MATH60026/MATH70026
Methods for Data Science
Lecture 8

Barbara Bravi, Imperial College London

Department of Mathematics, Academic year 2024-2025

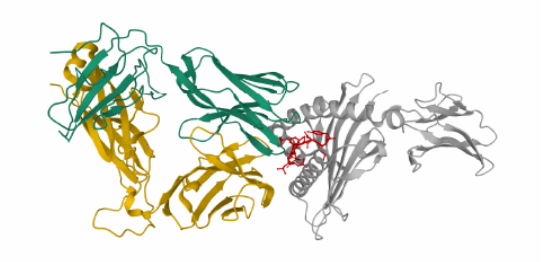
IMPERIAL

What do we mean by data dimensionality reduction?

In many fields, scientists deal with *high-dimensional* data

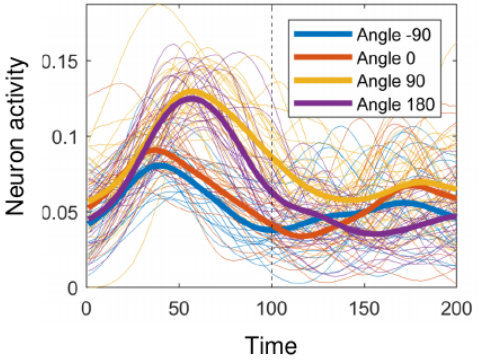
Images

Molecules



Protein complex, from PDB 71NF

Time trajectories

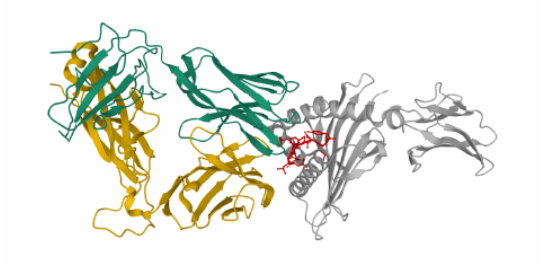


etc.

In many fields, scientists deal with *high-dimensional* data

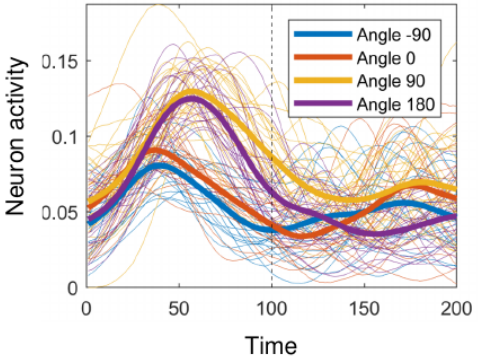
Images

Molecules



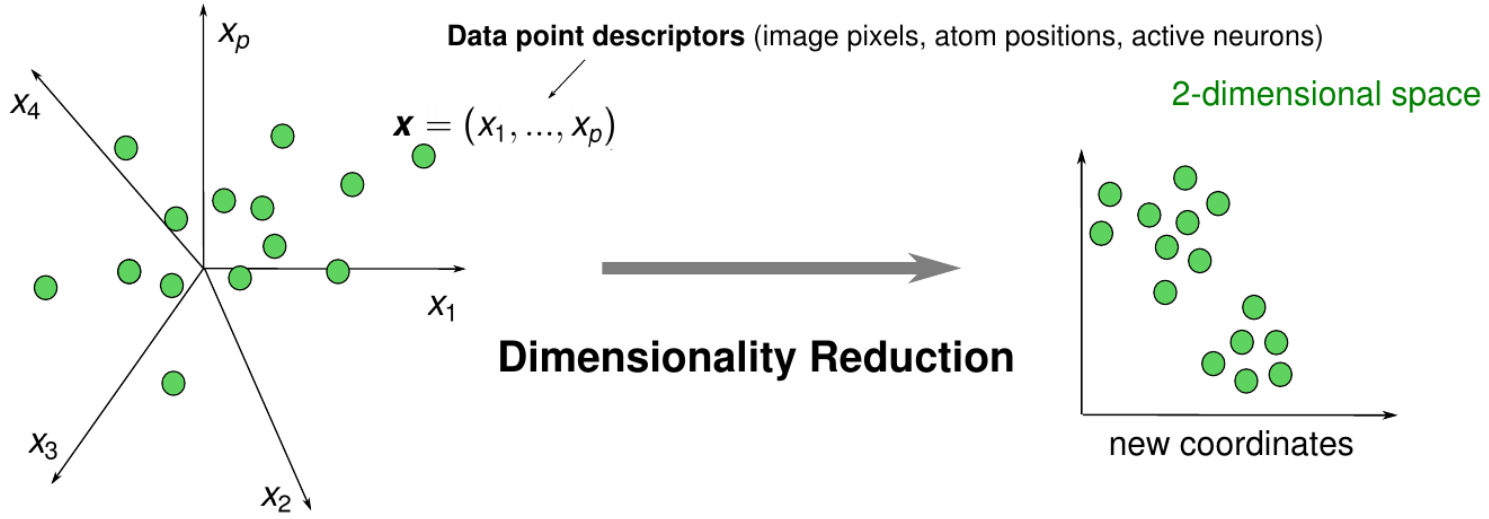
Protein complex, from PDB 71NF

Time trajectories



etc.

p-dimensional space

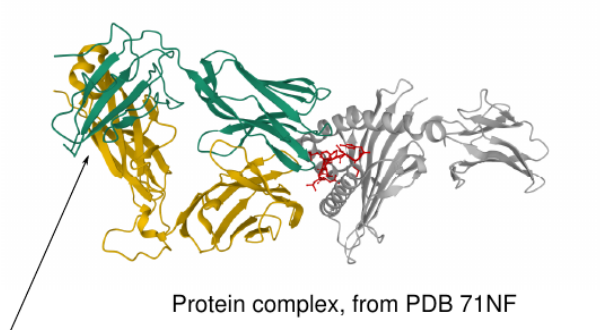


Data Dimensionality Reduction

it's about finding data representations in fewer dimensions

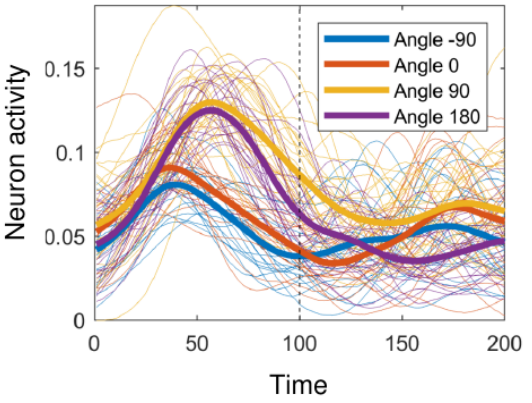
Images

Molecules



Protein complex, from PDB 71NF

Time trajectories

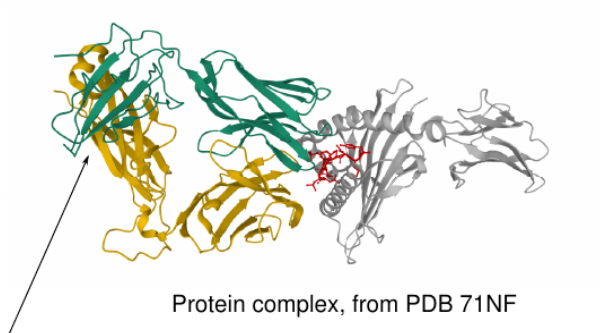


etc.

Data have an **internal structure**, giving rise to **correlations**:
we exploit them to combine dimensions into a few new coordinates

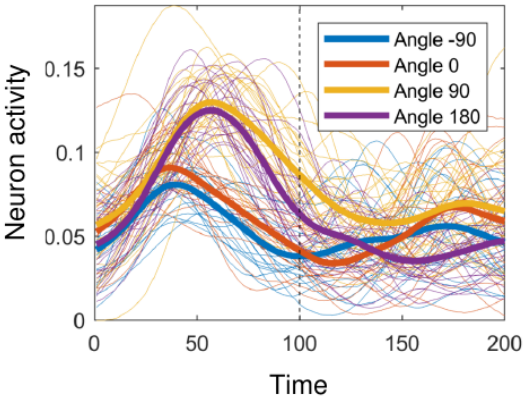
Images

Molecules



Protein complex, from PDB 71NF

Time trajectories



etc.

Data have an **internal structure**, giving rise to **correlations**:
we exploit them to combine dimensions into a few new coordinates

Goals of dimensionality reduction

- To discover properties and patterns within data
- To facilitate further analysis: with fewer variables, building a model becomes more effective, computationally feasible, and interpretable

There are several techniques to perform dimensionality reduction. In this lecture: **Principal Component Analysis**.

Problem setting. We consider the dataset $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ $\mathbf{x}^{(i)} \in \mathbb{R}^p$

Columns are descriptors

Rows are data points

$$\boldsymbol{X}_{N\times p} = \begin{pmatrix} x_1^{(1)} & \dots & x_j^{(1)} & \dots & x_k^{(1)} & \dots & x_p^{(1)} \\ \vdots & \dots & \dots & \dots & \dots & \dots & \vdots \\ x_1^{(i)} & \dots & x_j^{(i)} & \dots & x_k^{(i)} & \dots & x_p^{(i)} \\ \vdots & \dots & \dots & \dots & \dots & \dots & \vdots \\ x_1^{(N)} & \dots & x_j^{(N)} & \dots & x_k^{(N)} & \dots & x_p^{(N)} \end{pmatrix}$$

Problem setting. We consider the dataset $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ $\mathbf{x}^{(i)} \in \mathbb{R}^p$

Columns are descriptors

Rows are data points

$$\boldsymbol{X}_{N\times p} = \begin{pmatrix} x_1^{(1)} & \dots & x_j^{(1)} & \dots & x_k^{(1)} & \dots & x_p^{(1)} \\ \vdots & \dots & \dots & \dots & \dots & \dots & \vdots \\ x_1^{(i)} & \dots & x_j^{(i)} & \dots & x_k^{(i)} & \dots & x_p^{(i)} \\ \vdots & \dots & \dots & \dots & \dots & \dots & \vdots \\ x_1^{(N)} & \dots & x_j^{(N)} & \dots & x_k^{(N)} & \dots & x_p^{(N)} \end{pmatrix}$$

$$C_{jk} = \frac{1}{N}\sum_{i=1}^{N}(x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k) \qquad \boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}^{(i)}$$

An important statistical quantity for PCA is the **covariance matrix**

Principal Component Analysis (PCA)

PCA seeks a lower-dimensional representation in terms of an orthogonal data projection:

$$a_j^{(i)} = \mathbf{v}_j \cdot \mathbf{x}^{(i)}$$

for some collection of *orthogonal* vectors:

$$\{\mathbf{v}_j\}_{j=1}^{m}, \quad \mathbf{v}_j \in \mathbb{R}^p$$

Principal Component Analysis (PCA)

PCA seeks a lower-dimensional representation in terms of an orthogonal data projection:

$$a_j^{(i)} = \mathbf{v}_j \cdot \mathbf{x}^{(i)}$$

Coordinates of the data in the *m*-dimensional subspace the principal components

for some collection of *orthogonal* vectors:

$$\{\mathbf{v}_j\}_{j=1}^{m}, \quad \mathbf{v}_j \in \mathbb{R}^p$$

Directions spanning a *m*-dimensional subspace onto which we project the data - the **Principal Components (PCs)**

Principal Component Analysis (PCA)

PCA seeks a lower-dimensional representation in terms of an orthogonal data projection:

$$a_j^{(i)} = \mathbf{v}_j \cdot \mathbf{x}^{(i)}$$

Coordinates of the data in the *m*-dimensional subspace the principal components

for some collection of *orthogonal* vectors:

$$\{\mathbf{v}_j\}_{j=1}^{m}, \quad \mathbf{v}_j \in \mathbb{R}^p$$

Directions spanning a *m*-dimensional subspace onto which we project the data - the **Principal Components (PCs)**

To have dimensionality reduction, we choose $m \ll p$

Principal Component Analysis (PCA)

$a_j^{(i)}$: low-dimensional, compressed representations of $\mathbf{x}^{(i)}$

From it, we can construct an approximation of $\mathbf{x}^{(i)}$

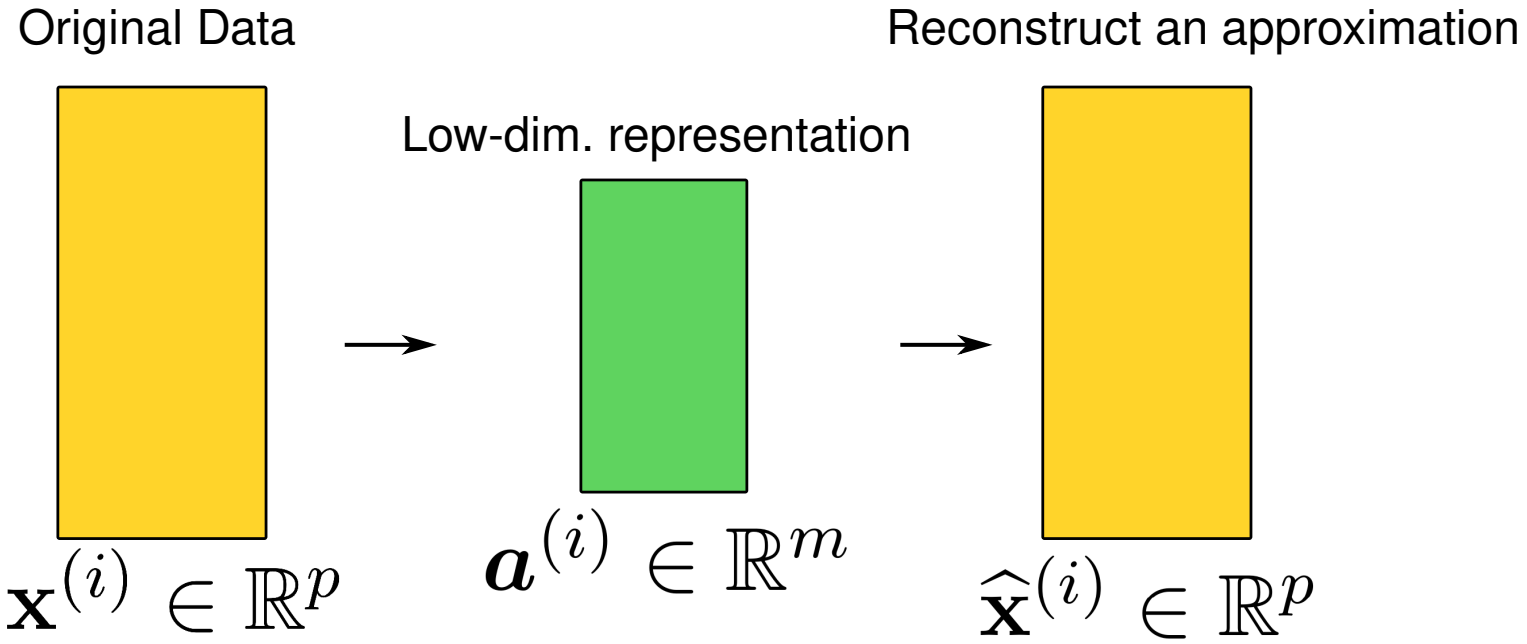
$$\widehat{\mathbf{x}}^{(i)} = \sum_{j=1}^{m} a_j^{(i)} \mathbf{v}_j$$

Principal Component Analysis (PCA)

$a_j^{(i)}$: low-dimensional, compressed representations of $\mathbf{x}^{(i)}$

From it, we can construct an approximation of $\mathbf{x}^{(i)}$

$$\widehat{\mathbf{x}}^{(i)} = \sum_{j=1}^{m} a_j^{(i)} \mathbf{v}_j$$



Minimising the approximation error

The principal components are found by minimising the error associated to PCA. Each data point can be written as a *linear* combination:

$$\mathbf{x}^{(i)} = \sum_{j=1}^{p} a_j^{(i)} \mathbf{v}_j$$

In PCA, we want to approximate by a projection on *m* dimensions:

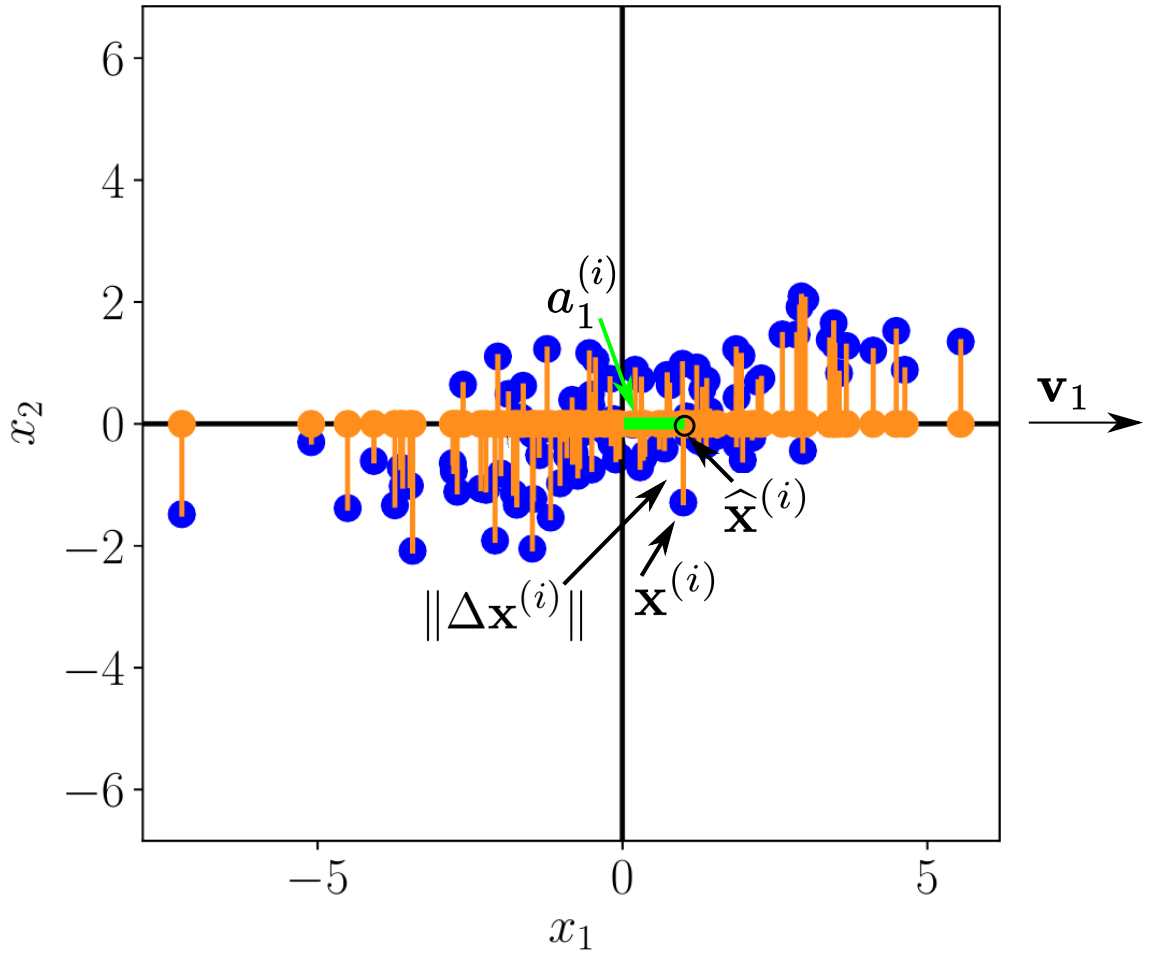
$$\widehat{\mathbf{x}}_m^{(i)} = \sum_{j=1}^{m} a_j^{(i)} \mathbf{v}_j + \underbrace{\sum_{j=m+1}^{p} b_j \mathbf{v}_j}_{\text{Error}}$$

The error of this approximation:

$$\Delta\mathbf{x}^{(i)} = \mathbf{x}^{(i)} - \widehat{\mathbf{x}}_m^{(i)} = \sum_{j=m+1}^{p} [a_j^{(i)} - b_j]\mathbf{v}_j$$

Sample-independent

Minimising the approximation error



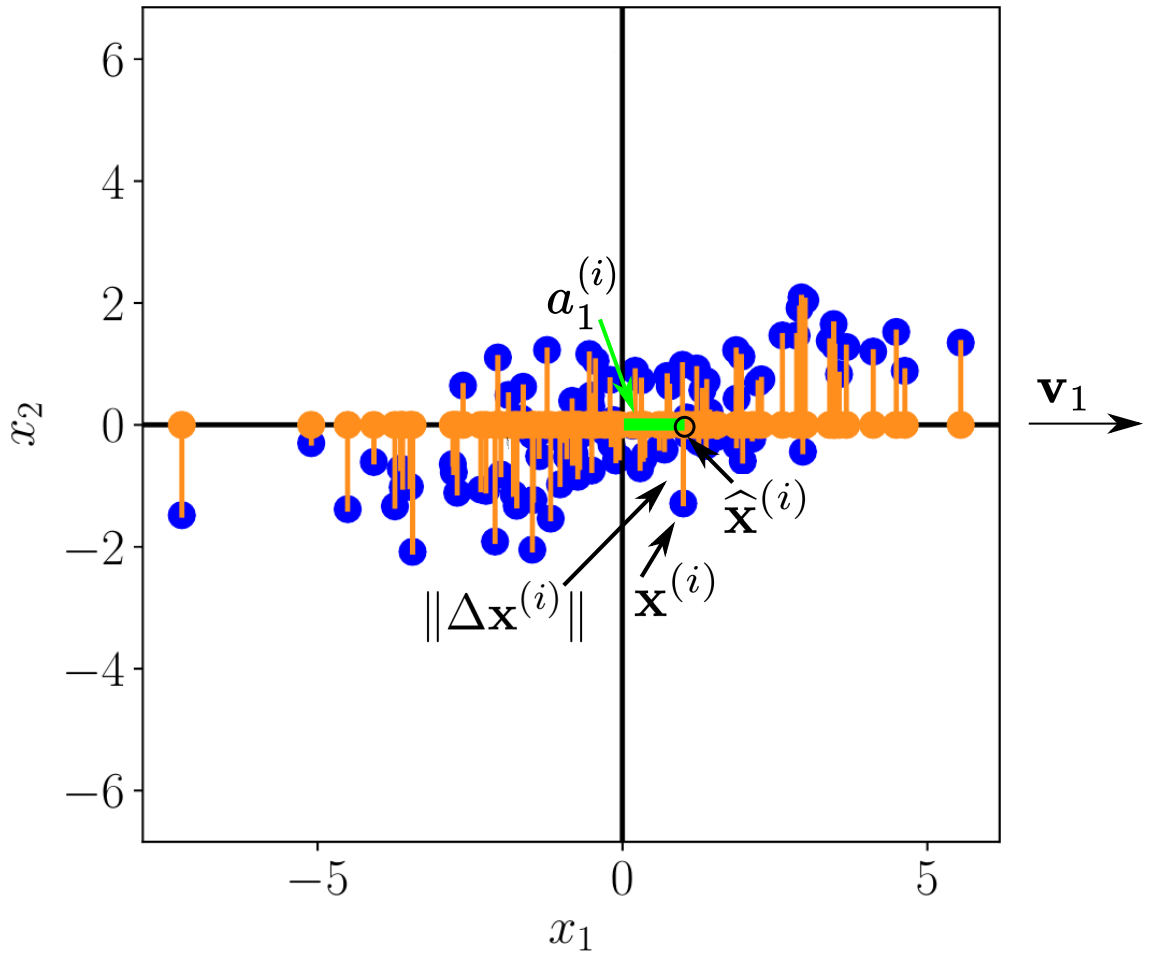
$$\underbrace{a_1^{(i)} = \mathbf{v}_1 \cdot \mathbf{x}^{(i)}}_{\text{Projection of } \mathbf{x}^{(i)} \text{ onto } 1^{st} \text{ PC}}$$

So the approximation in the 1-dim. space spanned by $\mathbf{v}_1$:

$$\widehat{\mathbf{x}}^{(i)} = a_1^{(i)} \mathbf{v}_1$$

Figures from M.P. Deisenroth, A.A. Faisal, and C. Soon Ong, Mathematics for Machine Learning 2020

Minimising the approximation error



$$\underbrace{a_1^{(i)} = \mathbf{v}_1 \cdot \mathbf{x}^{(i)}}_{\text{Projection of } \mathbf{x}^{(i)} \text{ onto } 1^{st} \text{ PC}}$$

So the approximation in the 1-dim. space spanned by $\mathbf{v}_1$:

$$\widehat{\mathbf{x}}^{(i)} = a_1^{(i)} \mathbf{v}_1$$

PCA can be derived as the linear dimensionality reduction that minimises the approximation error $\|\Delta\mathbf{x}^{(i)}\|$

Figures from M.P. Deisenroth, A.A. Faisal, and C. Soon Ong, Mathematics for Machine Learning 2020

Minimising the approximation error

Specifically one minimises the Mean Squared Error (MSE):

$$\begin{aligned}
\mathrm{MSE} &= \frac{1}{N}\sum_{i=1}^{N}\|\Delta\mathbf{x}^{(i)}\|^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{j,k=m+1}^{p}(a_j^{(i)}-b_j)(a_k^{(i)}-b_k)\mathbf{v}_j\cdot\mathbf{v}_k\right] \\
&= \frac{1}{N}\sum_{i=1}^{N}\sum_{j=m+1}^{p}(a_j^{(i)}-b_j)^2
\end{aligned}$$

Using: $\mathbf{v}_j\cdot\mathbf{v}_k = \delta_{jk} := \begin{cases} 1 & \text{if } j=k \\ 0 & \text{otherwise} \end{cases}$

Minimising the approximation error

Specifically one minimises the Mean Squared Error (MSE):

$$\begin{aligned}
\mathrm{MSE} &= \frac{1}{N}\sum_{i=1}^{N}\|\Delta\mathbf{x}^{(i)}\|^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{j,k=m+1}^{p}(a_j^{(i)}-b_j)(a_k^{(i)}-b_k)\mathbf{v}_j\cdot\mathbf{v}_k\right] \\
&= \frac{1}{N}\sum_{i=1}^{N}\sum_{j=m+1}^{p}(a_j^{(i)}-b_j)^2
\end{aligned}$$

Using: $\mathbf{v}_j\cdot\mathbf{v}_k = \delta_{jk} := \begin{cases} 1 & \text{if } j=k \\ 0 & \text{otherwise} \end{cases}$

We look for the *optimal* $\mathbf{v}_j, b_j$:

$$\mathbf{v}_j, b_j = \operatorname{argmin} MSE, \quad \{\mathbf{v}_j\}_{j=1}^{p}, \{b_j\}_{j=m+1}^{p}$$

Minimising the approximation error

We need to derivate the MSE wrt $\mathbf{v}_j$ and b_j :

$$\frac{\partial \text{MSE}}{\partial b_j}\Big|_{b_j} = 0 \quad \Longrightarrow \quad b_j = \frac{1}{N}\sum_{i=1}^{N} a_j^{(i)}$$

Leading to:

$$\text{MSE} = \sum_{j=m+1}^{p} \frac{1}{N}\sum_{i=1}^{N} \Big(\underbrace{\mathbf{x}^{(i)} \cdot \mathbf{v}_j}_{a_j^{(i)}} - \underbrace{\frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)} \cdot \mathbf{v}_j}_{b_j} \Big)^2$$

$$= \sum_{j=m+1}^{p} \frac{1}{N}\sum_{i=1}^{N} \left[\left(\mathbf{x}^{(i)} - \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)} \right) \cdot \mathbf{v}_j \right]^2$$

$$= \sum_{j=m+1}^{p} \mathbf{v}_j^T \left(\frac{1}{N}\sum_{i=1}^{N} \left[\left(\mathbf{x}^{(i)} - \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)} \right) \left(\mathbf{x}^{(i)} - \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)} \right)^T \right] \right) \mathbf{v}_j$$

Minimising the approximation error

We need to derivate the MSE wrt $\mathbf{v}_j$ and b_j :

$$\frac{\partial \text{MSE}}{\partial b_j}\Big|_{b_j} = 0 \quad \Longrightarrow \quad b_j = \frac{1}{N}\sum_{i=1}^{N} a_j^{(i)}$$

Leading to:

$$\begin{aligned}
\text{MSE} &= \sum_{j=m+1}^{p} \frac{1}{N}\sum_{i=1}^{N} \Big(\underbrace{\mathbf{x}^{(i)} \cdot \mathbf{v}_j}_{a_j^{(i)}} - \underbrace{\frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)} \cdot \mathbf{v}_j}_{b_j} \Big)^2 \\
&= \sum_{j=m+1}^{p} \frac{1}{N}\sum_{i=1}^{N} \left[\left(\mathbf{x}^{(i)} - \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)} \right) \cdot \mathbf{v}_j \right]^2 \\
&= \sum_{j=m+1}^{p} \mathbf{v}_j^T \underbrace{\left(\frac{1}{N}\sum_{i=1}^{N} \left[\left(\mathbf{x}^{(i)} - \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)} \right) \left(\mathbf{x}^{(i)} - \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)} \right)^T \right] \right)}_{(C_{\mathbf{x}})_{p\times p}} \mathbf{v}_j
\end{aligned}$$

Which can be rewritten compactly as:

$$\boxed{\text{MSE} = \sum_{j=m+1}^{p} \mathbf{v}_j^T C_{\mathbf{x}}\, \mathbf{v}_j}$$

Minimising the approximation error

There is the constraint of an **orthonormal** $\mathbf{v}_j$ set, we need Lagrange multipliers. We perform constrained optimisation of the Lagrangian:

$$\mathcal{L} = \underbrace{\sum_{j=m+1}^{p} \mathbf{v}_j^T C_{\mathbf{x}} \mathbf{v}_j}_{\text{MSE}} + \underbrace{\sum_{j=m+1}^{p} \lambda_j (1 - \mathbf{v}_j^T \mathbf{v}_j)}_{\text{Enforce normality}}$$

Minimising the approximation error

There is the constraint of an **orthonormal** $\mathbf{v}_j$ set, we need Lagrange multipliers. We perform constrained optimisation of the Lagrangian:

$$\mathcal{L} = \underbrace{\sum_{j=m+1}^{p} \mathbf{v}_j^T C_{\mathbf{x}} \mathbf{v}_j}_{\text{MSE}} + \underbrace{\sum_{j=m+1}^{p} \lambda_j (1 - \mathbf{v}_j^T \mathbf{v}_j)}_{\text{Enforce normality}}$$

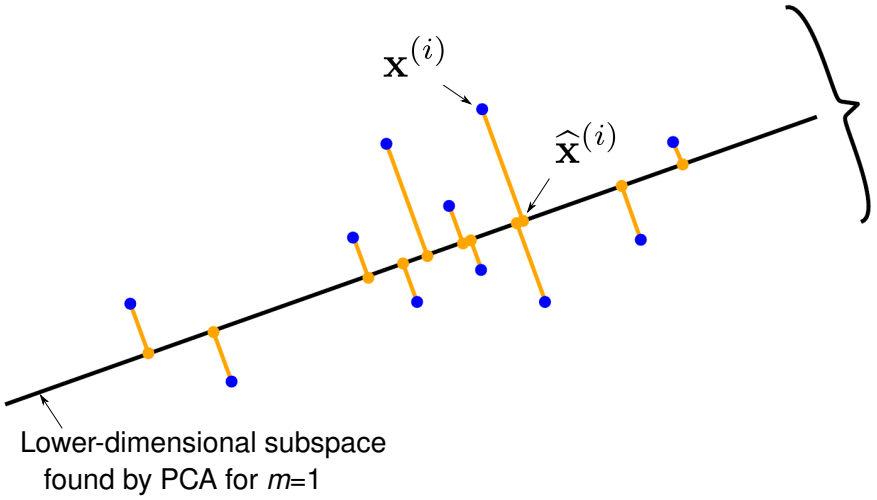
We seek for the minimum of the Lagrangian:

$$\min_{\mathbf{v}_j, \lambda_j} \mathcal{L}$$

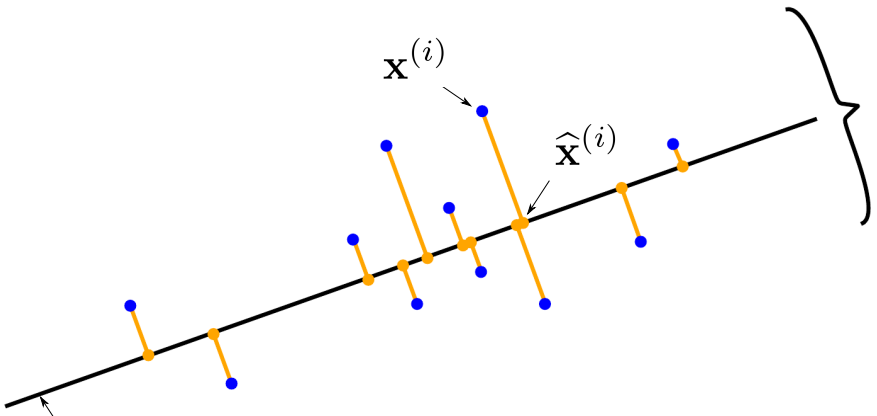
$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}_j} = 0 \quad \Longrightarrow \quad \boxed{C_{\mathbf{x}} \mathbf{v}_j = \lambda_j \mathbf{v}_j}$$

The principal components are the eigenvectors of the covariance matrix

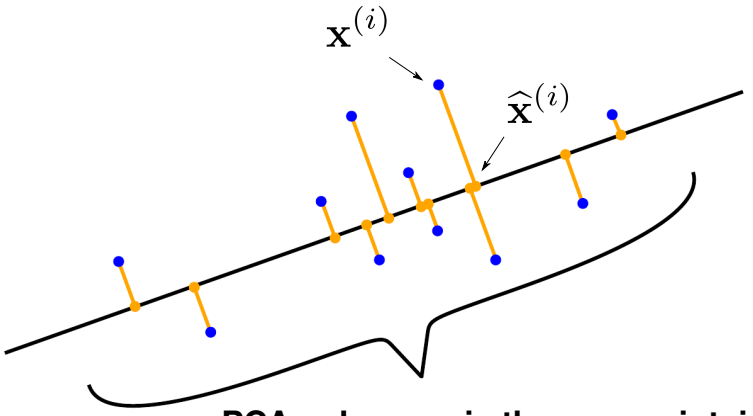
PCA subspace minimises the difference between approximated and original data



PCA subspace minimises the difference between approximated and original data



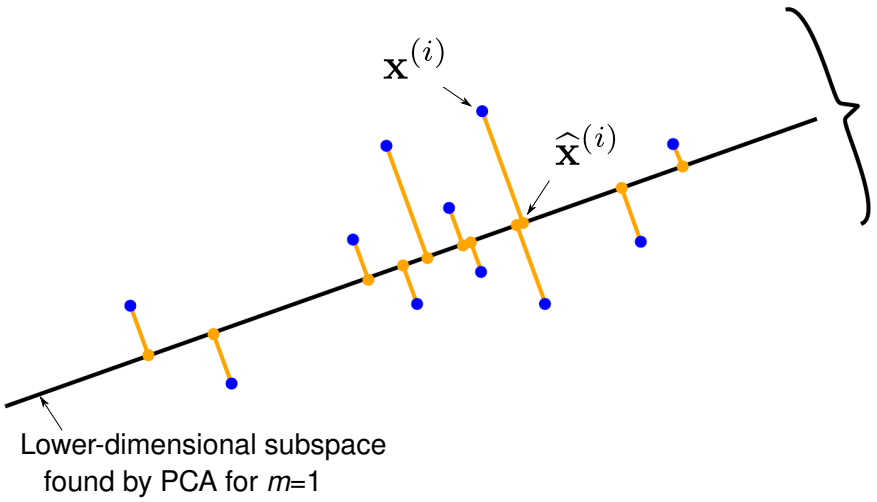
Lower-dimensional subspace found by PCA for m=1

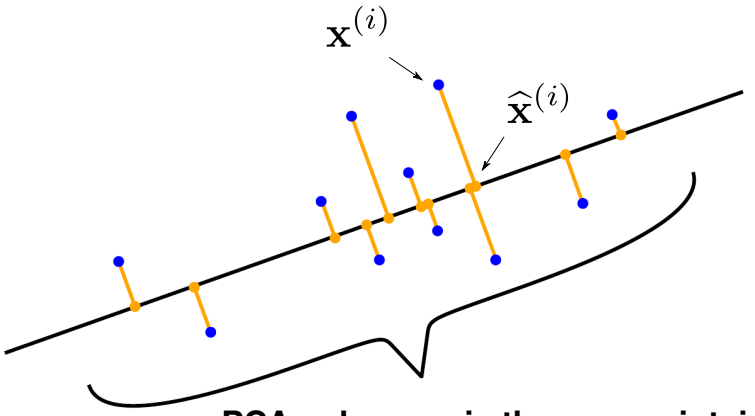


PCA subspace is the one maintaining as much variance of the data as possible

Figures from M.P. Deisenroth, A.A. Faisal, and C. Soon Ong, Mathematics for Machine Learning 2020

PCA subspace minimises the difference between approximated and original data





PCA subspace is the one maintaining as much variance of the data as possible

PCA can be derived by maximising the variance of the data representation!

Let's take the variance of the projection onto one direction (the first PC):

$$V_1 = Var(a_1^{(i)}) \qquad \text{where} \quad a_1^{(i)} = \mathbf{v}_1 \cdot \mathbf{x}^{(i)}$$

$$V_1 = \frac{1}{N}\sum_{i=1}^{N}\left(a_1^{(i)} - Mean(a_1^{(i)})\right)^2$$

$$= \mathbf{v}_1^T \underbrace{\frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}\right)\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}\right)^T}_{\text{Covariance } C_{\mathbf{x}}} \mathbf{v}_1 = \boxed{\mathbf{v}_1^T C_{\mathbf{x}} \mathbf{v}_1}$$

Figures from M.P. Deisenroth, A.A. Faisal, and C. Soon Ong, Mathematics for Machine Learning 2020

Maximising the variance

The first PC can be found by maximising V_1 under the orthonormality constraint:

$$\max_{\mathbf{v}_1, \lambda_1} \mathcal{L} \qquad \mathcal{L} = \mathbf{v}_1^T C_{\mathbf{x}} \mathbf{v}_1 + \underbrace{\lambda_1 (1 - \mathbf{v}_1^T \mathbf{v}_1)}_{\text{Enforce normality}}$$

Maximising the variance

The first PC can be found by maximising V_1 under the orthonormality constraint:

$$\max_{\mathbf{v}_1, \lambda_1} \mathcal{L} \qquad \mathcal{L} = \mathbf{v}_1^T C_{\mathbf{x}} \mathbf{v}_1 + \underbrace{\lambda_1 (1 - \mathbf{v}_1^T \mathbf{v}_1)}_{\text{Enforce normality}}$$

But this leads to the same solution as before!

$$C_{\mathbf{x}} \mathbf{v}_1 = \lambda_1 \mathbf{v}_1$$

The first PC is an eigenvector of the covariance matrix

$$V_1 = \mathbf{v}_1{}^T C_{\mathbf{x}} \mathbf{v}_1 = \lambda_1 \mathbf{v}_1 \cdot \mathbf{v}_1 = \lambda_1$$

To maximise the variance, the first PC corresponds to the largest eigenvalue

Maximising the variance

The first PC can be found by maximising V_1 under the orthonormality constraint:

$$\max_{\mathbf{v}_1, \lambda_1} \mathcal{L} \qquad \mathcal{L} = \mathbf{v}_1^T C_{\mathbf{x}} \mathbf{v}_1 + \underbrace{\lambda_1 (1 - \mathbf{v}_1^T \mathbf{v}_1)}_{\text{Enforce normality}}$$

But this leads to the same solution as before!

$$C_{\mathbf{x}} \mathbf{v}_1 = \lambda_1 \mathbf{v}_1$$

The first PC is an eigenvector of the covariance matrix

$$V_1 = \mathbf{v}_1{}^T C_{\mathbf{x}} \mathbf{v}_1 = \lambda_1 \mathbf{v}_1 \cdot \mathbf{v}_1 = \lambda_1$$

To maximise the variance, the first PC corresponds to the largest eigenvalue

Following the same logic (i.e., max variance under orthonormality constraint) gives:

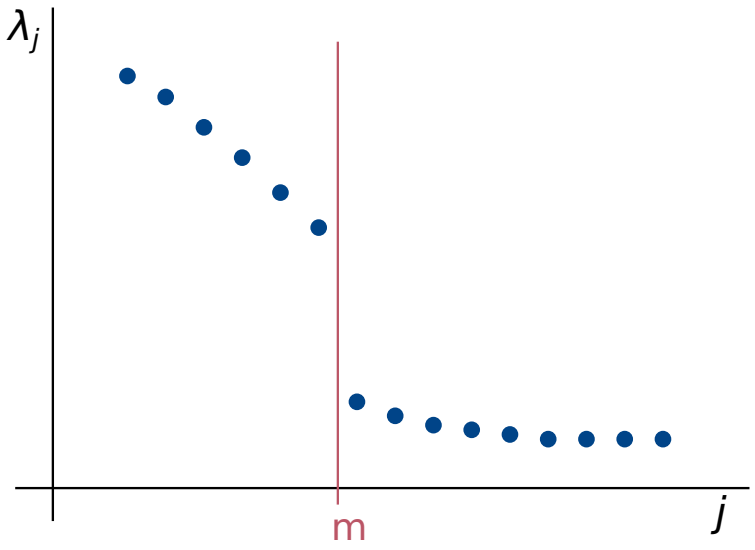
$C_{\mathbf{x}} \mathbf{v}_j = \lambda_j \mathbf{v}_j$ choosing $j = 1, ..., m,$ λ_j in decreasing order

Covariance eigenvalue spectrum

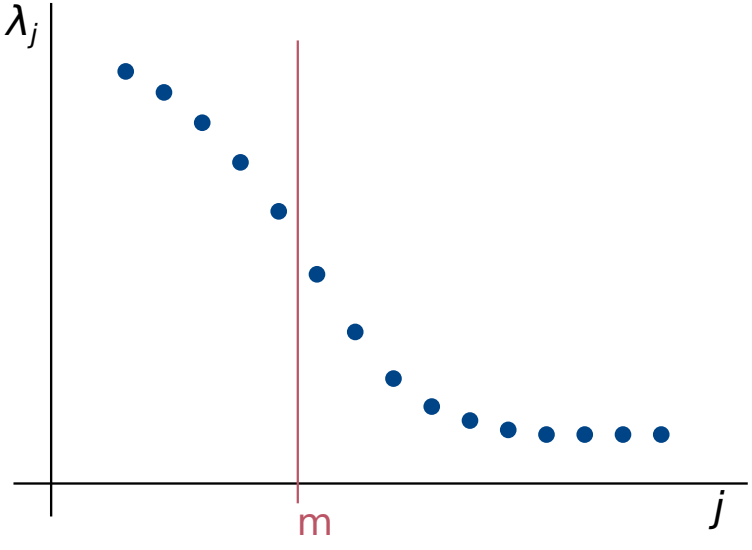
In summary, in PCA we need to diagonalise the covariance matrix:

$$C_{\mathbf{x}}\mathbf{v}_j = \lambda_j \mathbf{v}_j$$

How does the spectrum of these eigenvalues look like?
And how do we choose *m*?



ideal case



standard case

Setting the optimal m

By taking the first m principal components:

Total Variance $V = \sum_{j=1}^{p} \lambda_j$

Captured Variance $V_m = \sum_{j=1}^{m} \lambda_j$

Fraction of Captured Variance $\frac{V_m}{V}$

Fraction of Residual Variance $1 - \frac{V_m}{V}$

Setting the optimal *m*

By taking the first *m* principal components:

Total Variance $V = \sum_{j=1}^{p} \lambda_j$

Captured Variance $V_m = \sum_{j=1}^{m} \lambda_j$

Fraction of Captured Variance $\frac{V_m}{V}$

Fraction of Residual Variance $1 - \frac{V_m}{V}$

Which is equivalent to the MSE:

$$\text{MSE} = \sum_{j=m+1}^{p} \mathbf{v}_j{}^T C_{\mathbf{x}} \mathbf{v}_j = \sum_{j=m+1}^{p} \mathbf{v}_j{}^T \lambda_j \mathbf{v}_j = \sum_{j=m+1}^{p} \lambda_j$$

When choosing *m,* we want:
the residual variance, or, equivalently, the MSE to be low

Setting the optimal *m*

By taking the first *m* principal components:

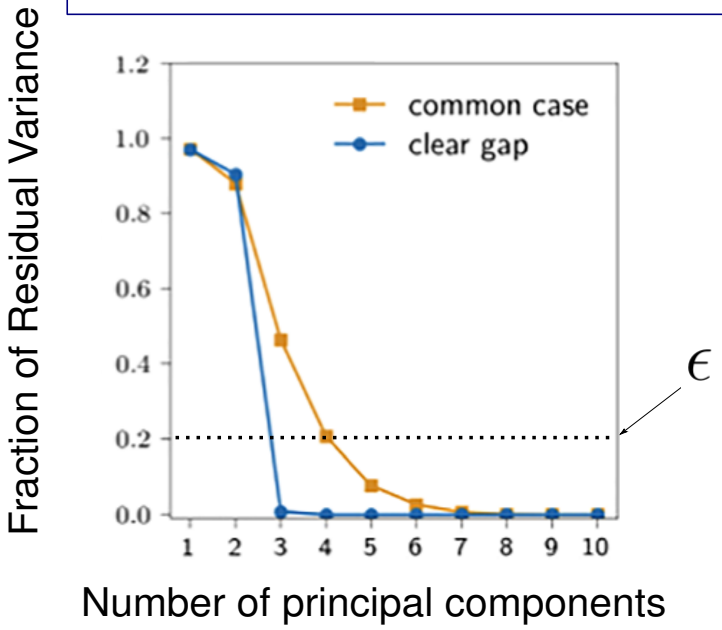
Set *m* such that $1 - \frac{V_m}{V} < \epsilon$

Total Variance $V = \sum_{j=1}^{p} \lambda_j$

Captured Variance $V_m = \sum_{j=1}^{m} \lambda_j$

Fraction of Captured Variance $\frac{V_m}{V}$

Fraction of Residual Variance $1 - \frac{V_m}{V}$

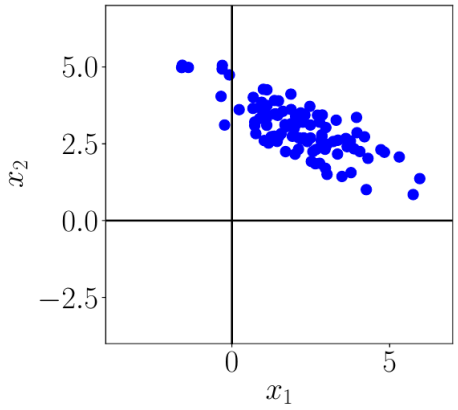


Which is equivalent to the MSE:

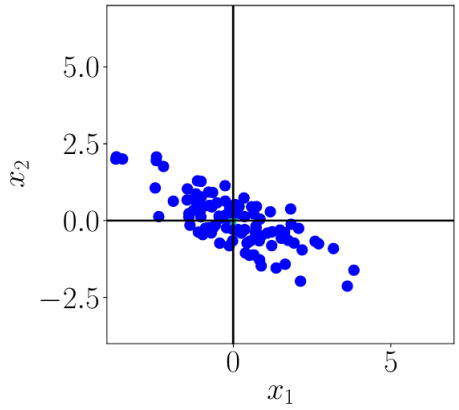
$$\text{MSE} = \sum_{j=m+1}^{p} \mathbf{v}_j{}^T C_{\mathbf{x}} \mathbf{v}_j = \sum_{j=m+1}^{p} \mathbf{v}_j{}^T \lambda_j \mathbf{v}_j = \sum_{j=m+1}^{p} \lambda_j$$

When choosing *m,* we want:
the residual variance, or, equivalently, the MSE to be low

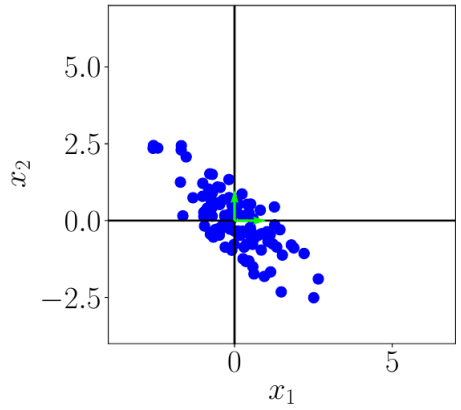
PCA steps in practice



(a) Original dataset.



(b) Step 1: Centering by subtracting the mean from each data point.

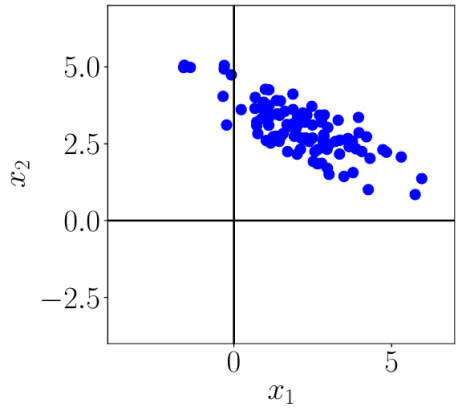


(c) Step 2: Dividing by the standard deviation to make the data unit free. Data has variance 1 along each axis.

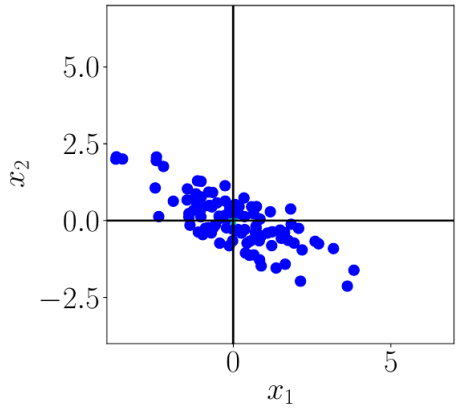
Standardising the data is important to ensure that the PCs capture only the *directions* of maximal variance

Figures from M.P. Deisenroth, A.A. Faisal, and C. Soon Ong, Mathematics for Machine Learning 2020

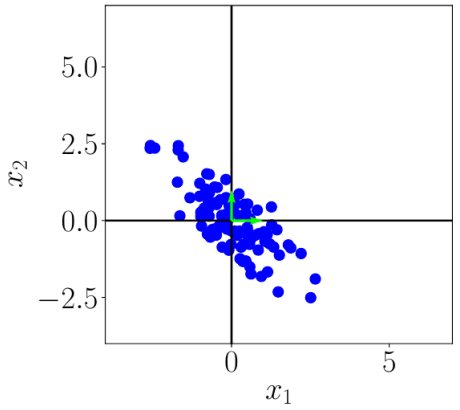
PCA steps in practice



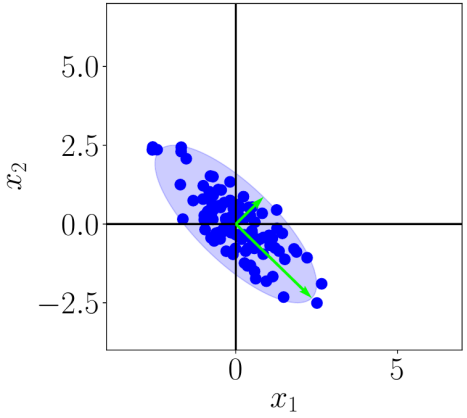
(a) Original dataset.



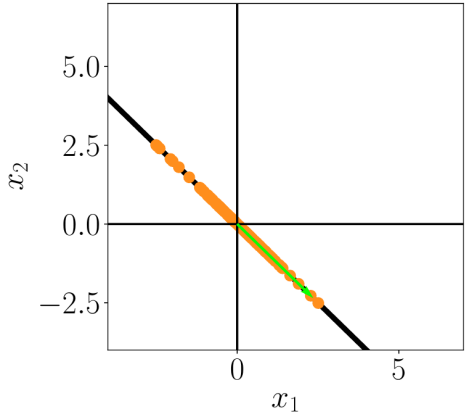
(b) Step 1: Centering by subtracting the mean from each data point.



(c) Step 2: Dividing by the standard deviation to make the data unit free. Data has variance 1 along each axis.



(d) Step 3: Compute eigenvalues and eigenvectors (arrows) of the data covariance matrix (ellipse).



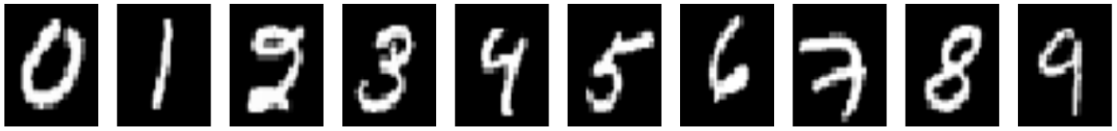
(e) Step 4: Project data onto the principal subspace.

Standardising the data is important to ensure that the PCs capture only the *directions* of maximal variance

Figures from M.P. Deisenroth, A.A. Faisal, and C. Soon Ong, Mathematics for Machine Learning 2020

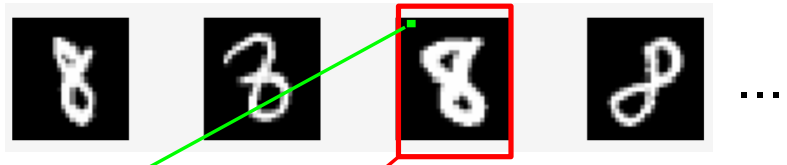
Example of application: PCA on image data

MNIST digits



MNIST digits: handwritten 0-9 digits (28x28 pixel images)

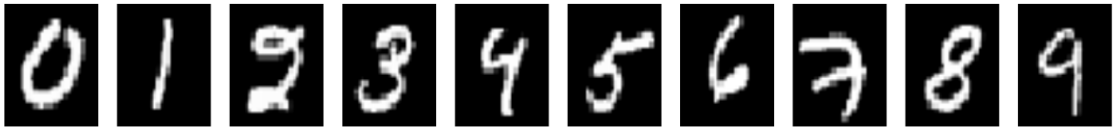
Dataset of thousands of images of the digit 8



$$\boldsymbol{X}_{N\times p} = \begin{pmatrix} x_1^{(1)} & \dots & x_p^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \dots & x_p^{(N)} \end{pmatrix} \qquad \mathbf{x}^{(i)} \in \mathbb{R}^{784}$$

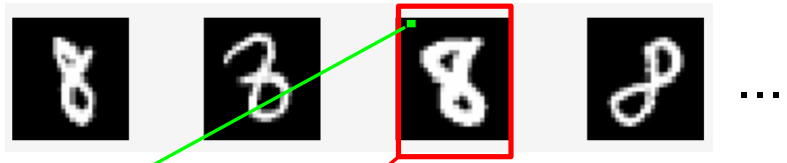
Figures from M.P. Deisenroth, A.A. Faisal, and C. Soon Ong, Mathematics for Machine Learning 2020

MNIST digits



MNIST digits: handwritten 0-9 digits (28x28 pixel images)

Dataset of thousands of images of the digit 8



$$\boldsymbol{X}_{N\times p} = \begin{pmatrix} x_1^{(1)} & \dots & x_p^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \dots & x_p^{(N)} \end{pmatrix} \qquad \mathbf{x}^{(i)} \in \mathbb{R}^{784}$$

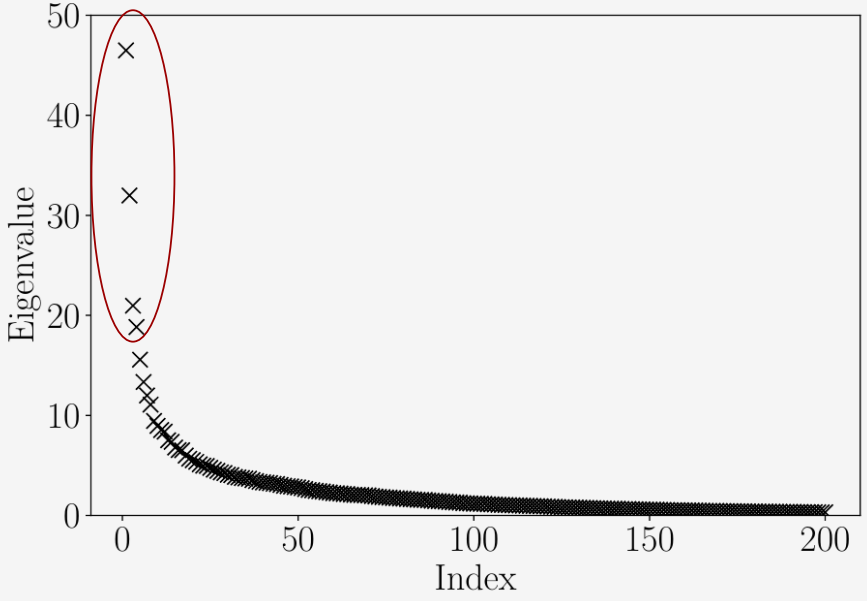
There are correlations between pixels across data because of a common pattern (the shape of 8) $\longrightarrow$ **Data are intrinsically lower-dimensional**

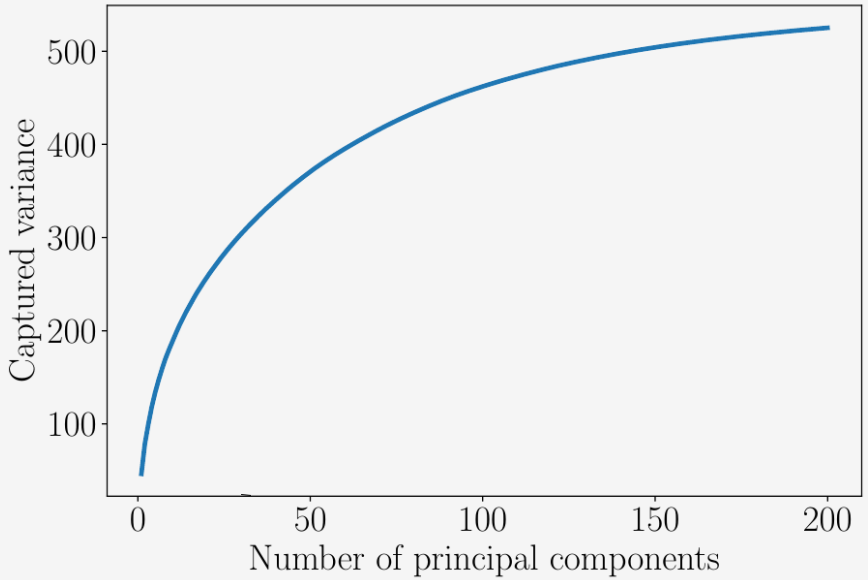
Figures from M.P. Deisenroth, A.A. Faisal, and C. Soon Ong, Mathematics for Machine Learning 2020

MNIST digits

A few larger eigenvalues, signalling strong correlations

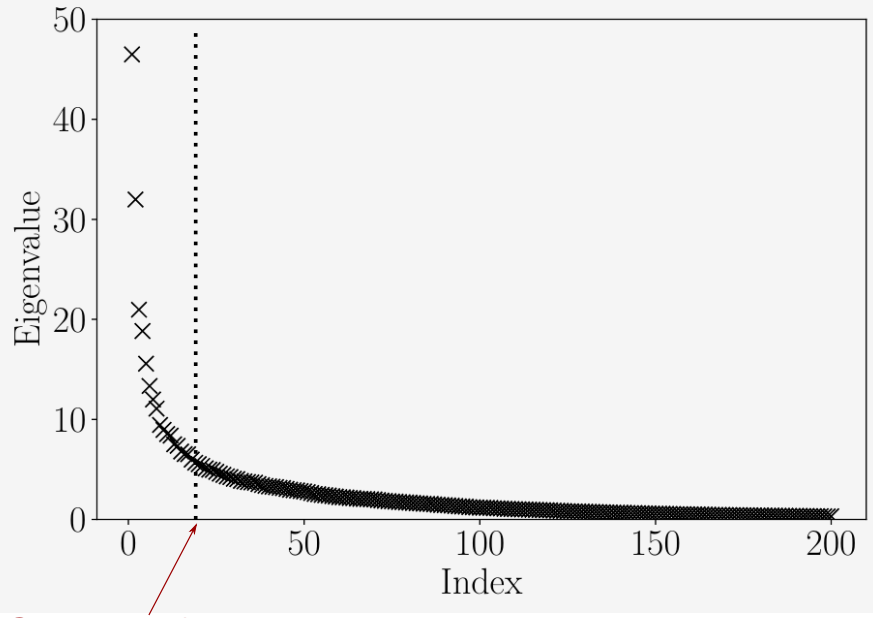
Captured variance $V_m = \sum_{j=1}^{m} \lambda_j$



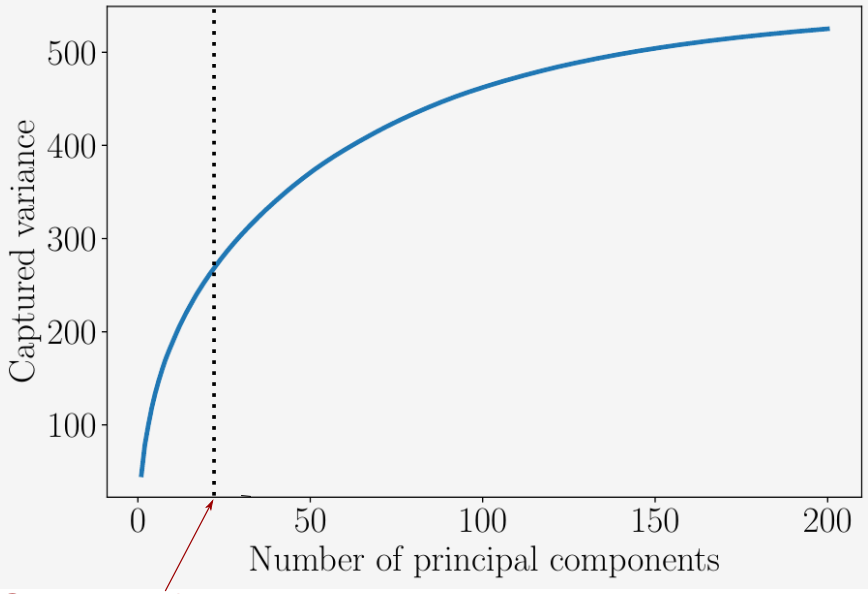


MNIST digits

Captured variance $V_m = \sum_{j=1}^{m} \lambda_j$



Choice of m



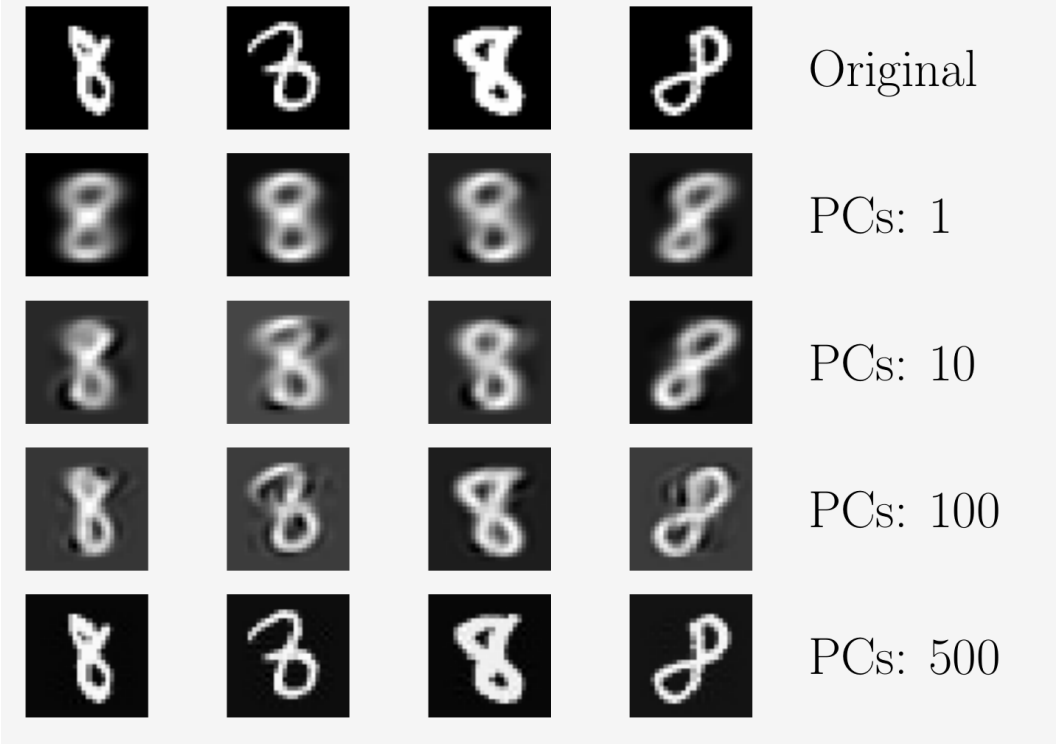
Choice of m

One looks at the eigenvalue spectrum and the captured variance to choose m

MNIST digits

Task 1. Approximating $\mathbf{x}^{(i)}$ by a few principal components (PCs)

$$\widehat{\mathbf{x}}^{(i)} = \sum_{j=1}^{m} \underbrace{\left(\mathbf{x}^{(i)} \cdot \mathbf{v}_j\right)}_{a_j^{(i)}} \mathbf{v}_j \quad \text{where} \quad C_{\mathbf{x}}\mathbf{v}_j = \lambda_j \mathbf{v}_j$$

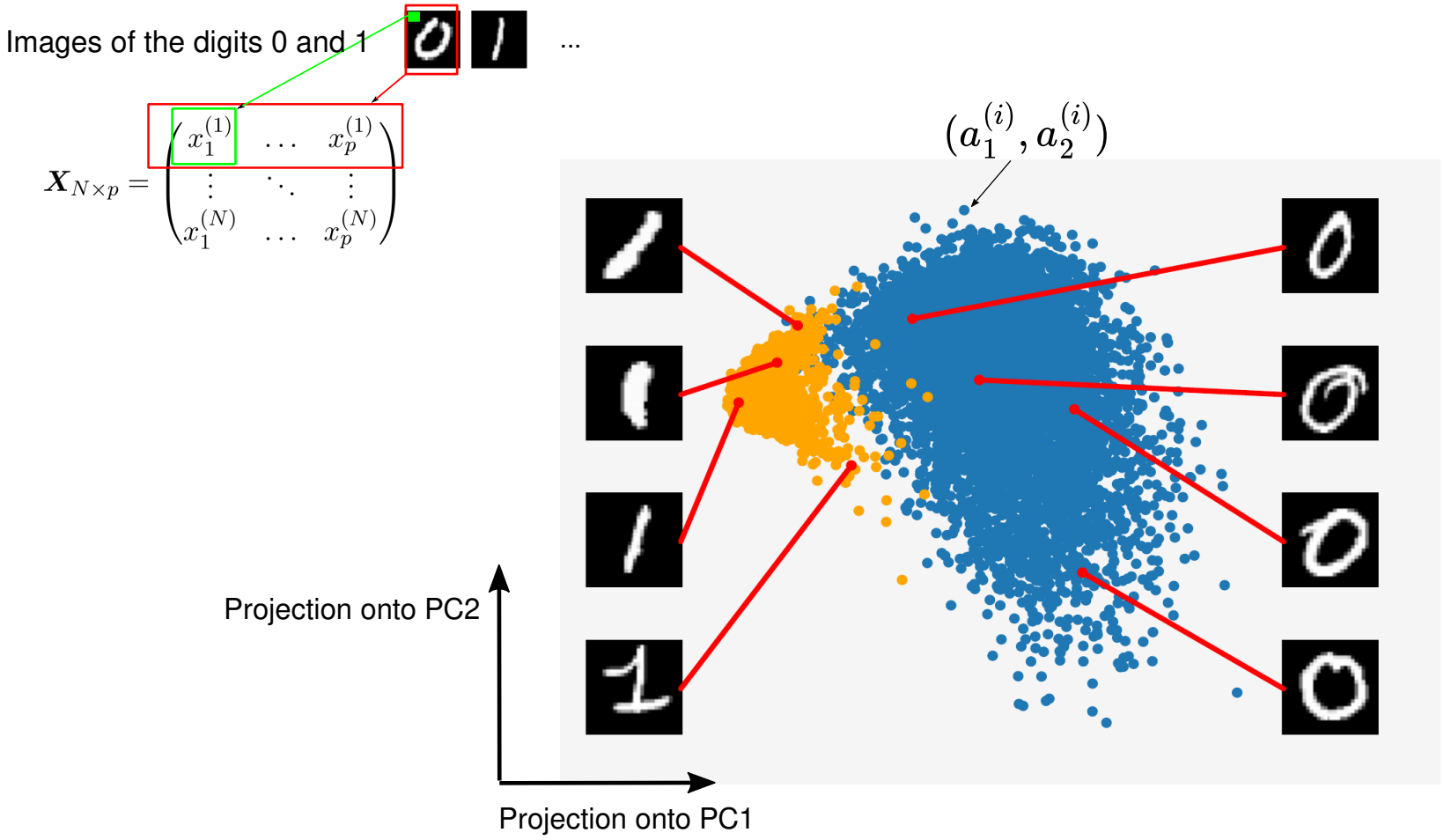


The more PCs, the more accurate the approximation

Figures from M.P. Deisenroth, A.A. Faisal, and C. Soon Ong, Mathematics for Machine Learning 2020

MNIST digits

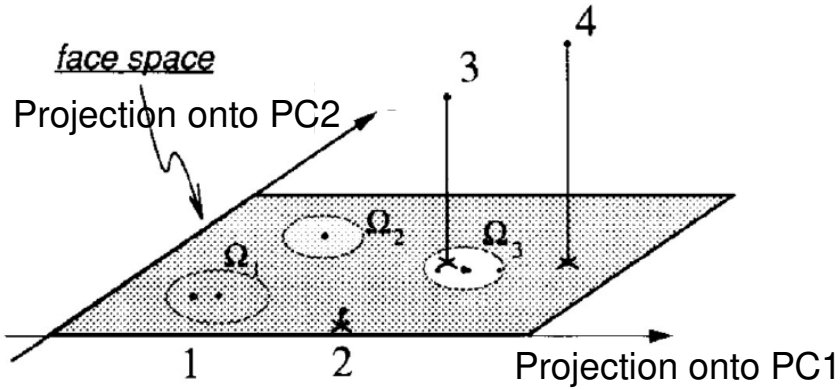
Task 2. Discovering properties and relationships in $\mathbf{x}^{(i)}$



In the lower-dim. representation, similar data group together

Figures from M.P. Deisenroth, A.A. Faisal, and C. Soon Ong, Mathematics for Machine Learning 2020

Interpretation of PCs: image data



The data are face images. As mentioned before, structurally similar images cluster in the lower-dimensional space of PCA.

Turk and Pentland, Face Recognition using eigenfaces 1991

Interpretation of PCs: image data

Low-dim space: 7 eigenfaces, look as modified versions of images ('prototypes')

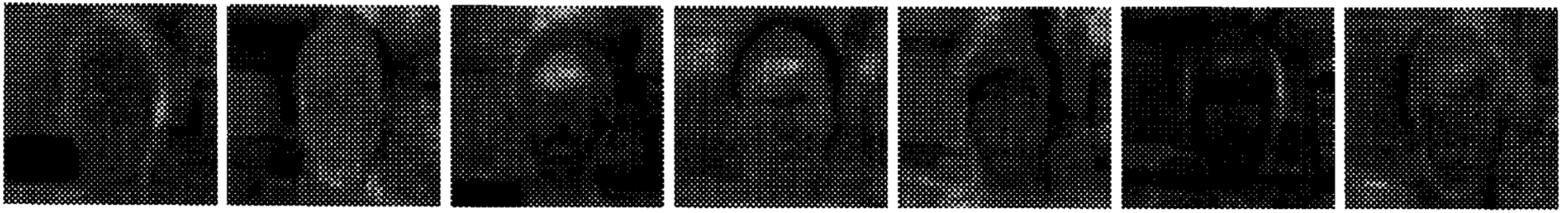


Figure 2: Seven of the eigenfaces calculated from the images of Figure 1, without the background removed.

Interpretation of PCs: image data

Low-dim space: 7 eigenfaces, look as modified versions of images (‘prototypes’)

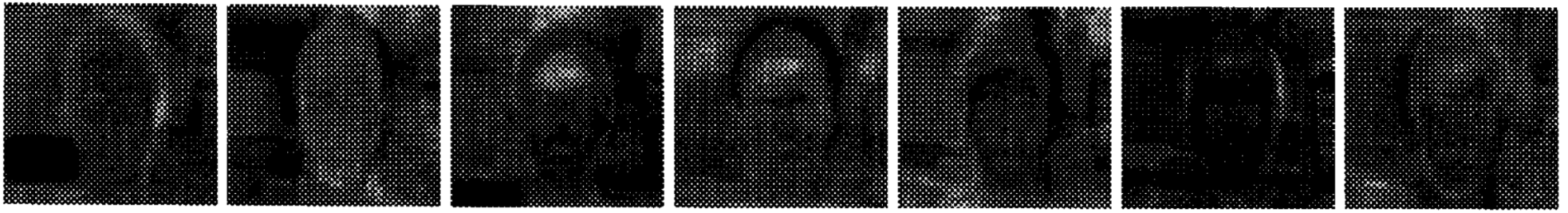


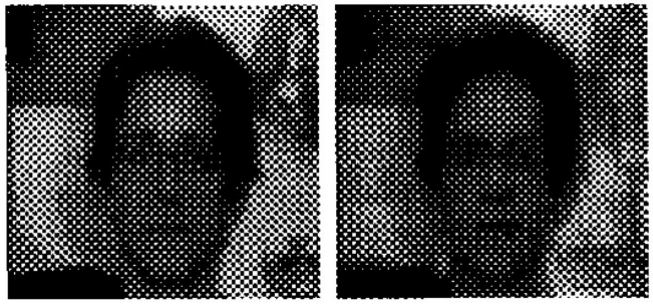
Figure 2: Seven of the eigenfaces calculated from the images of Figure 1, without the background removed.

Image

$$\mathbf{x}^{(i)}$$

Its projection

$$\widehat{\mathbf{x}}_m^{(i)} = \sum_{j=1}^{m} a_j^{(i)} \mathbf{v}_j$$



Can be used for image recognition tasks:
1. Image classifier
(minimal distance to one ‘prototype’)

Interpretation of PCs: image data

Low-dim space: 7 eigenfaces, look as modified versions of images (‘prototypes’)

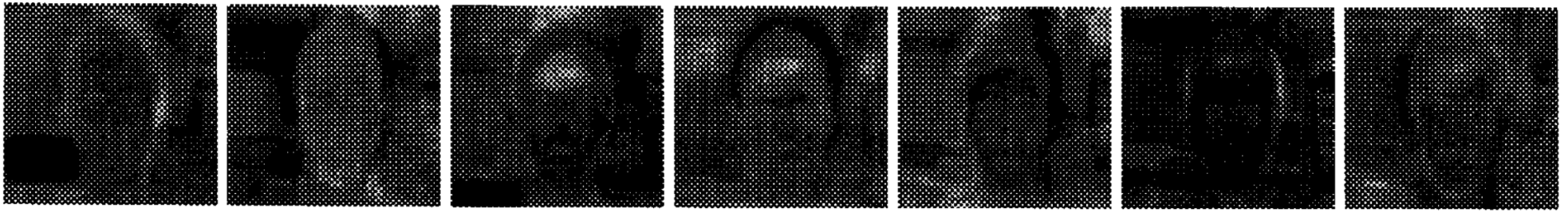


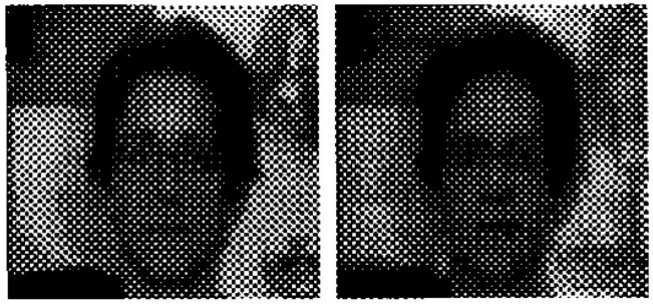
Figure 2: Seven of the eigenfaces calculated from the images of Figure 1, without the background removed.

Image

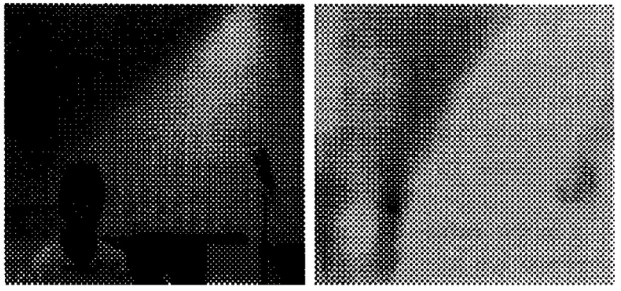
$$\mathbf{x}^{(i)}$$

Its projection

$$\widehat{\mathbf{x}}_m^{(i)} = \sum_{j=1}^{m} a_j^{(i)} \mathbf{v}_j$$



Can be used for image recognition tasks:
1. Image classifier
(minimal distance to one ‘prototype’)
2. **Face detection in images**
(difference between image and projection)



(a) (b)

Figure 4: (a) Original image. (b) Face map, where low values (dark areas) indicate the presence of a face.

Interpretation of PCs: image data

Low-dim space: 7 eigenfaces, look as modified versions of images (‘prototypes’)

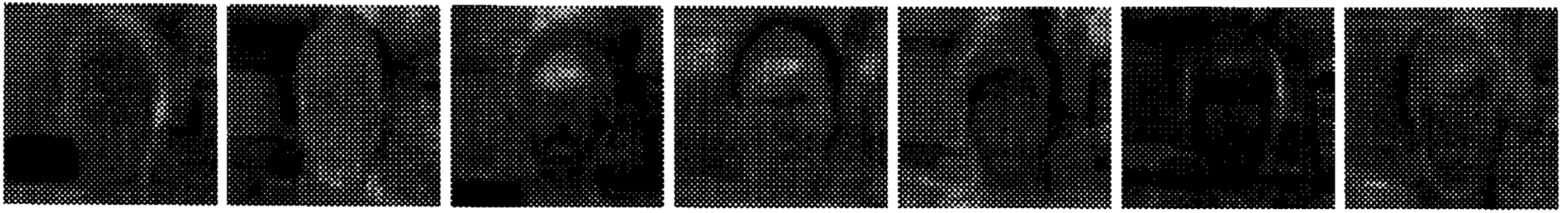


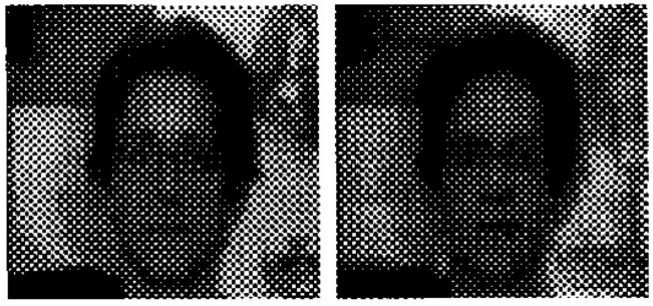
Figure 2: Seven of the eigenfaces calculated from the images of Figure 1, without the background removed.

Image

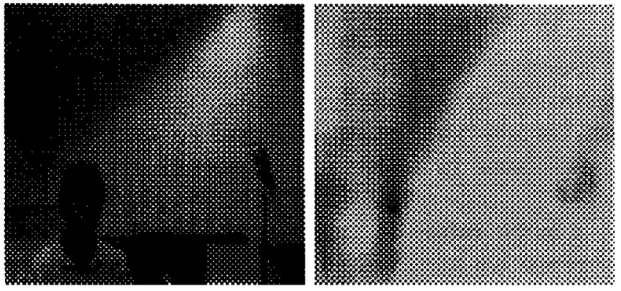
$$\mathbf{x}^{(i)}$$

Its projection

$$\widehat{\mathbf{x}}_m^{(i)} = \sum_{j=1}^{m} a_j^{(i)} \mathbf{v}_j$$



Can be used for image recognition tasks:
1. Image classifier
(minimal distance to one ‘prototype’)
2. **Face detection in images**
(difference between image and projection)



(a) (b)

Figure 4: (a) Original image. (b) Face map, where low values (dark areas) indicate the presence of a face.

Reduction of dimensions + computationally cheap still capture main information and structure

Beyond standard PCA

Suppose we project data onto PC1

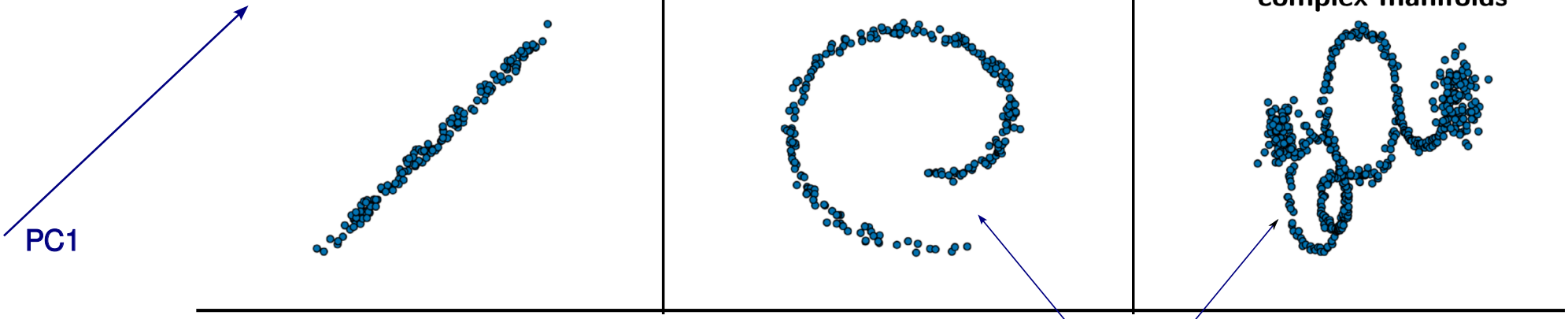


Figure from Glielmo et al. 2021

We are unable to detect these types of structure!

Beyond standard PCA

Suppose we project data onto PC1

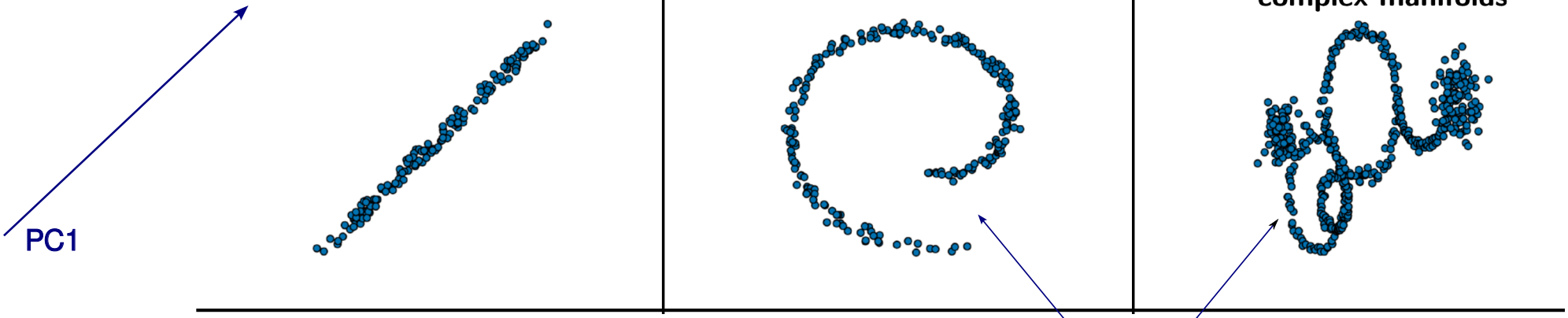


Figure from Glielmo et al. 2021

We are unable to detect these types of structure!

Difficult example

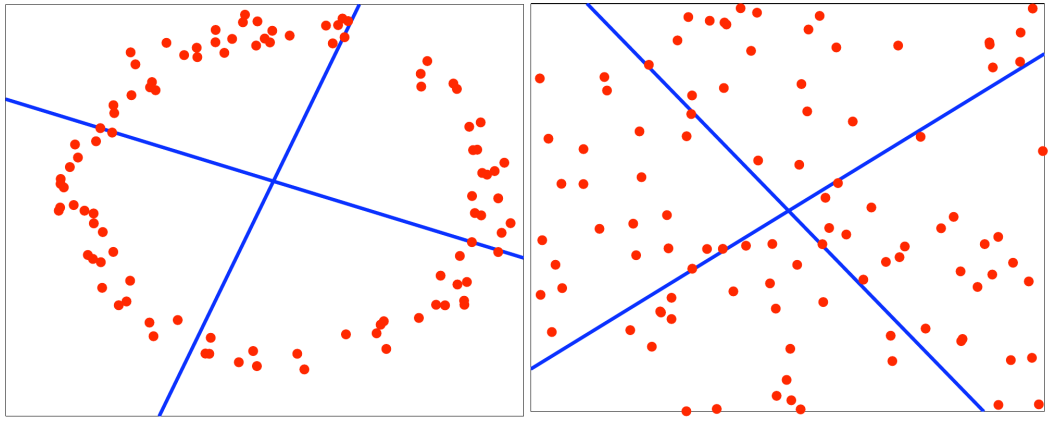


Figure from https://www.cs.mcgill.ca/~dprecup/courses/ML/lectures.html

Linear PCA does not make differences here!

How do we find non-linear, low-dimensional representations of the data?

Quick Aside: connection to SVD

Singular Value Decomposition (SVD) is a standard matrix factorisation method:

$$\mathbf{A}_{(N\times p)} = \mathbf{U}_{(N\times N)}\boldsymbol{\Sigma}_{(N\times p)}\mathbf{V}^T_{(p\times p)}$$

Left eigenvectors:
$\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_N)$

Right eigenvectors:
$\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_p)$

Diagonal matrix with the singular values $\sigma_1, \ldots, \sigma_R$:

$$\left\{ \begin{array}{l} (\mathbf{A}^T\mathbf{A})\mathbf{v}_k = \sigma_k^2\mathbf{v}_k \\ (\mathbf{A}\mathbf{A}^T)\mathbf{u}_k = \sigma_k^2\mathbf{u}_k \end{array} \right\}$$

i.e.: they are related to the left and right eigenvalues (same)

Quick Aside: connection to SVD

Singular Value Decomposition (SVD) is a standard matrix factorisation method:

$$\mathbf{A}_{(N\times p)} = \mathbf{U}_{(N\times N)}\boldsymbol{\Sigma}_{(N\times p)}\mathbf{V}^T_{(p\times p)}$$

Left eigenvectors:
$\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_N)$

Right eigenvectors:
$\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_p)$

Diagonal matrix with the singular values $\sigma_1, \ldots, \sigma_R$:

$$\left\{ \begin{array}{l} (\mathbf{A}^T\mathbf{A})\mathbf{v}_k = \sigma_k^2\mathbf{v}_k \\ (\mathbf{A}\mathbf{A}^T)\mathbf{u}_k = \sigma_k^2\mathbf{u}_k \end{array} \right\}$$

i.e.: they are related to the left and right eigenvalues (same)

A lower rank approximation is given by:

$$\widehat{\mathbf{A}}_m = \sum_{j=1}^{m} \sigma_j\, \mathbf{u}_j\mathbf{v}_j^T =: \mathbf{U}_m\boldsymbol{\Sigma}_m\mathbf{V}_m^T$$

Quick Aside: connection to SVD

Let's take $\mathbf{X}$ as the matrix $\mathbf{A}$ - we first define the **centred** data matrix:

$$\mathcal{X}_{N\times p} := \mathbf{X} - \frac{1}{N}\mathbf{1}\mathbf{1}^T\mathbf{X} = \underbrace{\left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right)}_{centring\ matrix}\mathbf{X}$$

From which the covariance can be derived:

$$(\mathcal{X}^T\mathcal{X})_{p\times p} = N\,C_{\mathbf{x}}$$

The PCA approximation is then written in matrix form as:

$$\widehat{\mathcal{X}}_{N\times p} = \mathbf{U}_m\mathbf{\Sigma}_m\mathbf{V}_m^T =: \mathbf{X}_{\mathrm{PCA}}\mathbf{V}_m^T$$

The PCA representation:

$$\mathbf{X}_{\mathrm{PCA}} = \mathbf{U}_m\mathbf{\Sigma}_m$$

$a_j^{(i)}$ with $j = 1, \ldots, m$ and $i = 1, \ldots, N$

Nonlinear extension: Kernel PCA

It relies on the introduction of kernels, and the kernel trick. We have dot products:

$$\left(\mathcal{X}\mathcal{X}^T\right)_{ij} = \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \quad \text{Assuming: } \mathbf{x}^{(i)} \leftarrow (\mathbf{x}^{(i)} - \langle \mathbf{x} \rangle)$$

Quick remark:

$$(\mathcal{X}^T\mathcal{X})_{p\times p}$$

$$(\mathcal{X}\mathcal{X}^T)_{N\times N}$$

have the same eigenvalues

These are the dot products that are replaced by kernel functions $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$:

$$K_{ij} = \boldsymbol{\phi}(\mathbf{x}^{(i)}) \cdot \boldsymbol{\phi}(\mathbf{x}^{(j)}) \equiv k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

Kernel PCA relies on the spectral decomposition of:

$$(C_{\boldsymbol{\phi}})_{D\times D} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\phi}(\mathbf{x}^{(i)})\boldsymbol{\phi}(\mathbf{x}^{(i)})^T$$

Dimensionality of the non-linearly transformed data through $\boldsymbol{\phi}$

Nonlinear extension: Kernel PCA

Its diagonalisation would lead to:

$$C_{\boldsymbol{\phi}}\mathbf{v}_j = \lambda_j \mathbf{v}_j \qquad j = 1, .., D$$

These eigenvectors being the **non-linear principal components**:

$$\mathbf{v}_j = \sum_{i=1}^{N} a_j^{(i)} \boldsymbol{\phi}(\mathbf{x}^{(i)})$$

Nonlinear extension: Kernel PCA

Its diagonalisation would lead to:

$$C_{\boldsymbol{\phi}}\mathbf{v}_j = \lambda_j \mathbf{v}_j \qquad j = 1, .., D$$

These eigenvectors being the **non-linear principal components**:

$$\mathbf{v}_j = \sum_{i=1}^{N} a_j^{(i)} \boldsymbol{\phi}(\mathbf{x}^{(i)})$$

Kernel trick: we don't need to work explicitly with $\boldsymbol{\phi}$ but only with the kernels.
Indeed, the projection of non-linearly transformed data onto the *j* non-linear PC:

$$\boldsymbol{\phi}(\mathbf{x}) \cdot \mathbf{v}_j = \sum_{i=1}^{N} a_j^{(i)} \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{x}^{(i)}) = \sum_{i=1}^{N} a_j^{(i)} k(\mathbf{x}, \mathbf{x}^{(i)})$$

$$\mathbf{K}\mathbf{a}_j = N\lambda_j \mathbf{a}_j \quad \text{where: } \mathbf{a}_j = \{a_j^{(i)}\}_{i=1}^{N}$$

simply the eigenvectors of the Gram matrix

Nonlinear extension: Kernel PCA

Remarks: Kernel PCA in practice.

In practice ϕ won't be not centred, one needs to replace $\mathbf{K} \to \tilde{\mathbf{K}}$ with:

$$\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N\mathbf{K} - \mathbf{K}\mathbf{1}_N + \mathbf{1}_N\mathbf{K}\mathbf{1}_N \qquad \tilde{K}_{ij} = \tilde{\boldsymbol{\phi}}(\mathbf{x}^{(i)}) \cdot \tilde{\boldsymbol{\phi}}(\mathbf{x}^{(j)})$$

centred Gram matrix

centred ϕ

Nonlinear extension: Kernel PCA

Remarks: Kernel PCA in practice.

In practice ϕ won't be not centred, one needs to replace $\mathbf{K} \rightarrow \tilde{\mathbf{K}}$ with:

$$\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N\mathbf{K} - \mathbf{K}\mathbf{1}_N + \mathbf{1}_N\mathbf{K}\mathbf{1}_N \qquad \tilde{K}_{ij} = \tilde{\boldsymbol{\phi}}(\mathbf{x}^{(i)}) \cdot \tilde{\boldsymbol{\phi}}(\mathbf{x}^{(j)})$$

centred Gram matrix

centred ϕ

The eigenvectors of the Gram matrix need to be normalised, using:

$$1 = \mathbf{v}_j \cdot \mathbf{v}_j = \mathbf{a}_j^T \mathbf{K} \mathbf{a}_j = N\lambda_j \mathbf{a}_j \cdot \mathbf{a}_j$$

To ensure the normalisation of the non-linear PCs $\mathbf{v}_j$

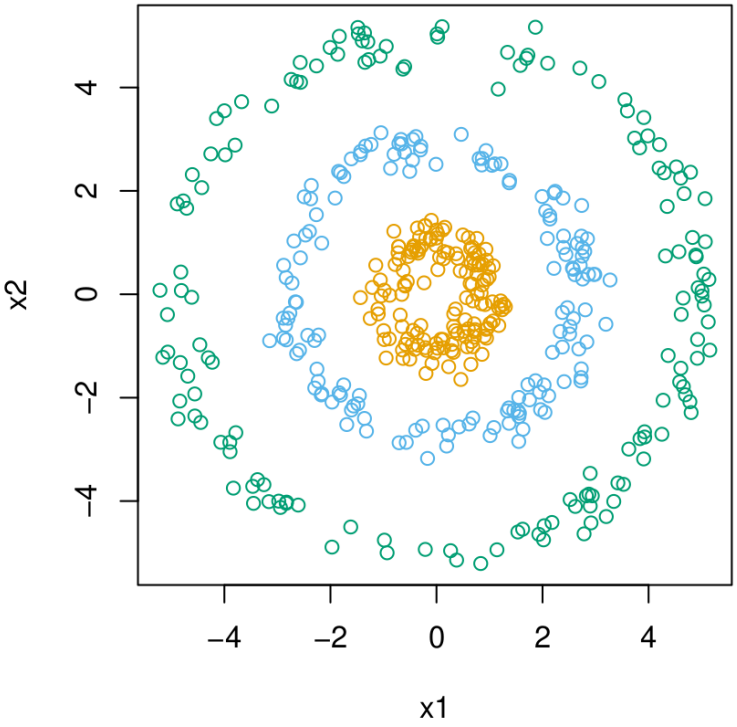
which becomes, for the centred Gram matrix:

$$1 = \mathbf{v}_j \cdot \mathbf{v}_j = \mathbf{a}_j^T \tilde{\mathbf{K}} \mathbf{a}_j = \lambda_j \mathbf{a}_j \cdot \mathbf{a}_j$$

since the centred Gram matrix contains already a scaling by N.

Nonlinear extension: Kernel PCA

Example:

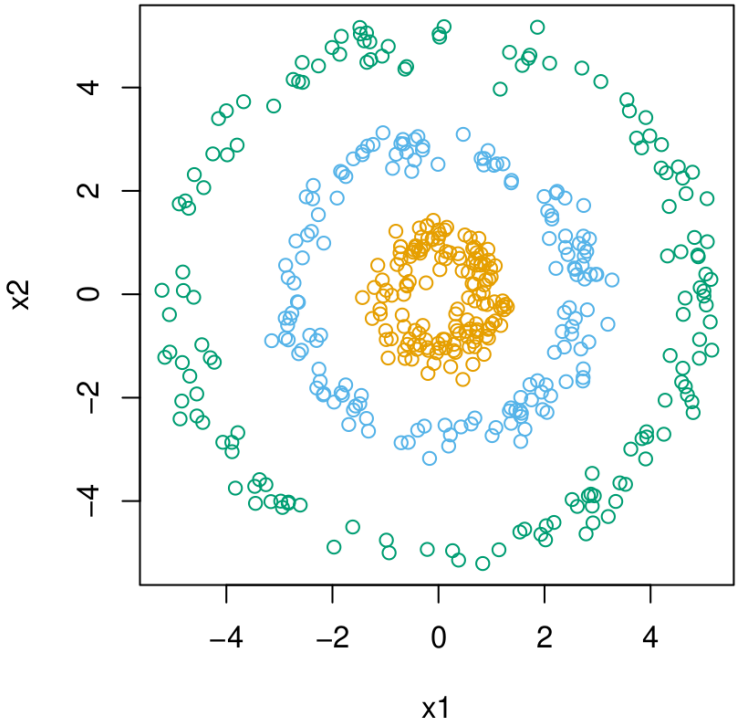


Data distributed on a non-linear manifold:
PCA does not capture the structure, and
the partition into 3 classes (colours)

Figures from Hastie, Tibshirani, Friedman, The Elements of statistical learning, Chap. 14

Nonlinear extension: Kernel PCA

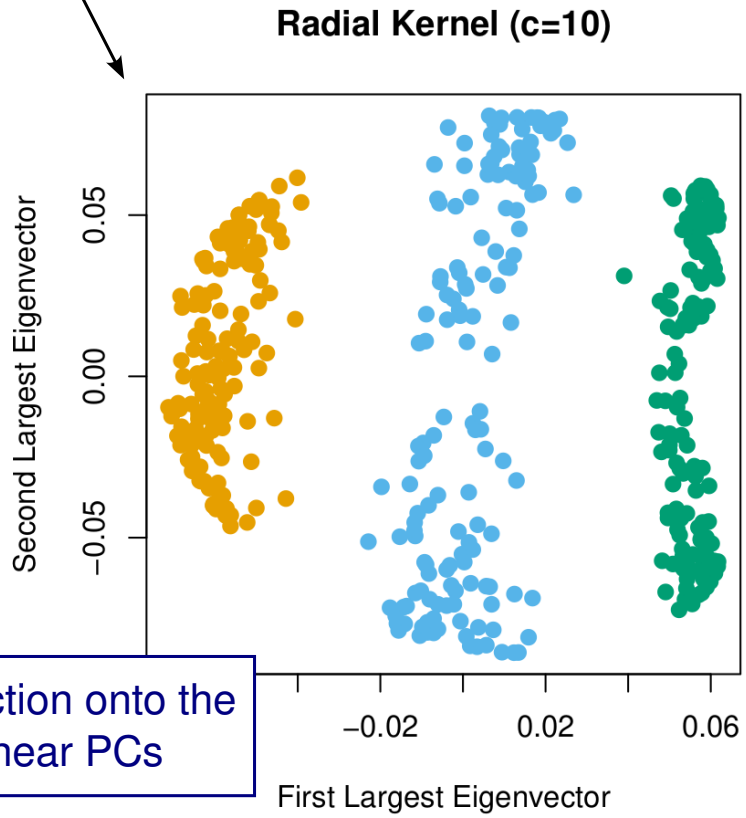
Example:



Data distributed on a non-linear manifold: PCA does not capture the structure, and the partition into 3 classes (colours)

Kernel PCA with radial kernel:

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = e^{-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2}{c}}$$



Projection onto the non-linear PCs

Figures from Hastie, Tibshirani, Friedman, The Elements of statistical learning, Chap. 14

Extension 2: Sparse PCA

One can enforce sparsity in PCA to increase interpretability, specifically in:

The non-sparse basis vectors $\mathbf{V}_m = \left(\mathbf{v}_1 \dots \mathbf{v}_m\right)_{p \times m}$

by introducing a LASSO-type penalty term in the function optimise in PCA.

Extension 2: Sparse PCA

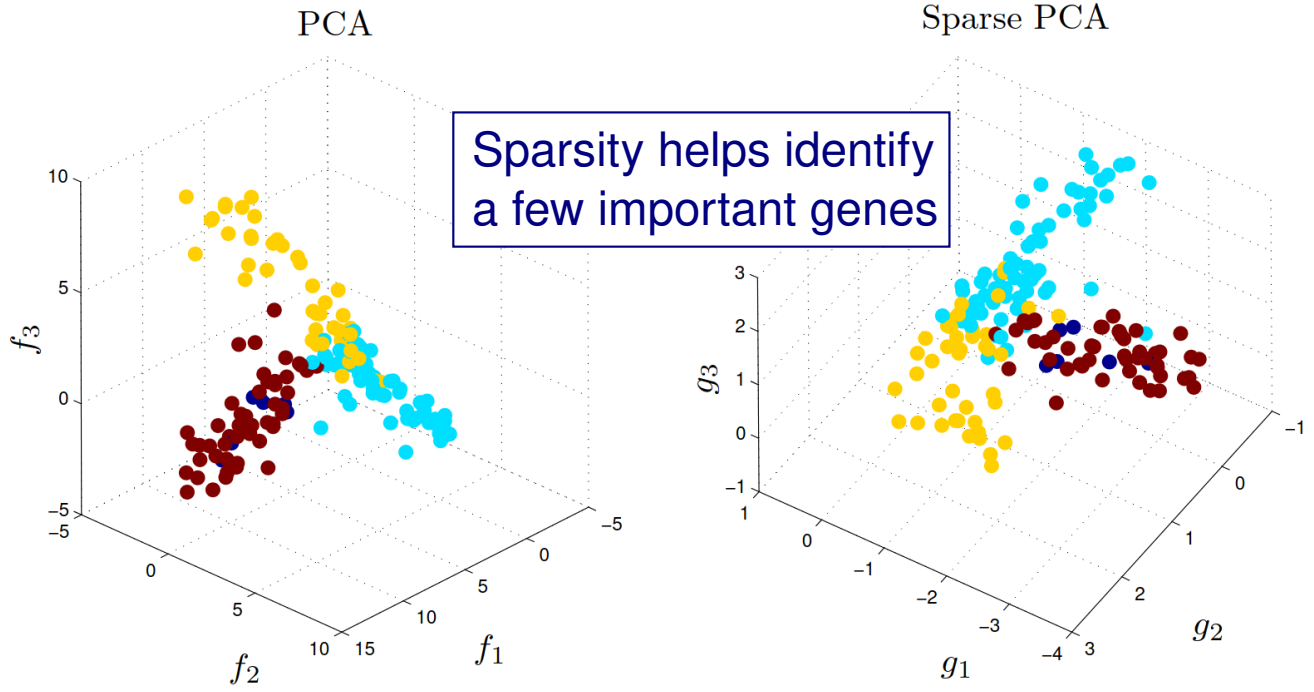
One can enforce sparsity in PCA to increase interpretability, specifically in:

The non-sparse basis vectors $\mathbf{V}_m = \left(\mathbf{v}_1 \dots \mathbf{v}_m\right)_{p \times m}$

by introducing a LASSO-type penalty term in the function optimise in PCA.

The result are sparser PCs, with fewer non-zero terms

$$\mathbf{v}_j^{\text{LASSO}} = \begin{pmatrix} \blacksquare \\ 0 \\ \vdots \\ \blacksquare \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$



Example from D'Aspremont et al. 2006

FIG. 6.3. *Clustering of the gene expression data in the PCA versus sparse PCA basis with 500 genes. The factors f on the left are dense and each use all 500 genes while the sparse factors g_1, g_2 and g_3 on the right involve 6, 4 and 4 genes respectively. (Data: Iconix Pharmaceuticals)*

Non-Negative Matrix Factorisation (NMF)

Connection to SVD: the PCA approximation is given by a matrix factorisation:

$$\widehat{\mathcal{X}}_{N\times p} = \mathbf{U}_m \mathbf{\Sigma}_m \mathbf{V}_m^T =: \mathbf{X}_{\text{PCA}} \mathbf{V}_m^T$$

Key Message: In NMF, the data matrix $\mathbf{X}$ is approximated by a lower-dimensional representation given by:

$$\boxed{\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H},} \quad \text{such that} \quad \boxed{W_{ij} \geq 0, H_{ij} \geq 0 \quad \text{for all} \quad i, j}$$

i.e., where we require a factorisation into matrices with non-negative elements.

Non-Negative Matrix Factorisation (NMF)

Connection to SVD: the PCA approximation is given by a matrix factorisation:

$$\widehat{\mathcal{X}}_{N\times p} = \mathbf{U}_m \mathbf{\Sigma}_m \mathbf{V}_m^T =: \mathbf{X}_{\mathrm{PCA}} \mathbf{V}_m^T$$

Key Message: In NMF, the data matrix $\mathbf{X}$ is approximated by a lower-dimensional representation given by:

$$\boxed{\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H},} \quad \text{such that} \quad \boxed{W_{ij} \geq 0, H_{ij} \geq 0 \quad \text{for all} \quad i,j}$$

i.e., where we require a factorisation into matrices with non-negative elements.

$$\mathbf{X}_{N\times p} = \begin{pmatrix} X_{11} & \dots & X_{1p} \\ \vdots & \ddots & \vdots \\ X_{N1} & \dots & X_{Np} \end{pmatrix} \approx \underbrace{\begin{pmatrix} W_{11} & \dots & W_{1r} \\ \vdots & \ddots & \vdots \\ W_{N1} & \dots & W_{Nr} \end{pmatrix}}_{\boldsymbol{W}_{N\times r}} \underbrace{\begin{pmatrix} H_{11} & \dots & H_{1p} \\ \vdots & \ddots & \vdots \\ H_{r1} & \dots & H_{rp} \end{pmatrix}}_{\boldsymbol{H}_{r\times p}}$$

Non-Negative Matrix Factorisation (NMF)

Connection to SVD: the PCA approximation is given by a matrix factorisation:

$$\widehat{\mathcal{X}}_{N\times p} = \mathbf{U}_m \mathbf{\Sigma}_m \mathbf{V}_m^T =: \mathbf{X}_{\mathrm{PCA}} \mathbf{V}_m^T$$

Key Message: In NMF, the data matrix $\mathbf{X}$ is approximated by a lower-dimensional representation given by:

$$\boxed{\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H},} \quad \text{such that} \quad \boxed{W_{ij} \geq 0, H_{ij} \geq 0 \quad \text{for all} \quad i,j}$$

i.e., where we require a factorisation into matrices with non-negative elements.

$$\mathbf{X}_{N\times p} = \begin{pmatrix} X_{11} & \dots & X_{1p} \\ \vdots & \ddots & \vdots \\ X_{N1} & \dots & X_{Np} \end{pmatrix} \approx \underbrace{\begin{pmatrix} W_{11} & \dots & W_{1r} \\ \vdots & \ddots & \vdots \\ W_{N1} & \dots & W_{Nr} \end{pmatrix}}_{\boldsymbol{W}_{N\times r}} \underbrace{\begin{pmatrix} H_{11} & \dots & H_{1p} \\ \vdots & \ddots & \vdots \\ H_{r1} & \dots & H_{rp} \end{pmatrix}}_{\boldsymbol{H}_{r\times p}}$$

$r \ll p$

Dimensions of the NMF representation

non-negative coefficients expressing data as a linear combination of the basis vectors

non-negative equivalent of principal components

Non-Negative Matrix Factorisation (NMF)

How do we learn the optimal factorisation? Via different *iterative* algorithms, to **minimise an objective function quantifying the quality of the approximation**.

Non-Negative Matrix Factorisation (NMF)

How do we learn the optimal factorisation? Via different *iterative* algorithms, to **minimise an objective function quantifying the quality of the approximation**.

Algorithm 1. $$||\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}||^2 = \sum_{ij}(X_{ij} - (\boldsymbol{W}\boldsymbol{H})_{ij})^2$$ **Objective function**

$$W_{ij} \leftarrow W_{ij} \frac{(\mathbf{X}\mathbf{H}^T)_{ij}}{(\mathbf{W}\mathbf{H}\mathbf{H}^T)_{ij}}$$

$$H_{jk} \leftarrow H_{jk} \frac{(\mathbf{W}^T\mathbf{X})_{jk}}{(\mathbf{W}^T\mathbf{W}\mathbf{H})_{jk}}$$

Multiplicative rules leading iteratively to a local minimum

See Lee and Seung, NIPS 2000

Non-Negative Matrix Factorisation (NMF)

How do we learn the optimal factorisation? Via different *iterative* algorithms, to **minimise an objective function quantifying the quality of the approximation**.

Algorithm 1. $$||\boldsymbol{X} - \boldsymbol{WH}||^2 = \sum_{ij}(X_{ij} - (\boldsymbol{WH})_{ij})^2$$ **Objective function**

$$W_{ij} \leftarrow W_{ij}\frac{(\mathbf{XH}^T)_{ij}}{(\mathbf{WHH}^T)_{ij}}$$

$$H_{jk} \leftarrow H_{jk}\frac{(\mathbf{W}^T\mathbf{X})_{jk}}{(\mathbf{W}^T\mathbf{WH})_{jk}}$$

Multiplicative rules leading iteratively to a local minimum

Algorithm 2. $$D(\boldsymbol{X}||\boldsymbol{WH}) = \sum_{ij}\left[X_{ij}\log\left(\frac{X_{ij}}{(\boldsymbol{WH})_{ij}}\right) - X_{ij} + (\boldsymbol{WH})_{ij}\right]$$ **Objective function**

$$W_{ij} \leftarrow W_{ij}\frac{\sum_k H_{jk}X_{ik}/(\boldsymbol{WH})_{ik}}{\sum_k H_{jk}}$$

$$H_{jk} \leftarrow H_{jk}\frac{\sum_i W_{ij}X_{ik}/(\boldsymbol{WH})_{ik}}{\sum_i W_{ij}}$$

Multiplicative rules

See Lee and Seung, NIPS 2000

PCA vs NMF on image data

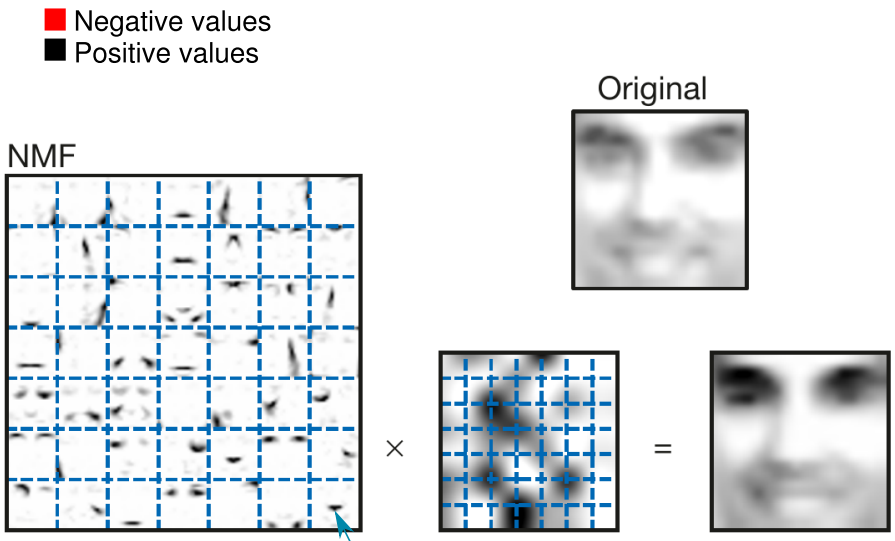


Figure 1 Non-negative matrix factorization (NMF) learns a parts-based representation of faces, whereas vector quantization (VQ) and principal components analysis (PCA) learn holistic representations. The three learning methods were applied to a database of $m = 2{,}429$ facial images, each consisting of $n = 19 \times 19$ pixels, and constituting an $n \times m$ matrix V. All three find approximate factorizations of the form $V \approx WH$, but with three different types of constraints on W and H, as described more fully in the main text and methods. As shown in the 7×7 montages, each method has learned a set of $r = 49$ basis images. Positive values are illustrated with black pixels and negative values with red pixels. A particular instance of a face, shown at top right, is approximately represented by a linear superposition of basis images. The coefficients of the linear superposition are shown next to each montage, in a 7×7 grid, and the resulting superpositions are shown on the other side of the equality sign. Unlike VQ and PCA, NMF learns to represent faces with a set of basis images resembling parts of faces.

$$\begin{pmatrix} W_{11} & \cdots & W_{1r} \\ \vdots & \ddots & \vdots \\ W_{N1} & \cdots & W_{Nr} \end{pmatrix} \begin{pmatrix} H_{11} & \cdots & H_{1p} \\ \vdots & \ddots & \vdots \\ H_{r1} & \cdots & H_{rp} \end{pmatrix} = \begin{pmatrix} X_{11} & \cdots & X_{1p} \\ \vdots & \ddots & \vdots \\ X_{N1} & \cdots & X_{Np} \end{pmatrix} \quad N = 2429$$

Each row is a basis image

$r = 49$ $\qquad$ $p = 361$

See D.D. Lee and H.S. Seung, Learning the parts of objects by non-negative matrix factorization, Nature 1999

PCA vs NMF on image data

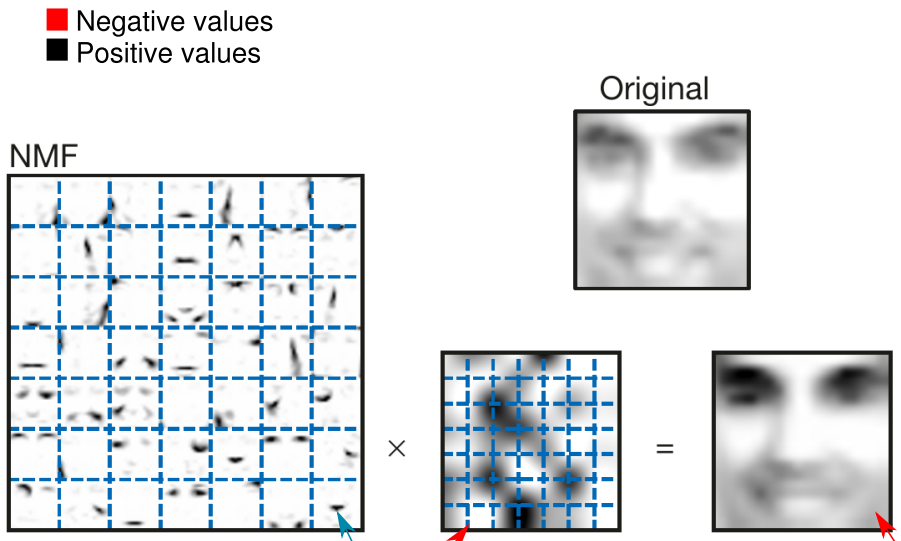


Figure 1 Non-negative matrix factorization (NMF) learns a parts-based representation of faces, whereas vector quantization (VQ) and principal components analysis (PCA) learn holistic representations. The three learning methods were applied to a database of $m =$ 2,429 facial images, each consisting of $n = 19 \times 19$ pixels, and constituting an $n \times m$ matrix V. All three find approximate factorizations of the form $V \approx WH$, but with three different types of constraints on W and H, as described more fully in the main text and methods. As shown in the 7×7 montages, each method has learned a set of $r = 49$ basis images. Positive values are illustrated with black pixels and negative values with red pixels. A particular instance of a face, shown at top right, is approximately represented by a linear superposition of basis images. The coefficients of the linear superposition are shown next to each montage, in a 7×7 grid, and the resulting superpositions are shown on the other side of the equality sign. Unlike VQ and PCA, NMF learns to represent faces with a set of basis images resembling parts of faces.

Coefficients

Approximated images

$$\begin{pmatrix} W_{11} & \dots & W_{1r} \\ \vdots & \ddots & \vdots \\ W_{N1} & \dots & W_{Nr} \end{pmatrix} \begin{pmatrix} H_{11} & \dots & H_{1p} \\ \vdots & \ddots & \vdots \\ H_{r1} & \dots & H_{rp} \end{pmatrix} = \begin{pmatrix} X_{11} & \dots & X_{1p} \\ \vdots & \ddots & \vdots \\ X_{N1} & \dots & X_{Np} \end{pmatrix} \quad N = 2429$$

Each row is a basis image

$r = 49$ $\qquad$ $p = 361$

See D.D. Lee and H.S. Seung, Learning the parts of objects by non-negative matrix factorization, Nature 1999

PCA vs NMF on image data

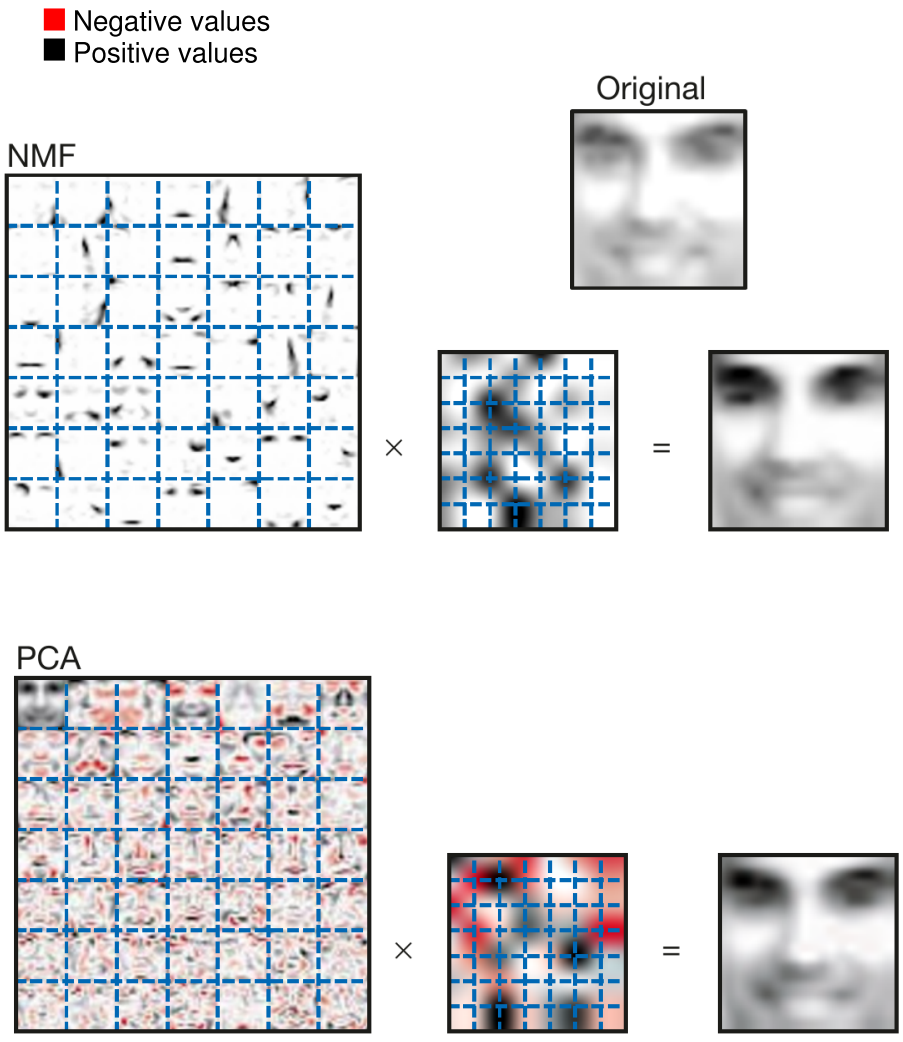


Figure 1 Non-negative matrix factorization (NMF) learns a parts-based representation of faces, whereas vector quantization (VQ) and principal components analysis (PCA) learn holistic representations. The three learning methods were applied to a database of $m = 2{,}429$ facial images, each consisting of $n = 19 \times 19$ pixels, and constituting an $n \times m$ matrix V. All three find approximate factorizations of the form $V \approx WH$, but with three different types of constraints on W and H, as described more fully in the main text and methods. As shown in the 7×7 montages, each method has learned a set of $r = 49$ basis images. Positive values are illustrated with black pixels and negative values with red pixels. A particular instance of a face, shown at top right, is approximately represented by a linear superposition of basis images. The coefficients of the linear superposition are shown next to each montage, in a 7×7 grid, and the resulting superpositions are shown on the other side of the equality sign. Unlike VQ and PCA, NMF learns to represent faces with a set of basis images resembling parts of faces.

See D.D. Lee and H.S. Seung, Learning the parts of objects by non-negative matrix factorization, Nature 1999

PCA vs NMF on image data

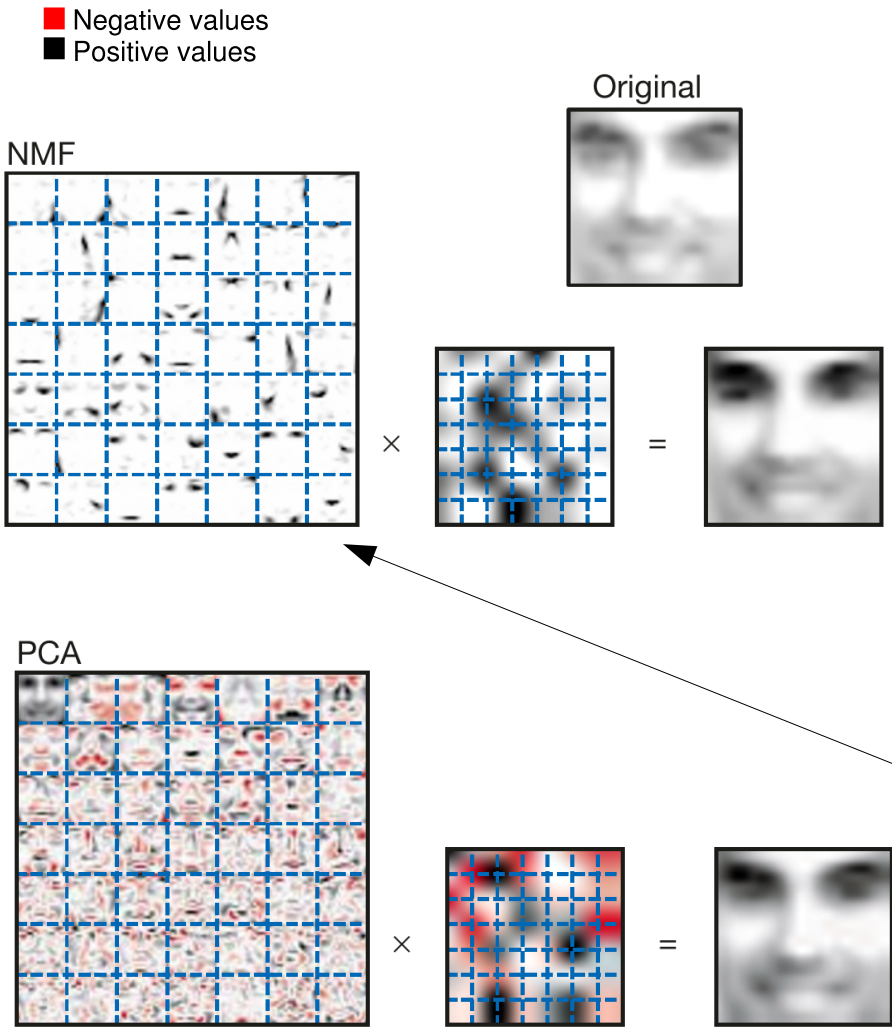


Figure 1 Non-negative matrix factorization (NMF) learns a parts-based representation of faces, whereas vector quantization (VQ) and principal components analysis (PCA) learn holistic representations. The three learning methods were applied to a database of $m = 2{,}429$ facial images, each consisting of $n = 19 \times 19$ pixels, and constituting an $n \times m$ matrix V. All three find approximate factorizations of the form $V \approx WH$, but with three different types of constraints on W and H, as described more fully in the main text and methods. As shown in the 7×7 montages, each method has learned a set of $r = 49$ basis images. Positive values are illustrated with black pixels and negative values with red pixels. A particular instance of a face, shown at top right, is approximately represented by a linear superposition of basis images. The coefficients of the linear superposition are shown next to each montage, in a 7×7 grid, and the resulting superpositions are shown on the other side of the equality sign. Unlike VQ and PCA, NMF learns to represent faces with a set of basis images resembling parts of faces.

whole face. The basis images for PCA are 'eigenfaces', some of which resemble distorted versions of whole faces[6]. The NMF basis is radically different: its images are localized features that correspond better with intuitive notions of the parts of faces.

See D.D. Lee and H.S. Seung, Learning the parts of objects by non-negative matrix factorization, Nature 1999

PCA vs NMF on image data

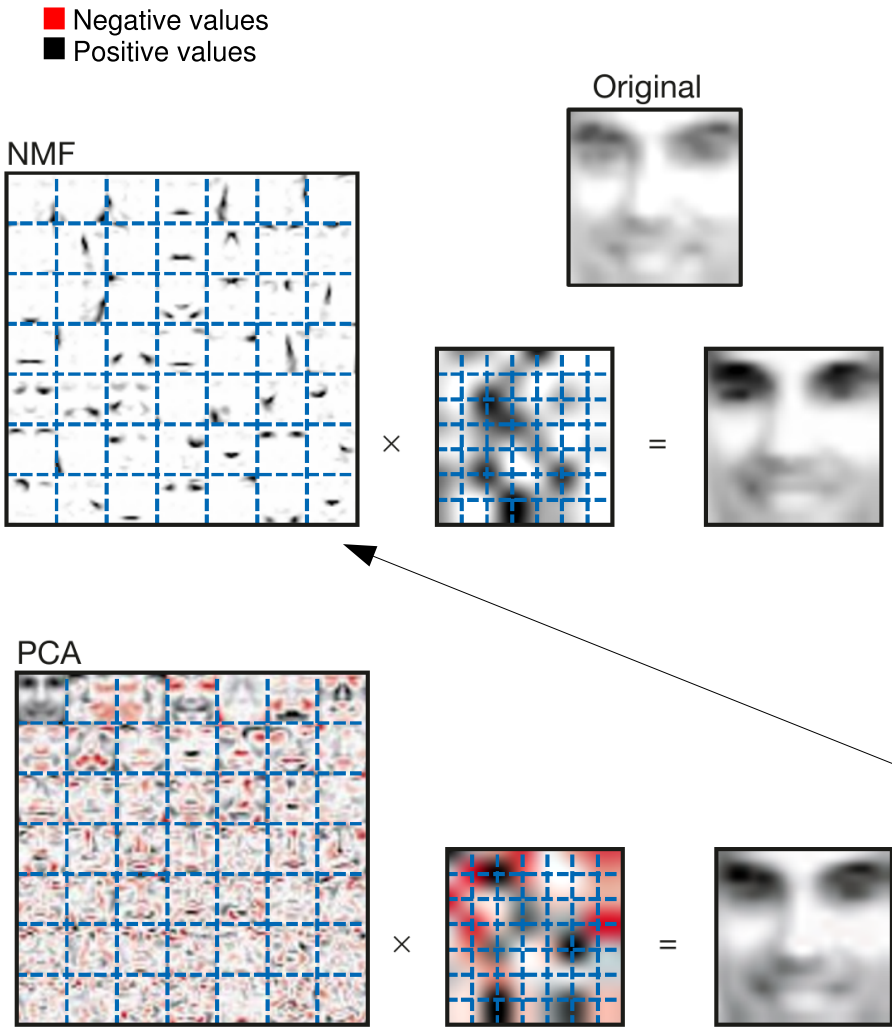


Figure 1 Non-negative matrix factorization (NMF) learns a parts-based representation of faces, whereas vector quantization (VQ) and principal components analysis (PCA) learn holistic representations. The three learning methods were applied to a database of $m = 2{,}429$ facial images, each consisting of $n = 19 \times 19$ pixels, and constituting an $n \times m$ matrix V. All three find approximate factorizations of the form $V \approx WH$, but with three different types of constraints on W and H, as described more fully in the main text and methods. As shown in the 7×7 montages, each method has learned a set of $r = 49$ basis images. Positive values are illustrated with black pixels and negative values with red pixels. A particular instance of a face, shown at top right, is approximately represented by a linear superposition of basis images. The coefficients of the linear superposition are shown next to each montage, in a 7×7 grid, and the resulting superpositions are shown on the other side of the equality sign. Unlike VQ and PCA, NMF learns to represent faces with a set of basis images resembling parts of faces.

whole face. The basis images for PCA are 'eigenfaces', some of which resemble distorted versions of whole faces[6]. The NMF basis is radically different: its images are localized features that correspond better with intuitive notions of the parts of faces.

The constraint of non-negativity helps the **interpretability** of the basis vectors and the low-dim. representation.

See D.D. Lee and H.S. Seung, Learning the parts of objects by non-negative matrix factorization, Nature 1999

Summary

1. Dimensionality reduction is the unsupervised learning task aimed to obtain low-dimensional representations of the data.

Summary

1. Dimensionality reduction is the unsupervised learning task aimed to obtain low-dimensional representations of the data.

2. PCA is based on a linear transformation (a projection onto an m-dimensional subspace) designed to retain maximal information on the original data (as quantified via maximal variance or minimum approximation MSE).

Summary

1. Dimensionality reduction is the unsupervised learning task aimed to obtain low-dimensional representations of the data.

2. PCA is based on a linear transformation (a projection onto an m-dimensional subspace) designed to retain maximal information on the original data (as quantified via maximal variance or minimum approximation MSE).

3. The choice of m is based on spectrum and variance captured by the PCs.

Summary

1. Dimensionality reduction is the unsupervised learning task aimed to obtain low-dimensional representations of the data.

2. PCA is based on a linear transformation (a projection onto an m-dimensional subspace) designed to retain maximal information on the original data (as quantified via maximal variance or minimum approximation MSE).

3. The choice of m is based on spectrum and variance captured by the PCs.

4. The PCA low-dimensional representation, by clustering similar data, is useful for feature discovery and data inspection.

Summary

1. Dimensionality reduction is the unsupervised learning task aimed to obtain low-dimensional representations of the data.

2. PCA is based on a linear transformation (a projection onto an m-dimensional subspace) designed to retain maximal information on the original data (as quantified via maximal variance or minimum approximation MSE).

3. The choice of m is based on spectrum and variance captured by the PCs.

4. The PCA low-dimensional representation, by clustering similar data, is useful for feature discovery and data inspection.

5. Kernel PCA gives a non-linear generalisation of PCA using the kernel trick.

Summary

1. Dimensionality reduction is the unsupervised learning task aimed to obtain low-dimensional representations of the data.

2. PCA is based on a linear transformation (a projection onto an m-dimensional subspace) designed to retain maximal information on the original data (as quantified via maximal variance or minimum approximation MSE).

3. The choice of m is based on spectrum and variance captured by the PCs.

4. The PCA low-dimensional representation, by clustering similar data, is useful for feature discovery and data inspection.

5. Kernel PCA gives a non-linear generalisation of PCA using the kernel trick.

6. Sparse PCA enables to find sparser, thus more interpretable, PCs.

Summary

1. Dimensionality reduction is the unsupervised learning task aimed to obtain low-dimensional representations of the data.

2. PCA is based on a linear transformation (a projection onto an m-dimensional subspace) designed to retain maximal information on the original data (as quantified via maximal variance or minimum approximation MSE).

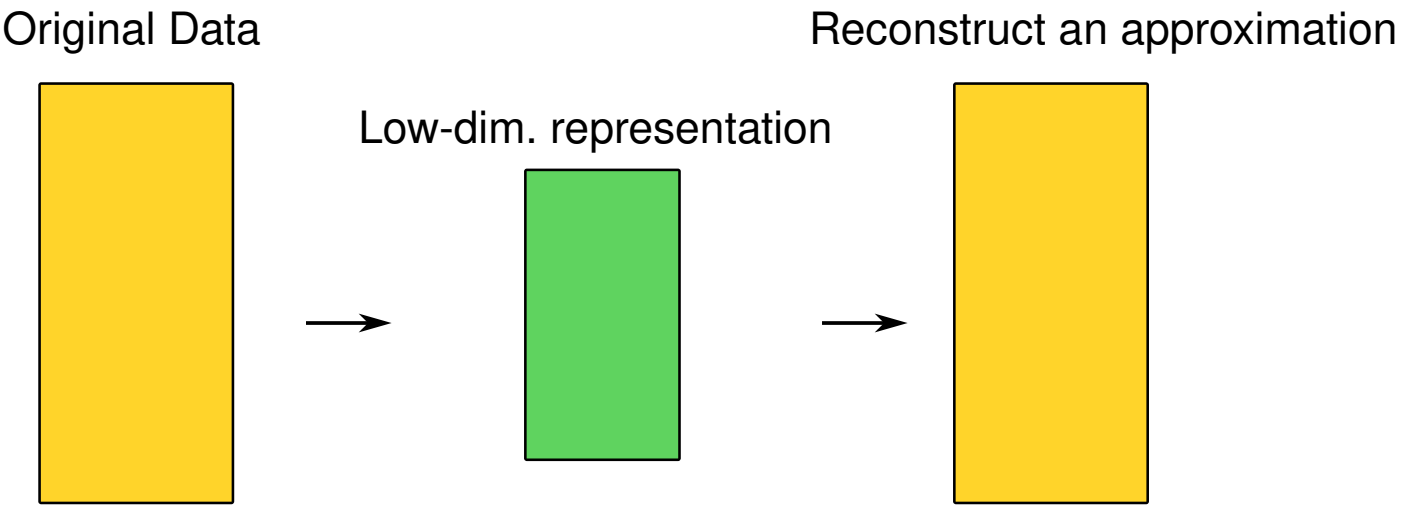
3. The choice of m is based on spectrum and variance captured by the PCs.

4. The PCA low-dimensional representation, by clustering similar data, is useful for feature discovery and data inspection.

5. Kernel PCA gives a non-linear generalisation of PCA using the kernel trick.

6. Sparse PCA enables to find sparser, thus more interpretable, PCs.

7. NMF constrains the matrix factorisation to be non-negative for interpretability



The idea of learning lower-dimensional representation (encodings) for a set of data by minimising the reconstruction error from the compressed data encoding can be generalised to more complex approaches, e.g.: neural network architectures

AUTOENCODER

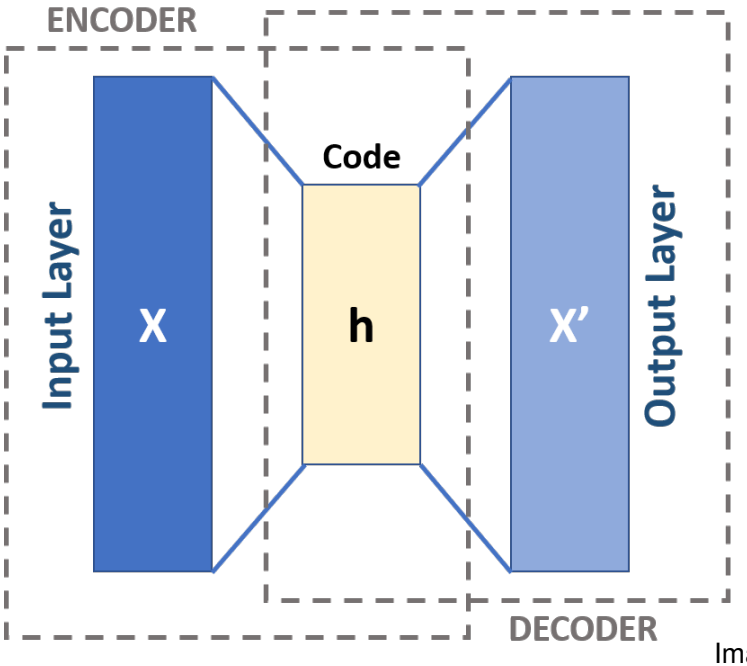


Image from Wikipedia