Aprenentatge Automàtic 2

GCED

Lluís A. Belanche

belanche@cs.upc.edu

Soft Computing Research Group
Dept. de Ciències de la Computació (Computer Science)

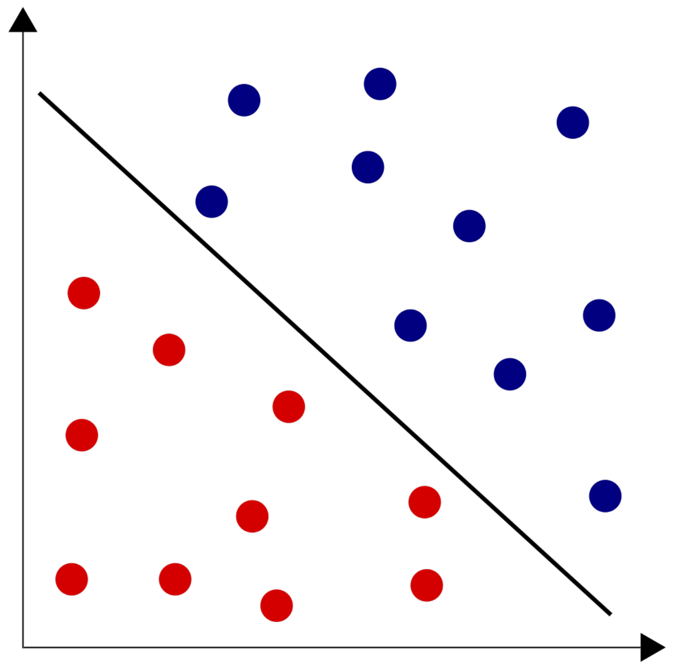
Universitat Politècnica de Catalunya

2021-2022

LECTURE 2: The Perceptron and the kernel Perceptron. Ridge regression and kernel ridge regression

Kernel-based learning

Departing scenario



Two classes, no distributional or *a priori* assumptions, linear separability ("linsep")

The goal is to **generalize** well to unseen data (as well as possible, topped by the Bayes error)

The Perceptron

Formalisation

We have a data set $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, with $\boldsymbol{x}_i \in \mathbb{R}^m$ and $y_i \in \{-1, +1\}$, describing a two-class problem.

We wish to find a linear function f which best models D:

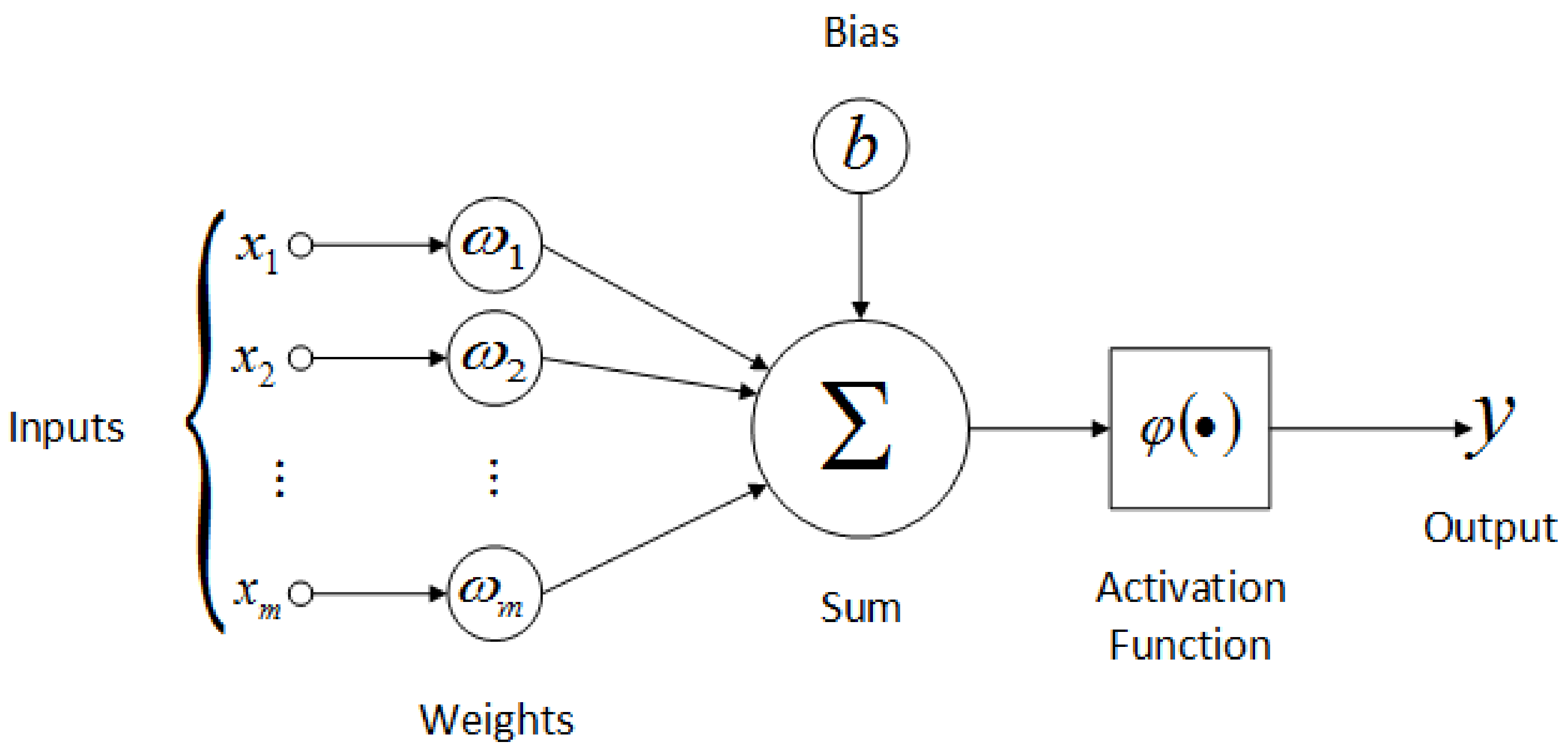
- Set up an **affine function** $g(\boldsymbol{x}) := \boldsymbol{\omega}^\top \boldsymbol{x} + b$
- Obtain a **linear discriminant** as $f(\boldsymbol{x}) := \mathrm{sgn}(g(\boldsymbol{x}))$
- We would like to find an iterative learning procedure for $\boldsymbol{\omega}, b$ such that:

  $\boldsymbol{\omega}^\top \boldsymbol{x}_i + b > 0$, when $y_i = +1$
  $\boldsymbol{\omega}^\top \boldsymbol{x}_i + b < 0$, when $y_i = -1$

  that is $y_i(\boldsymbol{\omega}^\top \boldsymbol{x}_i + b) > 0$ or simply $y_i\, g(\boldsymbol{x}_i) > 0 \qquad (1 \leq i \leq n)$

The Perceptron

Formalisation



Interpretation as a neural network with $m + 1$ trainable weights, where $\varphi(\cdot) = \text{sgn}(\cdot)$.

The Perceptron

Formalisation

Definition 1 *The functional margin γ_i^F of an example $(\boldsymbol{x}_i, y_i)$ wrt an hyperplane $\boldsymbol{\omega}, b$ is*

$$\gamma_i^F := y_i g(\boldsymbol{x}_i) = y_i(\boldsymbol{\omega}^\top \boldsymbol{x}_i + b)$$

We note that $\gamma_i^F > 0$ iff $(\boldsymbol{x}_i, y_i)$ is correctly classified by the hyperplane.

Definition 2 *The geometric margin γ_i^G of an example $(\boldsymbol{x}_i, y_i)$ is the functional margin wrt the hyperplane $\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}, \frac{b}{\|\boldsymbol{\omega}\|}$.*

We note that $\gamma_i^G = d(\boldsymbol{x}_i, \pi)$, where π stands for the hyperplane
$\pi : \boldsymbol{\omega}^\top \boldsymbol{x} + b = 0$. ¶

The Perceptron

Definition 3 *The margin* Γ *of a dataset* D *is the maximum geometric margin over all possible hyperplanes:*

$$\Gamma(D) := \sup_{(\boldsymbol{\omega},b)\in\mathbb{R}^m\times\mathbb{R}} \min_{i=1,\dots,n} \gamma_i^G(\boldsymbol{\omega}, b)$$

Any hyperplane such that:

1. *it realises* Γ *on* D
2. *all its functional margins* $\gamma_i^F > 0$

is known as a **maximum margin hyperplane***.*

We first set for the goal of learning a hyperplane satisfying 2.

The Perceptron

Formalisation

Definition 4 *The Perceptron cost function is defined as:*

$$J(\boldsymbol{\omega}, b) := -\sum_{i/\boldsymbol{x}_i \in M} y_i(\boldsymbol{\omega}^\top \boldsymbol{x}_i + b) = -\sum_{i/\boldsymbol{x}_i \in M} \gamma_i^F$$

being M the set of missclassified examples.

Proposition 1 ¶

1. $J \geq 0$; *moreover* $M = \emptyset \Rightarrow J = 0$

2. J *is continuous and piece-wise linear*

The Perceptron

Formalisation

Assuming M contains the set of missclassified examples by $\boldsymbol{\omega}(t), b(t)$, a gradient descent updating rule leads (¶) to Rosenblatt's famous **Perceptron learning algorithm**:

```
initialize:      Choose $\boldsymbol{\omega}(0), b(0)$ and $\eta > 0$
                 $t := 0$; $R := \max_i \|\boldsymbol{x}_i\|$
                 repeat
                         For $i = 1, \ldots, n$
                             Predict $\widehat{y}_i(t) := \mathrm{sgn}(y_i(\boldsymbol{\omega}^\top(t)\boldsymbol{x}_i + b(t)))$
correct mistake:             if $\widehat{y}_i(t) \neq y_i$ then
                               $\boldsymbol{\omega}(t+1) := \boldsymbol{\omega}(t) + \eta y_i \boldsymbol{x}_i$
                               $b(t+1) := b(t) + \eta y_i R^2$
                               $t := t + 1$
                             end
                 until no mistakes made within the for loop
result:          $\boldsymbol{\omega}(t), b(t)$
```

The Perceptron

Formalisation

Theorem 1 *(Novikoff'62; Block'62) Given a data set $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, with $\boldsymbol{x}_i \in \mathbb{R}^m$ and $y_i \in \{-1, +1\}$, for all $1 \leq i \leq n$, describing a two-class problem, and s.t. $\|\boldsymbol{x}_i\| \leq R$.*

Suppose there exists a vector $\boldsymbol{\omega}^*$ and a scalar b^* fulfilling $\|\boldsymbol{\omega}^*\| = 1$ and, for some μ, $y_i(\boldsymbol{\omega}^{*\top}\boldsymbol{x}_i + b^*) \geq \mu > 0$ holds.

Then the number of mistakes made by the Perceptron learning algorithm (until such a solution is found) is at most

$$4\left(\frac{1}{\mu} + \frac{R^2}{\mu^2}\right)$$

F. Rosenblatt, "The Perceptron–a perceiving and recognizing automaton", *Cornell Aeronautical Laboratory*, Report 85-460-1, 1957.

A. B. Novikoff, "On convergence proofs on perceptrons", *Symposium on the Mathematical Theory of Automata*, 12, 615-622, 1962.

The kernel Perceptron

Observations

The algorithm starts with an initial solution $\boldsymbol{\omega}(0), b(0)$ and iteratively updates it whenever a training data example is missclasified by the current solution $\boldsymbol{\omega}(t), b(t)$ until a global solution $\boldsymbol{\omega}^*, b^*$ is found.

The algorithm works simply by adding (subtracting) positive (negative) missclassified examples to the initial solution vector. In consequence:

$$\boldsymbol{\omega}^* = \boldsymbol{\omega}(0) + \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i$$

Without loss of generality we can assume that $\boldsymbol{\omega}(0) = \mathbf{0}$.

The α_i^* are equal to the number of times example i was involved in a correction ("difficult" points will tend to have higher α_i^*).

The kernel Perceptron

Formalisation

- This simple fact allows to derive an alternative "dual" version of the algorithm.
- We depart from a kernel k in $\mathbb{R}^m$.
- Then, invoking Aronszajn's theorem, we introduce a mapping $\phi : \mathbb{R}^m \to \mathcal{H}$, where $\mathcal{H}$ is a Hilbert space endowed with an inner product $\mathcal{H} \times \mathcal{H} \to \mathbb{R}$ s.t. $k(\boldsymbol{x}, \boldsymbol{x'}) := \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle_{\mathcal{H}}$.
- This in turn allows to obtain the **dual model** of f.
- The new vector of learned parameters $\boldsymbol{\alpha}^*$ may be found with the dual version of the learning algorithm.

The kernel Perceptron

Conclusions

1. The algorithm converges faster when the required margin (μ) is larger (if we could only specify this!)

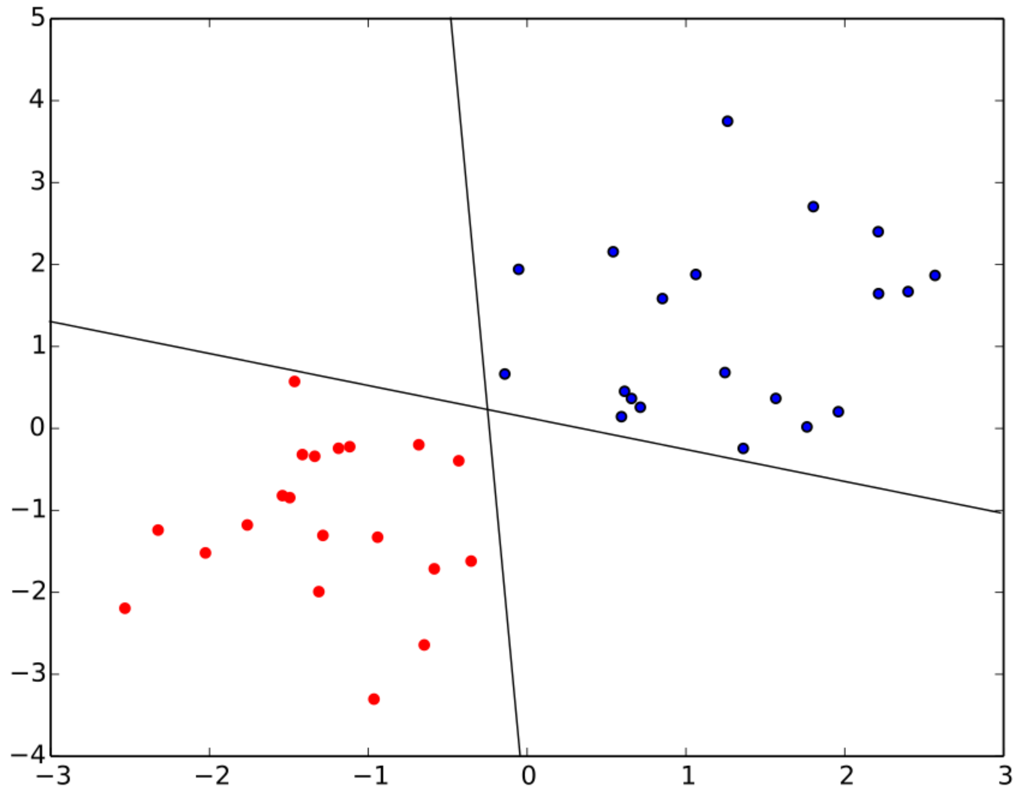
2. The bound on the number of corrections does not directly depend on the dimension of the space where the solution is being found.

3. We can relate the dual solution vector to a rough measure of problem hardness. ¶

4. It is not clear how the specify a required margin, and most important, if and when is there a “best” margin. ☹

The Perceptron

Formalisation



The problem of an infinite number of valid solutions ...

Kernel-based learning

Introduction

Problem: We wish to find a function $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$, for $\mathcal{X} = \mathbb{R}^m$ and $\mathcal{T} = \mathbb{R}$ which best models a data generation mechanism.

- We have a data sample $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}, \boldsymbol{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}$

- $(\boldsymbol{x}_i, y_i)$ are drawn i.i.d. from some unknown (joint) probability distribution

- In order to find a good model of the process $(\boldsymbol{x}, f_P(\boldsymbol{x}))$, we make the assumption that $f_P(\boldsymbol{x}) = f(\boldsymbol{x}; \boldsymbol{w}) + \epsilon$, where ϵ is a real-valued r.v. s.t. $\mathbb{E}[\epsilon] = 0, Var(\epsilon) = \sigma^2$ and ϵ and $\boldsymbol{x}$ are independent

Kernel-based learning

Introduction

- The **least squares** approach prescribes choosing the parameters $(\boldsymbol{w}, b) \in \mathbb{R}^m \times \mathbb{R}$ s.t.

$$(\widehat{\boldsymbol{w}}, \widehat{b}) := \min_{\boldsymbol{w},b} L(\boldsymbol{w}, b) := \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i - b)^2$$

- This procedure coincides with a maximum likelihood approach under the assumption $\epsilon \sim N(0, \sigma^2)$

- We rewrite to add one dimension as $\underline{\boldsymbol{x}}_i := \binom{\boldsymbol{x}_i}{1}$ and $\underline{\boldsymbol{w}}_i := \binom{\boldsymbol{w}_i}{b}$

- Call $\underline{\mathbf{X}}$ the matrix of the $\underline{\boldsymbol{x}}_i^\top$ and $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$

Kernel-based learning

Introduction

The least squares problem is now written:

$$\underset{\underline{\boldsymbol{w}} \in \mathbb{R}^{p+1}}{\text{mín}}\ L(\underline{\boldsymbol{w}}) := ||\boldsymbol{y} - \underline{\mathbf{X}}\underline{\boldsymbol{w}}||^2$$

Setting $\nabla_{\underline{\boldsymbol{w}}} L = 0$, we obtain the **normal equations**:

$$-2\underline{\mathbf{X}}^\top \boldsymbol{y} + 2\underline{\mathbf{X}}^\top \underline{\mathbf{X}}\underline{\boldsymbol{w}} = 0$$

with solution $\hat{\underline{\boldsymbol{w}}} = (\underline{\mathbf{X}}^\top \underline{\mathbf{X}})^{-1} \underline{\mathbf{X}}^\top \boldsymbol{y}$ ¶

and therefore $f(\boldsymbol{x}) = \hat{\underline{\boldsymbol{w}}}^\top \underline{\boldsymbol{x}} = \boldsymbol{y}^\top \underline{\mathbf{X}}(\underline{\mathbf{X}}^\top \underline{\mathbf{X}})^{-1} \underline{\boldsymbol{x}}$.

Kernel-based learning

Introduction

- If S is generated as $(\boldsymbol{x}, f_P(\boldsymbol{x}))$ for some f_P, the $\underline{\boldsymbol{x}}_i$ vectors are linearly independent and $n = p + 1$, then there is a unique solution for $\underline{\mathbf{X}}\underline{\boldsymbol{w}} = \boldsymbol{y}$ given by $\underline{\widehat{\boldsymbol{w}}} = \underline{\mathbf{X}}^{-1}\boldsymbol{y}$; in any other case, the problem is "ill-posed"

- Ridge regression was introduced in statistics by Hoerl and Kennard in 1970

- In the NNs community, it was proposed in the mid 60s in the form of an iterative procedure, known as the Widrow-Hoff learning rule

Kernel-Based Learning

Kernel ridge regression

A problem is **ill-posed** if the solution may not always exist, is not uniquely determined or is unstable (small variations in the initial conditions of the problem cause the solution to change quite a lot)

- A basic example would be the solution of linear systems of equations
- Learning problems are in general ill-posed
- The solution is to use an **inductive principle**; one of the most popular is the **regularization** principle

Kernel-based learning

Introduction

If the matrix $(\underline{\mathbf{X}}^\top \underline{\mathbf{X}})^{-1}$ does not exist (because $\underline{\mathbf{X}}^\top \underline{\mathbf{X}}$ is not full rank), or if it has a large condition number, or numerical stability problems occur because this matrix is close to singularity, we get into trouble ...

A very useful and general technique is to add a penalty term to the "size" of the model f. If we choose $||\boldsymbol{w}||^2$ as size we are developing a particular case of regularization known in statistics as **ridge regression**:

$$\min_{\underline{\boldsymbol{w}} \in \mathbb{R}^{p+1}} L_\lambda(\underline{\boldsymbol{w}}) := ||\boldsymbol{y} - \underline{\mathbf{X}}\underline{\boldsymbol{w}}||^2 + \lambda ||\boldsymbol{w}||^2, \qquad \lambda > 0$$

Kernel-based learning

Introduction

The general idea is to *stabilize* the optimization problem making it well-posed.

The λ hyper-parameter controls the trade-off between low loss and low norm of the solution

Setting again $\nabla_{\underline{\boldsymbol{w}}} L_\lambda = 0$, we obtain the solution

$$\underline{\widehat{\boldsymbol{w}}}_\lambda = (\underline{\mathbf{X}}^\top \underline{\mathbf{X}} + \lambda \mathbf{I}_p)^{-1} \underline{\mathbf{X}}^\top \boldsymbol{y}$$

where $\mathbf{I}_m$ is the $(m+1) \times (m+1)$ identity matrix with the $(m+1) \times (m+1)$ entry set to zero

and therefore $f(\boldsymbol{x}) = \underline{\widehat{\boldsymbol{w}}}^\top \underline{\boldsymbol{x}} = \boldsymbol{y}^\top \underline{\mathbf{X}} (\underline{\mathbf{X}}^\top \underline{\mathbf{X}} + \lambda \mathbf{I}_m)^{-1} \underline{\boldsymbol{x}}$.

Kernel-Based Learning

Ridge regression: primal representation

1. It can be shown that the matrix $(\underline{\mathbf{X}}^\top\underline{\mathbf{X}} + \lambda\mathbf{I}_m)^{-1}$ is PD, thus this inverse is well-defined ¶

2. Since $\underline{\mathbf{X}}$ is $n \times (m+1)$, the matrix $\underline{\mathbf{X}}^\top\underline{\mathbf{X}}$ is $(m+1) \times (m+1)$

3. The hyper-parameter λ is typically chosen by cross-validation

4. The model size does not grow with data size (a **parametric** model)

Kernel-based learning

Introduction

We now try to find a dual representation of ridge regression.

1. We depart from the necessary condition
$$\frac{\partial L_\lambda}{\partial \underline{\boldsymbol{w}}} = 0$$

2. This implies that
$$\lambda \underline{\boldsymbol{w}} = \underline{\mathbf{X}}^\top (\boldsymbol{y} - \underline{\mathbf{X}}\underline{\boldsymbol{w}})$$

3. Therefore, there exists a new parameter vector:
$$\boldsymbol{\alpha} = \frac{1}{\lambda}(\boldsymbol{y} - \underline{\mathbf{X}}\underline{\boldsymbol{w}})$$

¶

Kernel-Based Learning

Dual representation

It turns out that the regularized solution can also be written as:

$$\hat{\boldsymbol{w}} = \sum_{i=1}^{n} \hat{\alpha}_i \boldsymbol{x}_i$$

In consequence,

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \hat{\alpha}_i \, \boldsymbol{x}_i^\top \boldsymbol{x} \qquad \P$$

1. The new parameter vector $\hat{\boldsymbol{\alpha}} = (\hat{\alpha}_1, \ldots, \hat{\alpha}_n)^\top$ is $\hat{\boldsymbol{\alpha}} = (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_n)^{-1} \boldsymbol{y}$

2. The **Gram matrix** $\mathbf{X}\mathbf{X}^\top$ ("matrix of inner products") is $n \times n$

Kernel-Based Learning

Primal and dual

So we have the **primal** and the **dual** forms for $f(\boldsymbol{x})$:

$$f(\boldsymbol{x}) = \underline{\hat{\boldsymbol{w}}}_\lambda^\top \underline{\boldsymbol{x}} = \sum_{j=1}^{m+1} \underline{\hat{w}}_j \underline{x}_j \quad \text{and} \quad f(\boldsymbol{x}) = \sum_{i=1}^{n} \hat{\alpha}_i \, \boldsymbol{x}_i^\top \boldsymbol{x}$$

The dual form is more convenient when $m >> n$:

- The primal requires $O(nm^2 + m^3)$ operations
- The dual requires $O(mn^2 + n^3)$ operations

Kernel-Based Learning

How can we perform non-linear regression?

First create a *feature map*, a function $\phi : \mathbb{R}^m \to \mathbb{R}^D$, with $D \in \mathbb{N} \cup \{\infty\}$

$$\boldsymbol{x} \mapsto \phi(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \cdots, \phi_D(\boldsymbol{x}))^\top$$

- $\phi(\boldsymbol{x})$ is called a *feature vector*
- $\{\phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^d\}$ is the *feature space* (part of a Hilbert space, a larger vector space endowed with an **inner product** $\langle\, , \rangle$ whose associated norm defines a complete metric)
- As a technicality, we could easily add $\phi_0(\boldsymbol{x}) = 1$ as before, if desired

Kernel-Based Learning

The regression function has now the primal representation:

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \phi(\boldsymbol{x}) \rangle = \sum_{j=1}^{D} w_j \phi_j(\boldsymbol{x})$$

- This feature space still has the structure of a vector space

- In consequence, there is also the dual representation:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \widehat{\alpha}_i \, \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}) \rangle$$

(BTW, how general are all these results?)

Kernel-Based Learning

Feature maps and kernels (1)

Given a feature map $\phi : \mathbb{R}^m \to \mathcal{H}$, being $\mathcal{H}$ a Hilbert space, we define its associated **kernel function** $k : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ as:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle, \qquad \boldsymbol{x}, \boldsymbol{x'} \in \mathbb{R}^d$$

One key point is that, for some feature maps, computing $k(\boldsymbol{x}, \boldsymbol{x'})$ is independent of D (the dimension of $\mathcal{H}$)

$\rightarrow$ the "kernel trick"

Again the regularized solution can also be written as:

$$\hat{\boldsymbol{w}} = \sum_{i=1}^{n} \widehat{\alpha}_i \phi_i(\boldsymbol{x})$$

Kernel-Based Learning

Feature maps and kernels (2)

Our regression function has now the dual representation:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \widehat{\alpha}_i \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}) \right\rangle = \sum_{i=1}^{n} \widehat{\alpha}_i k(\boldsymbol{x}_i, \boldsymbol{x})$$

This dual representation:

- ... is a **non-linear model** (in the input space)
- ... is a **linear model** (in the feature space)

We then have a **non-parametric model** (complexity grows with data size)

Kernel-Based Learning

Back to ridge regression ...

The new vector of parameters $\widehat{\boldsymbol{\alpha}} = (\widehat{\alpha}_1, \ldots, \widehat{\alpha}_n)^\top$ is now given by

$$\widehat{\boldsymbol{\alpha}} = (\mathbf{K} + \lambda \mathbf{I}_n)^{-1} \boldsymbol{y},$$

where $\mathbf{I}_n$ is the $n \times n$ identity matrix and $\mathbf{K} = (k_{ij})$, with $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

[Q] What is f now? ¶

So we can do ridge regression based only on $\mathbf{K}$ (and throw away $\mathbf{X}$)

$\rightarrow$ this is called **Kernel ridge regression** (kRR)

Kernel-Based Learning

Kernel ridge regression

What if we take the (simplest) choices $\phi(\boldsymbol{x}) = \boldsymbol{x}$ for $\boldsymbol{x}$ in $\mathbb{R}^m$? In this case $m = D$ and $k(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{x}^\top \boldsymbol{x'}$. The regularized solution reads:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \widehat{\alpha}_i \boldsymbol{x}_i^\top \boldsymbol{x}$$

where

$$\widehat{\boldsymbol{\alpha}} = (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_n)^{-1} \boldsymbol{t},$$

(so $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$ in this particularly simple case)

This means we have generalized (the dual of) standard ridge regression via a kernel function (we have **kernelized** ridge regression)

Kernel-Based Learning

Kernel ridge regression

Pros and Cons:

1. We can use many kernel functions, or a single one and tune its hyper-parameter(s) ☺

2. There is no sparsity ☹

3. The computational complexity is rather high ☹

4. It is an elegant example of the power of kernel functions ☻