Advanced Machine Learning
Lecture 11: Kernel methods

Sandjai Bhulai
Vrije Universiteit Amsterdam

s.bhulai@vu.nl

Kernel methods

- The linear models that we have studied are parametric models $y(x, w)$ based on adaptive parameters w
  - Use training data to learn w
  - Discard training data
  - Make predictions based on $y(x, w)$

- Similar approach in non-linear models (NN)

Kernel methods

- Kernel methods keep the training data (memory-based methods)
- Fast to train, slow in predictions

- Linear parametric methods have a dual representation based on kernels

Kernel methods

- Consider ridge regression:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

- Setting the gradient of $J(w)$ to zero, yields

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}\boldsymbol{\phi}(\mathbf{x}_n) = \sum_{n=1}^{N} a_n\boldsymbol{\phi}(\mathbf{x}_n) = \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{a}$$

with $$a_n = -\frac{1}{\lambda}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}$$

Kernel methods

- Substitute $w = \Phi^T a$ into $J(w)$:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

- This yields:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{a}$$

- Define the Gram matrix: $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$

$$K_{nm} = \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$$

Kernel methods

- Substitute the Gram matrix $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}$ in:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{a}$$

- This yields

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}.$$

- Setting the derivative of $J(a)$ to zero, gives:

$$\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}.$$

Kernel methods

- Hence:

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}) = \mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^{\mathrm{T}}\left(\mathbf{K} + \lambda\mathbf{I}_N\right)^{-1}\mathbf{t}$$

with $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$

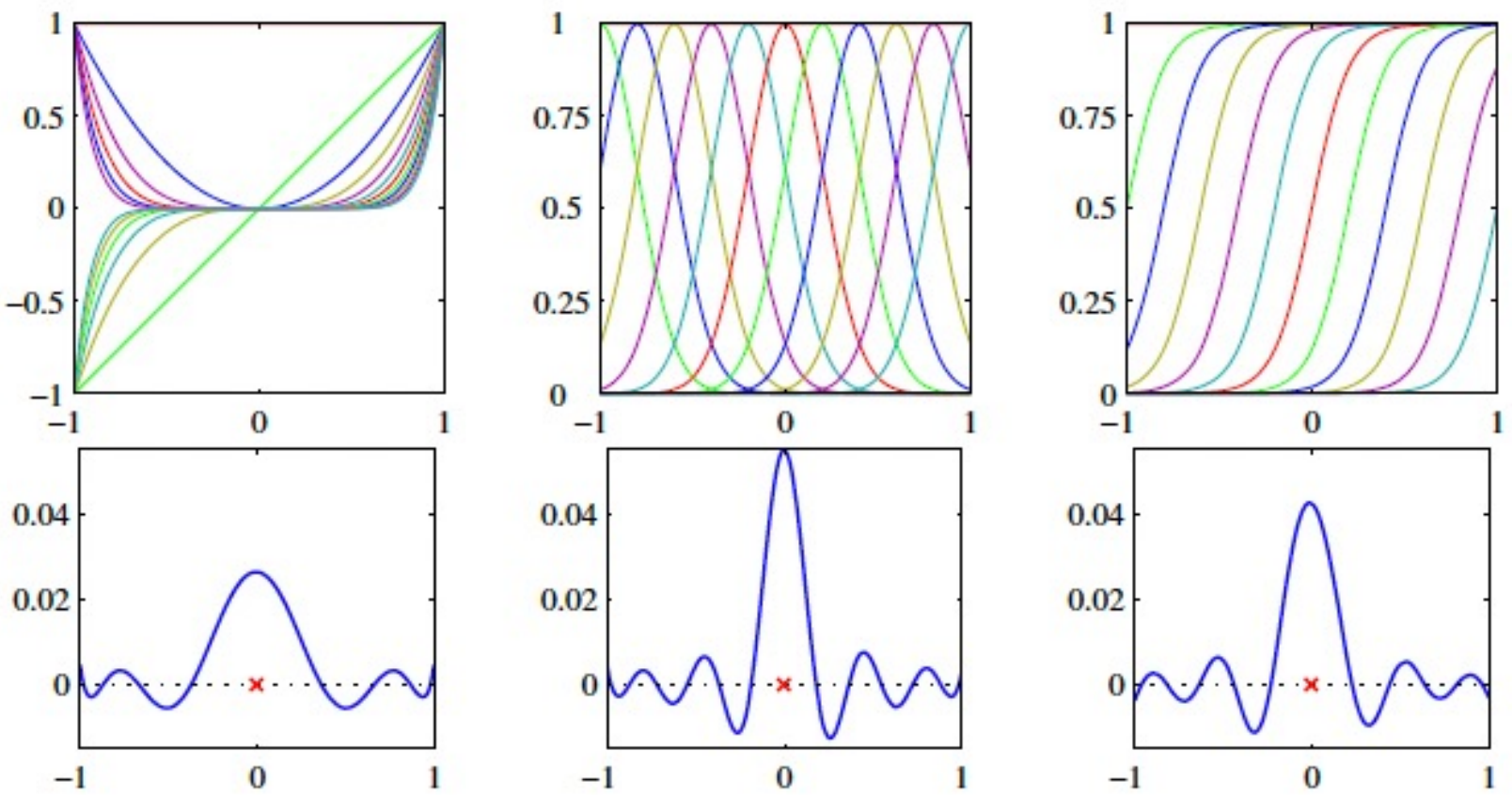
- The solution is fully expressed in $k(x, x')$
- Note that k is N x N whereas the original formulation had M x M parameters

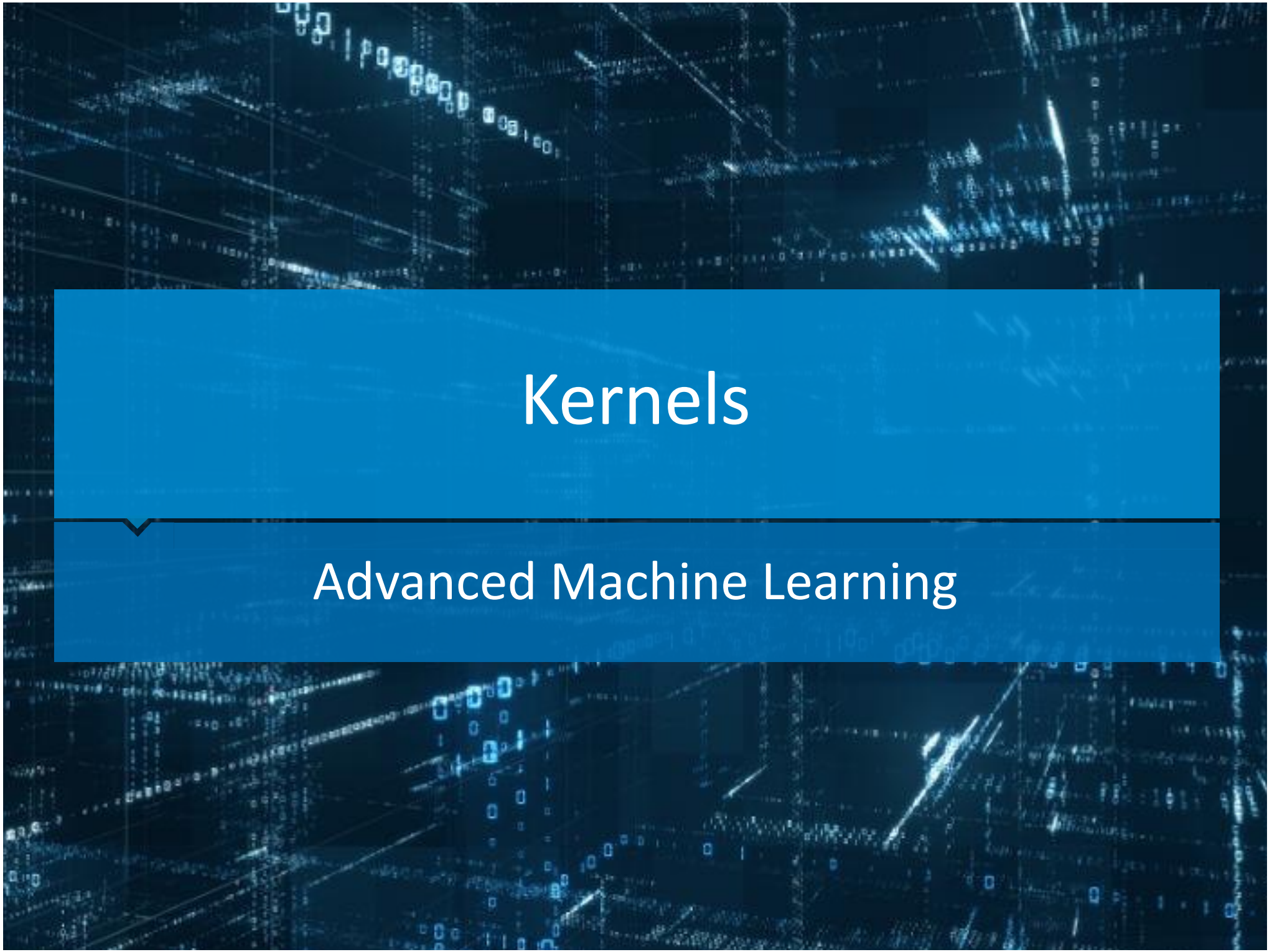
Kernel methods

Kernel trick:

- Any algorithm in which the examples only appear as dot products $x_i^T x_j$ can be kernelized by replacing $x_i^T x_j$ by $\varphi(x_i)^T \varphi(x_j) = k(x_i, x_j)$
- If we use algorithms that only depend on the Gram-matrix, then we never have to know (compute) the actual features
- This is the crucial point of kernel methods

Kernel methods



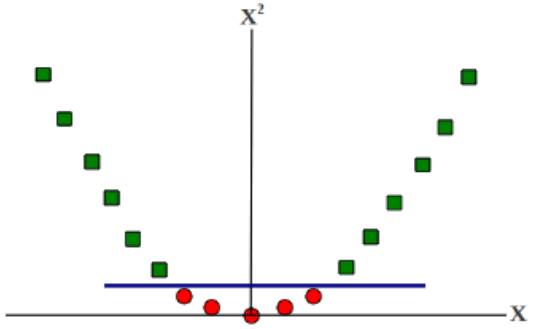

Kernels
Advanced Machine Learning

Kernels

- Kernels *map data to higher dimensions* (feature space) where it exhibits linear patterns

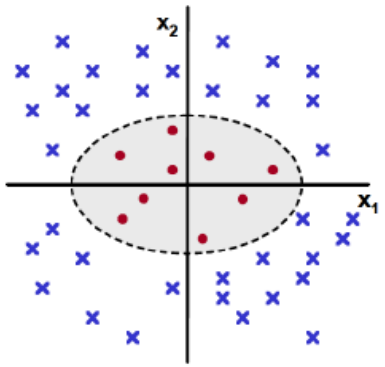
- Consider binary classification problem:

$$x \rightarrow \{x, x^2\}$$

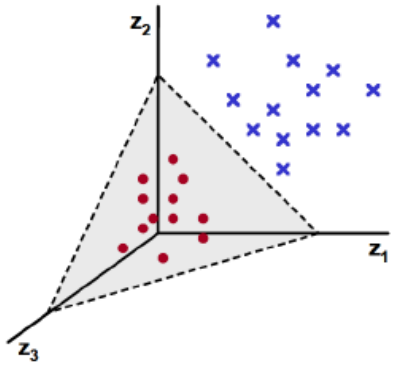


Kernels

- Another example:



$$\mathbf{x} = \{x_1, x_2\} \rightarrow \mathbf{z} = \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$$



Kernels

- Consider the following mapping for an example $\mathbf{x} = \{x_1, \ldots, x_D\}$

$$\phi : \mathbf{x} \rightarrow \{x_1^2, x_2^2, \ldots, x_D^2, x_1x_2, x_1x_2, \ldots, x_1x_D, \ldots\ldots, x_{D-1}x_D\}.$$

- This is a quadratic mapping
  - Computation of the mapping is inefficient
  - Computation using the mapping can be inefficient
- Kernels avoid these issues

Kernels

- Kernel function: $k(x, x') = \boldsymbol{\phi}(x)^{\mathrm{T}}\boldsymbol{\phi}(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x')$
- Consider kernel function $k(\mathbf{x}, \mathbf{z}) = \left(\mathbf{x}^{\mathrm{T}}\mathbf{z}\right)^2$
- Take $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{z} = (z_1, z_2)$

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= \left(\mathbf{x}^{\mathrm{T}}\mathbf{z}\right)^2 = (x_1 z_1 + x_2 z_2)^2 \\
&= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\
&= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^{\mathrm{T}} \\
&= \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{z}).
\end{aligned}
$$

Kernels

- We did not have to define the mapping φ
- Not any function can be used!
- The Gram matrix K must be a positive semidefinite matrix
- This is called Mercer's condition

Kernels

- Let $k_1(x, x')$ and $k_2(x, x')$ be valid kernels
- The following kernels will be valid as well

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b)
\end{aligned}
$$

Kernels

- Examples:
  - Linear (trivial) Kernel:

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z} \text{ (mapping function } \phi \text{ is identity - no mapping)}$$

  - Quadratic Kernel:

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2 \quad \text{or} \quad (1 + \mathbf{x}^\top \mathbf{z})^2$$

  - Polynomial Kernel (of degree d):

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^d \quad \text{or} \quad (1 + \mathbf{x}^\top \mathbf{z})^d$$

  - Radial Basis Function (RBF) Kernel:

$$k(\mathbf{x}, \mathbf{z}) = \gamma ||\mathbf{x} - \mathbf{z}||^2$$

Kernels

- RBF kernel is valid, because:

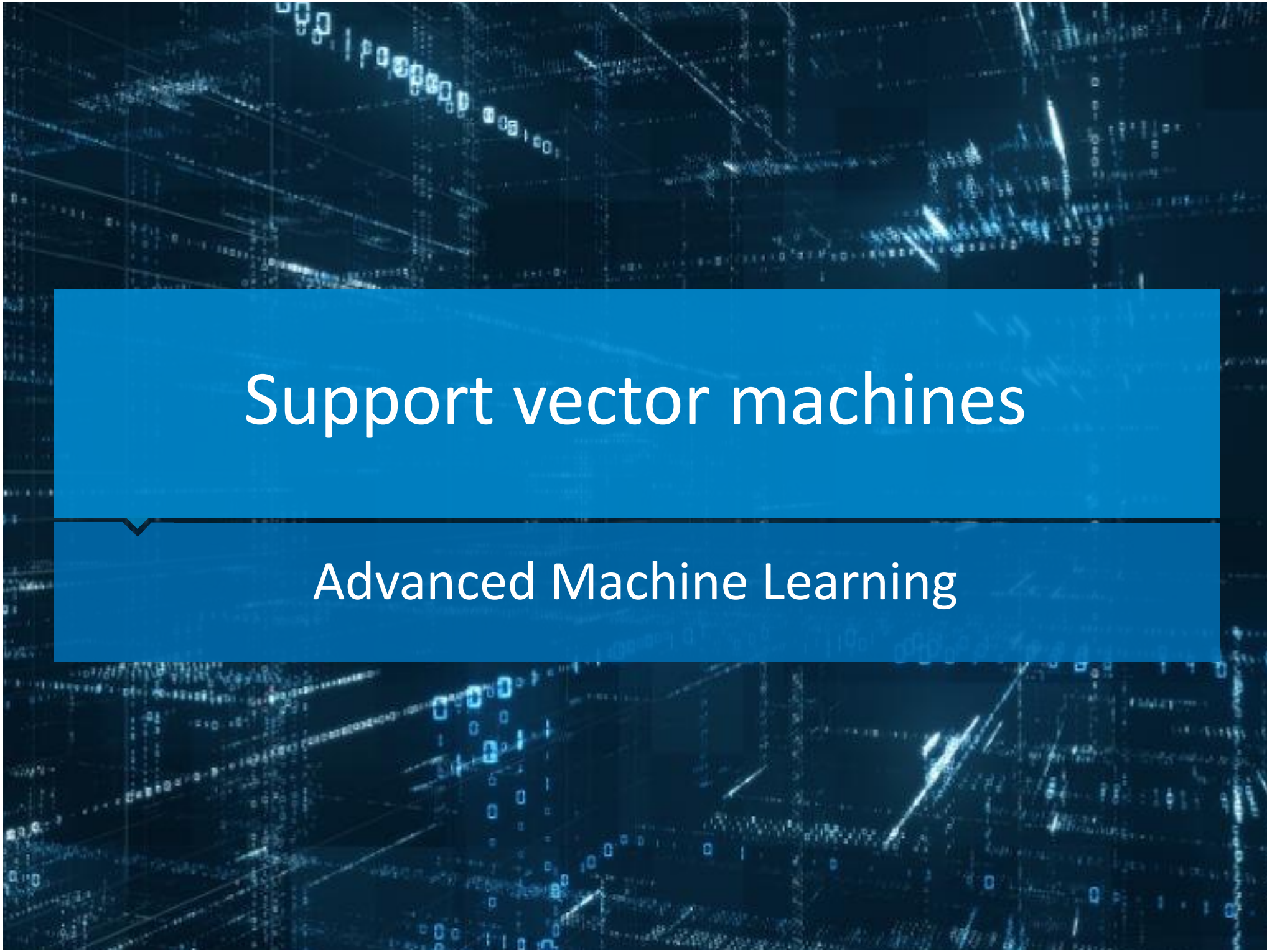
$$\|\mathbf{x} - \mathbf{x}'\|^2 = \mathbf{x}^{\mathrm{T}}\mathbf{x} + (\mathbf{x}')^{\mathrm{T}}\mathbf{x}' - 2\mathbf{x}^{\mathrm{T}}\mathbf{x}'$$

- For the Gaussian kernel, this gives:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\mathbf{x}^{\mathrm{T}}\mathbf{x}/2\sigma^2\right) \exp\left(\mathbf{x}^{\mathrm{T}}\mathbf{x}'/\sigma^2\right) \exp\left(-(\mathbf{x}')^{\mathrm{T}}\mathbf{x}'/2\sigma^2\right)$$

- Sigmoidal kernel is also valid:

$$k(\mathbf{x}, \mathbf{x}') = \tanh\left(a\mathbf{x}^{\mathrm{T}}\mathbf{x}' + b\right)$$


Support vector machines
Advanced Machine Learning

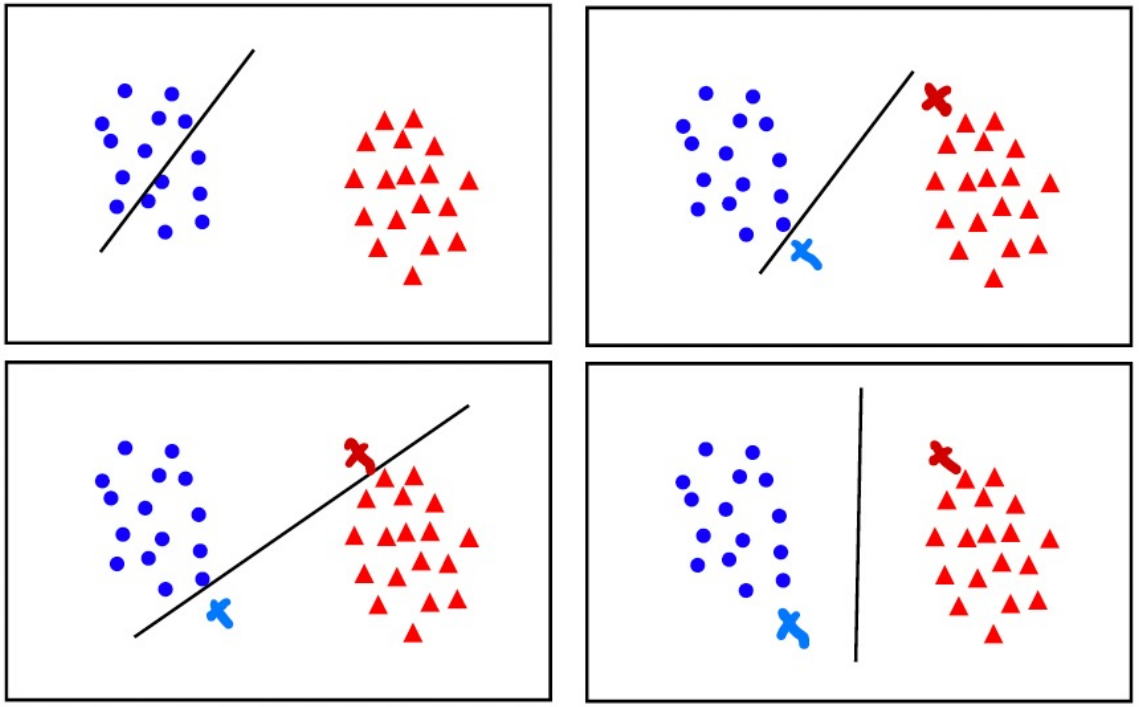
Sparse kernel machines

- Kernel methods consist of two modules:
  - The choice of kernel (this is non-trivial)
  - The algorithm which takes kernels as input

- Limitation:
  - Excessive computation times for prediction since $k(x_n, x_m)$ must be evaluated for all pairs x_n and x_m

Sparse kernel machines

- Linear classifier: $y(\mathbf{x_n}) = \mathbf{w}^t\mathbf{x}_n + b$
- Classification: $\begin{cases} t_n = +1 & \text{if } y(\mathbf{x}_n) \geq 0 \\ t_n = -1 & \text{if } y(\mathbf{x}_n) < 0 \end{cases}$

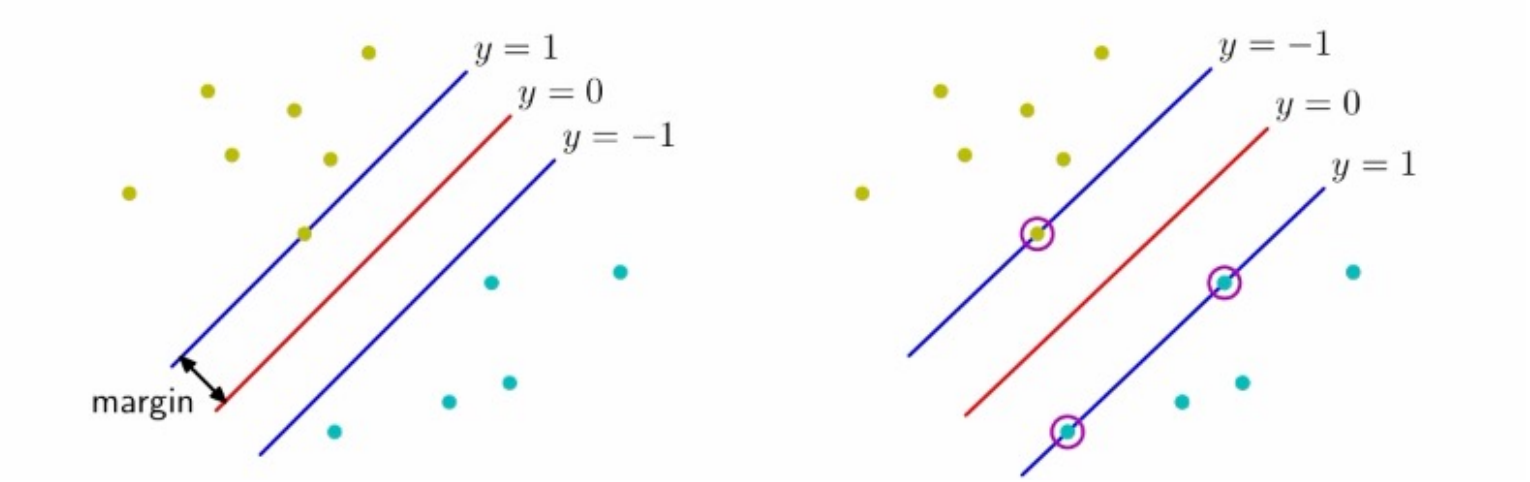


Sparse kernel machines

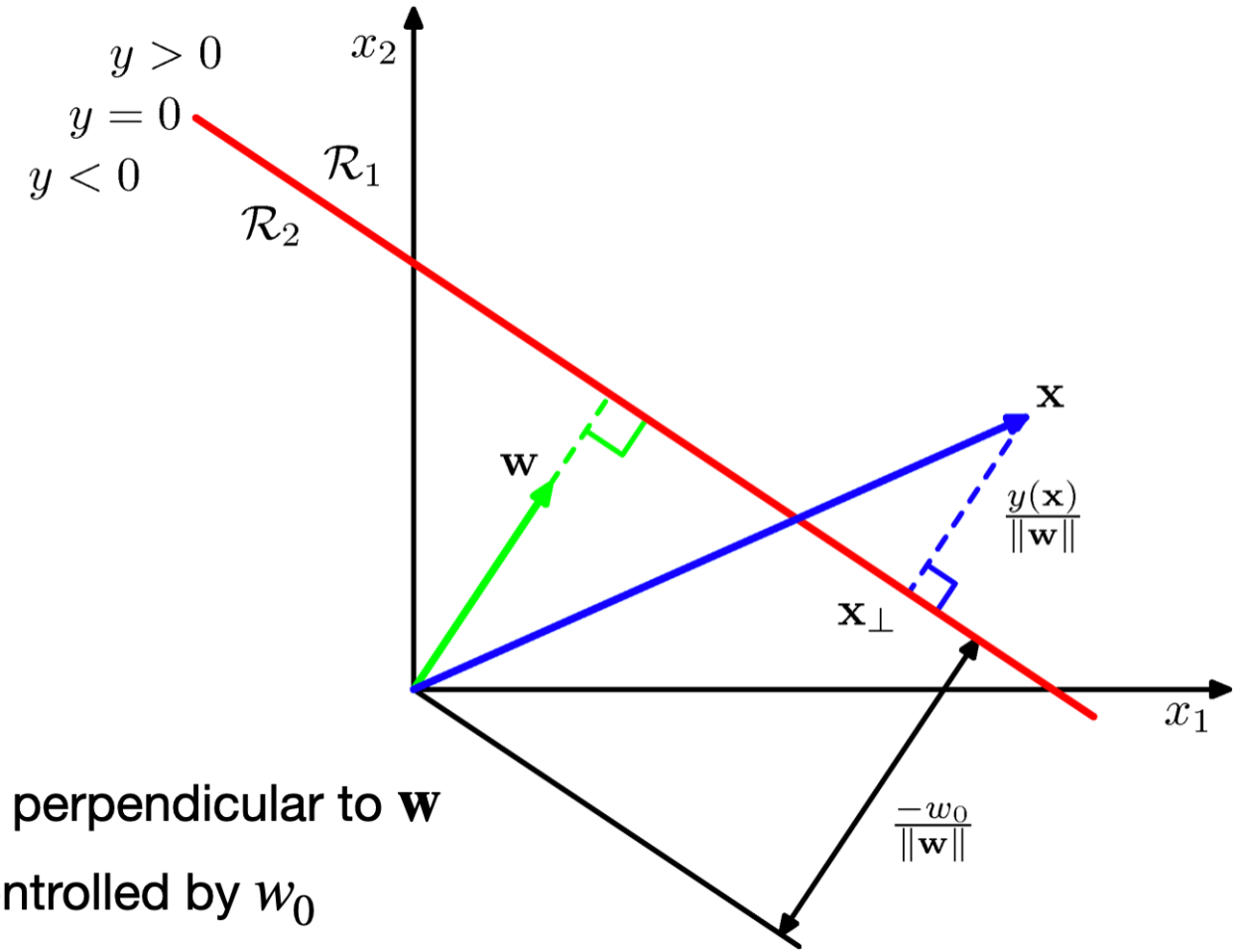
- Two-class classification problem:

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) + b$$

- Decision boundary $y(\boldsymbol{x})$



Sparse kernel machines



- Decision surface is perpendicular to $\mathbf{w}$
- Displacement is controlled by w_0

Sparse kernel machines

- Distance of a point x_n to decision surface:

$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}$$

- Optimization problem

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[t_n \left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b \right) \right] \right\}$$

- Difficult problem!

Sparse kernel machines

- By rescaling $\mathbf{w} \rightarrow \kappa \mathbf{w}$ and $b \rightarrow \kappa b$, the distance does not change
- For the closest point to the decision surface, set $t_n \left(\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) + b \right) = 1$

- Now, all points satisfy $t_n \left(\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) + b \right) \geqslant 1$
- Optimization problem now becomes to maximize $\|\mathbf{w}\|^{-1}$, or minimize $\|\mathbf{w}\|^2$

Sparse kernel machines

- Optimization problem:

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

s.t. $t_n\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)+b\right) \geqslant 1$

- Use Lagrange approach

$$L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{t_n(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)+b)-1\right\}$$

Sparse kernel machines

- Use Lagrange approach

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b) - 1 \right\}$$

- Take derivatives:

$$\begin{aligned} \mathbf{w} &= \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n) \\ 0 &= \sum_{n=1}^{N} a_n t_n . \end{aligned}$$

Sparse kernel machines

- Use Lagrange approach

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b) - 1 \right\}$$

- Substitution leads to:

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$a_n \geqslant 0,$$

$$\sum_{n=1}^{N} a_n t_n = 0.$$

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n)$$

$$0 = \sum_{n=1}^{N} a_n t_n.$$

Sparse kernel machines

- How to make predictions?

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) + b = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b.$$

$$\begin{aligned} \mathbf{w} &= \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n) \\ 0 &= \sum_{n=1}^{N} a_n t_n. \end{aligned}$$

- Karush-Kuhn-Tucker (KKT) conditions

$$\begin{aligned} a_n &\geqslant 0 \\ t_n y(\mathbf{x}_n) - 1 &\geqslant 0 \\ a_n \{t_n y(\mathbf{x}_n) - 1\} &= 0. \end{aligned}$$

$$\begin{aligned} a_n > 0 \quad &\rightarrow \quad t_n\, y(\mathbf{x}_n) = 1 \qquad \text{(support vectors)} \\ a_n = 0 \quad &\leftarrow \quad t_n\, y(\mathbf{x}_n) > 1 \qquad \text{(all other points)} \end{aligned}$$

Sparse kernel machines

- Prediction of class for datapoint $\mathbf{x}_n$:

$$y(\mathbf{x}_n) = \mathbf{w}^T\mathbf{x}_n + b$$

- Use $\mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n$ so that

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n^T\mathbf{x} + b \quad \xrightarrow{\text{kernel trick!}} \quad y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n, \mathbf{x}) + b$$

- Remember the KKT conditions:

(primal feasibility)	$t_n(\mathbf{w}^T\mathbf{x}_n + b) - 1 \geq 0$	for	$n = 1,\ldots,N$
(dual feasibility)	$a_n \geq 0$	for	$n = 1,\ldots,N$
(complimentary slackness)	$a_n(t_n(\mathbf{w}^T\mathbf{x}_n + b) - 1) = 0$	for	$n = 1,\ldots,N$

- Support vectors lie on maximum margin hyperplanes

$$a_n > 0 \quad \rightarrow \quad t_n\, y(\mathbf{x}_n) = 1 \qquad \text{(support vectors)}$$
$$a_n = 0 \quad \leftarrow \quad t_n\, y(\mathbf{x}_n) > 1 \qquad \text{(all other points)}$$

Sparse kernel machines

- Prediction of class for datapoint $\mathbf{x}$:

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n^T \mathbf{x} + b \quad \rightarrow \quad y(\mathbf{x}) = \sum_{m \in S} a_m t_m k(\mathbf{x}_m, \mathbf{x}) + b$$

- Find b by using that $t_n y_n(\mathbf{x}) = 1$ if $\mathbf{x}_n$ lies on the margin boundary! ($\mathbf{x}_n$ is a support vector)

- Then $t_n \left(\sum_{m \in S} a_m t_m k(\mathbf{x}_m, \mathbf{x}_n) + b \right) = 1$

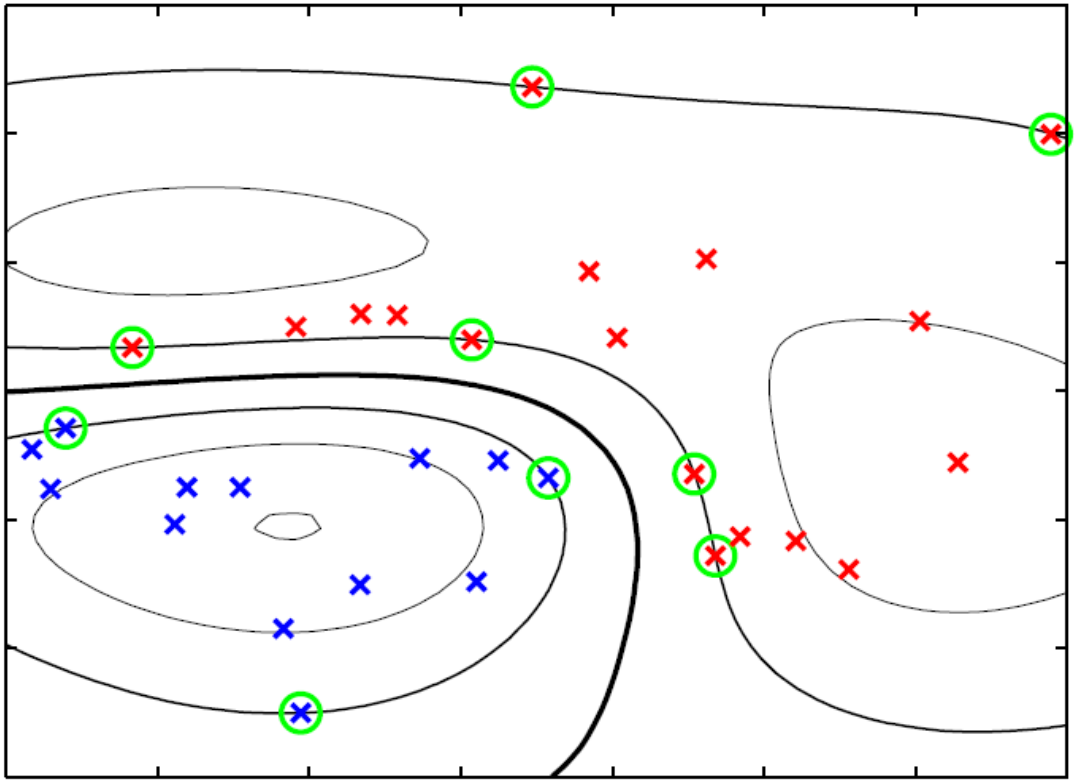
$$\sum_{m \in S} a_m t_m k(\mathbf{x}_m, \mathbf{x}_n) + b = t_n$$

$$\rightarrow \quad b = t_n - \sum_{m \in S} a_m t_m k(\mathbf{x}_m, \mathbf{x}_n)$$

- More stable to average over all support vectors (depending on optimizer, $\mathbf{a}_n$ may not be perfect

$$\rightarrow \quad b = \frac{1}{N_S} \sum_{n \in S} \left(t_n - \sum_{m \in S} a_m t_m k(\mathbf{x}_m, \mathbf{x}_n) \right)$$

Support vector machines



Support vector machines

- Minimization problem can be formulated as

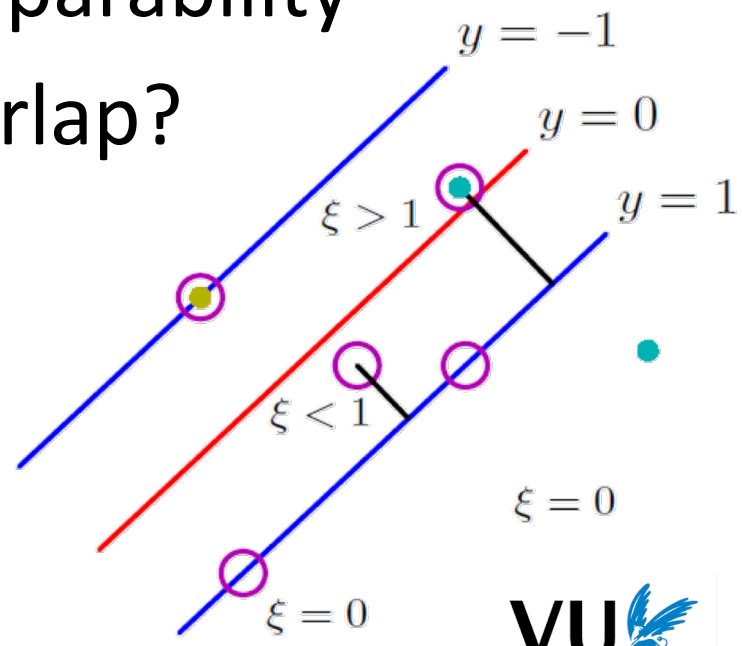
$$\sum_{n=1}^{N} E_\infty(y(\mathbf{x}_n)t_n - 1) + \lambda\|\mathbf{w}\|^2$$

- This only works for perfect separability
- What to do when classes overlap?

$$t_n y(\mathbf{x}_n) - 1 \geqslant 0$$

$$\downarrow$$

$$t_n y(\mathbf{x}_n) \geqslant 1 - \xi_n,$$



Support vector machines

- New optimization problem:

$$\text{minimize} \quad C\sum_{n=1}^{N}\xi_n + \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \quad t_n y(\mathbf{x}_n) \geqslant 1 - \xi_n,$$

$$\xi_n \geqslant 0$$

- Lagrangian:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N} a_n \{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n$$

Support vector machines

- Lagrangian:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N} a_n \{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n$$

- Take derivatives:

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n)$$

$$\frac{\partial L}{\partial b} = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} a_n t_n = 0$$

$$\frac{\partial L}{\partial \xi_n} = 0 \quad \Rightarrow \quad a_n = C - \mu_n.$$

Support vector machines

- Substitution leads to

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$0 \leqslant a_n \leqslant C$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

SVM for regression

- Normal regularized error function

$$\frac{1}{2}\sum_{n=1}^{N}\{y_n - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2.$$

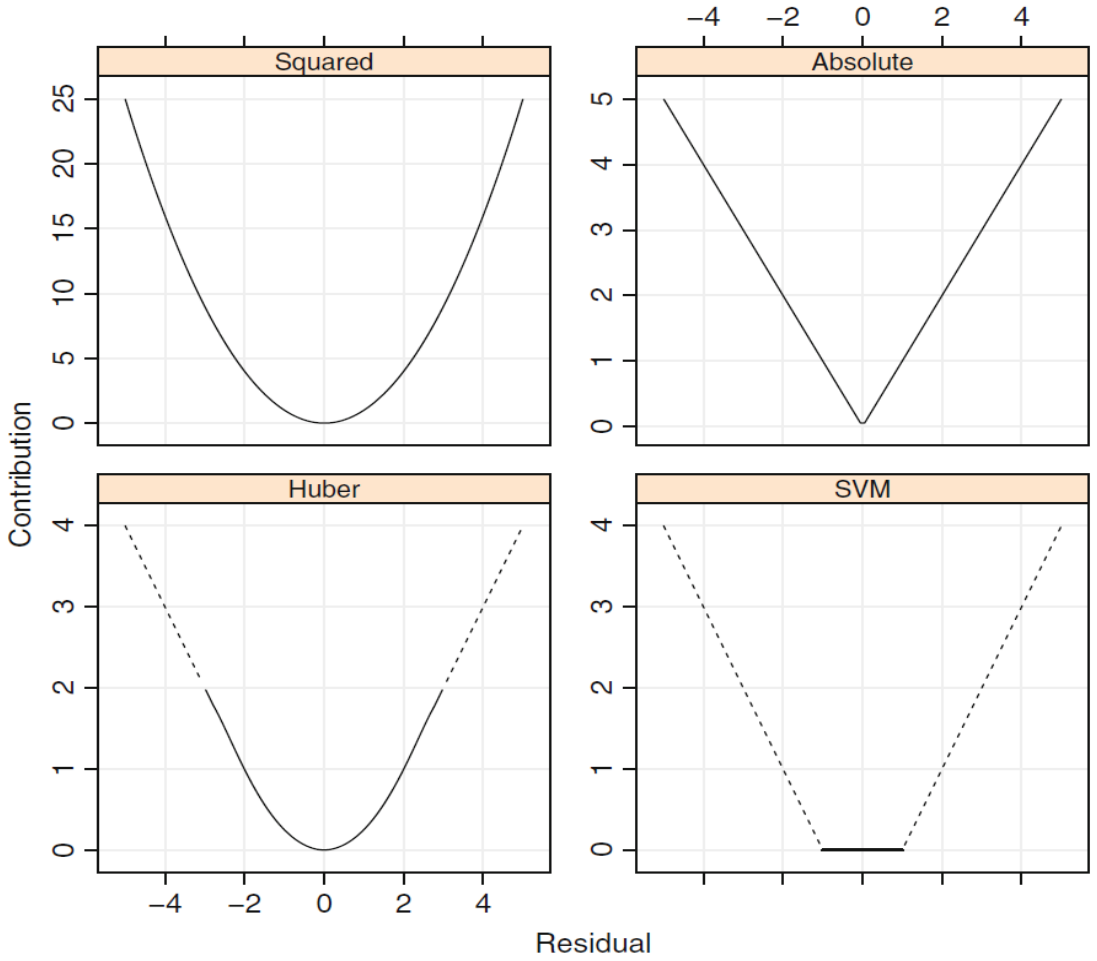
- Define

$$E_\epsilon(y(\mathbf{x}) - t) = \begin{cases} 0, & \text{if } |y(\mathbf{x}) - t| < \epsilon; \\ |y(\mathbf{x}) - t| - \epsilon, & \text{otherwise} \end{cases}$$

- SVM error function:

$$C\sum_{n=1}^{N} E_\epsilon(y(\mathbf{x}_n) - t_n) + \frac{1}{2}\|\mathbf{w}\|^2$$

SVM for regression



SVM for regression