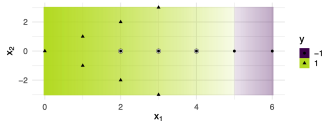


# Introduction to Machine Learning

## The Kernel Trick



### Learning goals

- Know how to efficiently introduce non-linearity via the kernel trick
- Know common kernel functions (linear, polynomial, radial)
- Know how to compute predictions of the kernel SVM

# DUAL SVM PROBLEM WITH FEATURE MAP

The dual (soft-margin) SVM is:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0, \end{aligned}$$

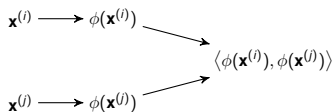
Here we replaced all features  $\mathbf{x}^{(i)}$  with feature-generated, transformed versions  $\phi(\mathbf{x}^{(i)})$ .

We see: The optimization problem only depends on **pair-wise inner products** of the inputs.

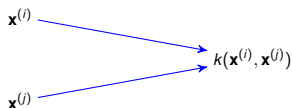
This now allows a trick to enable efficient solving.

# KERNEL = FEATURE MAP + INNER PRODUCT

Instead of first mapping the features to the higher-dimensional space and calculating the inner products afterwards,



it would be nice to have an efficient “shortcut” computation:



We will see: **Kernels** give us such a “shortcut”.

# MERCER KERNEL

**Definition:** A **(Mercer) kernel** on a space  $\mathcal{X}$  is a continuous function

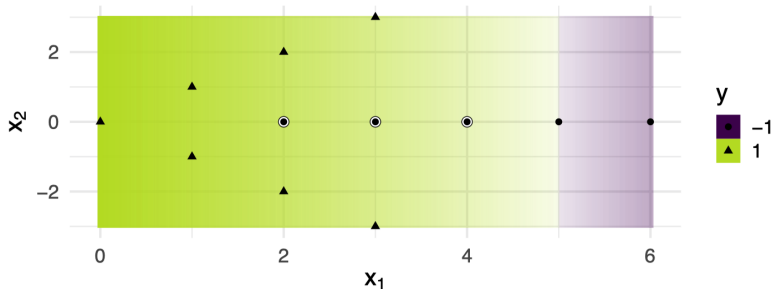
$$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$$

of two arguments with the properties

- Symmetry:  $k(\mathbf{x}, \tilde{\mathbf{x}}) = k(\tilde{\mathbf{x}}, \mathbf{x})$  for all  $\mathbf{x}, \tilde{\mathbf{x}} \in \mathcal{X}$ .
- Positive definiteness: For each finite subset  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$  the **kernel Gram matrix**  $\mathbf{K} \in \mathbb{R}^{n \times n}$  with entries  $K_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$  is positive semi-definite.

# CONSTANT AND LINEAR KERNEL

- Every constant function taking a non-negative value is a (very boring) kernel.
- An inner product is a kernel. We call the standard inner product  $k(\mathbf{x}, \tilde{\mathbf{x}}) = \mathbf{x}^\top \tilde{\mathbf{x}}$  the **linear kernel**. This is simply our usual linear SVM as discussed.



# SUM AND PRODUCT KERNELS

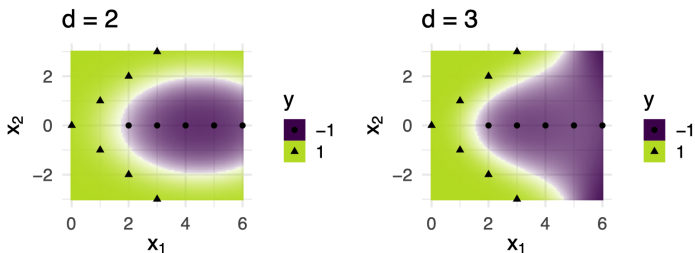
A kernel can be constructed from other kernels  $k_1$  and  $k_2$ :

- For  $\lambda \geq 0$ ,  $\lambda \cdot k_1$  is a kernel.
- $k_1 + k_2$  is a kernel.
- $k_1 \cdot k_2$  is a kernel (thus also  $k_1^n$ ).

The proofs remain as (simple) exercises.

# POLYNOMIAL KERNEL

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = (\mathbf{x}^\top \tilde{\mathbf{x}} + b)^d, \text{ for } b \geq 0, d \in \mathbb{N}$$



From the sum-product rules it directly follows that this is a kernel.

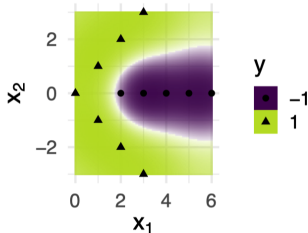
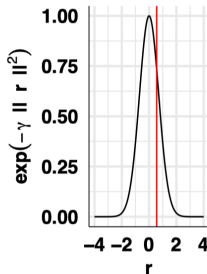
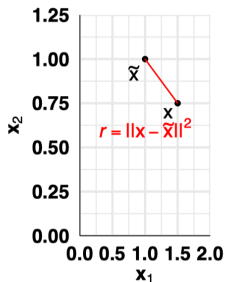
# RBF KERNEL

The “radial” **Gaussian kernel** is defined as

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

or

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp(-\gamma \|\mathbf{x} - \tilde{\mathbf{x}}\|^2), \gamma > 0$$





# KERNEL SVM

We kernelize the dual (soft-margin) SVM problem by replacing all inner products  $\langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle$  by kernels  $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0. \end{aligned}$$

This problem is still convex because  $\mathbf{K}$  is psd!

# KERNEL SVM

We kernelize the dual (soft-margin) SVM problem by replacing all inner products  $\langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle$  by kernels  $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0. \end{aligned}$$

This problem is still convex because  $\mathbf{K}$  is psd!

# KERNEL SVM

We kernelize the dual (soft-margin) SVM problem by replacing all inner products  $\langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle$  by kernels  $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0. \end{aligned}$$

In more compact matrix notation with  $\mathbf{K}$  denoting the kernel matrix:

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \mathbf{1}^\top \alpha - \frac{1}{2} \alpha^\top \text{diag}(\mathbf{y}) \mathbf{K} \text{diag}(\mathbf{y}) \alpha \\ \text{s.t.} \quad & \alpha^\top \mathbf{y} = 0, \\ & 0 \leq \alpha \leq C. \end{aligned}$$

This problem is still convex because  $\mathbf{K}$  is psd!

# KERNEL SVM: PREDICTIONS

For the linear soft-margin SVM we had:

$$f(\mathbf{x}) = \hat{\boldsymbol{\theta}}^T \mathbf{x} + \theta_0 \quad \text{and} \quad \hat{\boldsymbol{\theta}} = \sum_{i=1}^n \alpha_i y^{(i)} \mathbf{x}^{(i)}$$

After the feature map this becomes:

$$f(\mathbf{x}) = \langle \hat{\boldsymbol{\theta}}, \phi(\mathbf{x}) \rangle + \theta_0 \quad \text{and} \quad \hat{\boldsymbol{\theta}} = \sum_{i=1}^n \alpha_i y^{(i)} \phi(\mathbf{x}^{(i)})$$

Assuming that the dot-product still follows its bi-linear rules in the mapped space and using the kernel trick again:

$$\begin{aligned} \langle \hat{\boldsymbol{\theta}}, \phi(\mathbf{x}) \rangle &= \left\langle \sum_{i=1}^n \alpha_i y^{(i)} \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \right\rangle = \sum_{i=1}^n \alpha_i y^{(i)} \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \rangle = \\ &= \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}), \quad \text{so:} \quad f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0 \end{aligned}$$

# MNIST EXAMPLE

- Through this kernelization we can now conveniently perform feature generation even for higher-dimensional data. Actually, this is how we computed all previous examples, too.
- We again consider MNIST with  $28 \times 28$  bitmaps of gray values.
- A polynomial kernel extracts  $\binom{d+p}{d} - 1$  features and for the RBF kernel the dimensionality would be infinite.
- We train SVMs again on 700 observations of the MNIST data set and use the rest of the data for testing; and use  $C=1$ .



	Error
linear	0.134
poly (d = 2)	0.119
RBF (gamma = 0.001)	0.12
RBF (gamma = 1)	0.184

# FINAL COMMENTS

- The kernel trick allows us to make linear machines non-linear in a very efficient manner.
- Linear separation in high-dimensional spaces is **very flexible**.
- Learning takes place in the feature space, while predictions are computed in the input space.
- Both the polynomial and Gaussian kernels can be computed in linear time. Computing inner products of features is **much faster** than computing the features themselves.
- What if a good feature map  $\phi$  is already available? Then this feature map canonically induces a kernel by defining  $k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle$ . There is no problem with an explicit feature representation as long as it is efficiently computable.