ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Machine Learning

Non-linear prediction; Kernels

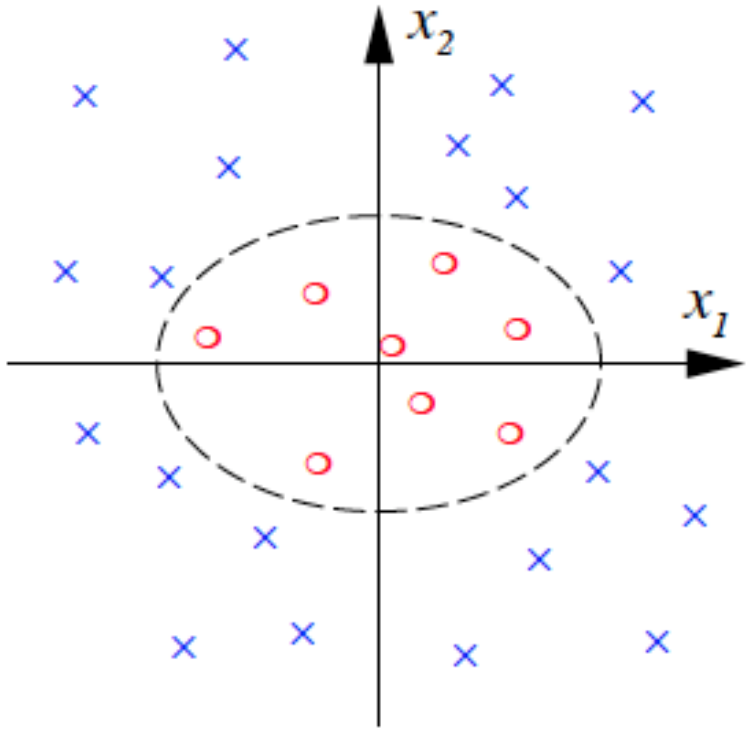
Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)

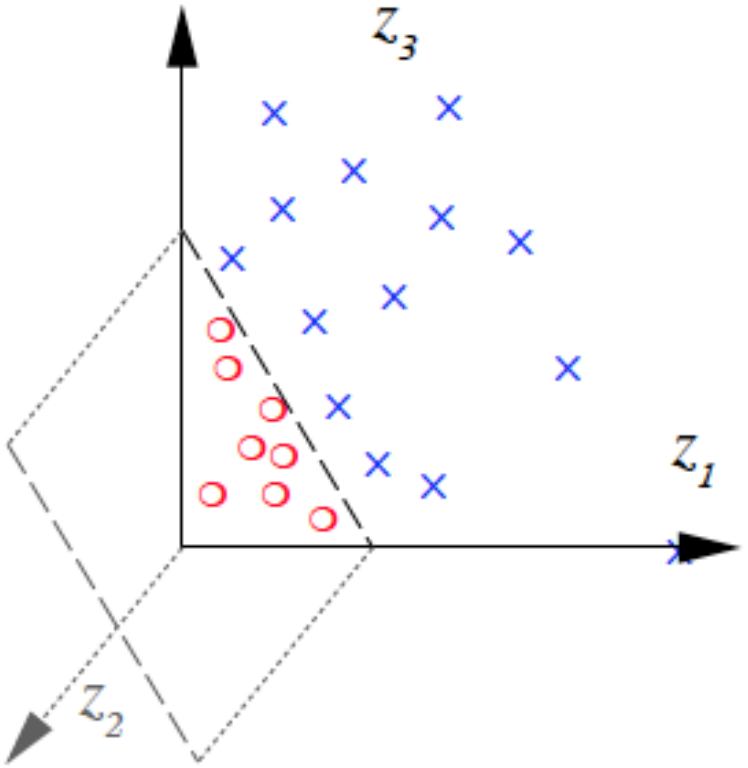
Announcement

- Project 1 out: Linear regression
- Grade based on prediction performance
- See tutorials for more information

Nonlinear classification via embedding

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$





Avoiding the feature explosion

- Need O(d^k) dimensions to represent (multivariate) polynomials of degree k on d features
- **Example**: d=10000, k=2 ➔ Need ~100M dimensions
- Key idea: Can avoid explicit mapping by working „implicitly“ (using inner products) only

Perceptron optimization

$$\min_{\alpha_{1:n}} \sum_{i=1}^{n} \max\{0, -\sum_{j} \alpha_j y_i \mathbf{x}_i^T \mathbf{x}_j\}$$

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

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

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*

1D: $(1 + xx')^3 = (1 + 3xx' + 3x^2x'^2 + x^3x'^3)$

$= [1, \sqrt{3}x, \sqrt{3}x^2, x^3]^T [1, \sqrt{3}x', \sqrt{3}x'^2, x'^3]$

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

The „Kernel Trick“

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

- Example: Perceptron

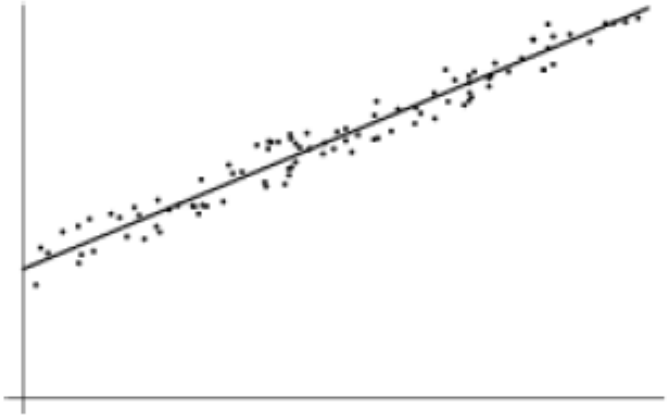
$$\min_{\alpha_{1:n}} \sum_{i=1}^{n} \max\{0, -\sum_j \alpha_j y_i \mathbf{x}_i^T \mathbf{x}_j\}$$

$$\min_{\alpha_{1:n}} \sum_{i=1}^{n} \max\{0, -\sum_j \alpha_j y_i k(\mathbf{x}_i, \mathbf{x}_j)\}$$

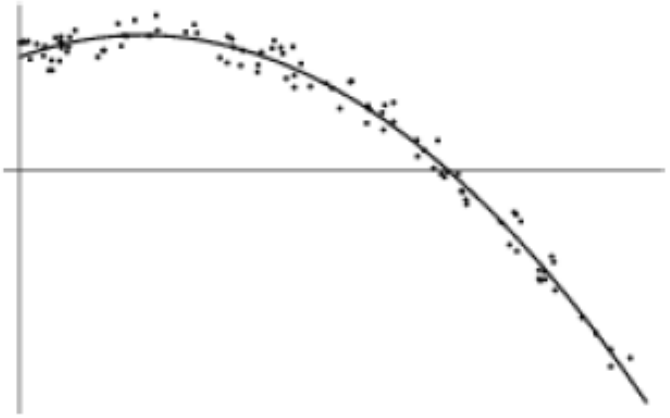
- This „trick“ is much more generally applicable

Preview: Kernelized Linear Regression

- From linear to nonlinear regression:



linear regression



nonlinear regression

- Will see how we can kernelize linear regression
- Predictor will have the form

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

nonlinear, e.g. $k(x,x') = (1 + x^T x')^m$

$\Rightarrow$ f degree m polynomial

Preview: Kernelized SVM

- The support vector machine

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda ||\mathbf{w}||_2^2$$

  can also be kernelized

- This requires convex duality (discussed in Advanced Machine Learning class).

Questions

- What are valid kernels?
- How can we select a good kernel for our problem?
- Kernels work in very high-dimensional spaces. Doesn‘t this lead to overfitting?

Properties of kernel functions

- Data space X , Eg. $X = \mathbb{R}^d$
- A kernel is a function $k : X \times X \to \mathbb{R}$
- Can we use any function?

- k must be an inner product in a suitable space

➔ k must be symmetric!

$$k(x, x') = \phi(x)^T \phi(x') = \phi(x')^T \phi(x) = k(x', x)$$

➔ Are there other properties that it must satisfy?

Positive semi-definite matrices

- A symmetric matrix M is called positive semidefinite iff

(1) $\forall x \in \mathbb{R}^n: \; x^T M x \geq 0$ $\qquad M \in \mathbb{R}^{n \times n}$

$\Leftrightarrow$ (2) All eigenvalues of M $\lambda_1 \dots \lambda_n \geq 0$

---

(2) $\Rightarrow$ (1) Any sym. M can be written

$$M = U^T D U, \qquad D = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}$$

U orthogonal

Define $D^{\frac{1}{2}} = \begin{pmatrix} \sqrt{\lambda_1} & & 0 \\ & \ddots & \\ 0 & & \sqrt{\lambda_n} \end{pmatrix}$

$$\forall x \in \mathbb{R}^n: \; x^T M x = \underbrace{x^T U^T D^{\frac{1}{2}T}}_{V^T} \overbrace{D^{\frac{1}{2}}}^{D} \underbrace{D^{\frac{1}{2}} U x}_{V} \geq 0$$

Kernels ➔ semi-definite matrices

- Data space X (possibly infinite)
- Kernel function $k : X \times X \to \mathbb{R}$
- Take any finite subset of data $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq X$
- Then the kernel (gram) matrix

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

is positive semidefinite

$$\Phi = \left(\phi(x_1) \mid \ldots \mid \phi(x_n) \right) \in \mathbb{R}^{D \times n}$$

$$K = \Phi^T \Phi \in \mathbb{R}^{n \times n}$$

$$\forall x \in \mathbb{R}^n : \quad x^T K x = \underbrace{x^T \Phi^T}_{v^T} \underbrace{\Phi x}_{v} \geq 0 \qquad \square$$

Semi-definite matrices ➔ kernels

- Suppose the data space $X=\{1,\ldots,n\}$ 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

$$\phi : X \to \mathbf{R}^n$$

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

$K = U^T D^{\frac{1}{2}T} D^{\frac{1}{2}} U$

$S^T \quad S$

$e_i = [0 \ldots 0 \underset{\uparrow}{1} 0 \ldots 0]$

i-th pos.

$K_{ij} = e_i^T K e_j = \underbrace{(e_i^T S^T)}_{\phi(i)^T} \underbrace{(S e_j)}_{j\text{-th column of } S}$

$=: \phi(j) \in \mathbb{R}^n$

□

Outlook: Mercer's Theorem

Theorem (Mercer 1909)

Let Ω be a compact subset of $\mathbb{R}^n$. Suppose K is a continuous symmetric function such that the integral operator
$T_K : L_2(X) \to L_2(X)$,

$$(T_K f)(\cdot) = \int_\Omega K(\cdot, \mathbf{x}) f(\mathbf{x}) d\mathbf{x},$$

is positive, that is

$$\int_{\Omega \times \Omega} K(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} > 0 \quad \forall f \in L_2(\Omega)$$

Then we can expand $K(\mathbf{x}, \mathbf{z})$ in a uniformly convergent series in terms of T_K's eigen-functions $\phi_j \in L_2(\Omega)$, with $||\phi_j||_{L_2} = 1$ and positive associated eigenvalues $\lambda_j > 0$.

Proof: theory of Fredholm integral equations.

Definition: kernel functions

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

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

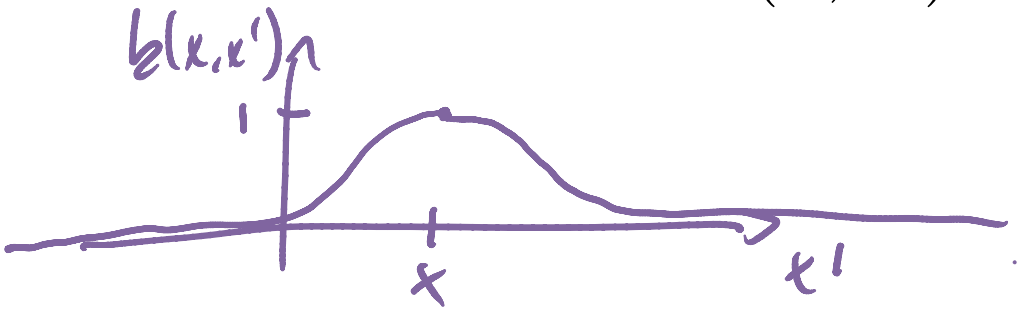
- Positive semi-definiteness: For any *n*, any set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq X$, the kernel (Gram) matrix

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

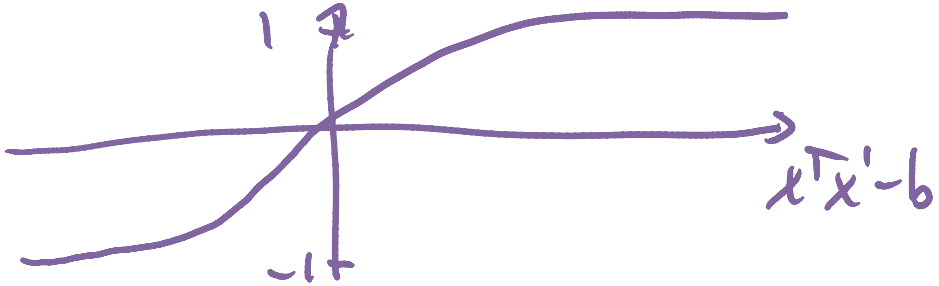
must be positive semi-definite

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

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



- Sigmoid (tanh) kernel: $k(\mathbf{x}, \mathbf{x}') = \tanh \kappa \mathbf{x}^T \mathbf{x}' - b$



Examples of (non)-kernels

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

$k(0, \frac{\pi}{2}) = 0 \neq k(\frac{\pi}{2}, 0) = 1$

$\Rightarrow$ not symmetric !!

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T M \mathbf{x}'$$

$x \in \mathbb{R}^d$
$M \in \mathbb{R}^{d \times d}$

if M not symm. $\Rightarrow$ k not symm.

Sps. $d = 1$, $M = -1$. Then $k(1, 1) = -1$

$\Rightarrow$ k not pos. semidefinite !!

Kernels as *similarity functions*

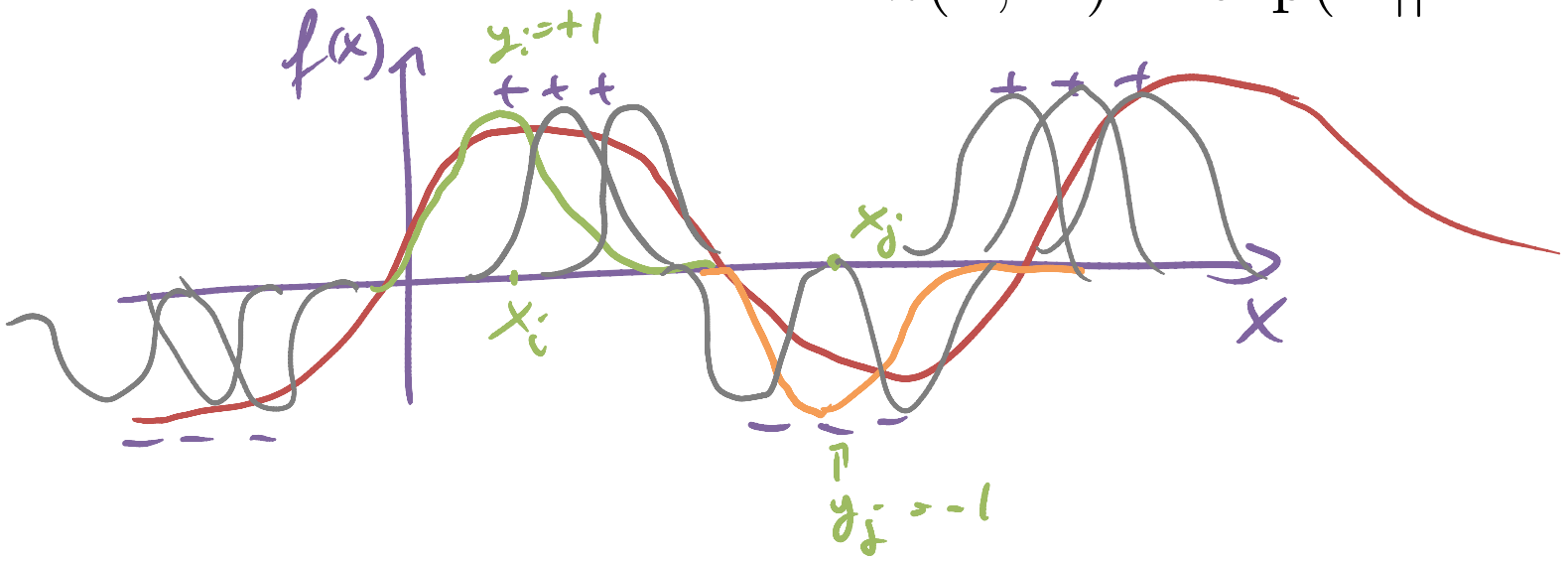
- Recall Perceptron (and SVM) classification rule:

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

obtained by training (e.g. SGD)

$f(x)$

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



Recall Kernelized Perceptron

- Initialize $\alpha_1 = \cdots = \alpha_n = 0$
- For t=1,2,...
  - Pick data point ($\boldsymbol{x}_i, y_i$) uniformly at random
  - Predict

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

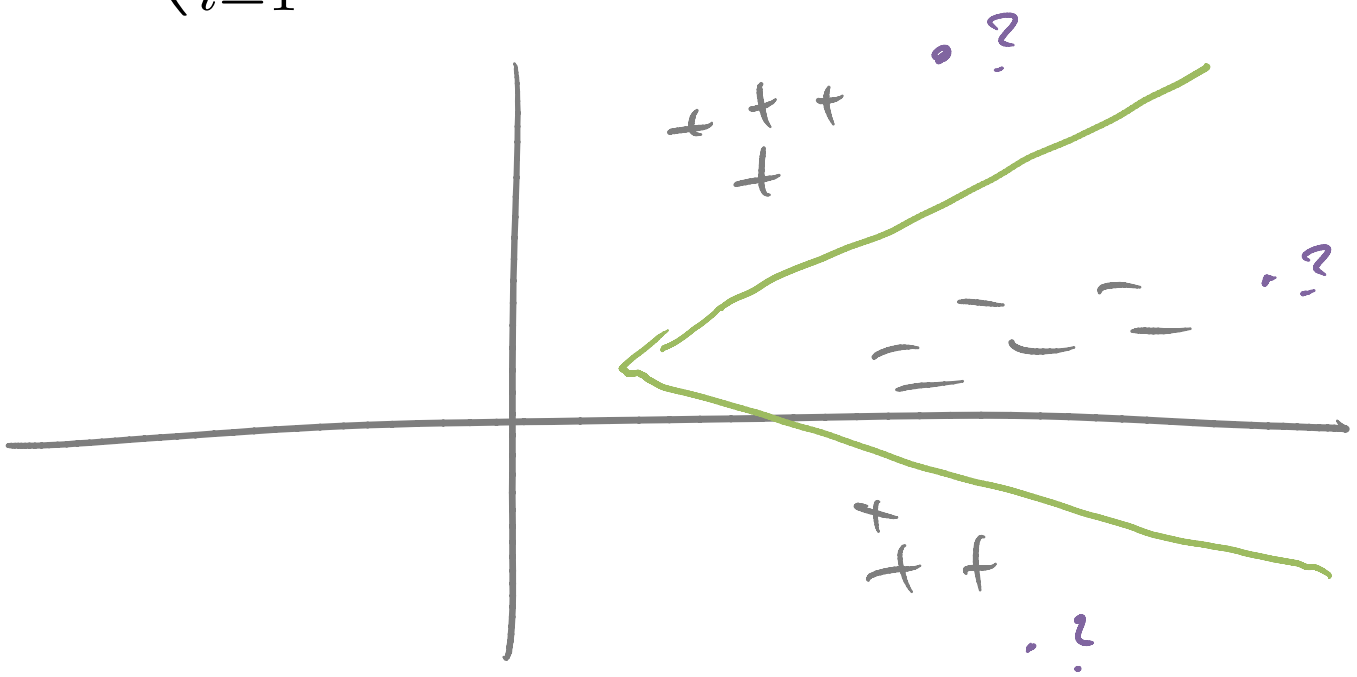
  - If $\hat{y} \neq y_i$ set $\alpha_i \leftarrow \alpha_i + \eta_t$

Matlab Demo: Kernelized Perceptron

Side note: Nearest-neighbor classifiers

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

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



Demo: k-NN

Nearest-neighbor classifiers

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

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

- How to choose *k*?
  - Cross-validation! ☺

K-NN vs. Kernel Perceptron

- k-Nearest Neighbor:

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

- Kernel Perceptron:

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

optimized

$\alpha_i \geq 0$

if x and x_i: "far"

k = Gaussian kernel

$= \exp\left(-\frac{dist^2}{h^2}\right)$

"bandwidth"

if x and x_i: "close"

Comparison: k-NN vs Kernelized Perceptron

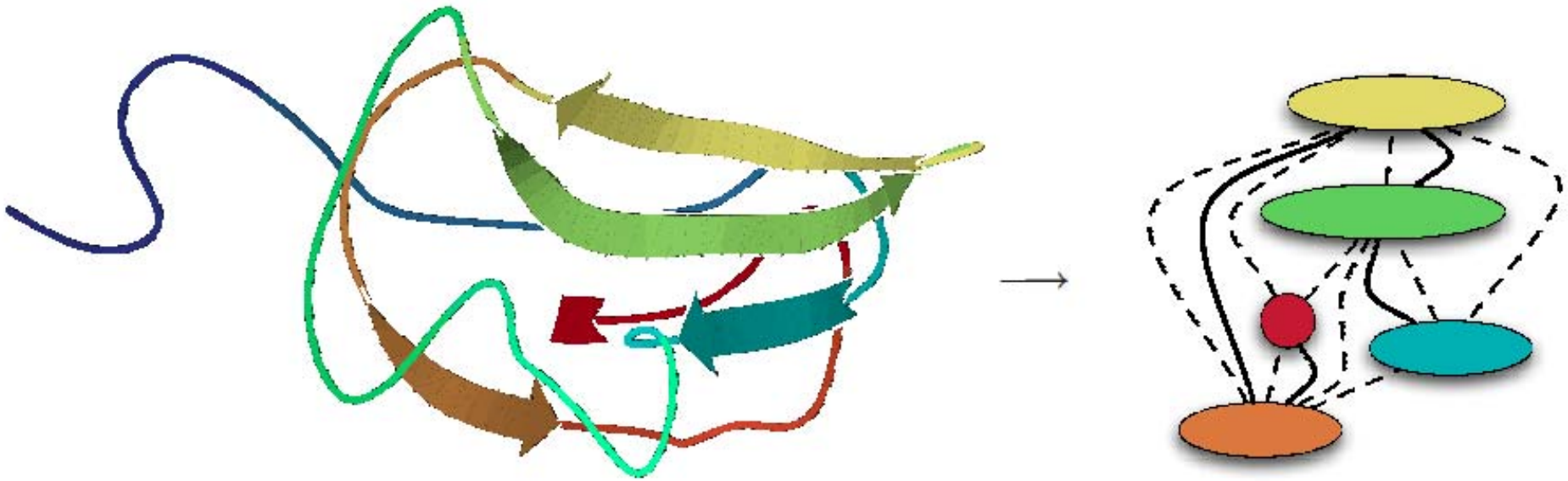
Method	*k-NN*	*Kernelized Perceptron*
Advantages	No training necessary	Optimized weights can lead to improved performance Can capture „global trends“ with suitable kernels Depends on „wrongly classified“ examples only
Disadvantages	Depends on all data ➔ inefficient	Training requires optimization

Kernels beyond $\mathbb{R}^d$

- Can define kernels on a variety of objects:

- Sequence kernels
- Graph kernels
- Diffusion kernels
- Kernels on probability distributions
- ...

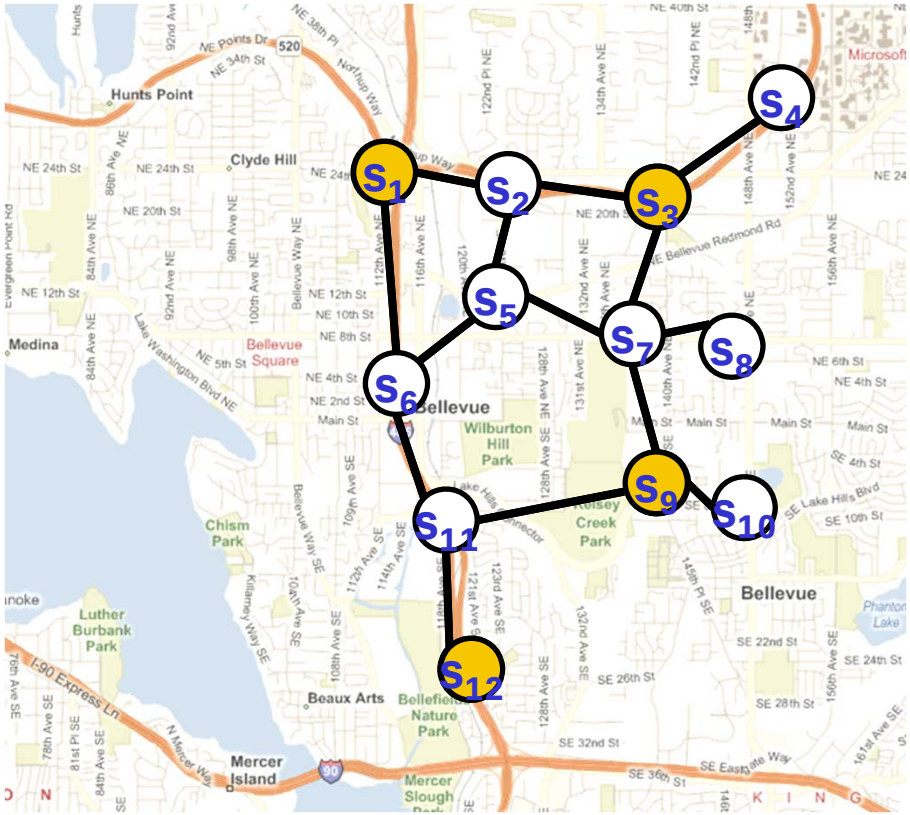
Example: Graph kernels



[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

Kernel composition rules

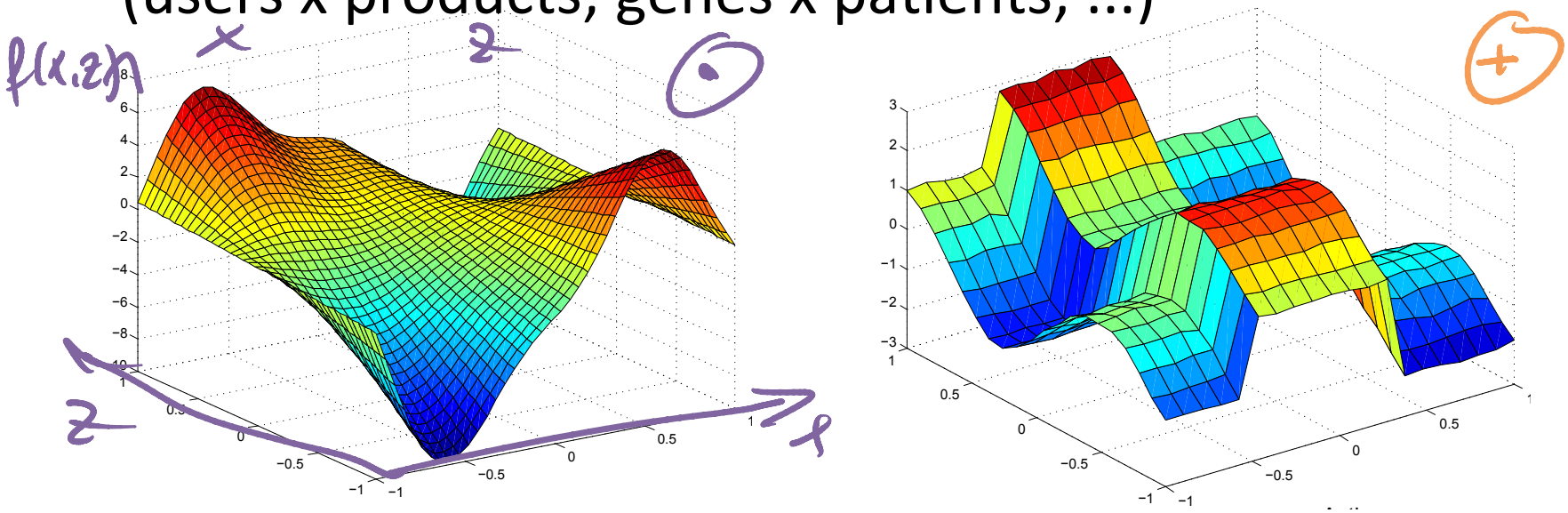
Let K_1, K_2 be kernels over $\Omega \times \Omega, \Omega \subseteq \mathbb{R}^d$, $a \in \mathbb{R}^+$, $f(.)$ a real-vealued function $\phi : \Omega \to \mathbb{R}^e$ with K_3 a kernel over $\mathbb{R}^e \times \mathbb{R}^e$.

Then the following functions are kernels:

1. $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z}) + K_2(\mathbf{x}, \mathbf{z})$
2. $K(\mathbf{x}, \mathbf{z}) = aK_1(\mathbf{x}, \mathbf{z})$
3. $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z})K_2(\mathbf{x}, \mathbf{z})$
4. $K(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$
5. $K(\mathbf{x}, \mathbf{z}) = K_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$
6. $K(\mathbf{x}, \mathbf{z}) = p(K_1(\mathbf{x}, \mathbf{z}))$, ($p(x)$ is a polynomial with positive coefficients)
7. $K(\mathbf{x}, \mathbf{z}) = \exp(K_1(\mathbf{x}, \mathbf{z}))$

Example: Modeling pairwise data

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



$$k((x,z),(x',z')) = \underbrace{k_x(x,x')}_{\text{e.g. linear kernel}} \cdot \underbrace{k_z(z,z')}_{\text{Gaussian}}$$

is valid kernel if k_x and k_z are kernels

Parametric vs nonparametric learning

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

$$f(x) = w^T x \,, \quad w \in \mathbb{R}^d \quad \text{indep. of } n \text{ (\#data pt.)}$$

- 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, ...

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

- Kernels provide a principled way of deriving non-parametric models from parametric ones

Example: Kernelized linear regression

- Original (parametric) linear optimization problem

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_i (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda ||\mathbf{w}||_2^2$$

$\hat{R}$

- Similar as in perceptron, optimal w^* lies in span of data:

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

follows from closed form formula for w^*

$$\hat{R} = \sum_{i=1}^{n} \left(\left(\sum_{j=1}^{n} \alpha_j x_j \right)^T x_i - y_i \right)^2 = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j \underbrace{(x_j^T x_i)}_{k(x_j, x_i)} - y_i \right)^2$$

w^*

$$\|w\|_2^2 = w^T w = \left(\sum_{i=1}^{n} \alpha_i x_i \right)^T \left(\sum_{j=1}^{n} \alpha_j x_j \right) = \sum_{i,j=1}^{n} \alpha_i \alpha_j \underbrace{x_i^T x_j}$$

$\alpha^T K \alpha$

Kernelized linear regression

$$\min_{\alpha_1,\ldots,\alpha_n} \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) - y_i \right)^2 + \lambda \alpha^T \mathbf{K} \alpha$$

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

$\alpha^T K_i$ ← ith column of K

$\| \alpha^T K - y \|_2^2$

$\alpha^T K \alpha = \sum_{ij} \alpha_i \alpha_j k(x_i, x_j)$

$\alpha = [\alpha_1 \ldots \alpha_n]^T \in \mathbb{R}^n$

Learning & Predicting with KLR

- Learning: Solve least squares problem

$$\min_{\alpha} ||\alpha^T \mathbf{K} - \mathbf{y}||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

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

- Prediction: For data point ***x*** predict response *y* as

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

$y = w^T x = \left(\sum_i \alpha_i^* x_i\right)^T x = \sum_i \alpha_i^* (x_i^T x)$ $\quad k(x_i, x)$

Demo: Kernelized linear regression

KLR for the linear kernel

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

$$f(x) = \sum_{i=1}^{n} \alpha_i^* k(x_i, x) = \sum_{i=1}^{n} \alpha_i^* (x_i^T x)$$

$$= \underbrace{\left(\sum_{i=1}^{n} \alpha_i^* x_i \right)^T}_{w} x$$

Application: semi-parametric regression

- Often, parametric models are too „rigid“, and non-parametric models fail to extrapolate
- **Solution**: Use additive combination of linear and non-linear kernel function

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

$c_1 > 0$, $c_2 > 0$

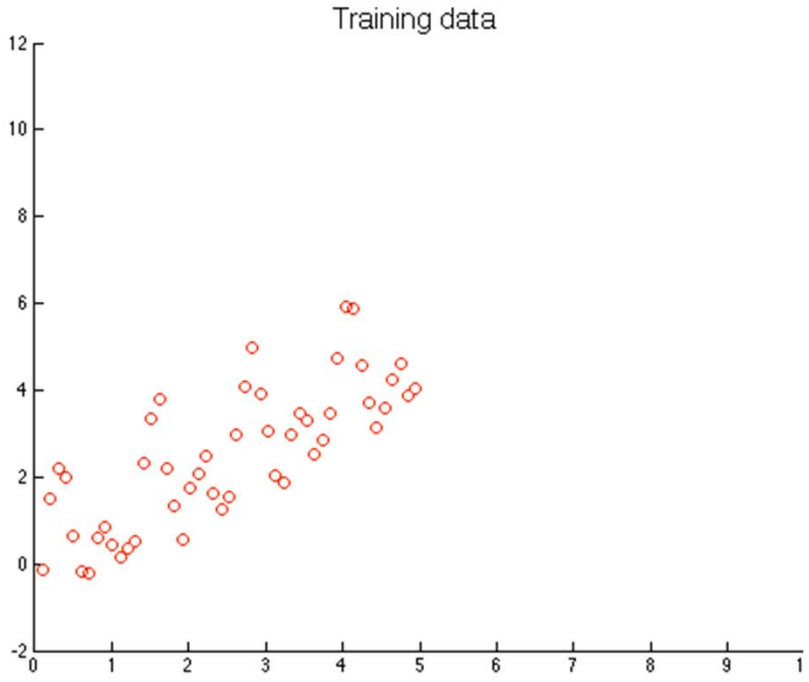
$$f(x) = \sum_i \alpha_i^* k(x_i, x) = c_1 \sum_{i=1}^{n} \exp\left(-\frac{\|x_i - x\|_2^2}{h^2}\right) \cdot \alpha_i + \underbrace{\left(c_2 \sum_{i=1}^{n} \alpha_i x_i\right)^T}_{w} x$$

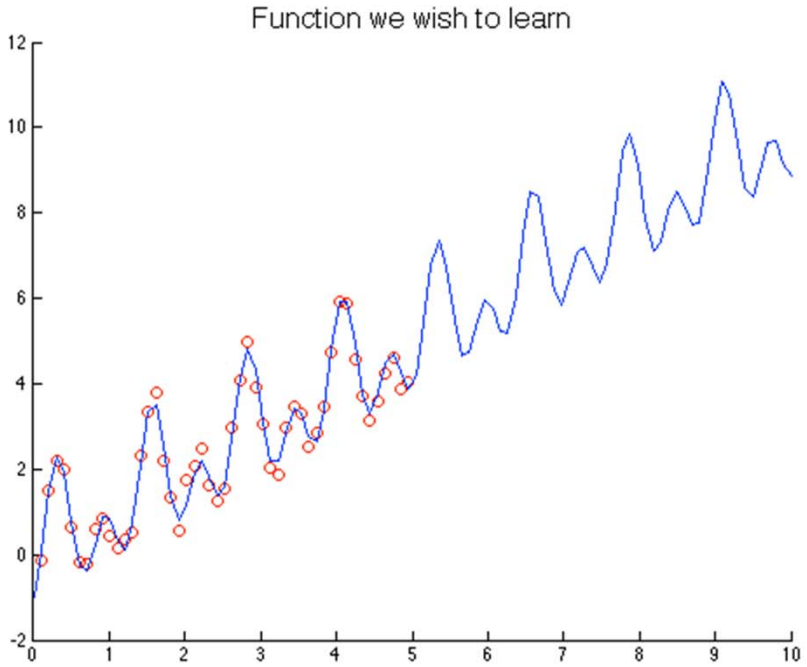
$$= w^T x + g(x)$$

linear; smooth nonlinear

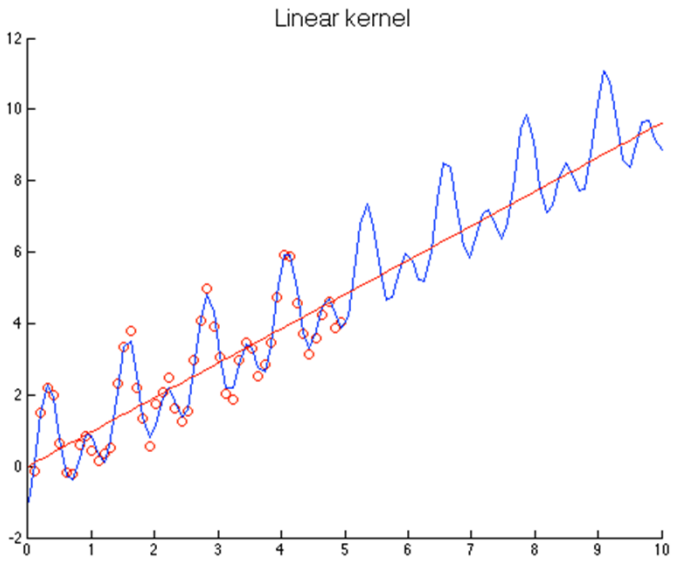
Demo: Semi-parametric KLR

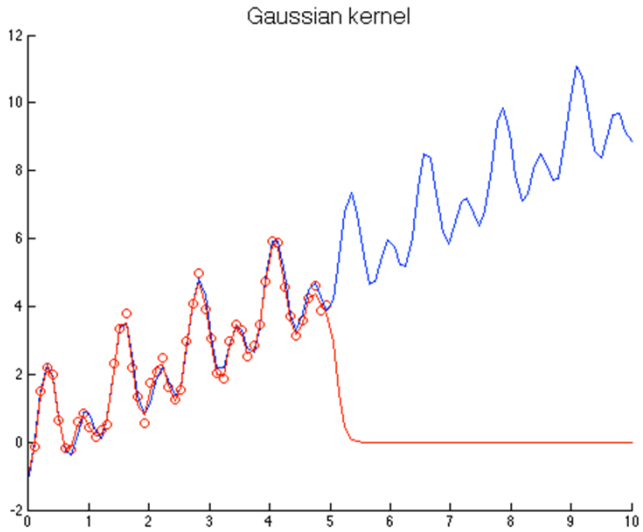
Example

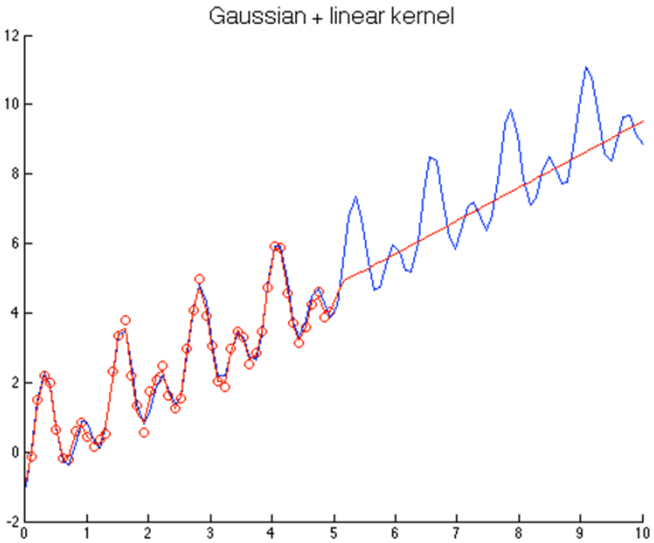

Training data
12
10
8
6
4
2
0
-2
0
1
2
3
4
5
6
7
8
9
1

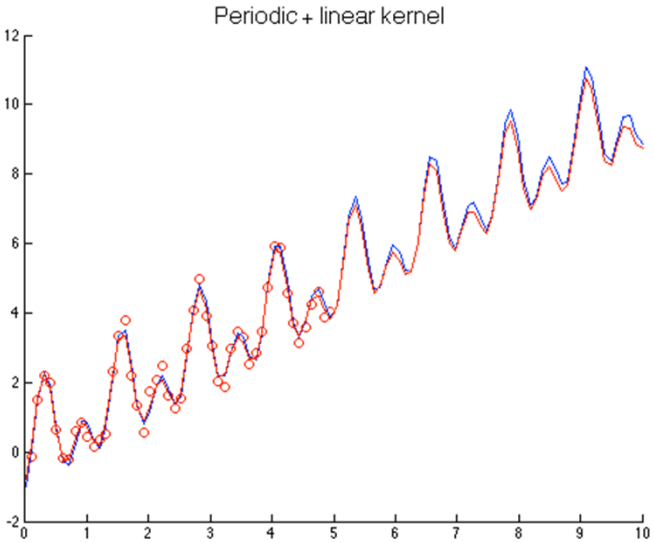

Function we wish to learn
12
10
8
6
4
2
0
-2
0
1
2
3
4
5
6
7
8
9
10

Example fits









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.)

Parameter demo

What about overfitting?

- Kernels map to (very) high-dimensional spaces.
- Why don't we overfit??

- 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

$$\min_{\alpha} \underbrace{||\alpha^T \mathbf{K} - \mathbf{y}||_2^2}_{\text{Data fit}} + \underbrace{\lambda \alpha^T \mathbf{K} \alpha}_{\text{Regularizer}}$$

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