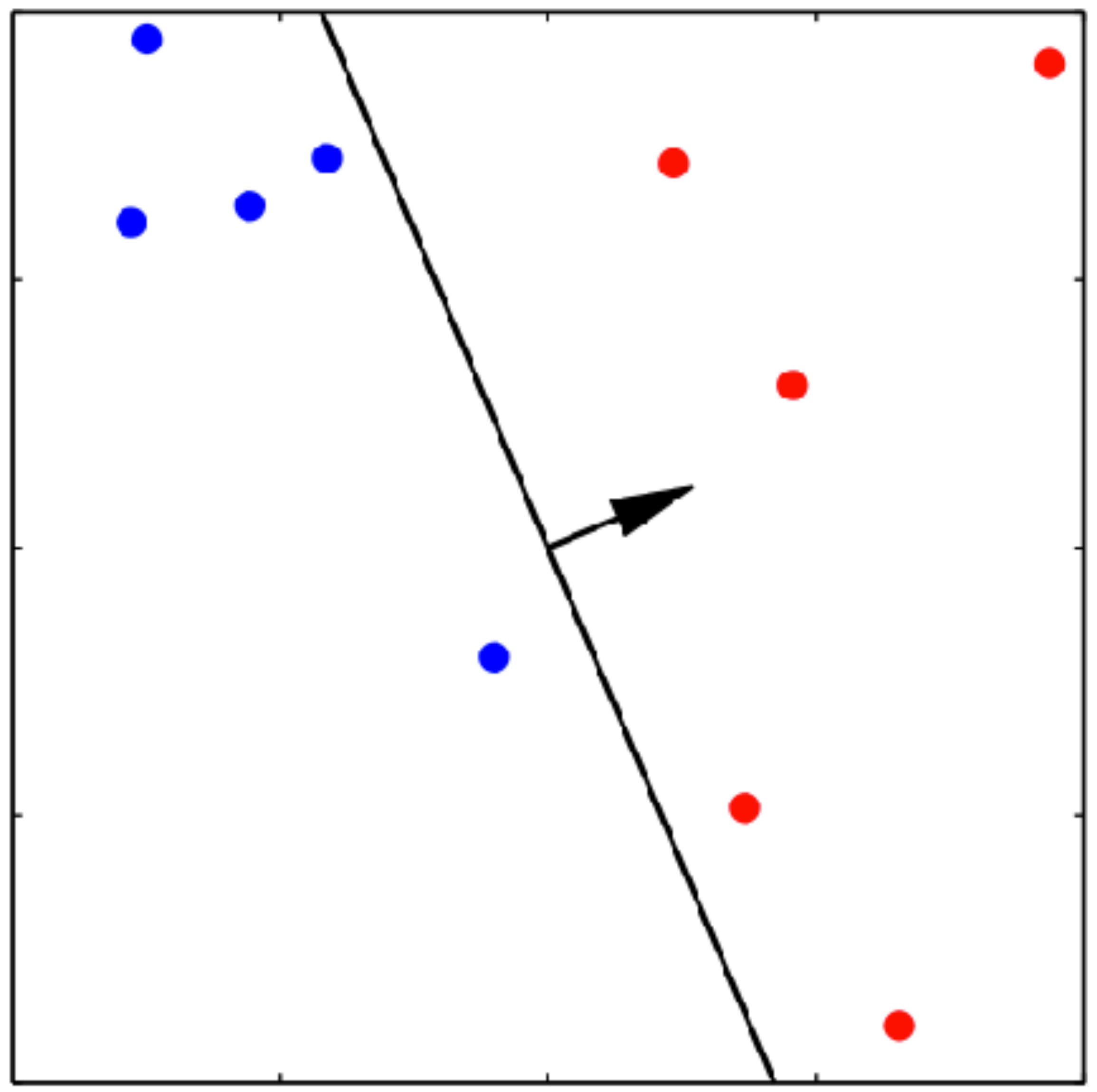
Applied Machine Learning

CIML Chaps 4-5 (A Geometric Approach)



"A ship in port is safe, but that is not what ships are for."

– Grace Hopper (1906-1992)

Week 5: Extensions and Variations of Perceptron, and Practical Issues

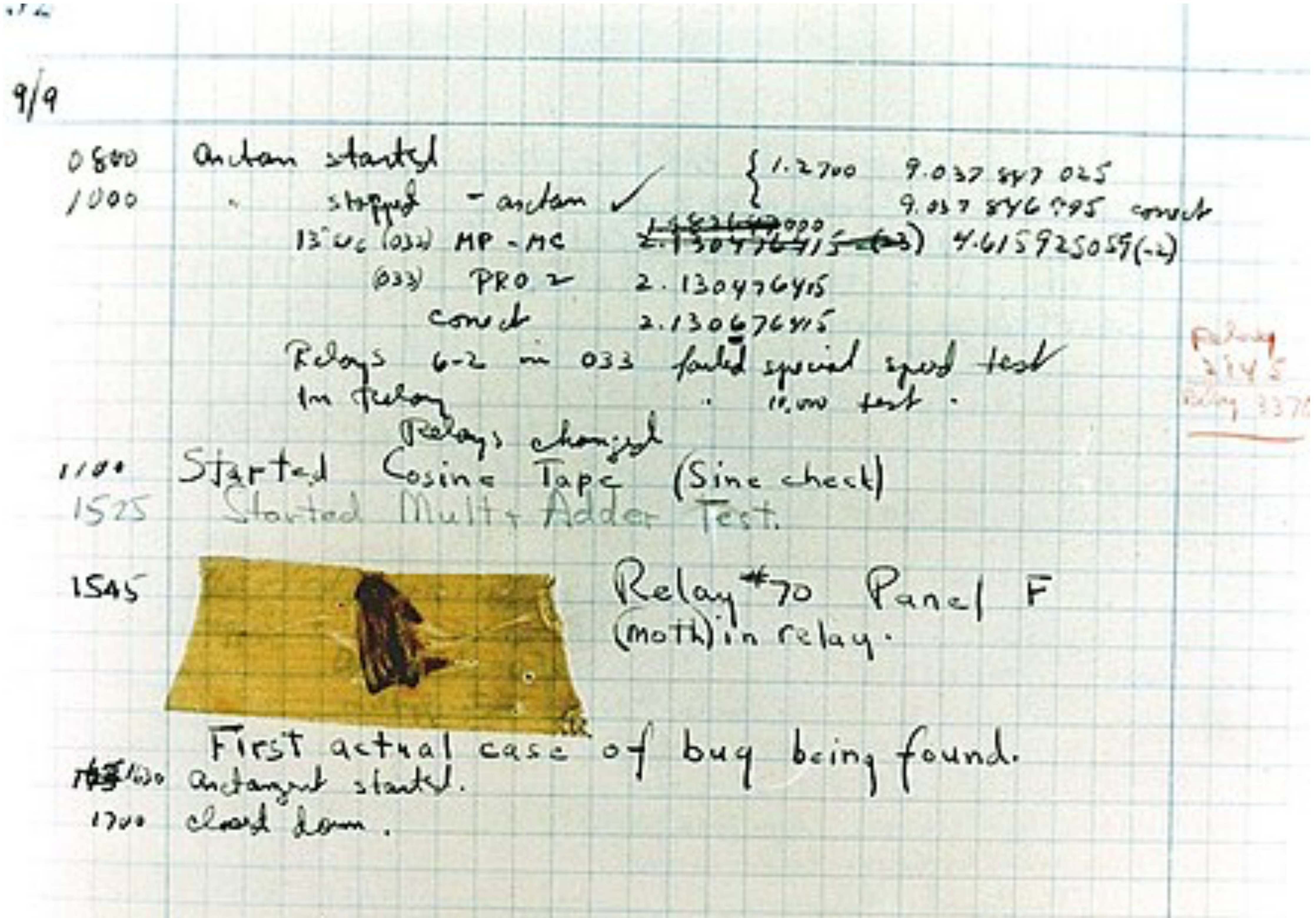
Professor Liang Huang

some slides from A. Zisserman (Oxford)

Trivia: Grace Hopper and the first bug

- Edison coined the term "bug" around 1878 and since then it had been widely used in engineering
- Hopper was associated with the discovery of the first computer bug in 1947 which was a moth stuck in a relay

ANITA BORG INSTITUTE
WOMEN TRANSFORMING TECHNOLOGY
GRACE HOPPER



Smithsonian National Museum of American History

Week 5: Perceptron in Practice

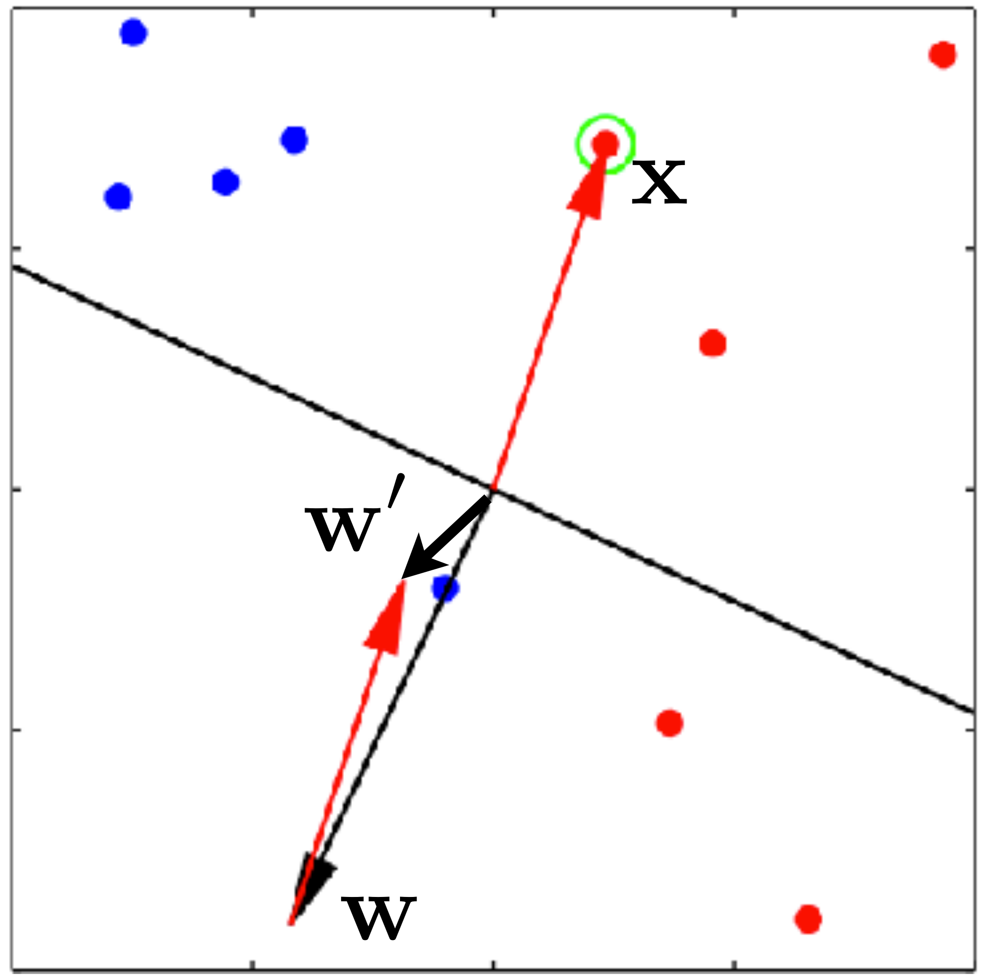
"A ship in port is safe, but that is not what ships are for."

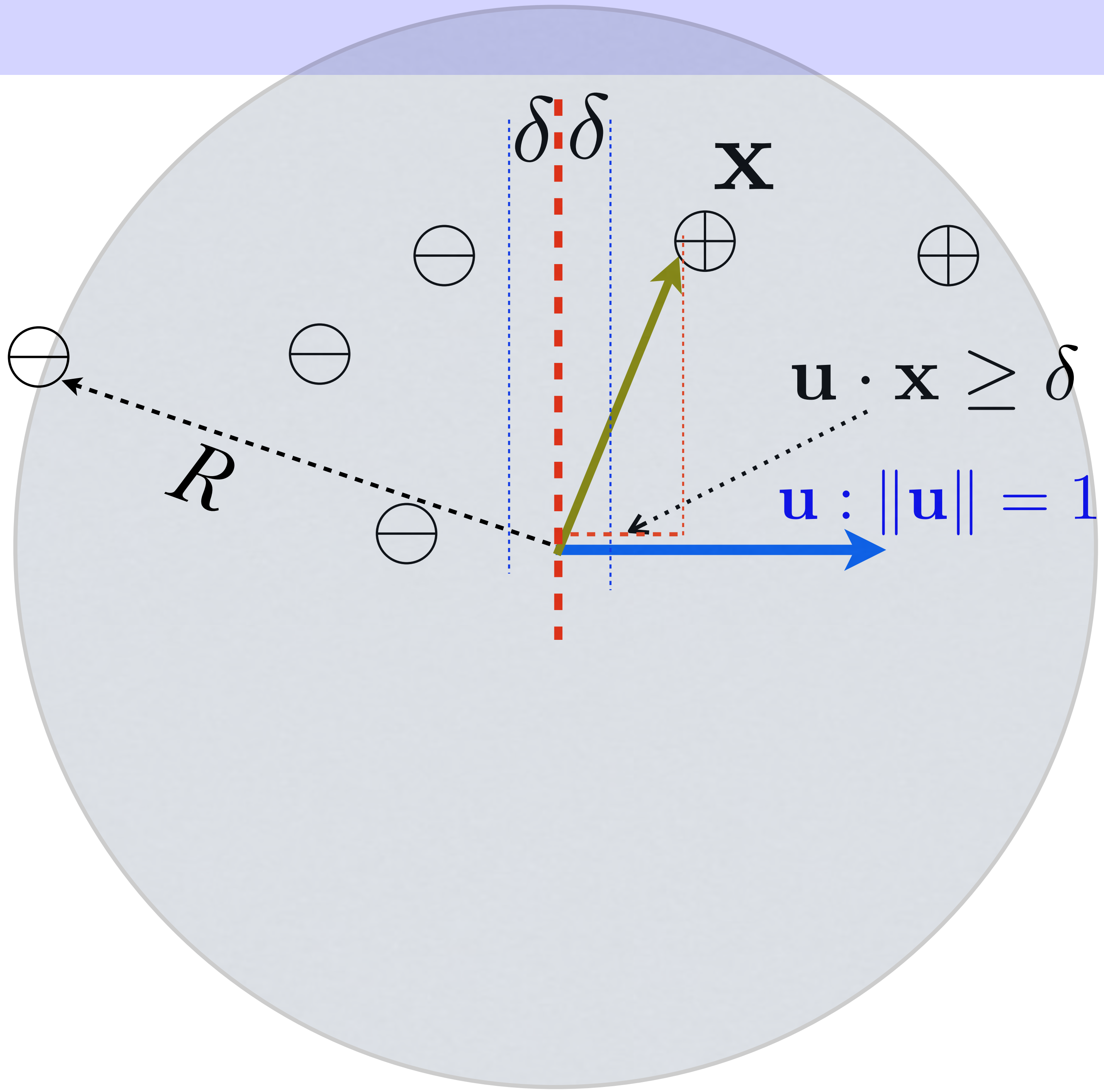
– Grace Hopper (1906-1992)

- Problems with Perceptron
  - doesn't converge with inseparable data
  - update might often be too "bold"
  - doesn't optimize margin
  - result is sensitive to the order of examples
- Ways to alleviate these problems (without SVM/kernels)
  - Part II: voted perceptron and average perceptron
  - Part III: MIRA (margin-infused relaxation algorithm)
- Part IV: Practical Issues and HW1
- Part V: "Soft" Perceptron: Logistic Regression

Recap of Week 4

input: training data D
output: weights $\mathbf{w}$
initialize $\mathbf{w} \leftarrow \mathbf{0}$
while not converged
  for $(\mathbf{x}, y) \in D$
    if $y(\mathbf{w} \cdot \mathbf{x}) \leq 0$
      $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$





“idealized” ML

Input $\mathbf{x}$ → Training → Model $\mathbf{w}$
Output y → Training

“actual” ML

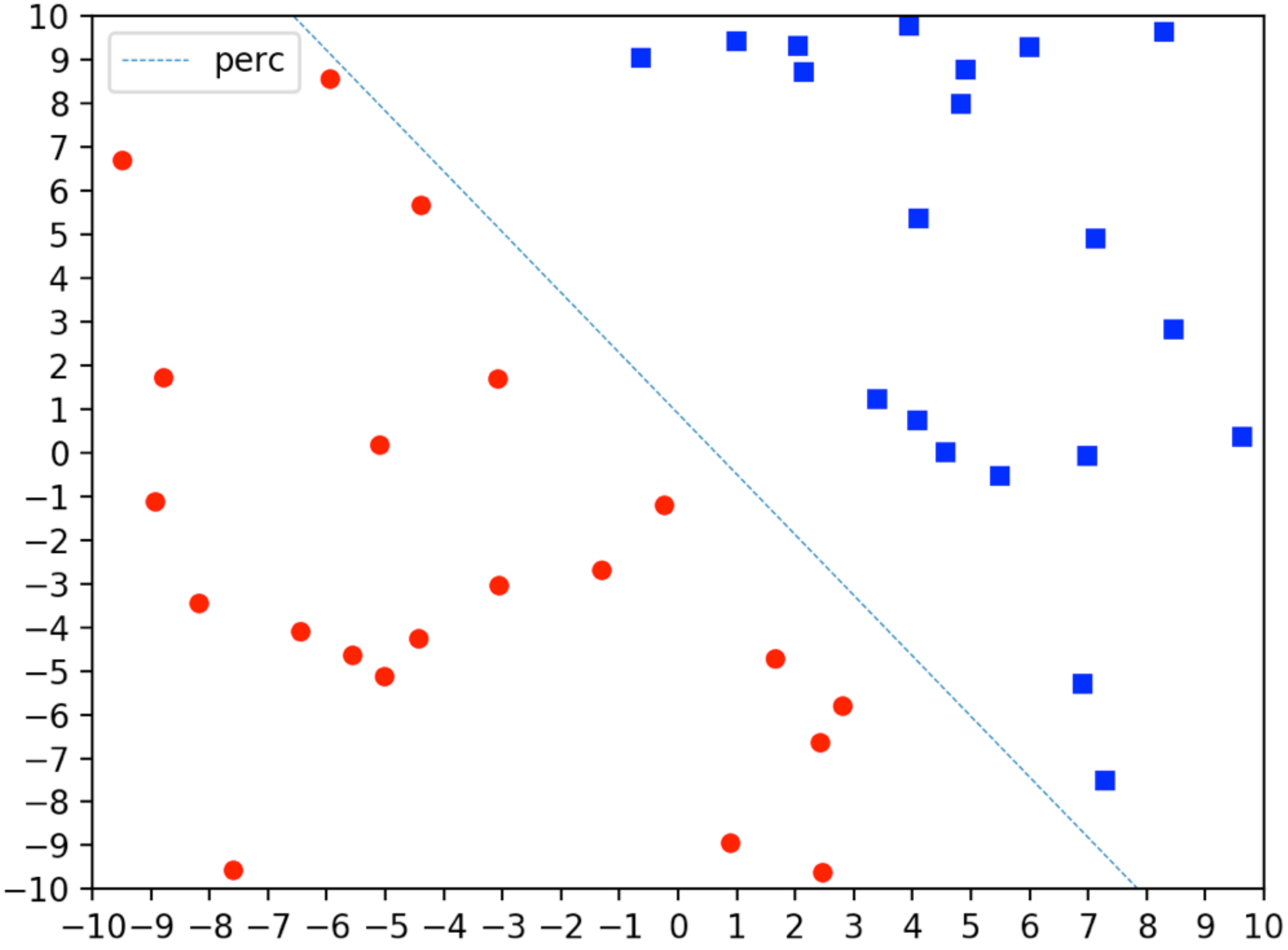
Input $\mathbf{x}$ → Training → Model $\mathbf{w}$
Output y → Training
feature map $\boldsymbol{\phi}$ → Training

deep learning ≈ representation learning

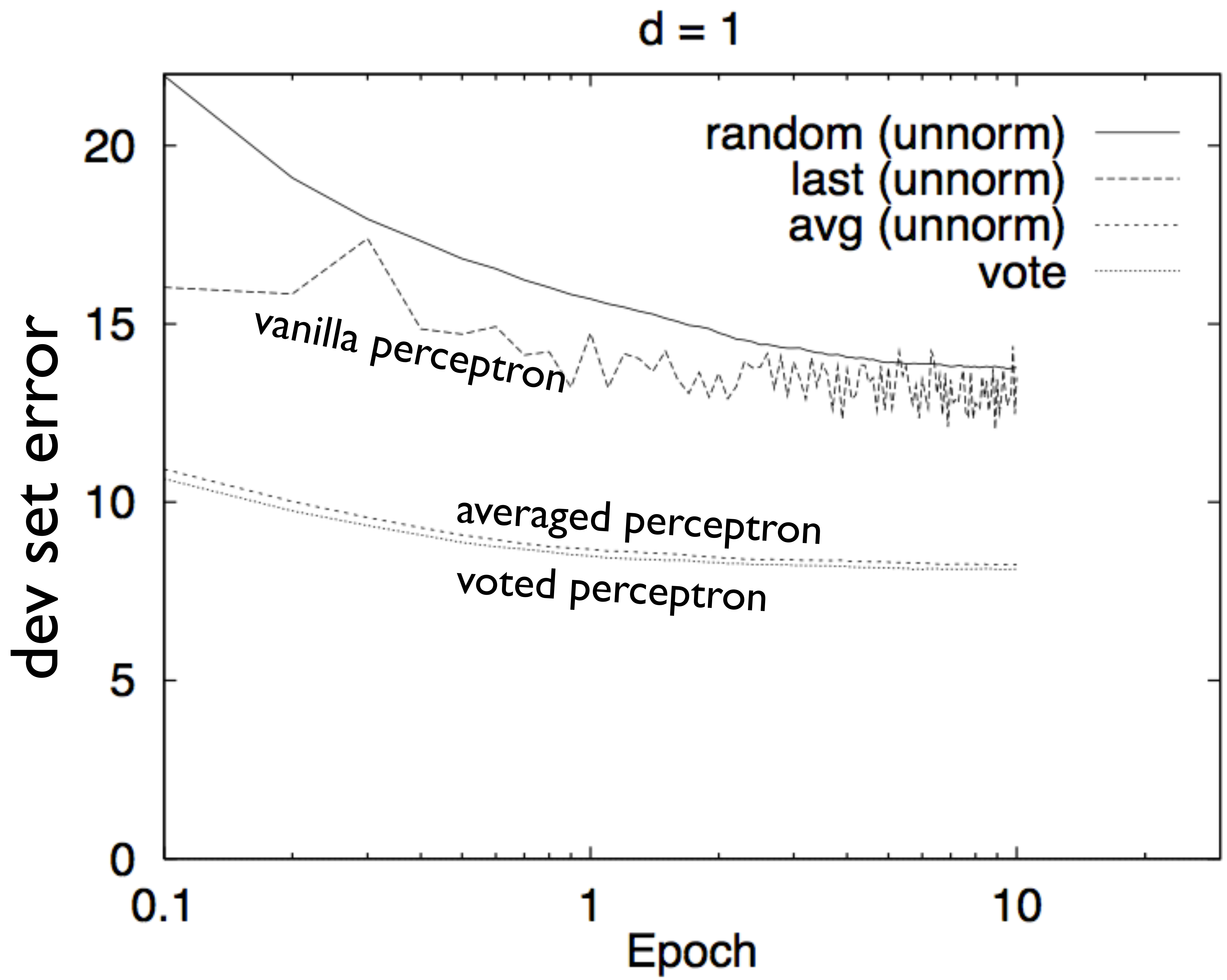
Input $\mathbf{x}$ → Training → feature map $\boldsymbol{\phi}$
Output y → Training → Model $\mathbf{w}$

Python Demo

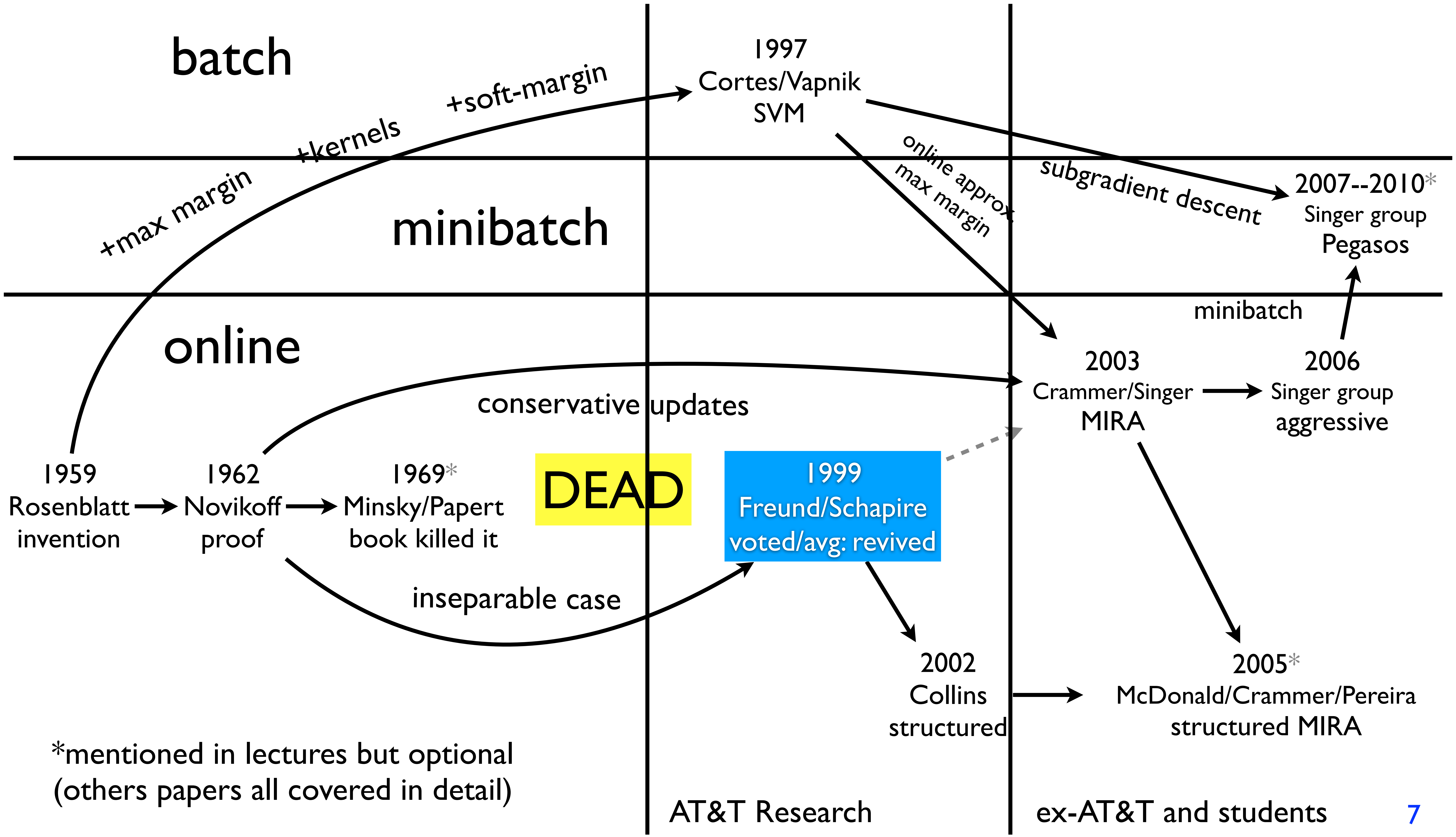
`$ python perc_demo.py` (requires numpy and matplotlib)



Part II: Voted and Averaged Perceptron



Brief History of Perceptron



Voted/Avged Perceptron

- problem: later examples dominate earlier examples
- solution: voted perceptron (Freund and Schapire, 1999)
  - record the weight vector after each example in D
    - <u>not</u> just after each update!
  - and vote on a new example using $|D|$ models
  - shown to have better generalization power
- averaged perceptron (from the same paper)
  - an approximation of voted perceptron
  - just use the average of all weight vectors
  - can be implemented efficiently

Voted Perceptron

Input: a labeled training set $\langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\rangle$
number of epochs T
Output: a list of weighted perceptrons $\langle(\mathbf{v}_1, c_1), \ldots, (\mathbf{v}_k, c_k)\rangle$

- Initialize: $k := 0$, $\mathbf{v}_1 := \mathbf{0}$, $c_1 := 0$.
- Repeat T times:
  - For $i = 1, \ldots, m$:
    * Compute prediction: $\hat{y} := \text{sign}(\mathbf{v}_k \cdot \mathbf{x}_i)$
    * If $\hat{y} = y$ then $c_k := c_k + 1$.
      else $\mathbf{v}_{k+1} := \mathbf{v}_k + y_i \mathbf{x}_i$;
      $c_{k+1} := 1$;
      $k := k + 1$.

our notation: $(\mathbf{x}^{(i)}, y^{(i)})$

v is weight,
c is its # of votes

Large Margin Classification Using the Perceptron Algorithm

YOAV FREUND yoav@research.att.com
AT&T Labs, Shannon Laboratory, 180 Park Avenue, Room A205, Florham Park, NJ 07932-0971

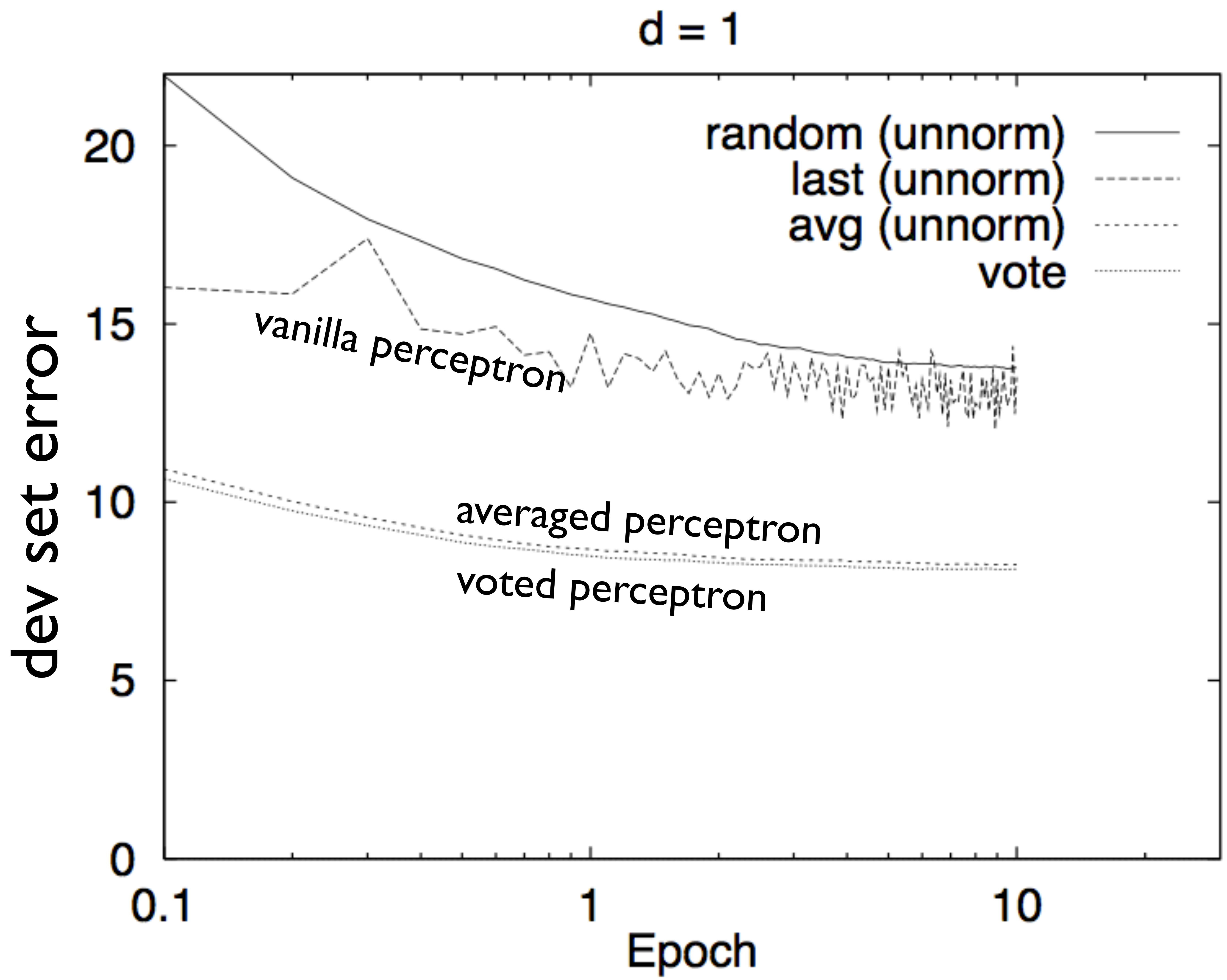
ROBERT E. SCHAPIRE schapire@research.att.com
AT&T Labs, Shannon Laboratory, 180 Park Avenue, Room A279, Florham Park, NJ 07932-0971

Prediction
Given: the list of weighted perceptrons: $\langle(\mathbf{v}_1, c_1), \ldots, (\mathbf{v}_k, c_k)\rangle$
an unlabeled instance: $\mathbf{x}$
compute a predicted label $\hat{y}$ as follows:

$$s = \sum_{i=1}^{k} c_i \,\text{sign}(\mathbf{v}_i \cdot \mathbf{x}); \qquad \hat{y} = \text{sign}(s)\ .$$

if correct, increase the current model's # of votes; otherwise create a new model with 1 vote

Experiments



Averaged Perceptron

- voted perceptron is not scalable
  - and does not output a single model
- avg perceptron is an approximation of voted perceptron
  - actually, summing all weight vectors is enough; no need to divide

$$
\begin{array}{l}
\textbf{initialize } \mathbf{w} \leftarrow \mathbf{0};\ \mathbf{w}_s \leftarrow \mathbf{0} \\
\textbf{while} \text{ not converged} \\
\quad \textbf{for } (\mathbf{x}, y) \in D \\
\qquad \textbf{if } y(\mathbf{w} \cdot \mathbf{x}) \leq 0 \\
\qquad\quad \mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x} \\
\qquad \mathbf{w}_s \leftarrow \mathbf{w}_s + \mathbf{w} \\
\textbf{output: } \text{summed weights } \mathbf{w}_s
\end{array}
$$

after each example, not after each update!

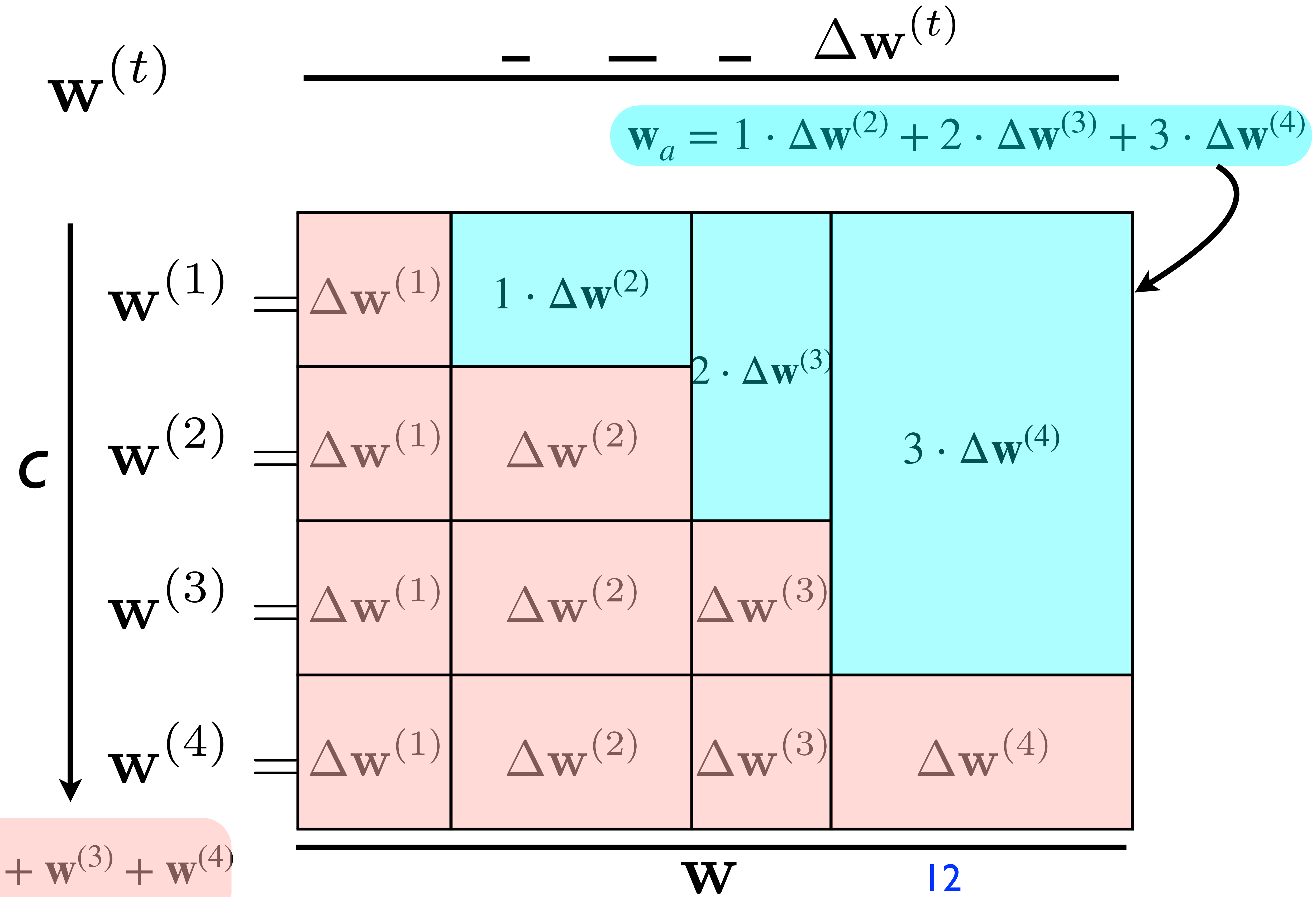
$$
\begin{array}{l}
\mathbf{w}^{(1)} = \Delta\mathbf{w}^{(1)} \\
\mathbf{w}^{(2)} = \Delta\mathbf{w}^{(1)}\ \ \Delta\mathbf{w}^{(2)} \\
\mathbf{w}^{(3)} = \Delta\mathbf{w}^{(1)}\ \ \Delta\mathbf{w}^{(2)}\ \ \Delta\mathbf{w}^{(3)} \\
\mathbf{w}^{(4)} = \Delta\mathbf{w}^{(1)}\ \ \Delta\mathbf{w}^{(2)}\ \ \Delta\mathbf{w}^{(3)}\ \ \Delta\mathbf{w}^{(4)}
\end{array}
$$

Efficient Implementation of Averaging

- naive implementation (running sum $\mathbf{w}_s$) doesn't scale either
  - OK for low dim. (HW1); too slow for high-dim. (HW3)
- very clever trick from Hal Daumé (2006, PhD thesis)

$$
\begin{array}{l}
\textbf{initialize } \mathbf{w} \leftarrow \mathbf{0};\ \mathbf{w}_a \leftarrow \mathbf{0};\ c \leftarrow 0 \\
\textbf{while} \text{ not converged} \\
\quad \textbf{for } (\mathbf{x}, y) \in D \\
\qquad \textbf{if } y(\mathbf{w} \cdot \mathbf{x}) \leq 0 \\
\qquad\quad \mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x} \\
\qquad\quad \mathbf{w}_a \leftarrow \mathbf{w}_a + cy\mathbf{x} \\
\qquad c \leftarrow c + 1 \\
\textbf{output: } c\mathbf{w} - \mathbf{w}_a
\end{array}
$$

after each update, not after each example!



$$c \cdot \mathbf{w} - \mathbf{w}_a = 4 \cdot \Delta\mathbf{w}^{(1)} + 3 \cdot \Delta\mathbf{w}^{(2)} + 2 \cdot \Delta\mathbf{w}^{(3)} + 1 \cdot \Delta\mathbf{w}^{(4)} = \mathbf{w}^{(1)} + \mathbf{w}^{(2)} + \mathbf{w}^{(3)} + \mathbf{w}^{(4)}$$

Part III: MIRA

- perceptron often makes bold updates (over-correction)
  - and sometimes too small updates (under-correction)
  - but hard to tune learning rate
- "just enough" update to correct the mistake?

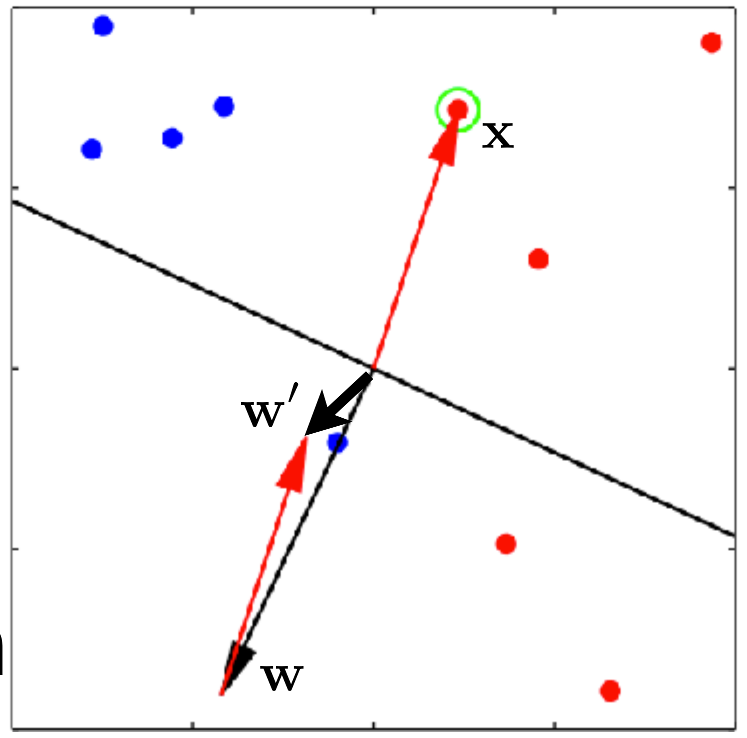
$$\mathbf{w}' \leftarrow \mathbf{w} + \frac{y - \mathbf{w} \cdot \mathbf{x}}{\|\mathbf{x}\|^2}\mathbf{x}$$

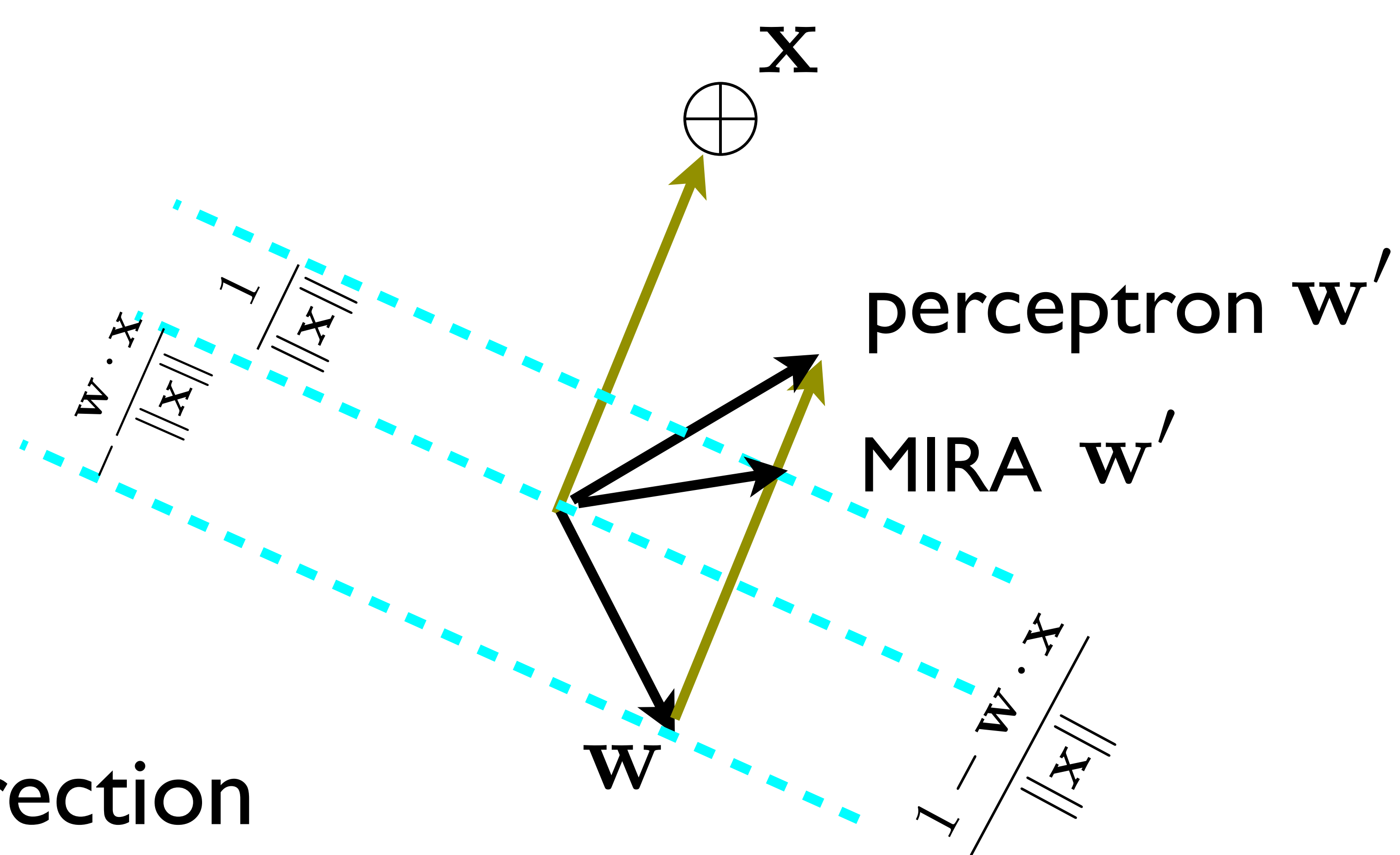
easy to show:

$$\mathbf{w}' \cdot \mathbf{x} = (\mathbf{w} + \frac{y - \mathbf{w} \cdot \mathbf{x}}{\|\mathbf{x}\|^2}\mathbf{x}) \cdot \mathbf{x} = y$$

margin-infused relaxation algorithm (MIRA)

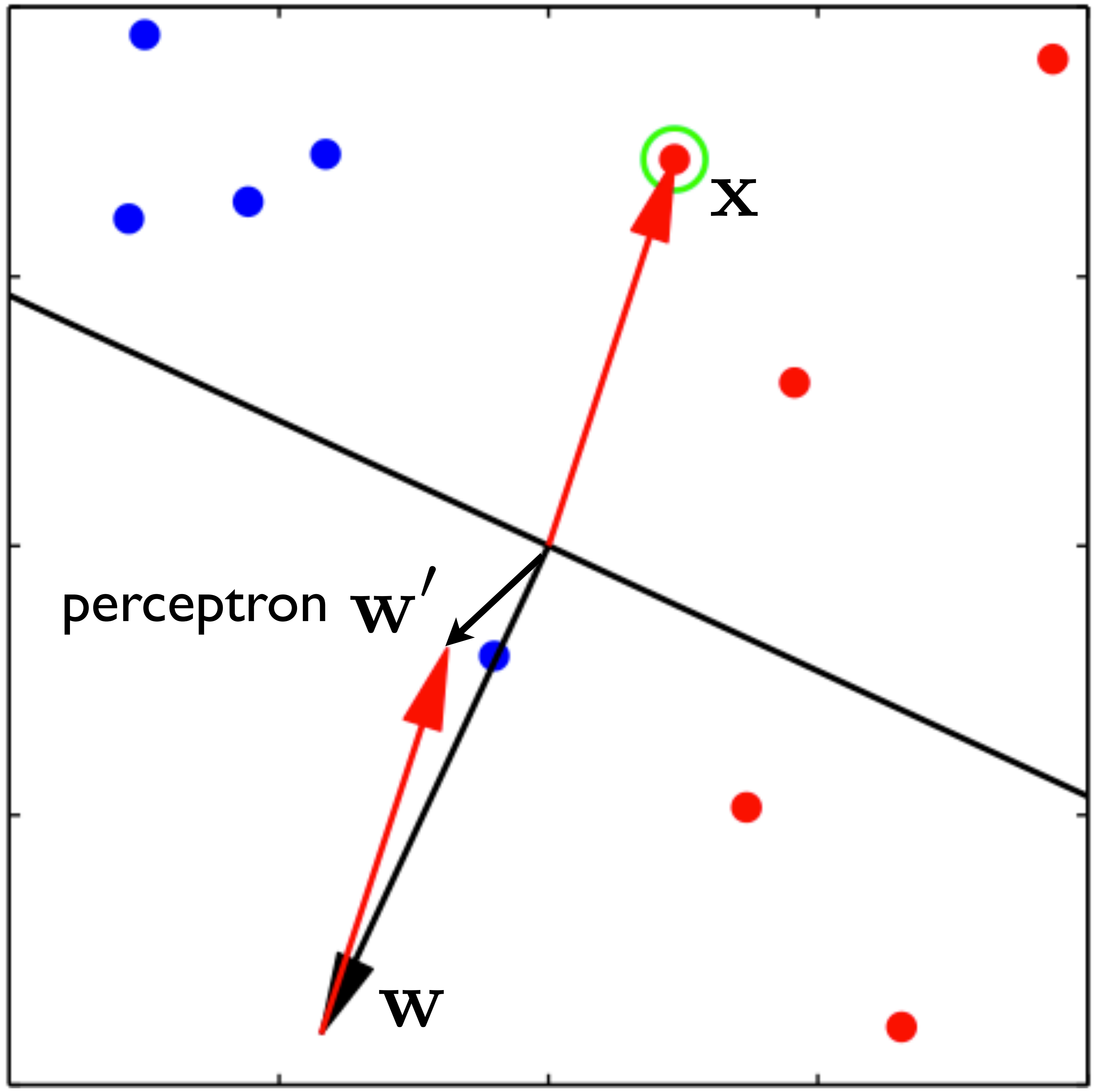
under-correction





over-correction

Example: Perceptron under-correction



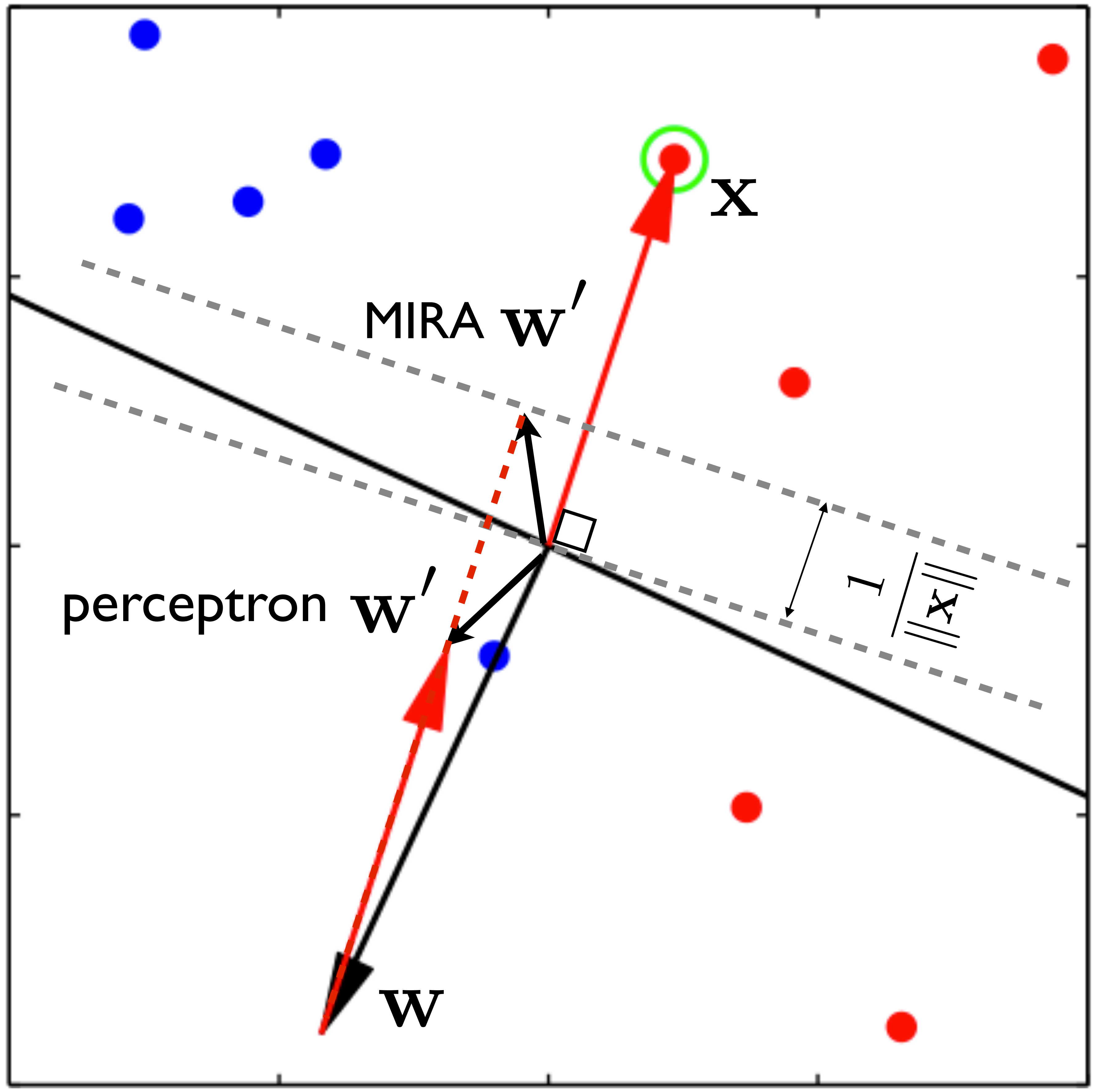
MIRA: just enough

$$\min_{\mathbf{w}'} \|\mathbf{w}' - \mathbf{w}\|^2$$

$$\text{s.t. } \mathbf{w}' \cdot \mathbf{x} \geq 1$$

minimal change to ensure functional margin of 1 (dot-product **w'** · **x**=1)

MIRA ≈ 1-step SVM



functional margin: $y(\mathbf{w} \cdot \mathbf{x})$

geometric margin: $\dfrac{y(\mathbf{w} \cdot \mathbf{x})}{\|\mathbf{w}\|}$

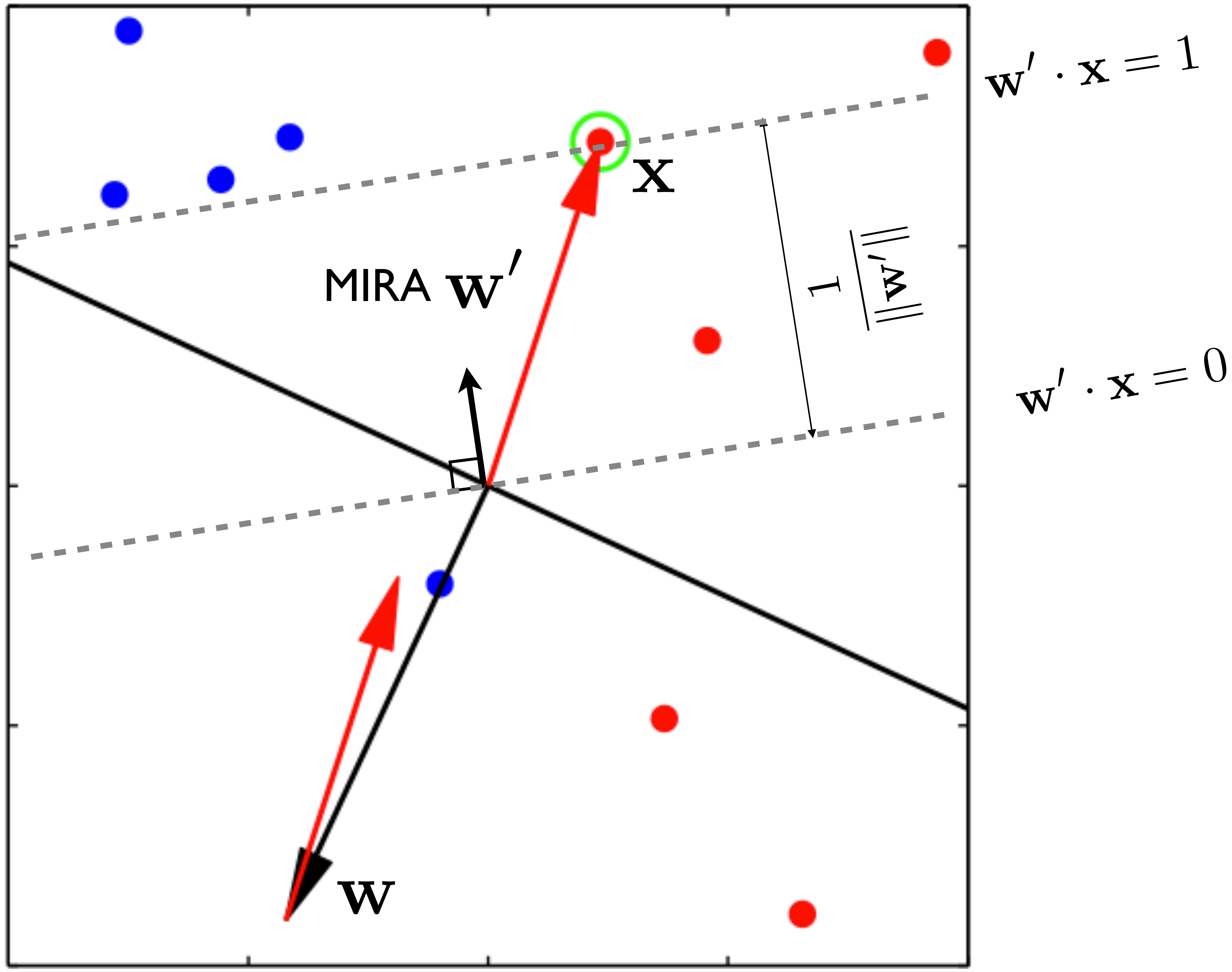
MIRA: functional vs geom. margin

$$\min_{\mathbf{w}'} \|\mathbf{w}' - \mathbf{w}\|^2$$

$$\text{s.t. } \mathbf{w}' \cdot \mathbf{x} \geq 1$$

minimal change to ensure functional margin of 1 (dot-product **w**' · **x**=1)

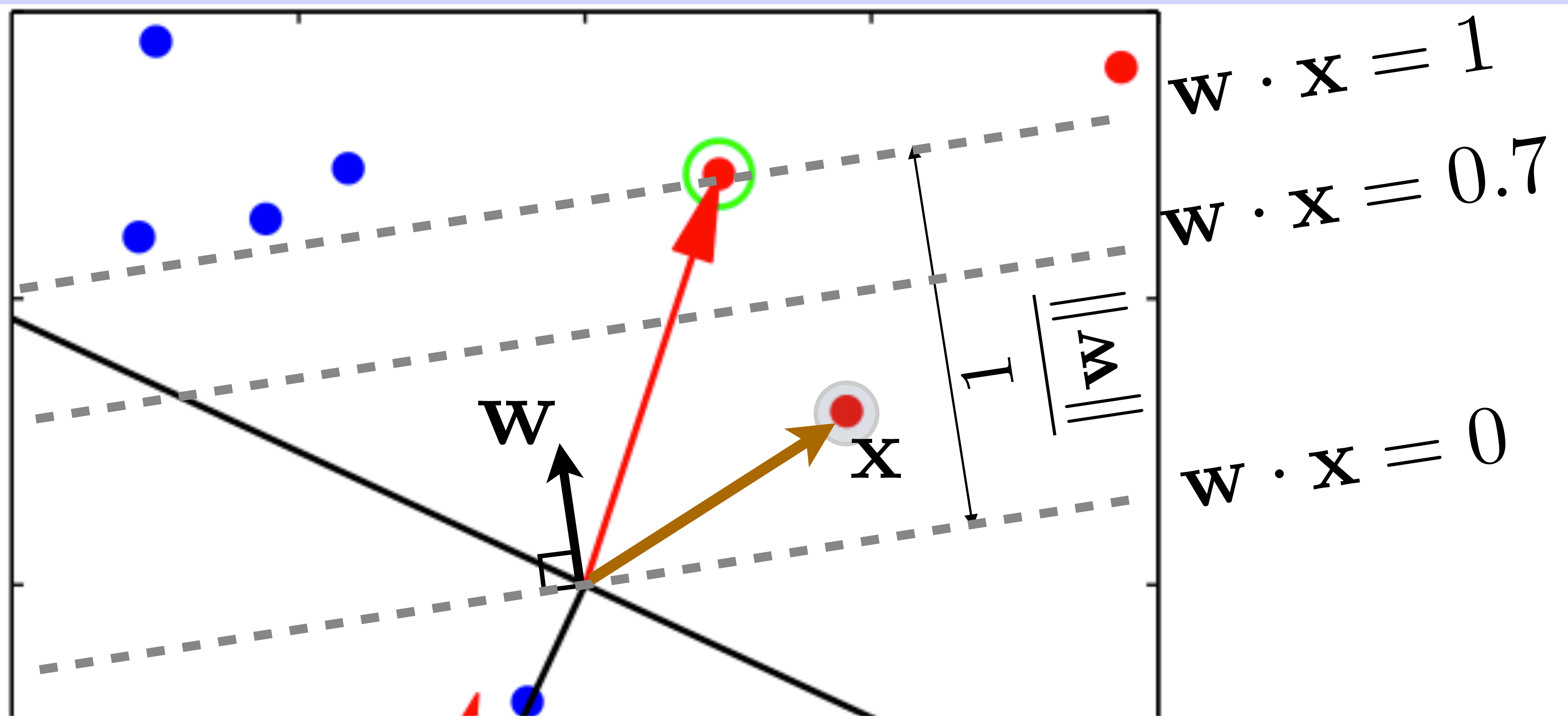
MIRA ≈ 1-step SVM



functional margin: $y(\mathbf{w} \cdot \mathbf{x})$

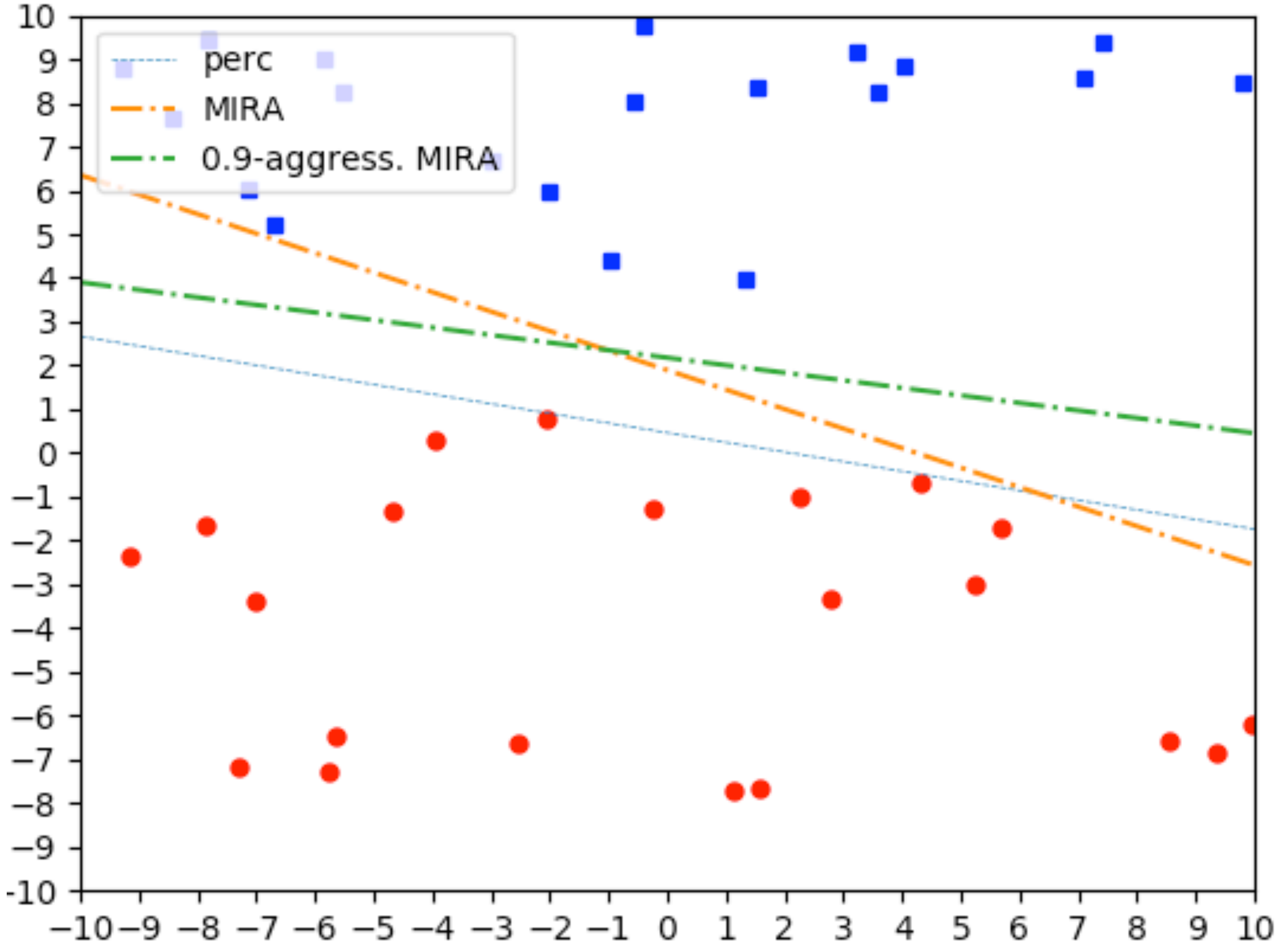
geometric margin: $\dfrac{y(\mathbf{w} \cdot \mathbf{x})}{\|\mathbf{w}\|}$

Optional: Aggressive MIRA

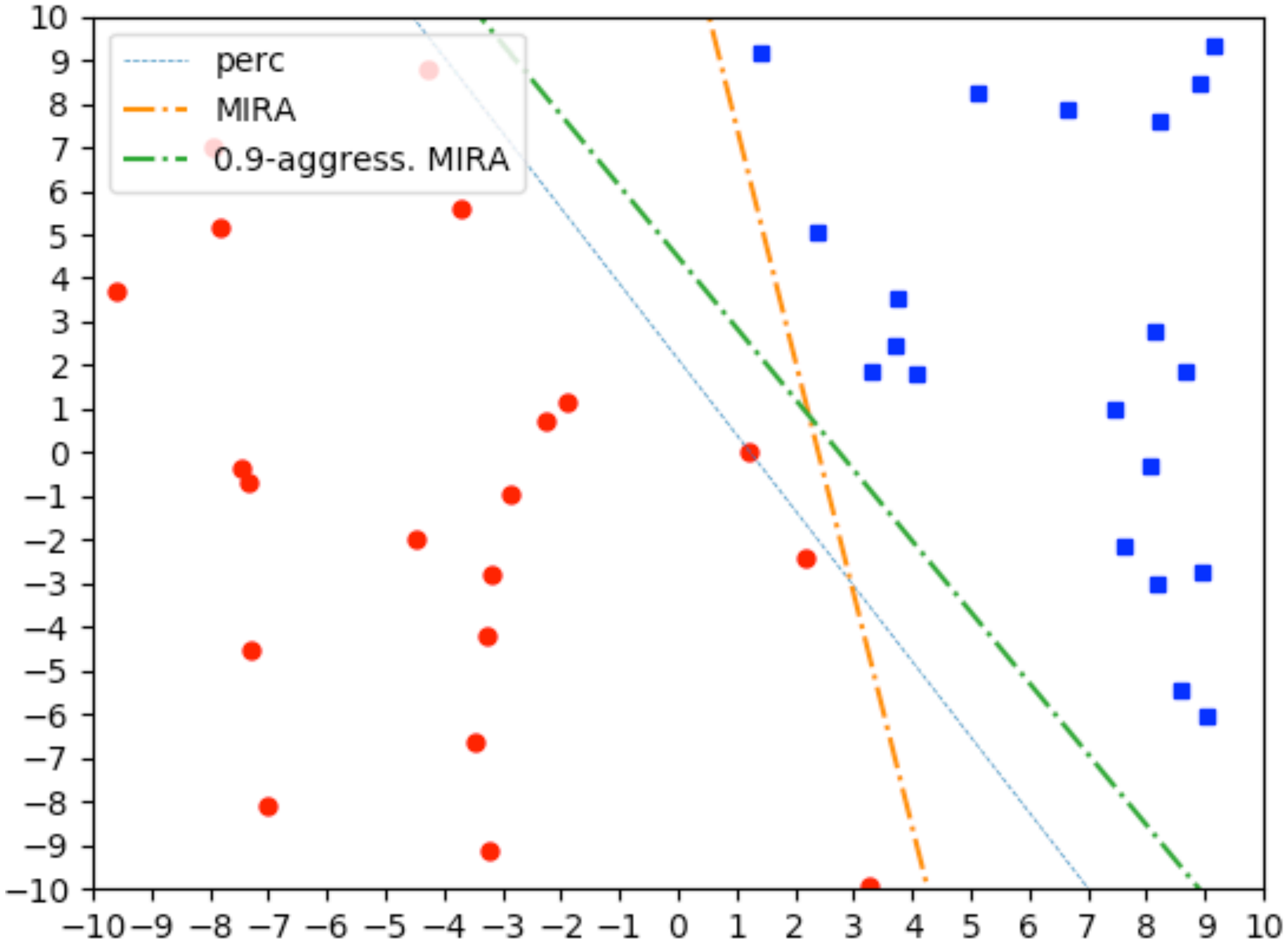


- aggressive version of MIRA
  - also update if correct but not confident enough
    - i.e., functional margin ($y\ \mathbf{w} \cdot \mathbf{x}$) not big enough
  - p-aggressive MIRA: update if $y\ (\mathbf{w} \cdot \mathbf{x}) < p$ $(0 <= p < 1)$
    - MIRA is a special case with p=0: only update if misclassified!
    - update equation is same as MIRA
      - i.e., after update, functional margin becomes 1
  - larger p leads to a larger geometric margin but slower convergence

Demo


perc
MIRA
0.9-aggress. MIRA
10
9
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−5
−6
−7
−8
−9
-10
−10 −9 −8 −7 −6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7 8 9 10

Demo



perc
MIRA
0.9-aggress. MIRA
10
9
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−5
−6
−7
−8
−9
−10
−10 −9 −8 −7 −6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7 8 9 10

Part IV: Practical Issues

"A ship in port is safe, but that is not what ships are for."

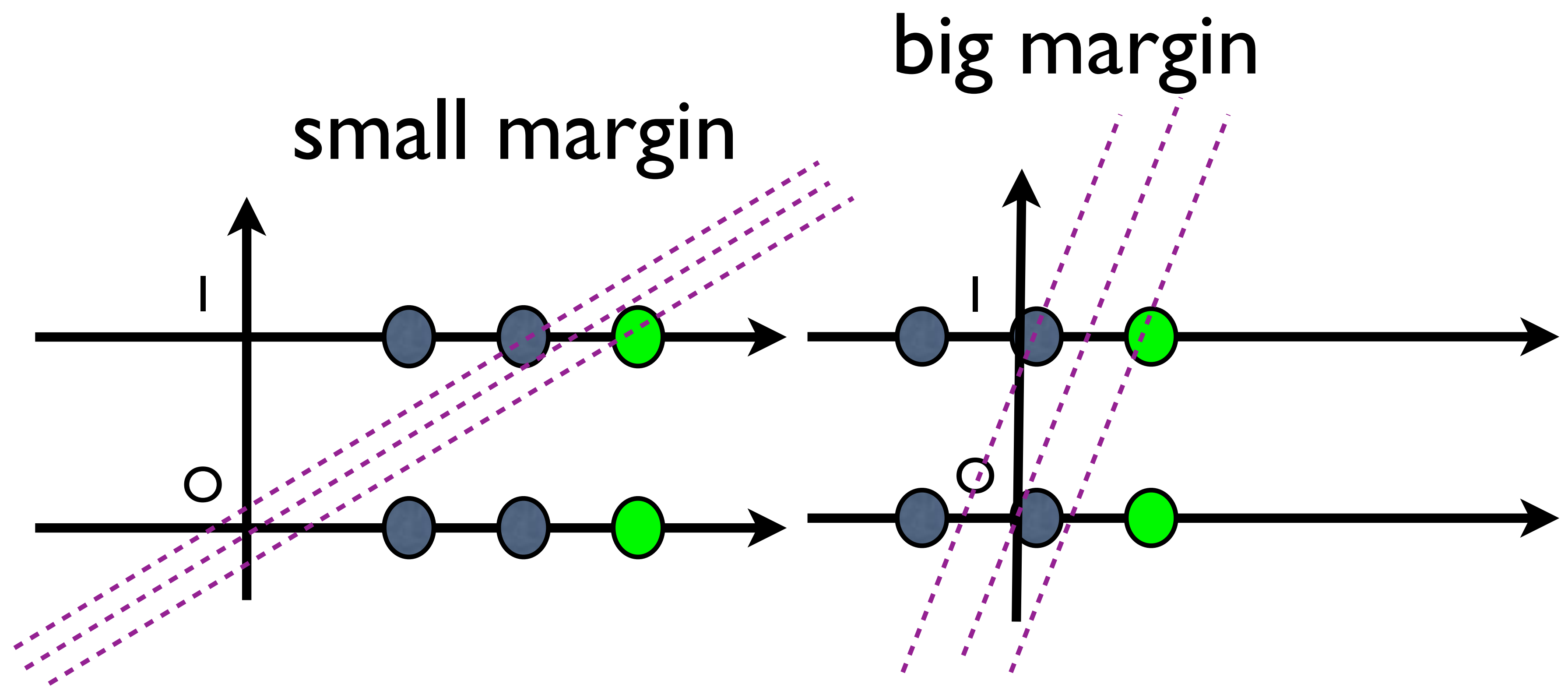
– Grace Hopper (1906-1992)

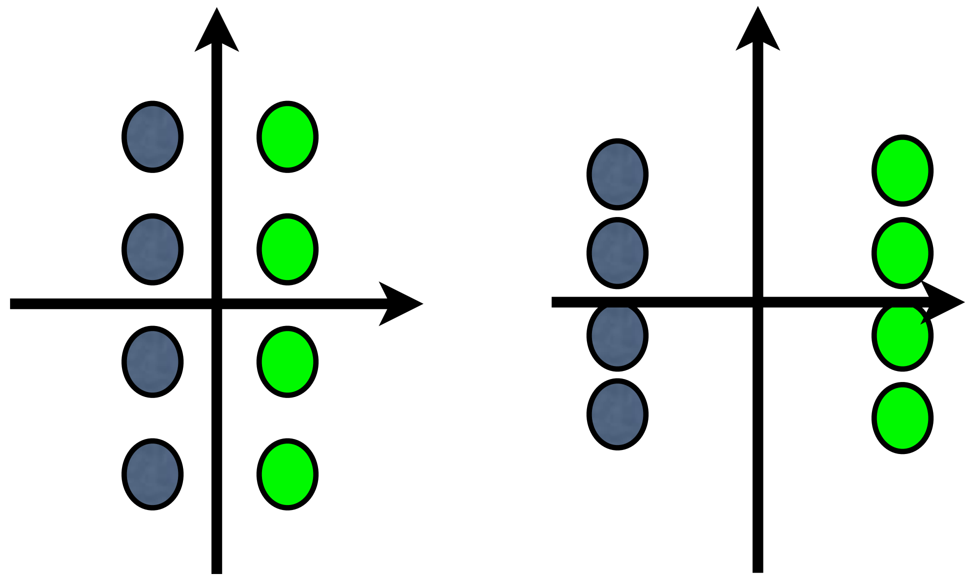
- you will build your own linear classifiers for HW2 (same data as HW1)
- slightly different binarizations
  - for k-NN, we binarize all categorical fields but keep the two numerical ones
  - for perceptron (and most other classifiers), we binarize numerical fields as well
  - why? hint: larger "age" always better? more "hours" always better?

Useful Engineering Tips:

averaging, shuffling, variable learning rate, fixing feature scale

- averaging helps significantly; MIRA helps a tiny little bit
  - perceptron < MIRA < avg. perceptron ≈ avg. MIRA ≈ SVM
- shuffling the data helps hugely if classes were ordered (HW1)
  - shuffling before each epoch helps a little bit
- variable (decaying) learning rate often helps a little
  - 1/(total#updates) or 1/(total#examples) helps
  - any requirement in order to converge?
    - how to prove convergence now?
- centering of each dimension helps (Ex1/HW1)
  - why? => smaller radius, bigger margin!
- unit variance also helps (why?) (Ex1/HW1)
  - 0-mean, 1-var => each feature ≈ a unit Gaussian





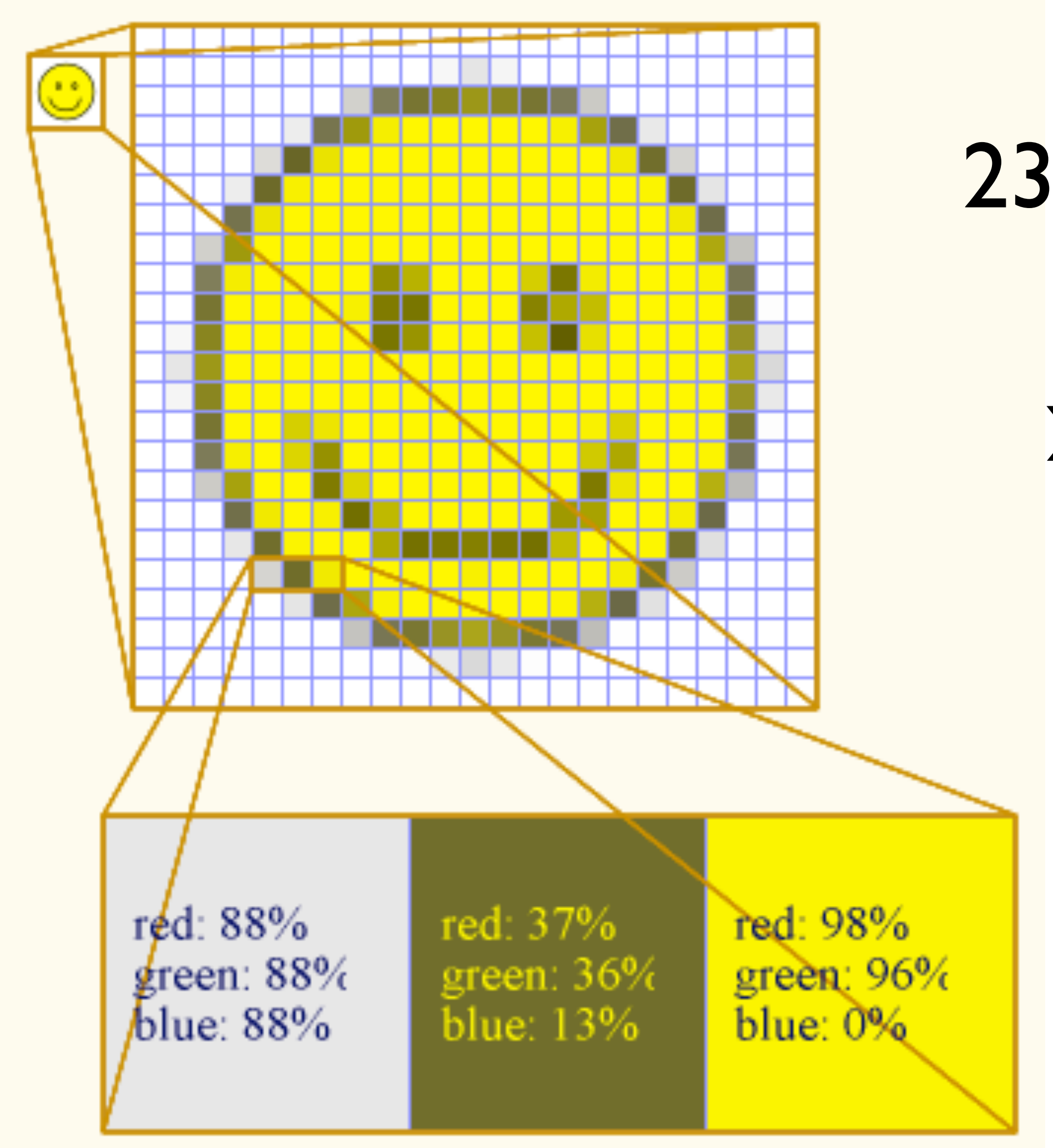
Feature Maps in Other Domains

- how to convert an image or text to a vector?

28x28 grayscale image

$\mathbf{x} \in \mathbb{R}^{784}$

- image



23x23 RGB image

$\mathbf{x} \in \mathbb{R}^{23x23x3}$

"a"	"abbreviations"	. . .	"zoology"
1	0		0
0	1		0
0	0		0
⋮	⋮	. . .	⋮
0	0		0
0	0		1
0	0		0

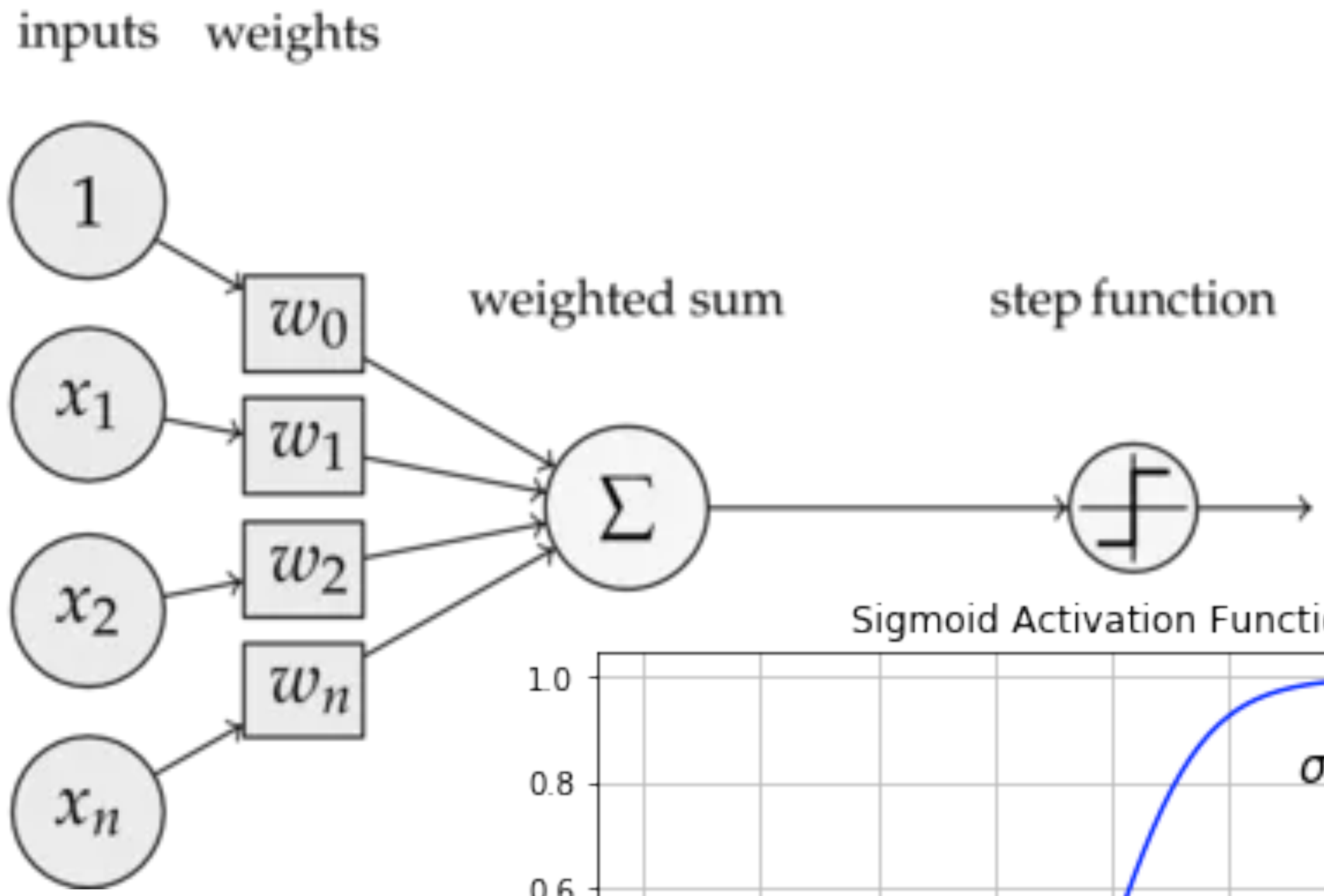
- text

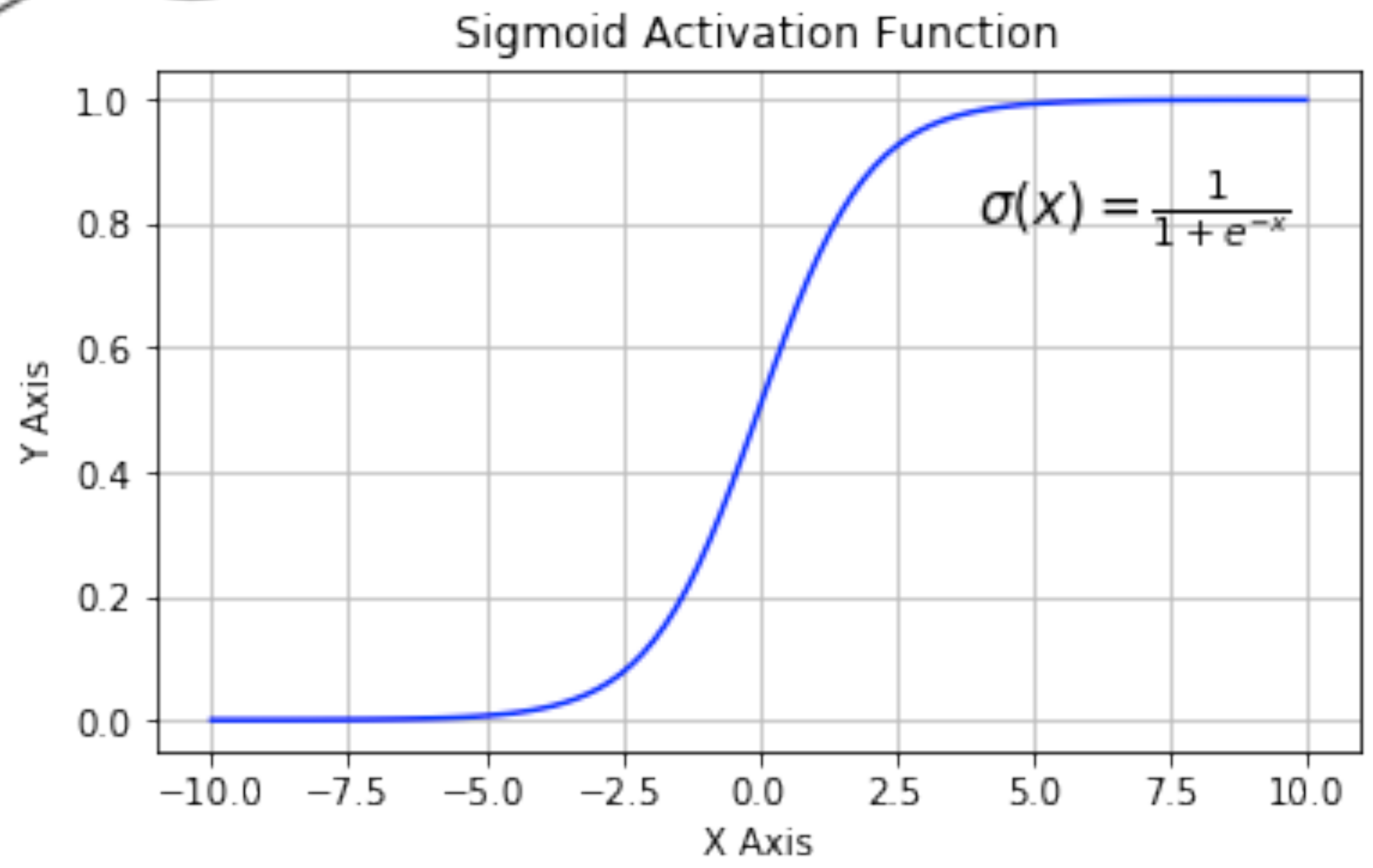
"one-hot" representation of words
(all binary features)

in deep learning there are other feature maps

Part V: Perceptron vs. Logistic Regression

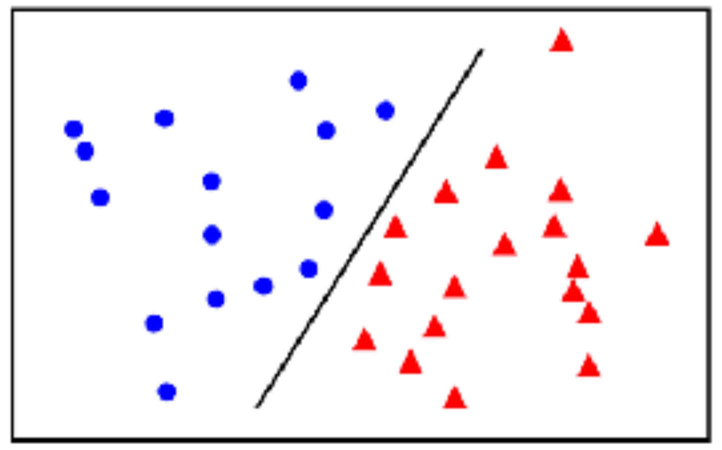
- logistic regression is another popular linear classifier
  - can be viewed as "soft" or "probabilistic" perceptron
  - same decision rule (sign of dot-product), but prob. output





perceptron

$$f(\mathbf{x}) = \mathbf{sign}(\mathbf{w} \cdot \mathbf{x})$$



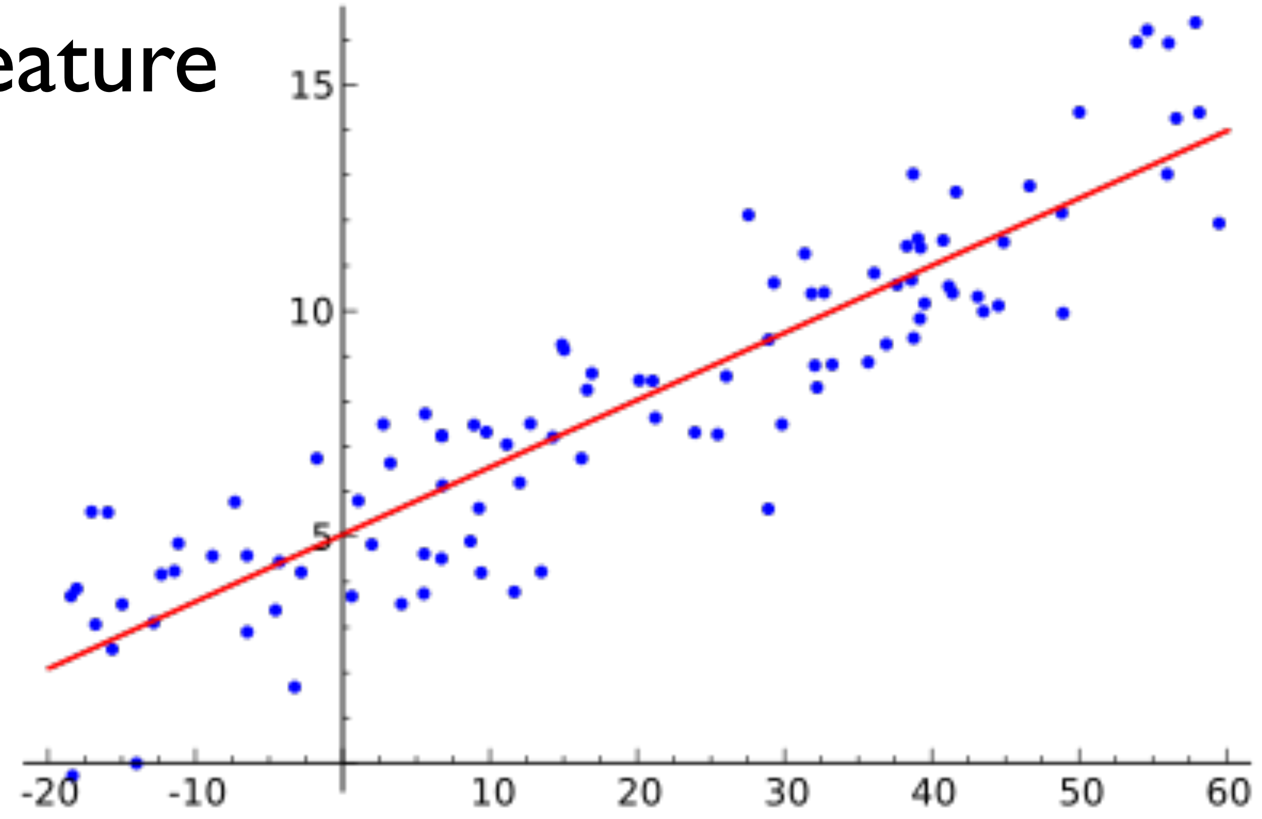
logistic regression

$$f(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

Logistic vs. Linear Regression

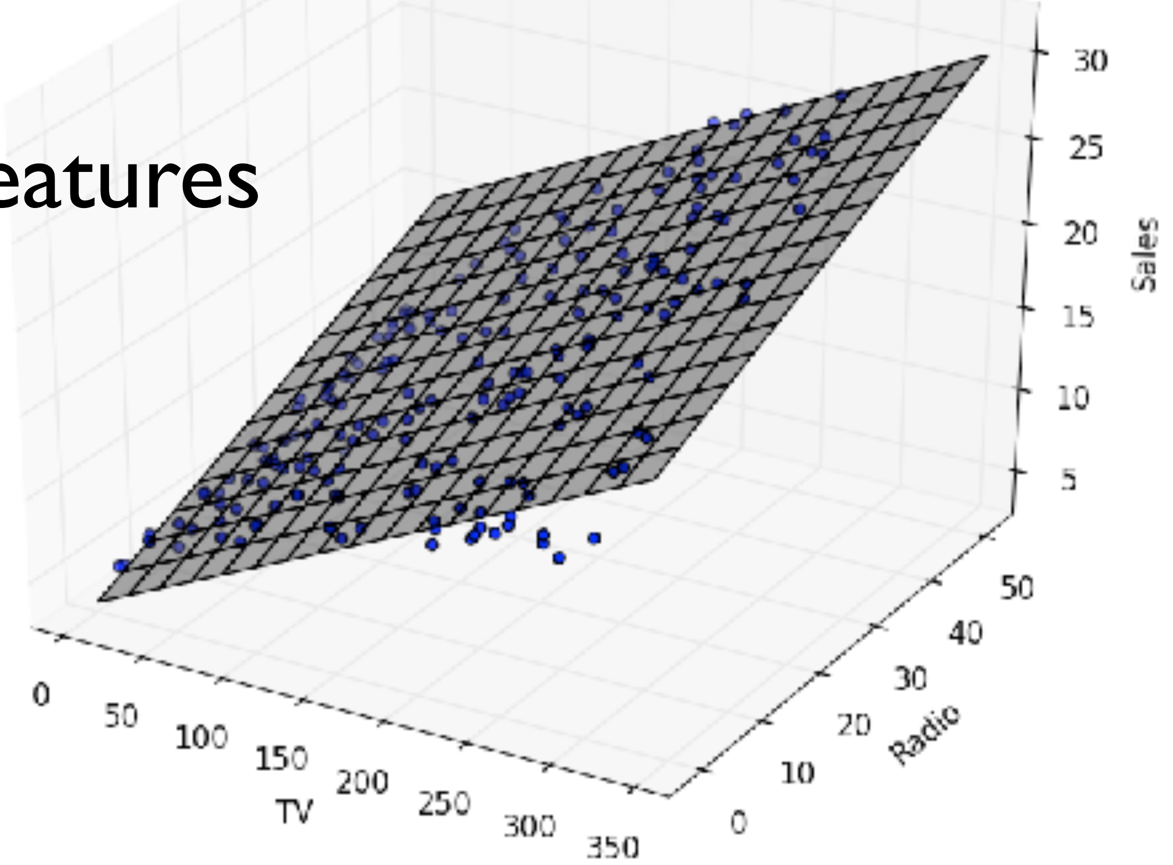
- linear regression is regression applied to real-valued output using linear function
- logistic regression is regression applied to 0-1 output using the sigmoid function

1 feature

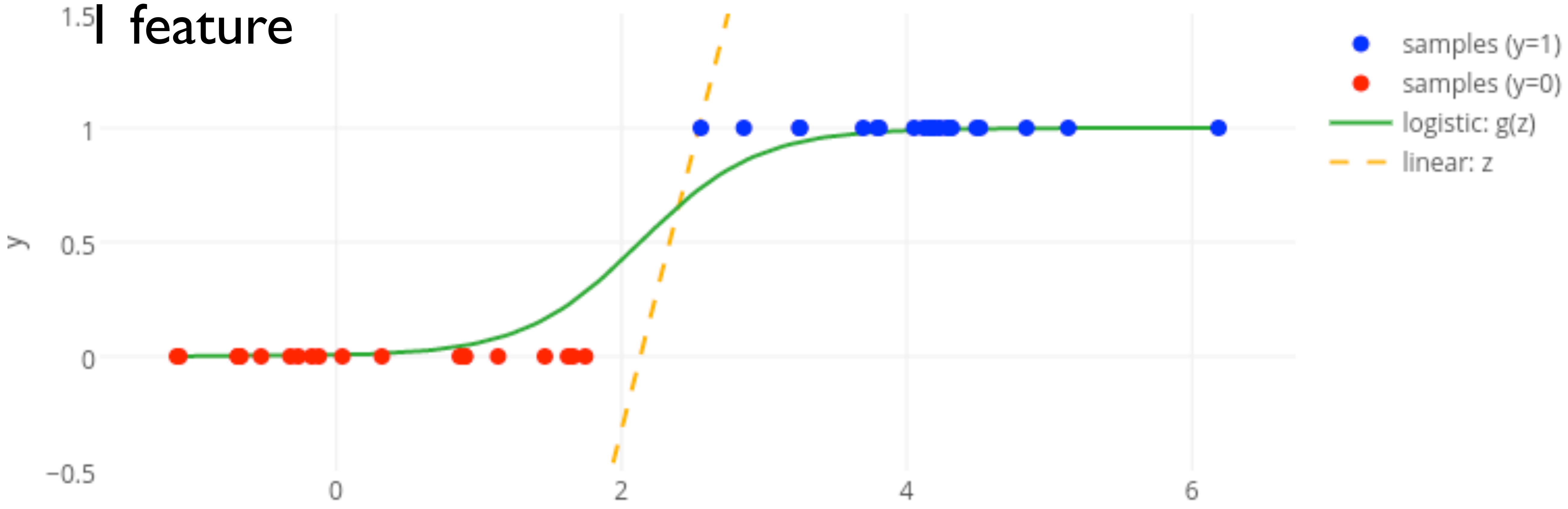


linear

2 features

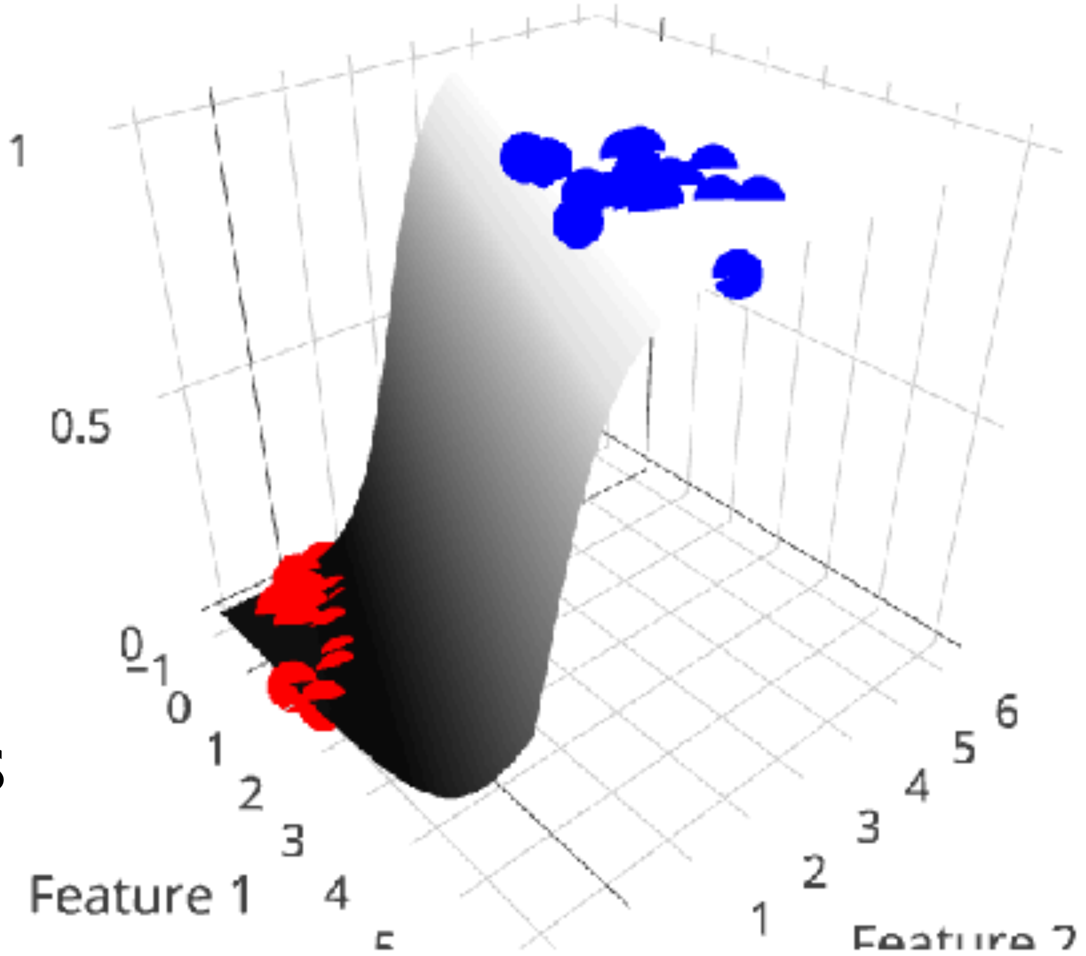


1 feature



logistic

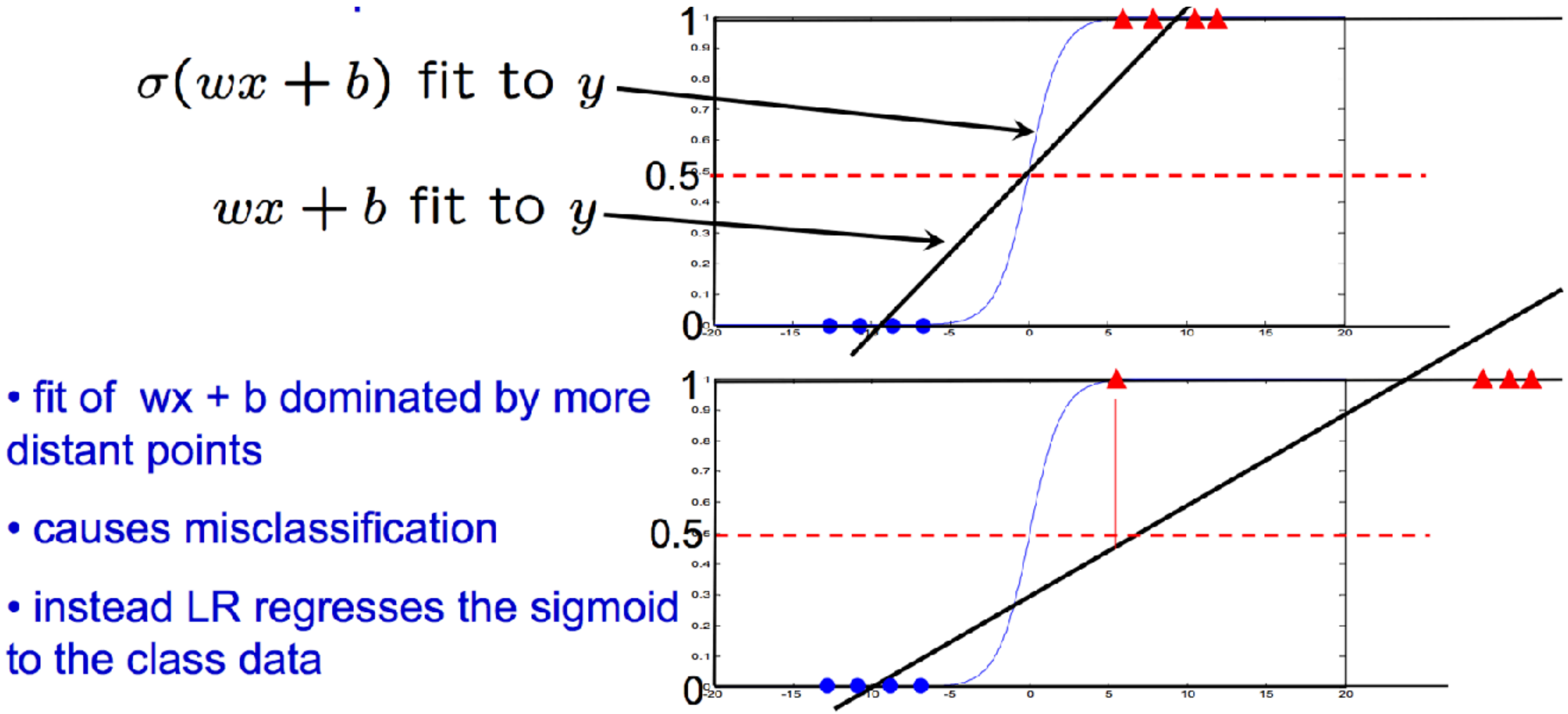
2 features



https://florianhartl.com/logistic-regression-geometric-intuition.html

Why Logistic instead of Linear

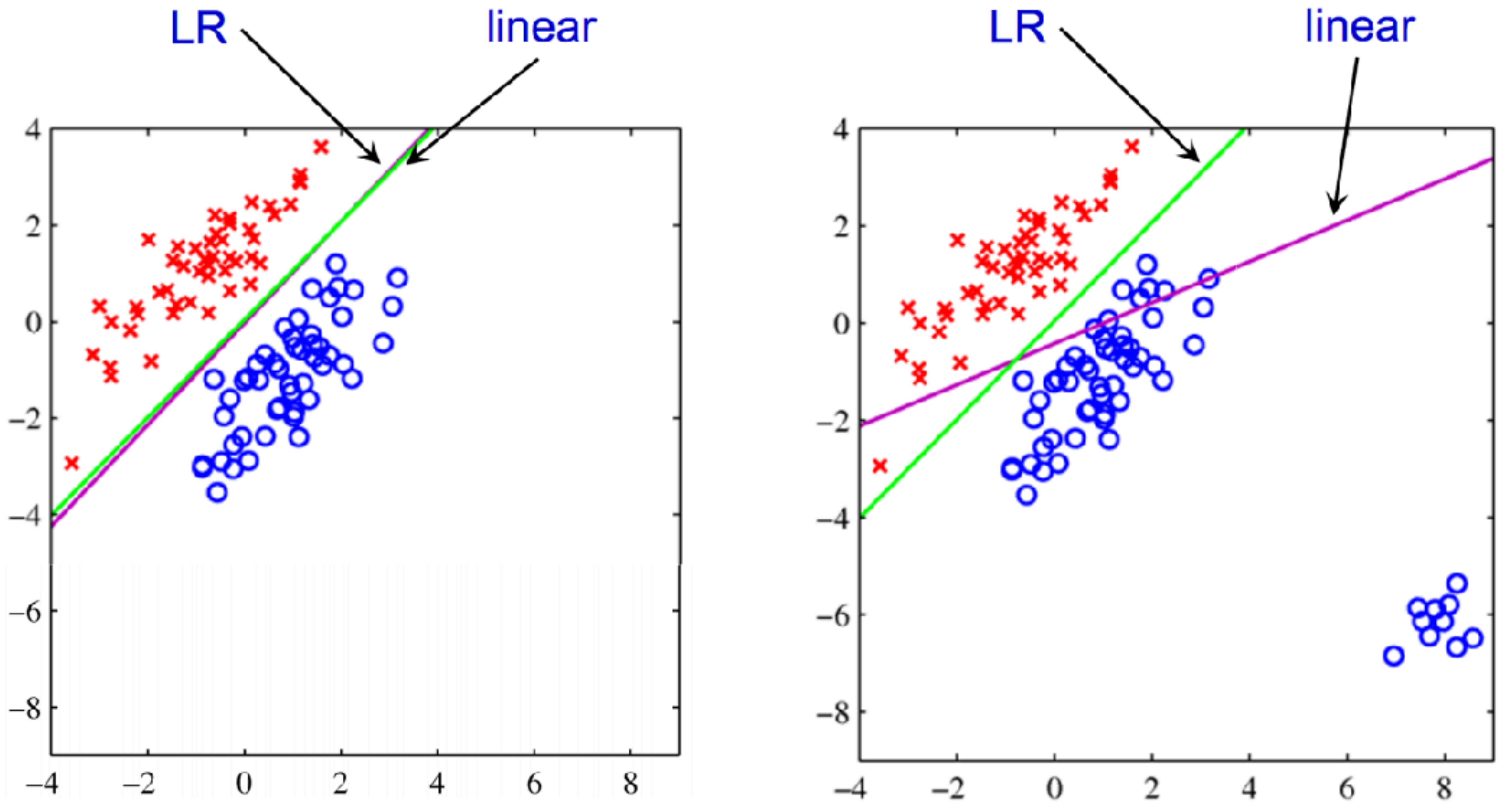
- linear regression easily dominated by distant points
  - causing misclassification



http://www.robots.ox.ac.uk/~az/lectures/ml/2011/lect4.pdf

Why Logistic instead of Linear

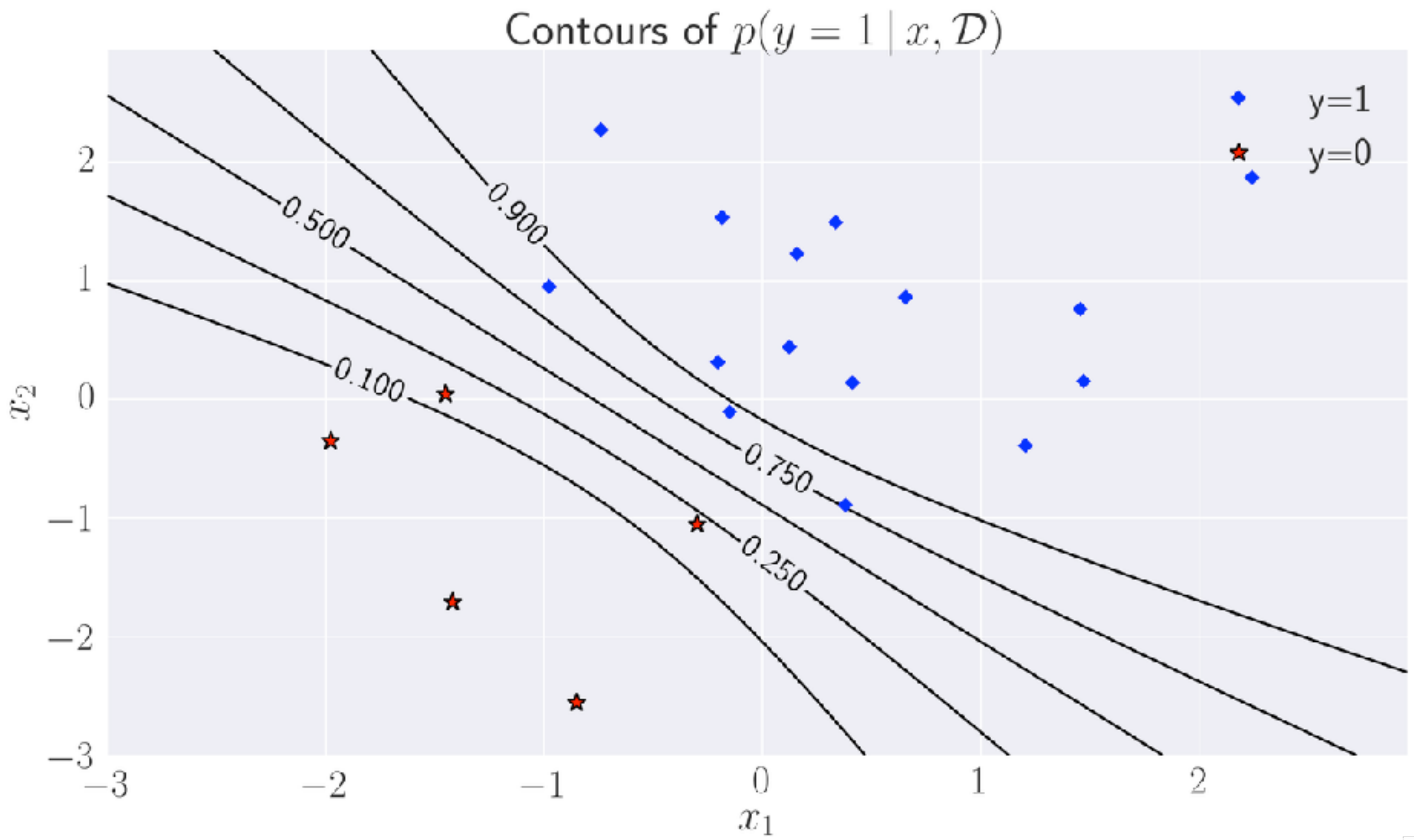
- linear regression easily dominated by distant points
  - causing misclassification



$\sigma(w_1x_1 + w_2x_2 + b)$ fit, vs $w_1x_1 + w_2x_2 + b$

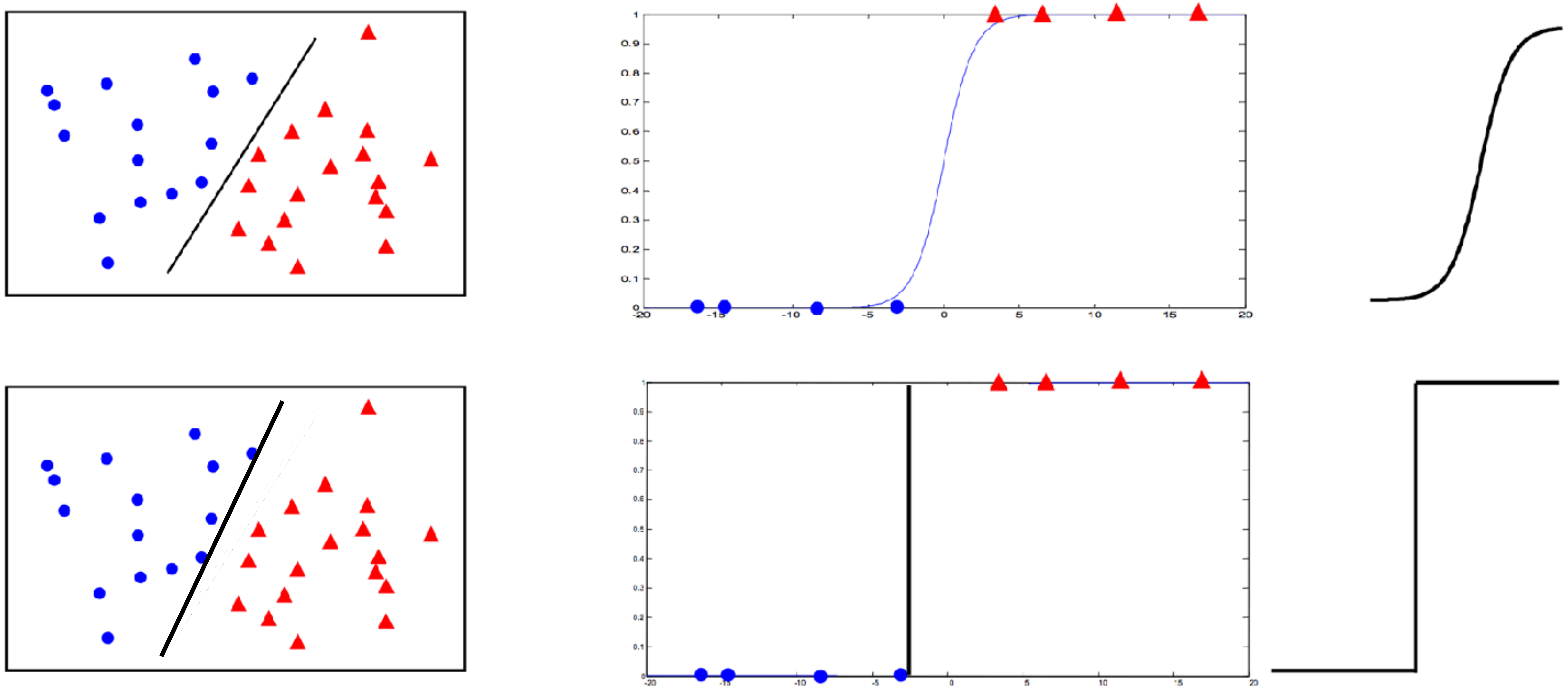
Why 0/1 instead of +/-1

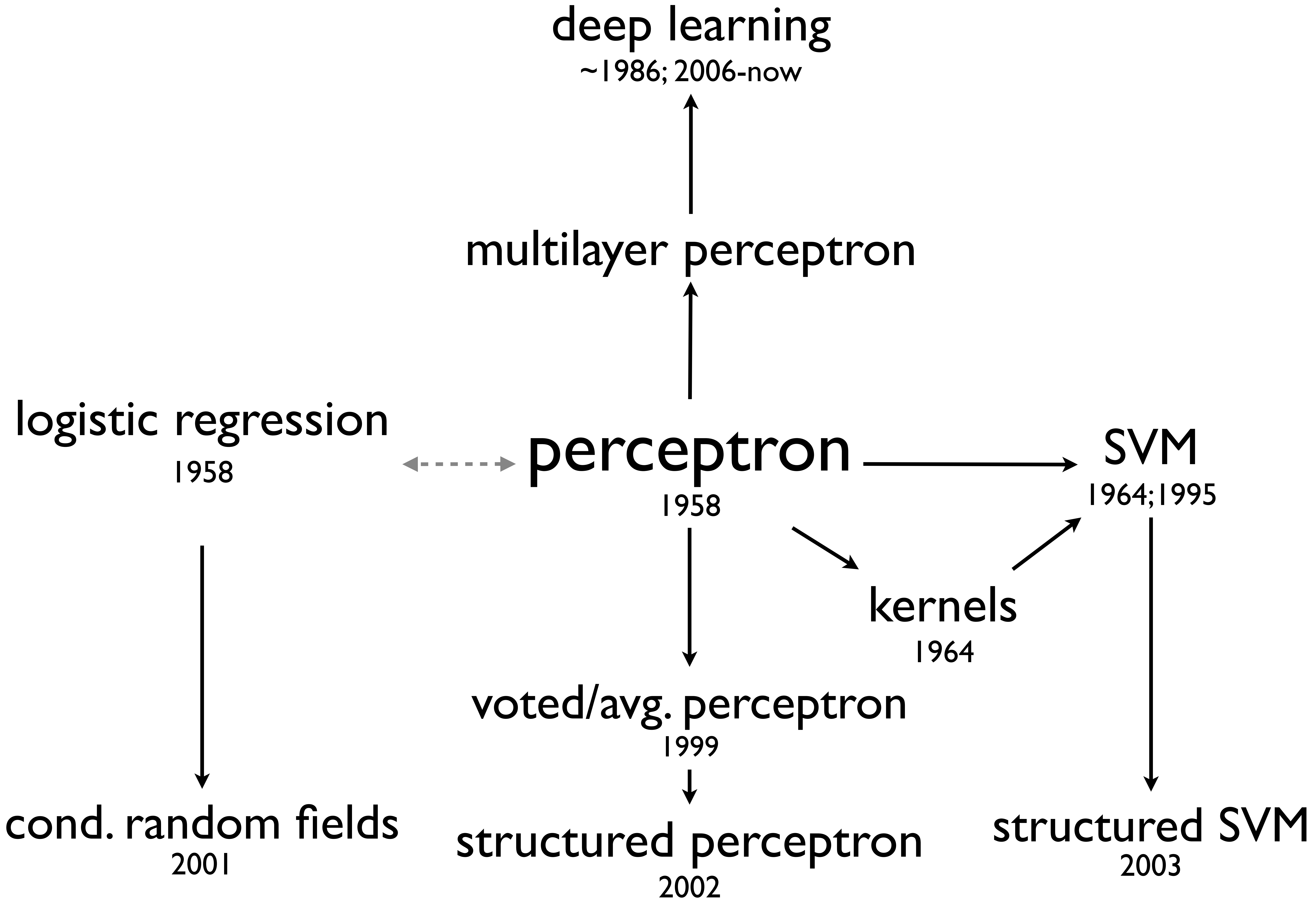
- perc: y=+1 or -1; logistic regression: y=1 or 0
- reason: want the output to be a probability
- decision boundary is still linear: p(y=1 | **x**) = 0.5



Logistic Regression: Large Margin

- perceptron can be viewed roughly as “step” regression
- logistic regression favors large margin; SVM: max margin
- in practice: perc. << avg. perc. ≈ logistic regression ≈ SVM





deep learning
~1986; 2006-now
multilayer perceptron
logistic regression
1958
perceptron
1958
SVM
1964;1995
kernels
1964
voted/avg. perceptron
1999
cond. random fields
2001
structured perceptron
2002
structured SVM
2003