Machine Learning

– winter term 2016/17 –

Chapter 06: Dimensionality Reduction

(and Anomaly Detection)

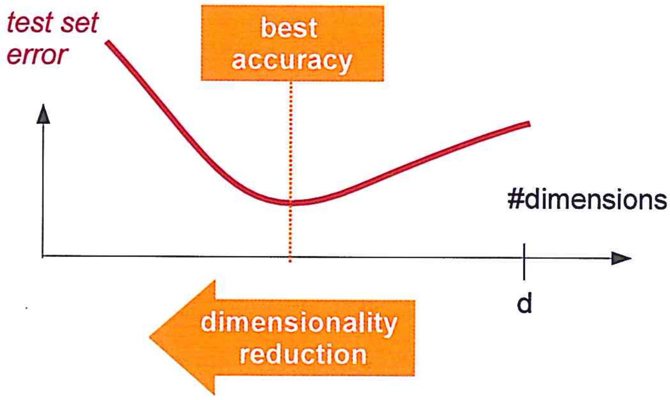
Prof. Adrian Ulges
Masters "Computer Science"
DCSM Department
University of Applied Sciences RheinMain

Outline

Dimensionality Reduction: Motivation

The Challenge

- **Reminder**: In machine learning, we are usually given **d-dimensional** samples $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$
- The number of features d can be high!



- **Goal: Reduce d** while preserving (or even improving) discriminativity

Why?

- **Efficiency**: faster training, faster application, less storage
- Avoid **overfitting**: overcoming the curse of dimensionality
- Better **interpretability** (and maybe visualization) of data

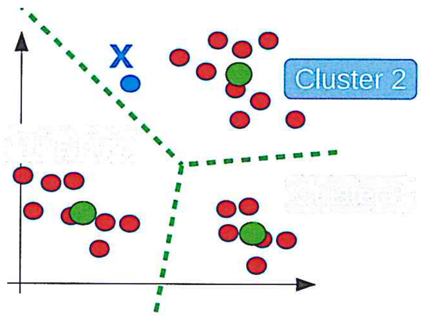
Dimensionality Reduction: Overview

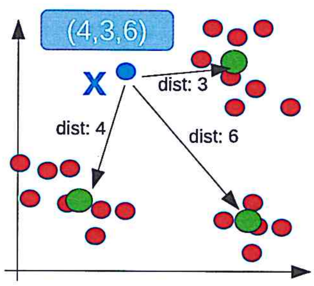
Approaches

- **feature selection**
- **feature derivation**, typically, by applying transformations

Example: K-Means for Feature Derivation

- **Approach**: map samples $\mathbf{x}$ to clusters *(vector quantization)*
- **Alternative 1**: store only the cluster number *(1 dimension!)*
- **Alternative 2**: store the distances to all centers *(K dimensions) (see* `sklearn > KMeans > transform()`*)*





Outline

1. Dimensionality Reduction: Motivation

2. Feature Selection

3. Principal Component Analysis

4. PCA Example: "Eigenfaces"

5. Anomaly Detection

Feature Selection: Strategies

- **Goal**: Find a **subset of features** $\mathcal{F} \subseteq \{1, ..., d\}$ for which we can solve our machine learning problem 'best' *(for example: minimizing classification error)*
- This is a **search problem** *(brute-force effort: $O(2^d)$)*
- There are **three common approaches**: **wrappers**, **filters**, and **embedded methods**

1. Wrappers

- Wrappers use an explicit evaluation of feature subsets *(training and validating classifiers)*
- Search can be done in a **greedy** fashion (adding or removing the 'best' features to the feature set), or by **backtracking**
- Benefit: It takes the **underlying classifier** into account!
- Usually the most reliable way, but very slow

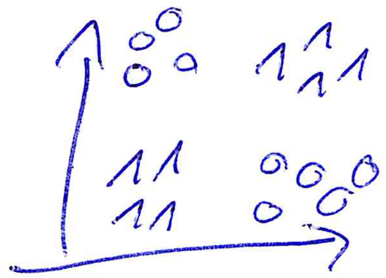
Feature Selection: Strategies

2. Filters

- Assess feature quality by a **proxy measure**
  - example: *mutual information* between feature X and class labels C

$$I(X,C) = \sum_{x \in X} \sum_{c \in C} p(x,c) \cdot \log_2 \left(\frac{P(x,c)}{P(x) \cdot P(c)} \right)$$

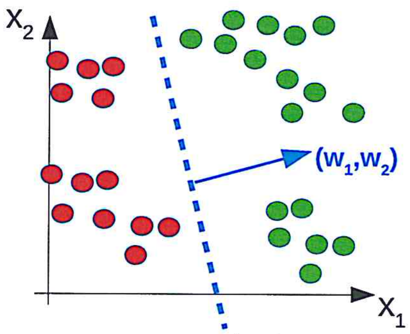
- **Search strategy**: rank features by their quality, pick the K top ones *(K determined via cross-validation)*
- Filters are cheaper than wrappers, but not as accurate



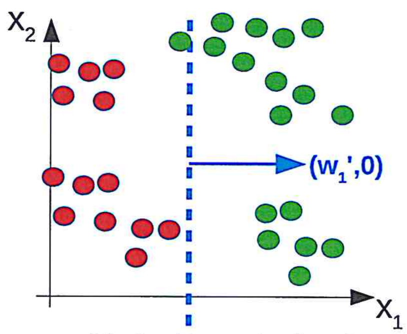
Feature Selection: Strategies

3. Embedded Methods

- ... treat feature selection as a part of **model construction** (i.e., we find classifier and features in the *same process*)
- Optimization is driven towards models with **few features**
- **Example**: logistic regression with **regularization**
  - the classifier penalizes feature weights, shrinking them to zero
  - features with weight zero are "filtered"!



no L_1-Regularization



with L_1-Regularization (feature x_2 has weight 0)

Outline

Principal Component Analysis (PCA)

Idea (unsupervised!)

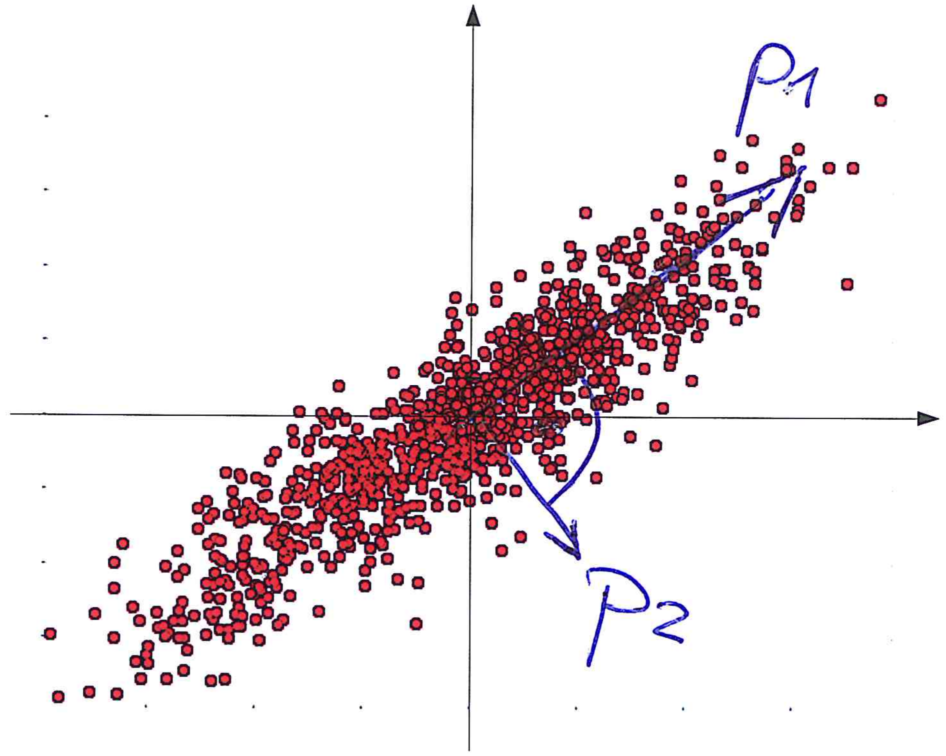
- The dataset exhibits a large stretch along some directions. These are the *important* directions.
- Along other directions, there is only little variation. These directions are merely *noise* and can be discarded.
- PCA is about finding the *important* directions *(or* **principal components***)* of the data

Principal Components: Formalization

- In the following, we assume our data points to be centered around the origin (otherwise, we simply shift the data beforehand)
- What is the *most important direction*? What is the *second most important* direction, etc.?

PCA: Illustration

What are the Principal Components here?

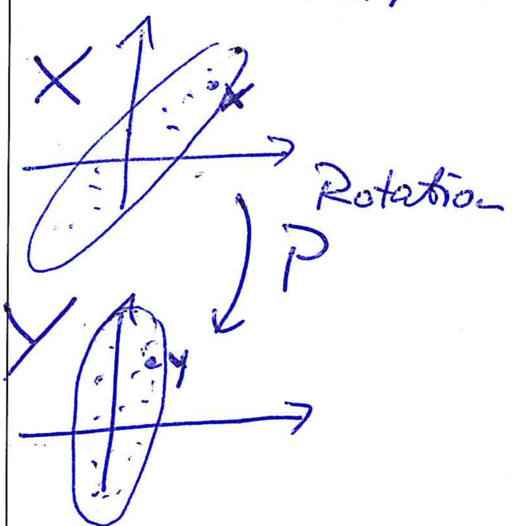


Principal Components: Derivation

Obviously, the principal components seem to be related to the data's **covariance matrix**. Let's find out how[1]

We assume: $\bar{X} = \vec{0}$

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} x_{11} & \dots & x_{1d} \\ \vdots & & \\ x_{n1} & \dots & x_{nd} \end{pmatrix}$$



Goal: Find rotation matrix $P \in \mathbb{R}^{d \times d}$

$$Y^T = P \cdot X^T$$

$P \cdot x$

Such that $\mathrm{cov}(Y) = \begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \ \lambda_d \end{pmatrix}$

[1] cmp. Marsland, page 228f

Principal Components: Derivation

$$\mathrm{cov}(Y) = \frac{1}{n} \cdot Y^T \cdot Y$$

$$Y_{ij} = \frac{1}{n} \sum_s x_{si} \cdot x_{sj}$$

$$= \frac{1}{n} (P \cdot X^T) \cdot (P \cdot X^T)^T$$

$$(A \cdot B)^T = B^T \cdot A^T$$

$$= \frac{1}{n} P \cdot X^T X \cdot P^T$$

$$= P \cdot \underbrace{\frac{1}{n} X^T X}\cdot P^T$$

$$\mathrm{cov}(Y) = P \cdot \mathrm{cov}(X) \cdot P^T \quad / P^T \cdot$$

$$P^T \cdot \mathrm{cov}(Y) = \cancel{P^T P} \cdot \mathrm{cov}(X) \cdot P^T$$

$$\begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \; \lambda_d \end{pmatrix} \qquad (\quad) \cdot (\quad)$$

Principal Components: Derivation

Let p_j be a column of P^T
(= a row of P)

$$\lambda_j \cdot p_j = \mathrm{cov}(X) \cdot p_j$$

$$\lambda \cdot x = A \cdot x$$

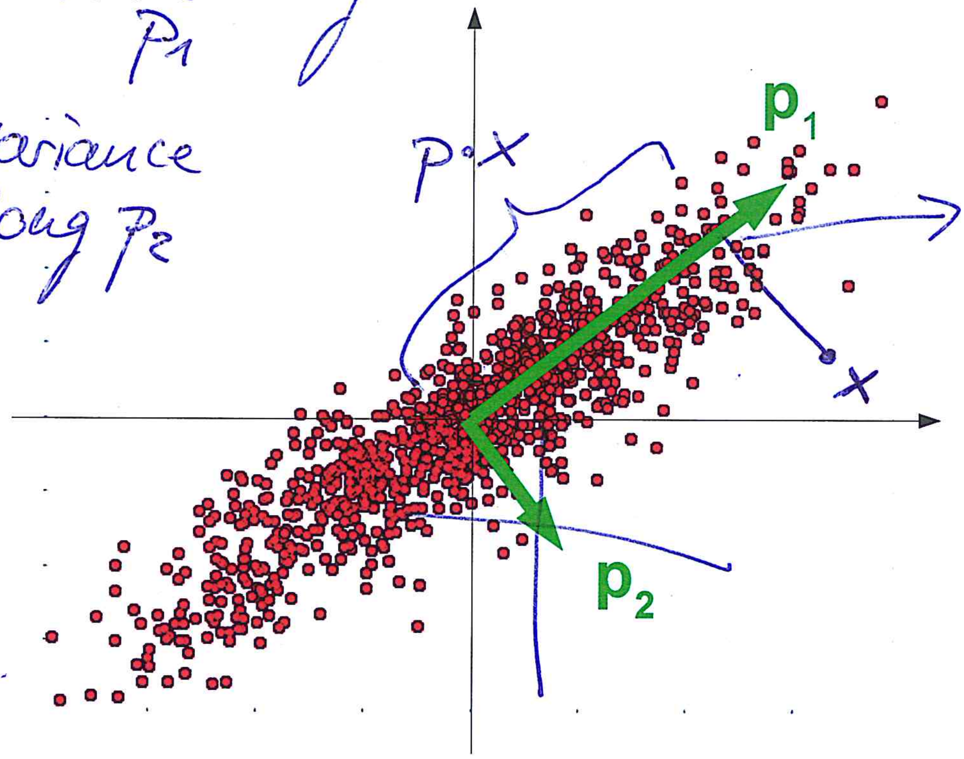
The rows $p_1, \ldots, p_d$ are the eigenvectors of the covariance matrix $\mathrm{cov}(X)$.

Principal Components: Illustration

λ_1 = variance along p_1

λ_2 = variance along p_2

$p_1 \circ x$



reconstructed data x'

PCA: Training

```
function PCA_TRAIN(x_1, ..., x_n, K)   // given: samples x_1, ..., x_n ∈ ℝ^d
    μ := 1/n Σ_i x_i
    for i = 1, ..., n:
        x_i := x_i − μ                 // shift the samples to mean zero
    stack the samples into an n × d data matrix X
    Σ := 1/n (X^T · X)                 // covariance matrix
    compute Σ's eigenvalues λ_1, ..., λ_d and eigenvectors p_1, ..., p_d
     (sorted in descending order of the eigenvalues)
    stack p_1, ..., p_K as rows into a K × d-Matrix P_K
    return P_K, μ
```

Remarks

- **Training**: Compute the top K eigenvectors of the data's covariance matrix
- **Application**: reduces the samples from d to K dimensions

```
function PCA_APPLY(x)                  // given: a new sample x
    return P_K · (x − μ)               // dimension of return value: K
```

PCA: Remarks

- Heuristic for choosing K: Preserve α (e.g., 90%) of the eigenvector's total energy

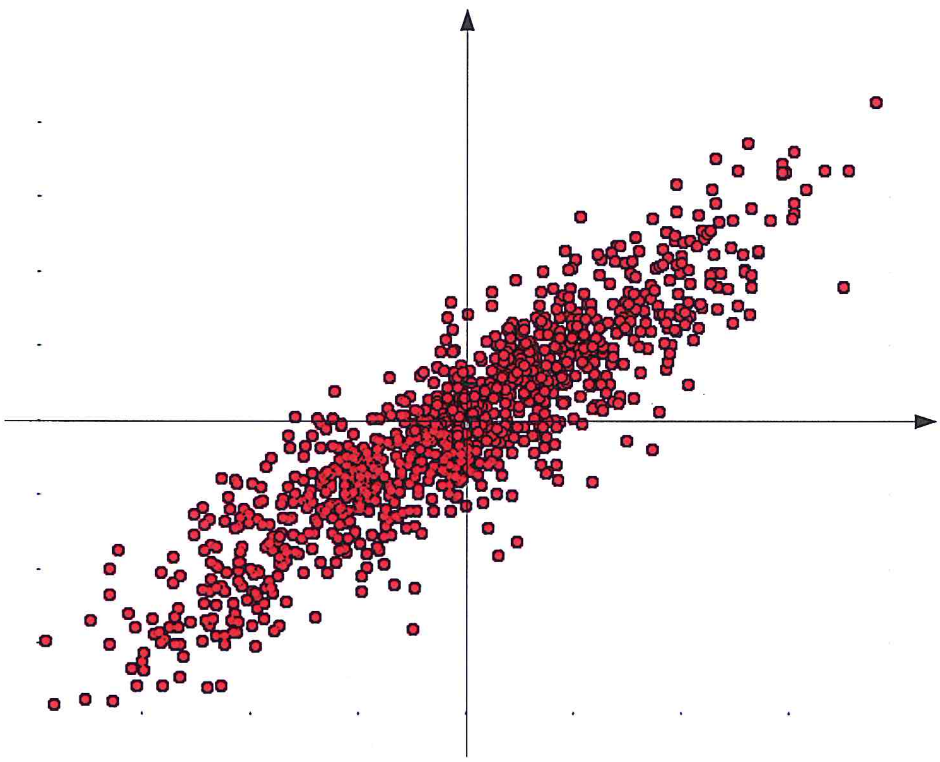
$$K := \min_{K \in \{1,...,d\}} \text{ such that } \sum_{k=1}^{K} \lambda_k \geq \alpha \cdot \sum_{k=1}^{d} \lambda_k$$

- Given a reduced K-dimensional feature $\mathbf{y} = (y_1, ..., y_K)$, we can reconstruct $\mathbf{x}$ (to some extent) from $\mathbf{x}'$:

$$\mathbf{x}' = \mu + y_1 \cdot \mathbf{p}_1 + y_2 \cdot \mathbf{p}_2 + ... + y_K \cdot \mathbf{p}_K$$

- We call $||\mathbf{x} - \mathbf{x}'||$ the **reconstruction error**.

PCA: Illustration



From PCA to Whitening

We learned above: Given a feature vector $\mathbf{x}$ (and the $d \times d$ matrix P with Σ's eigenvectors as rows), by applying the transformation $P \cdot \mathbf{x}$, the **covariance matrix** of the data becomes

$$\begin{pmatrix} \lambda_1 & 0 & & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & 0 & \lambda_d \end{pmatrix}$$

Each value λ_i is a variance within one dimension. We turn these variances into 1, simply by dividing by the standard deviation $\sqrt{\lambda_i}$. This leads to the **whitening** transform *(see Chapter 'Features')*:

$$Diag(\frac{1}{\sqrt{\lambda_1}}, \frac{1}{\sqrt{\lambda_2}}, \dots, \frac{1}{\sqrt{\lambda_d}}) \cdot P \cdot \mathbf{x}.$$

Applying this transformation turns the data's covariance matrix into the identity I and decorrelates the features.

Outline

Example: Applying PCA to Images

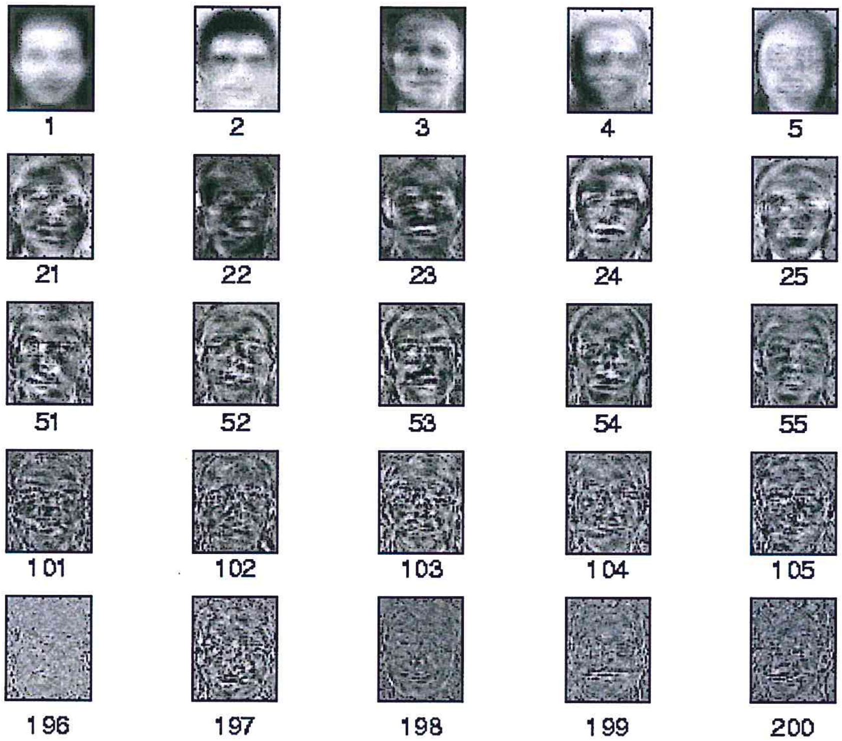
- We can apply PCA to **images**, simply by scaling images to a standard resolution (say, $N \times M$) and stacking all pixel values into sample vectors of dimension $d = N \times M$
- Note: The principal components are $(N \cdot M)$-dimensional, too! (i.e., we can *visualize them as images*)
- Example: PCA for **face recognition** $\rightarrow$ "eigenfaces" *(apply PCA to lots of face images)*

Example: Applying PCA to Images

Mean face (top left) and the first 7 principal components (=eigenfaces)

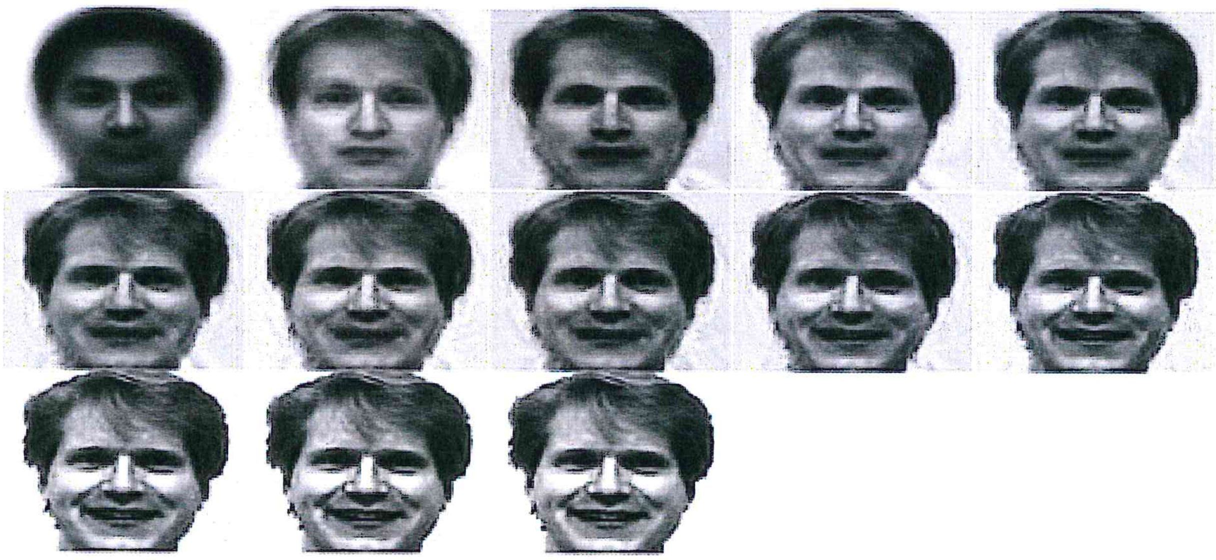
Example: Applying PCA to Images

Some eigenfaces from a different face dataset

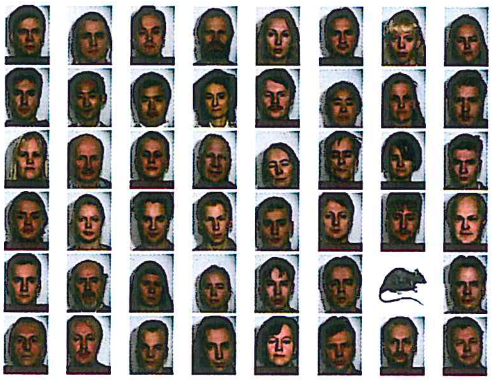


Example: Applying PCA to Images

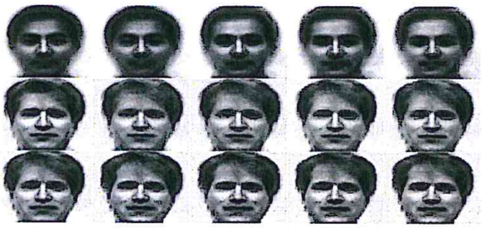
Reconstruction of a face using $0, 8, 16, \ldots$ eigenfaces



Example: Applying PCA to Images



- Which of these images does not show a human face?
- Approach: Compare the **original image** with its **PCA reconstruction**



faces are well reconstructed

non-faces are poorly reconstructed

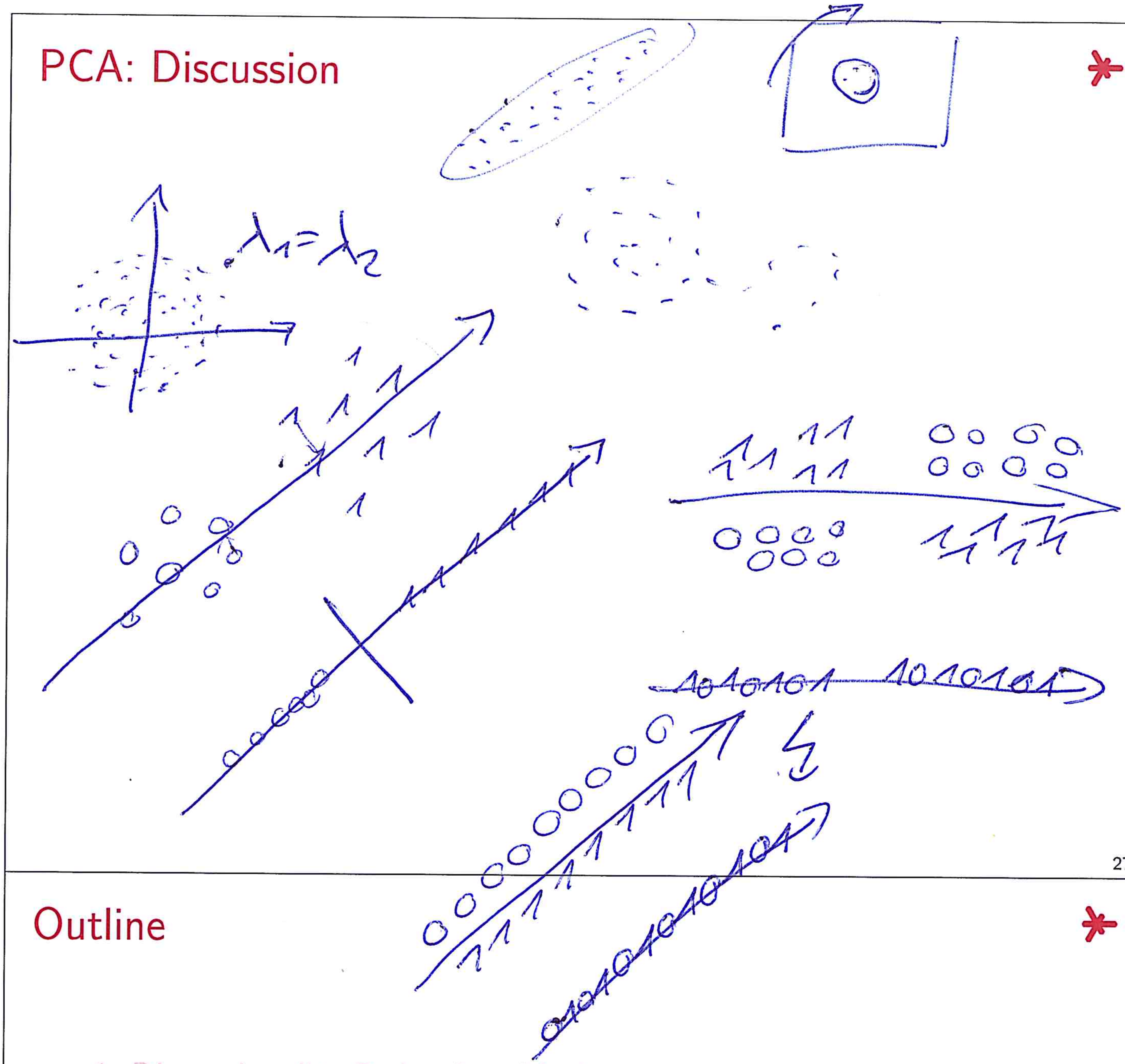
Eigenfaces: Discussion

The Eigenfaces approach for **face detection** is **simple and powerful**, but it has some shortcomings:

- The results depend strongly on *illumination*, *shadows*, and *local changes*, e.g. glasses or beards
  - include those effects in the training set
  - learn smaller components (eye, mouth, ...)
  - (illumination): normalize all images
- Only faces of *fixed size* are detected
  - create scaled versions of the input image before searching
- Already small *rotations* of the head change the result
  - include rotated faces into the training set.
  - try to make the image upright before the applying PCA

Since eigenfaces [8], there have been at least two generations of more elaborate face detection methods [9, 7].

PCA: Discussion



Outline

1. Dimensionality Reduction: Motivation
2. Feature Selection
3. Principal Component Analysis
4. PCA Example: "Eigenfaces"
5. Anomaly Detection