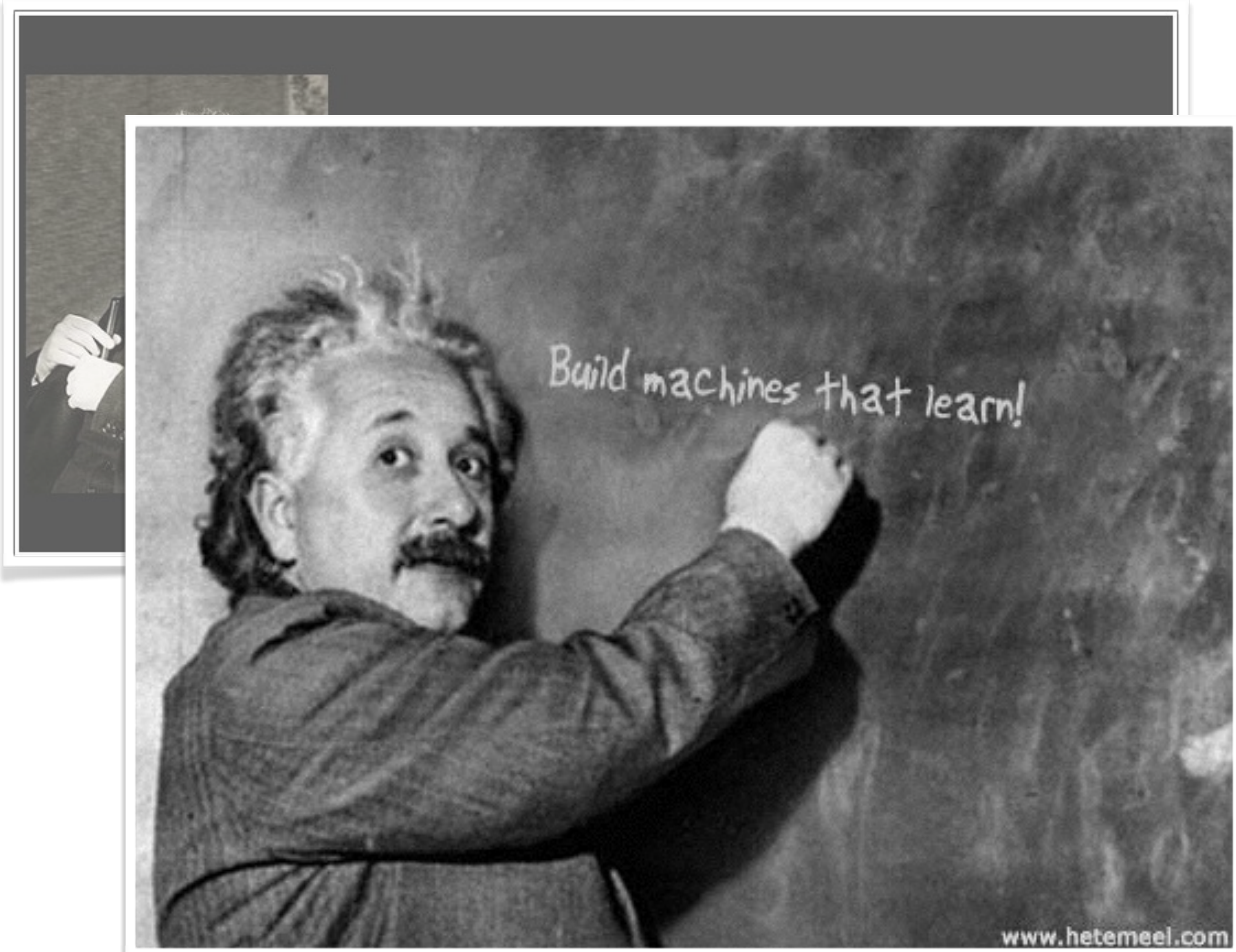
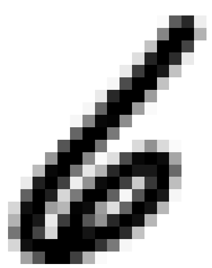
6.867 Fall 2017
Introduction to Classification

Lecture 5: 21st Sept., 2017


Build machines that learn!
www.hetemeel.com

Classification: features to predictions

x		$\phi(x)$		y
Mrs. MELISSA LEWIS **<qa.zx2015@yandex.com>** I am writing to seek your consent to conduct humanitarian projects, becos I suffer from advanced cancer that prevents me from realizing my dreams. That is why I want to Send these some of my money to you (EURO 4,000,000.00) Four million Euro so that you can use it to help the Orphanages,homeless and Widows and 35% for you while you use 65% for the project.. ..	⇒	Risky domain: 1 Misspelled: 2 From friend: 0 Money: 1 Your_name: 0	clf ···▶	Spam (+1)
	⇒	Pixel_12,12: 1 Pixel_12,13: 1 .. .. : 0 Pixel_28,28: 0 Has_loop: 1	clf ···▶	"6"
	↗	Raw pixels SIFT, HIST CNN features	clf ···▶	Albatross (+1)

Example: raw data to prediction

[http://detexify.kirelabs.org/classify.html]

Classification

Training data

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \sim \mathbb{P}(\mathcal{X}, \mathcal{Y})$$

Test data

$$\{x_{N+1}, \ldots, \}$$

Labels

$$\text{Predict} \quad \{y_{N+1}, \ldots, \}$$

Important:

Test data must be 'featurized' in the same way as training data
Test data should be drawn from same distribution
Typically we assume that we do not know $\mathbb{P}$
Many subtleties can arise — we must be careful

Classification

* So far we saw regression:
  * Noise model, Gaussian, least squares
  * Ridge-regression, regularization, Lasso
  * Bayesian linear regression
* Aim is to "predict" a continuous target Y | X

  * Classification usually involves predicting discrete variables
  * More precisely, the target 'Y' is categorical
    (e.g., yes vs no, good vs bad, {small, medium, large}, etc.)
  * Often categorical target encoded as +1, -1, 0, etc.
  * Refrain from treating these as numbers without care (why?)

    * Many common ideas / techniques
    * Today's focus: **discriminative classifiers**

Linear classifiers

Binary classification

Assume data is already encoded as features in $\mathbf{R}^p$

Assume labels drawn from $\{+1, -1\}$

Learn a map $h : \mathbf{R}^p \rightarrow \{+1,-1\}$ using training data
Predict using $h(\boldsymbol{x})$ on any test data point $\boldsymbol{x}$

Multiclass classification

Labels in a set {1,2,…,K}

Linear classifiers

Hyperplane:

$$y(x) = w^T x + w_0$$

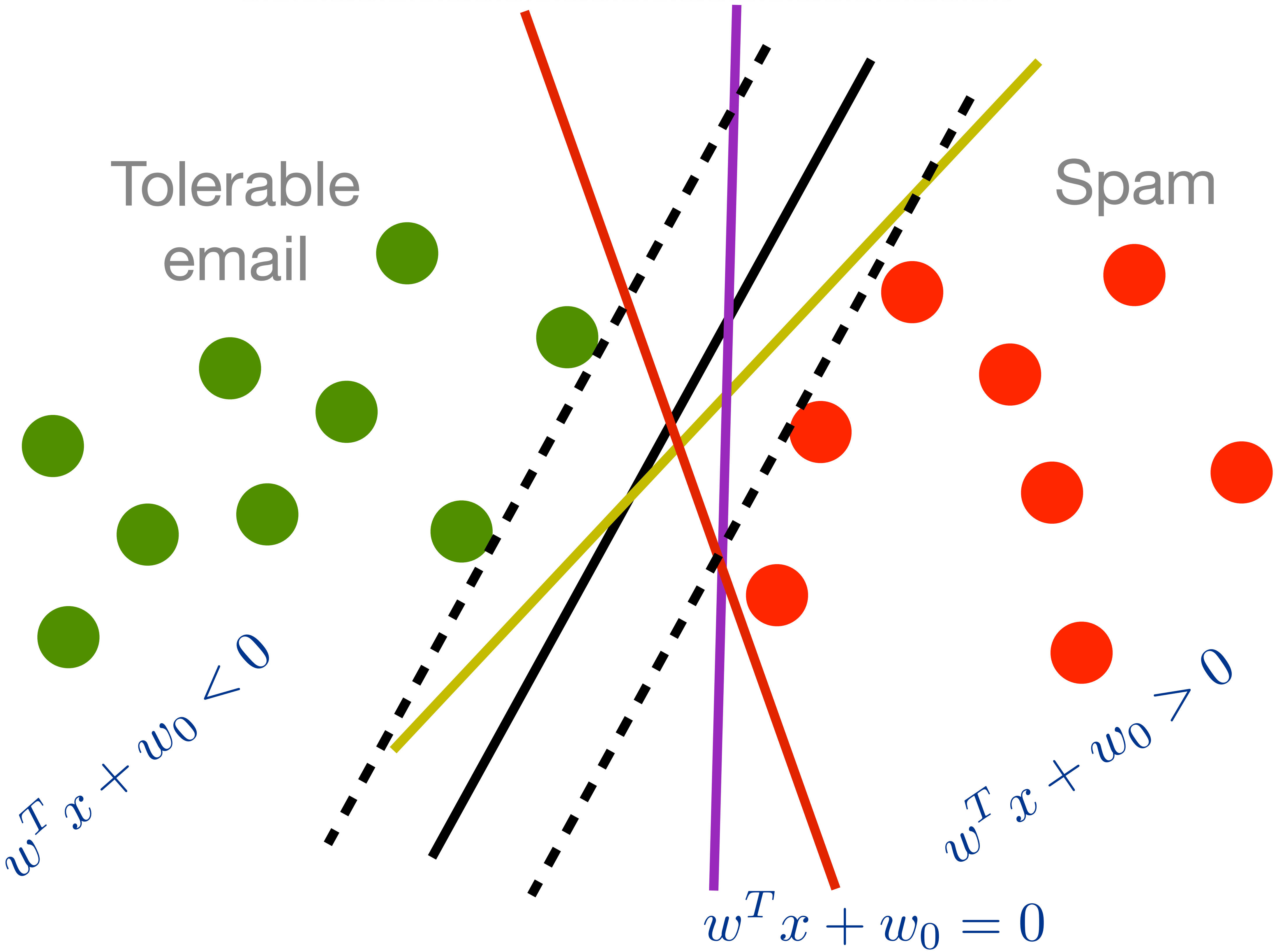
weight vector / parameter

offset / bias

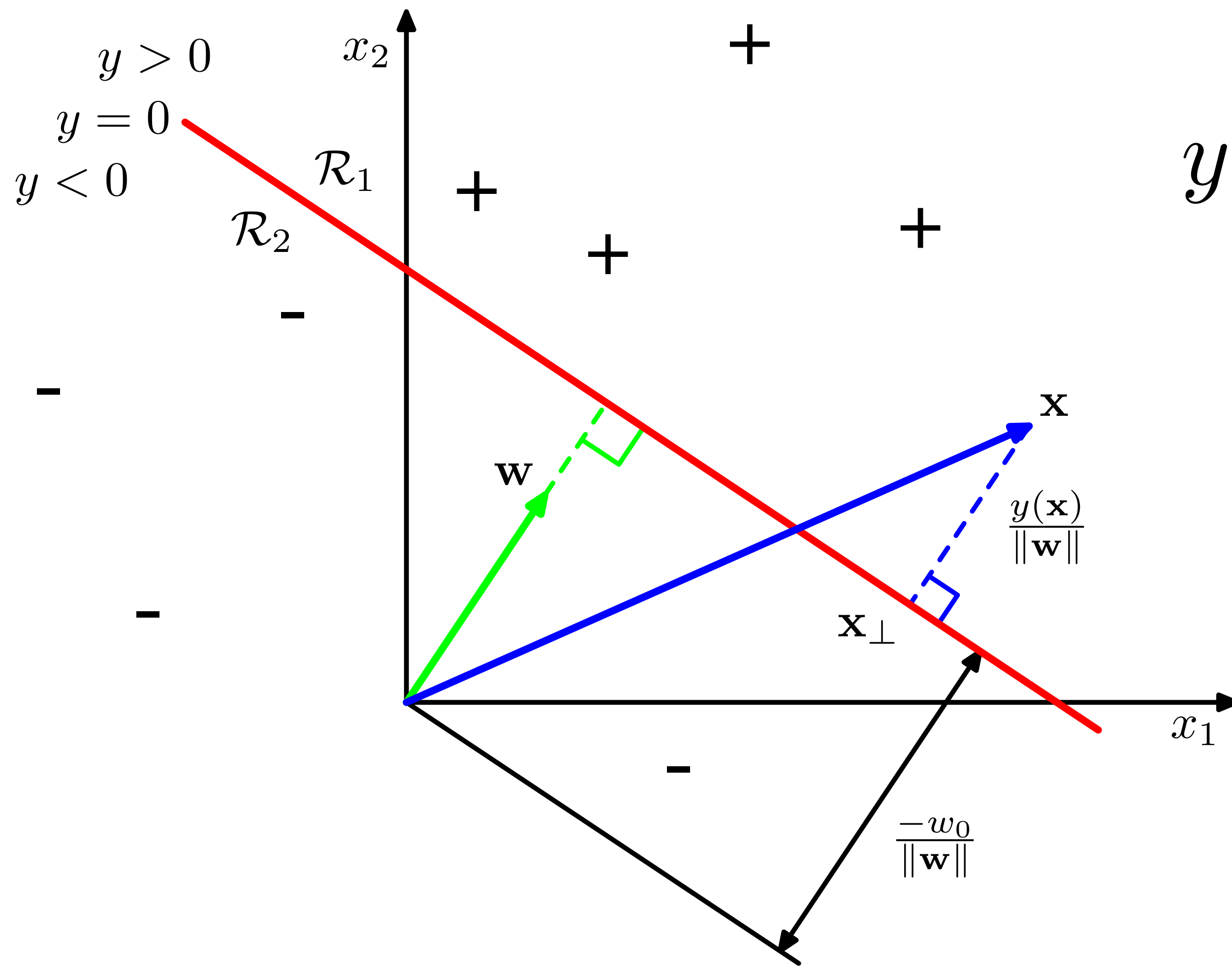
Linear classifier:

$$h(x; w, w_0) = \mathrm{sign}(w^T x + w_0) = \begin{cases} +1, & w^T x + w_0 > 0 \\ -1, & w^T x + w_0 \leq 0. \end{cases}$$

Linear classifiers



Linear classifiers



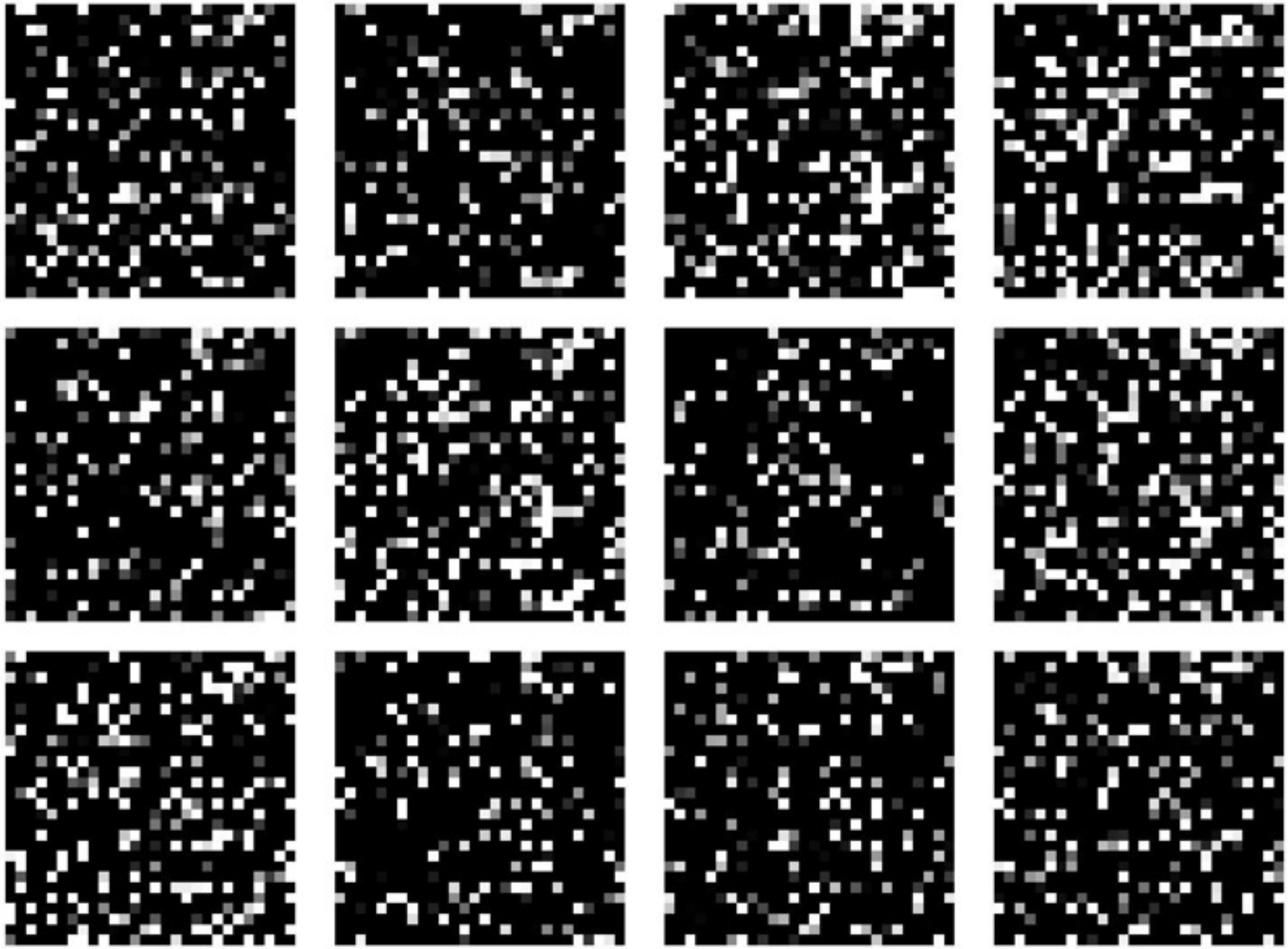
$$y(x) = w^T x + w_0$$

2D example
(Fig 4.1 in Bishop)

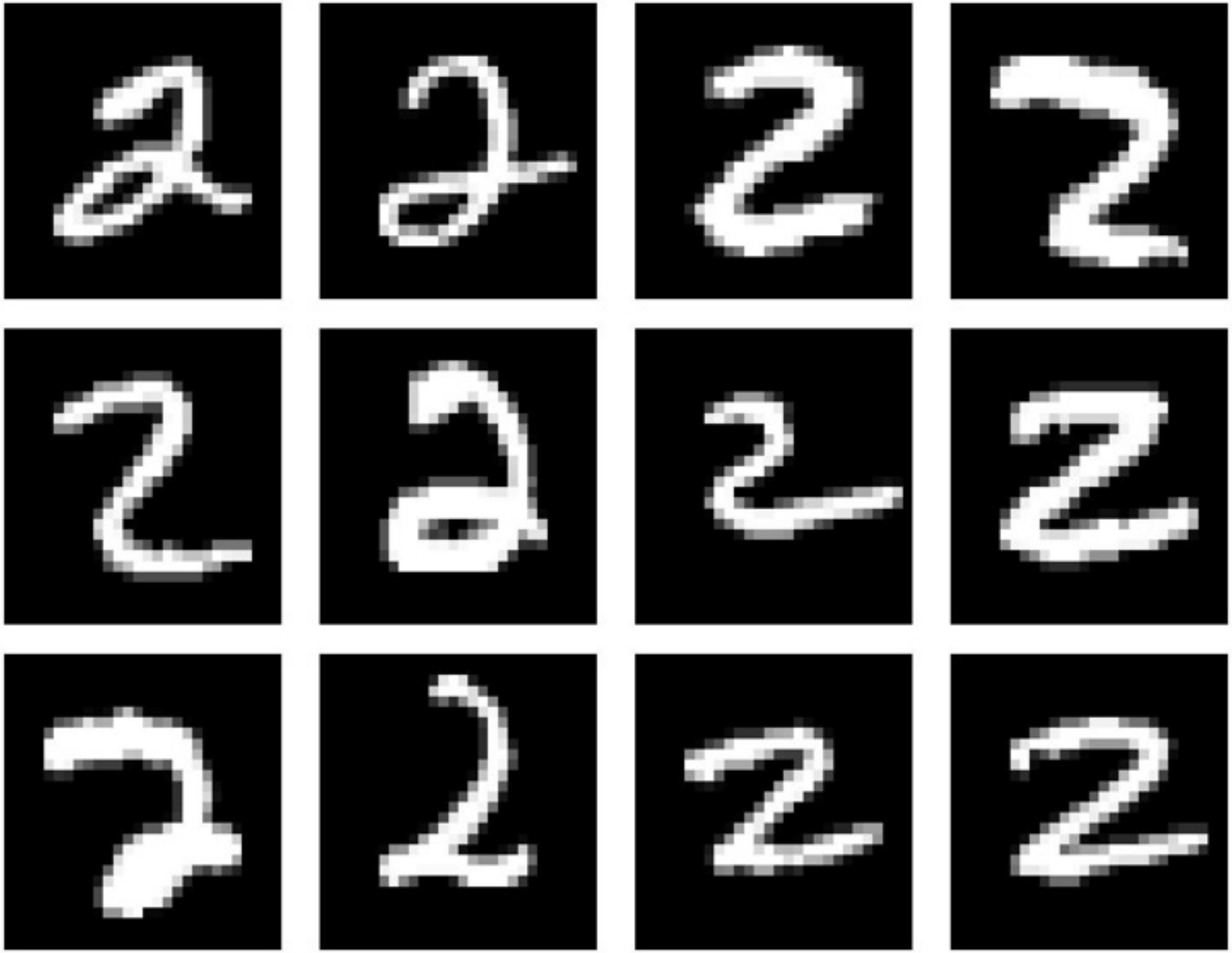
Exercise: Write $x = x_\perp + \gamma \frac{w}{\|w\|}$ and conclude that γ is given by

$$\gamma = \frac{w^T x + w_0}{\|w\|}$$ *(signed distance to the decision hyperplane)*

Are linear classifiers powerful?

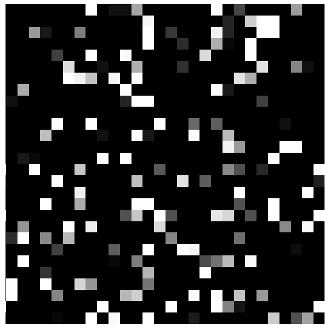


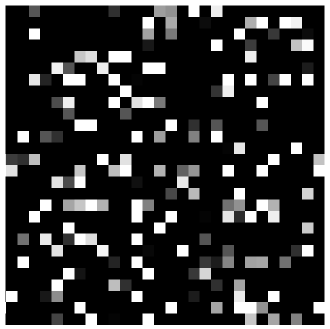
What do you see?

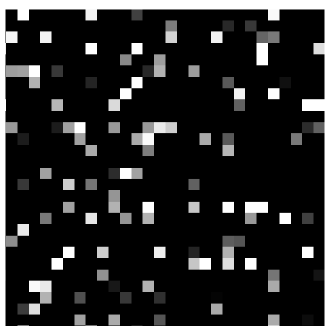


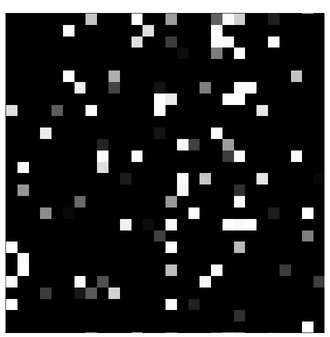
Some samples of digit “2” from MNIST

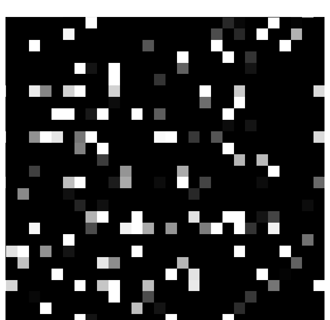
http://yann.lecun.com/exdb/mnist/

( , 5)

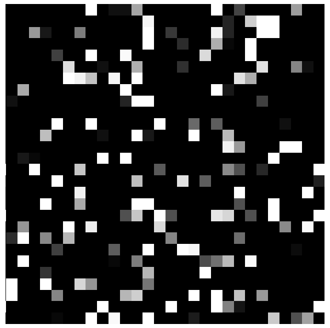
( , 0)

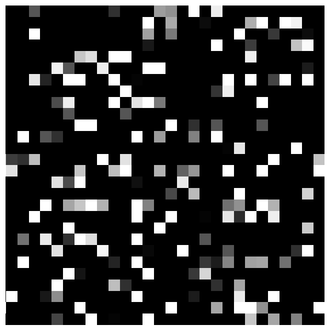
( , 4)

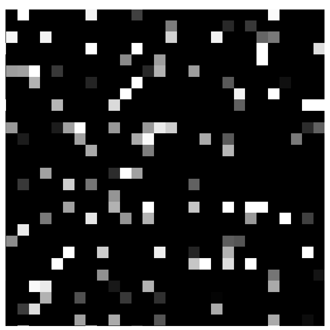
( , 1)

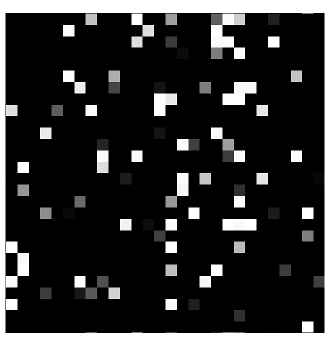
( , 9)

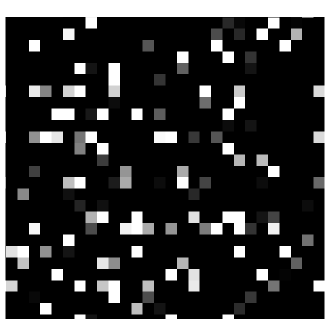
* Training data (x’[i], y[i]) pairs, where x’[i] is a permuted version of x[i]
* Classifier can learn this X ⇒ Y map
* If test data features have undergone same permutation, prediction should work

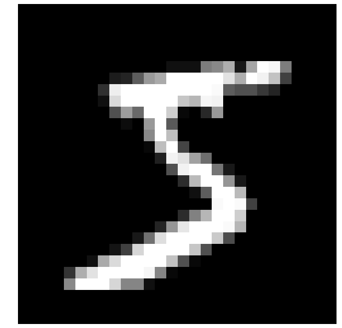
( , 5)

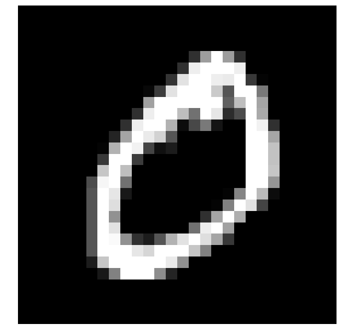
( , 0)

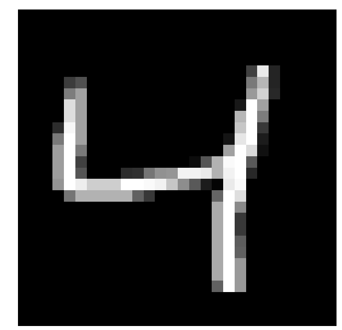
( , 4)

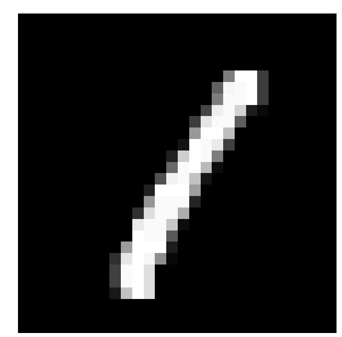
( , 1)

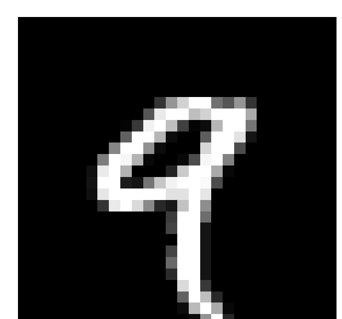
( , 9)

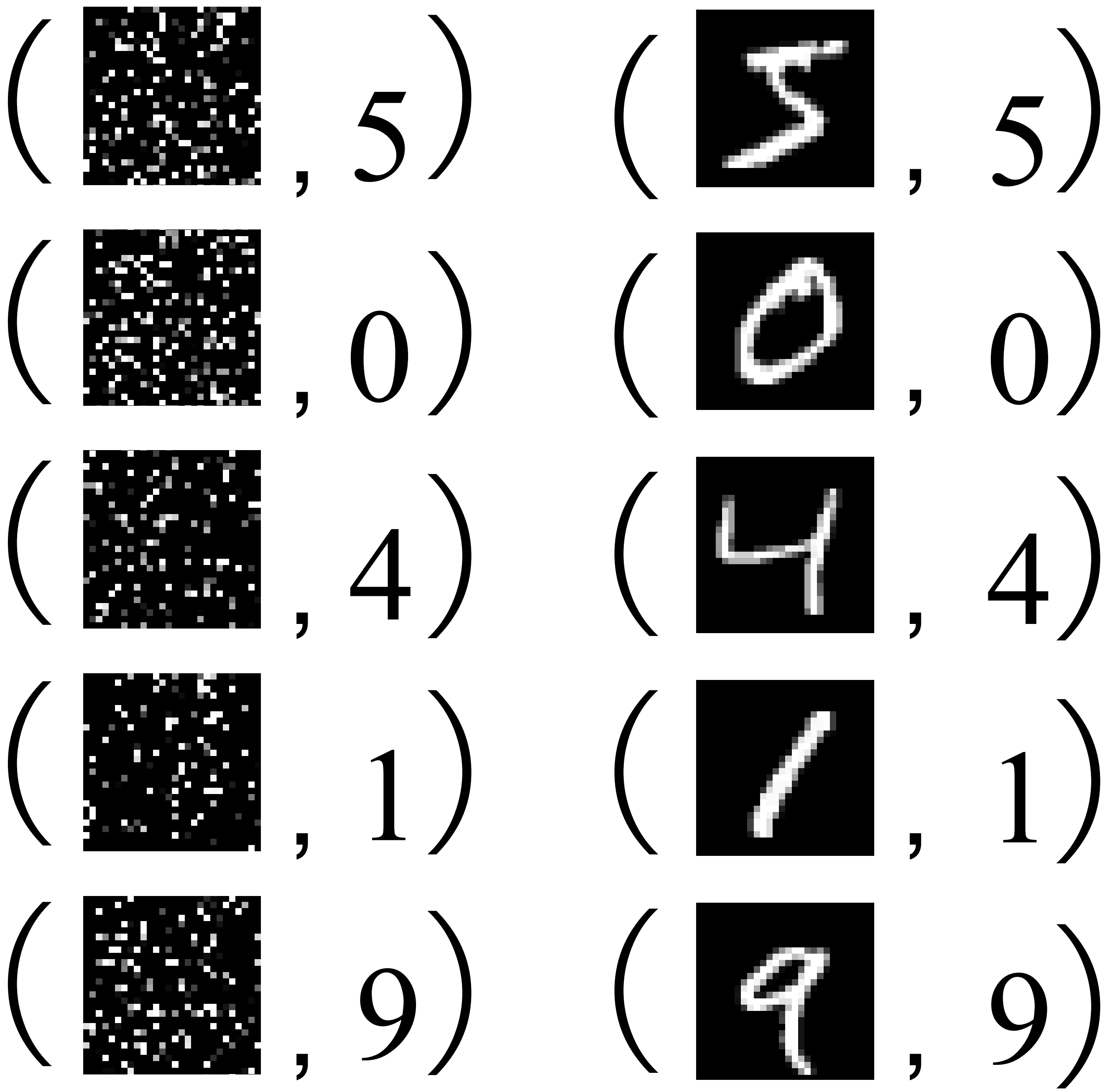
( , 5)

( , 0)

( , 4)

( , 1)

( , 9)



Classifier can learn both equally well!

Exercise: Try this out yourself. What if each digit is permuted differently?

Power of linear classifiers

Observe: If the original features are transformed as $x \rightsquigarrow Ax$, then using $w \rightsquigarrow A^{-1}x$ will work too, if 'w' worked for untransformed data.

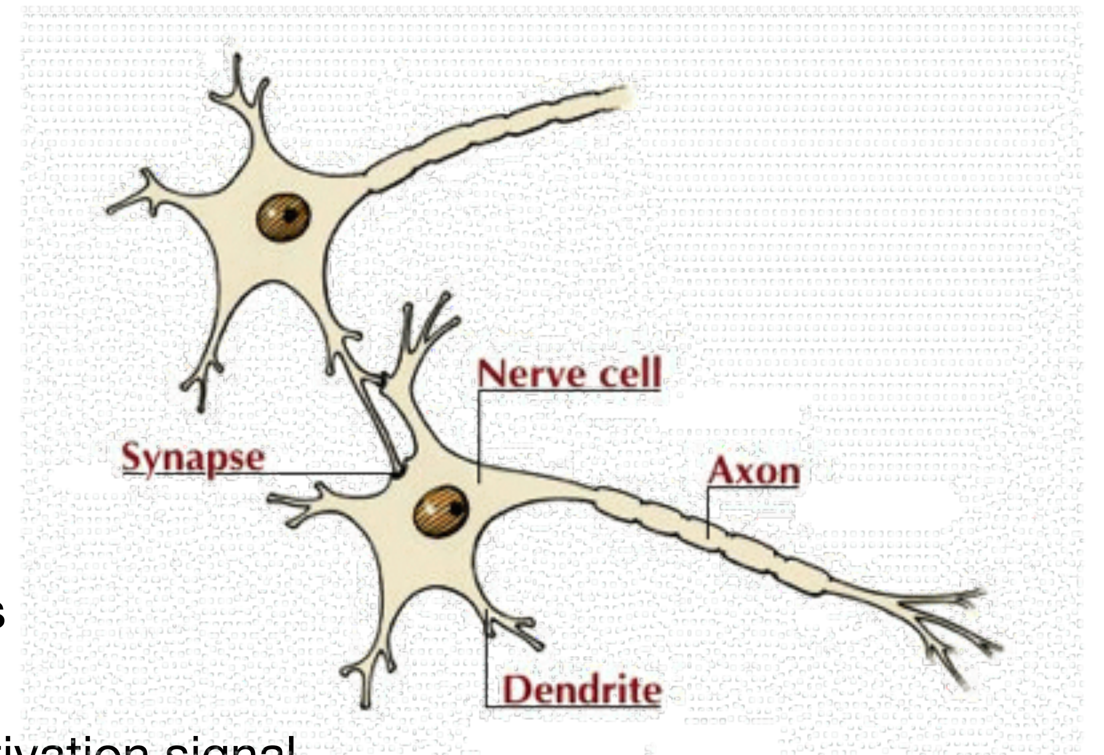
Previous example was random permutation of the features, so it can be written as working with Px, where P is a *permutation matrix*

This invariance can be both good and bad.

Training a linear classifier

Inspiration from biology: *mistake driven*
Bad behavior punished, good rewarded

- Soma (CPU)
  Cell body - combines signals
- Dendrite (input bus)
  Combines the inputs from several other nerve cells
- Synapse (interface)
  Interface and parameter store between neurons
- Axon (cable)
  May be up to 1m long and will transport the activation signal to neurons at different locations



Perceptron (Rosenblatt): Go through training examples one by one, if current classifier (w, w_0) makes a mistake, update it, else do nothing.

Perceptron

ARCHIVES | 1958

Electronic 'Brain' Teaches Itself

SPECIAL TO THE NEW YORK TIMES JULY 13, 1958

The Navy last week demonstrated the embryo of an electronic computer named the Perceptron which, when completed in about a year, is expected to be the first non-living mechanism able to "perceive, recognize and identify its surroundings without human training or control."

July 13, 1958, Page 9
The New York Times Archives

Subscribe and see the full article in TimesMachine

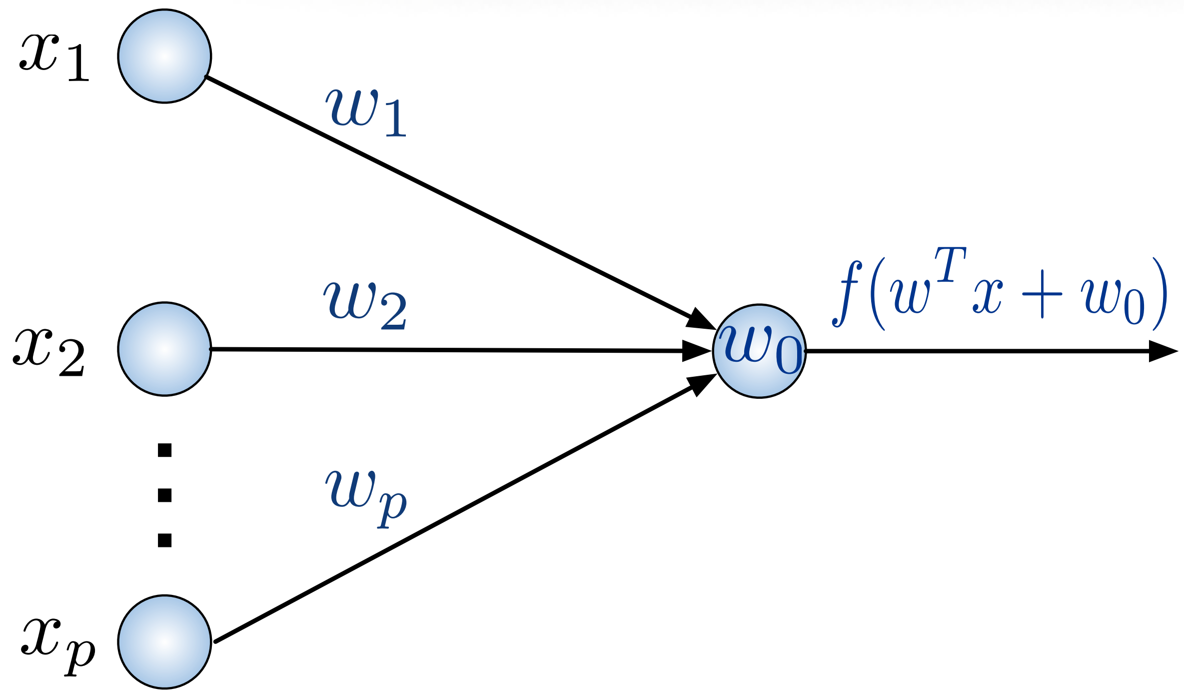
New York Times subscribers enjoy full access to TimesMachine—view 129 years of New York Times journalism, as it originally appeared. **99¢ for your first 4 weeks.**

Or, purchase this article individually for $3.95 and download a high-resolution PDF.

Subscribe   Buy Article

Wikipedia: *"the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence"*

Perceptron



Neural network with 1 neuron :-)

Algorithm:

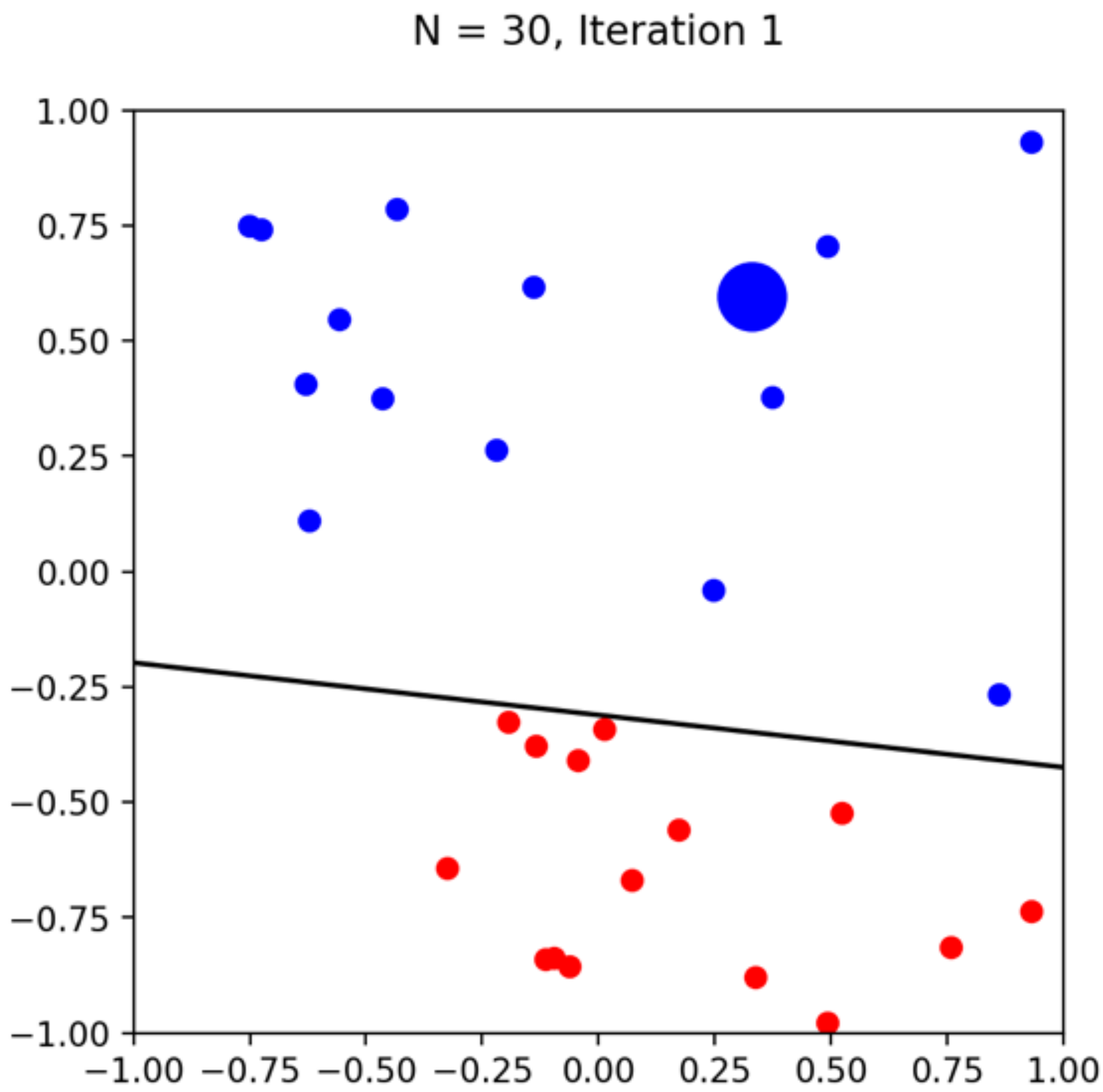
1. Initialize parameters; set iteration counter t = 1.
2. Cycle through training data $(x_1, y_1), \ldots, (x_N, y_N)$ and update

$$\left.\begin{aligned} &\text{if } y_i \neq h(x_i; w^t, w_0^t), \text{ then} \\ &w^{t+1} = w^t + y_i x_i \end{aligned}\right\} \quad y_i(x_i^T w^t + w_0^t) \leq 0$$

$$w_0^{t+1} = w_0^t + y_i$$

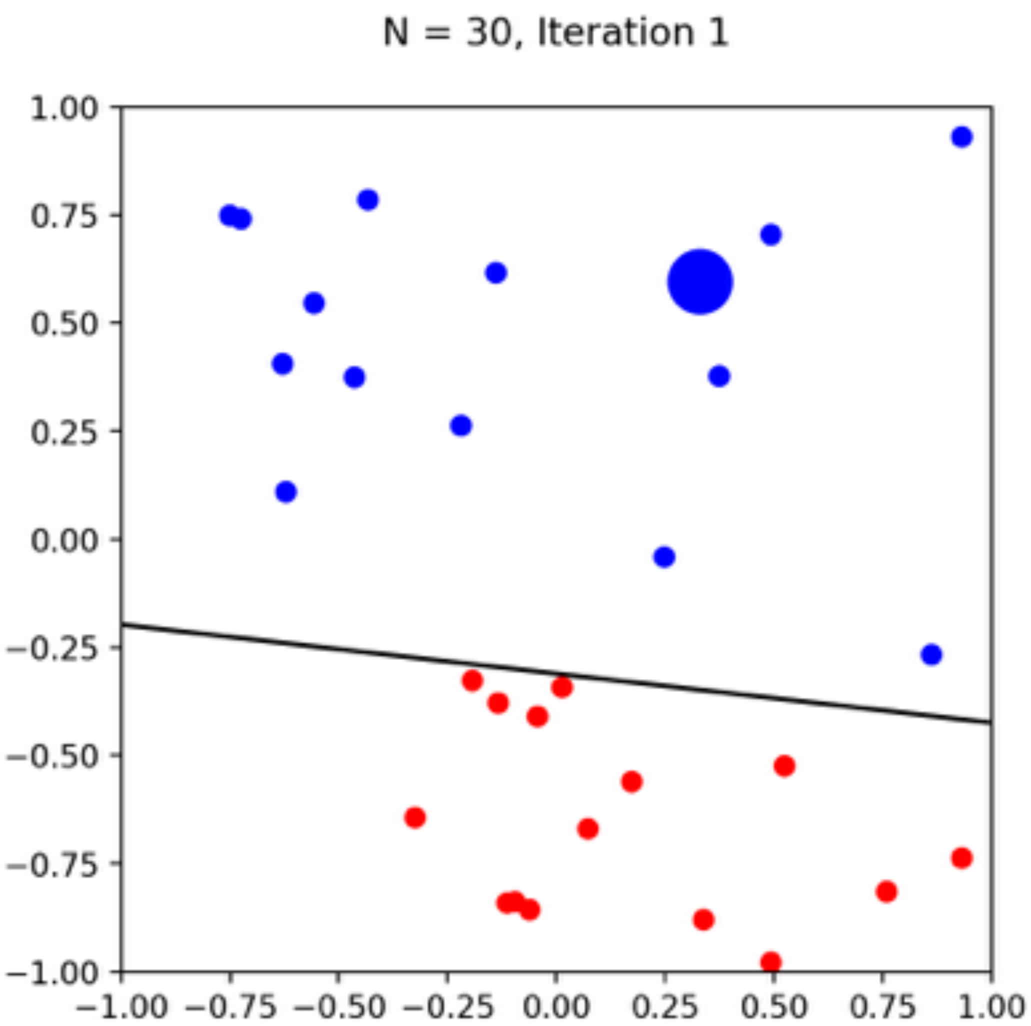
3. Repeat until?

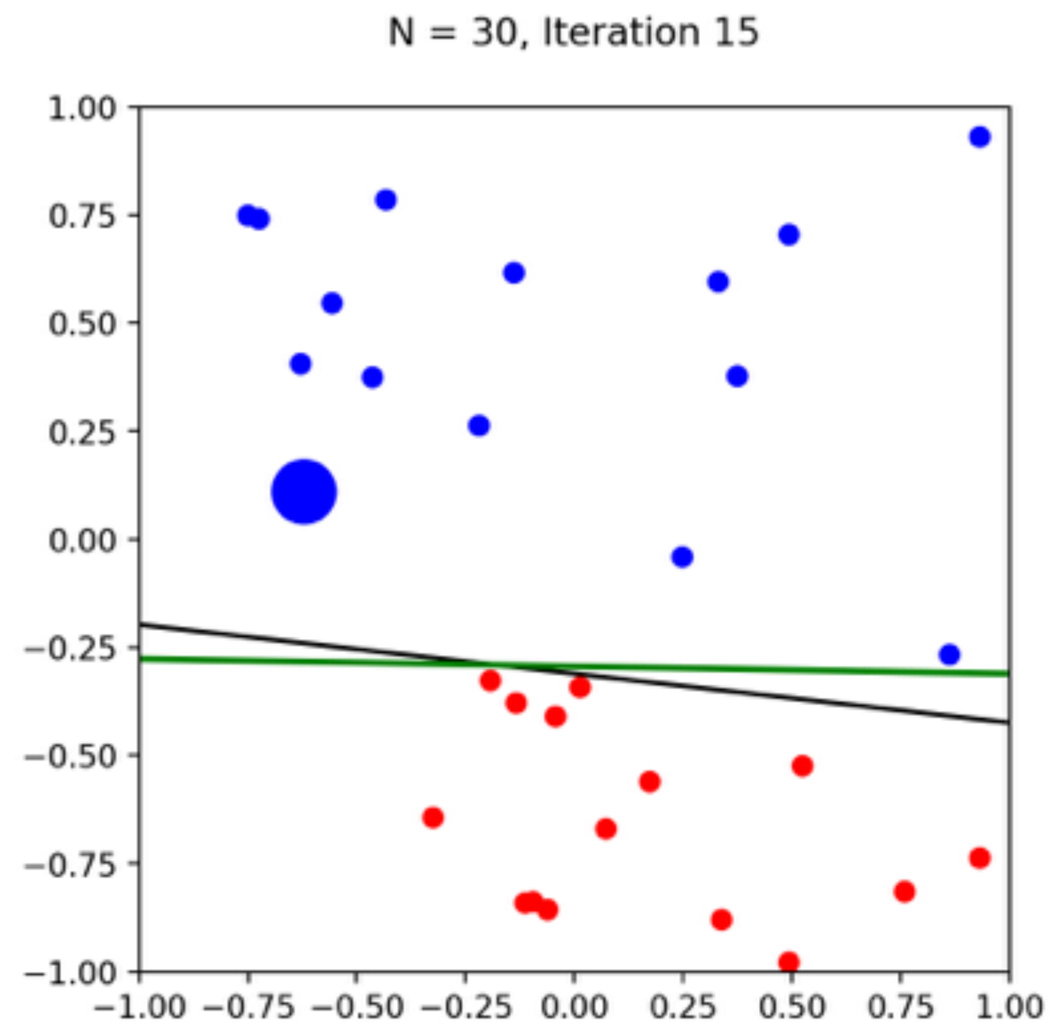
Online algorithm



N = 30, Iteration 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

Linear separability





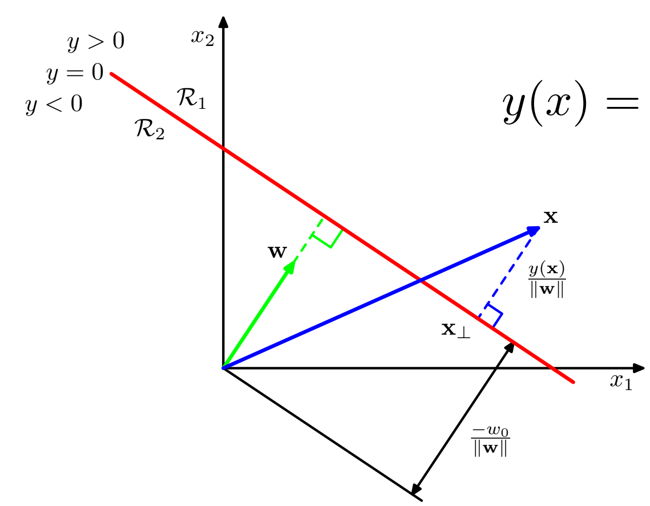
Claim: If the data are linearly separable, then the perceptron is guaranteed to stop in finite number of iterations

Proof of convergence

Assume: $||x_i|| \leq R$ for $1 \leq i \leq N$. *(Augment data as (x,1) to absorb the w0 term)*
Suppose there is unit vector *u* such that $y_i(u^T x_i) \geq \gamma$
for all the training examples.

Observe: linear separability assumption gives us such *u* and γ

We show that in this case, perceptron requires at most $(R/\gamma)^2$ steps



$$y(x) = w^T x + w_0$$

$$\gamma = \frac{w^T x + w_0}{\|w\|}$$

(signed distance to the hyperplane)

- **Proof idea**: we show that w^t keeps getting bigger but is upper bounded, so the process should stop in finite time.
- Let w^t denote the vector prior to the t-th mistake
- If t-th mistake occurs on (x_i, y_i), then

$$y_i(x_i^T w^t) \leq 0, \quad \text{and} \quad w^{t+1} = w^t + y_i x_i$$

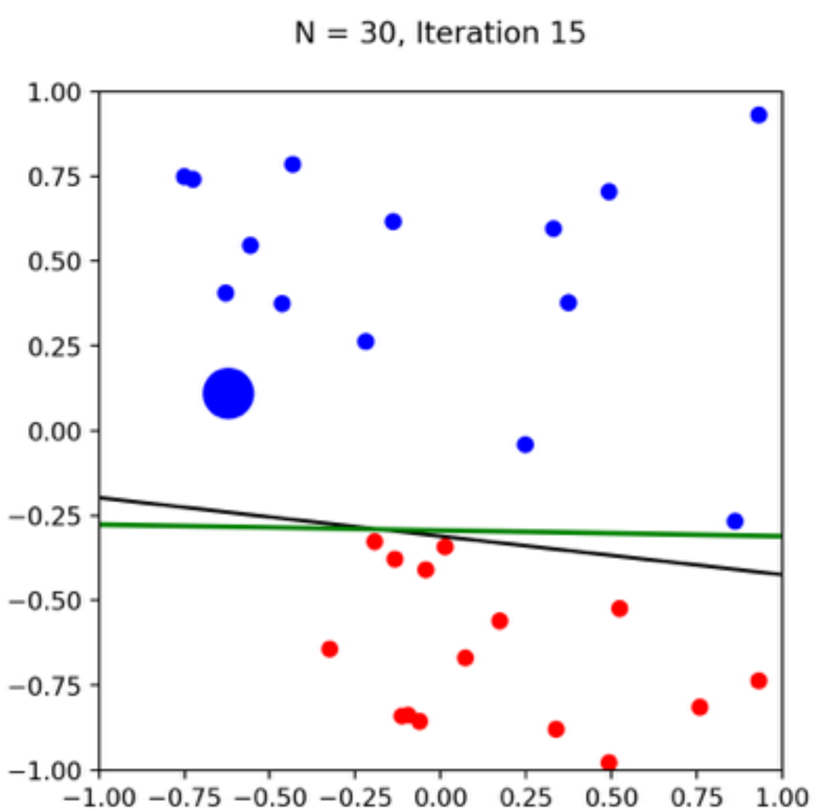
- $u^T w^{t+1} = u^T w^t + y_i(u^T x_i) \geq u^T w^t + \gamma$
- Thus, inductively we obtain $u^T w^{t+1} \geq t\gamma$
- Similarly, $\|w^{t+1}\|^2 = \|w^t\|^2 + \|x_i\|^2 + 2y_i(x_i^T w^t)$, which is bounded by $\|w^t\|^2 + R^2$ (notice last term is negative)
- Thus, inductively we obtain $\|w^{t+1}\|^2 \leq tR^2$.
- $\sqrt{t}R \geq \|w^{t+1}\| \overset{\text{Cauchy-Schwarz}}{\geq} |u^T w^{t+1}| \geq t\gamma$ which implies that

$$t \leq (R/\gamma)^2.$$

Perceptron analysis

This convergence theorem is a remarkable result:

* regardless of order in which we process training data, if they are linearly separable, perceptron will attain 100% training accuracy in ≤ $(R/\gamma)^2$ steps
* the bound is independent of dimensionality of the feature space
* similar analysis can be made for infinite number of data (generalization…)



Weaknesses?

What if data not separable?
What about the hyperplanes learned?
How long does it actually take?

The loss function viewpoint

The perceptron can be seen as implicitly trying to minimize the 0/1-loss, i.e., *misclassification error*.

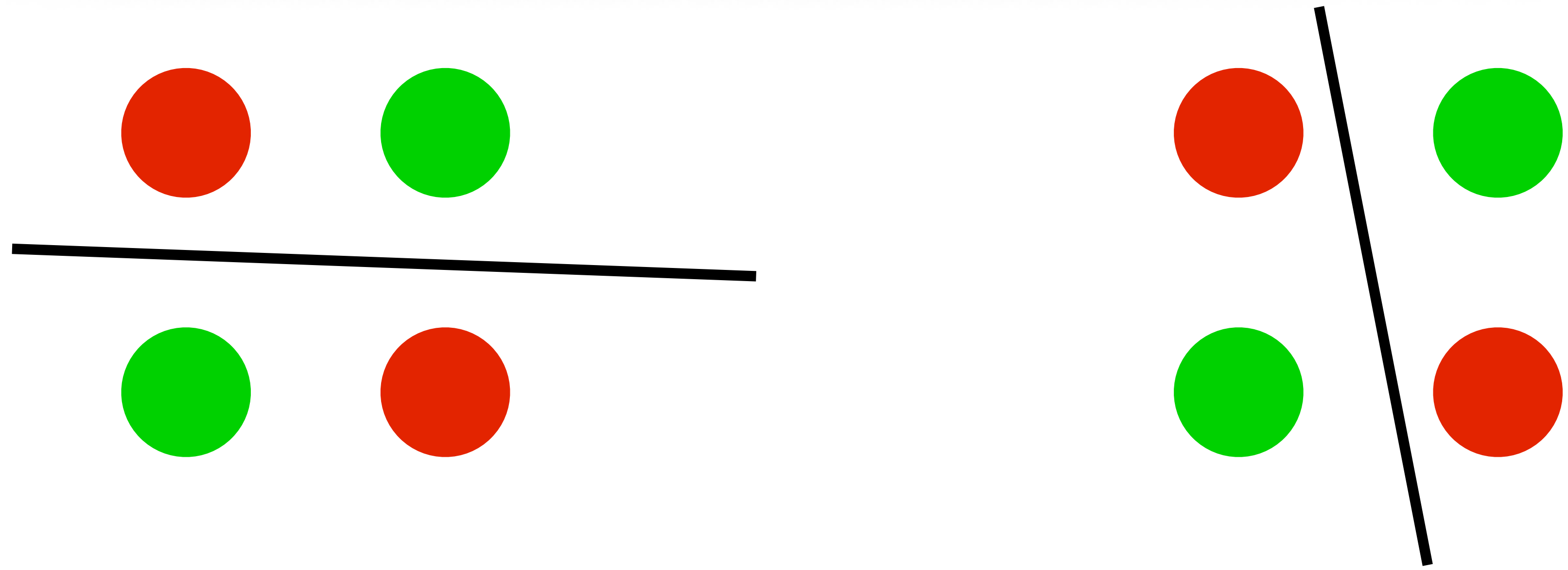
$$\ell_{0/1}(z) := [\![z \leq 0]\!]$$

$$R_{\text{emp}}(w, w_0) := \sum_{i=1}^{N} \ell_{0/1}(y_i(w^T x_i + w_0))$$

Notation: Iverson bracket

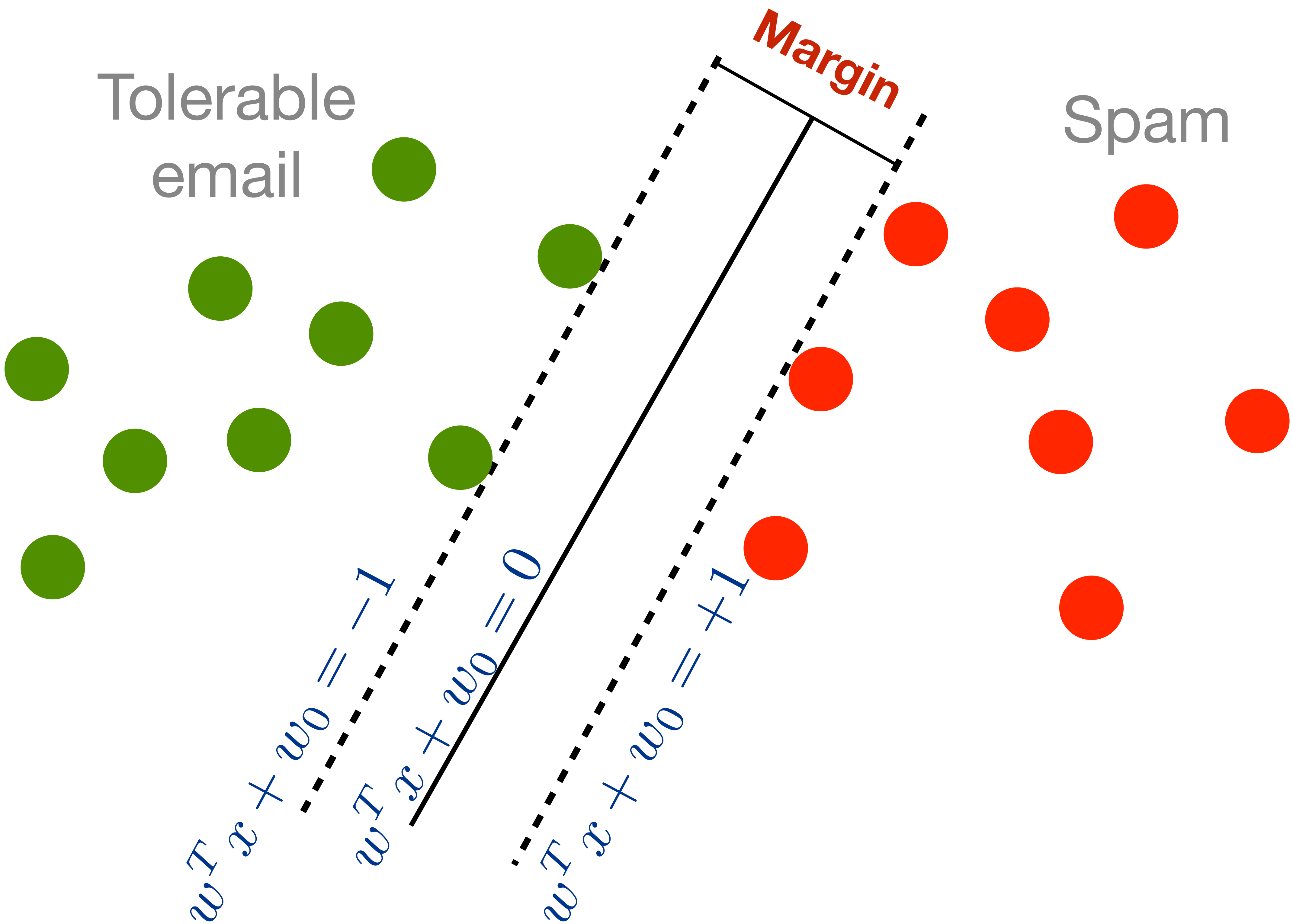
$$[\![\text{predicate}]\!] := \begin{cases} 1 & \text{if predicate is true} \\ 0, & \text{otherwise} \end{cases}$$

Minimum error linear separators



* XOR - not linearly separable
* Nonlinear separation is trivial
* Caveat (Minsky & Papert)
  Finding the *minimum error linear separator*
  is **NP hard** (this killed Neural Networks in the 70s).

Loss function viewpoint: margins



Loss function viewpoint: hinge loss

Not satisfied with just $y_i(w^T x_i + w_0) > 0$,want it at least 1

Hinge loss $\qquad \ell_h(z) := \max(0, 1 - z)$

$$R_{\text{emp}}(w, w_0) := \sum_{i=1}^{N} \ell_h(y_i(w^T x_i + w_0))$$

This is an **Empirical Risk Minimization** (ERM) problem;
the most important optimization problem in supervised learning

ERM with hinge loss

$$R_{\text{emp}}(w, w_0) := \sum_{i=1}^{N} \ell_h(y_i(w^T x_i + w_0))$$

Minimize using SGD (to be pedantic: **sub**gradient needed)

Algorithm:

1. Initialize weights. Set t = 0
2. For t=0,1,2,…
   2.1. Pick i in {1,…,N}
   2.2. Obtain subgradient of $\ell_h(y_i x_i^T w^t)$
   2.3. Update $w^{t+1} = w^t - \eta_t g_t$

Stochastic convex optimization problem; SGD can be shown to converge to an optimum at the rate $O(1/\sqrt{T})$ (T updates).