Machine Learning

3. Dimensionality reduction

Yannick Le Cacheux

CentraleSupélec - Université Paris Saclay

September 2024

Dimensionality reduction

Definition

Dimensionality reduction is the process of reducing the number of features in a dataset while retaining as much of the important information as possible.

- It can be useful for:
  - Data compression
  - Data visualization
  - Noise reduction
  - As a preprocessing step for other algorithms
  - ...

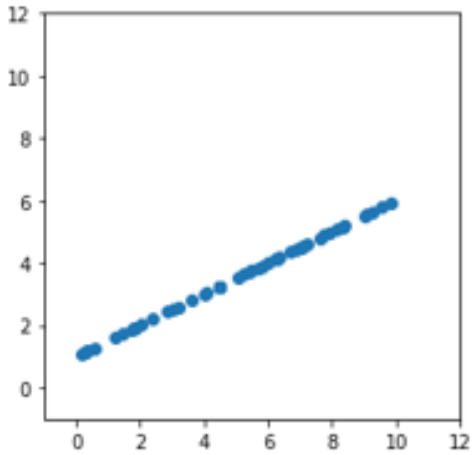
Application: 3D shape estimation

Figure: Faces sampled in 3 dimensions.

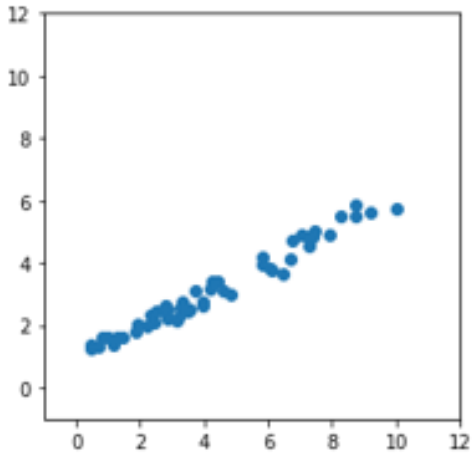
Figure: A weird face, with the corresponding 3D shape estimation.

Dimensions and hyperplanes

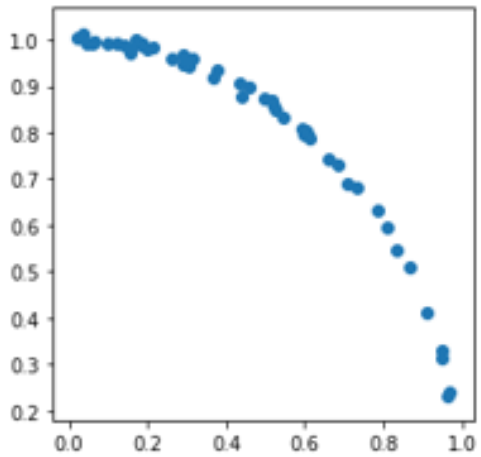
- All these points have two dimensions, but...



(a) These points are located on a hyperplane (a line)



(b) These points are *almost* located on a hyperplane



(c) These points are located on a curved line

- Knowing the orientation of the line, a single coordinate would suffice to accurately represent these points.

Outline

Principal Component Analysis (PCA)

- If we had to represent these points by their coordinates along a *single* direction: which one would we choose?

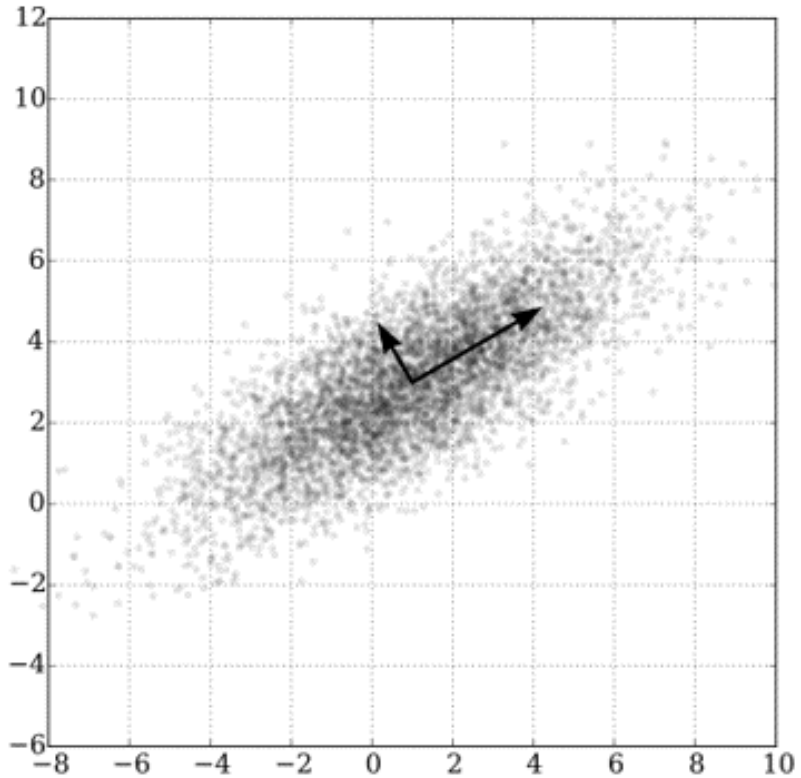


Figure: Points with some dispersion

Principal Component Analysis (PCA)

- If we had to represent these points by their coordinates along a *single* direction: which one would we choose?
- Intuition: we want to represent points by their coordinates on new (orthogonal) axes
- Such that variance (*i.e.* dispersion) along these axes is maximized
- We call such axes *principal components*

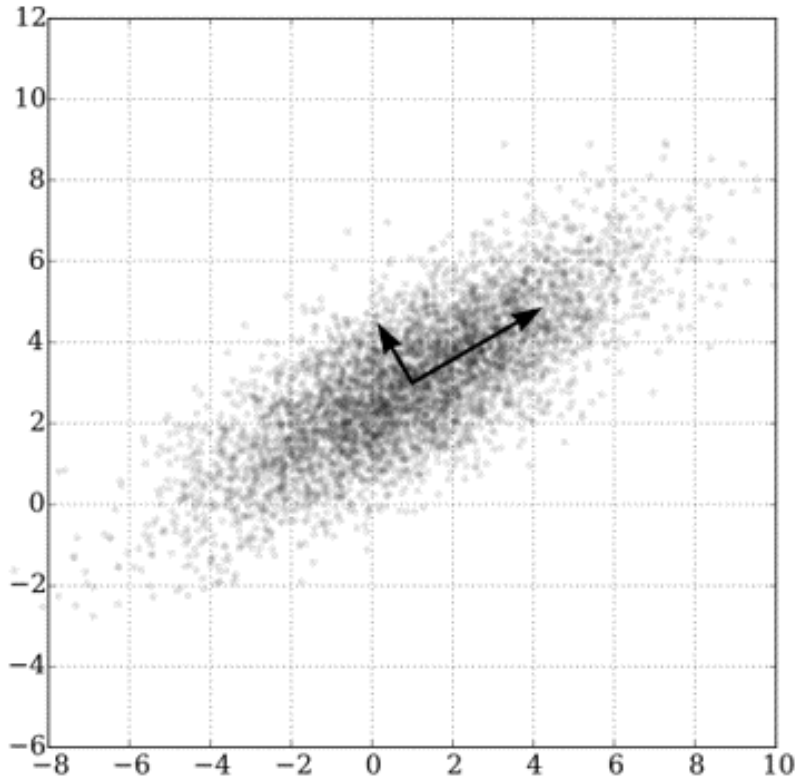


Figure: First and second principal components: axes maximizing variance

Formalization

- Given N points $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ $\quad \mathbf{x}_n \in \mathbb{R}^D \quad \mathbf{X} \in \mathbb{R}^{N \times D}$

  with mean $\boldsymbol{\mu} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$

- We want to find a vector $\mathbf{w}$ such that the variance along direction $\mathbf{w}$ is maximized:

$$\underset{\mathbf{w}}{\text{maximize}} \quad \frac{1}{N} \sum_{n=1}^{N} (\mathbf{w}^\top \mathbf{x}_n - \mathbf{w}^\top \boldsymbol{\mu})^2 \tag{1}$$

- Define $\mathbf{S} \in \mathbb{R}^{D \times D}$ the covariance matrix:

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{x}_n - \boldsymbol{\mu})(\mathbf{x}_n - \boldsymbol{\mu})^T$$

- Equation (1) can be rewritten as:

$$\mathbf{w}^\top \mathbf{S} \mathbf{w}$$

Resolution

(Temporary) objective

$$\underset{\mathbf{w}}{\text{maximize}} \quad \mathbf{w}^\top \mathbf{S} \mathbf{w}$$

[1] Cf. relevant section on constrained optimization with Lagrange multipliers in appendix.

Resolution

(Temporary) objective

$$\underset{\mathbf{w}}{\text{maximize}} \quad \mathbf{w}^\top \mathbf{S} \mathbf{w}$$

- We want to find a vector $\mathbf{w}$ with unit norm: $\|\mathbf{w}\|_2 = 1$ (why?)
- We therefore add the constraint that $\|\mathbf{w}\|_2 = 1$ (or equivalently $\|\mathbf{w}\|_2 \leq 1$)
- This can be done with a Lagrange multiplier [1] λ to enforce $1 - \mathbf{w}^\top \mathbf{w} = 0$ (equivalent to $\|\mathbf{w}\|_2^2 = 1$)

- We can thus define a loss function J:

$$J_\lambda(\mathbf{w}) = -\mathbf{w}^\top \mathbf{S} \mathbf{w} - \lambda(1 - \mathbf{w}^\top \mathbf{w})$$

[1] Cf. relevant section on constrained optimization with Lagrange multipliers in appendix.

Resolution

Objective

$$\underset{\mathbf{w}}{\text{minimize}} \quad -\mathbf{w}^\top \mathbf{S}\mathbf{w} - \lambda(1 - \mathbf{w}^\top \mathbf{w})$$

- We can calculate[2]
$$\frac{\partial J_\lambda(\mathbf{w})}{\partial \mathbf{w}} = 2\mathbf{S}\mathbf{w} - 2\lambda\mathbf{w}$$
- Thus
$$\frac{\partial J_\lambda(\mathbf{w})}{\partial \mathbf{w}} = 0 \quad \text{if} \quad \mathbf{S}\mathbf{w} = \lambda\mathbf{w}$$
- Which means:

Solution

$\mathbf{w}$ is an *eigenvector* of the covariance matrix $\mathbf{S}$, and λ is an *eigenvalue*.

[2] Using the identity $\frac{\partial \mathbf{w}^\top \mathbf{S}\mathbf{w}}{\partial \mathbf{w}} = (\mathbf{S}^\top + \mathbf{S})\mathbf{w}$ (homework 1) and the fact that $\mathbf{S}$ is symetrical.

Estimating the principal components

To obtain principal components from observations $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$:

- Compute mean vector $\boldsymbol{\mu}$:

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$$

- Center points by subracting $\boldsymbol{\mu}$:

$$\mathbf{x}_n^* = \mathbf{x}_n - \boldsymbol{\mu} \quad \forall n \qquad \mathbf{X}^* = (\mathbf{x}_1^*, \ldots, \mathbf{x}_N^*)$$

- Compute covariance matrix $\mathbf{S} = \frac{1}{N} \mathbf{X}^{*\top} \mathbf{X}^*$

Principal components

- The principal components $\mathbf{p}_1, \ldots, \mathbf{p}_D$ are the eigenvectors of the covariance matrix $\mathbf{S}$
- The variances $\lambda_1, \ldots, \lambda_D$ along these axes are the eigenvalues of $\mathbf{S}$

Illustration of principal components

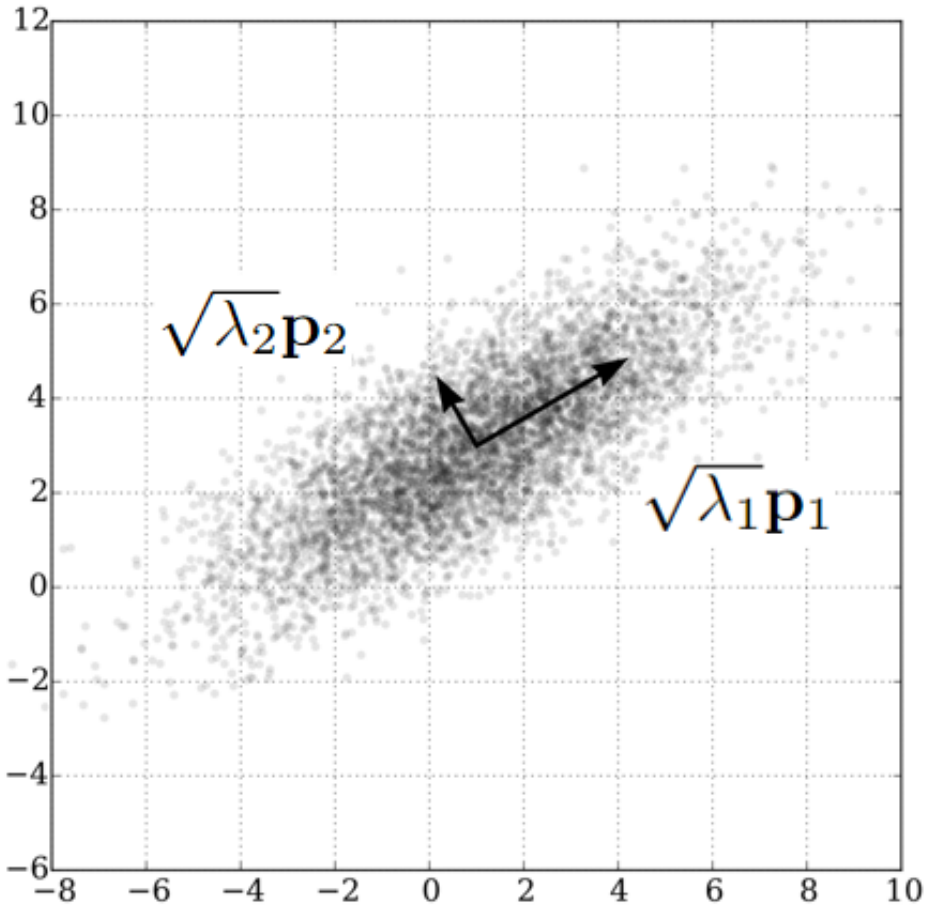


Figure: First and second principal components $\mathbf{p}_1$ and $\mathbf{p}_2$, scaled by standard deviations along these axes $\sqrt{\lambda_1}$ and $\sqrt{\lambda_2}$.

Projection in lower dimension

- For a point $\mathbf{x} = (x_1, \ldots, x_D)$, the first coordinate z_1 in the new, lower dimension basis can be obtained by centering and projecting it onto the first principal component $\mathbf{p}_1$:

$$z_1 = (\mathbf{x} - \boldsymbol{\mu})^\top \mathbf{p}_1$$

- We may also want to ensure that z_1 has a standard deviation of 1 by dividing by the square root of the corresponding variance λ_1:

$$z_1 = \frac{(\mathbf{x} - \boldsymbol{\mu})^\top \mathbf{p}_1}{\sqrt{\lambda_1}}$$

- Similarly, we can compute $z_2 = \frac{(\mathbf{x}-\boldsymbol{\mu})^\top \mathbf{p}_3}{\sqrt{\lambda_2}}$, $z_3 = \frac{(\mathbf{x}-\boldsymbol{\mu})^\top \mathbf{p}_3}{\sqrt{\lambda_3}}$, etc

Projection in lower dimension

- In general, if we want to obtain a P-dimensional representation $\mathbf{z} \in \mathbb{R}^P$ of a point $\mathbf{x} \in \mathbb{R}^D$, we compute

$$\mathbf{z} = (z_1, \ldots, z_P) = \begin{pmatrix} \mathbf{p}_1^\top (\mathbf{x} - \boldsymbol{\mu}) / \sqrt{\lambda_1} \\ \vdots \\ \mathbf{p}_P^\top (\mathbf{x} - \boldsymbol{\mu}) / \sqrt{\lambda_P} \end{pmatrix}$$

- In matrix notations, with $\mathbf{P} = (\mathbf{p}_1, \ldots, \mathbf{p}_P)^\top \in \mathbb{R}^{P \times D}$ the first P principal components:[3]

$$\mathbf{z} = \mathbf{P}(\mathbf{x} - \boldsymbol{\mu}) \oslash \boldsymbol{\lambda}$$

- For a whole dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$:

$$\mathbf{Z} = (\mathbf{X} - \boldsymbol{\mu})\mathbf{P}^\top \oslash \boldsymbol{\lambda} \quad \in \mathbb{R}^{N \times P}$$

[3] with $\oslash \boldsymbol{\lambda}$ indicating the element-wise division by the square roots of the variances $\boldsymbol{\lambda} = (\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_P}) \in \mathbb{R}^P$

PCA illustration

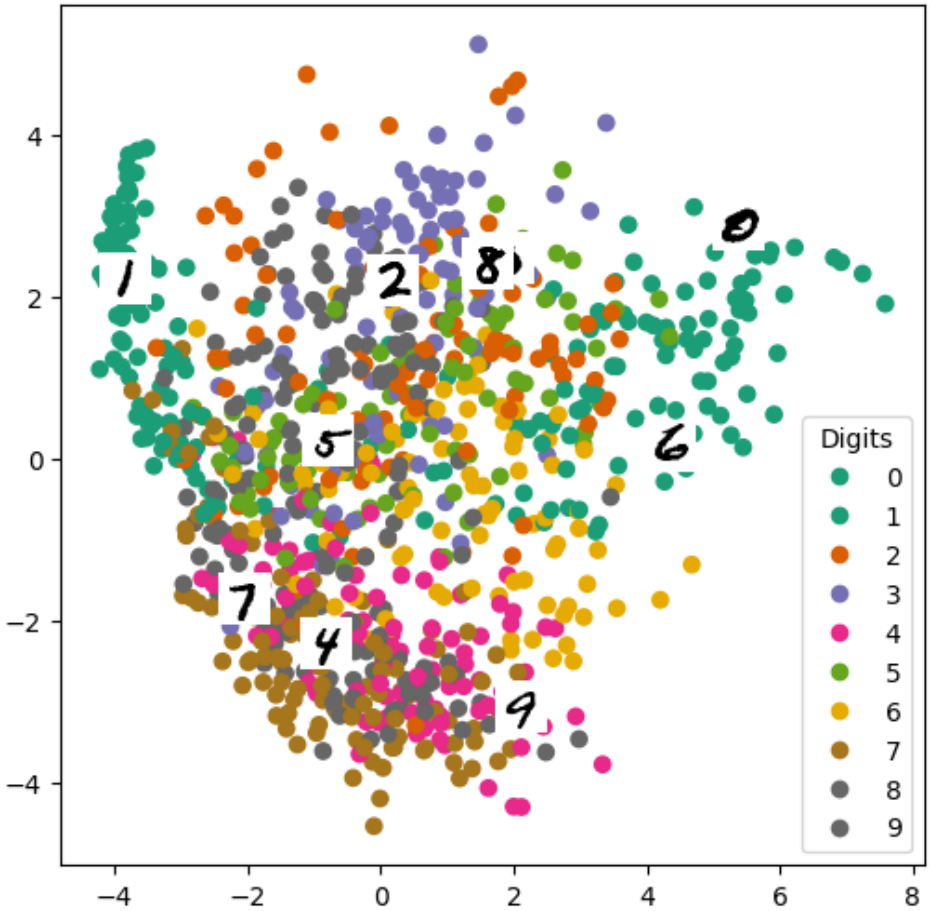


Figure: Using PCA, we can map a high-dimensional dataset ($D = 784$, 28×28 pixels per image) of handwritten digit images to 2 dimensions to visualize the data.

Inverse transform

- We can easily do the reverse transformation, to re-estimate $\hat{\mathbf{x}}$ from its low-dim representation $\mathbf{z}$[4]:

$$\hat{\mathbf{x}} = z_1(\sqrt{\lambda_1})\mathbf{p}_1 + \cdots + z_P(\sqrt{\lambda_P})\mathbf{p}_P + \boldsymbol{\mu}$$

- Matrix notation[5]:

$$\hat{\mathbf{x}} = \mathbf{P}^\top \mathbf{z} \odot \boldsymbol{\lambda} + \boldsymbol{\mu}$$

- To reconstruct a whole dataset $\hat{\mathbf{X}}$:

$$\hat{\mathbf{X}} = (\mathbf{Z} \odot \boldsymbol{\lambda})\mathbf{P} + \boldsymbol{\mu}$$

[4] Add the contribution z_p of each principal component $\mathbf{p}_p$ (rescaled by $\sqrt{\lambda_p}$), and "uncenter" by adding the mean $\boldsymbol{\mu}$

[5] $\odot$ being the element-wise product.

Desirable properties

- Now we have "compact" representations of points z in $P < D$ dimensions.
- Each coordinate $z_1, \ldots, z_P$ has mean 0 and standard deviation 1.
  - This is somehow similar to standardization we did previously for K-Means and linear regression, where $x_{\text{stand}} = \frac{x-\mu}{\sigma}$
- Additionaly, there is no covariance *i.e. no correlation* between coordinates:

$$\rho_{z_i, z_j} = 0 \quad \forall i, j$$

  - Consequently, we can sample new points in this new basis by sampling coordinates independently from one another.[6]
  - We can then project these back into the higher-dimensional space to obtain new points in the original space.
- We could also estimate the probability that a point belongs to our distribution[7]

[6] We can, and we *will* in lab 3!

[7] Out of scope but useful for eg face 3D

Data generation: examples

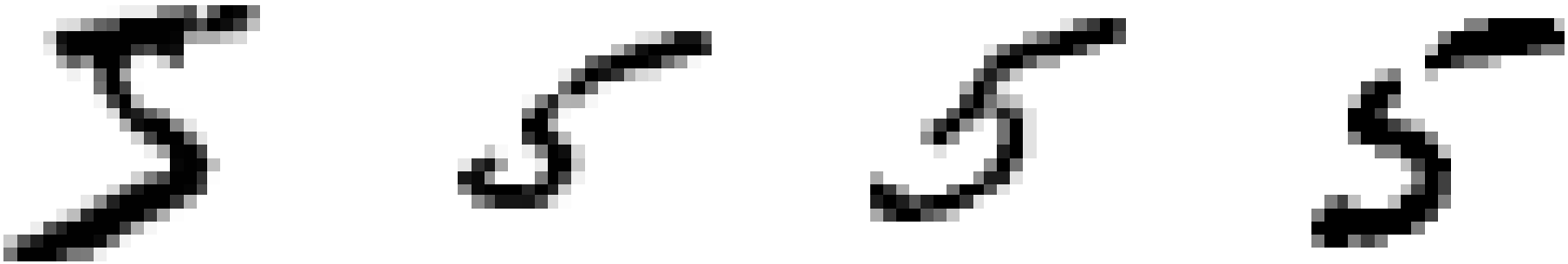


Figure: Examples of handwritten images of "5"

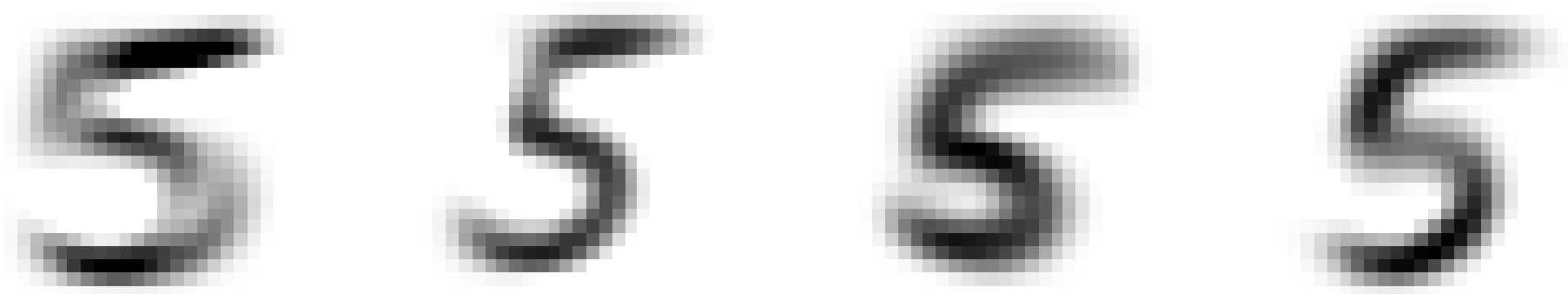


Figure: Samples generated from the same data distribution.

Limitations

- Same limitations as always: this model is linear.
- We can add non-linear features, similarly to linear regression
- However, be careful: *overfitting still exists with unsupervised learning*

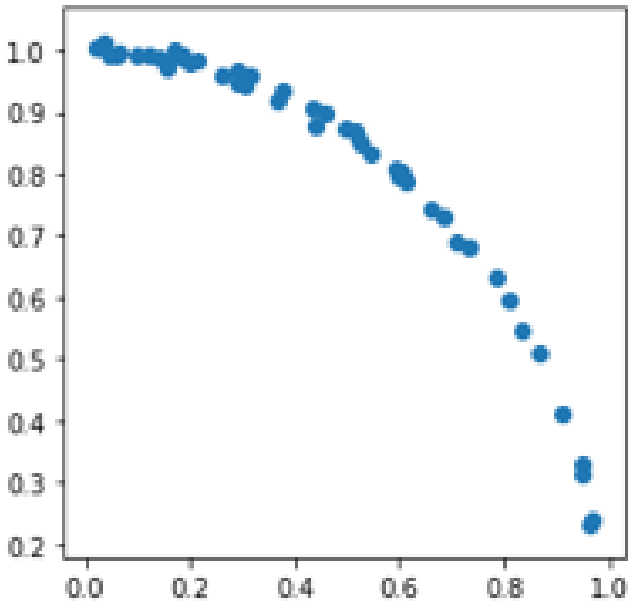


Figure: Non linear curve. PCA will fail to accurately capture this pattern.

Overfitting in unsupervised learning

In an unsupervised learning setting, we generally do not have a score that we can measure on a validation set: overfitting is thus harder to detect.

Outline

t-SNE: t-distributed Stochastic Neighbor Embedding

- t-SNE is another dimensionality reduction technique
- Unlike PCA, t-SNE can capture highly non linear structures
- It focuses on preserving local structure: this is sometimes called *manifold learning*.

Visualization

t-SNE is particularly effective for visualization.

t-SNE: t-distributed Stochastic Neighbor Embedding

- t-SNE is another dimensionality reduction technique
- Unlike PCA, t-SNE can capture highly non linear structures
- It focuses on preserving local structure: this is sometimes called *manifold learning*.

Visualization

t-SNE is particularly effective for visualization.

Be careful

t-SNE is a powerful tool, but it is easy to miss important things, or see patterns which do not exist.

t-SNE: t-distributed Stochastic Neighbor Embedding

- Key idea: convert high-dimensional Euclidean distances into conditional probabilities
- Process:
  1. Compute probability distribution over pairs of high-dimensional points
  2. Compute a similar distribution over low-dimensional points
  3. Minimize the Kullback-Leibler divergence between these distributions
- Uses t-distribution in low-dimensional space to allow dissimilar objects to be modeled far apart

t-SNE: t-distributed Stochastic Neighbor Embedding

- Key idea: convert high-dimensional Euclidean distances into conditional probabilities
- Process:
  1. Compute probability distribution over pairs of high-dimensional points
  2. Compute a similar distribution over low-dimensional points
  3. Minimize the Kullback-Leibler divergence between these distributions
- Uses t-distribution in low-dimensional space to allow dissimilar objects to be modeled far apart

Long story short: it is fairly complicated

A detailed explanation of how t-SNE works is (unfortunately) outside of the scope of this class.[a]

[a] If you are curious and determined, you can check out the original paper: jmlr.org/papers/v9/vandermaaten08a.html

t-SNE visualization

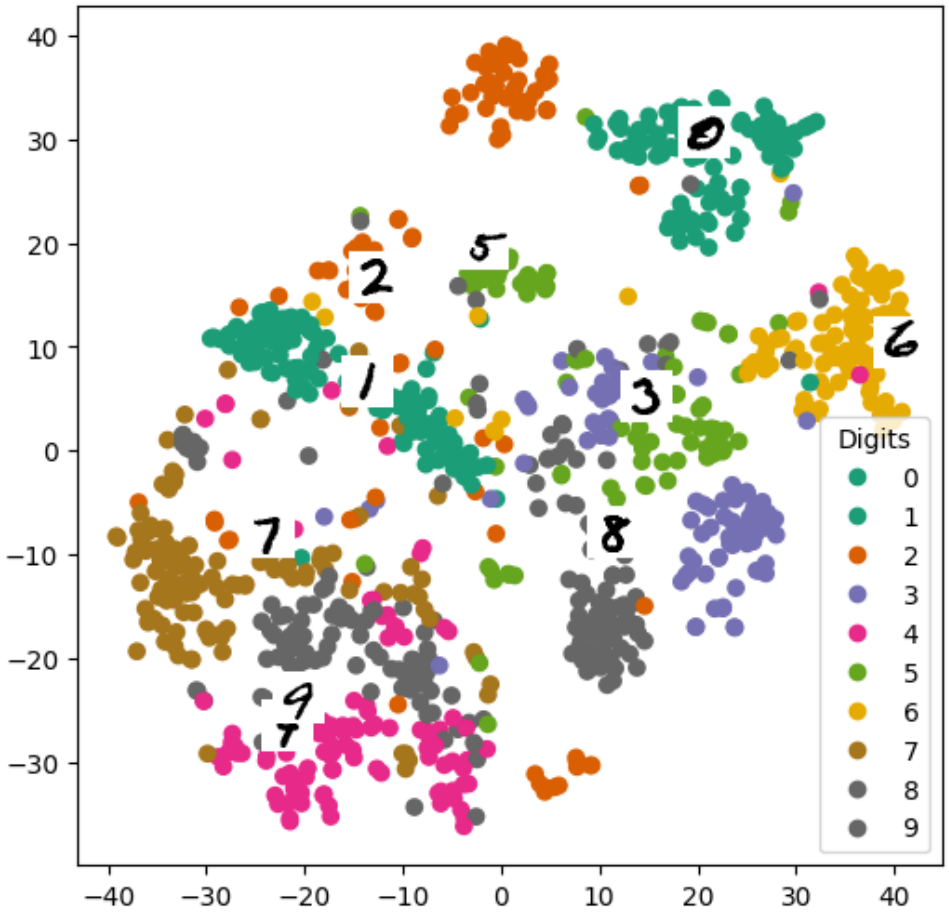


Figure: t-SNE visualization in 2 dimensions of handwritten digit images.

t-SNE in practice

Hyper-parameter

t-SNE has one main hyper-parameter: the *perplexity* p

- Low values of p will lead the model to focus more on local structures.
- High values of p will lead the model to focus more on global structures.
- There is no general rule: try different values of p (typically in $[1 - 100]$), and try to find one such that the resulting visualization seems to make sense.

t-SNE in practice

Hyper-parameter

t-SNE has one main hyper-parameter: the *perplexity* p

- Low values of p will lead the model to focus more on local structures.
- High values of p will lead the model to focus more on global structures.
- There is no general rule: try different values of p (typically in $[1 - 100]$), and try to find one such that the resulting visualization seems to make sense.

Be careful

t-SNE is a powerful tool, but it is easy to miss important things, or see patterns which do not exist.

Perplexity: example

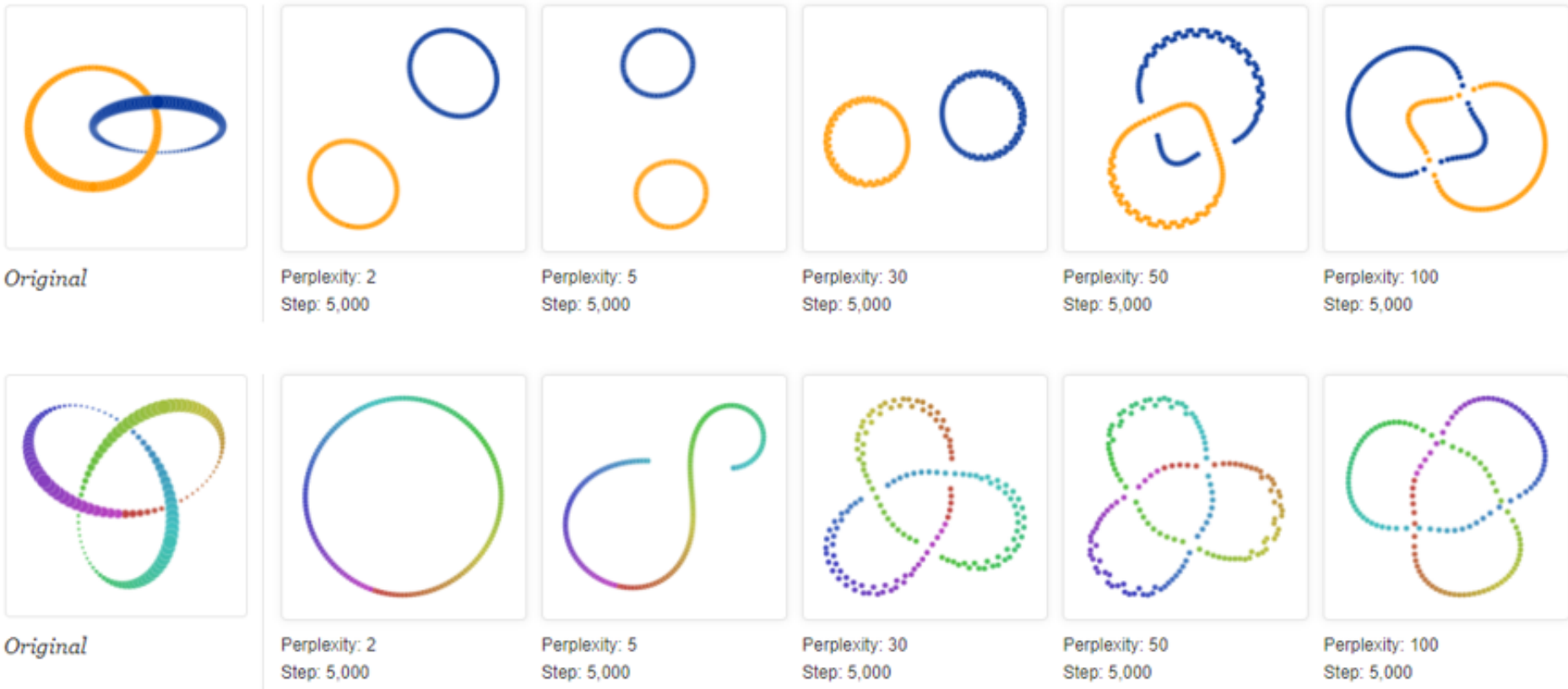


Figure: Illustration of the impact of p on two datasets.[8]

[8] Source: distill.pub/2016/misread-tsne/

Outline

Other manifold learning algorithms

- **Multidimensional scaling (MDS)**: tries to find a lower dimensional representation such that distances between all pairs of points are similar to distances in high. dim. space.
- **Spectral embedding**: roughly similar idea, but we generate a graph based on similarities and do operations on this graph (using the graph Laplacian) to achieve the desired result.
- ...