Introduction to Machine Learning

Non-linear Prediction with Kernels

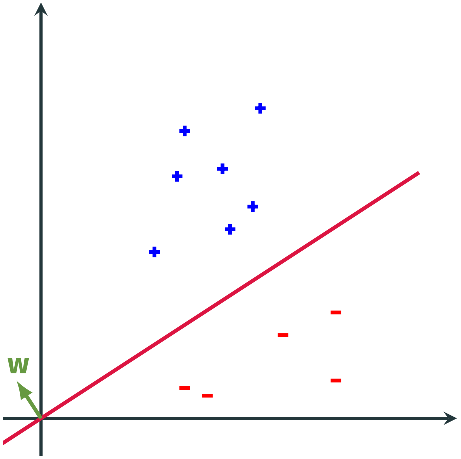
Nils M. Kriege

WS 2023

Data Mining and Machine Learning
Faculty of Computer Science
University of Vienna

Credit: Slides based on the IML Lectures by Sebastian Tschiatschek and Andreas Krause

Recall: Linear classifiers



$$\hat{y} = \text{sign}(\mathbf{w}^T\mathbf{x})$$

Recall: The Perceptron problem

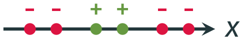
- Solve
$$\hat{w} = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell_P(\mathbf{w}; y_i, \mathbf{x}_i)$$
where
$$\ell_P(\mathbf{w}; y_i, \mathbf{x}_i) = \max(0, -y_i \mathbf{w}^T \mathbf{x}_i)$$
- Optimize via Stochastic Gradient Descent

Solving non-linear classification tasks

- How can we find nonlinear classification boundaries?

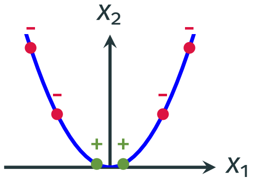
Solving non-linear classification tasks

- How can we find nonlinear classification boundaries?
- Similar as in regression, can use non-linear transformations of the feature vectors, followed by linear classification



$x \mapsto [x_1, x_2]$

$\Rightarrow$



Recall: Linear regression for polynomials

- We can fit non-linear functions via linear regression, using nonlinear features of our data (basis functions):

$$f(\mathbf{x}) = \sum_{i=1}^{d} w_i \phi_i(\mathbf{x})$$

- For example: polynomials (in 1d):

$$f(x) = \sum_{i=0}^{m} w_i x^i$$

Polynomials in higher dimensions

- Suppose we wish to use polynomial features, but our input is higher-dimensional
- Can still use monomial features
- **Example**: Monomials in 2 variables, degree = 2;

$$\mathbf{x} = [x_1, x_2] \quad \mapsto \quad \phi(\mathbf{x}) = [x_1^2, x_2^2, x_1 x_2]$$

Avoiding the feature explosion

- Need $O(d^k)$ dimensions to represent (multivariate) polynomials of degree k on d features

Avoiding the feature explosion

- Need $O(d^k)$ dimensions to represent (multivariate) polynomials of degree k on d features
- ⚠ Example: $d = 10000$, $k = 2 \Rightarrow$ Need $\sim 100M$ dimensions

Avoiding the feature explosion

- Need $O(d^k)$ dimensions to represent (multivariate) polynomials of degree k on d features
- ⚠ Example: $d = 10000$, $k = 2 \Rightarrow$ Need $\sim 100M$ dimensions
- 💡 In the following, we can see how we can efficiently implicitly operate in such high-dimensional feature spaces (i.e., without ever explicitly computing the transformation)

Revisiting the Perceptron/SVM

- 💡 Fundamental insight: Optimal hyperplane lies in the span of the data:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

Revisiting the Perceptron/SVM

- 💡 Fundamental insight: Optimal hyperplane lies in the span of the data:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

- **(Handwavy) proof**: (Stochastic) gradient descent starting from 0 constructs such a representation:
    - Perceptron: $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t y_t \mathbf{x}_t [y_t \mathbf{w}_t \mathbf{x}_t < 0]$
    - SVM: $\mathbf{w}_{t+1} = \mathbf{w}_t (1 - 2\lambda\eta_t) + \eta_t y_t \mathbf{x}_t [y_t \mathbf{w}_t \mathbf{x}_t < 1]$

Revisiting the Perceptron/SVM

- 💡 Fundamental insight: Optimal hyperplane lies in the span of the data:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

- **(Handwavy) proof**: (Stochastic) gradient descent starting from 0 constructs such a representation:
    - Perceptron: $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t y_t \mathbf{x}_t [y_t \mathbf{w}_t \mathbf{x}_t < 0]$
    - SVM: $\mathbf{w}_{t+1} = \mathbf{w}_t (1 - 2\lambda\eta_t) + \eta_t y_t \mathbf{x}_t [y_t \mathbf{w}_t \mathbf{x}_t < 1]$
- **More abstract proof:** Follows from so called “representer theorems” (discussed later)

Reformulating the Perceptron

Idea: Replace $\mathbf{w}$ by $\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \max(0, -y_i \mathbf{w}^T \mathbf{x}_i)$$

$$\hat{\boldsymbol{\alpha}} = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \max \left\{ 0, -y_i \left(\sum_{j=1}^n \alpha_j y_j \mathbf{x}_j \right)^T \mathbf{x}_i \right\}$$

$$= \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \max \left\{ 0, -y_i \sum_{j=1}^n \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i \right\}$$

$$= \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \max \left\{ 0, - \sum_{j=1}^n \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \right\}$$

Advantage of reformulation

$$\hat{\alpha} = \operatorname*{argmin}_{\boldsymbol{\alpha}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j\}$$

Advantage of reformulation

$$\hat{\alpha} = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j\}$$

- 💡 **Key observation:** Objective only depends on inner products of pairs of data points
- Thus, we can implicitly work in high-dimensional spaces, as long as we can do inner products efficiently:

$$\mathbf{x} \mapsto \phi(\mathbf{x})$$
$$\mathbf{x}^T\mathbf{x}' \mapsto \phi(\mathbf{x})^T\phi(\mathbf{x}') =: k(\mathbf{x}, \mathbf{x}')$$

“Kernels = efficient inner products”

- Often, $k(\mathbf{x}, \mathbf{x}')$ can be computed much more efficiently than $\phi(\mathbf{x})^T\phi(\mathbf{x})'$

“Kernels = efficient inner products”

- Often, $k(\mathbf{x}, \mathbf{x}')$ can be computed much more efficiently than $\phi(\mathbf{x})^T\phi(\mathbf{x})'$
- Simple example: Polynomial kernel in degree 2

$$
\begin{aligned}
\mathbf{x} = [x_1, x_2]^T &\mapsto \phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2] \\
\phi(x)^T\phi(x') &= x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2x_1x_2x'_1x'_2 \\
&= \left(\mathbf{x}^T\mathbf{x}'\right)^2 =: k(\mathbf{x}, \mathbf{x}')
\end{aligned}
$$

Embedding: #+: 2 $\quad$ #$*$: 3+3+4=10

Functional: #+: 1 $\quad$ #$*$: 3

Polynomial kernels (degree 2)

- Suppose $\mathbf{x} = [x_1, \ldots, x_d]^T$ and $\mathbf{x}' = [x'_1, \ldots, x'_d]^T$
- Then

$$
\begin{aligned}
(\mathbf{x}^T\mathbf{x}')^2 = \left(\sum_{i=1}^{d} x_i x'_i\right)^2 &= \sum_{i=1}^{d} x_i^2 x'^2_i + 2 \sum_{1\le i<j\le d} x_i x'_i x_j x'_j \\
&= \phi(\mathbf{x})^T \phi(\mathbf{x}'), \text{ where} \\
\phi(\mathbf{x}) = [x_1^2, \ldots, x_d^2, &\sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \ldots, \sqrt{2}x_{d-1}x_d]^T
\end{aligned}
$$

Polynomial kernels: Fixed degree

- The kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^m$ implicitly represents all monomials of degree m

Polynomial kernels: Fixed degree

- The kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^m$ implicitly represents all monomials of degree m

$$x_1^m, x_2^m, \ldots, x_d^m, x_1^{m-1} x_2, \ldots, x_1^{m-1} x_d, \ldots, x_1 \cdots x_m, \cdots, x_{d-m-1} \cdots x_d$$

# Monomials of degree m in d variables:

$$\binom{d+m-1}{m} = O(d^m)$$

Polynomial kernels: Fixed degree

- The kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^m$ implicitly represents all monomials of degree m

  $x_1^m, x_2^m, \ldots, x_d^m, x_1^{m-1} x_2, \ldots, x_1^{m-1} x_d, \ldots, x_1 \cdots x_m, \cdots, x_{d-m-1} \cdots x_d$

  # Monomials of degree m in d variables:

$$\binom{d + m - 1}{m} = O(d^m)$$

- How can we get monomials up to order m?

Polynomial kernels

- The polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^m$ implicitly represents all monomials of up to degree m

Polynomial kernels

- The polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^m$ implicitly represents all monomials of up to degree m

$$1, x_1, x_2, \ldots, x_d, x_1^2, x_2^2, \ldots, x_d^2, x_1 x_2 \cdots x_m, \ldots$$

# Monomials of degree m in d variables:

$$\binom{d+m}{m} = O(d^m)$$

Polynomial kernels

- The polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^m$ implicitly represents all monomials of up to degree m

$$1, x_1, x_2, \ldots, x_d, x_1^2, x_2^2, \ldots, x_d^2, x_1 x_2 \cdots x_m, \ldots$$

# Monomials of degree m in d variables:

$$\binom{d+m}{m} = O(d^m)$$

- ⚠ Representing the monomials (and computing inner product explicitly) is exponential in m!

The “Kernel Trick”

Kernel Trick

- Express problem s.t. it only depends on inner products
- Replace inner products by kernels

The "Kernel Trick"

Kernel Trick

- Express problem s.t. it only depends on inner products
- Replace inner products by kernels

$$\mathbf{x}_i^T \mathbf{x}_j \quad \Rightarrow \quad k(\mathbf{x}_i, \mathbf{x}_j)$$

- This "trick" is very widely applicable!

The “Kernel Trick”

Kernel Trick

- Express problem s.t. it only depends on inner products
- Replace inner products by kernels

- Example: Perceptron

$$\hat{\alpha} = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j\}$$

$$\Downarrow$$

$$\hat{\alpha} = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j k(\mathbf{x}_i^T \mathbf{x}_j)\}$$

The "Kernel Trick"

Kernel Trick

- Express problem s.t. it only depends on inner products
- Replace inner products by kernels

- Example: Perceptron

$$\hat{\alpha} = \underset{\alpha}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j\}$$

$$\Downarrow$$

$$\hat{\alpha} = \underset{\alpha}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j k(\mathbf{x}_i^T \mathbf{x}_j)\}$$

- Will see further examples later

Derivation: Kernelized Perceptron

Perceptron

Training:

```
$\mathbf{w}_0 \leftarrow \mathbf{0}$
For $t = 0, 1, 2, \ldots$ do
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$
    Sample $(\mathbf{x}_i, y_i) \in \mathcal{D}$
    if $y_i \mathbf{w}_t^T \mathbf{x}_i < 0$
        $\mathbf{w}_{t+1} \leftarrow \eta_t \mathbf{w}_{t+1} + y_i \mathbf{x}_i$
```

Prediction:

$\text{sign}(\mathbf{w}^T \mathbf{x})$

Derivation: Kernelized Perceptron

Perceptron

Training:

```
$\mathbf{w}_0 \leftarrow \mathbf{0}$
For $t = 0, 1, 2, \ldots$ do
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$
    Sample $(\mathbf{x}_i, y_i) \in \mathcal{D}$
    if $y_i \mathbf{w}_t^T \mathbf{x}_i < 0$
        $\mathbf{w}_{t+1} \leftarrow \eta_t \mathbf{w}_{t+1} + y_i \mathbf{x}_i$
```

Prediction:

$\text{sign}(\mathbf{w}^T \mathbf{x})$

Kernelized Perceptron

Training:

```
$\boldsymbol{\alpha}^{(0)} \leftarrow \mathbf{0}$
For $t = 0, 1, 2, \ldots$ do
    $\boldsymbol{\alpha}^{(t+1)} \leftarrow \boldsymbol{\alpha}^{(t)}$
    Sample $(\mathbf{x}_i, y_i) \in \mathcal{D}$
    if $y_i \sum_{j=1}^{n} \alpha_j^{(t)} y_j \mathbf{x}_j^T \mathbf{x}_i < 0$
        $\alpha_i^{(t+1)} \leftarrow \alpha_i^{(t+1)} + \eta_t$
```

Prediction:

$\text{sign}(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x})$

Kernelized Perceptron

Training

- Initialize $\alpha_1 = \cdots = \alpha_n = 0$
- For $t = 1, 2, \ldots$
  - Pick data point $(\mathbf{x}_i, y_i)$ uniformly at random
  - Predict
$$\hat{y} = \text{sign}\Big(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i)\Big)$$
  - if $\hat{y} \neq y_i$ set $\alpha_i \leftarrow \alpha_i + \eta_t$

Prediction

- For new point $\mathbf{x}$ predict
$$\hat{y} = \text{sign}\Big(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x})\Big)$$

Demo: Kernelized Perceptron

[kernelized-perceptron.ipynb]

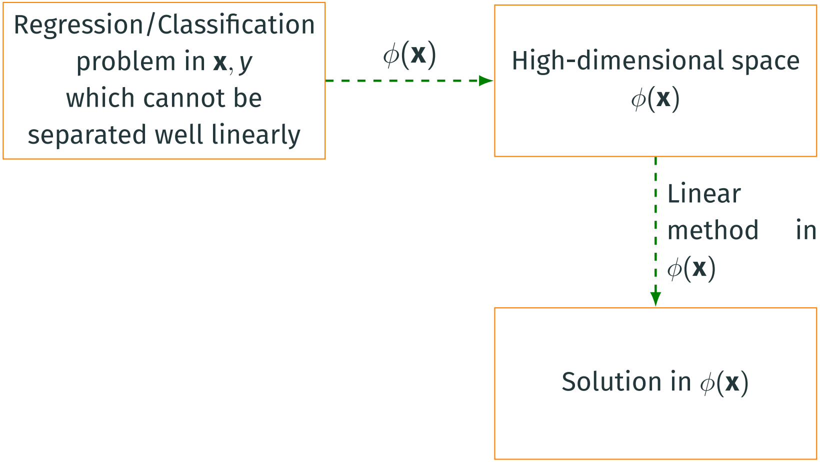
The Kernel Trick: Summary

Regression/Classification problem in $\mathbf{x}, y$ which cannot be separated well linearly

The Kernel Trick: Summary

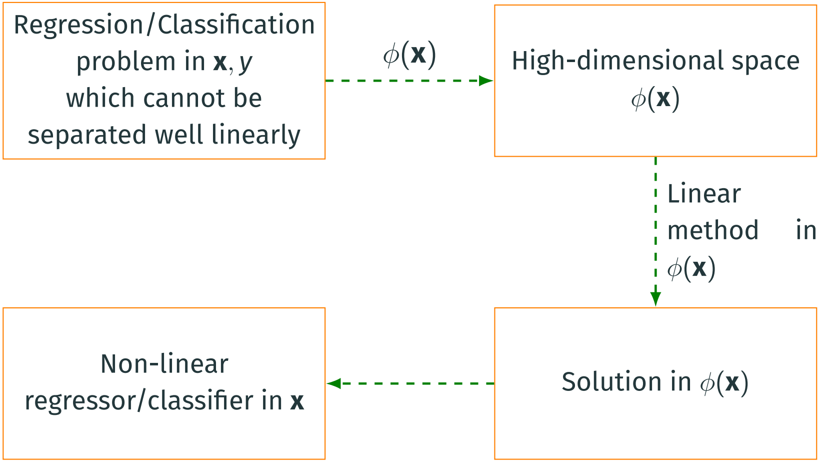
$$\begin{tikzcd} \boxed{\begin{array}{c}\text{Regression/Classification}\\ \text{problem in } \mathbf{x}, y\\ \text{which cannot be}\\ \text{separated well linearly}\end{array}} \arrow[r, dashed, "\phi(\mathbf{x})"] & \boxed{\begin{array}{c}\text{High-dimensional space}\\ \phi(\mathbf{x})\end{array}} \end{tikzcd}$$

The Kernel Trick: Summary

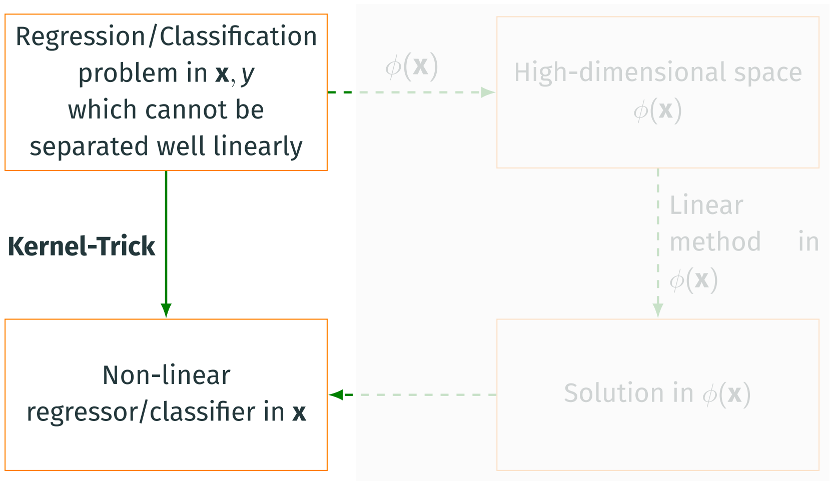


Regression/Classification problem in $\mathbf{x}, y$ which cannot be separated well linearly
$\phi(\mathbf{x})$
High-dimensional space $\phi(\mathbf{x})$
Linear method in $\phi(\mathbf{x})$
Solution in $\phi(\mathbf{x})$

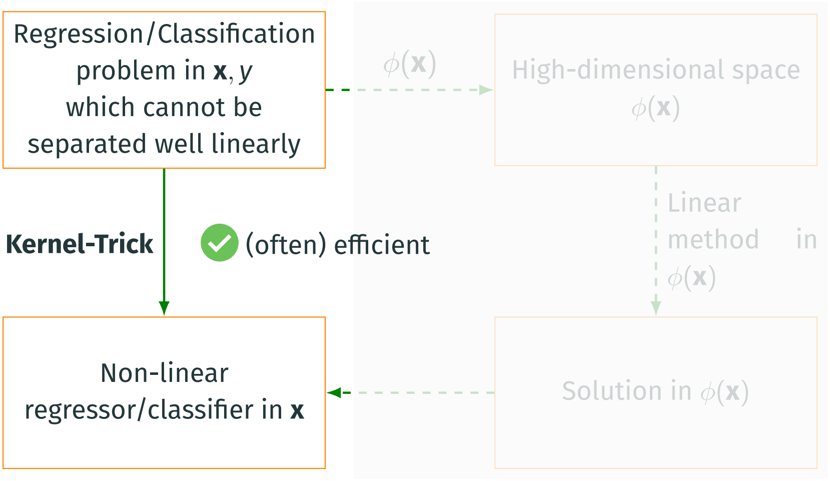
The Kernel Trick: Summary


Regression/Classification problem in $\mathbf{x}, y$ which cannot be separated well linearly
$\phi(\mathbf{x})$
High-dimensional space $\phi(\mathbf{x})$
Linear method in $\phi(\mathbf{x})$
Solution in $\phi(\mathbf{x})$
Non-linear regressor/classifier in $\mathbf{x}$

The Kernel Trick: Summary


Regression/Classification problem in $\mathbf{x}, y$ which cannot be separated well linearly
$\phi(\mathbf{x})$
High-dimensional space $\phi(\mathbf{x})$
Kernel-Trick
Linear method in $\phi(\mathbf{x})$
Non-linear regressor/classifier in $\mathbf{x}$
Solution in $\phi(\mathbf{x})$

The Kernel Trick: Summary


Regression/Classification problem in $\mathbf{x}, y$ which cannot be separated well linearly
$\phi(\mathbf{x})$
High-dimensional space $\phi(\mathbf{x})$
Linear method in $\phi(\mathbf{x})$
Solution in $\phi(\mathbf{x})$
Kernel-Trick
✓ (often) efficient
Non-linear regressor/classifier in $\mathbf{x}$

Questions

- What are valid kernels?
- How can we select a good kernel for our problem?
- Can we use kernels beyond the perceptron?
- Kernels work in very high-dimensional spaces. Doesn’t this lead to overfitting?

Properties of kernel functions

- Data space $\mathcal{X}$
- A kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- ? Can we use any function?

Properties of kernel functions

- Data space $\mathcal{X}$
- A kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- ❓ Can we use any function?

- k must be an inner product in a suitable space

Properties of kernel functions

- Data space $\mathcal{X}$
- A kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- ? Can we use any function?

- k must be an inner product in a suitable space
  $\Rightarrow$ k must be symmetric!

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} : k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \phi(\mathbf{x}')^T \phi(\mathbf{x}) = k(\mathbf{x}', \mathbf{x})$$

Properties of kernel functions

- Data space $\mathcal{X}$
- A kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- ? Can we use any function?

- k must be an inner product in a suitable space
  $\Rightarrow$ k must be symmetric!

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} \colon k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \phi(\mathbf{x}')^T \phi(\mathbf{x}) = k(\mathbf{x}', \mathbf{x})$$

$\Rightarrow$ Are there any other properties that it must satisfy?

Positive semi-definite matrices

Symmetric matrix $M \in \mathbb{R}^{n \times n}$ is positive semidefinite iff any of the following two conditions holds

1. $\forall \mathbf{x} \in \mathbb{R}^n : \mathbf{x}^T M \mathbf{x} \geq 0$
2. all eigenvalues of M are ≥ 0

Kernels $\Rightarrow$ Semi-definite matrices

- Data space $\mathcal{X}$ (possibly finite)
- Kernel $k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with feature map $\phi \colon \mathcal{X} \to \mathbb{R}^d$
- Take any finite subset of data $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq \mathcal{X}$
- Then the kernel (gram) matrix

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} = \begin{pmatrix} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_n) \\ \vdots & & \vdots \\ \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_n) \end{pmatrix}$$

is positive semidefinite

Kernels $\Rightarrow$ Semi-definite matrices

- Data space $\mathcal{X}$ (possibly finite)
- Kernel $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with feature map $\phi\colon \mathcal{X} \to \mathbb{R}^d$
- Take any finite subset of data $S = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subseteq \mathcal{X}$
- Then the kernel (gram) matrix

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} = \begin{pmatrix} \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_n) \\ \vdots & & \vdots \\ \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_n) \end{pmatrix}$$

is positive semidefinite

Proof.
$\mathbf{K} = \Phi^T\Phi$, where $\Phi = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \dots, \phi(\mathbf{x}_n)]$ ($d \times n$ matrix)
$\mathbf{x}^T\mathbf{K}\mathbf{x} = \mathbf{x}^T(\Phi^T\Phi)\mathbf{x} = (\mathbf{x}^T\Phi^T)(\Phi\mathbf{x}) = (\Phi\mathbf{x})^T(\Phi\mathbf{x}) = \|\Phi\mathbf{x}\|_2^2 \geq 0 \qquad \square$

Semi-definite matrices ⇒ Kernels

- Suppose the data space $\mathcal{X} = \{1, \dots, n\}$ is finite, and we are given a positive semidefinite matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$
- Then we can always construct a feature map

$$\phi \colon \mathcal{X} \to \mathbb{R}^n$$

such that $\mathbf{K}_{i,j} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$

Proof.
Since $\mathbf{K}$ is psd, its eigendecomposition $\mathbf{K} = UDU^T$ exists with $D = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$ and $\lambda_i \geq 0$. Hence, $D = D^{1/2}(D^{1/2})^T$ with $D^{1/2} = \mathrm{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_n})$ for all $1 \leq i \leq n$.

$$\mathbf{K} = UDU^T = U(D^{1/2}(D^{1/2})^T)U^T = \underbrace{(UD^{1/2})}_{=:\Phi^T} \underbrace{((D^{1/2})^T U^T)}_{=:\Phi} = \Phi^T \Phi,$$

where $\Phi = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \dots, \phi(\mathbf{x}_n)]$. $\square$

Outlook: Mercer’s Theorem

Mercer’s Theorem

Let $\mathcal{X}$ be a compact subset of $\mathbb{R}^n$ and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a kernel function.
Then one can expand k in a uniformaly convergent series of bounded functions ϕ_i s.t.

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$$

Outlook: Mercer's Theorem

Mercer's Theorem

Let $\mathcal{X}$ be a compact subset of $\mathbb{R}^n$ and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a kernel function.
Then one can expand k in a uniformaly convergent series of bounded functions ϕ_i s.t.

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$$

Can be generalized even further

Definition: kernel functions

- Data space $\mathcal{X}$
- A kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfying:

Definition: kernel functions

- Data space $\mathcal{X}$
- A kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfying:
    1. Symmetry: For any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ it must hold that $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$

Definition: kernel functions

- Data space $\mathcal{X}$
- A kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfying:
  1. Symmetry: For any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ it must hold that $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
  2. Positive semi-definiteness: For any n, any set $S = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subseteq \mathcal{X}$, the kernel (Gram) matrix

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$$

     must be positive semi-definite

Examples of kernels on $\mathbb{R}^d$

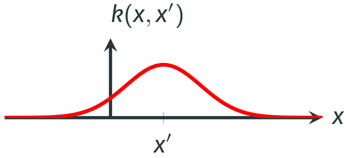
- Linear kernel: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$

Examples of kernels on $\mathbb{R}^d$

- Linear kernel: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
- Polynomial kernel: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^d$

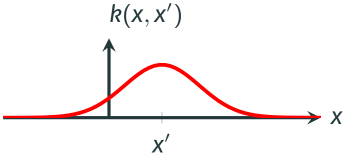
Examples of kernels on $\mathbb{R}^d$

- Linear kernel: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$
- Polynomial kernel: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + 1)^d$
- Gaussian (RBF, squared exp. kernel): $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|_2^2/h^2)$

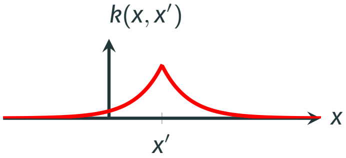


Examples of kernels on $\mathbb{R}^d$

- Linear kernel: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$
- Polynomial kernel: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + 1)^d$
- Gaussian (RBF, squared exp. kernel): $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|_2^2/h^2)$



- Laplacian kernel: $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|_1/h)$



Examples of (non)-kernels

- $k(x, x') = \sin(x) cos(x')$

Examples of (non)-kernels

- $k(x, x') = \sin(x) cos(x')$
  Not symmetric: Consider $x = 0, x' = \pi/2$:

  $$k(x, x') = 0 \cdot 0 = 0 \neq 1 = 1 \cdot 1 = k(x', x)$$

  Hence, k is not a valid kernel.

Examples of (non)-kernels

- $k(x, x') = \sin(x) cos(x')$
  Not symmetric: Consider $x = 0, x' = \pi/2$:

  $$k(x, x') = 0 \cdot 0 = 0 \neq 1 = 1 \cdot 1 = k(x', x)$$

  Hence, k is not a valid kernel.
- $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{M} \mathbf{x}'$

Examples of (non)-kernels

- $k(x, x') = \sin(x) cos(x')$
  Not symmetric: Consider $x = 0, x' = \pi/2$:
  $$k(x, x') = 0 \cdot 0 = 0 \neq 1 = 1 \cdot 1 = k(x', x)$$
  Hence, k is not a valid kernel.
- $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{M} \mathbf{x}'$
  If $\mathbf{M}$ is symmetric:
  $$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{M} \mathbf{x}' = \mathbf{x}^T \mathbf{M}^T \mathbf{x}' = (\mathbf{x}^T \mathbf{M}^T \mathbf{x}')^T = \mathbf{x}'^T M \mathbf{x} = k(\mathbf{x}', \mathbf{x})$$
  If $\mathbf{M}$ is not symmetric, then k not guaranteed to be symmetric.
  Claim: k valid kernels $\Leftrightarrow$ $\mathbf{M}$ psd

Effect of kernel on function class

- Given kernel k, predictors (for kernelized classification) have the form

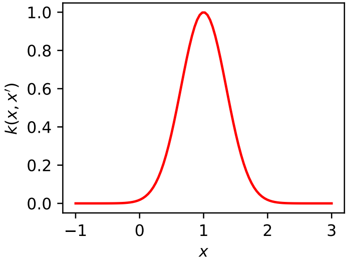
$$\hat{y} = \operatorname{sign}\left(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x})\right)$$

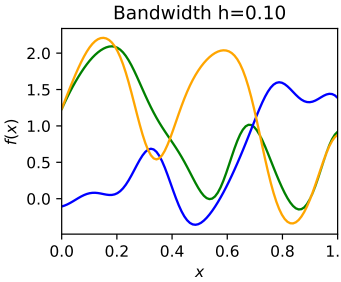
[kernels/decision-boundary.ipynb]

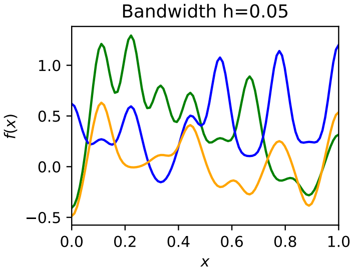
Example: Gaussian kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x}-\mathbf{x}'\|_2^2/h^2\right)$$

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})$$





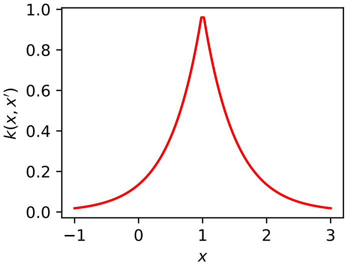


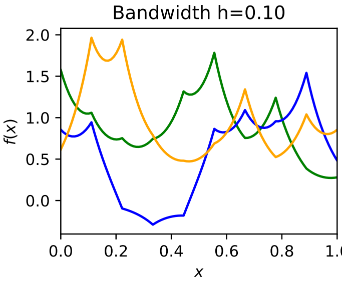
[kernels/kernels.py]

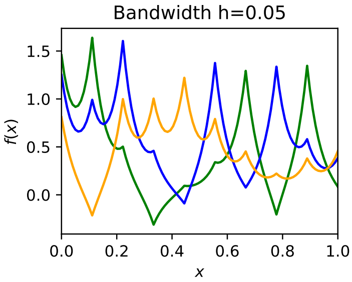
Example: Laplace/Exponential kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\big(- \|\mathbf{x} - \mathbf{x}'\|_1 / h \big)$$

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})$$







[kernels/kernels.py]

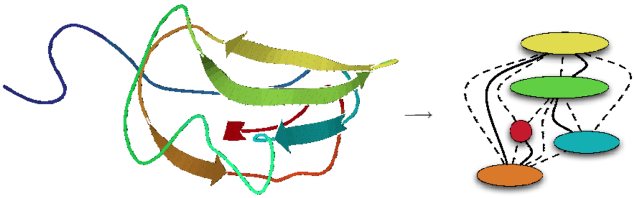
Demo: Effect on decision boundary

[kernels/decision-boundary2.ipynb]

Kernels beyond $\mathbb{R}^d$

- Can define kernels on a variety of objects:
    - Sequence kernels
    - Graph kernels
    - Diffusion kernels
    - Kernels on probability distributions
    - ...

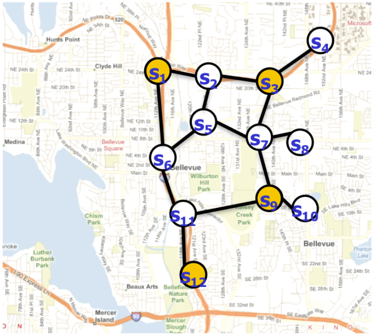
Example: Graph kernels



[Borgwardt et al.]

- Can define a kernel for measuring similarity between graphs by comparing random walks on both graphs (not further defined here)

Example: Diffusion kernels on graphs



$$\mathbf{K} = \exp(-\beta \mathbf{L})$$

- Can measure similarity among nodes in a graph via diffusion kernels (not defined here)

Kernel engineering (composition rules)

- Suppose we have two kernels:

$$k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \qquad k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

  defined on data space $\mathcal{X}$

Kernel engineering (composition rules)

- Suppose we have two kernels:

$$k_1 \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R} \qquad k_2 \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

  defined on data space $\mathcal{X}$

- Then the following functions are valid kernels:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\ k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\ k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \text{ for } c > 0 \\ k(\mathbf{x}, \mathbf{x}') &= f(k_1(\mathbf{x}, \mathbf{x}')), \end{aligned}$$

  where f is a polynomial with positive coefficients or the exponential function

Example: ANOVA kernel

$$\textbf{General:} \quad k_P(\mathbf{x}, \mathbf{x}') = \sum_{1 \le j_1 < \dots < j_P \le d} \prod_{p=1}^{P} k_{j_p}(x_{j_p}, x'_{j_p}) \quad \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$$

$$\textbf{P=1:} \quad \underbrace{k(\mathbf{x}, \mathbf{x}')}_{k:\ \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}} = \sum_{j=1}^{d} \underbrace{k_j(x_j, x'_j)}_{k_j:\ \mathbb{R} \times \mathbb{R} \to \mathbb{R}} \quad \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$$

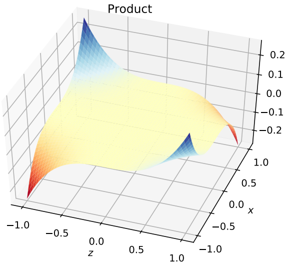
The function k is a valid kernel according to composition rules. Which functions f are learned?

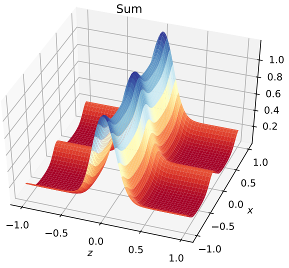
$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}^{(i)}, \mathbf{x})$$

$$= \sum_{i=1}^{n} \alpha_i y_i \sum_{j=1}^{d} k_j(x_j^{(i)}, x_j) = \sum_{j=1}^{d} \underbrace{\sum_{i=1}^{n} \alpha_i y_i k_j(x_j^{(i)}, x_j)}_{f_j(x_j)}$$

Example: Modeling pairwise data

- May want to use kernels to model pairwise data (users $\times$ products; genes $\times$ patients; ...)





$$k((x, z), (x', z')) = k_{\text{user}}(x, x') \cdot k_{\text{product}}(z, z')$$
$$k((x, z), (x', z')) = k_{\text{user}}(x, x') + k_{\text{product}}(z, z')$$

Where are we?

- We’ve seen how to kernelize the perceptron
- Discussed properties of kernels, and seen examples

Where are we?

- We’ve seen how to kernelize the perceptron
- Discussed properties of kernels, and seen examples
- Next questions:
  - What kind of predictors / decision boundaries do kernel methods entail?
  - Can we use the kernel trick beyond the perceptron?

Kernels as similarity functions

- Recall Perceptron (and SVM) classification rule:

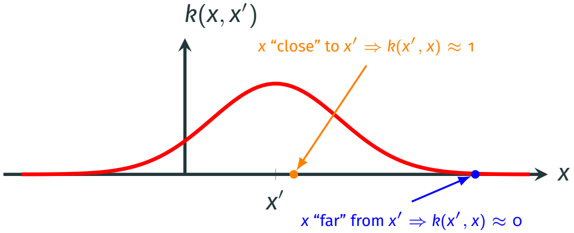
$$y = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})\right)$$

Kernels as similarity functions

- Recall Perceptron (and SVM) classification rule:

$$y = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})\right)$$

- Consider Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/h^2)$

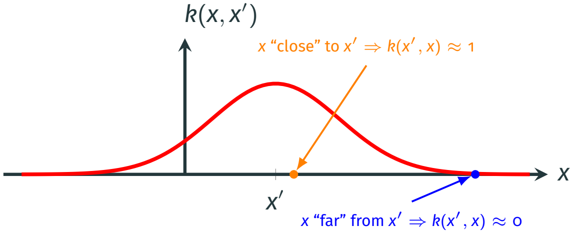


Kernels as similarity functions

- Recall Perceptron (and SVM) classification rule:

$$y = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})\right) \approx \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i [\mathbf{x}_i \text{ “close” to } \mathbf{x}]\right)$$

- Consider Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/h^2)$



$k(x, x')$
x “close” to $x' \Rightarrow k(x', x) \approx 1$
x
x'
x “far” from $x' \Rightarrow k(x', x) \approx 0$

Side note: Nearest-neighbor classifiers

- For data point **x**, predict majority of labels of *k* nearest neighbors:

$$y = \text{sign}\left(\sum_{i=1}^{n} y_i [\mathbf{x}_i \text{ among } k \text{ nearest neighbors } \mathbf{x}]\right)$$

Demo: k-NN

[kernels/knn.ipynb]

Nearest-neighbor classifiers

- For data point **x**, predict majority of labels of *k* nearest neighbors:

$$y = \text{sign}\left(\sum_{i=1}^{n} y_i [\mathbf{x}_i \text{ among } k \text{ nearest neighbors } \mathbf{x}]\right)$$

? How to choose *k*?

Nearest-neighbor classifiers

- For data point $\mathbf{x}$, predict majority of labels of k nearest neighbors:

$$y = \operatorname{sign}\left(\sum_{i=1}^{n} y_i [\mathbf{x}_i \text{ among } k \text{ nearest neighbors } \mathbf{x}]\right)$$

? How to choose k? 💡 Cross-validation!

K-NN vs. Kernel Perceptron

- *k*-nearest neighbor:

$$y = \text{sign}\left(\sum_{i=1}^{n} y_i [\mathbf{x}_i \text{ among } k \text{ nearest neighbors } \mathbf{x}]\right)$$

- Kernel perceptron:

$$y = \text{sign}\left(\sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x})\right)$$

Comparison: k-NN vs Kernelized Perceptron

Method	*k-NN*	*Kernelized Perceptron*
Advantages	No training necessary	Optimized weights can lead to improved performance Can capture "global trends" with suitable kernels Depends on "wrongly classified" examples only
Disadvantages	Depends on all data $\Rightarrow$ inefficient	Training requires optimization

Parametric vs nonparametric learning

- Parametric models have finite set of parameters
- Example: Linear regression, linear Perceptron, . . .

Parametric vs nonparametric learning

- Parametric models have finite set of parameters
- Example: Linear regression, linear Perceptron, . . .
- Nonparametric models grow in complexity with the size of the data
    - Potentially much more expressive
    - But also more computationally complex. Why?
- Example: Kernelized Perceptron, k-NN, …

Parametric vs nonparametric learning

- Parametric models have finite set of parameters
- Example: Linear regression, linear Perceptron, . . .
- Nonparametric models grow in complexity with the size of the data
    - Potentially much more expressive
    - But also more computationally complex. Why?
- Example: Kernelized Perceptron, k-NN, ...
- 💡 Kernels provide a principled way of deriving nonparametric models from parametric ones

Where are we?

- We’ve seen how to kernelize the perceptron
- Discussed properties of kernels, and seen examples

Where are we?

- We’ve seen how to kernelize the perceptron
- Discussed properties of kernels, and seen examples
- Next questions:
    - Can we use the kernel trick beyond the perceptron?

Kernelized SVM

- The support vector machine

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda \|\mathbf{w}\|_2^2$$

  can also be kernelized

How to kernelize the objective?

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda\|\mathbf{w}\|_2^2$$

$$\begin{aligned}
\max\left\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right\} &= \max\left\{0, 1 - y_i \left(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j\right)^T \mathbf{x}_i\right\} \\
&= \max\left\{0, 1 - y_i \sum_{j=1}^{n} \alpha_j y_j \underbrace{\mathbf{x}_j^T \mathbf{x}_i}_{=k(\mathbf{x}_j, \mathbf{x}_i)}\right\} \\
&= \max\left\{0, 1 - y_i \boldsymbol{\alpha}^T \boldsymbol{k}^{(i)}\right\},
\end{aligned}$$

where $\boldsymbol{\alpha}^T = (\alpha_1, \ldots, \alpha_n)$, $\boldsymbol{k}^{(i)} = (y_1 k(\mathbf{x}_1, \mathbf{x}_i), \ldots, y_n k(\mathbf{x}_n, \mathbf{x}_i))^T$.

How to kernelize the regularizer?

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda \|\mathbf{w}\|_2^2$$

$$\lambda \|\mathbf{w}\|_2^2 = \lambda \mathbf{w}^T \mathbf{w} = \lambda \left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \right)^T \left(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j \right)$$

$$= \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \underbrace{\mathbf{x}_i^T \mathbf{x}_j}_{=k(\mathbf{x}_i, \mathbf{x}_j)} = \lambda \boldsymbol{\alpha}^T \mathbf{D}_y \mathbf{K} \mathbf{D}_y \boldsymbol{\alpha},$$

where $\mathbf{D}_y = \begin{pmatrix} y_1 & & \\ & \ddots & \\ & & y_n \end{pmatrix}$ and $\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_1, \mathbf{x}_n) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$

Learning & prediction with kernel classifier

Learning

Solve the following problem.

- Perceptron: $\text{argmin}_\alpha \frac{1}{n}\sum_{i=1}^n \max\{0, -y_i\alpha^T\mathbf{k}_i\}$
- SVM: $\text{argmin}_\alpha \frac{1}{n}\sum_{i=1}^n \max\{0, 1 - y_i\alpha^T\mathbf{k}_i\} + \lambda\alpha^T\mathbf{D_y K D_y}\alpha$

$$\mathbf{k}_i = [y_1 k(\mathbf{x}_i, \mathbf{x}_1), \ldots, y_n k(\mathbf{x}_i, \mathbf{x}_n)]$$

Learning & prediction with kernel classifier

Learning

Solve the following problem.

- Perceptron: $\text{argmin}_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -y_i \alpha^T \mathbf{k}_i\}$
- SVM: $\text{argmin}_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \alpha^T \mathbf{k}_i\} + \lambda \alpha^T \mathbf{D_y K D_y} \alpha$

$$\mathbf{k}_i = [y_1 k(\mathbf{x}_i, \mathbf{x}_1), \ldots, y_n k(\mathbf{x}_i, \mathbf{x}_n)]$$

Prediction

For data point $\mathbf{x}$ predict label y as

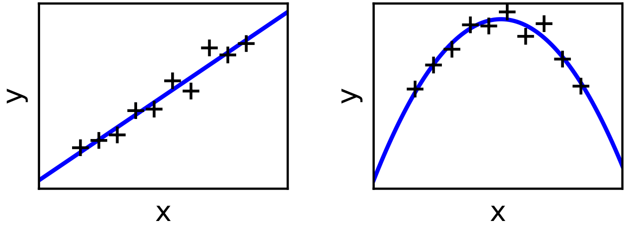
$$\hat{y} = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})\right)$$

Demo: Kernelized SVM

[kernels/kernelized-svm.ipynb]

Kernelized Linear Regression

- From linear to nonlinear regression:



- Can also kernelize linear regression
- Predictor has the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$$

Example: Kernelized linear regression

- Original (parametric) linear optimization problem:

$$\hat{w} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \|\mathbf{w}\|_2^2$$

- Similar as in perceptron, optimal $\hat{\mathbf{w}}$ lies in span of data:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$$

Kernelizing linear regression

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \|\mathbf{w}\|_2^2$$

$$\mathbf{w}^T \mathbf{x}_i - y_i = \sum_{j=1}^{n} \alpha_j \mathbf{x}_j^T \mathbf{x}_i - y_i = \sum_{j=1}^{n} \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) - y_i = \boldsymbol{\alpha}^T \mathbf{k}_i - y_i$$

$$\lambda \|\mathbf{w}\|_2^2 = \lambda \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha},$$

where $\mathbf{k}_i = (k(\mathbf{x}_1, \mathbf{x}_i), \ldots, k(\mathbf{x}_n, \mathbf{x}_i))^T$ and $\mathbf{K} = [\mathbf{k}_1 | \cdots | \mathbf{k}_n]$.

$$\hat{\alpha} = \operatorname*{argmin}_{\boldsymbol{\alpha}} \frac{1}{n} \underbrace{\sum_{i=1}^{n} (\boldsymbol{\alpha}^T \mathbf{k}_i - y_i)^2}_{= \|\boldsymbol{\alpha}^T \mathbf{K} - \mathbf{y}\|_2^2 \text{ with } \mathbf{y} = (y_1, \ldots, y_n)^T} + \lambda \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha}$$

Learning & Predicting with KLR

- Learning: Solve least squares problem

$$\hat{\alpha} = \operatorname*{argmin}_{\boldsymbol{\alpha}} \frac{1}{n}\|\alpha^T\mathbf{K} - \mathbf{y}\|_2^2 + \lambda\alpha^T\mathbf{K}\alpha$$

Learning & Predicting with KLR

- Learning: Solve least squares problem

$$\hat{\alpha} = \operatorname*{argmin}_{\boldsymbol{\alpha}} \frac{1}{n}\|\alpha^T \mathbf{K} - \mathbf{y}\|_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

  Closed-form solution: $\hat{\alpha} = (\mathbf{K} + n\lambda \mathbf{I})^{-1}\mathbf{y}$

- Prediction: For data point $\mathbf{x}$ predict response y as

$$\hat{y} = \sum_{i=1}^{n} \hat{\alpha}_i k(\mathbf{x}_i, \mathbf{x})$$

Demo: Kernelized linear regression

[kernels/kernelized-linear-regression.ipynb]

KLR for the linear kernel

- What if $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$?

KLR for the linear kernel

- What if $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$?

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) = \left(\sum_{i=1}^{n} \alpha_i \mathbf{x}_i^T\right)\mathbf{x} = \mathbf{w}^T\mathbf{x}$$

Application: Semi-parametric regression

- Often, parametric models are too “rigid”, and nonparametric models fail to extrapolate
- Solution: Use additive combination of linear and nonlinear kernel function:

$$k(\mathbf{x}, \mathbf{x}') = c_1 \exp(-\|\mathbf{x} - \mathbf{x}'\|_2^2 / h^2) + c_2 \mathbf{x}^T \mathbf{x}'$$

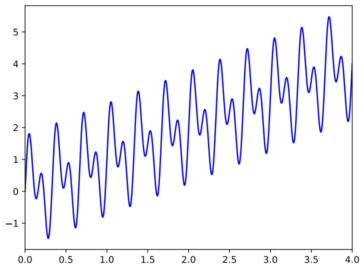
$$\begin{aligned}
f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) &= \sum_{i=1}^{n} \alpha_i \left(c_1 \exp(-\|\mathbf{x}_i - \mathbf{x}\|_2^2 / h^2) + c_2 \mathbf{x}_i^T \mathbf{x} \right) \\
&= c_1 \sum_{i=1}^{n} \alpha_i \exp(-\|\mathbf{x}_i - \mathbf{x}\|_2^2 / h^2) + c_2 \sum_{i=1}^{n} \alpha_i \mathbf{x}_i^T \mathbf{x} \\
&= c_1 f_1(\mathbf{x}) + c_2 f_2(\mathbf{x})
\end{aligned}$$

Demo: Semi-parametric KLR

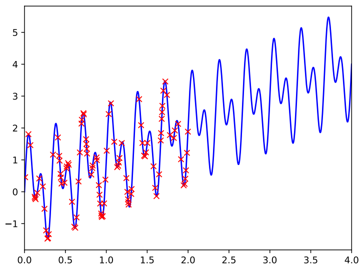
[semiparametric-demo.ipynb]

Example

Function

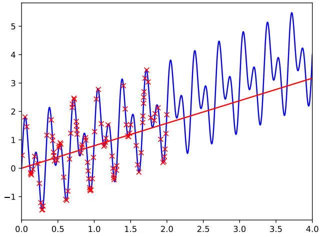


Training data

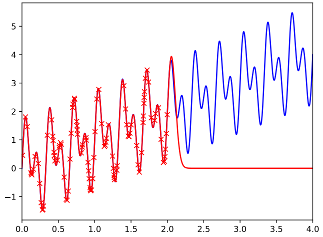


Example fits

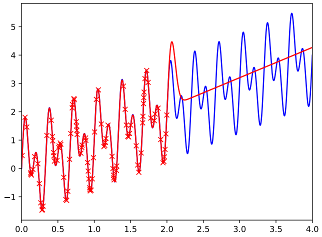
Linear kernel



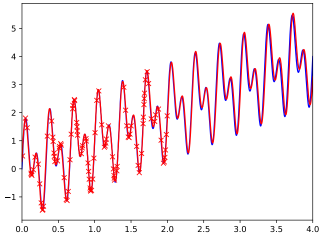
Gaussian kernel



Gaussian + linear kernel



Periodic + linear kernel



Choosing kernels

- For a given kernel, how should we choose parameters?

Choosing kernels

- For a given kernel, how should we choose parameters?
  - 💡 Cross-validation

Choosing kernels

- For a given kernel, how should we choose parameters?
    - 💡 Cross-validation
- How should we select suitable kernels?
    - Domain knowledge (dependent on data type)
    - “Brute force” (or heuristic) search
    - Use cross-validation

Choosing kernels

- For a given kernel, how should we choose parameters?
    - 💡 Cross-validation
- How should we select suitable kernels?
    - Domain knowledge (dependent on data type)
    - “Brute force” (or heuristic) search
    - Use cross-validation
- Learning kernels
    - Much research on automatically selecting good kernels (Multiple Kernel Learning; Hyperkernels; etc.)

What about overfitting?

- Kernels map to (very) high-dimensional spaces.
- Why do we hope to be able to learn?
- **First attempt of an answer**:
  (typically) # parameters $\ll$ # dimensions. Why?

What about overfitting?

- Kernels map to (very) high-dimensional spaces.
- Why do we hope to be able to learn?
- **First attempt of an answer**:
  (typically) # parameters $\ll$ # dimensions. Why?
- Number of parameters = number of data points
  ("non-parametric learning")

What about overfitting?

- Kernels map to (very) high-dimensional spaces.
- Why do we hope to be able to learn?
- **Second attempt of an answer**:

What about overfitting?

- Kernels map to (very) high-dimensional spaces.
- Why do we hope to be able to learn?
- **Second attempt of an answer**:
- Overfitting can of course happen (if we choose poor parameters)

What about overfitting?

- Kernels map to (very) high-dimensional spaces.
- Why do we hope to be able to learn?
- **Second attempt of an answer**:
- Overfitting can of course happen (if we choose poor parameters)
- Can combat overfitting by regularization:
  - This is already built into kernelized linear regression (and SVMs), but not the kernelized Perceptron

$$\text{KLR:} \quad \operatorname*{argmin}_{\alpha} \frac{1}{n} \|\alpha^T \mathbf{K} - \mathbf{y}\|_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

$$\text{SVM:} \quad \operatorname*{argmin}_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \alpha^T \mathbf{k}_i\} + \lambda \alpha^T \mathbf{D}_y \mathbf{K} \mathbf{D}_y \alpha$$

Before we conclude

- Our proof for why we can use the ansatz $\hat{\mathbf{w}} = \sum_i \alpha_i \phi(\mathbf{x}_i)$ was hand-wavy
- Principled proofs: Representer Theorems

Before we conclude

- Our proof for why we can use the ansatz $\hat{\mathbf{w}} = \sum_i \alpha_i \phi(\mathbf{x}_i)$ was hand-wavy
- Principled proofs: Representer Theorems

Representer Theorem

Let $\mathcal{X}$ be the data space and $\phi \colon \mathcal{X} \to \mathcal{H}$ a mapping from $\mathcal{X}$ to Hilbert space $\mathcal{H}$. Consider the optimization problem

$$\min_{\mathbf{w}} f(\langle \mathbf{w}, \phi(\mathbf{x}_1)\rangle, \ldots, \langle \mathbf{w}, \phi(\mathbf{x}_n)\rangle) + R(\|\mathbf{w}\|),$$

where f is an arbitrary function from $f \colon \mathbb{R}^n \to \mathbb{R}$ and $R \colon \mathbb{R}_{\geq 0} \to \mathbb{R}$ is a strictly monotonic function. Then, there exist $\alpha_1, \ldots, \alpha_n$ such that $\mathbf{w} = \sum_{i=1}^n \alpha_i \phi(\mathbf{x}_i)$ is an optimal solution for the above optimization problem.

Representer Theorem I

Many functions satisfy the above form:

- Ridge regression with non-linear functions:

$$f(\underbrace{\langle \mathbf{w}, \phi(\mathbf{x}_1)\rangle}_{\hat{y}_1}, \ldots, \underbrace{\langle \mathbf{w}, \phi(\mathbf{x}_n)\rangle}_{\hat{y}_n}) = \sum_{i=1}^{n} (\hat{y}_n - y_i)^2 \quad R(\|\mathbf{w}\|) = \|\mathbf{w}\|_2^2$$

Proof of the Representer Theorem:
Let $\mathbf{w}^*$ be an optimal solution of the problem. Since $\mathbf{w}^* \in \mathcal{H}$,

$$\mathbf{w}^* = \underbrace{\sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i)}_{\text{subspace of } \mathcal{H}} + \boldsymbol{u},$$

where $\boldsymbol{u}$ is orthogonal to the subspace, i.e., $\langle \boldsymbol{u}, \phi(\mathbf{x}_i)\rangle = 0$ for all $i \in \{1, \ldots, m\}$.

Representer Theorem II

We prove the theorem by showing that the same value of the objective function can be reached if $\boldsymbol{u} = \mathbf{0}$.

Let $\mathbf{w} = \mathbf{w}^* - \boldsymbol{u}$.

Regularizer:

$\|\mathbf{w}^*\|^2 = \|\mathbf{w}\|^2 + \|\boldsymbol{u}\|^2 \Rightarrow \|\mathbf{w}\| \leq \|\mathbf{w}^*\| \Rightarrow R(\|\mathbf{w}\|) \leq R(\|\mathbf{w}^*\|)$

***f*-term:**

$\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle = \langle \mathbf{w}^* - \boldsymbol{u}, \phi(\mathbf{x}_i) \rangle = \langle \mathbf{w}^*, \phi(\mathbf{x}_i) \rangle$, since $\langle u, \phi(\mathbf{x}_i) \rangle = 0$

$f(\langle \mathbf{w}, \phi(\mathbf{x}_1) \rangle, \dots, \langle \mathbf{w}, \phi(\mathbf{x}_n) \rangle) = f(\langle \mathbf{w}^*, \phi(\mathbf{x}_1) \rangle, \dots, \langle \mathbf{w}^*, \phi(\mathbf{x}_n) \rangle)$

Representer Theorem III

We have shown that the value of the objective function for **w** is at most the value for $\mathbf{w}^*$. Hence,

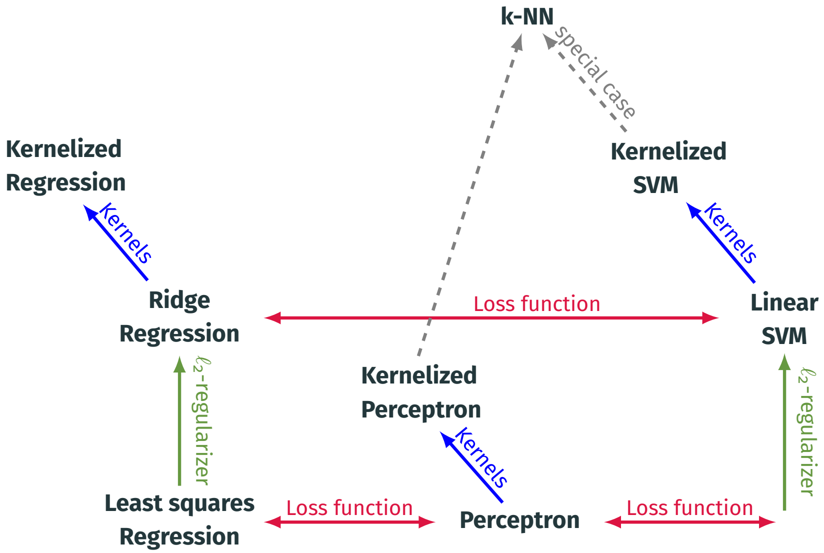
$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i)$$

also is an optimal solution, proving the theorem. $\square$

What you need to know

- Kernels are
    - (efficient, implicit) inner products
    - Positive (semi-)definite functions
    - Many examples (linear, polynomial, Gaussian/RBF, ...)
- The “Kernel trick”
    - Reformulate learning algorithm so that inner products appear
    - Replace inner products by kernels
- K-Nearest Neighbor classifier (and relation to Perceptron)
- How to choose kernels (kernel engineering etc.)
- Applications: Kernelized Perceptron / SVM; kernelized linear regression

Supervised learning big picture so far



Supervised learning summary so far

Representation/ features	Linear hypotheses, non-linear hypotheses through feature transformations, kernels
Model/ objective	Loss-function (squared loss, ℓ_p loss, 0/1 loss, Perceptron loss, Hinge loss) + Regularization (ℓ_2 norm)
Method	Exact solution, Gradient Descent, (mini-batch) SGD
Evaluation metric	Empirical risk = (mean) squared error, Accuracy
Model selection	k-fold cross-validation, Monte Carlo cross-validation

Further Reading / References

- S. Shalev-Schwartz & S. Ben-David, “Understanding Machine Learning: From Theory to Algorithms”, Chapter 16
- C. Bishop, “Pattern Recognition and Machine Learning”, Chapter 6 (6.1 & 6.2)