FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
NATURWISSENSCHAFTLICHE
FAKULTÄT

Kernel Principal Component Analysis

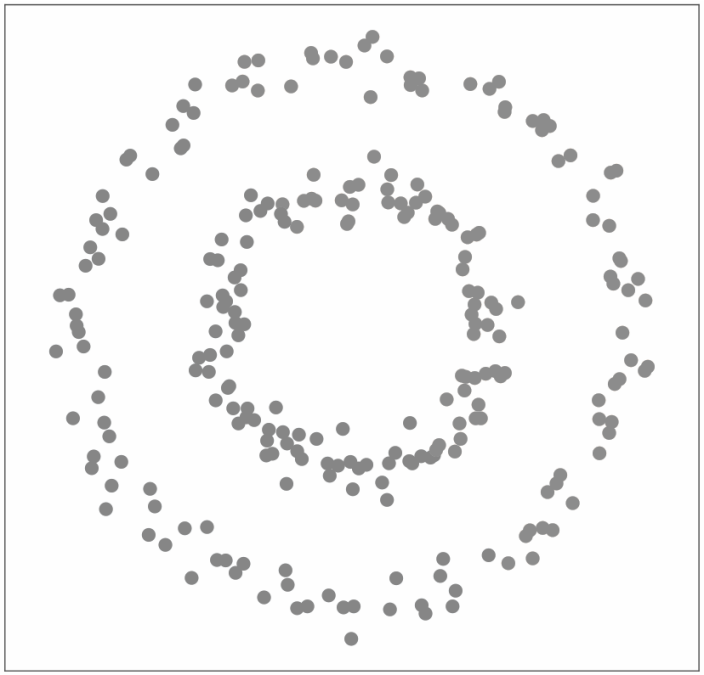
Lecture “Mathematics of Learning” 2022

Andreas Bärmann
Friedrich-Alexander-Universität Erlangen-Nürnberg

WIRTSCHAFTS
MATHEMATIK

Motivation for Extension of (linear) PCA

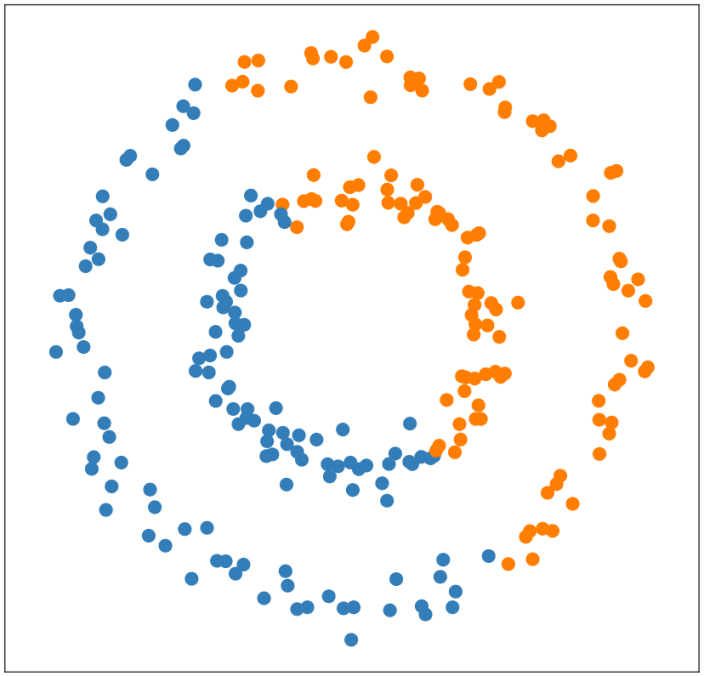
Given input data with circular structure:



do PCA in higher dimension (idea from Schölkopf, Smola, Müller, 1998)

Motivation for Extension of (linear) PCA

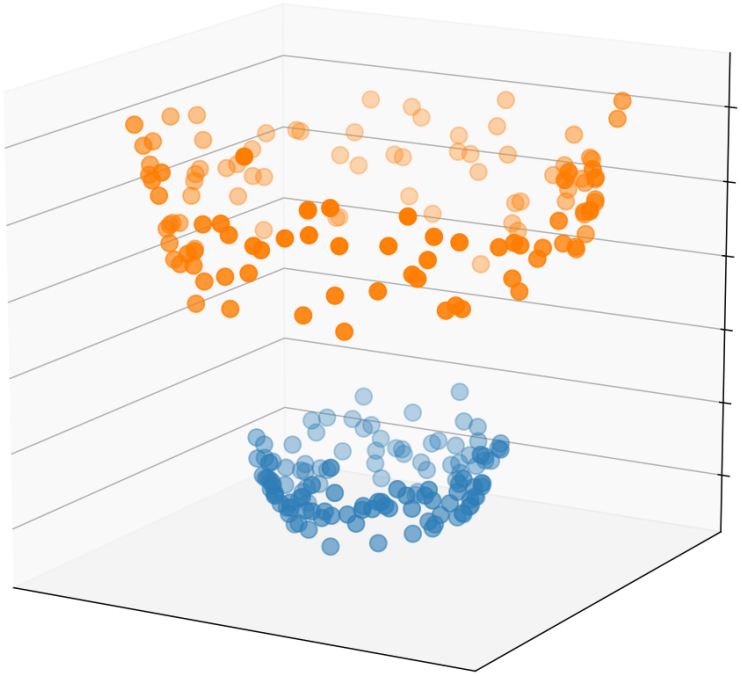
Clustering using features from linear PCA directly on data:



do PCA in higher dimension (idea from Schölkopf, Smola, Müller, 1998)

Motivation for Extension of (linear) PCA

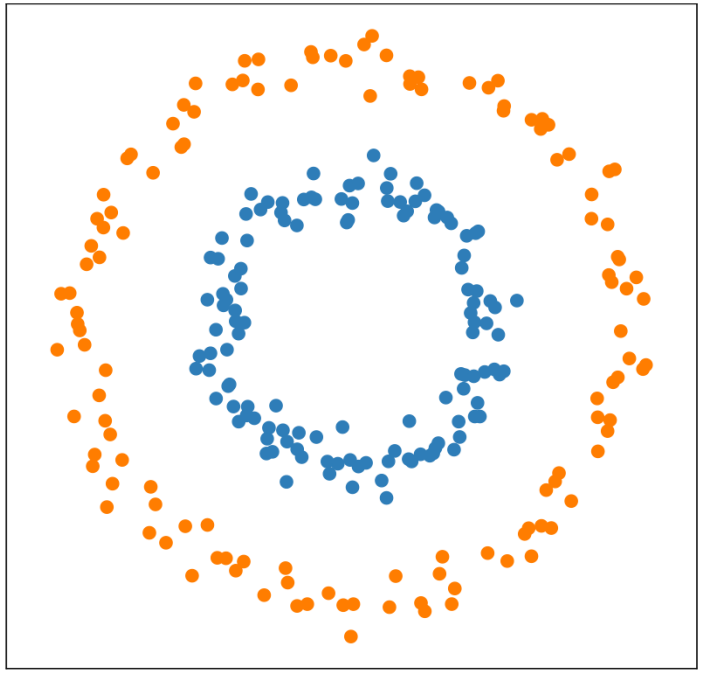
Nonlinear mapping of the data yields **linear separability in higher dimensions**:



do PCA in higher dimension (idea from Schölkopf, Smola, Müller, 1998)

Motivation for Extension of (linear) PCA

Clustering using features from linear PCA after nonlinear mapping :



do PCA in higher dimension (idea from Schölkopf, Smola, Müller, 1998)

Non-linear transformation map Ψ

Idea:

Transform the given input data $x^{(i)}, i = 1, \ldots, N$ via a map Ψ to a higher-dimensional vector space $\mathcal{H}$:

$$\begin{aligned} \Psi : \mathbb{R}^M &\to \mathcal{H} \\ x &\mapsto \Psi(x) \end{aligned}$$

Goal:

- Ψ can potentially be a **non-linear transformation map**
- Perform linear principal component analysis (PCA) in $\mathcal{H}$ e.g. for clustering. However, avoid actual work in high-dimensional $\mathcal{H}$ via *Kernel functions* to be specified later.

Aim: Efficient algorithms for performing PCA. Never calculate Ψ explicitely, i.e. assume it is given and work with it. Allows high-dimensional Ψ! Later we will explicitly give some formulas for possible 'kernel functions' that are built from Ψ.

Covariance matrix in $\mathcal{H}$

We follow Chapter 12.3 from Bishop 'Pattern Recognition and Machine Learning'

First Assume: Similar to linear PCA, transformed data $\Psi(x^{(1)}), \ldots, \Psi(x^{(N)})$ already centered in $\mathcal{H}$:

$$\sum_{i=1}^{N} \Psi(x^{(i)}) \;=\; 0.$$

Then how to compute covariance matrix?

Computation of covariance matrix $\mathbf{C}$ in $\mathcal{H}$:

$$\mathbf{C} := \frac{1}{N} \sum_{i=1}^{N} \Psi(x^{(i)}) \Psi(x^{(i)})^T$$

Recall how linear PCA works

- principal components are defined by the eigenvectors v_i of the covariance matrix via $Cv_i = \lambda_i v_i, i = 1, \ldots, M$.
- If we assume that $x^{(1)}, \ldots, x^{(N)}$ is centred, then the $M \times M$ covariance matrix C is $C = \frac{1}{N} \sum_{i=1}^{N} x^{(i)} {x^{(i)}}^T$
- eigenvectors need to be normalized such that $v_i^\top v_i = 1$.

Derivation of kernel PCA

Consider some non-linear transformation $\Psi(x)$ such that a data point x_n is projected onto a point $\Psi(x_n)$. Let us perform linear PCA in feature space. This implicitly defines a non-linear principal component model in the original space.

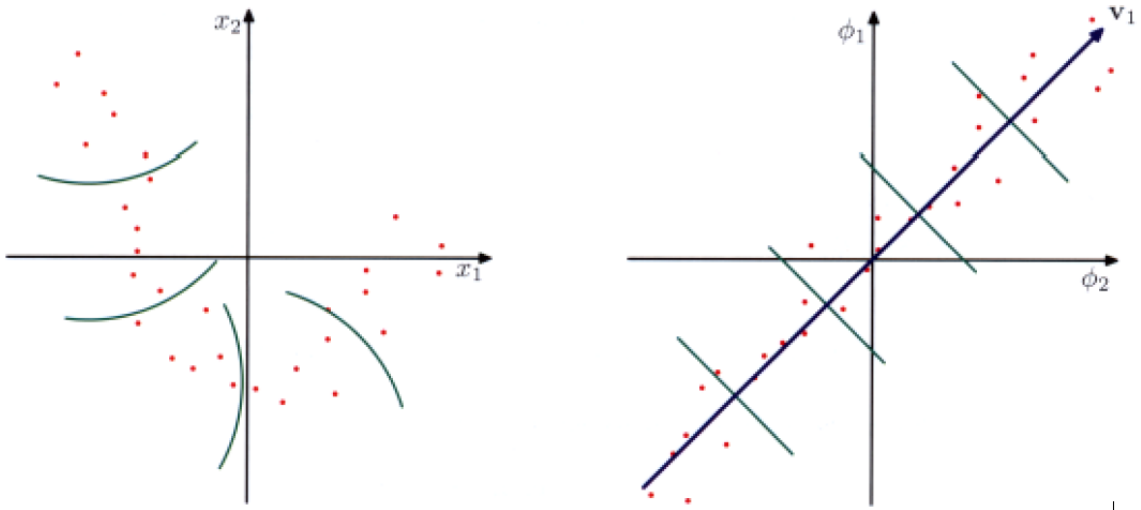


Figure 12.16 Schematic illustration of kernel PCA. A data set in the original data space (left-hand plot) is projected by a nonlinear transformation $\phi(\mathbf{x})$ into a feature space (right-hand plot). By performing PCA in the feature space, we obtain the principal components, of which the first is shown in blue and is denoted by the vector $\mathbf{v}_1$. The green lines in feature space indicate the linear projections onto the first principal component, which correspond to nonlinear projections in the original data space. Note that in general it is not possible to represent the nonlinear principal component by a vector in $\mathbf{x}$ space.

The eigenvalue problem in $\mathcal{H}$

Let us consider the covariance matrix **C** in feature space:

$$\mathbf{C} := \frac{1}{N}\sum_{i=1}^{N} \Psi(x^{(i)})\Psi(x^{(i)})^T$$

- The eigenvalue problem in $\mathcal{H}$ is given as:

$$\lambda \mathbf{v} = \mathbf{C}\mathbf{v}$$

- Our goal: Solve this eigenvalue problem without working explicitly in feature space.
- To this end, insert definition of C in eigenvector equation:

$$\frac{1}{N}\sum_{i=1}^{N} \Psi(x^{(i)})\big(\Psi(x^{(i)})^\top \mathbf{v}\big) = \lambda \mathbf{v} \tag{1}$$

- What do we see from this formula? Provided $\lambda > 0$, the left-hand side is a sum of scalars with $\Psi(x^{(i)})$. This means that each eigenvector v is *a linear combination of the* $\Psi(x^{(i)})$!

Investigating the span of the $\Psi(x^{(j)})$

This means in formulas: for an eigenvector $\mathbf{v}$ there exist $\alpha_i \in \mathbb{R}, i = 1, \ldots, N$ such that:

$$\mathbf{v} = \sum_{i=1}^{N} \alpha_i \Psi(x^{(i)}). \tag{2}$$

Substituting this expression back into the eigenvector equation (1) we obtain

$$\frac{1}{N} \sum_{i=1}^{N} \Psi(x^{(i)}) \Psi(x^{(i)})^\top \sum_{m=1}^{N} \alpha_m \Psi(x^{(m)}) = \lambda \sum_{i=1}^{N} \alpha_i \Psi(x^{(i)}).$$

There now follows a *key step*: We express this formula in terms of a *kernel function* that is of (abstract) form $k(x^i, x^m) = \Psi(x^{(i)})^\top \Psi(x^{(m)})$. (We will see later some typical kernel functions that are often used.)
We achieve this by first multiplying both sides by $\Psi(x^{(l)})^\top$. Doing this, we obtain

$$\frac{1}{N} \sum_{i=1}^{N} k(x^{(l)}, x^{(i)}) \sum_{m=1}^{N} \alpha_m k(x^{(i)}, x^{(m)}) = \lambda \sum_{i=1}^{N} \alpha_i k(x^{(l)}, x^{(i)}).$$

Investigating the span of the $\Psi(x^{(j)})$

This means in formulas: for an eigenvector $\mathbf{v}$ there exist $\alpha_i \in \mathbb{R}, i = 1, \ldots, N$ such that:

$$\mathbf{v} = \sum_{i=1}^{N} \alpha_i \Psi(x^{(i)}). \tag{2}$$

Substituting this expression back into the eigenvector equation (1) we obtain

$$\frac{1}{N} \sum_{i=1}^{N} \Psi(x^{(i)}) \Psi(x^{(i)})^\top \sum_{m=1}^{N} \alpha_m \Psi(x^{(m)}) = \lambda \sum_{i=1}^{N} \alpha_i \Psi(x^{(i)}).$$

There now follows a *key step*: We express this formula in terms of a *kernel function* that is of (abstract) form $k(x^i, x^m) = \Psi(x^{(i)})^\top \Psi(x^{(m)})$. (We will see later some typical kernel functions that are often used.)
We achieve this by first multiplying both sides by $\Psi(x^{(l)})^\top$. Doing this, we obtain

$$\frac{1}{N} \sum_{i=1}^{N} k(x^{(l)}, x^{(i)}) \sum_{m=1}^{N} \alpha_m k(x^{(i)}, x^{(m)}) = \lambda \sum_{i=1}^{N} \alpha_i k(x^{(l)}, x^{(i)}).$$

Kernel Approach

We repeat the last formula from the previous slide:

$$\frac{1}{N}\sum_{i=1}^{N} k(x^{(l)}, x^{(i)}) \sum_{m=1}^{N} \alpha_m k(x^{(i)}, x^{(m)}) = \lambda \sum_{i=1}^{N} \alpha_i k(x^{(l)}, x^{(i)}).$$

Kernel Approach

We repeat the last formula from the previous slide:

$$\frac{1}{N}\sum_{i=1}^{N} k(x^{(l)}, x^{(i)}) \sum_{m=1}^{N} \alpha_m k(x^{(i)}, x^{(m)}) = \lambda \sum_{i=1}^{N} \alpha_i k(x^{(l)}, x^{(i)}).$$

We write it easier in matrix notation as

$$\mathbf{K}^2\alpha = \lambda N \mathbf{K}\alpha \tag{3}$$

where α is an N-dimensional column vector of the $\alpha_i, i = 1, \ldots, N$, and $\mathbf{K} = (k(x^{(i)}, x^{(j)}))_{i,j}$ is the *kernel matrix*.

Important Observation

We now can solve for the α's by solving the following eigenvalue problem:

$$\mathbf{K}\alpha = \lambda N \alpha \tag{4}$$

We note we have removed a factor of $\mathbf{K}$ from both sides of (3). (We note: this only changes the solutions by eigenvectors of $\mathbf{K}$ having zero eigenvalues. This does not affect the principal components projection. We will prove this formally in the exercises.)

Normalization of Eigenvectors

We are almost done, but we need to make sure that the eigenvectors in feature space are normalized. We achieve this by enforcing the condition

$$1 = v^\top v = \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j \Psi(x^{(i)})^\top \Psi(x^{(j)}) = \alpha^\top \mathbf{K} \alpha = \lambda N \alpha^\top \alpha. \tag{5}$$

(The last equation follows from (4), please double-check!)).
Finally, the last thing to do: The goal was to be able to compute the projection z of a data point x onto the eigenvector v. This can also be done via a *kernel*:

$$z = \Psi(x)^\top v = \sum_{i=1}^{N} \alpha_i \Psi(x)^\top \Psi(x^{(i)}) = \sum_{i=1}^{N} \alpha_i k(x, x^i)$$

which is an expression in the kernel function only.

How to achieve a zero mean in projected data?

Typically, this mean is not zero. In addition, we cannot simply compute the mean and subtract it, as we want to avoid working in feature space.
Again, the kernel gives us a solution to this problem:

Let a projected data point after centralizing be denoted by $\widetilde{\Psi(x^{(i)})}$:

$$\widetilde{\Psi(x)} = \Psi(x) - \frac{1}{N}\sum_{l=1}^{N}\Psi(x^{(l)})$$

The elements of the Kernel matrix are then given by

$$\begin{aligned}
\widetilde{K}_{ij} &= \widetilde{\Psi(x^i)}^\top \widetilde{\Psi(x^j)} \\
&= \Psi(x^i)^\top\Psi(x^j) - \frac{1}{N}\sum_{l=1}^{N}\Psi(x^i)^\top\Psi(x^l) \\
&\quad - \frac{1}{N}\sum_{l=1}^{N}\Psi(x^j)^\top\Psi(x^l) + \frac{1}{N^2}\sum_{l=1}^{N}\sum_{m=1}^{N}\Psi(x^l)^\top\Psi(x^m) \\
&= k(x^i,x^j) - \frac{1}{N}\sum_{l=1}^{N}k(x^l,x^j) - \frac{1}{N}\sum_{l=1}^{N}k(x^i,x^l) + \frac{1}{N^2}\sum_{m=1}^{N}\sum_{l=1}^{N}k(x^m,x^l).
\end{aligned}$$

The Advantage of Using Kernels

Let 1_N denote the $N \times N$ matrix where every element takes value $\frac{1}{N}$.
We write the last formula in matrix notation:

$$\widetilde{K} = (K - 1_N K - K 1_N + 1_N K 1_N)$$

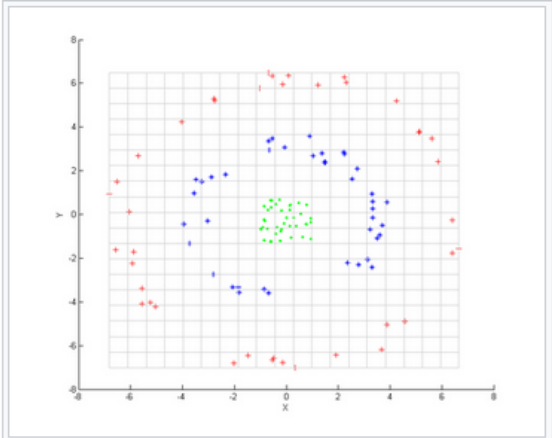
Therefore: If we define an appropriate kernel function $k(x, x')$, we can calculate the kernel matrix $\mathbf{K}$. We can thus evaluate $\widetilde{\mathbf{K}}$ by using only the kernel functions! We can also simply use $\widetilde{\mathbf{K}}$ to determine eigenvalues and eigenvectors.

Typical kernels

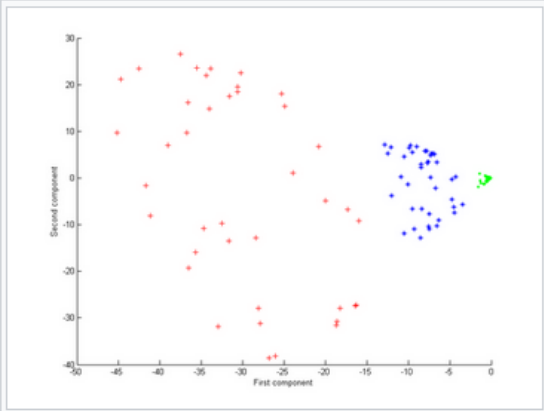
- Standard linear PCA is recovered with a linear kernel $k(x, x') = x^\top x'$
- 'Gaussian' kernel by Schölkopf et al: $k(x, x') = \exp(\frac{-||x-x'||^2}{0.1})$

This means: Although we have done the theoretical derivation of Kernel PCA via the nonlinear and high-dimensional functions Ψ, we never work with them! We only need to work with (much easier!) Kernel functions that we define beforehand. As a result, the resulting Kernel PCA algorithm only works with a pre-defined Kernel function, see next slide.

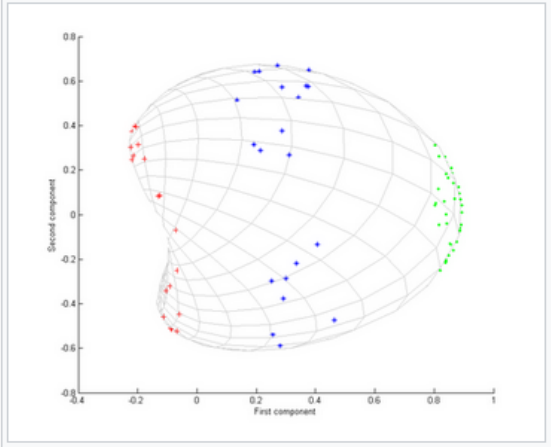
An Example of the Use of Different Kernels



Input points before kernel PCA



Output after kernel PCA with $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} + 1)^2$. The three groups are distinguishable using the first component only.



Output after kernel PCA, with a Gaussian kernel.

Summary of Kernel PCA

For given data $x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^M$ we can compute **non-linear** features via:

The Kernel PCA algorithm:

1. Choose a problem-tailored kernel k.
2. Compute the kernel matrix K with $K_{i,j} = k(x^{(i)}, x^{(j)})$.
3. Center the kernel matrix and hence the transformed input data via:
$$\widetilde{K} = (K - 1_N K - K 1_N + 1_N K 1_N) \text{ with } 1_N \in \mathbb{R}^{N \times N},\ (1_N)_{ij} := \frac{1}{N}$$
4. Approximate numerically the N eigenvalues and eigenvectors of $\widetilde{K}$.
5. Choose $1 \leq k \leq N$ eigenvectors of the k largest non-vanishing eigenvalues.
6. Normalize computed eigenvectors via (5)
7. Compute for a point $x \in \mathbb{R}^M$ the (non-linear) principal components as:
$$z_j = \sum_{i=1}^{N} \vec{\alpha}_i^{(j)} k(x^{(i)}, x), \text{ for } j = 1, \ldots, k.$$

Discussion

- In the original M-dimensional x-space there are M many orthogonal eigenvectors, hence we can find *at most M linear principal components*.
- The dimension of the feature space, however, can be much larger than that, even infinite. Thus, we can find a number of non-linear principal components that can exceed M.
- However: The number of *non-zero* eigenvalues cannot exceed the number of data points N, as the covariance matrix in feature space has rank at most N.
- This is reflected in the fact that kernel PCA involves eigenvalue expansion of the $N \times N$-matrix $\mathbf{K}$, see step 5. in the Kernel PCA algorithm.

Applications

Using a Kernel PCA is meaningful in the following situations:

- as a first tool for data reduction
- when there is not yet a model known for the data

Possible applications:

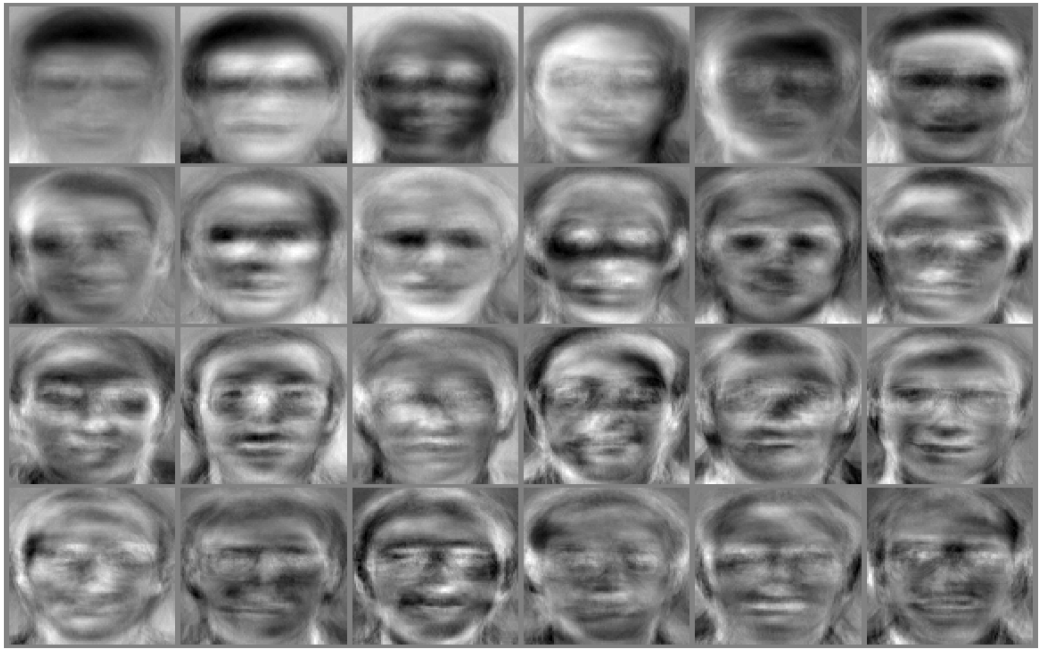
- Data clustering
- Parameter reduction
- Feature extraction
- . . .

Example:
In the following we will discuss the application of Kernel PCA for face recognition tasks.

Face recognition)

(see Ming-Hsuan Yang, NIPS '01)

1. Linear PCA $\rightarrow$ 'Eigenfaces'



Face recognition

Experiment: compare linear PCA with Kernel PCA

- two public test data sets with gray value images of faces:
  1. AT&T data set
  2. Yale data set
- best results with (inhomogeneous) polynomial kernels:

$$k(x, y) := (\langle x, y \rangle + a)^d, \text{ for } a \in {\mathbb{R}_0}^+, \ d = 2, 3$$

- evaluation via the *Leave-One-Out validation method* (see later lectures)

Face recognition on AT&T data set

AT&T data set:

- Images of size 23×28 pixels $\Rightarrow$ 644-dimensional feature vector
- 400 images of 40 different persons

- small variations in face scale and angle
- constant ligthing and face expression

Results:

Method	**Principal components**	**Classification error %**
Eigenface	30	2.75 (11/400)
Kernel-Eigenface, $d = 2$	50	2.50 (10/400)
Kernel-Eigenface, $d = 3$	50	2.00 (08/400)

Face recognition on Yale data set

Yale data set:

- Images of size 29×41 pixels $\Rightarrow$ 1189-dimensional feature vector
- 165 images of 11 persons

- strong changes in lighting and face expression
- constant face scaling and angle

Results:

Method	**Principle components**	**Classification error %**
Eigenface	30	28.48 (47/165)
Kernel-Eigenface, $d = 2$	80	27.27 (45/165)
Kernel-Eigenface, $d = 3$	60	24.24 (40/165)

Conclusion

We can summarize the following observations:

Kernel PCA is a good choice, if:

- there is a lot of training data and not many features
- one needs a nonlinear classifier to separate data
- the dimension M of the input data space is much bigger than the amount of input data N, i.e., $M \gg N$
  $\rightarrow$ computational complexity much lower (compare linear kernel!)

Problems of Kernel PCA:

- Computational complexity also depends on choice of kernel k
- Choice of "best" kernel is a research topic on its own
  $\rightarrow$ Testing many kernels makes computational effort cumbersome

Thank you for your attention!