Dimensionality Reduction

AP HOGESCHOOL ANTWERPEN

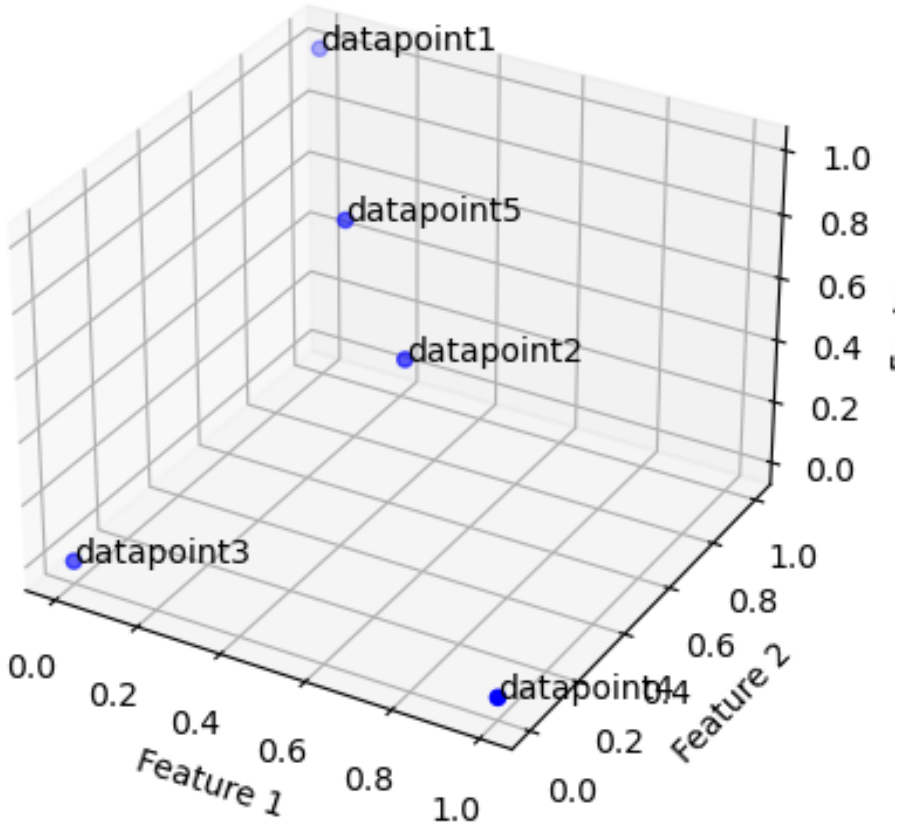
AP.BE

Belangrijke opmerking voor deze les

- In deze les ga ik technieken tonen die, alhoewel ze vaak gebruikt worden en wettelijk perfect OK zijn, op een niet ethische manier kunnen worden gebruikt
- In bepaalde landen worden deze technieken gebruikt om de bevolking te controleren en in de pas te laten lopen
- Gebruik **nooit** facial recognition op data van familie, vrienden, … zonder hun expliciete toestemming. Op het oneigenlijk gebruik van fotomateriaal bestaat trouwens wel wetgeving: dat mag in België niet zomaar.

Features zijn dimensies

	Feature1	Feature2	Feature3
Datapunt 1	0	1	1
Datapunt 2	0.5	0.5	0.5
Datapunt 3	0	0	0
Datapunt 4	1	0	0
Datapunt 5	0.3	0.6	0.8



Het probleem met veel features=dimensies

- We zijn gewoon om figuren te beoordelen met 2 of 3 dimensies
- Wij zijn, als menselijke soort, niet getraind in het interpreteren van beelden met meer dan 3 dimensies

'The curse of dimensionality'

- Als er heel veel dimensies zijn, zijn er ook heel veel mogelijkheden voor de datapunten
  - Het risico op overfitten is groot bij data met heel veel features
  - Er is in een hoog-dimensionale ruimte heel veel 'plaats', daarom het risico op overfitting

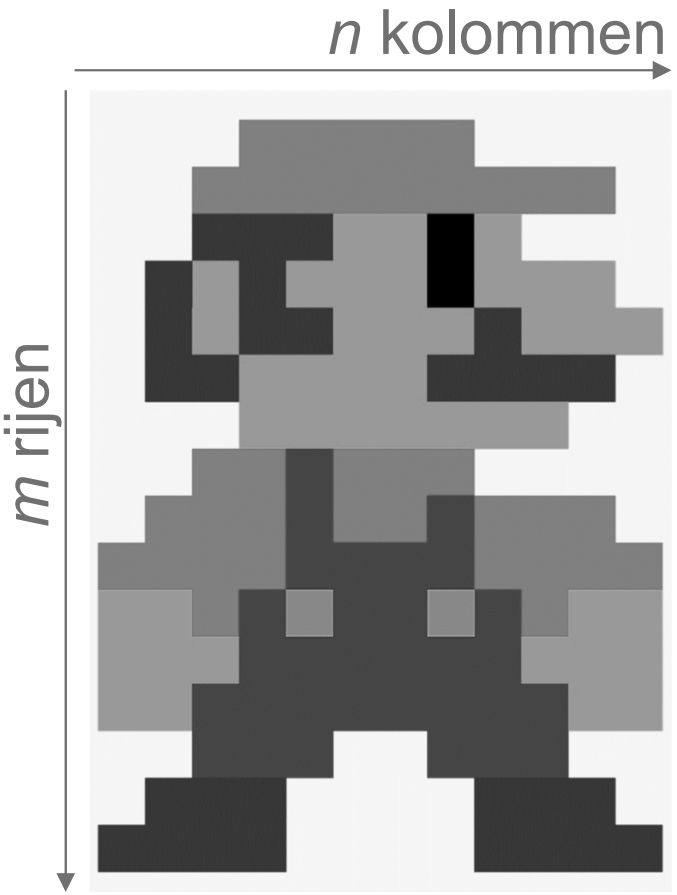
De oplossing voor teveel dimensies

- **Aantal dimensies verlagen**
- Maar: we willen zoveel mogelijk variatie (= 'informatie') van de dataset bijhouden
  - Voorbeeld hiernaast: veel slimmer om **f1** te schrappen dan **f2** of **f3**
- We zouden ook een nieuwe combinatie kunnen maken van f2 en f3, en dit als een nieuwe kolom opslaan, en enkel die bijhouden

Voorbeelddataset	f1	f2	f3
Rij1	1	0	1
Rij2	1	3	2
Rij3	1	-2	1
rij4	1	2	6

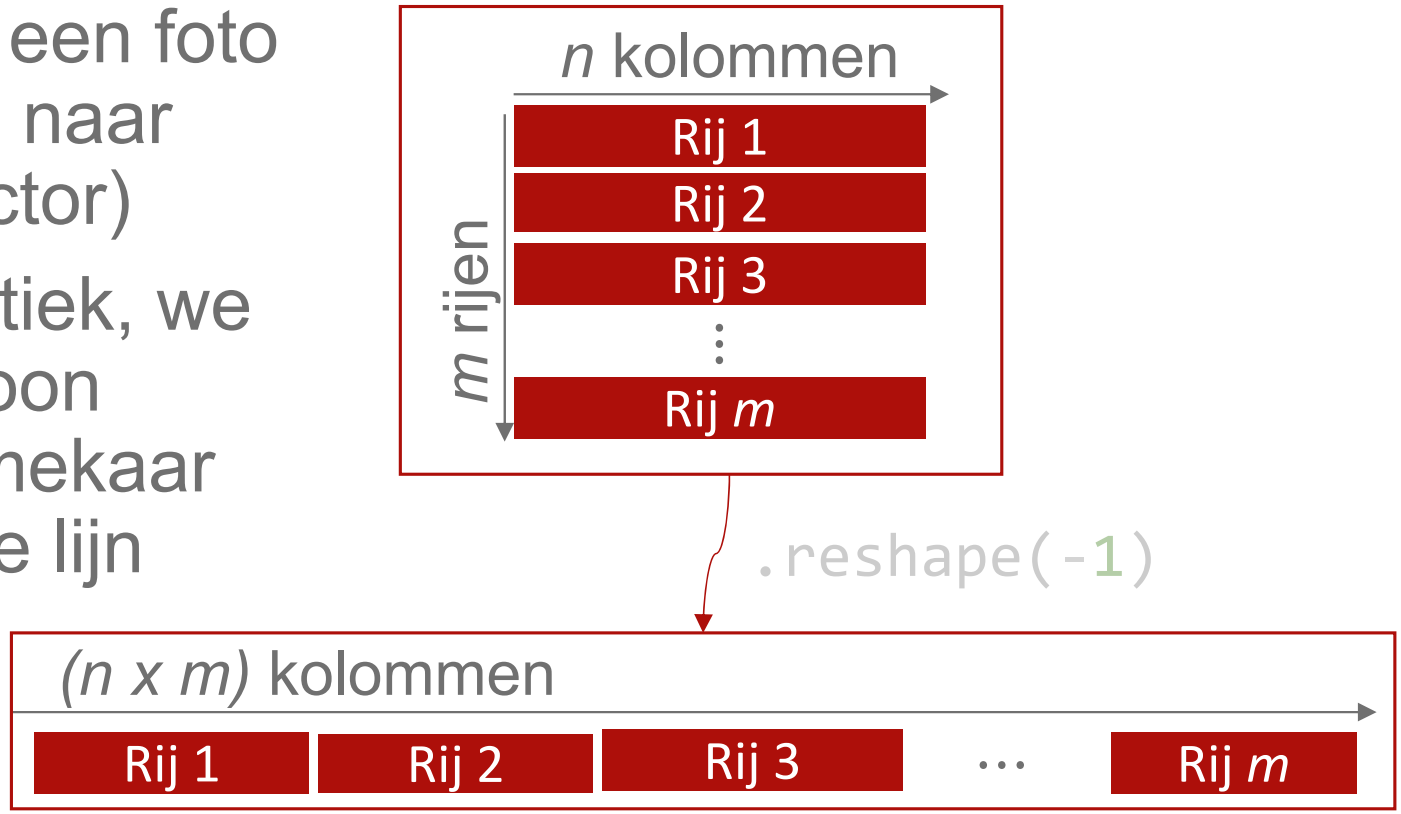
Voorbeeld van dimensiereductie

- We gaan een simpele vorm doen van datareductie, op zwart-witte fotos (ZW-fotos)
- ZW fotos zijn eigenlijk matrices met *m* rijen en *n* kolommen, en elke pixel bevat een waarde tussen 0 en 256



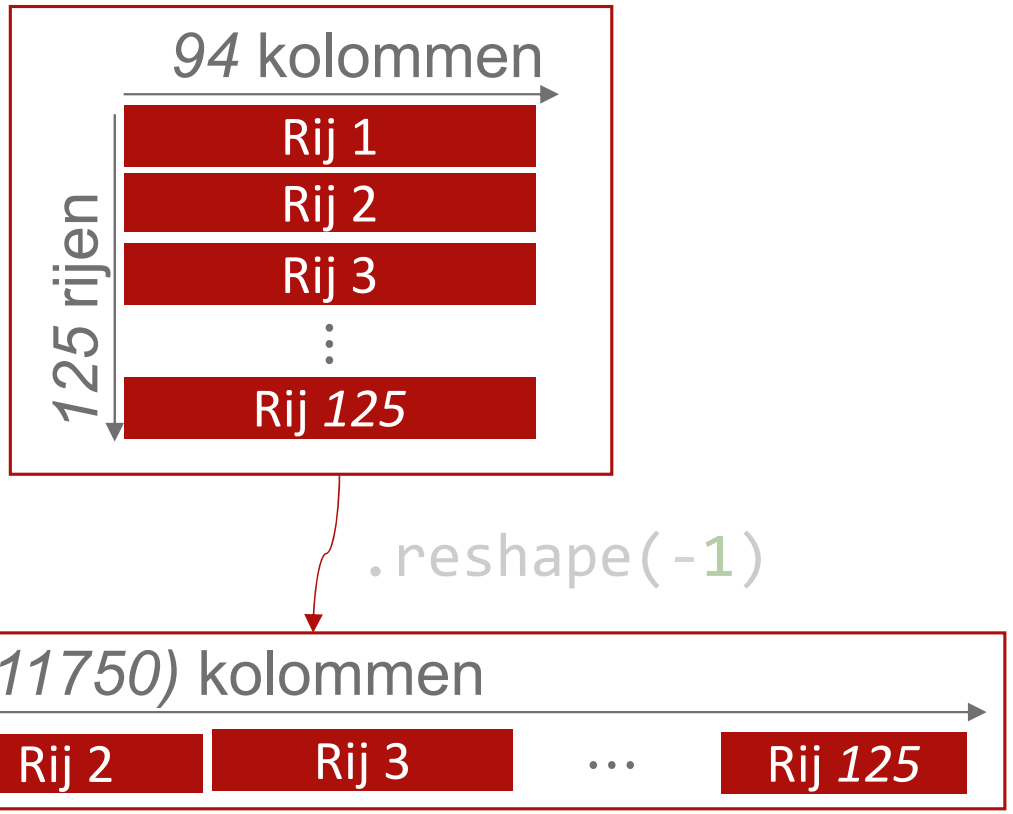
Afbeeldingen platdrukken

- We moeten eerst een foto (matrix) omzetten naar een datapunt (vector)
- De data blijft identiek, we verschuiven gewoon pixels die onder mekaar stonden in 1 lange lijn



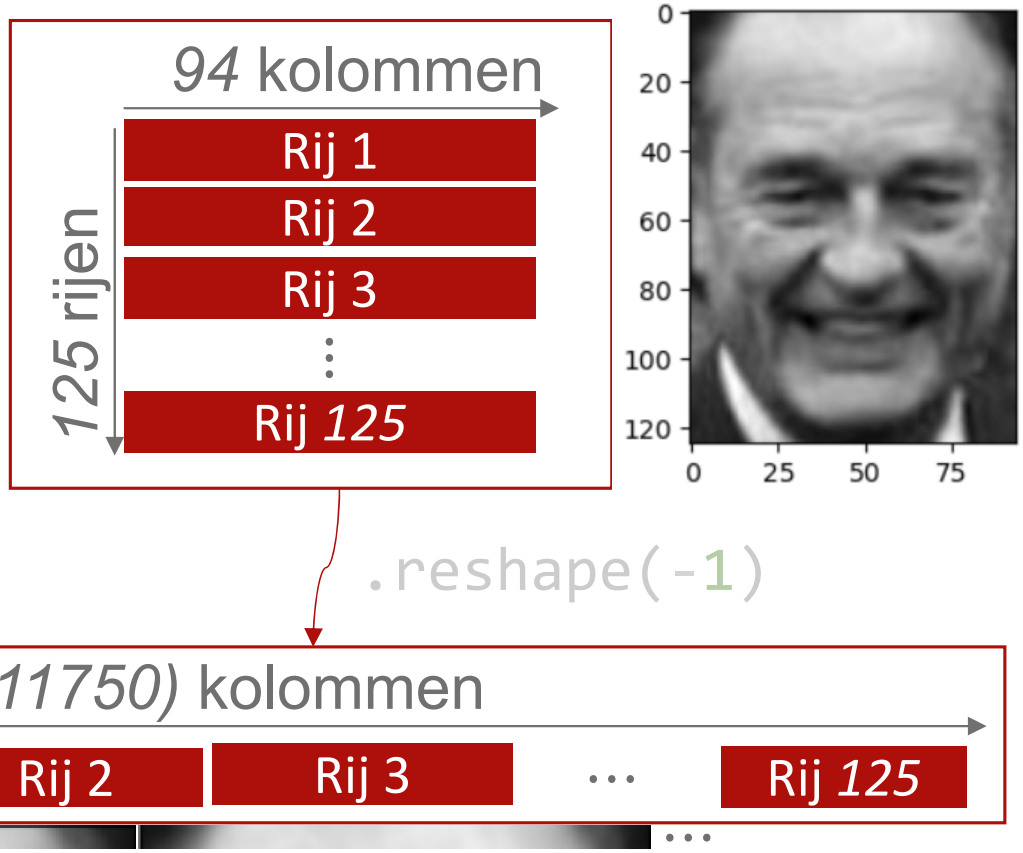
Afbeeldingen platdrukken

- We kijken naar prentjes van 125 pixels hoog en 94 breed
- Een prentje is nu een rij van 11750 waardes tussen 0 en 256.
- Dit proces heet *flattening*



Afbeeldingen platdrukken

- We kijken naar prentjes van 125 pixels hoog en 94 breed
- Een prentje is nu een rij van 11750 waardes (dimensies) tussen 0 en 256.



Fotos inladen

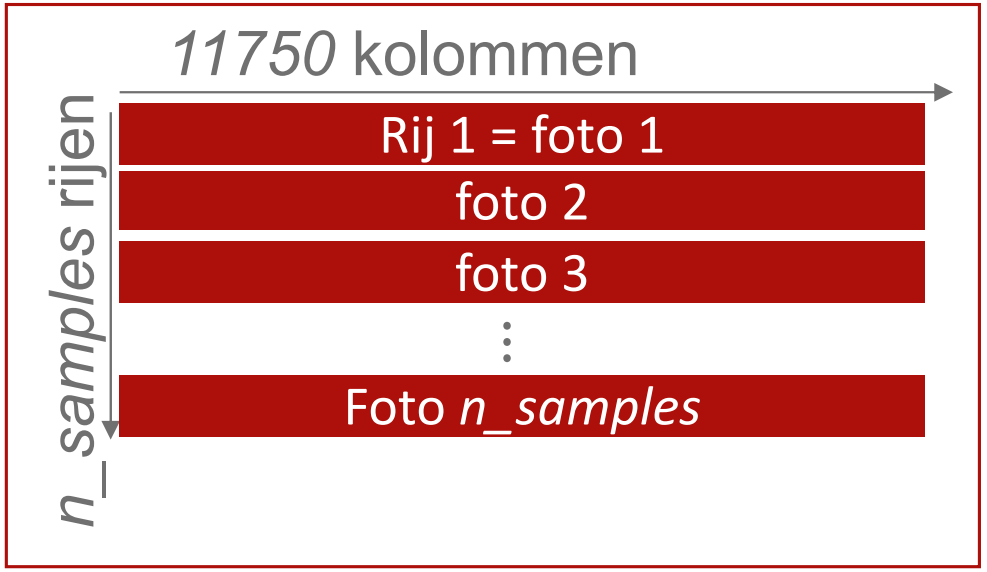
```
from sklearn.datasets import fetch_lfw_people
people =
    fetch_lfw_people(min_faces_per_person=5,
    resize=1)
X = people.data
n_features = X.shape[1]

y = people.target
```

PCA gaat de data automatisch flattenen, dit moet je niet zelf doen

Dimensiereductie op fotos

- Heel de dataset ziet er dan uit als alle fotos als rijen onder mekaar
  - Alle fotos hebben dezelfde afmetingen!
- Een kolom komt overeen met een specifieke pixel in alle fotos op dezelfde locatie



Dimensie reductie

- Doel: aantal dimensies naar beneden halen
- Weinig ‘informatie’ laten verdwijnen
  - Hiervoor gebruiken we **PCA: Principal Component Analysis**

- We hebben in onze dataset (veel meer) features dan datapunten
- Onze trainingsmatrix is dus heel ‘breed’

Features >> samples

- Idealiter wil je een matrix die langer is dan hij breed is

Features < samples

PCA in sklearn

```
pca = PCA(n_components = n_components,  svd_solver = 'randomized’, whiten=True).fit(X_train_scaled)
X_train_pca = pca.transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)


# data
eigenvectors =
    pca.components_.reshape((n_components, h, w))
```

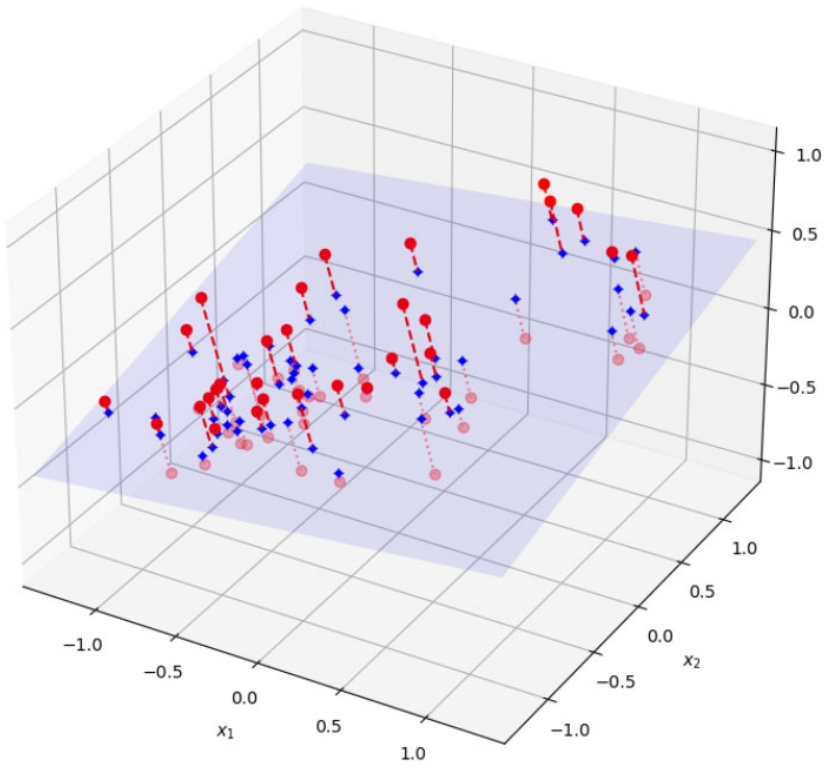
PCA in sklearn

```
pca = PCA(n_components = 0.95,   svd_solver =
'randomized',    whiten=True).fit(X_train_scaled)
```

Nu kiest PCA exact voldoende element zodat 95% van de variatie van je data behouden blijft

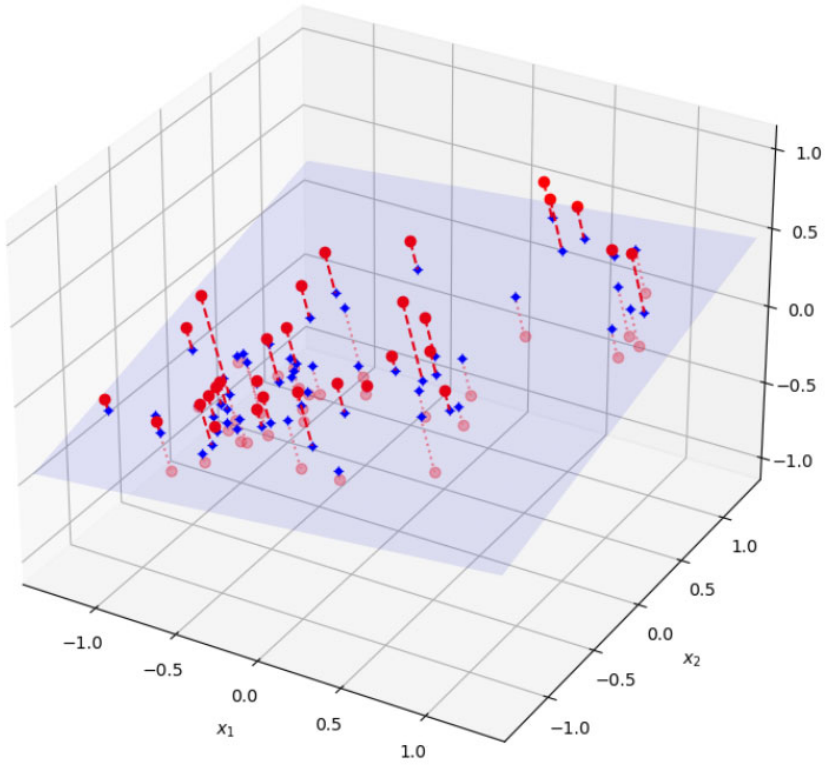
Dimensie reductie: PCA

- Principle Component Analysis (PCA):
  - Zoekt de meest ‘dominante richtingen’ in een dataset
  - Geeft deze terug als **principal components:** hoe minder je er vraagt, hoe sterker de datareductie
  - Alle oorspronkelijke datapunten kunnen worden uitgedrukt als lineaire combinatie van die **principale componenten**



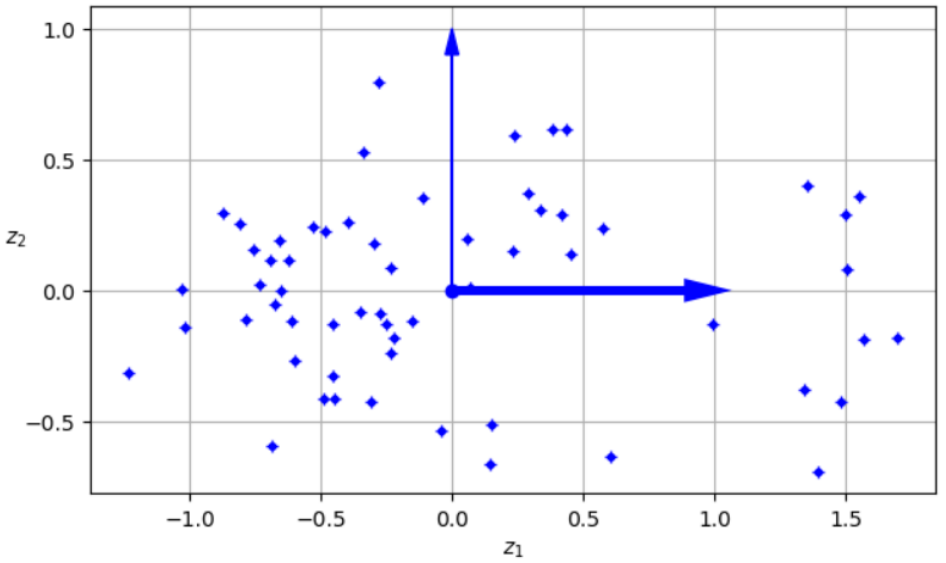
Dimensie reductie: PCA

- Om te begrijpen hoe PCA achterliggend werkt, kijken we nu naar een simpeler probleem met slechts 3 dimensies
- De data ligt hier in de buurt van een vlak. PCA gaat voor ons de data op dit vlak **projecteren**



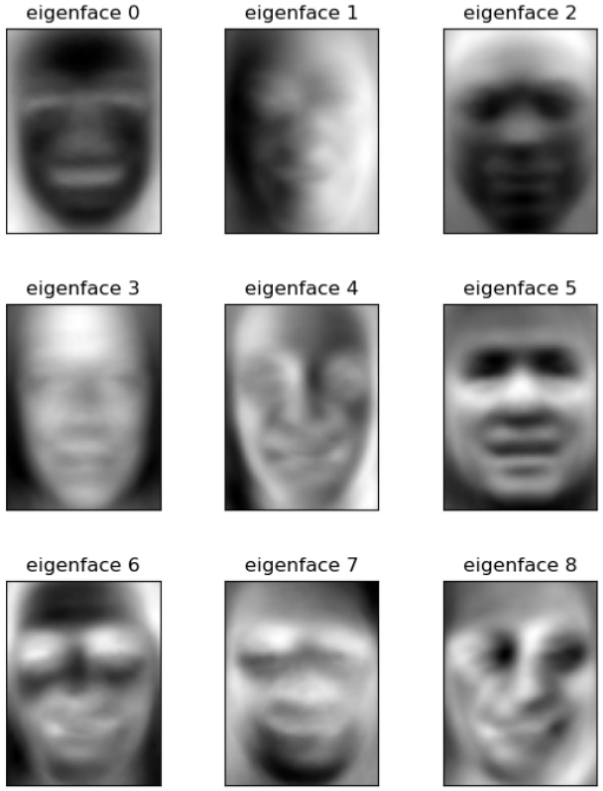
Dimensie reductie: PCA

- PCA berekent automatisch:
  - Het beste vlak om op te projecteren
  - De projecties van alle datapunten
  - Binnen het vlak, **de dominante richtingen**
    - Dit noemen we de '**eigen**vectoren'



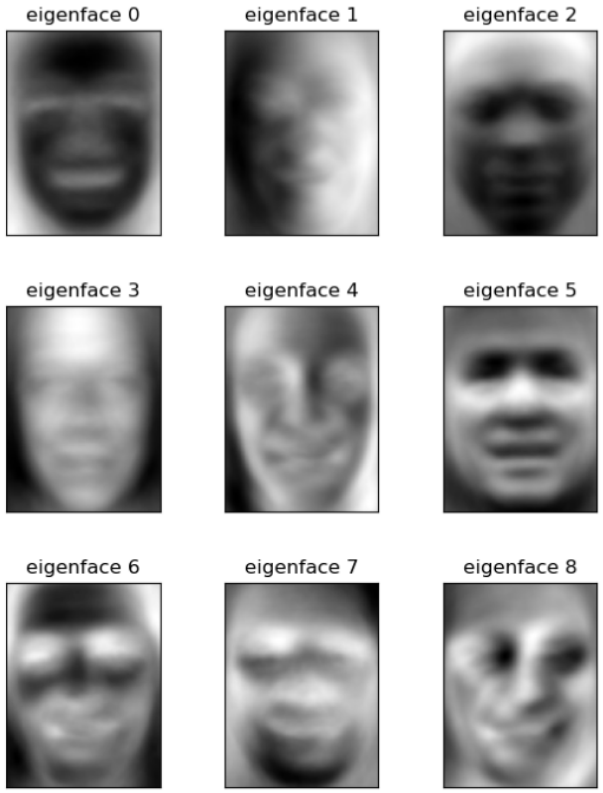
Dimensie reductie: PCA

- Hoe zien die eigenvectoren ‘eigenfaces’ eruit ?
  - **Het zijn generieke gezichten**
  - Elke gezicht kan nu worden uitgedrukt als de som van deze ‘eigenfaces’



Dimensie reductie: PCA

- Hoe zien die ‘eigenfaces’ eruit ?
  - **Het zijn generieke gezichten**
  - Elke gezicht kan nu worden uitgedrukt als de som van deze ‘eigenfaces’



Dimensie reductie: PCA

- Voorbeeld: Kim Clijsters

$$= \sum_{i=1}^{200} c_i \cdot \text{eigenface}_i =$$

-0,56

-0,40

+0,22

+ …

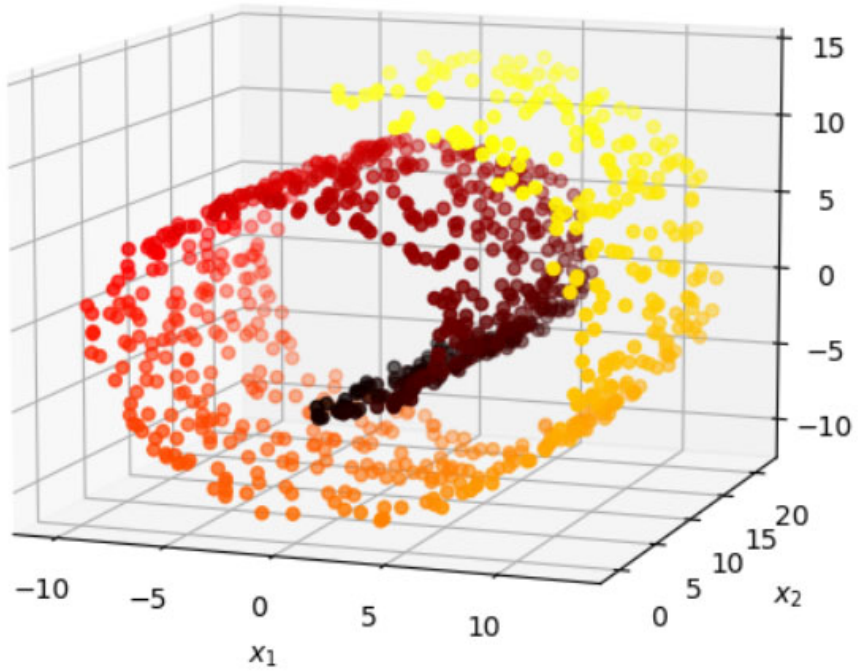
Dimensiereductie met PCA

- PCA behoudt zoveel mogelijk de variantie in een dataset
- De geïdentificeerde principale componenten staan in volgorde: de eerste is de belangrijkste, dan de tweede, etc.
- De reductie bestaat erin de laatste principale componenten 'te laten vallen', eg niet meer te gebruiken, omdat ze weinig waarde bijbrengen.

- Nadelen
  - Heel de dataset moet in memory
    - Dit kan je eventueel oplossen met **IncrementalPCA**
  - PCA werkt met lineaire projecties (rechte bewegingen)
    - Voor complexere datasets kun je **Kernel PCA** gebruiken, met kernels zoals we ook gebruikt hebben in de les rond Support Vector Machines

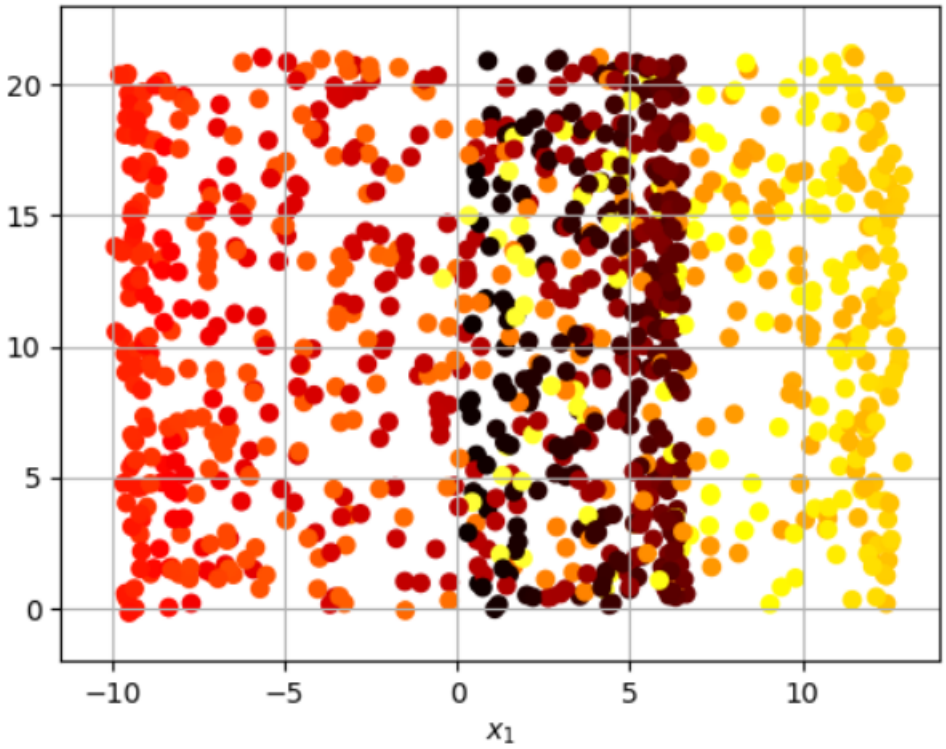
Dimensiereductie met LLE

- Voor niet-lineaire datasets is PCA minder geschikt
- Bekijk de dataset hiernaast (‘swiss roll’)
  - Deze dataset is eigenlijk een soort opgerold blad
  - Dit willen we gebruiken: heel lokaal, lijkt de data op een 2D vlak



Dimensiereductie met LLE

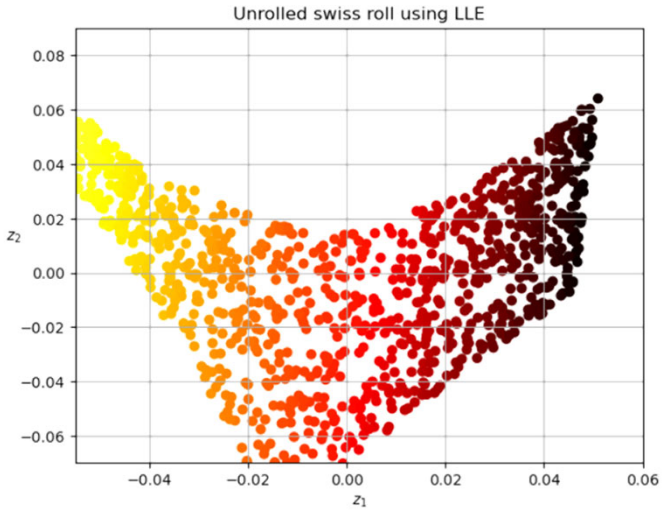
- Door PCA te gebruiken, zouden we die structuur verliezen:
  - In alle richtingen zou na projectie data die nu op een andere plaats is op dezelfde regio worden geprojecteerd:



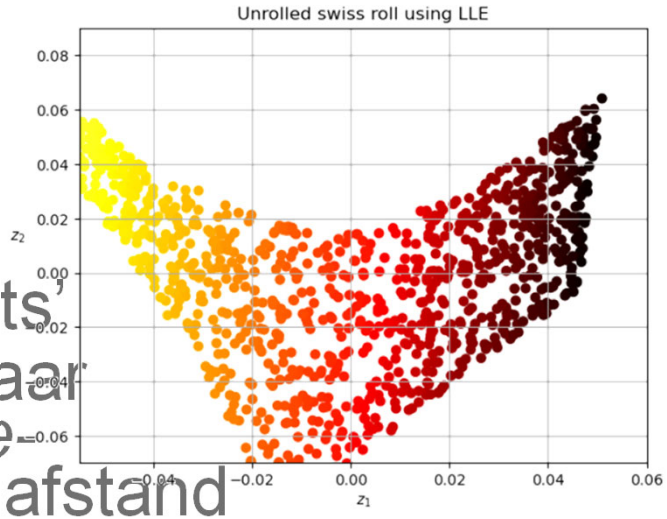
Dimensiereductie met LLE

- LLE:

```
from sklearn.manifold import LocallyLinearEmbedding
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10, random_state=42)
X_unrolled = lle.fit_transform(X_swiss)
```



Dimensiereductie met LLE



- LLE:
  - Vooral geschikt voor 'opgerolde datasets'
  - Punten die oorspronkelijk dicht bij mekaar lagen, blijven dicht bij mekaar (*distance-preserving on local scale*) – op grotere afstand blijven afstanden echter niet bewaard.
- (Werking – niet te kennen):
  - Zoek voor elk punt de *k* dichtstbijzijnde buren
  - Schrijf dit punt nu als lineaire functie van die *k* buren (lineair modelleren van lokale relaties) in een object *W*
  - Projecteer vervolgens naar een lager dimensionele ruimte, en behoudt deze lokale relaties