Kernel-Based Learning & Multivariate Modeling

MIRI Master

Lluís A. Belanche
belanche@cs.upc.edu

Soft Computing Research Group

Universitat Politècnica de Catalunya

2021-2022

Kernel-Based Learning & Multivariate Modeling

Syllabus

Part 1

Sep 15 Introduction to kernel-based learning
Sep 22 The SVM for classification, regression & novelty detection (I)
Sep 29 The SVM for classification, regression & novelty detection (II)
Oct 06 Kernel design (I): theoretical issues
Oct 13 Kernel design (II): practical issues
Oct 20 Kernelizing ML & statistical algorithms
Oct 27 Advanced topics

Kernel-Based Learning

Introduction

Desiderata for satisfactory learning methods (my particular view):

Robustness to outliers, errors and/or wrong model assumptions
tolerance

Efficiency in the computational sense (necessary to handle large datasets)
complexity, fast languages, use of supercomputers etc

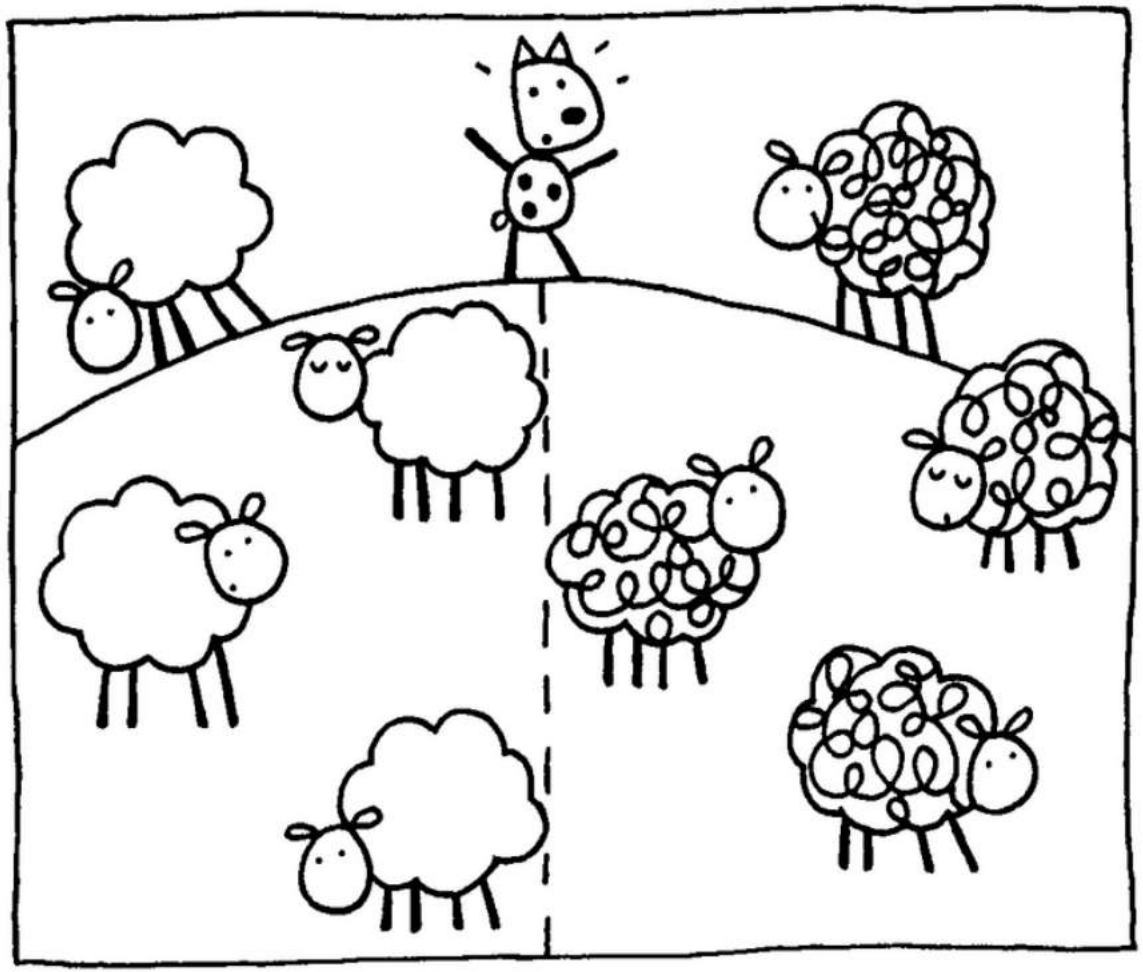
Flexibility to perform different tasks

Controlable non-linearity to deliver complexity surplus and accept explicit complexity control

Versatility to accept different data types and incorporate prior knowledge

$\Longrightarrow$ **Generalize** well to unseen data (as well as possible)

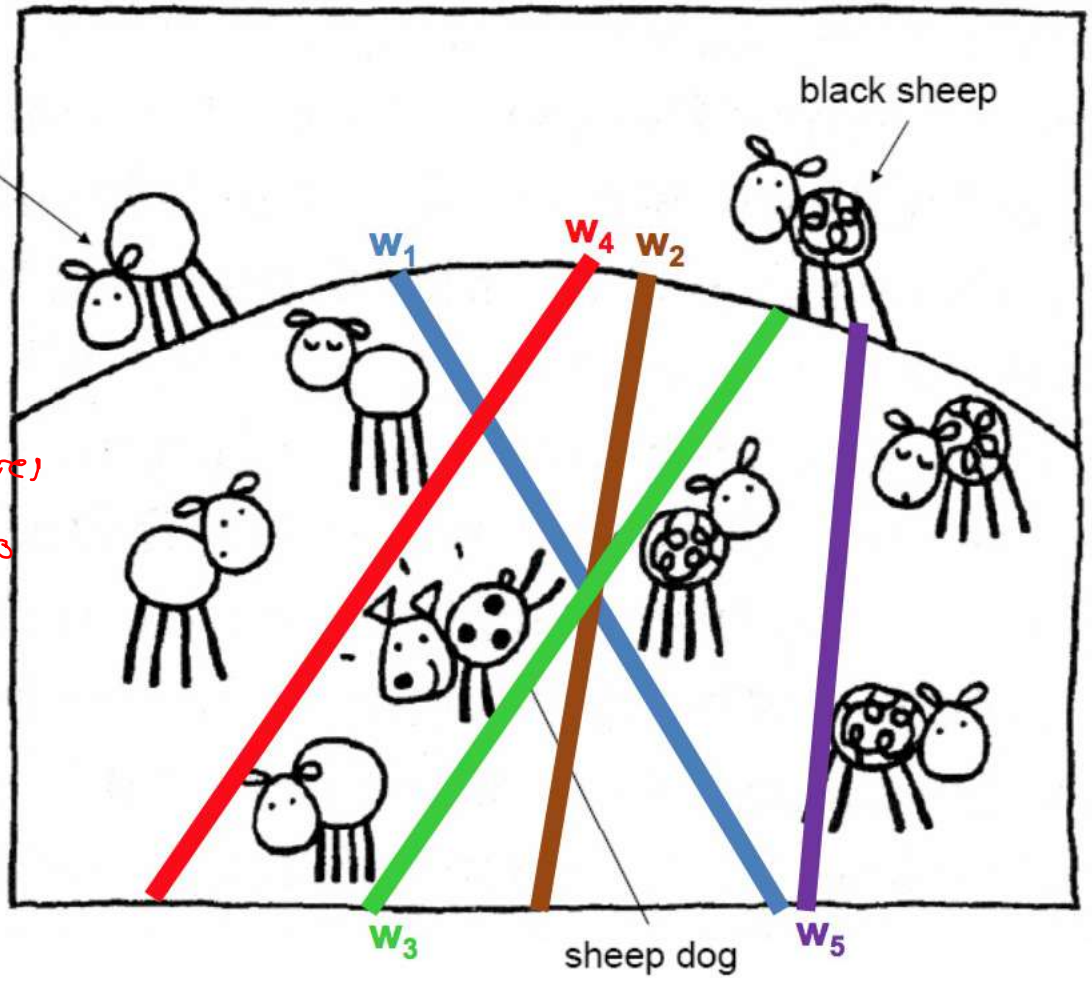
Kernel-Based Learning



Kernel-Based Learning

calculation is only estimation and classifier is random because it depends on sample

SVMs are systems that add info
↳ use hypothesis of function w/ many features
↳ trained w/ opt algo w/ assumptions from statistical learning



add info to solve

"SVMs are regularized learning systems that leverage a hypothesis space of linear functions in a high dimensional feature space, and trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory."

set of all hypo

n-dim space w/ all att of data

assumptions

Kernel-Based Learning

Introduction

Kernel methods constitute a very powerful paradigm:

- provide a principled way to perform non linear, nonparametric learning;
- rely on solid functional analytic foundations and enjoy optimal statistical properties;
- they have been successfully applied to a wide range of data analysis problems (e.g. in computational biology);
- are specially well suited for high-dimensional data, noisy environments or heterogenous information (unlike other approaches like neural networks) but small data sets

Kernel-Based Learning

Introduction

- the number of hyper-parameters is usually very small compared to other approaches (again unlike neural networks);

- their main limitations are:

  – their difficult applicability in large data scenarios because of stringent computational requirements in terms of time and (especially) memory;

  – the unability to extract increasingly abstract features from the data ("single hidden layer", unlike, yes, neural networks!)

$\longrightarrow$ *Kernel methods are a central tool in machine learning*

Kernel-Based Learning

Key aspects of kernel methods

1. Input data is embedded (mapped) into a **vector space**

2. **Linear relations** are sought among the elements of this vector space

3. The coordinates of the images (mapped data) are *not* needed: only their **pairwise inner products**
   dot product: a1*b1+...+an*bn

4. Often these inner products can be computed (efficiently and implicitly) in the input space (via a **kernel function**) $\rightarrow$ the PROMISE

Kernel-Based Learning

Introduction

Kernel-based methods consist of two ingredients:

$$K = \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & \cdots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \cdots & k(x_2,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n,x_1) & k(x_n,x_2) & \cdots & k(x_n,x_n) \end{bmatrix}$$

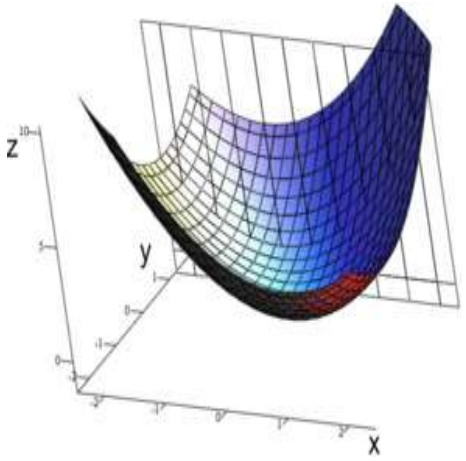
1. The (right) **kernel function** to convert input data into a similarity matrix

2. The (chosen) **learning algorithm** that uses the kernel matrix and the data to produce a model that performs a prescribed task

Kernel-Based Learning

Introduction

A kernel function is:



1. a special case of similarity measure;
   keep in mind: not all similarity measures are kernel functions
2. a positive semi-definite function; all eigenvalues ≥0
   $K(x,x) \geq 0$
3. a general inner product in a (special) Hilbert space

necessary to have only 1 optimum

It must be psd so that optimization problems can be convex and, therefore, *uniquely* solvable by standard algorithms (moreover, there is a *global* optimum)

Kernel-Based Learning

Introduction

- kernel functions can be used to analyze virtually any kind of data (and to express many forms of previous knowledge) without preprocessing the data

- kernel functions allow the obtention of more general (non-linear) versions of learning algorithms as long as they are base on inner products or Euclidean distances

Kernel-Based Learning

Introduction

Examples of **kernel functions**:

- RBF kernels
- Polynomial kernels
- String kernels
- Anova kernels
- Fisher kernels

Examples of (kernel) **learning algorithms**:

- Support Vector Machine (SVM) and Relevance Vector Machine (RVM)
- kernel Fisher Discriminant Analysis (FDA)
- kernel Principal Components analysis (PCA)
- kernel Canonical Correspondence analysis (CCA)
- kernel (regularized) regression (logistic, linear)
- kernel k-means

Kernel-Based Learning

Introduction

Designing/choosing an appropriate kernel function is not easy in general:

- Consider the **RBF kernel**:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2}\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{\sigma^2}\right), \quad \sigma^2 \in \mathbb{R}$$

- All information of a problem (besides the target values) is tunneled through the kernel matrix $\mathbf{K} = (k_{ij})$, where $k_{ij} := k(\boldsymbol{x}_i, \boldsymbol{x}_j)$

- if σ^2 is very small, then $\mathbf{K} \approx \mathbf{I}$ (all data are dissimilar): over-fitting

- if σ^2 is very large, then $\mathbf{K} \approx \mathbf{J}$ (all data are similar): under-fitting

Kernel-Based Learning

Notation

We have a data sample $S = \{(\bar{\boldsymbol{x}}_1, t_1), \ldots, (\bar{\boldsymbol{x}}_n, t_n)\}$

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

  independently & identically distributed

- $\bar{\boldsymbol{x}}_i \in \mathcal{X}, t_i \in \mathcal{T}$ (input space, output space)

Kernel-Based Learning

Linear regression

Problem: We wish to find a function $f(\bar{\boldsymbol{x}}) = \bar{\boldsymbol{w}}^\top \bar{\boldsymbol{x}}$ which best models a data set S for the case $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{T} = \mathbb{R}$

d = dimension

- If $f(\bar{\boldsymbol{x}}) = \boldsymbol{w}^\top \bar{\boldsymbol{x}} + w_0$, we rewrite to add one dimension as $\boldsymbol{x} := (1, \bar{\boldsymbol{x}}^\top)^\top$; $\boldsymbol{w} := (w_0, \bar{\boldsymbol{w}}^\top)^\top$

  ↙ transposed

  $w_0 \cdot 1 + w^\top x$

- Call $\mathbf{X}_{n\times(d+1)}$ the matrix of the $\boldsymbol{x}_i^\top$ and $\boldsymbol{t} = (t_1, \ldots, t_n)^\top$

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

data sample $S = \{(x, f(x))\}$ w. x_i lin. independent & $n = d+1$
$\Rightarrow$ $\exists$ unique sol $Xw = t$ by $\hat{w} = X^{-1} \cdot t$

Kernel-Based Learning

Ill-posed & well-posedness

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).

Jacques Hadamard (1865-1963)

- 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

$t_i = f(x_i) + \varepsilon_i$

$t_i = w^T x + x_0 + \varepsilon_i$

$\Leftrightarrow \varepsilon_i = t_i - w^T x - x_0$

not σ^2 from kernel!

This is only for regression

Kernel-Based Learning

Ridge regression

Under the standard assumptions $t_i = f(\boldsymbol{x}_i) + \varepsilon_i$ and $\varepsilon_i \sim N(0, \sigma^2)$, a maximum likelihood argument leads to the minimization of the regularized empirical error (a.k.a penalized likelihood):

unobserved var

Gaussian assump

bc Gaussian

well $\varepsilon()^2 = ()^T \cdot ()$

$$E_\lambda(\boldsymbol{w}) := \sum_{i=1}^{n} (t_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + \lambda \sum_{j=1}^{d} w_j^2 = (\boldsymbol{t} - \mathbf{X}\boldsymbol{w})^\top (\boldsymbol{t} - \mathbf{X}\boldsymbol{w}) + \lambda \bar{\boldsymbol{w}}^\top \bar{\boldsymbol{w}}$$

$$= \|t - Xw\|^2 + \lambda \|\bar{w}\|^2$$

1. Note $E_\lambda(\boldsymbol{w}) = \|\boldsymbol{t} - \mathbf{X}\boldsymbol{w}\|^2 + \lambda \|\bar{\boldsymbol{w}}\|^2$

2. The parameter $\lambda > 0$ defines a trade-off between the fit to the data and the complexity of the model (2-norm of $\bar{\boldsymbol{w}}$ in this case)

Kernel-Based Learning

Ridge regression: primal representation

grows w/ param

Setting $\nabla_{\boldsymbol{w}} E_\lambda = 0$, we obtain the (regularized) normal equations

$$-2\mathbf{X}^\top(\boldsymbol{t} - \mathbf{X}\boldsymbol{w}) + 2\lambda\boldsymbol{w} = 0$$

estimation

with solution $\widehat{\boldsymbol{w}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_{d+1})^{-1}\mathbf{X}^\top\boldsymbol{t}$

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

à abv -1 t

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

2. $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_{d+1}$ always has an inverse, for all $\lambda > 0$

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

Kernel-Based Learning

Dual representation grows w/ data

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

$$\hat{\boldsymbol{w}} = \sum_{i=1}^{n} \hat{\alpha}_i \boldsymbol{x}_i$$ → use this to go from primal to dual

In consequence,

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

very diff compared to $X^T X$

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

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

how grows w/ data

Kernel-Based Learning

Primal and dual

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

primal

$$f(\boldsymbol{x}) = \hat{\boldsymbol{w}}^\top \boldsymbol{x} = \sum_{j=0}^{d} \hat{w}_j x_j$$

and

dual

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

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

param data size

- The primal requires the computation and inversion of $\mathbf{X}^\top\mathbf{X}+\lambda\mathbf{I}_(d+1)$, requiring $O(nd^2 + d^3)$ operations

param model

- The dual requires the computation and inversion of $\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}_n$, requiring $O(dn^2 + n^3)$ operations

Kernel-Based Learning

How can we perform non-linear regression?

First create a *feature map*, a function $\phi : \mathbb{R}^d \to \mathbb{R}^D$, 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* (possibly part of a larger vector space)
- as a technicality, we could easily add $\phi_0(\boldsymbol{x}) = 1$ as before, if desired

Kernel-Based Learning

Hilbert spaces

The right structure for this vector space $\mathcal{H}$ is that of a **Hilbert space**:

David Hilbert (1862-1943)

- A vector space endowed with an **inner product** $\langle , \rangle$ whose associated norm defines a complete metric
  1. define inner prod $\langle \cdot,\cdot \rangle : H \times H \to \mathbb{R}$
  2. define norm out of inner prod $\|f\| := \sqrt{\langle f, f \rangle_H}$
  3. check completeness
- Completeness means that all Cauchy sequences defined in $\mathcal{H}$ converge to an element of $\mathcal{H}$
  $\Leftrightarrow \forall \varepsilon > 0, \exists N : \|x_n - x_m\| < \varepsilon, \forall n, m > N$

- Example: the l_2 space of square-summable sequences
  $l_2 : \{ [a_n]_{n \in \mathbb{N}} \mid \sum_{n=0}^{\infty} (a_n)^2 < \infty \}$ complete $\subset \mathbb{R}$ is complete

(informally) Generalizes the notion of Euclidean space: an infinite-dimensional space with the structure of $\mathbb{R}^d$ (distances, lenghts and angles are well-defined)

Kernel-Based Learning

The (non-linear) regression function has now the primal representation:

compute dual rep as exercise

feature map

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

- This feature space has the structure of $\mathbb{R}^D$ (a vector space)

- In consequence, there is also the dual representation:

$w = \sum a_i x_i$

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

(BTW, how general are all these results?)

Kernel-Based Learning

Feature maps and kernels (1)

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

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

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

→ the PROMISE!!!

if $k : X \times X \to \mathbb{R}$ is symm PSD func,
then $\exists \phi : X \to H$ s.t. $\langle \phi(x), \phi(x') \rangle_H = k(x, x') \quad \forall x, x' \in X$
⇒ reverse also applies

Kernel-Based Learning

Feature maps and kernels (2)

Our regression function has now the dual representation:

often 0 (sparse)

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

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)

n

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

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

where (as introduced earlier) $\mathbf{K} = (k_{ij})$, with $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

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) choice $\phi(\boldsymbol{x}) = \boldsymbol{x}$? In this case $d = D$ and $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle = \boldsymbol{x}^\top \boldsymbol{x}'$. The regularized solution reads:

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

where

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

(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** it!)

Kernel-Based Learning

Kernelizing ...

Many (classical and new) learning algorithms can be kernelized:

1. They require solving a problem where the data appear in the form of pairwise inner products (or pairwise Euclidean distances)

2. The solution is expressed as a linear combination of the kernel function centered at the data (ideally we only use *some* of the data: **sparsity**)

3. Examples include SVMs, ridge regression, perceptrons, FDA, PLS [supervised], as well as PCA, k-means, Parzen Windows [unsupervised]

Kernel-Based Learning

General feature maps

A feature map is of the general form $\phi : \mathcal{X} \to \mathcal{H}$

The associated kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \Big\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \Big\rangle, \qquad \boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}$$

$\mathcal{X}$ can be any space, $\mathcal{H}$ is always a (special case of) Hilbert space

---

In our starting development, $\mathcal{X} = \mathbb{R}^d$

Kernel-Based Learning

Regularization-based learning algorithms

The key for this is the **regularization framework**; consider again the regularized empirical error for mapped data:

$$E_\lambda(\boldsymbol{w}) = \sum_{i=1}^{n} (t_i - \langle \boldsymbol{w}, \phi(\boldsymbol{x}_i) \rangle)^2 + \lambda \|\bar{\boldsymbol{w}}\|^2, \qquad \lambda > 0$$

which we now generalize to arbitrary **loss** functions $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$:

very general function

$$E_\lambda(\boldsymbol{w}) = \sum_{i=1}^{n} L(t_i, \langle \boldsymbol{w}, \phi(\boldsymbol{x}_i) \rangle) + \lambda \|\boldsymbol{w}\|^2, \qquad \lambda > 0$$

$(t_i - \langle w, \phi(x_i) \rangle)^2$ regularizer

Kernel-Based Learning

Regularization-based learning algorithms

Theorem (informal). If L is differentiable w.r.t. its second argument and $\widehat{\boldsymbol{w}}$ is a minimizer of $E_\lambda(\boldsymbol{w})$, then we can represent:

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

and therefore

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

This result is usually called the **Representer Theorem**. In the case of KRR, $\widehat{\boldsymbol{\alpha}} = (\mathbf{K} + \lambda \mathbf{I}_n)^{-1} \boldsymbol{t}$.

Organization of the course

Theory

Practice Lab

In-class exercises

Practical work

Partial exam: 08/11/21 10:00-11:30h

Kernel-Based Learning

Suggested theoretical exercise

1. Derive the (primal) solution of ridge regression by minimization of the regularized empirical error, and derive a formula for the model

2. Prove that the obtained (regularized) solution can be also written as:

$$\boldsymbol{w}^* = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n$$

In consequence,

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n \langle \boldsymbol{x}_n, \boldsymbol{x} \rangle, \text{ where } \boldsymbol{\alpha} = (XX^T + \lambda I_N)^{-1} \boldsymbol{t}$$

Kernel-Based Learning

Suggested practical exercise

Consider the function

$$f(x) = 0.5\frac{\sin(x-a)}{x-a} + 0.8\frac{\sin(x-b)}{x-b} + 0.3\frac{\sin(x-c)}{x-c},\ x \in \mathbb{R}$$

where $a = 10, b = 50, c = 80$.

1. Generate a dataset of $N = 1052$ examples where the x are equally-spaced in $[0.1, 100]$ and the targets for regression are obtained as $t = f(x) + N(0, 0.05^2)$

2. Fit standard polynomial regression with some degrees of your choice

3. Fit kernel ridge regression with the RBF kernel and some σ and λ of your (careful) choice, until you are satisfied with the fit.

Kernel-Based Learning & Multivariate Modeling

MIRI Master

Lluís A. Belanche
belanche@cs.upc.edu

Soft Computing Research Group

Universitat Politècnica de Catalunya

2021-2022

Kernel-Based Learning & Multivariate Modeling

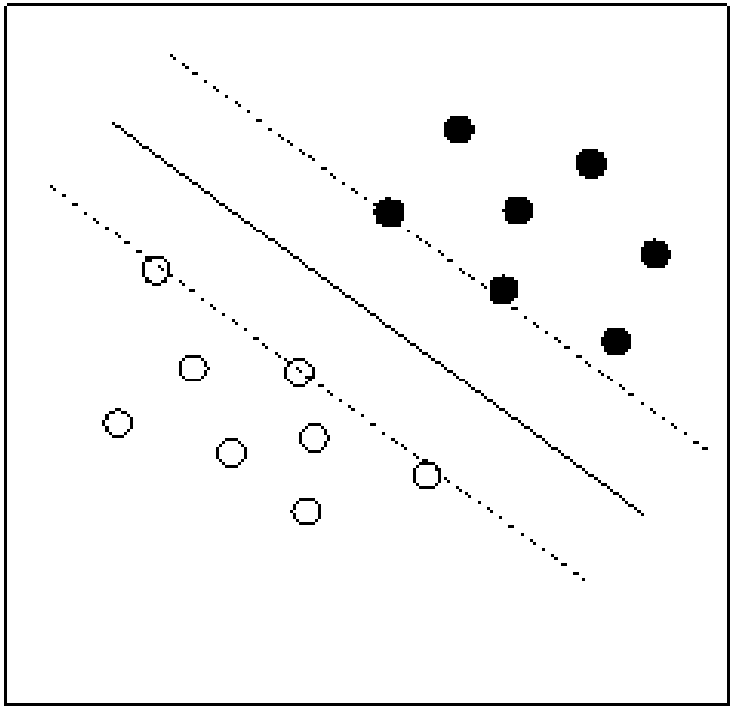
Syllabus

Part 1

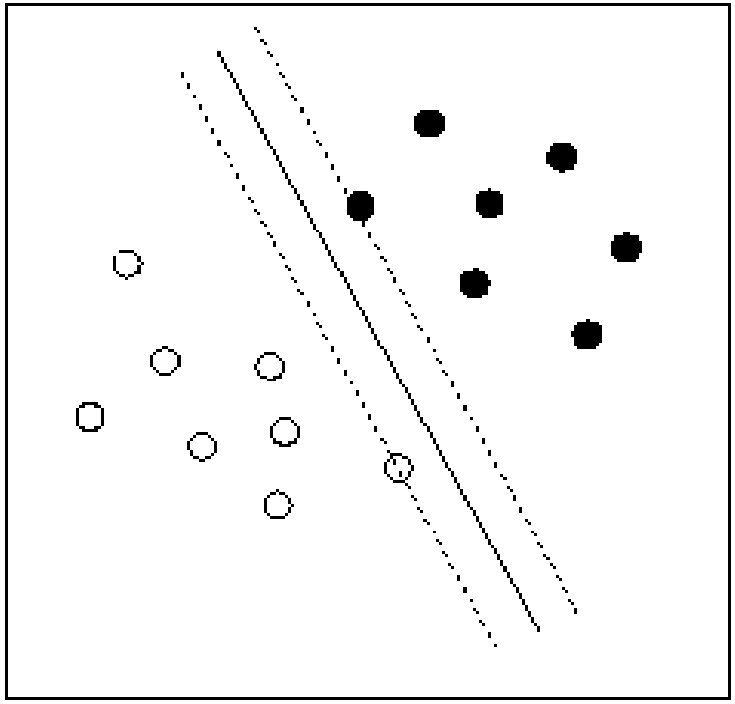
Sep 15 Introduction to kernel-based learning
Sep 22 The SVM for classification, regression & novelty detection (I)
Sep 29 The SVM for classification, regression & novelty detection (II)
Oct 06 Kernel design (I): theoretical issues
Oct 13 Kernel design (II): practical issues
Oct 20 Kernelizing ML & statistical algorithms
Oct 27 Advanced topics

Support Vector Machines

Preliminaries



(a) Larger margin



(b) Smaller margin

Which solution is more likely to lead to better **generalization**?

Works for all models & data sets

Support Vector Machines

Preliminaries

- Criterion for building a two-class classifier:

  Maximize the **margin = width of the separation** between the classes, defined by the distance to the nearest training examples

- **Working Hypotheses**:

  1. The data are linearly separable ("linsep") –very unlikely, but see later

  2. The larger the margin, the better the generalization (a first intuition)

Goal: find the separating hyperplane with the **largest margin**

Support Vector Machines

Preliminaries

For a two-class classifier, the **VC dimension** ϑ is the maximum number of points that can be separated in all possible 2^{ϑ} ways (**shattered**) by using functions representable by the classifier.

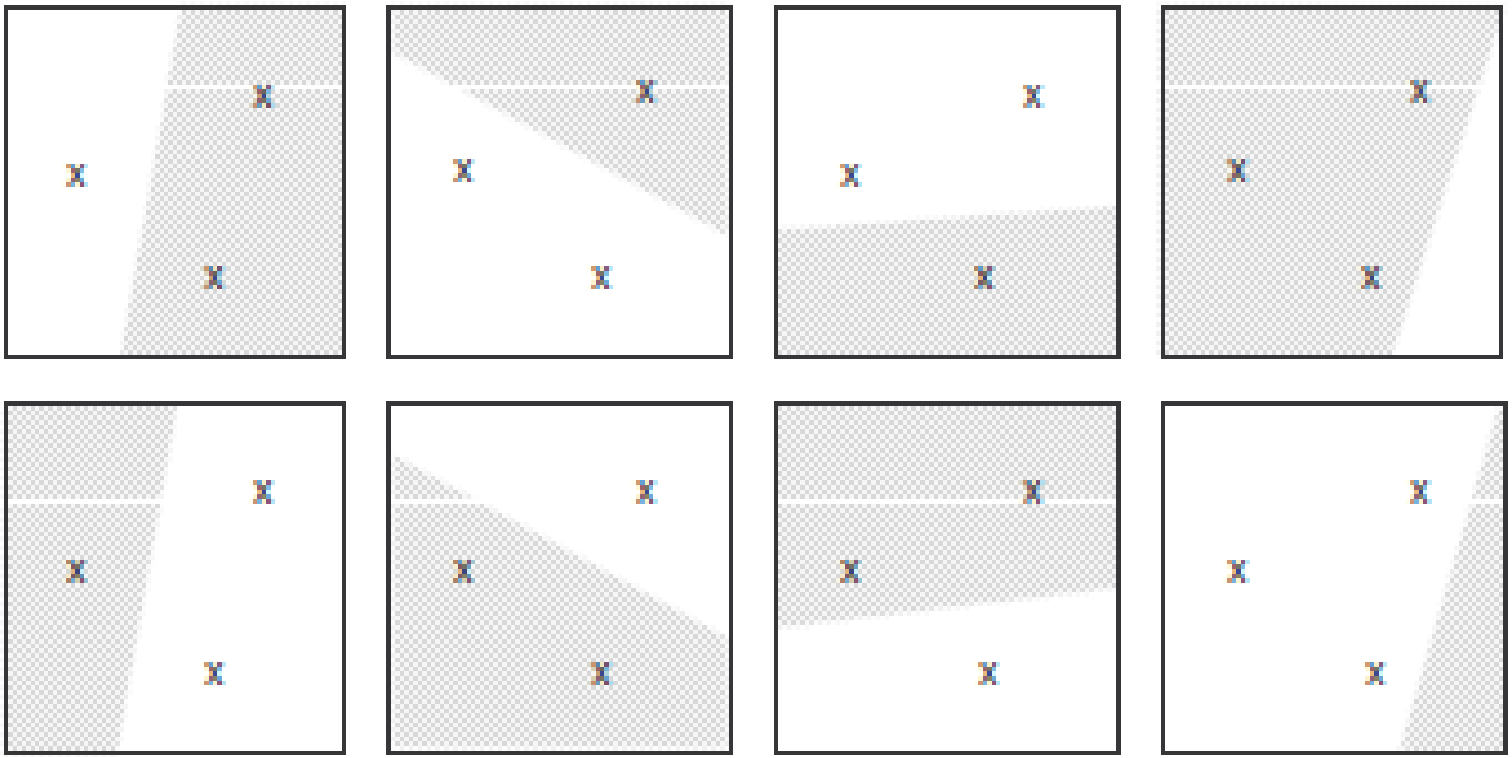
- Note it is *sufficient* that one set of ϑ points exists that can be shattered for the VC dimension to be at least ϑ

- If the VC dimension of a class is ϑ, this means there is at least one set of ϑ points that can be shattered by members of the class. It does not mean that every set of ϑ points can be shattered

- If no set of $\vartheta + 1$ points can be shattered by members of the class, then the VC dimension of the class is at most ϑ

Support Vector Machines

An example



$v(\mathbb{R}^2) = 3$

- In $\mathbb{R}^2$ we can shatter these three points (VC dim is ≥ 3)

- No set of four or more points can be shattered (VC dim is < 4)

Support Vector Machines

Why is the VC dimension relevant?

Theorem (Vapnik and Chervonenkis, 1974). Let D be an i.i.d data sample of size n and $\mathcal{Y}$ ($\{0,1\}$) a class of parametric binary classifiers. Let ϑ denote the VC dimension of $\mathcal{Y}$. Take $y \in \mathcal{Y}$ with empirical error $R_n(y)$ on D. For all $\eta > 0$ it holds true that, with probability at least $1-\eta$, the true error of y is bounded by:

$$R(y) \leq R_n(y) + H(n, \vartheta, \eta)$$

("error term")

where

$$H(n, \vartheta, \eta) = \sqrt{\frac{\vartheta(\ln(2n/\vartheta) + 1) - \ln(\eta/4)}{n}}$$

Support Vector Machines

Formalisation

We have a data set $D = \{(\boldsymbol{x}_1, t_1), \ldots, (\boldsymbol{x}_n, t_n)\}$, with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $t_i \in \{-1, +1\}$, describing a two-class problem.

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

1. • Set up an **affine function** $g(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$

2. • Obtain a **linear discriminant** as $f(\boldsymbol{x}) = \operatorname{sgn}(g(\boldsymbol{x}))$

   sign (Vorzeichen = {1, 0, −1})

3. • We would like to find $\boldsymbol{w}, b$ such that:

   $\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b > 0$, when $t_i = +1$

   $\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b < 0$, when $t_i = -1$ ← error

   Find w, b s.t. $(w_i x_i) + b > 0$

   that is $t_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) > 0$ or simply $t_i g(\boldsymbol{x}_i) > 0 \qquad (1 \leq i \leq n)$

Support Vector Machines

Formalisation

- The quantity $t_i\, g(\boldsymbol{x}_i)$ is called the **functional margin** of $\boldsymbol{x}_i$ (there will be an "error" whenever $t_i\, g(\boldsymbol{x}_i) < 0$)

- Define the **loss** $L(t_i, \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle) := \max(1 - t_i\, g(\boldsymbol{x}_i), 1)$

  has error $t_i > 0$

- Given the plane $\pi : g(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0$, the distance $d(\boldsymbol{x}, \pi) = \frac{|g(\boldsymbol{x})|}{\|\boldsymbol{w}\|}$ is called the **geometrical margin** of $\boldsymbol{x}$.

  $= \frac{|\langle w, x \rangle + b|}{\|w\|}$

- The **optimal separating hyperplane** (OSH) for linsep data is the one that maximizes the geometrical margin:

$$\max_{\boldsymbol{w}, b} \left\{ \min_{1 \le i \le n} d(\boldsymbol{x}_i, \pi) \right\} \qquad \text{subject to } t_i\, (\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) > 0 \ (1 \le i \le n)$$

max dist to nearest x

Support Vector Machines

Formalisation

- Rescaling $\boldsymbol{w}, b$ such that $|\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b| = 1$ for the points closest to the hyperplane, we obtain $|\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b| \geq 1$. The **support vectors** (SVs) are those $\{\boldsymbol{x}_i \;/\; |\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b| = 1\}$.

- The new loss is $\max(1 - t_i\, g(\boldsymbol{x}_i), 0) =: (1 - t_i\, g(\boldsymbol{x}_i))_+$ **(hinge loss)**

- The **margin** is twice the distance of any SV to the plane π:

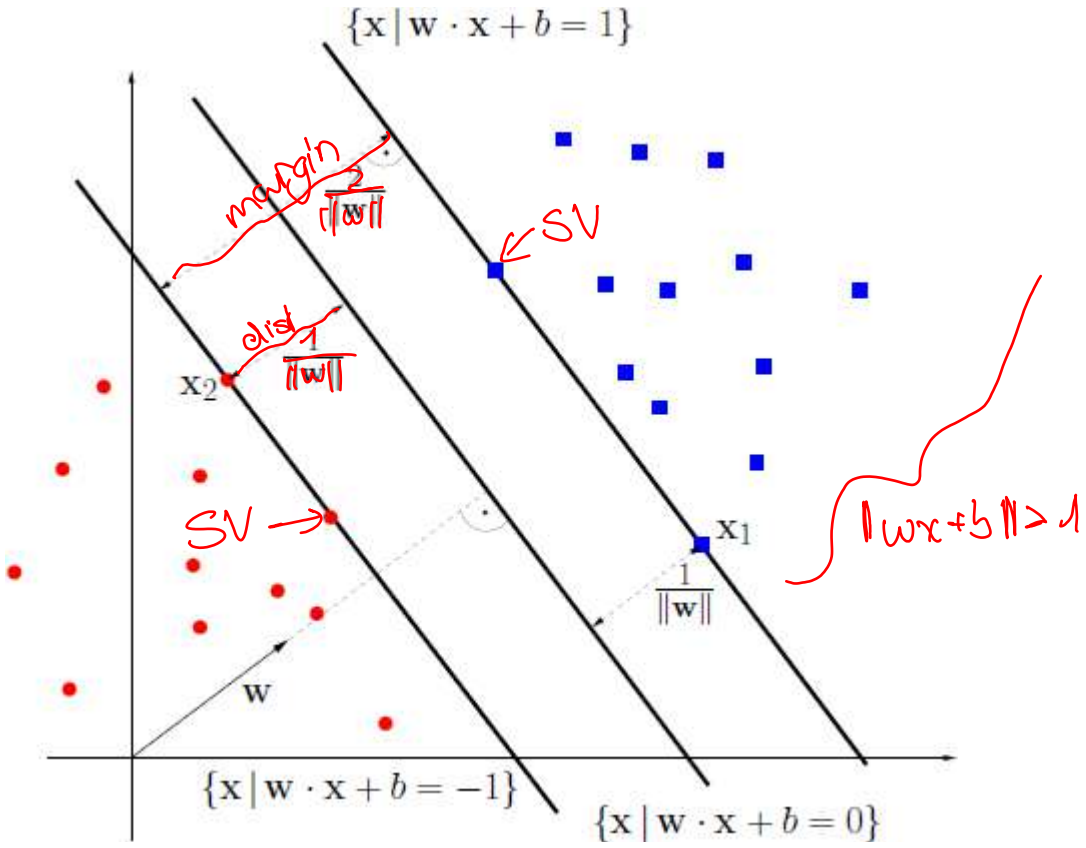
  $2\, d(\boldsymbol{x}_{\mathrm{SV}}, \pi) = 2/\|\boldsymbol{w}\|$, since $|g(\boldsymbol{x}_{\mathrm{SV}})| = 1$

- Therefore we find the **canonical** OSH by solving

$$\max_{\boldsymbol{w}, b} \left\{ \frac{2}{\|\boldsymbol{w}\|} \;\Big/\; t_i\, (\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geq 1, \qquad 1 \leq i \leq n \right\}$$

Support Vector Machines

Geometrical view of the OSH



Support Vector Machines

A look on what's to come

1. The solution for $\boldsymbol{w}$ can be expressed as $\boldsymbol{w} = \sum_{i=1}^{n} t_i \alpha_i \boldsymbol{x}_i,\ \alpha_i \geq 0.$

   (as a consequence of the **Representer theorem**)

2. A fraction of the training data vectors will have $\alpha_i = 0$ (**sparsity**, as a consequence of the chosen error function)

3. The $\boldsymbol{x}_i$ for which $\alpha_i > 0$ will coincide with the **support vectors**

4. The **discriminant function** (classifier) is written

$$f_{\text{SVM}}(\boldsymbol{x}) = \text{sgn}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b) = \text{sgn}\left(\sum_{i=1}^{n} t_i \alpha_i \langle \boldsymbol{x}, \boldsymbol{x}_i \rangle + b \right)$$

Support Vector Machines

More than an intuition

- Separating hyperplanes in $\mathbb{R}^d$ have VC dimension $d + 1$
- When we use a feature map into a very high dimension $D \in (\mathbb{N} \cup \{\infty\})$, VC dimension will grow accordingly
- If we bound the margin of the hyperplanes, we limit VC dimension (therefore, we have an explicit control on complexity)

Support Vector Machines

More than an intuition

Theorem. Consider canonical hyperplanes $f(\boldsymbol{x}) = \text{sgn}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)$ and a data set $D = \{(\boldsymbol{x}_1, t_1), \ldots, (\boldsymbol{x}_n, t_n)\}$, with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $t_i \in \{-1, +1\}$. The **subclass** of linear classifiers with margin $m \geq m_0$ has VC dimension ϑ bounded by

VC dim bounded by dist/radius

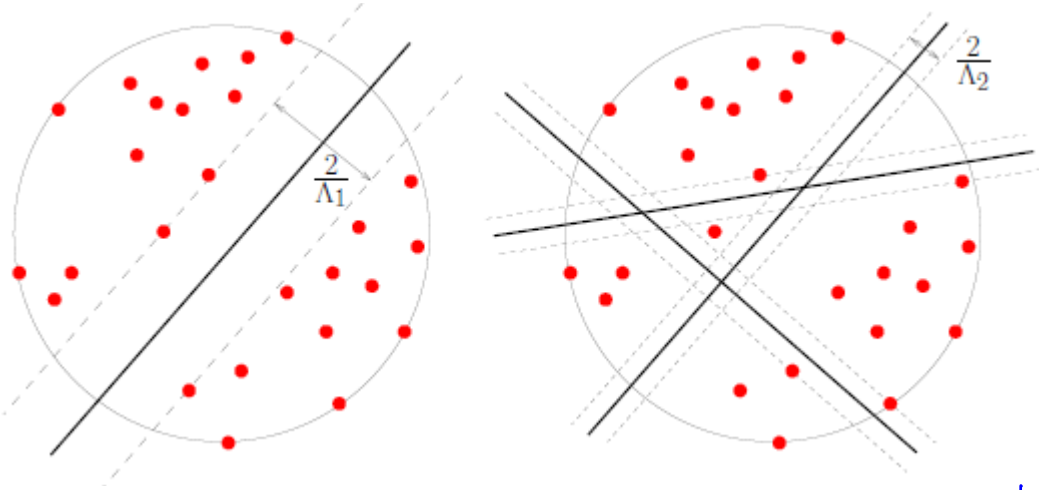
$$\vartheta \leq \min\left(\left\lceil \frac{R^2}{m_0^2} \right\rceil, d\right) + 1$$

$\vartheta = \min \left(\frac{R}{m}\right)^2$ oder dim

where R is the radius of the smallest sphere centered at the origin containing the $\boldsymbol{x}_i$.

Support Vector Machines

More than an intuition



smaller margin: more ways to shatter (2^v possib)

- Left: hyperplanes with a large margin have reduced chances to separate the data (the VC dimension is small)

- Right: smaller margins allow more separating hyperplanes (the VC dimension is large)

Support Vector Machines

Formulation

$$\underset{\boldsymbol{w},b}{\textbf{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$\textbf{subject to } t_i\left(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b\right) \geq 1, \qquad 1 \leq i \leq n$$

This is solved (numerically) by QP techniques:

- Quadratic (therefore convex) function subject to linear constraints
- Unique solution (or set of equivalent ones)
- Therefore, no local minima

Support Vector Machines

Formulation

For the set of constraints to be satisfied, the data set must be linsep; this is a very unrealistic requirement in practice

- We could aim at minimizing the **number of** violated constraints $|\{n \;\; / \;\; t_i\,(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) < 1\}|$, but this turns out to be NP-hard ...

- Instead, we can minimize a convex function of $\boldsymbol{w}$:

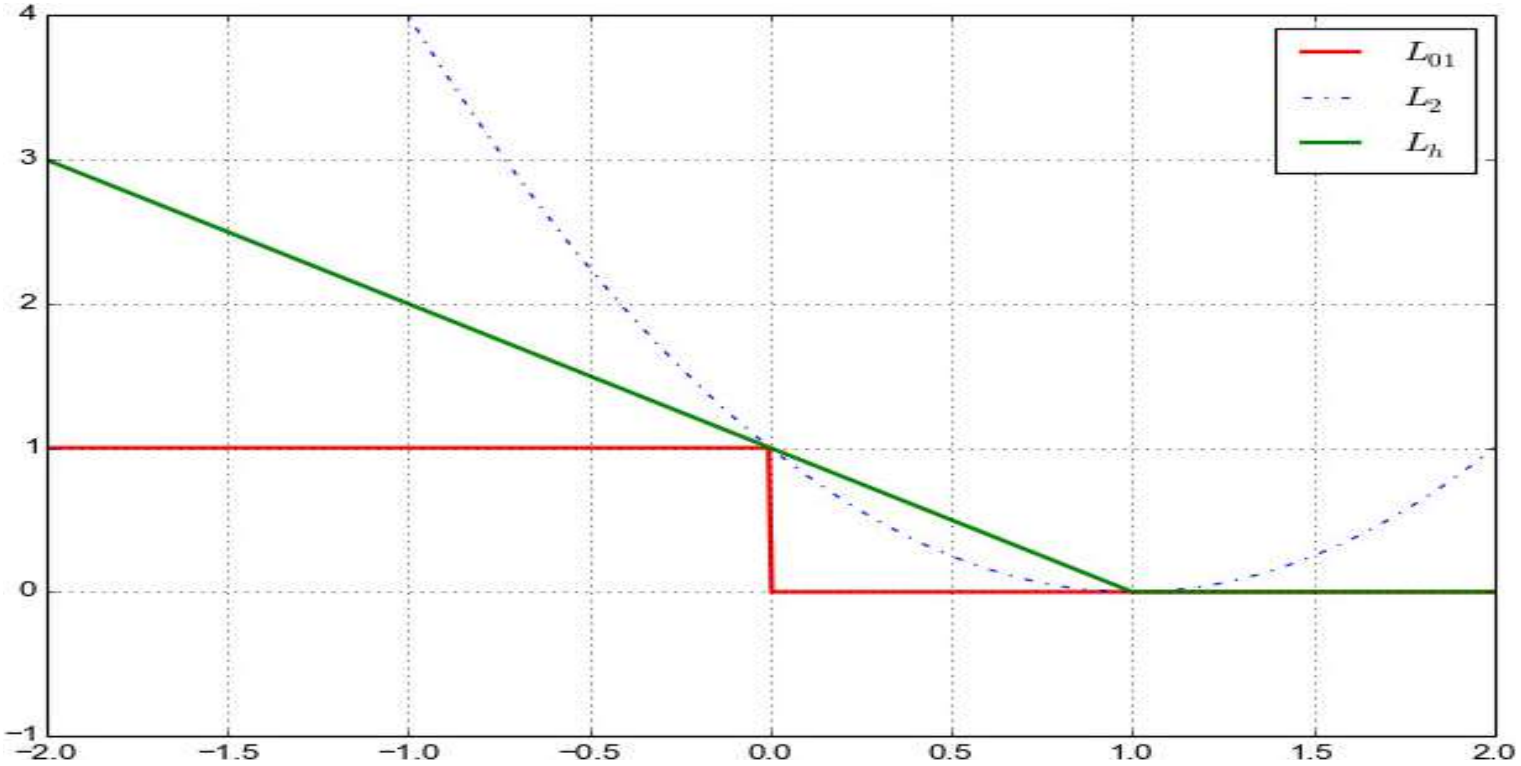
$$\underset{\boldsymbol{w},b}{\textbf{minimize}} \qquad \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} (1 - t_i\, g(\boldsymbol{x}_i))_+, \qquad C > 0$$

new part
error term
cost param regularizing sol → doesn't directly affect margin value

- Yes, the new term is the total **hinge loss**!

Support Vector Machines

Formulation



L_{01} is the 0/1 loss; L_2 is the square loss; L_h is the hinge loss

$(1 - t(g(x)))_+$ $\quad \frac{\|w\|^2}{2}$ $\quad$ combi $\quad \frac{\|w\|^2}{2} + C\sum(1 - t(g(x)))_+$

Support Vector Machines

Margin violations

- This problem is rewritten as another QP, by introducing a set of margin violations ε_i –called **slack** variables–, for each $\boldsymbol{x}_i$:

$$\underset{\boldsymbol{w},b,\{\varepsilon_i\}}{\textbf{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\varepsilon_i$$

$$\textbf{subject to } t_i\left(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b\right) \geq 1 - \varepsilon_i \textbf{ and } \varepsilon_i \geq 0 \ (1 \leq i \leq n)$$

- This is a **soft** margin ($\varepsilon_i > 0$ implying $\boldsymbol{x}_i$ would violate the original constraint)
- For a training error to occur $\varepsilon_i > 1$ and so $\sum_{i=1}^{n}\varepsilon_i$ is an upper bound on the number of training errors
- The optimal slacks satisfy $\varepsilon_i = (1 - t_i\, g(\boldsymbol{x}_i))_+$

Support Vector Machines

Excursion: Lagrange multipliers

The famous method of Lagrange multipliers allows the optimization of smooth functions subject to **equality constraints**. (local optimum)

The Karush, Kuhn and Tucker (KKT) theory extends Lagrange's method to include **inequality constraints**.

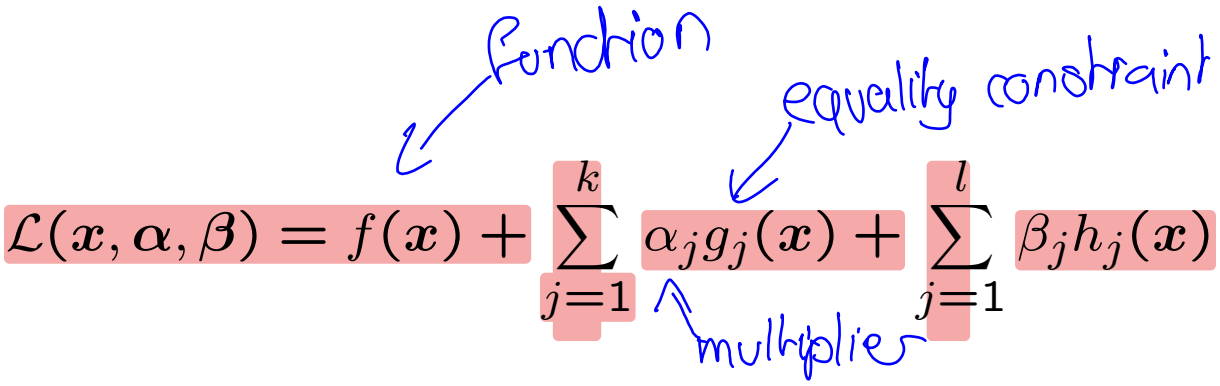
Consider the problem of minimizing $f(\boldsymbol{x})$ in a convex $\Omega \subset \mathbb{R}^d$, subject to:

- $g_j(\boldsymbol{x}) \leq 0$ affine functions, $1 \leq j \leq k$
- $h_j(\boldsymbol{x}) = 0$ affine functions, $1 \leq j \leq l$

Support Vector Machines

Excursion: Lagrange multipliers

Define the **Lagrangian** as:



where f, g_j, h_j are continuously differentiable functions.

Support Vector Machines

Excursion: Lagrange multipliers

Theorem. Necessary and sufficient conditions for a point $\boldsymbol{x}^*$ to be an optimum are the existence of $\boldsymbol{\alpha}^* \in \mathbb{R}^k$ and $\boldsymbol{\beta}^* \in \mathbb{R}^l$ such that:

ableiten

1. $\frac{\partial \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{x}} = 0$
2. $\frac{\partial \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} = 0$

} nach x & β ableiten
↳ beides $= 0$

3. $\alpha_j^* g_j(\boldsymbol{x}^*) = 0, 1 \leq j \leq k$ (KKT complementarity conditions)

for opt $\alpha^* \rightarrow \alpha^* \cdot g(x^*) = 0$

4. $g_j(\boldsymbol{x}^*) \leq 0, 1 \leq j \leq k$

≥ 0 ≤ 0

5. $\alpha_j^* \geq 0, 1 \leq j \leq k$

eins von beiden ist 0

Support Vector Machines

SVM Lagrangian (primal)

We construct the **Lagrangian**:

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \Big\{ t_i \left(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b\right) - 1 + \varepsilon_i \Big\} + C \sum_{i=1}^{n} \varepsilon_i - \sum_{i=1}^{n} \mu_i \varepsilon_i$$

only for error term

cost param

multipliers s.t. $\varepsilon \geq 0$

- The $\alpha_i, \mu_i \geq 0$ are the **Lagrange multipliers**; the μ_i ensure that $\varepsilon_i \geq 0$
- The solution is a **saddle point** of $\mathcal{L}$: minimum w.r.t. $\boldsymbol{w}, b$ and the ε_i and maximum w.r.t. the α_i and μ_i

Support Vector Machines

Lagrangian form

The gradient of $\mathcal{L}$ with respect to $\boldsymbol{w}, b$ and ε_i must vanish:

s.t. only α_i & x left ⇒ Kernel

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{n} \alpha_i t_i = 0, \qquad \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i t_i \boldsymbol{x}_i = 0, \qquad \frac{\partial \mathcal{L}}{\partial \varepsilon_i} = C - \alpha_i - \mu_i = 0$$

In addition, the KKT complementarity conditions must hold:

$$\alpha_i \Big(t_i \left(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b \right) - 1 + \varepsilon_i \Big) = 0$$

$$L(w,b,a) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i \underbrace{[t_i(\langle w, x_i\rangle + b)}_{t_i(w^T x_i + b)} - 1]$$

$$w = \sum \alpha_i t_i x_i$$

$$= \frac{1}{2}\left\|\sum \alpha_i t_i x_i\right\|^2 - \sum_{i=1}^{n} \alpha_i \underbrace{[t_i((\sum_j^n a_i t_i x_i)^T x_i + b)}_{-bt_i - (\sum a_i t_i x_i)^T (\sum a_i t_i x_i)} - 1]$$

$$= \frac{1}{2}(\sum a_i t_i x_i)^T (\sum a_i t_i x_i) - \sum a_i - b\underbrace{\sum a_i t_i}_{0} - (\sum a_i t_i x_i)^T (\sum a_i t_i x_i)$$

$$= \sum a_i - \frac{1}{2}(\sum a_i t_i x_i)^T (\sum a_i t_i x_i) \quad \left(\frac{1}{2}x - x\right)$$

$$= \sum a_i - \frac{1}{2}\sum_{i}^{n}\sum_{j}^{n} a_i a_j t_i t_j (x_i^T x_j)$$

$$= \sum a_i - \frac{1}{2}\sum\sum a_i a_j t_i t_j (x_i, x_j)$$

Support Vector Machines

Dual formulation

The Lagrangian $\mathcal{L}$ is convex; its optimization is equivalent to the maximization of its concave **dual problem** $\mathcal{L}_D$:

(over n)

$j \le k, l$

$$\underset{\boldsymbol{w},b,\{\alpha_i\}}{\textbf{minimize}} \quad \mathcal{L}_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j t_i t_j \left\langle \boldsymbol{x}_i, \boldsymbol{x}_j \right\rangle$$

$$\textbf{subject to } 0 \le \alpha_i \le C \ (1 \le i \le n), \qquad \textbf{and} \ \sum_{i=1}^{n} \alpha_i t_i = 0$$

- Neither $\mu_i, \varepsilon_i, \boldsymbol{w}, b$ appear in the dual form; maximization is only w.r.t. the α_i
- This optimization problem is expressed *only* in terms of inner products of the data points: the dual lends itself to kernelisation
- How many free parameters? n (independent of data dimension)

Support Vector Machines

Dual formulation

A closer look at the KKT complementarity conditions:

- $\alpha_i = 0$ implies $t_i\, g(\boldsymbol{x}_i) > 1$ and $\varepsilon_i = 0$ ($\boldsymbol{x}_i$ is **not a SV**) liegt außerhalb margin
- $\alpha_i \in (0, C)$ implies $t_i\, g(\boldsymbol{x}_i) = 1$ and $\varepsilon_i = 0$ ($\boldsymbol{x}_i$ is a **non-bound SV**) kann auf margin liegen, aber nicht immer SV
- $\alpha_i = C$ implies $t_i\ g(\boldsymbol{x}_i) < 1$ and $\varepsilon_i > 0$ ($\boldsymbol{x}_i$ is a **bound SV**)

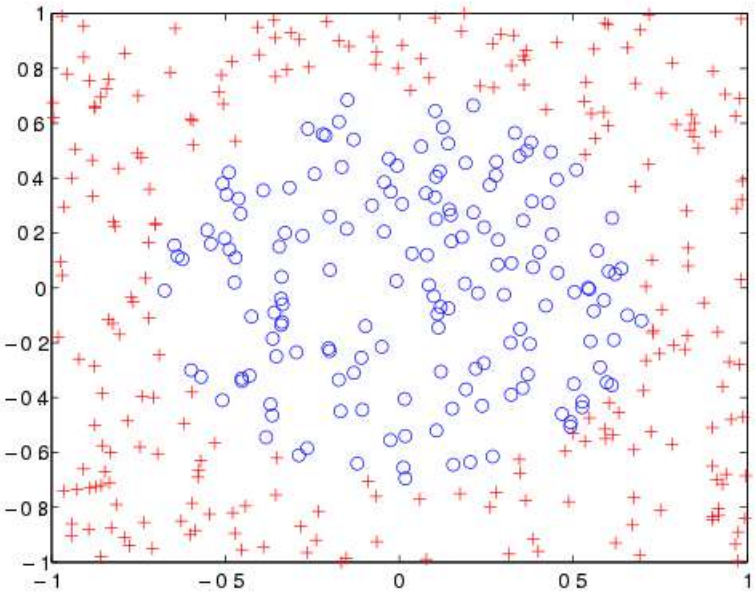
  (in particular, $\varepsilon_i > 1$ implies $\boldsymbol{x}_i$ is a **training error**)

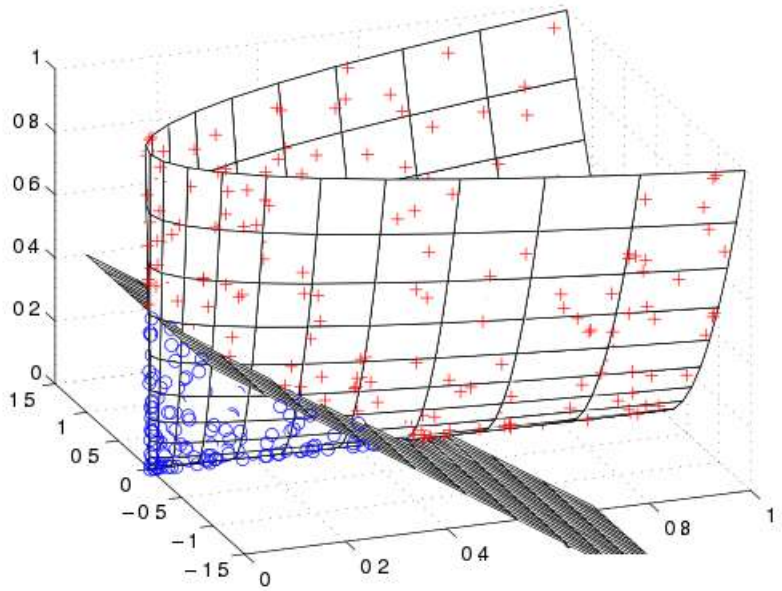
Support Vector Machines

The SVM goes non-linear

Recall the idea of mapping input data into some Hilbert space (called the **feature space**) via a non-linear mapping $\phi : \mathcal{X} \to \mathcal{H}$

The associated kernel function is $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle_{\mathcal{H}}, \ \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$





Support Vector Machines

SVM kernelization

- We now substitute $\boldsymbol{x}_i$ by $\phi(\boldsymbol{x}_i)$, then build the OSH in $\mathcal{H}$

- The dual of the new QP problem is formulated exactly as before, replacing $\left\langle \boldsymbol{x}_i, \boldsymbol{x}_j \right\rangle$ with $\left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle_{\mathcal{H}} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$

- The discriminant function becomes:

$= \text{sgn}(w^T \phi(x) + b) = \text{sgn}\left(\sum \alpha_i t_i \phi(x_i)^T \phi(x) + b \right)$

(w; $k(x, x_i)$)

$$f_{\text{SVM}}(\boldsymbol{x}) = \text{sgn}\left(\sum_{i=1}^{n} \alpha_i t_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b \right)$$

Support Vector Machines

LOOCV bounds (I)

A rough but simple bound on LOOCV (leave-one-out CV) error can be computed as:

$$\text{LOOCV}(n) \leq \frac{1}{n}\mathbb{E}(n_{\text{SV}})$$

n_{SV} is the number of SVs for a given sample of size n
The $\mathbb{E}()$ is taken over all such samples

Support Vector Machines

LOO bounds (II)

Theorem. The LOOCV error of a stable SVM$^{(*)}$ on a set of training patterns $\boldsymbol{x}_i$ is bounded by $|\{i \,/\, (2\alpha_i R^2 + \varepsilon_i) \geq 1\}|/n$, where R^2 is an upper bound on $k(\boldsymbol{x}, \boldsymbol{x})$ and $k(\boldsymbol{x}, \boldsymbol{x}') \geq 0$.

$\alpha \leq \frac{|i|}{n}$ s.t. $2\alpha_i R^2 + \varepsilon_i \geq 1$

- This quantity can be extracted easily from the solution
- This LOOCV error is an unbiased estimate of true error

$^{(*)}$ A SVM is stable if there is at least one non-bound SV (see *Estimating the Generalization Performance of a SVM Efficiently*. T. Joachims; In ICML, 2000)

Support Vector Machines

Final remarks (I)

- The fact that the **OSH** is determined only by the support vectors is most remarkable, since usually this number will be small

- The **support vectors** (SVs) are:

  1. the only training examples that define the solution

  2. the most difficult examples to classify

- This means all the **relevant information** in the data set is summarized by the SVs: we would have obtained the same result by using *only* the SVs from the outset

Support Vector Machines

Final remarks (II)

- The SVM is specially well suited for "large d, low n" problems, because:

  1. complexity grows with n (non-parametric model)

  2. space requirements (the kernel matrix) also grows with n

  3. generalization error does not depend on d

- The "architecture" is determined automatically by the method (not by experimentation, as in neural networks)

Support Vector Machines

Hot topics

- Choice of the **best kernel** is an open issue; **kernel design** is an active area of research

- More efficient algorithms for solving **big QP** problems are being developed

- Sometimes the **fraction of SVs** is very high (indicating a poor fit); it is possible to control this fraction directly (ν-SVMs)

- Performance usually depends on a careful choice of the external parameters: C and those of the kernel function; we need principled ways for **hyper-parameter** selection

Support Vector Machines

Where to look for more ...

- *An Introduction to Kernel-based Learning Algorithms*. K.-R. Mueller, S. Mika, G. Raetsch, K. Tsuda, and B. Schoelkopf, IEEE Neural Networks, 12(2):181-201, 2001.

- *A Tutorial on Support Vector Machines for Pattern Recognition*. Christopher Burges.
  `https://research.microsoft.com/en-us/um/people/cburges/papers/svmtutorial.pdf`

- *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Bernhard Schoelkopf and Alexander J. Smola, MIT Press, 2001.

- *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Nello Cristianini and John Shawe-Taylor, Cambridge University Press, 2000.

- *Kernel Methods for Pattern Analysis*. John Shawe-Taylor and Nello Cristianini, Cambridge University Press, 2004.

- *The Nature of Statistical Learning Theory*. V. Vapnik, Springer, 2nd ed., 1999

Kernel-Based Learning & Multivariate Modeling

MIRI Master

Lluís A. Belanche
belanche@cs.upc.edu

Soft Computing Research Group

Universitat Politècnica de Catalunya

2021-2022

Kernel-Based Learning & Multivariate Modeling

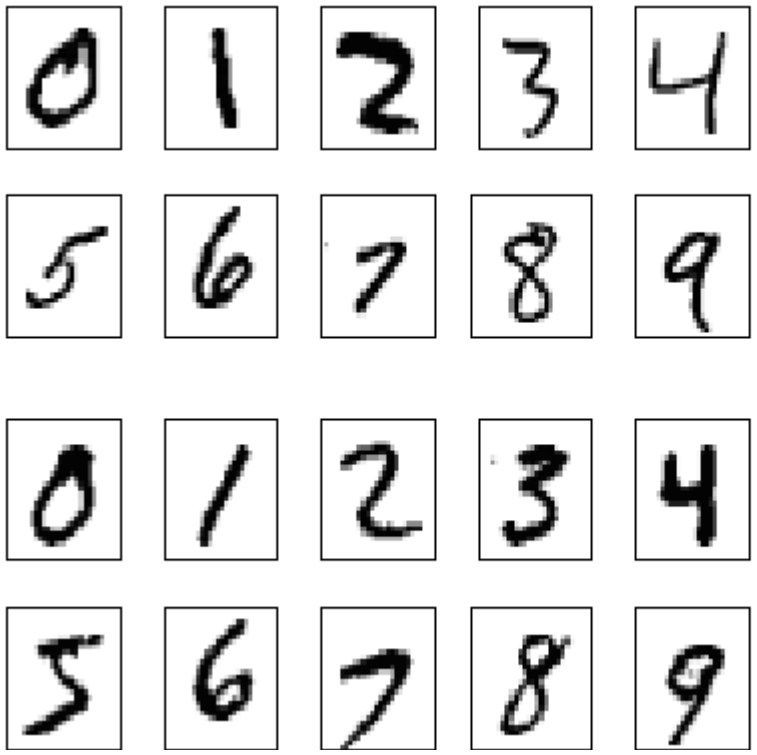
Syllabus

Part 1

Sep 15 Introduction to kernel-based learning
Sep 22 The SVM for classification, regression & novelty detection (I)
Sep 29 The SVM for classification, regression & novelty detection (II)
Oct 06 Kernel design (I): theoretical issues
Oct 13 Kernel design (II): practical issues
Oct 20 Kernelizing ML & statistical algorithms
Oct 27 Advanced topics

Support Vector Machines

Application (digit recognition)



- Handwritten zip code recognition traces back to the 1960's

Support Vector Machines

Application (digit recognition)

- MNIST handwritten zip code recognition

- 60.000 training, 10.000 test examples (28×28 pixels)

Classifier	test error
Linear classifier	8.4 %
3-nearest-neighbor	2.4 %
SVM	1.4 %
Tangent distance	1.1 %
LeNet4	1.1 %
Boosted LeNet4	0.7 %
Translation invariant SVM	0.56 %

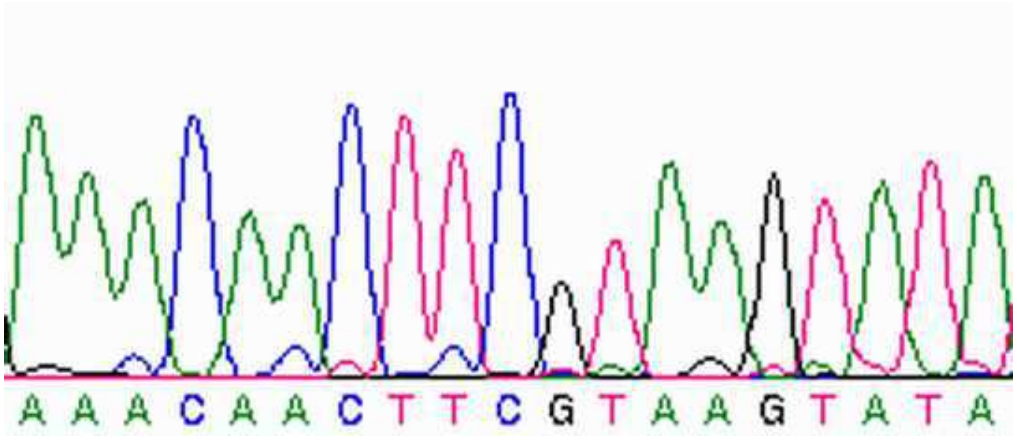
Support Vector Machines

Application: Classification of DNA sequences

- A *promoter* is a region of DNA that initiates or facilitates transcription of a particular gene

- The dataset consists of 106 DNA sequences described by 57 categorical variables, represented as the nucleotide at each position: [A]denine, [C]ytosine, [G]uanine, [T]hymine

- The response is a binary class: "+" for a promoter gene and "−" for a non-promoter gene



Support Vector Machines

Application: Classification of DNA sequences

The similarity between two multivariate categorical vectors is the fraction of the number of matching values.

Overlap/Dirac kernel]

$$k_0(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{d} \sum_{i=1}^{d} \mathbb{I}_{\{x_i = x_i'\}}$$

no matching values

Another kernel that can be used is the RBF kernel:

Gaussian Radial Basis Function kernel:

$$k_{\text{RBF}}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma ||\boldsymbol{x} - \boldsymbol{x}'||^2\right), \gamma > 0$$

In order to use this kernel, categorical variables with m modalities are coded using a binary expansion representation.

Support Vector Machines

Application: Classification of DNA sequences

Univariate kernel $k_1^{(U)}$:

$$k_1^{(U)}(x_i, x_i') = \begin{cases} h_\alpha(P_Z(x_i)) & \text{if } x_i = x_i' \\ 0 & \text{if } x_i \neq x_i' \end{cases}$$

where

$$h_\alpha(z) = (1 - z^\alpha)^{1/\alpha}, \ \alpha > 0$$

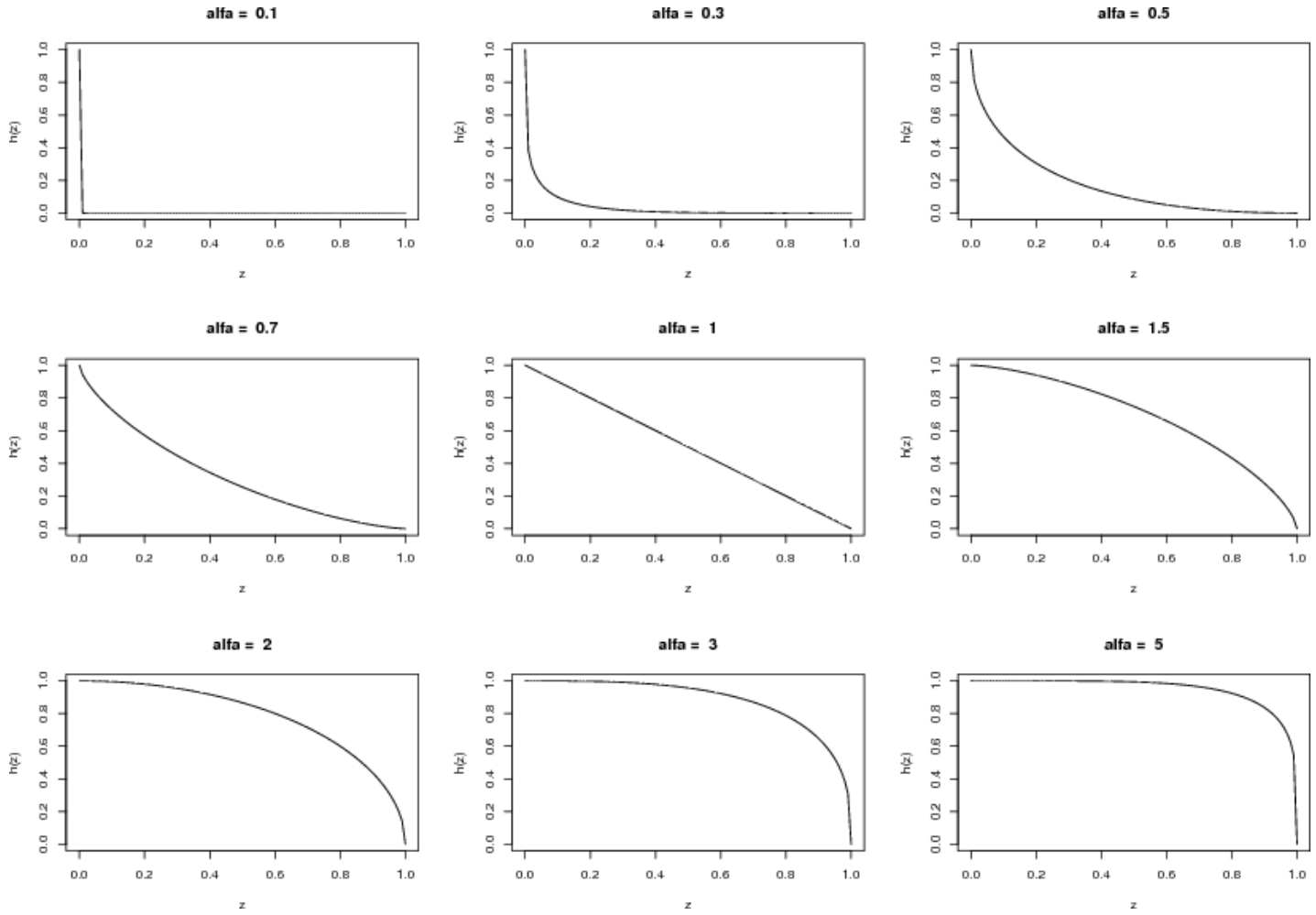
Multivariate kernel k_1:

$$k_1(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(\frac{\gamma}{d} \sum_{i=1}^{d} k_1^{(U)}(x_i, x_i')\right), \ \gamma > 0$$

The kernel matrices generated by k_1 are p.s.d.

Support Vector Machines

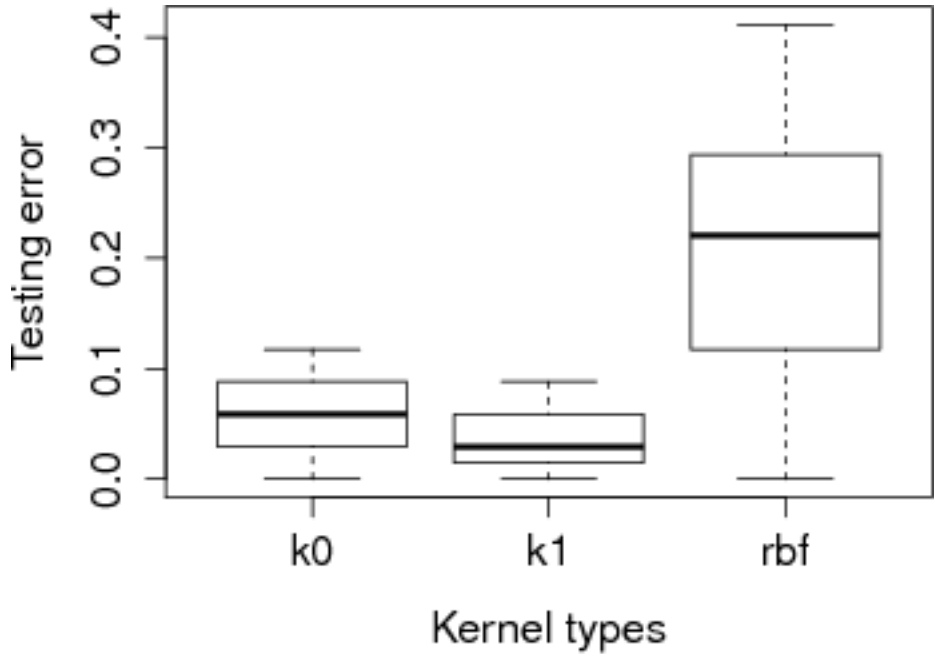
Application: Classification of DNA sequences



The family of inverting functions $h_\alpha(z)$ for different values of α

Support Vector Machines

Application: Classification of DNA sequences



Test error distributions on the PromoterGene problem

(*joint work with M. Villegas*)

Support Vector Machines

The role of the C parameter

Increasing the value of C ... (only recommended)

can affect but not always

- penalizes margin errors more $\Rightarrow$ narrower margin $\Rightarrow$ larger VC-dimension
- allows the $\alpha_i \leq C$ to be larger (so more opportunities for outliers)
- increases training times

Support Vector Machines

The role of the C parameter

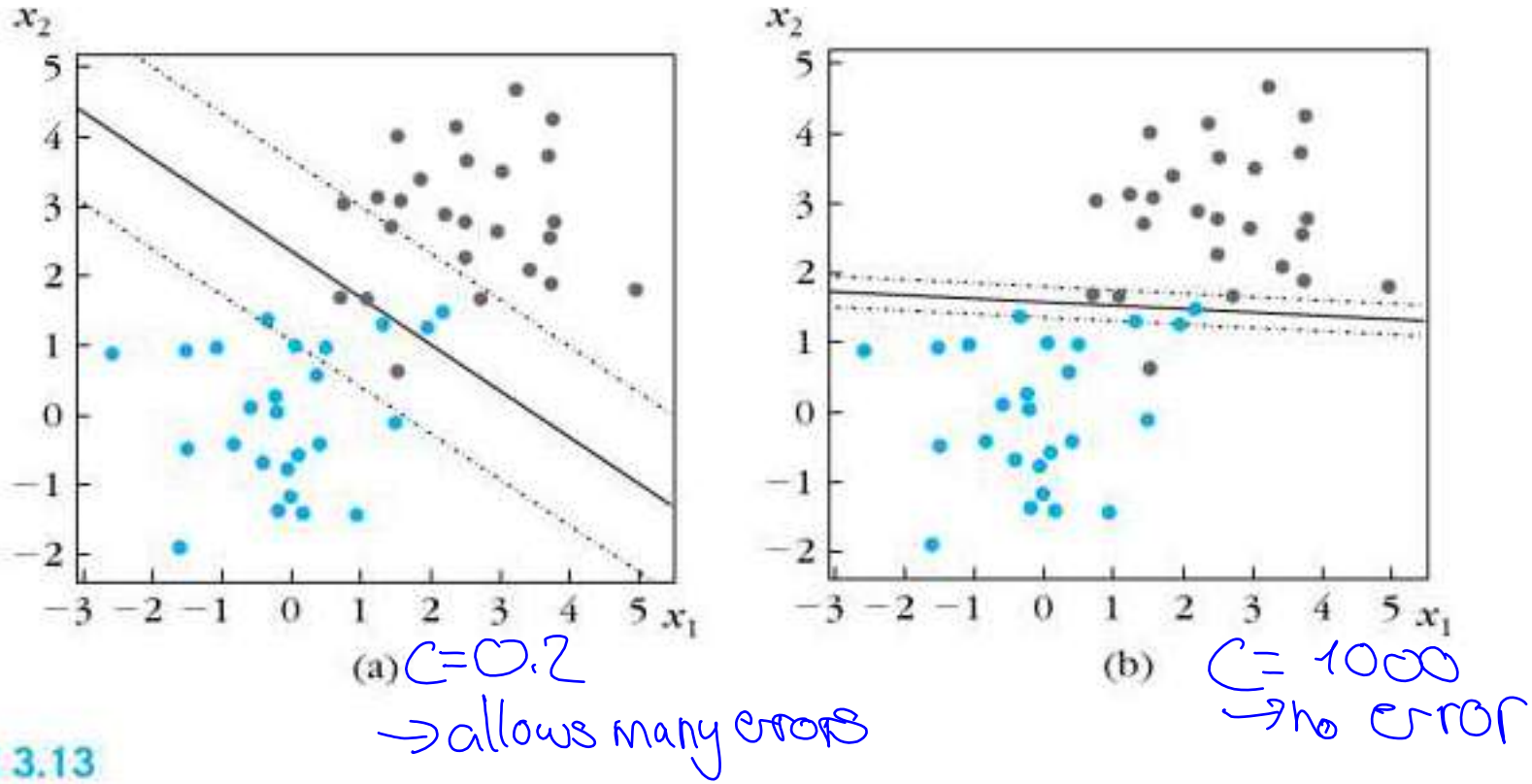


FIGURE 3.13

An example of two nonseparable classes and the resulting SVM linear classifier (full line) with the associated margin (dotted lines) for the values (a) $C = 0.2$ and (b) $C = 1000$. In the latter case, the location and direction of the classifier as well as the width of the margin have changed in order to include a smaller number of points inside the margin.

–from *Pattern Recognition (Fourth Edition)*, S. Theodoridis and K. Koutroumbas

Support Vector Machines

ν-SVMs

There are two commonly used versions of the SVM for classification:

'C-SVC': original SVM formulation, uses a parameter $C \in (0, \infty)$ to apply a penalty to the optimization for those data points not entirely separated by the OSH (violating the margins) penalty for opt

'nu-SVC': C is replaced by $\nu \in (0, 1)$: lower & upper bound

- upper bound on the fraction of examples which are training errors (missclassified)
- lower bound on the fraction of points which are SVs.

Support Vector Machines

SVMs for regression

"The Support Vector method can also be applied to the case of regression, maintaining all the main features that characterise the maximal margin algorithm: a non-linear function is learned by a linear learning machine in a kernel-induced feature space while the capacity of the system is controlled by a parameter that does not depend on the dimensionality of the space."

–from N. Cristianini and J. Shawe-Taylor, *An introduction to Support Vector Machines* (2000)

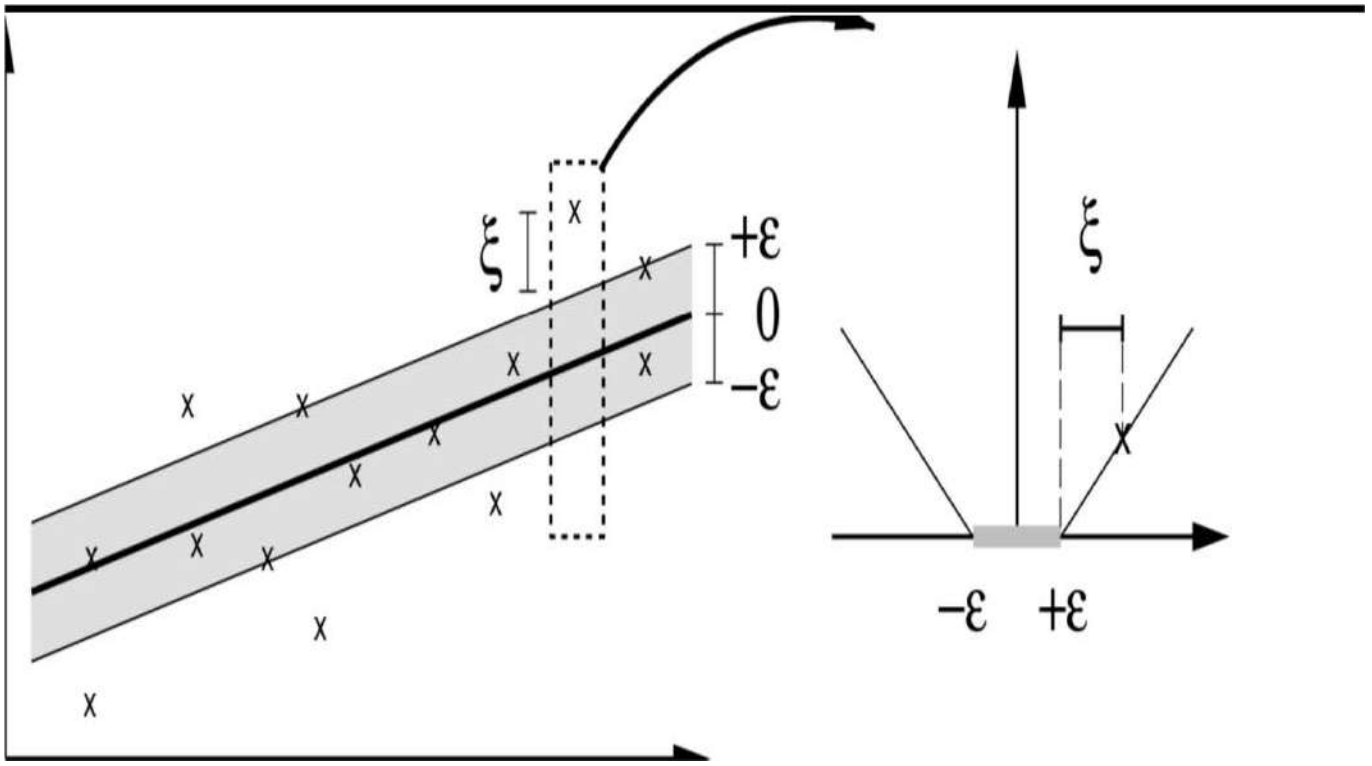
Support Vector Machines

SVMs for regression

Here we choose the ε-**insensitive** loss:

$$L(t_i, \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle) = |t_i - g(\boldsymbol{x}_i)|_\varepsilon = \max(|t_i - g(\boldsymbol{x}_i)| - \varepsilon, 0)$$

where $g(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$



Support Vector Machines

SVMs for regression

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

cost param

subject to
$$\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b - t_i \leq \varepsilon + \xi_i,$$
$$t_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b \leq \varepsilon + \xi_i^*,$$
$$\xi_i, \xi_i^* \geq 0$$

where the ξ_i, ξ_i^* are again slack variables controlling the "violations"

Support Vector Machines

SVMs for regression

Then feature maps $\phi(\cdot)$ are introduced, the primal optimization problem is transformed into the dual, and kernelised, to give:

$$y_{\text{SVM}}(\boldsymbol{x}) = \sum_{i=1}^{n} \beta_i k(\boldsymbol{x}, \boldsymbol{x}_i)$$

if $\xi = 0 \Rightarrow \beta = 0$, (not SV)
otherwise $\neq 0$ (is SV)
$\hookrightarrow$ only use SVs

$$= \sum_{i}^{n} \beta_i \langle \phi(x), \phi(x_i) \rangle = \sum \beta_i \, x^T x_i$$

with $0 \leq \alpha_i, \alpha_i^* \leq C$

For convenience, we have defined $\beta_i := \alpha_i - \alpha_i^*$.

Support Vector Machines

SVMs for regression

A closer look at the structure of the solution:

- Data points that end up **within** the ε-tube have inactive slacks (*i.e.*, $\xi_i = \xi_i^* = 0$) and therefore $\beta_i = 0$ (**not SVs**) between lines

- Data points that end up **not within** the ε-tube have exactly one active slack (*i.e.*, either $\xi_i > 0$ and $\xi_i^* = 0$, or vice versa) and therefore $\beta_i \neq 0$ (**non-bound SVs**) One on line, other not

- Data points that end up **outside** the ε-tube have exactly one bound slack (*i.e.*, $\xi_i = C$ and $\xi_i^* = 0$, or vice versa) and therefore $\beta_i \neq 0$ (**bound SVs**) one on line, one outside

$(0, \infty)$ penalizer

Support Vector Machines

SVMs for regression

In comparison to ridge regression, the only difference is in the choice of the loss (since both are **regularized machines** and both are amenable to **kernelisation**) and its consequences:

- Deviations lower than ε are ignored

dev $< \varepsilon$ ignored

- The loss grows linearly (and not quadratically) in the residual, making it more robust against outliers

← ridge

- The solution is **sparse** (the number of **basis functions** $\phi(\boldsymbol{x}_i)$ is automatically adapted)

Support Vector Machines

C versus ε

- C determines the trade off between model complexity (flatness) and tolerance to deviations larger than ε

- ε controls the width of the ε-insensitive tube

Larger ε or C implies less SVs (while smaller ε or C implies more SVs); but larger ε gives flatter models while larger C implies more complex models

---

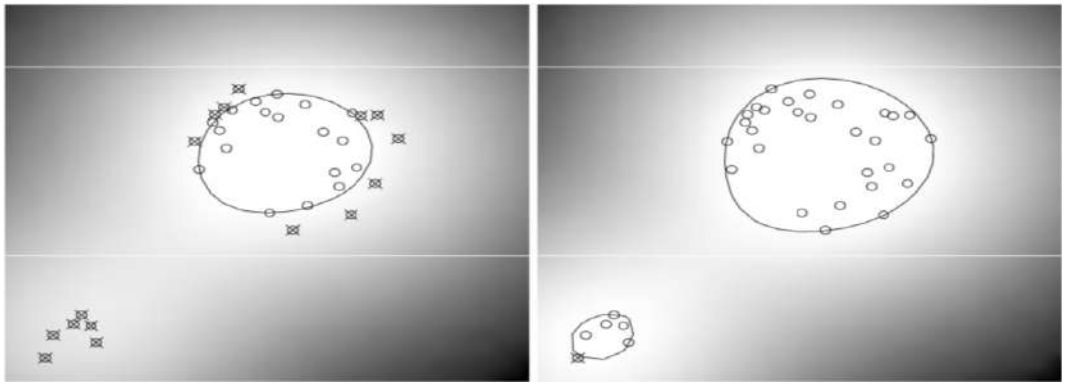
Hence, both parameters affect model complexity and number of SVs (but in a different way).

Support Vector Machines

SVMs for novelty detection (I)

- You are given a dataset drawn from a pdf $p(\boldsymbol{x})$; the $\boldsymbol{x}$ can be handwritten digits (recognizable/strange), process status (normal/faulty), credit card transactions (normal/fraudulent), ...

- The goal is to estimate a "simple" subset S of input space s.t. the *probability* that a test point drawn from p lies *outside* S equals some a priori specified $\rho \in (0, 1)$:



–from Alex Smola: Hilbert Space Methods: Basics, Applications, Open Problems
`http://alex.smola.org/talks/rsisesvm.pdf`

Support Vector Machines

SVMs for novelty detection (II)

USPS dataset of handwritten digits: $9,298$ digit images of size 16×16

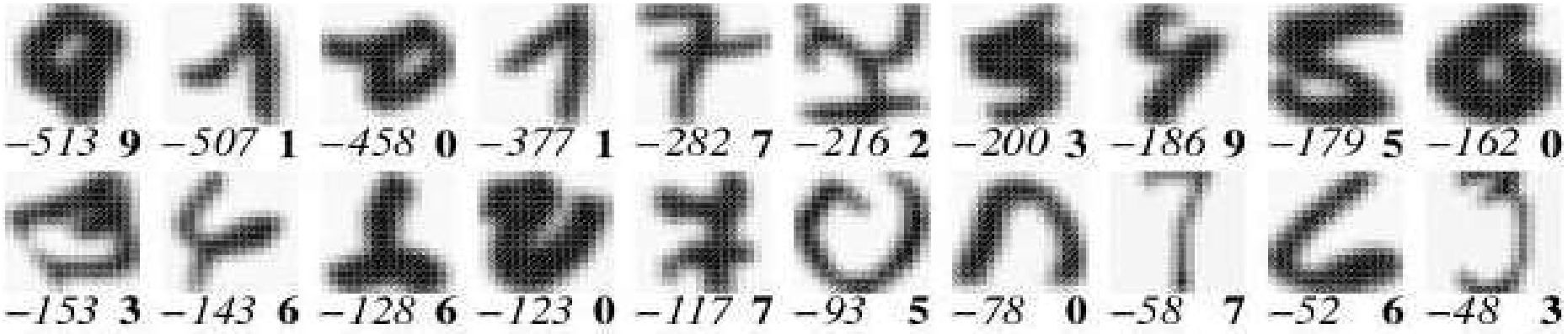


Figure 2: Outliers identified by the proposed algorithm, ranked by the negative output of the SVM (the argument of the sgn in the decision function). The outputs (for convenience in units of 10^{-5}) are written underneath each image in italics, the (alleged) class labels are given in bold face. Note that most of the examples are "difficult" in that they are either atypical or even mislabelled.

The 20 worst outliers for the USPS test set (here $\rho = 0.05$)

–from Schölkopf et al, *Support Vector Method for Novelty Detection*, NIPS'2000

Support Vector Machines

Tricks of the trade for the kernel

1. **Standardizing** the variables is in general good (assumed numerical)

$$x_S = \frac{x - \mu_x}{\sigma_x}$$

2. Kernel matrices close to the identity or close the the "all-ones" matrix are also an indication of bad kernel parameter: avoid these situations

3. Kernel matrix values should not be very large or very small (or both): if so, **normalize** the kernel matrix

$$k_n(x,x') = \frac{k(x,x')}{\sqrt{k(x,x')}\sqrt{k(x,x')}} = \frac{\langle \phi(x), \phi(x') \rangle}{\|\phi(x)\| \cdot \|\phi(x')\|} = \cos \langle \phi(x), \phi(x') \rangle$$

$\in [-1, 1]$

Kernel-Based Learning & Multivariate Modeling

MIRI Master

Lluís A. Belanche
belanche@cs.upc.edu

Soft Computing Research Group

Universitat Politècnica de Catalunya

2021-2022

Kernel-Based Learning & Multivariate Modeling

Syllabus

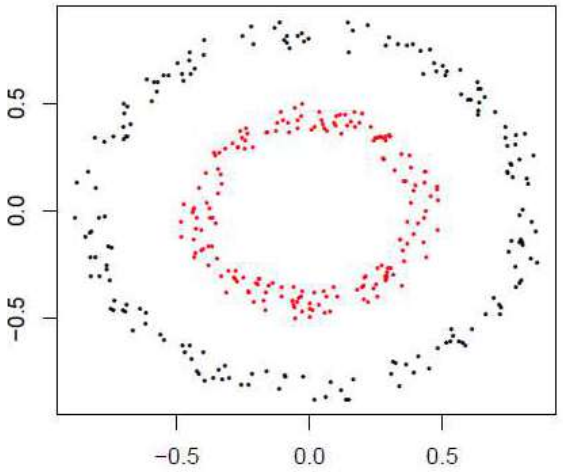
Part 1

Sep 15 Introduction to kernel-based learning
Sep 22 The SVM for classification, regression & novelty detection (I)
Sep 29 The SVM for classification, regression & novelty detection (II)
Oct 06 Kernel design (I): theoretical issues
Oct 13 Kernel design (II): practical issues
Oct 20 Kernelizing ML & statistical algorithms
Oct 27 Advanced topics

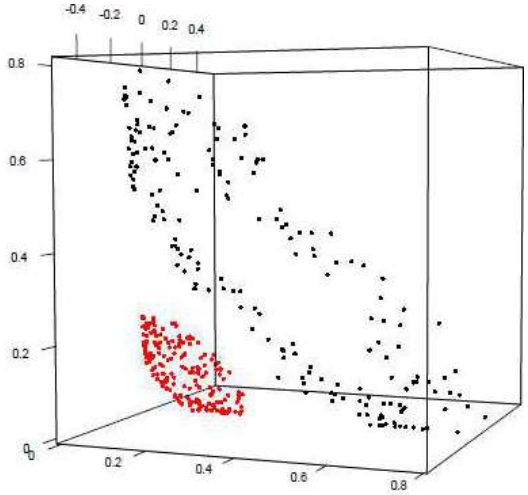
Kernel design (I): theoretical issues

General feature maps

Recall the idea of mapping input data into some Hilbert space (called the *feature space*) via a non-linear mapping $\phi : \mathcal{X} \to \mathcal{H}$



(a) Input Space (data not linearly separable)



(b) Feature Space (data linearly separable)

Kernel design (I): theoretical issues

Hilbert spaces

An abstract complete **vector space** endowed with an inner product:

Inner product requires symmetry, bilinearity and PSD-ness

Completeness means all Cauchy sequences converge to an element within the space (w.r.t. the norm induced by the inner product)

Show by adding & multiplying elem in $\mathcal{H}$ -> result in $\mathcal{H}$

Kernel design (I): theoretical issues

Characterization of Kernels

Given a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, which properties make it a valid kernel function for ML?

$\Rightarrow$ existence of a map $\phi : \mathcal{X} \to \mathcal{H}$ s.t.

1. $\mathcal{H}$ is a Hilbert space and

2. $k(\boldsymbol{x}, \boldsymbol{x'}) = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle_{\mathcal{H}}$ holds?

Kernel design (I): theoretical issues

Characterization of Kernels

A symmetric function k is called **positive semi-definite** (PSD) in $\mathcal{X}$ if:

for every $n \in \mathbb{N}$, and every choice $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in \mathcal{X}$,

the Gram matrix $\mathbf{K} = (k_{ij})$, where $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, is PSD.

Theorem. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ admits the existence of a map $\phi : \mathcal{X} \to \mathcal{H}$ s.t. $\mathcal{H}$ is a Hilbert space and $k(\boldsymbol{x}, \boldsymbol{x'}) = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle_{\mathcal{H}}$ if and only if k is a symmetric and PSD function in $\mathcal{X}$.

Prove example on next slide

$\Rightarrow$ We depart from a $\phi: \mathcal{X} \to \mathcal{H}$. We define $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$

$$(x, x') \mapsto k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$$

- Symmetry ✓
- $\forall m \in \mathbb{N}, \forall x_1, \dots, x_n, x_i \in \mathcal{X}$,
  $\forall c \in \mathbb{R}^n$, $c^T K c = \sum_{i=1}^{n} \sum_{j=1}^{n} c_i k_{ij} c_j$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}} c_j$$

$$= \langle \sum_{i=1}^{n} c_i \phi(x_i), \sum_{j=1}^{n} c_j \phi(x_j) \rangle_{\mathcal{H}}$$

$$= \| \sum_{i=1}^{n} c_i \phi(x_i) \|^2 \geq 0$$

PSD

alles in $\|\cdot\|^2$ packen, weil ≥ 0

Kernel design (I): theoretical issues

On positive semi-definiteness

There are many **equivalent characterizations** of the PSD property for real symmetric matrices. Here are some: $A_{n\times n}$ is PSD if and only if ...

1. all of its eigenvalues are non-negative $A \cdot \vec{v} = \lambda \cdot \vec{v}$ (eigenvalue)

2. the determinants of all of its leading principal minors are non-negative
   (+ ↘ −) oder Dreiecksmatrix Diagonale mult — of all matrixes ixi w/ i=1...n

3. there is a PSD matrix B such that $BB^{\mathsf{T}} = A$ (this matrix is unique, denoted with $B = A^{1/2}$, and called the *principal square root* of A)

4. $\forall \boldsymbol{c} \in \mathbb{R}^n,\ \boldsymbol{c}^{\mathsf{T}} A \boldsymbol{c} \geq 0$ the def used by prof on prev slide
   bzw Quadrat in Eig ideal

Kernel design (I): theoretical issues

Generating the inner product

Given a kernel k symmetric and PSD, consider the space of functions:

$$\phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}}$$
$$\boldsymbol{x} \mapsto \phi(\boldsymbol{x}) : k(\boldsymbol{x}, \cdot)$$

stand-in für beliebigen Wert

Define the (soon-to-be) vector space

$$\mathcal{H}_{\text{pre}} := \text{span}\{\phi(\boldsymbol{x}) / \ \boldsymbol{x} \in \mathcal{X}\}$$

set of all linear combis

all possible finite combos

$$= \left\{ f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \cdot) / \ n \in \mathbb{N}, \boldsymbol{x}_i \in \mathcal{X}, \alpha_i \in \mathbb{R} \right\}$$

intersection of in subspaces

linear combi's of $\phi(x) = k(x_i, \cdot)$

Kernel design (I): theoretical issues

Generating the inner product

Let $f, g \in \mathcal{H}_{\text{pre}}$; define an **inner product** in $\mathcal{H}_{\text{pre}}$ as

$$\langle f, g \rangle = \left\langle \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \cdot), \sum_{j=1}^{m} \beta_j k(\boldsymbol{x}'_j, \cdot) \right\rangle := \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k(\boldsymbol{x}_i, \boldsymbol{x}'_j)$$

Note that $\langle f, k(\boldsymbol{x}, \cdot) \rangle = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) = f(\boldsymbol{x})$

$f(\cdot) = \sum \alpha_i k(x_i, \cdot)$

This is called the **reproducing property** of the kernel

Kernel design (I): theoretical issues

Generating the inner product

Let's check we have a valid inner product space:

1. $\langle f, g \rangle = \langle g, f \rangle$ (symmetry)

2. $\langle f, g \rangle = \sum_{i=1}^{n} \alpha_i g(\boldsymbol{x}_i) = \sum_{j=1}^{m} \beta_j f(\boldsymbol{x}'_j)$ (bilinearity)

   Funktionen sind gleich mit bestimmter Gewichtung

3. $\langle f, f \rangle \geq 0$ with equality iff f is the zero function (PSD-ness)

   almost always f(x)=0

This inner product satisfies the Cauchy-Schwartz inequality:

$$|\langle f, g \rangle| \leq \sqrt{\langle f, f \rangle} \cdot \sqrt{\langle g, g \rangle}, \quad \forall f, g \in \mathcal{H}_{\text{pre}}$$

Kernel design (I): theoretical issues

Generating the inner product

1. Once we have an inner product, we have a **norm** $\|f\| := \sqrt{\langle f, f \rangle}$

2. Moreover, we have a **metric** $d(f,g) := \|f - g\|$ $= \sqrt{\langle f-g, f-g \rangle}$

3. For any metric space, one can construct a **complete** metric space which contains the former as a dense subspace*; if completion is applied to an inner product space, the result is a Hilbert space $\mathcal{H}$

(*): Let (X,d) be a metric space, and $X_0 \subset X$. Then X_0 is dense in X if and only if $\forall x \in X$ there is a sequence of points $x_n \in X_0$ that has limit x.

Kernel design (I): theoretical issues

The Kernel Trick

Such a space is called a **Reproducing Kernel Hilbert Space** (RKHS)

Given the mapping $\phi : \mathcal{X} \to \mathcal{H}$, the **kernel trick** consists in performing the mapping and the inner product simultaneously by defining its associated kernel function:

inner prod w/o map info

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \Big\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \Big\rangle_{\mathcal{H}}, \quad \boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}$$

This way it is possible to compute inner products in $\mathcal{H}$ without explicitly performing/knowing the map (*e.g.* Gram matrices, the OSH)

We depart from kernel function: symmetric & PSD
Let's build associated Hilbert space.

Build Hilbert space (same as slides but sum on 1 page)

1) Let $\mathcal{H} := \emptyset$

2) $\forall x \in X$, add the function $k_x : X \to \mathbb{R}^X$
$x \mapsto k(x, \cdot)$

3) Make $\mathcal{H}$ the span of the set in 2), that is
$$\mathcal{H} := \text{span}\{k_x / x \in X\} = \{f(\cdot) = \sum_{i=1}^{n} \alpha_i k_{x_i}(\cdot) / \alpha_i \in \mathbb{R}, x_i \in X, i = 1, \dots, m\}$$

4) Define inner prod in this space
For f, g as $f = \sum_{i=1}^{n} \alpha_i k_{x_i}$, $g = \sum_{j=1}^{m} \beta_j k_{x_j}$ $\to$ $\langle \sum_{i=1}^{n} \alpha_i k_{x_i}, \sum_{j=1}^{m} \beta_j k_{x_j} \rangle_H := \sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_i \beta_j k(x_i, x_j)$

5) We define a norm & a distance
- $\forall f \in \mathcal{H}$, $\|f\|_{\mathcal{H}} := \sqrt{\langle f, f \rangle_H}$
- $\forall f, g$, $d_H(f, g) := \|f - g\|_H$
- add to H the limits of all converging seq (complete)
↳ HS of entirely generated by $k \to H_k$

Norm
1) $\|z\| = 0 \iff z = 0$
2) $\|a \cdot z\| = |a| \cdot \|z\|$
3) $\|z + t\| \leq \|z\| + \|t\|$

Property of HS generated this way

$\forall f \in \mathcal{H}_k$. $\langle f, k_x \rangle_{H_k} = f(x)$, the reproducing property of the kernel

⇒ Reproducing kernel Hilbert Space (RKHS)

Take $x, x' \in X$, define $\phi : X \to \mathbb{R}$, $x \mapsto \phi(x) := k_x$

$\langle \phi(x), \phi(x') \rangle_{H_k} = \langle k_x, k_{x'} \rangle_{H_k} = k(x, x') = k_x(x')$

Kernel design (I): theoretical issues

The Kernel Trick: an example

Take $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle^q$, for $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$. What is the underlying feature map ϕ?

$\Rightarrow$ Answer: the space spanned by all products of exactly q dimensions of $\mathbb{R}^d$.

Example: $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^3$, and $q = 2$:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle^2 = \left\langle \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} \right\rangle^2$$

write how inner prod is calc

$$= (x_1 x'_1 + x_2 x'_2 + x_3 x'_3)^2 = (x_1 x'_1 + x_2 x'_2)^2 + 2(x_1 x'_1 + x_2 x'_2) x_3 x'_3 + (x_3 x'_3)^2$$

d lang w/ jewels q

$$= \left\langle \begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ \sqrt{2} x_1 x_3 \\ \sqrt{2} x_2 x_3 \\ x_2^2 \\ x_3^2 \end{pmatrix}, \begin{pmatrix} (x'_1)^2 \\ \sqrt{2} x'_1 x'_2 \\ \sqrt{2} x'_1 x'_3 \\ \sqrt{2} x'_2 x'_3 \\ (x'_2)^2 \\ (x'_3)^2 \end{pmatrix} \right\rangle$$

d-q lang

$$= \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$$

bring it in form $\langle \phi(x), \phi(x') \rangle$

Kernel design (I): theoretical issues

Popular choices for the Kernel

Polynomial kernels (relation to GLDs)

$$k(\boldsymbol{x}, \boldsymbol{x'}) = (a \langle \boldsymbol{x}, \boldsymbol{x'} \rangle + c)^q, \ q \in \mathbb{N}, a > 0, c \geq 0 \in \mathbb{R}$$

Gaussian kernels (relation to RBFNNs)

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \exp\left(-\gamma \|\boldsymbol{x} - \boldsymbol{x'}\|^2\right), \ \gamma > 0 \in \mathbb{R}$$

Laplacian kernels (relation to ???)

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \exp\left(-\gamma \|\boldsymbol{x} - \boldsymbol{x'}\|\right), \ \gamma > 0 \in \mathbb{R}$$

Sigmoidal kernels (relation to MLPs)

$$k(\boldsymbol{x}, \boldsymbol{x'}) = g(a \langle \boldsymbol{x}, \boldsymbol{x'} \rangle + c)$$

with g a sigmoidal (*e.g.*, logistic, tanh, ...) and particular choices for a, c

Kernel design (I): theoretical issues

Kernel construction

Which **operations** (*e.g.*, products, sums, composition, etc) on kernels produce new kernels? (*closure properties*)

Example:

Consider functions $p : \mathbb{R} \to \mathbb{R}$.

If k is a kernel, when is $p \circ k$ a kernel?

$p(x) = \sum_{i=1}^{q} \underbrace{a_i}_{\geq 0} x^q$, $a_q \neq 0$

$p(z) = (z+1)^q$

$k(x, x') = (x^T x' + 1)^q$

Kernel design (I): theoretical issues

Closure properties

- Inner products: finite (sums), infinite countable (series) or infinite uncountable (integrals)
- Scalar operations, sums and direct sums
- Products and tensor products
- Limits of point-wise convergent sequences
- Composition with certain analytic functions
- Normalization

Kernel design (I): theoretical issues

Inner products

1. Let $f_1, \ldots, f_n : \mathcal{X} \to \mathbb{R}$ be a finite collection of functions:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{n} f_i(\boldsymbol{x}) \cdot f_i(\boldsymbol{x}')$$

2. Let $\{f_n\}_n$ be a sequence of functions $\mathcal{X} \to \mathbb{R}$; if the series is convergent:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{n=1}^{\infty} f_n(\boldsymbol{x}) \cdot f_n(\boldsymbol{x}')$$

3. Let $f : \mathcal{X} \times W \to \mathbb{R}$ be a parameterized (indexed) set of functions; if the integral is well-defined:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \int_W f(\boldsymbol{x}; \boldsymbol{w}) \cdot f(\boldsymbol{x}'; \boldsymbol{w}) \, d\boldsymbol{w}$$

Kernel design (I): theoretical issues

Scalar operations, sums and direct sums

Take $k_1, k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $k' : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ kernels

$k_1, k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \Rightarrow k_1(x, x') + k_2(x, x')$ sum

$k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
$k_2 : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$
$\Rightarrow k((x,y),(x',y')) = k_1(x,x') + k_2(y,y')$ direct sum

- $a \cdot k_1(\boldsymbol{x}, \boldsymbol{x'}) + b$, $a > 0, b \geq 0$

- $k_+ : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ defined as

$$k_+(\boldsymbol{x}, \boldsymbol{x'}) = k_1(\boldsymbol{x}, \boldsymbol{x'}) + k_2(\boldsymbol{x}, \boldsymbol{x'})$$

- $k_\oplus : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$ defined as

$$k_\oplus((\boldsymbol{x}, \boldsymbol{y}), (\boldsymbol{x'}, \boldsymbol{y'})) = k_1(\boldsymbol{x}, \boldsymbol{x'}) + k'(\boldsymbol{y}, \boldsymbol{y'})$$

Kernel design (I): theoretical issues

Products and tensor products

Take $k_1, k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $k' : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ kernels

- $k_.$: $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$ defined as

$$k_.(\boldsymbol{x}, \boldsymbol{x'}) = k_1(\boldsymbol{x}, \boldsymbol{x'}) \cdot k_2(\boldsymbol{x}, \boldsymbol{x'})$$

- $k_\odot$: $(\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$ defined as

$$k_\odot((\boldsymbol{x}, \boldsymbol{y}), (\boldsymbol{x'}, \boldsymbol{y'})) = k_1(\boldsymbol{x}, \boldsymbol{x'}) \cdot k'(\boldsymbol{y}, \boldsymbol{y'})$$

Kernel design (I): theoretical issues

Limits of sequences

Let $\{k_n\}_n : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a sequence of kernels; if, for all $\boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}$, the limit exists,

then $k_\infty : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ defined as

$$k_\infty(\boldsymbol{x}, \boldsymbol{x'}) := \lim_{n \to \infty} k_n(\boldsymbol{x}, \boldsymbol{x'}), \ \forall \boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}$$

is a valid kernel.

Kernel design (I): theoretical issues

Composition with analytic functions

Theorem. Let f be a real analytic function with radius of convergence $R > 0$ s.t. all the coefficients in its power series expansion are non-negative. Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel fulfilling $|k(\boldsymbol{x}, \boldsymbol{x'})| < R$.

Then $k_f : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ given by $k_f(\boldsymbol{x}, \boldsymbol{x'}) := f(k(\boldsymbol{x}, \boldsymbol{x'}))$ is a valid kernel.

Example: $f(z) = \exp(z)$

A real function f is *analytic* in an open set $\Omega \subset \mathbb{R}$ iff for every $x_0 \in \Omega$ there is a neighborhood of x_0 for which the Taylor series expansion of f in x_0 coincides with $f(x)$.

Kernel design (I): theoretical issues

Operations in feature space

Norms in feature space:

$$||\phi(\boldsymbol{x})||_{\mathcal{H}} = \sqrt{\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}) \rangle_{\mathcal{H}}} = \sqrt{k(\boldsymbol{x}, \boldsymbol{x})}$$

Norms of linear combinations in feature space:

$$\left\| \sum_i \alpha_i \phi(\boldsymbol{x}_i) \right\|_{\mathcal{H}}^2 = \langle K\boldsymbol{\alpha}, \boldsymbol{\alpha} \rangle = \boldsymbol{\alpha}^\top K \boldsymbol{\alpha}$$

$\langle \alpha_i \phi(x_i), \alpha_i \phi(x_i) \rangle$ $\quad \langle k(x_i, x_i)\alpha, \alpha \rangle = \alpha^T k(x, x)\alpha$

Kernel design (I): theoretical issues

Operations in feature space

Distances in feature space:

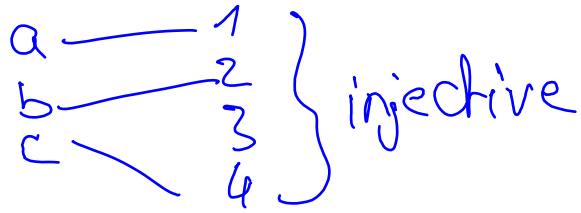
$$\|\phi(x)-\phi(x')\|^2 = \|\phi(x)\|^2 + \|\phi\|^2 - 2\langle\phi(x),\phi(x')\rangle$$
$$= \langle\phi(x),\phi(x)\rangle + \langle\phi(x'),\phi(x')\rangle - 2\langle\phi(x),\phi(x')\rangle = k(x,x)+k(x',x') - 2k(x,x')$$

$$\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x'})\|_{\mathcal{H}} = \sqrt{\langle\phi(\boldsymbol{x}),\phi(\boldsymbol{x})\rangle_{\mathcal{H}} + \left\langle\phi(\boldsymbol{x'}),\phi(\boldsymbol{x'})\right\rangle_{\mathcal{H}} - 2\left\langle\phi(\boldsymbol{x}),\phi(\boldsymbol{x'})\right\rangle_{\mathcal{H}}}$$

and then $d_{\mathcal{H}}(\boldsymbol{x},\boldsymbol{x'}) := \sqrt{k(\boldsymbol{x},\boldsymbol{x}) + k(\boldsymbol{x'},\boldsymbol{x'}) - 2k(\boldsymbol{x},\boldsymbol{x'})}$ is an Euclidean metric (distance) when ϕ is injective (otherwise it would be a pseudo-metric).

Suppose ϕ not injective & I have a kernel k s.t. $k(x,x)=1 \quad \forall x,x' \in \mathcal{X}$

$\Rightarrow d_{\mathcal{H}}(x,x') = \sqrt{1+1-2k(x,x')} = \sqrt{2}\cdot\sqrt{1-k(x,x')} \propto \sqrt{1-k(x,x')}$



Kernel design (I): theoretical issues

Normalizing kernels

If k is a kernel, then so is:

$$k_n(\boldsymbol{x}, \boldsymbol{x'}) := \frac{k(\boldsymbol{x}, \boldsymbol{x'})}{\sqrt{k(\boldsymbol{x}, \boldsymbol{x})} \cdot \sqrt{k(\boldsymbol{x'}, \boldsymbol{x'})}}$$

Moreover, $|k_n(\boldsymbol{x}, \boldsymbol{x'})| \leq 1$ and $k_n(\boldsymbol{x}, \boldsymbol{x}) = 1$.

The effect is to project each point onto the unit sphere, since

$$1 = k_n(\boldsymbol{x}, \boldsymbol{x}) = \langle \phi_n(\boldsymbol{x}), \phi_n(\boldsymbol{x}) \rangle = ||\phi_n(\boldsymbol{x})||^2$$

Kernel design (I): theoretical issues

General linear kernel

Theorem. If $A_{d\times d}$ is a PSD matrix, then the function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ given by $k(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{x}^\mathsf{T} A \boldsymbol{x'}$ is a kernel.

Proof. Since A is PSD we can write it in the form $A = BB^\mathsf{T}$. For every $n \in \mathbb{N}$, and every choice $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in \mathbb{R}^d$, we form the matrix $\mathbf{K} = (k_{ij})$, where $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\mathsf{T} A \boldsymbol{x}_j$. Then for every $\boldsymbol{c} \in \mathbb{R}^n$:

$$= x_i^\mathsf{T} B B^\mathsf{T} x_j = B^\mathsf{T} x_i \cdot B^\mathsf{T} x_j$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \boldsymbol{x}_i^\mathsf{T} A \boldsymbol{x}_j = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j (B^\mathsf{T}\boldsymbol{x}_i)^\mathsf{T}(B^\mathsf{T}\boldsymbol{x}_j)$$

$$= \left\| \sum_{i=1}^{n} c_i (B^\mathsf{T}\boldsymbol{x}_i) \right\|^2 \geq 0. \qquad \text{Note that } \phi(\boldsymbol{x}) = B^\mathsf{T}\boldsymbol{x}$$

Kernel design (I): theoretical issues

Polynomial kernels

1. If k is a kernel and p is a (non-zero) polynomial of degree q with non-negative coefficients, then the function

$$k_p(\boldsymbol{x}, \boldsymbol{x'}) := p(k(\boldsymbol{x}, \boldsymbol{x'}))$$

   is also a kernel.

2. The special case where k is linear and $p(z) = (az + c)^q, a > 0, c \geq 0 \in \mathbb{R}$ leads to the so-called **polynomial kernel**

Kernel design (I): theoretical issues

Translation invariant and radial kernels

We say that a kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is:

Translation invariant if it has the form $k(\boldsymbol{x}, \boldsymbol{x'}) = T(\boldsymbol{x} - \boldsymbol{x'})$, where $T : \mathbb{R}^d \to \mathbb{R}$ is a differentiable function

Radial if it has the form $k(\boldsymbol{x}, \boldsymbol{x'}) = t(\|\boldsymbol{x} - \boldsymbol{x'}\|)$, where $t : [0, \infty) \to [0, \infty)$ is a differentiable function

Radial kernels fulfill $k(\boldsymbol{x}, \boldsymbol{x}) = t(0)$.

Kernel design (I): theoretical issues

The Gaussian kernel

Consider the function $t(z) = \exp(-\gamma z^2), \gamma > 0$. The resulting radial kernel is known as the **Gaussian RBF kernel**:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{x'}\|^2)$$

Many people refer to it simply as "the RBF kernel"

You can also find it as:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x'}\|^2}{2\sigma^2}\right)$$

Kernel design (I): theoretical issues

Using the exponential

1. If k is a kernel and $\gamma > 0$, then the function

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \exp(\gamma k(\boldsymbol{x}, \boldsymbol{x'}))$$

is also a kernel.

2. If k is a kernel and $\gamma > 0$, then the function

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \exp\Big(- \gamma[k(\boldsymbol{x}, \boldsymbol{x}) + k(\boldsymbol{x'}, \boldsymbol{x'}) - 2k(\boldsymbol{x}, \boldsymbol{x'})]\Big)$$

is also a kernel.

Kernel design (I): theoretical issues

Characterization of Kernels

A symmetric function k is called **conditionally positive semi-definite** (CPSD) in $\mathcal{X}$ if for every $n \in \mathbb{N}$, and every choice $x_1, \cdots, x_n \in \mathcal{X}$, the matrix $\mathbf{K} = (k_{ij})$, where $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, is CPSD.

A real symmetric matrix $A_{n \times n}$ is CPSD if and only if $\forall \boldsymbol{c} \in \mathbb{R}^n$ such that $\boldsymbol{c}^\top \mathbf{1} = 0$, $\boldsymbol{c}^\top A \boldsymbol{c} \geq 0$.

new part

It turns out that it suffices for a kernel to be CPSD! Since the class of CPSD kernels is larger than that of PSD kernels:

1. a larger set of kernel functions are usable by kernel machines

2. a larger set of learning algorithms are prone to kernelization

Kernel-Based Learning & Multivariate Modeling

MIRI Master

Lluís A. Belanche
belanche@cs.upc.edu

Soft Computing Research Group

Universitat Politècnica de Catalunya

2021-2022

Kernel-Based Learning & Multivariate Modeling

Syllabus

Part 1

Sep 15 Introduction to kernel-based learning
Sep 22 The SVM for classification, regression & novelty detection (I)
Sep 29 The SVM for classification, regression & novelty detection (II)
Oct 06 Kernel design (I): theoretical issues
Oct 13 Kernel design (II): practical issues
Oct 20 Kernelizing ML & statistical algorithms
Oct 27 Advanced topics

Kernel design (II): practical issues

Euclidean space $\mathbb{R}^d$, but not only ...

- Kernels on real vectors (whole families)
- Kernels on binary vectors (bitstrings = sets)
- General structured kernels:
  - All-subsets kernel
  - Convolution kernels
- Kernels on discrete structures:
  - Tree kernels
  - Graph kernels
- Kernels on distributions (generative kernels):
  - P-kernels
  - Marginalized kernels
- String kernels (text)

... and many others (functional data, categorical data, ...)

Kernel design (II): practical issues

All-subsets kernel

Consider a feature space with one feature for every subset $A \subseteq \{1, \ldots, d\}$ of the input variables:

For $\boldsymbol{x} \in \mathbb{R}^d$, feature A is given by $\phi_A(\boldsymbol{x}) = \prod_{i \in A} x_i$ (note $\phi_\emptyset(\boldsymbol{x}) = 1$)

The kernel is defined by the mapping $\phi : \boldsymbol{x} \to (\phi_A(\boldsymbol{x}))_{A \subseteq \{1,\ldots,d\}}$

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle = \sum_{A \subseteq \{1,\ldots,d\}} \phi_A(\boldsymbol{x}) \phi_A(\boldsymbol{x'})$$

$$= \sum_{A \subseteq \{1,\ldots,d\}} \prod_{i \in A} x_i x'_i = \prod_{i=1}^{d} (1 + x_i x'_i)$$

The last step is obtained by expanding $(1 + x_1 x'_1)(1 + x_2 x'_2) \ldots (1 + x_d x'_d)$

Kernel design (II): practical issues

All-subsets kernel

We have the freedom to downplay some features (and thus emphasize others) by introducing weighting factors $w_i \geq 0$ for each feature i:

$$\phi_A(\boldsymbol{x}) = \prod_{i \in A} \sqrt{w_i} x_i$$

therefore

$$k_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{x'}) = \prod_{i=1}^{d} (1 + w_i x_i x'_i)$$

Kernel design (II): practical issues

Bitstring/Binary variables/Sets

Let $\boldsymbol{x}, \boldsymbol{x'} \in \{0,1\}^d$, representing absence/presence of a binary trait:

1. The Simple Matching Coefficient (SMC) is the fraction of $1-1$ matches, and it is a kernel on $\{0,1\}^d$.

$$\frac{\#\text{ matching values}}{\#\text{all values}}$$

Proof. For every $n \in \mathbb{N}$, and every choice $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \{0,1\}^d$, we form the matrix $K = (k_{ij})$, where $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{d}\boldsymbol{x}_i^\top \boldsymbol{x}_j$.

2. The Jaccard Coefficient is the fraction of $1-1$ matches among the traits present in either data vector, and it is a kernel on $\{0,1\}^d$.

$$\frac{|A \cap B|}{|A \cup B|}$$

Kernel design (II): practical issues

Bitstring/Binary variables/Sets

- Given two sets $A, B \subset U$, where U is finite, consider

$$k(A,B) = \frac{1}{|U|} \sum_{a \in A} \sum_{b \in B} k_{\text{base}}(a,b)$$

- If k_{base} is the overlap kernel $k(a,b) = \begin{cases} 1 & \text{if } a = b; \\ 0 & \text{otherwise.} \end{cases}$

  we get $k(A,B) = \frac{|A \cap B|}{|U|}$, the equivalent of the SMC.

- The equivalent of the Jaccard kernel would be $k(A,B) = \frac{|A \cap B|}{|A \cup B|}$.

Kernel design (II): practical issues

Generative kernels

Given a probability distribution on $\mathcal{X} \times \mathcal{Z}$, we can compare data points by assigning a high value if both have high conditional probability:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \sum_{z \in \mathcal{Z}} p(\boldsymbol{x}|z) p(\boldsymbol{x'}|z) P(z) \qquad \text{discrete case}$$

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \int_{\mathcal{Z}} p(\boldsymbol{x}|z) p(\boldsymbol{x'}|z) p(z)\, dz \qquad \text{continuous case}$$

$p(x, x'|z)$

The feature maps are $(\phi(\boldsymbol{x}))_z = p(\boldsymbol{x}|z)\sqrt{p(z)}$

Idea: $p(\boldsymbol{x}, \boldsymbol{x'}, z) = p(\boldsymbol{x}, \boldsymbol{x'}|z)p(z) = p(\boldsymbol{x}|z)p(\boldsymbol{x'}|z)p(z)$

Kernel design (II): practical issues

Generative kernels

Given a probability distribution on $\mathcal{X} \times \mathcal{Z}$, and a kernel on $\mathcal{X} \times \mathcal{Z}$ pairs, we can define:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \sum_z \sum_{z'} k\Big((\boldsymbol{x}, z), (\boldsymbol{x'}, z')\Big) p(z|\boldsymbol{x}) p(z'|\boldsymbol{x'})$$

Typical applications of generative kernels are found in **graphical models**:

- $\mathcal{X}$ are the **observed** variables and $\mathcal{Z}$ are the **hidden** (latent) variables
- A kernel for the observed ones is obtained by taking the expectation w.r.t. the hidden ones (*marginalizing* them away)
- Examples: HMMs for sequences or stochastic context-free grammars for RNA sequences –see "Kernel methods in genomics and computational biology" by J.P. Vert

Kernel design (II): practical issues

The Spectrum (aka n-Gram) kernel

- Let Σ be a finite alphabet: an n-Gram is a block of n adjacent characters in Σ

  Define $k(\boldsymbol{x}, \boldsymbol{x'}) := \sum_{s \in \Sigma^n} |s \in \boldsymbol{x}| \cdot |s \in \boldsymbol{x'}|$ X is word

---

Example: Word `aababc` in alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, $n = 2$:

aa	ab	ac	ba	bb	bc ...
1	2	0	1	0	1 ...

- While the feature space is large, the feature vectors are sparse; this kernel can be computed in $O(|\boldsymbol{x}| + |\boldsymbol{x'}|)$ time and memory (the actual number of distinct n-Grams in a text is very small)

Kernel design (II): practical issues

Kernels from graphs

- Consider a graph $G = (V, E)$, where the set of vertices (nodes) V are the data points and E is the set of edges. Call $N = |V|$, the number of nodes
- The idea is to compute a (base) matrix $S_{N \times N}$ whose entries are the weights of the edges and consider $S^2 = SS$ (S need not be symmetric)
- Typical use: **connectivity matrix** of G: the (i, j) element of S^2 is the number of paths of length exactly 2 between i and j

---

Examples:

1. protein-protein interactions
2. people-to-people interactions

In 2, the (i, j) element of S^2 is the number of common friends between data points i and j (it can be thought of as a measure of their similarity)

Kernel design (II): practical issues

Kernels from graphs

Notes:

- The entries of S may be real-valued numbers (e.g., symmetric bounded similarities)
- Higher powers of S measure higher-order similarities
- Only the even powers are guaranteed to be PSD

Consider, for a given $\lambda \in (0, 1)$:

$$\sum_{k=0}^{\infty} \frac{1}{k!} \lambda^k S^k = \exp(\lambda S)$$

1. If S is symmetric, then $S = U \Lambda U^T$ (spectral decomposition), so $S^2 = (U \Lambda U^T)(U \Lambda U^T) = U \Lambda^2 U^T$.

2. In general, we have $S^k = U \Lambda^k U^T$ and therefore:

$$K := \exp(\lambda S) = U \exp(\lambda \Lambda) U^T$$

   is an example of a **diffusion** kernel.

Example in a real application domain

Handling missing values in microbiology

- Modern modelling problems are difficult for a number of reasons, including the challenge of dealing with a significant amount of missing information

- **Missing values** almost always represent a serious problem because they force to preprocess the dataset and a good deal of effort is normally put in this part of the modelling

- In order to process such datasets with kernel methods, an imputation procedure is then deemed a necessary but demanding step

Example in a real application domain

Handling missing values in microbiology

- The study of fecal source pollution in waterbodies is a major problem in ensuring the welfare of human populations

- **Microbial source tracking** (MST) methods attempt to identify the source of contamination, allowing for improved risk analysis and better water management

- The available dataset includes 148 observations about 10 chemical, microbial, and eukaryotic markers of fecal pollution in water

- All variables (except the class variable) are binary, i.e., they signal the presence or absence of a particular marker

Example in a real application domain

Handling missing values in microbiology

Origin	HF183	HF134	CF128	Humito	Pomito	Bomito	ADO	DEN
Human :50	0 :68	0 :81	0 :104	0 :35	0 :83	0 :78	0 :56	0 :80
Cow :26	1 :40	1 :26	1 : 5	1 :79	1 :32	1 :32	1 :59	1 :34
Poultry:31	**?**:31	**?**:32	**?**:30	**?**:25	**?**:24	**?**:29	**?**:24	**?**:25
Pig :32								

Summary (counts) table for the full dataset. The first column is the target class. The symbol **?** denotes a missing value.

The percentage of missing values is around 19.8 %, and all the predictive variables have percentages between 17 % and 23 %

Example in a real application domain

Handling missing values in microbiology

Let the symbol **?** denote a missing element, for which only equality is defined. Let $k : X \times X \to \mathbb{R}$ be a symmetric kernel in X and P a probability mass function (PMF) in X. Then the function $k^{?}(x, y)$ given by

$$k^{?}(x,y) = \begin{cases} k(x,y), & \text{if } x, y \neq ? \ ; \\ g(x) = \sum\limits_{y' \in X} P(y')k(x,y'), & \text{if } x \neq ? \text{ and } y = ?; \\ g(y) = \sum\limits_{x' \in X} P(x')k(x',y), & \text{if } x = ? \text{ and } y \neq ?; \\ G = \sum\limits_{x' \in X} P(x') \sum\limits_{y' \in X} P(y')k(x',y'), & \text{if } x = y = ? \end{cases}$$

is a kernel in $X \cup \{?\}$.

Example in a real application domain

Handling missing values in microbiology

For the particular case of binary variables $x, y \in \{v_1, v_2\}$, a convenient approach is to define the kernel:

$$k_{0/1}(x, y) = \mathbb{I}_{\{x=y\}}$$

where

$$\mathbb{I}_{\{z\}} = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{if } z \text{ is false} \end{cases}$$

Example in a real application domain

Handling missing values in microbiology

Consider now $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^d$. When we apply the Theorem to this kernel, we obtain an extended multivariate kernel:

$$\mathcal{K}_1(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{d}\sum_{i=1}^{d} \begin{cases} 1 & \text{if } x_i = y_i = 1 \ ; \\ P_i(x_i), & \text{if } x_i \neq \textbf{?} \text{ and } y_i = \textbf{?}; \\ P_i(y_i), & \text{if } x_i = \textbf{?} \text{ and } y_i \neq \textbf{?}; \\ (P_i(0))^2 + (P_i(1))^2, & \text{if } x_i = y_i = \textbf{?}; \\ 0, & \text{otherwise} \end{cases}$$

This kernel is a generalization of the classical *Simple Matching Coefficient*, proposed by Sokal and Michener for numerical taxonomy

Example in a real application domain

Handling missing values in microbiology

Let the symbol **?** denote a missing element, for which only equality is defined. Let $k : X \times X \to \mathbb{R}$ be a symmetric kernel in $X = \{0, 1\}^d$. Let $c(\boldsymbol{x})$ be the set of completions of $\boldsymbol{x}$. Given two vectors $\boldsymbol{x}, \boldsymbol{y} \in X$, the function

$$\mathcal{K}_2(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{|c(\boldsymbol{x})||c(\boldsymbol{y})|} \sum_{\boldsymbol{x}' \in c(\boldsymbol{x})} \sum_{\boldsymbol{y}' \in c(\boldsymbol{y})} k(\boldsymbol{x}', \boldsymbol{y}') \tag{1}$$

is a kernel in $X \cup \{\mathbf{?}\}$.

Example in a real application domain

Handling missing values in microbiology

			10x10cv for each class			
Approach	C	10x10cv	Human	Cow	Poultry	Swine
$\mathcal{K}_1$	2.0	79.3	95.4	64.5	75.2	69.4
$\mathcal{K}_2$	1.6	78.2	92.6	62.8	71.8	74.2
MI-1	1.0	79.9	92.7	66.4	69.4	80.2
MI-2	1.0	79.0	94.5	57.5	70.8	78.8

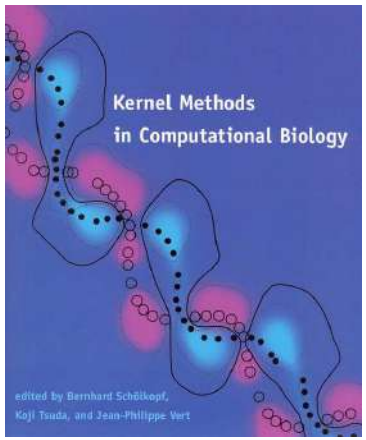
Mean 10x10cv accuracies for the four approaches to handle missing values. Also shown are best cost parameter C and detailed class performance.

(*joint work with G. Nebot, T. Aluja and V. Kobayashi*)

Kernel design (II): practical issues

More Kernels!

Kernels abound in computational biology and computational chemistry (e.g., phylogenetic profiles, protein 3D structures)



Example: the prediction of **interacting proteins** to reconstruct an interaction network can be posed as a binary classification problem: given a pair of proteins, do they interact or not?

$\longrightarrow$ we need kernel between *pairs* of proteins!

Kernel design (II): practical issues

More Kernels!

The available data is about each single protein; it is then natural to derive kernels for **pairs** of proteins k_{pair} from any kernel k for **single** proteins:

$$k_{\text{pair}}\Big((A,B),(C,D)\Big) := k(A,C)k(B,D) + k(A,D)k(B,C)$$

(there is usually no order in a protein pair, so we try both matches)

· Using Product Kernels to Predict Protein Interactions. *Advances in Biochemical Engineering/Biotechnology* (110), pp 215-245 (2007)

· Kernel methods for predicting protein-protein interactions. *Bioinformatics*. 2005

Kernel design (II): practical issues

Conclusions

- The power of kernel methods partly relies in the ability to process virtually any sort of data as soon as a valid kernel is defined
- Importance of designing kernels that do not constitute explicit inner products between objects, and therefore fully exploit the kernel trick
- Possibility of learning the kernel function (or the kernel matrix) from the training data
- Theoretical analyses are needed on the implications of the kernel choice for the success of specific kernel-based methods

Kernel-Based Learning & Multivariate Modeling

MIRI Master

Lluís A. Belanche
belanche@cs.upc.edu

Soft Computing Research Group

Universitat Politècnica de Catalunya

2021-2022

Kernel-Based Learning & Multivariate Modeling

Syllabus

Part 1

Sep 15 Introduction to kernel-based learning
Sep 22 The SVM for classification, regression & novelty detection (I)
Sep 29 The SVM for classification, regression & novelty detection (II)
Oct 06 Kernel design (I): theoretical issues
Oct 13 Kernel design (II): practical issues
Oct 20 Kernelizing ML & statistical algorithms
Oct 27 Advanced topics

Kernelizing ML & stats algorithms

[Q] What is "kernelizing"?

[A] do the inner product/distance in feature space:

- If an algorithm is **inner product-based**, the idea is:
  1. substitute $\left\langle \boldsymbol{x}_i, \boldsymbol{x}_j \right\rangle$ by $\left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle_{\mathcal{H}}$
  2. replace this by $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$

- If an algorithm is (Euclidean) **distance-based**, the idea is:
  1. substitute $||\boldsymbol{x}_i - \boldsymbol{x}_j||$ by $||\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)||_{\mathcal{H}}$
  2. replace this by $\sqrt{k(\boldsymbol{x}_i, \boldsymbol{x}_i) + k(\boldsymbol{x}_j, \boldsymbol{x}_j) - 2k(\boldsymbol{x}_i, \boldsymbol{x}_j)}$

Kernelizing ML & stats algorithms

A problem with the inner product

- If two data points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are translated as:

$$\begin{cases} \boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \boldsymbol{x}_0 \\ \boldsymbol{x}_j \leftarrow \boldsymbol{x}_j - \boldsymbol{x}_0 \end{cases}$$

  then the inner product $\left\langle \boldsymbol{x}_i, \boldsymbol{x}_j \right\rangle$ changes substantially

- This is not suitable for algorithms that should be **translation-invariant** (e.g. PCA), unless we center the data in feature space

Kernelizing ML & stats algorithms

Conditional positive semi-definiteness

A symmetric function k is called **conditionally positive semi-definite** (CPSD) in $\mathcal{X}$ if for every $n \in \mathbb{N}$, and every choice $x_1, \ldots, x_n \in \mathcal{X}$, the matrix $\mathbf{K} = (k_{ij})$, where $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is CPSD.

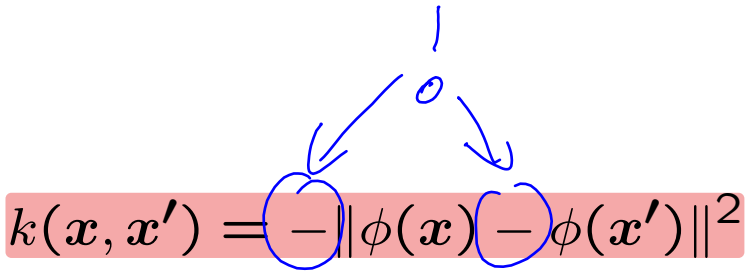
A real symmetric matrix $A_{n \times n}$ is CPSD if and only if $\forall \boldsymbol{c} \in \mathbb{R}^n$ such that $\boldsymbol{c}^\mathsf{T}\mathbf{1} = 0,\ \boldsymbol{c}^\mathsf{T} A \boldsymbol{c} \geq 0$.

Example: in $\mathcal{X} = \mathbb{R}^d$, $k(\boldsymbol{x}, \boldsymbol{x'}) = -\|\boldsymbol{x} - \boldsymbol{x'}\|^2$ is CPSD (but not PSD).
Exercise: For which values of β is $-\|\boldsymbol{x} - \boldsymbol{x'}\|^\beta$ CPSD? ¶

Kernelizing ML & stats algorithms

The kernel trick for distances

Theorem. Let k be a CPSD kernel on $\mathcal{X}$, satisfying $k(\boldsymbol{x}, \boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \mathcal{X}$. Then there exists a Hilbert space $\mathcal{H}$ and a mapping $\phi : \mathcal{X} \to \mathcal{H}$ such that



$$k(\boldsymbol{x}, \boldsymbol{x}') = -\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\|^2$$

It turns out that it suffices for a kernel to be CPSD! Since the class of CPSD kernels is larger than that of PSD kernels, a larger set of learning algorithms are prone to kernelization.

Kernelizing ML & stats algorithms

Example: k-nearest neighbours

(kNN classifies new examples by finding the k closest examples in the sample and taking a majority vote)

$$
\begin{aligned}
k(\boldsymbol{x},\boldsymbol{x}) + k(\boldsymbol{x'},\boldsymbol{x'}) - 2k(\boldsymbol{x},\boldsymbol{x'}) = \\
\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}) \rangle + \left\langle \phi(\boldsymbol{x'}), \phi(\boldsymbol{x'}) \right\rangle - 2\left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle = \\
\|\phi(\boldsymbol{x})\|^2 + \|\phi(\boldsymbol{x'})\|^2 - 2\left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle = \\
\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x'})\|^2 =: d^2_{\mathcal{H}}(\boldsymbol{x},\boldsymbol{x'})
\end{aligned}
$$

- This (square) Euclidean distance in *feature space* can be calculated using 3 calls to the kernel function (or 1 if k is normalized), for k PSD
- Note that $\sqrt{-k(\boldsymbol{x},\boldsymbol{x'})}$ is also a Euclidean distance, for k CPSD

Kernelizing ML & stats algorithms

Standard PCA

- We are given a data set $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, $\boldsymbol{x}_i \in \mathbb{R}^d$ for $i = 1, \ldots n$ which is centered around the origin, *i.e.* $\sum_{i=1}^{n} \boldsymbol{x}_i = 0$

- The sample covariance matrix C of the data is defined as

$$C = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathsf{T}}$$

Kernelizing ML & stats algorithms

Standard PCA

What are the PCs?

reduce dimensions -> simplify data

1. The goal of PCA is to replace the original axes with the principal components (PC) of the data

2. We choose the first PC as a **projection direction** such that the projections of the data onto it have **maximum variance**

3. All subsequent components can be defined as **orthogonal** projection directions with the next largest variance

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

- If we could define our principal components to be arbitrary **manifolds** we could get higher variance and better separability
- If we first (non-linearly) map our data onto a higher-dimensional space where the data falls neatly onto some hyperplane we can perform Standard PCA in that space
- These PCs will map back onto the arbitrary manifolds that we need in our lower-dimensional space

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

Kernel PCA allows us to perform PCA in this higher dimension using the kernel trick, doing all our calculations in a lower dimension.

Recall the idea of mapping input data into some Hilbert space (called the *feature space*) via a non-linear mapping $\phi : \mathcal{X} \to \mathcal{H}$:

$\to$ the new sample covariance matrix C of the data is given by

$$C = \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^\mathsf{T}$$

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

To find the first PC, again we need to solve $C\boldsymbol{v} = \lambda \boldsymbol{v}$. Therefore:

$$\frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \left(\phi(\boldsymbol{x}_i)^\top \boldsymbol{v} \right) = \lambda \boldsymbol{v}$$

Since $\lambda \neq 0$, $\boldsymbol{v}$ must be in the span of the set of vectors $\phi(\boldsymbol{x}_i)$, *i.e.* it can be written as some linear combination thereof:

$\rightarrow$ there must exist some α_i such that $\boldsymbol{v} = \sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{x}_i)$.

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

$$\lambda \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j) = \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \phi(\boldsymbol{x}_i)^\mathsf{T} \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j)$$

switch to our alternative notation ...

$$\lambda \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j) = \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \left\langle \phi(\boldsymbol{x}_i), \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j) \right\rangle$$

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

and now a trick ... introduce an inner product with $\phi(\boldsymbol{x}_l)$ for an arbitrary l on both sides ($1 \leq l \leq n$):

siehe slide vorher

$$\left\langle \lambda \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j), \phi(\boldsymbol{x}_l) \right\rangle = \lambda \sum_{j=1}^{n} \alpha_j \left\langle \phi(\boldsymbol{x}_j), \phi(\boldsymbol{x}_l) \right\rangle$$

$$\left\langle \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \left\langle \phi(\boldsymbol{x}_i), \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j) \right\rangle, \phi(\boldsymbol{x}_l) \right\rangle = \left\langle \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \sum_{j=1}^{n} \alpha_j \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle, \phi(\boldsymbol{x}_l) \right\rangle$$

$$= \left\langle \frac{1}{n} \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle, \phi(\boldsymbol{x}_l) \right\rangle = \frac{1}{n} \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{n} \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_l) \right\rangle \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle$$

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

Now we can introduce the kernel; since

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x'}) = \left\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \right\rangle_{\mathcal{H}}, \;\; \boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}$$

$$\lambda \sum_{j=1}^{n} \alpha_j \left\langle \phi(\boldsymbol{x}_j), \phi(\boldsymbol{x}_l) \right\rangle = \frac{1}{n} \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{n} \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_l) \rangle \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle$$

is rewritten as:

$$\lambda \sum_{j=1}^{n} \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}_l) = \frac{1}{n} \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{n} k(\boldsymbol{x}_i, \boldsymbol{x}_l) k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

Now let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^\mathsf{T}$ and $\mathbf{K} = (k_{ij})$, where $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$:

$$\lambda \mathbf{K} \boldsymbol{\alpha} = \frac{1}{n} \mathbf{K}^2 \boldsymbol{\alpha}$$

Again an eigenvalue problem! put $\boldsymbol{p} := \mathbf{K}\boldsymbol{\alpha}$ and rewrite:

$$\frac{1}{n} \mathbf{K} \boldsymbol{p} = \lambda \boldsymbol{p}$$

eigenvector

eigenvalue

where $\boldsymbol{p}, \lambda$ are the eigenvectors and eigenvalues of $\frac{1}{n}\mathbf{K}$ (we get rid of $\boldsymbol{\alpha}$)

$\frac{1}{n}\mathbf{K}$ has the same eigenvectors/values as $\mathbf{K}$, with eigenvalues scaled by n

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

- In Standard PCA, the matrix $XX^\top$ grows with the **dimension** of the data points d

- In Kernel PCA, the matrix $\mathbf{K}$ grows with the **number** of data points n; in consequence, we get n non-linear PCs

- When the kernel function is standard dot product, Kernel PCA solution reduces to Standard PCA

Kernelizing ML & stats algorithms

Kernel PCA from Standard PCA

- Centered data is required to perform an effective PCA

- Even though the $\{\boldsymbol{x}_i\}$ were centered, the $\{\phi(\boldsymbol{x}_i)\}$ are *not* guaranteed to be centered in feature space!

- We have to "center" the kernel matrix before doing Kernel PCA:

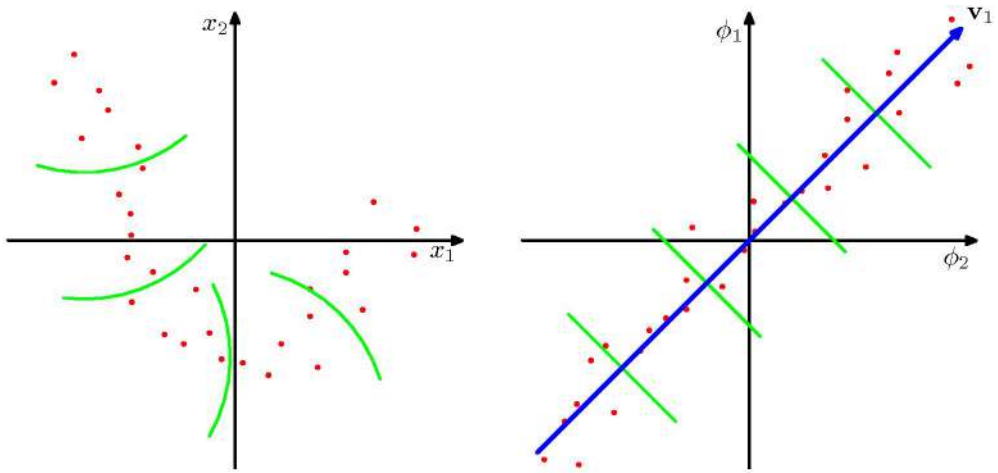
$$\mathbf{K} := \mathbf{K} - \frac{1}{n}\mathbf{1}\mathbf{K} - \frac{1}{n}\mathbf{K}\mathbf{1} + \frac{1}{n^2}\mathbf{1}\mathbf{K}\mathbf{1}$$

only these for correct dims

where $\mathbf{1}$ is a $n \times n$ matrix of ones

Kernelizing ML & stats algorithms

An illustration of KPCA



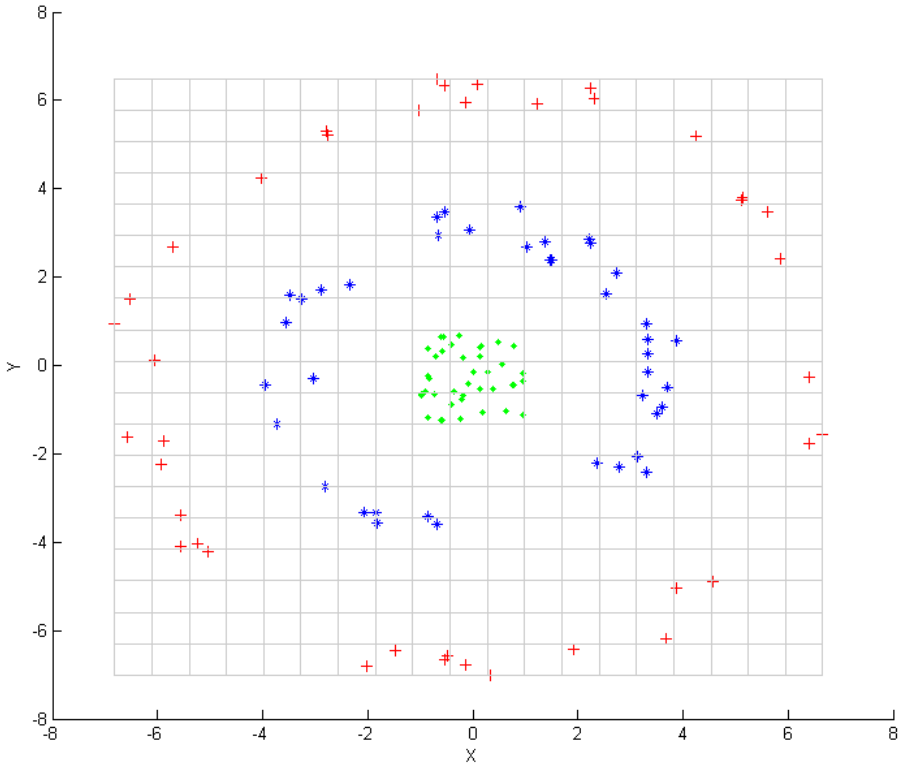
Left (input space): Green lines are desired non-linear projections

Right (feature space): Green lines are linear projections onto the first non-linear PC

(from *Pattern Recognition and Machine Learning*, C. M. Bishop, Springer, 2006)

Kernelizing ML & stats algorithms

Example (I)

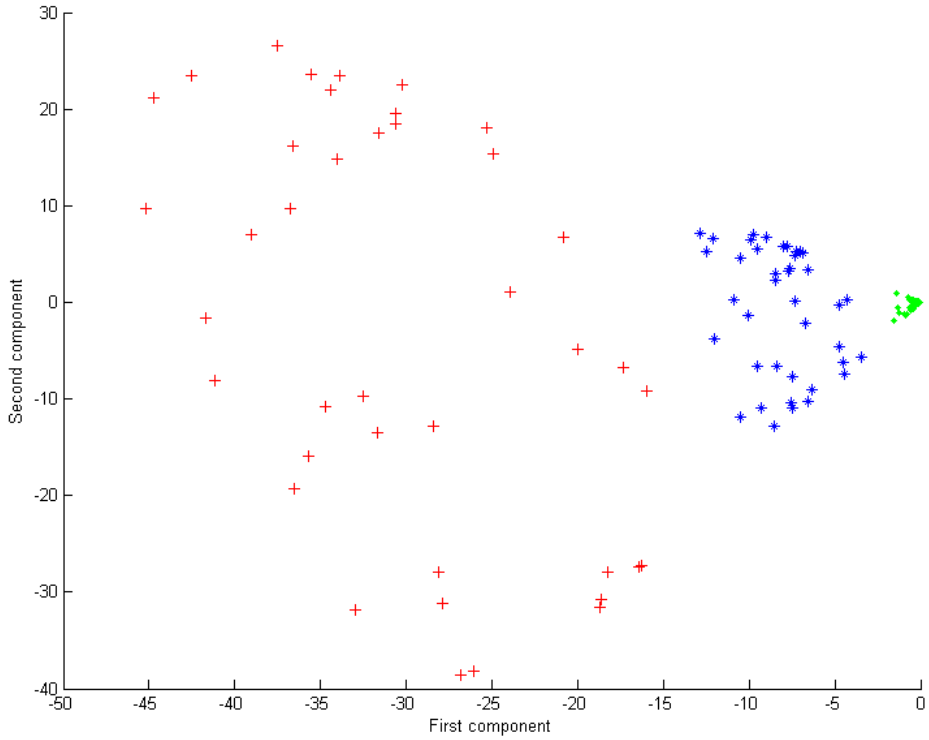


Three concentric clouds of points

(Source: Wikipedia, Petter Strandmark)

Kernelizing ML & stats algorithms

Example (II)



$$k(\boldsymbol{x}, \boldsymbol{x'}) = (\boldsymbol{x}^\top \boldsymbol{x'} + 1)^2$$

(the color of the points is not part of the algorithm; (Wikipedia, P. Strandmark)

Kernelizing ML & stats algorithms

Bad and good news

In Standard PCA, it is common to:

1. use the eigenvalues to rank the eigenvectors based on how much of the data variation is captured by each PC

2. estimate a "convenient" number of PCs and perform dimensionality reduction by projecting the original data onto the reduced set of eigenvectors

We cannot compute the kPCs (since they reside in the high-dimensional feature space), only the projections of our data onto the kPCs; however, we can perform new projections on test data (as in Standard PCA).

Kernelizing ML & stats algorithms

Summary of Standard PCA

1. We are given a data set of d-dimensional vectors $X = \{\boldsymbol{x}_i\}$ for $i = 1, \ldots, n$ which we **center** around the origin, as $\boldsymbol{x}_i := \boldsymbol{x}_i - \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j$

2. Compute the **sample covariance matrix** of the data as $C = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathsf{T}}$

3. Compute the **eigenvalues** and **eigenvectors** $(\lambda_i, \boldsymbol{v}_i)$ of C; **decide** a number $l < d$. **Let** $\widehat{\boldsymbol{x}}_i = V \boldsymbol{x}_i$, where $V = [\boldsymbol{v}_1; \ldots; \boldsymbol{v}_l]$

4. **Return** the l-dimensional data sample $\widehat{X} = \{\widehat{\boldsymbol{x}}_i\}$ for $i = 1, \ldots, n$

Kernelizing ML & stats algorithms

Summary of Kernel PCA

1. We are given a data set of objects $X = \{\boldsymbol{x}_i\}$ for $i = 1, \ldots, n$

2. Choose a **kernel** function k and compute the **kernel matrix** $\mathbf{K}$

3. **Center** the kernel matrix as: $\mathbf{K} := \mathbf{K} - \frac{1}{n}\mathbf{1}\mathbf{K} - \frac{1}{n}\mathbf{K}\mathbf{1} + \frac{1}{n^2}\mathbf{1}\mathbf{K}\mathbf{1}$

4. Compute the **eigenvalues** and **eigenvectors** $(\lambda_i, \boldsymbol{v}_i)$ of $\mathbf{K}$; **decide** a number $l < n$. Set $\boldsymbol{\alpha}_j = \boldsymbol{v}_j / \sqrt{\lambda_j}$ and let $\widehat{\boldsymbol{x}}_i = \left(\sum_{k=1}^{n} \boldsymbol{\alpha}_{jk}\, k(\boldsymbol{x}_i, \boldsymbol{x}_k) \right)_{j=1}^{l}$

5. **Return** the l-dimensional data sample $\widehat{X} = \{\widehat{\boldsymbol{x}}_i\}$ for $i = 1, \ldots, n$

Kernelizing ML & stats algorithms

Key aspects of kernel methods

Kernel-based methods consist of two ingredients:

1. The kernel function (this is non-trivial)

2. The algorithm taking kernels as input

   - Data items are embedded into a vector space where linear relations are computed
   - The coordinates of these images are not needed (and are usually unknown): only their pairwise inner products matter
   - These inner products can sometimes be computed efficiently and implicitly in the input space (kernel function)
   - This requires expressing a problem wherein the data appear in the form of inner products or distances
   - The solution vector is expressed as a linear combination of kernel evaluations centered at the data

Kernelizing ML & stats algorithms

Work so far ...

Many (classical and new) learning algorithms can be "kernelized":

- Support Vector Machine (SVM) and Relevance Vector Machine (SVM)
- Fisher Discriminant Analysis (kFDA), Principal Components Analysis (kPCA) and Canonical Correlation Analysis (kCCA), Independent Component Analysis (kICA)
- Kernel (regularized) linear and logistic regression
- kernel kNN (k Nearest Neighbours)
- Clustering (Spectral Clustering, Kernel k-means)
- PLS, Parzen Windows, Vector Quantization, ...
- Statistical indexes (e.g. Kendall's tau correlation coefficient to measure similarity between permutations)