

Dimensionality reduction

Guillaume TOCHON & Joseph CHAZALON

LRE



Why do we care?

We have at hand n points $\mathbf{x}_1, \dots, \mathbf{x}_n$ lying in some N -dimensional space, $\mathbf{x}_i \in \mathbb{R}^N$, $\forall i = 1, \dots, n$, compactly written as a $n \times N$ matrix \mathbf{X}

→ One row of \mathbf{X} = one sample

→ One column of \mathbf{X} = a given feature value for all samples

one sample

$$\mathbf{X} = \begin{pmatrix} 1.1 & 2.2 & 3.4 & 5.6 & 1.0 \\ 6.7 & 0.5 & 0.4 & 2.6 & 1.6 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.5 & 0.0 & 4.3 & 8.3 & 3.4 \\ 0.5 & 3.5 & 8.1 & 3.6 & 4.6 \\ 5.1 & 9.7 & 3.5 & 7.9 & 5.1 \\ 3.7 & 7.8 & 2.6 & 3.2 & 6.3 \end{pmatrix}$$

one feature

Why do we care?

We have at hand n points $\mathbf{x}_1, \dots, \mathbf{x}_n$ lying in some N -dimensional space, $\mathbf{x}_i \in \mathbb{R}^N$, $\forall i = 1, \dots, n$, compactly written as a $n \times N$ matrix \mathbf{X}

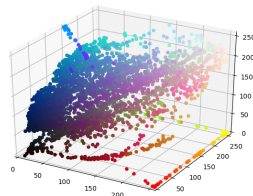
→ One row of \mathbf{X} = one sample

→ One column of \mathbf{X} = a given feature value for all samples

one sample

1.1	2.2	3.4	5.6	1.0
6.7	0.5	0.4	2.6	1.6
2.4	9.3	7.3	6.4	2.8
1.5	0.0	4.3	8.3	3.4
0.5	3.5	8.1	3.6	4.6
5.1	9.7	3.5	7.9	5.1
3.7	7.8	2.6	3.2	6.3

one feature



$N = 3$

Why do we care?

Example of real high-dimensional data

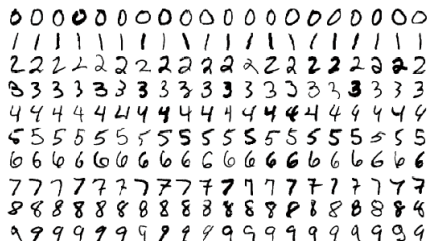
Real world data is very often high-dimensional

Why do we care?

Example of real high-dimensional data

Real world data is very often high-dimensional

MNIST image classification:



0
1
2
3
4
5
6
7
8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

Sample \mathbf{x} : image with 28x28 pixels

Data set: 60000 samples

Dimensionality: $\mathbf{x} \in \mathbb{R}^{28 \times 28 = 784}$

Why do we care?

Example of real high-dimensional data

Real world data is very often high-dimensional

MNIST image classification:

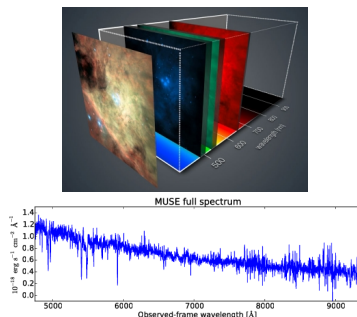


Sample \mathbf{x} : image with 28×28 pixels

Data set: 60000 samples

Dimensionality: $\mathbf{x} \in \mathbb{R}^{28 \times 28 = 784}$

MUSE hyperspectral image analysis:



Sample \mathbf{x} : pixel with 3600 spectral bands

Data set: image with 300×300 pixels

Dimensionality: $\mathbf{x} \in \mathbb{R}^{3600}$

Why do we care?

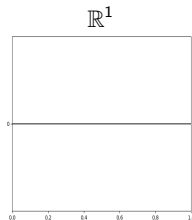
The curse of dimensionality

High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)

Why do we care?

The curse of dimensionality

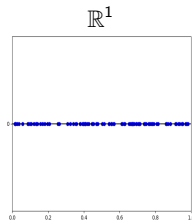
High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)



Why do we care?

The curse of dimensionality

High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)

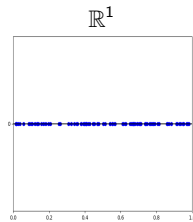


100 points uniformly
distributed in $[0:1]$

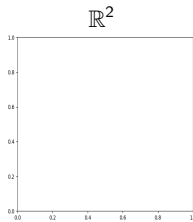
Why do we care?

The curse of dimensionality

High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)



100 points uniformly
distributed in $[0:1]$

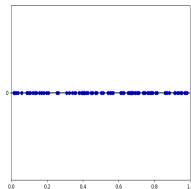


Why do we care?

The curse of dimensionality

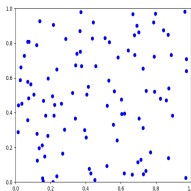
High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)

\mathbb{R}^1



100 points uniformly distributed in $[0:1]$

\mathbb{R}^2



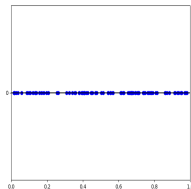
100 points uniformly distributed in $[0:1]^2$

Why do we care?

The curse of dimensionality

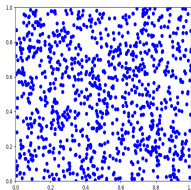
High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)

\mathbb{R}^1



100 points uniformly
distributed in $[0:1]$

\mathbb{R}^2



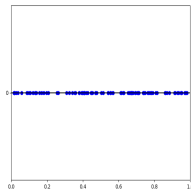
1000 points uniformly
distributed in $[0:1]^2$

Why do we care?

The curse of dimensionality

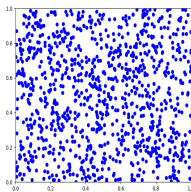
High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)

\mathbb{R}^1



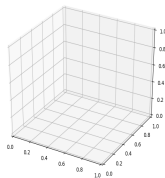
100 points uniformly distributed in $[0:1]$

\mathbb{R}^2



1000 points uniformly distributed in $[0:1]^2$

\mathbb{R}^3

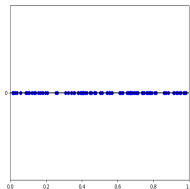


Why do we care?

The curse of dimensionality

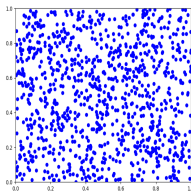
High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)

\mathbb{R}^1



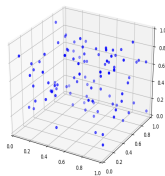
100 points uniformly distributed in $[0:1]$

\mathbb{R}^2



1000 points uniformly distributed in $[0:1]^2$

\mathbb{R}^3



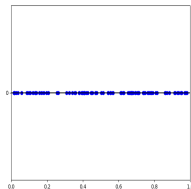
100 points uniformly distributed in $[0:1]^3$

Why do we care?

The curse of dimensionality

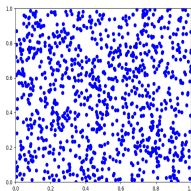
High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)

\mathbb{R}^1



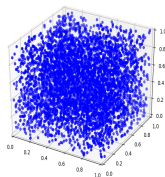
100 points uniformly distributed in $[0:1]$

\mathbb{R}^2



1000 points uniformly distributed in $[0:1]^2$

\mathbb{R}^3



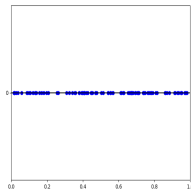
10000 points uniformly distributed in $[0:1]^3$

Why do we care?

The curse of dimensionality

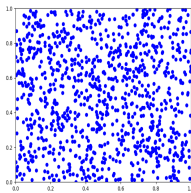
High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)

\mathbb{R}^1



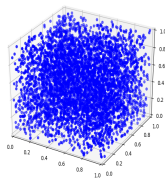
100 points uniformly distributed in $[0:1]$

\mathbb{R}^2



1000 points uniformly distributed in $[0:1]^2$

\mathbb{R}^3



10000 points uniformly distributed in $[0:1]^3$

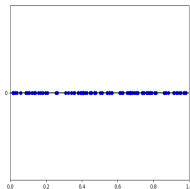
$$\frac{\mathcal{V}(\mathbb{S}^n)}{\mathcal{V}([-1:1]^n)} = \frac{\pi^{n/2}}{2^n \Gamma(\frac{n}{2} + 1)} \xrightarrow{n \rightarrow +\infty} 0$$

Why do we care?

The curse of dimensionality

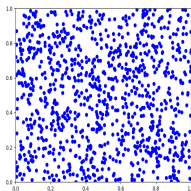
High-dimensional spaces ~~suck donkey balls~~ suffer from the *curse of dimensionality* (also called Hughes' phenomenon)

\mathbb{R}^1



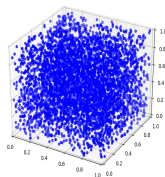
100 points uniformly distributed in $[0:1]$

\mathbb{R}^2



1000 points uniformly distributed in $[0:1]^2$

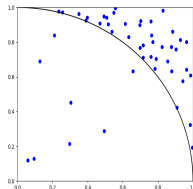
\mathbb{R}^3



10000 points uniformly distributed in $[0:1]^3$

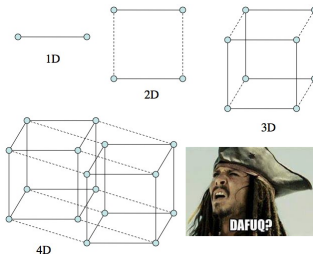
$$\frac{\mathcal{V}(\mathbb{S}^n)}{\mathcal{V}([-1:1]^n)} = \frac{\pi^{n/2}}{2^n \Gamma(\frac{n}{2} + 1)} \xrightarrow{n \rightarrow +\infty} 0$$

Points uniformly distributed in a n -cube of side 2 mostly fall outside of the unit sphere!



Why is it tricky?

→ We naturally cannot picture anything that is more than 3D in our mind



- Picturing something 3D in a 2D flat screen can already be misleading
- Real data naturally lives in (complex) high-dimensional space
- Real data is often strongly correlated

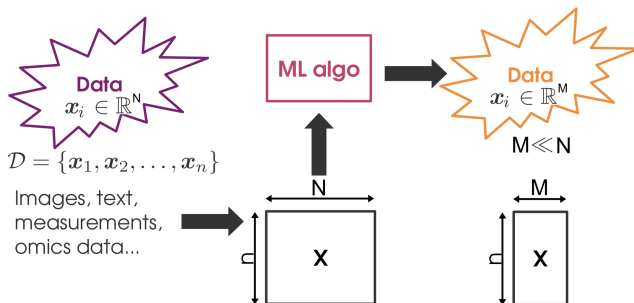
And somehow, we want to have a good look to our data before feeding it to some machine learning algorithm.

⇒ can I use the inherent structure of my data to pimp my machine learning performances?

How?

Dimensionality reduction: transform data set \mathbf{X} with dimensionality N into a new data set \mathbf{Y} ($n \times M$ matrix) with dimensionality $M < N$ (hopefully $M \ll N$) such that **as few information as possible is lost** in the process.

\mathbf{y}_i (i th row of \mathbf{Y}) is the low-dimensional counterpart (the *projection*) of \mathbf{x}_i .



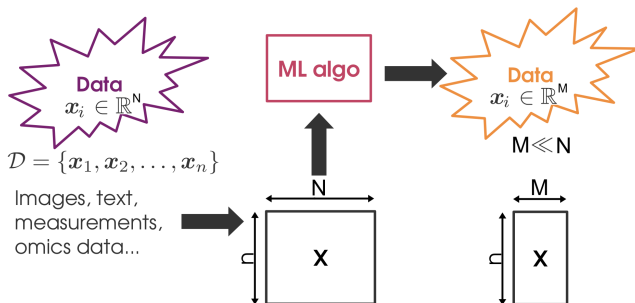
For a good read (from which the majority of these classnotes are extracted): Van Der Maaten, L., Postma, E., & Van den Herik, J. (2009). Dimensionality reduction: a comparative review. *Journal of Machine Learning Research*.

How?

Dimensionality reduction: transform data set \mathbf{X} with dimensionality N into a new data set \mathbf{Y} ($n \times M$ matrix) with dimensionality $M < N$ (hopefully $M \ll N$) such that **as few information as possible is lost** in the process.

y_i (i th row of \mathbf{Y}) is the low-dimensional counterpart (the *projection*) of x_i .

Information???

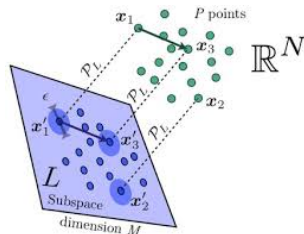


For a good read (from which the majority of these classnotes are extracted): Van Der Maaten, L., Postma, E., & Van den Herik, J. (2009). Dimensionality reduction: a comparative review. *Journal of Machine Learning Research*.

Linear approaches

Somehow trying to find a low-dimensional subspace in which the projected data would not be too much distorted after projection.

- Johnson-Lindenstrauss lemma
- Classical scaling
- (The one and only) Principal Component Analysis
- And much more...



Johnson-Lindenstrauss lemma

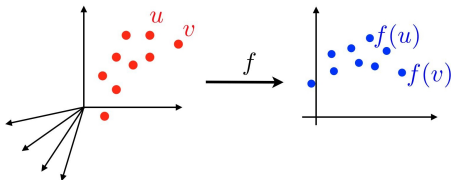
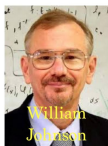
It's not because you *can* that you *will*

Let $0 < \varepsilon < 1$ and let x_1, x_2, \dots, x_n be n points in \mathbb{R}^N . Then there exists a linear map $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$ such that for every two points x_i and x_j ,

$$(1 - \varepsilon) \|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1 + \varepsilon) \|x_i - x_j\|^2$$

with $M = 4 \log n / (\varepsilon^2/2 - \varepsilon^3/3)$.

Johnson, W. B., & Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*.



f can be defined as a random projection $f(x) = \frac{1}{\sqrt{k}}Ax$ with A a $M \times N$ matrix whose entries are sampled iid from a Gaussian law $\mathcal{N}(0, 1)$ or a Uniform law $\mathcal{U}(\{-1, 1\})$

See for instance Dasgupta, S., & Gupta, A. (1999). An elementary proof of the Johnson-Lindenstrauss lemma. *International Computer Science Institute, Technical Report*.

Johnson-Lindenstrauss lemma

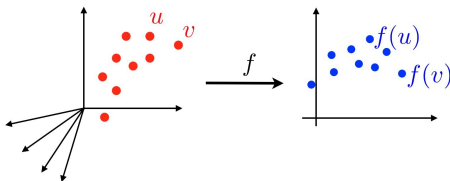
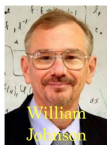
It's not because you *can* that you *will*

Let $0 < \varepsilon < 1$ and let x_1, x_2, \dots, x_n be n points in \mathbb{R}^N . Then there exists a linear map $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$ such that for every two points x_i and x_j ,

$$(1 - \varepsilon) \|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1 + \varepsilon) \|x_i - x_j\|^2$$

with $M = 4 \log n / (\varepsilon^2/2 - \varepsilon^3/3)$.

Johnson, W. B., & Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*.



f can be defined as a random projection $f(x) = \frac{1}{\sqrt{k}}Ax$ with A a $M \times N$ matrix whose entries are sampled iid from a Gaussian law $\mathcal{N}(0, 1)$ or a Uniform law $\mathcal{U}(\{-1, 1\})$

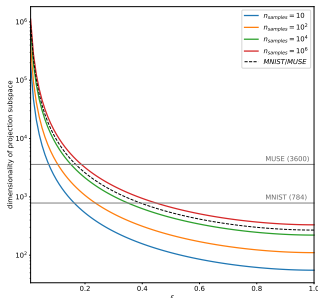
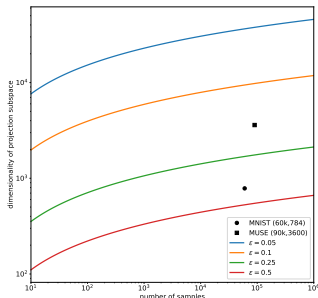
See for instance Dasgupta, S., & Gupta, A. (1999). An elementary proof of the Johnson-Lindenstrauss lemma. *International Computer Science Institute, Technical Report*.

So, where is the poop? 🐛

Johnson-Lindenstrauss lemma

Here is the 🤖

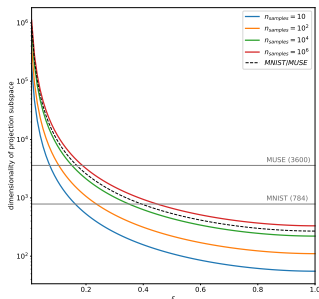
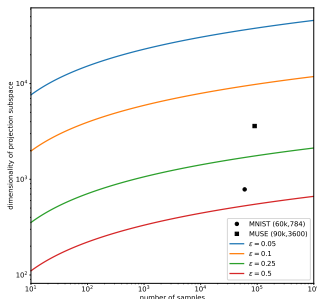
The dimension of the projection space $M = 4 \log n / (\varepsilon^2/2 - \varepsilon^3/3)$ depends only on the number of input samples n and the distortion factor ε and not on the dimension of the input space N .



Johnson-Lindenstrauss lemma

Here is the 🤖

The dimension of the projection space $M = 4 \log n / (\varepsilon^2/2 - \varepsilon^3/3)$ depends only on the number of input samples n and the distortion factor ε and not on the dimension of the input space N .



Most of the time, the JL lemma is not a good deal:

- $M > N$ for a fixed (low) $\varepsilon \Rightarrow$ dimensionality augmentation
- $M < N$ for a high $\varepsilon \Rightarrow$ distortion \equiv loss of information

Classical scaling

Also called Principal Coordinates Analysis (PCoA)

Lots of formula here, but you just need to retain the overall idea

PCoA: project data points \mathbf{X} onto \mathbf{Y} with a linear mapping \mathbf{M} such that $\mathbf{Y} = \mathbf{XM}$ such that all pairwise distances between points do not change too much before/after projection

Classical scaling

Also called Principal Coordinates Analysis (PCoA)

Lots of formula here, but you just need to retain the overall idea

PCoA: project data points \mathbf{X} onto \mathbf{Y} with a linear mapping \mathbf{M} such that $\mathbf{Y} = \mathbf{XM}$ such that all pairwise distances between points do not change too much before/after projection

If \mathbf{D} is the $n \times n$ Euclidean distance matrix with entries $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ and $\mathbf{D}^{(2)} = [d_{ij}^2]$, PCoA seeks the linear mapping \mathbf{M} that minimizes

$$\phi(\mathbf{Y}) = \sum_{i,j} (d_{ij}^2 - \|\mathbf{y}_i - \mathbf{y}_j\|^2)$$

with $\mathbf{y}_i = \mathbf{x}_i \mathbf{M}$ and $\|\mathbf{m}_i\|^2 = 1 \ \forall i$

Classical scaling

Also called Principal Coordinates Analysis (PCoA)

Lots of formula here, but you just need to retain the overall idea

PCoA: project data points \mathbf{X} onto \mathbf{Y} with a linear mapping \mathbf{M} such that $\mathbf{Y} = \mathbf{X}\mathbf{M}$ such that all pairwise distances between points do not change too much before/after projection

If \mathbf{D} is the $n \times n$ Euclidean distance matrix with entries $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ and $\mathbf{D}^{(2)} = [d_{ij}^2]$, PCoA seeks the linear mapping \mathbf{M} that minimizes

$$\phi(\mathbf{Y}) = \sum_{i,j} (d_{ij}^2 - \|\mathbf{y}_i - \mathbf{y}_j\|^2)$$

with $\mathbf{y}_i = \mathbf{x}_i\mathbf{M}$ and $\|\mathbf{m}_i\|^2 = 1 \ \forall i$

Solution: eigendecomposition of the Gram matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^T = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^T$

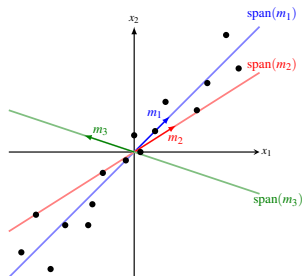
\mathbf{K} can be obtained by double centering $\mathbf{D}^{(2)}$: $\mathbf{K} = -\frac{1}{2}\mathbf{C}_n\mathbf{D}^{(2)}\mathbf{C}_n$ with centering matrix $\mathbf{C}_n = \mathbf{I}_n - \frac{1}{n}\mathbf{ones}(n, n)$

Optimal projection onto the first M dimensions: $\mathbf{Y} = \mathbf{\Lambda}_M^{1/2}\mathbf{E}_M^T$ with \mathbf{E}_M matrix of the M largest eigenvectors of \mathbf{E} .

Principal component analysis

Also known as the Karhunen-Loève transform

Closely related to PCoA, but operates on the covariance matrix $\frac{1}{n-1} \mathbf{X}_c^T \mathbf{X}_c$
PCA seeks the main spreading directions in the data point cloud \mathbf{X}

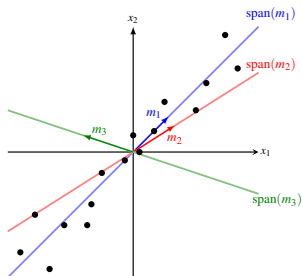


Which axis better represents the main spreading direction of the data point cloud?

Principal component analysis

Also known as the Karhunen-Loève transform

Closely related to PCoA, but operates on the covariance matrix $\frac{1}{n-1} \mathbf{X}_c^T \mathbf{X}_c$
PCA seeks the main spreading directions in the data point cloud \mathbf{X}



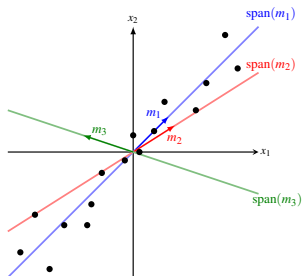
Which axis better represents the main spreading direction of the data point cloud?

→ $\text{span}(m_1)$ (easy...)

Principal component analysis

Also known as the Karhunen-Loève transform

Closely related to PCoA, but operates on the covariance matrix $\frac{1}{n-1} \mathbf{X}_c^T \mathbf{X}_c$
PCA seeks the main spreading directions in the data point cloud \mathbf{X}



Which axis better represents the main spreading direction of the data point cloud?

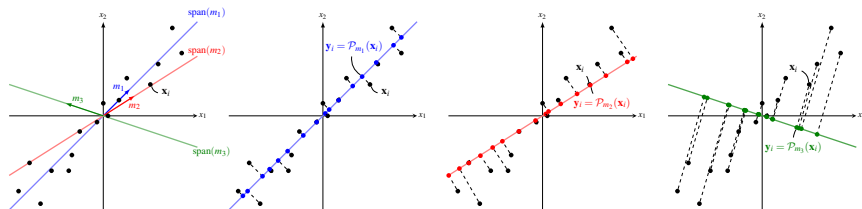
→ $\text{span}(m_1)$ (easy...)

Which mathematical criterion could we define to automatically derive m_1 ?

Principal component analysis

Also known as the Karhunen-Loève transform

Closely related to PCoA, but operates on the covariance matrix $\frac{1}{n-1} \mathbf{X}_c^T \mathbf{X}_c$
PCA seeks the main spreading directions in the data point cloud \mathbf{X}



Define $\mathbf{y}_i = \mathcal{P}_m(\mathbf{x}_i)$ the projection of data point \mathbf{x}_i on the axe spanned by m .

→ The best axis minimizes the sum of projection errors $\sum_{i=1}^n \|\mathbf{x}_i - \mathbf{y}_i\|^2$

→ Equivalent to maximizes the dispersion (*the variance*) after projection

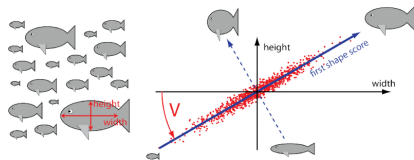
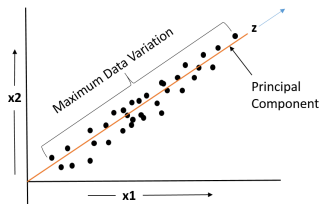
⇒ $\mathbf{u}^* = \arg \min_{\|\mathbf{u}\|=1} \sum_{i=1}^n \|\mathbf{x}_i - \mathcal{P}_m(\mathbf{x}_i)\|^2 = \arg \max_{\|\mathbf{u}\|=1} \text{Var}(\mathbf{Y})$

Principal component analysis

Also known as the Karhunen-Loève transform

Closely related to PCoA, but operates on the covariance matrix $\frac{1}{n-1} \mathbf{X}_c^T \mathbf{X}_c$. PCA seeks the linear mapping \mathbf{M} that maximizes the projection variance $\text{tr}(\mathbf{M}^T \text{cov}(\mathbf{X}) \mathbf{M})$ with $\|\mathbf{m}_i\|^2 = 1 \ \forall i$.

1. Center the data $\mathbf{X}_c = \mathbf{C}_n \mathbf{X}$
- 1.b (opt) Reduce the data
2. Compute covariance matrix $\Sigma = \frac{1}{n-1} \mathbf{X}_c^T \mathbf{X}_c$
3. Perform eigendecomposition $(\mathbf{E}, \mathbf{\Lambda})$ of Σ
4. Project on the first M principal axes $\mathbf{Y} = \mathbf{X} \mathbf{E}_M$

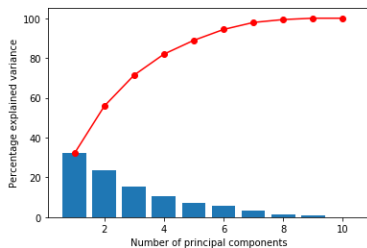
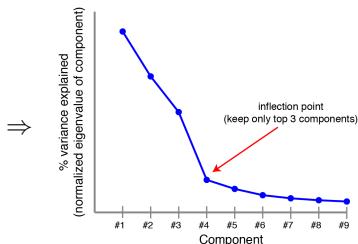


Data after projection is uncorrelated, but has lost some interpretability

Major challenges related to PCA

PCA is probably the most popular and used unsupervised linear dimensionality reduction technique, but it comes with a bunch of operability questions, the 2 principles being:

1. How to automatically select the right number of dimensions to project?

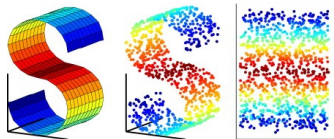


2. How to project a new data point on a learned projection subspace?

⇒ See you in lab session for the answer

Non-linear approaches

When it is assumed that the data does not live in an Euclidean subspace (why would it anyway?), some more advanced techniques must be relied on.



- Isomap
- Locally linear embedding
- Kernel Principal Component Analysis (aka PCA on steroids)
- Multilayer autoencoders
- And much more...

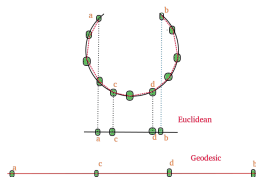


Manifold unfolding

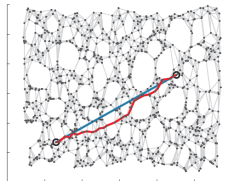
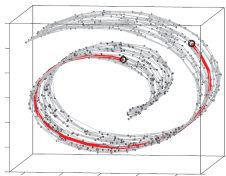
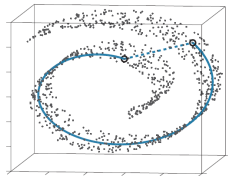
Isomap

Geodesic distance rocks

Isometric feature mapping: same idea as classical scaling, but using geodesic distance instead of Euclidean distance.



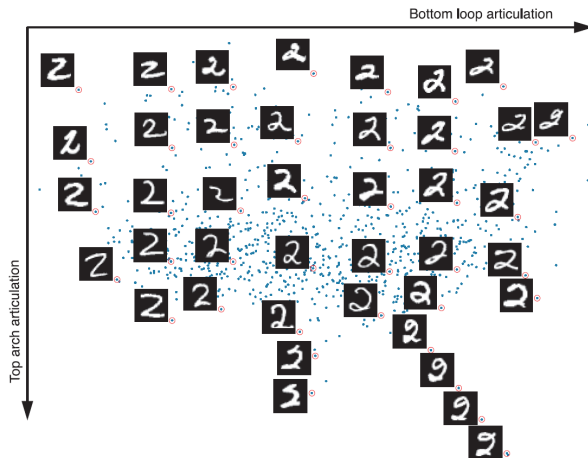
1. Compute k-nearest neighbor graph of data $\mathbf{x}_1, \dots, \mathbf{x}_n$
2. Compute all pairwise geodesic distances
3. Apply classical scaling



Isomap

An illustrative example

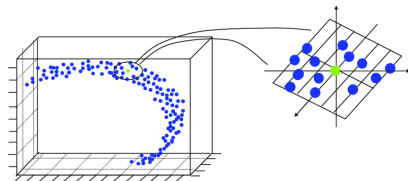
Isomap applied to some images of the digit 2 in MNIST data



See Tenenbaum, J. B., De Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*. for more details.

Locally linear embedding

Locally linear embedding: the manifold can be locally considered Euclidean

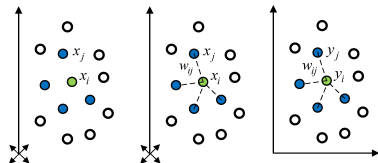


For each point \mathbf{x}_i :

1. get its k -nearest neighbors \mathbf{x}_j , $j=1,\dots,k$
2. Get weights w_{ij} that best linearly reconstruct \mathbf{x}_i with \mathbf{x}_j :

$$\text{minimize } \sum_{i=1}^n \|\mathbf{x}_i - \sum_j w_{ij} \mathbf{x}_j\|^2$$

with constraint $\sum_j w_{ij} = 1$ (closed-form solution)



3. Low-dimensional embedding \rightarrow reconstruct \mathbf{y}_i with \mathbf{y}_j and same weights w_{ij} :

$$\text{minimize } \sum_{i=1}^n \|\mathbf{y}_i - \sum_j w_{ij} \mathbf{y}_j\|^2$$

with constraints $\frac{1}{n} \sum_i \mathbf{y}_i \mathbf{y}_i^T = I$ and $\sum_i \mathbf{y}_i = \mathbf{0}$ (eigendecomposition of a Gram matrix).

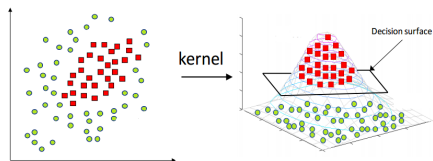
For a good read, see for instance Ghoggh, B., Ghodsi, A., Karray, F., & Crowley, M. (2020). Locally Linear Embedding and its Variants: Tutorial and Survey. *arXiv preprint arXiv:2011.10925*.

The kernel trick

When one actually wants to increase the dimension

Base idea: map n non linearly separable points to a (possibly infinite) space where they would be with a function φ

- How should we define φ ?
- Do we really want to compute stuff in a (possibly infinite) feature space?



Mercer theorem: we do not need to know the mapping φ explicitly as long as we have a positive semi-definite kernel/Gram matrix $\mathbf{K} = [\kappa(\mathbf{x}_i, \mathbf{x}_j)] = [\langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle]$

Widely used kernel functions:

Polynomial kernel: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^d$

Gaussian RBF kernel: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$

Sigmoid kernel: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{b} \mathbf{x}_i^T \mathbf{x}_j + c)$

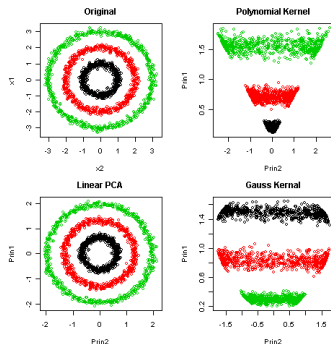
A (challenging) must-read on this topic: Schölkopf, B., Smola, A. J., & Bach, F. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

Kernel PCA

PCA on steroids

The maths behind are quite hard, but the following scikit-learn recipe works fine:

1. Compute kernel matrix $\mathbf{K} = [\kappa(\mathbf{x}_i, \mathbf{x}_j)] = [\langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle]$ and double-center it $\mathbf{K}_c = \mathbf{C}_n \mathbf{K} \mathbf{C}_n$
2. Eigendecomposition of \mathbf{K}_c is strongly related to this of the (intractable) covariance matrix in the feature space \rightarrow get eigenvectors \mathbf{V} and corresponding eigenvalues $\mathbf{\Lambda}$ of \mathbf{K}_c .
3. Keep the first M columns of $\sqrt{\mathbf{\Lambda}} \mathbf{V}$ to get the coordinates of projected data points in the low M -dimensional space.



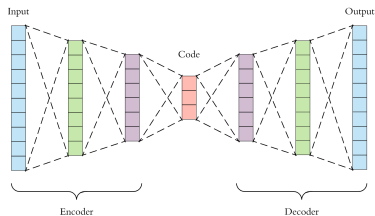
But things get nasty when one wants to project a new data point \mathbf{x} that was not known when constructing the kernel...

Non-linear PCA

Also known as autoencoder

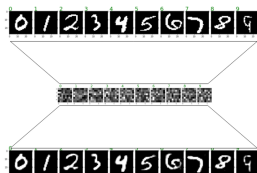
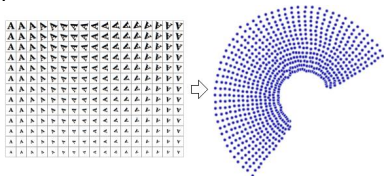
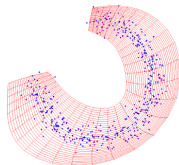
Overall idea:

1. train an autoencoder (neural network with an autoassociative architecture) to perform an identity mapping.
2. use the output of the bottleneck layer as low-dimensional code.



Bottleneck code is a non-linear combination of entries (thanks to activation functions on the encoder layers) → learned mapping is a non-linear PCA.

Principal components are generalized from straight lines to curves: the projection subspace which is described by all nonlinear components is also curved.



Let's recap

High-dimensional data set \mathbf{X} is a $n \times N$ matrix, with n = number of samples and N = dimensionality of underlying space.

Technique	Parametric	Parameters	Computational	Memory
PCoA	nope	None	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
PCA	yay	None	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$
Isomap	nope	k	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
LLE	nope	k	$\mathcal{O}(pn^2)$	$\mathcal{O}(pn^2)$
K-PCA	nope	$\kappa(\cdot, \cdot)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
NL-PCA	yay	net size	$\mathcal{O}(inw)$	$\mathcal{O}(w)$

Parametric \equiv explicit embedding from high-dimensional space to low-dimensional one

For LLE: p is the ratio of non-zero elements in a sparse matrix to the total number of elements

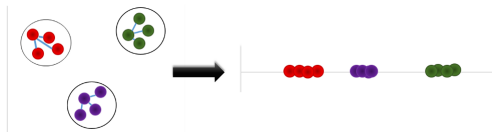
For NL-PCA: i is the number of iterations and w is the number of weights in the neural network

Extracted from Van Der Maaten, L., Postma, E., & Van den Herik, J. (2009). Dimensionality reduction: a comparative review. *Journal of Machine Learning Research*.

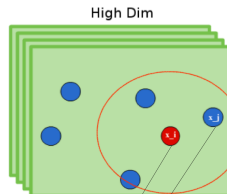
t-Distributed Stochastic Neighbor Embedding

t-SNE is a popular method to see in 2D or 3D wtf is going on in a high-dimensional spaces.

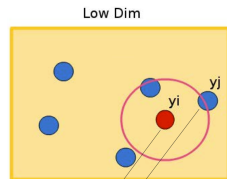
1. Construct a probability distribution p over pairs of points in the high-dim space: the more similar (the closer) the two points, the higher the probability.
2. Define a second probability distribution q over the points in the low-dim space, and dispatch the points such that the distance between p and q is minimized (for the Kullback-Leibler divergence)



- t-SNE is excellent in visualizing the well-separated clusters, but fails to preserve the global geometry of the data.
- t-SNE depends on a *perplexity* parameter, which reflects the scale of search for close points.



$$p_{ji} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)}$$



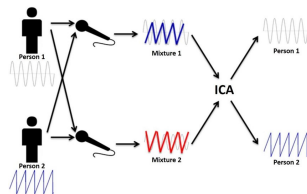
$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - y_l\|^2)^{-1}}$$

For more details, see Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*.
Or go play with <https://distill.pub/2016/misread-tsne/>

Independent component analysis

Yet another kind of principal component analysis

ICA aims to provide a solution to the so-called *cocktail party*: retrieving independent sources that got mixed-up together with unknown scaling coefficients.

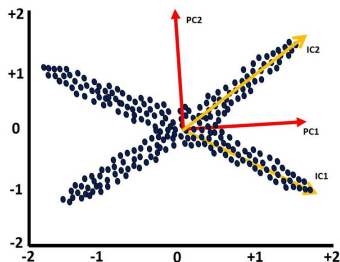
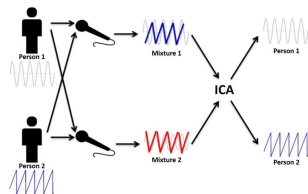


Goal: estimate source **s** and mixing matrix **A** from observation $\mathbf{x} = \mathbf{A}\mathbf{s}$.

Independent component analysis

Yet another kind of principal component analysis

ICA aims to provide a solution to the so-called *cocktail party*: retrieving independent sources that got mixed-up together with unknown scaling coefficients.



Goal: estimate source **s** and mixing matrix **A** from observation $\mathbf{x} = \mathbf{A}\mathbf{s}$.

- Ill-posed \Rightarrow enforce independence on source components
- Work on higher order statistics (PCA limits to order-2 statistics)
- Unknown source must **not** be Gaussian-distributed

Contrarily to PCA vectors, ICA vectors are not orthogonal and not ranked by importance, but they are mutually independent.

For a (long) tutorial on ICA, see Stone, J. V. (2004). Independent component analysis. *A Bradford Book*.