

M2R IGI

Cours « Méthodes avancées en Image et Vidéo »

Machine Learning 1

Christian Wolf
INSA-Lyon
LIRIS

Méthodes avancées en image et vidéo

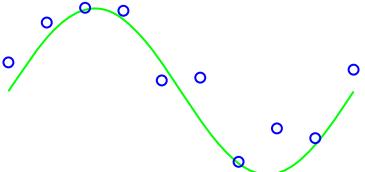


C. Wolf	Doua	Machine Learning 1 (introduction, classification)
C. Wolf	Doua	Machine Learning 2 (features, <i>deep learning</i> , modèles graphiques)
M. Ardabilian	ECL	Indexation par le contenu : évaluation des techniques et systèmes à base d'images
M. Ardabilian	ECL	Acquisition multi-image et applications
M. Ardabilian	ECL	Approches de super-résolution et applications
M. Ardabilian	ECL	Méthodes de fusion en imagerie
M. Ardabilian	ECL	Détection, analyse et reconnaissance d'objets
J. Mille	Doua	Méthodes variationnelles
M. Ardabilian	ECL	La biométrie faciale
E. Dellandréa	ECL	Le son au sein de données multimédia : codage, compression et analyse
S. Duffner	Doua	Suivi d'objets
E. Dellandréa	ECL	Un exemple d' analyse audio : Reconnaissance de l'émotion à partir de signaux de parole et de musique
S. Bres	Doua	Indexation d'images et de vidéos
V. Eglin	Doua	Analyse de documents numériques 1
F. LeBourgeois	Doua	Analyse de documents numériques 2

Sommaire

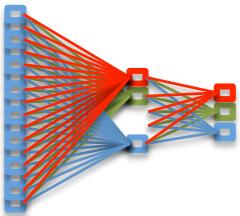
Introduction

- Principes générales
- *Fitting* et généralisation, complexité des modèles
- Minimisation du risque empirique



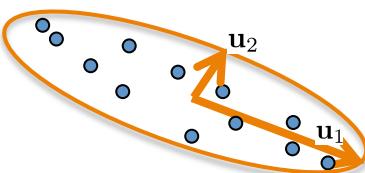
Classification supervisée

- *KPPV* (k plus proches voisins)
- Modèles linéaires pour la classification
- Réseaux de neurones
- Arbres de décisions et forêts aléatoires
- SVM (*Support Vector machines*)

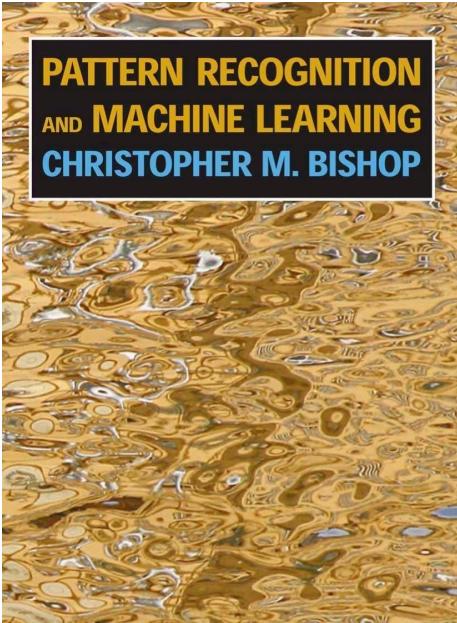


Extraction de caractéristiques

- Manuelle
- ACP (Analyse de composantes principales)
- Réseaux de neurones convolutionnels
(« *deep learning* »)



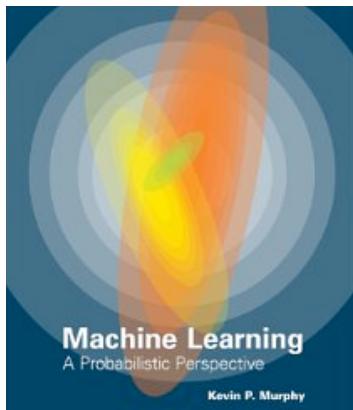
Quelques sources



Christopher M. Bishop
“Pattern Recognition and Machine Learning”
Springer Verlag, 2006



Microsoft Research,
Cambridge, UK



Kevin P. Murphy, “Machine Learning”
MIT Press, 2013



Google
Formerly : University of British Columbia,

Reconnaissance de formes

Classification supervisée



7210414959
0690159734
9665407401
3134727121
1742351244



Reconnaissance d'objets à partir d'images:

- Chiffres, lettres, logos
- Biométrie: visages, empreintes, iris, ...
- Classes d'objets (voitures, arbres, bateaux, ...)
- Objects spécifiques (une voiture, un mug, ...)

Prédiction

De manière générale, nous aimerais prédire une valeur t à partir d'une observation x

$$t = y(x, w)$$

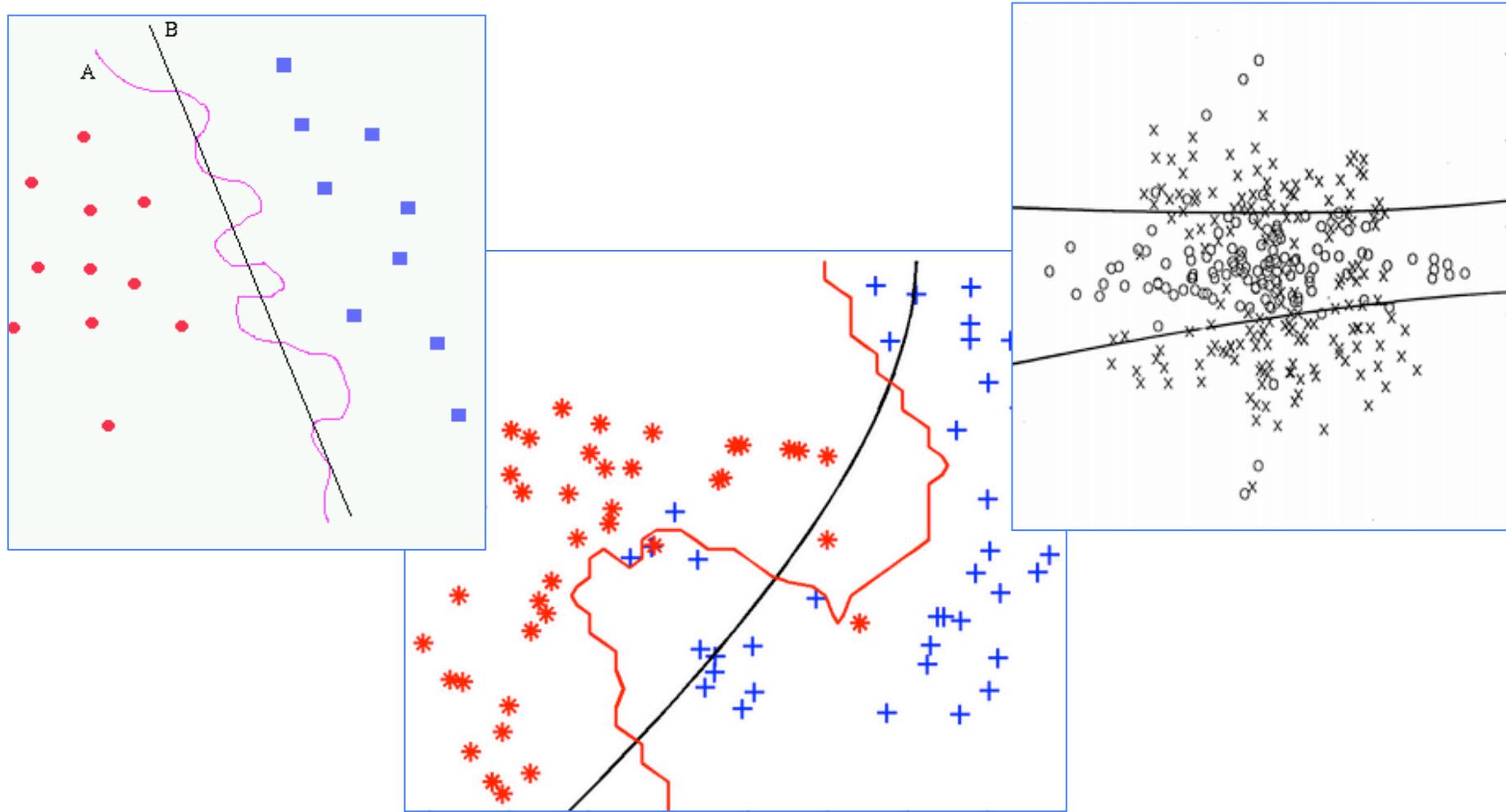
Si t est continue : [régression](#)

Si t est discret : [classification](#)

Apprentissage des paramètres w à partir de données.

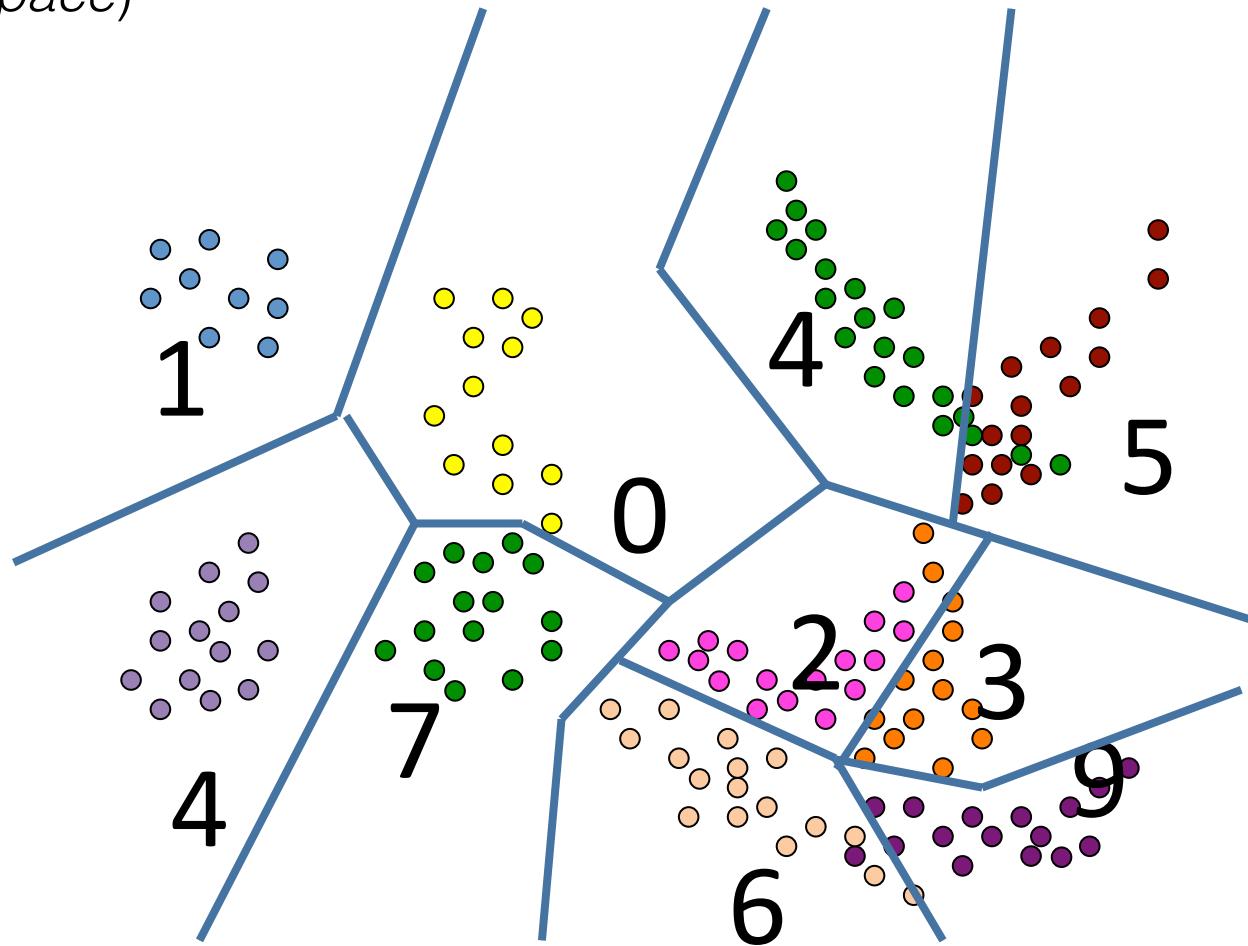
Apprentissage et généralisation

Apprendre à classifier des données revient à apprendre une fonction de décision : la frontière entre les classes.



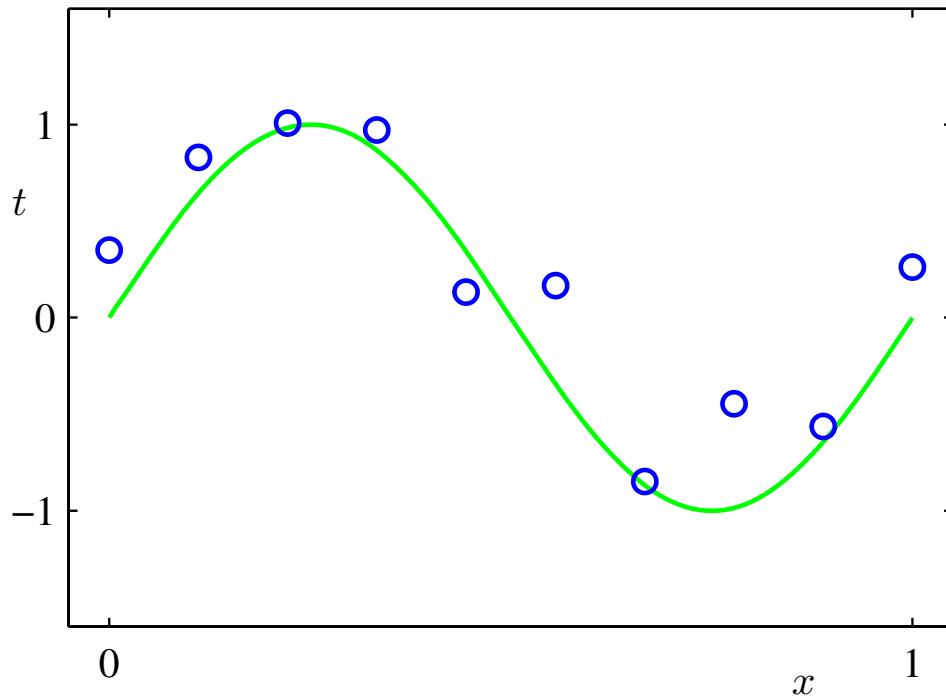
Apprentissage et généralisation

La complexité d'une fonction de décision dépend de la complexité du regroupement des étiquettes dans l'espace de caractéristiques (*feature space*)



Fitting et Généralisation

- Données générées par une fonction $t = \sin(2\pi x)$
- Objectif : supposant la fonction inconnue, prédire t à partir de x



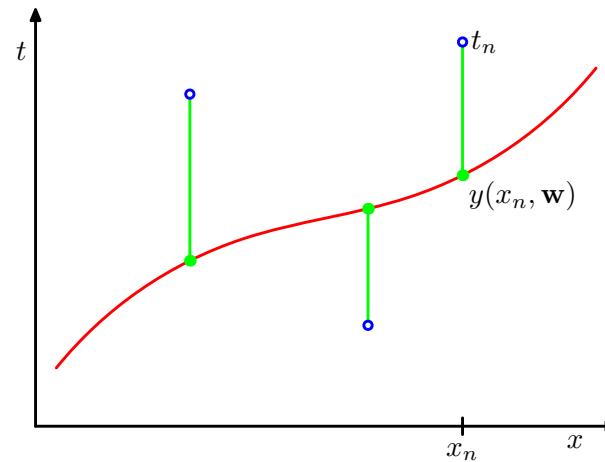
Fitting et Généralisation

« Fitting » d'un polynôme d'ordre M

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{j=0}^M w_j x^j$$

Critère des « moindres carrés » (des erreurs)

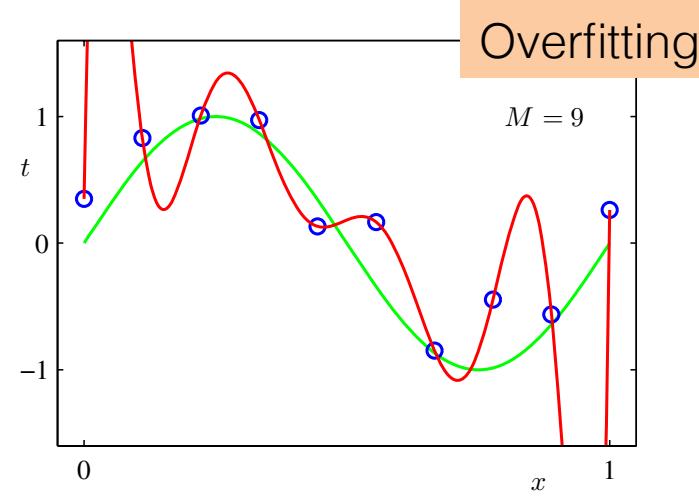
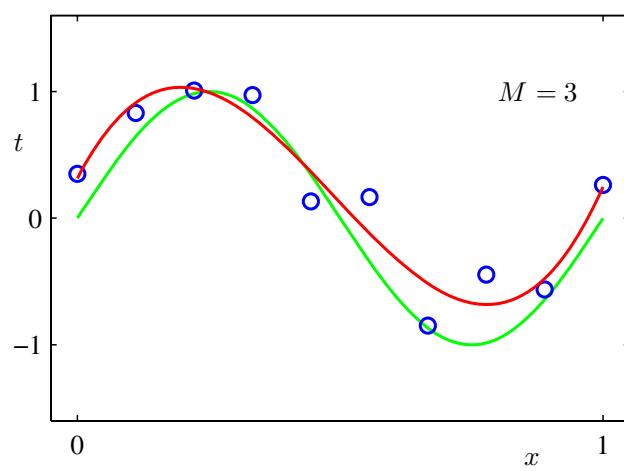
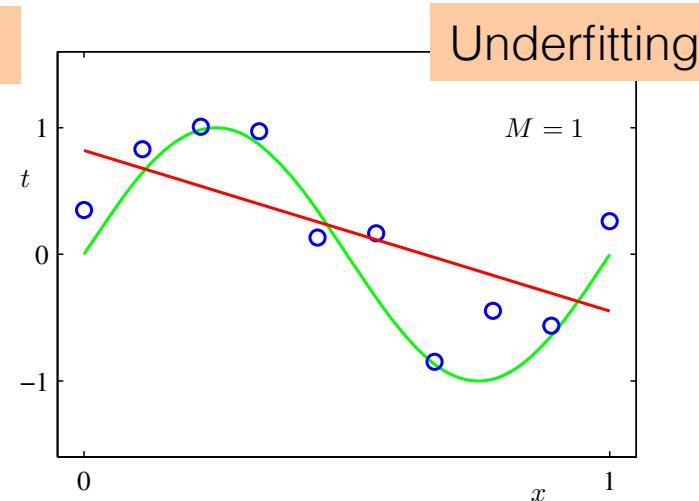
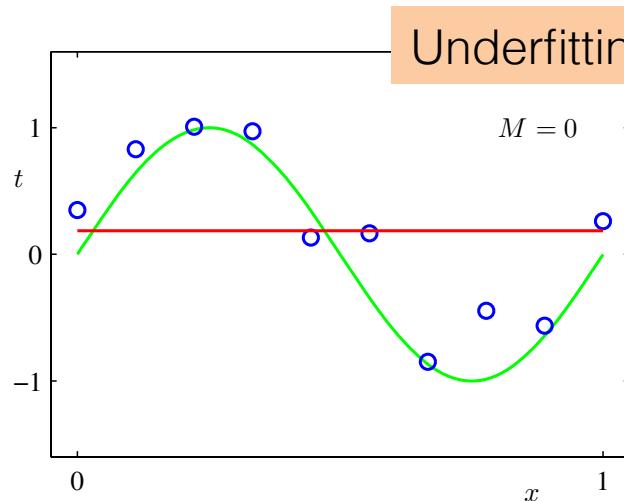
$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2$$



Dérivée linéaire -> solution directe

Sélection d'un modèle

Quel ordre M pour le polynôme?



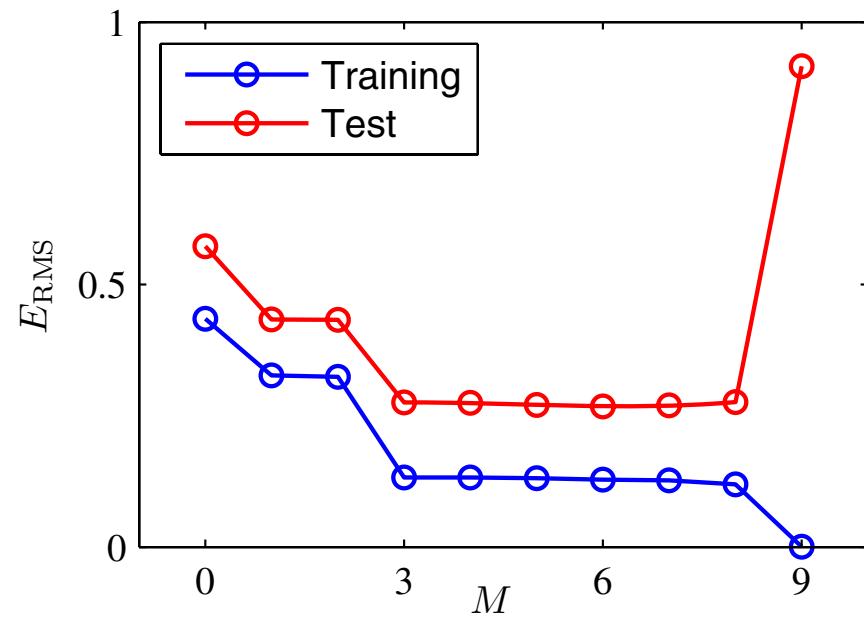
Sélection d'un modèle

Séparation des données en deux parties :

- Base d'apprentissage (*Training set*)
- Base de test (*test set*)

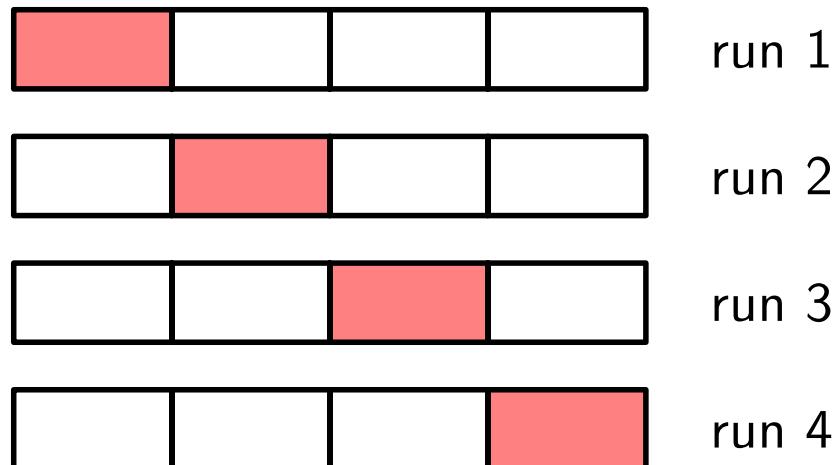
Root Mean Square Error (RMS)

$$E_{\text{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$$



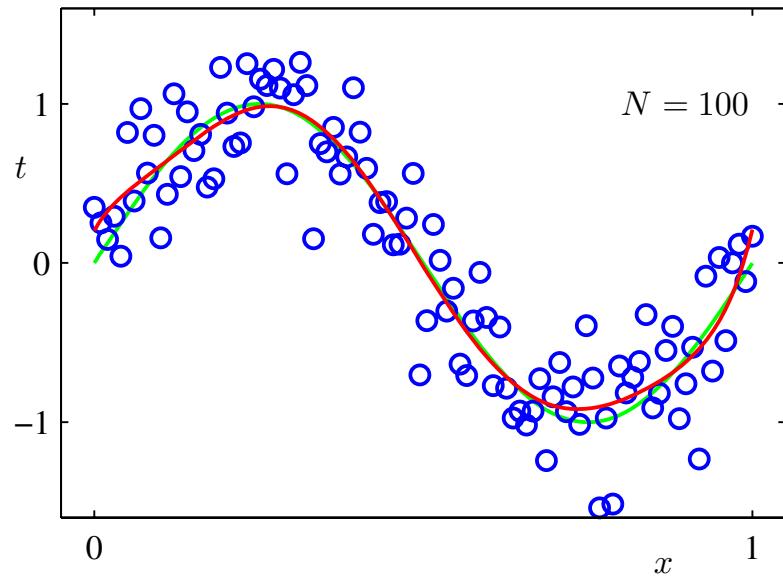
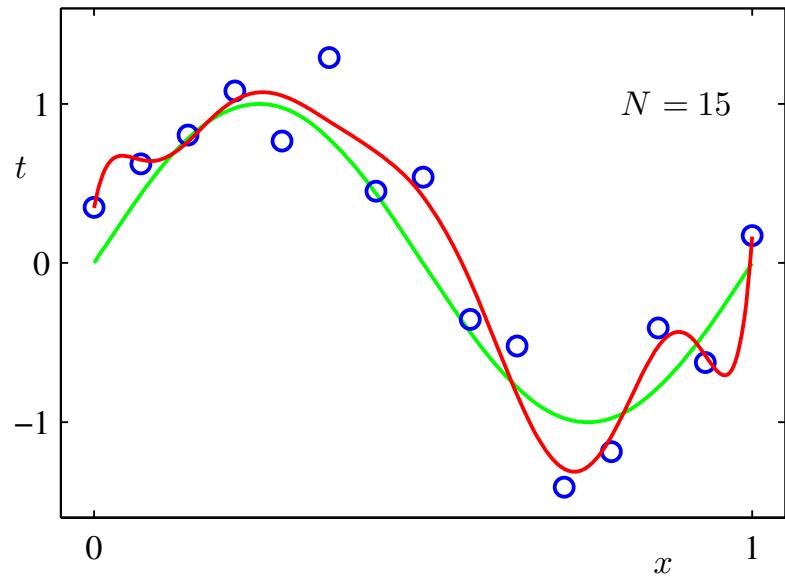
Validation croisée

La séparation des données en deux parties change itérativement. La mesure de performance est la moyenne sur toutes les itérations.



Big Data!

Le problème de l'*overfitting* diminue en augmentant la taille de la base d'apprentissage.



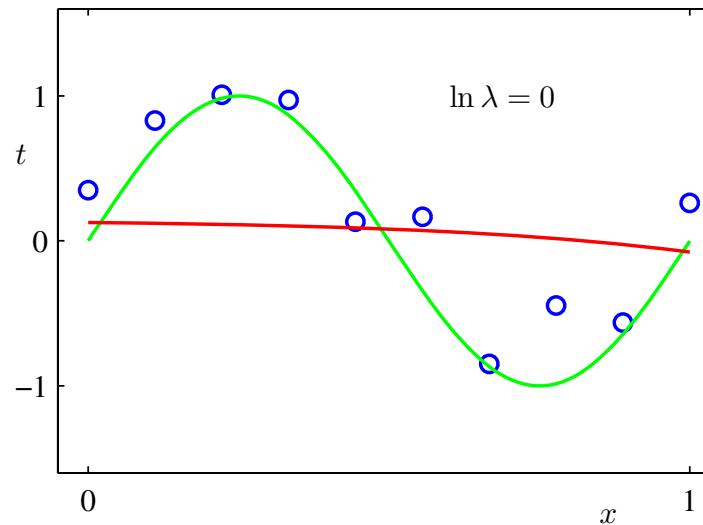
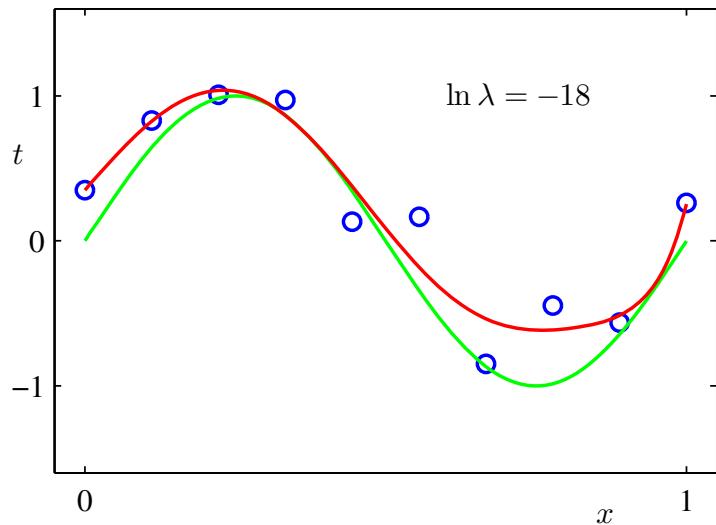
$M=9$

Régularisation

Ajouter un terme supplémentaire : restriction des paramètres du modèle

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2 + \boxed{\frac{\lambda}{2}} \|\mathbf{w}\|^2$$

Paramètre de régularisation
w₀ est souvent exclu

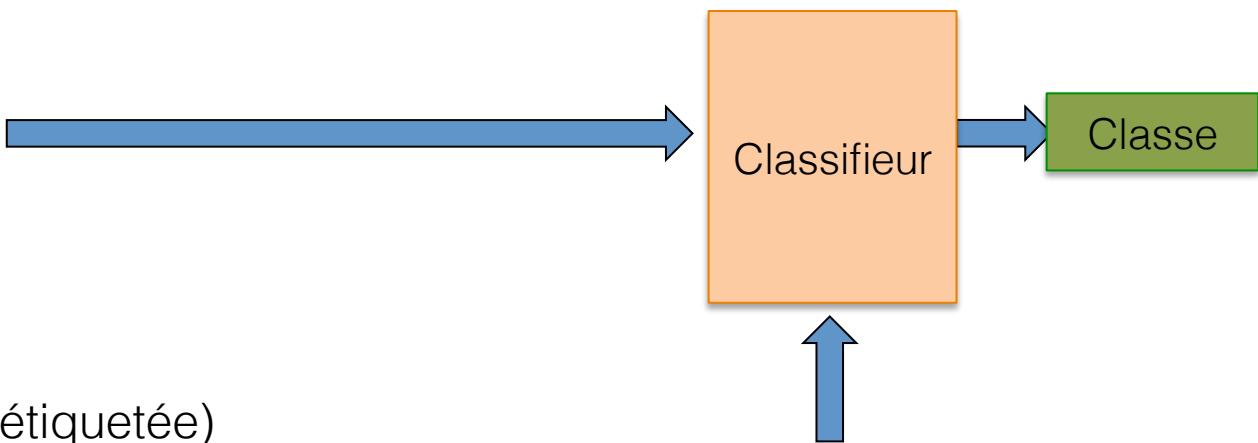


$M=9$

La classification supervisée

Nouvelle entrée

4



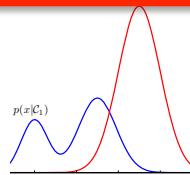
Base d'apprentissage (étiquetée)

7210414959
0690159784
7219665407401
0693137210414959
972104149599784
306901597347401
196654074017121
31347271211244
1742351244

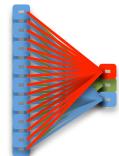
La classification supervisée



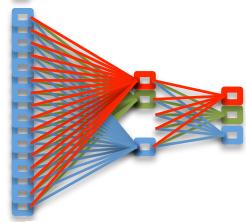
kppV (k plus proches voisins)



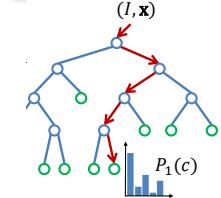
Classificateurs génératifs linéaires
(Bayésiens)



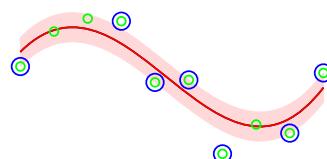
Classificateurs discriminatifs linéaires



Réseaux de neurones



Forêts aléatoires



SVM (*Support Vector Machines*)

“k-plus proches voisins”

Probablement le classifieur le plus simple.

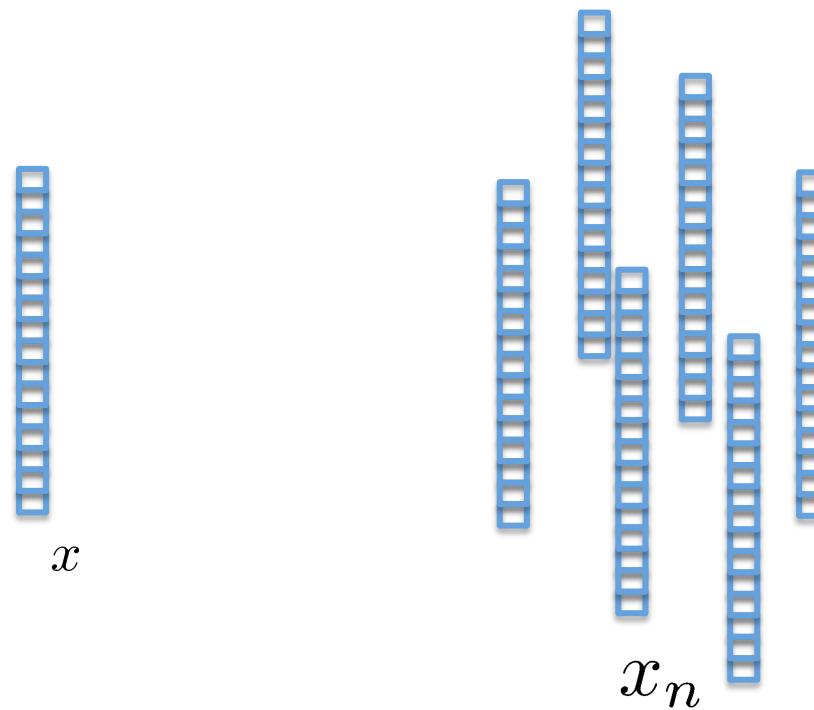
La base d'apprentissage (vecteurs x_n et étiquettes t_n) est stocké.

Pour un nouveau vecteur x , la donnée x_n la plus proche est cherchée et son étiquette sert comme prédiction.

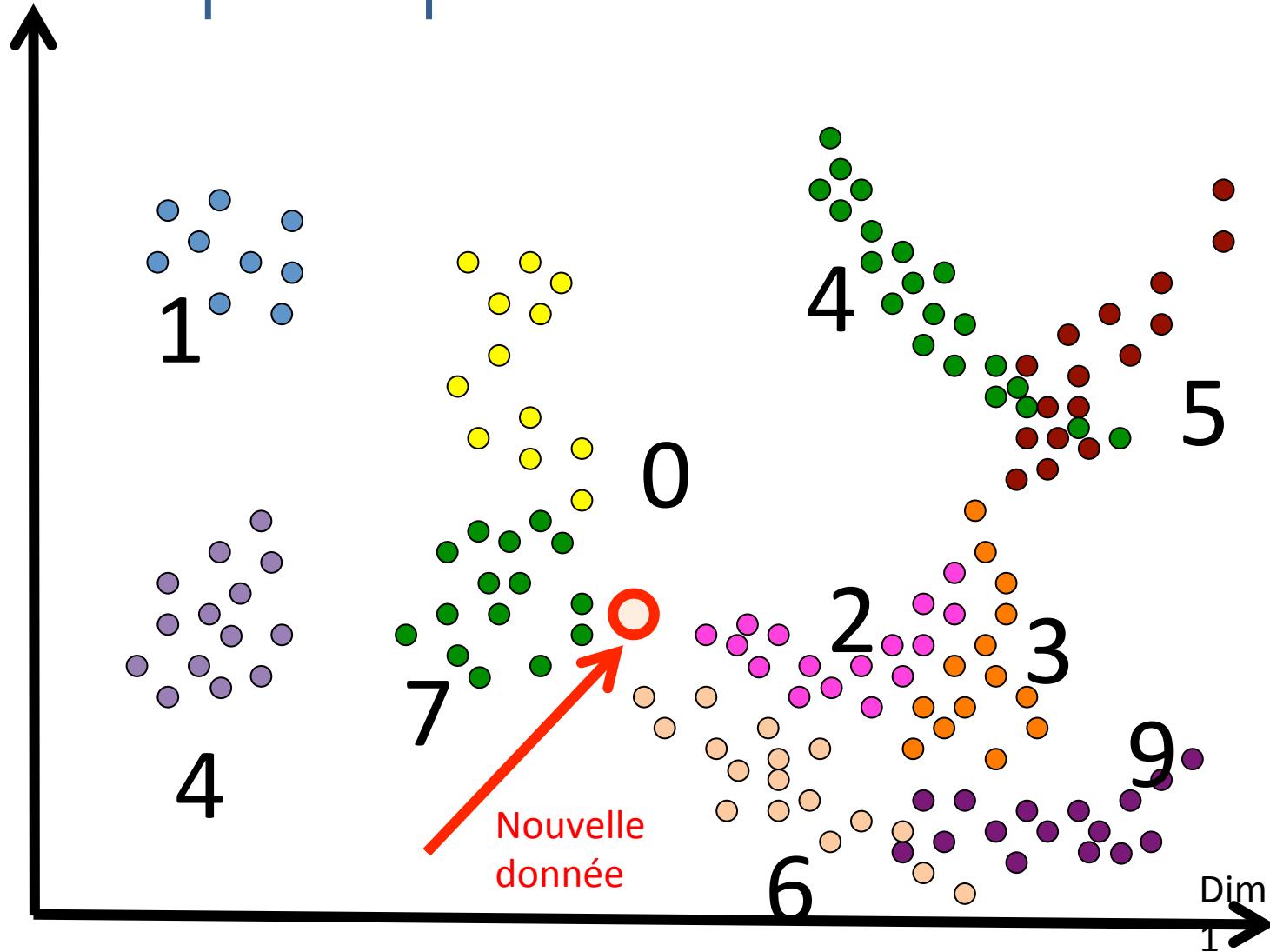
$$\hat{t} = t_m$$

où

$$m = \arg \min ||x - x_n||$$

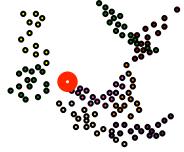


“k-plus proches voisins”

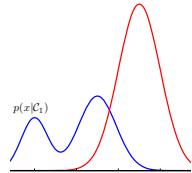


Optimal dans la limite d'un nombre infini de données d'apprentissage.

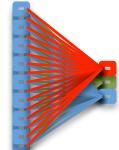
La classification supervisée



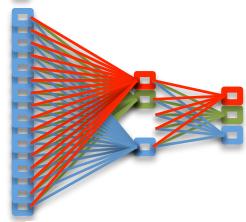
kppV (k plus proches voisins)



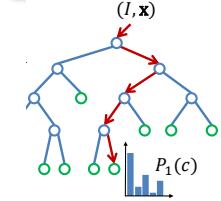
Classificateurs génératifs (Bayésiens)



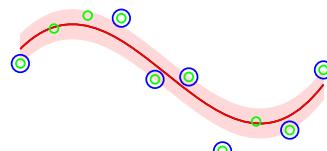
Classificateurs discriminatifs linéaires



Réseaux de neurones



Forêts aléatoires



SVM (*Support Vector Machines*)

Modélisation probabiliste

Supposons un problème de classification en deux classes \mathcal{C}_1 et \mathcal{C}_2 à partir de données 1D modélisées par une variable aléatoire x (L'observation)

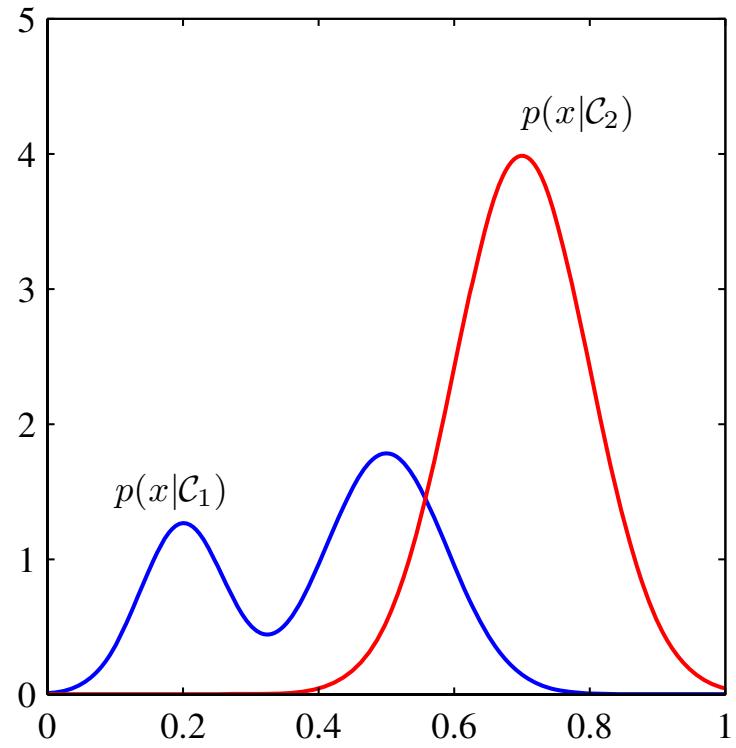
Modélisation probabiliste :

$$\hat{\mathcal{C}} = \arg \max_{\mathcal{C}_k} p(\mathcal{C}_k | x)$$

Probabilité postérieure

A notre disposition : la vraisemblance des données (*likelihood*)

$$p(x|\mathcal{C}_k)$$



Modélisation Bayesienne

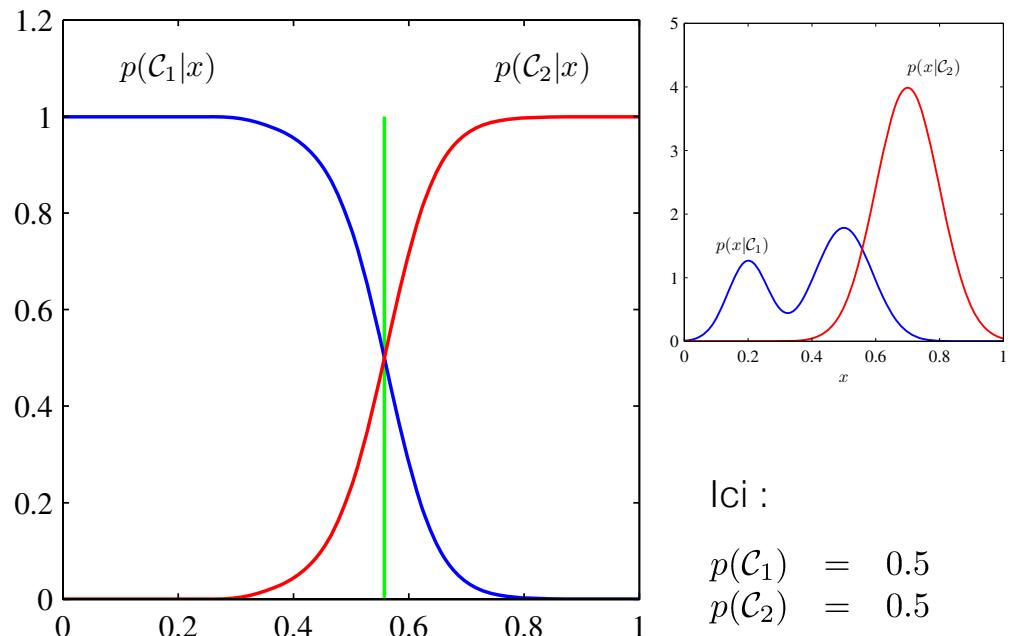
Pour obtenir la probabilité postérieure, il faut inverser le modèle à l'aide de la règle de Bayes :

$$p(\mathcal{C}_k | \mathbf{x}) = \frac{Vraisemblance}{Probabilité a priori}$$
$$p(\mathcal{C}_k | \mathbf{x}) = \frac{p(\mathbf{x} | \mathcal{C}_k) p(\mathcal{C}_k)}{p(\mathbf{x})}$$

Peut être ignorée pour la maximisation

La probabilité *a priori* modélise nos connaissances sur le résultat, indépendamment des observations x . Dans les cas simples, elle est tabulée, e.g.

$$\begin{aligned} p(\mathcal{C}_1) &= 0.7 \\ p(\mathcal{C}_2) &= 0.3 \end{aligned}$$



Ici :

$$\begin{aligned} p(\mathcal{C}_1) &= 0.5 \\ p(\mathcal{C}_2) &= 0.5 \end{aligned}$$

Génération de nouvelles données

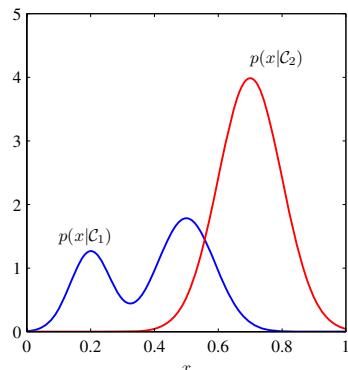
Cette famille de modèles porte le nom « modèles génératifs » parce qu'elle permet de générer des nouvelles données respectant le modèle.

1. On échantillonne une classe \mathcal{C}_k selon la distribution *a priori*
e.g.

$$\begin{aligned} p(\mathcal{C}_1) &= 0.7 \\ p(\mathcal{C}_2) &= 0.3 \end{aligned}$$

2. Pour la classe donnée, on échantillonne l'observation selon la vraisemblance $p(x|\mathcal{C}_k)$

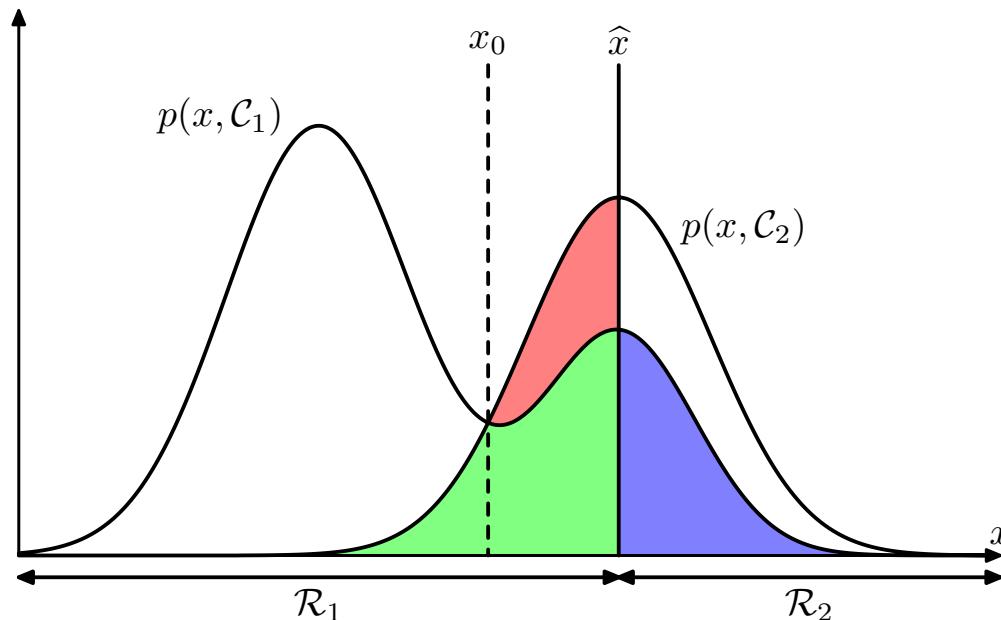
e.g.



[C. Bishop, Pattern recognition and Machine learning, 2006]

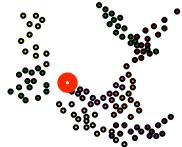
Probabilité d'erreur

La probabilité de commettre une erreur d'un certain type peut être calculée de manière directe :

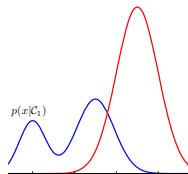


En changeant le seuil de décision \hat{x} , la zone rouge est compressible.

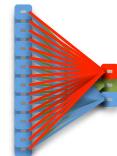
La classification supervisée



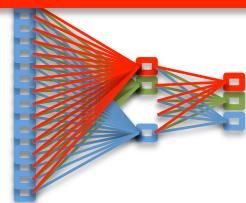
kppV (k plus proches voisins)



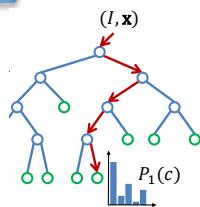
Classificateurs génératifs linéaires
(Bayésiens)



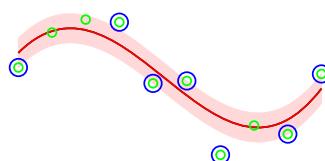
Classificateurs discriminatifs linéaires



Réseaux de neurones



Forêts aléatoires



SVM (*Support Vector Machines*)

Modèles linéaires (2 classes)

Modèle linéaire pour la classification

Une fonction de décision est modélisée par une fonction paramétrique :

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

w_0 est un biais qui peut être intégré dans les autres paramètres en ajoutant la constante « 1 » aux entrées

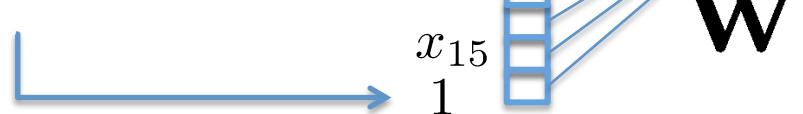
$$y(\mathbf{x}) = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}$$

Interprétation :

$y(x) \geq 0 \rightarrow$ Classe 1

$y(x) < 0 \rightarrow$ Classe 2

La constante « 1 » ajoute
un « biais » au modèle



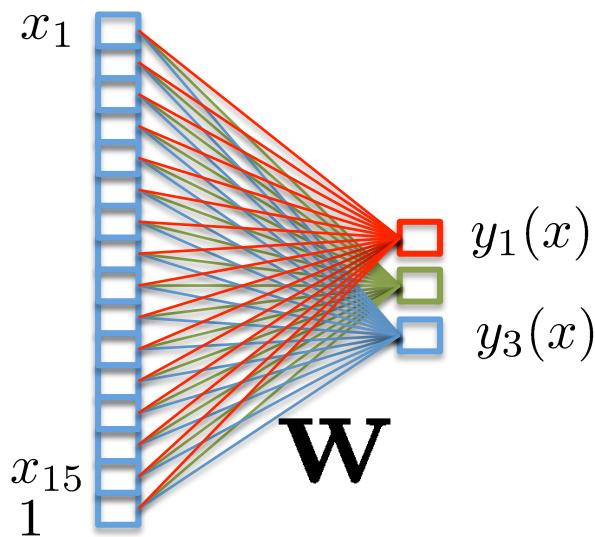
Modèles linéaires (K classes)

Plusieurs fonctions paramétriques :

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

En notation vectorielle, et en intégrant le biais :

$$\mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^T \widetilde{\mathbf{x}}$$



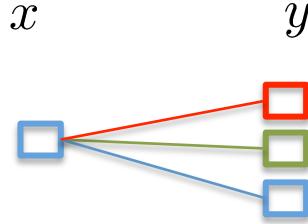
Interprétation :

Classe k si $y_k(x) > y_j(x) \quad \forall j \neq k$

« *The winner takes it all* »

Problème simple (1D, 3 classes)

Classifieur linéaire : entrées 1D, 3 classes



Entrée :

$$x = \begin{bmatrix} \cdot \\ 1 \end{bmatrix} \quad \mathbf{w}_k = \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$$

Paramètres :

Sortie d'une classe :

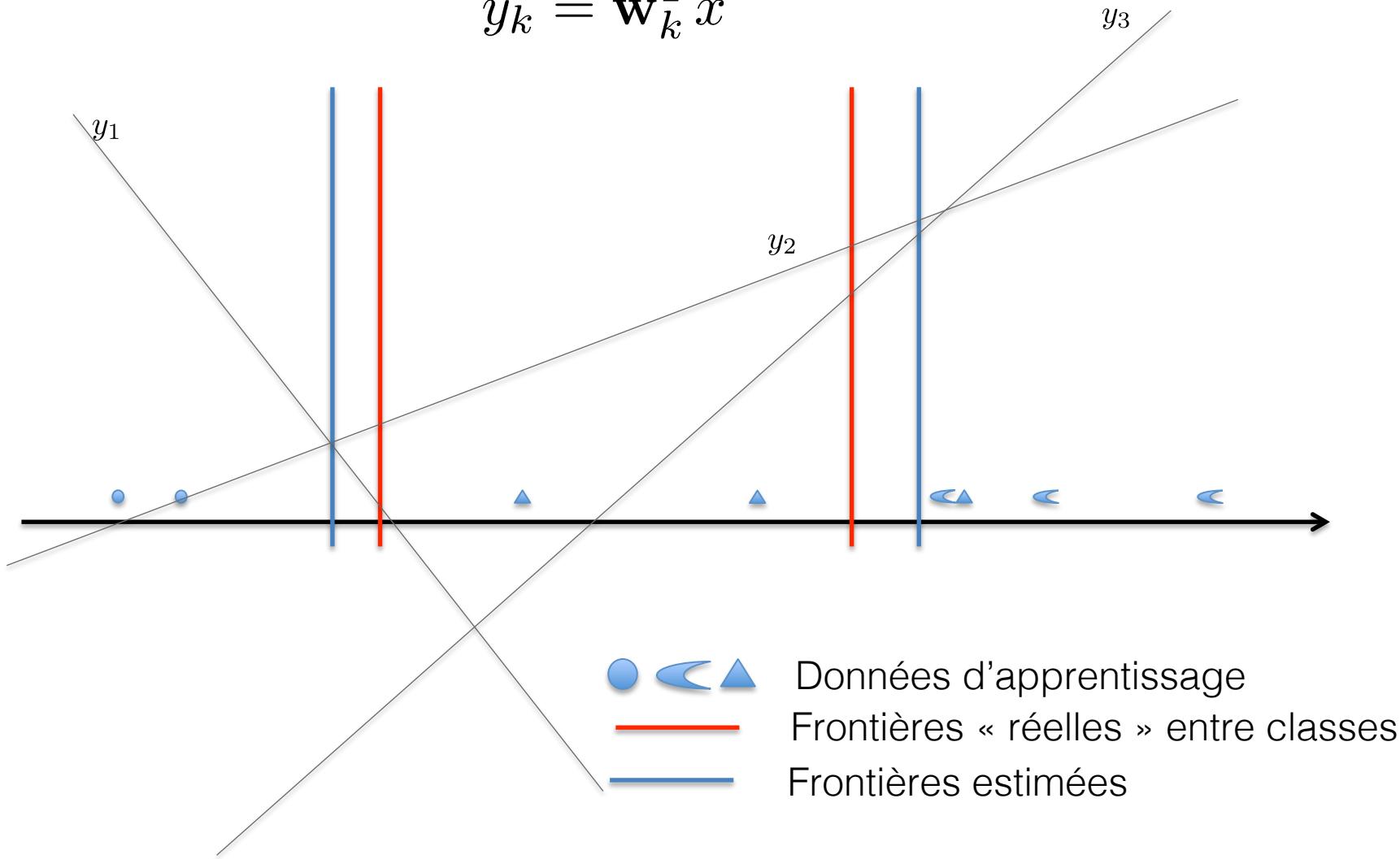
$$y_k = \mathbf{w}_k^T x$$

Sortie de toutes les classes :

$$y = Wx$$

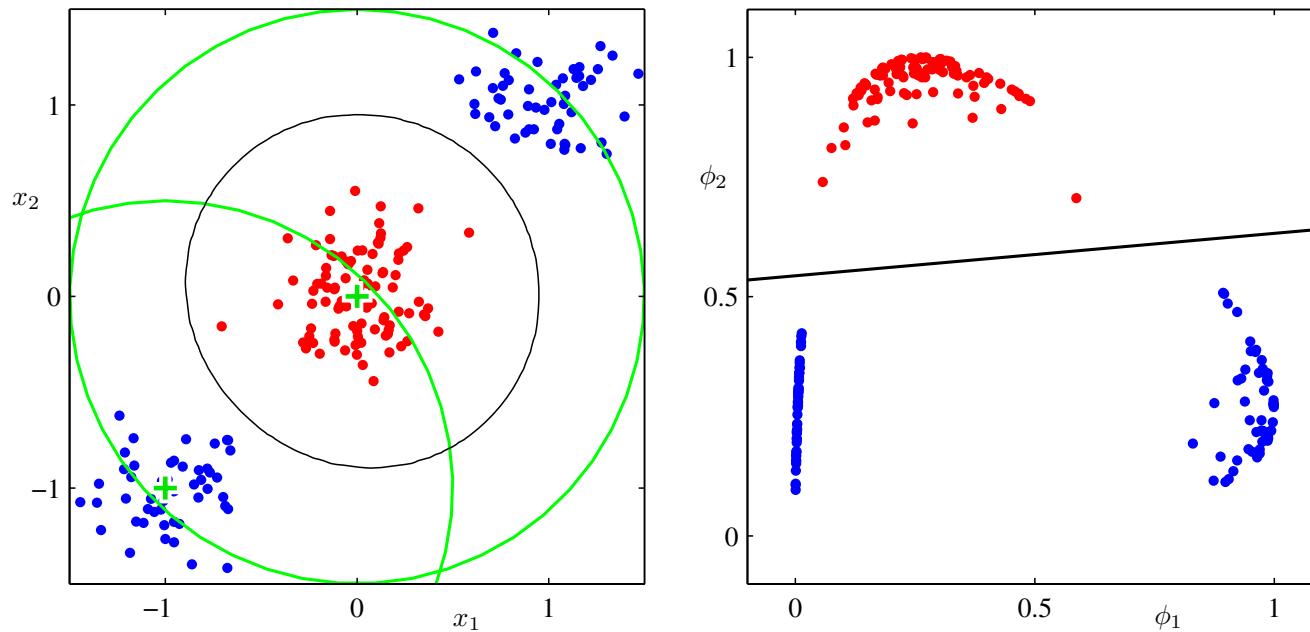
Problème simple (1D, 3 classes)

$$y_k = \mathbf{w}_k^T x$$



Le cas non-linéaire

Prétraitement : transformation non-linéaire des données
(à choisir préalablement selon l'application) :



Fonctions de bases Gaussiennes

Régression logistique (2 classes)

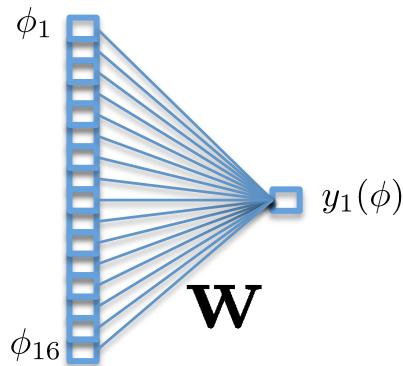
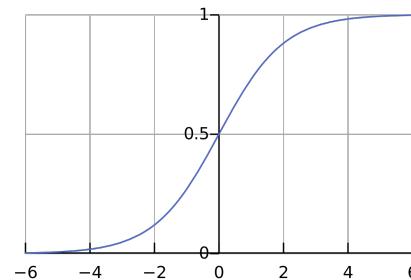
Modèle linéaire (sur entrées transformées) + non-linéarité à la sortie

$$p(\mathcal{C}_1 | \phi) = y(\phi) = \sigma(\mathbf{w}^T \phi)$$

Modélisation directe de la probabilité postérieure

σ est la fonction logistique (« sigmoïde ») assurant que les sorties sont entre 0 et 1

$$\sigma(\eta) = \frac{1}{1 + \exp(-\eta)}$$



Régression logistique (K classes)

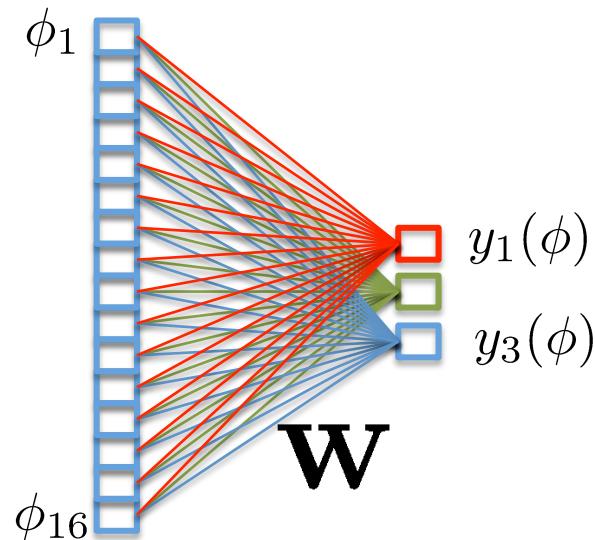
Extension similaire au cas linéaire

$$p(\mathcal{C}_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

« Softmax » pour assurer que les sorties sont des probabilités

$$a_k = \mathbf{w}_k^T \phi.$$

Linéarités



Régression logistique : apprentissage

Un ensemble de données d'apprentissage est supposé connu :

Les entrées x_n , transformées par les fonctions des bases $\phi(x_n)$

Les sorties t_n sont connues, codées en « 1-à-K »

(« *hot-one-encoded* ») :

$$t_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \quad \text{Classe « réelle » de l'échantillon n}$$

Objectif : apprendre les paramètres \mathbf{w} selon un critère d'optimalité

Apprentissage des paramètres

La vraisemblance des données est la probabilité d'obtenir les observations étant donné les paramètres et les entrées :

$$p(\mathbf{T}|\mathbf{w}_1, \dots, \mathbf{w}_K) = \prod_{n=1}^N \prod_{k=1}^K p(\mathcal{C}_k | \boldsymbol{\phi}_n)^{t_{nk}} = \prod_{n=1}^N \prod_{k=1}^K y_{nk}^{t_{nk}}$$

Pour estimer les paramètres \mathbf{w} , on minimisera une fonction d'erreur (le logarithme négatif de la vraisemblance) :

$$E(\mathbf{w}_1, \dots, \mathbf{w}_K) = -\ln p(\mathbf{T}|\mathbf{w}_1, \dots, \mathbf{w}_K) = -\sum_{n=1}^N \sum_{k=1}^K t_{nk} \ln y_{nk}$$

Cette fonction de cout est connue sous le nom

« Cross-entropy loss »

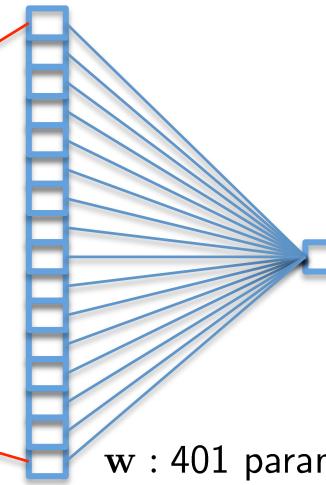
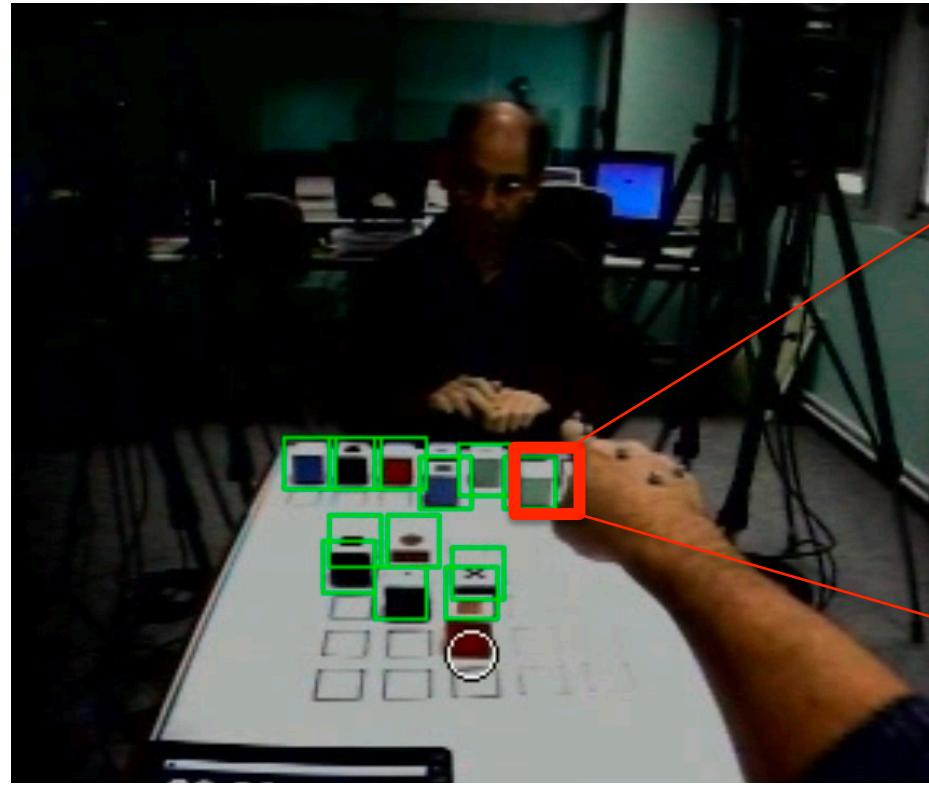
En calculant son gradient, elle peut être minimisée :

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \dots, \mathbf{w}_K) = \sum_{n=1}^N (y_{nj} - t_{nj}) \boldsymbol{\phi}_n$$

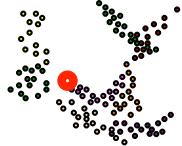
Application : détection d'objets simples

Une fenêtre de taille 20x20 est glissée sur l'image.

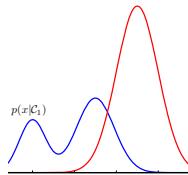
Les pixels d'une fenêtre sont donnés comme entrées



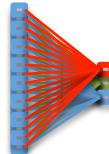
La classification supervisée



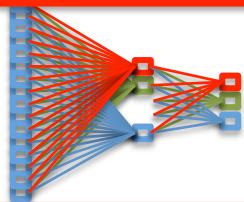
kppV (k plus proches voisins)



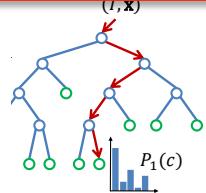
Classifieurs génératifs linéaires
(Bayésiens)



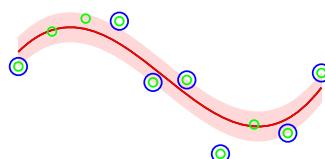
Classifieurs discriminatifs linéaires



Réseaux de neurones



Forêts aléatoires



SVM (*Support Vector Machines*)

Réseaux de neurones

Régression logistique : les fonctions de bases sont limitées, à choisir manuellement pour chaque application.

Et si on les apprenait?

$$y(\mathbf{x}, \mathbf{w}) = f \left(\sum_{j=1}^M w_j \phi_j(\mathbf{x}) \right)$$

Si les fonctions de base sont de la même forme que les fonctions des sorties :

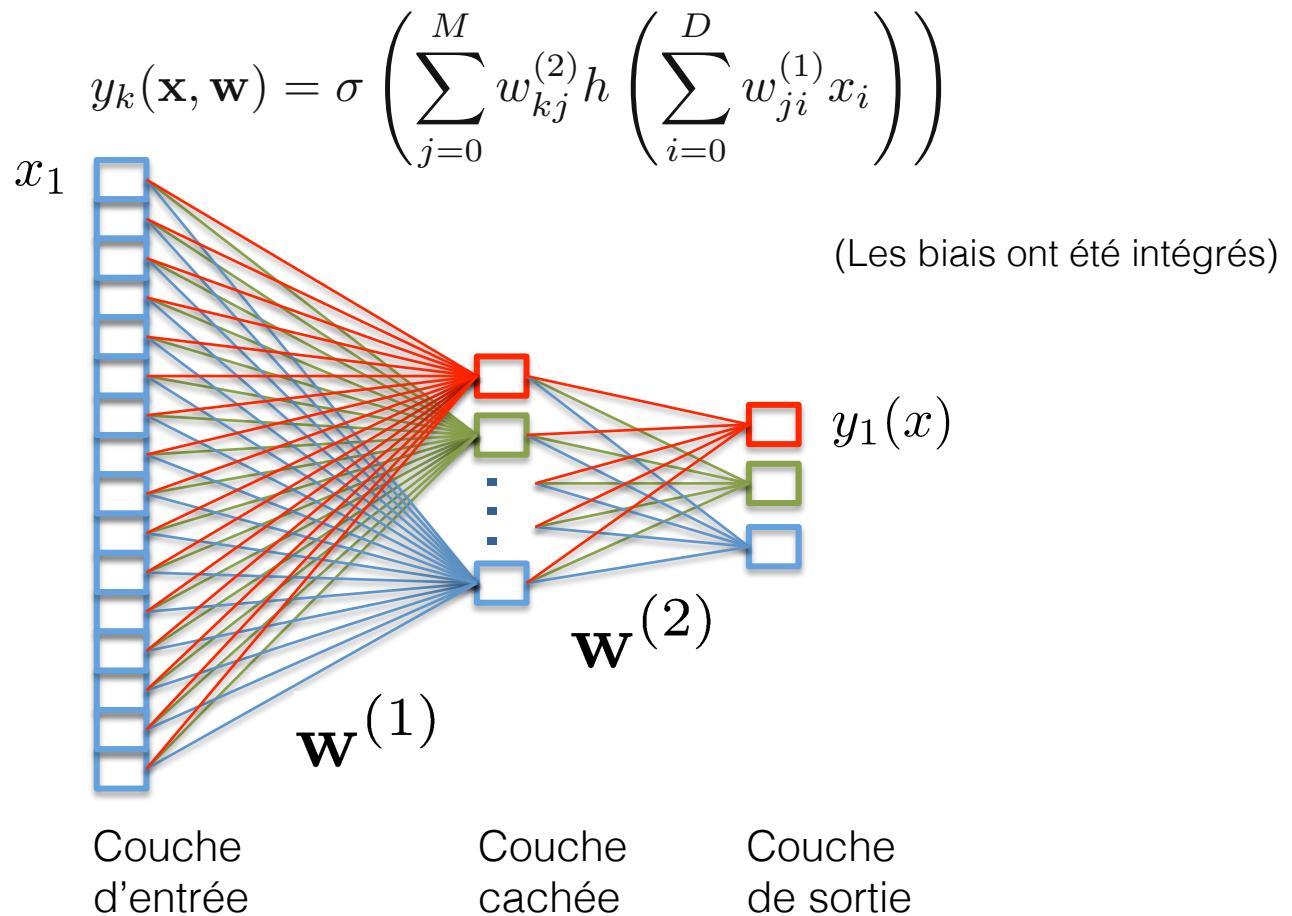
$$\phi_j(x) = h \left(\sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right)$$

f et h sont des fonctions non-linéaires (« fonctions d'activations »). Généralement, il s'agit de sigmoid, de tanh ou de softmax.

Réseaux de neurones

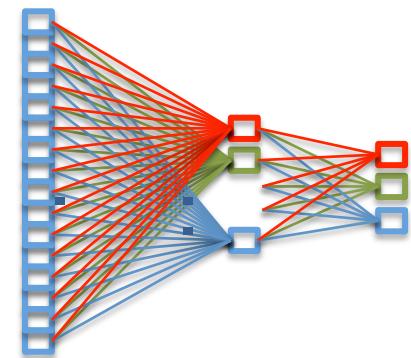
Cela donne un réseau de neurones à deux couches (une couche cachée, une couche de sortie) :

« Multi-layer perceptron » (MLP)



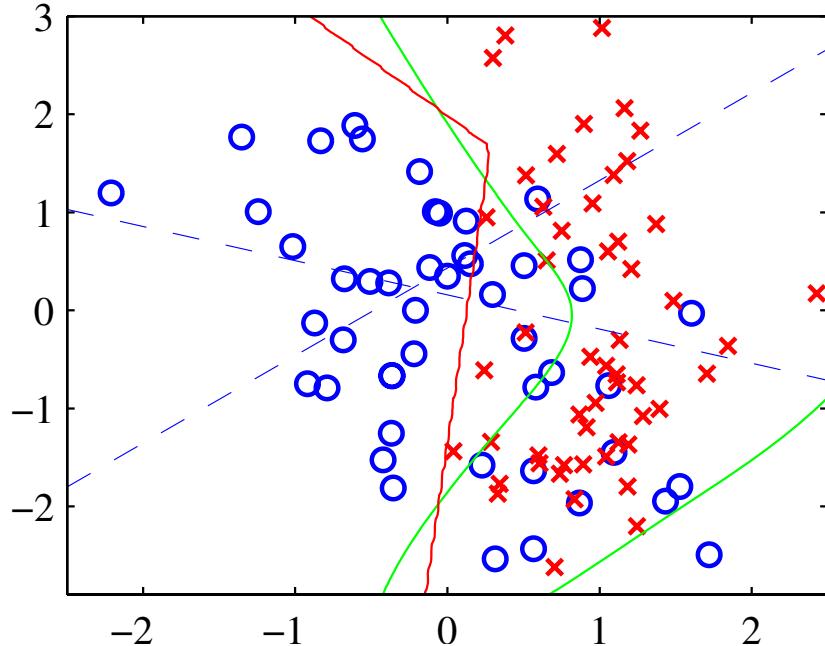
Quelques remarques

- Le nombre d'unités dans la couche caché est adaptable.
- Plusieurs couches cachées sont possibles. Nous traiterons les type *feed-forward* : absence de cycles dans les connexions.
- Les fonctions de sorties sont dérivables par rapport aux paramètres du réseau.
- Si les fonctions d'activations sont linéaires, alors un réseau équivalent peut être trouvé sans couche cachée.
- Les réseaux de neurones peuvent approximer n'importe quelle fonction à une précision arbitraire si la quantité d'unités est suffisante.



Un exemple

Example of the solution of a simple two-class classification problem involving synthetic data using a neural network having two inputs, two hidden units with ‘tanh’ activation functions, and a single output having a logistic sigmoid activation function. The dashed blue lines show the $z = 0.5$ contours for each of the hidden units, and the red line shows the $y = 0.5$ decision surface for the network. For comparison, the green line denotes the optimal decision boundary computed from the distributions used to generate the data.



[C. Bishop, Pattern recognition and Machine learning, 2006]

Apprentissage : descente de gradient

La fonction d'erreur est celle de la régression logistique :

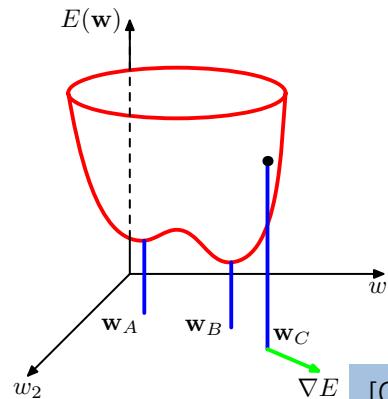
$$E(\mathbf{w}) = - \sum_{n=1}^N \sum_{k=1}^K t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

Minimisation itérative par **descente de gradient** (un pas dans la direction du plus grand changement) :

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

Vitesse (*Learning rate*)

Possibilité de blocage dans un minimum local :



Deux stratégies : batch vs. en ligne

Batch : toutes les données sont utilisées pour chaque étape

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w})$$

En ligne (stochastic gradient descent) : le gradient est calculé pour un seul point (une seule donnée) à chaque itération :

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$

Calcul du gradient

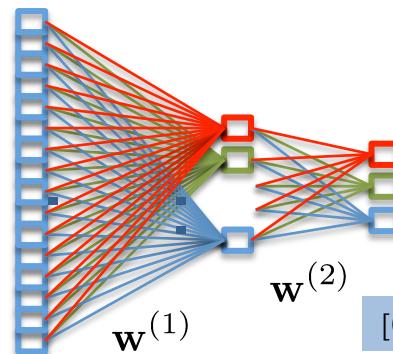
La minimisation de l'erreur nécessite le calcul de son gradient

$$\nabla E_n(\mathbf{W}) = \left[\frac{\partial E_n}{\partial w_{ij}^{(l)}} \right]_{ijl}$$

Chaque dérivée partielle pourrait (en théorie) se calculer par une différence finie :

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji} - \epsilon)}{2\epsilon} + O(\epsilon^2)$$

Problème : efficacité très mauvaise. Pour chaque donnée, pour W paramètres (poids), nous avons 2^*W « stimulations » à faire, chacune demandant W calculs $\rightarrow O(W^2)$



Retro-propagation du gradient (1)

Il est possible de calculer le gradient de manière directe.

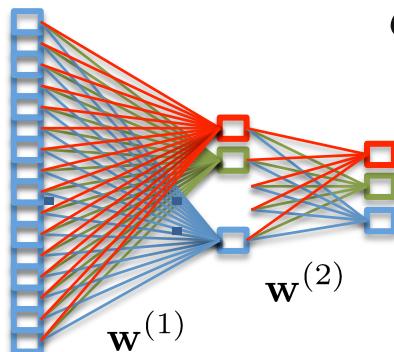
Exemple très simple : fonction d'activation linéaire (à la sortie), erreur de type « somme des carrés » :

$$y_k = \sum_i w_{ki} x_i$$

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

Pour les poids de la couche de sortie, on peut donner les dérivées directement :

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$



Retro-propagation du gradient (2)

Il manque les dérivées par rapport aux paramètres des couches cachées. Rappelons le calcul fait par une unité caché :

$$a_j = \sum_i w_{ji} z_i \quad z_j = h(a_j)$$

Pour simplifier, on calculera d'abord les dérivées par rapport à a_j

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

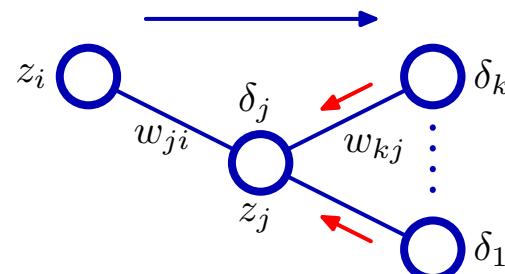
Pour la couche de sortie :

$$\delta_k = y_k - t_k$$

Chaque dérivée peut être calculée à partir des dérivées de la couche suivante :

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

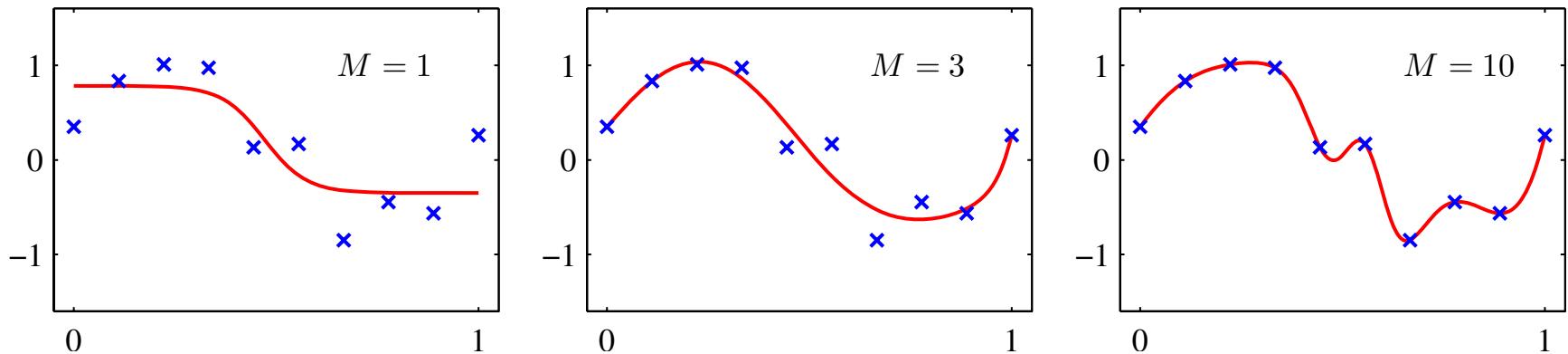
$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$



Retour : généralisation, sélection de modèles

Complexité du modèle de prédiction :

- Nombre de couches cachées
- Nombre d'unités cachées par couche



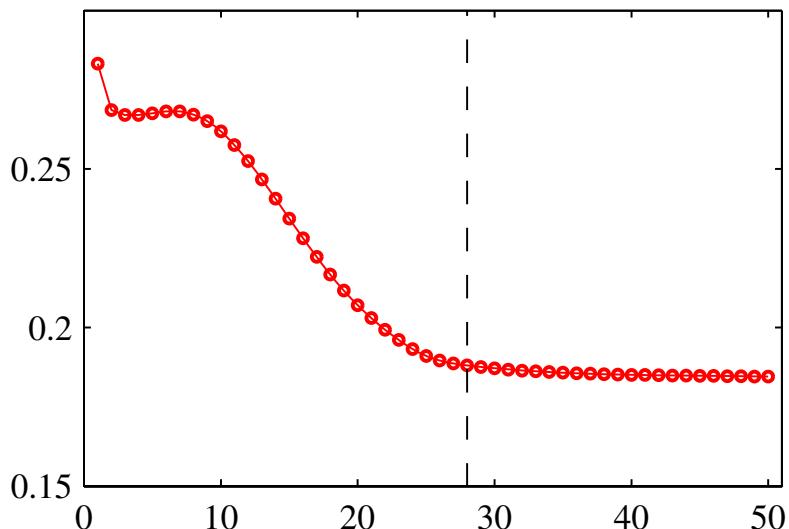
1 couche cachée avec M unités

Données générées avec $t = \sin(2\pi x)$

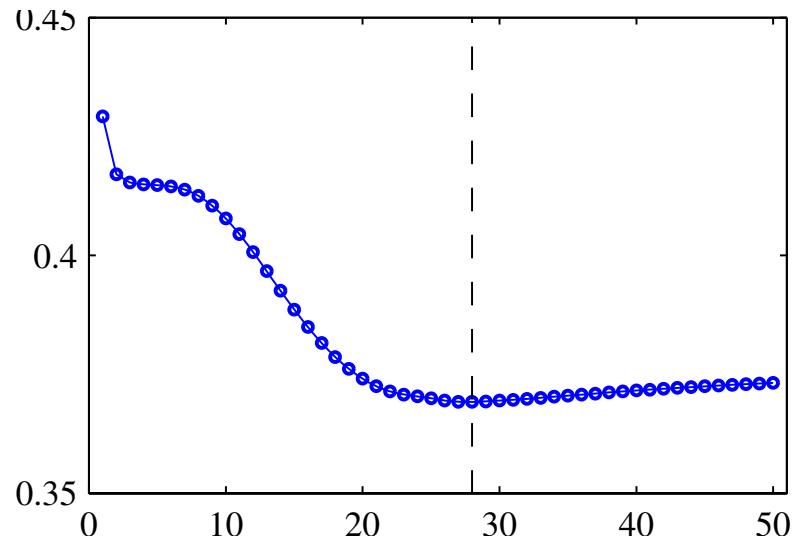
Arrêt prématué (*early stopping*)

L'apprentissage est itérative (1 itération est appelée 1 « *epoch* »).

On peut diminuer le sur apprentissage (*overfitting*) en arrêtant l'apprentissage lorsque l'erreur sur la base de validation commence à augmenter.



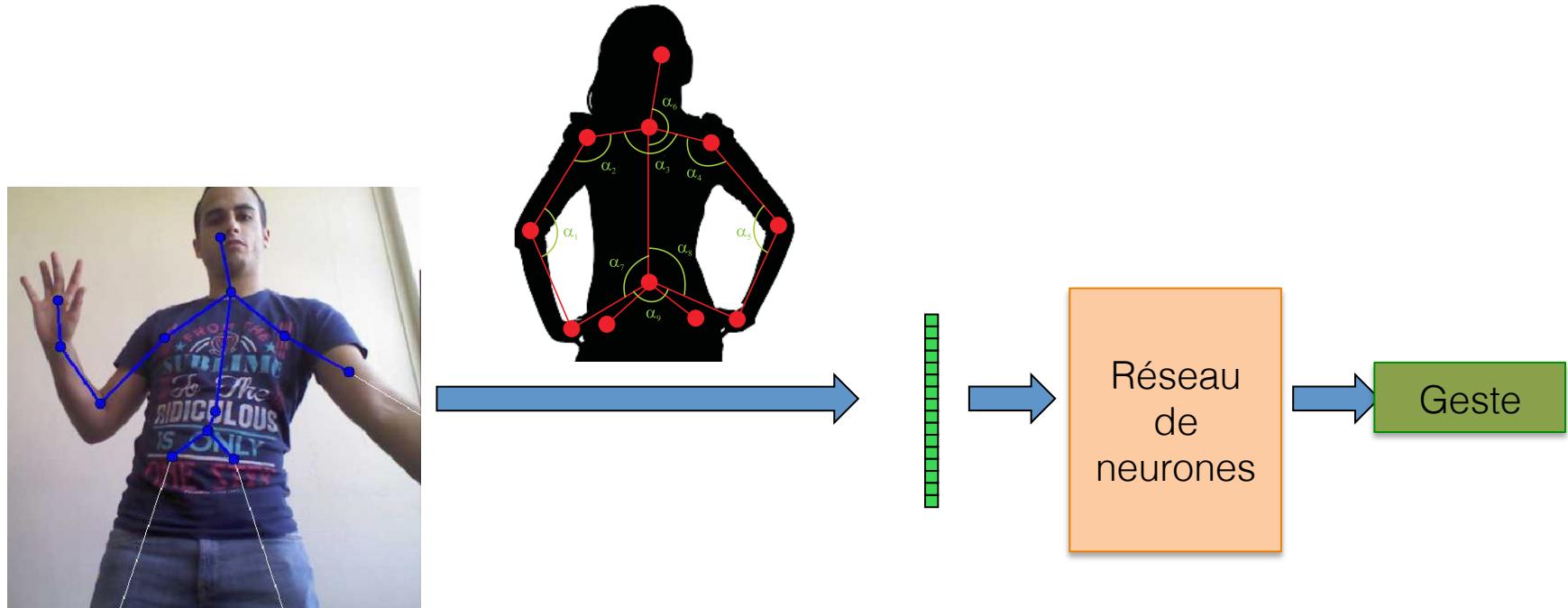
Erreur sur la base d'apprentissage



Erreur sur la base de validation

Application : reconnaissance de gestes

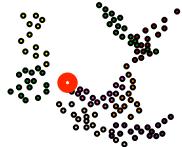
Classification de gestes à partir de la pose articulée d'un humain
(positions des articulations estimées à l'aide d'un capteur Kinect).



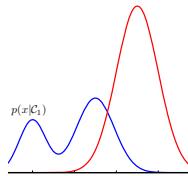
Poses consécutives
(Positions des articulations)

Caractéristiques
invariantes (point de vue,
personnes etc.)

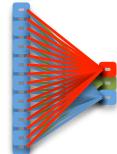
La classification supervisée



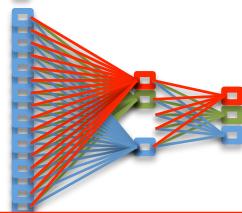
kppV (k plus proches voisins)



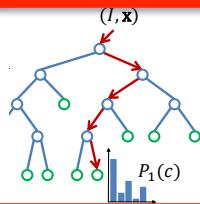
Classificateurs génératifs linéaires
(Bayésiens)



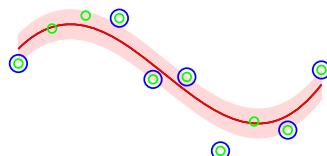
Classificateurs discriminatifs linéaires



Réseaux de neurones



Forêts aléatoires



SVM (*Support Vector Machines*)

Bases : théorie de l'information

Entropie $h(x)$ d'une variable aléatoire x : mesure du contenu d'information.

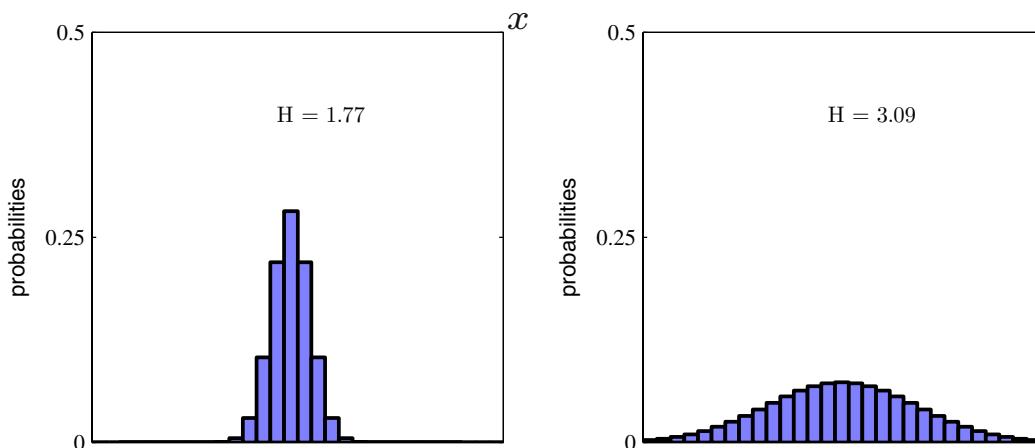
Peut être dérivé de deux variables indépendantes :

$$h(x, y) = h(x) + h(y).$$

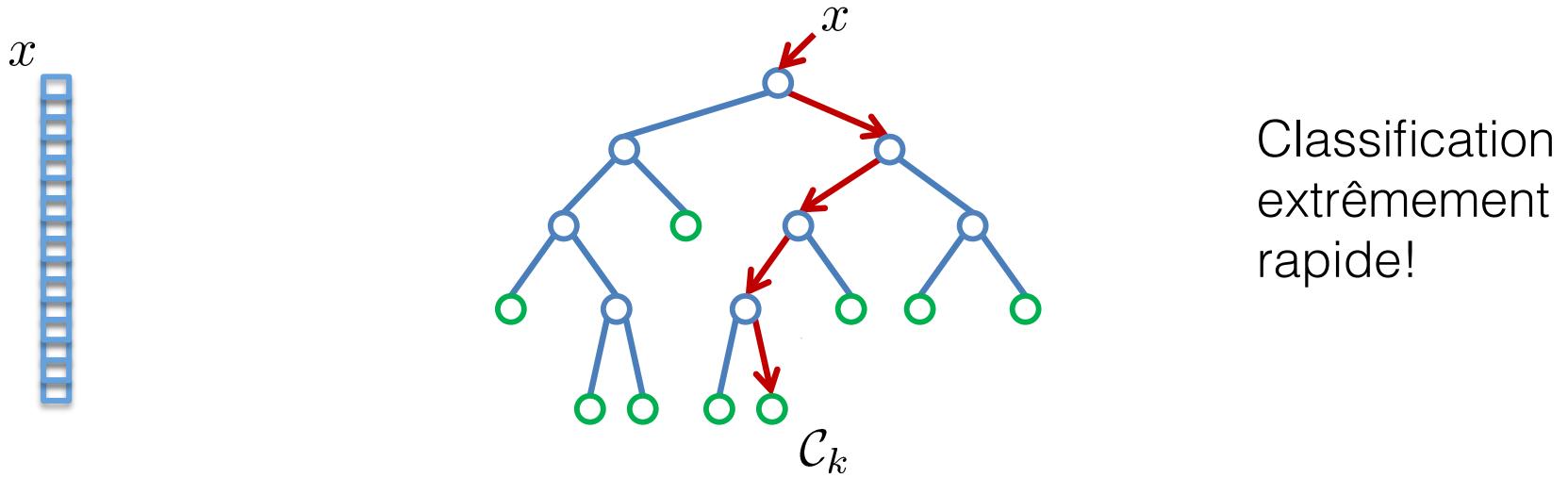
$$p(x, y) = p(x)p(y)$$

$$h(x) = -\log_2 p(x)$$

$$H[x] = - \sum p(x) \log_2 p(x).$$

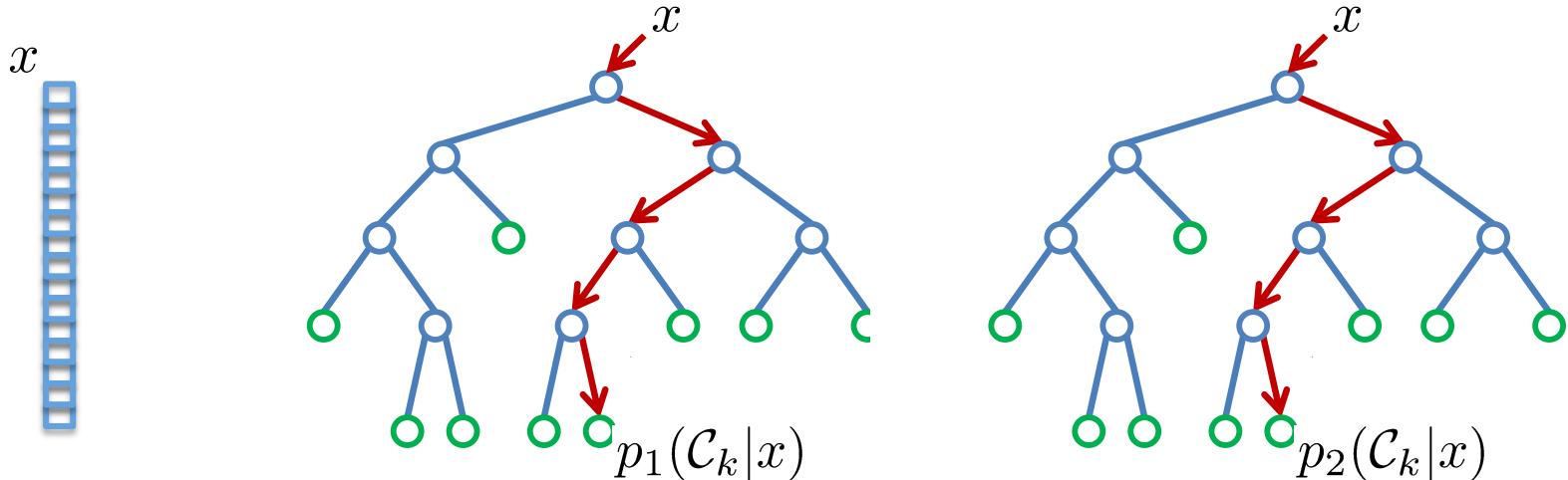


Arbres de décision



- À chaque nœud d'un arbre, une valeur du vecteur est examinée
- Parcours à gauche ou droite selon comparaison avec un seuil (différent pour chaque sommet)
- Chaque feuille contient une décision (une classe \mathcal{C}_k)
- Paramètres :
 - Choix des éléments x_i pour chaque sommet
 - Seuil pour chaque sommet
 - \mathcal{C}_k pour chaque feuille

Forêts aléatoires



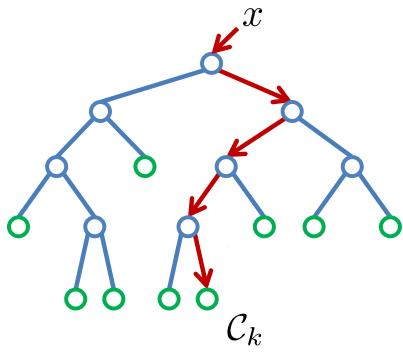
Plusieurs arbres sont entraînés sur des sous-ensembles différents des données
Chaque feuille contient une distribution d'étiquettes

$$p_j(\mathcal{C}_k|x)$$

Addition des distributions correspondantes :

$$p(\mathcal{C}_k|x) = \sum_j p_j(\mathcal{C}_k|x)$$

Apprentissage des paramètres



Chaque arbre est appris de manière indépendante
Apprentissage couche par couche ➔ le gradient de l'erreur n'est pas disponible!!

1. Pour un couche donnée, proposition d'un ensemble de candidats ϕ de paramètres (choix élément x_i , seuil τ)
2. Séparation des données d'apprentissage en 2 parties

$$\begin{aligned} Q_l(\phi) &= \{x \mid f(x_i) < \tau\} \\ Q_r(\phi) &= Q \setminus Q_l(\phi) \end{aligned}$$

3. Choix des paramètres candidats ϕ maximisant le gain en information (mesure d'entropie)

$$H(Q) - \sum_{s \in \{l, r\}} \frac{|Q_s(\phi)|}{|Q|} H(Q_s(\phi))$$

$H(Q)$... Shannon entropie

4. Récursion

Application : estimation de la pose humaine (MS Kinect)



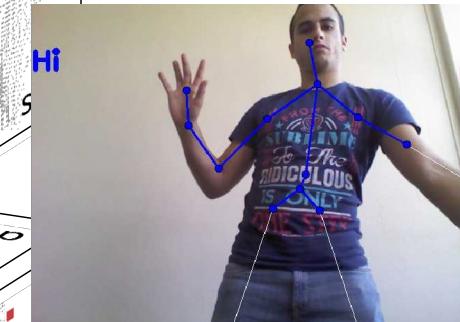
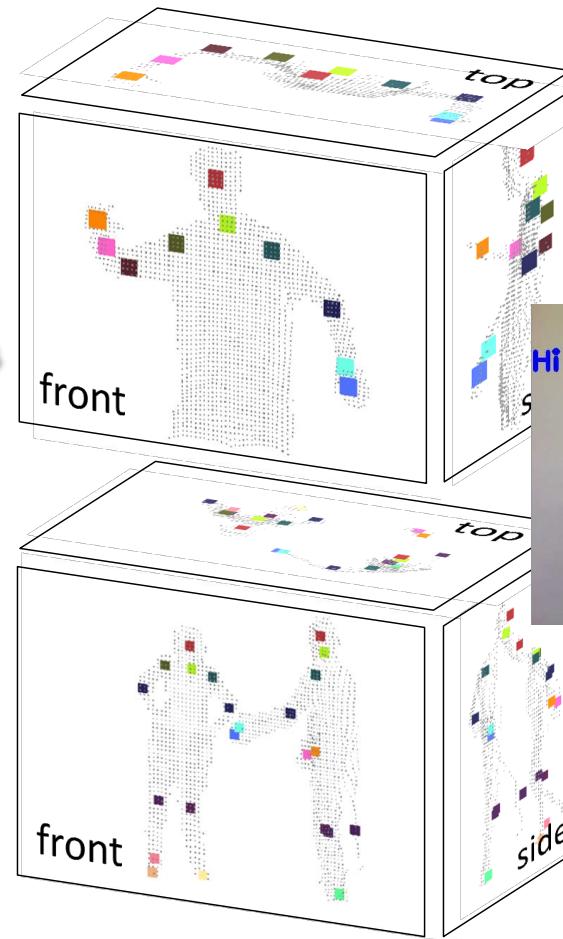
depth image



body parts



3D joint proposals



[J. Shotton et al., 2011]

Données d'apprentissage



31 parties du corps humains

1 Million d'images synthétiques issues d'une pipeline de rendu (la moitié à partir d'une capture de mouvements humains)

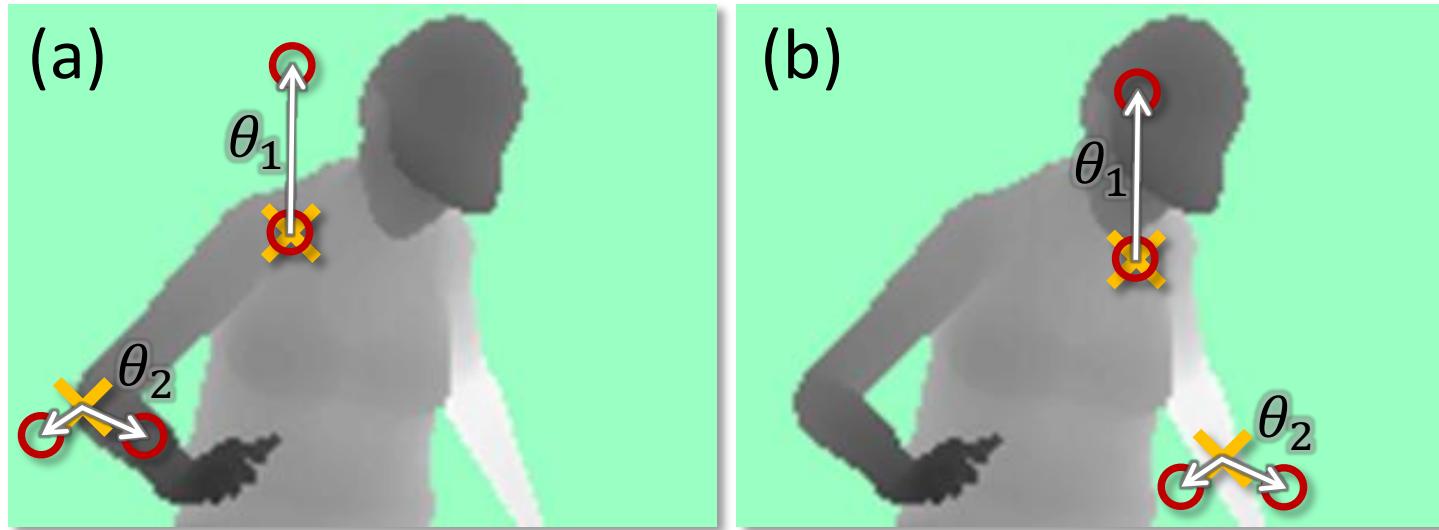
2000 pixels de chaque image = 2.000.000.000 de vecteurs d'apprentissage

3 arbres, chacun de 20 niveaux

Chaque feuille : 1 histogramme de 31 valeurs = 32.505.856 valeurs

$$2^{20}-1 = 1.048.575 \text{ seuils}$$

Les caractéristiques



$$f_{\theta}(I, \mathbf{x}) = d_I \left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})} \right) - d_I \left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})} \right)$$

\mathbf{x}

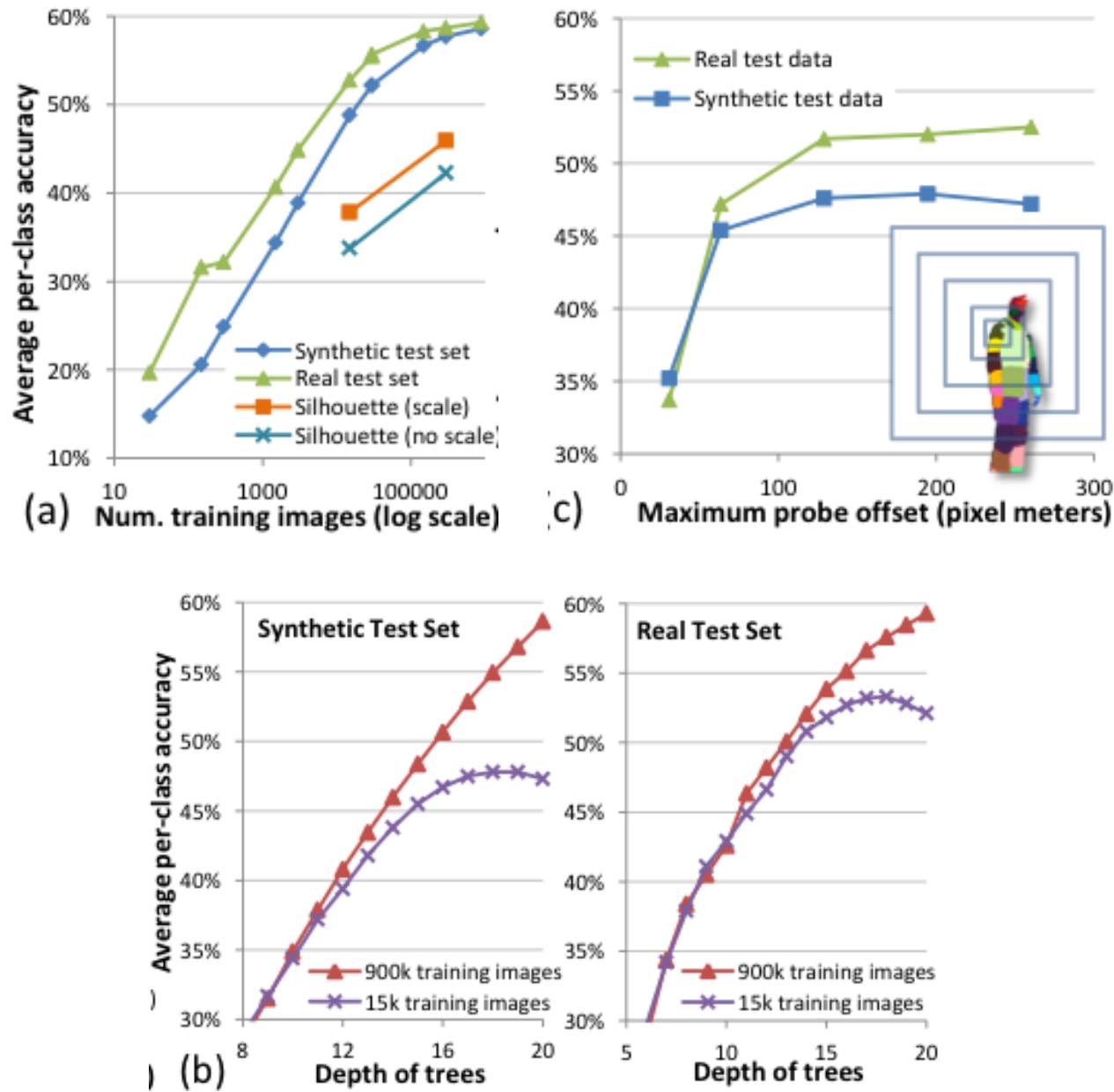
position du pixel,

\mathbf{u}, \mathbf{v} ...

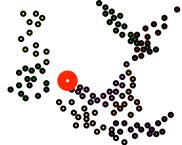
offsets

d_I

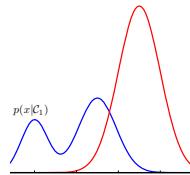
profondeur



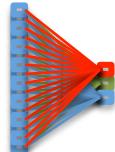
La classification supervisée



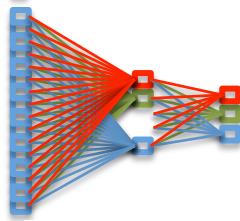
kppV (k plus proches voisins)



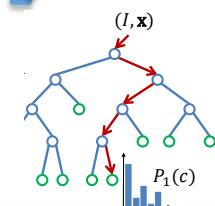
Classificateurs génératifs linéaires
(Bayésiens)



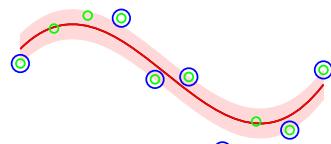
Classificateurs discriminatifs linéaires



Réseaux de neurones



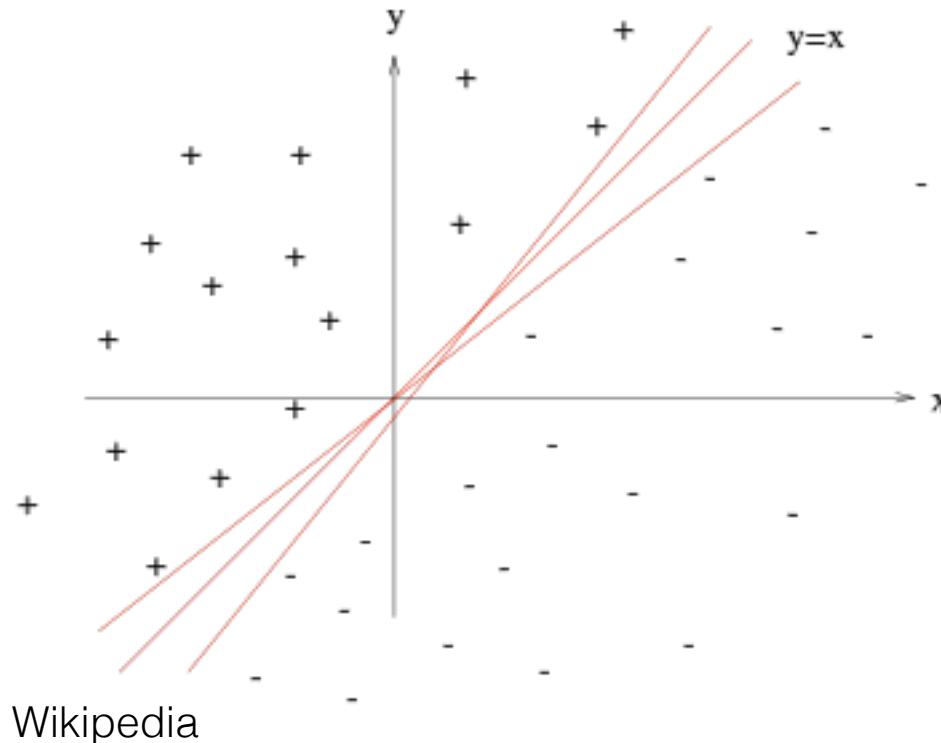
Forêts aléatoires



SVM (*Support Vector Machines*)

SVM (« Support Vector Machines »)

L'hypothèse de départ est que les données sont linéairement séparables (cette hypothèse sera partiellement relâchée)

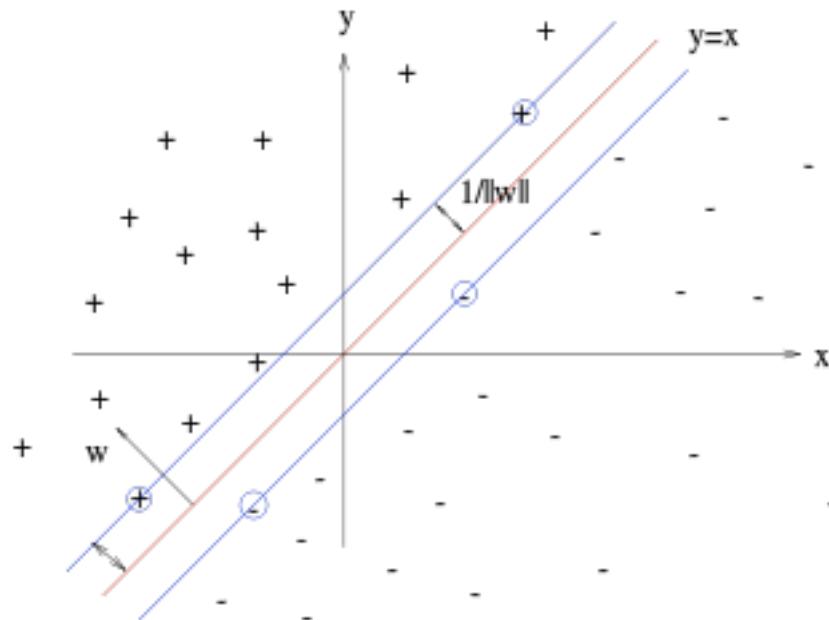


Pour un ensemble de points linéairement séparables, il existe une infinité d'hyperplans séparateurs

SVM : la marge

On cherche obtenir l'hyperplan ayant la marge maximale (distance entre les échantillons de classes différentes).

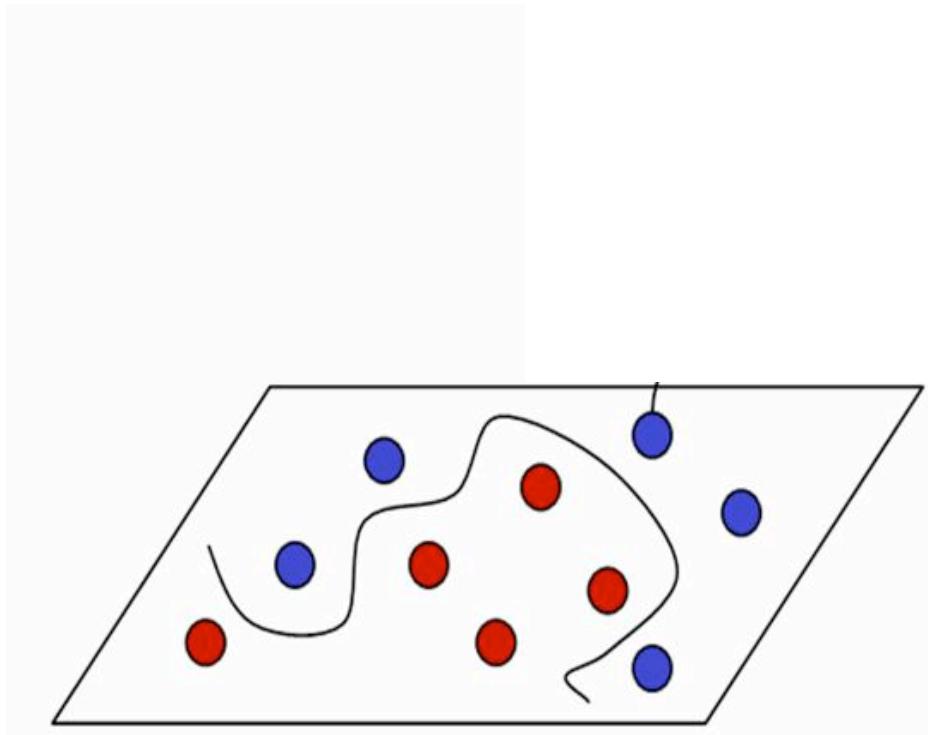
Objectif : réduire l'erreur sur les données non vues.



SVM : « *kernel trick* »

Certaines données ne sont pas linéairement séparables.

Astuce : projection des données dans un espace où elles le sont.



SVM : applications

Les SVM ont été utilisées pour quasiment toutes les applications en image / vision par ordinateur.

Quelques applications développées au LIRIS :



Reconnaissance d'actions individuelles

[Wolf et Taylor, 2010]



Reconnaissance d'actions collectives

[Baccouche/Mamalet/Wolf/Garcia/Baskurt, 2012]

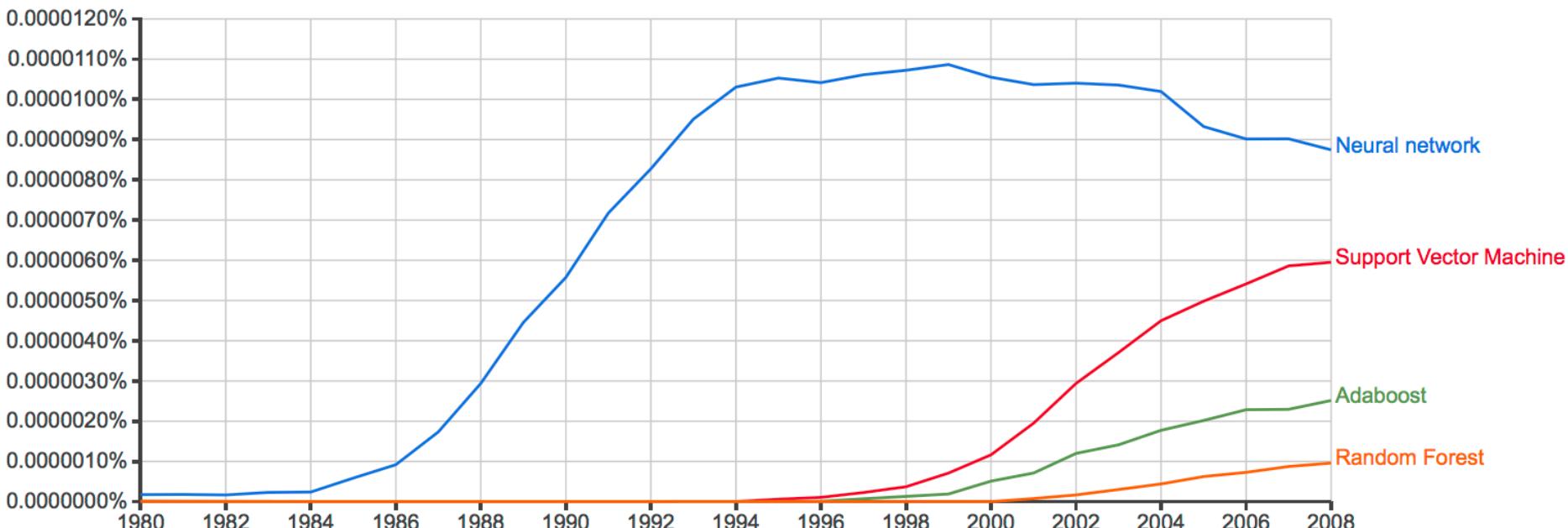


Reconnaissance de textes

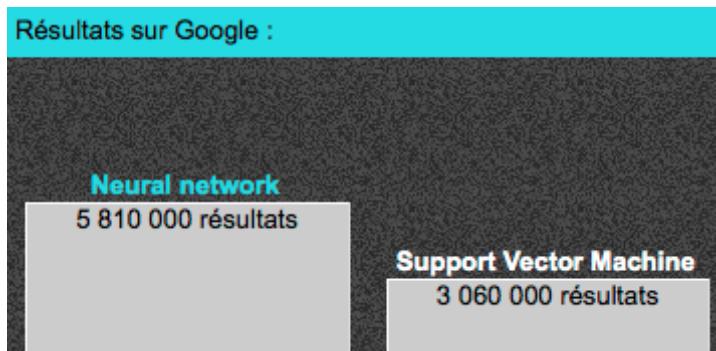
[Wolf et Jolion, 2003]

Popularité des classifieurs

Google N-gram viewer, requête le 24.7.2014 :



Résultats sur Google :



Google fight
(24.7.2014)