Machine Learning Fundamentals - LS2026
Kernel functions

Czech Technical University in Prague
V. Franc

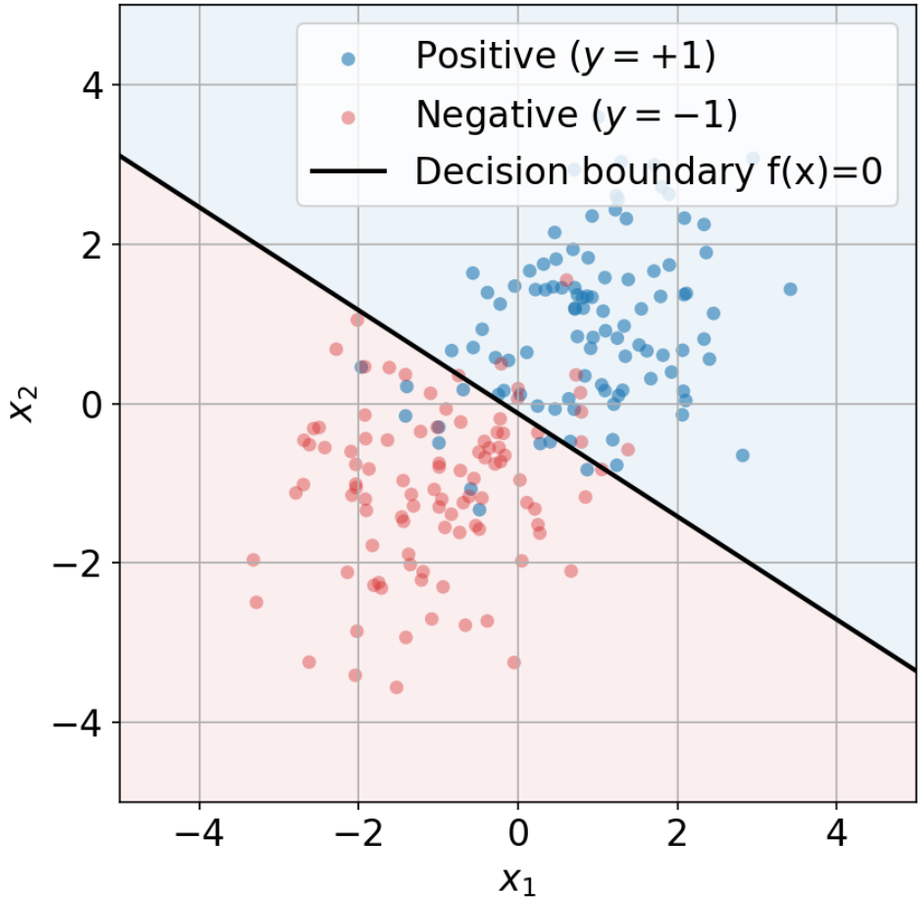
Linear models

- Linear classifier $h(\boldsymbol{x}; \boldsymbol{w}, b) = \operatorname{sign}(f(\boldsymbol{x}; \boldsymbol{w}, b))$ uses (affine) discriminant function

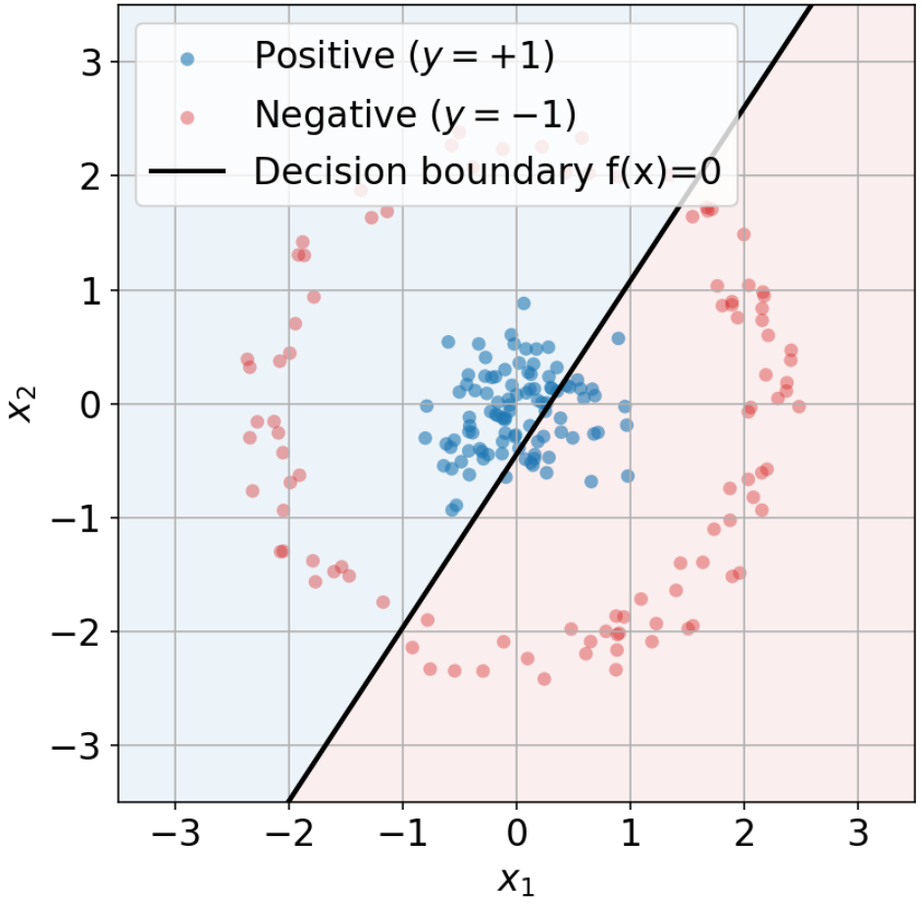
$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = \sum_{i=1}^{d} w_i \, x_i + b$$

where $\boldsymbol{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$ are parameters and $\boldsymbol{x} = [x_1, \ldots, x_d] \in \mathbb{R}^d$ is an input.

Example: Linear model is good



Linear model is not good

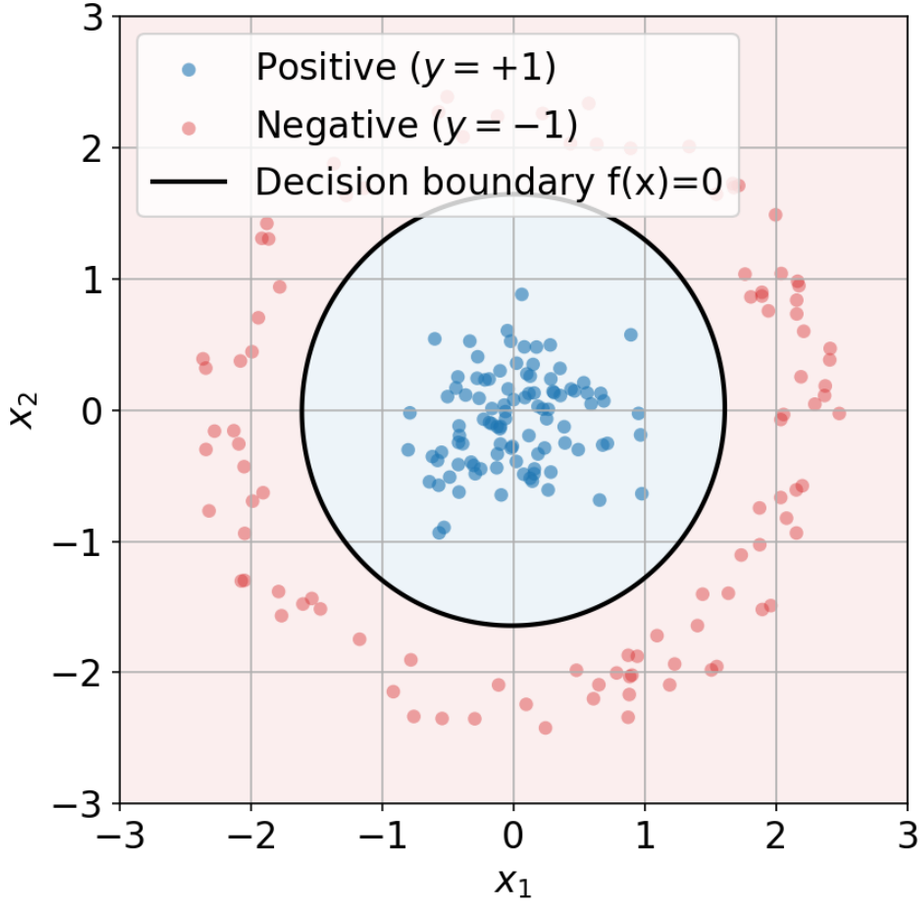


Non-linear feature mapping

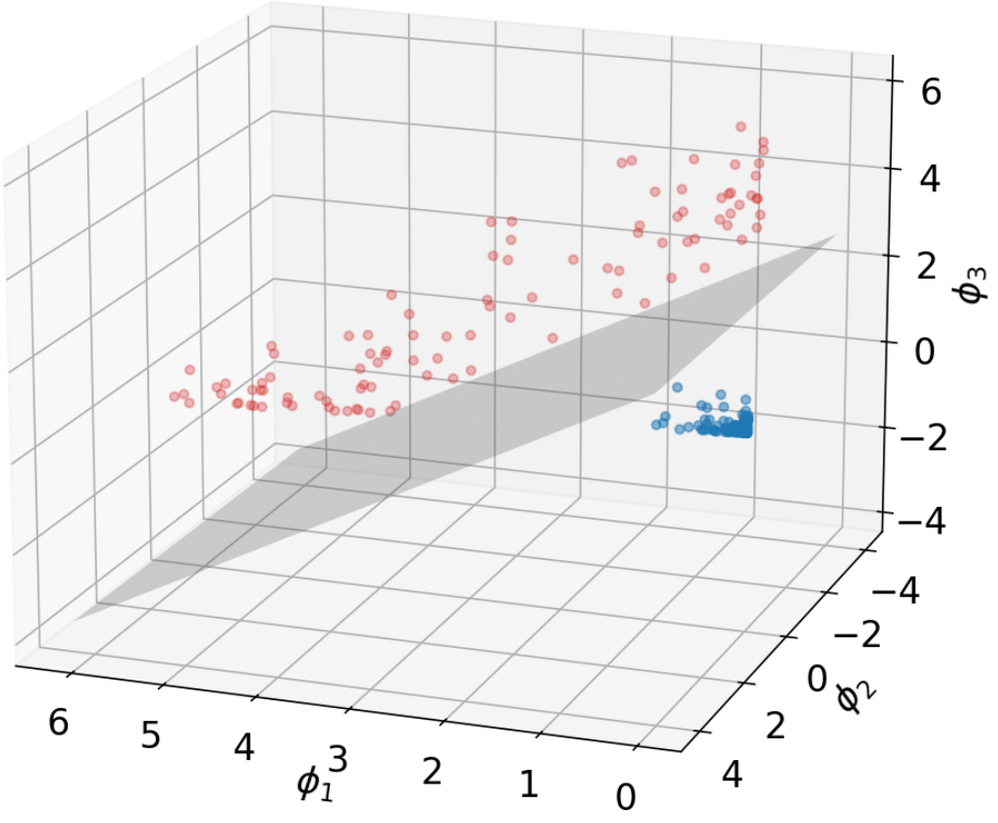
- Linear classifier $h(\boldsymbol{x}; \boldsymbol{w}, b) = \text{sign}(f(\boldsymbol{x}; \boldsymbol{w}, b))$ using non-linear features:

$$\begin{aligned} f(\boldsymbol{x}; \boldsymbol{w}, b) &= w_1 \phi_1(\boldsymbol{x}) + w_2 \phi_2(\boldsymbol{x}) + w_3 \phi_3(\boldsymbol{x}) + b \\ &= w_1 x_1^2 + w_2 \sqrt{2}\, x_1 x_2 + w_3 x_2^2 + b = \langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle + b \end{aligned}$$

Original features
$\boldsymbol{x} = [x_1, x_2] \in \mathbb{R}^2$



New features
$\boldsymbol{\phi}(\boldsymbol{x}) = [x_1^2, \sqrt{2} x_1 x_2, x_2^2] \in \mathbb{R}^3$



Perceptron Learning Algorithm

Task: Given training set $T_m = ((x_i, y_i) \in \mathcal{X} \times \{+1, -1\} \mid i = 1, \ldots, m)$ linearly separable under the feature map $\boldsymbol{\phi}\colon \mathcal{X} \to \mathbb{R}^n$, find

$$h(x; \mathbf{w}, b) = \operatorname{sign}(\langle \boldsymbol{\phi}(x), \boldsymbol{w} \rangle + b)$$

with zero training error.

Perceptron Learnig Algorithm (PLA)

---

1. $\boldsymbol{w} \leftarrow \mathbf{0}$, $b \leftarrow 0$
2. Find $(x_u, y_u) \in T_m$ such that:

$$y_y \neq \hat{y}_u = \operatorname{sign}(\langle \boldsymbol{w}, \boldsymbol{\phi}(x_u) \rangle + b)$$

3. If such (x_u, y_u) does not exist, return $(\boldsymbol{w}, b)$, otherwise update parameters:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y_u \, \boldsymbol{\phi}(x_u)\,, \quad b \leftarrow b + y_u$$

and go to step 2.

Perceptron Learning Algorithm

Example: Use the Perceptron Learning Algorithm to learn a quadratic classifier $h(\mathbf{x}; \mathbf{w}, b) = \operatorname{sign}\big(f(\mathbf{x}; \mathbf{w}, b)\big)$, where

$$\begin{aligned} f(\mathbf{x}; \mathbf{w}, b) &= x_1^2 w_1 + \cdots + x_d^2 w_d + \sqrt{2}\, x_1 x_2\, w_{d+1} + \cdots + \sqrt{2}\, x_{d-1} x_d\, w_n + b \\ &= \langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b \end{aligned}$$

with $\mathbf{w} \in \mathbb{R}^n$, $n = \frac{d(d+1)}{2}$, and feature map

$$\phi(\mathbf{x}) = \big[x_1^2, \ldots, x_d^2, \sqrt{2}\, x_1 x_2, \ldots, \sqrt{2}\, x_{d-1} x_d\big].$$

- Assume the number of input features is $d = 1{,}000{,}000$.
- Storing a single input vector $\mathbf{x} \in \mathbb{R}^d$ requires $d \cdot 4$ bytes ≈ 4 MB.
- Explicitly computing or storing $\phi(\mathbf{x})$ requires $\mathcal{O}(n)$ operations and memory, where

$$n = \frac{d(d+1)}{2} = \frac{1{,}000{,}000 \cdot 1{,}000{,}001}{2} \approx 5 \times 10^{11}.$$

- This corresponds to roughly 5×10^{11} multiplications/additions and

$$n \cdot 4 \text{ bytes} \approx 1.8 \text{ TiB of memory.}$$

Polynomial kernel of degree 2

- Consider quadratic feature map $\phi\colon \mathbb{R}^d \to \mathbb{R}^n$:

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left[x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{d-1}x_d\right]$$

  which maps d dimensional feature space to $n = \frac{d(d+1)}{2}$ feature space.

- Polynomial kernel of degree 2 computes the dot product in this feature space:

$$\boxed{k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle^2 = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle}$$

- In case of $d = 2$, we have

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{x}') &= (x_1\, x'_1 + x_2\, x'_2)^2 \\ &= x_1^2\, {x'}_1^2 + 2\, x_1\, x_2\, x'_1\, {x'}_2 + x_2^2\, {x'_2}^2 \\ &= \left\langle \left[x_1^2,\ \sqrt{2}\, x_1\, x_2\, , x_2^2\right], \left[{x'_1}^2,\ \sqrt{2}\, x'_1\, x'_2\, , {x'_2}^2\right] \right\rangle \\ &= \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle \end{aligned}$$

Why kernels matter: polynomial kernel of degree 2

Example: Let the input dimension be $d = 1{,}000{,}000$, and consider the quadratic feature map induced by the degree-2 polynomial kernel.

- Storing a single input vector $\boldsymbol{x} \in \mathbb{R}^d$ requires $d \cdot 4$ bytes ≈ 4 MB
- The corresponding quadratic feature map is

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left[x_1^2, \ldots, x_d^2, \sqrt{2}\, x_1 x_2, \ldots, \sqrt{2}\, x_{d-1} x_d\right]$$

  Its dimension is $n = \frac{d(d+1)}{2} = \frac{1{,}000{,}000 \cdot 1{,}000{,}001}{2} \approx 5 \times 10^{11}$.
- Explicitly computing or storing $\boldsymbol{\phi}(\boldsymbol{x})$ therefore requires $\mathcal{O}(n)$ operations and memory, that is roughly 5×10^{11} products/additions and $n \cdot 4$ bytes ≈ 1.8 TiB.
- In contrast, the degree-2 polynomial kernel can be evaluated directly as

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

  which requires only $\mathcal{O}(d)$ operations, i.e. about $1{,}000{,}000$ multiplications/additions.

The kernel computes the dot product in a huge feature space *without ever constructing the feature vector explicitly*.

Kernel perceptron

Task: Given training set $T_m = ((x_i, y_i) \in \mathcal{X} \times \{+1, -1\} \mid i = 1, \ldots, m)$ linearly separable under the feature map $\boldsymbol{\phi}\colon \mathcal{X} \to \mathbb{R}^n$, find $h(x) = \operatorname{sign}(\langle \boldsymbol{\phi}(x), \boldsymbol{w} \rangle + b)$ with zero training error.

Key idea: Represent the parameter vector as a linear combination of the training inputs

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \boldsymbol{\phi}(x_i)$$

Perceptron Learnig Algorithm

1. $\boldsymbol{w} \leftarrow \mathbf{0}$, $b \leftarrow 0$
2. Find $(x_u, y_u) \in T_m$ such that:

$$y_y \neq \hat{y}_u = \operatorname{sign}\left(\langle \boldsymbol{w}, \boldsymbol{\phi}(x_u) \rangle + b\right)$$

3. If such (x_u, y_u) does not exist, return $(\boldsymbol{w}, b)$, otherwise update parameters:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y_u \, \boldsymbol{\phi}(x_u)\,, \quad b \leftarrow b + y_u$$

and go to step 2.

PLA using only dot products

1. $\boldsymbol{\alpha} \leftarrow \mathbf{0}$, $b \leftarrow 0$
2. Find $(x_u, y_u) \in T_m$ such that:

$$y_y \neq \hat{y}_u = \operatorname{sign}\left(\sum_{i=1}^{m} \alpha_i \langle \boldsymbol{\phi}(x), \boldsymbol{\phi}(x_i) \rangle + b\right)$$

3. If such (x_u, y_u) does not exist return $(\boldsymbol{\alpha}, b)$, otherwise update parameters:

$$\alpha_u \leftarrow \alpha_u + y_u \quad b \leftarrow b + y_u$$

and go to step 2.

Kernel perceptron

Kernel Perceptron

Input: training sequence $T_m = ((x_i, y_i) \in \mathcal{X} \times \{+1, -1\} \mid i = 1, \ldots, m)$; kernel function $k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ such that $k(x, x') = \langle \boldsymbol{\phi}(x), \boldsymbol{\phi}(x') \rangle$ under some feature map $\boldsymbol{\phi}$.

Output: kernel classifier reprezented by the weights $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m] \in \mathbb{R}^m$ and bias $b \in \mathbb{R}$:

$$h(x; \boldsymbol{\alpha}, b) = \operatorname{sign}\left(\sum_{i=1}^{m} \alpha_i \, k(x, x_i) + b \right).$$

1. $\boldsymbol{\alpha} \leftarrow \mathbf{0}$, $b \leftarrow 0$
2. Find a misclassified example $(x_u, y_u) \in T_m$ such that:

$$y_y \neq \hat{y}_u = \operatorname{sign}\left(\sum_{i=1}^{m} \alpha_i k(x, x_i) \; + b \right)$$

3. If such (x_u, y_u) does not exist, return $(\boldsymbol{w}, b)$. Otherwise update parameters:

$$\alpha_u \leftarrow \alpha_u + y_u \quad \text{and} \quad b \leftarrow b + y_u$$

and go to step 2.

Explicit vs. kernel Perceptron

Compute and memory requirements

	Training set memory	Prediction compute	Single epoch compute
Standard PLA	$\mathcal{O}(m \cdot n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \cdot m)$
Kernel PLA	$\mathcal{O}(m^2)$	$\mathcal{O}(m \cdot K)$	$\mathcal{O}(m^2 \cdot K)$

m...number of training examples, d..input space dimension, n...feature space dimension, K...kernel eval. complexity

Examples of kernel functions

	$k(x, x')$	Input space $\mathcal{X}$	Feature space dimesnion n	Evaluation compute K
2nd polynomial	$\langle \boldsymbol{x}, \boldsymbol{x}' \rangle^2$	$\mathbb{R}^d$	$\frac{d(d+1)}{2}$	$\mathcal{O}(d)$
RBF	$\exp\left(-\gamma \lVert \boldsymbol{x} - \boldsymbol{x}' \rVert^2 \right)$	$\mathbb{R}^d$	∞	$\mathcal{O}(d)$
string sub-sequence	dynamic programming	$\cup_{d=0}^{\infty} \Sigma^d$	$\lvert\Sigma\rvert^q$	$\mathcal{O}(q\lvert s\rvert\lvert t\rvert)$

Recall that the number of iterations of PLA is at most $\frac{R^2}{\gamma^2}$, where $R = \max_{i=1,\ldots,m} \lVert \boldsymbol{\phi}(x_i) \rVert$ and $\gamma = \max_{\lVert \boldsymbol{w} \rVert = 1, b} \min_{i=1,\ldots,m} (y_i(\langle \boldsymbol{w}, \boldsymbol{\phi}(x_i) \rangle + b))$, i.e. it does no depend on m and n.

Radial basis function kernel

- Assume the observations are real-valued features: $\mathcal{X} = \mathbb{R}^d$
- The RBF kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|^2)$$

corresponds to dot product $\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle$ in infinite dimensional space.

- In one dimensional case, $\mathcal{X} = \mathbb{R}$ the RBF kernel reads

$$\begin{aligned} k(x, x') &= \exp(-\gamma (x - x')^2) \\ &= \exp(-\gamma\, x^2) \exp(-\gamma\, x'^2) \exp(2\, \gamma\, x\, x') \\ &= \exp(-\gamma\, x^2) \exp(-\gamma\, x'^2) \sum_{n=0}^{\infty} \frac{(2\, x\, x'\, \gamma)^n}{n!} \\ &= \exp(-\gamma\, x^2)\, \exp(-\gamma\, x'^2) \sum_{n=0}^{\infty} \frac{(\sqrt{2\,\gamma}\, x)^n}{\sqrt{n!}} \frac{(\sqrt{2\,\gamma}\, x')^n}{\sqrt{n!}} \end{aligned}$$

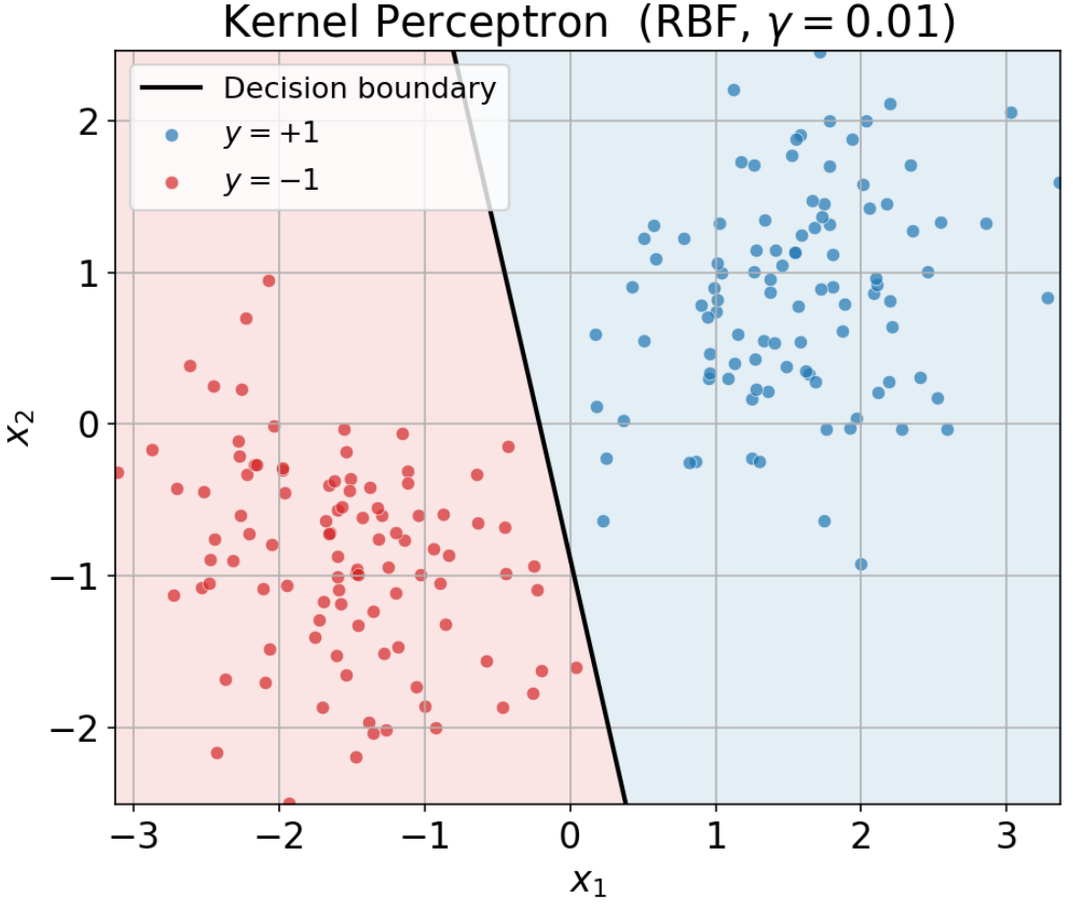
where we used the Taylor expansion $\exp(a) = \sum_{n=0}^{\infty} \frac{a^n}{n!}$

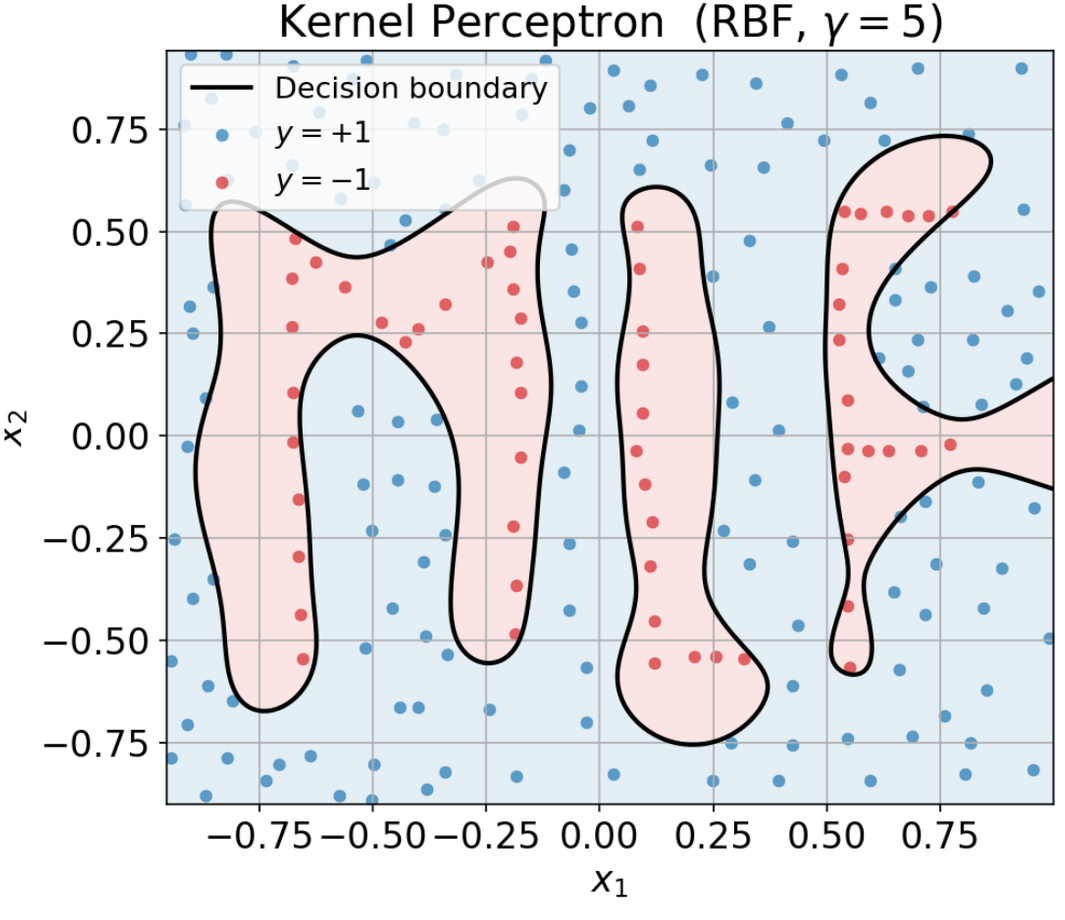
- The feature map $\boldsymbol{\phi}\colon \mathbb{R} \to \mathbb{R}^{\infty}$ (which cannot be evaluate explicitly) then reads:

$$\boldsymbol{\phi}(x) = \left[\exp(-\gamma\, x^2)\, , \exp(-\gamma\, x^2) \sqrt{2\gamma} x\, , \exp(-\gamma\, x^2) \frac{(\sqrt{2\gamma} x)^2}{\sqrt{2!}}\, , \exp(-\gamma\, x^2) \frac{(\sqrt{2\gamma} x)^3}{\sqrt{3!}}, \dots \right]$$

Radial basis function kernel

- Universal: can approximate arbitrary continuous decision functions on compact domains.
- Highly flexible, with a single scale parameter γ:
  - small γ: smoother / less complex boundary
  - large γ: more intricate boundary, higher overfitting risk





String Subsequence kernel

- Input space $\mathcal{X} = \cup_{d=1}^{\infty} \Sigma^d$ contains all strings on a finite alphabet Σ
- The features: occurrence+continuity of sub-sequences of length q:

$$\boldsymbol{\phi}\colon \mathcal{X} \to \mathbb{R}^{|\Sigma|^q} \qquad \text{and} \qquad \phi_u(s) = \sum_{\boldsymbol{i}\colon\, u=s[\boldsymbol{i}]} \lambda^{l(\boldsymbol{i})}\,, \quad \forall u \in \Sigma^q$$

where $s[\boldsymbol{i}]$ denotes a sub-sequence of $\boldsymbol{s}$ and $\lambda \in (0, 1]$ is a decay factor.

- Example for strings "cat", "car", "bat" and "bar" and $q = 2$:

	ca	ct	at	ba	bt	cr	ar	br
$\boldsymbol{\phi}(\text{"}cat\text{"})$	λ^2	λ^3	λ^2	0	0	0	0	0
$\boldsymbol{\phi}(\text{"}car\text{"})$	λ^2	0	0	0	0	λ^3	λ^2	0
$\boldsymbol{\phi}(\text{"}bat\text{"})$	0	0	λ^2	λ^2	λ^3	0	0	0
$\boldsymbol{\phi}(\text{"}bar\text{"})$	0	0	0	λ^2	0	0	λ^2	λ^3

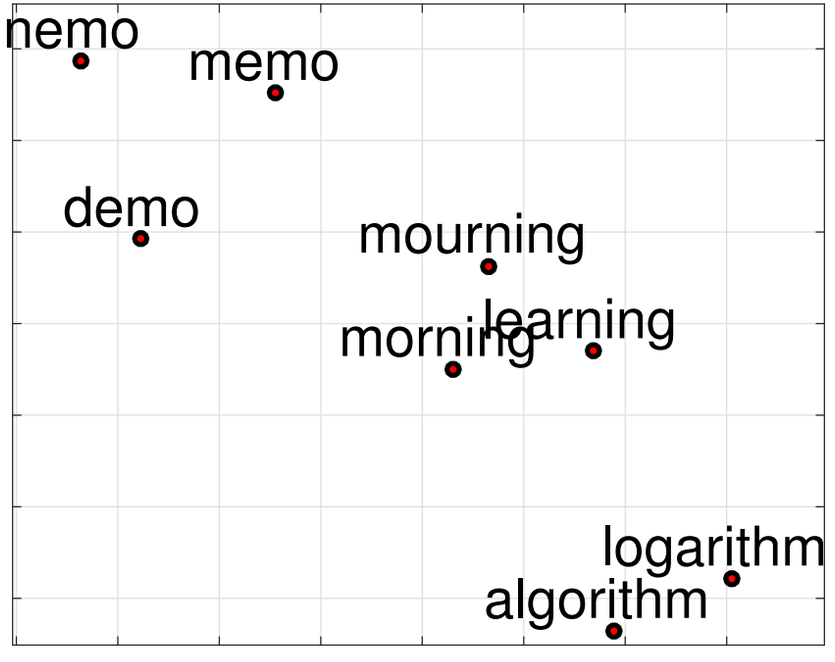
$k(\text{"}cat\text{"}, \text{"}car\text{"}) = \lambda^4$, $k(\text{"}cat\text{"}, \text{"}bat\text{"}) = \lambda^4$, $k(\text{"}cat\text{"}, \text{"}bar\text{"}) = 0$, ...

- The kernel value $k(s, t)$ can be computed by dynamic programming in time $\mathcal{O}(q\,|s|\,|t|)$.

String Subsequence kernel

Example: Consider substring kernel for sub-sequence length $q = 2$ and decay factor $\lambda = 0.4$:

Feature space
(2D embedding via tSNE)



Kernel matrix

	algorithm	logarithm	learning	morning	mourning	demo	memo	nemo
algorithm	0.24	0.15	0.01	0.04	0.02	0.00	0.00	0.00
logarithm	0.15	0.24	0.04	0.02	0.01	0.00	0.00	0.00
learning	0.01	0.04	0.23	0.13	0.13	0.00	0.00	0.00
morning	0.04	0.02	0.13	0.19	0.17	0.03	0.03	0.03
mourning	0.02	0.01	0.13	0.17	0.23	0.03	0.03	0.03
demo	0.00	0.00	0.00	0.03	0.03	0.09	0.06	0.06
memo	0.00	0.00	0.00	0.03	0.03	0.06	0.09	0.06
nemo	0.00	0.00	0.00	0.03	0.03	0.06	0.06	0.09

- Kernel $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ can be thought of as a similarity measure between the inputs embedded to a feature space:

$$k(\mathrm{A}, \mathrm{B}) = \text{similarity between input A and B}$$

Positive definite kernel

Definition 1. *Let $\mathcal{X}$ be a non-empty set. The function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a* positive definite kernel *if it is* symmetric *and for any finite set of inputs $x_1, \ldots, x_m$, the* kernel matrix $\mathbf{K} \in \mathbb{R}^{m \times m}$ *with elements $K_{i,j} = k(x_i, x_j)$ is positive semi-definite.*

- The kernel matrix $\mathbf{K} \in \mathbb{R}^{m \times m}$ represents similarities between each pair of inputs $(x_1, \ldots, x_m)$:

$$\mathbf{K} = \begin{bmatrix} k(x_1, x_1), & k(x_1, x_2), & \ldots, & k(x_1, x_m) \\ k(x_2, x_1), & k(x_2, x_2), & \ldots, & k(x_2, x_m) \\ \vdots & & & \\ k(x_m, x_1), & k(x_m, x_2), & \ldots, & k(x_m, x_m) \end{bmatrix}$$

- A matrix $\mathbf{K} \in \mathbb{R}^{m \times m}$ is PSD if for every $\boldsymbol{\alpha} \in \mathbb{R}^m$, $\boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \geq 0$.

Every kernel defines a feature space

Theorem 1. *For every positive definite kernel* $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *there exists a Hilbert space* $\mathcal{H}$ *and a feature map* $\phi\colon \mathcal{X} \to \mathcal{H}$ *such that* $k(x, x') = \langle \phi(x), \phi(x') \rangle$.

Proof for a kernel defined on a finite input space $\mathcal{X}$:

The kernel matrix $\mathbf{K} \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{X}|}$ is symmetric and PSD hence the spectral decomposition exists $\mathbf{K} = \mathbf{V}\mathbf{D}\mathbf{V}^T$ where $\mathbf{V} \in \mathbb{R}^{m \times m}$ is orthogonal and $\mathbf{D} = \operatorname{diag}(\lambda_1, \ldots, \lambda_{|\mathcal{X}|})$ is diagonal matrix of non-negative eigenvalues.

Therefore $\mathbf{K} = \mathbf{\Phi}^T\mathbf{\Phi}$ where $\mathbf{\Phi}^T = \mathbf{V}\mathbf{D}^{\frac{1}{2}}$.

- A Hilbert space is a vector space, i.e. a set of objects that you can add together and multiply by a number, equipped with a dot product, which lets you measure lengths and angles between vectors.

  It generalizes ordinary Euclidean space to possibly infinite dimensions.

Construction of valid kernels

Let $k_1 \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $k_2 \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be p.d. kernels, $\lambda \in [0, 1]$, $\alpha \geq 0$, $\boldsymbol{\phi} \colon \mathcal{X} \to \mathbb{R}^n$ and $k_3 \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ a p.d. kernel, $\mathbf{K}$ a symmetric positive definite matrix. Then the following functions are also p.d. kernels:

$$
\begin{aligned}
k(x, z) &= (1 - \lambda)\, k_1(x, z) + \lambda\, k_2(x, z) \\
k(x, z) &= \alpha\, k_1(x, z) \\
k(x, z) &= k_1(x, z)\, k_2(x, z) \\
k(x, z) &= k_3(\boldsymbol{\phi}(x), \boldsymbol{\phi}(z)) \\
k(\boldsymbol{x}, \boldsymbol{z}) &= \boldsymbol{x}^T \mathbf{K} \boldsymbol{z}
\end{aligned}
$$

Summary

- **Motivation:** Linear models may fail in the original input space, but can become linear after a nonlinear feature map $f(x) = \langle \mathbf{w}, \boldsymbol{\phi}(x) \rangle + b$.
- **Kernel trick:** avoid constructing the high-dimensional feature vector explicitly by using $k(x, x') = \langle \boldsymbol{\phi}(x), \boldsymbol{\phi}(x') \rangle$.
- **Kernel perceptron:** represent the classifier by training examples

$$h(x) = \operatorname{sign}\left(\sum_{i=1}^{m} \alpha_i \, k(x, x_i) + b \right),$$

  so learning and prediction use only kernel evaluations.
- **Examples of kernels:**
  - polynomial kernel: finite nonlinear feature space,
  - RBF kernel: implicit infinite-dimensional feature space,
  - string subsequence kernel: similarity for structured inputs such as strings.
- **Key takeaway:** Kernels enable nonlinear decision boundaries while keeping algorithms formulated only through similarities between examples.