IFT3395/6390
Fondements de l'apprentissage machine

L'astuce du noyau
(kernel trick)

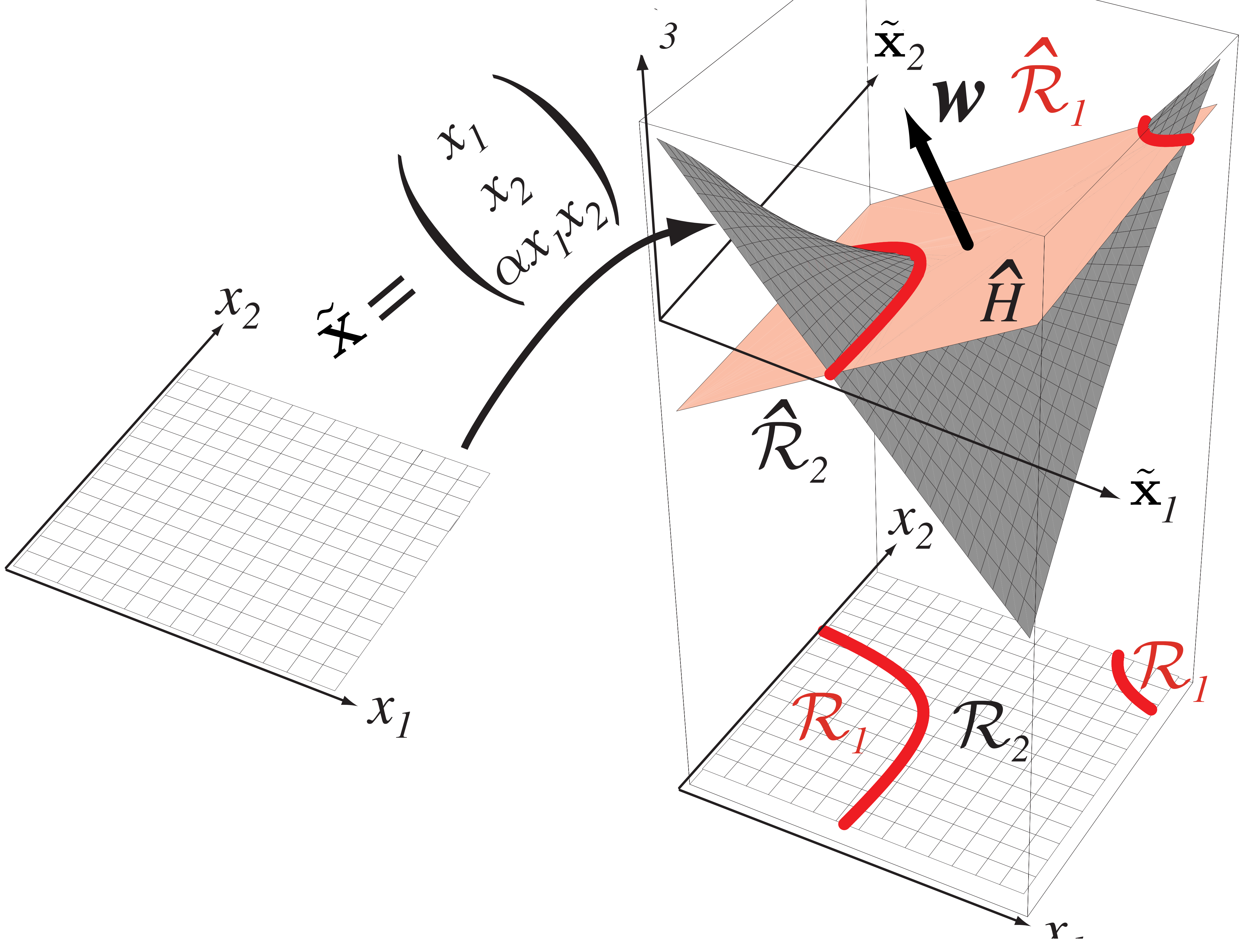
Professeur: Pascal Vincent

LISA Laboratoire d'Informatique des Systèmes d'Apprentissage

Rappel

- Il existe de nombreuses méthodes pour construire un classifieur linéaire. (Ex: l'algorithme du Perceptron, la régression logistique, SVM.)
- Nous avons vu qu'il est possible d'obtenir des classifieurs non linéaires à partir de classifieurs linéaires, avec un simple prétraitement des données.
- Il suffit d'appliquer une transformation non-linéaire *(mapping)* φ aux points de données x qui les transforme dans un espace de plus haute dimension en $\tilde{\mathbf{x}} = \varphi(\mathbf{x})$

Ex. transformation non-linéaire à priori (pré-traitement)



Pour obtenir un classifieur non-linéaire

Il est possible de:

- Utiliser une fonction φ **explicite choisie à priori**, et de calculer explicitement les $\tilde{\mathbf{x}} = \varphi(\mathbf{x})$

Ex: $\varphi : \underbrace{(x_{[1]}, x_{[2]})}_{\mathbf{x}} \mapsto \underbrace{(1, x_{[1]}, x_{[2]}, x_{[1]}x_{[2]}, x_{[1]}^2, x_{[2]}^2, \sin x_{[1]}, \cos x_{[2]})}_{\tilde{\mathbf{x}}}$

- **Apprendre une transformation** non-linéaire φ ayant une forme particulière. On peut voir sous cet angle ce que fait la première couche cachée d'un réseau de neurones.

Ex: $\tilde{\mathbf{x}} = \varphi(\mathbf{x}) = \text{sigmoid}(\mathbf{W}\mathbf{x} + \mathbf{b})$

- Utiliser l'**astuce du noyau** *(kernel trick)*

Problème avec un *mapping* explicite

- Si x est en haute dimension, un *mapping* polynômial mêne rapidement à calculer un $\tilde{\mathrm{x}}$ dans un espace gigantesque.
- Ex: $\mathbf{x} \in \mathbb{R}^d$ et *mapping* polynômial de degré *k* (tous les produits entre *k* éléments de x), on doit calculer un $\tilde{\mathrm{x}}$ dans un espace de dimension de l'ordre de d^k. Ex: d=100, k=5 donne 10 000 000 000

L'astuce du noyau

- S'applique à tout algorithme qui peut s'exprimer sous forme d'une expression basée sur des produits scalaires entre des vecteurs observations d'entrée.
- L'astuce est de supposer qu'on peut calculer le produit scalaire $\langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle$ directement sans jamais avoir à calculer explicitement un $\varphi(\mathbf{x})$
- On choisit un noyau K tel que

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle$$

Exemple

$$\varphi : \underbrace{(x_{[1]},\ x_{[2]})}_{\mathbf{x}} \quad \mapsto \quad (x_{[1]}^2,\ \sqrt{2}\, x_{[1]} x_{[2]},\ x_{[2]}^2)$$

$$\begin{aligned}
K(\mathbf{x}, \mathbf{y}) &= \langle \varphi(\mathbf{x}),\ \varphi(\mathbf{y}) \rangle \\
&= \left\langle (x_{[1]}^2,\ \sqrt{2}\, x_{[1]} x_{[2]},\ x_{[2]}^2), (y_{[1]}^2,\ \sqrt{2}\, y_{[1]} y_{[2]},\ y_{[2]}^2) \right\rangle \\
&= x_{[1]}^2 y_{[1]}^2 + \sqrt{2} x_{[1]} x_{[2]} \sqrt{2} y_{[1]} y_{[2]} + x_{[2]}^2 y_{[2]}^2 \\
&= x_{[1]}^2 y_{[1]}^2 + 2 x_{[1]} x_{[2]} y_{[1]} y_{[2]} + x_{[2]}^2 y_{[2]}^2 \\
&= (x_{[1]} y_{[1]} + x_{[2]} y_{[2]})^2 \\
&= (\langle \mathbf{x}, \mathbf{y} \rangle)^2
\end{aligned}$$

Il est possible de calculer le produit scalaire des vecteurs mappés sans jamais calculer le mapping explicitement !

Terminologie

- On appellera l'espace de départ dans lequel se trouvent les x, *l'espace de départ* ou l'*espace des x* ou l'*espace des entrées brutes* (*raw input space*)
- L'espace dans lequel φ *mappe* les donnée sera appelé *espace-φ* ou *espace des traits* (*feature space*)
- K correspond à un produit scalaire dans l'*espace-φ*.

L'astuce du noyau en détail

- Un modèle linéaire va avoir la forme

$$\begin{aligned} g(\mathbf{x}) &= \underbrace{\mathbf{w}^T\mathbf{x}}_{\text{produit scalaire}} + b & \mathbf{x} &\in \mathbb{R}^d \\ g(\mathbf{x}) &= b + \underbrace{\langle \mathbf{w}, \mathbf{x} \rangle}_{\text{produit scalaire}} & \mathbf{w} &\in \mathbb{R}^d, b \in \mathbb{R} \end{aligned}$$

- Pour beaucoup d'algorithmes d'apprentissages **w** va à tout moment être une combinaison linéaire d'exemples d'entrée:

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

On peut ainsi représenter $\mathbf{w}$ implicitement en retenant les $\alpha_i \in \mathbb{R}$ (remarque: plusieurs α_i peuvent être nuls).

- Ex: le Perceptron va trouver **w** en partant d'un vecteur nul et en y ajoutant ou soustrayant des vecteurs proportionnels à des exemples de l'ensemble d'entraînement (les $\mathbf{x}_i$).

L'astuce du noyau en détail

En particulier, si on travaille avec des vecteurs mappés dans l'espace φ: $\tilde{\mathbf{x}} = \varphi(\mathbf{x})$, on peut représenter implicitement $\mathbf{w}$ (potentiellement en très haute dimension) par:

$$\tilde{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i \tilde{\mathbf{x}}_i = \sum_{i=1}^{n} \alpha_i \varphi(\mathbf{x}_i)$$

auquel cas on peut calculer

$$\begin{aligned}
g(\tilde{\mathbf{x}}) &= b + \langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}} \rangle \\
&= b + \langle \tilde{\mathbf{w}}, \varphi(\mathbf{x}) \rangle \\
&= b + \left\langle \left(\sum_{i=1}^{n} \alpha_i \varphi(\mathbf{x}_i) \right), \varphi(\mathbf{x}) \right\rangle \\
&= b + \sum_{i=1}^{n} \alpha_i \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}) \rangle \\
&= b + \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x})
\end{aligned}$$

Puisqu'il suffit de calculer un produit scalaire entre des points mappés, on peut le faire avec un noyau K sans jamais mapper les points explicitement.

Attention!

$$\tilde{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i \tilde{\mathbf{x}}_i = \sum_{i=1}^{n} \alpha_i \varphi(\mathbf{x}_i) \neq \varphi\left(\sum_{i=1}^{n} \alpha_i \mathbf{x}_i\right)$$

Donc il ne s'agit **pas** simplement d'exécuter un algorithme dans l'espace de départ des x, et d'appliquer φ au vecteur de poids trouvé.

Il faut vraiment implicitement exécuter l'algorithme sur les $\varphi(x)$.

L'astuce du noyau en résumé

- Exprimer un algorithme dans l'*espace-φ* de manière qu'il n'utilise que des produits scalaires entre points d'entrée.
- Généralement cela implique de garder la trace d'un vecteur α de pondération des exemples d'entrée utilisés par l'algorithme.
- Remplacer ces produits scalaires par un noyau *K*.
- Ainsi on peut exécuter l'algorithme sans jamais avoir à projeter les points explicitement dans l'*espace-φ*.

Noyaux fréquemment utilisés

- Le produit scalaire usuel: $K(a,b) = \langle a, b \rangle$
- Noyau polynômial de degré k: $K_k(a,b) = (1 + \langle a, b \rangle)^k$
- Noyau RBF (ou Gaussien): $K_\sigma(a,b) = e^{-\frac{1}{2}\frac{\|a-b\|^2}{\sigma^2}}$

Remarques:

- Il existe de nombreux autres noyaux utiles.
- Les noyaux peuvent comporter des (hyper)-paramètres. ex: σ ou k
- On ne peut pas toujours exprimer explicitement la fonction φ correspondant à un noyau K donné. Ainsi le noyau RBF correspond à un *espace-φ* de dimension infinie.

Propriétés qu’un noyau doit respecter

- Pour qu’il existe un *espace-φ* et une fonction φ correspondant à un noyau K, il suffit que K soit un *“noyau de Mercer”*

- Pour satisfaire le *théorème de Mercer,* un noyau $K(a,b)$ doit être continu, symmétrique et semi défini positif.

  - diagonaliser la matrice de Gram: $\mathbf{G}_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{V}\Lambda\mathbf{V}$
  - valeurs propres: λ_t, vecteurs propres: $\mathbf{v}_t$
  - condition suffisante: $\mathbf{G}$ est positif semi-definit $(\forall t : \lambda_t > 0)$ pour tout $X_n = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$
  - D peut être ∞!!!

Algorithmes auxquels l'astuce du noyau s'applique

L'astuce du noyau peut s'appliquer à de nombreux algorithmes produisant une projection linéaire, pour en faire des versions non-linéaires dites "à noyau" ou "kernelisées" notamment:

classification

- Le Perceptron => perceptron à noyau (Kernel Perceptron)
- Les Machines à Vecteurs de Support (SVM)
- La régression logistique

régression

- La régression linéaire
- la régression de ridge
- variante des Machines à Vecteurs de Support pour la régression
- régression linéaire Bayesienne
  => Régresseur à Processus Gaussien (Kriging en géostats).

non supervisé

- L'Analyse en Composantes Principales (PCA)