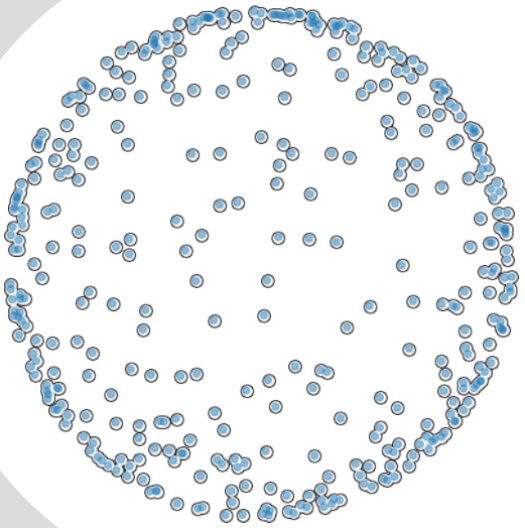
WiSe 2024/25

Machine Learning for Data Science

Lecture by G. Montavon

FREIE UNIVERSITÄT BERLIN



Lecture 10a | Kernels

Outline

Making ML models nonlinear

- Feature Expansions

From Feature Expansions to Kernels

- Computational benefits
- Kernelizing the distance computation
- Kernelizing the one-class SVM

From Kernels to Feature Expansions

- Testing positive semi-definiteness
- Commonly used kernels

From Linear to Nonlinear Models

Observation:

- Linear or quadratic models (e.g. Fisher discriminant, linear regression, etc.) are easy to train, but often lack the representation power to learn complex tasks.

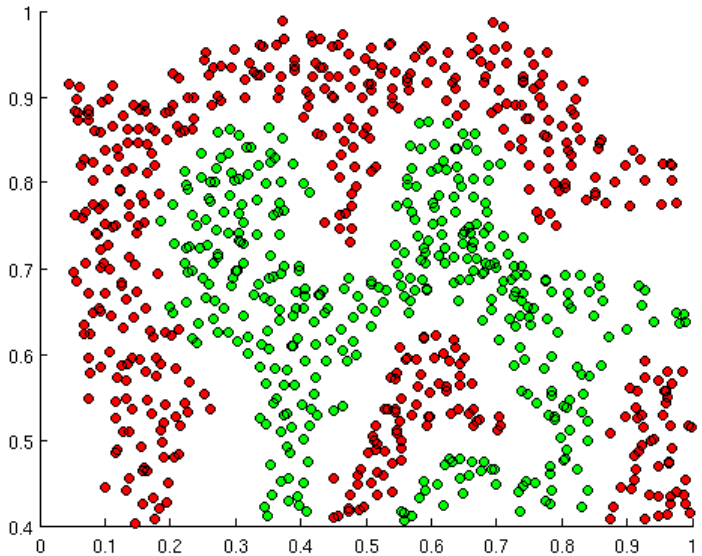


Image source: Ng. Stanford OpenClassroom ML course

Idea:

- No need to invent new algorithms. Apply some nonlinear transformation $\phi : \mathbb{R}^d \to \mathbb{R}^h$ to the data, and learn a *linear* model in the image domain:

$$f(\boldsymbol{x}) = \boldsymbol{\beta}^\top \phi(\boldsymbol{x}) + b$$

- The function ϕ is usually called a 'feature map'.

Part 1

Feature Expansions

Quadratic Features Expansions

General expression:

$$\phi(\boldsymbol{x}) = \begin{bmatrix} 1 \\ (x_i)_{i=1}^d \\ (x_i x_j)_{i,j=1}^d \end{bmatrix} \in \mathbb{R}^{(d+1)\cdot(d+2)/2}$$

Observation:

- If building a linear model on top of ϕ, we can show that it is equivalent to a quadratic model of $\boldsymbol{x}$:

  $$f(\boldsymbol{x}) = \boldsymbol{w}^\top \phi(\boldsymbol{x}) = \boldsymbol{x}^\top A \boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{x} + c$$

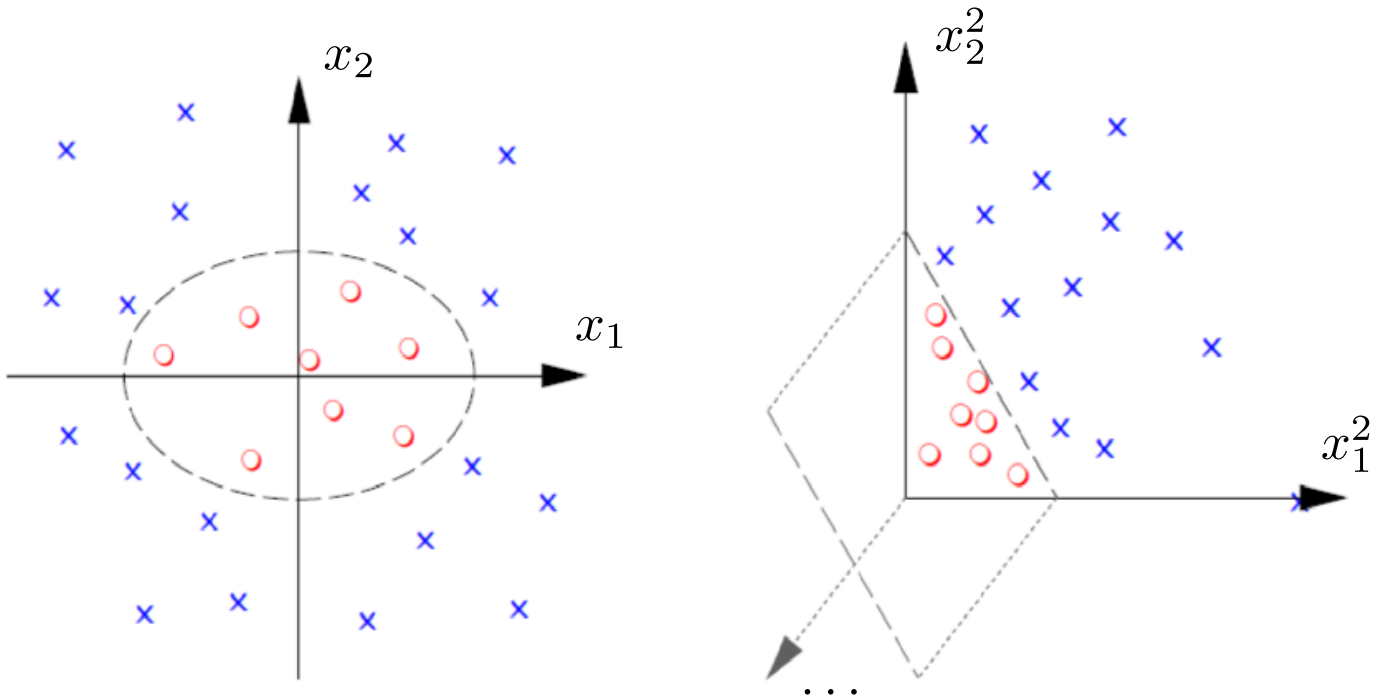
  for some parameters $A, \boldsymbol{b}, c$.

- We can using existing methods (e.g. PCA, linear regression, Fisher discriminant, ...), and increase their predictive power.

Linear Model in Feature Space

Example for $d = 2$:

$$\phi : \mathbb{R}^2 \to \mathbb{R}^6$$
$$(x_1, x_2) \mapsto (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$



Observation:

- The nonlinear (quadratic) problem is made linear in feature space.

From simple to more powerful discriminants

Difference of means:

- finds a linear projection that maximizes the difference between the two classes.

$$f(\boldsymbol{x}) = \boldsymbol{x}^\top (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

Fisher discriminant:

- finds a linear projection that maximizes the separability of the two classes.

$$f(\boldsymbol{x}) = \boldsymbol{x}^\top \Sigma^{-1} (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

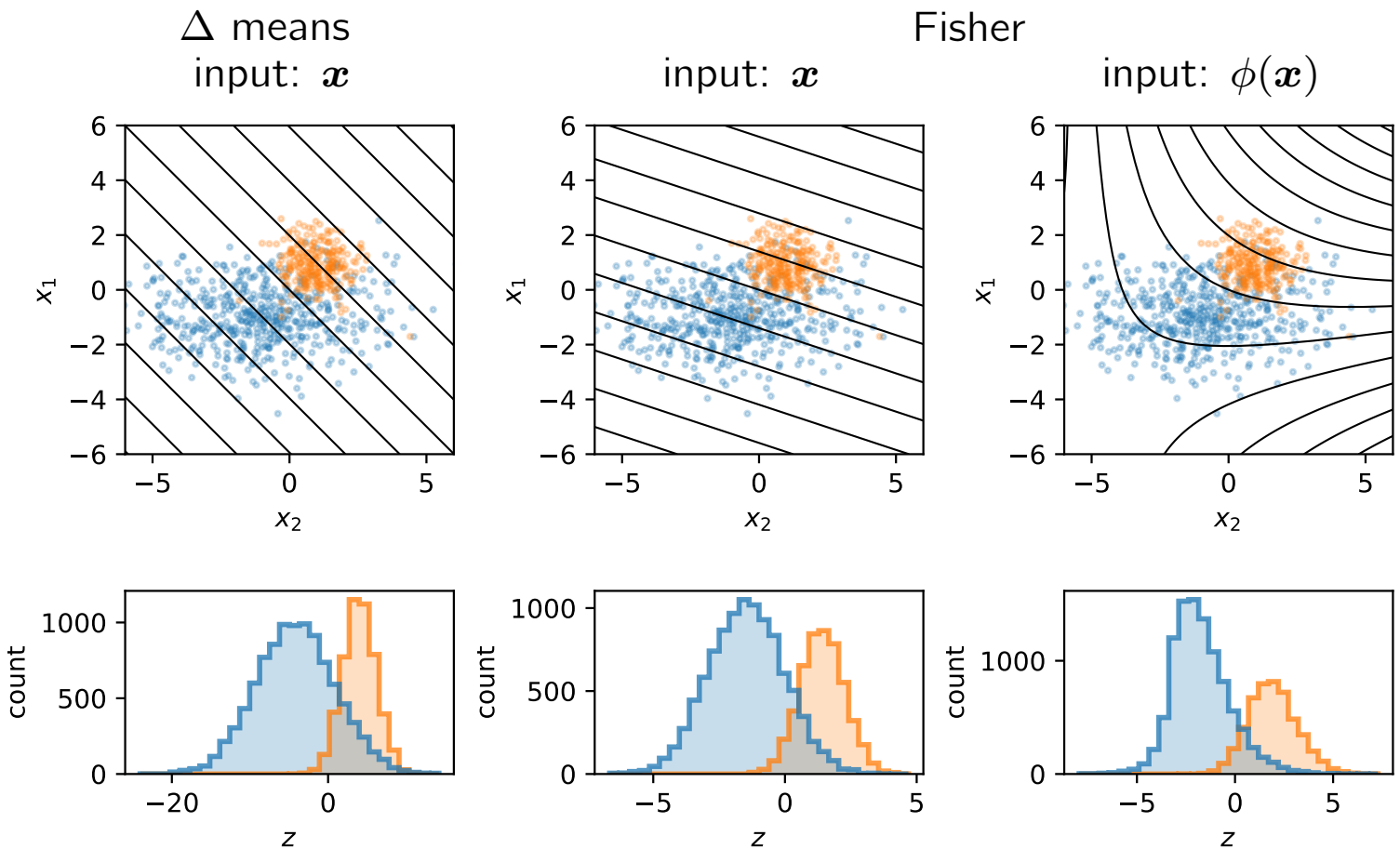
Fisher discriminant (with quadratic features):

- finds a nonlinear mapping that maximizes the separability of the two classes.

$$f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \Sigma_\phi^{-1} (\boldsymbol{\mu}_2^\phi - \boldsymbol{\mu}_1^\phi)$$

where $\boldsymbol{\mu}_1^\phi$, $\boldsymbol{\mu}_2^\phi$, Σ_ϕ indicate the means and covariances computed on $\phi(\boldsymbol{x})$ instead of $\boldsymbol{x}$.

Example



Observations:

- The nonlinear transformation gives more flexibility for increasing class separability, e.g. by bending the contour lines to mitigate the higher variance in the first class.

Other Nonlinear Expansions

Polynomial expansions:

- Generalization of quadratic expansions to higher degree:

$$\phi(\boldsymbol{x}) = \begin{bmatrix} 1 \\ (x_i)_{i=1}^d \\ (x_i x_j)_{i,j=1}^d \\ (x_i x_j x_k)_{i,j,k=1}^d \\ \vdots \end{bmatrix}$$

Hashing:

- Map to binary vectors testing for membership to some subspace, e.g.

$$\phi(\boldsymbol{x}) = \begin{bmatrix} \sigma(\boldsymbol{w}_1^\top \boldsymbol{x} + b_1) \\ \sigma(\boldsymbol{w}_2^\top \boldsymbol{x} + b_2) \\ \vdots \\ \sigma(\boldsymbol{w}_h^\top \boldsymbol{x} + b_h) \end{bmatrix}$$

with $(\boldsymbol{w}_i, b_i) \sim p(\boldsymbol{w}, b)$ some distribution and σ is some nonlinear function, e.g. a sign function, or a sinusoid.

Part 2

From Feature Expansions to Kernels

Distance Computation in Feature Space

Many prediction functions can be expressed in terms of dot products between data points in feature space (called 'kernels'):

Example:

- Distance to the mean:

$$
\begin{aligned}
d(\boldsymbol{x}) &= \|\phi(\boldsymbol{x}) - \overbrace{\tfrac{1}{N}\textstyle\sum_i \phi(\boldsymbol{x}_i)}^{\boldsymbol{\mu}^\phi}\| \\
&= \left(\|\phi(\boldsymbol{x}) - \tfrac{1}{N}\textstyle\sum_i \phi(\boldsymbol{x}_i)\|^2\right)^{\frac{1}{2}} \\
&= \Big(\underbrace{\phi(\boldsymbol{x})^\top \phi(\boldsymbol{x})}_{\text{dot product}} - \tfrac{2}{N}\textstyle\sum_i \underbrace{\phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}_i)}_{\text{dot product}} + \tfrac{1}{N^2}\textstyle\sum_{ij} \underbrace{\phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)}_{\text{dot product}}\Big)^{\frac{1}{2}}
\end{aligned}
$$

Question: Can we bypass the computation of the feature map when evaluating dot products?

The Kernel Trick

Example: Dot product computations can be computed very quickly for certain choices of feature maps ϕ (we refer to such dot products as kernels)

$$\left\langle \overbrace{\begin{bmatrix} 1 \\ \sqrt{2}(x_i)_{i=1}^d \\ (x_i x_j)_{i,j=1}^d \end{bmatrix}}^{\phi(\boldsymbol{x})}, \overbrace{\begin{bmatrix} 1 \\ \sqrt{2}(x'_i)_{i=1}^d \\ (x'_i x'_j)_{i,j=1}^d \end{bmatrix}}^{\phi(\boldsymbol{x}')} \right\rangle = 1 + \sqrt{2}\sqrt{2}\sum_{i=1}^d x_i x'_i + \sum_{i=1}^d \sum_{j=1}^d x_i x_j x'_i x'_j$$

$$= 1 + 2\sum_{i=1}^d x_i x'_i + \Big(\sum_{i=1}^d x_i x'_i\Big)\Big(\sum_{j=1}^d x_j x'_j\Big)$$

$$= \Big(1 + \sum_{i=1}^d x_i x'_i\Big)^2$$

$$= \underbrace{(1 + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle)^2}_{k(\boldsymbol{x}, \boldsymbol{x}')}$$

Observation:

- Evaluating $k(\boldsymbol{x}, \boldsymbol{x}')$ only requires $\approx d$ computations (instead of $\approx d^2$ computations if evaluating ϕ).

Computing ϕ vs. Computing k

Cost of making one prediction:

$$f(\boldsymbol{x}) = \overbrace{\sum_i \alpha_i \underbrace{\phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x})}_{k(\boldsymbol{x}_i, \boldsymbol{x})}}^{\boldsymbol{w}} + \theta$$

$\boldsymbol{w} \longrightarrow$ approach 1 (classic): $O(h)$

$k(\boldsymbol{x}_i, \boldsymbol{x}) \longrightarrow$ approach 2 (kernel): $O(N \cdot d)$

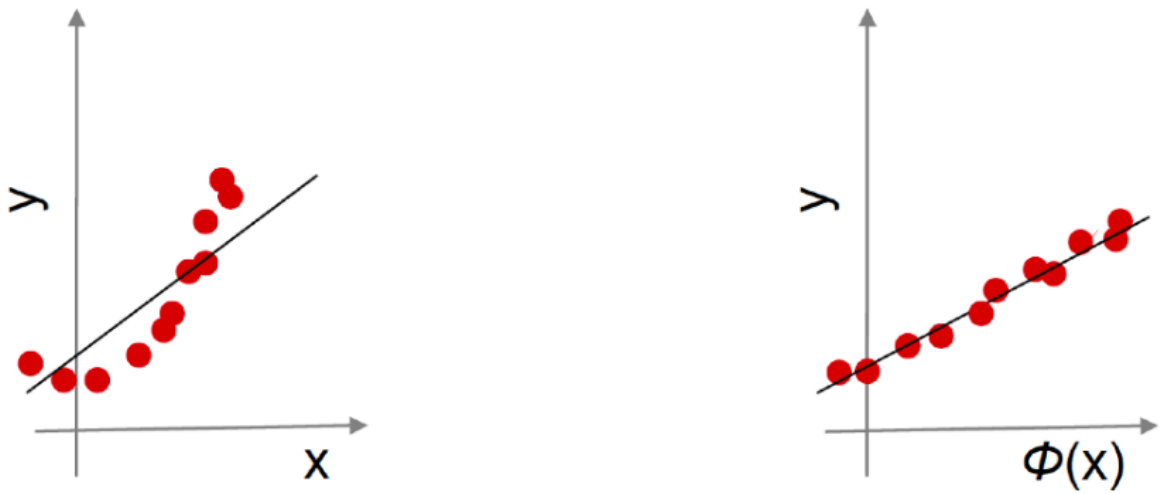
Examples:

N	d	(ϕ, k)	Approach 1 (classic)	Approach 2 (kernel)
1000	25	Linear	25	—
1000	25	Poly($q = 2$)	625	25000
1000	25	Poly($q = 3$)	15625	25000
1000	25	Poly($q = 4$)	390625	25000
1000	25	Poly($q = 5$)	9765625	25000

Kernel Ridge Regression

Ideas:

- Perform standard ridge regression in some feature space.



- Rewrite the resulting model in a way that the feature map computations only appear as dot products (so that they can be replaced by kernels).

Kernel Ridge Regression

- Redefine the prediction function $y = \boldsymbol{w}^\top \phi(\boldsymbol{x})$ where $\boldsymbol{w} \in \mathbb{R}^h$ and the objective to minimize as

$$\mathcal{E}(\boldsymbol{w}) = \frac{1}{N} \sum_{k=1}^{N} (\boldsymbol{w}^\top \phi(\boldsymbol{x}_k) - t_k)^2 \qquad \text{subject to} \quad \|\boldsymbol{w}\|^2 \leq C.$$

- As we have seen in the lecture on regression, the solution to this optimization problem has the form:

$$\boldsymbol{w} = (\phi(X)\phi(X)^\top + \lambda I)^{-1} \phi(X)\boldsymbol{t}$$

  where $\phi(X) = (\phi(\boldsymbol{x}_1)| \ldots |\phi(\boldsymbol{x}_N))$, and for an appropriate choice of parameter λ.

- A new data point is then predicted as

$$\begin{aligned} y = \boldsymbol{w}^\top \phi(\boldsymbol{x}) &= \phi(\boldsymbol{x})^\top \boldsymbol{w} \\ &= \phi(\boldsymbol{x})^\top (\phi(X)\phi(X)^\top + \lambda I)^{-1} \phi(X)\boldsymbol{t} \end{aligned}$$

Kernel Ridge Regression

from: $y = \phi(\boldsymbol{x})^\top (\underbrace{\phi(X)\phi(X)^\top}_{\bar{\Phi}} + \lambda I)^{-1} \phi(X)\boldsymbol{t}$

$$y = \phi(x)^T \underbrace{\left(\bar{\Phi}\bar{\Phi}^T + \lambda I\right)^{-1} \bar{\Phi}}_{\bar{\Phi}\left(\bar{\Phi}^T\bar{\Phi} + \lambda I\right)^{-1}} \left(\bar{\Phi}^T\bar{\Phi} + \lambda I\right)^{-1} t$$

$$= \underbrace{\phi(x)^T \bar{\Phi}}_{1\times h} \underbrace{\left(\bar{\Phi}^T\bar{\Phi}\right)}_{N\times N} + \lambda I)^{-1} t \quad (N\times 1)$$

$$= k(x, X)\left(k(X, X) + \lambda I\right)^{-1} \cdot t$$

we arrive at: $\boxed{y = k(\boldsymbol{x}, X)(K + \lambda I)^{-1}\boldsymbol{t}}$

Kernel Fisher Discriminant

We can kernelize the Fisher discriminant model by first observing that the projection on the Fisher discriminant can be written as:

$$
\begin{aligned}
y &= \phi(\boldsymbol{x})^\top \boldsymbol{w} \\
&= \phi(\boldsymbol{x})^\top (\Sigma_\phi + \lambda I)^{-1} \cdot (\boldsymbol{\mu}_2^\phi - \boldsymbol{\mu}_1^\phi) \\
&= \phi(\boldsymbol{x})^\top (\tfrac{1}{N}\phi(X)\phi(X)^\top + \lambda I)^{-1} (\boldsymbol{\mu}_2^\phi - \boldsymbol{\mu}_1^\phi) \\
&= \phi(\boldsymbol{x})^\top (\tfrac{1}{N}\phi(X)\phi(X)^\top + \lambda I)^{-1} \phi(X)\boldsymbol{t}
\end{aligned}
$$

where

$$
t_i = \begin{cases} \frac{1}{|\mathcal{G}_2|} & \text{if } \; i \in \mathcal{G}_2 \\ -\frac{1}{|\mathcal{G}_1|} & \text{if } \; i \in \mathcal{G}_1 \end{cases}
$$

and then proceed similarly as for kernelizing ridge regression.

Further Kernelizations

Many linear models can be kernelized.

Examples:

- Kernel PCA
- Kernel CCA
- Kernel SVM
- Kernel SVR
- Kernel One-Class SVM

Often, the term 'kernel' is omitted (e.g. the SVM modules in scikit learn uses by default a nonlinear kernel.

Part 3

From Kernels to Feature Spaces

Can we Choose any Function k in Practice?

Negative example:

- Suppose you would like to use the candidate kernel $k(\boldsymbol{x}, \boldsymbol{x}') = I(\boldsymbol{x} \neq \boldsymbol{x}')$ in an algorithm based on distance computation.
- Similarly to what we did before, the distance can be kernelized as:

$$\begin{aligned} d(\boldsymbol{x}, \boldsymbol{x}') &= \|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\| \\ &= \sqrt{\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\|^2} \\ &= \sqrt{k(\boldsymbol{x}, \boldsymbol{x}) - 2k(\boldsymbol{x}, \boldsymbol{x}') + k(\boldsymbol{x}', \boldsymbol{x}')}. \end{aligned}$$

Injecting our candidate kernel into this expression, we get

$$\begin{aligned} &= \sqrt{I(\boldsymbol{x} \neq \boldsymbol{x}) - 2I(\boldsymbol{x} \neq \boldsymbol{x}') + I(\boldsymbol{x}' \neq \boldsymbol{x}')} \\ &= \sqrt{0 - 2 + 0} \end{aligned}$$

which throws an error, because the square root does not apply to negative numbers.

Conclusion:

- Not all functions are valid kernels. Some conditions need to be verified.

Testing the Validity of a Kernel

[Mercer] If k is continuous, symmetric, and for all square-integrable functions $g(\boldsymbol{x})$ satisfies the condition:

$$\iint g(\boldsymbol{x})\, k(\boldsymbol{x}, \boldsymbol{x}')\, g(\boldsymbol{x}')\, d\boldsymbol{x}\, d\boldsymbol{x}' \geq 0$$

then the kernel can be expanded as:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{N_F} \lambda_i \psi_i(\boldsymbol{x}) \psi_i(\boldsymbol{x}')$$

with $\lambda_i > 0$, and $N_F \in \mathbb{N}$ or $N_F = \infty$ and we can construct the feature map

$$\phi(\boldsymbol{x}) := \begin{pmatrix} \sqrt{\lambda_1}\psi_1(\boldsymbol{x}) \\ \sqrt{\lambda_2}\psi_2(\boldsymbol{x}) \\ \vdots \end{pmatrix}$$

satisfying $\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle = k(\boldsymbol{x}, \boldsymbol{x}')$

Testing the Validity of a Kernel

In practice, to test whether the kernel is valid, we need to:

1. verify that the kernel is symmetric, i.e.:

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$$

   for all $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$.

2. show that it is positive semi-definite, i.e.:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$$

   for all sequences of points $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ and sequences of real numbers $(c_1, \ldots, c_N)$.

Example

Show that $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ with $k(\boldsymbol{x}, \boldsymbol{x}') = \alpha + x_1 x_1'$ and $\alpha > 0$, is a Mercer kernel.

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j \left(\alpha + x_1^{(i)} \cdot x_1^{(j)}\right)$$

$$= \sum_i \sum_j c_i c_j \alpha + \sum_i \sum_j c_i c_j x_1^{(i)} x_1^{(j)}$$

$$= \alpha \left(\sum_i c_i\right)\left(\sum_j c_j\right) + \left(\sum_i c_i x_1^{(i)}\right)\left(\sum_j c_j x_1^{(j)}\right) \geq 0$$

≥ 0 ≥ 0 ≥ 0 ≥ 0

Find a feature map $\phi(\boldsymbol{x})$ associated to this kernel

$$\phi(x) = \begin{pmatrix} \sqrt{\alpha} \\ x_1 \end{pmatrix}$$

$$\begin{pmatrix} \sqrt{\alpha} \\ x_1 \end{pmatrix}^T \begin{pmatrix} \sqrt{\alpha} \\ x_1' \end{pmatrix} = \alpha + x_1 x_1'$$

Example of Kernels

Examples of commonly used kernels satisfying the Mercer property

Linear	$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$	
Polynomial	$k(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \beta)^{\gamma}$	$\beta \in \mathbb{R}_{\geq 0}, \gamma \in \mathbb{N}$
Gaussian	$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma \lVert \boldsymbol{x} - \boldsymbol{x}' \rVert^2)$	$\gamma \in \mathbb{R}_{>0}$
Student	$k(\boldsymbol{x}, \boldsymbol{x}') = \dfrac{1}{\alpha + \lVert \boldsymbol{x} - \boldsymbol{x}' \rVert^2}$	$\alpha \in \mathbb{R}_{>0}$

Summary

Summary

- One approach to make models nonlinear is to pass the data through a nonlinear feature map ϕ, and then use a linear model on top of it.
- Feature maps can either be engineered based on existing knowledge, or it can be of generic type (e.g. quadratic expansion, polynomial expansion, or random hashing).
- Many learning algorithms applied in feature space can be expressed in terms of dot products in feature space.
- For certain types of feature maps, dot products can be computed without explicitly mapping to the feature space, and result in much faster computation, and we call these dot product computations 'kernels'.
- In practice, one cannot choose any kernel function. The kernel must be symmetric and positive semi-definite so that the existence of a corresponding feature map is ensured, and so that the various ML algorithms are guaranteed to remain applicable.