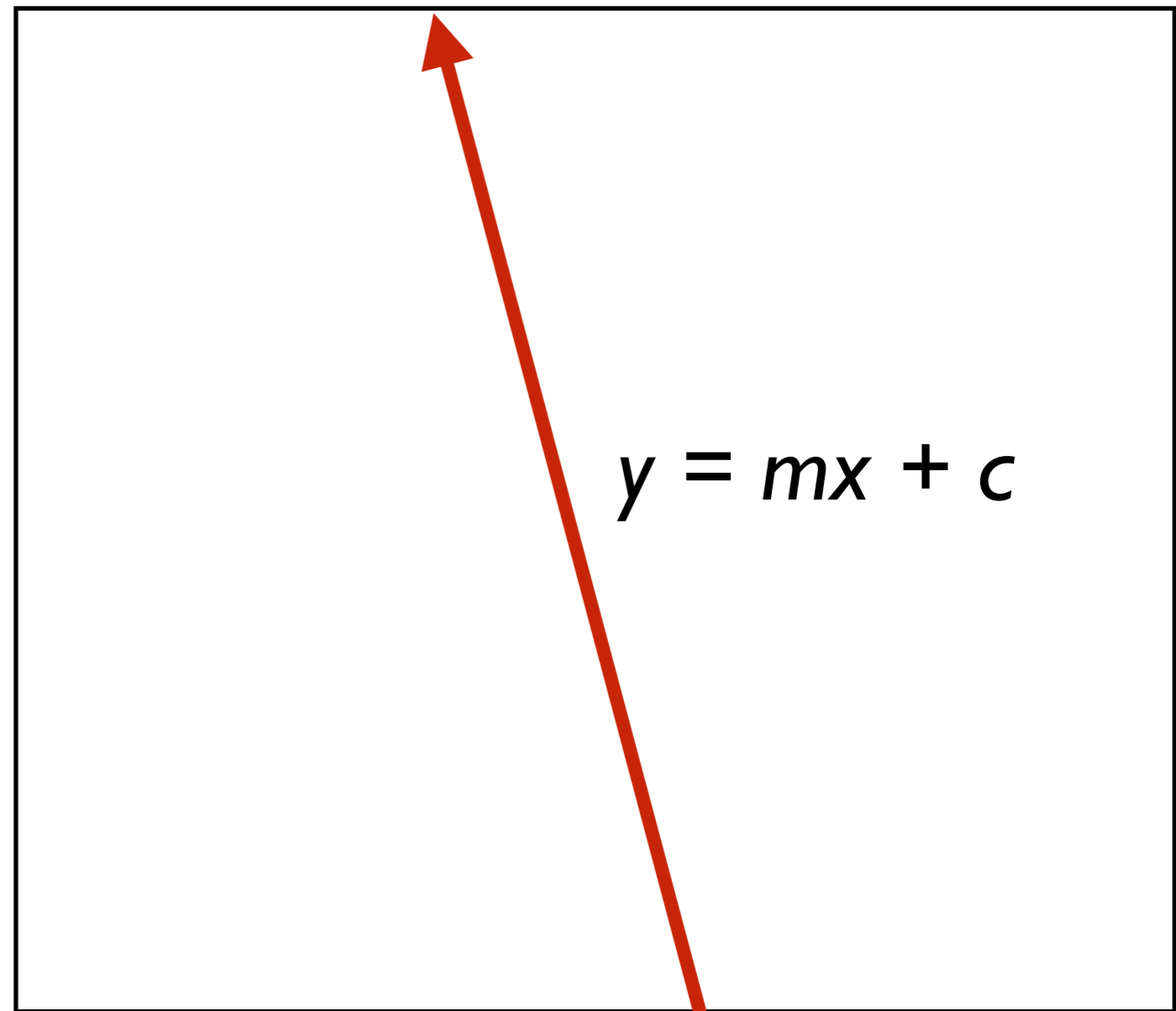
For $x = [x(1), \dots, x(n)]$:

- Create an n -d line
- Slope for each $x(i)$
- Constant offset

$$f(x) = \text{sign}(w \cdot x - c)$$

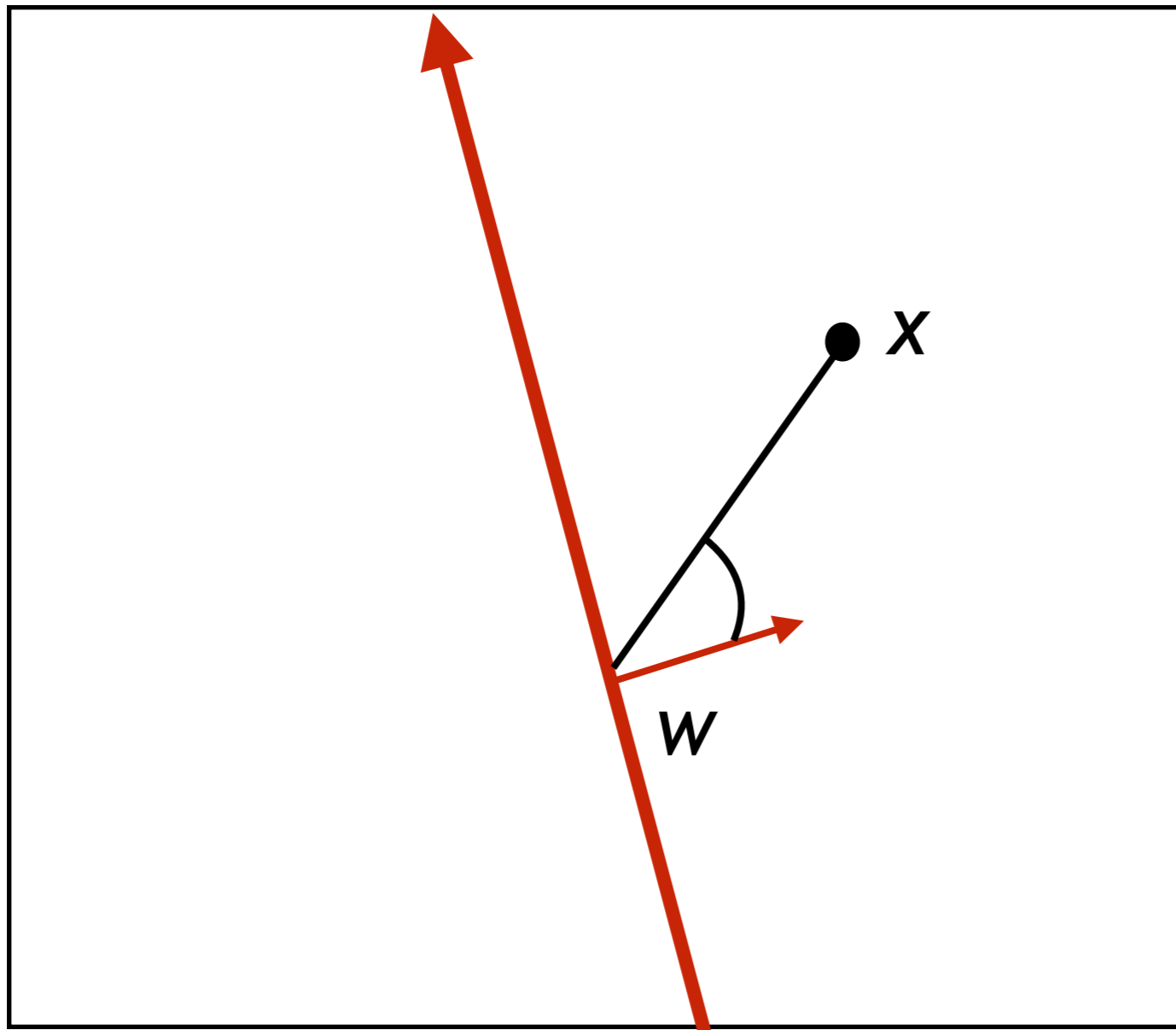
gradient

offset

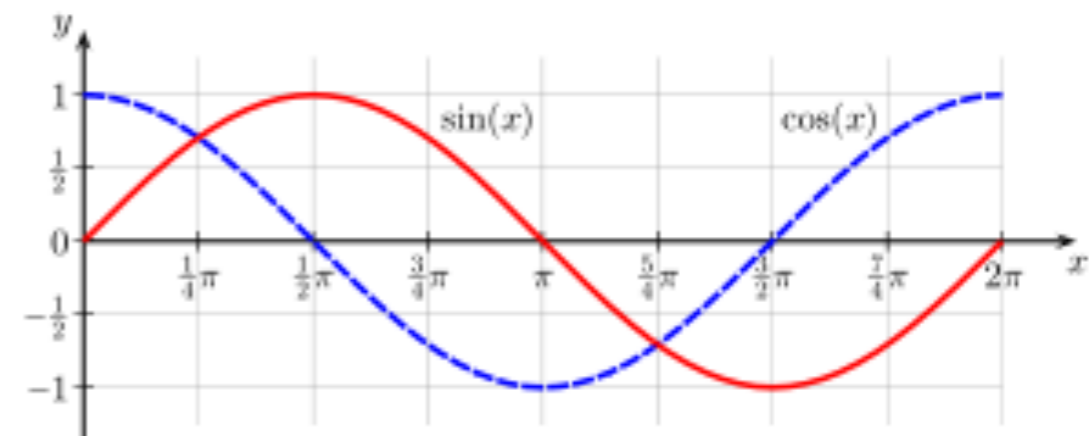


The Perceptron

Which side of a line are you on?

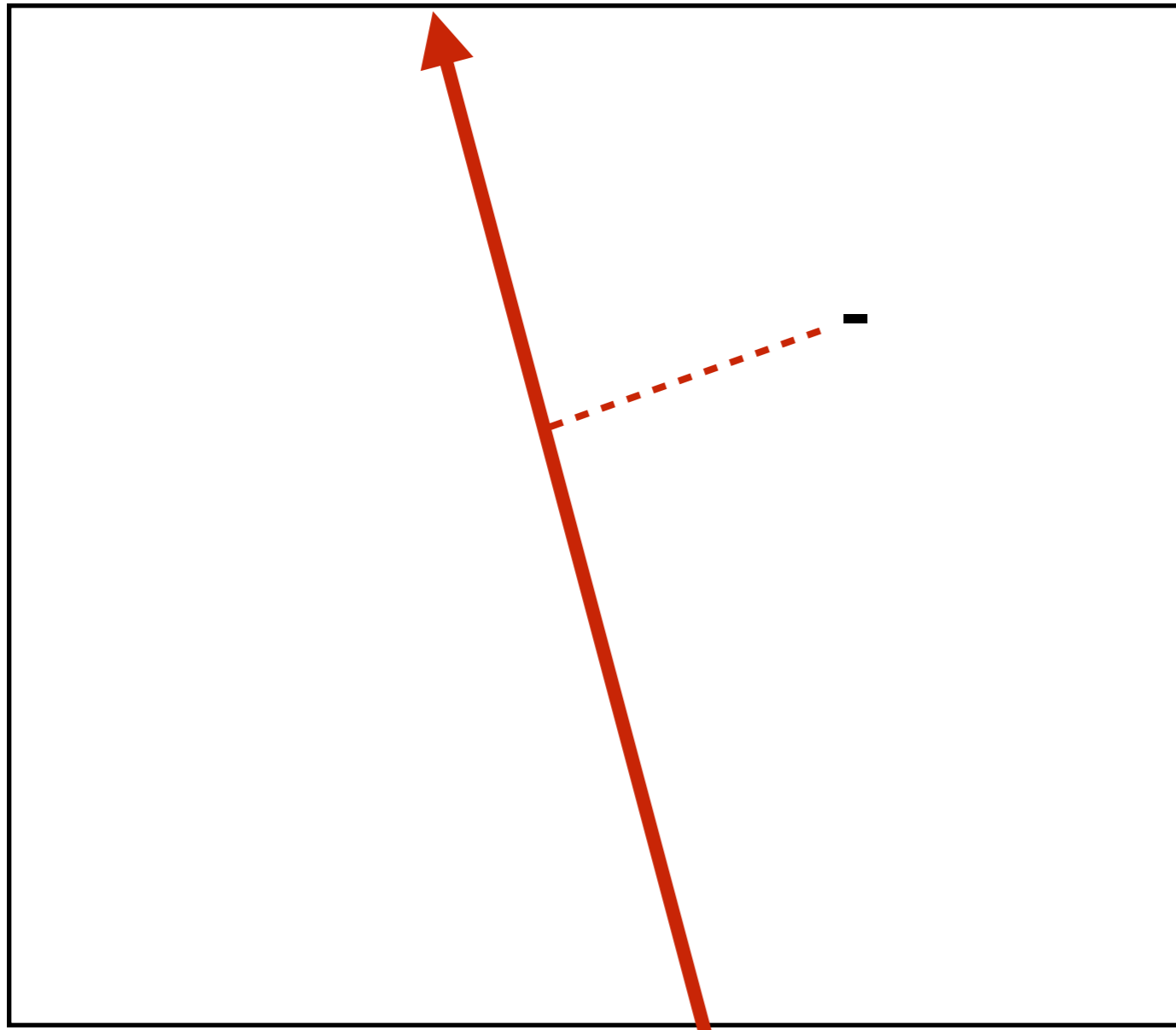


$$w \cdot x = \|w\| \|x\| \cos(\theta)$$



The Perceptron

How do you reduce error?



$$e = (y_i - (w \cdot x_i + c))^2$$
$$\frac{\partial e}{\partial w_j} = -2(y_i - (w_i \cdot x_i + c))x_i(j)$$



descend this gradient
to reduce error

The Perceptron Algorithm

Assume you have a *batch* of data:

$$X = \{x_1, \dots, x_n\}$$

$$Y = \{y_1, \dots, y_n\}$$

set w, c to 0.

for each x_i :

predict $z_i = \text{sign}(w \cdot x_i + c)$

if $z_i \neq y_i$:

$$w = w + a(y_i - z_i)x_i$$

learning rate



**converges if data
is linearly separate**

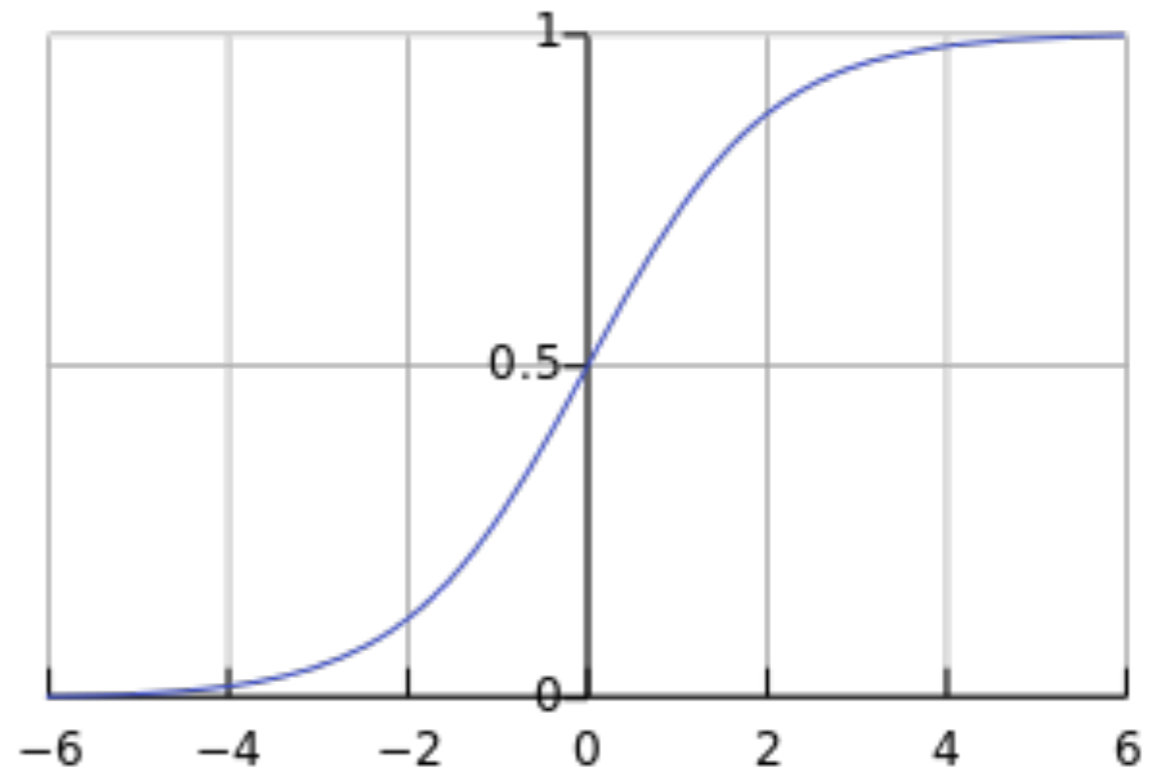
Probabilities

What if you want a *probabilistic classifier*?

Instead of *sign*, squash output of linear sum down to $[0, 1]$:

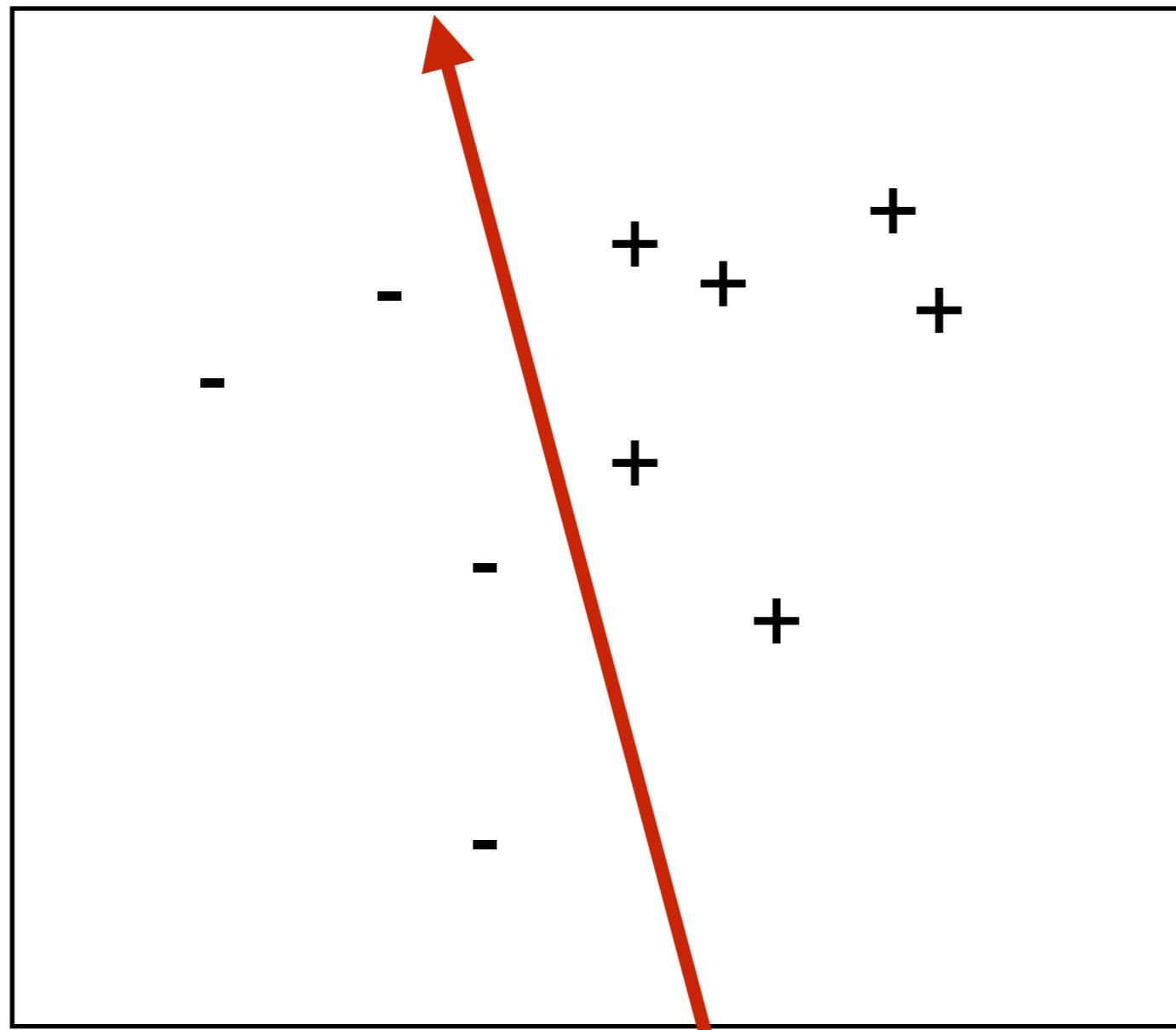
$$\sigma(w \cdot x + c)$$

Resulting algorithm:
logistic regression.



Perceptrons

What can't you do?

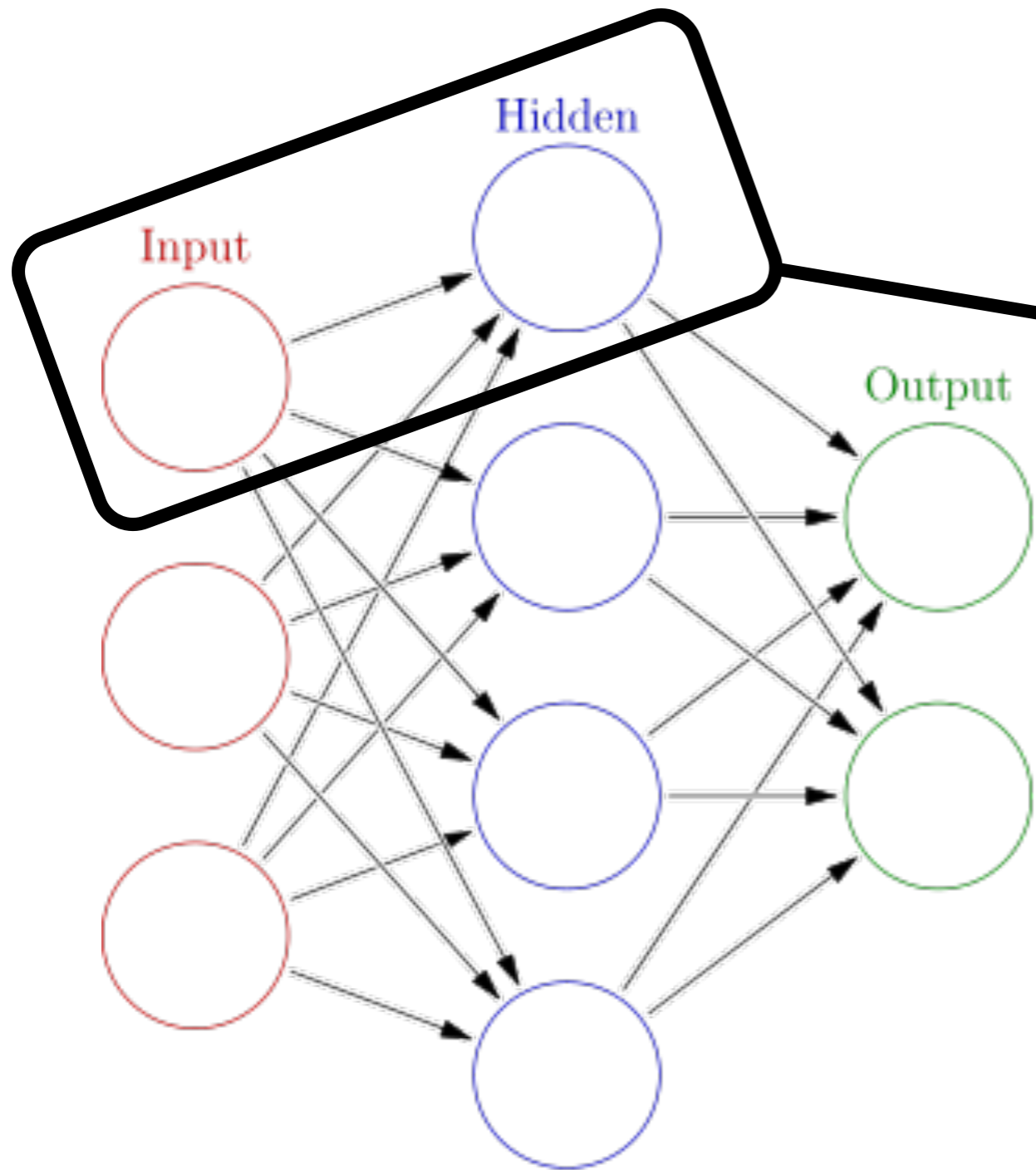


Frank Rosenblatt

Built the *Mark I* in 1960.

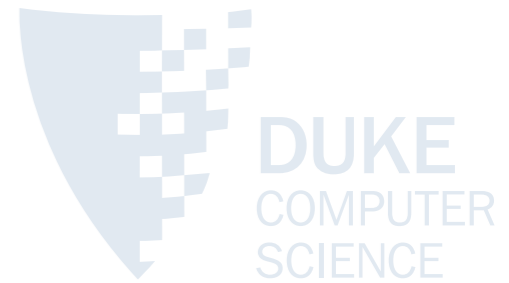


Neural Networks



$\sigma(w \cdot x + c)$
logistic regression

Nonparametric Methods



Most ML methods can be characterized by setting a few parameters.

Alternative approach:

- Let the data speak for itself.
- No finite-sized parameter vector.
- Usually more interesting decision boundaries.

K-Nearest Neighbors

Given training data:

$$X = \{x_1, \dots, x_n\}$$

$$Y = \{y_1, \dots, y_n\}$$

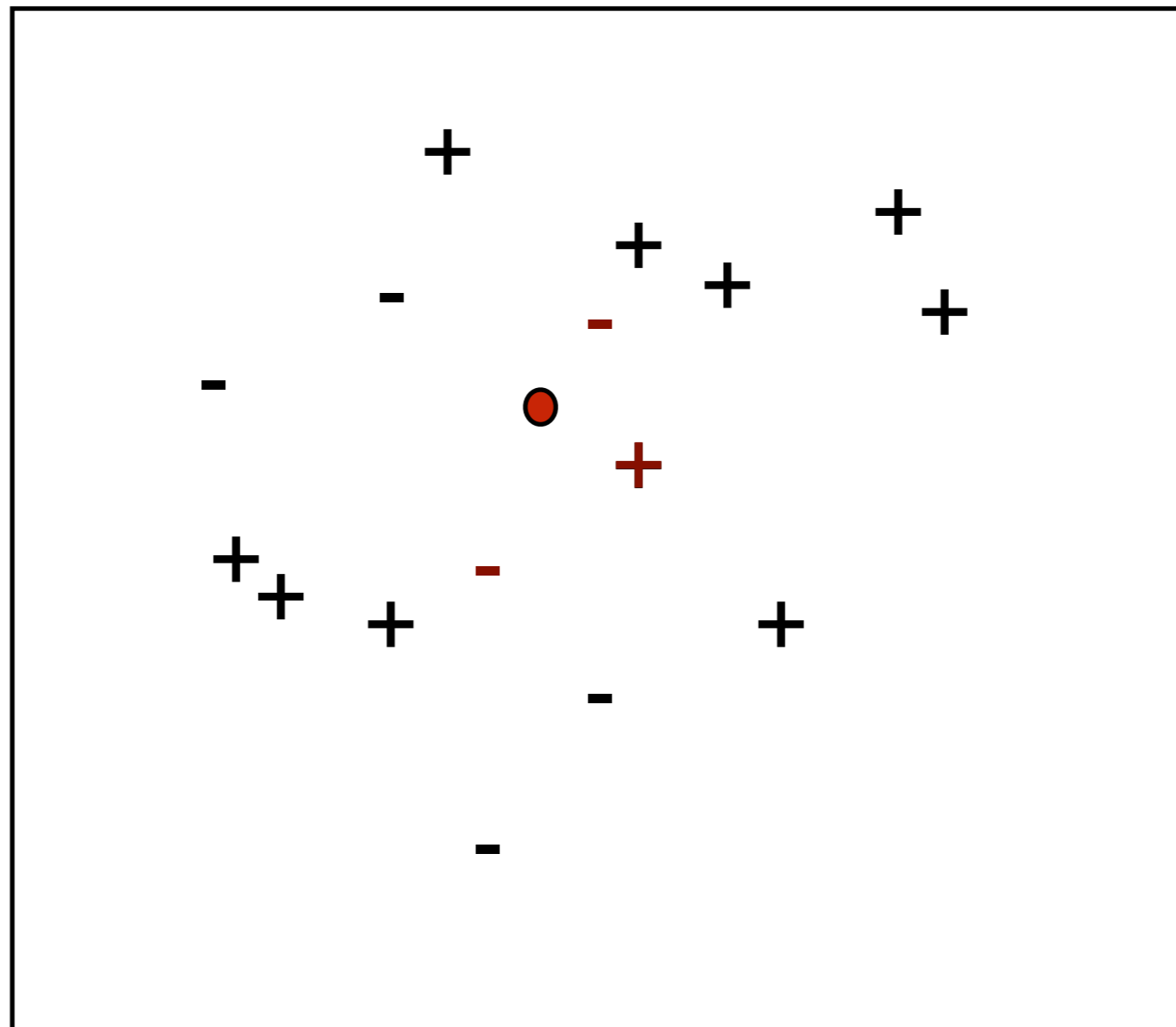
Distance metric $D(x_i, x_j)$

For a new data point x_{new} :

find k nearest points in X (measured via D)

set y_{new} is the majority label

K-Nearest Neighbors



K-Nearest Neighbors

Properties:

- No learning phase.
- Must store all the data.
- $\log(n)$ computation per sample - *grows with data*.

Decision boundary: *any function, given enough data*.

Classic trade-off: memory and compute time for flexibility.

Applications

Training set: 60k digits

Test set: 10k digits



Best performance: ~0.3% error rate.

Applications

- Fraud detection
- Internet advertising
- Friend or link prediction
- Sentiment analysis
- Face recognition
- Spam filtering

**“A breakthrough in machine learning would
be worth ten Microsofts.”
- Bill Gates**