# Kernel - Passerini

Mattia Carolo - @Carolino96

## Kernel Machines

For learning non-linear models we can apply feature mapping (you have to know *which* mapping to apply). Even if polynomial mapping is useful, in general it could be *expensive* to explicitly compute the mapping and deal with a high dimension feature space.

If we look at dual formulation of SVM problem, the feature mapping only appears in dot products. The **kernel trick** replace the dot product with an equivalent kernel function over the inputs, that produces the output of the dot product but in feature space, without mapping (explicitly) $\mathbf{x}$ and $\mathbf{x}'$ to it.

So the dual problem for svm becomes

$$\max_{\alpha \in \mathbb{R}^m} \quad \sum_{i-1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \overbrace{\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}^{k(\mathbf{x}_i, \mathbf{x}_j)} \text{ subject to } \quad 0 \leq \alpha_i \leq C, \quad i = 1, ..., m \sum_{i=1}^{m} \alpha_i y_i = 0$$

and the dual *decision function* becomes : $f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i \overbrace{\phi(\mathbf{x}_i)^t \phi(\mathbf{x})}^{k(\mathbf{x}_i, \mathbf{x})}$

This is useful beacuse if $k$ computes the dot product without computing the mapping

### Example in polynomial mapping

- **Homogeneous** polynomial kernel: the same result can be achieved by taking as kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^d$ it is the dot product in input space not feature space and the result is a scalar raised to $d$
  - $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^d, \quad k(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}) = (x_1 x_1' + x_2 x_2')^2$
- **Inhomogeneous** polynomial kernel: $k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^d$
  - $k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^d, \quad k(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}) = (1 + x_1 x_1' + x_2 x_2')$

# Kernel validity

A kernel, if valid, always corresponds to a dot product in some feature space.

A kernel is valid if it's defined as a *similarity function* defined as **cartesian product** of input space $k : X \times X \to \mathbb{R}$. It corresponds to a dot product in a certain feature space $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$

You can compute kernels even with objects that are not vectors, like sequences $\to$ the kernel generalizes the notion of dot product to arbitrary input space

## Condition for validity

The **Gram matrix** is a *symmetric matrix* of the kernels between pairs of examples: $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \quad \forall i, j$

**Positive definite** matrix: a symmetric matrix is positive definite if for any possible vector c $\mathbb{R}ightarrow \sum_{i,j=1}^{m} c_i c_j K_{ij} \geq 0, \quad \forall \mathbf{c} \in \mathbb{R}^m$

If equality only holds for $c = 0$, the matrix is **strictly positive definite**

**Alternative definitions**:

- all eigenvalues of $K$ are non-negative
- there exists a matrix $B$ such that $K = B^T B$

### Positive definite kernel

- A positive definite kernel is a function $k : X \times X \to \mathbb{R}$ giving rise to a positive definite *Gram matrix* for any $m$ and $\{\mathbf{x}_1, ..., \mathbf{x}_m\}$
- positive definiteness is **necessary** and **sufficient** condition for a kernel to correspond to a dot product of some feature map $\phi$

To verify if the kernel is valid, either we have to:

- show how is the $\phi$
- make the *feature map explicit*
- using *kernel combination properties* (using already valid kernels combined)

In the dual problem for SVM regression, $\phi(\mathbf{x})$ appears only in the dot product (and also in the decision function $f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + w_0 = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) \overbrace{\phi(\mathbf{x}_i)^T \phi(\mathbf{x})}^{k(\mathbf{x}_i, \mathbf{x})} + w_0$):

$$\max_{\alpha \in \mathbb{R}^m} \quad -\frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) \overbrace{\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}^{k(\mathbf{x}_i, \mathbf{x}_j)} - \epsilon \sum_{i=1}^{m} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} y_i (\alpha_i^* - \alpha_i) \text{ subject to } \quad \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) = 0 \alpha_i, \alpha$$

## Kernelize the Perceptron

- *stochastic perceptron* (already seen) $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$:
  - initialize $\mathbf{w} = 0$
  - iterate until all examples are classified correctly $\rightarrow$ for each incorrectly classified training example $(\mathbf{x}_i, y_i) : \mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$
- **kernel perceptron** $f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x})$:
  - initialize $\alpha_i = 0 \quad \forall i$
  - iterate until all examples are classified correctly $\rightarrow$ for each incorrectly classified training example $(\mathbf{x}_i, y_i) : \alpha_i \leftarrow \alpha_i + \eta y_i$

$\mathbb{R}ightarrow$ in kernel machines you always have that the decision function is the *sum over the training sample of coefficient times* $k(\mathbf{x}_i, \mathbf{x})$. The coefficient changes depending on the problem.

## Basic kernels

- **linear kernel** $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$: it computes only the dot product: it's useful when combining kernel because when you don't have a kernel, in reality you have the linear kernel;
- **polynomial kernel** $k_{d,c}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + c)^d$: homogeneous if $c = 0$, inhomogeneous otherwise
- **gaussian kernel** $k_\sigma(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}) = \exp(-\frac{\mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{x}' + \mathbf{x}'^T\mathbf{x}'}{2\sigma^2})$
  - depends on $\sigma$, the **width** parameter
  - the smaller the width, the more predictions on a point only depends on its nearest neighbours
  - it's a **universal kernel**: it can uniformly approximate any arbitrary continuous target function (but you first have to find the correct value of $\sigma$)

  Gaussian has infinite dimensional feature space

The **choice** of a Kernel is made **before** training.

## Kernel on structured data

We don't need to think input as vectors anymore: now the kernel can be seen as a way to *generalize dot products to arbitrary domains*. It's possible to design kernels over structured objects such as sequences, graphs and trees.

The idea is to create a pairwise function measuring the **similarity between two objects**. This measure has to satisfy the *valid kernel conditions*.

### Examples of Kernel over structure

Every kernel over structure is built as combination of kernels over pieces.

The simplest **kernel over pieces** (of a structure) is the **delta kernel** (or *match* kernel) $k_\delta(x, x') = \delta(x, x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases}$ where x does not need to be a vector

**Kernel on sequeces** **Spectrum kernel**: (looking at frequencies of subsequences of $n$ symbols) and the result is the number of time each subsequence appears in the starting sequence. The feature space is the space of all possible $k$-subsequences.

### Kernel Combinations

Building the kernel combining pieces, provided the basic kernels are valid and the operations are valid, the result is valid.

- **sum**: concatenating the features of the two kernels (that are valid); can be defined on different spaces ($k_1$ and $k_2$ could look at different things):

  $$(k_1+k_2)(x, x') = k_1(x, x')+k_2(x, x') = \phi_1(x)^T\phi_1(x')+\phi_2(x)^T\phi_2(x') = (\phi_1(x)\phi_2(x)) \begin{pmatrix} \phi_1(x') \\ \phi_2(x') \end{pmatrix}$$

- **product**: the resulting feature space is the cartesian product between the feature space of the first kernel and the feature space of the second kernel

  $$(k_\times k_1)(x, x') = k_1(x, x')k_2(x, x') = \sum_{i=1}^{n} \phi_{1i}(x)\phi_{1i}(x') \sum_{j=1}^{m} \phi_{2j}(x)\phi_{2j}(x') = \sum_{i=1}^{n}\sum_{j=1}^{m}(\phi_{1i}(x)\phi_{2j}(x))(\phi_{1i}(x')\phi_{2j}$$

  - where $\phi_{12}(x) = \phi_1 \times \phi_2$ is the Cartesian product
  - the product can be between kernels in different spaces (**tensor** product)

- **linear combination**: you can always multiply by a positive scalar (if it's negative the kernel becomes invalid). If I don't know which kernel to use I just take a bunch of them, combine them linearly and learn the parameters (in addition to learn the $\alpha$s of the dual) $\rightarrow$ this is called **kernel learning**
  - a kernel can be rescaled by an arbitrary positive constant $k_\beta(x, x') = \beta k(x, x')$
  - we can define, for example, linear combinations of kernels (each scaled by a desired weight): $k_{\text{sum}}(x, x') = \sum_{k=1}^{k} \beta_k k_k(x, x')$

- **normalization**: kernel values can often be influenced by the dimension of objects (a longer string contains more substrings → higher kernel value). This effect can be reduced with *normalizing* the kernel. **Cosine normalization** computes the cosine of the dot product in feature space: $\hat{k}(x, x') = \frac{k(x, x')}{\sqrt{k(x, x)k(x', x')}}$
- **composition**: you can combine also by composition

  $(k_{d,c} \circ k)(x, x') = (k(x, x')+c)^d (k_\sigma \circ k)(x, x') = \exp(-\frac{k(x, x) - 2k(x, x') + k(x', x')}{2\sigma^2})$

  it corresponds to the composition of the mappings associated with the two kernels.

### Kernel on Graphs

**Wistfeiler-Lehman graph kernel** It's an efficient **graph kernel** for large graphs. It relies on (approximation of) Weistfeiler-Lehman test of graph isomorphism and it describes a family of graph kernel:

- Let $\{G_0, G_1, ..., G_h\} = \{(V, E, I_0), (V, E, I_1), ..., (V, E, I_h)\}$ be a sequence of graphs made from $G$ where $I_i$ is the node labelling the $i$th WL iteration
- Let $k : G \times G' \to \mathbb{R}$ be any kernel on graphs
- the Weistfeiler-Lehman graph kernel is defined as: $k^h_{WL}(G, G') = \sum_{i=0}^{h} k(G_i, G'_i)$

  Graphs G and H are isomorphic if there is a structure that preserves a one-to-one correspondence between the vertices and edges.