Fachhochschule Graubünden
University of Applied Sciences

# Methods & Algorithms

(formerly known as Introduction to Data Science)

FS 2023

Prof. Dr. rer. nat. habil. Ralf-Peter Mundani

DAViS

## Lost in Dimension – Geschichten aus der *n*-ten Dimension

- nochmal *k-NN*
  - Klassifizierung basiert auf nächsten Nachbarn (bzw. deren Klassifizierung)
  - Beispiel Empfehlung: Vorlieben Nutzer:in *A* basieren auf Vorlieben der nächsten Nachbarn

- ABER
  - Fluch der Dimension: nächste Nachbarn nicht mehr wirklich 'nahe'
  - Overfitting: nächster Nachbar könnte nur 'Rauschen' (noise) darstellen
  - Wichtung: manche Attribute sind wichtiger / prägnanter als andere
  - Komplexität: je höher die Dimension, desto höher die Berechnungskosten

- Idee
  - Dimensionsreduktion auf wenige (latente) Attribute

# Wiederholung: ein bisschen lineare Algebra ☺

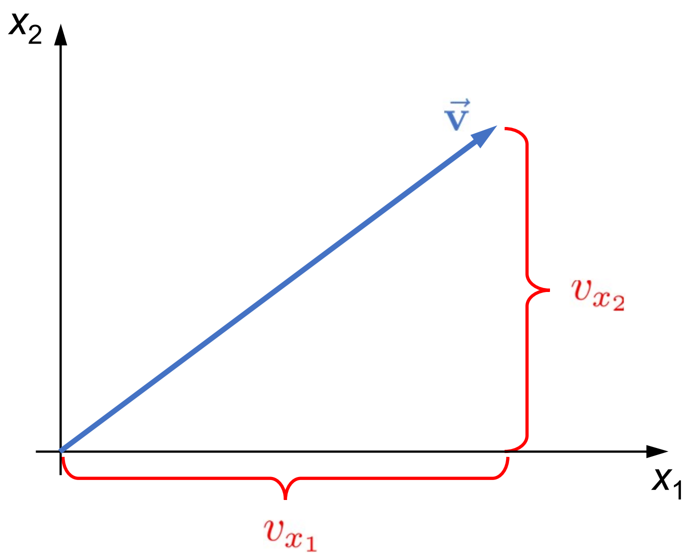
- Nomenklatur
  - Skalare: *s*
  - Vektoren: $\vec{\mathbf{v}}$ (1D Reihung)
  - Matrizen: $\mathbf{A}$ (2D Reihung)

- Operationen
  - Skalarprodukt: $\vec{\mathbf{v}}^T \cdot \vec{\mathbf{w}} = s$
  - $\boxed{\text{dyadisches Produkt: } \vec{\mathbf{v}} \cdot \vec{\mathbf{w}}^T = \mathbf{A}}$ (nicht heute)
  - Matrix-Vektor-Multiplikation: $\mathbf{A} \cdot \vec{\mathbf{v}} = \vec{\mathbf{w}}$
  - Matrix-Matrix-Multiplikation: $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$

# Wiederholung: ein bisschen lineare Algebra ☺

- Vektoren

$$\vec{\mathbf{v}} = \begin{pmatrix} v_{x_1} \\ v_{x_2} \\ \vdots \\ v_{x_n} \end{pmatrix}$$




- Transponierte $\vec{\mathbf{v}}^T$ und Länge $|\vec{\mathbf{v}}|$ eines Vektors $\vec{\mathbf{v}}$

$$\vec{\mathbf{v}}^T = \begin{pmatrix} v_{x_1} & v_{x_2} & \cdots & v_{x_n} \end{pmatrix}$$

$$|\vec{\mathbf{v}}| = \sqrt{\sum_{i=1}^{n} v_{x_i}^2}$$ (auch $||\vec{\mathbf{v}}||$ )

`Euklidischer Abstand`

## Wiederholung: ein bisschen lineare Algebra ☺

- Skalarprodukt $\vec{\mathbf{v}}^T \cdot \vec{\mathbf{w}}$

$$\vec{\mathbf{v}}^T \cdot \vec{\mathbf{w}} = \begin{pmatrix} v_{x_1} & v_{x_2} & \cdots & v_{x_n} \end{pmatrix} \cdot \begin{pmatrix} w_{x_1} \\ w_{x_2} \\ \dots \\ w_{x_n} \end{pmatrix} = v_{x_1} \cdot w_{x_1} + v_{x_2} \cdot w_{x_2} + \ldots + v_{x_n} \cdot w_{x_n} = \sum_{i=1}^{n} v_{x_i} \cdot w_{x_i}$$

- Beispiel: $\vec{\mathbf{v}} = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}, \vec{\mathbf{w}} = \begin{pmatrix} 4 \\ 0 \\ 3 \end{pmatrix}$

$$\vec{\mathbf{v}}^T \cdot \vec{\mathbf{w}} = \begin{pmatrix} 3 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ 0 \\ 3 \end{pmatrix} = 3 \cdot 4 + 1 \cdot 0 + 2 \cdot 3 = 18$$

## Wiederholung: ein bisschen lineare Algebra ☺

- Matrizen

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

mit $m$ Zeilen und $n$ Spalten (kurz: $m \times n$)

Hauptdiagonale (für $m = n$)

- Transponierte $\mathbf{A}^T$ einer Matrix $\mathbf{A}$

$$\mathbf{A}^T = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}$$

mit $n$ Zeilen und $m$ Spalten (kurz: $n \times m$)

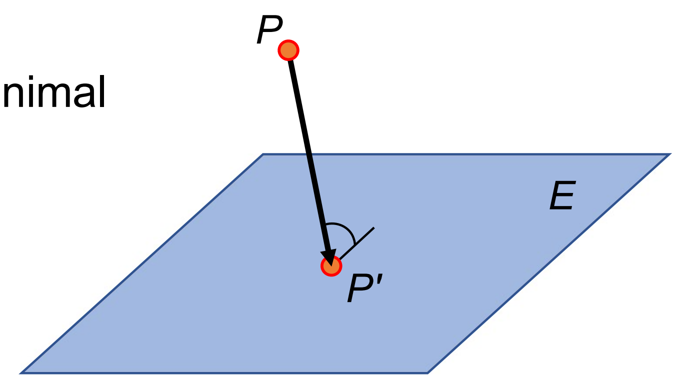
## Dimensionsreduktion: wenn der Raum mal wieder zu gross ist…

- Problem
  - viele Aufgaben enthalten unzählige (manchmal tausende) Attribute je Datenpunkt
  - Trainieren des Modells / Auffinden von Lösungen deutlich erschwert
  - bekannt als *Fluch der Dimension*
- Ausweg: Dimensionsreduktion
  - Reduktion der Attribute auf wenige prägnante
  - aber welche…?
- unnützes Wissen: je mehr Dimension vorhanden sind, desto grösser wird der Abstand zwischen zwei beliebigen Datenpunkten (Gefahr des Overfitting) $\rightarrow$ zur besseren Vorstellung: vergleiche Einheitsquadrat mit $d$-dimensionalem Einheitswürfel

## Dimensionsreduktion: wenn der Raum mal wieder zu gross ist…

- Dimensionsreduktion
  - Daten meist nicht gleichmässig über alle Dimensionen verteilt
  - viele Merkmale annähernd konstant / in hohem Masse miteinander korreliert
  - d.h. Daten liegen innerhalb eines niedrig dimensionalen Subraums

- Idee: Projektion auf Ebene niederer Dimensionalität (z.B. 3D $\rightarrow$ 2D)
  - Orthogonalprojektion senkrecht auf Ebene $E$
  - d.h. Abstand zwischen Punkt $P$ und Abbild $P'$ minimal
  - ABER: Projektion nicht immer beste Lösung



# Dimensionsreduktion: wenn der Raum mal wieder zu gross ist…

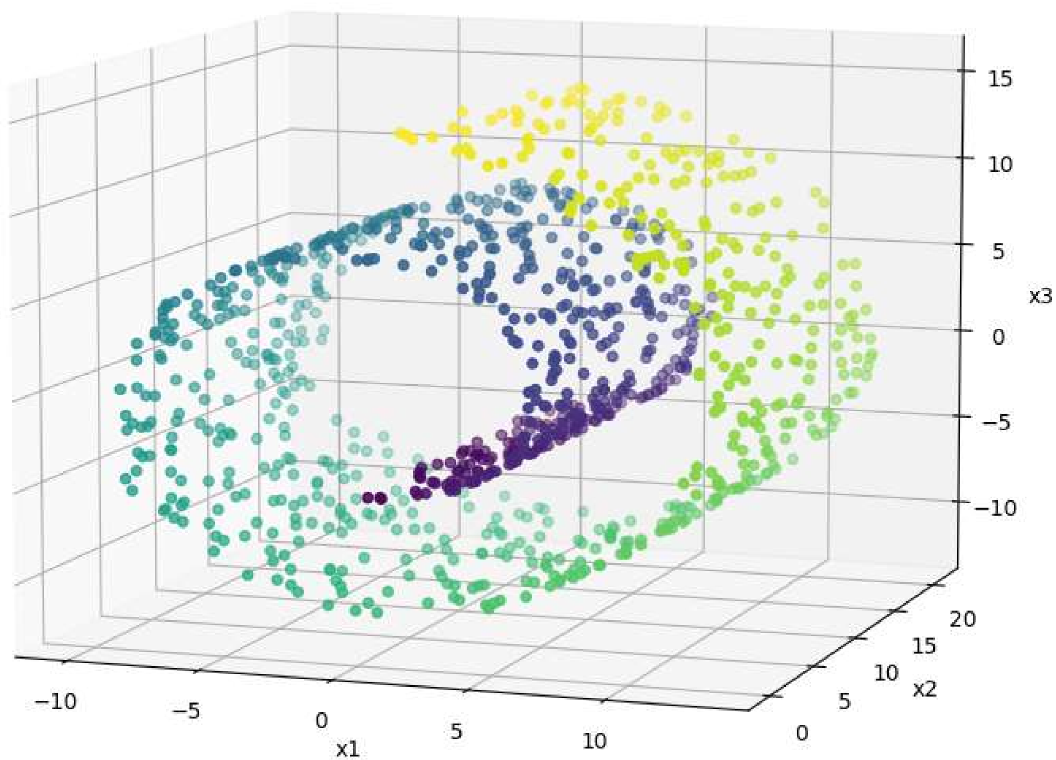
- Beispiel: 'Schweizer Rolle'

```
from sklearn import datasets
import matplotlib.pyplot as plt

data, shape = datasets.make_swiss_roll(n_samples=1000, noise=0.0)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(data[:,0], data[:,1], data[:,2], c=shape)
plt.show()
```

erzeugt leeres 3D-Koordinatensystem

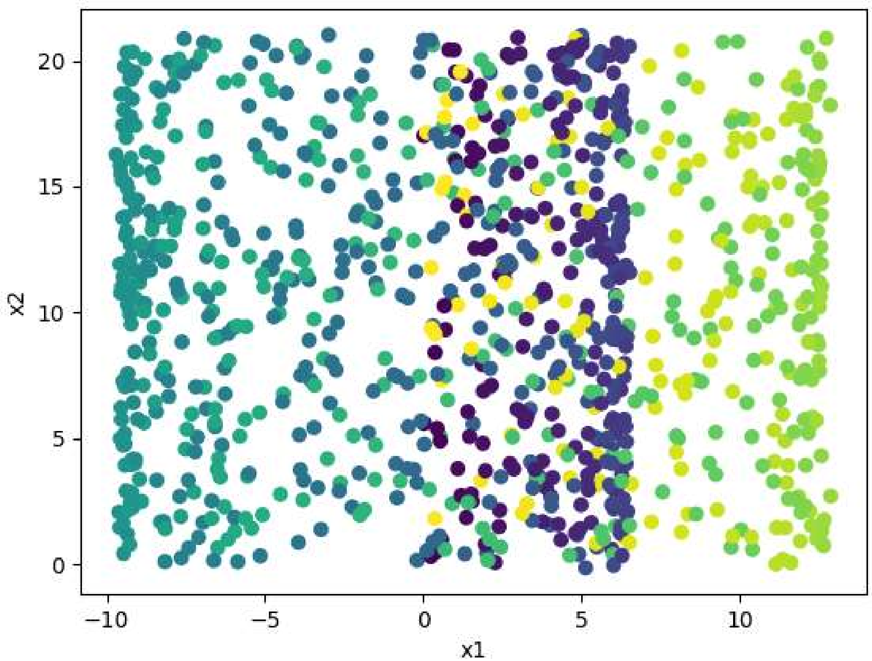
# Dimensionsreduktion: wenn der Raum mal wieder zu gross ist…

- Beispiel: 'Schweizer Rolle'

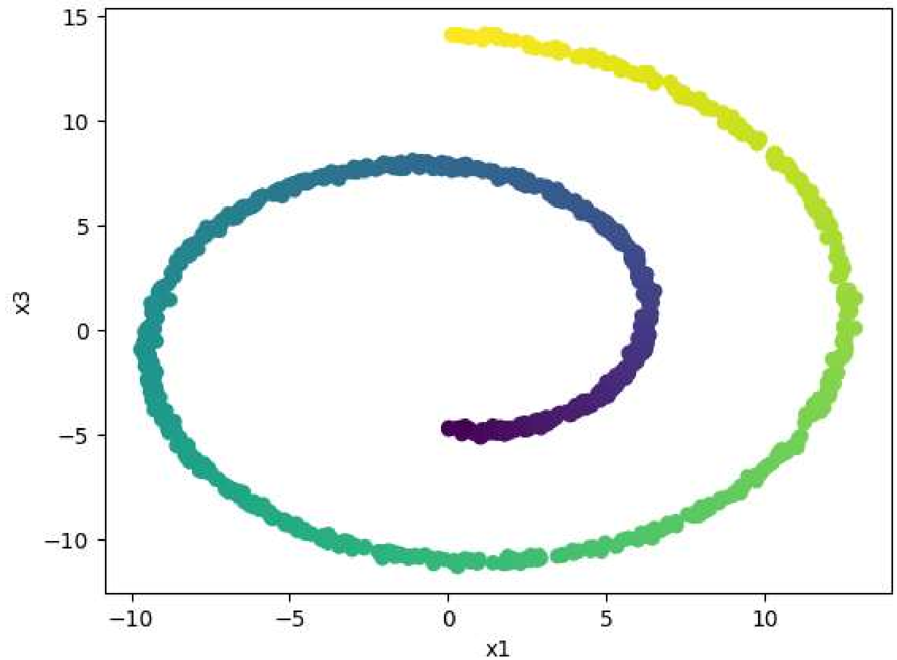


## Dimensionsreduktion: wenn der Raum mal wieder zu gross ist…

- Beispiel: 'Schweizer Rolle'




Projektion auf $x_1$–$x_2$-Ebene




Projektion auf $x_1$–$x_3$-Ebene

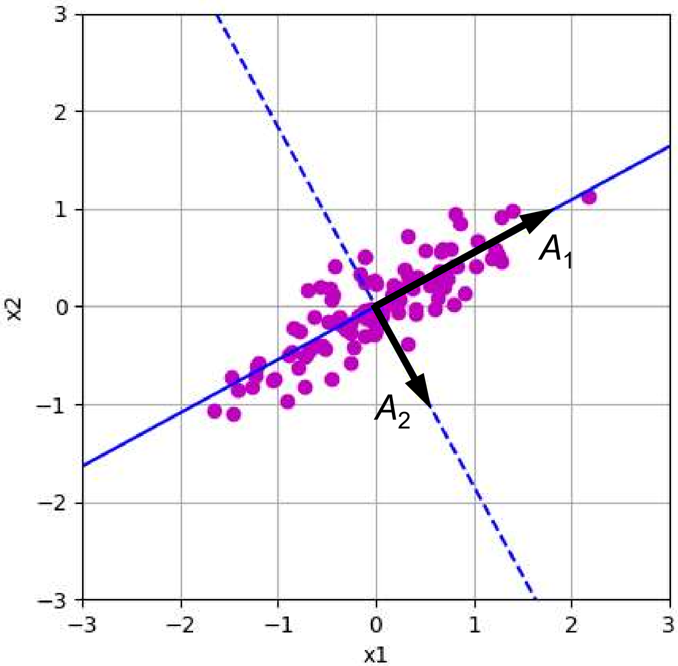
## Hauptkomponentenzerlegung

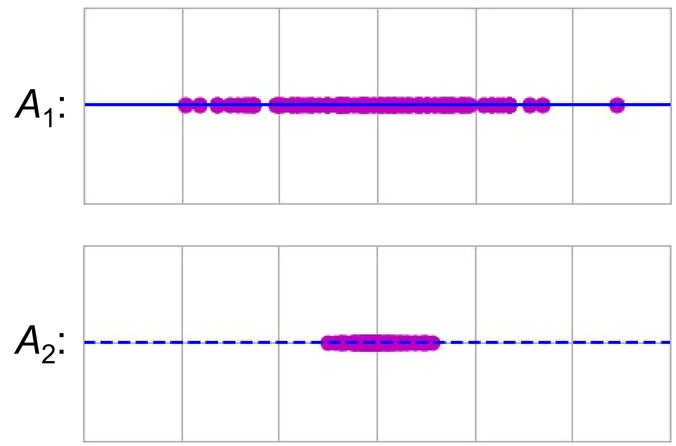
- *Principal Component Analysis* (PCA)
  - vermutlich das beliebteste Verfahren zur Dimensionsreduktion `von 3D: Hyperebene 2D`
  - PCA findet diejenige Hyperebene, die den Daten am nächsten ist und projiziert sie darauf
  - d.h. mittlerer quadratischer Abstand zwischen Daten und Hyperebene ist minimal
  - anderes ausgedrückt: grösste Varianz bleibt erhalten ($\rightarrow$ Informationsverlust)

- wie darf man sich eine Hyperebene vorstellen…?
  - 2D $\rightarrow$ Gerade
  - 3D $\rightarrow$ Ebene
  - $n$D $\rightarrow$ ($n$–1)-dimensionale Ebene, also eine Hyperebene ☺

## Hauptkomponentenzerlegung

- Beispiel: Achsen (Geraden) im 2D
  - $A_1$: grösste Varianz; $A_2$ (orthogonal zu $A_1$): grösste verbliebene Varianz

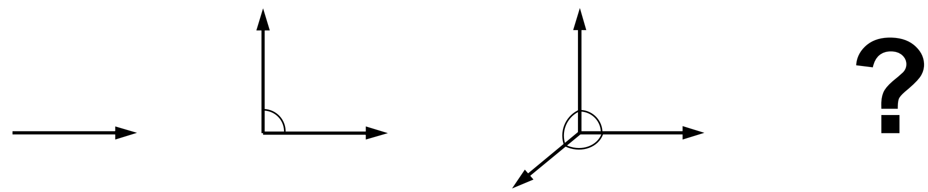



'Varianz' der projizierten Daten

## Hauptkomponentenzerlegung

- Hauptkomponenten
  - PCA findet in einem *d*-dimensionalen Raum (in absteigender Folge) *d* orthogonale Achsen
  - *i*-te Achse wird *i*-te Hauptkomponente (PC) der Daten genannt
  - Achsen als Vektoren durch den Ursprung vorstellbar



1D 2D 3D 4D

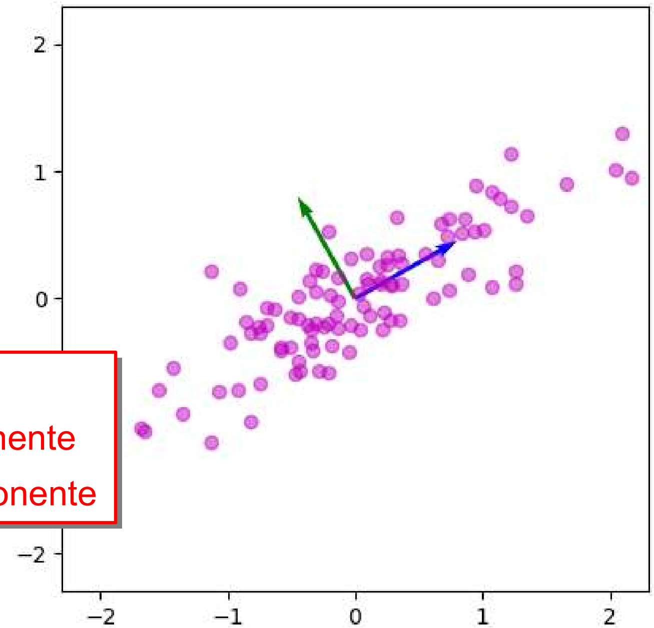
  - Speicherung als Matrix (mit Vektoren als Spalten) $\mathbf{V}^T = \begin{pmatrix} \vec{\mathbf{v}}_\mathbf{1} & \vec{\mathbf{v}}_\mathbf{2} & \cdots & \vec{\mathbf{v}}_\mathbf{d} \end{pmatrix}$

# Hauptkomponentenzerlegung

- Principal Component Analysis (Datei 'decomp_test.dat')
  - Model (Paket **`sklearn`**) trainieren und Varianz ausgeben

```
from sklearn.decomposition import PCA
data = np.loadtxt("decomp_test.dat", delimiter=",")
model = PCA(n_components=2)
model.fit(data)
print(model.explained_variance_ratio_)
plt.plot(data[:,0], data[:,1], 'mo')
plt.show()
```

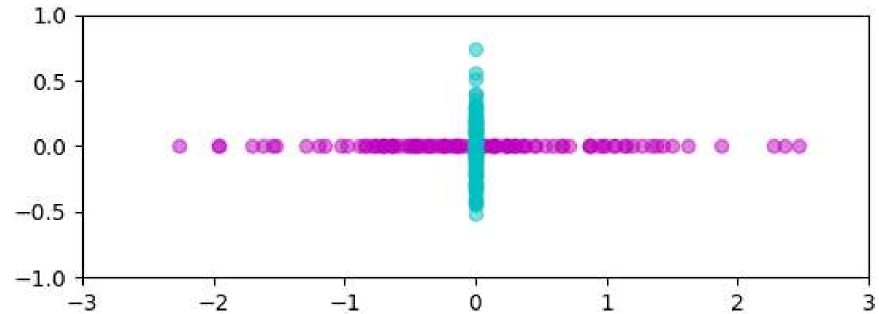
für Datensatz 'decomp_test.dat'
→ 94.06% entlang der ersten Hauptkomponente
→ 5.94% entlang der zweiten Hauptkomponente



## Hauptkomponentenzerlegung

- Principal Component Analysis (Datei 'decomp_test.dat')
  - Punkte auf Hauptkomponenten (Achsen) projizieren

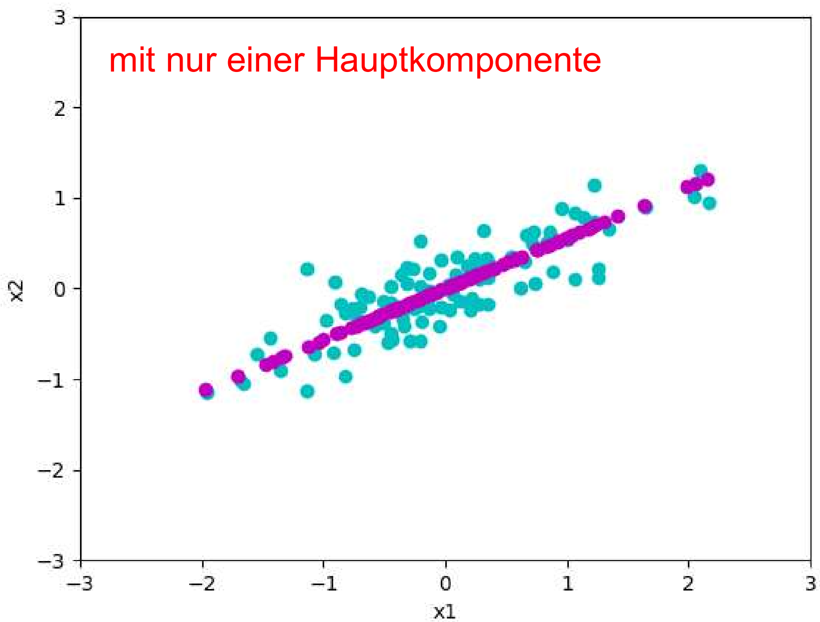
```
model = PCA(n_components=2)
data_proj = model.fit_transform(data)
y = np.zeros([len(data_proj)])
plt.plot(data_proj[:,0], y, 'mo')
plt.plot(y, data_proj[:,1], 'co')
plt.show()
```

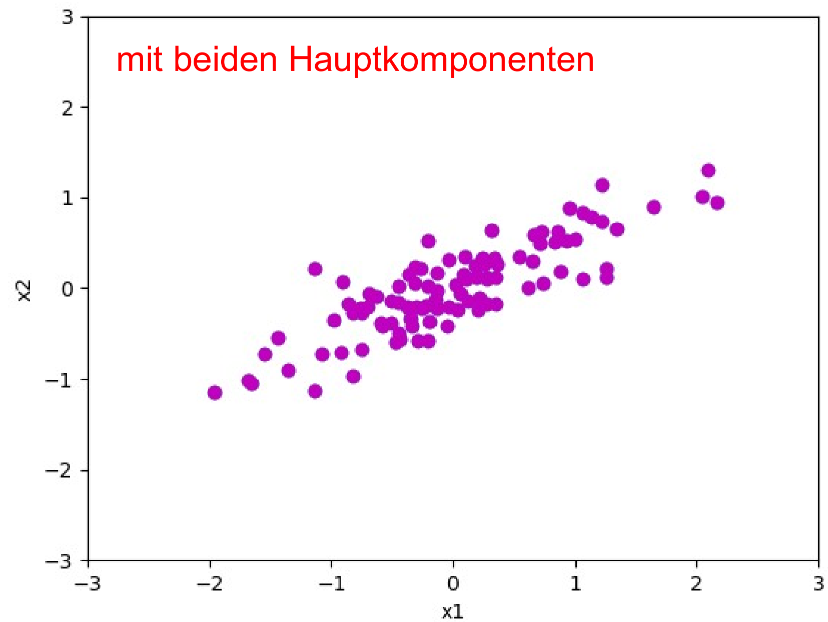


# Hauptkomponentenzerlegung

- Principal Component Analysis (Datei 'decomp_test.dat')
  - Projektion umkehren

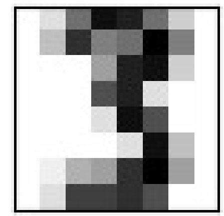
```
data_recovered = model.inverse_transform(data_proj)
```



## Praktische Übung

- Datensatz: handschriftlicher Ziffern (Paket **`sklearn`**)
  - Matrix der Grösse 1797×64 (d.h. 1797 Bilder mit Auflösung 8×8)

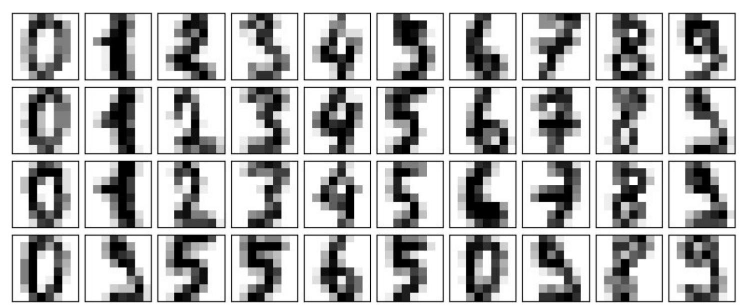


- Datensatz laden und Grösse anzeigen

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

- Datensatz als 4×10 Graubilder anzeigen (Datei **`myplot.py`**)

```
from myplot import plot_digits
plot_digits(digits.data)
plt.show()
```



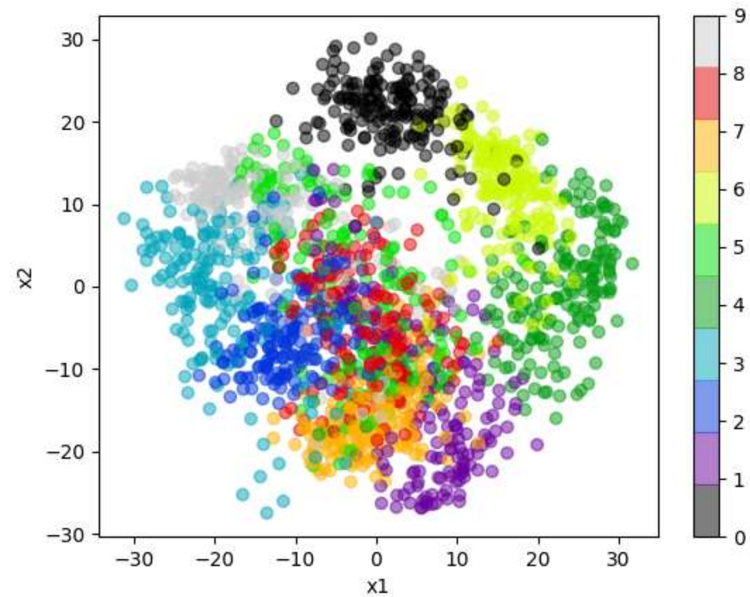
## Praktische Übung

- Dimensionsreduktion auf zwei Hauptkomponenten

```
from sklearn.decomposition import PCA
model = PCA(n_components=2)
d_proj = model.fit_transform(digits.data)
print(model.explained_variance_ratio_)
```

- projizierte Daten als Scatterplot anzeigen

```
plt.scatter(d_proj[:,0], d_proj[:,1],
                   c=digits.target, alpha=0.5)¹
plt.show()
```

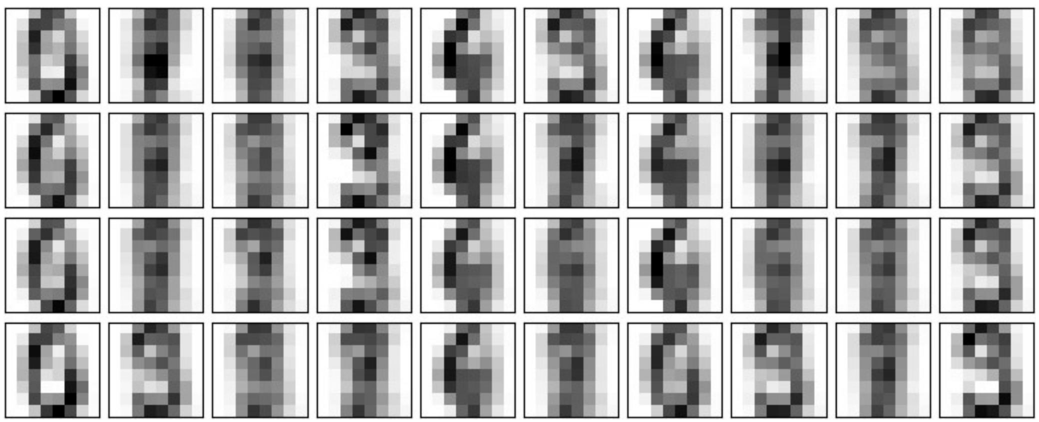



[1] `cmap=plt.cm.get_cmap('nipy_spectral', 10)` und `plt.colorbar()`

## Praktische Übung

- Projektion umkehren und reduzierten Datensatz plotten

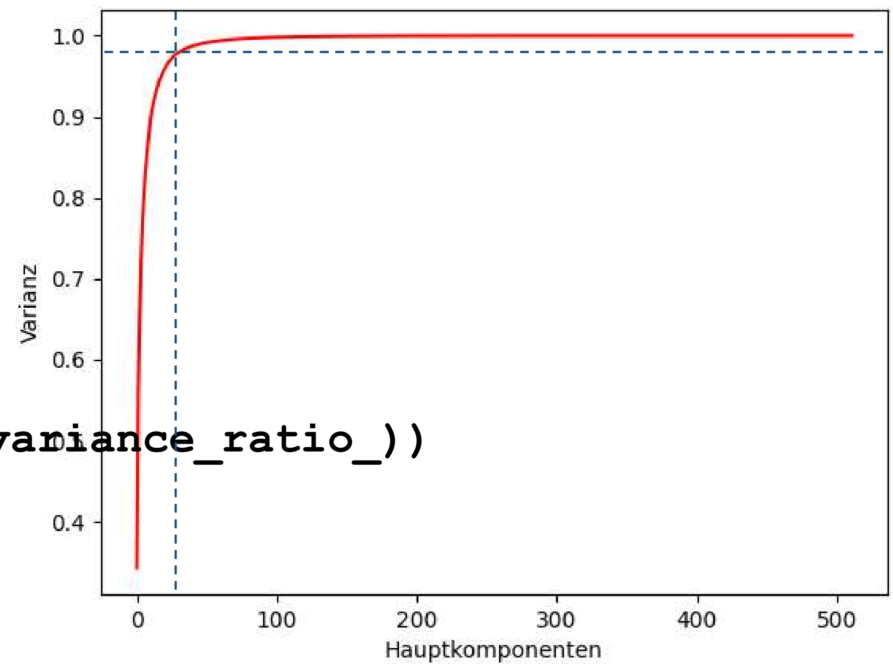
```
d_recov = model.inverse_transform(d_proj)
plot_digits(d_recov)
plt.show()
```



# Mal wieder ein kleiner Blick über den Tellerrand…

- Bild einlesen (Paket **`matplotlib`**) und anzeigen (Datei 'papa.png')

```
img = plt.imread('papa.png')
plt.imshow(img, cmap='gray')
plt.show()
```




- Experiment 1: Verteilung der Varianz

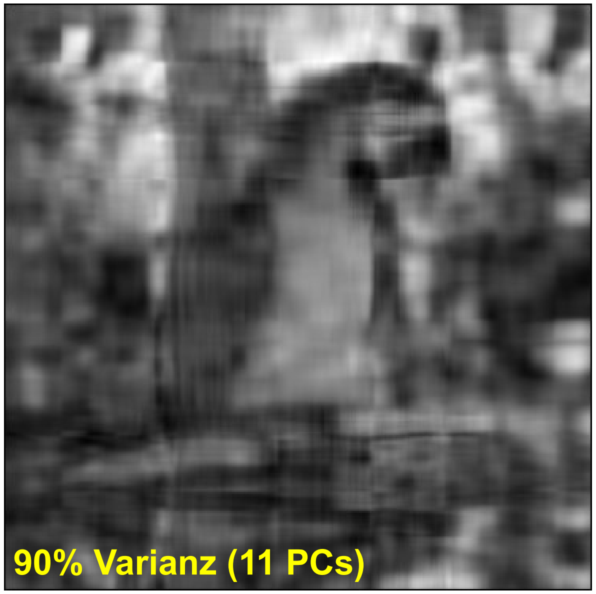
```
model = PCA()
model.fit(img)
plt.plot(np.cumsum(model.explained_variance_ratio_))
plt.show()
```

kumulative Summe

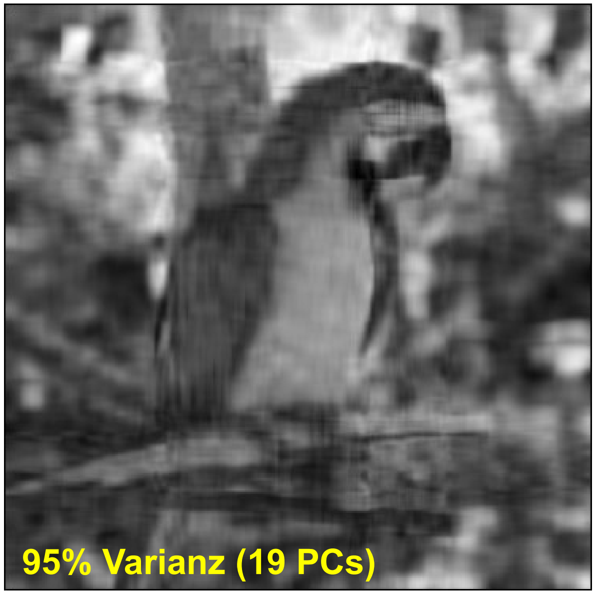
# Mal wieder ein kleiner Blick über den Tellerrand…

- Experiment 2: Hauptkomponentenzerlegung mit *x*% Varianz (0.*x*)

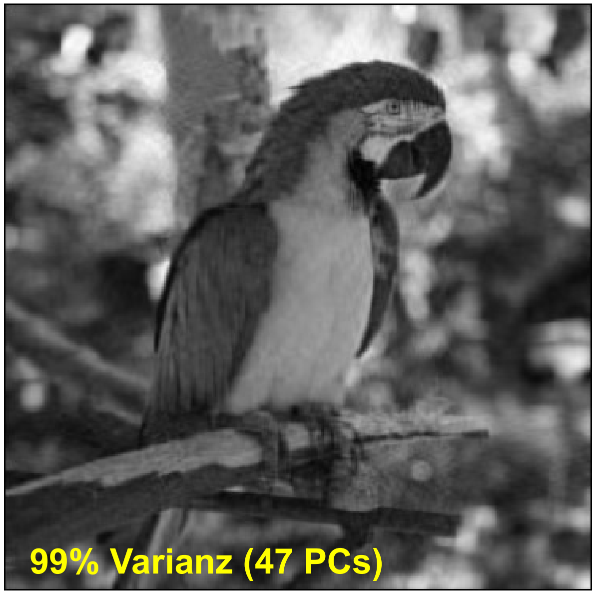
```
model = PCA(0.x)
img_rec = model.inverse_transform(model.fit_transform(img))
```



90% Varianz (11 PCs)



95% Varianz (19 PCs)

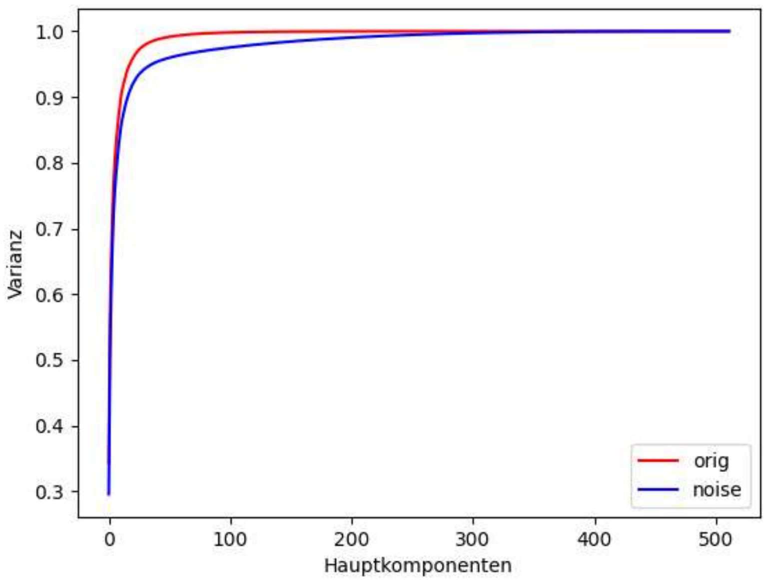


99% Varianz (47 PCs)

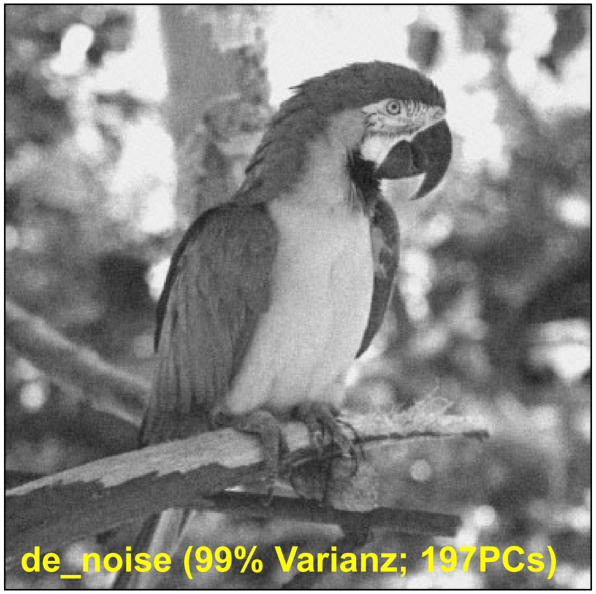
# Mal wieder ein kleiner Blick über den Tellerrand…

- Experiment 3: Entrauschung (Datei 'papa_noise.png')

```
model = PCA(0.99)
de_noise = model.inverse_transform(model.fit_transform(noise))
```







# Fragen…?