

# **Apprentissage Automatique**

## Régularisation / SVM + régression

**Stéphane Herbin**

[stephane.herbin@onera.fr](mailto:stephane.herbin@onera.fr)

# Aujourd'hui

- Approfondissement:
  - Régularisation
  - Un algorithme efficace: Support Vector Machines (SVM)
  - Multiclasse
  - Régression
- TD:
  - SVM: étude de l'influence des paramètres
  - Validation croisée (reprise!)

# Apprentissage supervisé (rappel)

- On veut construire une fonction de décision  $F$  à partir d'exemples
- On dispose d'un **ensemble d'apprentissage**  $\mathcal{L}$  sous la forme de paires  $\{x_i, y_i\}$  où  $x_i$  est la donnée à classer et  $y_i$  est la classe vraie:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1 \dots n}$$

- L'apprentissage consiste à identifier cette fonction de classification dans un certain espace **paramétrique**  $W$  optimisant un certain **critère**  $L$ :

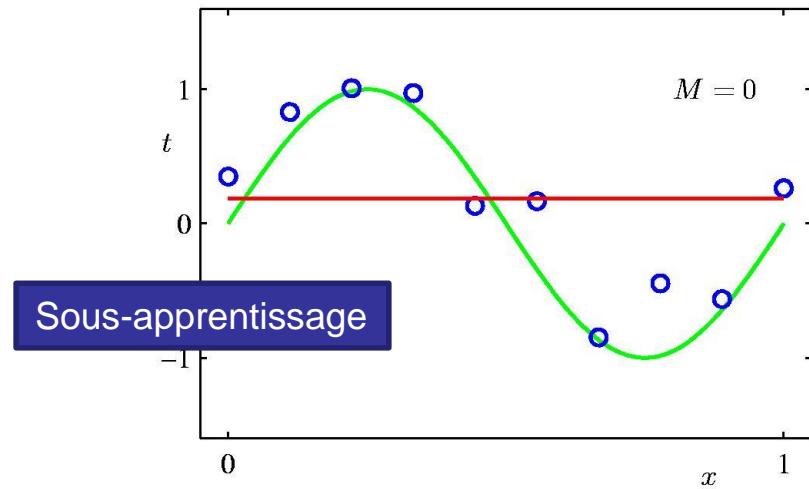
$$W = \arg \min_{W'} L(\mathcal{D}, W')$$

- On l'applique ensuite à de nouvelles données.

$$y = f(x; W)$$

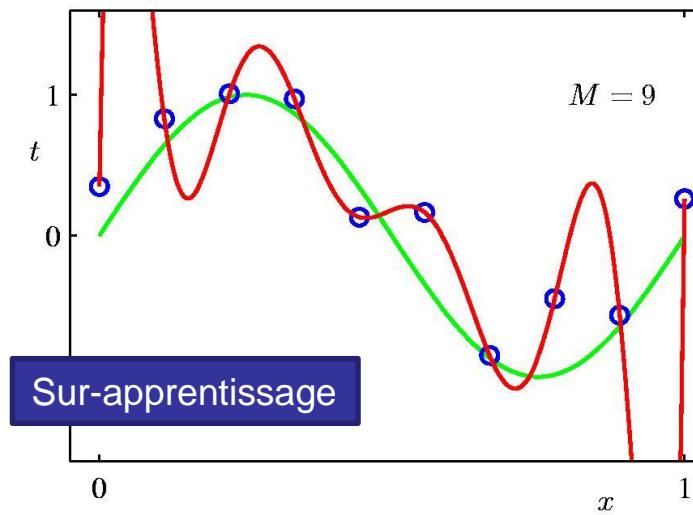
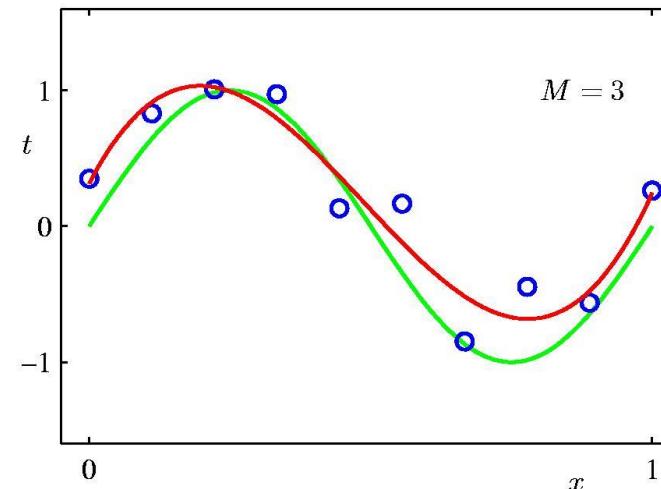
# Régularisation

# Retour sur le sur-apprentissage



	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^*$	0.19	0.82	0.31	0.35
$w_1^*$		-1.27	7.99	232.37
$w_2^*$			-25.43	-5321.83
$w_3^*$			17.37	48568.31
$w_4^*$				-231639.30
$w_5^*$				640042.26
$w_6^*$				-1061800.52
$w_7^*$				1042400.18
$w_8^*$				-557682.99
$w_9^*$				125201.43

Coefficients des polynômes



Très grandes valeurs!

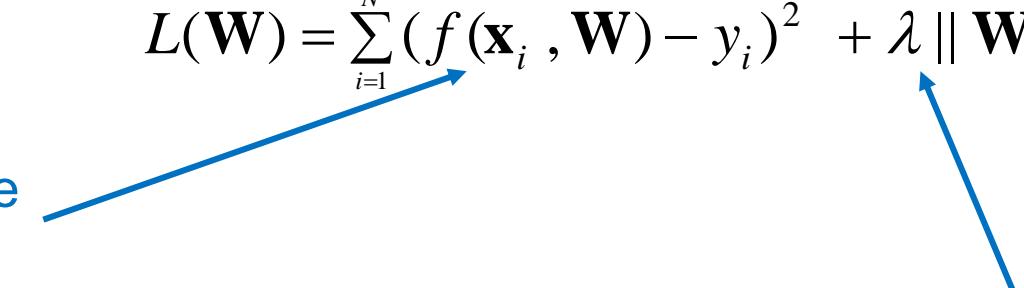
# Moindre carrés régularisés

Idée: on rajoute une pénalisation des grandes valeurs des paramètres à la fonction de coût:

$$L(\mathbf{W}) = \sum_{i=1}^N (f(\mathbf{x}_i, \mathbf{W}) - y_i)^2 + \lambda \|\mathbf{W}\|^2$$

Coût d'attaché aux données

Paramètre de régularisation

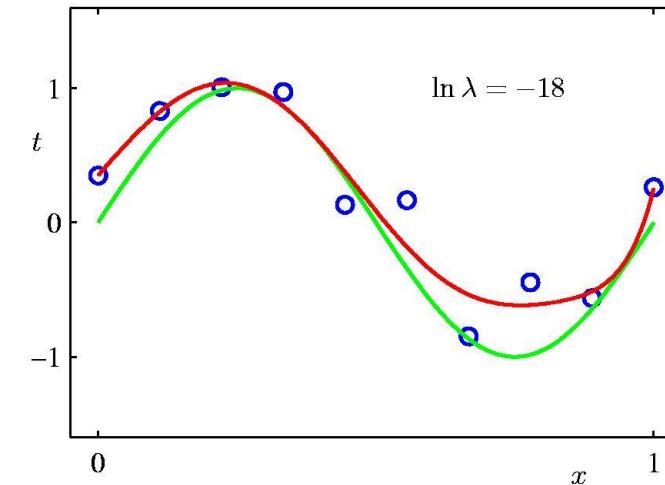
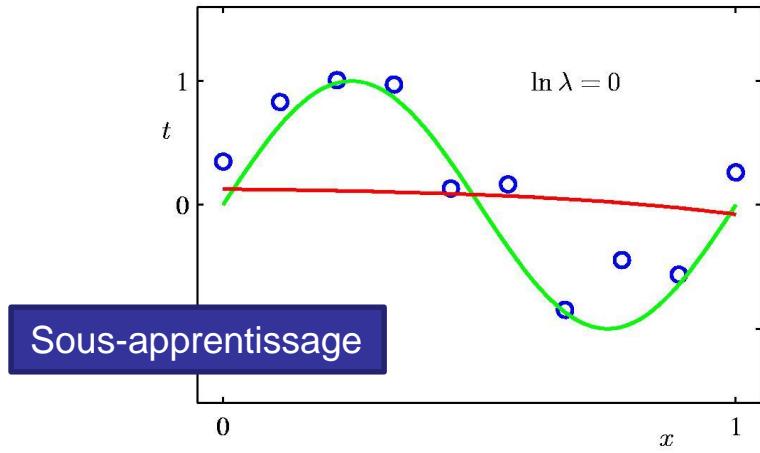


Dont l'optimum exact est alors:

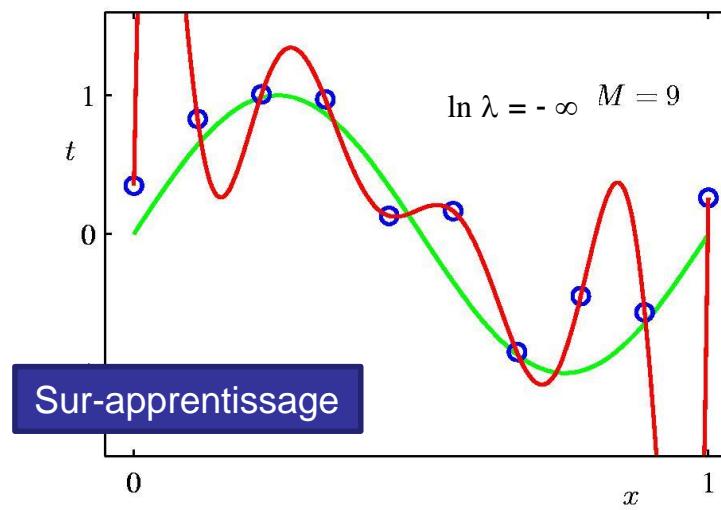
$$\mathbf{W}^* = (\Phi^t \Phi + \lambda I)^{-1} \Phi^t Y$$

Si on pénalise les grandes valeurs des coefficients du polynôme, on obtient une fonction moins « zigzagante »

# Effet de la régularisation

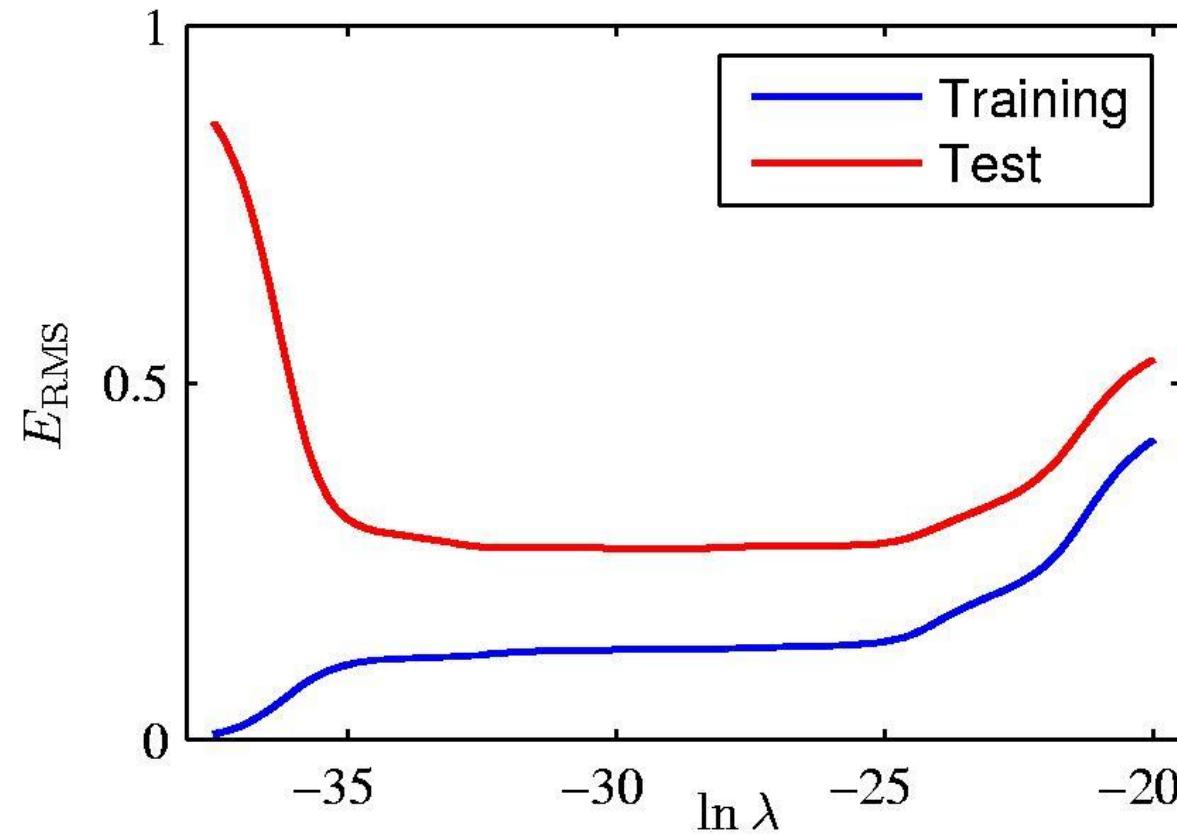


	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^*$	0.35	0.35	0.13
$w_1^*$	232.37	4.74	-0.05
$w_2^*$	-5321.83	-0.77	-0.06
$w_3^*$	48568.31	-31.97	-0.05
$w_4^*$	-231639.30	-3.89	-0.03
$w_5^*$	640042.26	55.28	-0.02
$w_6^*$	-1061800.52	41.32	-0.01
$w_7^*$	1042400.18	-45.95	-0.00
$w_8^*$	-557682.99	-91.53	0.00
$w_9^*$	125201.43	72.68	0.01



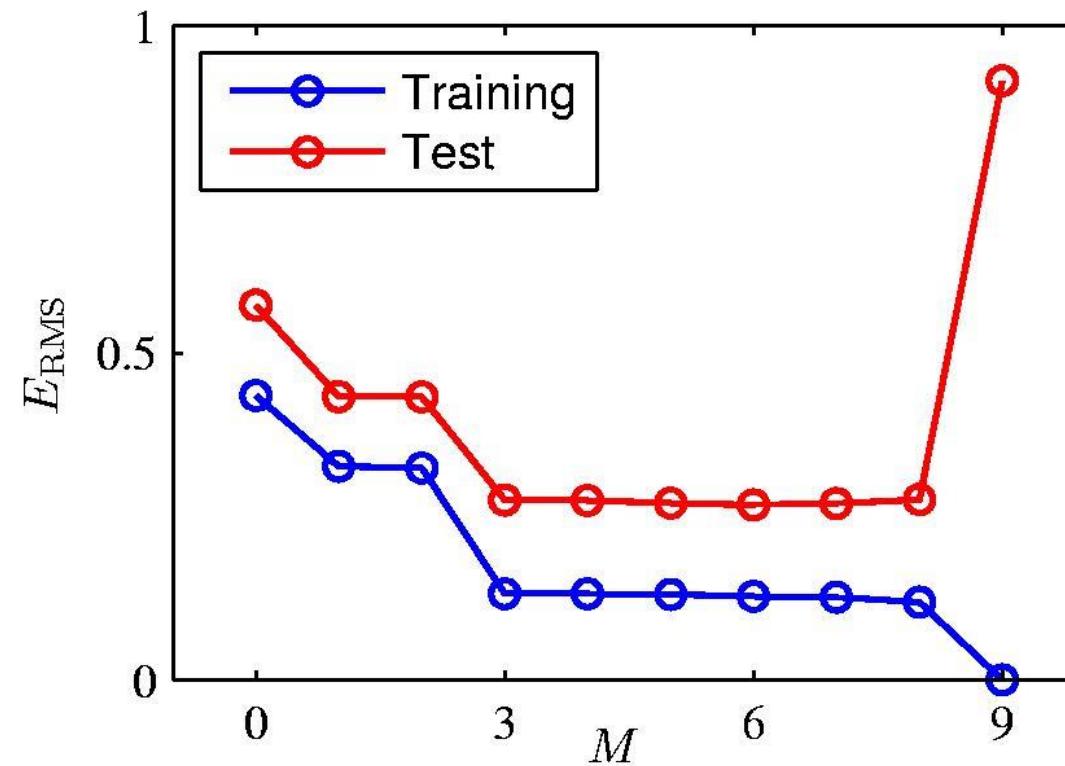
# Régularisation: $\mathcal{E}_{\text{RMS}}$ vs. $\ln(\lambda)$

$$\mathcal{E}_{\text{RMS}}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (F(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$



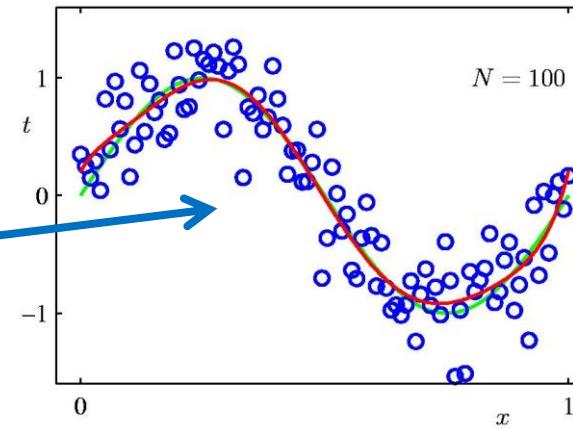
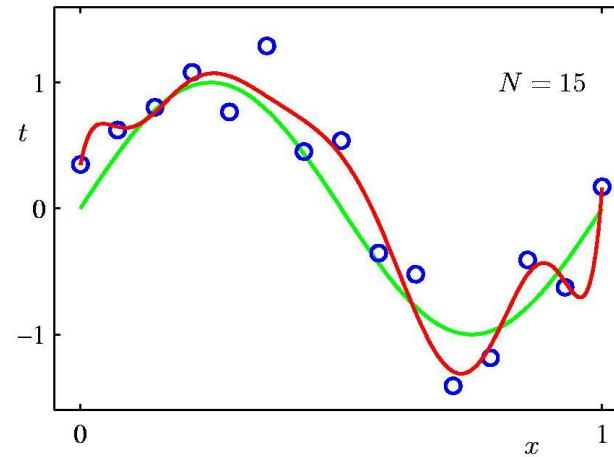
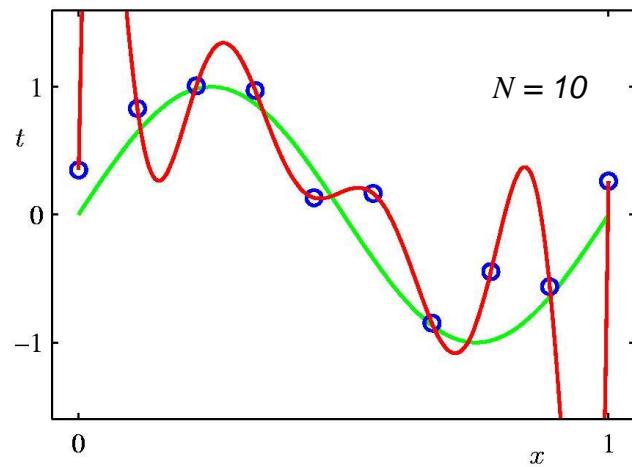
# Régularisation: $\mathcal{E}_{\text{RMS}}$ vs. $M$

$$\mathcal{E}_{\text{RMS}}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (F(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$



# Influence de la quantité de données

## Polynôme d'ordre 9



C'est aussi un moyen de contrôler la régression

# Trois critères à ne pas confondre

- Risque ou erreur empirique

$$\mathcal{E}_{\text{test}}(\mathbf{w}, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N \{ F(\mathbf{x}_i, \mathbf{w}) \neq y_i \}$$

- Erreur de généralisation (ou idéale...)

$$\mathcal{E}(\mathbf{w}) = E_{\mathbf{X}, Y} [\{ F(\mathbf{x}, \mathbf{w}) \neq y \}]$$

- Critère à optimiser (forme assez générique)

$$L(\mathbf{w}, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N l(F(\mathbf{x}_i, \mathbf{w}), y_i) + r(\mathbf{w})$$

Adéquation aux données

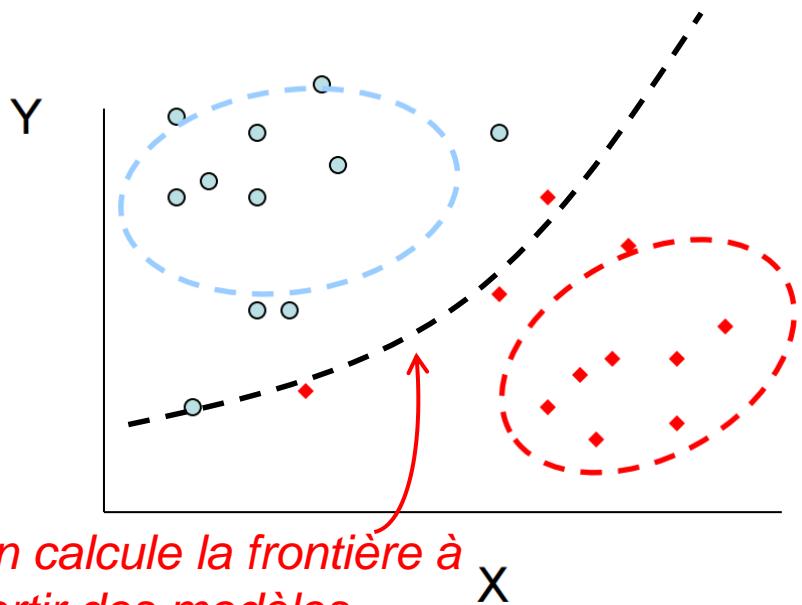
Régularisation

# « Support Vector Machines »

# Deux types d'approches: génératives vs. discriminatives

Objectif = modéliser les distributions de données puis les exploiter

Generative model

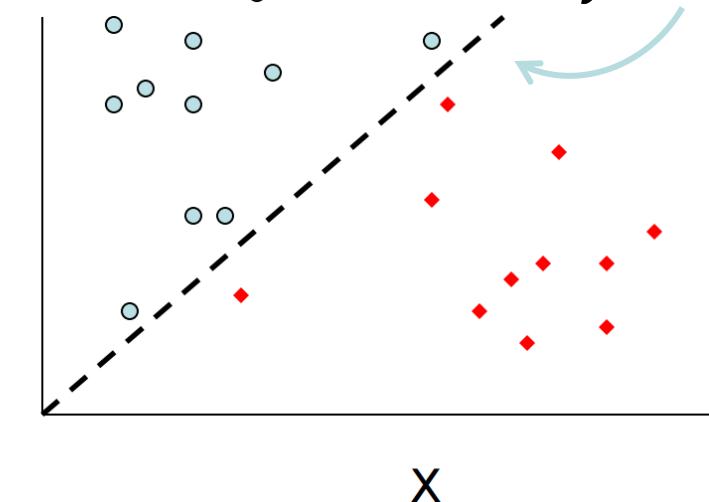


Le premier cours

*On estime directement*

Discriminative model

$$y = D(x)$$



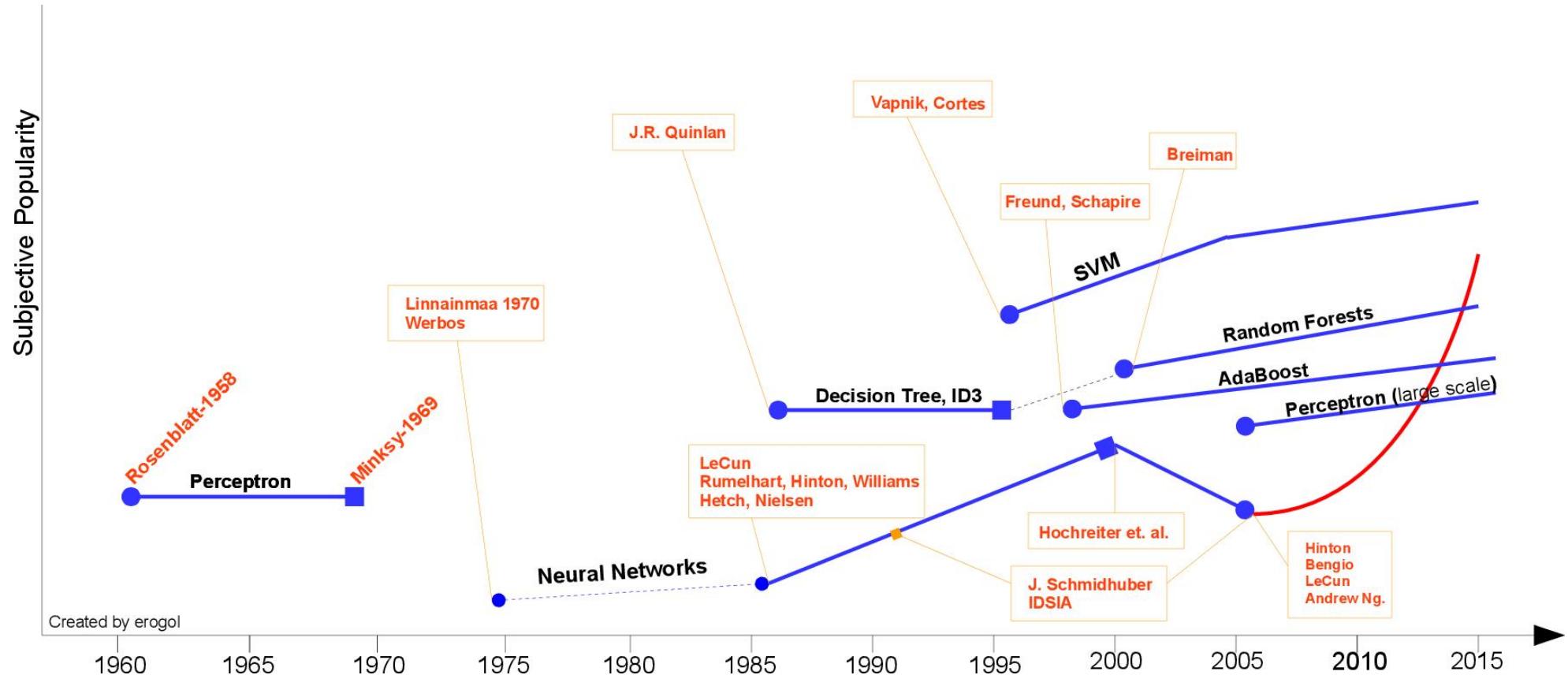
Objectif = construire les meilleures frontières

Aujourd'hui

# Support Vector Machines

- Historique
- Principe: maximiser la marge de séparation d'un hyperplan
- Le cas séparable
- Le cas non séparable: les fonctions de perte (« hinge loss »)
- L'extension au cas non linéaire: les noyaux
- Parcimonie
- Les paramètres de contrôle

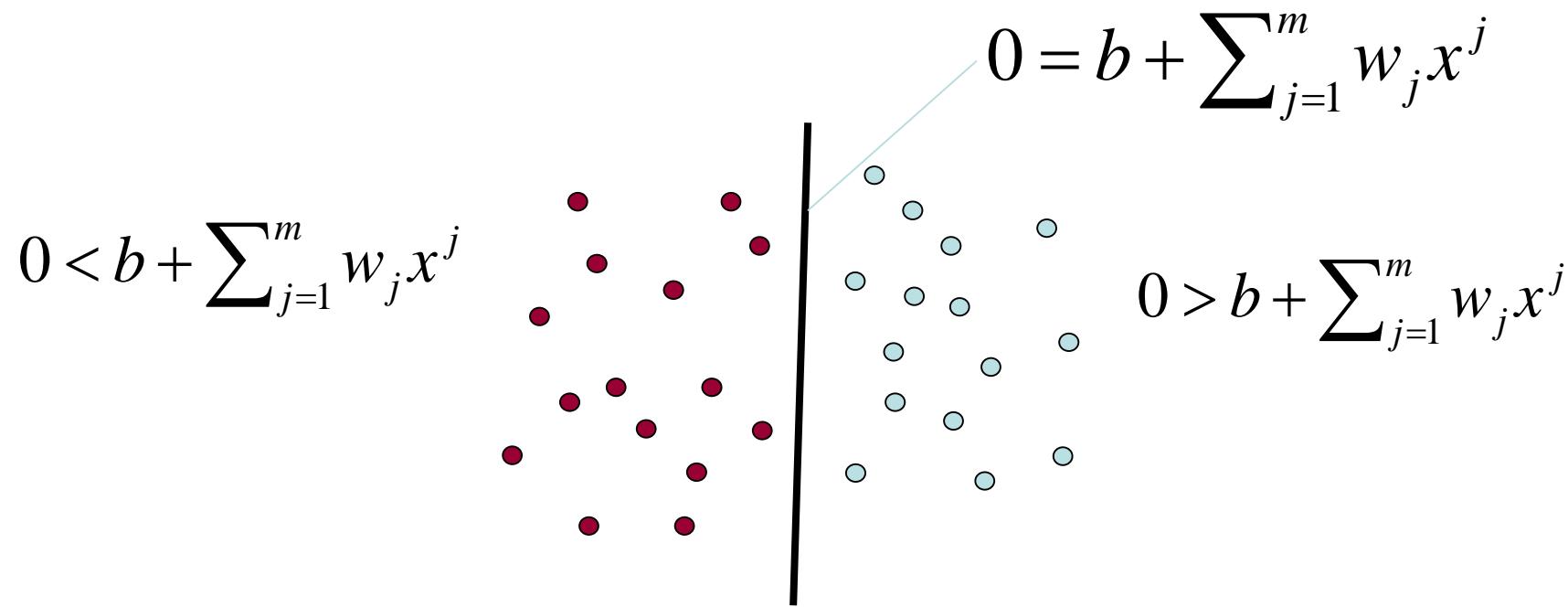
# Historique du Machine Learning



# Modèles linéaires de décision

Hypothèse = les données sont linéairement séparables.

- En 2D, par une droite
- En ND, par un hyperplan.



# Classifieur linéaire

- Equation de l'hyperplan séparateur

$$b + \mathbf{w} \cdot \mathbf{x} = 0$$

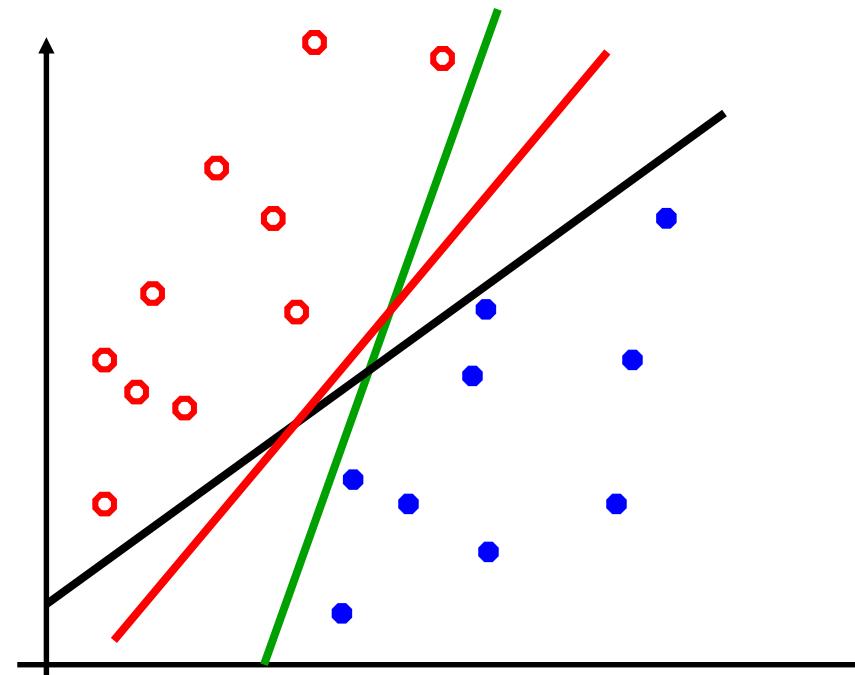
- Expression du classifieur linéaire (pour  $y_i$  valant -1 et 1)

$$F(\mathbf{x}; \mathbf{w}) = \text{sign}(b + \mathbf{w} \cdot \mathbf{x})$$

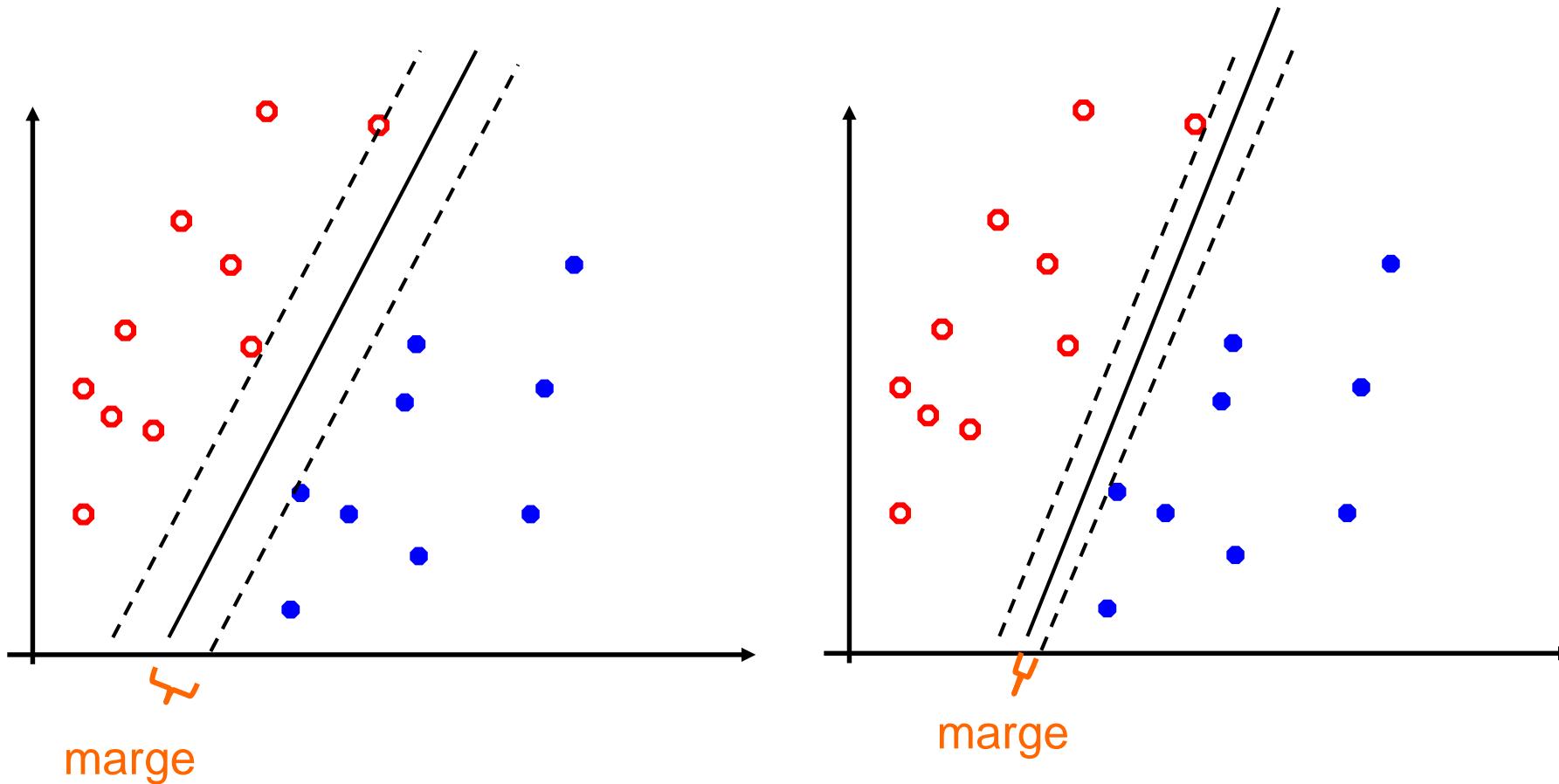
- Erreur

$$\mathcal{E}_{test}(\mathbf{w}, \mathcal{L}) = \frac{1}{N} \sum_{i=1}^N \left\{ y_i \cdot \text{sign}(b + \mathbf{w} \cdot \mathbf{x}_i) < 0 \right\}$$

# Quel hyperplan choisir?



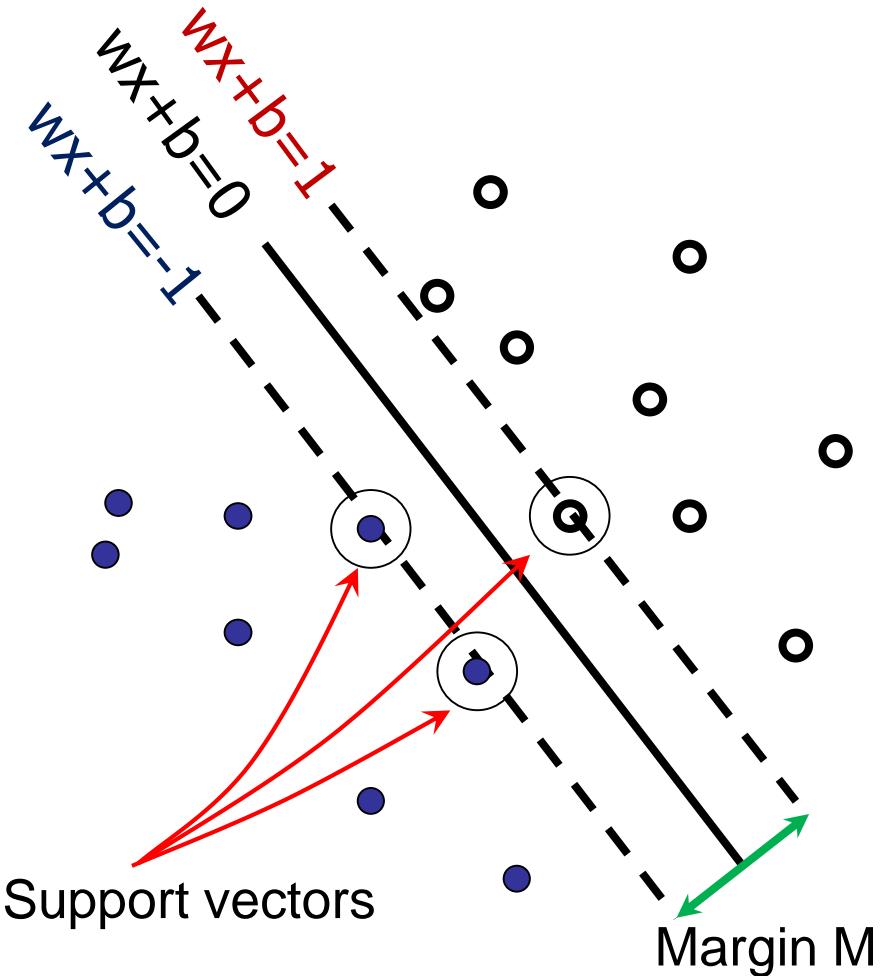
# Classifieur « Large margin »



Choisir l'hyperplan qui maximise la distance  
aux points les plus proches

# Support Vector Machines

- On cherche l'hyperplan qui maximise la marge.



$$\mathbf{x}_i \text{ positif } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ négatif } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

Pour les vecteurs de support,

Distance entre point et hyperplan:

$$\frac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$$

Pour les « support vectors »:

$$\frac{\mathbf{w}^T \mathbf{x} + b}{\|\mathbf{w}\|} = \frac{\pm 1}{\|\mathbf{w}\|} \quad M = \left| \frac{1}{\|\mathbf{w}\|} - \frac{-1}{\|\mathbf{w}\|} \right| = \frac{2}{\|\mathbf{w}\|}$$

# Principe du SVM (Large Margin)

- Maximiser la marge = distance des vecteurs à l'hyperplan séparateur des vecteurs de supports

$$\max \frac{1}{\|\mathbf{w}\|^2}$$

- Sous contraintes

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i$$

- Les vecteurs de support vérifiant:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$$

Le 1 est conventionnel.  
N'importe quelle  
constante  $>0$  est valable.

# Formulation du SVM

$$\min_{w,b} \|w\|^2$$

Tel que:

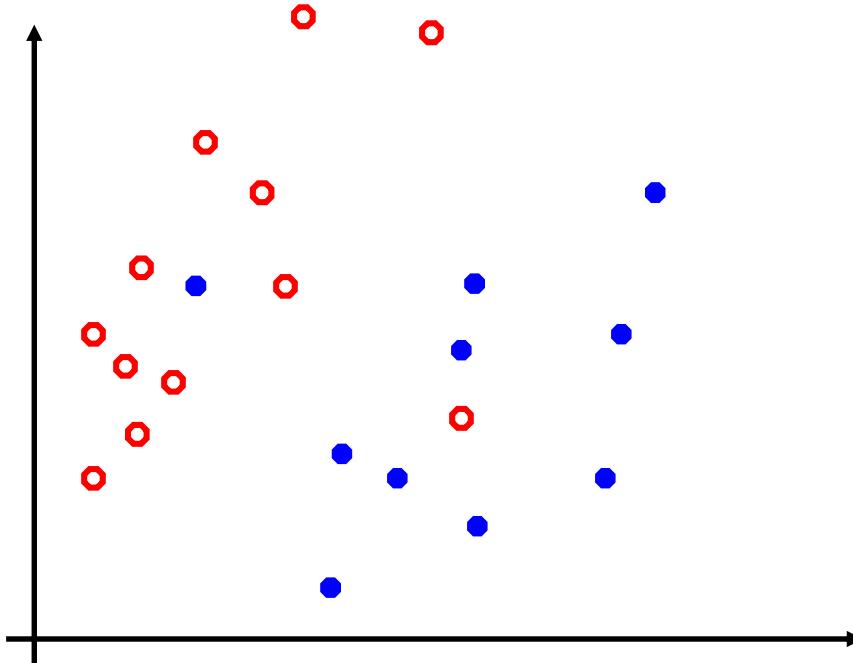
$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

Si les données sont séparables  
Problème d'optimisation quadratique  
Avec contraintes linéaires

## Problème d'optimisation quadratique classique

Mais avec beaucoup de contraintes! (autant que d'exemples d'apprentissage)

# Classification « Soft Margin »



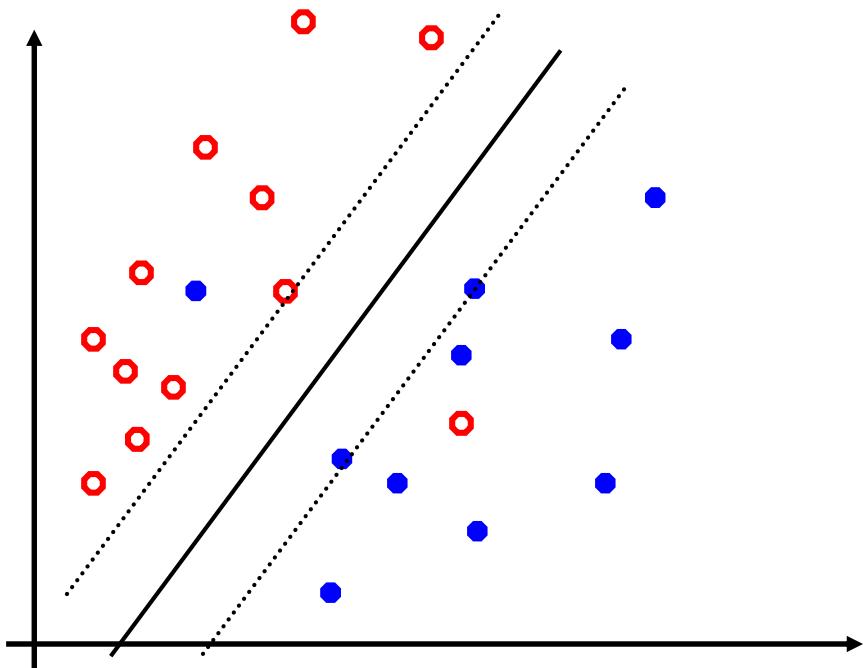
$$\min_{w,b} \|w\|^2$$

Tel que:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

Comment traiter le cas non linéairement séparable?

# Classification « Soft Margin »



$$\min_{w,b} \|w\|^2$$

Tel que:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

On aimeraient obtenir une séparation robuste à quelques données non séparées

# Idée: « Slack variables »

$$\min_{w,b} \|w\|^2$$

tq:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$



$$\min_{w,b} \|w\|^2 + C \sum_i \xi_i$$

tq:

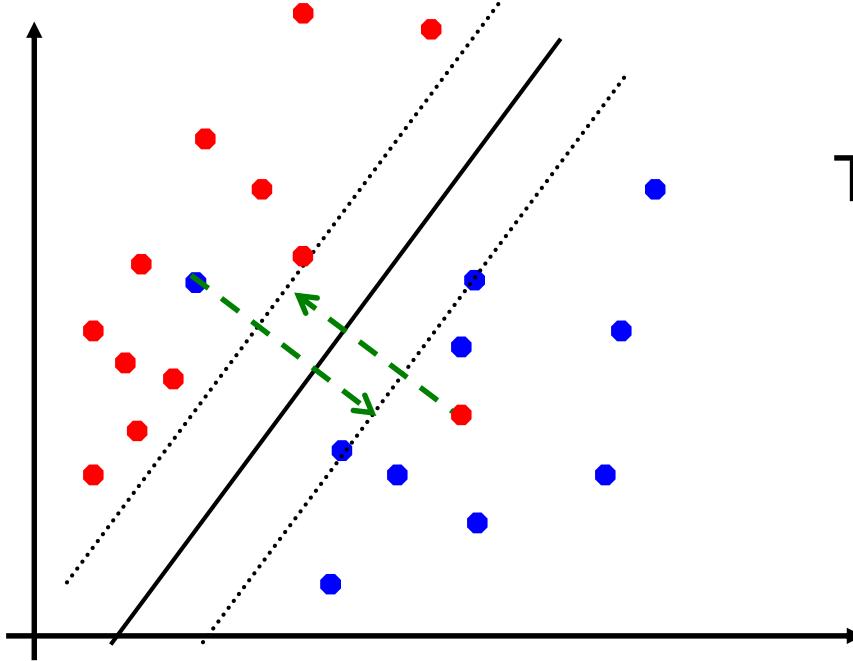
$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \forall i$$

$$\xi_i \geq 0$$

Permet de relâcher la contrainte de séparabilité pour chaque exemple.

slack variables  
(une par exemple)

# « Slack variables »



$$\min_{w,b} \|w\|^2 + C \sum_i \xi_i$$

Tel que:

$$y_i(w \cdot x_i + b) + \xi_i \geq 1 \quad \forall i$$

$$\xi_i \geq 0$$

Relâchement de la contrainte

# Utilisation des « Slack variables »

marge

tq

$$\min_{w,b} \|w\|^2 + C \sum_i \xi_i$$

Compromis entre marge et pénalisation de la contrainte

Valeur du relâchement de la contrainte

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \forall i$$

$\xi_i \geq 0$

Contrainte autorisée à être relachée

# Soft margin SVM

$$\min_{w,b} \|w\|^2 + C \sum_i \xi_i$$

Tel que

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \forall i$$
$$\xi_i \geq 0$$

**On garde un problème quadratique!**

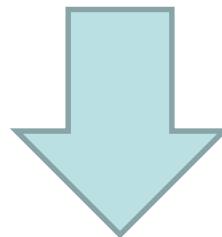
**Mais avec un très grand nombre de variables+contraintes**

# Autre formulation

$$\min_{w,b} \|w\|^2 + C \sum_i \xi_i$$

tq:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \forall i$$
$$\xi_i \geq 0$$



$$\min_{w,b} \|w\|^2 + C \sum_i \max(0, 1 - y_i(w \cdot x_i + b))$$

**Problème d'optimisation non contraint**  
→ Autres méthodes d'optimisation (descente de gradient)

# Interprétation du « Soft Margin SVM »

$$\min_{w,b} \|w\|^2 + C \sum_i \max(0, 1 - y_i(w \cdot x_i + b))$$

On retrouve la formulation:

$$\text{Loss}(\mathbf{w}, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N l(F(\mathbf{x}_i, \mathbf{w}), y_i) + r(\mathbf{w})$$

Avec

$$r(\mathbf{w}) = \frac{1}{C} \|\mathbf{w}\|^2$$

$$l(F(\mathbf{x}_i, \mathbf{w}), y_i) = \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b))$$

**Le SVM est un cas particulier du formalisme:  
« erreur empirique + régularisation »**

# Autres Fonctions de coût

0/1 loss:

$$l(y, y') = \mathbf{1}[yy' \leq 0]$$

Hinge:

$$l(y, y') = \max(0, 1 - yy')$$

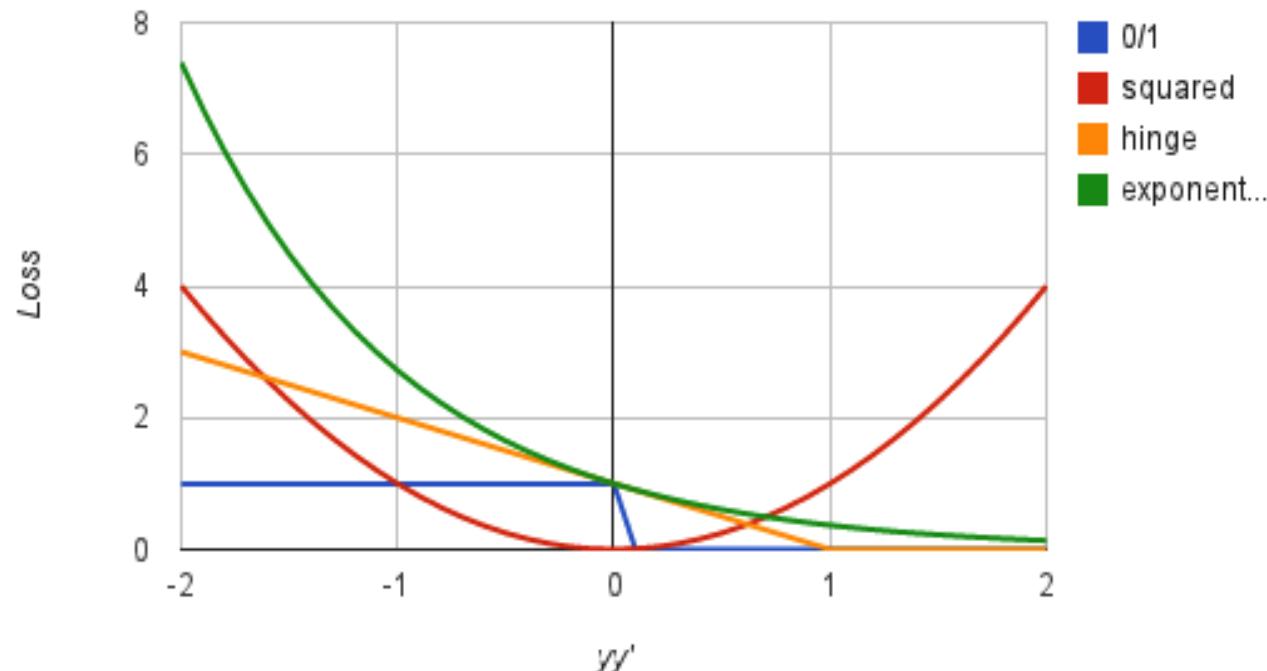
Squared loss:

$$l(y, y') = (y - y')^2$$

Exponential:

$$l(y, y') = \exp(-yy')$$

## Surrogate loss functions



# Forme duale du SVM

- Problème d'optimisation sous contrainte

*Pour simplifier l'expression des calculs*

Primal

$$\operatorname{argmin}_{\mathbf{w}} \frac{\|\mathbf{w}\|^2}{2} + C \sum_i \xi_i \quad \text{Multiplicateurs de Lagrange}$$

$$s.t. \forall i, y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \quad \alpha_i$$

$$\xi_i \geq 0 \quad \beta_i$$

Dual (Lagrangien)

$$L(\mathbf{w}, \xi, \alpha, \beta) = \frac{\|\mathbf{w}\|^2}{2} + \sum_i (C\xi_i - \alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i) - \beta_i\xi_i)$$

$$s.t. \forall i, \alpha_i \geq 0, \beta_i \geq 0$$

# Forme duale du SVM

- Lagrangien

$$L(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

Maximisation dans le dual!

$$s.t. \forall i, 0 \leq \alpha_i \leq C$$

On garde un pb. quadratique

Dual des contraintes « slack »

Solution optimale (conditions de Kuhn-Tucker):  $\alpha_i(y_i w^T x_i - 1 + \xi_i) = 0$

Interprétation:  $\alpha_i = 0$  si la contrainte est satisfaite (bonne classification)

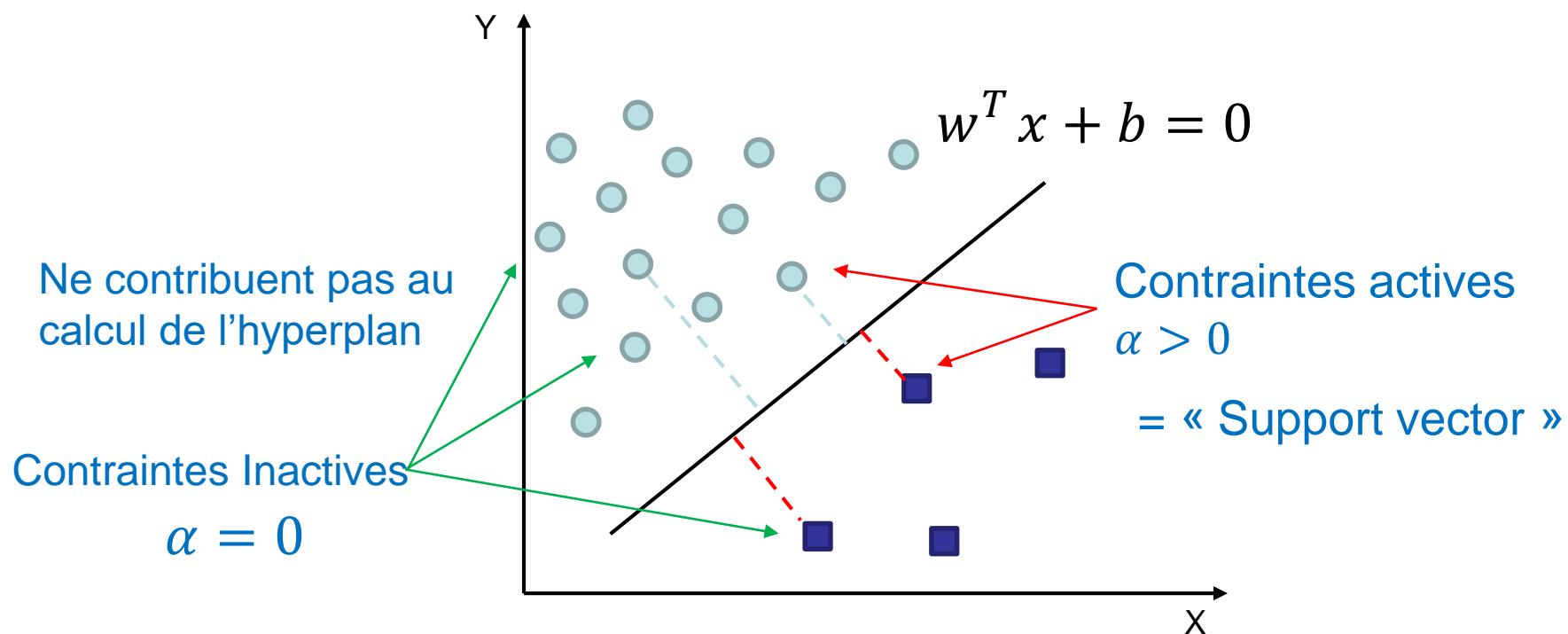
$\alpha_i > 0$  si la contrainte n'est pas satisfaite (mauvaise classification)

# Parcimonie du SVM

- Seuls certains  $\alpha$  sont non nuls = autre manière de définir les vecteurs de support.

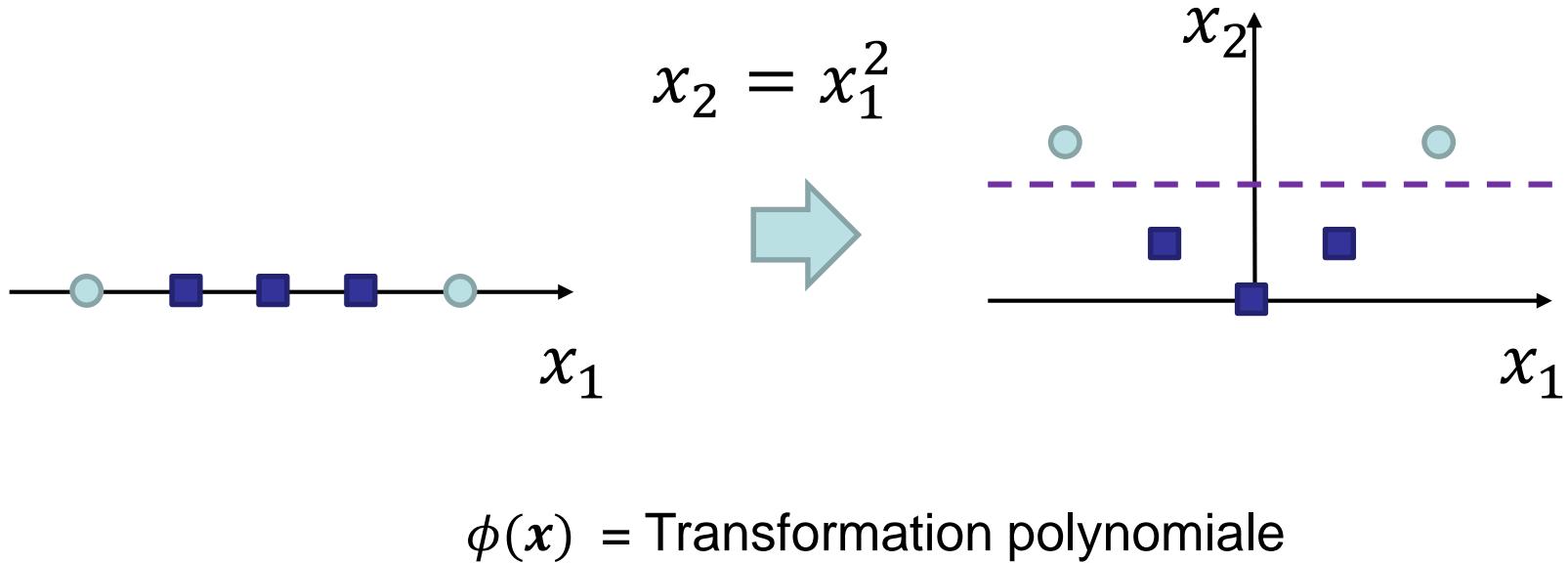
$$\text{Optimalité} = \alpha_i(y_i w^T x_i - 1 + \xi_i) = 0$$

Direction de l'hyperplan séparateur  $w = \sum_i \alpha_i y_i x_i$



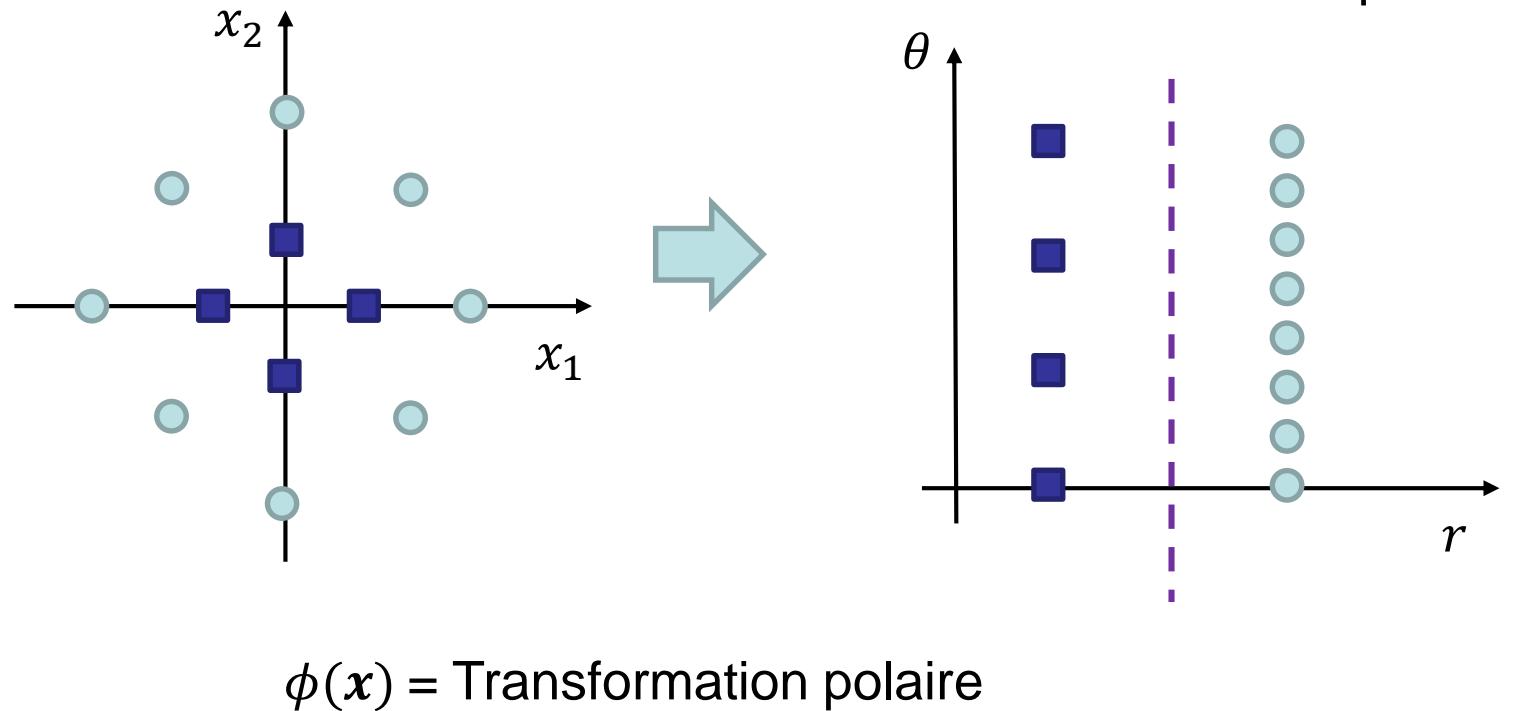
# Données non linéairement séparables

- Transformation non linéaire  $\phi(x)$  pour séparer linéairement les données d'origine



# Données non linéairement séparables

- Transformation non linéaire  $\phi(x)$  pour séparer linéairement les données d'origine



# Retour sur la formulation duale du SVM

Lagrangien

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \boxed{\mathbf{x}_i \mathbf{x}_j}$$

tq  $\forall i, 0 \leq \alpha_i \leq C$

Produit scalaire  
uniquement

## « Kernel trick »

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

tq  $\forall i, 0 \leq \alpha_i \leq C$

**Noyau**

Le noyau  $K$  est un produit scalaire dans l'espace transformé:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Il est uniquement nécessaire de connaître la similarité entre données pour introduire la non linéarité dans le problème (avec des conditions...)

# Utilisation de noyaux dans les SVM

- Permet d'introduire des mesures de similarités propres au domaine étudié et sans avoir à gérer la complexité de la transformation
- Permet de séparer modélisation = noyau de la classification et SVM (optimisation)
- Définit la fonction de classification à partir de noyaux « centrés » sur les vecteurs de support

$$F(\mathbf{x}, \mathbf{w}) = b + \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$$

# Noyaux courants

- Polynômes de degrés supérieurs à  $d$

$$K(x, y) = (x \cdot y + 1)^d$$

- Noyau gaussien

$$K(x, y) = \exp\left(-\frac{(x - y)^T(x - y)}{2\sigma^2}\right)$$

Paramètres à définir  
= degré de liberté supplémentaire

- Intersection d'histogrammes

$$K(x, y) = \sum_i \min(x^i, y^i)$$

# Résumé sur SVM

- Une formulation optimale quadratique du problème de classification binaire:
  - Primal: optimisation d'un critère empirique + régularisation
  - Dual: permet d'introduire parcimonie et « kernel trick »➔ plusieurs manières d'optimiser
- Les solutions s'expriment comme des combinaisons linéaires éparses de noyaux:

$$F(\mathbf{x}) = \text{sign}(b + \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}))$$

où  $\alpha_i > 0$  seulement pour les vecteurs de support, 0 sinon.

- En pratique, ce qu'il faut régler:
  - Le coefficient de régularisation: C
  - Le type de noyau et ses caractéristiques
  - Les paramètres de l'optimiseur

# Multiclasse

# Différents types de classification

- Binaire

$$\mathcal{A} = \{-1,1\}$$

- Multi classe

$$\mathcal{A} = \{1,2\dots L\}$$

- Détection (quoi et où)

$$\mathcal{A} = \{1,2\dots L\} \times R^4$$

- Caractérisation des données:

- Rejet
  - Anomalie

$$\mathcal{A} = \{1,2\dots L, \text{ambigu}, \text{inconnu}\}$$

# Hypothèses multiples

- Toutes les classes/hypothèses ne se valent pas
  - Classes plus rares que d'autres (non équilibrées)
  - Coût d'une erreur de classification dépend des classes (Zèbre vs. Gazelle vs. Lion)
- Deux stratégies:
  - Optimiser un critère multi-hypothèse dans l'apprentissage
    - Par exemple entropie dans arbre de décision, softmax dans réseaux de neurones...
  - Utiliser un ensemble de classifieurs binaires
    - SVM, adaboost, perceptron...

# Multiclasse à partir de classifieurs

- Comment passer d'une classification binaire à  $N$  classes?
- Plusieurs techniques:
  - One vs Rest
  - One vs One (ou All vs All)
- OVO:
  - On apprend autant de classifieurs que de **paires de classes** ( $N(N-1)/2$ )
  - Classification = choix de la classe ayant le plus de **votes**
  - **Pb:** peut être indécidable dans certains cas
- OVR:
  - On apprend **un classifieur par classe**
  - Classification = choix de la classe ayant **le meilleur score**
  - **Pb:** déséquilibre des données entre classe cible et « reste »

# Evaluation du multi-classe

- Erreur globale:

$$Err = \frac{\text{nombre d'échantillons mal classés}}{\text{nombre d'échantillons testés}}$$

- Matrice de confusion:

$\text{conf}(i, j)$ =probabilité de classer comme i | vraie classe est j  
estimée sur données de test

- Risque ou coût moyen

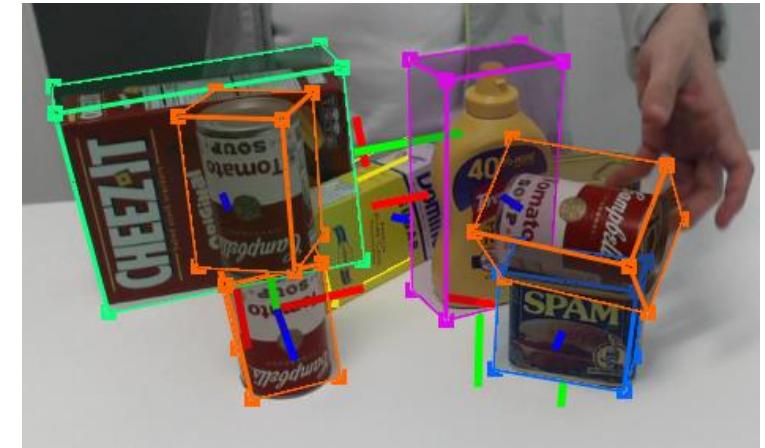
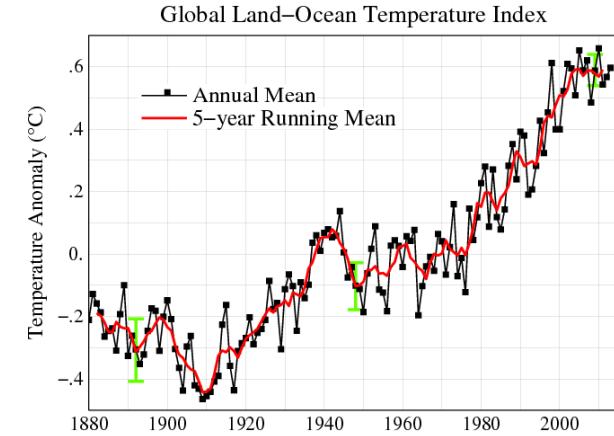
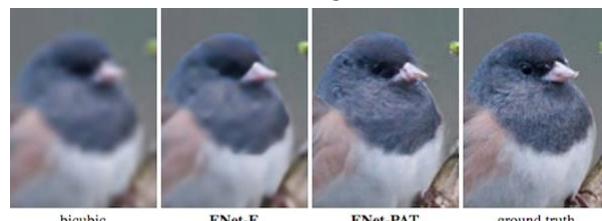
$$R = \sum_j \sum_i \lambda(i, j) \text{conf}(i, j) p(j)$$

où  $\lambda(i, j)$  est le coût de décider  $i$  lorsque  $j$  est vrai

# Régression

# A quoi ça sert

- **Prédire** une valeur (interpolation, extrapolation)
- Expliquer/déetecter/repérer des corrélations/tendances
- Exemples de prédiction:
  - Prédiction de régime moteur
  - Prédiction de durée de survie
  - Estimation de prix
  - Prévision météo ou climatique
  - Prévision de cours de la bourse
  - Super résolution
  - Estimation de pose d'objet



# Formulation de la régression

- Prédicteur:  $W, x \mapsto y$
- Modèle :  $P(y | x, W)$  ou  $y = f(x, W) + \varepsilon$

Où

- entrée  $x \in \mathbb{R}^M$ , prédiction  $y \in \mathbb{R}^p$
  - $W$  paramètres du prédicteur estimé à partir d'un ensemble d'apprentissage:  $\mathcal{L} = \{x_i, y_i\}_{i=1..N}$
  - $\varepsilon$  variable aléatoire décrivant l'erreur de prédiction (souvent considérée gaussienne)
- 
- C'est du « Machine Learning » (Apprentissage supervisé)
  - C'est du « Data Mining » (Estimation de dépendances entre variables)

# Modèles linéaires

- La base de la régression
  - Fondements mathématiques solides
  - Calculs analytiques ou optimisation séquentielle
- Plusieurs manières de dépasser la linéarité
  - Modèles linéaires généralisés
  - Modèles à base de noyaux (« kernels »)
- Plusieurs manières de maîtriser la complexité
  - « Large margin » (cf. SVM)
  - Régularisateurs

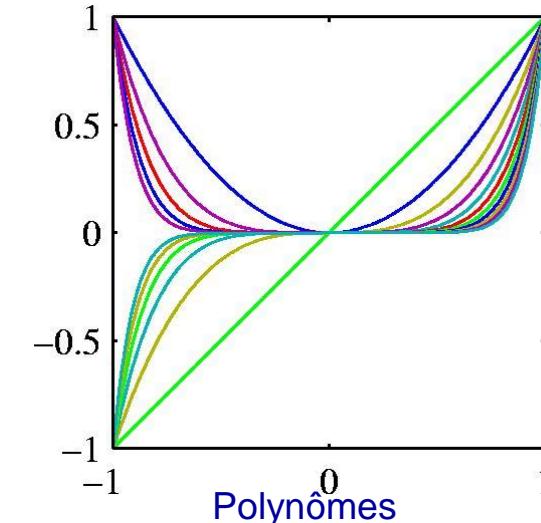
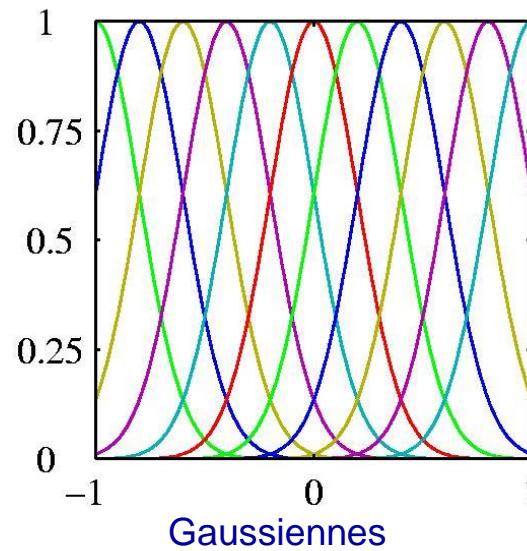
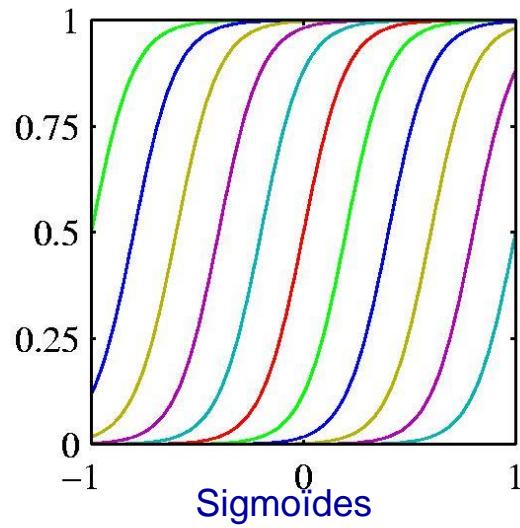
# Modèles linéaires généralisés (cas scalaire $y \in \mathbb{R}$ )

- Principe simple: utiliser plusieurs **fonctions de base**  $\phi$  encodant les données source (« features »)

$$f(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \dots = \mathbf{w}^T \mathbf{x}$$
$$f(\mathbf{x}, \mathbf{w}) = w_0 + w_1 \phi_1(\mathbf{x}) + w_2 \phi_2(\mathbf{x}) + \dots = \mathbf{w}^T \Phi(\mathbf{x})$$

- Une fois définies les fonctions, le problème reste **linéaire!**
- Comment trouver ces fonctions?
  - Se les donner
  - Les apprendre

# Exemples classiques de fonctions de base 1D



Remarque: les sigmoïdes et gaussiennes sont des fonctions d'activation usuelles dans les réseaux de neurones

→ les RN permettent d'apprendre les  $\phi$

# Apprentissage = trouver $W_{ML}$

- Forme du prédicteur

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

- Critère d'erreur (évaluation)

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

- Principe statistique: maximum de vraisemblance

$$\mathbf{W}_{ML} = \underset{\mathbf{W}}{\operatorname{argmax}} P(y | \mathbf{x}, \mathbf{W})$$

# Solution générale ( $y \in \mathbb{R}^P$ )

Lorsque les cibles sont vectorielles

$$\mathbf{W}_{\text{ML}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{T}.$$

où  $\mathbf{T} = [\mathbf{t}_1, \dots, \mathbf{t}_N]^T$  est la matrice des cibles ( $N \times p$ )  $\mathbf{t}_k = [t_{1k}, \dots, t_{Nk}]^T$

Pour une cible scalaire on a

$$\mathbf{w}_k = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}_k = \Phi^\dagger \mathbf{t}_k$$

où  $\Phi^\dagger$  ne dépend que des données d'entrée.

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}.$$

# Moindre carrés régularisés (retour)

Idée: on rajoute une pénalisation des grandes valeurs des paramètres à la fonction de coût:

$$L(\mathbf{W}) = \sum_{i=1}^N (f(\mathbf{x}_i, \mathbf{W}) - y_i)^2 + \lambda \|\mathbf{W}\|^2$$

Coût d'attaché aux données

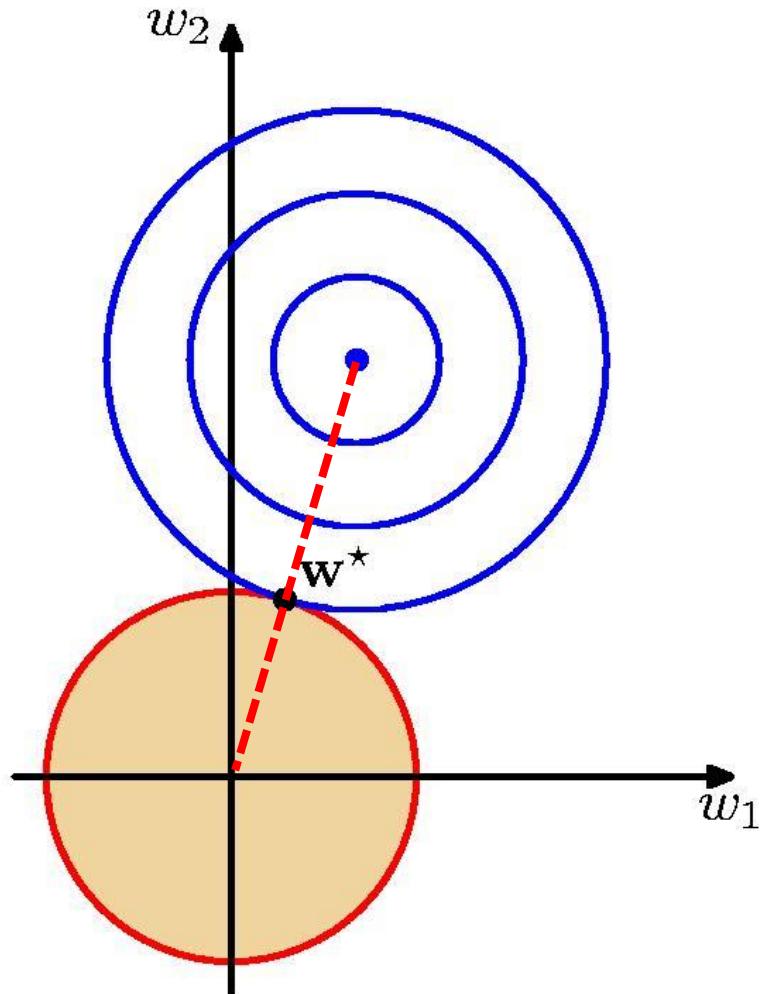
Paramètre de régularisation

Dont l'optimum exact est alors:

$$\mathbf{W}^* = (\Phi^t \Phi + \lambda I)^{-1} \Phi^t Y$$

Si on pénalise les grandes valeurs des coefficients du polynôme, on obtient une fonction moins « zigzagante »

# Comment fonctionne la régularisation L2

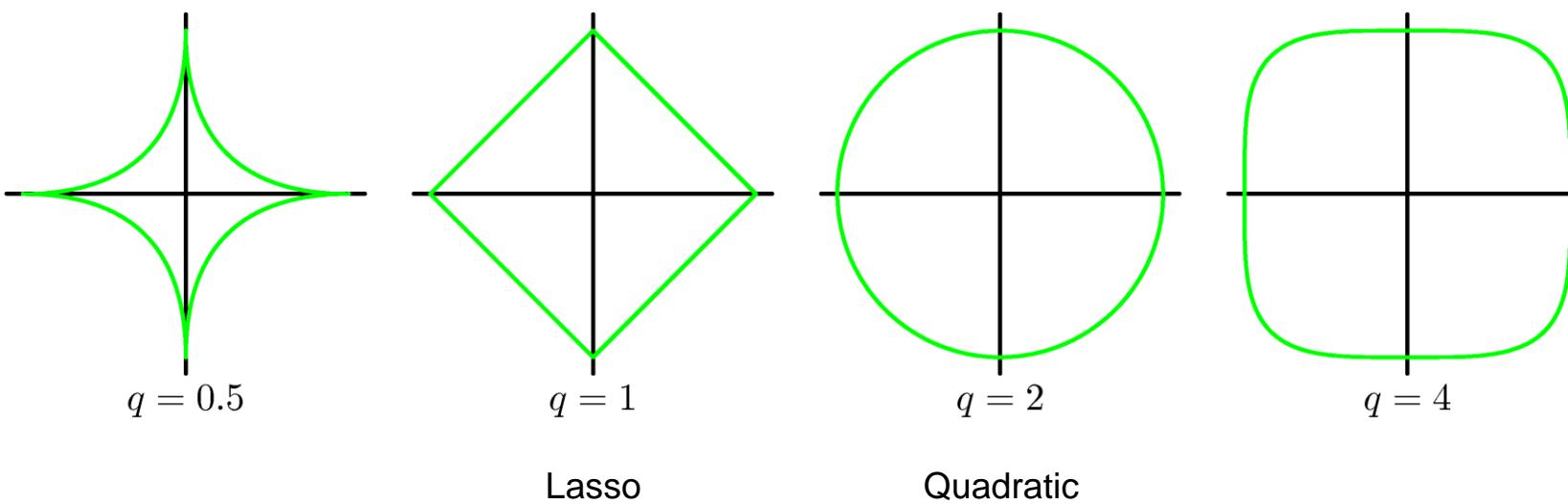


- Le coût global est la somme de deux « cuvettes ».
- La somme est aussi quadratique.
- Le minimum global est sur une ligne joignant l'origine et le minimum sans contrainte.
- La régularisation a pour effet de diminuer les poids.

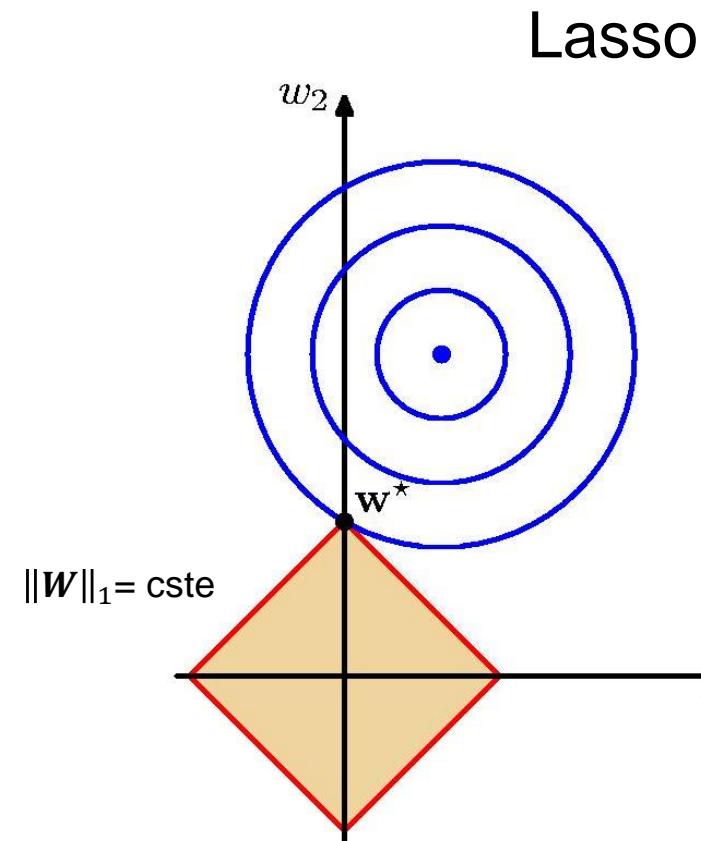
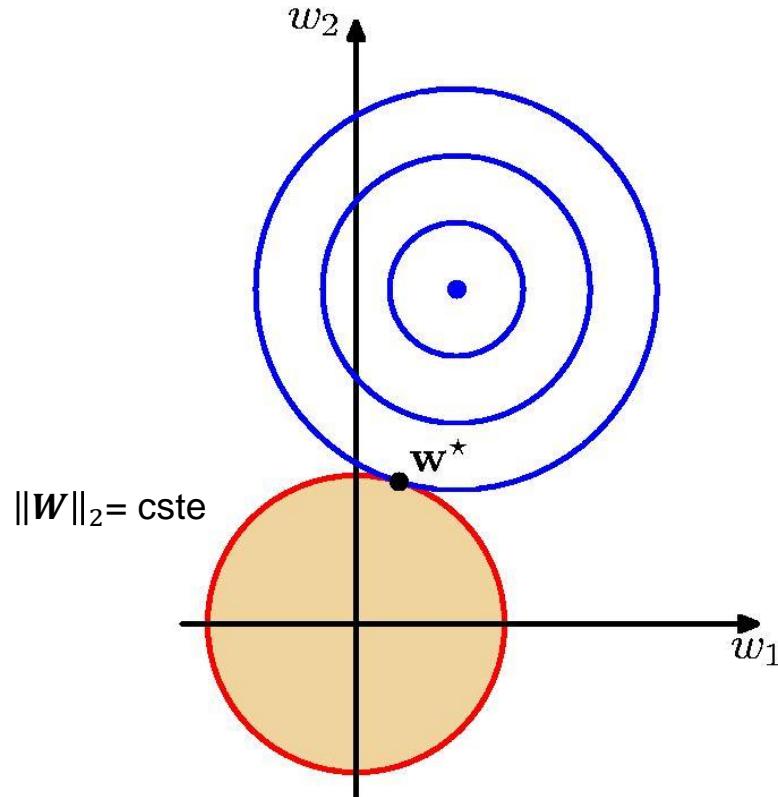
# Autres types de régularisation

Une manière simple: utiliser une autre norme

$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$



# Interprétation géométrique



Remarque:  $w_1=0$  à l'optimum.  
**Parcimonie** de la solution.

# D'autres pénalisations...

- **The Tikhonov regularization:**  $\psi(\mathbf{w}) = ^\Delta 1/2 \|\mathbf{w}\|_2^2$ .
- **The  $\ell_1$ -norm:**  $\psi(\mathbf{w}) = ^\Delta \|\mathbf{w}\|_1$ .
- **The Elastic-Net:**  $\psi(\mathbf{w}) = ^\Delta \|\mathbf{w}\|_1 + \gamma \|\mathbf{w}\|_2^2$ .
- **The Fused-Lasso:**  $\psi(\mathbf{w}) = ^\Delta \|\mathbf{w}\|_1 + \gamma \|\mathbf{w}\|_2^2 + \gamma_2 \sum_{i=1}^{p-1} |\mathbf{w}_{i+1} - \mathbf{w}_i|$ .
- **The group Lasso:**  $\psi(\mathbf{w}) = ^\Delta \sum_{g \in \textcolor{red}{G}} \eta_g \|\mathbf{w}_g\|_2$ , where  $\textcolor{red}{G}$  are groups of variables.
- **The group Lasso with  $\ell_\infty$ -norm:**  $\psi(\mathbf{w}) = ^\Delta \sum_{g \in \textcolor{red}{G}} \eta_g \|\mathbf{w}_g\|_\infty$ , where  $\textcolor{red}{G}$  are groups of variables.
- **The sparse group Lasso:** same as above but with an additional  $\ell_1$  term.
- **The tree-structured sum of  $\ell_2$ -norms:**  $\psi(\mathbf{w}) = ^\Delta \sum_{g \in \textcolor{red}{G}} \eta_g \|\mathbf{w}_g\|_2$ , where  $\textcolor{red}{G}$  is a tree-structured set of groups [15], and the  $\eta_g$  are positive weights.
- **The tree-structured sum of  $\ell_\infty$ -norms:**  $\psi(\mathbf{w}) = ^\Delta \sum_{g \in \textcolor{red}{G}} \eta_g \|\mathbf{w}_g\|_\infty$ . See [15]
- **General sum of  $\ell_\infty$ -norms:**  $\psi(\mathbf{w}) = ^\Delta \sum_{g \in \textcolor{red}{G}} \eta_g \|\mathbf{w}_g\|_\infty$ , where no assumption are made on the groups  $\textcolor{red}{G}$ .
- **The path-coding penalties of [24].**
- **the  $\ell_1$ -constraint.**

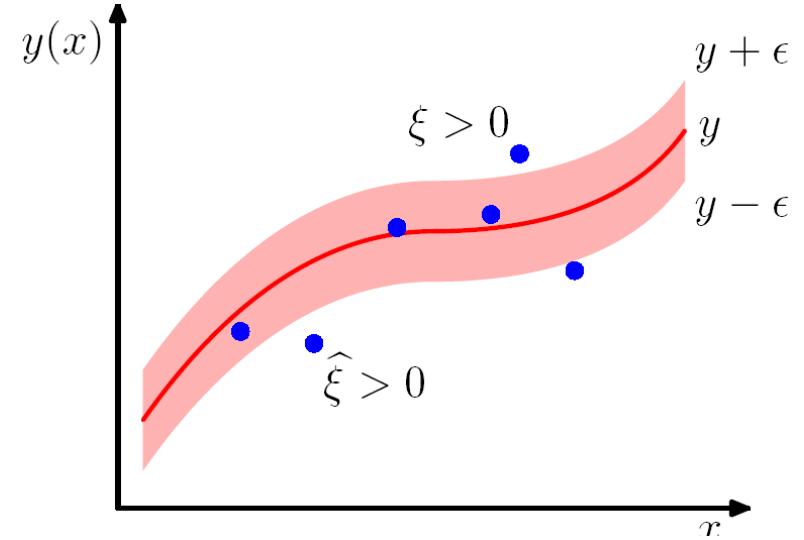
<http://spams-devel.gforge.inria.fr/documentation.html>

# Support Vector Regression

$$\text{Min } \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n (\xi_i + \xi_i^*)$$

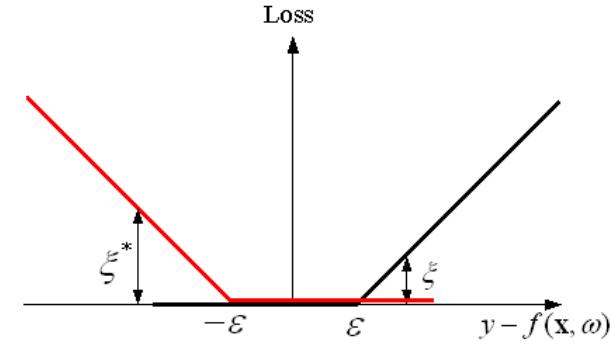
subject to

$$\begin{cases} u_i - \mathbf{w}^T \mathbf{x}_i - b \leq \epsilon + \xi_i \\ \mathbf{w}^T \mathbf{x}_i + b - u_i \leq \epsilon + \xi_i^* \\ \xi_i \geq 0, \xi_i^* \geq 0 \end{cases}$$



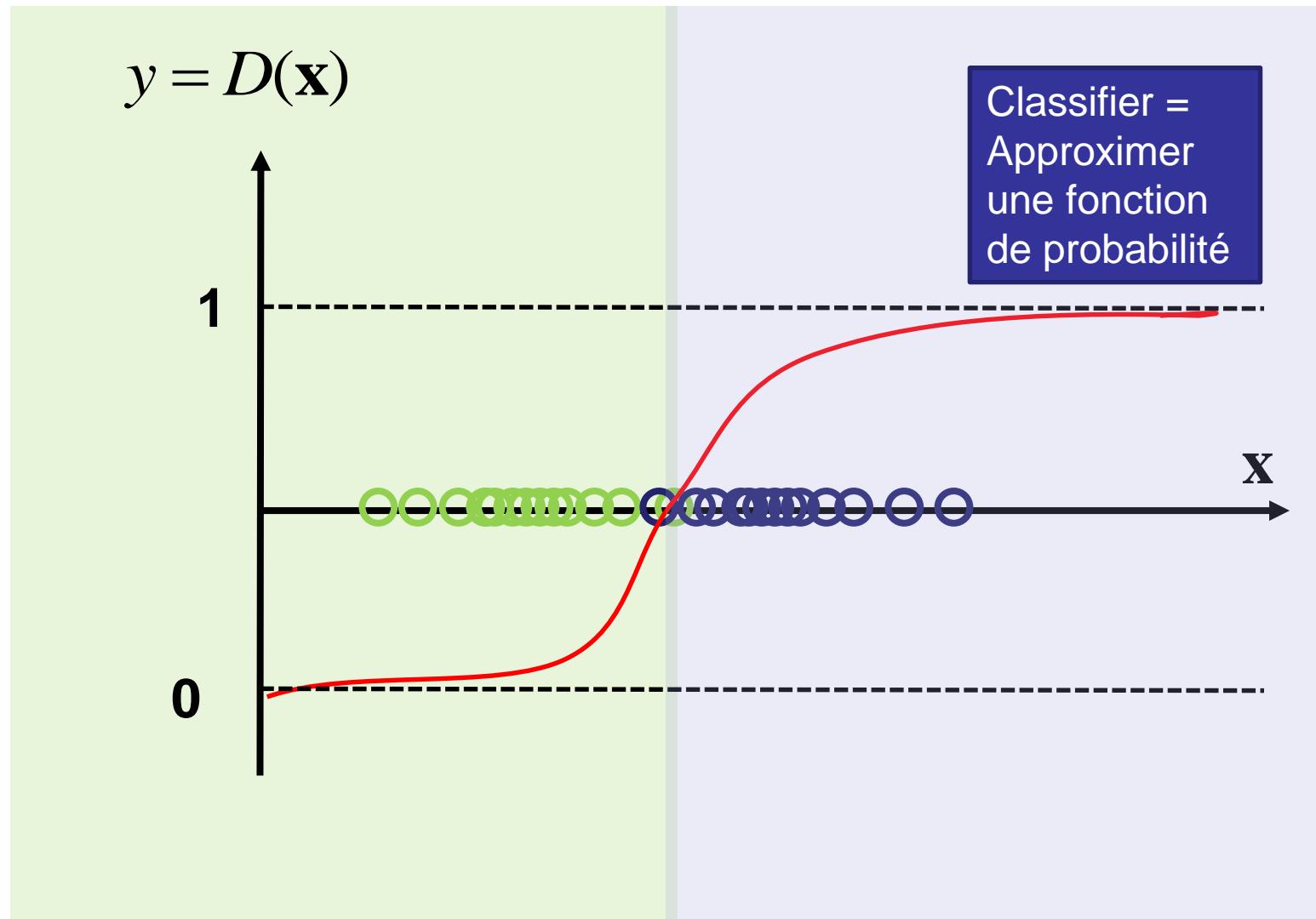
Formulation comparable à celle de la classification

- Fonction de coût différente (dépend d'un paramètre)
- Expression du « soft margin » symétrique
- « Kernel trick » applicable
- Sparsité de la solution



$$L_\epsilon(y, f(\mathbf{x}, \omega)) = \max(|y - f(\mathbf{x}, \omega)| - \epsilon, 0)$$

# Classification et Régression: : même combat



# Problème liés à la régularisation globale

- La pénalisation est isotrope: toutes les dimensions sont considérées simultanément, avec le même poids
  - ➔ on fait l'hypothèse que les dimensions sont comparables (en unité et signification)
  - ➔ Besoin de normaliser ou « blanchir » les données. Mais risque alors de louper les fortes corrélations entre données.
- Beaucoup de types de régularisation (et d'algorithmes d'optimisation...): comment choisir?
  - Validation croisée
  - Structure du problème (on veut forcer certaines dimensions ou certaines corrélations entre dimensions)
  - On veut obtenir une solution interprétable ➔ sparsité. Mais il y a d'autres approches algorithmiques pour la rechercher directement .
- Remarque: la recherche sur les algorithmes « sparse » était très active avant le « deep learning era ». Maintenant moins...

# Evaluer une régression

- Deux objectifs: prédiction ou modélisation
- Prédiction: on cherche la fonction ayant l'erreur de généralisation la plus faible
  - ➔ Erreur estimée sur base de test
  - ➔ Recherche des paramètres sur base de validation (validation croisée)
- Modélisation: on cherche la fonction qui explique au mieux les corrélations entre données
  - ➔ Tests statistiques ( $R^2$  et p-values)

# Formulation bayésienne

- Principes
  - Considérer la distribution jointe des sorties  $t$  conditionnellement aux entrées  $x$  comme gaussiennes
  - considérer les poids  $W$  comme des variables aléatoires
  - ➔ on évolue dans des espaces de distributions
- Loi a priori  $P(W)$
- Vraisemblance des entrées  $P(t | x, W)$
- Loi a posteriori  $P(W|t)$
- Plus hyper paramètres de modélisation des lois (en général prises gaussiennes)

Gaussienne

$$p(\mathbf{t} | \mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n | \mathbf{w}^T \mathbf{x}_n, \beta^{-1})$$

variance du  
bruit de sortie

vraisemblance

$$p(\mathbf{w} | \alpha) = \mathcal{N}(\mathbf{w} | 0, \alpha^{-1} \mathbf{I})$$

prior

$$-\ln p(\mathbf{w} | \mathbf{t}) = \frac{\beta}{2} \sum_{n=1}^N (t_n - \mathbf{w}^T \mathbf{x}_n)^2 + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} + const$$

Inverse  
de la  
variance  
du prior

On retrouve une formulation  
avec régularisation

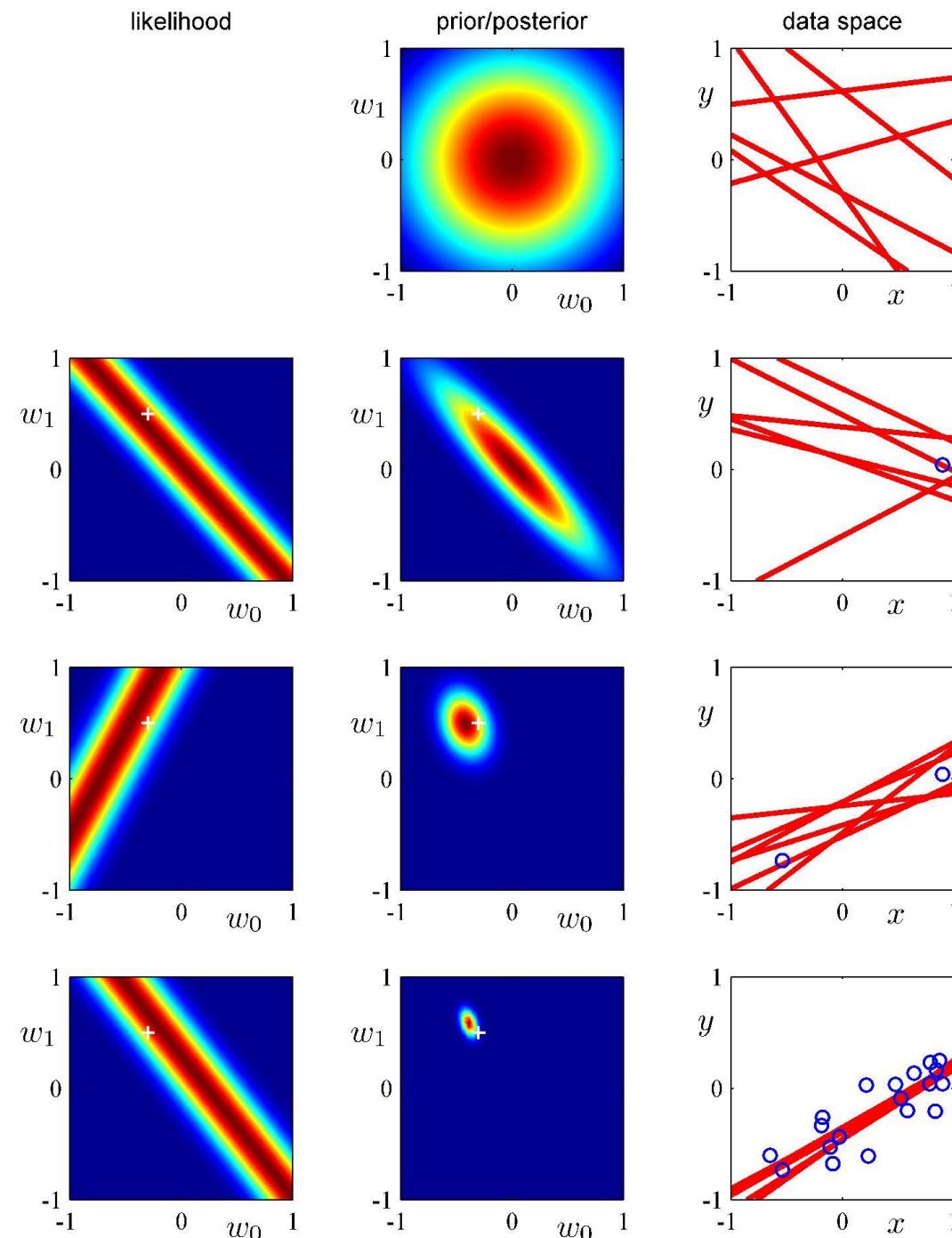
$$\lambda = \frac{\alpha}{\beta}$$

# Exemple

- Modèle à deux paramètres:

$$y(x, \mathbf{w}) = w_0 + w_1 x$$

- On peut imager la loi a posteriori sur les poids
- La vraisemblance est gaussienne  
→ loi a posteriori est aussi gaussienne si la loi a priori l'est



- Sans données, on échantillonne la loi a priori

- La loi a priori a peu d'impact si 20 points

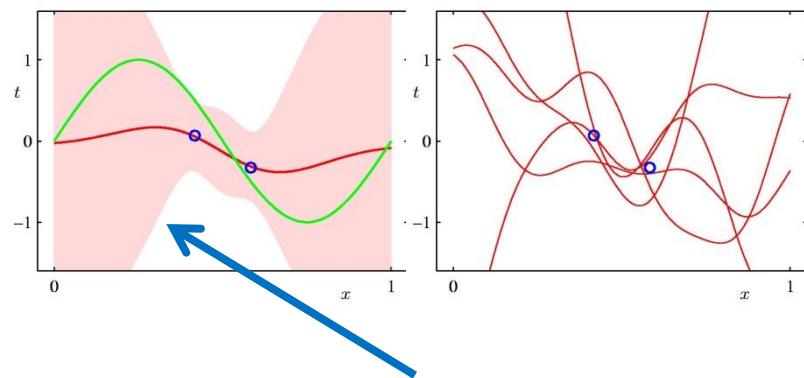
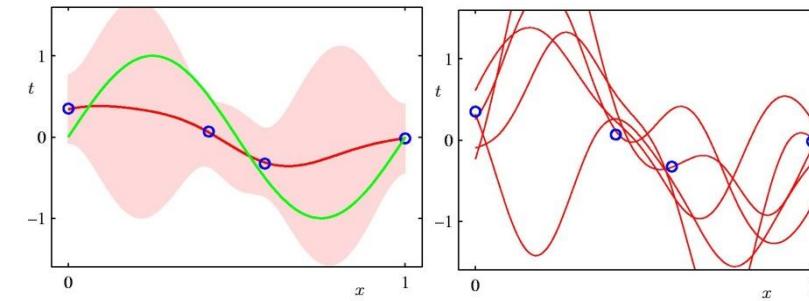
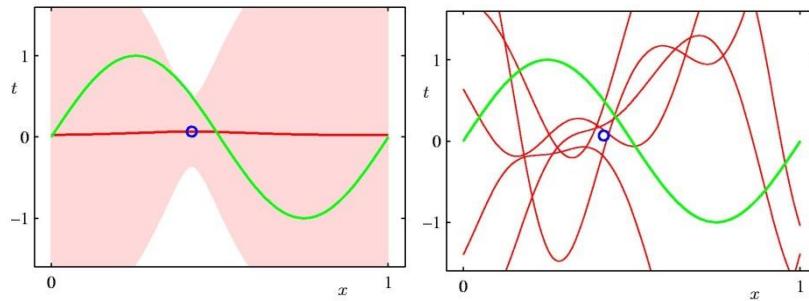
# Utilisation de la loi a posteriori

- Elle permet de faire des prédictions, des tirages aléatoires, des estimations d'intervalle de confiance...
  - Dans le cas gaussien, on peut calculer explicitement en intégrant sur les paramètres

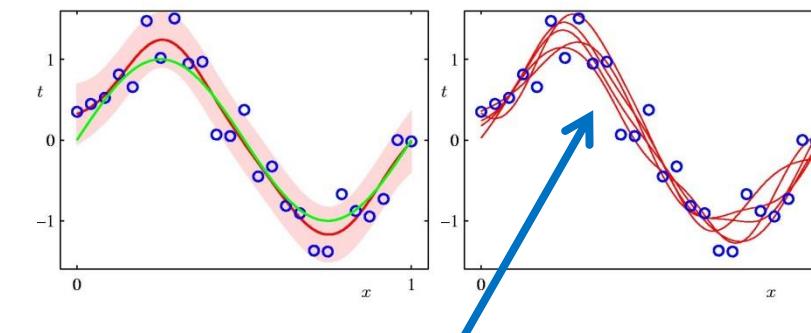
- On peut aussi introduire directement les corrélations dans les modèles et utiliser le « kernel trick »

➔ Processus Gaussiens (« kriging »)

# Exemple de prédiction à partir d'une base de fonctions sinusoïdales



Intervalle de confiance  
à un écart-type +  
moyenne

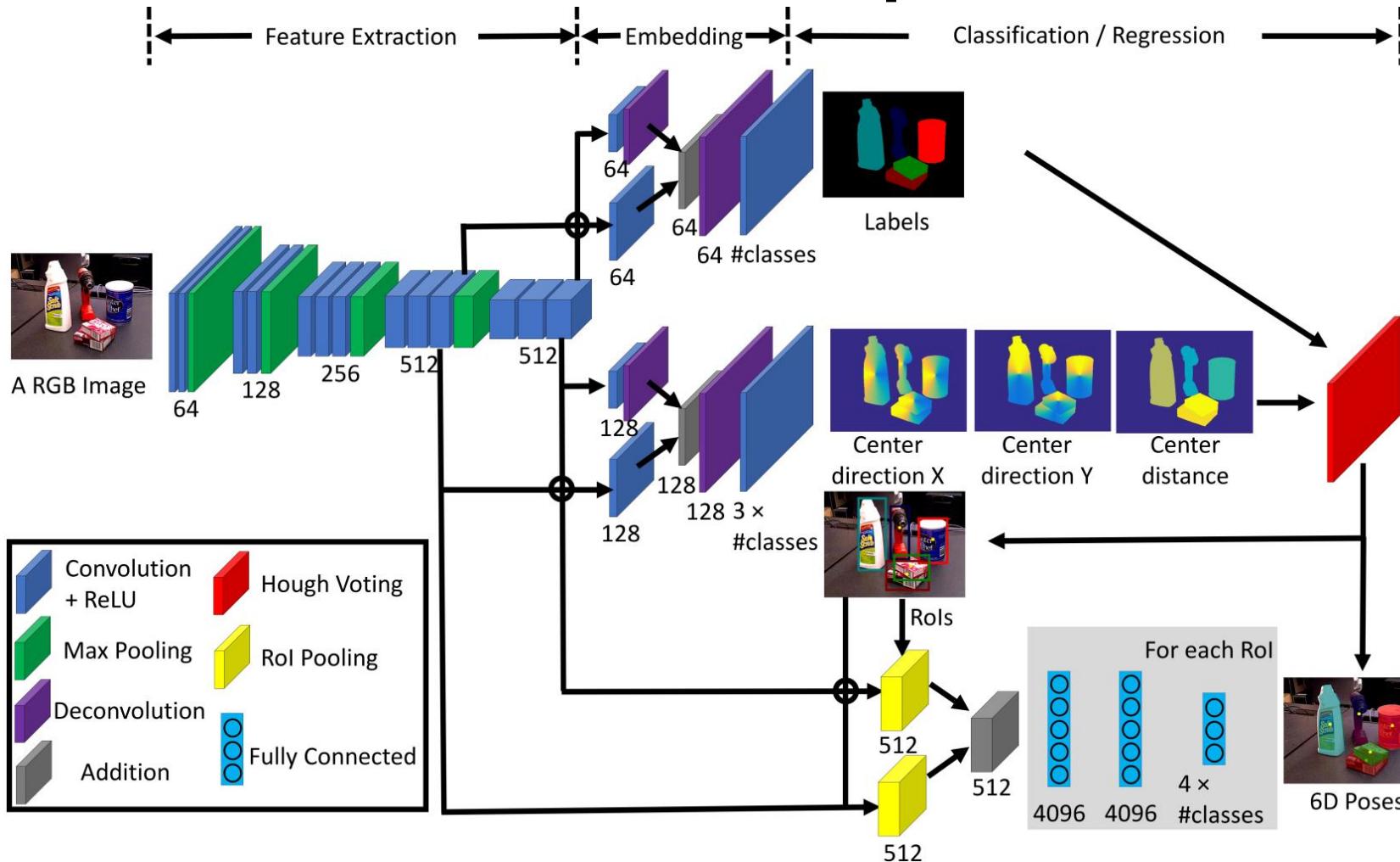


Echantillonnage de  
la loi a posteriori

# Régression et Deep Learning

- Les RN sont des fonctions paramétriques
- On dispose d'un algorithme « générique » d'optimisation: gradient stochastique (et variantes)
- Fonction de coût: erreur quadratique, cosinus...
- Difficulté: comment introduire la régularisation?
  - « Weight decay » (pénalisation L2)
  - « Drop-out »
  - « Early stopping »
  - Ajout de bruit
  - Multi-tâche
  - Lasso ?
  - Discussion générale ici: <https://www.deeplearningbook.org/contents/regularization.html>

# Estimation de pose 6D



<https://www.groundai.com/project/posecnn-a-convolutional-neural-network-for-6d-object-pose-estimation-in-cluttered-scenes/>

# « Single image super-resolution » en deep learning

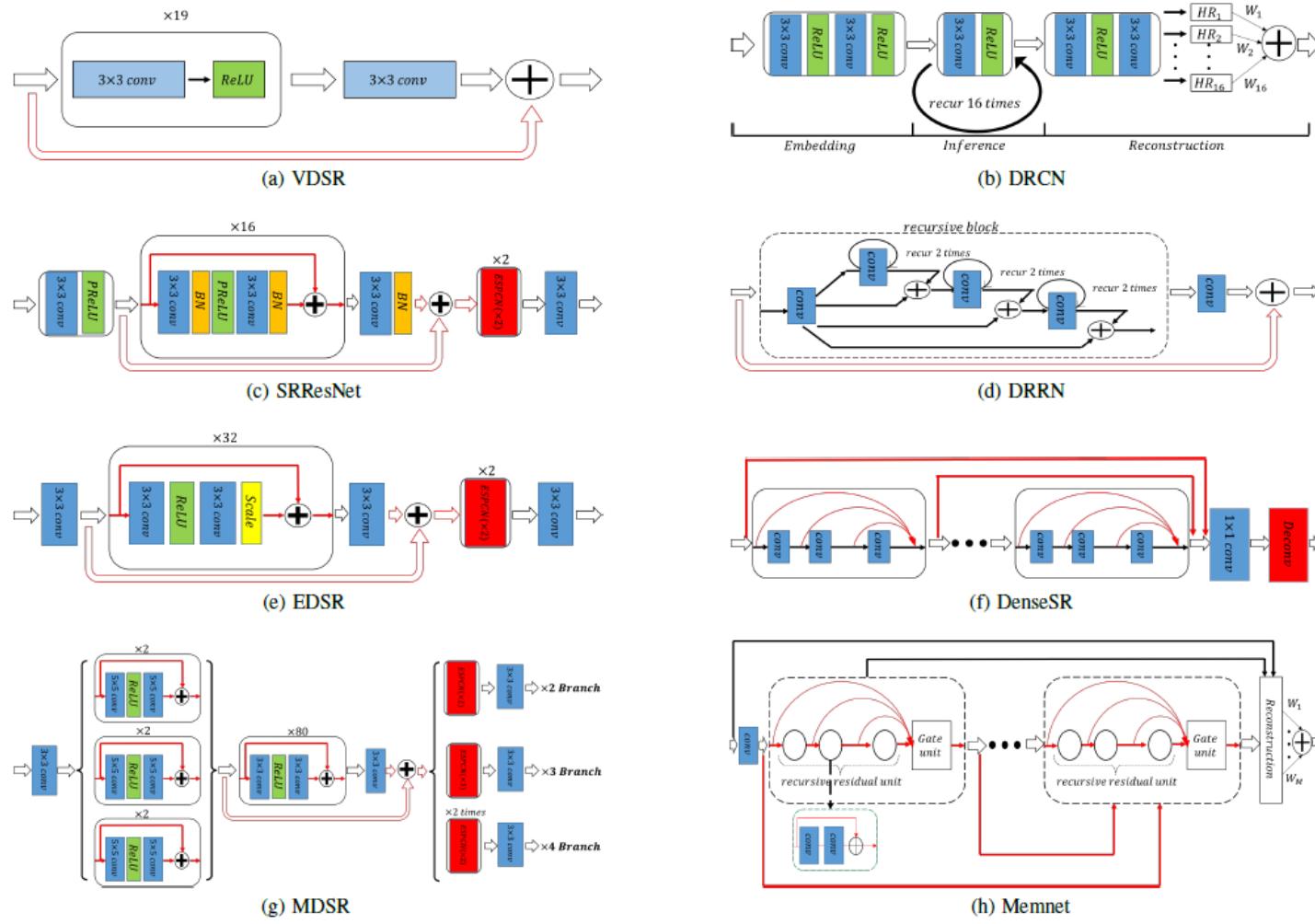


Figure 5: Sketch of several deep architectures for SISR.

Yang, W., Zhang, X., Tian, Y., Wang, W., & Xue, J. H. (2018). Deep learning for single image super-resolution: A brief review. *arXiv preprint arXiv:1808.03344*.

# Régression: les questions à se poser

- Quels sont les modèles & algorithmes?
  - Modèles linéaires généralisés
  - Processus gaussiens (krigeage)
  - Ensembles de prédicteurs (random forest, boosting, bagging...)
  - Réseaux de neurones
- Comment valider les modèles?
  - Validation croisée
  - Tests statistiques
- Comment maîtriser les grandes dimensions?
  - Projection / Construction de caractéristiques (« feature construction »)
  - Sélection de caractéristiques (« feature selection »)
  - Sparsité
- Comment contrôler les données aberrantes (« outliers »)?
  - Régularisation
  - Estimateurs robustes
  - RANSAC

# A retenir

- Régularisation
  - Un moyen de contrôler le compromis biais-variance
- SVM
  - Un algorithme optimal et flexible qui permet de traiter un grand nombre de configurations de données (en dimension raisonnable)
- Validation croisée
  - Un moyen empirique d'estimer l'erreur de généralisation
  - Une technique pour optimiser les hyper-paramètres (par ex. ceux du SVM)
- Régression
  - Plusieurs types de pénalisation
- Multi-classe
  - Un problème qui peut s'exprimer et se résoudre de différentes manières

# Implémentations logicielles

## SPAMS

- Bibliothèque orientée sparsité
- <http://spams-devel.gforge.inria.fr/documentation.html>

## Scikit-learn

- La plupart des modèles classiques implémentés
- Réseaux de neurones (mais pas profonds)
- [https://scikit-learn.org/stable/supervised\\_learning.html#supervised-learning](https://scikit-learn.org/stable/supervised_learning.html#supervised-learning)

## Pytorch

- Quelques exemples sur le site (<https://github.com/pytorch/examples>)
  - Régression polynomiale, Super-resolution
- Quelques tutoriels sur le web

## Tensorflow

- Quelques modules élémentaires
  - LinearRegressor, DNNRegressor
- Un tutoriel [https://www.tensorflow.org/tutorials/keras/basic\\_regression](https://www.tensorflow.org/tutorials/keras/basic_regression)

## Kaggle

- Plein d'exemples dans les « kernels »

# Références et sources

- Présentations et livre de C. Bishop (<https://www.microsoft.com/en-us/research/people/cmbishop/#!prml-book>)
- Cours de G. Hinton (<http://www.cs.toronto.edu/~hinton/csc2515/lectures.html>)
- Autres livres (en ligne):
  - Elements of Statistical Learning  
<https://web.stanford.edu/~hastie/Papers/ESLII.pdf>
  - An Introduction to Statistical Learning  
<http://www-bcf.usc.edu/~gareth/ISL/>
  - Gaussian processes  
<http://www.gaussianprocess.org/gpml/>
  - Machine Learning: a Probabilistic Perspective  
<https://www.cs.ubc.ca/~murphyk/MLbook/index.html>