

IFT6390

Fondements de l'apprentissage machine

Kernel Trick

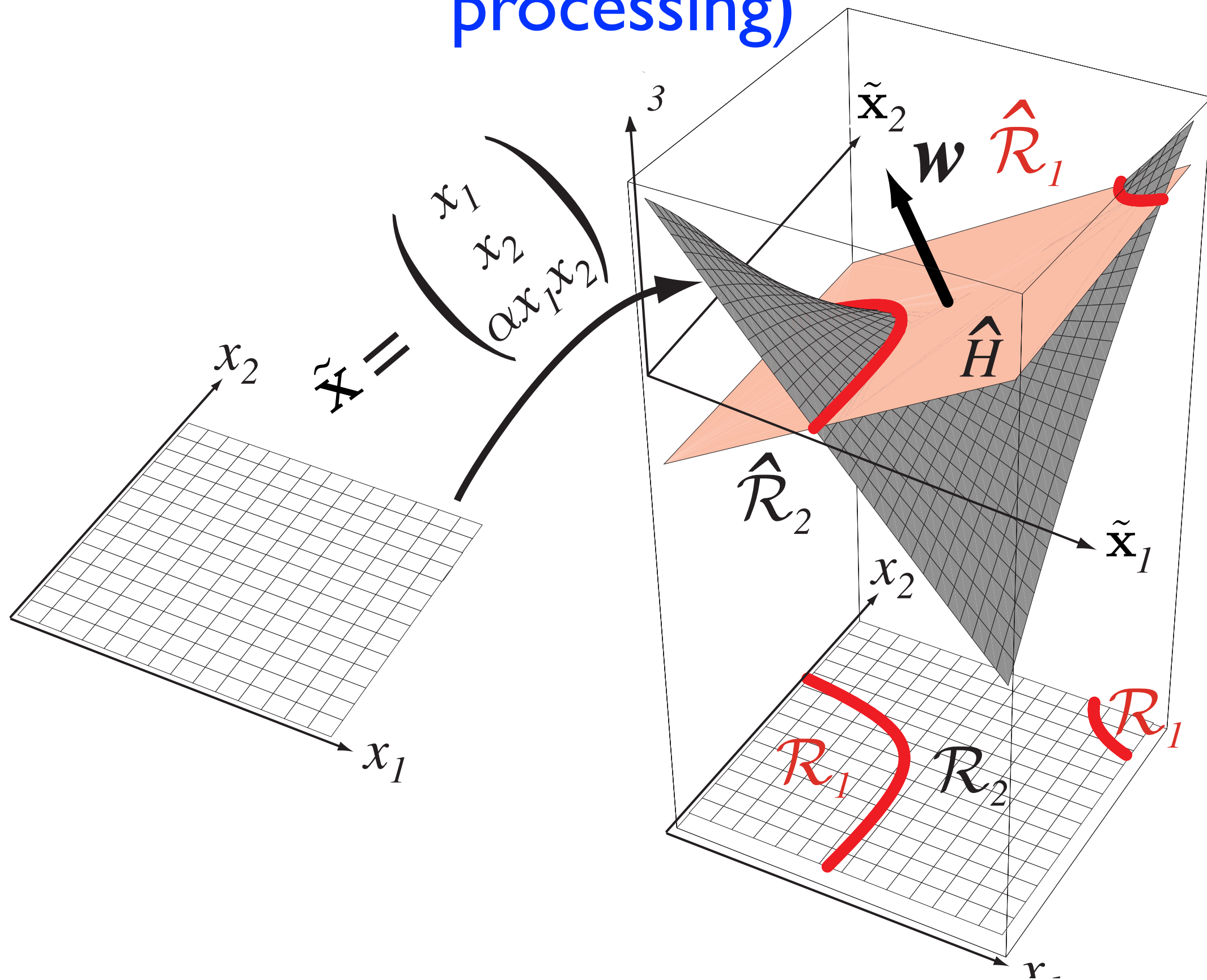
Professor: Ioannis Mitliagkas
Slides: Pascal Vincent

Reminder

- There is a lot of ways to learn a linear classifier (e.g. Perceptron, logistic regression, SVM.)
- We saw that we can obtain a non-linear classifier using a linear classifier by simply pre-processing the input data.
- We just need to apply a non φ to data points x to project them into a feature space of higher dimension:

$$\tilde{\mathbf{x}} = \varphi(\mathbf{x})$$

Ex. *a priori* linear transform (pre-processing)



To obtain a non-linear classifier

We can:

- Use a mapping φ **that we explicitly choose a priori** and explicitly compute each $\tilde{\mathbf{x}} = \varphi(\mathbf{x})$

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

- **Learn a non-linear mapping** φ within some parametrized class of functions. Neural networks can be seen from this perspective (e.g. first layer computes φ)

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

- Use the **kernel trick**

What's bad with *explicitly choosing the mapping?*

- If x is already of high dimension, a polynomial mapping will lead to computing \tilde{x} in a space of very high dimension.
- Ex: $x \in \mathbb{R}^d$ and *polynomial mapping* of degree k (all products within k components of x), we need to compute \tilde{x} in space of dimension $\sim d^k$.

Ex: $d=100, k=5 \longrightarrow 10\,000\,000\,000$ (100^5)

*That's
huge*

The kernel trick

- Can be applied to any learning algorithm that can be expressed in terms of dot products between input points.
- The trick consists in supposing that we can compute the dot product $\langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle$ directly without having to explicitly compute $\varphi(\mathbf{x})$.
- We choose a kernel K satisfying

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

Example

2D

3D

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

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

We can compute dot products in the new feature space
without having to explicitly use the feature map!

Terminology

- We call the input space where the original inputs x belong the *starting/original/raw input space*.
- The space where φ maps the data is *feature space*)
- The kernel K corresponds to a dot *product in the feature space*.

Kernel trick: details

- A linear model takes the form

$$\begin{aligned} g(\mathbf{x}) &= \underset{\text{dot product}}{\mathbf{w}^T \mathbf{x}} + b & \mathbf{x} &\in \mathbb{R}^d \\ g(\mathbf{x}) &= b + \underset{\text{dot product}}{\langle \mathbf{w}, \mathbf{x} \rangle} & \mathbf{w} &\in \mathbb{R}^d, b \in \mathbb{R} \end{aligned}$$

- For many learning algorithms \mathbf{w} will always be a **linear combination of input points**:

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

Hence \mathbf{w} can be implicitly represented by using the scalars α_i
(sometimes, most of them will be 0, e.g. in SVM)

- Ex: Perceptron find \mathbf{w} starting from 0 and adding/subtracting vectors that are collinear to input points from the training set (the \mathbf{x}_i 's).

Kernel trick: details

In particular, if we work with input points mapped in the feature space $\tilde{\mathbf{x}} = \Phi(\mathbf{x})$, we can implicitly represent \mathbf{w} (potentially of high dimension) with

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

In which case we can compute

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

Since we only need to compute **dot products between points in the feature space**, we can use the kernel K without ever having to explicitly use the feature map!

! Warning !

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

So you **cannot** simply run the algorithm in the original space, and then apply φ to the weight vector returned by the algorithm!

You need to *implicitly* run the algorithm on the points $\varphi(x)$ (in the feature space).

Kernel trick: summary

- Express an algorithm in the *feature space (corresponding to some feature map ϕ)* using only dot products between input points.
- In general, this will imply keeping track of some weight vector α used by the algorithm (for a linear combination of input points)
- Replace all dot products with a **kernel function K** .
- This corresponds to running the algorithm in the feature space without ever having to explicitly compute the mappings with the feature map ϕ .

Common kernel functions

- Usual scalar product: $K(a, b) = \langle a, b \rangle$
- Polynomial kernel of degree k : $K_k(a, b) = (1 + \langle a, b \rangle)^k$
- RBF (or Gaussian) kernel: $K_\sigma(a, b) = e^{-\frac{1}{2} \frac{\|a-b\|^2}{\sigma^2}}$

Remarks:

- There are a lot of other useful kernels
- There are also kernels on strings, trees, graphs...
- Kernels can have (hyper)-parameters. ex: σ ou k
- The feature map φ associated with a kernel K cannot always be expressed in a nice analytical form: the RBF kernel corresponds to a feature space of infinite dimension!

What it takes to be a kernel...

- Not all functions $K(\cdot, \cdot)$ are kernels: there is not always a mapping φ such that $K(a, b) = \langle \varphi(a), \varphi(b) \rangle$
- *Mercer Theorem*: This will be the case if, and only if, K is continuous, symmetric and positive semi-definite.

Kernel-trick friendly algorithms:

Many learning algorithms that usually return a linear classifier/regressor can be kernelized. Among others:

classification

- Perceptron => Kernel Perceptron
- Support Vector Machines (SVM)
- Logistic Regression

regression

- Linear Regression
- Ridge Regression
- Support Vector Regression (SVM variant for regression)
- Bayesian Linear Regression => Gaussian Processes (also Kriging in stats).

unsupervised

- Principal Component Analysis (PCA)