Kernel-Based Learning & Multivariate Modeling

MIRI Master - DMKM Master

Lluís A. Belanche
`belanche@cs.upc.edu`

Soft Computing Research Group

Universitat Politècnica de Catalunya

2016-2017

Kernel-Based Learning & Multivariate Modeling

Contents by lecture

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 $A \subseteq \{1, \dots, d\}$ of the input variables:

for $\boldsymbol{x} \in \mathbb{R}^d$, the 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 $\phi : \boldsymbol{x} \to (\phi_A(\boldsymbol{x}))_{A \subseteq \{1,\dots,d\}}$

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

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

The last step is obtained by expanding $(1 + x_i x'_i)(1 + u_2 v_2) \dots (1 + u_d v_d)$

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

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 $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{d}\sum_{k=1}^{d} x_{ik}x_{jk} = \frac{1}{d}\left\langle \boldsymbol{x}_i, \boldsymbol{x}_j \right\rangle$

Then we have, for any $\boldsymbol{c} \in \mathbb{R}^N$:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j \frac{1}{d}\left\langle \boldsymbol{x}_i, \boldsymbol{x}_j \right\rangle = \frac{1}{d}\left\langle \sum_{i=1}^{N} c_i \boldsymbol{x}_i, \sum_{j=1}^{N} c_j \boldsymbol{x}_j \right\rangle = \frac{1}{d}\left\| \sum_{i=1}^{N} c_i \boldsymbol{x}_i \right\|^2 \geq 0$$

Kernel design (II): practical issues

Bitstring/Binary variables/Sets

Given two finite sets A, B, consider

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

If k_{base} is the overlap kernel:

$$k(a,b) = \begin{cases} 1 & \text{if } a = b; \\ 0 & \text{otherwise.} \end{cases}$$

we get $k(A,B) = |A \cap B|$, the same kernel as before.

Kernel design (II): practical issues

Convolution kernels

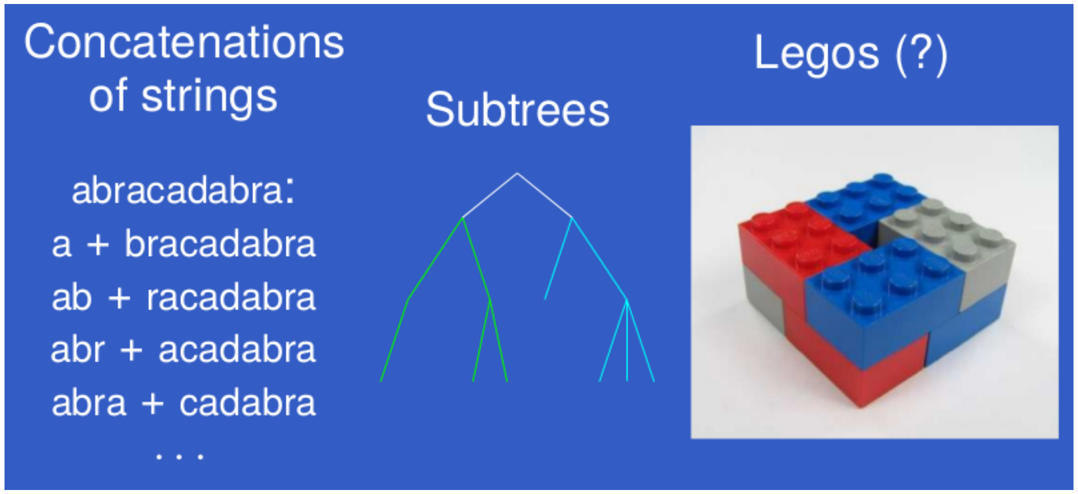
- Kernels between composite objects by decomposition on the their respective parts, compare components and combine results

- Given two composite objects $\boldsymbol{x}, \boldsymbol{x}'$ and a way $\mathcal{D}$ to obtain all possible decompositions into a finite number P of parts, their $\mathcal{D}$-convolution is a kernel:

$$k_{\mathcal{D}}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{\boldsymbol{x}_0 \in \mathcal{D}(\boldsymbol{x})} \sum_{\boldsymbol{x}'_0 \in \mathcal{D}(\boldsymbol{x}')} \prod_{p=1}^{P} k_p(x_0(p), x'_0(p))$$

Kernel design (II): practical issues

Convolution kernels



Examples of decomposable objects

Kernel design (II): practical issues

P-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:

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

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

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

Marginalized 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 both kernels are found on **graphical models**:

- $\mathcal{X}$ are the *observed* variables and $\mathcal{Z}$ are the *hidden* 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" (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 $k(\boldsymbol{x},\boldsymbol{x'}) := \sum_{s\in\Sigma^n} |s \in \boldsymbol{x}| \cdot |s \in \boldsymbol{x'}| = \sum_{s\in\Sigma^n} f_s(\boldsymbol{x}) f_s(\boldsymbol{x'})$

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 $B_{N\times N}$ whose entries are the weights of the edges and consider $B^2 = BB$ (B need not be symmetric)
- Typical use: **connectivity matrix** of G: the (i, j) element of B^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 B^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:

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

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

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

If $B = U \Lambda U^T$ is the spectral decomposition of B, then $B^2 = (U\Lambda U^T)(U\Lambda U^T) = U\Lambda^2 U^T$.
In general, we have $B^k = U\Lambda^k U^T$ and therefore:

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

is an example of a **diffusion** kernel (the name comes from the *heat equation* in physics)

Kernel design (II): practical issues

A worked example

Suppose we have designed a "kernel" on **image** objects $\boldsymbol{x}, \boldsymbol{x'}$ in $\mathcal{X} = [-1,+1]^d$ from a set of D descriptors $f_i : \mathcal{X} \to \mathbb{R}$ as the function:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \frac{\left(\sum_{i=1}^{D} f_i(\boldsymbol{x}) f_i(\boldsymbol{x'}) + \theta \right)^3}{\sqrt{\theta - \exp(-||\boldsymbol{x} - \boldsymbol{x'}||^2)}}$$

where $\theta \in \mathbb{R}$ is a free parameter.

Under what conditions does this expression define a kernel?

Kernel design (II): practical issues

A worked example

1. The numerator is simply the polynomial kernel of degree 3 applied to the descriptors and hence a kernel in itself for all $\theta \geq 0$

2. Since the denominator must be well-defined, we conclude that $\theta > 1$

3. Now define $f(z) = \dfrac{1}{\sqrt{\theta - z}}$, for $\theta > 1$

4. We find the Taylor series expansion for f as $f(z) = \sum_{n=0}^{\infty} a_n z^n$, with $a_n = \left(2^n \theta^{(2n+1)/2}\right)^{-1} \prod_{i=1}^{n} (2i - 1)$, which is positive for all $\theta > 0$

5. Therefore both the denominator and numerator are kernels and so is their product, for $\theta > 1$ (note the kernel is *not* normalized)

Kernel design (II): practical issues

Kernels as similarity measures

- A kernel can be conceptually regarded as a similarity measure on the dtata, though many kernels do not fulfill the classical properties for a similarity (e.g. boundedness)

- In a very general sense, a similarity $s : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ should at least satisfy the **Similarity Principle**:

$$s(x, x) > s(x, y) \geq 0,\ \forall x \neq y \in \mathcal{X}$$

- It seems natural to base kernel design on classical similarity functions, for example on **distance-based** similarities

- In many domains, data is described by a **mixture** of variable types (binary, categorical, real, ...)

Kernel design (II): practical issues

Gower-like similarity measures

- For any two vectors $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}$ to be compared on the basis of feature k, a score s_{ijk} is defined based on their type

- Set $\delta_{ijk} = 0$ when the comparison of $\boldsymbol{x}_i, \boldsymbol{x}_j$ cannot be performed on the basis of feature k (e.g. presence of missing values), or we want to ignore it; otherwise, let $\delta_{ijk} = 1$

Gower's **similarity** is defined as the average over all partial comparisons:

$$s_{ij} := s(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\sum_{k=1}^{d} s_{ijk}\delta_{ijk}}{\sum_{k=1}^{d} \delta_{ijk}}$$

Kernel design (II): practical issues

Gower-like similarity measures

Binary (dichotomous) variables indicate the presence/absence of a trait, marked by the symbols $+$ and $-$. Their similarities are given by:

	Values of feature k			
Object $\boldsymbol{x}_i$	$+$	$+$	$-$	$-$
Object $\boldsymbol{x}_j$	$+$	$-$	$+$	$-$
s_{ijk}	1	0	0	0
δ_{ijk}	1	1	1	0

Kernel design (II): practical issues

Gower-like similarity measures

- A **categorical** variable takes discrete and unordered values, which are commonly known as **modalities**

- No order should be defined (the only meaningful relation being equality / non-equality)

Their *overlap* is:

$$s_{ijk} = \mathbb{I}_{\{x_{ik}=x_{jk}\}} = \begin{cases} 1, & \text{if } x_{ik} = x_{jk}; \\ 0, & \text{if } x_{ik} \neq x_{jk} \end{cases}$$

Kernel design (II): practical issues

Gower-like similarity measures

Real-valued variables are compared with the standard metric in $\mathbb{R}$:

$$s_{ijk} = 1 - \frac{|x_{ik} - x_{jk}|}{R_k},$$

where R_k is the *range* of feature k (the difference between the maximum and minimum values)

Kernel design (II): practical issues

Gower-like similarity measures

Theorem. If there are no missing values, the similarity matrix $S = (s_{ij})$ is PSD (Gower, 1971).

Consider three objects in $[1, 5] \subset \mathbb{R}^4$, that is, $R_k = 4$ given by:

Feature no.	#1	#2	#3	#4
Object $\boldsymbol{x}_1$	1.0	2.0	3.0	1.0
Object $\boldsymbol{x}_2$	1.0	3.0	3.0	?
Object $\boldsymbol{x}_3$	1.0	3.0	3.0	5.0

Kernel design (II): practical issues

Gower-like similarity measures

Then we have

$$S = \begin{pmatrix} 1 & \frac{11}{12} & \frac{11}{16} \\ \frac{11}{12} & 1 & 1 \\ \frac{11}{16} & 1 & 1 \end{pmatrix}, \qquad \det(S) = -\frac{121}{2304} < 0$$

and therefore S is not PSD. However, if we replace ? by *any* value in $[1, 5]$, then the matrix S is certainly PSD.

Examples in real application domains

Classification of DNA sequences

- DNA sequences of promoter and non-promoters

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

- The dataset consists of 106 observations and 57 categorical variables describing the DNA sequence, represented as the nucleotide at each position: [A] adenine, [C] cytosine, [G] guanine, [T] thymine

- The response is a binary class: “$+$” for a promoter gene and “$-$” for a non-promoter gene

Examples in real application domains

Classification of DNA sequences

Overlap kernel: the similarity between two multivariate categorical vectors is the fraction of the number of variables in which they match:

$$k_{\mathrm{O}}(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{d} \sum_{l=1}^{d} \mathbb{I}_{\{x_l = x'_l\}}$$

Another kernel that can be used is the Gaussian RBF kernel:

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

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

Examples in real application domains

Classification of DNA sequences

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

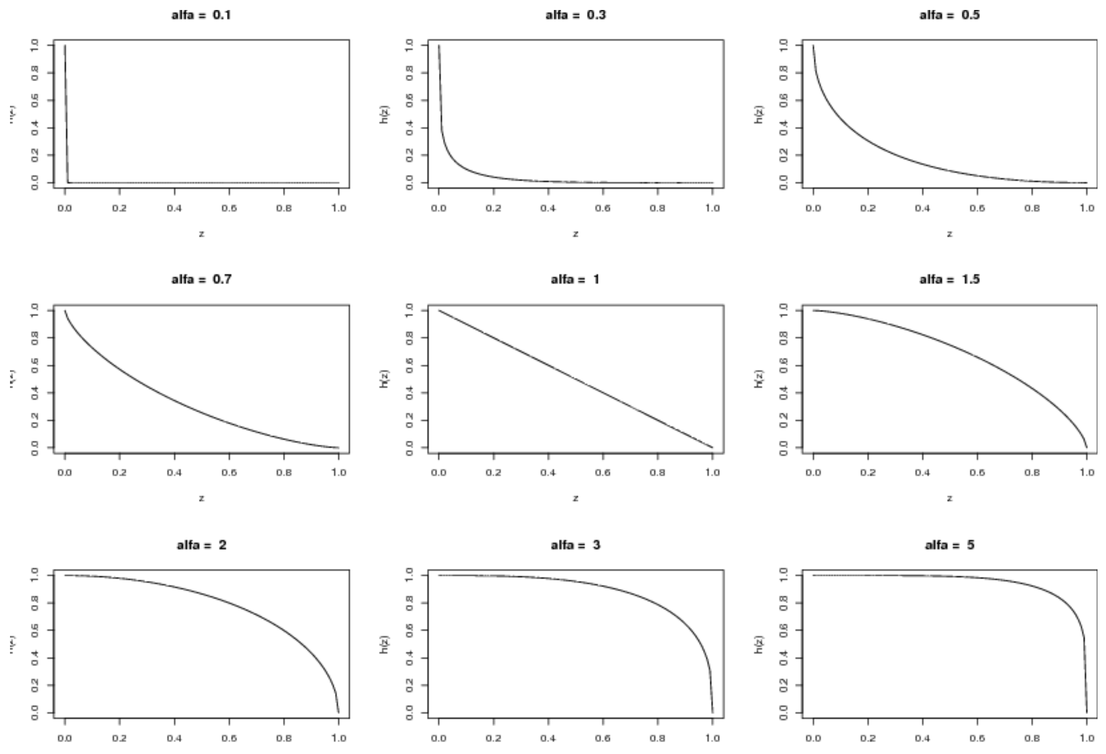
$$k_1^{(U)}(z_i, z_j) = \begin{cases} h_\alpha(P_Z(z_i)) & \text{if } z_i = z_j \\ 0 & \text{if } z_i \neq z_j \end{cases}$$

Inverting function:

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

Examples in real application domains

Classification of DNA sequences



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

Examples in real application domains

Classification of DNA sequences

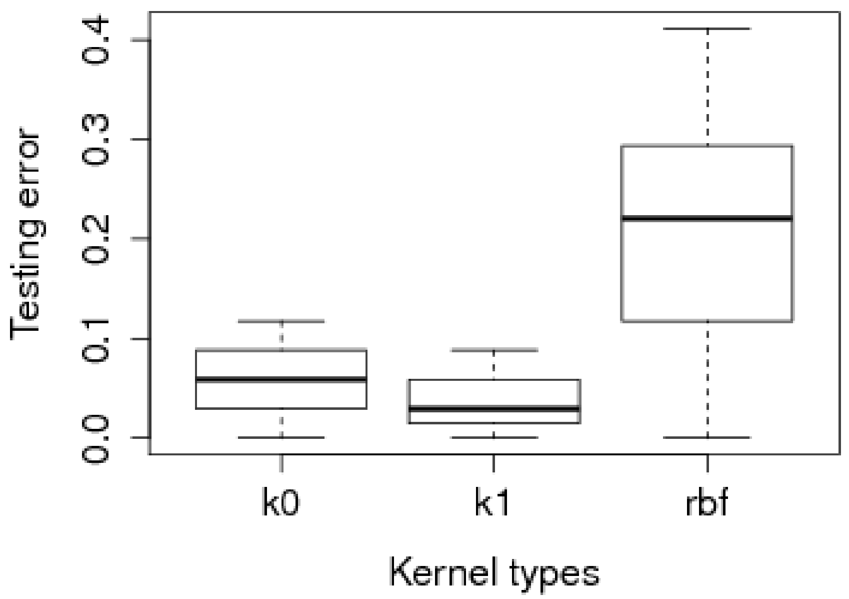
Multivariate kernel k_1:

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

Proposition. The kernel matrix generated by the multivariate kernel k_1 is PSD.

Examples in real application domains

Classification of DNA sequences



Test error distributions on the PromoterGene problem

(*joint work with M. Villegas*)

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