6. Kernel Methods

Motivation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

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

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n)\phi(\mathbf{x}_n) = \Phi^T\mathbf{a} \quad \text{where} \quad a_n = -\frac{1}{\lambda}(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n)$$

Thus, we can express J as

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

Resolving for $\mathbf{a}$ we obtain: $\mathbf{a} = (K + \lambda I_N)^{-1}\mathbf{t}$ substituting:

$$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \mathbf{a}^T\Phi\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(K + \lambda I_N)^{-1}\mathbf{t}$$

Motivation

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T (K + \lambda I_N)^{-1} \mathbf{t}$$

where:

$$\mathbf{k}(\mathbf{x}) = \begin{pmatrix} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}) \\ \vdots \\ \phi(\mathbf{x}_N)^T \phi(\mathbf{x}) \end{pmatrix} \quad K = \begin{pmatrix} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_N) \end{pmatrix}$$

Thus, y is expressed only in terms of **dot products** between different pairs of $\phi(\mathbf{x})$, or in terms of the **kernel function**

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Motivation

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T (K + \lambda I_N)^{-1} \mathbf{t}$$

Now we have to invert a matrix of size $N \times N$, before it was $M \times M$ where $M < N$, but:

By expressing everything with the kernel function, we can deal with very high-dimensional or even **infinite**-dimensional feature spaces!

Idea: Don’t use features at all but simply define a similarity function expressed as the kernel!

Constructing Kernels

The straightforward way to define a kernel function is to first find a basis function $\phi(\mathbf{x})$ and to define:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

This means, k is an inner product in some space $\mathcal{H}$, i.e:

1.Symmetry: $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_j), \phi(\mathbf{x}_i) \rangle = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$

2.Linearity: $\langle a(\phi(\mathbf{x}_i) + \mathbf{z}), \phi(\mathbf{x}_j) \rangle = a\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle + a\langle \mathbf{z}, \phi(\mathbf{x}_j) \rangle$

3.Positive definite: $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_i) \rangle \geq 0$, equal if $\phi(\mathbf{x}_i) = \mathbf{0}$

Can we find conditions for k under which there is a (possibly infinite dimensional) basis function into $\mathcal{H}$, where k is an inner product?

Constructing Kernels

It turns out that if k is

1. symmetric, i.e. $k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_j, \mathbf{x}_i)$ and
2. positive definite, i.e.

$$K = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

“Gram Matrix”

is positive definite, then there exists a mapping $\phi(\mathbf{x})$ into a feature space $\mathcal{H}$ so that k can be expressed as an inner product in $\mathcal{H}$.

This means, we don’t need to find $\phi(\mathbf{x})$ explicitly!

We can directly work with k

A Simple Example

Define a kernel function as

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^2 \qquad \mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^2$$

This can be written as:

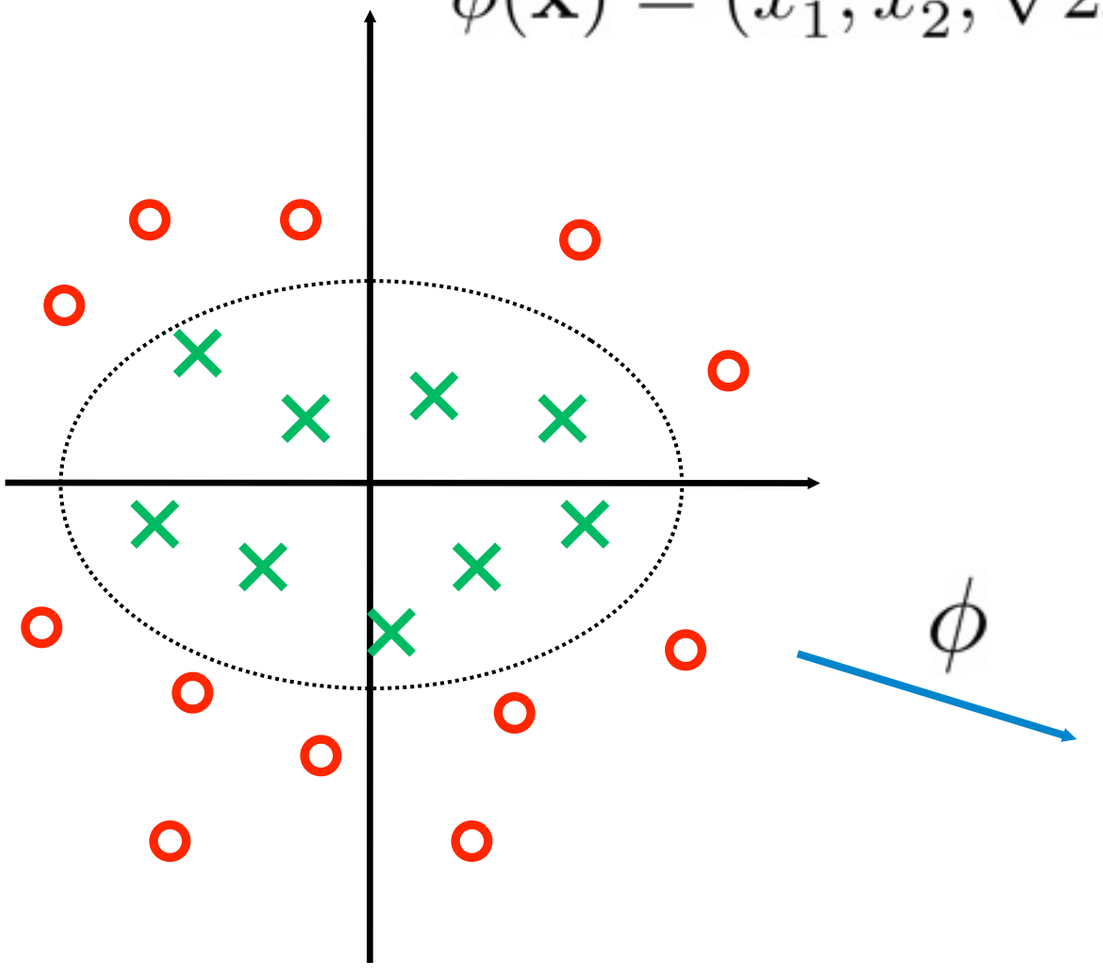
$$\begin{aligned}
(x_{i1}x_{j1} + x_{i2}x_{j2})^2 &= x_{i1}^2 x_{j1}^2 + 2x_{i1}^2 x_{j1} x_{i2}^2 x_{j2} + x_{i2}^2 x_{j2}^2 \\
&= (x_{i1}^2, x_{i2}^2, \sqrt{2}x_{i1}x_{i2})(x_{j1}^2, x_{j2}^2, \sqrt{2}x_{j1}x_{j2})^T \\
&= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)
\end{aligned}$$

It can be shown that this holds in general for

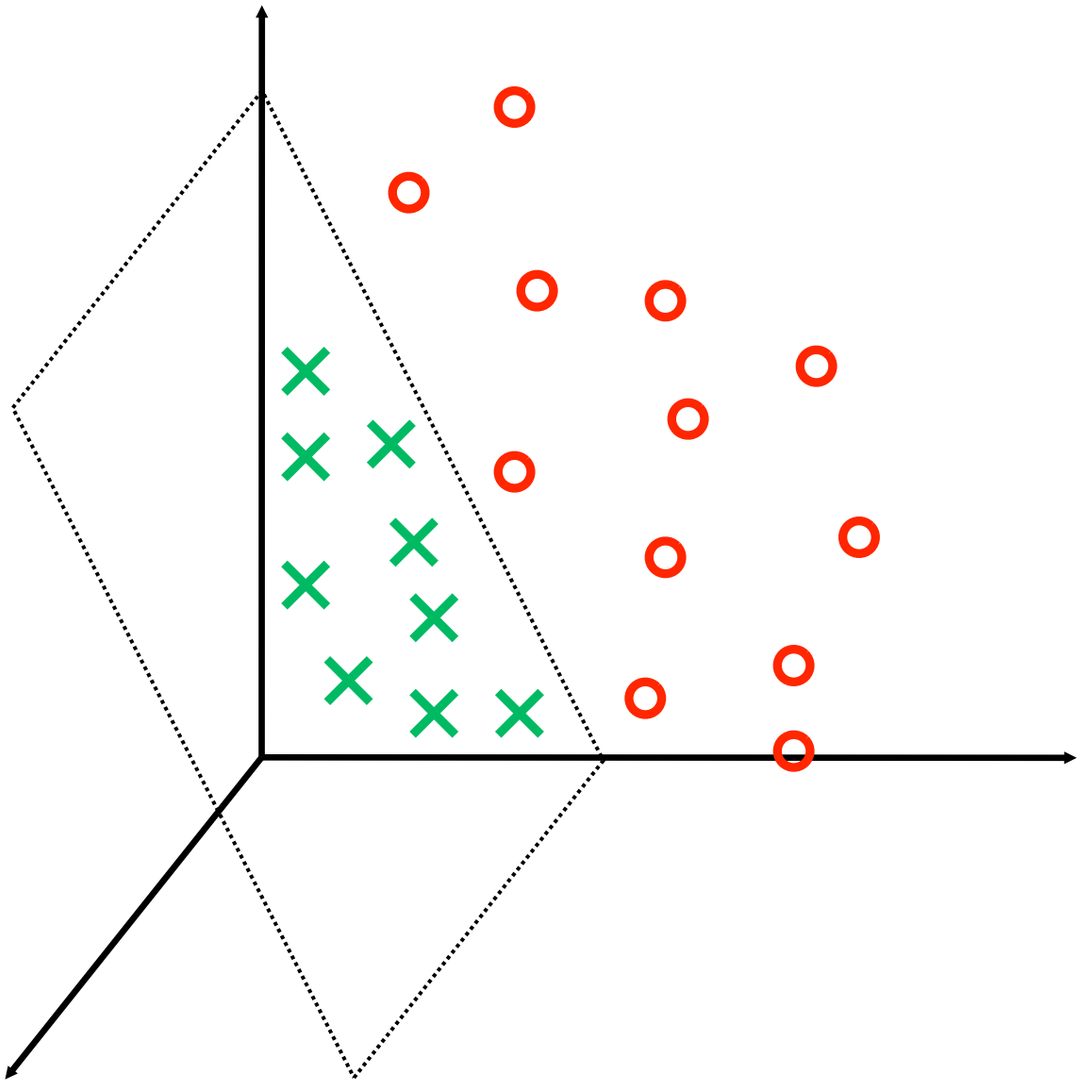
$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^d$$

Visualization of the Example

$$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$



ϕ



Decision boundary becomes a hyperplane

Original decision boundary is an ellipse

Constructing Kernels

Finding valid kernels from scratch is hard, but:

A number of rules exist to create a new valid kernel k from given kernels k_1 and k_2. For example:

$$k(\mathbf{x}_1, \mathbf{x}_2) = ck_1(\mathbf{x}_1, \mathbf{x}_2), \quad c > 0$$

$$k(\mathbf{x}_1, \mathbf{x}_2) = f(\mathbf{x}_1)k_1(\mathbf{x}_1, \mathbf{x}_2)f(\mathbf{x}_2)$$

$$k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(k_1(\mathbf{x}_1, \mathbf{x}_2)\right)$$

$$k(\mathbf{x}_1, \mathbf{x}_2) = k_1(\mathbf{x}_1, \mathbf{x}_2) + k_2(\mathbf{x}_1, \mathbf{x}_2)$$

$$k(\mathbf{x}_1, \mathbf{x}_2) = k_1(\mathbf{x}_1, \mathbf{x}_2)k_2(\mathbf{x}_1, \mathbf{x}_2)$$

$$k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T A \mathbf{x}_2$$

Examples of Valid Kernels

- Polynomial Kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + c)^d \quad c > 0 \quad d \in \mathbb{N}$$

- Gaussian Kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2)$$

- Kernel for sets:

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|}$$

- Matern kernel:

$$k(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{l} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu} r}{l} \right) \qquad r = \|\mathbf{x}_i - \mathbf{x}_j\|, \nu > 0, l > 0$$

Smoothing Kernels

A smoothing kernel κ is a function of one argument that satisfies the following properties:

$$\int \kappa(x)dx = 1 \qquad \int x\kappa(x)dx = 0 \qquad \int x^2\kappa(x)dx > 0$$

From this it follows that every symmetric pdf is a smoothing kernel.

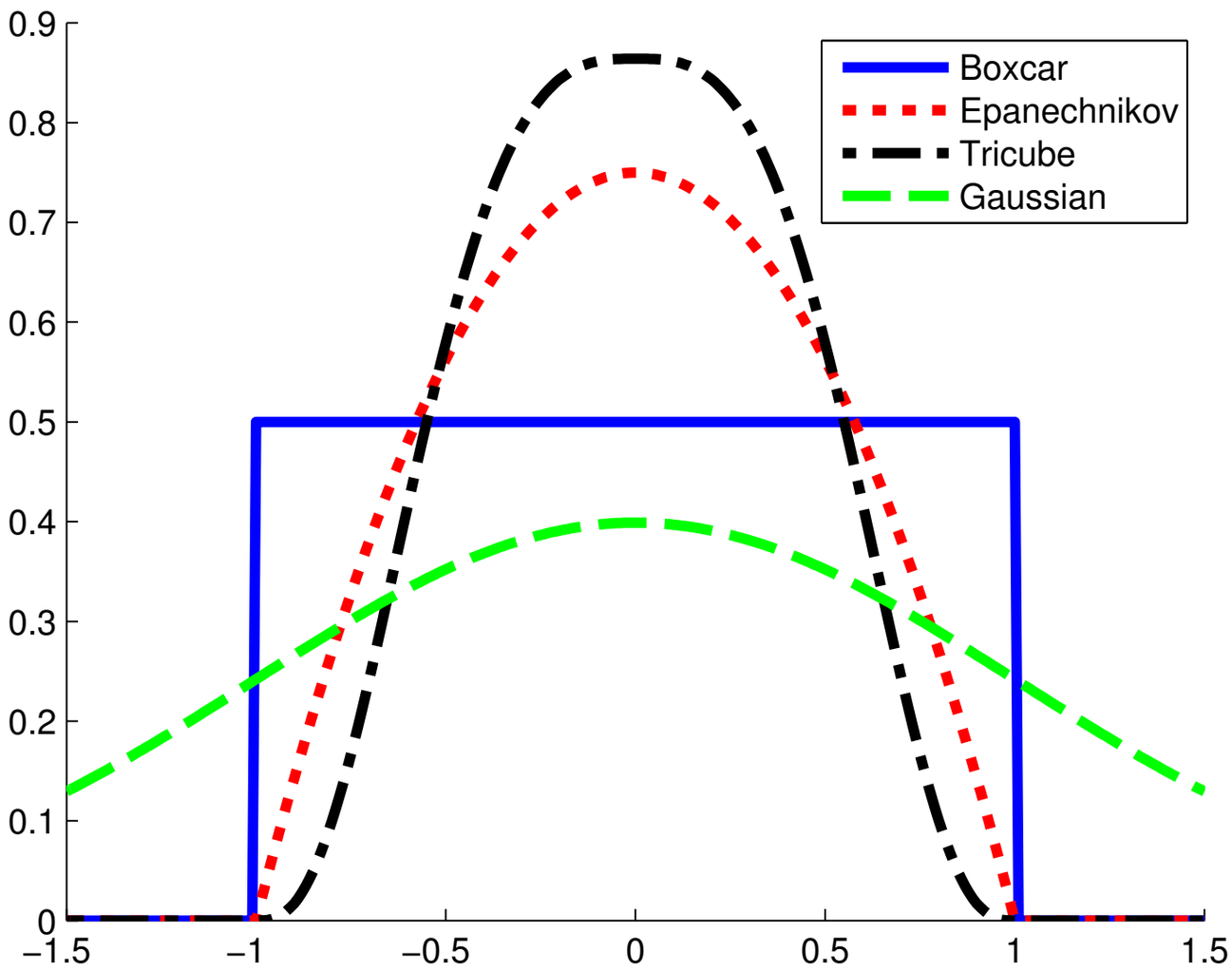
We can control the **bandwidth** h of κ:

$$\kappa_h(\mathbf{x}) = \frac{1}{h}\kappa(\frac{\|\mathbf{x}\|}{h})$$

We can build a kernel from a smoothing kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \kappa_h(\mathbf{x}_i - \mathbf{x}_j)$$

Some Smoothing Kernels



- Kernels can be smooth and non-smooth
- Kernels can have compact and non-compact support

Application Examples

Kernel Methods can be applied for many different problems, e.g.:

- Density estimation (unsupervised learning)
- Regression
- Principal Component Analysis (PCA)
- Classification

Most important Kernel Methods are

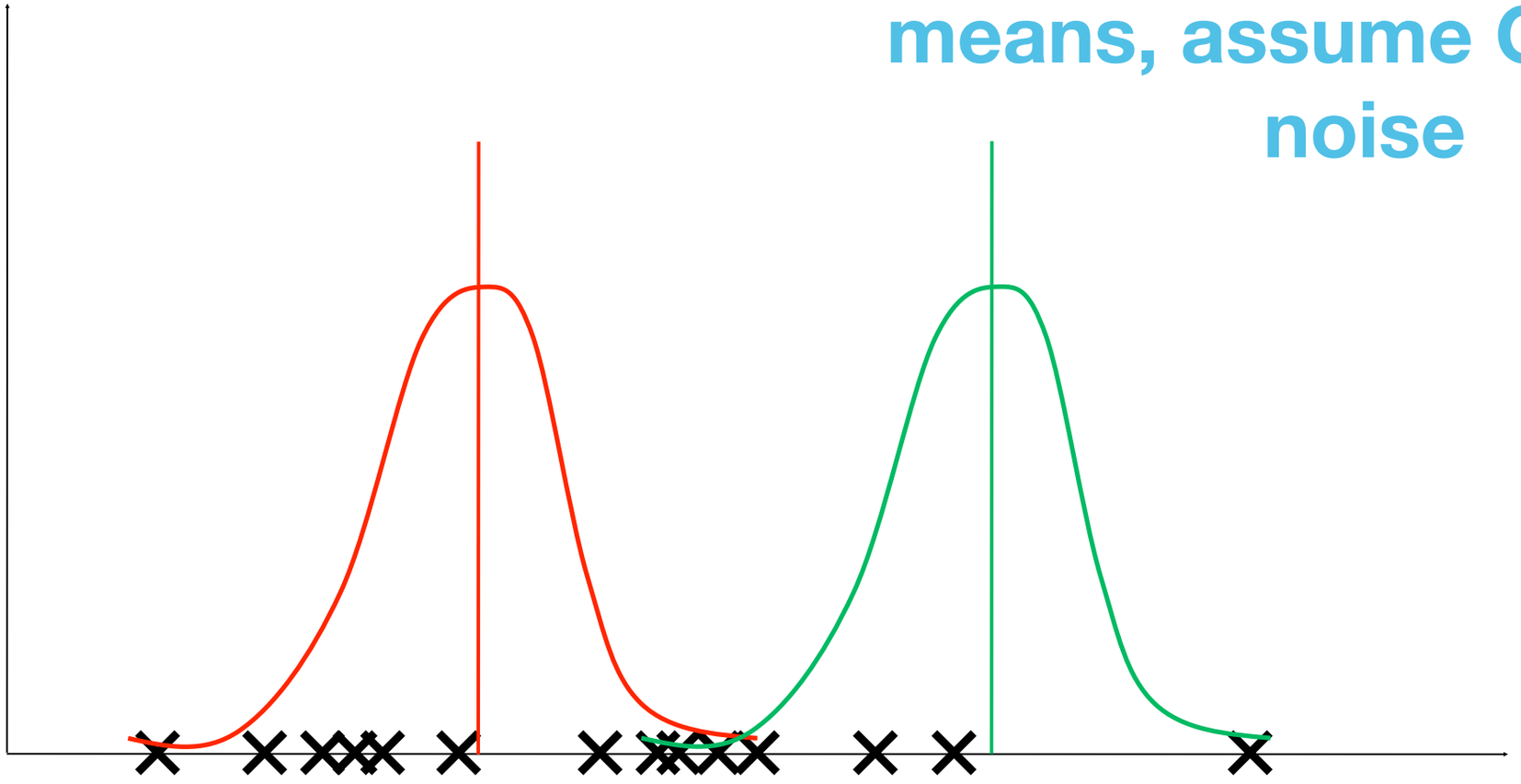
- Support Vector Machines
- Gaussian Processes

Density Estimation

Problem: Given input data (no labels), find a probabilistic model

Possible solution: K-means clustering, i.e. find an assignment from data points to K clusters iteratively where K is given

1. Initial estimate of cluster means, assume Gaussian noise

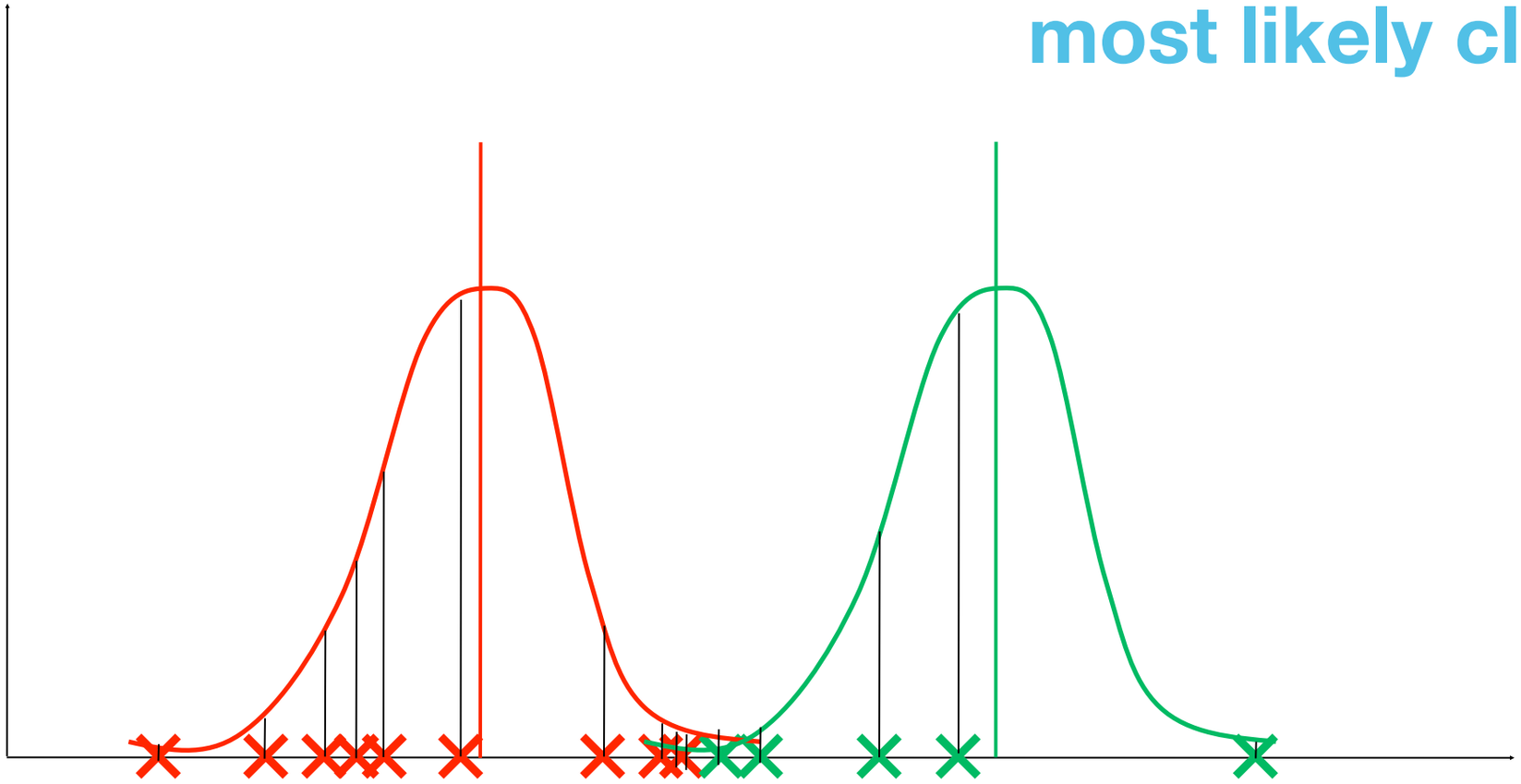


Density Estimation

Problem: Given input data (no labels), find a probabilistic model

Possible solution: K-means clustering, i.e. find an assignment from data points to K clusters iteratively where K is given

2. Assign each data point to the most likely cluster

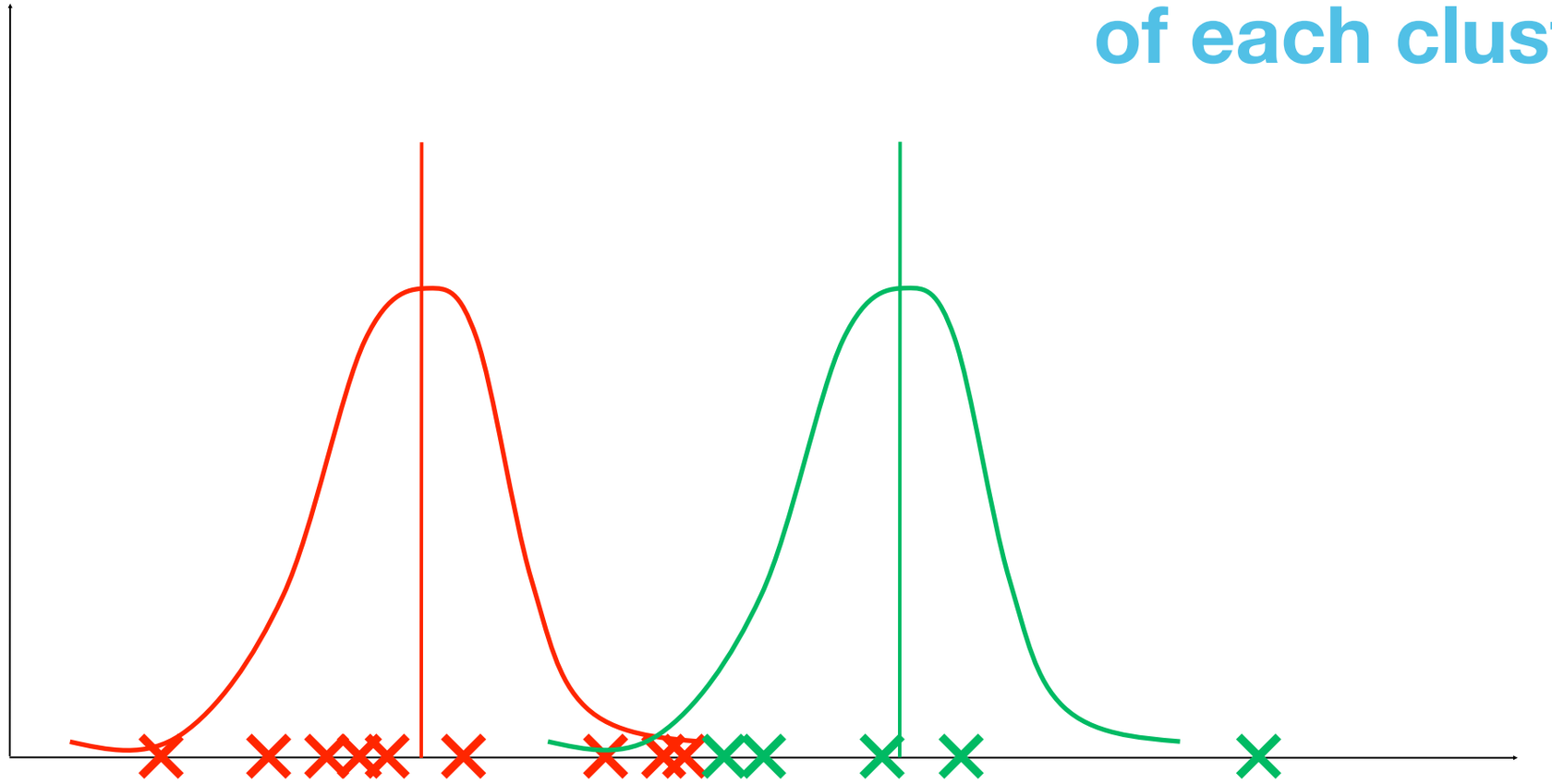


Density Estimation

Problem: Given input data (no labels), find a probabilistic model

Possible solution: K-means clustering, i.e. find an assignment from data points to K clusters iteratively where K is given

3. Recompute the mean values of each cluster

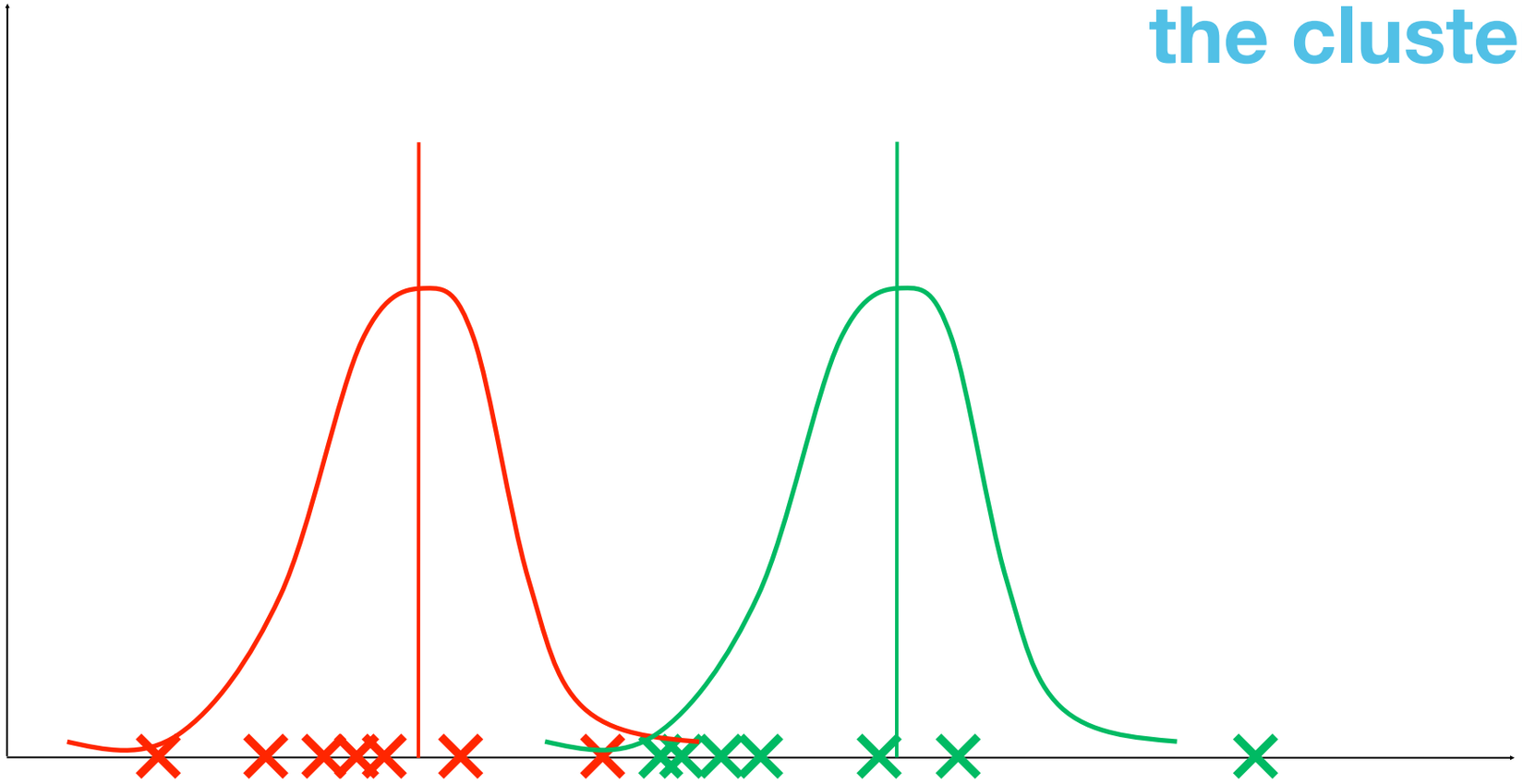


Density Estimation

Problem: Given input data (no labels), find a probabilistic model

Possible solution: K-means clustering, i.e. find an assignment from data points to K clusters iteratively where K is given

4. Reassign the data points to the clusters

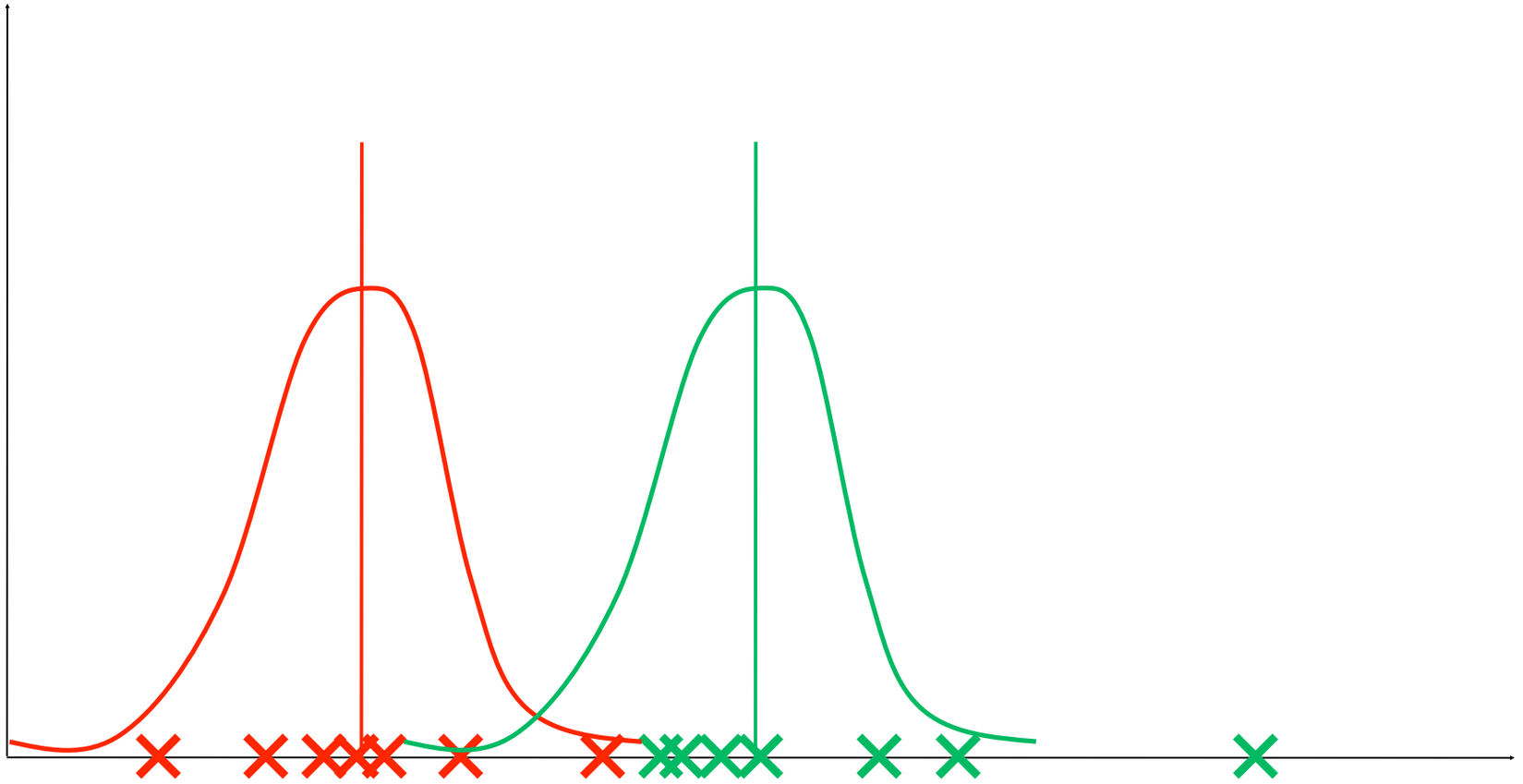


Density Estimation

Problem: Given input data (no labels), find a probabilistic model

Possible solution: K-means clustering, i.e. find an assignment from data points to K clusters iteratively where K is given

5. Iterate until convergence

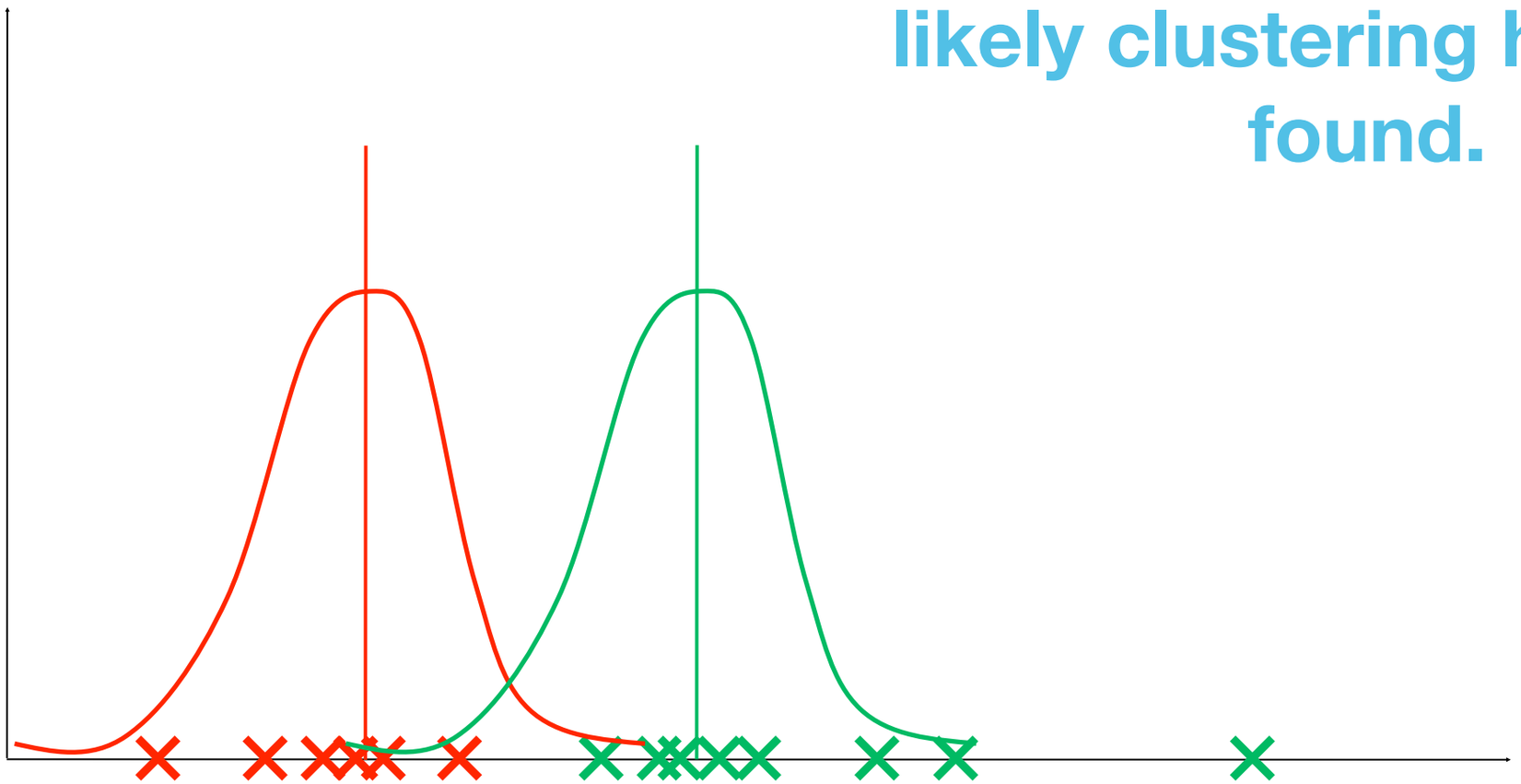


Density Estimation

Problem: Given input data (no labels), find a probabilistic model

Possible solution: K-means clustering, i.e. find an assignment from data points to K clusters iteratively where K is given

6. Convergence! The most likely clustering has been found.



K-Medoids

- We can define the cluster centers to be actual data points instead of means of data points
- The new centroids are those that minimize the sum of distances to all other cluster members

Algorithm k-medoids:

1.Initialize centroids with random subset of size K

2.repeat until convergence:

$$z_i = \arg\min_k d(i, m_k) \quad \forall i = 1, \ldots, N$$

$$m_k \leftarrow \arg\min_{i:z_i=k} \sum_{i':z_{i'}=k} d(i, i') \quad \forall k = 1, \ldots, K$$

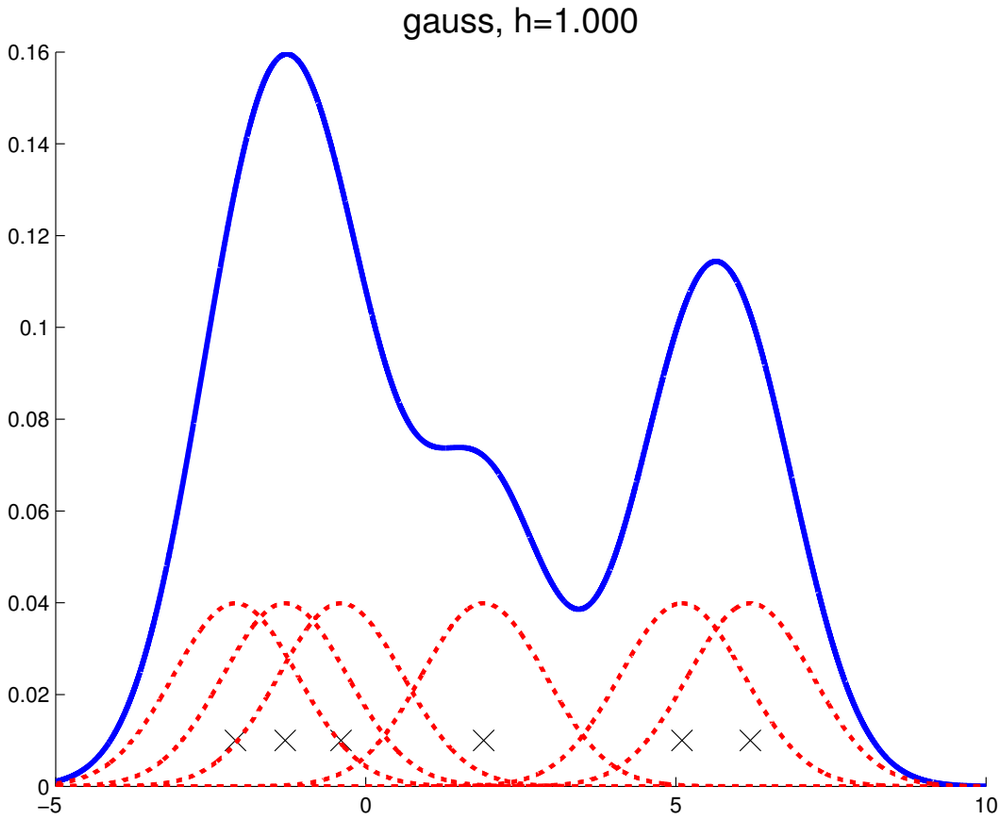
This can be kernelized, as it depends only on d!

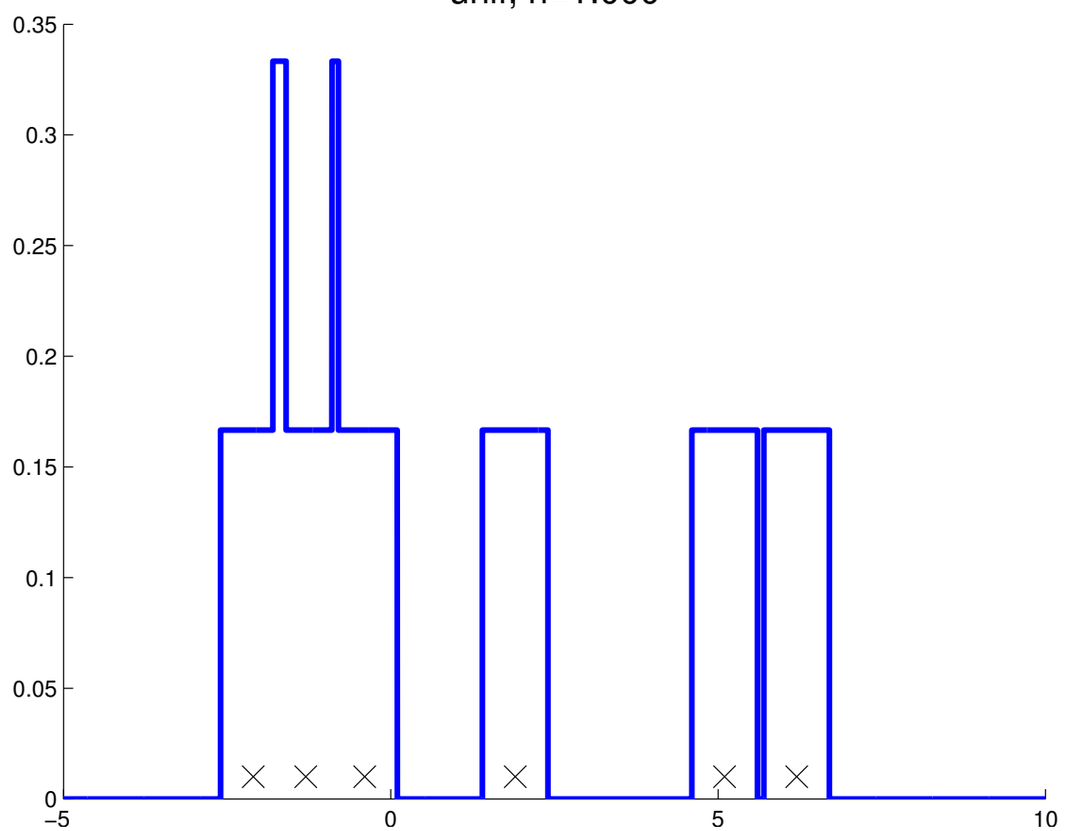
Kernel Density Estimation (KDE)

If we don’t want to specify K, we can use one cluster center per data point

The model does not have to be Gaussian:

$$p(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \kappa_h(\mathbf{x} - \mathbf{x}_i)$$





Example: Kernel Regression

$$y(\mathbf{x}) = \mathbb{E}[t \mid \mathbf{x}] = \int y \; p(y \mid \mathbf{x}) = \frac{\int y \; p(\mathbf{x}, y) dy}{\int p(\mathbf{x}, y)}$$

we can represent the joint using KDE:

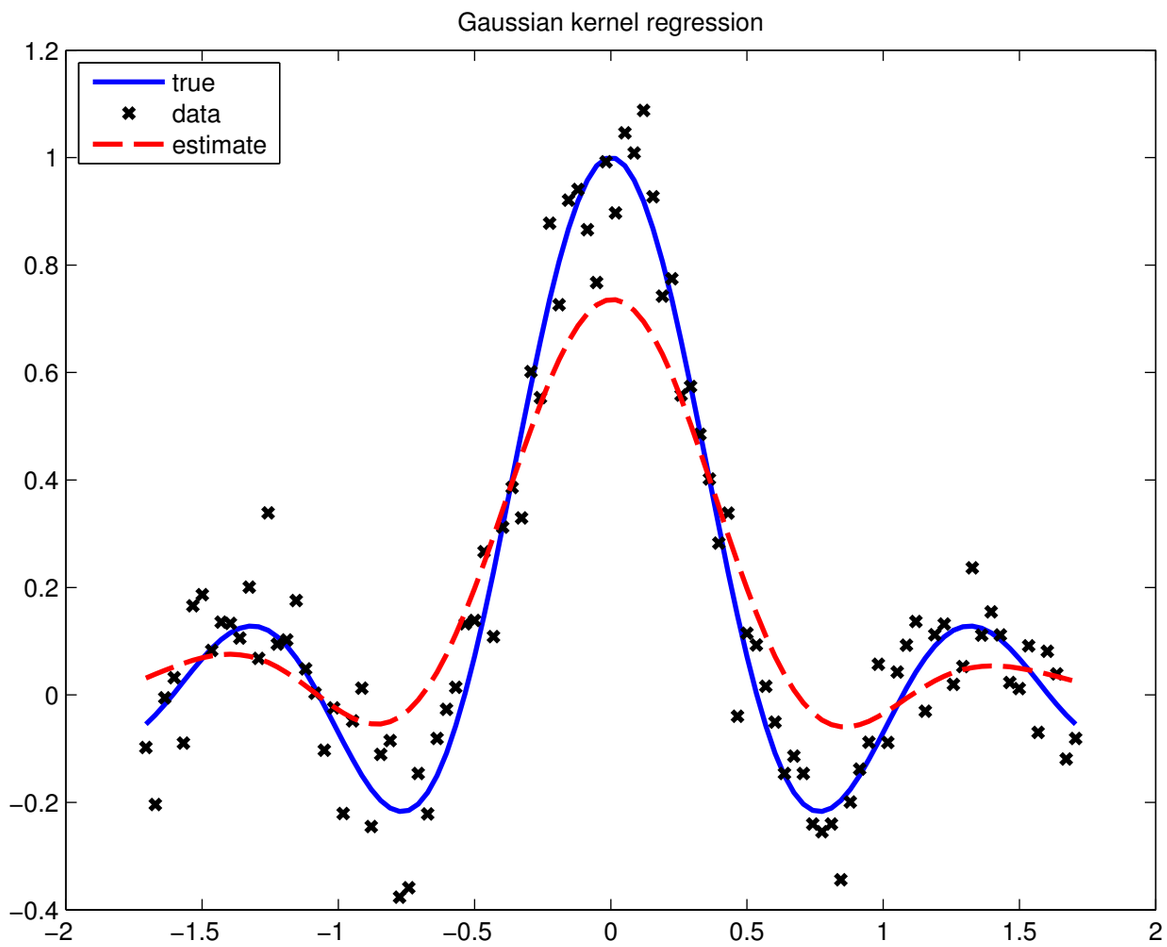
$$p(\mathbf{x}, y) \approx \frac{1}{N} \sum_{i=1}^{N} \kappa_h(\mathbf{x} - \mathbf{x}_i) \kappa_h(y - y_i)$$

which results in:

$$y(\mathbf{x}) = \frac{\sum_{i=1}^{N} \kappa_h(\mathbf{x} - \mathbf{x}_i) y_i}{\sum_{i=1}^{N} \kappa_h(\mathbf{x} - \mathbf{x}_i)}$$

This is also called the Nadaraya-Watson model.

Kernel Regression



- Example of the Nadaraya-Watson model with Gaussian kernel
- In this case, if true dist. is Gaussian then the optimal bandwidth is: $h = \left(\frac{4}{3N}\right)^{1/5} \hat{\sigma}$

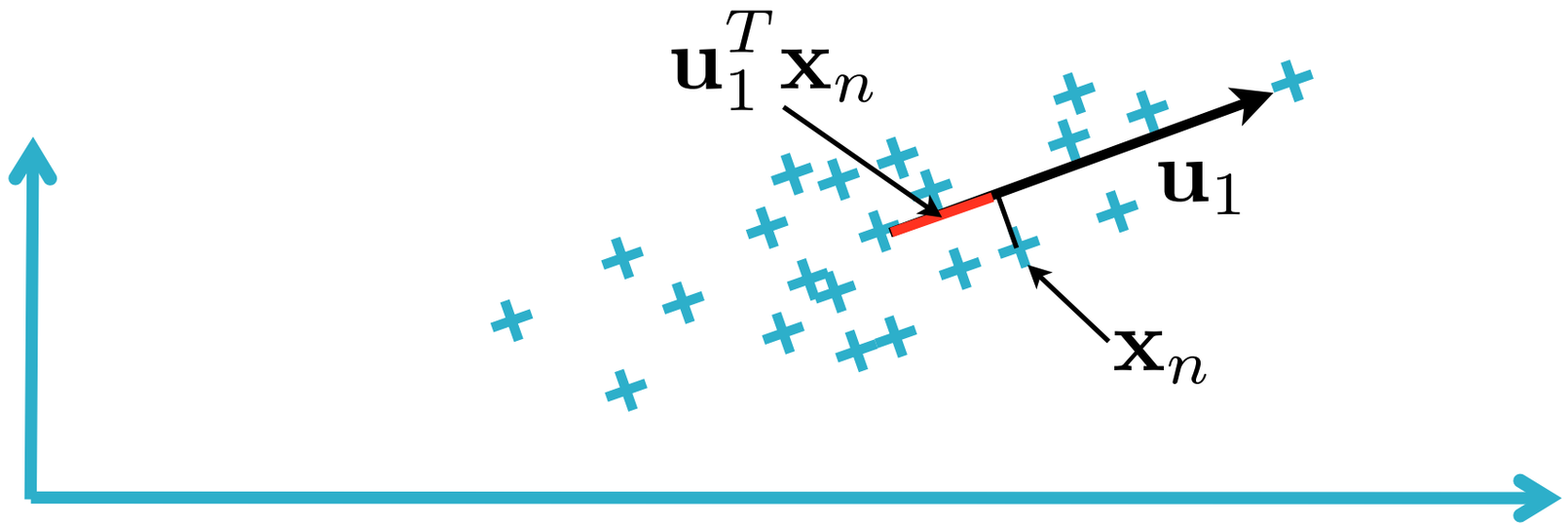
Example: Principal Component Analysis

- Given: data set $\{\mathbf{x}_n\} \quad n = 1, \ldots, N \quad \mathbf{x}_n \in \mathbb{R}^D$
- Project data onto a subspace of dimension M so that the variance is maximized ("decorrelation")
- For now: assume M is equal to 1

- Thus: the subspace can be described by a D-dimensional unit vector $\mathbf{u}_1$, i.e.: $\mathbf{u}_1^T \mathbf{u}_1 = 1$
- Each data point is projected onto the subspace using the dot product: $\mathbf{u}_1^T \mathbf{x}_n$

Principal Component Analysis

Visualization:



Mean:

$$\mu = \frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_1^T\mathbf{x}_n = \frac{1}{N}\mathbf{u}_1^T\sum_{n=1}^{N}\mathbf{x}_n = \mathbf{u}_1^T\bar{\mathbf{x}}$$

Variance:

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\bar{\mathbf{x}})^2 = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T(\mathbf{x}_n - \bar{\mathbf{x}}))^2 = \mathbf{u}_1^T\underbrace{\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T}_{S}\mathbf{u}_1$$

Principal Component Analysis

Goal: Maximize $\mathbf{u}_1^T S \mathbf{u}_1$ s.t. $\mathbf{u}_1^T \mathbf{u}_1 = 1$

Using a Lagrange multiplier:

$$\mathbf{u}^* = \arg\max_{\mathbf{u}_1} \mathbf{u}_1^T S \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$

Setting the derivative wrt. $\mathbf{u}_1$ to 0 we obtain:

$$S\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

Thus: $\mathbf{u}_1$ must be an eigenvector of S.

Multiplying with $\mathbf{u}_1^T$ from left gives: $\mathbf{u}_1^T S \mathbf{u}_1 = \lambda_1$

Thus: σ^2 is largest if $\mathbf{u}_1$ is the eigenvector of the largest eigenvalue of S

Principal Component Analysis

We can continue to find the best one-dimensional subspace that is orthogonal to $\mathbf{u}_1$

If we do this M times we obtain:

$\mathbf{u}_1, \ldots, \mathbf{u}_M$ are the eigenvectors of the M largest eigenvalues of S: $\lambda_1, \ldots, \lambda_M$

To project the data onto the M-dimensional subspace we use the dot-product:

$$\mathbf{x}^{\perp} = \begin{pmatrix} \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_M^T \end{pmatrix} (\mathbf{x} - \bar{\mathbf{x}})$$

Reconstruction using PCA

- We can interpret the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_M$ as a basis if $M = D$
- A reconstruction of a data point $\mathbf{x}$ into an M-dimensional subspace (M<D) can be written:

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} z_{ni}\mathbf{u}_i + \sum_{i=M+1}^{D} b_i\mathbf{u}_i$$

- Goal is to minimize the squared error:

$$J = \frac{1}{N}\sum_{n=1} \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2$$

- This results in:

$$z_{ni} = \mathbf{x}_n^T\mathbf{u}_i \qquad\qquad b_i = \bar{\mathbf{x}}^T\mathbf{u}_i$$

These are the coefficients of the eigenvectors

Reconstruction using PCA

Plugging in, we have:

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} (\mathbf{x}_n^T \mathbf{u}_i)\mathbf{u}_i + \sum_{i=M+1}^{D} (\bar{\mathbf{x}}^T \mathbf{u}_i)\mathbf{u}_i$$

$$= \sum_{i=1}^{D} (\bar{\mathbf{x}}^T \mathbf{u}_i)\mathbf{u}_i - \sum_{i=1}^{M} (\bar{\mathbf{x}}^T \mathbf{u}_i)\mathbf{u}_i + \sum_{i=1}^{M} (\mathbf{x}_n^T \mathbf{u}_i)\mathbf{u}_i$$

$$= \bar{\mathbf{x}} + \sum_{i=1}^{M} (\mathbf{x}_n^T \mathbf{u}_i)\mathbf{u}_i - \sum_{i=1}^{M} (\bar{\mathbf{x}}^T \mathbf{u}_i)\mathbf{u}_i$$

$$= \bar{\mathbf{x}} + \sum_{i=1}^{M} ((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i)\mathbf{u}_i$$

4. Add mean

3. Back-project

1. Substract mean

2. Project onto first M eigenvectors

Application of PCA: Face Recognition

Database

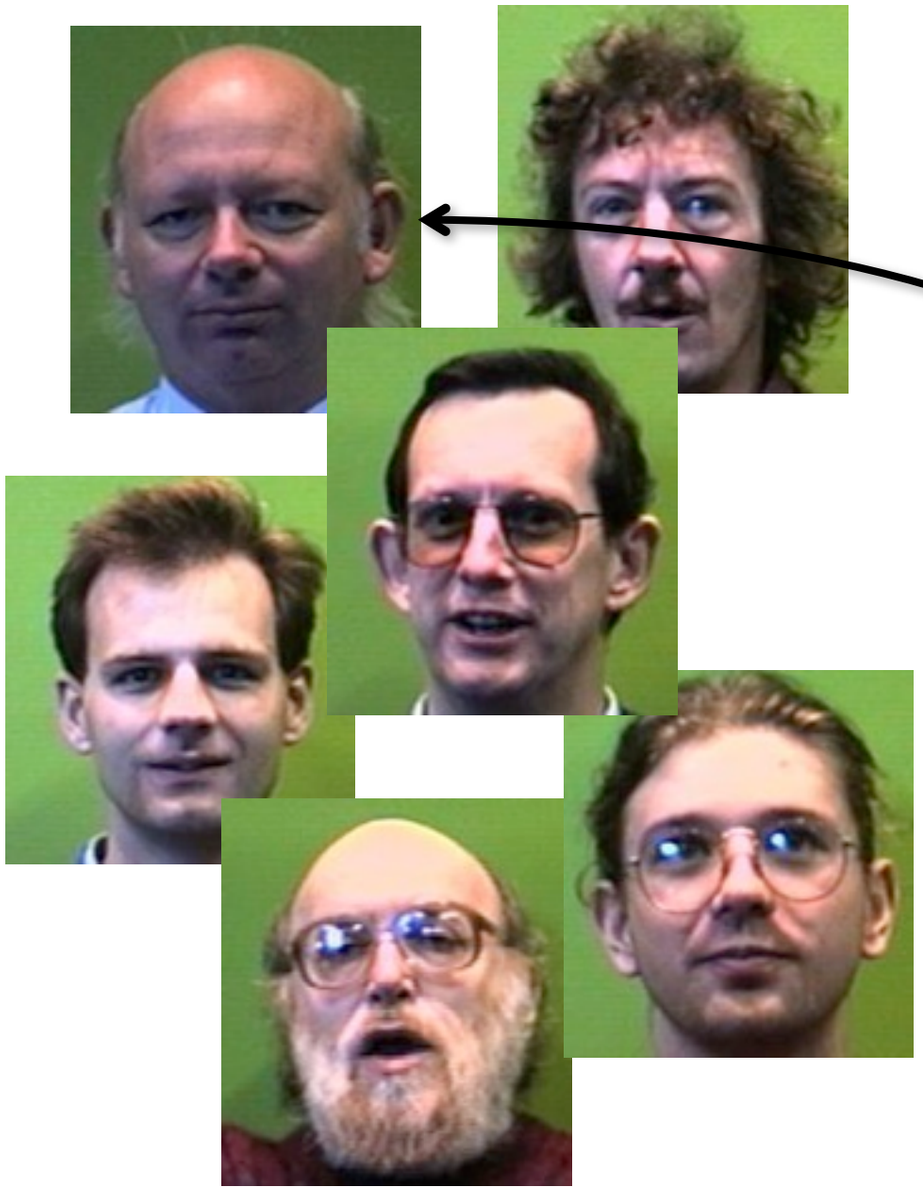


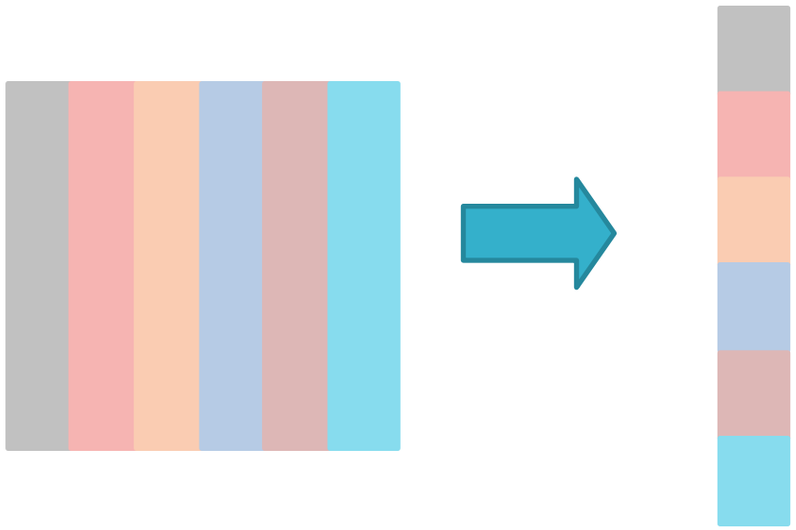
Image to identify

Identification

Application of PCA: Face Recognition

Approach:

- Convert the image into a nm vector by stacking the columns:



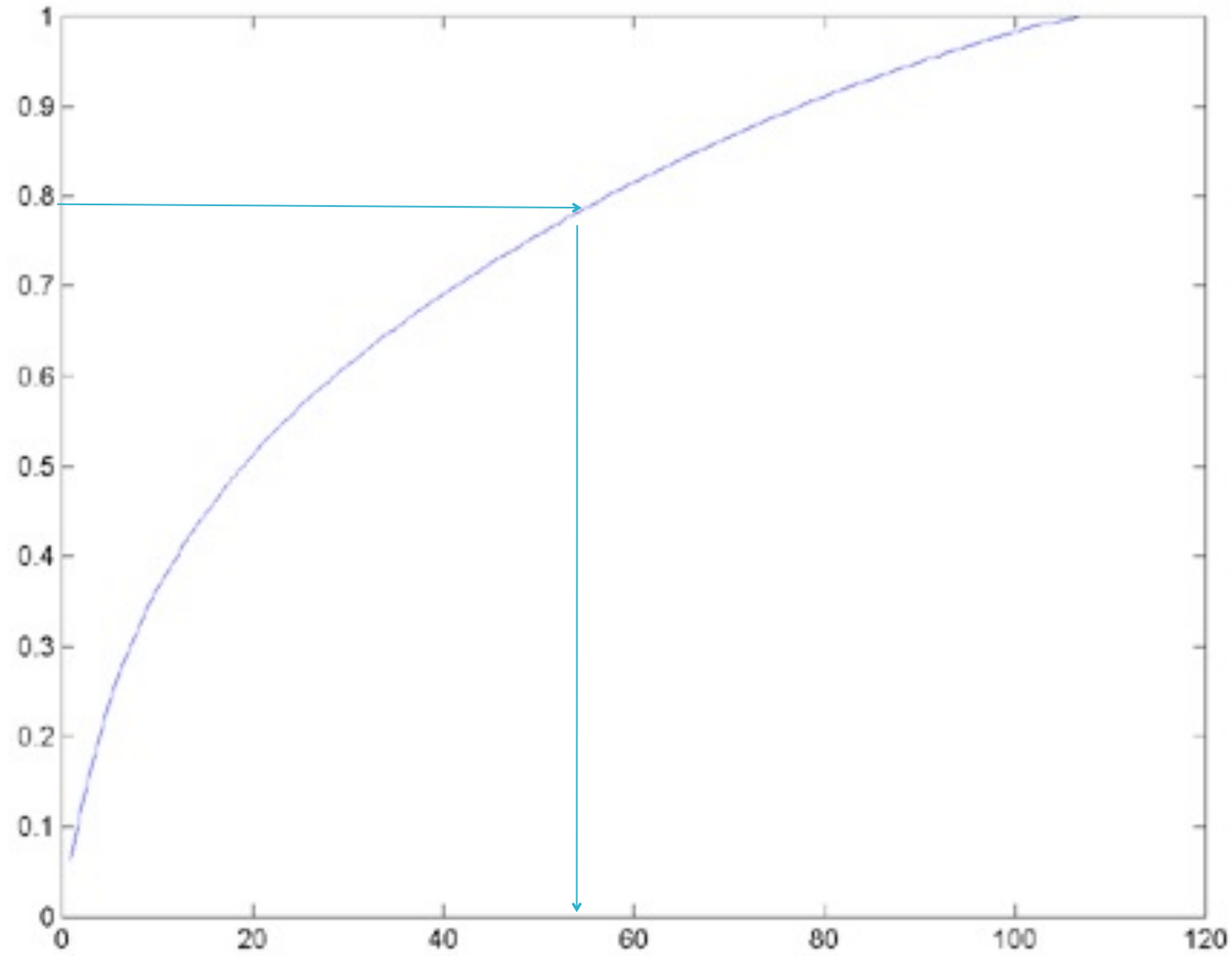
- A small image is 100x100 -> a 10000 element vector, i.e. a point in a 10000 dimension space
- Then compute covariance matrix and eigenvectors
- Select number of dimensions in subspace
- Find nearest neighbor in subspace for a new image

Results of Face Recognition

- 30% of faces used for testing, 70% for learning.

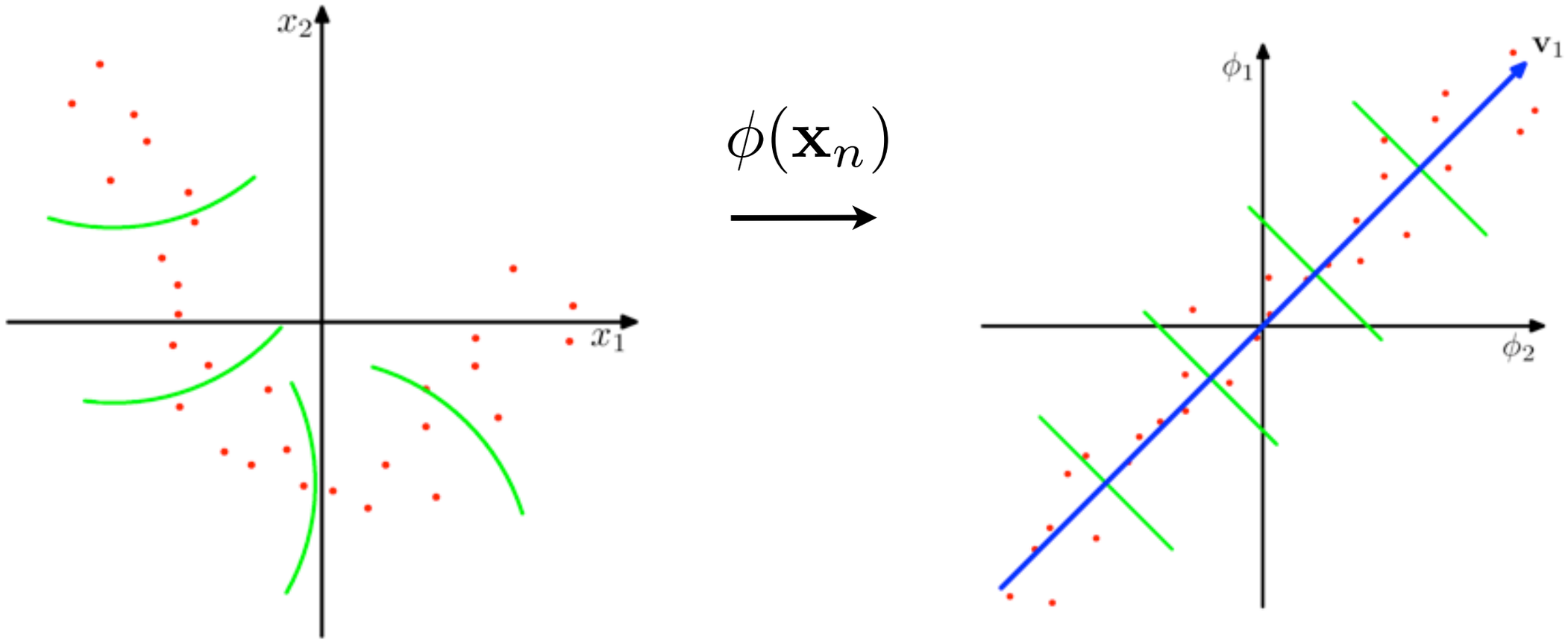
How many eigenfaces are required?

Plot: normalised cumulative sum of the eigenvalues.



About 55 eigenfaces are required to represent 80% of the information

Can We Use Kernels in PCA?



- What if data is distributed along non-linear principal components?
- **Idea:** Use non-linear kernel to map into a space where PCA can be done

Kernel PCA

Here, assume that the mean of the data is zero:

$$\sum_{n=1}^{N} \mathbf{x}_n = \mathbf{0}$$

Then, in standard PCA we have the eigenvalue problem:

$$S\mathbf{u}_i = \lambda_i \mathbf{u}_i \quad S = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T$$

Now, we use a non-linear transformation $\phi(\mathbf{x}_n)$ and we assume $\sum_{n=1}^{N} \phi(\mathbf{x}_n) = \mathbf{0}$. We define C as

$$C = \frac{1}{N}\sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T \text{ , with } \quad C\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

Goal: find eigenvalues without using features!

Kernel PCA

Plugging in:

$$\frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T\mathbf{v}_i = \lambda_i\mathbf{v}_i$$

This means, there are values a_{in} so that $\mathbf{v}_i = \sum_{i=1}^{N} a_{in}\phi(\mathbf{x}_n)$.
With this we have:

$$\frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T\sum_{m=1}^{N}a_{im}\phi(\mathbf{x}_m) = \lambda_i\sum_{i=1}^{N}a_{in}\phi(\mathbf{x}_n)$$

Multiplying both sides by $\phi(\mathbf{x}_l)$ gives:

$$\frac{1}{N}\sum_{n=1}^{N}k(\mathbf{x}_l,\mathbf{x}_n)\sum_{m=1}^{N}a_{im}k(\mathbf{x}_n,\mathbf{x}_m) = \lambda_i\sum_{i=1}^{N}a_{in}k(\mathbf{x}_l,\mathbf{x}_n)$$

where $k(\mathbf{x}_l,\mathbf{x}_n) = \phi(\mathbf{x}_l)^T\phi(\mathbf{x}_n)$. This is our expression in terms of the kernel function!

Kernel PCA

The problem can be cast as finding eigenvectors of the kernel matrix K:

$$K\mathbf{a}_i = \lambda_i N \mathbf{a}_i$$

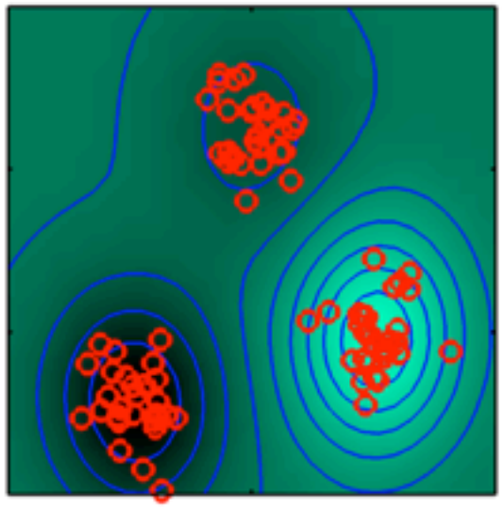
With this, we can find the projection of the image of $\mathbf{x}$ onto a given principal component as:

$$\phi(\mathbf{x})^T \mathbf{v}_i = \sum_{n=1}^{N} a_{in} \phi(\mathbf{x})^T \phi(\mathbf{x}_n) = \sum_{n=1}^{N} a_{in} k(\mathbf{x}, \mathbf{x}_n)$$

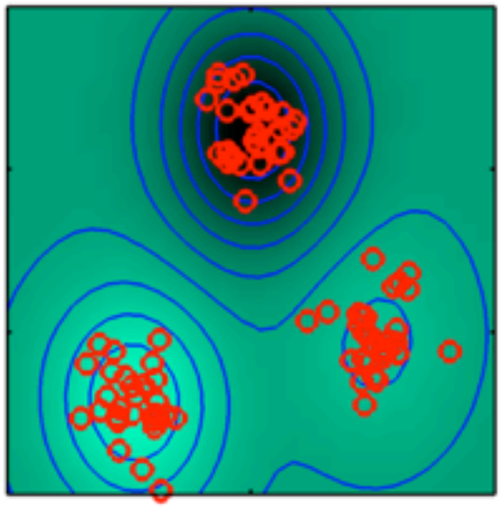
Again, this is expressed in terms of the kernel function.

Kernel PCA: Example

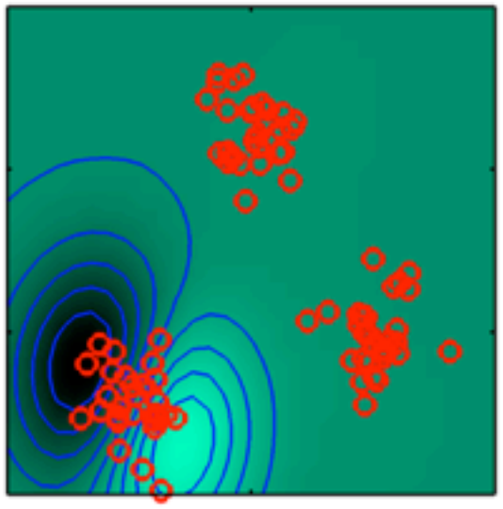
Eigenvalue=21.72



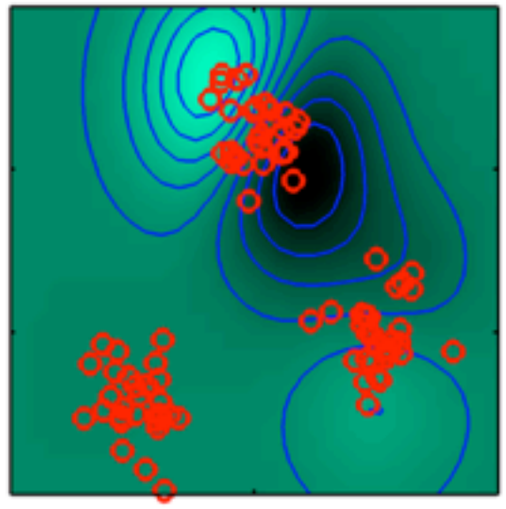
Eigenvalue=21.65



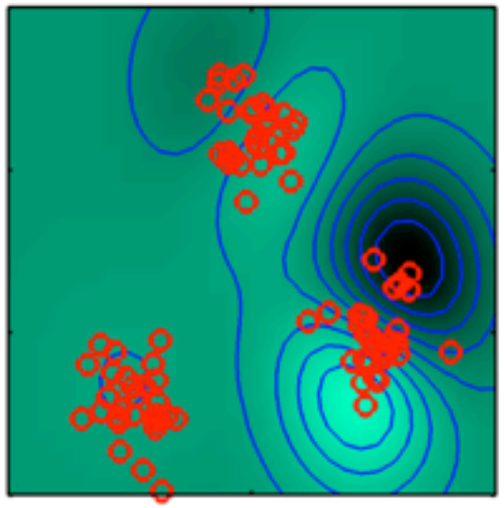
Eigenvalue=4.11



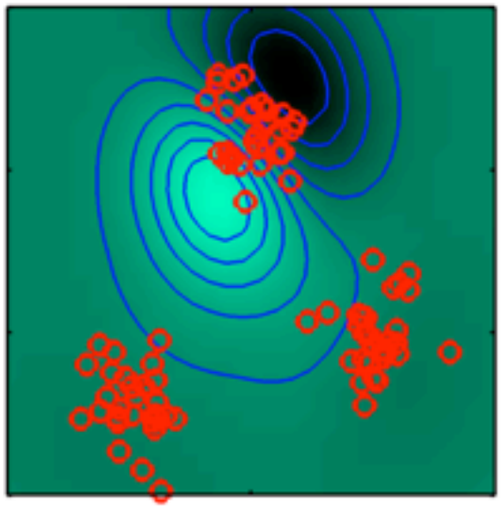
Eigenvalue=3.93



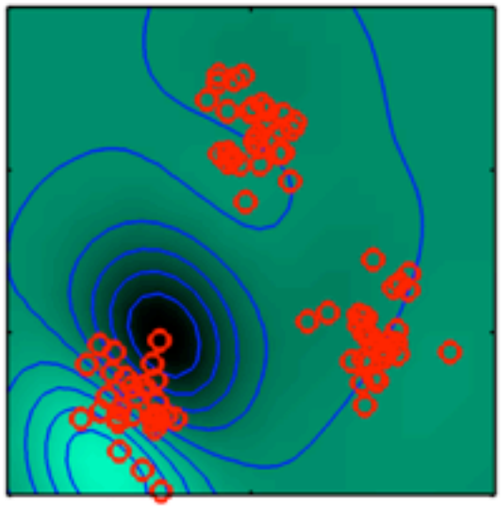
Eigenvalue=3.66



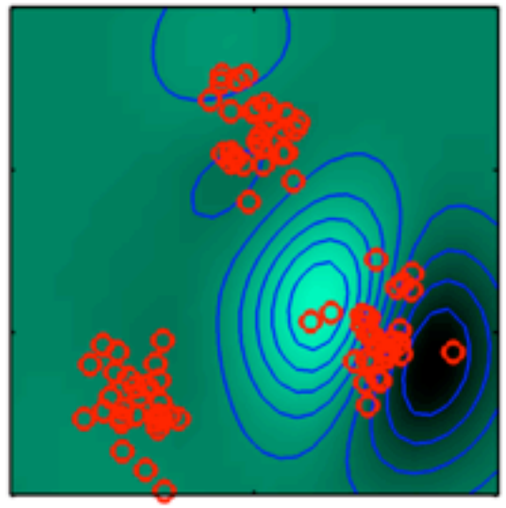
Eigenvalue=3.09



Eigenvalue=2.60



Eigenvalue=2.53



Example: Classification

- We have seen kernel methods for density estimation, PCA and regression
- For classification there are two major kernel methods: Support Vector Machines (SVMs) and Gaussian Processes
- SVMs are probably the most used classification algorithm
- Main idea: use kernelisation to map into a high-dimensional feature space, where a linear separation between the classes can be found (“hyper-plane”)

Support Vector Machines

Support Vector Machines learn a linear discriminant function (“hyper-planes”):

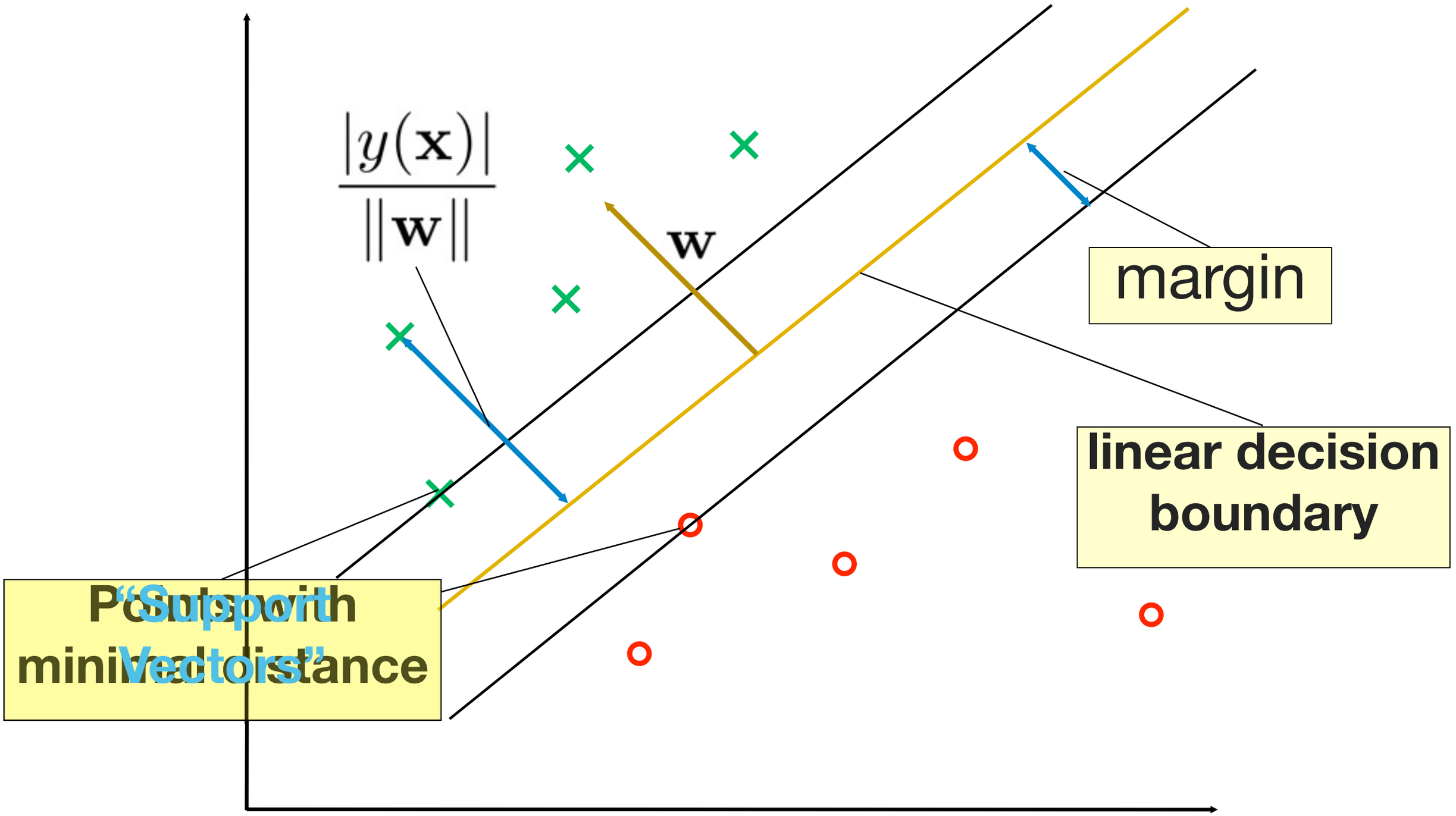
$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x}) - b$$

parameters of the hyperplane (normal vector)	feature function	data point	Bias parameter

Assumptions for now: Data is linearly separable, Binary classification ($y(\mathbf{x}, \mathbf{w}) \in \{-1, 1\}$).

“Maximum Margin”: find the decision boundary that maximizes the distance to the closest data point

Maximum Margin



Maximum Margin

- The distance of a point $\mathbf{x}_n$ to the decision hyperplane is

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

- This distance is independent of the scale of $\mathbf{w}$ and b

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

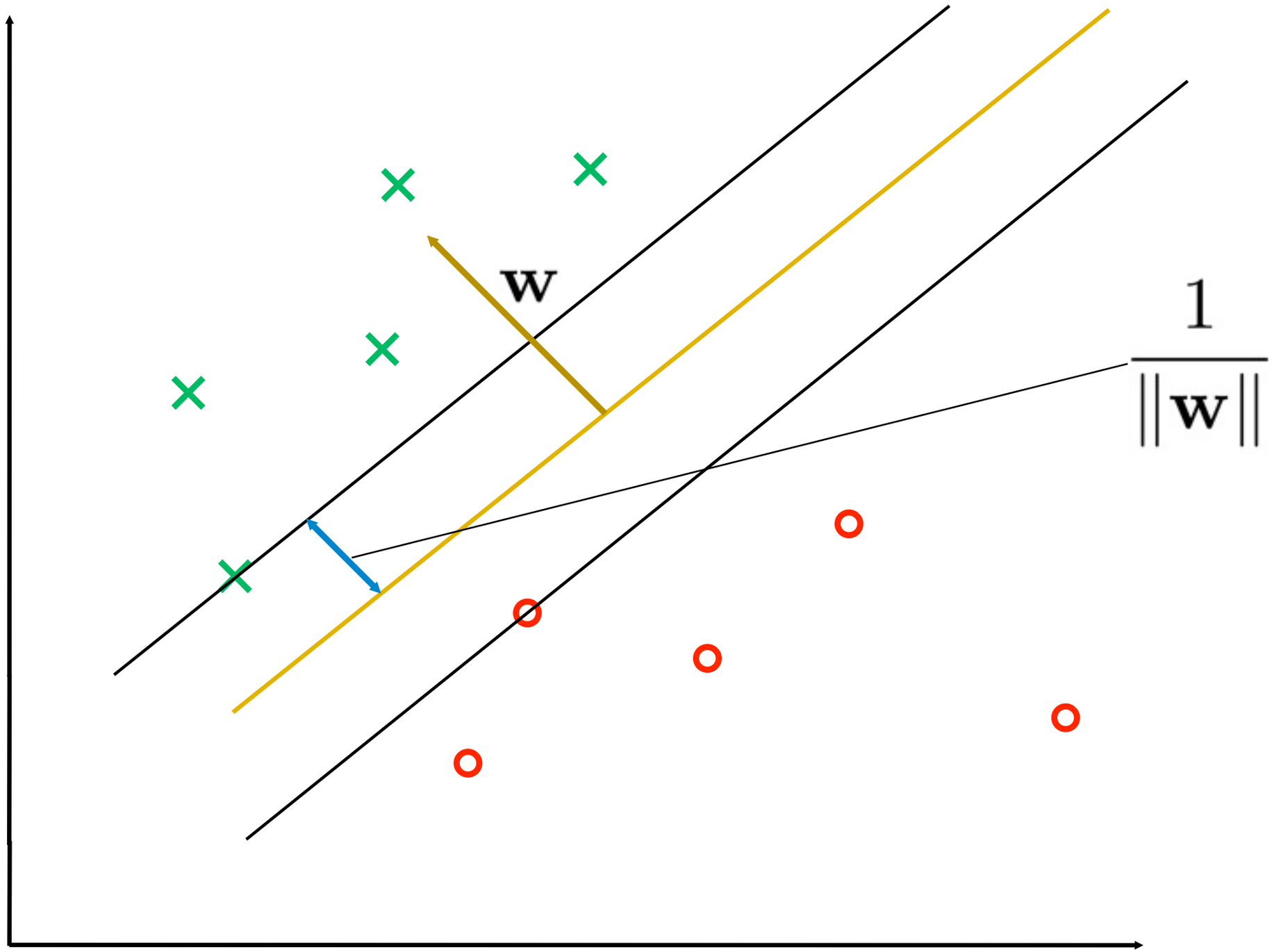
- Maximum margin is found by

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

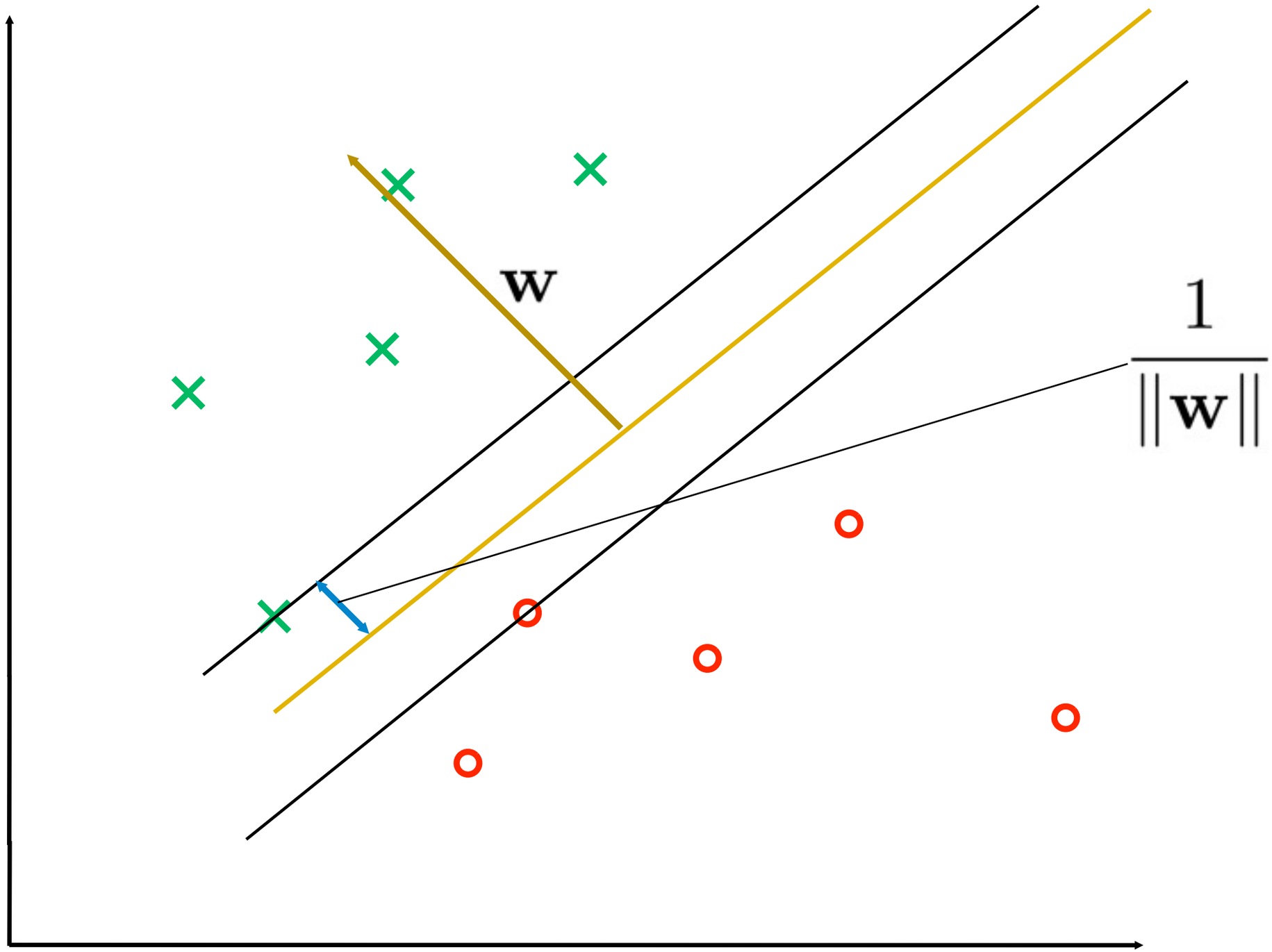
- Rescaling: We can choose α so that

$$t_n(\alpha\mathbf{w}^T\phi(\mathbf{x}_n)+\alpha b) = 1$$

Rescaling



Rescaling



Maximum Margin

For all data points we have the constraint

$$t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b) \geq 1, \qquad n = 1, \ldots, N$$

This means we have to maximize:

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \right\} \quad \text{s.th.} \quad t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b) \geq 1, \qquad n = 1, \ldots, N$$

which is equivalent to

$$\arg\min_{\mathbf{w},b} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 \right\} \quad \text{s.th.} \quad t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b) \geq 1, \qquad n = 1, \ldots, N$$

Maximum Margin

$$\arg\min_{\mathbf{w},b}\left\{\frac{1}{2}\|\mathbf{w}\|^2\right\} \quad \text{s.th.} \quad t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)\geq 1, \qquad n=1,\ldots,N$$

This is a constrained optimization problem.
It can be solved with a technique called quadratic programming.

Dual Formulation

For the constrained minimization we can introduce **Lagrange multipliers** a_n:

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

Setting the derivatives of this wrt. $\mathbf{w}$ and b to 0 yields:

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \qquad 0 = \sum_{n=1}^{N} a_n t_n$$

If we plug these constraints back into $L(\mathbf{w}, b, \mathbf{a})$:

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

Dual Formulation

subject to the constraints

$$a_n \geq 0, \qquad n = 1, \ldots, N \qquad \sum_{n=1}^{N} a_n t_n = 0$$

This is called the **dual formulation** of the constrained optimization problem. The function k is called the **kernel function** and is defined as:

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

The simplest example of a kernel function is given for Φ= I. It is also known as the **linear kernel**.

$$k(\mathbf{x}_n, \mathbf{x}_m) = \mathbf{x}_n^T \mathbf{x}_m$$

The Kernel Trick

- Other kernels are possible, e.g. the polynomial:

$$\phi(\mathbf{x}) = (x_1^2, x_2^2, x_1 x_2, x_2 x_1) \qquad \mathbf{x} \in \mathbb{R}^2$$

$$k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n^T)\phi(\mathbf{x}_m) = (\mathbf{x}^T\mathbf{x})^2$$

Kernel Trick for SVMs: If we find an optimal solution to the dual form of our constrained optimization problem, then we can replace the kernel by any other valid kernel and obtain again an optimal solution.

- Consequence: Using a non-linear feature transform Φ we obtain non-linear decision boundaries.

Observations and Remarks

- The kernel function is evaluated for each pair of training data points during training
- It can be shown that for every training data point it holds either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1$. In the latter case, they are support vectors.
- For classifying a new feature vector $\mathbf{x}$ we evaluate:

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

We only need to compute that for the support vectors

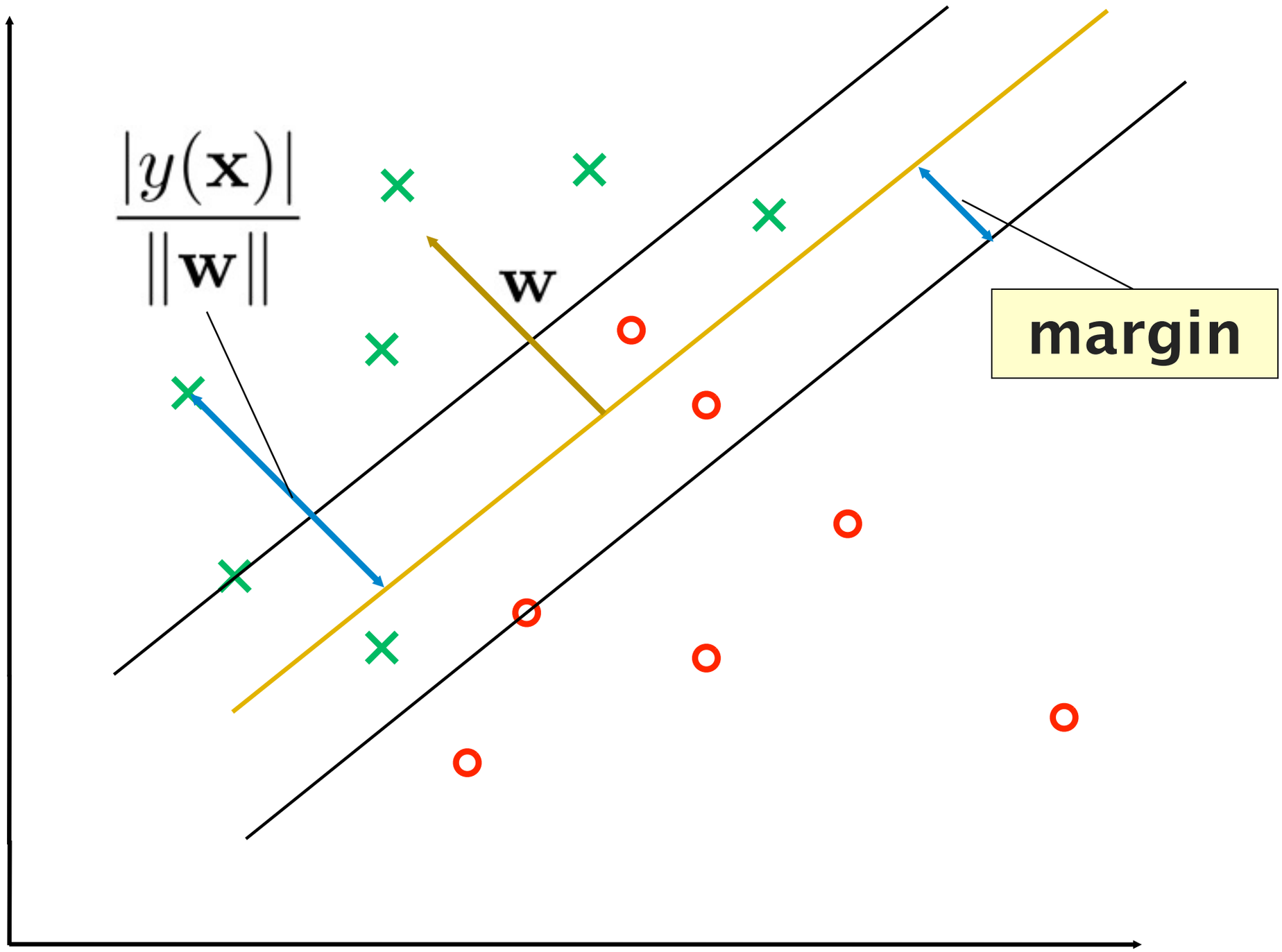
Multiple Classes

We can generalize the binary classification problem for the case of multiple classes.

This can be done with:

- one-to-many classification
- Defining a single objective function for all classes
- Organizing pairwise classifiers in a directed acyclic graph (DAGSVM)

Extension: Non-separable problems



Slack Variables

- The slack variable ξ_n is defined as follows:
- For all points on the correct side: $\xi_n = 0$
- For all other points: $\xi_n = |t_n - y(\mathbf{x}_n)|$
- This means that points with $0 < \xi_n \leq 1$ are correct classified, but inside the margin, points with $\xi_n > 1$ are misclassified.

- In the optimization, we modify the constraints:

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n, \qquad n = 1, \ldots, N$$

- and $\xi_n \geq 0$

Summary

- Kernel methods are used to solve problems by implicitly mapping the data into a (high-dimensional) feature space
- The feature function itself is not used, instead the algorithm is expressed in terms of the kernel
- Applications are manifold, including density estimation, regression, PCA and classification
- An important class of kernelized classification algorithms are Support Vector Machines
- They learn a linear discriminative function, which is called a hyper-plane
- Learning in SVMs can be done efficiently