FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
NATURWISSENSCHAFTLICHE
FAKULTÄT

Kernel Princpial Component Analysis

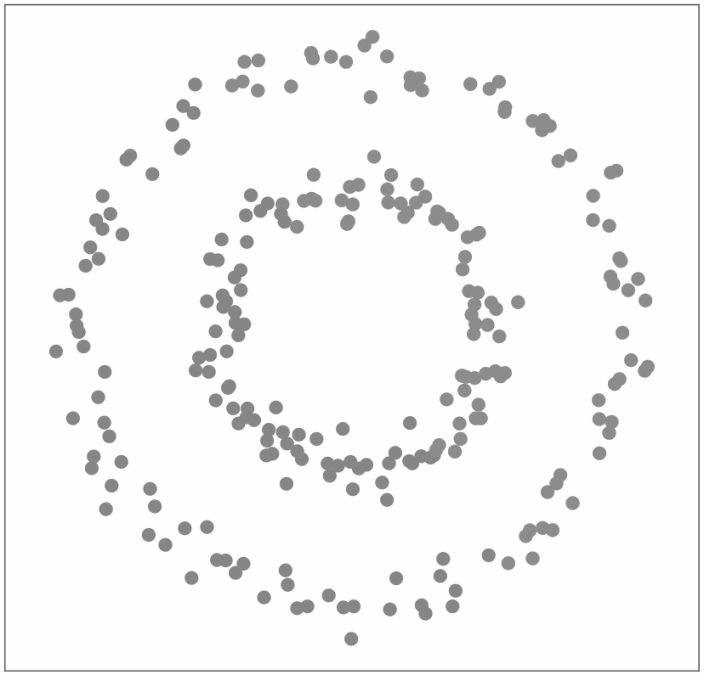
Lecture “Mathematical Data Science” 2021/2022

Frauke Liers
Friedrich-Alexander-Universität Erlangen-Nürnberg
3.11.2021

DISCRETE
OPTIMIZATION

Motivation for Extension of (linear) PCA

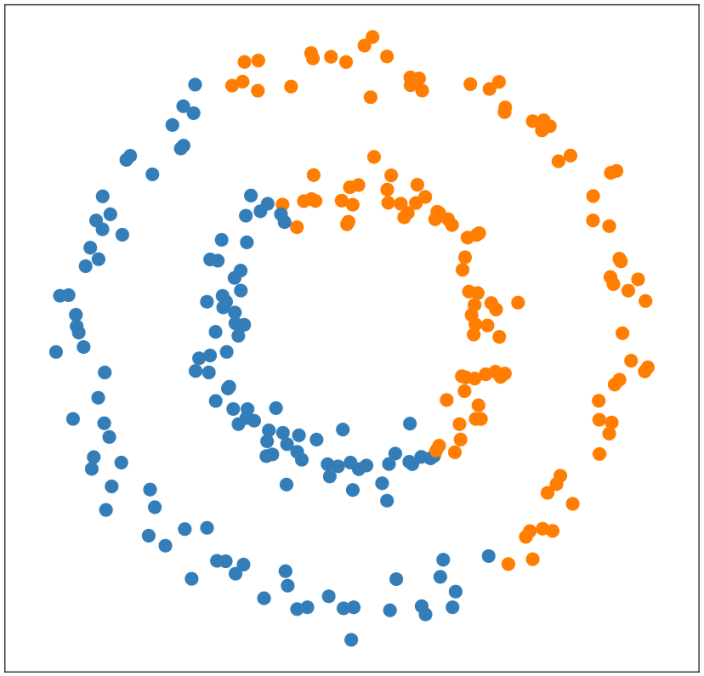
Given input data with circular structure:



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

Motivation for Extension of (linear) PCA

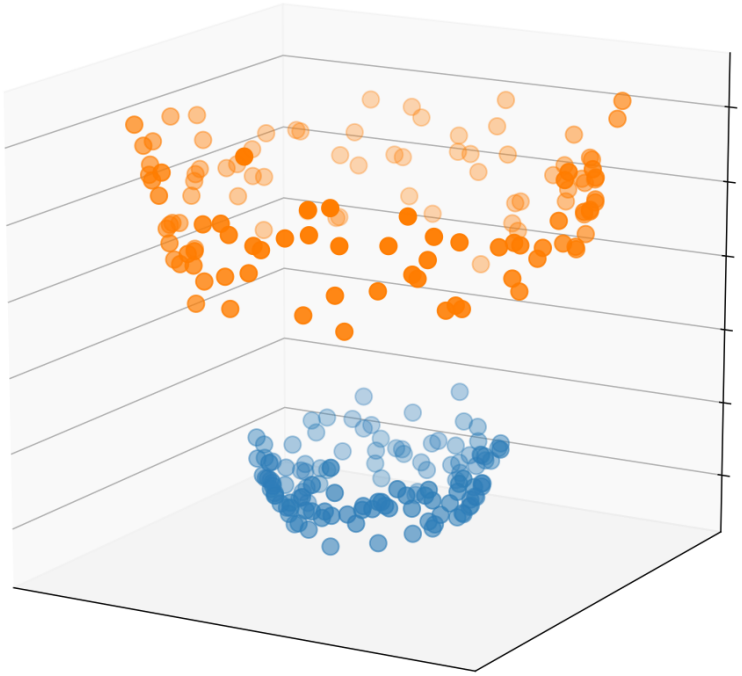
Clustering using features from linear PCA directly on data:



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

Motivation for Extension of (linear) PCA

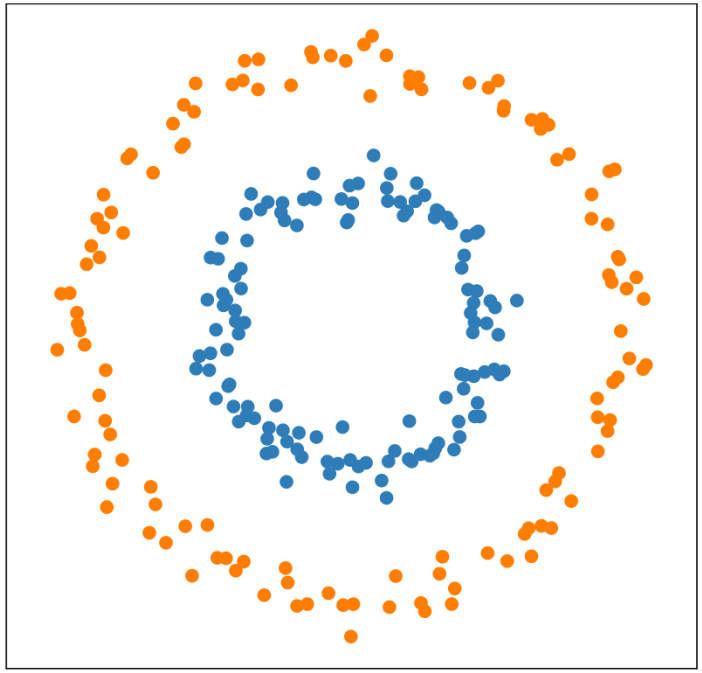
Nonlinear mapping of the data yields **linear separability**:



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

Nonlinear 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 $\mathcal{H}$:

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

Then, perform linear principal component analysis (PCA) in $\mathcal{H}$ e.g., for clustering.

Observations:

- Ψ can potentially be a **nonlinear transformation map**

Aim: Efficient algorithms for performing PCA in 'feature space' $\mathcal{H}$. Never calculate Ψ explicitely, i.e. assume it is given and work with it. Allows high-dimensional Ψ! Later we will explicitely give some formulas for possible 'kernel functions' that are built from Ψ (to be defined later).

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 linear PCA:

- principal components are defined by the eigenvecturs v_i of the covariance matrix $\mathbf{C}v_i = \lambda_i v_i, i = 1, \ldots, M$.
- $M \times M$ covariance matrix C defined by $C = \frac{1}{N} \sum_{i=1}^{N} x^{(i)} x^{(i)^T}$
- eigenvectors are normalized such that $v_i^\top v_i = 1$.

Derivation of kernel PCA

Consider nonlinear transformation $\Psi(x)$ into D-dimensional 'feature space', such that a data point x_n is projected onto a point $\Psi(x_n)$. Let us perform linear PCA in feature space. Implicitely defines nonlinear principal component model in original space.

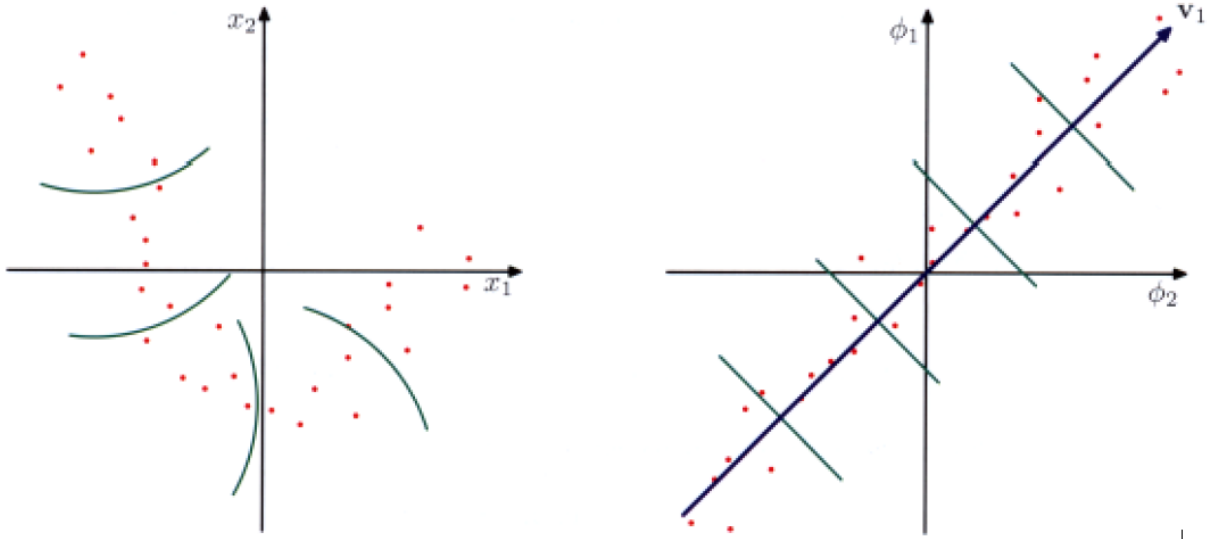


Figure 12.16 Schematic illustration of kernel PCA. A data set in the original data space (left-hand plot) is projected by a nonlinear transformation $\phi(\mathbf{x})$ into a feature space (right-hand plot). By performing PCA in the feature space, we obtain the principal components, of which the first 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 $\mathcal{H}$

again 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:

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

- goal: solve this eigenvalue problem without having to work explicitely in feature space.
- 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 we see from this formula? provided $\lambda > 0$, left-hand side is sum of scalars with $\Psi(x^{(i)})$, i.e.: each eigenvector v is given by a linear combination of the $\Psi(x_i)$.

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

in formulas: for an eigenvector $\mathbf{v}$ $\exists \alpha_i \in \mathbb{R}, i = 1, \ldots, N$ 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

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

Key step: express this in terms of an *kernel function* of (abstract) form $k(x_i, x_m) = \Psi(x_{(i)})^\top \Psi(x_{(m)})$. (We will see later some 'typical' kernel functions.) We do this by multiplying both sides by $\Psi(x_{(l)})^\top$. We receive

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

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

in formulas: for an eigenvector $\mathbf{v}$ $\exists \alpha_i \in \mathbb{R}, i = 1, \ldots, N$ 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

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

Key step: express this in terms of an *kernel function* of (abstract) form $k(x_i, x_m) = \Psi(x_{(i)})^\top \Psi(x_{(m)})$. (We will see later some 'typical' kernel functions.) We do this by multiplying both sides by $\Psi(x_{(l)})^\top$. We receive

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

Kernel Approach

Last formula from last 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

Last formula from last slide

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

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

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

note we have corrected the above formula that before had both α and **a** in which we have removed a factor of **K** from both sides of (3). (Note this only changes the solutions by eigenvectors of **K** having zero eigenvalues that do not affect the principal components projection, see exercises.)

Normalization of Eigenvectors

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

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

(last equation follows from (4)).
Finally, 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 be done via the *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.

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 nonlinear principal components that can exceed M.

However: number of *nonzero* eigenvalues cannot exceed number of data points N, as 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 $\mathbf{K}$.

How about zero mean in projected data?

This it in general not the case. Cannot compute mean and substract it, as we want to avoid working in feature space. Again via kernel: Let a projected data point after centralizing be denoted by $\Psi(\tilde{x}^{(i)})$:

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

The elements of the Kernel matrix are then given by

$$\begin{aligned}
\tilde{K}_{ij} &= \Psi(\tilde{x}_i)^\top \Psi(\tilde{x}_j) \\
&= \Psi(x_i)^\top \Psi(x_j) - \frac{1}{N}\sum_{l=1}^{N}\Psi(x_i)^\top \Psi(x_l) \\
&\quad - \frac{1}{N}\sum_{l=1}^{N}\Psi(x_j)^\top \Psi(x_l) + \frac{1}{N^2}\sum_{l=1}^{N}\sum_{m=1}^{N}\Psi(x_l)^\top \Psi(x_m) \\
&= k(x_i, x_j) - \frac{1}{N}\sum_{l=1}^{N}k(x_l, x_j) - \frac{1}{N}\sum_{l=1}^{N}k(x_i, x_l) + \frac{1}{N^2}\sum_{m=1}^{N}\sum_{l=1}^{N}k(x_m, x_l).
\end{aligned}$$

In matrix notation, this reads (with 1_N the $N \times N$ matrix where every element takes value $\frac{1}{N}$):
$\widetilde{K} = (K - 1_N K - K 1_N + 1_N K 1_N), 1_N \in \mathbb{R}^{N \times N},\ (1_N)_{ij} := \frac{1}{N}$
Therefore: If we define a Kernel function $k(x, x')$, we can calculate the kernel matrix $\mathbf{K}$. We can thus evaluate $\widetilde{\mathbf{K}}$ by using unly the kernel functions, use $\widetilde{\mathbf{K}}$ to determine eigenvalue and eigenvectors.

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

Summary of Kernel PCA

For given data $x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^M$ we can compute **nonlinear** 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 $\tilde{K}$.
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 test point $x \in \mathbb{R}^M$ the (nonlinear) principal components as:
$$z_j = \sum_{i=1}^{N} \vec{\alpha}_i^{(j)} k(x_{(i)}, x), \text{ for } j = 1, \ldots, k.$$

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:

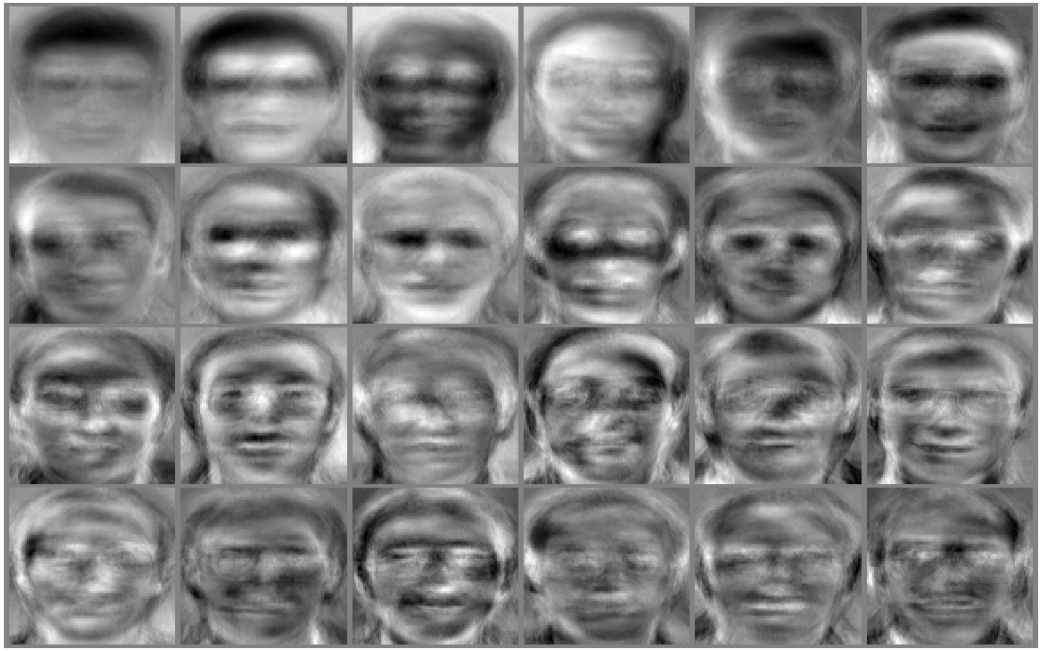
- Data clustering
- Parameter reduction
- Feature extraction
- . . .

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

Face recognition)

(see Ming-Hsuan Yang, NIPS ’01)

- Two possible ways for feature extraction in face recognition are:
  1. Linear PCA $\rightarrow$ Eigenfaces
  2. Linear Kernel Fisher Discriminant



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*

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!