Kernels and representation

Corso di AA, anno 2018/19, Padova

Fabio Aiolli

12 Dicembre 2018

Outline

- Representation with kernels
- Kernel extensions
- Kernel learning

Representation with kernels

We are given a set of objects $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$. How can they be represented?

- Classical (explicit) representation: $\varphi(x) \to \mathcal{F}$
- Kernel (implicit) representation:
  - $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ (paired comparisons, symmetric function)
  - $\mathcal{S}$ represented by a symmetric matrix $\mathbf{K} = [K(x_i, x_j)]_{i,j} \in \mathbb{R}^{n \times n}$

Kernel and Gram matrix: definitions

Definition

A kernel function is a function $K(\cdot,\cdot)$ such that for all $\mathbf{x},\mathbf{z}\in\mathcal{X}$, satisfies $K(\mathbf{x},\mathbf{z})=\varphi(\mathbf{x})\cdot\varphi(\mathbf{z})$ where $\varphi(\mathbf{x})$ is a mapping from $\mathcal{X}$ to an (inner product or Hilbert) space $\mathcal{H}$.

Definition

The Gram (or kernel) matrix associated with the kernel function $K(\cdot,\cdot)$, evaluated on a finite subset of examples $X=\{\mathbf{x}_1,\ldots,\mathbf{x}_n\},\mathbf{x}_i\in\mathcal{X}$, is the matrix $\mathbf{K}\in\mathbb{R}^{n\times n}$ such that

$$\mathbf{K}_{i,j}=K(\mathbf{x}_i,\mathbf{x}_j).$$

The matrix $\mathbf{K}$ is symmetric and positive definite by definition.

A simple example

A first example:

$$
\begin{aligned}
\forall \mathbf{x}, \mathbf{z} \in \mathbb{R}^2, \; K(\mathbf{x}, \mathbf{z}) &= (\mathbf{x} \cdot \mathbf{z})^2 \\
&= (x_1 z_1 + x_2 z_2)^2 \\
&= \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{bmatrix} \cdot \begin{bmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2} z_1 z_2 \end{bmatrix} \\
&= \varphi(\mathbf{x}) \cdot \varphi(\mathbf{z})
\end{aligned}
$$

where $\varphi(\mathbf{x}), \varphi(\mathbf{z}) \in \mathbb{R}^3$

Advantages of using kernels

- Representation with kernel matrices has some advantages:
  - same algorithm for different typologies of data
  - modularity of the design of kernel and algorithms
  - the integration of different views is simpler
- The dimensionality of data depends on the number of objects and not from their vector dimensionality
- Comparison between objects can result computationally simpler than using the explicit object representation: kernel computation vs. dot product

Kernel methods

- Many kernel methods, including SVM, can be interpreted as solving the following problem:

$$\arg\min_{f \in \mathcal{H}} \mathcal{L}(f(x_1), \ldots, f(x_n)) + \Lambda ||f||_{\mathcal{H}}$$

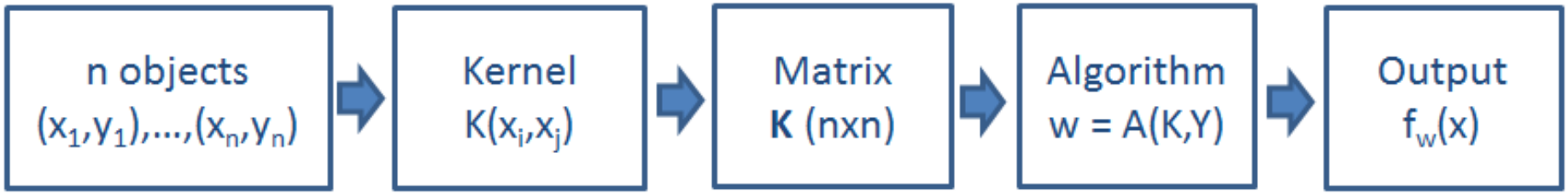
- $\mathcal{L}$ is a loss (or cost) function associated to the empirical risk
- The norm is the "smoothness" of the function. In fact, the meaning of "smoothness" depends on the considered kernel and feature space.
- Λ is a trade-off regularization coefficient

The problem above can be shown to always have a solution of type:

$$f(x) = \mathbf{w} \cdot \varphi(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$$

That is the optimization problem can be formulated with n variables. If $n \ll d$ then we get a computational advantage

Kernel Modularity



Modularity in the design/definition of the kernel (representation) and the learning algorithm used for model computation (classification, regression, ranking, etc.)

The Kernel Trick

Any algorithm for vectorial data which can be expressed in terms of dot-products can be implicitly executed in the feature space associated to a kernel, by simply replacing dot-products with kernel evaluations.

- Kernelization of popular linear or distance-based methods (e.g. Perceptron and kNN)
- Application of algorithms for vectorial data (SVM, Perceptron, etc.) to non-vectorial data using ad-hoc kernels (e.g. kernel for structures)

The Kernel Trick - Distances in feature space

Given two objects, $x, z \in \mathcal{X}$, the distance between the two objects in feature space is computed by:

$$d(x,z) = ||\varphi(x) - \varphi(z)||$$

$$\begin{aligned} d^2(x,z) &= ||\varphi(x) - \varphi(z)||^2 \\ &= \varphi(x) \cdot \varphi(x) + \varphi(z) \cdot \varphi(z) - 2\varphi(x) \cdot \varphi(z) \\ &= K(x,x) + K(z,z) - 2K(x,z) \end{aligned}$$

That is, $d(x,z) = \sqrt{K(x,x) + K(z,z) - 2K(x,z)}$.
Note that the values $\varphi(x), \varphi(z)$ are not explicitly used!

Common kernels for vectors

- Linear Kernel $K(\mathbf{x}, \mathbf{z}) = \mathbf{x} \cdot \mathbf{z}$
- Polynomial Kernel $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + u)^p$
- Exponential $K(\mathbf{x}, \mathbf{z}) = \exp(\mathbf{x} \cdot \mathbf{z})$
- Radial Basis Function (RBF) Kernel $K(\mathbf{x}, \mathbf{z}) = \exp(-\gamma ||\mathbf{x} - \mathbf{y}||^2)$

Polynomial Kernels

Homogeneous polynomial kernels $k(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^s$ can be constructed by defining an embedding map, indexed by all monomials of degree s:

$$\phi_{\mathbf{i}}(\mathbf{x}) = \prod_{k=1}^{n} x_k^{i_k}$$

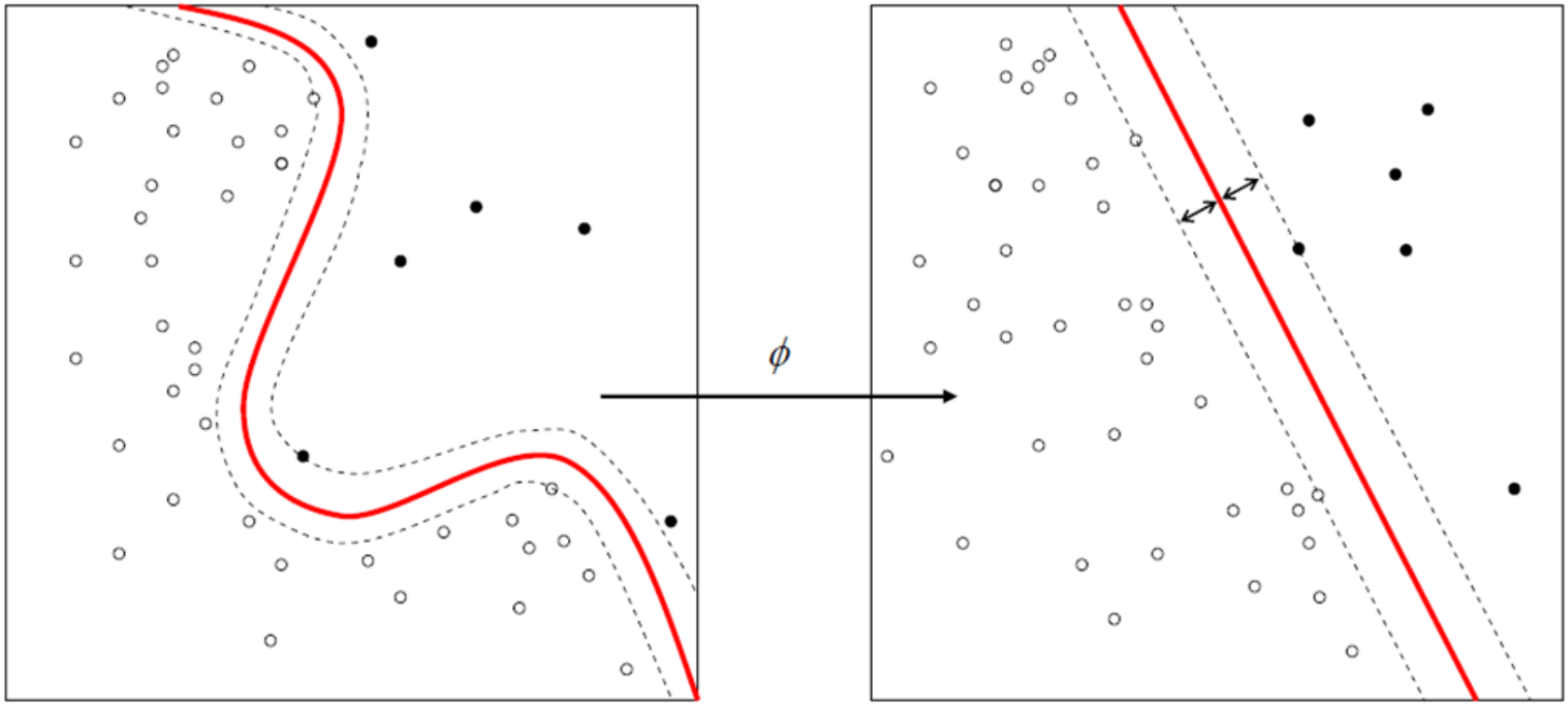
such that $\mathbf{i} = (i_1, \ldots, i_n)$ and $\sum_{k=1}^{n} i_k = s$

Non-homogeneous polynomial kernels $k(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + c)^d$ can be constructed by defining an embedding map, indexed by all monomials of degree less or equal to s:

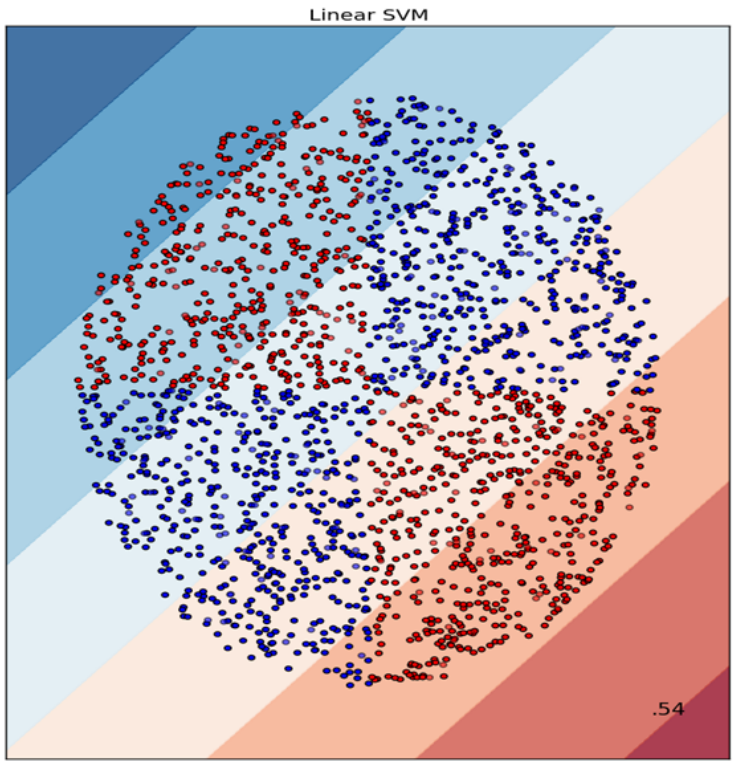
$$\phi_{\mathbf{i}}(\mathbf{x}) = \prod_{k=1}^{n} x_k^{i_k}$$

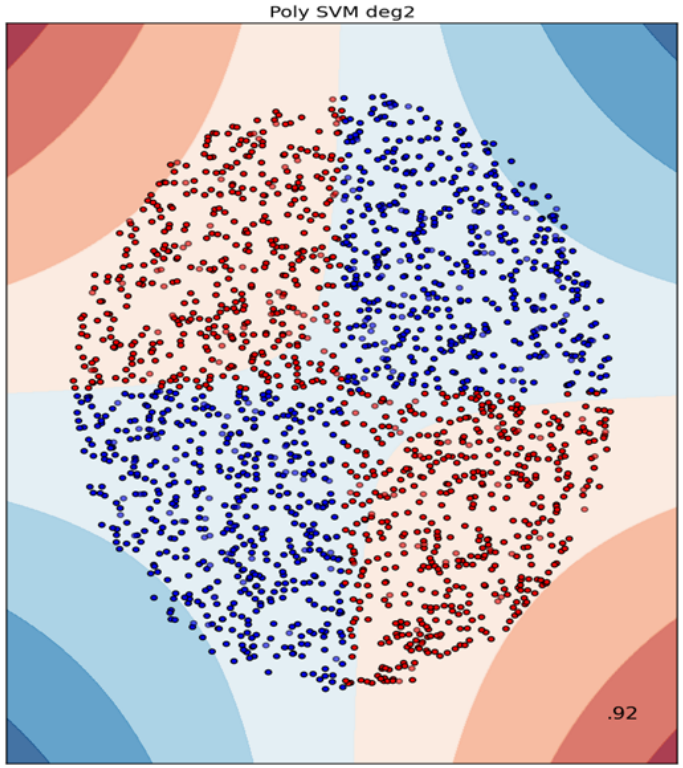
such that $\mathbf{i} = (i_1, \ldots, i_n)$ and $\sum_{k=1}^{n} i_k \leq d$

SVM with Kernels



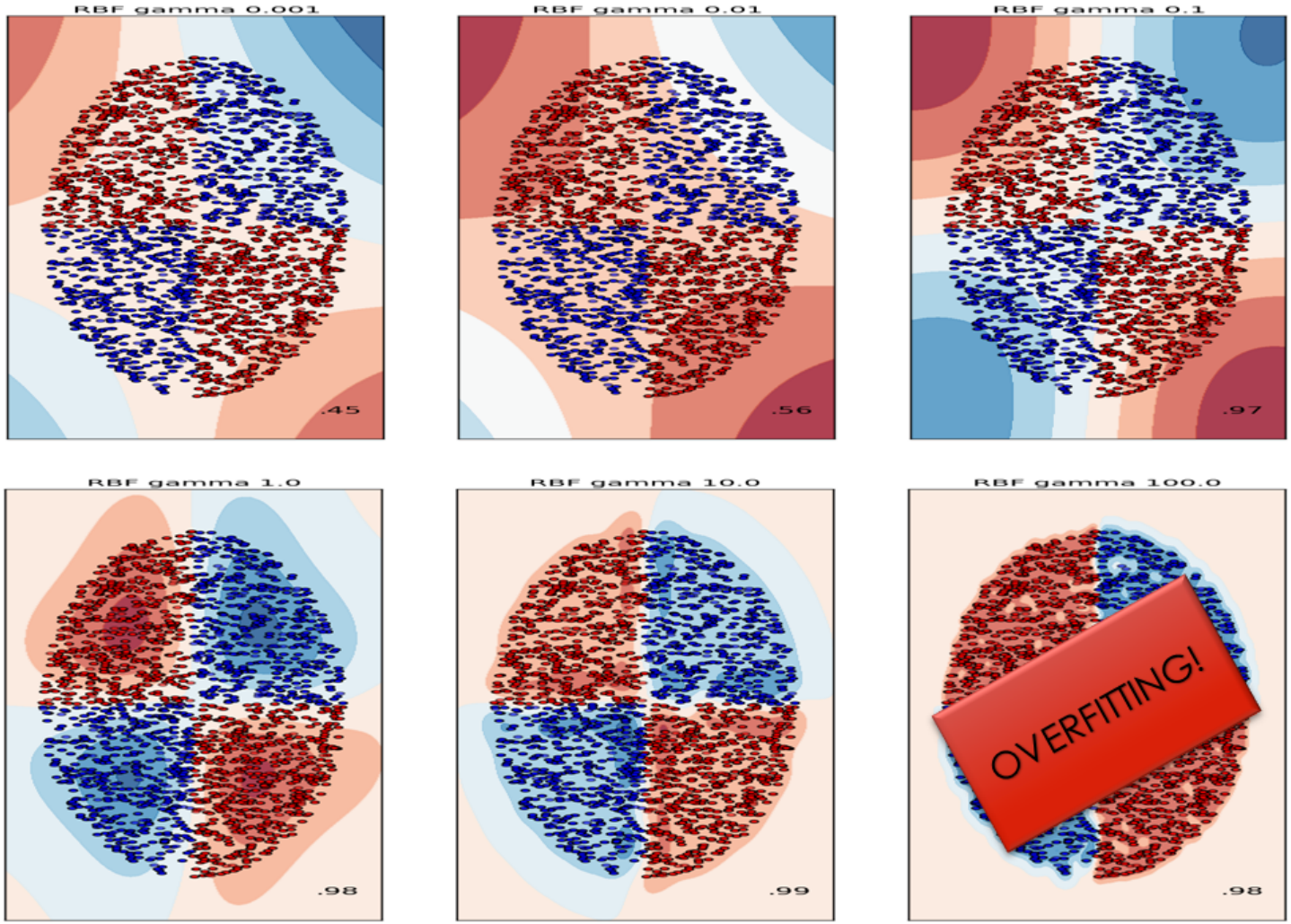
Linear vs. Poly kernel





RBF kernels

$$K(\mathbf{x}, \mathbf{z}) = \exp(-\gamma ||\mathbf{x} - \mathbf{z}||^2)$$



Closure Properties

Let K_1, K_2 be kernels defined on $\mathcal{X} \times \mathcal{X}$. $a \in \mathbb{R}^+$, $\phi : \mathcal{X} \to \mathbb{R}^N$ with K_3 a kernel over $\mathbb{R}^N \times \mathbb{R}^N$. Then,

- $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z}) + K_2(\mathbf{x}, \mathbf{z})$ is a kernel
- $K(\mathbf{x}, \mathbf{z}) = aK_1(\mathbf{x}, \mathbf{z})$ is a kernel
- $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z}) \cdot K_2(\mathbf{x}, \mathbf{z})$ is a kernel
- $K(\mathbf{x}, \mathbf{z}) = K_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$ is a kernel

A kernel can be easily normalized (such to have normalized data in feature space $||\phi(\mathbf{x})|| = 1$):

$$\tilde{K}(\mathbf{x}, \mathbf{z}) = \frac{K(\mathbf{x}, \mathbf{z})}{\sqrt{K(\mathbf{x}, \mathbf{x})K(\mathbf{z}, \mathbf{z})}}$$

Kernel extensions to other types of inputs

- Kernel for strings
  Idea: given two strings, compute the number of shared sub-strings (dynamic programming algorithms exist to make efficient the computation of these kernels)
- Kernel for trees
  Idea: given two trees, compute the number of shared sub-trees (also here dynamic programming algorithms exist to make efficient the computation of these kernels)
- Kernel for graphs
  Idea: similar to the ones above, e.g. counting common walks.

Spectrum kernels for strings

A kernel for strings can be defined by an explicit embedding map from the space of all finite sequences over an alphabet Σ, that is

$$\phi^p : (\phi^p_u(s))_{u \in \Sigma^p}$$

and Σ^p the set of strings of size p. The vector length will be $|\Sigma|^p$.
The associated kernel is defined as:

$$k_p(s,t) = \langle \phi^p(s), \phi^p(t) \rangle = \sum_{u \in \Sigma^p} \phi^p_u(s) \phi^p_u(t)$$

Example: 2-Spectrum kernels for strings

Consider the strings `bar`, `bat`, `car`, `cat`. The 2-spectra are given by:

ϕ	ar	at	ba	ca
bar	1	0	1	0
bat	0	1	1	0
car	1	0	0	1
cat	0	1	0	1

So the resulting kernel matrix is:

K	bar	bat	car	cat
bar	2	1	1	0
bat	1	2	0	1
car	1	0	2	1
cat	0	1	1	2

There exists a recursive method that, for any p, can compute this kernel in time $O(p(|s| + |t|)) = O(p \max(|s|, |t|))$.

Boolean Kernels

- Kernels for binary data
  Idea: given two binary vectors, compute the number of shared logical propositions, of a fixed form, over the input binary variables
- A generic boolean kernel is a function $k : \{0,1\}^n \times \{0,1\}^n \to \mathbb{N}$ such that $k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$ where $\phi : \{0,1\}^n \to \{0,1\}^N$ maps the binary input vectors into a space formed by logical formulas
- The linear kernel, when applied to binary data, is the simplest boolean kernel since it computes the number of shared boolean literals between the input vectors, i.e., the mapping function is the identity.

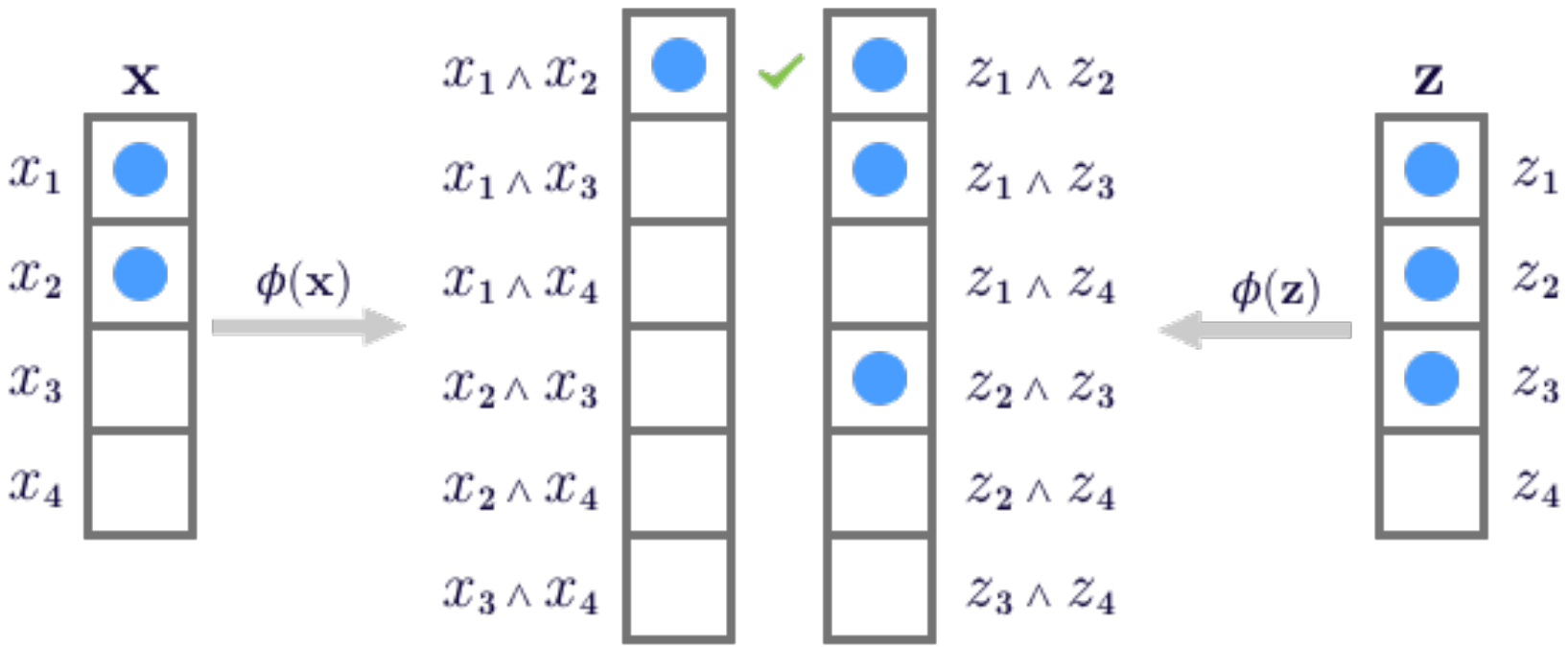
Conjunctive Kernel

- Given two binary vectors $\mathbf{x}, \mathbf{z} \in \{0,1\}^n$, the Conjunctive Kernel (C-kernel) of degree c computes the number of common conjunctions of arity c between $\mathbf{x}$ and $\mathbf{z}$
- The features in the embedding space are all the possible combinations without repetition of c objects (i.e., input variables) taken from n (i.e., the dimension of the input space)
- The value of these features are computed as the logical AND operator among the involved variables in the combination
- Formally the kernel is computed by

$$k_{\wedge}^{c}(\mathbf{x}, \mathbf{z}) = \binom{\langle \mathbf{x}, \mathbf{z} \rangle}{c}$$

Conjunctive Kernel Example

- Recall: $k^c_\wedge(\mathbf{x}, \mathbf{z}) = \binom{\langle \mathbf{x}, \mathbf{z} \rangle}{c}$



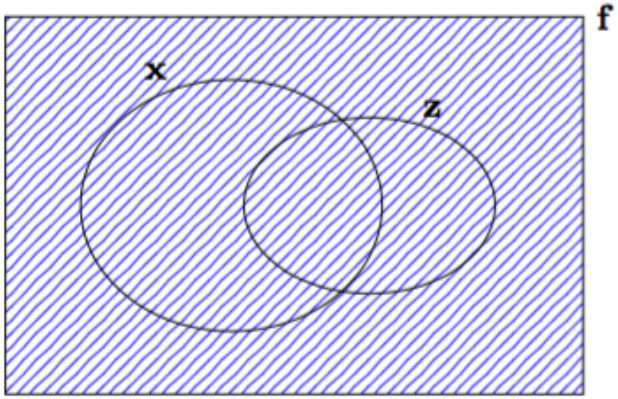
- In the example: $k^2_\wedge(\mathbf{x}, \mathbf{z}) = \binom{\langle \mathbf{x}, \mathbf{z} \rangle}{2} = \binom{2}{2} = 1.$

Disjunctive Kernel

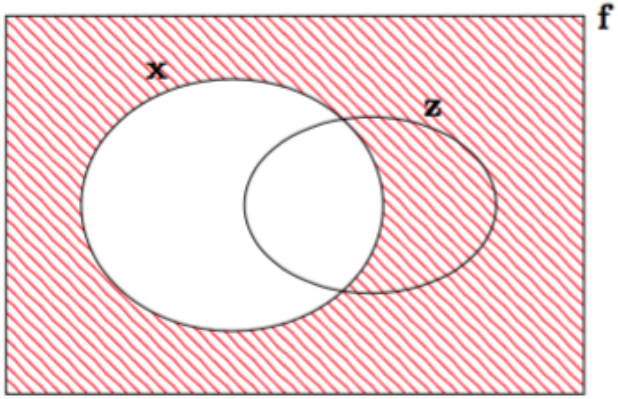
- Given two binary vectors $\mathbf{x}, \mathbf{z} \in \{0,1\}^n$, the Disjunctive Kernel (D-kernel) of degree d computes the number of common disjunctions of arity d between $\mathbf{x}$ and $\mathbf{z}$
- The features in the embedding space are all the possible combinations without repetition of d objects (i.e., input variables) taken from n (i.e., the dimension of the input space)
- The value of these features are computed as the logical OR operator among the involved variables in the combination
- Formally the kernel is computed by

$$k_{\vee}^{d}(\mathbf{x}, \mathbf{z}) = \binom{n}{d} - \binom{n - \langle \mathbf{x}, \mathbf{x} \rangle}{d} - \binom{n - \langle \mathbf{z}, \mathbf{z} \rangle}{d} + \binom{n - \langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{z}, \mathbf{z} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle}{d}$$

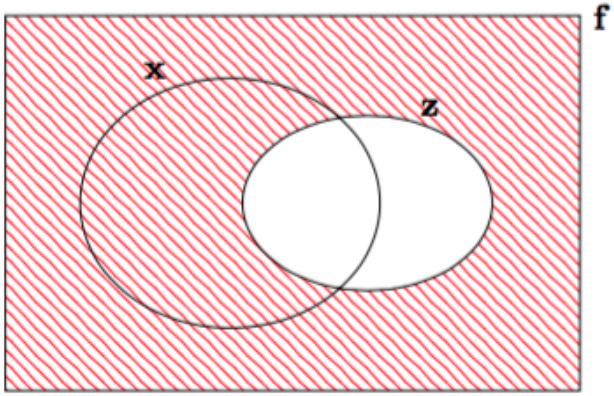
Disjunctive Kernel Computation



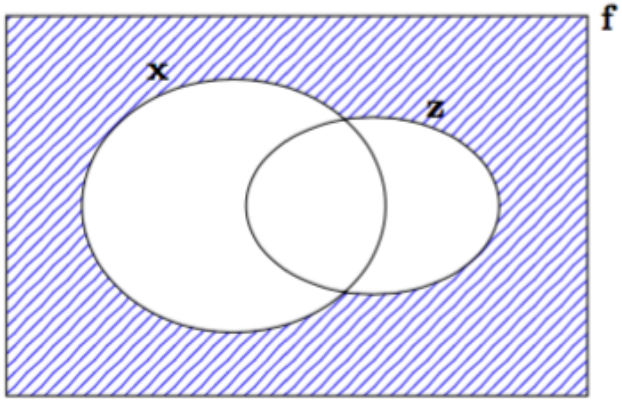
(a) $\binom{n}{d}$



(b) $-\binom{n-\langle \mathbf{x},\mathbf{x} \rangle}{d}$



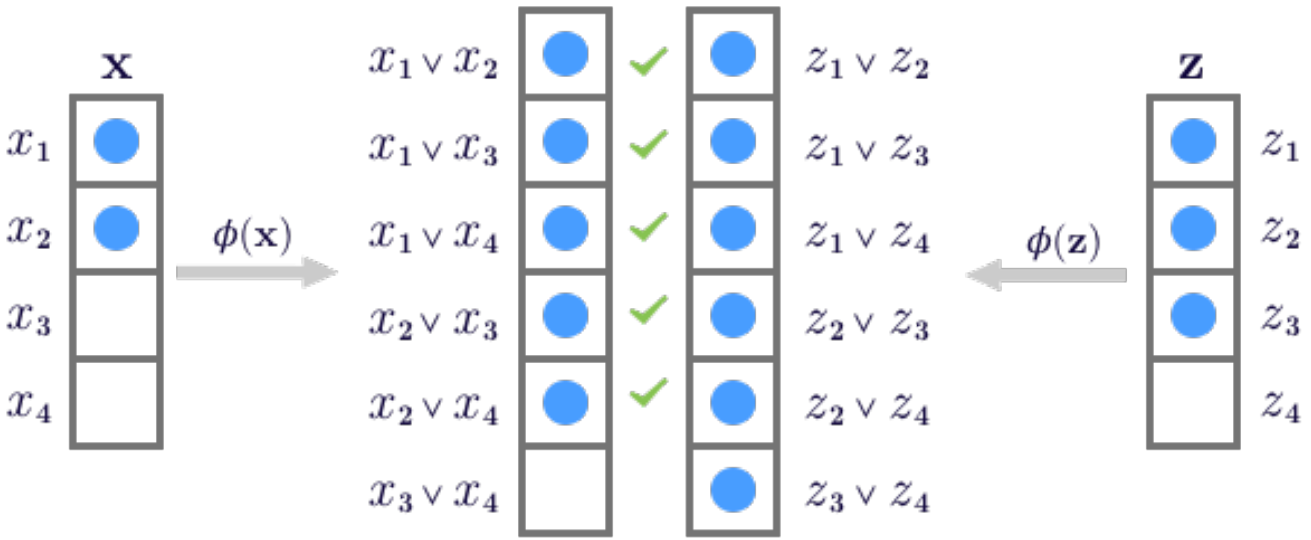
(c) $-\binom{n-\langle \mathbf{z},\mathbf{z} \rangle}{d}$



(d) $\binom{n-\langle \mathbf{x},\mathbf{x} \rangle-\langle \mathbf{z},\mathbf{z} \rangle+\langle \mathbf{x},\mathbf{z} \rangle}{d}$

Disjunctive Kernel Example

- Recall: $k_{\vee}^{d}(\mathbf{x}, \mathbf{z}) = \binom{n}{d} - \binom{n-\langle \mathbf{x},\mathbf{x} \rangle}{d} - \binom{n-\langle \mathbf{z},\mathbf{z} \rangle}{d} + \binom{n-\langle \mathbf{x},\mathbf{x} \rangle-\langle \mathbf{z},\mathbf{z} \rangle+\langle \mathbf{x},\mathbf{z} \rangle}{d}$



- In the example:
  $k_{\vee}^{2}(\mathbf{x}, \mathbf{z}) = \binom{4}{2} - \binom{4-2}{2} - \binom{4-3}{2} + \binom{4-2-3+2}{2} = 6 - 1 - 0 + 0 = 5.$

Normal Form Kernels

- Both the C-kernel and the D-kernel are defined as function of dot-products of the input vectors
- We can exploit the kernel trick to build more complex boolean kernels by composition
- DNF-kernel: the feature space is formed by Disjunctive Normal form formulas of exactly d conjunctive clauses of arity c

$$k^{d,c}_{DNF}(\mathbf{x},\mathbf{z}) = \binom{\binom{n}{c}}{d} - \binom{\binom{n}{c} - \binom{\langle \mathbf{x},\mathbf{x}\rangle}{c}}{d} - \binom{\binom{n}{c} - \binom{\langle \mathbf{z},\mathbf{z}\rangle}{c}}{d} + \binom{\binom{n}{c} - \binom{\langle \mathbf{x},\mathbf{x}\rangle}{c} - \binom{\langle \mathbf{z},\mathbf{z}\rangle}{c} + \binom{\langle \mathbf{x},\mathbf{z}\rangle}{c}}{d}$$

- CNF-kernel: the feature space is formed by Conjunctive Normal form formulas of exactly c disjunctive clauses of arity d

$$k^{d,c}_{CNF}(\mathbf{x},\mathbf{z}) = \binom{\binom{n}{d} - \binom{n-\langle \mathbf{x},\mathbf{x}\rangle}{d} - \binom{n-\langle \mathbf{z},\mathbf{z}\rangle}{d} + \binom{n-\langle \mathbf{x},\mathbf{x}\rangle-\langle \mathbf{z},\mathbf{z}\rangle+\langle \mathbf{x},\mathbf{z}\rangle}{d}}{c}$$

Kernel Learning

IDEA: Why not to learn the kernel function (or matrix)?

- Parametric methods for kernel learning
- Transductive feature extraction with non-linear kernels
- Spectral Kernel Learning
- Multiple Kernel Learning

In all these cases margin maximization criteria are typically used.

Parametric methods

Optimization of the parameters of a kernel function (e.g. RBF, Poly)

$$K(\mathbf{x}, \mathbf{z}) = \exp(-(\mathbf{x} - \mathbf{z})^\top \mathbf{M}(\mathbf{x} - \mathbf{z}))$$

Particular choices:

- $\mathbf{M} = \beta_0 \mathbf{I} = \begin{pmatrix} \beta_0 & 0 & 0 & \dots \\ 0 & \beta_0 & 0 & \dots \\ 0 & 0 & \beta_0 & \\ \vdots & \vdots & & \ddots \end{pmatrix}$ RBF kernel
- $\mathbf{M} = \text{diag}(\beta_1, \dots, \beta_m) = \begin{pmatrix} \beta_0 & 0 & 0 & \dots \\ 0 & \beta_1 & 0 & \dots \\ 0 & 0 & \beta_2 & \\ \vdots & \vdots & & \ddots \end{pmatrix}$ Anysotropic RBF

Transductive feature extraction with non-linear kerne

- Feature extraction is performed implicitly in feature space.
- Kernel Principal Component Analysis (KPCA) which is a generalization of PCA to kernels is the most popular instance of this approach. The trick is that the projections onto the eigenvectors (principal components in feature space) can be implicitly computed using the kernel trick
- Issue: This is a transductive approach since it works on kernel matrices and cannot be promptly generalized to new data!
- Solution: Out-of-sample techniques can be used to approximate the values of the kernels on new examples

Spectral Kernel Learning

The kernel matrix (positive definite) can always be written as:

$$\mathbf{K} = \sum_{s=1}^{n} \lambda_s \mathbf{u}_s \mathbf{u}_s^\top$$

where λ_s and $\mathbf{u}_s$ are the eigenvalues, and corresponding eigenvectors, of the matrix $\mathbf{K}$

- Using a transformation $\tau(\lambda)$ we can act (implicitly) on the mapping.
- The idea is to optimize the spectrum of the kernel matrix. This is also a transductive approach. Hence, it suffers of the same issues as the previous methods!

Multiple Kernel Learning

In Multiple Kernel Learning (MKL), given a set of a-priori defined kernels, these are linearly combined. Hence, the task is to learn the positive coefficients of the combination:

$$K(\mathbf{x}, \mathbf{z}) = \sum_{r=1}^{R} \eta_r K_r(\mathbf{x}, \mathbf{z}) \Rightarrow \mathbf{K} = \sum_{r=1}^{R} \eta_r \mathbf{K}_r$$

Note that, a positive combination of positive definite matrices is still a positive definite matrix. Hence, any positive combination of valid kernels is still a valid kernel!

- Fixed methods or heuristic methods: a very simple rule (or a heuristics) is used to compute the coefficients (e.g. simple kernel mean, based on the accuracy of individual kernels, etc.)
- Optimization methods: the coefficients are included as further variables to learn in the optimization problem (e.g. SVM formulation)

Recap

Notions

- Representation with kernels and kernel methods
- The kernel trick
- Kernels for vectors and extension to other types of data
- Learning the kernel

Exercises

- Implementation of some kernel for non-vectorial objects
- Definition and implementation of the Kernel Perceptron.
  Hint: note that the perceptron can always be seen in its "implicit" form as $\mathbf{w} = \sum y_i \alpha_i \mathbf{x}_i$ with $\alpha_i \in \mathbb{N}$
- Try to average multiple (kernel) perceptrons each trained processing training examples in a different order