

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 7: Kernel methods

---

Juho Rousu

25. October, 2022

Department of Computer Science  
Aalto University

# Kernel methods

Key characteristics of kernel methods:

- **Embedding:** Inputs  $\mathbf{x} \in X$  from some input space  $X$  are embedded into a *feature space*  $F$  via a feature map  $\phi : X \mapsto F$ .  $\phi$  may be highly **non-linear** and  $F$  potentially very high-dimensional vector space
- **Linear models:** are built for the the patterns in the feature space (typically  $\mathbf{w}^T \phi(\mathbf{x})$ ); efficient to find the optimal model, convex optimization
- **Kernel trick:** Algorithms work with kernels, inner products of feature vectors  $\kappa(\mathbf{x}, \mathbf{z}) = \sum_j \phi_j(\mathbf{x}) \phi_j(\mathbf{z})$  rather than the explicit features  $\phi(\mathbf{x})$ ; side-step the efficiency problems of high-dimensionality
- **Regularized learning:** To avoid overfitting, large feature weights are penalized, separation by large margin is favoured

# Data analysis tasks via kernels

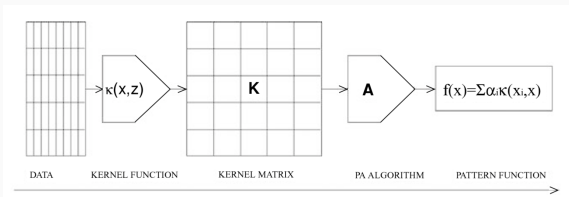
Many data analysis algorithms can be 'kernelized', i.e. transformed to an **equivalent** form by replacing object descriptions (feature vectors) by pairwise similarities (kernels):

- Classification (SVM)
- Regression
- Ranking
- Novelty detection
- Clustering
- Principal component analysis, canonical correlation analysis
- Multi-label/Multi-task/Structured output
- ...

More of the tasks beyond classification will be discussed at the course Kernel Methods in Machine Learning (Spring 2023 by Rohit Babbar)

# Modularity of kernel methods

- Algorithms are designed that work with arbitrary inner products (or kernels) between inputs
- The same algorithm will work with any inner product (or kernel)
- This allows theoretical properties of the learning algorithm to be investigated and the results will carry to all application domains
- Kernel will depend on the application domain; prior information is encoded into the kernel



# What is a kernel?

- Informally, a **kernel** is a function that calculates the similarity between two objects, e.g.
  - two proteins
  - two images
  - two documents
  - ...
- $x_i \in X$  and  $x_j \in X$ 
  - $X$  = set of all proteins in the nature (finite set)
  - $X$  = all possible images (infinite set)
  - $X$  = all possible documents (infinite set)
- $\kappa: X \times X \rightarrow \mathbb{R}$

# Data and Feature maps

- We assume inputs  $\mathbf{x}$  to come from an arbitrary set  $X$ :
  - Vectors, matrices, tensors
  - Structured objects: Sequences, hierarchies, graphs
- We further assume the data items can be expressed as objects in some feature space  $F$
- Typically  $F$  is a space of feature vectors,  $F \subseteq \mathbb{R}^N$ , where  $N$  is the dimension of the feature space, or more generally matrices or tensors.
- Inputs  $\mathbf{x}$  are mapped to this space by a feature map  $\phi : X \mapsto F$
- $\phi(\mathbf{x})$  is the image of the data item in the feature space

# What is a kernel?

- Formally: a kernel function is an inner product (scalar product, dot product) in a feature space  $F$ , denoted by  $\langle \cdot, \cdot \rangle_F$ 
  - Often the subscript  $F$  is dropped when it is clear from context
- Linear kernel: If  $\mathbf{x} \in \mathbb{R}^n$  and the feature map  $\phi(\mathbf{x}) = \mathbf{x}$  is the identity, then  $F = \mathbb{R}^n$  and the resulting kernel

$$\kappa_{lin}(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle_F = \langle \mathbf{x}, \mathbf{z} \rangle_{\mathbb{R}^n}$$

is called the linear kernel

- Linear kernel therefore corresponds to the dot product in  $\mathbb{R}^n$

$$\kappa_{lin}(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^n x_j z_j = \mathbf{x}^T \mathbf{z}$$

# Geometric interpretation

- Geometric interpretation of the linear kernel: cosine angle between two feature vectors

$$\cos \beta = \frac{\mathbf{x}^T \mathbf{z}}{\|\mathbf{x}\|_2 \|\mathbf{z}\|_2} = \frac{\kappa_{lin}(\mathbf{x}, \mathbf{z})}{\sqrt{\kappa_{lin}(\mathbf{x}, \mathbf{x})} \sqrt{\kappa_{lin}(\mathbf{z}, \mathbf{z})}},$$

where

$$\|\mathbf{x}\|_2 = \sqrt{\kappa_{lin}(\mathbf{x}, \mathbf{x})} = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\sum_{j=1}^n x_j^2}$$

is the Euclidean norm.



## Kernel vs. Euclidean distance

- Assume two vectors  $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$  with unit length  $\|\mathbf{x}\|_2 = \|\mathbf{z}\|_2 = 1$
- Kernel:  $\kappa(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$
- Euclidean Distance:  $d(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|_2 = \sqrt{\sum_{k=1}^n (x_k - z_k)^2}$
- Expanding the squares and using unit length of the vectors we get:

$$\begin{aligned}\frac{1}{2}d(\mathbf{x}, \mathbf{z})^2 &= \frac{1}{2}\|\mathbf{x} - \mathbf{z}\|_2^2 = \frac{1}{2}(\mathbf{x} - \mathbf{z})^T (\mathbf{x} - \mathbf{z}) = \\ &= \frac{1}{2} (\|\mathbf{x}\|_2^2 - 2\mathbf{x}^T \mathbf{z} + \|\mathbf{z}\|_2^2) \\ &= 1 - \mathbf{x}^T \mathbf{z} = 1 - \kappa(\mathbf{x}, \mathbf{z})\end{aligned}$$

# Hilbert space\*

Formally the underlying space of a kernel is required to be a Hilbert space

A Hilbert space is a real vector space  $\mathcal{H}$ , with the following additional properties

- Equipped with a **inner product**, a map  $\langle \cdot, \cdot \rangle$ , which satisfies for all objects  $x, x', z \in \mathcal{H}$ 
  - linear:  $\langle ax + bx', z \rangle = a\langle x, z \rangle + b\langle x', z \rangle$
  - symmetric:  $\langle x, x' \rangle = \langle x', x \rangle$
  - positive semi-definite:  $\langle x, x \rangle \geq 0$ ,  $\langle x, x \rangle = 0$  if and only if  $x = 0$
- Complete: every Cauchy sequence  $\{h_n\}_{n \geq 1}$  of elements in  $\mathcal{H}$  converges to an element of  $\mathcal{H}$
- Separable: there is a countable set of elements  $\{h_1, h_2, \dots\}$  in  $\mathcal{H}$  such that for any  $h \in \mathcal{H}$  and every  $\epsilon > 0$   $\|h_i - h\| < \epsilon$ .

On this lecture  $\mathcal{H} = \mathbb{R}^N$ , where the dimension  $N$  is finite or infinite. Both cases are Hilbert spaces.

# The kernel matrix

- In kernel methods, a **kernel matrix**, also called the **Gram matrix**, an  $m \times m$  matrix of pairwise similarity values is used:

$$\mathbf{K} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \kappa(\mathbf{x}_1, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_m) \\ \kappa(\mathbf{x}_2, \mathbf{x}_1) & \kappa(\mathbf{x}_2, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_2, \mathbf{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_m, \mathbf{x}_1) & \kappa(\mathbf{x}_m, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

- Each entry is an inner product between two data points  
 $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ , where  $\phi : X \mapsto \mathcal{F}$  is a feature map
- Since an inner product is symmetric,  $\mathbf{K}$  is a symmetric matrix

# Kernel matrix example

The heatmap on the right illustrates a typical kernel matrix

- Rows and columns index examples in the data
- The colors corresponds to the kernel values (red=high value, blue=low value)
- Notice the dark red diagonal elements: they correspond to the kernel value between an example and itself ( $\kappa(\mathbf{x}_i, \mathbf{x}_i)$ )
- Red/orange diagonal blocks correspond to clusters of similar examples as defined by the kernel values  $\kappa(\mathbf{x}_i, \mathbf{x}_j)$

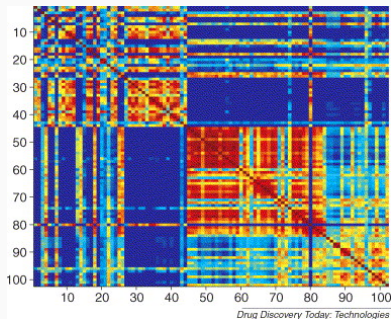


Image source; Mark Girolami et al. 2006. Analysis of complex, multidimensional datasets, *Drug Discovery Today: Technologies*, 3, 1, pp. 13–19

# Processing the kernel matrix

$$\mathbf{K} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \kappa(\mathbf{x}_1, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_m) \\ \kappa(\mathbf{x}_2, \mathbf{x}_1) & \kappa(\mathbf{x}_2, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_2, \mathbf{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_m, \mathbf{x}_1) & \kappa(\mathbf{x}_m, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

- Processing the kernel matrix during preprocessing, training and prediction time is the major factor of the time-complexity of kernel methods
- Compare the two matrices assuming  $m$  examples of dimension  $\mathbb{R}^N$ 
  - The kernel matrix has  $m^2$  items, independently from  $N$
  - The data matrix, matrix of feature vectors of the training data has size  $mN$
- Consequently, the kernel matrix scales better than the data matrix when  $N > m$

# The kernel matrix

- A symmetric matrix  $\mathbf{A} \in \mathbb{R}^{m \times m}$  is positive semi-definite (PSD) if for any vector  $\mathbf{v} \in \mathbb{R}^m$ , we have  $\mathbf{v}^T \mathbf{A} \mathbf{v} \geq 0$
- A symmetric PSD matrix has non-negative eigenvalues  
 $\lambda_1 \geq \dots \geq \lambda_m \geq 0$
- The **kernel matrix** corresponding to the kernel function  $\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$  on a set of data points  $\{\mathbf{x}_i\}_{i=1}^m$  is positive semidefinite:

$$\begin{aligned} \mathbf{v}^T \mathbf{K} \mathbf{v} &= \sum_{i,j=1}^n v_i \mathbf{K}_{ij} v_j = \sum_{i,j=1}^m v_i \langle \phi(x_i), \phi(x_j) \rangle v_j = \\ &= \left\langle \sum_{i=1}^m v_i \phi(x_i), \sum_{j=1}^m v_j \phi(x_j) \right\rangle = \left\| \sum_{i=1}^m v_i \phi(x_i) \right\|^2 \geq 0 \end{aligned}$$

# PSD property and optimization

- Consider objective of the dual SVM optimization problem

$$OBJ(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \alpha^T \mathbf{H} \alpha$$

where we denoted  $\mathbf{H} = (y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}^m$

- It is easy to verify that  $\mathbf{H}$  is the Hessian matrix of second derivatives of the objective;

$$\mathbf{H} = \left( \frac{\partial^2 OBJ(\alpha)}{\partial \alpha_i \partial \alpha_j} \right)_{i,j=1}^m$$

- If  $\mathbf{H}$  is PSD  $OBJ(\alpha)$  is concave ( $-OBJ(\alpha)$  is convex), and has no non-optimal local maxima
- However,  $\mathbf{H}$  is PSD if and only if  $\mathbf{K}$  is PSD
- Thus, a PSD kernel matrix  $\mathbf{K}$  ensures that we can find a global optimum by gradient descent approaches

# Rademacher complexity

- Assume a symmetric positive definite kernel  $\kappa : X \times X \mapsto \mathbb{R}$  with associated feature map  $\phi$ , and a sample  $S$  of size  $m$ , with the kernel matrix  $\mathbf{K} = (\kappa(x_i, x_j))_{i,j=1}^m$ , and  $\kappa(\mathbf{x}_i, \mathbf{x}_i) \leq r^2$  for all  $i = 1, \dots, m$
- Empirical Rademacher complexity of the hypothesis class containing support vector machines

$$\mathcal{H} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \phi(\mathbf{x}) \rangle : \|\mathbf{w}\| \leq B\}$$

for some  $B \geq 0$  satisfies (c.f. Mohri book for the proof)

$$\hat{\mathcal{R}}_S(\mathcal{H}) \leq \frac{B\sqrt{\text{trace}(\mathbf{K})}}{m}$$

- The key quantities are
  - the upper bound  $B$  of the norm of weight vector – relates to the margin
  - the trace of the kernel matrix  
 $\text{trace}(\mathbf{K}) = \sum_{i=1}^m \kappa(\mathbf{x}_i, \mathbf{x}_i) = \sum_{i=1}^m \|\phi(\mathbf{x}_i)\|^2 \leq mr^2$  – relates to the norm of the data points



# Generalization error bound

- We can plug the above to a Rademacher complexity based generalization bound (c.f. Lecture 3)

$$\begin{aligned} R(h) &\leq \hat{R}(h) + \hat{\mathcal{R}}(\mathcal{H}) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}} \\ &\leq \hat{R}(h) + \frac{B\sqrt{\text{trace}(\mathbf{K})}}{m} + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}} \end{aligned}$$

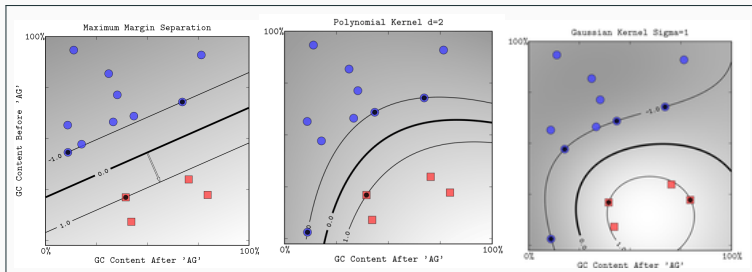
- Evaluating this bound required observing (1) the empirical risk of the hypothesis  $\hat{R}(h)$  on training data, (2) the norm of the weight vector  $B = \|\mathbf{w}\| = \sqrt{\sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)}$ , and (3) the trace of the kernel matrix
- Note that we do not need to run simulations with random labelings to use this bound!

## Non-linear kernels

---

# Non-linear kernels

- By defining kernels that are non-linear functions of the original feature vectors, a linear models (e.g. SVM classifier) can be turned into a non-linear model
- However, the learning algorithm does not need to be changed, apart from plugging in the new kernel matrix
- The most commonly used non-linear kernels:
  - Polynomial kernel:  $\kappa_{pol}(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + c)^q$
  - Gaussian (or radial basis function, RBF) kernel:  
 $\kappa_{RBF}(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2 / (2\sigma^2))$



# Non-linear kernels: Polynomial kernel

- Given inputs  $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$ , the **polynomial kernel** is given by

$$\kappa_{pol}(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + c)^q$$

- Integer  $q > 0$  gives the **degree** of the polynomial kernel
- Real value  $c \geq 0$  is a weighting factor for lower order polynomial terms
- The underlying features are **non-linear**: monomial combinations  $x_1 \cdot x_2 \cdots x_k$  of degree  $k \leq q$  of the original features  $x_j$

## Example: Polynomial kernel on 2D inputs

- Consider two-dimensional inputs  $\mathbf{x} = [x_1, x_2]^T \in \mathbb{R}^2$
- The second degree polynomial kernel is given by  
 $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^2$
- We can write it as a inner product in  $\mathbb{R}^6$ :

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{x}') &= (\mathbf{x}^T \mathbf{x}' + c)^2 = (x_1 x'_1 + x_2 x'_2 + c)^2 = \\&= x_1 x'_1 x_1 x'_1 + x_2 x'_2 x_2 x'_2 + c^2 + \\&+ 2x_1 x'_1 x_2 x'_2 + 2c x_1 x'_1 + 2c x_2 x'_2 \\&= \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 x_2 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ c \end{bmatrix}^T \begin{bmatrix} x_1'^2 \\ x_2'^2 \\ \sqrt{2}x'_1 x'_2 \\ \sqrt{2c}x'_1 \\ \sqrt{2c}x'_2 \\ c \end{bmatrix} \\&= \phi(\mathbf{x})^T \phi(\mathbf{x}'),\end{aligned}$$

where  $\phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1 x_2, \sqrt{2c}x_1, \sqrt{2c}x_2, c]^T$

## Non-linear kernels: Polynomial kernel

$$\kappa_{pol}(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + c)^q$$

A linear model in the polynomial feature space corresponds to a **non-linear model** in the original feature space

- In the previous example, the model

$$\mathbf{w}^T \phi(\mathbf{x}) = w_1 x_1^2 + w_2 x_2^2 + w_3 \sqrt{2} x_1 x_2 + w_4 \sqrt{2c} x_1 + w_5 \sqrt{2c} x_2 + w_6 c = 0$$

is a second degree polynomial in the original inputs space, but a hyperplane in the new 6-dimensional feature space

- Using the dual representation  $\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \phi(\mathbf{x}_i)$ , the polynomial kernel allows non-linear classification in the input space by

$$\mathbf{w}^T \phi(\mathbf{x}) = \sum_i \alpha_i y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle = \sum_i \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x})$$

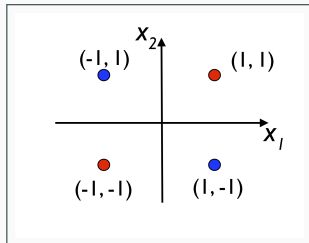
## Example: XOR with polynomial kernel

Consider the following simple example:

- Input data points  
 $\{(-1, -1), (-1, 1), (1, -1), (1, 1)\}$   
and the label (red = 1, blue = -1)  
given by a XOR type function

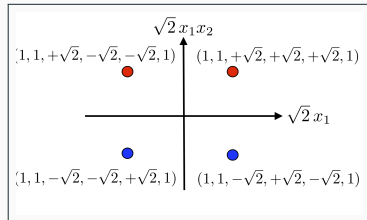
$$y = x_1 x_2 = \begin{cases} +1 & \text{if } x_1 = x_2 \\ -1 & \text{if } x_1 \neq x_2 \end{cases}$$

- The classes are not linearly separable:  
there is no consistent line that  
separates the two classes



## Example: XOR with polynomial kernel

- However, map the data using the feature map  $\phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}cx_1, \sqrt{2}cx_2, c]^T$  underlying the polynomial kernel function



- Now, the example data is linearly separable in the feature space, for example, choose  $\alpha_i = 1/(4\sqrt{2})$ , for all  $i$ :

$$\begin{aligned}\mathbf{w} &= \sum_i \alpha_i y_i \phi(\mathbf{x}_i) = \\ &= (\phi([-1, -1]^T) - \phi([-1, 1]^T) - \phi([1, -1]^T) + \phi([1, 1]^T)) / (4\sqrt{2}) \\ &= [0, 0, 1, 0, 0, 0]^T\end{aligned}$$

- We can consistently classify the example data by using the kernel function  $\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ :  $h(\mathbf{x}) = \text{sgn}(\sum_{i=1}^m \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}))$



## Non-linear kernels: Polynomial kernel

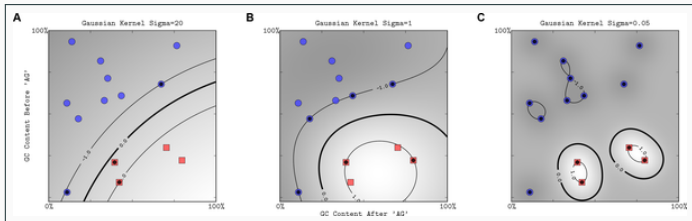
- The dimension of the polynomial feature space is  $\binom{d+q}{q} = O((d+q)^q)$  where  $d$  is the dimension of the input space  $X$  and  $q$  is the degree of the polynomial.
- Explicitly maintaining the feature map  $\phi(\mathbf{x})$  and the weight vector  $\mathbf{w}$ , and evaluating the model  $\mathbf{w}^T \phi(\mathbf{x})$  takes  $O(d^q)$  time and space
- However, the polynomial kernel  $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^q$  can be computed in time  $O(d)$  in preprocessing, and evaluated in constant time
- Evaluating the model using the dual representation  $\sum_{i=1}^m \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x})$  takes  $O(m)$  time
- Trade-off: No computational overhead from working in the high-dimensional feature space, but linear dependency in the size of training data

# Non-linear kernels: Gaussian kernel (Radial basis function kernel, RBF)

Gaussian kernel between two inputs  $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$  with bandwidth parameter  $\sigma > 0$ :

$$\kappa_{RBF}(\mathbf{x}, \mathbf{z}) = \exp \left( -\|\mathbf{x} - \mathbf{z}\|^2 / (2\sigma^2) \right)$$

- Large values for  $\sigma$  give a smoother kernel, slower decay of kernel values as a function of the squared euclidean distance  $\|\mathbf{x} - \mathbf{z}\|^2$ , and a more linear decision boundary
- Small values for  $\sigma$  give a less smooth kernel, faster decay of kernel values as a function of the squared euclidean distance  $\|\mathbf{x} - \mathbf{z}\|^2$ , and a more non-linear decision boundary

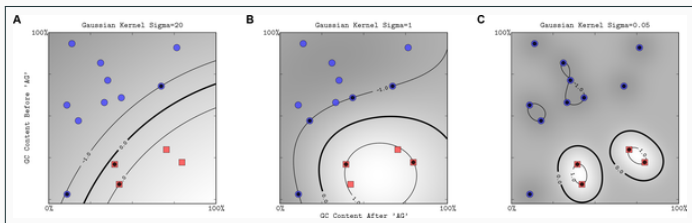


# Non-linear kernels: Gaussian kernel (Radial basis function kernel, RBF)

Gaussian kernel can be seen to correspond to an **infinite dimensional** polynomial kernel:

$$\kappa_{RBF}(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2 / (2\sigma^2)) = \sum_{n=0}^{\infty} \frac{(\mathbf{x}^T \mathbf{z})^n}{\sigma^{2n} n!}$$

- It can be shown through the power series expansion of  $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$
- Each individual term is an degree- $n$  polynomial kernel  $(\mathbf{x}^T \mathbf{z})^n$ , exponentially down-weighted by  $\frac{1}{\sigma^{2n} n!}$



# Designing kernels

---

# Several ways to get to a kernel

Approach I. Construct a feature map  $\phi$  and think about efficient ways to compute the inner product  $\langle \phi(x), \phi(x) \rangle$

- If  $\phi(x)$  is very high-dimensional, computing the inner product element by element is slow, we don't want to do that
- For several cases, there are efficient algorithms to compute the kernel in low polynomial time, even with exponential or infinite dimension of  $\phi$

## Several ways to get to a kernel

Approach II. Construct similarity measure and show that it qualifies as a kernel:

- Show that for any set of examples the matrix  $K = (\kappa(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}^m$  is positive semi-definite (PSD).
- In that case, there always is an underlying feature representation, for which the kernel represents the inner product
- Example: if you can show the matrix is a covariance matrix for some variates, you will know the matrix will be PSD.

# Several ways to get to a kernel

Approach III. Convert a distance or a similarity into a kernel

- Take any distance  $d(\mathbf{x}, \mathbf{z})$  or a similarity measure  $s(\mathbf{x}, \mathbf{z})$  (that do not need to be a kernel)
- In addition a set of data points  $Z = \{\mathbf{z}_j\}_{j=1}^m$  from the same domain is required (e.g. training data)
- Construct feature vector from distances (similarly for  $s$ ):  
$$\phi(\mathbf{x}) = (d(\mathbf{x}, \mathbf{z}_1), d(\mathbf{x}, \mathbf{z}_2), \dots, d(\mathbf{x}, \mathbf{z}_m))$$
- Compute linear kernel, also known as the **empirical kernel map**:  
$$\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$
- This will always work technically, but requires that the data  $Z$  captures the essential patterns in the input space  $\implies$  need enough data

# Several ways to get to a kernel

## Approach IV. Making kernels from kernels

- Examples of elementary operations that give valid kernels when applied to kernels  $\kappa_n$ ,  $n = 1, 2, \dots$ 
  1. Sum:  $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$
  2. Scaling with a positive scalar:  $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z})$ ,  $a > 0$
  3. Itemwise product:  $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$
  4. Normalization:  $\kappa(\mathbf{x}, \mathbf{z}) = \frac{\kappa_1(\mathbf{x}, \mathbf{z})}{\sqrt{\kappa_1(\mathbf{x}, \mathbf{x})\kappa_1(\mathbf{z}, \mathbf{z})}} = \left\langle \frac{\phi(\mathbf{x})}{\|\phi(\mathbf{x})\|}, \frac{\phi(\mathbf{z})}{\|\phi(\mathbf{z})\|} \right\rangle$
  5. Pointwise limit:  $\kappa(\mathbf{x}, \mathbf{z}) = \lim_{n \rightarrow \infty} \kappa_n(\mathbf{x}, \mathbf{z})$
  6. Composition with a power series of radius of convergence  $\rho$ :  
$$\kappa(\mathbf{x}, \mathbf{z}) = \sum_{n=0}^{\infty} a_n \kappa(\mathbf{x}, \mathbf{z})^n, \text{ with } a_n \geq 0 \text{ for all } n, \text{ and } |\kappa(\mathbf{x}, \mathbf{z})| < \rho$$
- The operations can be combined to construct arbitrarily complex kernels, e.g. polynomial kernels and Gaussian kernels can be derived this way (see details in the Mohri book ch. 6)



## Kernels for structured data

---

# Kernels for structured data

- In many applications the data does not come in the form of numerical vectors or data matrices, but from a general set of objects  $X$ , for example:
  - Sequential data - text analysis kernels, string kernels
  - Molecular data - graph kernels
  - Structured documents, context-free grammars, classification taxonomies, ... - tree kernels
- To compute kernels we could define a feature map  $\phi : X \mapsto F$  and compute feature vectors  $\phi(x)$  for our data, and the kernel as the inner product:  $\kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_F$
- Alternatively, we can use an algorithm that directly computes the kernel values  $\kappa(x_i, x_j)$  for any pair of objects  $x_i, x_j$
- The latter approach can be very effective if the feature space  $F$  has a high dimension

# Kernels for structured data

- The commonly seen form for kernels for two structured objects  $x_i$  and  $x_j$

$$\kappa(x_i, x_j) = \sum_{s \in S} \phi_s(x_i) \phi_s(x_j),$$

where  $S$  is the set of substructures of interest and  $\phi_s(x)$  is either

- An indicator function:  $\phi_s(x) = 1$  if and only if  $x$  contains  $s$
- A count:  $x$  contains  $\phi_s(x)$  instances of  $s$
- In many cases, the set  $S$  has exponential size in the size of the objects and the feature vectors  $\phi(x)$  are high dimensional and sparse
- Efficient algorithms exist for computing the kernels  $\kappa(x_i, x_j)$  directly from the structured data, skipping writing down the feature vectors  $\phi(x)$

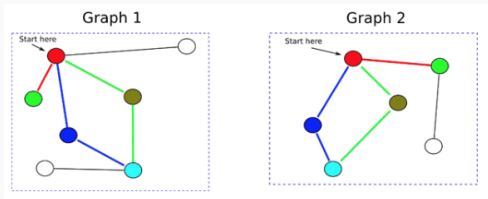
## Example: String kernels

- String kernels is a family of kernels between sequences based on "counting" common subsequences two sequences have
- Underlying feature map, also called subsequence spectrum, contains a feature for each possible substring
- The feature spaces induced by subsequences are generally exponential in the length of the subsequences
- However, low polynomial time algorithms (linear to quadratic time) exist to compute string kernels
- For more information see Lodhi et al. "Text classification using string kernels." Journal of Machine Learning Research 2.Feb (2002): 419-444.

$x$    AAACAAATAAGTAACTAATCTTTAGGAAGAACGTTTCAACCATTTTGAG  
 $x'$    TACCTAATTATGAAATTAAATTCAGTGTGCTGATGGAACGGAGAAGTC

## Example: Graph kernels

- Basic idea: count common substructures in two graphs
- Challenging problem in general due to the underlying NP-hard subgraph isomorphism problem: given two graphs  $G$  and  $S$ , does a graph  $S$  appear in  $G$  as a subgraph
- Polynomial-time kernel computation possible for restricted substructures:
  - Random walks
  - Tree-shaped subgraphs
  - Small general subgraphs
- For more information see Vishwanathan et al. "Graph kernels." Journal of Machine Learning Research 11 (2010): 1201-1242.



# Summary

- Kernel methods are a broad class of data analysis methods
- Kernels allow efficient non-linear learning in high-dimensional feature spaces
- Special kernels can be designed for different data types such as sequential or graph data
- Time-complexity of kernel methods generally scale quadratically in the number of training points (due to the kernel matrix), which can be a limitation when huge datasets are processed
- More on kernel methods on the course CS-E4830 Kernel methods in machine learning (Spring 2023)