Dimension reduction

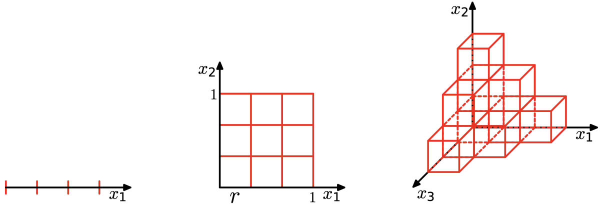
Mathurin Massias & Titouan Vayer
email: firstname.lastname@inria.fr

Table of contents

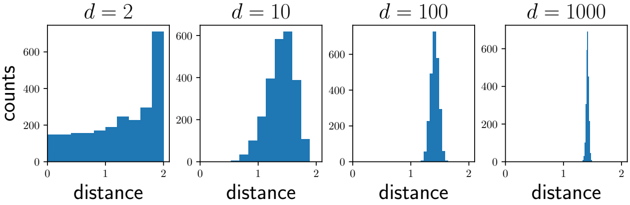
Why dimension reduction?

- Visualize high-dimensional data (in 2D or 3D).
- Interpret the data: find meaningful compact representations.
- Compress the data: advantages for storage and robustness.
- Reveal the “structure of the data”.
- Avoid the curse of dimensionality.

Curse of dimensionality

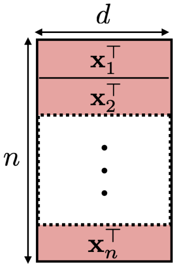


- Number of points needed to cover the hypercube cube $[0,1]^d$ with precision r: $\left(\frac{2}{r}\right)^d$
- Distances between points become meaningless. Pairwise distances between 50 points on the unit sphere of $\mathbb{R}^d$:



Unsupervised dataset

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

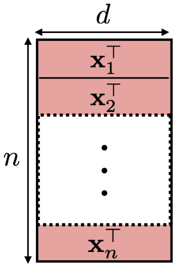


Unsupervised learning

- The dataset contains the samples $(\mathbf{x}_i)_{i=1}^n$ where n is the number of samples of size d.
- d and n define the dimensionality of the learning problem.
- Data stored as a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ that contains the training samples as rows.

Unsupervised dataset

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

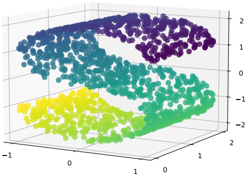


Unsupervised learning

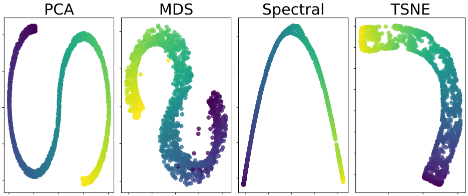
- The dataset contains the samples $(\mathbf{x}_i)_{i=1}^n$ where n is the number of samples of size d.
- d and n define the dimensionality of the learning problem.
- Data stored as a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ that contains the training samples as rows.
- ⚠ in ML vectors are sometimes described **in row instead of column**

The big picture

Original dataset



$\Rightarrow$



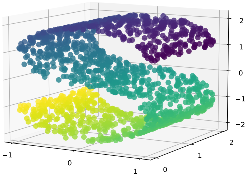
Objective

$$(\mathbf{x}_i)_{i=1}^n \quad \Rightarrow \quad \left(\tilde{\mathbf{x}}_i \in \mathbb{R}^k\right)_{i=1}^n \text{ with } k \ll d$$

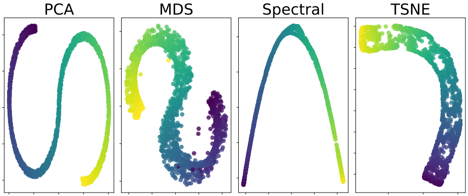
- Project the data into a low dimensional space $\mathbb{R}^k$ with $k \ll d$
- Preserve the information in the data (class, subspace, similarities)

The big picture

Original dataset



$\Rightarrow$



Objective

$$(\mathbf{x}_i)_{i=1}^n \quad \Rightarrow \quad \left(\tilde{\mathbf{x}}_i \in \mathbb{R}^k\right)_{i=1}^n \text{ with } k \ll d$$

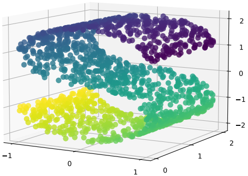
- Project the data into a low dimensional space $\mathbb{R}^k$ with $k \ll d$
- Preserve the information in the data (class, subspace, similarities)

Modeling choices

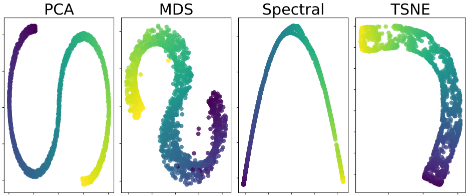
- Linear, non linear projection?
- Similarity between samples.

The big picture

Original dataset



$\Rightarrow$



Objective

$$(\mathbf{x}_i)_{i=1}^n \quad \Rightarrow \quad \left(\tilde{\mathbf{x}}_i \in \mathbb{R}^k\right)_{i=1}^n \text{ with } k \ll d$$

- Project the data into a low dimensional space $\mathbb{R}^k$ with $k \ll d$
- Preserve the information in the data (class, subspace, similarities)

Modeling choices

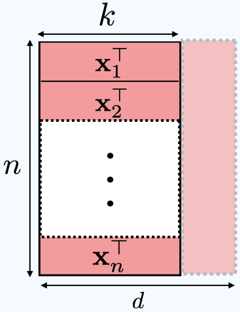
- Linear, non linear projection?
- Similarity between samples.

Methods

- PCA, random projections.
- Non-linear methods (MDS, tSNE, Auto-Encoder)

Dimension reduction vs subsampling

Dimension reduction

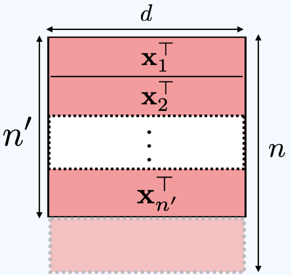


Random projections

Feature selection, sparsity

Minimum distortion embedding, PCA

Subsampling



Coresets

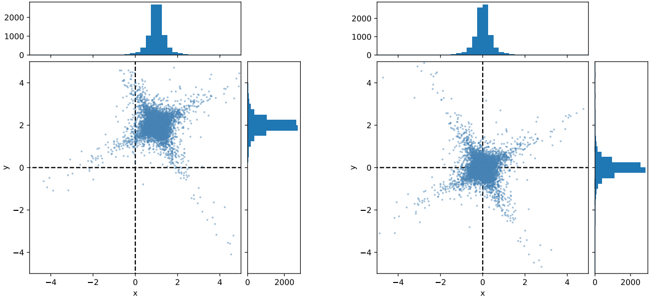
Importance sampling

Table of contents

The principle

Setting

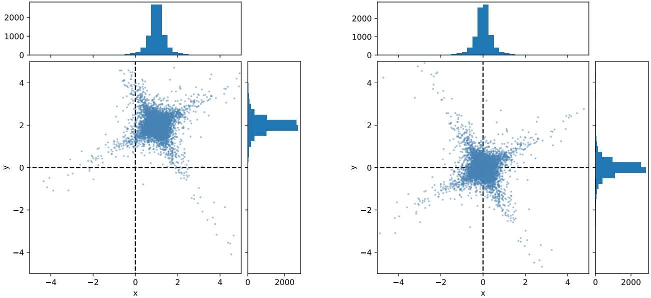
- A dataset $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)^\top \in \mathbb{R}^{n \times d}$ with d big.
- Suppose for simplicity $\sum_{i=1}^n \mathbf{x}_i = 0$ (centered data), *i.e.* $\mathbf{X}^\top \mathbf{1}_n = 0$.



The principle

Setting

- A dataset $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)^\top \in \mathbb{R}^{n \times d}$ with d big.
- Suppose for simplicity $\sum_{i=1}^n \mathbf{x}_i = 0$ (centered data), *i.e.* $\mathbf{X}^\top \mathbf{1}_n = 0$.



Goal

- Find coordinates $\tilde{\mathbf{x}}_i = f(\mathbf{x}_i)$ in $\mathbb{R}^k$ with $k \ll d$.
- The new data $(\tilde{\mathbf{x}}_i)_{i \in [\![n]\!]}$ should "look like" $\mathbf{X}$ (to be defined).

The principle of PCA

Linear mapping according to a reconstruction principle

- Find $\mathbf{U} = (\mathbf{u}_1, \cdots, \mathbf{u}_k) \in \mathbb{R}^{d \times k}$ with $\mathbf{u}_n^\top \mathbf{u}_m = \delta_{nm}$ (orthonormal vectors)
- Dimension reduction via linear mapping: $\tilde{\mathbf{x}}_i = \mathbf{U}^\top \mathbf{x}_i \in \mathbb{R}^k$

The principle of PCA

Linear mapping according to a reconstruction principle

- Find $\mathbf{U} = (\mathbf{u}_1, \cdots, \mathbf{u}_k) \in \mathbb{R}^{d \times k}$ with $\mathbf{u}_n^\top \mathbf{u}_m = \delta_{nm}$ (orthonormal vectors)
- Dimension reduction via linear mapping: $\tilde{\mathbf{x}}_i = \mathbf{U}^\top \mathbf{x}_i \in \mathbb{R}^k$
- What make a $\mathbf{U}$ "better" than another?

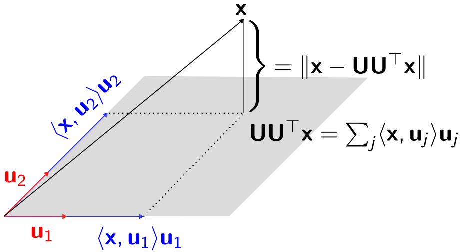
The principle of PCA

Linear mapping according to a reconstruction principle

- Find $\mathbf{U} = (\mathbf{u}_1, \cdots, \mathbf{u}_k) \in \mathbb{R}^{d \times k}$ with $\mathbf{u}_n^\top \mathbf{u}_m = \delta_{nm}$ (orthonormal vectors)
- Dimension reduction via linear mapping: $\tilde{\mathbf{x}}_i = \mathbf{U}^\top \mathbf{x}_i \in \mathbb{R}^k$
- Reconstruction principle (Pearson 1901):

$$\min_{\substack{\mathbf{U} \in \mathbb{R}^{d \times k} \\ \mathbf{U}^\top \mathbf{U} = \mathbf{I}_k}} \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{U}\tilde{\mathbf{x}}_i\|_2^2 = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{U}\mathbf{U}^\top \mathbf{x}_i\|_2^2$$

- $\mathbf{U}\mathbf{U}^\top \mathbf{x}_i$ is the linear projection of $\mathbf{x}_i$ onto $\mathrm{span}(\mathbf{u}_1, \cdots, \mathbf{u}_k)$



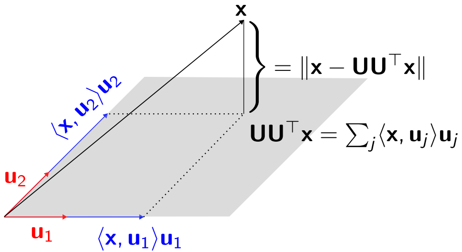
The principle of PCA

Linear mapping according to a reconstruction principle

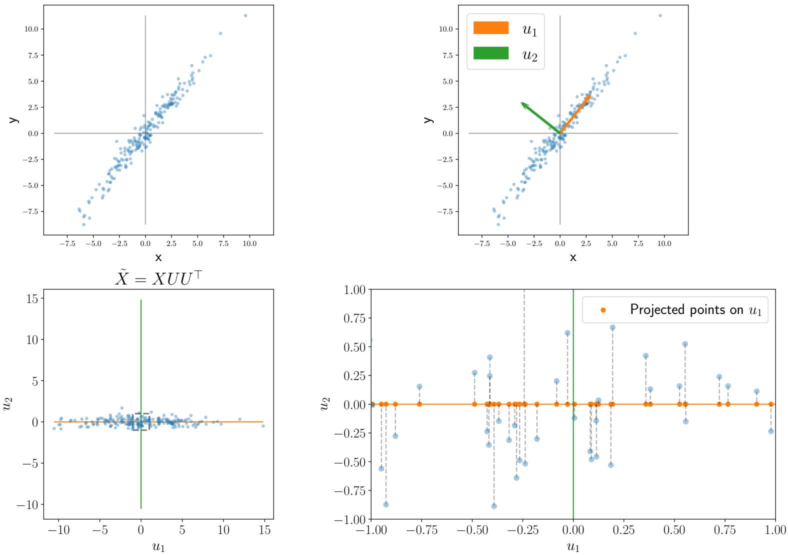
- Find $\mathbf{U} = (\mathbf{u}_1, \cdots, \mathbf{u}_k) \in \mathbb{R}^{d\times k}$ with $\mathbf{u}_n^\top \mathbf{u}_m = \delta_{nm}$ (orthonormal vectors)
- Dimension reduction via linear mapping: $\tilde{\mathbf{x}}_i = \mathbf{U}^\top \mathbf{x}_i \in \mathbb{R}^k$
- Reconstruction principle (Pearson 1901):

$$\min_{\substack{\mathbf{U}\in\mathbb{R}^{d\times k}\\ \mathbf{U}^\top\mathbf{U}=\mathbf{I}_k}} \frac{1}{n}\sum_{i=1}^n \|\mathbf{x}_i - \mathbf{U}\tilde{\mathbf{x}}_i\|_2^2 = \frac{1}{n}\sum_{i=1}^n \|\mathbf{x}_i - \mathbf{U}\mathbf{U}^\top\mathbf{x}_i\|_2^2$$

- After finding a sol. $\mathbf{U}^\star$, $\mathbf{x}_i \approx \sum_{j=1}^k \langle \mathbf{x}_i, \mathbf{u}_j^\star\rangle \mathbf{u}_j^\star$ (equality when $k = d$).



Illustration



Variance interpretation

Equivalent problem

- The PCA problem is equivalent to the *non-convex* quadratic problem:

$$\max_{\substack{\mathbf{U}\in\mathbb{R}^{d\times k}\\ \mathbf{U}^\top\mathbf{U}=\mathbf{I}_k}} \frac{1}{n}\sum_{i=1}^{n}\|\mathbf{U}^\top\mathbf{x}_i\|_2^2 = \mathrm{tr}\left(\mathbf{U}^\top \overbrace{(\frac{1}{n}\mathbf{X}^\top\mathbf{X})}^{\widehat{\mathbf{\Sigma}}}\mathbf{U}\right)$$

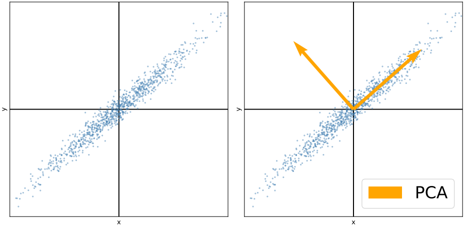
Variance interpretation

Equivalent problem

- The PCA problem is equivalent to the *non-convex* quadratic problem:

$$\max_{\substack{\mathbf{U}\in\mathbb{R}^{d\times k}\\ \mathbf{U}^\top\mathbf{U}=\mathbf{I}_k}} \frac{1}{n}\sum_{i=1}^{n}\|\mathbf{U}^\top\mathbf{x}_i\|_2^2 = \mathrm{tr}\left(\mathbf{U}^\top \overbrace{\left(\frac{1}{n}\mathbf{X}^\top\mathbf{X}\right)}^{\widehat{\boldsymbol{\Sigma}}}\mathbf{U}\right)$$

- Equivalent to maximizing the **variance of the projected samples $\tilde{\mathbf{x}}_i$**.



- *Empirical covariance matrix* $\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\mathbf{X}^\top\mathbf{X} \in \mathbb{R}^{d\times d}$

recap: Two views on PCA

The PCA is the linear mapping $\mathbf{x} \mapsto \tilde{\mathbf{x}} = \mathbf{U}\mathbf{x} \in \mathbb{R}^k$ that (equivalently):

- minimizes the reconstruction error

$$\min_{\substack{\mathbf{U} \in \mathbb{R}^{d \times k} \\ \mathbf{U}^\top \mathbf{U} = \mathbf{I}_k}} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{U}\mathbf{U}^\top \mathbf{x}_i\|^2$$

- maximizes the variance of the projected data

$$\max_{\substack{\mathbf{U} \in \mathbb{R}^{d \times k} \\ \mathbf{U}^\top \mathbf{U} = \mathbf{I}_k}} \sum_{i=1}^{n} \|\mathbf{U}\mathbf{x}_i\|^2$$

Now how do we compute this optimal $\mathbf{U}$?

Computing PCA: the Ky-Fan theorem

Fan 1949

Let $\mathbf{A} \in \mathbb{R}^{d\times d}$ **symmetric** with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_d$ and $k \leq d$. Then,

$$\max_{\mathbf{U}\in\mathbb{R}^{d\times k}, \mathbf{U}^\top\mathbf{U}=\mathbf{I}_k} \operatorname{tr}(\mathbf{U}^\top\mathbf{A}\mathbf{U}) = \sum_{i=1}^{k} \lambda_i \,. \tag{1}$$

A solution of (1) is given by $\mathbf{U}^\star = (\mathbf{u}_{\mathbf{A}_1}, \ldots, \mathbf{u}_{\mathbf{A}_k})$ where $\mathbf{u}_{\mathbf{A}_1}, \ldots, \mathbf{u}_{\mathbf{A}_k}$ are eigenvectors of $\mathbf{A}$ respectively associated to the top-k eigenvalues.

Computing PCA: the Ky-Fan theorem

Fan 1949

Let $\mathbf{A} \in \mathbb{R}^{d \times d}$ **symmetric** with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_d$ and $k \leq d$. Then,

$$\max_{\mathbf{U} \in \mathbb{R}^{d \times k}, \mathbf{U}^\top \mathbf{U} = \mathbf{I}_k} \operatorname{tr}(\mathbf{U}^\top \mathbf{A} \mathbf{U}) = \sum_{i=1}^{k} \lambda_i \,. \tag{1}$$

A solution of (1) is given by $\mathbf{U}^\star = (\mathbf{u}_{\mathbf{A}_1}, \dots, \mathbf{u}_{\mathbf{A}_k})$ where $\mathbf{u}_{\mathbf{A}_1}, \dots, \mathbf{u}_{\mathbf{A}_k}$ are eigenvectors of $\mathbf{A}$ respectively associated to the top-k eigenvalues.

Consequences for PCA

- Solution of PCA: find the k largest eigenvalues of $\widehat{\mathbf{\Sigma}} = \frac{1}{n} \mathbf{X}^\top \mathbf{X} \in \mathbb{R}^{d \times d}$.
- Solution $\mathbf{U}^\star = (\mathbf{u}_1^\star, \dots, \mathbf{u}_k^\star)$ associated to the top-k eigenvalues of $\widehat{\mathbf{\Sigma}}$.
- $\mathbf{u}_1^\star \in \mathbb{R}^d, \dots, \mathbf{u}_k^\star \in \mathbb{R}^d$ are called *principal components*.
- ⚠ the decomposition is not unique ! (eigenvectors sign flip)

Computing PCA: the Ky-Fan theorem

Fan 1949

Let $\mathbf{A} \in \mathbb{R}^{d \times d}$ **symmetric** with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_d$ and $k \leq d$. Then,

$$\max_{\mathbf{U} \in \mathbb{R}^{d \times k}, \mathbf{U}^\top \mathbf{U} = \mathbf{I}_k} \operatorname{tr}(\mathbf{U}^\top \mathbf{A} \mathbf{U}) = \sum_{i=1}^{k} \lambda_i \,. \tag{1}$$

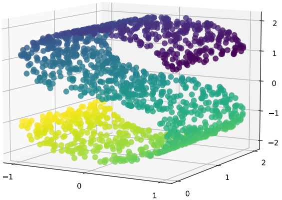
A solution of (1) is given by $\mathbf{U}^\star = (\mathbf{u}_{\mathbf{A}_1}, \dots, \mathbf{u}_{\mathbf{A}_k})$ where $\mathbf{u}_{\mathbf{A}_1}, \dots, \mathbf{u}_{\mathbf{A}_k}$ are eigenvectors of $\mathbf{A}$ respectively associated to the top-k eigenvalues.

Consequences for PCA

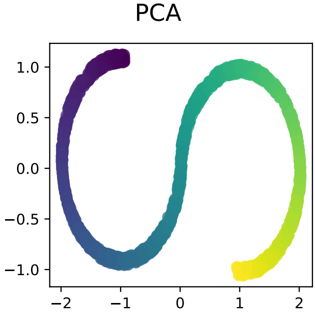
- Solution of PCA: find the k largest eigenvalues of $\widehat{\mathbf{\Sigma}} = \frac{1}{n} \mathbf{X}^\top \mathbf{X} \in \mathbb{R}^{d \times d}$.
- Solution $\mathbf{U}^\star = (\mathbf{u}_1^\star, \dots, \mathbf{u}_k^\star)$ associated to the top-k eigenvalues of $\widehat{\mathbf{\Sigma}}$.
- $\mathbf{u}_1^\star \in \mathbb{R}^d, \dots, \mathbf{u}_k^\star \in \mathbb{R}^d$ are called *principal components*.
- ⚠ the decomposition is not unique ! (eigenvectors sign flip)

Example with 3D data

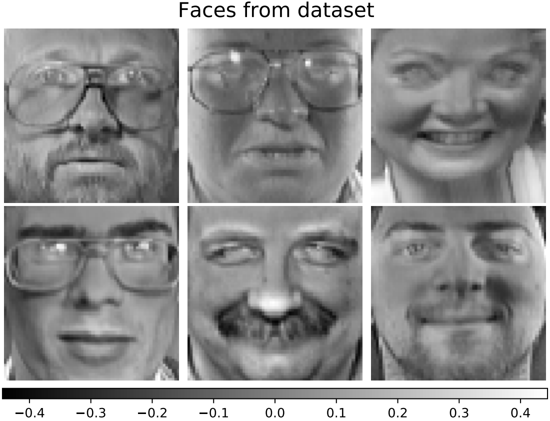
- Simple 3D data.



- Projection onto the two firsts principal components ($d = 3 \to k = 2$).



Example with eigenfaces

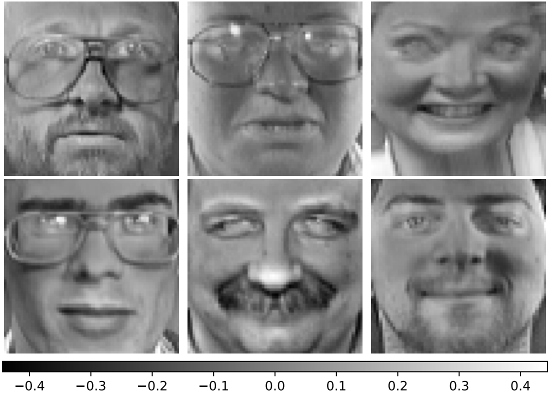


Setting

- Each image is a vector $\mathbf{x}_i \in \mathbb{R}^{4096}$ ($d = 4096$ pixels), $n = 400$ images.

Example with eigenfaces

Faces from dataset



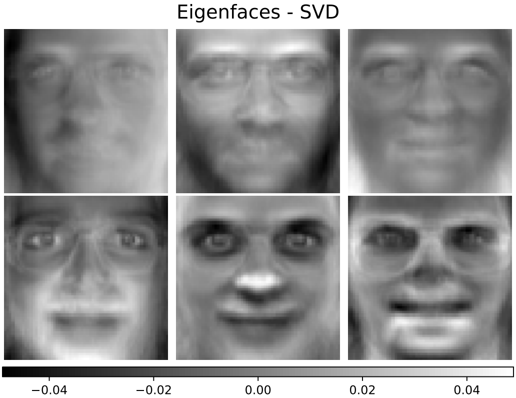
Setting

- Each image is a vector $\mathbf{x}_i \in \mathbb{R}^{4096}$ ($d = 4096$ pixels), $n = 400$ images.
- Find k "eigenfaces": principal components $\mathbf{u}_1^\star, \ldots, \mathbf{u}_k^\star \in \mathbb{R}^d$.
- Idea: explain images via $\mathbf{x}_i \approx \sum_{j=1}^k \langle \mathbf{x}_i, \mathbf{u}_j^\star \rangle \mathbf{u}_j^\star$, in other words

$$\text{image } \approx \alpha_1 \times \text{ eigenface 1} + \cdots + \alpha_k \times \text{ eigenface k}$$

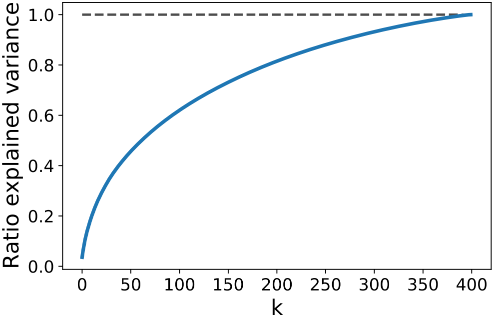
Example with eigenfaces

- Result with $k = 6$



Example with eigenfaces

- How to choose k? ratio explained variance: $r = \sum_{i=1}^{k} \lambda_i / \sum_{i=1}^{d} \lambda_i$



- Explanation:

$$
\begin{aligned}
r &= \sum_{i=1}^{k} \lambda_i / \sum_{i=1}^{d} \lambda_i \\
&= \mathrm{tr}\left((\mathbf{U}^\star)^\top \widehat{\boldsymbol{\Sigma}} \mathbf{U}^\star \right) / \mathrm{tr}(\widehat{\boldsymbol{\Sigma}}) \\
&= \frac{1}{n} \sum_{i=1}^{n} \|\tilde{\mathbf{x}}_i\|_2^2 / \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i\|_2^2
\end{aligned}
$$

How to compute the PCA?

The "naive" way

- Find the eigenvalue decomposition of $\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\mathbf{X}^\top\mathbf{X} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^\top$
- Compute $\widehat{\boldsymbol{\Sigma}}$: $\mathcal{O}(nd^2)$ operations.
- Eigenvalue decomposition : $\mathcal{O}(d^3)$ operations.
- Keep only the k-largest eigenvalues associated to k eigenvectors.
- Space complexity: $\mathcal{O}(d^2)$
- Time complexity: $\mathcal{O}(nd^2 + d^3)$

How to compute the PCA?

The "naive" way

- Find the eigenvalue decomposition of $\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\mathbf{X}^\top\mathbf{X} = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i\mathbf{x}_i^\top$
- Compute $\widehat{\boldsymbol{\Sigma}}$: $\mathcal{O}(nd^2)$ operations.
- Eigenvalue decomposition : $\mathcal{O}(d^3)$ operations.
- Keep only the k-largest eigenvalues associated to k eigenvectors.
- Space complexity: $\mathcal{O}(d^2)$
- Time complexity: $\mathcal{O}(nd^2 + d^3)$

The right way

- Compute the singular value decomposition (SVD) of $\mathbf{X}$!

Table of contents

One of the most useful tools in linear algebra

Singular value decomposition

Let $\mathbf{X} \in \mathbb{R}^{n \times d}$. Then $\mathbf{X}$ can be decomposed as

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top \tag{2}$$

where $\mathbf{U} \in \mathbb{R}^{n \times n}, \mathbf{V} \in \mathbb{R}^{d \times d}$ are *unitary* ($\mathbf{U}^\top\mathbf{U} = \mathbf{U}\mathbf{U}^\top = \mathbf{I}_n$, $\mathbf{V}^\top\mathbf{V} = \mathbf{V}\mathbf{V}^\top = \mathbf{I}_d$) and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times d}$ is a rectangular diagonal matrix with *non-negative* real numbers $(\sigma_i)_{i \in [\![\min\{n,d\}]\!]}$ on the diagonal, called *singular values*.

One of the most useful tools in linear algebra

Singular value decomposition

Let $\mathbf{X} \in \mathbb{R}^{n \times d}$. Then $\mathbf{X}$ can be decomposed as

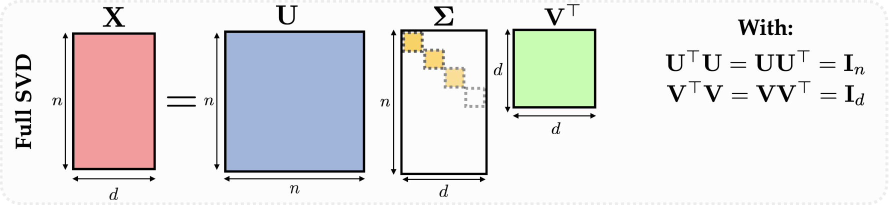
$$\mathbf{X} = \mathbf{U\Sigma V}^\top \tag{2}$$

where $\mathbf{U} \in \mathbb{R}^{n \times n}, \mathbf{V} \in \mathbb{R}^{d \times d}$ are *unitary* ($\mathbf{U}^\top\mathbf{U} = \mathbf{UU}^\top = \mathbf{I}_n$, $\mathbf{V}^\top\mathbf{V} = \mathbf{VV}^\top = \mathbf{I}_d$) and $\mathbf{\Sigma} \in \mathbb{R}^{n \times d}$ is a rectangular diagonal matrix with *non-negative* real numbers $(\sigma_i)_{i \in [\![\min\{n,d\}]\!]}$ on the diagonal, called *singular values*.

Properties

- $\mathrm{rank}(\mathbf{X})$ = number of non-zero $\sigma_i's$.
- The *columns* of $\mathbf{V}$ are eigenvectors of $\mathbf{X}^\top\mathbf{X} \in \mathbb{R}^{d \times d}$.
- The *columns* of $\mathbf{U}$ are eigenvectors of $\mathbf{XX}^\top \in \mathbb{R}^{n \times n}$.
- We have $\sigma_i = \sqrt{\mathrm{eigenvalue}_i(\mathbf{X}^\top\mathbf{X})} = \sqrt{\mathrm{eigenvalue}_i(\mathbf{XX}^\top)}$.
- We have the relations $\mathbf{Xv}_i = \sigma_i\mathbf{u}_i, \mathbf{X}^\top\mathbf{u}_i = \sigma_i\mathbf{v}_i$.

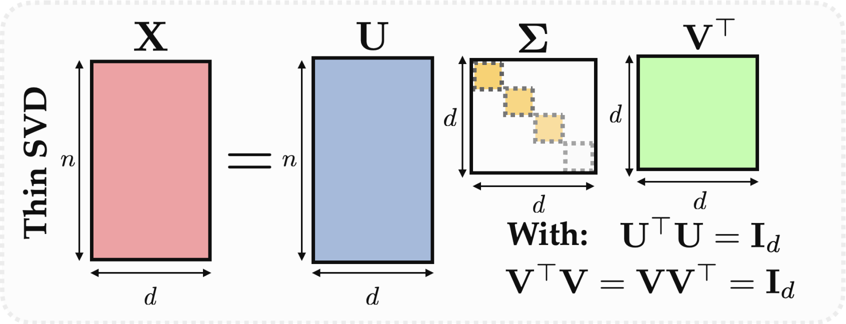
SVD: many flavors



Full SVD (image in case $n \geq d$)

- Generalizes eigenvalue decomposition for non-symmetric matrices.
- Complexity: $\mathcal{O}(nd \min\{n, d\})$ (Golub–Reinsch algorithm see Cline and Dhillon 2006).
- To find the eigenvalues of $\mathbf{X}^\top\mathbf{X}$ or $\mathbf{X}\mathbf{X}^\top$ we do not even have to compute these matrices!

SVD: many flavors

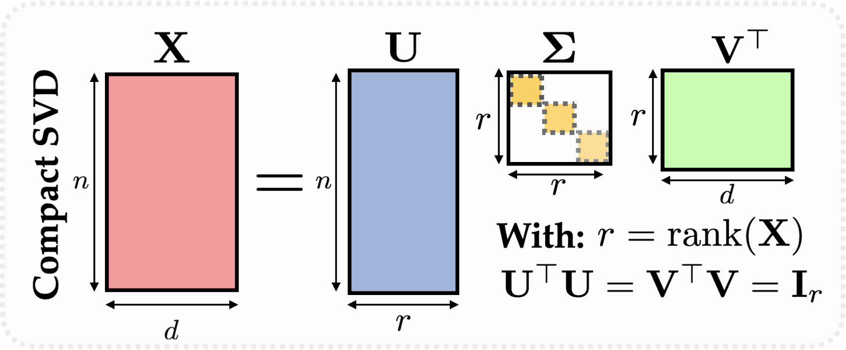


Thin SVD

- If we write $\mathbf{U} = (\mathbf{u}_1, \cdots, \mathbf{u}_n), \mathbf{V} = (\mathbf{v}_1, \cdots, \mathbf{v}_d)$ the full SVD, then Thin SVD gives:

$$\mathbf{X} = \sum_{i=1}^{\min\{n,d\}} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$$

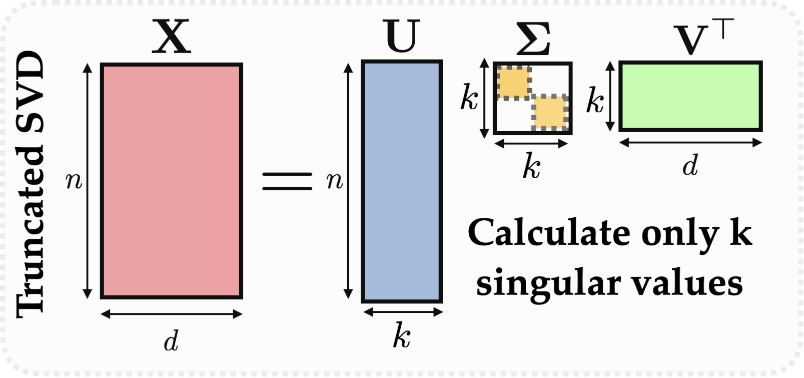
SVD: many flavors



Compact SVD

- Keep only the non-zero singular values. In particular $r = \mathrm{rank}(\mathbf{X})$.
- The pseudo-inverse of $\mathbf{X}$ is given by $\mathbf{X}^\dagger = \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^\top$.

SVD: many flavors



Truncated SVD

- Best rank-k approximation of $\mathbf{X}$ (in the sense of $\|\cdot\|_F, \|\cdot\|_{2\to 2}$).
- The solution of the PCA is given by $\mathbf{V} \in \mathbb{R}^{d\times k}$.
- The embedding $(\tilde{\mathbf{x}}_i)_{i\in[\![n]\!]}$ in low dim of PCA is given by $\mathbf{U\Sigma} \in \mathbb{R}^{n\times k}$.
- Efficient algorithms $\mathcal{O}(ndk)$ (Halko, Martinsson, and Tropp 2011).

PCA: a recap

- Dimension reduction $\mathbb{R}^d \to \mathbb{R}^k$ via a linear mapping.
- Defined with a matrix $\mathbf{U}$ with orthonormal columns.
- Follows a reconstruction principle.
- Maximizes the variance of the projected samples.
- Used for compression, interpretation, robustness.
- Can be computed with SVD in $\mathcal{O}(ndk)$ time with truncated SVD (randomness).
- In practice it is common to normalize your data before doing PCA.

A word on Kernel PCA

From PCA...

- PCA solves $\max_{\substack{\mathbf{U} \in \mathbb{R}^{d \times k} \\ \mathbf{U}^\top \mathbf{U} = \mathbf{I}_k}} \operatorname{tr}\left(\mathbf{U}^\top (\mathbf{X}^\top \mathbf{X}) \mathbf{U}\right)$.
- Eigenvalue decomposition of $\mathbf{X}^\top \mathbf{X} = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top \in \mathbb{R}^{d \times d}$.
- Same non-zero eigenvalues as $\mathbf{X}\mathbf{X}^\top = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle)_{ij} \in \mathbb{R}^{n \times n}$ (exercise).

A word on Kernel PCA

From PCA...

- PCA solves $\max\limits_{\substack{\mathbf{U}\in\mathbb{R}^{d\times k}\\ \mathbf{U}^\top\mathbf{U}=\mathbf{I}_k}} \operatorname{tr}\left(\mathbf{U}^\top(\mathbf{X}^\top\mathbf{X})\mathbf{U}\right)$.
- Eigenvalue decomposition of $\mathbf{X}^\top\mathbf{X} = \sum_{i=1}^n \mathbf{x}_i\mathbf{x}_i^\top \in \mathbb{R}^{d\times d}$.
- Same non-zero eigenvalues as $\mathbf{X}\mathbf{X}^\top = (\langle \mathbf{x}_i, \mathbf{x}_j\rangle)_{ij} \in \mathbb{R}^{n\times n}$ (exercise).

... to Kernel PCA (Schölkopf, Smola, and Müller 2005)

- PCA in a *high-dimensional non-linear* embedding $\Phi(\mathbf{x})$ of the data.
- *Kernel trick*: embedding is *implicit* we only need $\mathbf{K} = \left(\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j)\rangle\right)_{ij}$.
- Kernel PCA: eigenvalue decomposition of $\mathbf{K} \in \mathbb{R}^{n\times n}$.
- More powerful but expensive for large n.

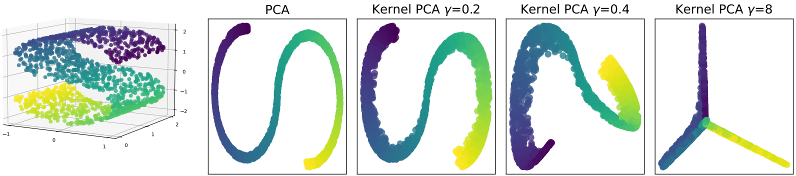
A word on Kernel PCA

From PCA...

- PCA solves $\max_{\substack{\mathbf{U}\in\mathbb{R}^{d\times k}\\ \mathbf{U}^\top\mathbf{U}=\mathbf{I}_k}} \operatorname{tr}\left(\mathbf{U}^\top(\mathbf{X}^\top\mathbf{X})\mathbf{U}\right)$.
- Eigenvalue decomposition of $\mathbf{X}^\top\mathbf{X} = \sum_{i=1}^n \mathbf{x}_i\mathbf{x}_i^\top \in \mathbb{R}^{d\times d}$.
- Same non-zero eigenvalues as $\mathbf{X}\mathbf{X}^\top = (\langle \mathbf{x}_i, \mathbf{x}_j\rangle)_{ij} \in \mathbb{R}^{n\times n}$ (exercise).

... to Kernel PCA (Schölkopf, Smola, and Müller 2005)

- PCA in a *high-dimensional non-linear* embedding $\Phi(\mathbf{x})$ of the data.
- *Kernel trick*: embedding is *implicit* we only need $\mathbf{K} = (\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j)\rangle)_{ij}$.
- Kernel PCA: eigenvalue decomposition of $\mathbf{K} \in \mathbb{R}^{n\times n}$.
- More powerful but expensive for large n.



A more general setting: dictionary learning

From PCA...

- One principle of PCA is to represent a sample as a linear combination $\mathbf{x} \approx \sum_{j=1}^{k} \alpha_j \mathbf{d}_j$ with $\mathbf{d}_j \in \mathbb{R}^d$.
- PCA: $\alpha_j = \langle \mathbf{x}, \mathbf{u}_j^\star \rangle, \mathbf{d}_j = \mathbf{u}_j^\star$ principal component.

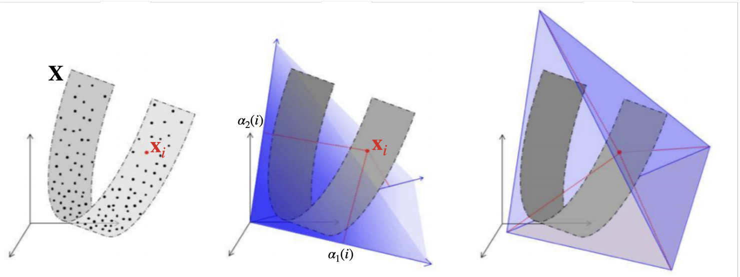
A more general setting: dictionary learning

From PCA...

- One principle of PCA is to represent a sample as a linear combination $\mathbf{x} \approx \sum_{j=1}^{k} \alpha_j \mathbf{d}_j$ with $\mathbf{d}_j \in \mathbb{R}^d$.
- PCA: $\alpha_j = \langle \mathbf{x}, \mathbf{u}_j^\star \rangle, \mathbf{d}_j = \mathbf{u}_j^\star$ principal component.

... to dictionary learning (DL)

- Represent $\mathbf{x}$ in another "basis": $\mathbf{x} \approx \mathbf{D}\boldsymbol{\alpha}$ (e.g. Fourier/Wavelet basis).
- $\mathbf{D} \in \mathbb{R}^{d \times k}$ is the *dictionary*. k might be bigger than d (overcomplete)
- $\boldsymbol{\alpha} \in \mathbb{R}^k$ is the representation of $\mathbf{x}$ in the dictionary $\mathbf{D}$.



A more general setting: dictionary learning

Find the representation

- Given a point $\mathbf{x}$ and a dictionary $\mathbf{D}$:

$$\widehat{\boldsymbol{\alpha}} = \underset{\boldsymbol{\alpha} \in C}{\arg\min} \; \|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 \tag{3}$$

- When $C = \mathbb{R}^k, \mathbf{D}^\top \mathbf{D} = \mathbf{I}_k$ then $\widehat{\boldsymbol{\alpha}} = \mathbf{D}^\top \mathbf{x}$.
- Can also be used with different losses than $\| \cdot \|_2^2$.

A more general setting: dictionary learning

Find the representation

- Given a point $\mathbf{x}$ and a dictionary $\mathbf{D}$:

$$\widehat{\boldsymbol{\alpha}} = \underset{\boldsymbol{\alpha} \in C}{\arg\min} \; \|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 \tag{3}$$

- When $C = \mathbb{R}^k, \mathbf{D}^\top \mathbf{D} = \mathbf{I}_k$ then $\widehat{\boldsymbol{\alpha}} = \mathbf{D}^\top \mathbf{x}$.
- Can also be used with different losses than $\| \cdot \|_2^2$.

Learn the representation **and** the dictionary

- Given a dataset $\mathbf{x}_1, \cdots, \mathbf{x}_n$, learn the dictionary and the representations

$$\widehat{\mathbf{D}}, \widehat{\boldsymbol{\alpha}_1}, \cdots, \widehat{\boldsymbol{\alpha}_n} = \underset{\substack{\mathbf{D} \in \mathcal{D} \\ \boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_n \in C}}{\arg\min} \; \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{D}\boldsymbol{\alpha}_i\|_2^2 \,. \tag{4}$$

- When $C = \mathbb{R}^k, \mathcal{D} = \{\mathbf{D} \in \mathbb{R}^{d \times k}, \mathbf{D}^\top \mathbf{D} = \mathbf{I}_k\}$ we retrieve the PCA.
- But various possibilities (Mairal et al. 2009)!
- Scikit-learn implementation : `sklearn.decomposition.DictionaryLearning`

One example

Sparse dictionary learning

- Given a dataset $\mathbf{x}_1, \cdots, \mathbf{x}_n$, learn the dictionary and the representations

$$\widehat{\mathbf{D}}, \widehat{\boldsymbol{\alpha}_1}, \cdots, \widehat{\boldsymbol{\alpha}_n} = \underset{\substack{\mathbf{D} \in \mathcal{D} \\ \boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_n \in C}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{D}\boldsymbol{\alpha}_i\|_2^2 . \tag{5}$$

- Take $\mathcal{D} = \{\mathbf{D} \in \mathbb{R}^{d \times k} : \forall i, \|\mathbf{d}_i\|_2 = 1\}$ (normalized columns).
- Take $C = \{\boldsymbol{\alpha} : \|\boldsymbol{\alpha}\|_1 \leq \lambda\}$ sparsity promoting regularization.
- Example $d = 2, k = 3$ (not dimension reduction!!)

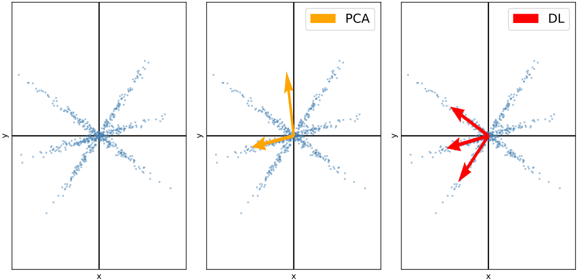


Table of contents

General setting

Preserve the pairwise distances (Agrawal, Ali, and Boyd 2021)

- A dataset $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_n), \mathbf{x}_i \in \mathbb{R}^d$.
- Find a dataset $\widetilde{\mathbf{X}} = (\widetilde{\mathbf{x}}_1, \cdots, \widetilde{\mathbf{x}}_n), \widetilde{\mathbf{x}}_i \in \mathbb{R}^k, k \ll d$ such that

$$\forall(i,j),\ \text{similarity}_{\mathbb{R}^d}(\mathbf{x}_i, \mathbf{x}_j) \approx \text{similarity}_{\mathbb{R}^k}(\widetilde{\mathbf{x}}_i, \widetilde{\mathbf{x}}_j) \text{ or}$$
$$\forall(i,j),\ \text{dissimilarity}_{\mathbb{R}^d}(\mathbf{x}_i, \mathbf{x}_j) \approx \text{dissimilarity}_{\mathbb{R}^k}(\widetilde{\mathbf{x}}_i, \widetilde{\mathbf{x}}_j)$$

- Optional: find a mapping $f : \mathbb{R}^d \to \mathbb{R}^k, \widetilde{\mathbf{x}}_i = f(\mathbf{x}_i)$.

General setting

Preserve the pairwise distances (Agrawal, Ali, and Boyd 2021)

- A dataset $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_n), \mathbf{x}_i \in \mathbb{R}^d$.
- Find a dataset $\widetilde{\mathbf{X}} = (\widetilde{\mathbf{x}}_1, \cdots, \widetilde{\mathbf{x}}_n), \widetilde{\mathbf{x}}_i \in \mathbb{R}^k, k \ll d$ such that

$$\forall (i,j),\ \text{similarity}_{\mathbb{R}^d}(\mathbf{x}_i, \mathbf{x}_j) \approx \text{similarity}_{\mathbb{R}^k}(\widetilde{\mathbf{x}}_i, \widetilde{\mathbf{x}}_j) \text{ or}$$
$$\forall (i,j),\ \text{dissimilarity}_{\mathbb{R}^d}(\mathbf{x}_i, \mathbf{x}_j) \approx \text{dissimilarity}_{\mathbb{R}^k}(\widetilde{\mathbf{x}}_i, \widetilde{\mathbf{x}}_j)$$

- Optional: find a mapping $f : \mathbb{R}^d \to \mathbb{R}^k, \widetilde{\mathbf{x}}_i = f(\mathbf{x}_i)$.

Example with $\|\cdot\|_2^2$

- Can we find $\delta \in [0,1]$ and $f : \mathbb{R}^d \to \mathbb{R}^k$ such that

$$(1-\delta)\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \leq \|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|_2^2 \leq (1+\delta)\|\mathbf{x}_i - \mathbf{x}_j\|_2^2\,? \tag{6}$$

Table of contents

Johnson-Lindenstrauss lemma

Johnson and Lindenstrauss 1984

Let $0 < \delta < 1$ and *any* dataset $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$. Provided that

$$k > 15\delta^{-2}\log(n)\,,$$

there is a matrix $\mathbf{A} \in \mathbb{R}^{k \times d}$, such that,

$$\forall (i,j) \in [\![n]\!]^2, (1-\delta)\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \leq \|\mathbf{A}\mathbf{x}_i - \mathbf{A}\mathbf{x}_j\|_2^2 \leq (1+\delta)\|\mathbf{x}_i - \mathbf{x}_j\|_2^2\,.$$

Johnson-Lindenstrauss lemma

Johnson and Lindenstrauss 1984

Let $0 < \delta < 1$ and *any* dataset $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$. Provided that

$$k > 15\delta^{-2}\log(n),$$

there is a matrix $\mathbf{A} \in \mathbb{R}^{k\times d}$, such that,

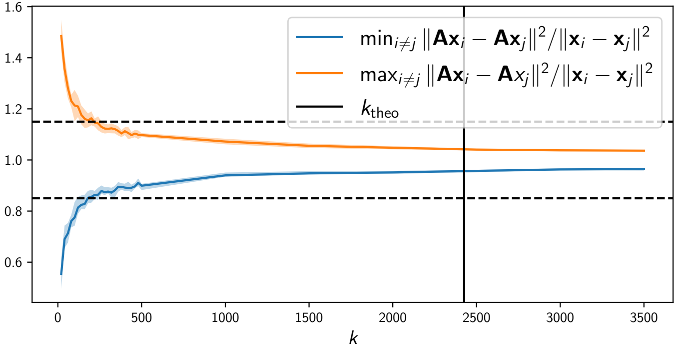
$$\forall (i,j) \in [\![n]\!]^2, (1-\delta)\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \leq \|\mathbf{A}\mathbf{x}_i - \mathbf{A}\mathbf{x}_j\|_2^2 \leq (1+\delta)\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 .$$

Important comments

- The mapping is linear + exists for any dataset!
- k does not depend on the dimension d!
- Magical: $\mathbf{A}$ can be drawn randomly: $A_{ij} \sim \mathcal{N}(0, \frac{1}{k})$.
- This is tight in some sense (Larsen and Nelson 2017).
- Caveat: $n = 300$ samples, $\delta = 10\%$ already requires $k > 8555$ (smaller in practice).

Johnson-Lindenstrauss in practice

- Real dataset in $\mathbb{R}^{38\times 7129}$, $\delta = 0.15$.
- For various choices of k, draw random Gaussian $A \sim \mathcal{N}(0, \frac{1}{k}) \in \mathbb{R}^{k\times d}$.
- Compare distances between $\mathbf{Ax}_i$s to distances between $\mathbf{x}_i$s.



Multidimensional scaling (MDS)

Learn from pairwise distances

- Distances in the big space: $D_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$ for some "metric" D.
- Find $\tilde{\mathbf{x}_1}, \cdots, \tilde{\mathbf{x}}_n \in \mathbb{R}^k$ that minimizes:

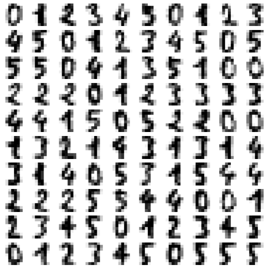
$$\mathrm{stress}_D(\tilde{\mathbf{x}_1}, \cdots, \tilde{\mathbf{x}}_n) = \sum_{i \neq j} \left(D_{ij} - \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2\right)^2 \tag{7}$$

- Eq. (7) usually called *stress minimization*: $\|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2 \approx D_{ij}$.
- Can also be used for embedding nodes $\mathbf{x}_i$ of a graph (not only Euclidean).
- Can be solved with eigenvalue decomposition.
- ⚠ No mapping from the high dim space to the lower dim space.

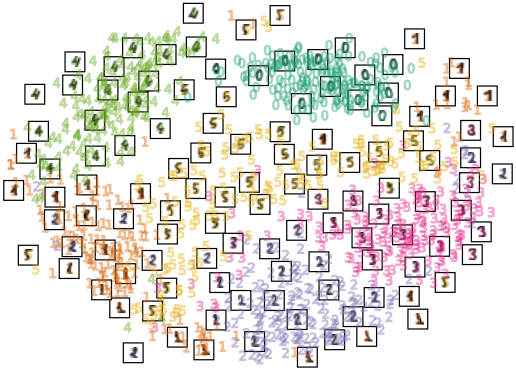
Multidimensional scaling (MDS)

- With digits dataset:

A selection from the 64-dimensional digits dataset



MDS embedding (time 2.959s)



Laplacian/spectral embedding

Learn from pairwise similarities

- Similarities in high-dim space encoded as a graph with weights $\mathbf{W}$.
- Examples: $W_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 / 2\sigma^2)$, nearest neighbors graph.

Laplacian/spectral embedding

Learn from pairwise similarities

- Similarities in high-dim space encoded as a graph with weights $\mathbf{W}$.
- Examples: $W_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/2\sigma^2)$, nearest neighbors graph.
- Find $\tilde{\mathbf{x}_1}, \cdots, \tilde{\mathbf{x}}_n \in \mathcal{S} \subset (\mathbb{R}^k)^n$ that minimizes:

$$\sum_{(i,j)\in\mathcal{E}} W_{ij}\|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2 = \mathrm{tr}(\tilde{\mathbf{X}}\mathbf{L}\tilde{\mathbf{X}}^\top) \qquad (8)$$

- Interpretation: $\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j$ close when W_{ij} is high *i.e.* high-dim points similar.

Laplacian/spectral embedding

Learn from pairwise similarities

- Similarities in high-dim space encoded as a graph with weights $\mathbf{W}$.
- Examples: $W_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/2\sigma^2)$, nearest neighbors graph.
- Find $\tilde{\mathbf{x}_1}, \cdots, \tilde{\mathbf{x}}_n \in \mathcal{S} \subset (\mathbb{R}^k)^n$ that minimizes:

$$\sum_{(i,j)\in\mathcal{E}} W_{ij}\|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2 = \mathrm{tr}(\tilde{\mathbf{X}}\mathbf{L}\tilde{\mathbf{X}}^\top) \tag{8}$$

- Interpretation: $\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j$ close when W_{ij} is high *i.e.* high-dim points similar.

- $\mathcal{S}$: constraints on the embedding (e.g. centered, standardized)
- Recover PCA with $\mathbf{W} = \mathbf{X}\mathbf{X}^\top$.
- $\mathbf{L}$ is the Laplacian of the graph with weights $\mathbf{W}$

Laplacian/spectral embedding

Learn from pairwise similarities

- Similarities in high-dim space encoded as a graph with weights $\mathbf{W}$.
- Examples: $W_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 / 2\sigma^2)$, nearest neighbors graph.
- Find $\tilde{\mathbf{x}_1}, \cdots, \tilde{\mathbf{x}}_n \in \mathcal{S} \subset (\mathbb{R}^k)^n$ that minimizes:

$$\sum_{(i,j)\in\mathcal{E}} W_{ij} \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2 = \operatorname{tr}(\tilde{\mathbf{X}}\mathbf{L}\tilde{\mathbf{X}}^\top) \tag{8}$$

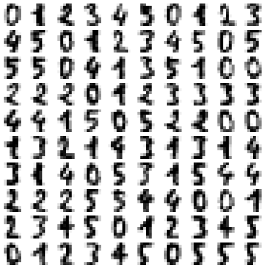
- Interpretation: $\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j$ close when W_{ij} is high *i.e.* high-dim points similar.

- $\mathcal{S}$: constraints on the embedding (e.g. centered, standardized)
- Recover PCA with $\mathbf{W} = \mathbf{X}\mathbf{X}^\top$.
- $\mathbf{L}$ is the Laplacian of the graph with weights $\mathbf{W}$
- Can be solved with eigenvalue decomposition.
- ⚠ No mapping from the high dim space to the lower dim space.

Laplacian/spectral embedding

- With digits dataset:

A selection from the 64-dimensional digits dataset



Spectral embedding (time 0.240s)

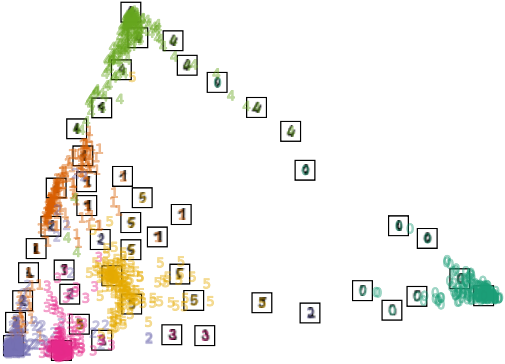


Table of contents

Stochastic neighbor embedding (SNE)

Learn from pairwise similarities

- One of the most used algorithm (G. E. Hinton and Roweis 2002).
- Similarities in the high-dim space:

$$P_{ij} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|_2^2/2\sigma_i^2)}, P_{ii} = 0\,.$$

- Similarities in the low-dim space:

$$Q_{ij} = \frac{\exp(-\|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2)}{\sum_{k \neq i} \exp(-\|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_k\|_2^2)}, Q_{ii} = 0\,.$$

Stochastic neighbor embedding (SNE)

Learn from pairwise similarities

- One of the most used algorithm (G. E. Hinton and Roweis 2002).
- Similarities in the high-dim space:

$$P_{ij} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|_2^2 / 2\sigma_i^2)}, P_{ii} = 0\,.$$

- Similarities in the low-dim space:

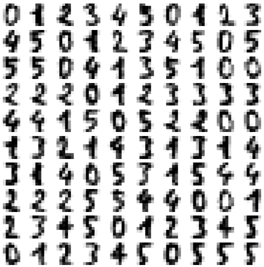
$$Q_{ij} = \frac{\exp(-\|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2)}{\sum_{k \neq i} \exp(-\|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_k\|_2^2)}, Q_{ii} = 0\,.$$

- $\overline{\mathbf{P}} = \frac{1}{2}(\mathbf{P} + \mathbf{P}^\top), \overline{\mathbf{Q}} = \frac{1}{2}(\mathbf{Q} + \mathbf{Q}^\top)$
- SNE: find $(\tilde{\mathbf{x}}_1, \cdots, \tilde{\mathbf{x}}_n)$ that minimizes $\mathrm{KL}(\overline{\mathbf{P}}|\overline{\mathbf{Q}}) = \sum_{ij} \overline{P}_{ij} \log(\frac{\overline{P}_{ij}}{\overline{Q}_{ij}})$.
- σ_i local scaling, tuned with *entropic affinities* with fixed *perplexity*.
- t-SNE variant for the kernel $\mathbf{Q}$ (t-Student) (Van der Maaten and G. Hinton 2008).

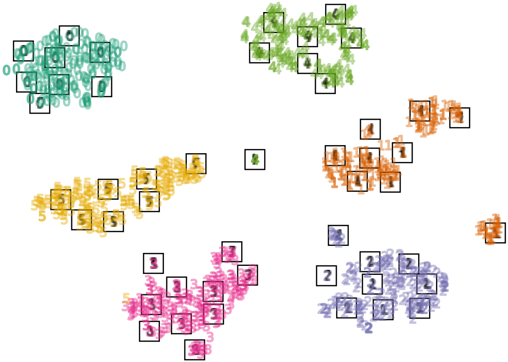
Stochastic neighbor embedding (SNE)

- With digits dataset:

A selection from the 64-dimensional digits dataset



t-SNE embeedding (time 1.324s)



Stochastic neighbor embedding (SNE)

On the perplexity parameter

- $(\sigma_i)_i$ local scalings are found so that (Vladymyrov and Carreira-Perpinan 2013):

$$\forall i \in [\![n]\!], \mathrm{H}(P_{i,:}) = -\sum_j P_{ij} \log(P_{ij}) = \log(\mathrm{perp})$$

Stochastic neighbor embedding (SNE)

On the perplexity parameter

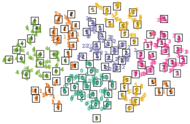
- $(\sigma_i)_i$ local scalings are found so that (Vladymyrov and Carreira-Perpinan 2013):

$$\forall i \in [\![n]\!], \mathrm{H}(P_{i,:}) = -\sum_j P_{ij} \log(P_{ij}) = \log(\text{perp})$$

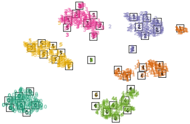
Be aware of ...

- (T)SNE has tendency to show non-existent clusters for small perplexity
- (T)SNE struggles in high-dim! In practice: PCA first.
- No geometrical relations between clusters.
- Difficult to interpret, sensitive to perplexity.
- ⚠ No mapping from the high dim space to the lower dim space.

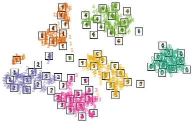
t-SNE perp=3



t-SNE perp=50



t-SNE perp=200



t-SNE perp=500

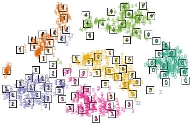
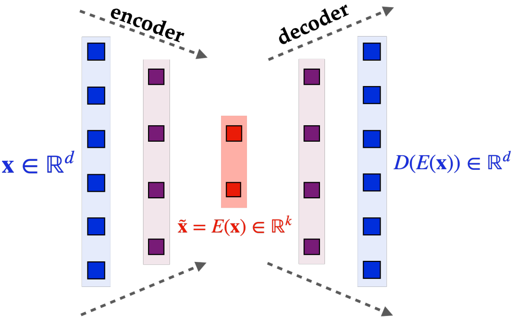


Table of contents

Autoencoders



Principle

- Send the point x from $\mathbb{R}^d$ to $\mathbb{R}^k$ with an *encoder* E.
- Map back the *code/latent variable* $E(x)$ to the original space with a *decoder* D.
- Decoded code should be close to the original point.
- Code dimension $k \ll$ original dimension d.
- Autoencoder: E and D are neural networks!

Learning autoencoders

- Architecture of E and D is fixed (number of layers, non-linearity, type of layers)
- Typical fully-connected neural networks are a combination of matrix multiplication + bias with pointwise non-linearity:

$$g_K \circ \cdots \circ g_1 \text{ where } g_k(\mathbf{x}) = \sigma(\mathbf{W}_k \mathbf{x} + \mathbf{b}_k)$$

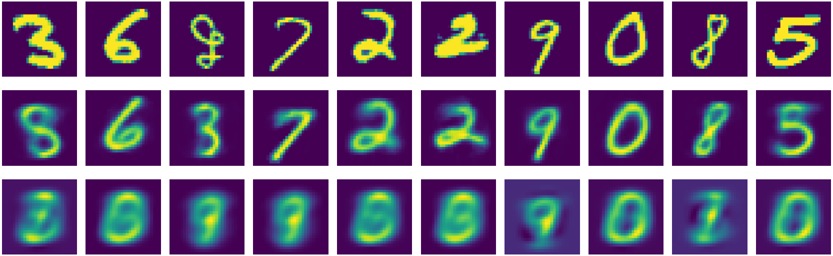
- Weights are learned by solving:

$$\min_{D,E} \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - D(E(\mathbf{x}_i))\|_2^2$$

- Optimisation is performed with first-order methods (SGD, Adam).

Autoencoder application to MNIST

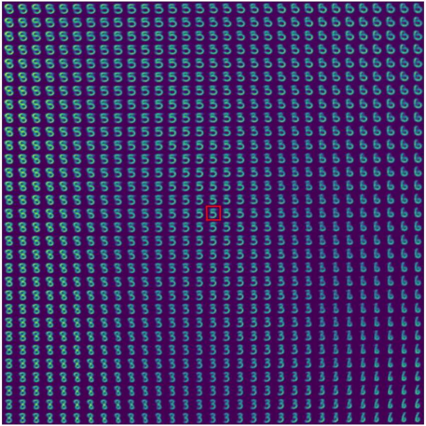
- MNIST data: 28×28 images (784 pixels)
- Compressed into $\mathbb{R}^2$ with Autoencoder or PCA



Top: original, middle: autoencoder, bottom: PCA.

Visualizing the latent space

- Pick a test image, find its code $(a, b) \in \mathbb{R}^2$.
- Plot decoder output for $(a \pm i\delta, b \pm j\delta)$.
- Continuous deformation from one digit to another.



Extensions: variational autoencoders (codes should follow a fixed law, e.g. Gaussian); different objective function (Kingma and Welling 2013)

References I

Agrawal, Akshay, Alnur Ali, and Stephen Boyd (2021). "Minimum-Distortion Embedding". In: *Foundations and Trends® in Machine Learning* 14.

Cline, Alan Kaylor and Inderjit S. Dhillon (2006). *Computation of the Singular Value Decomposition*. CRC Press.

Fan, Ky (1949). "On a Theorem of Weyl Concerning Eigenvalues of Linear Transformations I*". In: *Proceedings of the National Academy of Sciences* 35.11.

Halko, N., P. G. Martinsson, and J. A. Tropp (2011). "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". In: *SIAM Review* 53.2, pp. 217–288.

Hinton, Geoffrey E and Sam Roweis (2002). "Stochastic neighbor embedding". In: *Advances in neural information processing systems* 15.

Johnson, William and Joram Lindenstrauss (1984). "Extensions of Lipschitz maps into a Hilbert space". In: *Contemporary Mathematics* 26.

Kingma, Diederik P and Max Welling (2013). "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114*.

References II

Larsen, Kasper Green and Jelani Nelson (2017). "Optimality of the Johnson-Lindenstrauss Lemma". In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 633–638.

Mairal, Julien et al. (2009). "Online Dictionary Learning for Sparse Coding". In: *International Conference on Machine Learning*.

Pearson, Karl (1901). "On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine Series 1* 2, pp. 559–572.

Schölkopf, Bernhard, Alexander Smola, and Klaus-Robert Müller (2005). "Kernel principal component analysis". In: *Artificial Neural Networks—ICANN'97: 7th International Conference Lausanne, Switzerland, October 8–10, 1997 Proceeedings*. Springer, pp. 583–588.

Van der Maaten, Laurens and Geoffrey Hinton (2008). "Visualizing data using t-SNE.". In: *Journal of machine learning research* 9.11.

Vladymyrov, Max and Miguel Carreira-Perpinan (2013). "Entropic affinities: Properties and efficient numerical computation". In: *International conference on machine learning*. PMLR, pp. 477–485.