Computer Vision Group
Prof. Daniel Cremers
Technische Universität München

6. Kernel Methods

Motivation

- Usually learning algorithms assume that some kind of feature function is given
- Reasoning is then done on a feature vector of a given (finite) length
- But: some objects are hard to represent with a fixed-size feature vector, e.g. text documents, molecular structures, evolutionary trees
- Idea: use a way of measuring similarity without the need of features, e.g. the edit distance for strings
- This we will call a **kernel function**

Dual Representation

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

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

Dual Representation

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

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

if we write this in vector form, we get

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad \mathbf{t} \in \mathbb{R}^N$$

Dual Representation

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

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

if we write this in vector form, we get

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad \mathbf{t} \in \mathbb{R}^N$$

and the solution is

$$\mathbf{w} = (\Phi^T\Phi + \lambda I_D)^{-1}\Phi^T\mathbf{t}$$

Dual Representation

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

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

$$\mathbf{w} = (\Phi^T\Phi + \lambda I_D)^{-1}\Phi^T\mathbf{t}$$

However, we can express this result in a different way using the **matrix inversion lemma:**

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

Dual Representation

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

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

$$\mathbf{w} = (\Phi^T\Phi + \lambda I_D)^{-1}\Phi^T\mathbf{t}$$

However, we can express this result in a different way using the **matrix inversion lemma:**

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

$$\mathbf{w} = \Phi^T(\Phi\Phi^T + \lambda I_N)^{-1}\mathbf{t}$$

Dual Representation

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

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

$$\mathbf{w} = (\Phi^T\Phi + \lambda I_D)^{-1}\Phi^T\mathbf{t}$$

$$\mathbf{w} = \Phi^T\underbrace{(\Phi\Phi^T + \lambda I_N)^{-1}\mathbf{t}}_{=:\mathbf{a}}$$

"Dual Variables"

Plugging $\mathbf{w} = \Phi^T\mathbf{a}$ into $J(\mathbf{w})$ gives:

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

Dual Representation

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

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

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

This is called the **dual formulation**.

Note: $\mathbf{a} \in \mathbb{R}^N \qquad \mathbf{w} \in \mathbb{R}^D$

Dual Representation

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

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

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

This is called the **dual formulation**.

The solution to the dual problem is:

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

Dual Representation

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

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

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

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

This we can use to make **predictions**:

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

(now $\mathbf{x}$ is unknown and $\mathbf{a}$ is given from training)

Dual Representation

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

where:

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

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

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

Representation using the Kernel

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

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

Theorem (Mercer): 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 “Kernel Trick”

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:

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

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

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

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

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

$$k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^TA\mathbf{x}_2$$ where A is positive semidefinite and symmetric

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:

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

A Simple Example

Define a kernel function as

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^2 \qquad \mathbf{x}, \mathbf{x}' \in \mathbb{R}^2$$

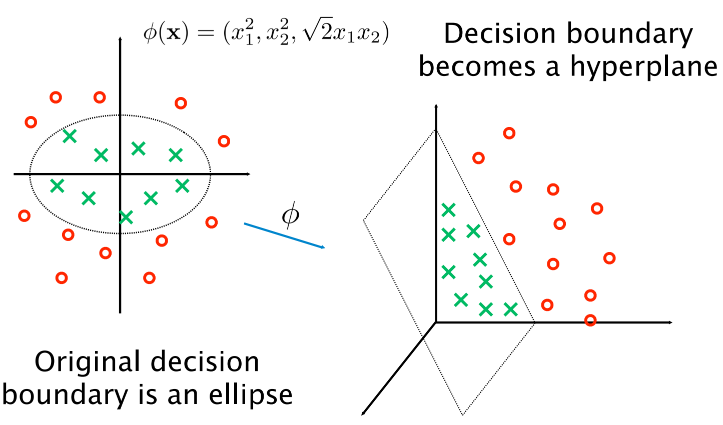
This can be written as:

$$(x_1 x_1' + x_2 x_2')^2 = x_1^2 x_1'^2 + 2 x_1 x_1' x_2 x_2' + x_2^2 x_2'^2$$
$$= (x_1^2, x_2^2, \sqrt{2} x_1 x_2)(x_1'^2, x_2'^2, \sqrt{2} x_1' x_2')^T$$
$$= \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

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


$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$
Decision boundary becomes a hyperplane
ϕ
Original decision boundary is an ellipse

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

Kernelization

- Many existing algorithms can be converted into kernel methods
- This process is called “kernelization”

Idea:

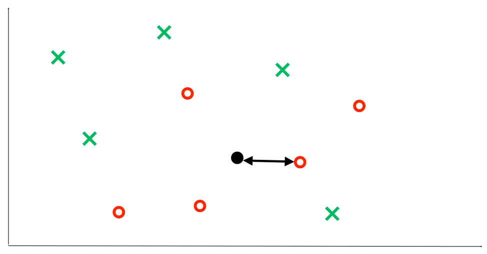
- express similarities of data points in terms of an inner product (dot product)
- replace all occurrences of that inner product by the kernel function

This is called the **kernel trick**

Example: Nearest Neighbor

- The NN classifier selects the label of the nearest neighbor in Euclidean distance

$$\|\mathbf{x}_i, \mathbf{x}_j\|^2 = \mathbf{x}_i^T \mathbf{x}_i + \mathbf{x}_j^T \mathbf{x}_j + 2 \mathbf{x}_i^T \mathbf{x}_j$$



Example: Nearest Neighbor

- The NN classifier selects the label of the nearest neighbor in Euclidean distance

$$\|\mathbf{x}_i, \mathbf{x}_j\|^2 = \mathbf{x}_i^T \mathbf{x}_i + \mathbf{x}_j^T \mathbf{x}_j + 2 \mathbf{x}_i^T \mathbf{x}_j$$

- We can now replace the dot products by a valid Mercer kernel and we obtain:

$$d(\mathbf{x}_i, \mathbf{x}_j)^2 = k(\mathbf{x}_i, \mathbf{x}_i) + k(\mathbf{x}_j, \mathbf{x}_j) + 2k(\mathbf{x}_i, \mathbf{x}_j)$$

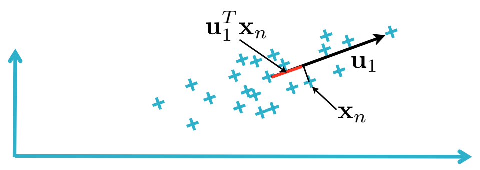
- This is a kernelized nearest-neighbor classifier
- We do not explicitly compute feature vectors!

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:

S symmetric

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

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:

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

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 $\mathbf{u}_1, \dots, \mathbf{u}_M$ 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:
$$z_{ni} = \mathbf{x}_n^T\mathbf{u}_i \qquad b_i = \bar{\mathbf{x}}^T\mathbf{u}_i$$

These are the coefficients of the eigenvectors

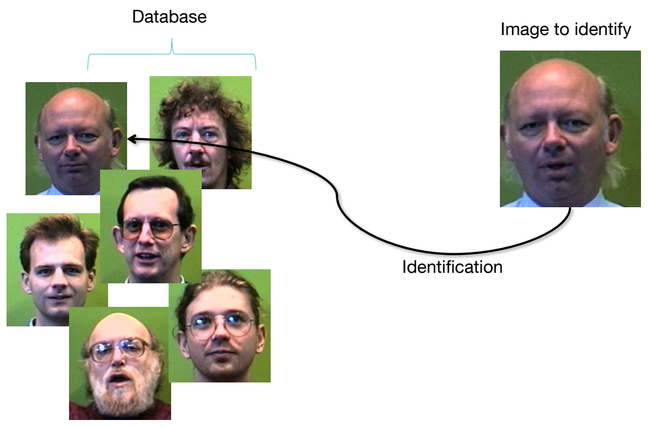
Reconstruction using PCA

Plugging in, we have:

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

4. Add mean
3. Back-project
1. Substract mean
2. Project onto first M eigenvectors

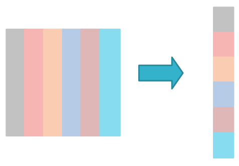
Application of PCA: Face Recognition



Application of PCA: Face Recognition

Approach:

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

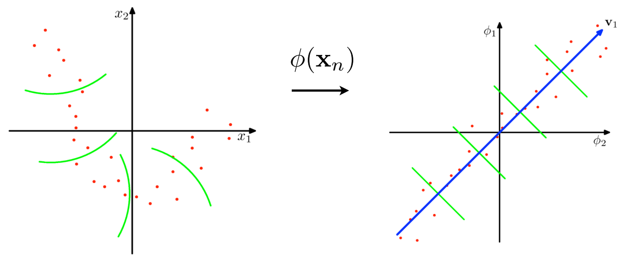


- A small image is 100x100 -> a 10000 element vector, i.e. a point in a 10000 dimension space
- Then compute covariance matrix and eigenvectors
- Select number of dimensions in subspace
- Find nearest neighbor in subspace for a new image

Results of Face Recognition

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

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:

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

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

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

Goal: find eigenvalues without using features!

Kernel PCA

Plugging in: $$\frac{1}{N}\sum_{n=1}^{N} \phi(\mathbf{x}_n)\underbrace{\phi(\mathbf{x}_n)^T \mathbf{v}_i}_{\in \mathbb{R}} = \lambda_i \mathbf{v}_i$$

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:

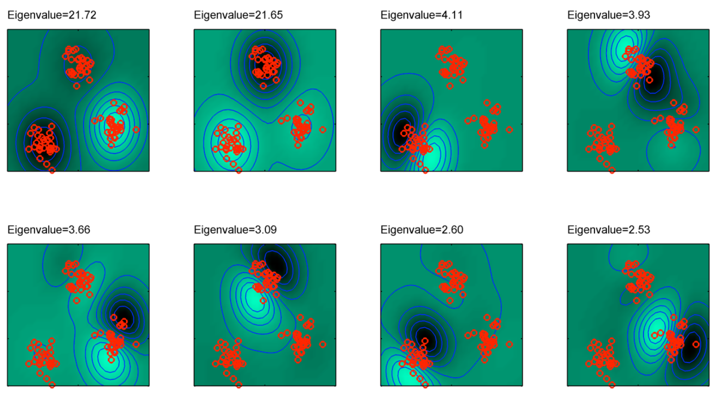
$$K\mathbf{a}_i = \lambda_i N \mathbf{a}_i$$

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



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"):

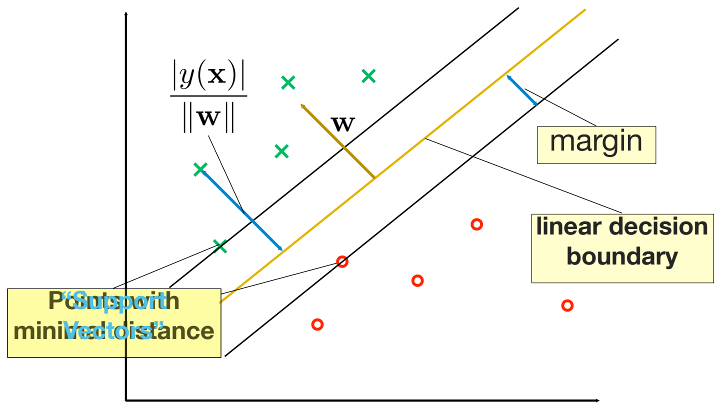
$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\phi(\mathbf{x}) - b$$

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

Assumptions for now: Data is linearly separable, Binary classification ($t_i \in \{-1; +1\}$).

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

Maximum Margin


$\frac{|y(\mathbf{x})|}{\|\mathbf{w}\|}$
$\mathbf{w}$
margin
linear decision boundary
Points with minimum distance
"Support Vectors"

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

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

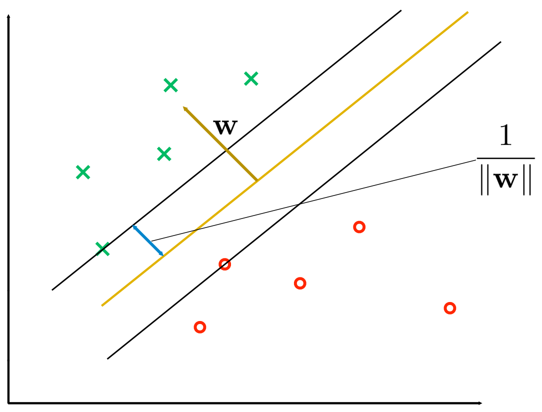
- Maximum margin is found by

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

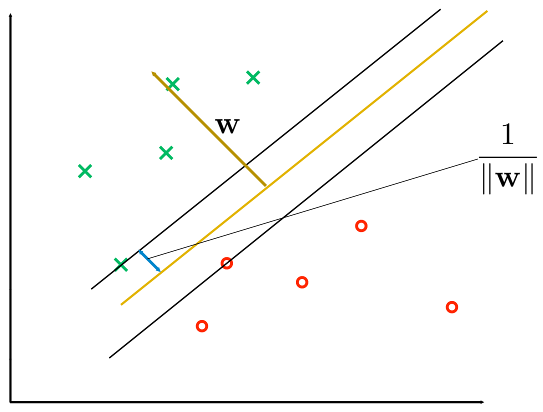
- Rescaling: We can choose α so that

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

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

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

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 again 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 in SVMs

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

$$\phi(\mathbf{x}) = (x_1^2, x_2^2, x_1x_2, x_2x_1) \qquad \mathbf{x} \in \mathbb{R}^2$$

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

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

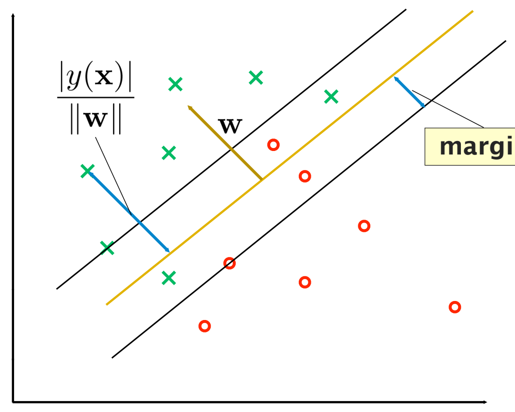
Multiple Classes

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:

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

- 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