Kernel Methods

Machine Learning

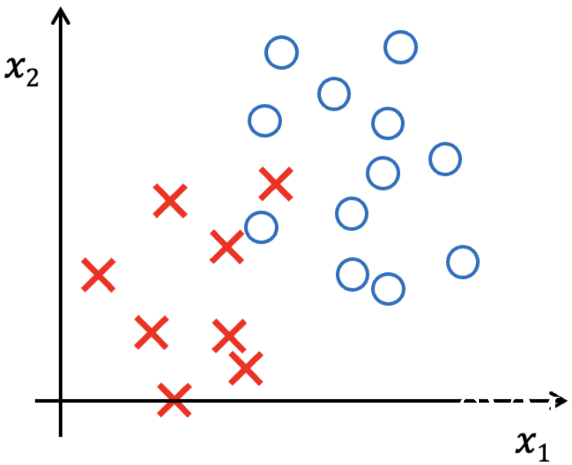
Hamid R Rabiee – Zahra Dehghanian

Spring 2025

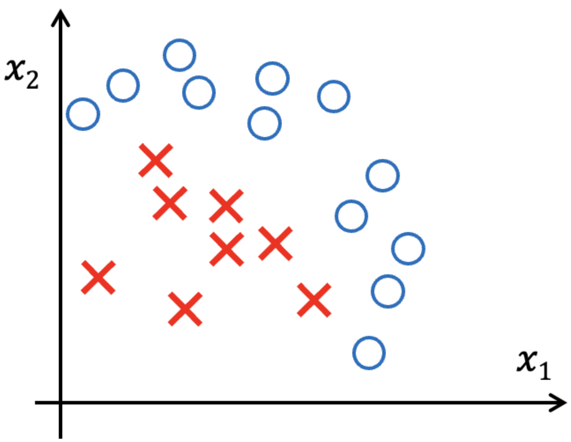
Sharif University
of Technology

Not linearly separable data

- Noisy data or overlapping classes (we discussed about it: soft margin)
  - Near linearly separable



- Non-linear decision surface
  - Transform to a new feature space



Nonlinear SVM

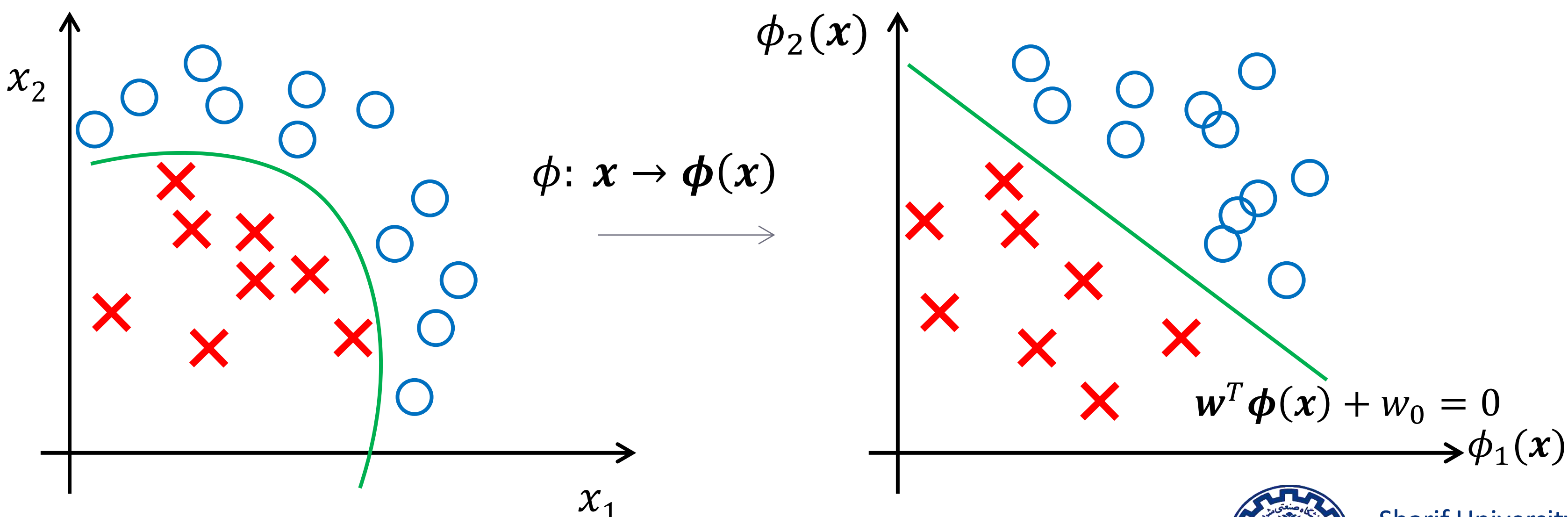
- Assume a transformation $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^m$ on the feature space
  - $\boldsymbol{x} \rightarrow \boldsymbol{\phi}(\boldsymbol{x})$

$\boldsymbol{\phi}(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \dots, \phi_m(\boldsymbol{x})]$

$\{\phi_1(\boldsymbol{x}), \dots, \phi_m(\boldsymbol{x})\}$: set of basis functions (or features)

$\phi_i(\boldsymbol{x}): \mathbb{R}^d \rightarrow \mathbb{R}$

- Find a hyper-plane in the transformed feature space:



Soft-margin SVM in a transformed space: Primal problem

- Primal problem:

$$\min_{\boldsymbol{w}, w_0} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N}\xi_n$$

$$\text{s.t.} \quad y^{(n)}\left(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}^{(n)}) + w_0\right) \geq 1 - \xi_n \quad n = 1, \dots, N$$

$$\xi_n \geq 0$$

  - $\boldsymbol{w} \in \mathbb{R}^m$: the weights that must be found
  - If $m \gg d$ (very high dimensional feature space) then there are many more parameters to learn

Soft-margin SVM in a transformed space: Dual problem

- Optimization problem:

$$\max_{\boldsymbol{\alpha}} \left\{ \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y^{(n)} y^{(m)} \boldsymbol{\phi}\left(\boldsymbol{x}^{(n)}\right)^T \boldsymbol{\phi}\left(\boldsymbol{x}^{(m)}\right) \right\}$$

Subject to $\sum_{n=1}^{N} \alpha_n y^{(n)} = 0$

$$0 \leq \alpha_n \leq C \quad n = 1, \dots, N$$

- If we have inner products $\boldsymbol{\phi}\left(\boldsymbol{x}^{(i)}\right)^T \boldsymbol{\phi}\left(\boldsymbol{x}^{(j)}\right)$, only $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_N]$ needs to be learnt.
  - not necessary to learn m parameters as opposed to the primal problem

Classifying a new data

$$\hat{y} = sign\left(w_0 \; + \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})\right)$$

where $\boldsymbol{w} = \sum_{\alpha_n > 0} \alpha_n \, y^{(n)} \boldsymbol{\phi}(\boldsymbol{x}^{(n)})$

and $w_0 \; = y^{(s)} - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}^{(s)})$

Kernel SVM

- Learns linear decision boundary in a high dimension space without explicitly working on the mapped data
- Let $\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}') = K(\boldsymbol{x}, \boldsymbol{x}')$ (kernel)
- Example: $\boldsymbol{x} = [x_1, x_2]$ and second-order $\boldsymbol{\phi}$:

$$\boldsymbol{\phi}(\boldsymbol{x}) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]$$

$$K(\boldsymbol{x}, \boldsymbol{x}') = 1 + x_1 x_1' + x_2 x_2' + x_1^2 x_1'^2 + x_2^2 x_2'^2 + x_1 x_1' x_2 x_2'$$

Kernel trick

- Compute $K(\boldsymbol{x}, \boldsymbol{x}')$ without transforming $\boldsymbol{x}$ and $\boldsymbol{x}'$

- Example: Consider $K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^T \boldsymbol{x}')^2$

$$= (1 + x_1 x_1' + x_2 x_2')^2$$

$$= 1 + 2x_1 x_1' + 2x_2 x_2' + x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2x_1 x_1' x_2 x_2'$$

This is an inner product in:

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

$$\boldsymbol{\phi}(\boldsymbol{x}') = \left[1, \sqrt{2}x_1', \sqrt{2}x_2', x'^2_1, x'^2_2, \sqrt{2}x_1' x_2'\right]$$

Polynomial kernel: Degree two

- We instead use $K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + 1)^2$ that corresponds to:

d-dimensional feature space $\boldsymbol{x} = [x_1, \dots, x_d]^T$

$$\boldsymbol{\phi}(\boldsymbol{x}) \\ = \left[1, \sqrt{2}x_1, \dots, \sqrt{2}x_d, x_1^2, . . , x_d^2, \sqrt{2}x_1x_2, \dots, \sqrt{2}x_1x_d, \sqrt{2}x_2x_3, \dots, \sqrt{2}x_{d-1}x_d\ \right]^T$$

Polynomial kernel

- This can similarly be generalized to d-dimensioan $\boldsymbol{x}$ and ϕs are polynomials of order M:

$$K(\boldsymbol{x},\boldsymbol{x}') = (1+\boldsymbol{x}^T\boldsymbol{x}')^M$$
$$= (1 + x_1x_1' + x_2x_2' + \cdots + x_dx_d')^M$$

- Example: SVM boundary for a polynomial kernel
  - $w_0 + \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) = 0$

$$\Rightarrow w_0 + \textstyle\sum_{\alpha_i>0}\alpha_i y^{(i)}\boldsymbol{\phi}\left(\boldsymbol{x}^{(i)}\right)^T\boldsymbol{\phi}(\boldsymbol{x}) = 0$$
$$\Rightarrow w_0 + \textstyle\sum_{\alpha_i>0}\alpha_i y^{(i)}k(\boldsymbol{x}^{(i)},\boldsymbol{x}) = 0$$
$$\Rightarrow w_0 + \textstyle\sum_{\alpha_i>0}\alpha_i y^{(i)}\left(1+\boldsymbol{x}^{(i)T}\boldsymbol{x}\right)^M = 0$$

$\Rightarrow$ Boundary is a polynomial of order M

Why kernel?

- kernel functions K can indeed be efficiently computed, with a cost proportional to d (the dimensionality of the input) instead of m.

- Example: consider the second-order polynomial transform:

$$\boldsymbol{\phi}(\boldsymbol{x}) = [1, x_1, \dots, x_d, x_1^2, x_1 x_2, \dots, x_d x_d]^T \qquad q = 1 + d + d^2$$

$$\boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}') = 1 + \sum_{i=1}^{d} x_i x_i' + \underbrace{\sum_{i=1}^{d} \sum_{j=1}^{d} x_i x_j x_i' x_j'}_{\sum_{i=1}^{d} x_i x_i' \times \sum_{j=1}^{d} x_j x_j'} \qquad O(q)$$

$$\boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}') = 1 + (x^T x') + (x^T x')^2$$

$$O(d)$$

Gaussian or RBF kernel

- If $K(\boldsymbol{x}, \boldsymbol{x}')$ is an inner product in some transformed space of x, it is good

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

- Take one dimensional case with $\gamma = 1$:

$$K(x, x') = \exp(-(x - x')^2)$$

$$= \exp(-x^2)\exp(-x'^2)\sum_{k=0}^{\infty}\frac{2^k x^k x'^k}{k!}$$

Some common kernel functions

- Linear: $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}'$
- Polynomial: $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + 1)^M$
- Gaussian: $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{\gamma})$
- Sigmoid: $k(\boldsymbol{x}, \boldsymbol{x}') = \tanh(a\boldsymbol{x}^T \boldsymbol{x}' + b)$

Kernel formulation of SVM

- Optimization problem:

$$\max_{\boldsymbol{\alpha}} \left\{ \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y^{(n)} y^{(m)} \quad k(\boldsymbol{x}^{(n)}, \boldsymbol{x}^{(m)}) \right\}$$

Subject to $\sum_{n=1}^{N} \alpha_n y^{(n)} = 0$

$$0 \leq \alpha_n \leq C \quad n = 1, \dots, N$$

$$\boldsymbol{Q} = \begin{bmatrix} y^{(1)} y^{(1)} K(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(1)}) & \cdots & y^{(1)} y^{(N)} K(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(1)}) \\ \vdots & \ddots & \vdots \\ y^{(N)} y^{(1)} K(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(1)}) & \cdots & y^{(N)} y^{(N)} K(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(N)}) \end{bmatrix}$$

Classifying a new data

$$\hat{y} = sign\left(w_0 \ + \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})\right)$$

where $\boldsymbol{w} = \sum_{\alpha_n > 0} \alpha_n \, y^{(n)} \boldsymbol{\phi}(\boldsymbol{x}^{(n)})$

and $w_0 \ = \qquad \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}^{(s)})$

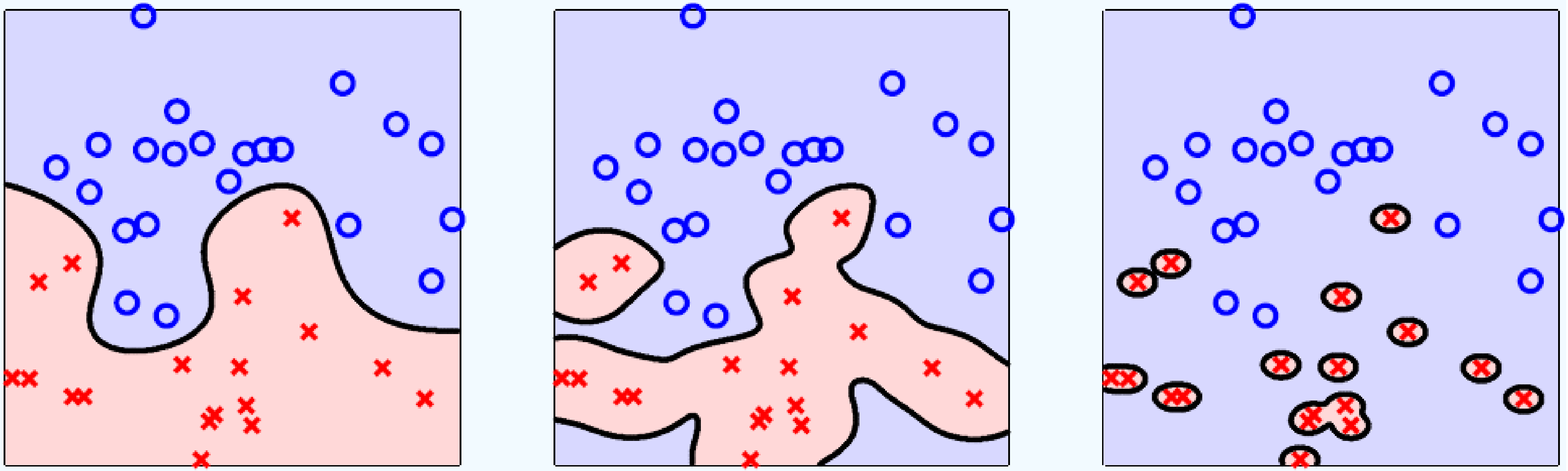
$$\hat{y} = sign\left(w_0 \ + \sum_{\alpha_n > 0} \alpha_n \, y^{(n)} \quad k(\boldsymbol{x}^{(n)}, \boldsymbol{x}) \qquad \right)$$

$$w_0 \ = y^{(s)} - \sum_{\alpha_n > 0} \alpha_n \, y^{(n)} \quad k(\boldsymbol{x}^{(n)}, \boldsymbol{x}^{(s)})$$

Gaussian kernel

- Example: SVM boundary for a Gaussian kernel
  - Considers a Gaussian function around each data point.
  - $w_0 + \sum_{\alpha_i > 0} \alpha_i y^{(i)} \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{x}^{(i)}\|^2}{\sigma}) = 0$
    - SVM + Gaussian Kernel can classify any arbitrary training set
      - Training error is zero when $\sigma \to 0$
        - All samples become support vectors (likely overfiting)

Hard margin Example



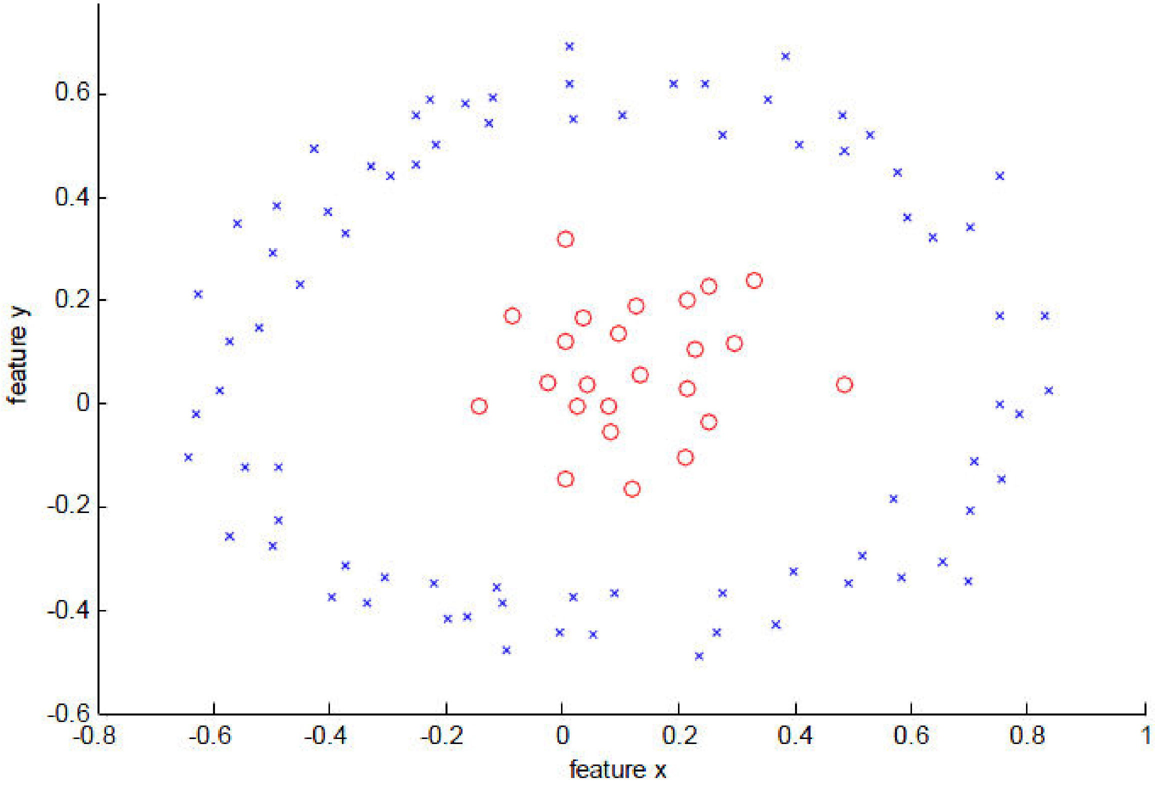
$\exp(-1\|\mathbf{x}-\mathbf{x}'\|^2)$ $\exp(-10\|\mathbf{x}-\mathbf{x}'\|^2)$ $\exp(-100\|\mathbf{x}-\mathbf{x}'\|^2)$

Y. Abu-Mostafa et. Al, 2012

- For narrow Gaussian (large σ), even the protection of a large margin cannot suppress overfitting.

SVM Gaussian kernel: Example

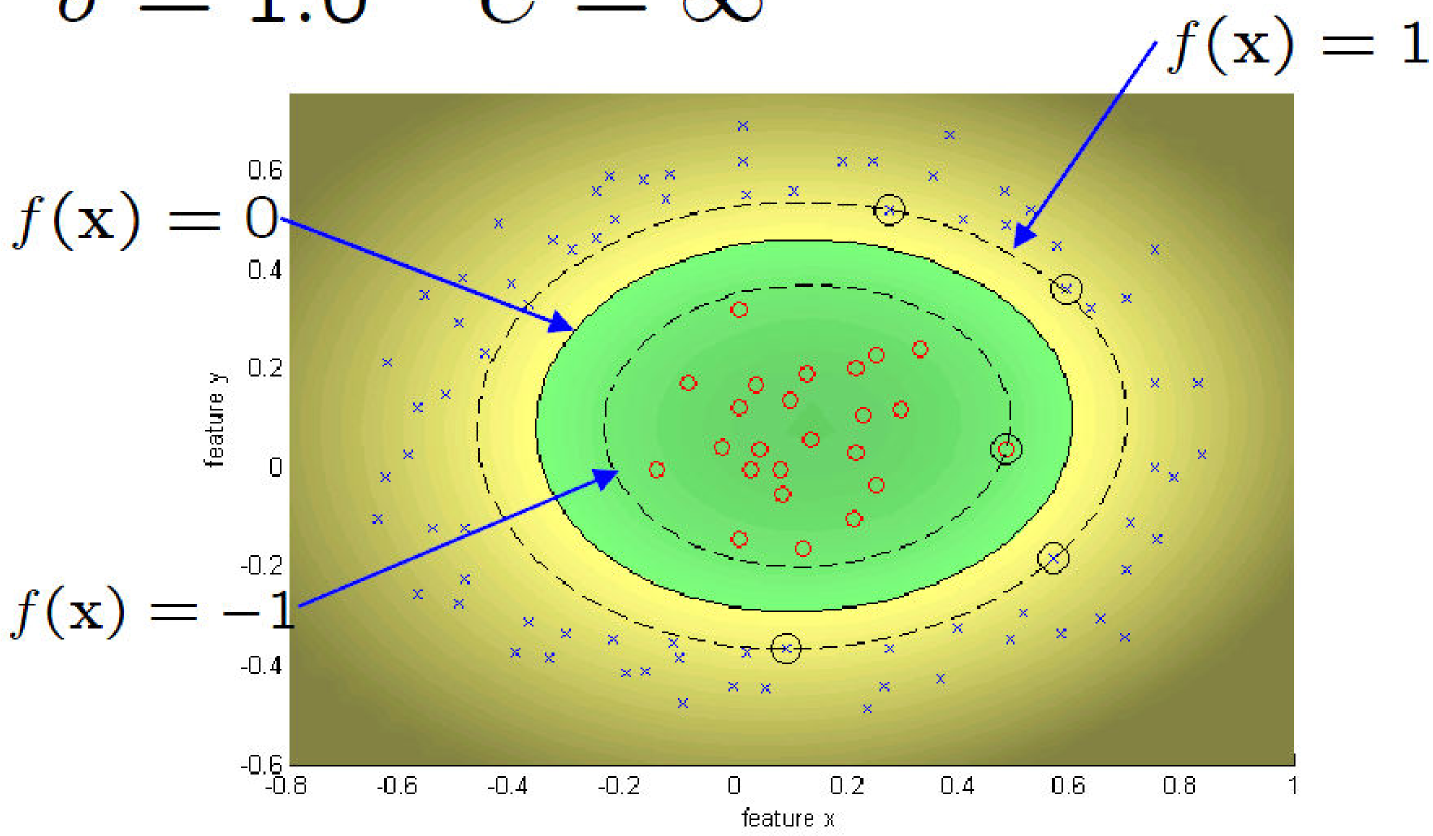
$$f(\boldsymbol{x}) = w_0 + \sum_{\alpha_i > 0} \alpha_i y^{(i)} \exp\left(-\frac{\left\|\boldsymbol{x} - \boldsymbol{x}^{(i)}\right\|^2}{2\sigma^2}\right)$$



This example has been adopted from Zisserman's slides

SVM Gaussian kernel: Example

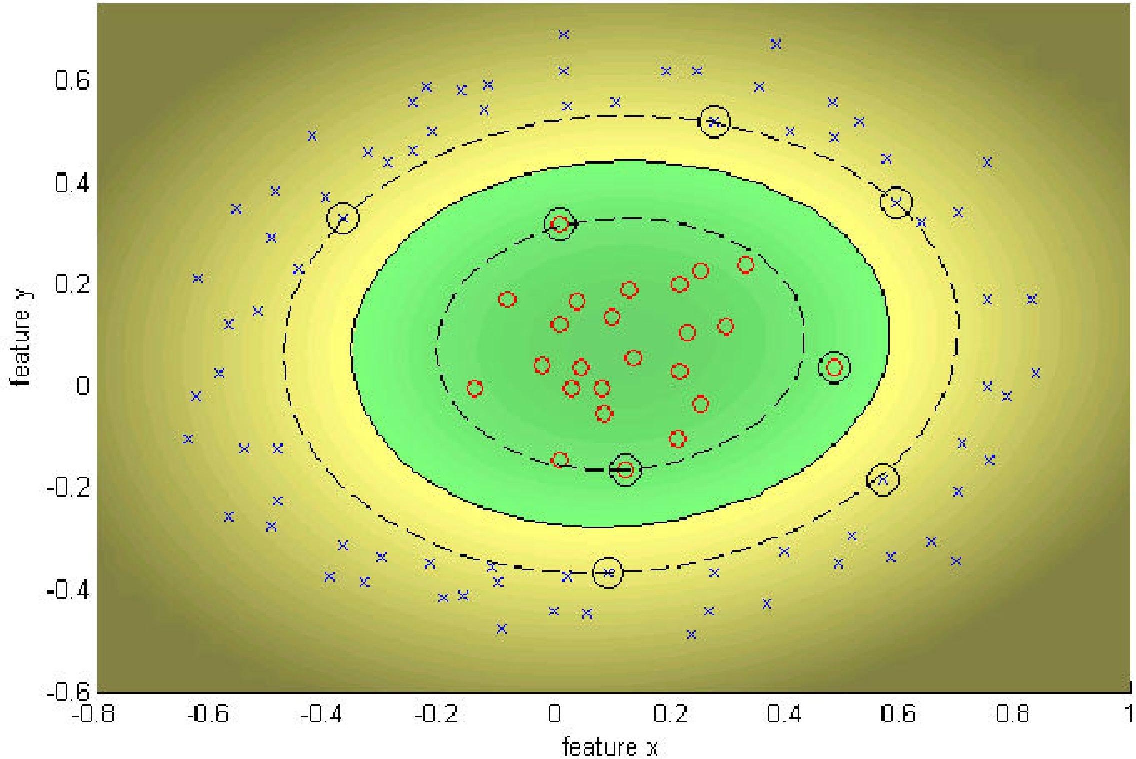
$$\sigma = 1.0 \qquad C = \infty$$



This example has been adopted from Zisserman's slides

SVM Gaussian kernel: Example

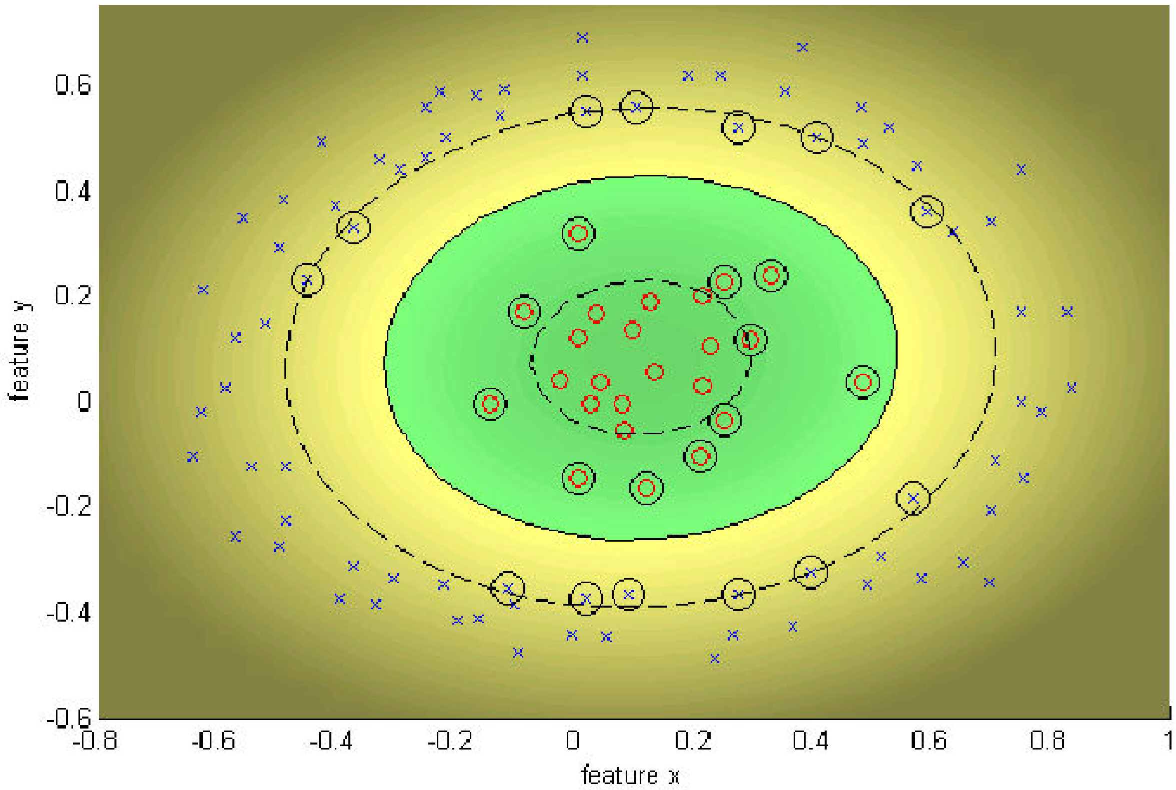
$$\sigma = 1.0 \quad C = 100$$



This example has been adopted from Zisserman's slides

SVM Gaussian kernel: Example

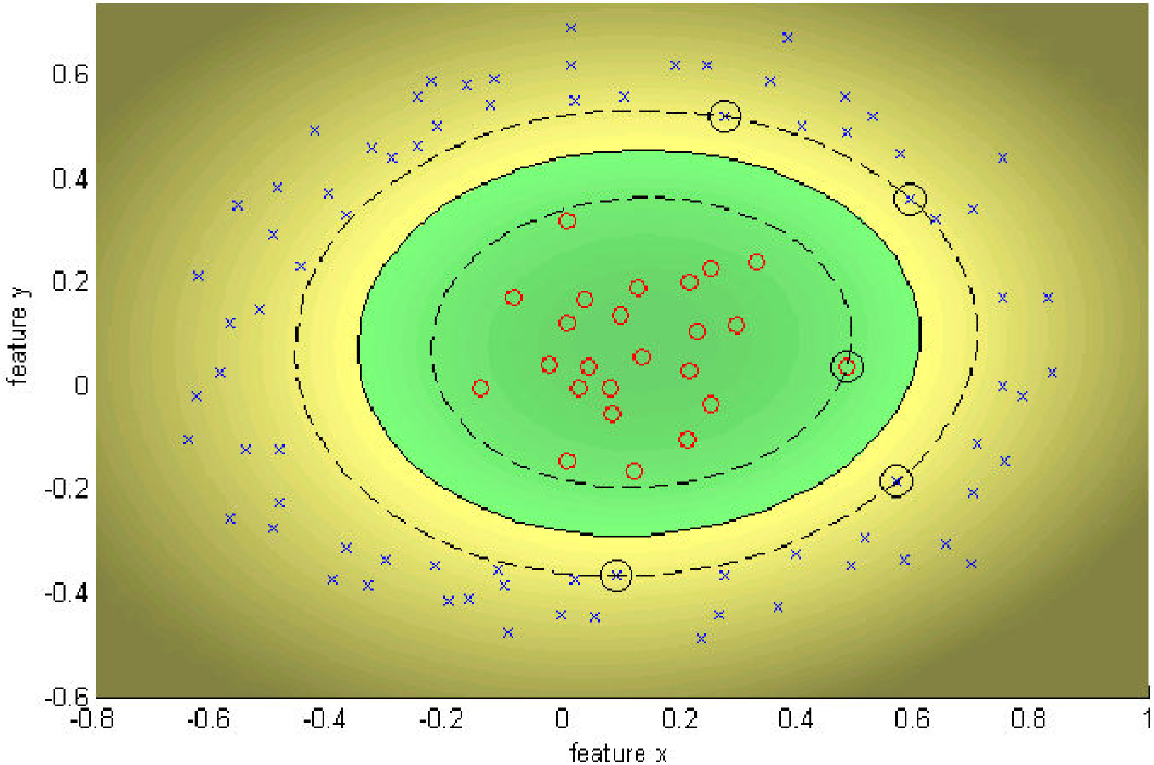
$$\sigma = 1.0 \quad C = 10$$



This example has been adopted from Zisserman's slides

SVM Gaussian kernel: Example

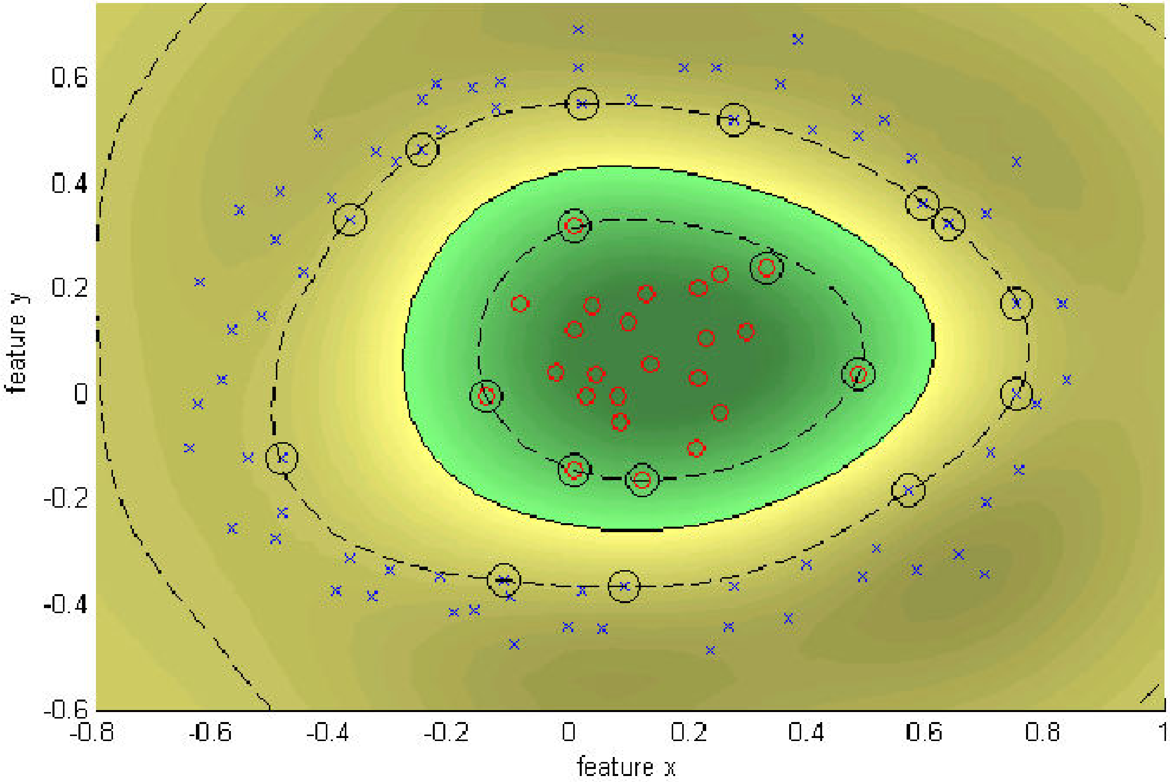
$$\sigma = 1.0 \quad C = \infty$$



This example has been adopted from Zisserman's slides

SVM Gaussian kernel: Example

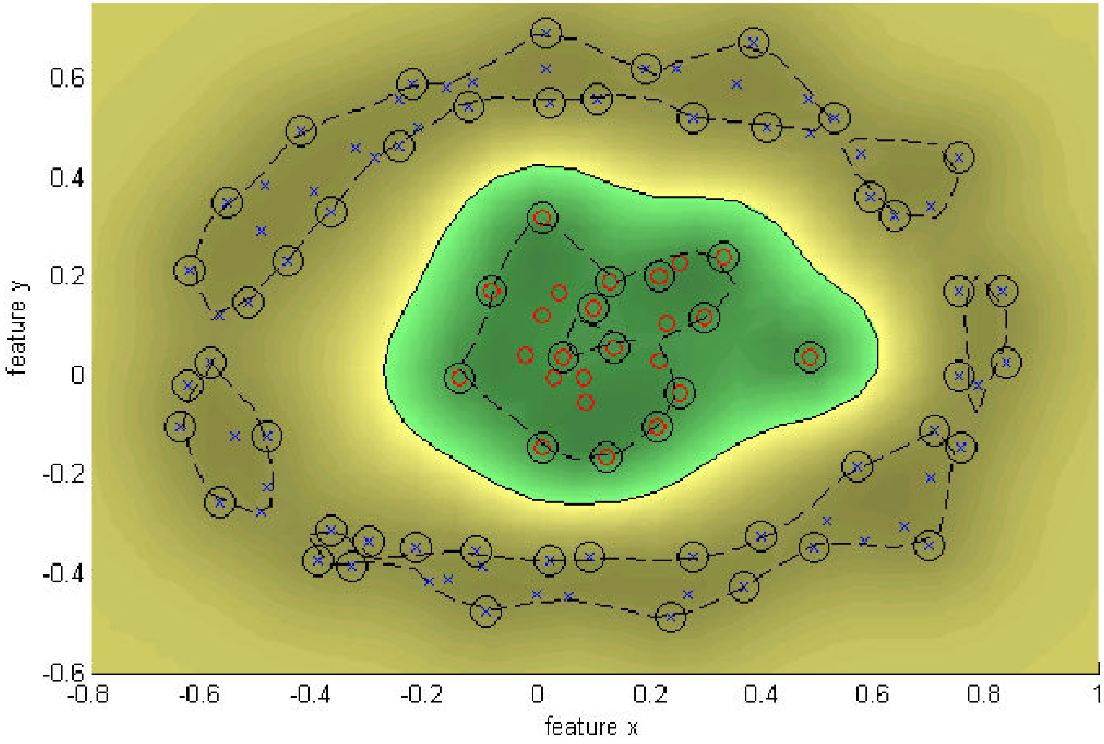
$$\sigma = 0.25 \qquad C = \infty$$



This example has been adopted from Zisserman's slides

SVM Gaussian kernel: Example

$$\sigma = 0.1 \qquad C = \infty$$



This example has been adopted from Zisserman's slides

Kernel trick: Idea

- Kernel trick → Extension of many well-known algorithms to kernel-based ones
  - By substituting the dot product with the kernel function
    - $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$
    - $k(\boldsymbol{x}, \boldsymbol{x}')$ shows the dot product of $\boldsymbol{x}$ and $\boldsymbol{x}'$ in the transformed space.

- Idea: when the input vectors appears only in the form of dot products, we can use kernel trick
  - Solving the problem without explicitly mapping the data
    - Explicit mapping is expensive if $\boldsymbol{\phi}(\boldsymbol{x})$ is very high dimensional

Kernel trick: Idea (Cont'd)

- Instead of using a mapping $\boldsymbol{\phi}: \mathcal{X} \leftarrow \mathcal{F}$ to represent $\boldsymbol{x} \in \mathcal{X}$ by $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathcal{F}$, a kernel function $k: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is used.
  - We specify only an inner product function between points in the transformed space (not their coordinates)
  - In many cases, the inner product in the embedding space can be computed efficiently.

Constructing kernels

- Construct kernel functions directly
  - Ensure that it is a valid kernel
    - Corresponds to an inner product in some feature space.

- Example: $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}')^2$
  - Corresponding mapping: $\boldsymbol{\phi}(\boldsymbol{x}) = \left[x_1^2, \sqrt{2}x_1 x_2, x_2^2\right]^T$ for $\boldsymbol{x} = [x_1, x_2]^T$

- We need a way to test whether a kernel is valid without having to construct $\boldsymbol{\phi}(\boldsymbol{x})$

Construct Valid Kernels

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

- $c > 0$, $k_{1:}$ valid kernel
- $f(.)$: any function
- $q(.)$: a polynomial with coefficients≥ 0
- $k_1, k_{2:}$ valid kernels
- $\boldsymbol{\phi}(\boldsymbol{x})$: a function from $\boldsymbol{x}$ to $\mathbb{R}^M$
  $k3(.,.)$: a valid kernel in $\mathbb{R}^M$
- $\boldsymbol{A}$: a symmetric positive semi-definite matrix
- $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ are variables (not necessarily disjoint) with $\boldsymbol{x} = (\boldsymbol{x}_a, \boldsymbol{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

[Bishop]

Valid kernel: Necessary & sufficient conditions

- Gram matrix $\boldsymbol{K}_{N\times N}$: $K_{ij} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$
  - Restricting the kernel function to a set of points $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(N)}\}$

$$K = \begin{bmatrix} k(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(1)}) & \cdots & k(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(N)}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(1)}) & \cdots & k(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(N)}) \end{bmatrix}$$

- **Mercer** Theorem: The kernel matrix is **Symmetric Positive Semi-Definite** (for any choice of data points)
  - Any symmetric positive definite matrix can be regarded as a kernel matrix, that is as an inner product matrix in some space

[Shawe-Taylor & Cristianini 2004]

Extending linear methods to kernelized ones

- Kernelized version of linear methods
  - Linear methods are famous
    - Unique optimal solutions, faster learning algorithms, and better analysis
  - However, we often require nonlinear methods in real-world problems and so we can use kernel-based version of these linear algorithms

- Replacing inner products with kernels in linear algorithms $\Rightarrow$ very flexible methods
  - We can operate in the mapped space without ever computing the coordinates of the data in that space

Example: kernelized minimum distance classifier

- If $\|\boldsymbol{x}-\boldsymbol{\mu}_1\| < \|\boldsymbol{x}-\boldsymbol{\mu}_2\|$ then assign $\boldsymbol{x}$ to $\mathcal{C}_1$

$$(\boldsymbol{x}-\boldsymbol{\mu}_1)^T(\boldsymbol{x}-\boldsymbol{\mu}_1) < (\boldsymbol{x}-\boldsymbol{\mu}_2)^T(\boldsymbol{x}-\boldsymbol{\mu}_2)$$

$$-2\boldsymbol{x}^T\boldsymbol{\mu}_1 + \boldsymbol{\mu}_1^T\boldsymbol{\mu}_1 < -2\boldsymbol{x}^T\boldsymbol{\mu}_2 + \boldsymbol{\mu}_2^T\boldsymbol{\mu}_2$$

$$-2\frac{\sum_{y^{(n)}=1}\boldsymbol{x}^T\boldsymbol{x}^{(n)}}{N_1} + \frac{\sum_{y^{(n)}=1}\sum_{y^{(m)}=1}\boldsymbol{x}^{(n)^T}\boldsymbol{x}^{(m)}}{N_1\times N_1} < -2\frac{\sum_{y^{(n)}=2}\boldsymbol{x}^T\boldsymbol{x}^{(n)}}{N_2} + \frac{\sum_{y^{(n)}=2}\sum_{y^{(m)}=2}\boldsymbol{x}^{(n)^T}\boldsymbol{x}^{(m)}}{N_2\times N_2}$$

$$-2\frac{\sum_{y^{(n)}=1}K\left(\boldsymbol{x},\boldsymbol{x}^{(n)}\right)}{N_1} + \frac{\sum_{y^{(n)}=1}\sum_{y^{(m)}=1}K\left(\boldsymbol{x}^{(n)},\boldsymbol{x}^{(m)}\right)}{N_1\times N_1} < -2\frac{\sum_{y^{(n)}=2}K\left(\boldsymbol{x},\boldsymbol{x}^{(n)}\right)}{N_2} + \frac{\sum_{y^{(n)}=2}\sum_{y^{(m)}=2}K\left(\boldsymbol{x}^{(n)},\boldsymbol{x}^{(m)}\right)}{N_2\times N_2}$$

Which information can be obtained from kernel?

- Example: we know all pairwise distances
  - $d\big(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{z})\big)^2 = \|\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{z})\|^2 = k(\boldsymbol{x}, \boldsymbol{x}) + k(\boldsymbol{z}, \boldsymbol{z}) - 2k(\boldsymbol{x}, \boldsymbol{z})$
  - Therefore, we also know distance of points from center of mass of a set

- Many dimensionality reduction, clustering, and classification methods can be described according to pairwise distances.
  - This allow us to introduce kernelized versions of them

Kernels for structured data

- Kernels also can be defined on general types of data
  - Kernel functions do not need to be defined over vectors
    - just we need a symmetric positive definite matrix

- Thus, many algorithms can work with general (non-vectorial) data
  - Kernels exist to embed strings, trees, graphs, …

- This may be more important than nonlinearity
  - kernel-based version of classical learning algorithms for recognition of structured data

Kernel function for objects

- Sets: Example of kernel function for sets:

$$k(A, B) = 2^{|A \cap B|}$$

- Strings: The inner product of the feature vectors for two strings can be defined as
  - e.g., sum over all common subsequences weighted according to their frequency of occurrence and lengths

A	E	G	A	T	E	A	G	G

E	G	T	E	A	G	A	E	G	A	T	G

Kernel trick advantages: summary

- Operating in the **mapped space without ever computing** the coordinates of the data in that space
- Besides vectors, we can introduce kernel functions for **structured data** (graphs, strings, etc.)
- Much of the **geometry** of the data in the embedding space is contained in all pairwise dot products
- In many cases, inner product in the embedding space can be computed efficiently.

Resources

- C. Bishop, "Pattern Recognition and Machine Learning", Chapter 6.1-6.2, 7.1.

- Yaser S. Abu-Mostafa, et al., "Learning from Data", Chapter 8.