Université de Montréal
Département d'informatique et de recherche opérationnelle

IFT6390
Fondements de l'apprentissage machine

Linear classifiers
(and linear discriminant functions)

Professeur: Ioannis Mitliagkas
Slides: Pascal Vincent

What is a linear classifier?

- A learning algorithm for classification that makes it possible to learn a decision function *f(x)* for a classification problem
- If its decision function **can be expressed** in one of the following simple forms:

Binary classification (2 classes):

$$f(\mathbf{x}) = \text{sign}(g(\mathbf{x}))$$

response: class -1 or +1

where $g(\mathbf{x})$ is a linear discriminant function:

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \mathbf{w}^t \mathbf{x} + b$$

Parameters:

$$\theta = \{\mathbf{w}, b\}, \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$$

weights vector; bias

Multiclass classification (*m* classes):

$$f(\mathbf{x}) = \text{argmax}(g(\mathbf{x}))$$

response: class between 1 and m

$$g(\mathbf{x}) = \mathbf{W}^T \mathbf{x} + \mathbf{b}$$

b is not a scalar. it has seperate b for every class

$$g(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_m(\mathbf{x}))$$

Parameters:

$$\theta = \{\mathbf{W}, \mathbf{b}\}, \quad \mathbf{W} \in \mathbb{R}^{d \times m}, \quad b \in \mathbb{R}^m$$

matrix of weights; vector of biases

Relationship of binary and multiclass with m=2

$$
\begin{aligned}
f(\mathbf{x}) &= \arg\max[\mathbf{W}^T\mathbf{x} + \mathbf{b}] \\
&= \arg\max[(\mathbf{W}_{:1}^T\mathbf{x} + b_1, \quad \mathbf{W}_{:2}^T\mathbf{x} + b_2)] \\
&= 2 \quad \text{if} \quad \mathbf{W}_{:2}^T\mathbf{x} + b_2 > \mathbf{W}_{:1}^T\mathbf{x} + b_1 \quad (\text{and 1 o.w.}) \\
&= 2 \quad \text{if} \quad (\mathbf{W}_{:2} - \mathbf{W}_{:1})^T\mathbf{x} + (b_2 - b_1) > 0 \quad (\text{and 1 o.w.}) \\
&= 2 \quad \text{if} \quad \text{sign}((\underbrace{\mathbf{W}_{:2} - \mathbf{W}_{:1}}_{\mathbf{w}})^T\mathbf{x} + (\underbrace{b_2 - b_1}_{b})) = +1 \quad (\text{and 1 o.w.}) \\
&= 1.5 + \frac{1}{2}\text{sign}(\mathbf{w}^T\mathbf{x} + b)
\end{aligned}
$$

o.w. = otherwise

Geometry of a linear discriminant function

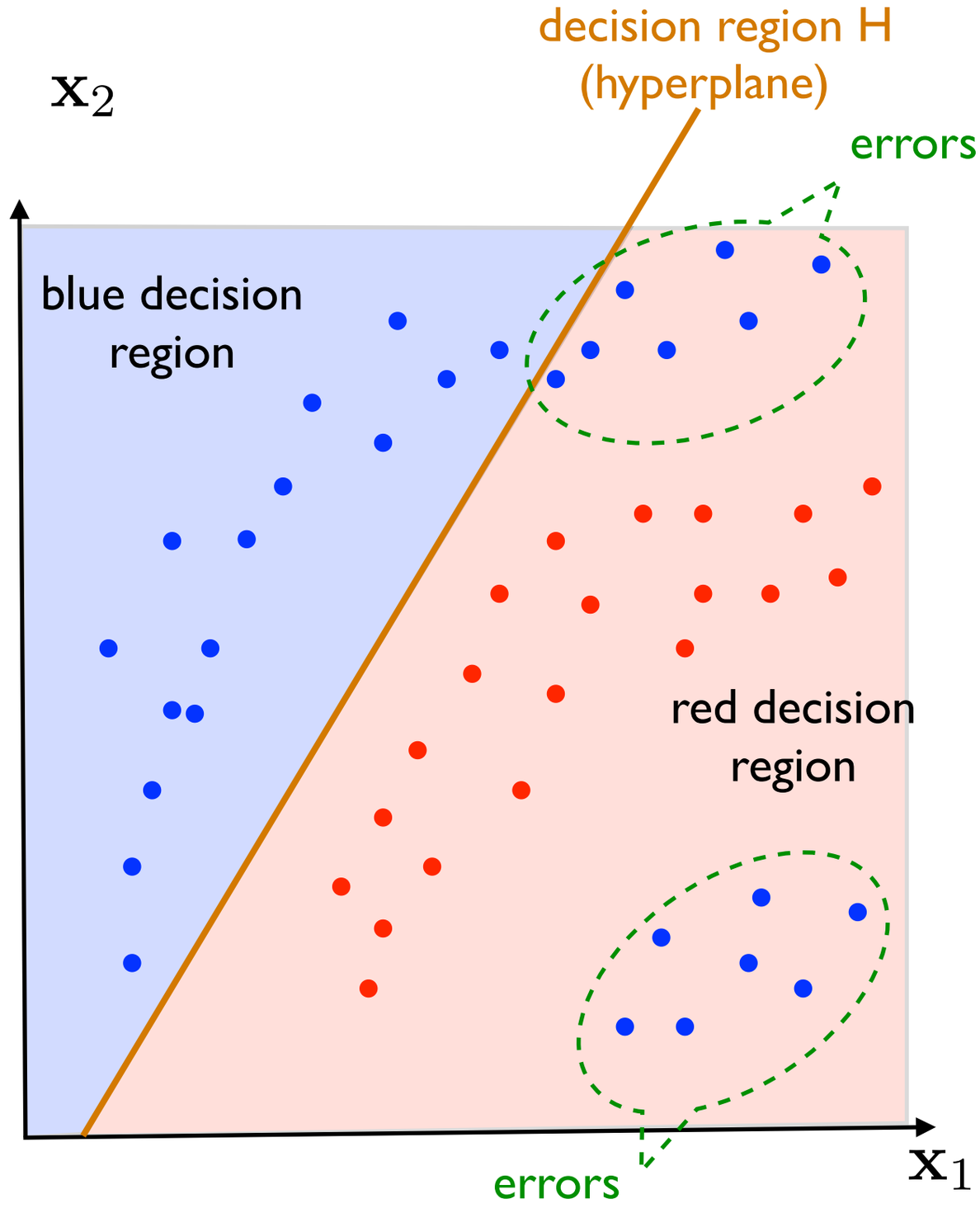
- Decision regions:
  $R_+=\{x \mid g(x)>0\}$ and $R_-=\{x \mid g(x)<0\}$
- Decision boundary/surface $H=\{x \mid g(x)=0\}$
- Linear discriminant: $g(x)= w^T x +b$

➡Decision boundary: $H=\{x \mid w^T x +b =0\}$
A **hyperplane**.

➡Decision regions: the two **half-spaces** defined by the hyperlpane.

Geometry of a linear discriminant function

The decision boundary is a hyperplane.



Cost of misclassification

- Decision function associated with a linear discriminant function:

$$y = f_\theta(\mathbf{x}) = \mathrm{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

- Cost (loss) of misclassification:

targets

$$t^{(i)} \in \{-1, +1\}$$

$$L(y,t) = I_{\{y \neq t\}} = I_{\{(\langle \mathbf{w},\mathbf{x} \rangle + b)t < 0\}}$$

- Rate of misclassification (error rate) of such a function on a set: $D_n = \{(x^{(1)}, t^{(1)}), \ldots, (x^{(n)}, t^{(n)})\}$

$$\hat{R}_{\mathrm{classif}}(f_\theta, D_n) = \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(x^{(i)}), t^{(i)})$$

Linear separability of Data

- We say that a dataset $D_n = \{(x^{(1)}, t^{(1)}), \ldots, (x^{(n)}, t^{(n)})\}$ is linearly separable if (and only if) there exists a linear discriminant function that makes 0 errors on this set.
- Geometrically: if there is a hyper-plane that perfectly separates the examples of D_n (so that all of those in one class are in a half-space and those in the other half in the other half-space)
- Formally. If $t^{(i)} \in \{-1, +1\}$ D_n is linearly separable if and only if

$$\exists \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \quad | \quad \forall i \in \{1, \ldots, n\}, \quad \overbrace{\text{sign}(\mathbf{w}^T x^{(i)} + b)}^{f(x^{(i)})} = t^{(i)}$$

Many algorithms to learn a "linear classifier"

- Linear regression (regularized or not) with targets +1, -1. (But not very appropriate for classification).
- Nearest centroid classifier (or average, center of gravity)
- Bayes classifier with Gaussian densities (if forced to have the same covariance).
- Logistic regression
- Perceptron Algorithm
- Fisher Linear Discriminant Analysis (LDA)
- (linear) Support vector machines...

Many algorithms to learn a "linear classifier"

How are they similar?

- They all generate a decision function **expressible** in the parameterized form previously defined.
- For the binary classification (2 classes), their decision boundary is always a **hyperplane**

How are they different?

- For the same training set D_n they will typically learn different parameter values.
  => giving rise to a different hyperplane.
  => Some will generalize better to new examples

Ex 1: nearest centroid (centroid/ barycenter)

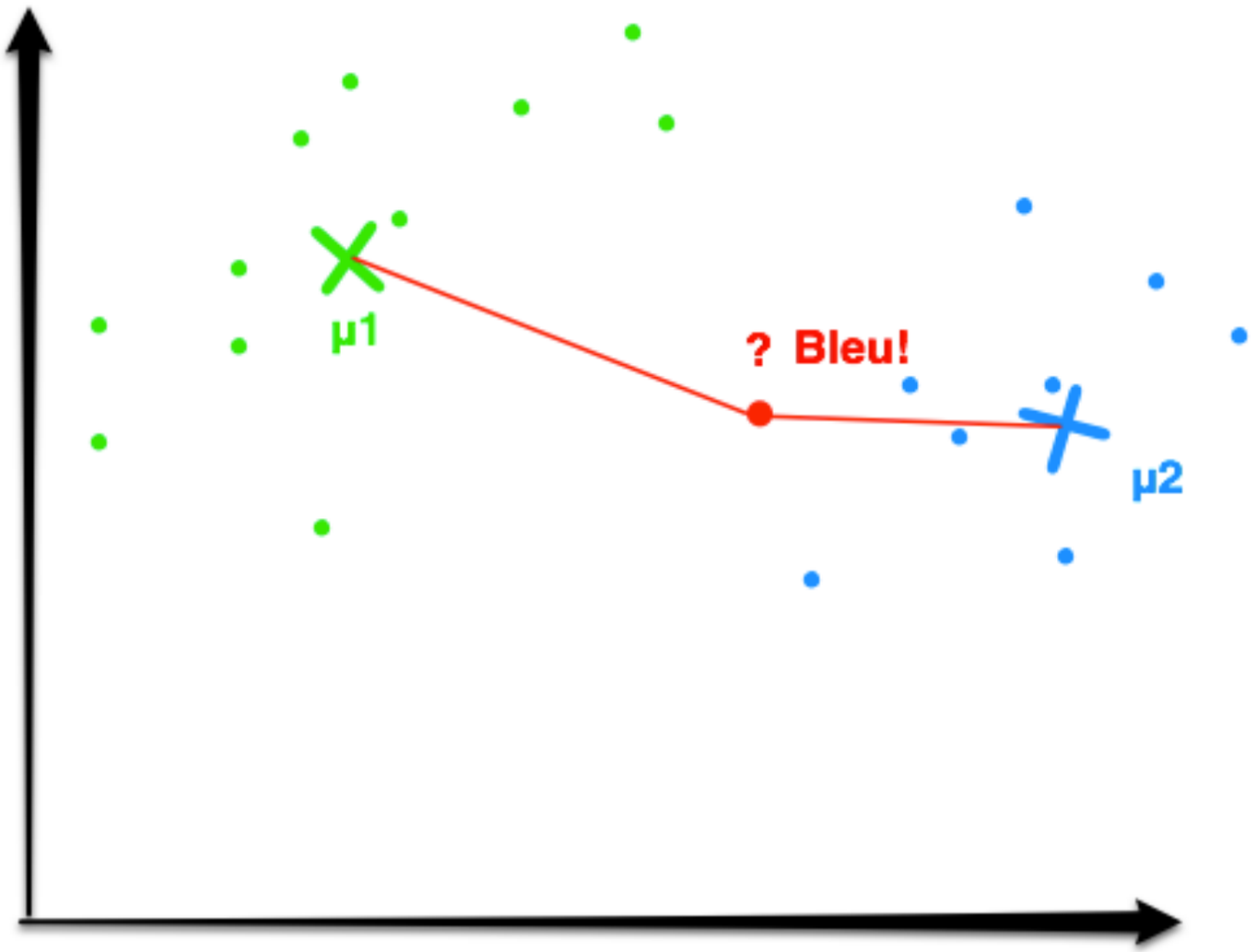
- Parameters: $\boldsymbol{\theta}=\{\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_m\}$ all vectors of dimension d.
- Training:

  $\boldsymbol{\mu}_m$ = average of all examples in class *m*
- Decision:

$$f(x) \quad = \quad \arg\min[(d_{L_2}(\mathbf{x}, \mu_1), \ldots, d_{L_2}(\mathbf{x}, \mu_m))]$$

where d_{L2} is the Euclidean distance

Ex 1 : nearest centroid (centroid/ barycenter)



The nearest centroid classifier is a linear classifier

Multiclass:

$$
\begin{aligned}
f(x) &= \arg\min[(d_{L_2}(\mathbf{x}, \mu_1), \ldots, d_{L_2}(\mathbf{x}, \mu_m))] \\
&= \arg\min[(\|\mathbf{x} - \mu_1\|^2, \ldots, \|\mathbf{x} - \mu_m\|^2)] \\
&= \arg\min[(\|\mathbf{x}\|^2 - 2\mu_1^T\mathbf{x} + \|\mu_1\|^2, \ldots, \|\mathbf{x}\|^2 - 2\mu_m^T\mathbf{x} + \|\mu_m\|^2)] \\
&= \arg\min[(-2\mu_1^T\mathbf{x} + \|\mu_1\|^2, \ldots, -2\mu_m^T\mathbf{x} + \|\mu_m\|^2)] \\
&= \arg\max[(2\mu_1^T\mathbf{x} - \|\mu_1\|^2, \ldots, 2\mu_m^T\mathbf{x} - \|\mu_m\|^2)] \\
&= \arg\max[\mathbf{W}^T\mathbf{x} + \mathbf{b}]
\end{aligned}
$$

$$
\text{avec} \quad \mathbf{W}_{:k} = 2v\mu_k \quad \text{et} \quad \mathbf{b}_k = -\|\mu_k\|^2
$$

Binary:

$$
\begin{aligned}
f(\mathbf{x}) &= \text{sign}(\|\mathbf{x} - \mu_-\|^2 - \|\mathbf{x} - \mu_+\|^2) \\
&= \text{sign}(\underbrace{2(\mu_+ - \mu_-)^T}_{\mathbf{w}}\mathbf{x} + \underbrace{(\|\mu_-\|^2 - \|\mu + \|^2))}_{b})
\end{aligned}
$$

Ex 2: Bayes classifier with Gaussian priors

- If the covariance matrices are constrained to be identical $\Sigma_i = \Sigma$ this gives rise to a linear classifier.

- If also the classes are equiprobable, $P(Y = C_i) = P(Y = C_j)$ this is equivalent to a nearest centroid classifier (based on the Euclidean distance if the Gaussians are spherical/isotropic, and on the Mahalanobis distance in the more general case)

- If the covariance matrices are different, we obtain (non-linear) quadratic discriminant functions=> it no longer gives a linear classifier.

Ex 3: linear regression

not a good idea, to use linear regression for classificaion, however it will yield a hyper plane

$$D_n = \{(x^{(1)}, t^{(1)}), \ldots, (x^{(n)}, t^{(n)})\} \qquad t^{(i)} \in \{-1, +1\}$$

- Nothing prevents us from applying linear regression (or ridge) with +1 and -1 targets
- But regression minimizes a quadratic error
  $L(y,t) = (y-t)^2$
  rather than the rate of misclassification...
- => very sensitive to «outliers» (extreme values)
  The decision boundary will be greatly influenced "pulled" by the points farthest from it.
- => will not generalize well for a classification problem.

Linear regression for multi-class classification

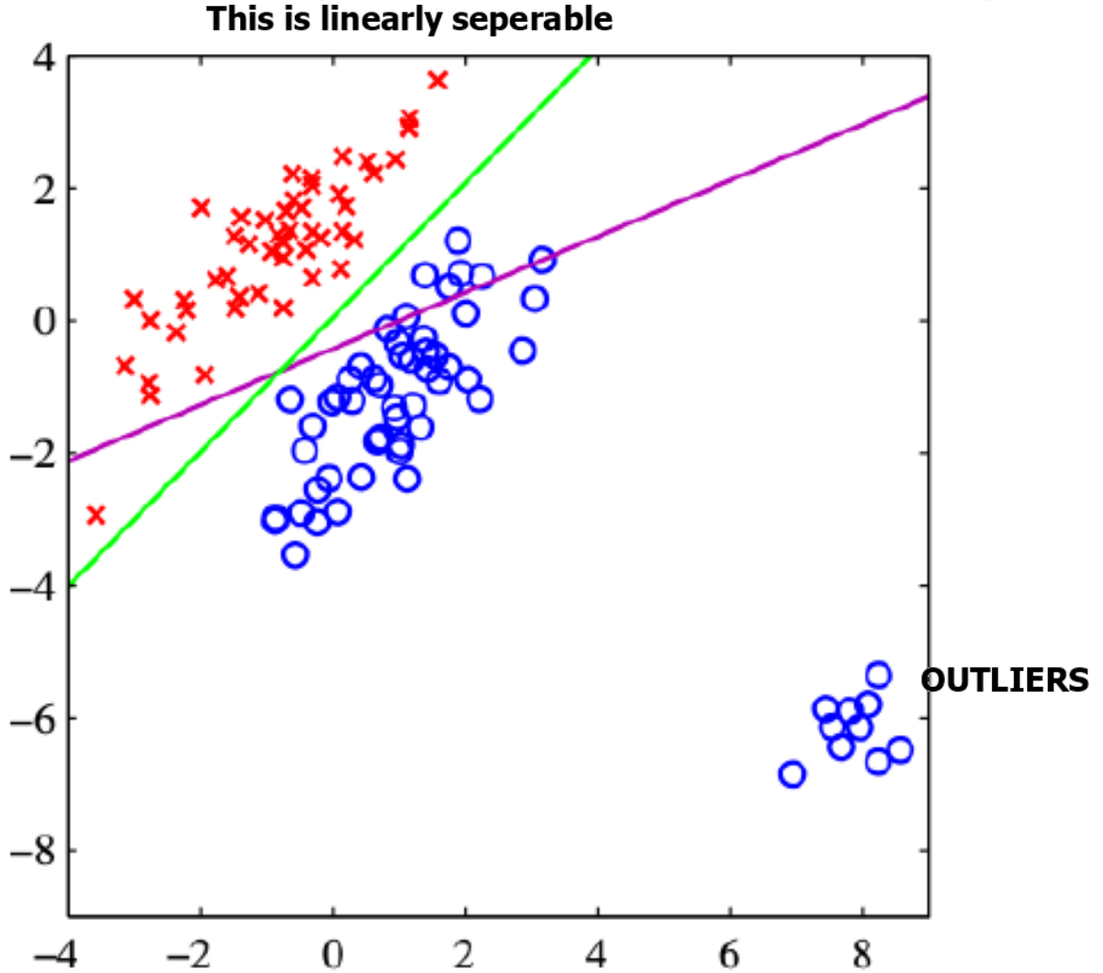
- Least-squares for classification:

$$\mathcal{Y} = \{1, \cdots, k\} \quad \text{and} \quad h_{\mathbf{W}}(\mathbf{x}) = \arg\max_{i=1,\cdots,k} (\mathbf{W}^\top \mathbf{x})_i$$

where $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_k \end{bmatrix} \in \mathbb{R}^{n \times k}$.

  - Learning: encode each output sample y_i to a vector in $\mathbb{R}^k$ using one-hot encoding and minimize the squared error (closed form solution).

- Very sensitive to outliers:



Linear regression for multi-class classification

- Least-squares for classification:

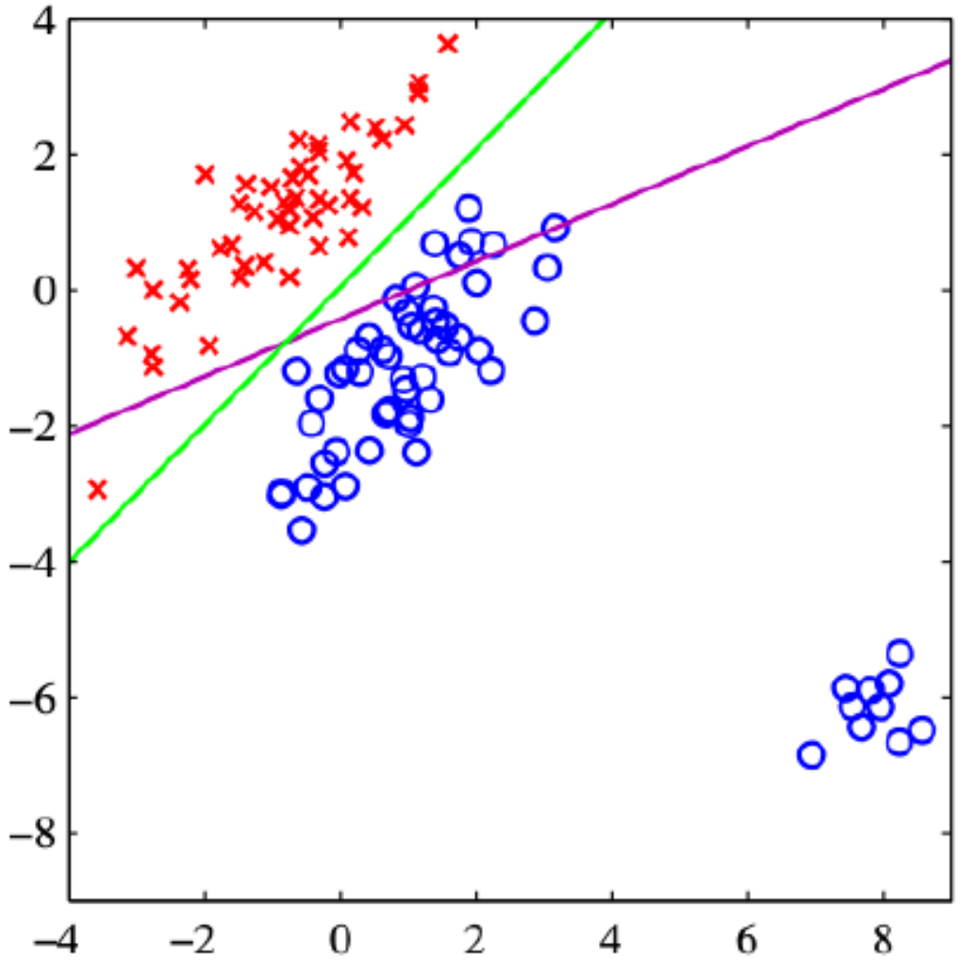
$$\mathcal{Y} = \{1, \cdots, k\} \quad \text{and} \quad h_{\mathbf{W}}(\mathbf{x}) = \arg\max_{i=1,\cdots,k} (\mathbf{W}^\top \mathbf{x})_i$$

where $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_k \end{bmatrix} \in \mathbb{R}^{n \times k}$.

  - Learning: encode each output sample y_i to a vector in $\mathbb{R}^k$ using one-hot encoding and minimize the squared error (closed form solution).

- Very sensitive to outliers:

minimizing the squared error corresponds to maximizing the likelihood under the assumption of a Gaussian conditional distribution.



And if we directly minimized the classification error?

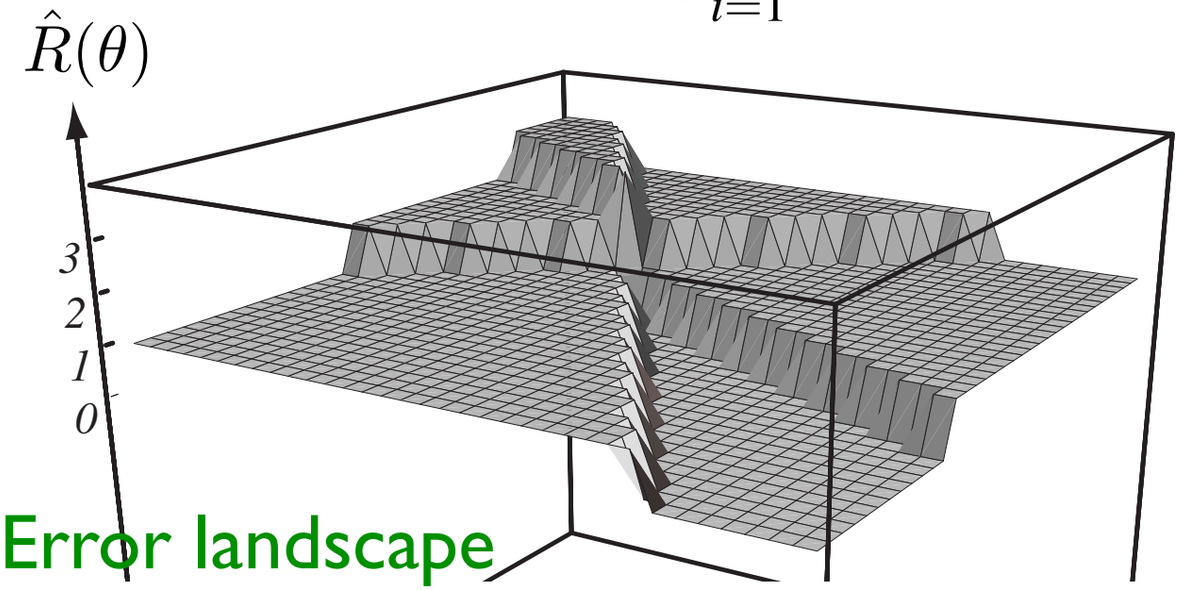
$$D_n = \{(x^{(1)}, t^{(1)}), \ldots, (x^{(n)}, t^{(n)})\} \qquad t^{(i)} \in \{-1, +1\}$$

$$y = f_\theta(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

$$L(y, t) = I_{\{y \neq t\}} = I_{\{(\langle \mathbf{w}, \mathbf{x} \rangle + b)t < 0\}}$$

$$J(\theta) = \hat{R}(f_\theta, D_n) = \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{w}^T x^{(i)} + b, t^{(i)})$$

$$= \frac{1}{n} \sum_{i=1}^{n} I_{\{(\mathbf{w}^T x^{(i)} + b)t^{(i)} < 0\}}$$



Error landscape

With gradient descent?

Problem: zero gradient everywhere!

(because the derivative of the indicator function with respect to the parameters is zero)

=> You can not use gradient descent.

Ex 4: the perceptron algorithm

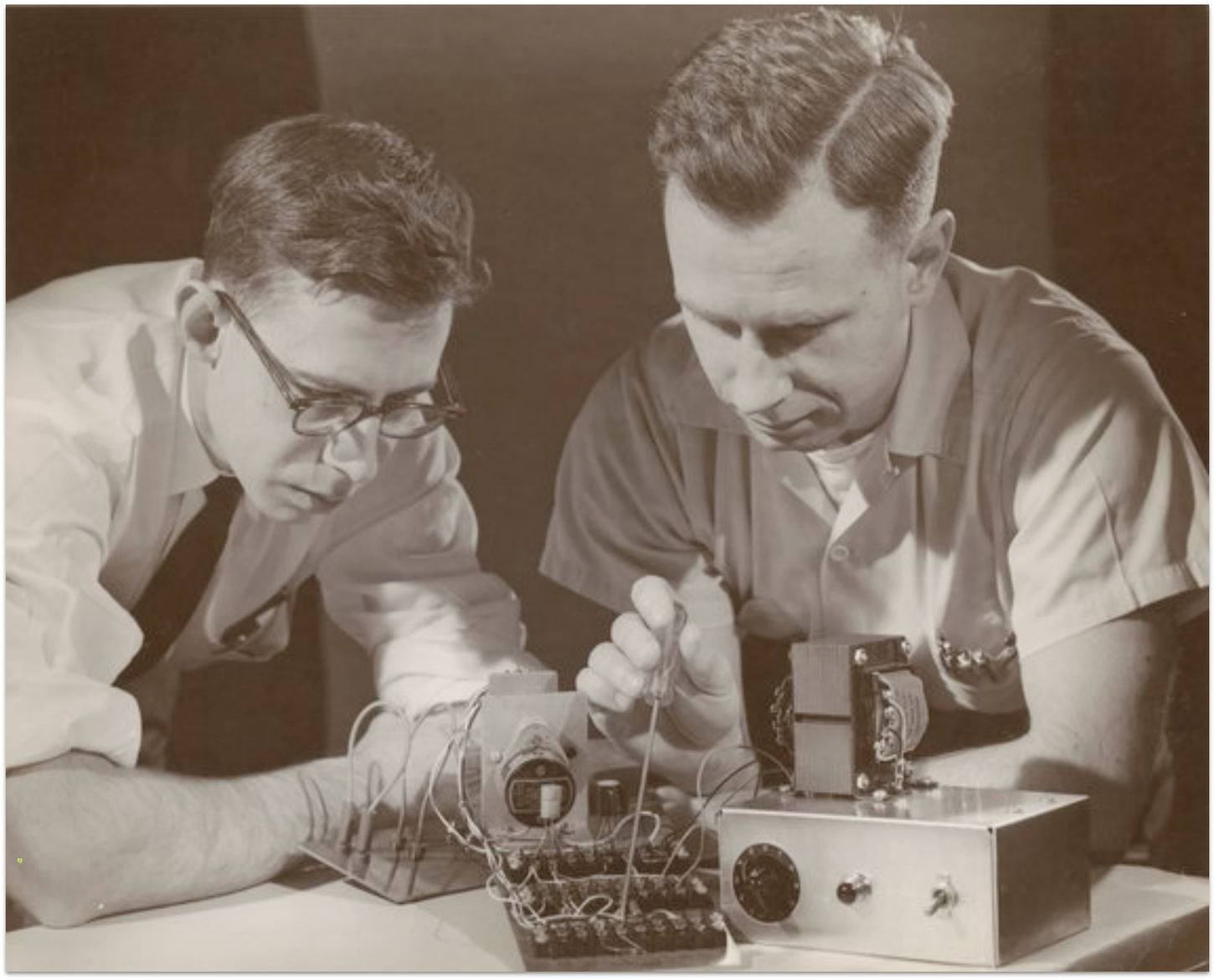
(simple description, online/stochastic version)

$$D_n = \{(x^{(1)}, t^{(1)}), \ldots, (x^{(n)}, t^{(n)})\} \qquad t^{(i)} \in \{-1, +1\}$$

- Initialize the parameters w and b *with* 0
- Repeat as long as misclassified points remain:
  - Choose the next example (x,t) randomly from the dataset D_n.
  - If (with the current parameter values) x is classified correctly i.e. if $(\mathrm{w}^{\mathrm{T}}x+b)t > 0$ we do not change the parameters.
  - Otherwise (x is classified incorrectly) we modify:
    $\mathrm{w} \leftarrow \mathrm{w} + t\,x$
    $b \leftarrow b + t$

Facts about the perceptron

- Invented in 1957 by Frank Rosenblatt.
- Inspired by the cognitive theories of Friedrich Hayek and Donald Hebb.
- One of the first attempts for an "artificial neuron"



The Mark I Perceptron

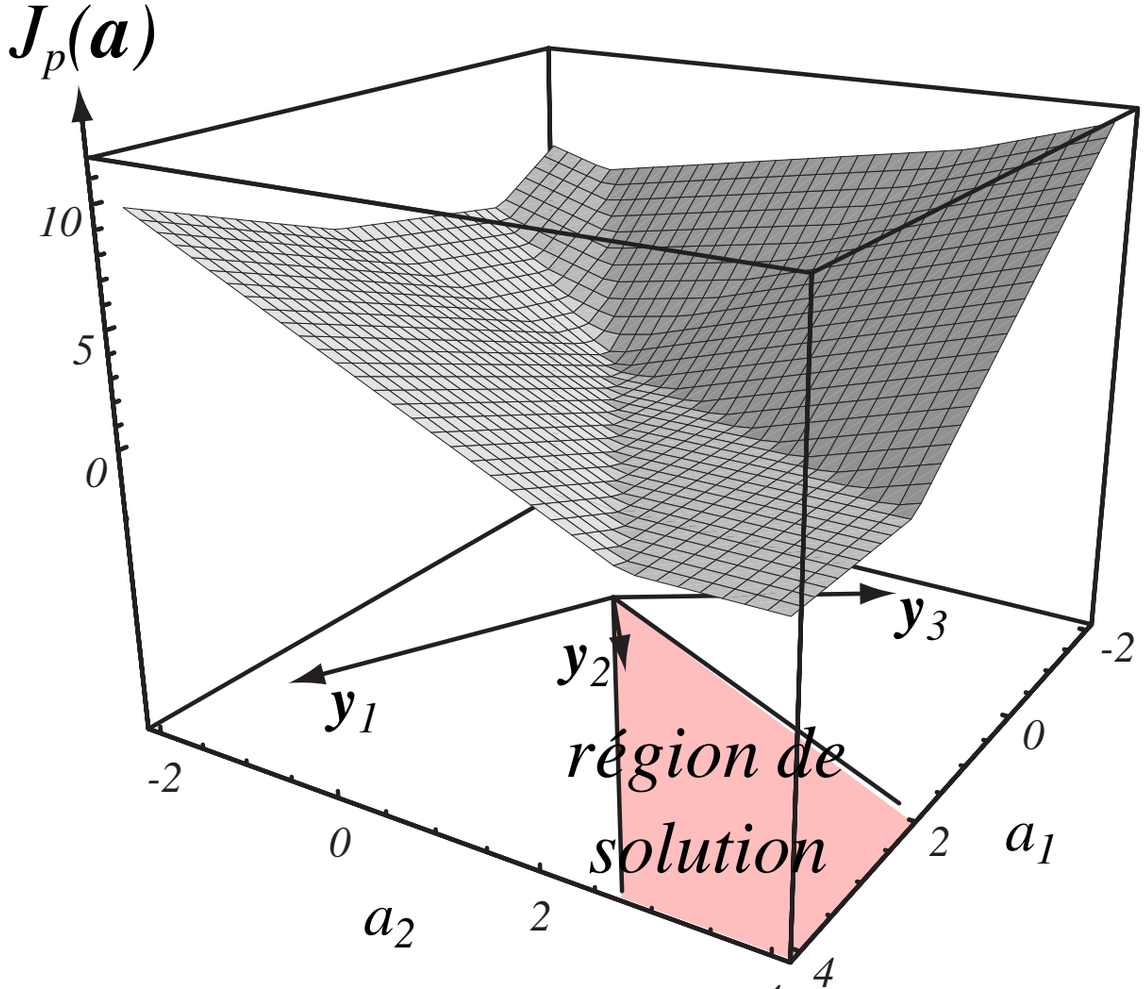
- If the data is linearly separable, the stochastic version (with random selection of the example) is guaranteed to find a hyper-plane separator (giving 0 training errors) in a finite number of iterations.
- If not linearly separable: the method does not terminate.
  => we need another stopping criterion.

Perceptron and gradient descent

$$D_n = \{(x^{(1)}, t^{(1)}), \ldots, (x^{(n)}, t^{(n)})\} \qquad t^{(i)} \in \{-1, +1\}$$

- The perceptron corresponds to a gradient descent on the following objective:

$$J_{\text{perceptron}}(\theta) = \frac{1}{n} \sum_{i=1}^{n} I_{\{(\mathbf{w}^T x^{(i)} + b) t^{(i)} \leq 0\}} (-(\mathbf{w}^T x^{(i)} + b) t^{(i)})$$



- We can derive optimizations by stochastic gradient descent (gives the previously detailed algo), batch, or mini-batch.

Ex 5: logistic regression

(case of two classes) [David Cox 1958]

Attention: despite the name it is a classification algorithm

For a binary classification task: $t \in \{0, 1\}$

We want to estimate the conditional **probability**: $y \simeq P(t = 1|\mathbf{x})$

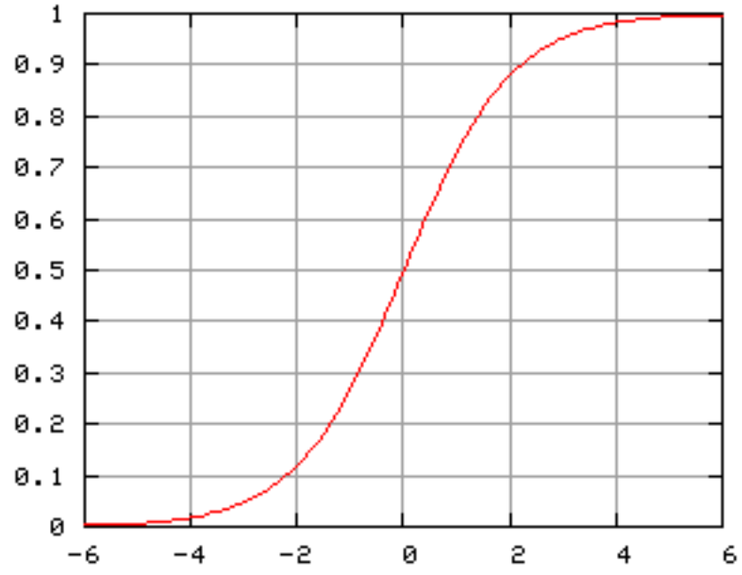
We choose $y \in [0, 1]$

A non-linear prediction function giving a probability

$$y = f_\theta(\mathbf{x}) = f_{\mathbf{w},b}(\mathbf{x}) = \underbrace{\text{sigmoid}(\langle \mathbf{w}, \mathbf{x} \rangle + b)}$$

non-linearity, transfer or activation function

logistic $$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



The logistic sigmoid is the inverse of the logit link function in the terminology of Generalized Linear Models (GLMs).

Cross-entropy loss

$$L(y, t) = -(t \ln(y) + (1 - t) \ln(1 - y))$$

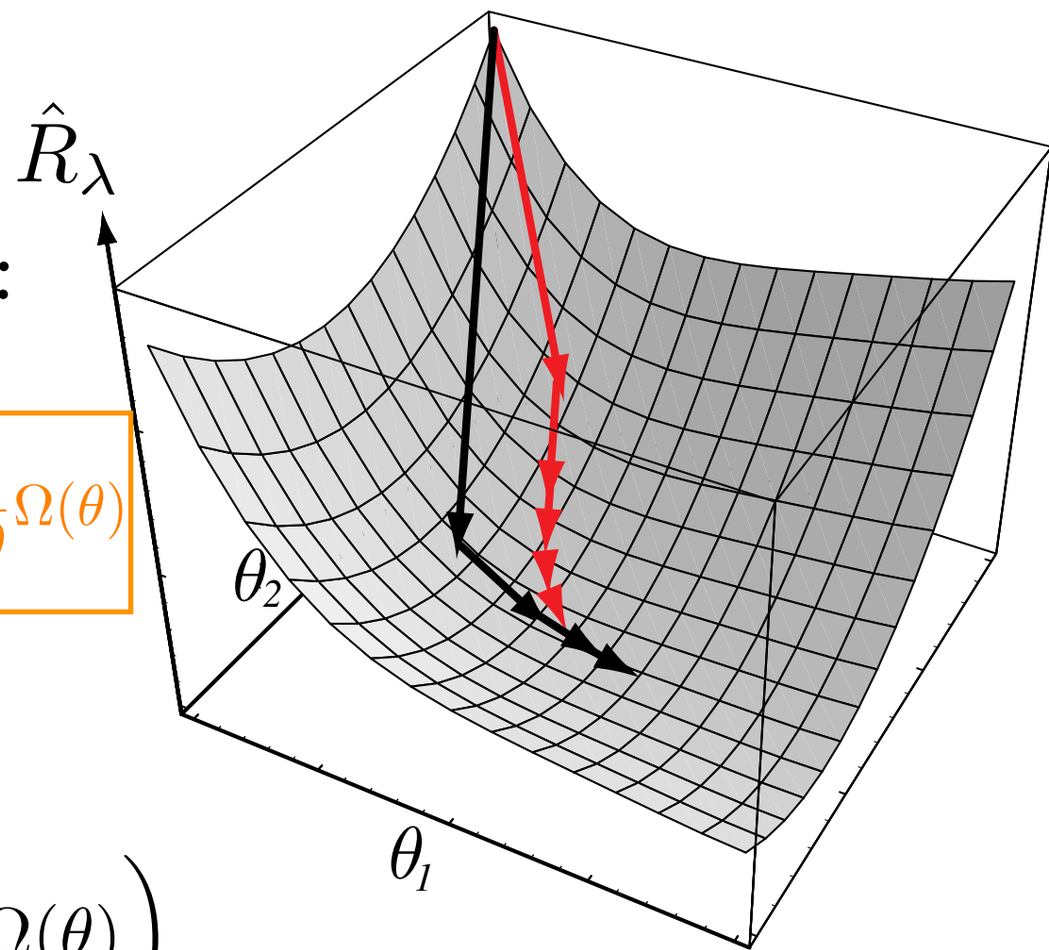
Here, no analytical solution
but the optimization is convex

Optimization by gradient descent

$$D_n = \{(x^{(1)}, t^{(1)}), \ldots, (x^{(n)}, t^{(n)})\} \qquad t^{(i)} \in \{0, 1\}$$

$$\hat{R}_\lambda(f_\theta, D_n) = \underbrace{\left(\sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)}) \right)}_{\text{empirical risk}} + \underbrace{\lambda \Omega(\theta)}_{\text{regularization term}}$$

- we initialize the parameters randomly
- we update them iteratively following the gradient



Either batch gradient descent (whole dataset):

Loop: $\theta \leftarrow \theta - \eta \frac{\partial \hat{R}_\lambda}{\partial \theta}$

$$= \left(\sum_{i=1}^{n} \frac{\partial}{\partial \theta} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)}) \right) + \lambda \frac{\partial}{\partial \theta} \Omega(\theta)$$

Or stochastic gradient descent:

Loop:

For i in 1...n

$$\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} \left(L(f_\theta(\mathbf{x}^{(i)}), t^{(i)}) + \frac{\lambda}{n} \Omega(\theta) \right)$$

Or other variants of the gradient descent idea

(conjugate gradient, Newton's method, natural gradient, ...)

Limitation of logistic regression:

it remains a linear classifier!

We decide class 1 if P(t=1|x) > 0.5 (class 0 otherwise).

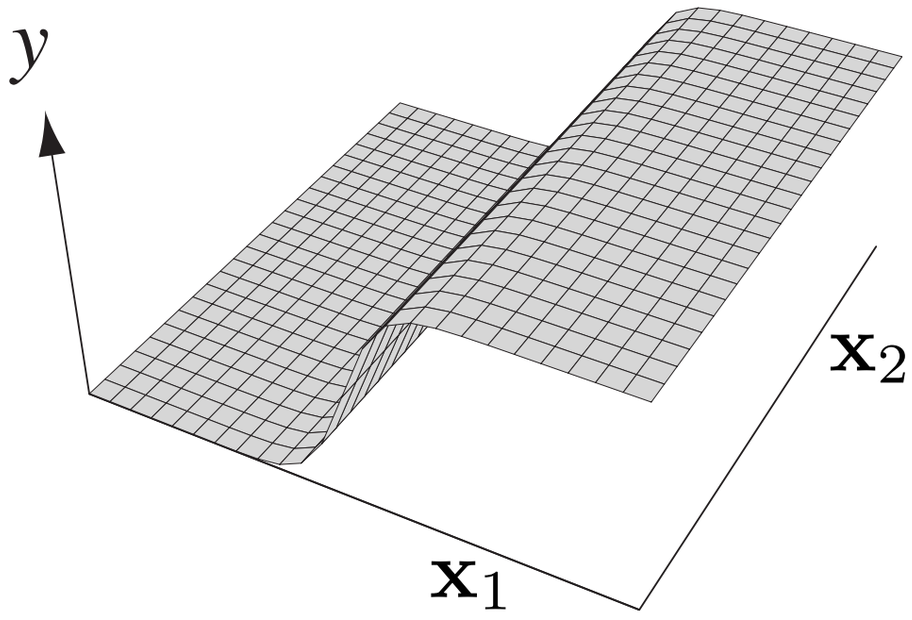
Or sigmoid($\langle \mathbf{w},\mathbf{x} \rangle + b$) > 0.5 $\equiv \langle \mathbf{w},\mathbf{x} \rangle + b > 0$.

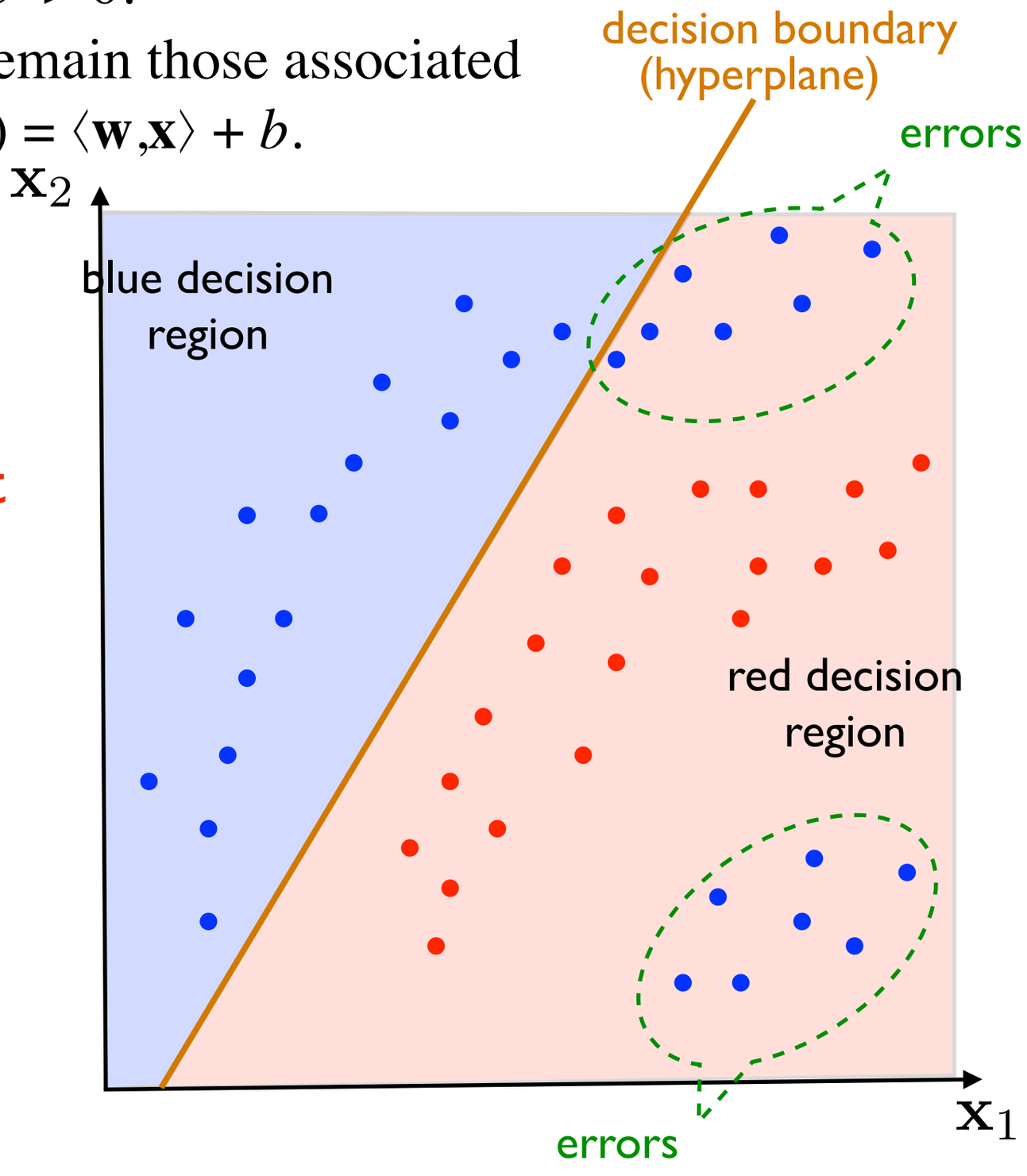
So the regions and decision boundaries remain those associated with the linear discriminant function $g(\mathbf{x}) = \langle \mathbf{w},\mathbf{x} \rangle + b$.

The decision boundary (surface) is a hyperplane

➠ inappropriate if the classes are not linearly separable

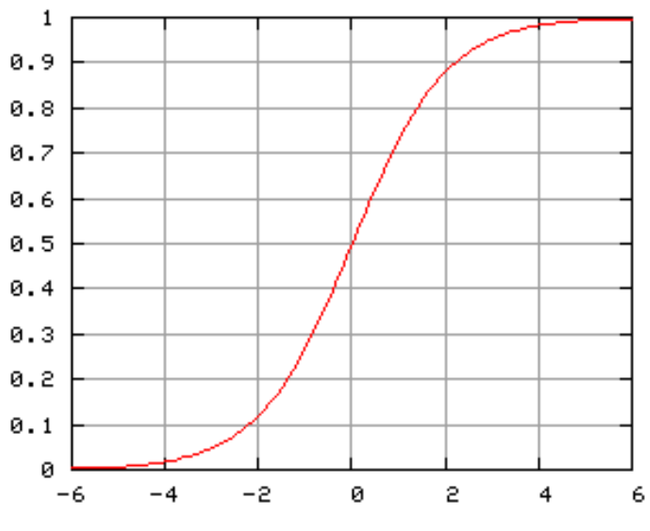
(eg: figure)





Logistic regression

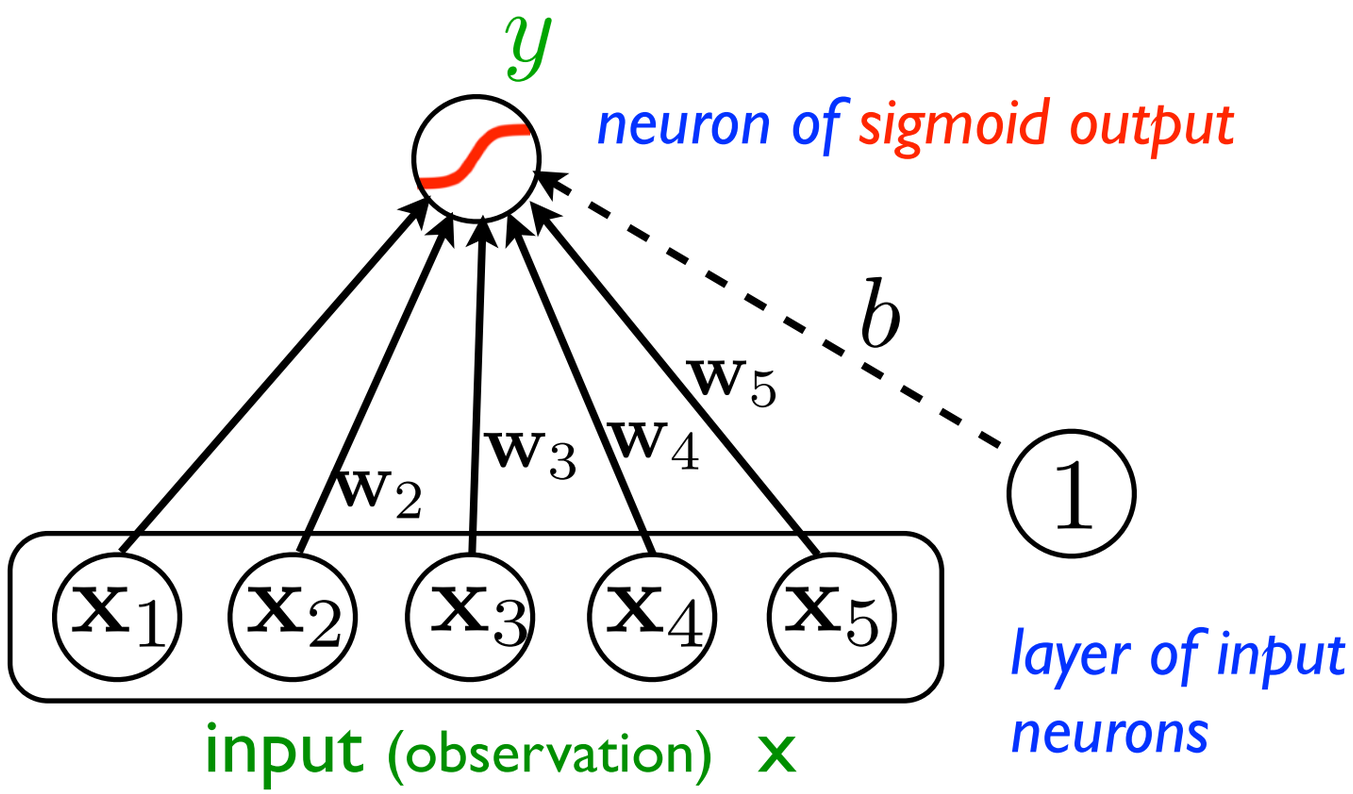
Neuron vision



We can see the sigmoid as

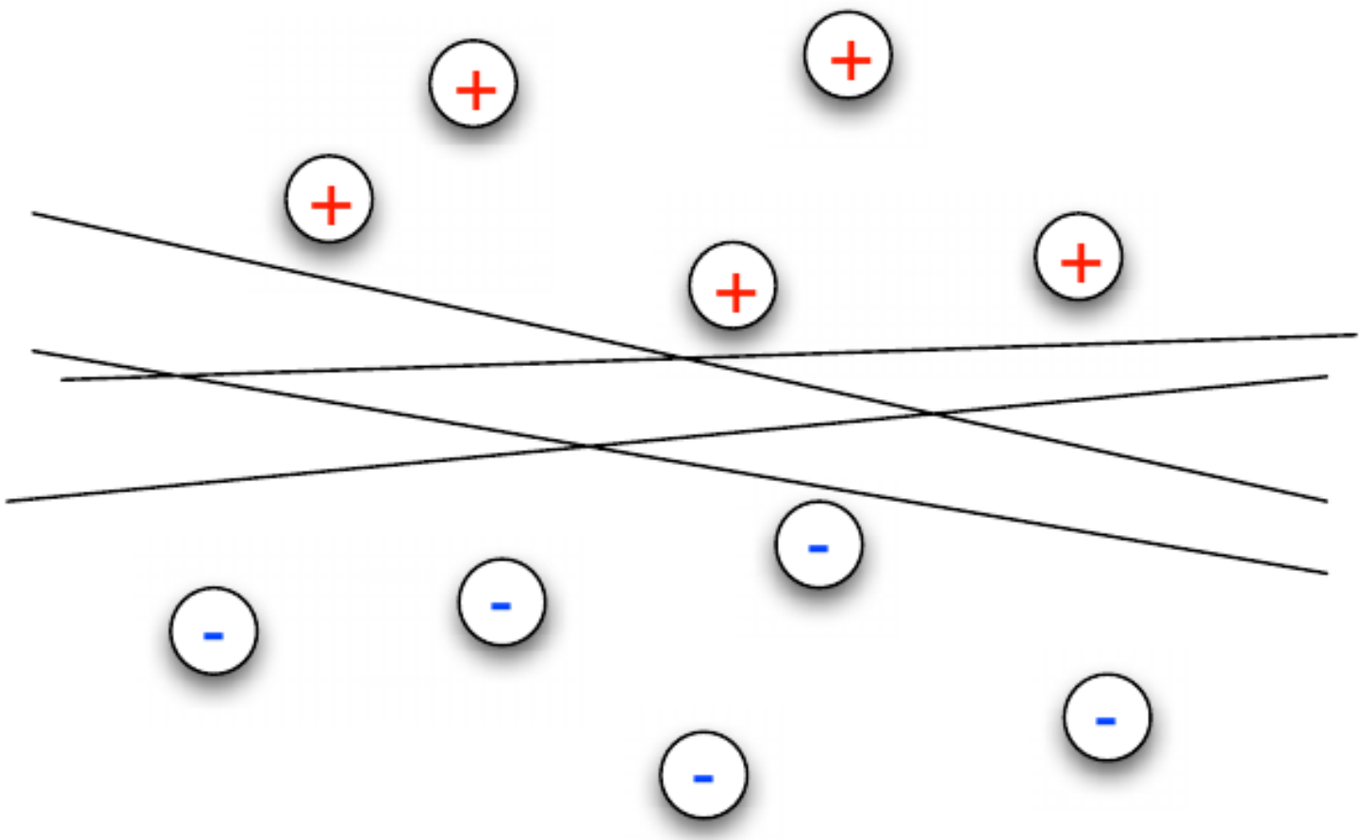
- An differentiable alternative of the indicator function (step function)

- An approximation of the "pulse rate" response in biological neurons



Binary classification revisited

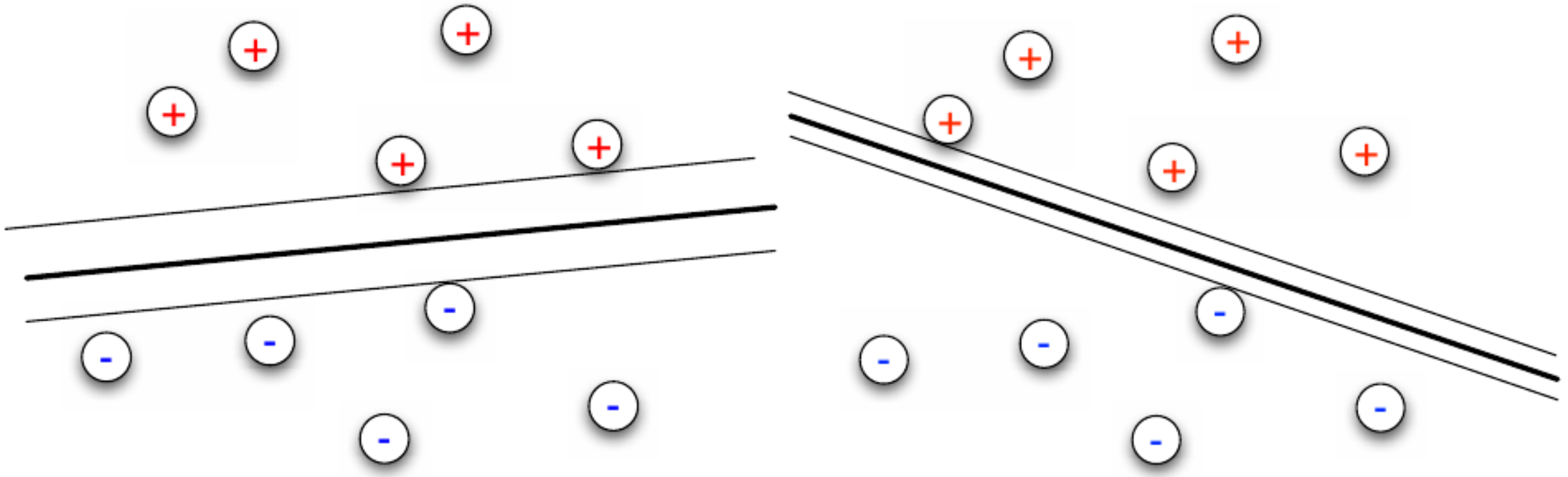
- Consider a linearly separable binary classification data set $\{\mathbf{x}_i, y_i\}_{i=1}^m$.
- There is an infinite number of hyperplanes that separate the classes:



- Which plane is best?
- Relatedly, for a given plane, for which points should we be most confident in the classification?

The margin, and linear SVMs

- For a given separating hyperplane, the *margin* is two times the (Euclidean) distance from the hyperplane to the nearest training example.



- It is the width of the "strip" around the decision boundary containing no training examples.
- A linear SVM is a perceptron for which we choose $\mathbf{w}$, b so that margin is maximized

Ex 6:
Support Vector Machines
(Vapnik & Lerner 1963, Cortes & Vapnik 1995)

- At the base: another algorithm that learns a simple **linear discriminant function** $g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$
- Choose a particular hyperplane, with geometric motivation: **aim for maximum margin.**
- Nonlinear version uses the kernel trick.
- Very popular method
  (in its linear form and in its nonlinear kernel form).

SVM: geometric motivation

Data:

$$D_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$$

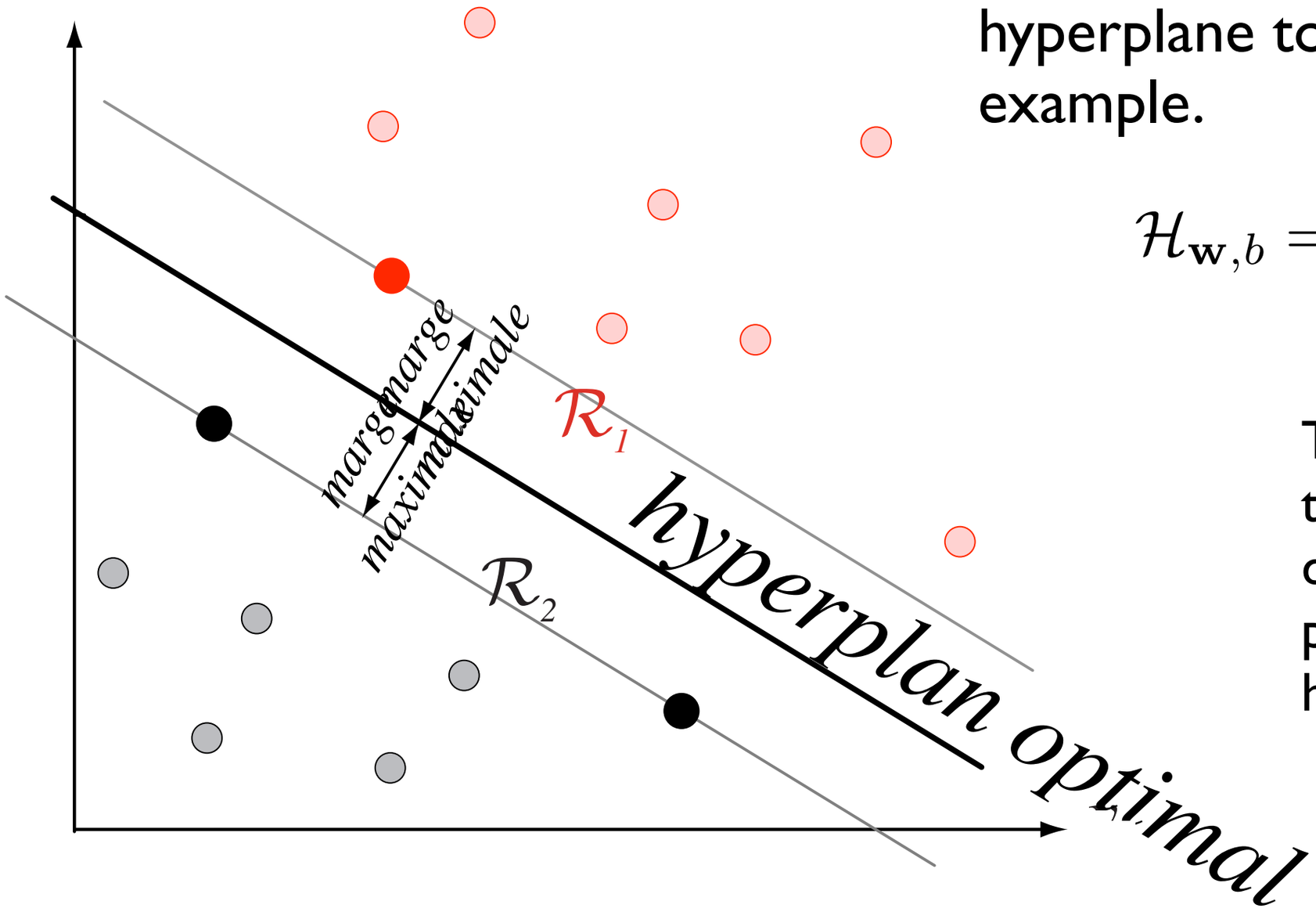
$$\mathbf{x}_i \in \mathbb{R}^d,\ y_i \in \{-1, +1\}$$

- Algorithm learns a maximum margin linear discriminant function.

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

- The margin is the distance from the hyperplane to the nearest training example.

$$\mathcal{H}_{\mathbf{w},b} = \{\mathbf{x} | \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}$$



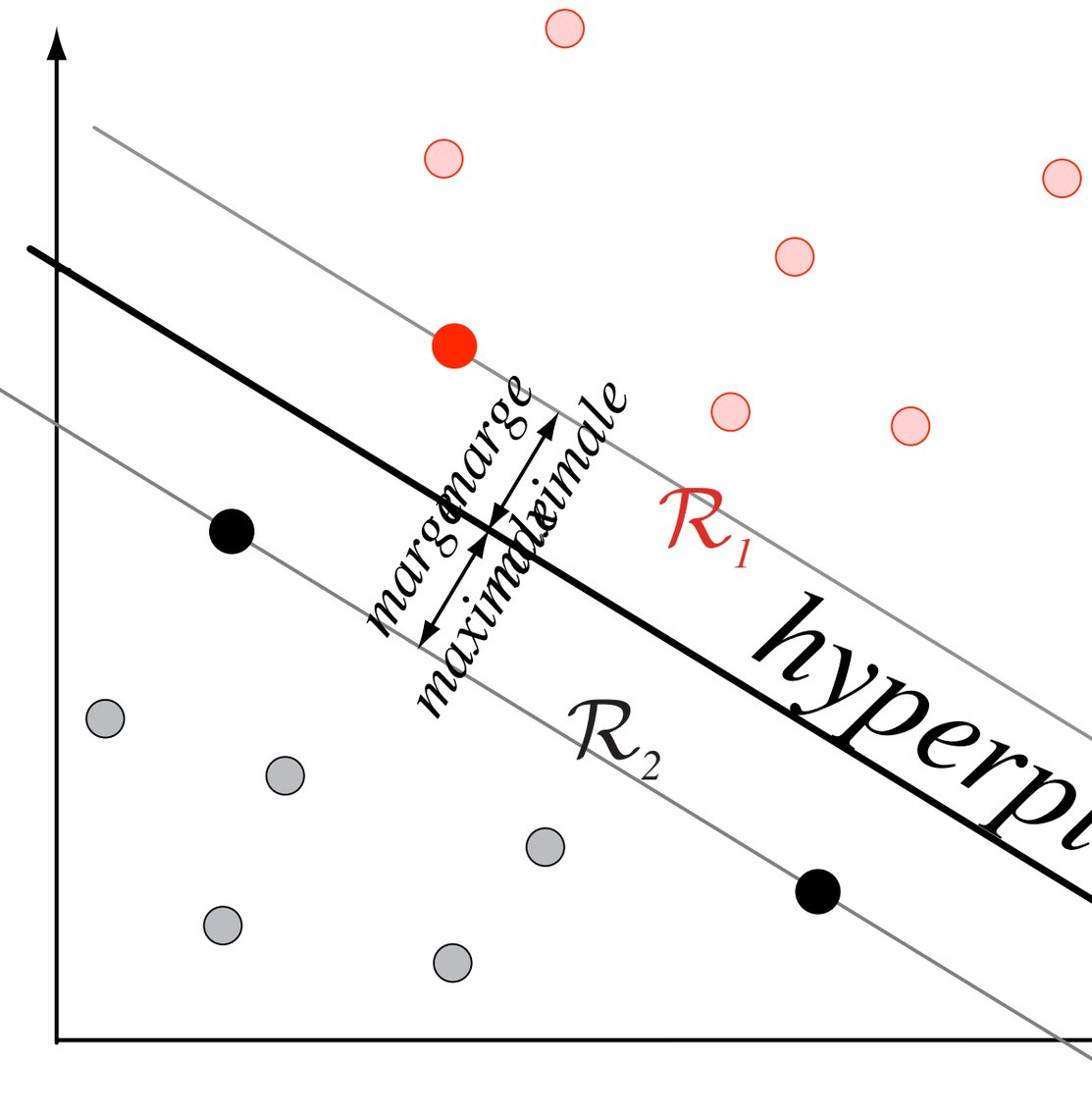
The distance from a point x to the hyperplane is the distance to its orthogonal projection on the hyperplane:

$$\left| \frac{\langle \mathbf{w}, \mathbf{x} \rangle + b}{\|\mathbf{w}\|} \right|$$

SVM with hard margin

The distance from a point x to the hyperplane:

$$\mathbf{d}(\mathbf{x}, \mathcal{H}_{\mathbf{w},b}) = \left| \frac{\langle \mathbf{w}, \mathbf{x} \rangle + b}{\|\mathbf{w}\|} \right|$$



Find the "optimal" hyperplane that:

- perfectly separates the data
  $\forall i \in \{1, \ldots, n\},\ (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) y_i > 0$
- and has the largest margin
  i.e. which is furthest away from the closest training point

$$\arg\max_{\mathbf{w},b} \left\{ \min_i \mathbf{d}(\mathbf{x}_i, \mathcal{H}_{\mathbf{w},b}) \right\}$$

$$= \arg\max_{\mathbf{w},b} \left\{ \min_i \frac{\langle \mathbf{w}, \mathbf{x}_i \rangle + b}{\|\mathbf{w}\|} y_i \right\}$$

Margin = confidence

- Definition: geometric margin of an example (x,y) is its signed distance from the hyperplane (sign indicates if classification was correct):

$$\gamma^{(g)}(\mathbf{x}, y) = \frac{\langle \mathbf{w}, \mathbf{x} \rangle + b}{\|\mathbf{w}\|} y$$

- Remark: the norm of w does not change the hyperplane

- Definition: functional margin of an example (x,y)

$$\gamma(\mathbf{x}, y) = g(\mathbf{x})y = (\langle \mathbf{w}, \mathbf{x} \rangle + b)y$$

- The signs of the margin indicates if classification is correct
- The magnitude (absolue value) of the margin indicates the «confidence» of the classifier in its prediction.
- An example with a large positive margin is classified correctly with great confidence.
- A point with a negative margin of high magnitude is misclassified with great confidence.

SVM with hard margin

Equivalent optimization problems

(with linearly separable data)

- Maximize the smallest geometric margin

$$\arg\max_{\mathbf{w},b}\left\{\min_i \frac{\langle \mathbf{w}, \mathbf{x}_i\rangle + b}{\|\mathbf{w}\|} y_i\right\}$$

- Maximize the smallest functional margin, under constraint ||w||=1

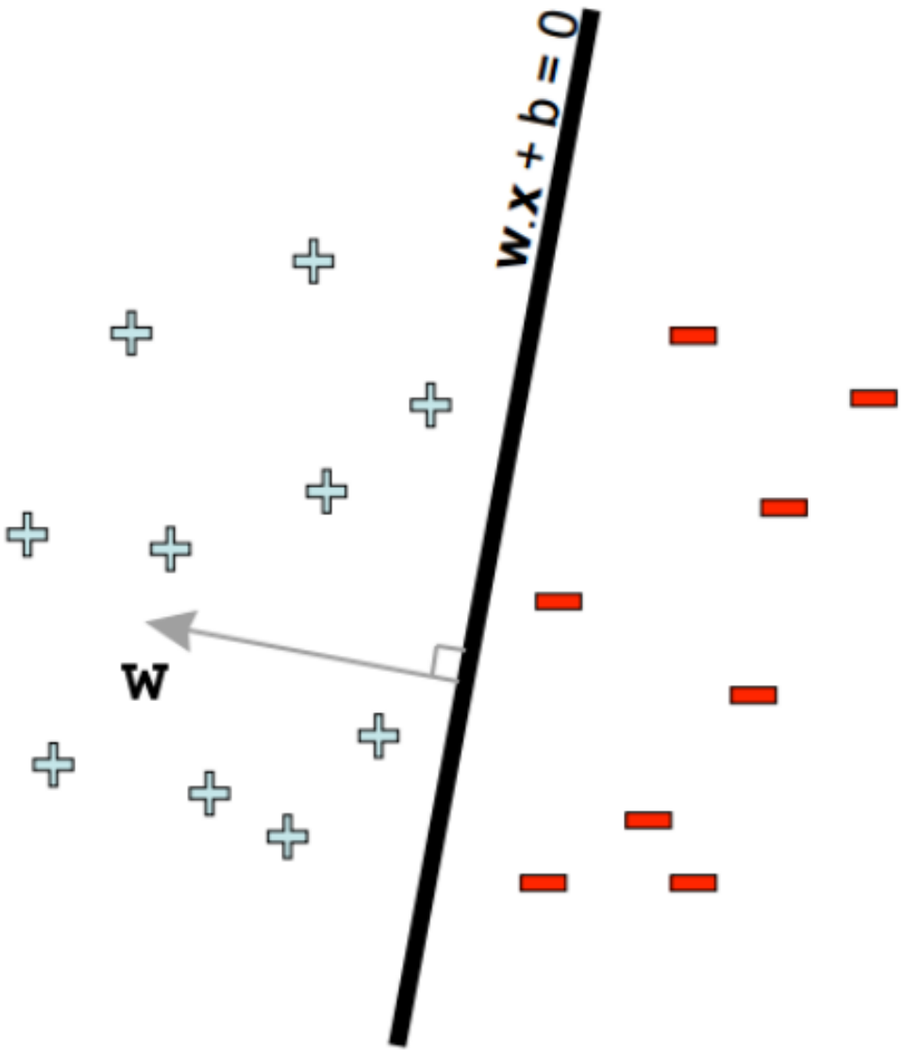
$$\arg\max_{\mathbf{w},b}\left\{\min_i (\langle \mathbf{w}, \mathbf{x}_i\rangle + b) y_i\right\}, \text{ s.c.} \|\mathbf{w}\|^2 = 1$$

- Minimize ||w|| under a functional margin constraint ≥1

$$\arg\min_{\mathbf{w}} \|\mathbf{w}\|^2$$
$$\text{s.c. } \forall i \in \{1,\ldots,n\}\ (\langle \mathbf{w}, \mathbf{x}_i\rangle + b) y_i \geq 1$$

- In these three cases, we obtain the same solution, with a potentially different norm for w (which does not change the hyperplane)
- The last wording above corresponds to a problem of Quadratic programming (QP) with constraints (for an SVM with hard margin).

Hyperplanes and scale invariance

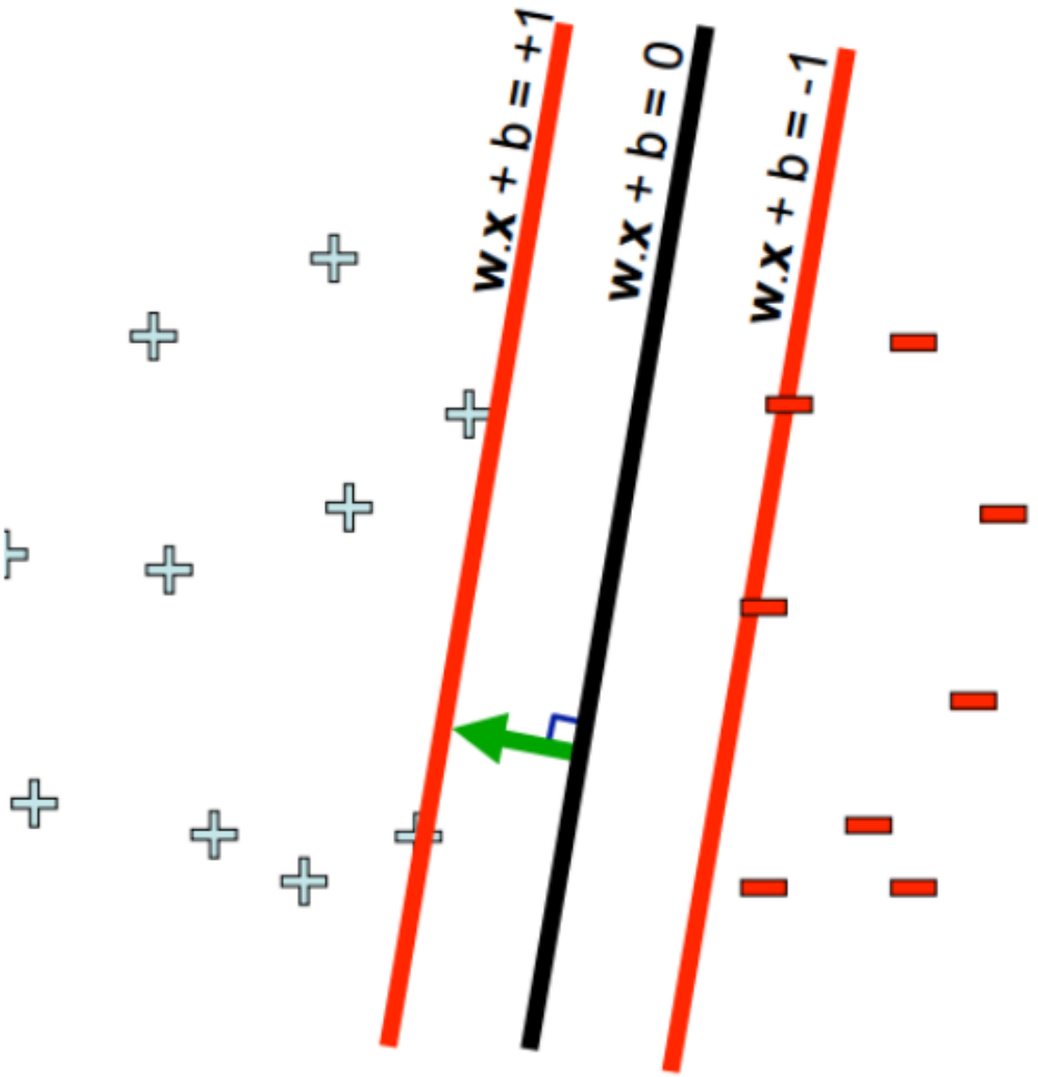


- Hyperplane: $\mathbf{w}^\top \mathbf{x} + b = 0$.

$\Rightarrow$ Scale invariance: $(\alpha \mathbf{w})^\top \mathbf{x} + \alpha b = 0$ define the same hyperplane for any scalar $\alpha \in \mathbb{R}$.

Figures taken form David Sontag's lecture slides:
`http://people.csail.mit.edu/dsontag/courses/ml13/slides/lecture3.pdf`

Hyperplanes and scale invariance



- Hyperplane: $\mathbf{w}^\top \mathbf{x} + b = 0$.

$\Rightarrow$ Scale invariance: $\alpha \mathbf{w}^\top \mathbf{x} + \alpha b = 0$...

- Workaround: we set the scale by enforcing

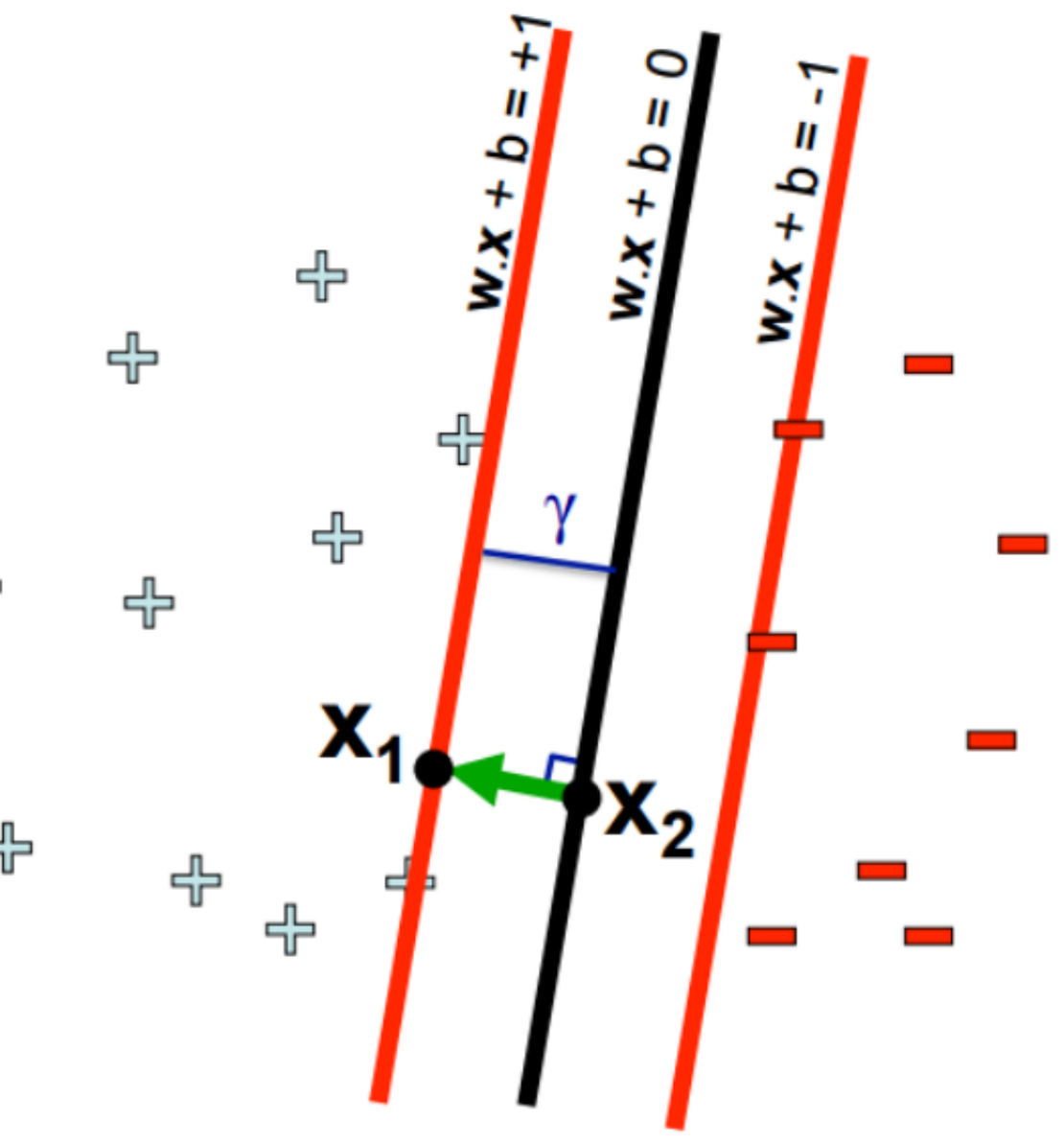
$$\mathbf{w}^\top \mathbf{x} + b \geq 1 \quad \text{for all positive examples}$$
$$\mathbf{w}^\top \mathbf{x} + b \leq -1 \text{ for all negative examples}$$

$\rightarrow$ equivalently, we want to satisfy the linear constraints

$$y_i(\mathbf{w}^\top \mathbf{x} + b) \geq 1$$

for all i (here $\mathcal{Y} = \{-1, 1\}$).

Margin as a function of the normal vector



- Let $\mathbf{x}_1$ be a point on the boundary and $\mathbf{x}_2$ its projection on the hyperplane:

$$\mathbf{w}^\top \mathbf{x}_1 + b = 1$$

$$\mathbf{w}^\top \mathbf{x}_2 + b = 0$$

  We get $\mathbf{w}^\top(\mathbf{x}_1 - \mathbf{x}_2) = 1$.

- But we also know that

$$\mathbf{x}_1 - \mathbf{x}_2 = \gamma \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

- Hence $\gamma = \dfrac{1}{\|\mathbf{w}\|}$: maximizing the margin is equivalent to minimizing the norm of $\mathbf{w}$!

SVM with hard margin

Equivalent optimization problems

(with linearly separable data)

- Maximize the smallest geometric margin

$$\arg\max_{\mathbf{w},b}\left\{\min_i \frac{\langle \mathbf{w}, \mathbf{x}_i\rangle + b}{\|\mathbf{w}\|} y_i\right\}$$

- Maximize the smallest functional margin, under constraint ||w||=1

$$\arg\max_{\mathbf{w},b}\left\{\min_i (\langle \mathbf{w}, \mathbf{x}_i\rangle + b) y_i\right\}, \text{ s.c.} \|\mathbf{w}\|^2 = 1$$

- Minimize ||w|| under a functional margin constraint ≥1

$$\arg\min_{\mathbf{w}} \|\mathbf{w}\|^2$$
$$\text{s.c. } \forall i \in \{1,\ldots,n\}\ (\langle \mathbf{w}, \mathbf{x}_i\rangle + b) y_i \geq 1$$

- In these three cases, we obtain the same solution, with a potentially different norm for w (which does not change the hyperplane)
- The last wording above corresponds to a problem of Quadratic programming (QP) with constraints (for an SVM with hard margin).

Soft margin SVM

- Relaxation of constraints to tolerate errors on training set D_n

$$\min_{\mathbf{w},\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

avec les contraintes $\forall i \in \{1,\ldots,n\}$

$(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) y_i \geq 1 - \xi_i$

$\xi_i \geq 0$

- The examples that violate the margin have $\xi_i > 0$ (=0 for the others)
- => Quadratic programming (QP) with constraints.
- C is a hyperparameter (positive scalar) which controls tolerance to errors
- C→+∞ corresponds to an SVM with a hard margin (no errors tolerated, D_n must be linearly separable)

Soft margin SVM
formulation without constraints

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{s.c.} \quad \xi_i \geq 0 \text{ et } (\langle \mathbf{w}, \mathbf{x}_i\rangle + b)y_i \geq 1 - \xi_i$$

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{s.c.} \quad \xi_i \geq 0 \text{ et } \xi_i \geq 1 - (\langle \mathbf{w}, \mathbf{x}_i\rangle + b)y_i$$

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{s.c.} \quad \xi_i \geq \max(0, 1 - (\langle \mathbf{w}, \mathbf{x}_i\rangle + b)y_i)$$

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{s.c.} \quad \xi_i = \max(0, 1 - (\langle \mathbf{w}, \mathbf{x}_i\rangle + b)y_i)$$

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max(0, 1 - (\langle \mathbf{w}, \mathbf{x}_i\rangle + b)y_i)$$

hinge loss: $\max(0, 1 - m)$ where *m* is the functional margin

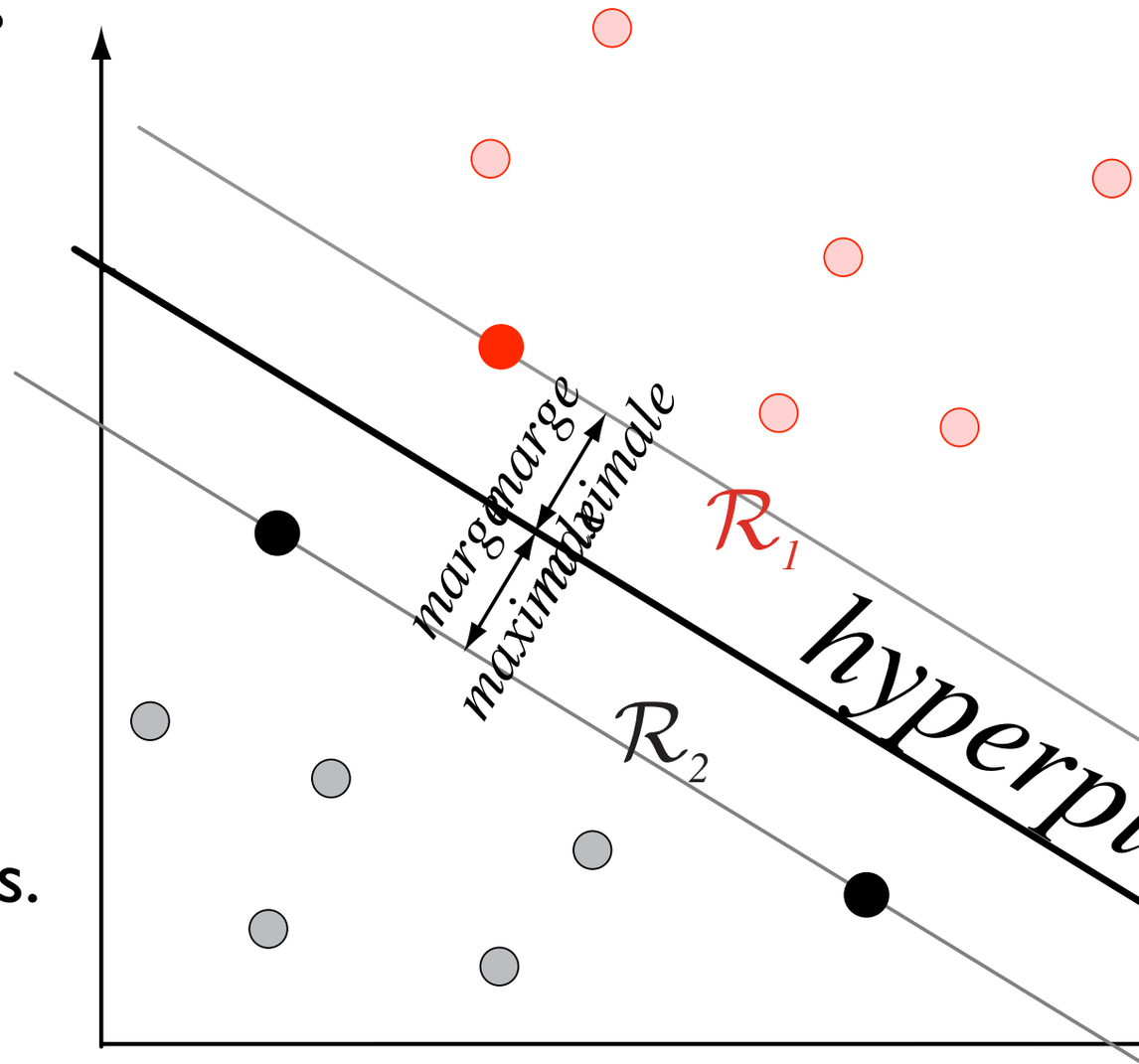
SVM: form of the solution

- The solution of the minimization problem gives a w that has the following form:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

where $\alpha_i \geq 0$

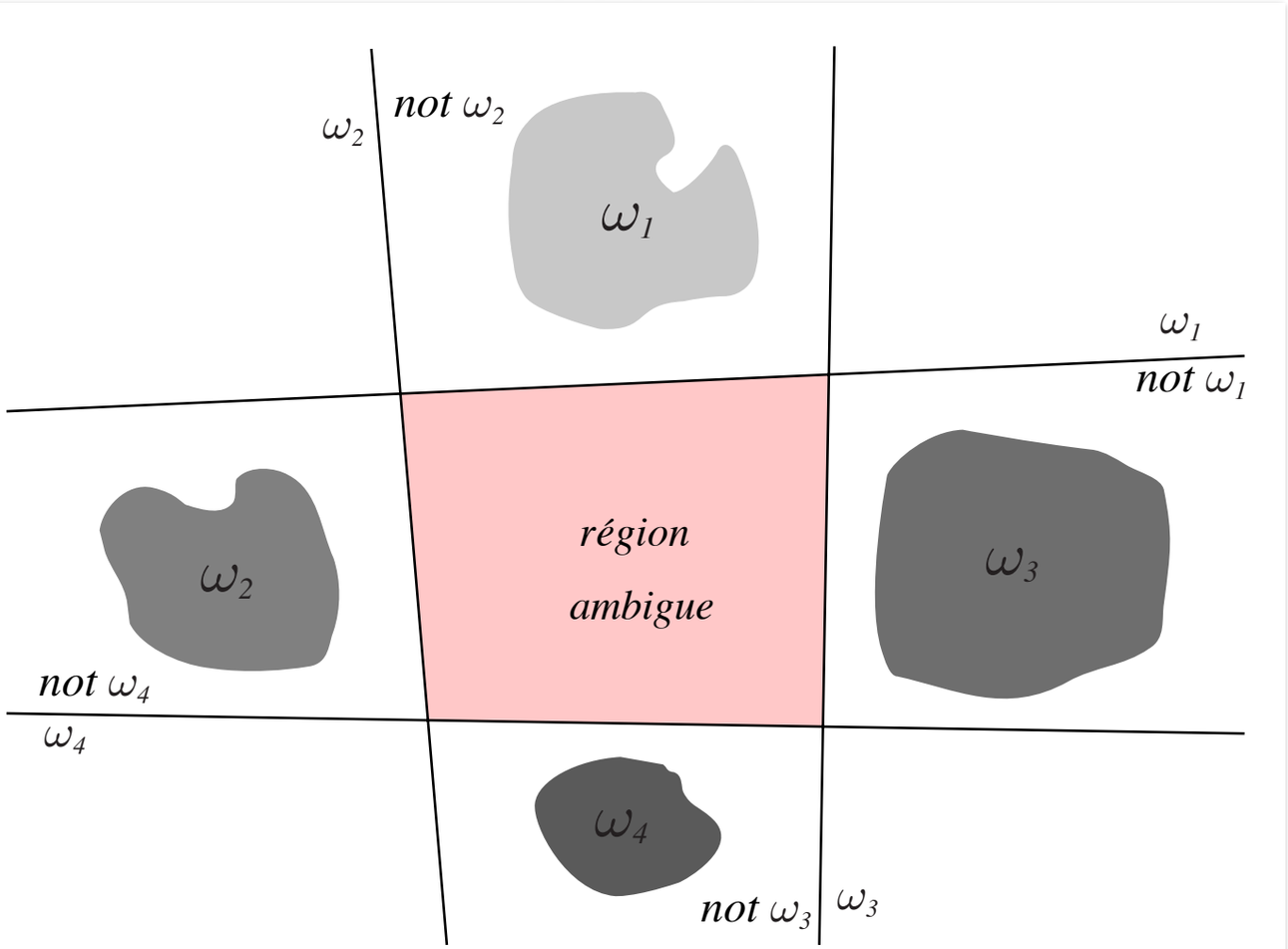
- Typically only a small fraction of the learning examples will have their $\alpha_i > 0$.
- The solution is therefore expressed according to these few training examples.
- We call those points support vectors.
- They are on the margin (or inside it if we tolerate errors, flexible margin).



Allows for a "dual formulation" of this optimization problem: where we learn α_i rather than w_j

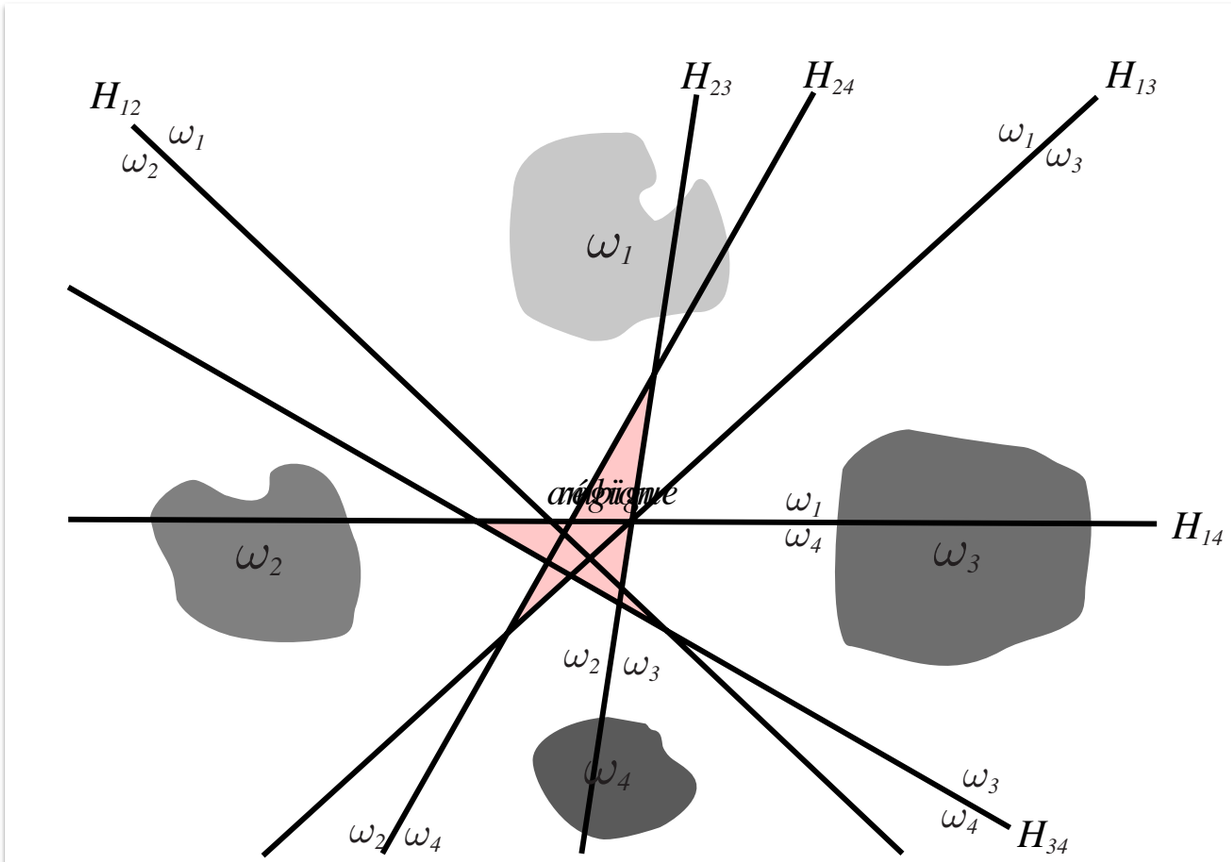
Using binary classifiers for a problem of m classes

One against all: m decisions or scores



We can choose the highest score

One against one: *m(m-1)/2* decisions



We can make all these classifiers vote

Nonlinear and multiclass SVM

- To obtain a nonlinear SVM we apply the **kernel trick** (which we will see in a next class)
- To obtain a classifier with m classes, one trains several SVM of binary classification:
  - either m classifiers, each "one class against all other examples". => we get then m scores
  - a classifier for each class pair: we then obtain m (m-1) / 2 values
- There are many other SVM extensions, especially for regression.

SVM summary

- Geometric idea leads to a quadratic optimization problem with constraints.
- This gives the "hard margin SVM" algorithm and requires linearly separable data.
- An extension called "soft margin" can find a solution in the case not separable.
- In addition, the kernel trick can be applied to give a nonlinear classifier.
- SVMs with soft margin (kernel or not) are a very very popular algorithm (good performance, few hyper-parameters).

Numerous techniques to learn a linear discriminant function

- Nearest centroid algorithm
- Bayes classifier with Gaussian densities (with identical covariance constraints for both classes)
- Linear Regression (or ridge regression) with targets -1 +1
- Perceptron
- Logistic regression (possibly regularized as ridge)
- (Fisher) Linear Discriminant Analysis (LDA)
- Support Vector Machine (Linear)

All these methods will give a different decision boundary but it will always be a hyperplane! (➟ limited capacity)