SME3006 Machine Learning – 2025 Fall

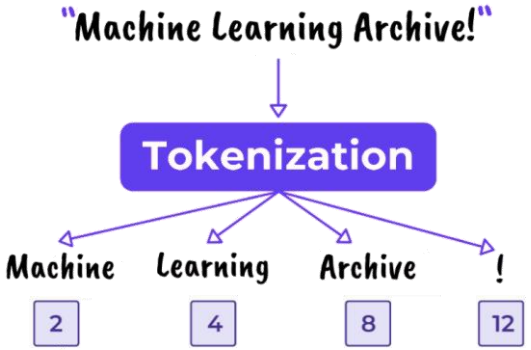
Dimensionality reduction and PCA

INHA UNIVERSITY

Why dimension reduction?

- high-resolution images
  - 1080p: 1920 * 1080 (pixels) * 3 (RGB) = 6,200,800 dims
  - Raw data~ 6Mb
  - image compression: PNG: 1~3Mb, JPEG: 0.1~0.5Mb

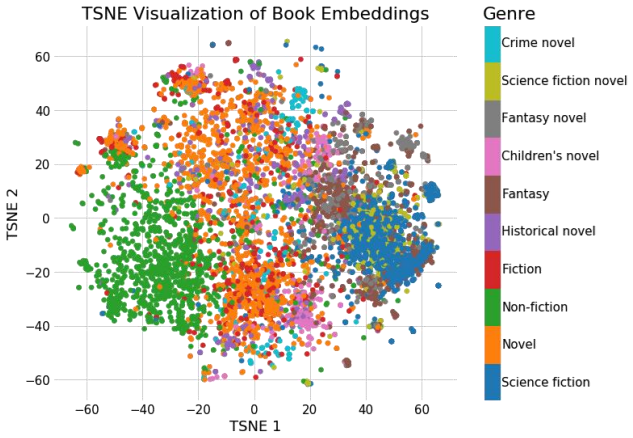
- language tokens
  - One-hot encoding
    - Dog = [1, 0, 0, ...], Cat = [0, 1, 0, ...], Rabbit = [0, 0, 1, ...] $x \in \mathbb{R}^{\dim(V)}$
    - Oxford English Dictionary $\dim(V) \approx 5\sim600{,}000$
    - 표준국어대사전 $\dim(V) \approx 4\sim500{,}000$



Why dimension reduction?

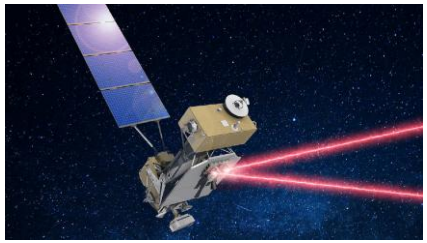
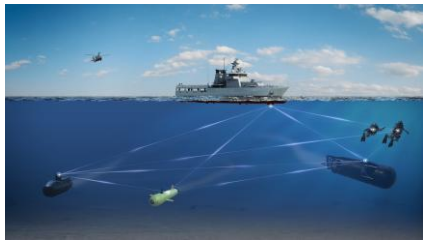
- Good things
  - memory friendly, computationally efficient
  - extract only meaningful data → performance improvement
  - avoid the curse of dimensionality
  - generalizability (prevent overfitting)
  - enable visualization (2D, 3D)

- Bad things
  - might lose some (important) information
  - might reduce interpretability
  - require extra steps to process the data



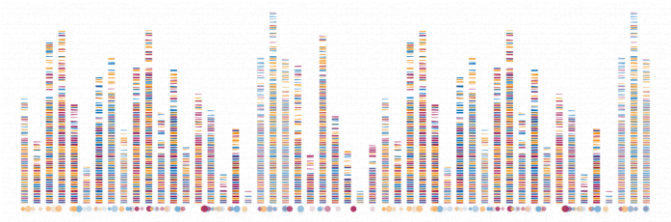
Why dimension reduction?

> Application
  - Communication

  - Robot control / autonomous driving

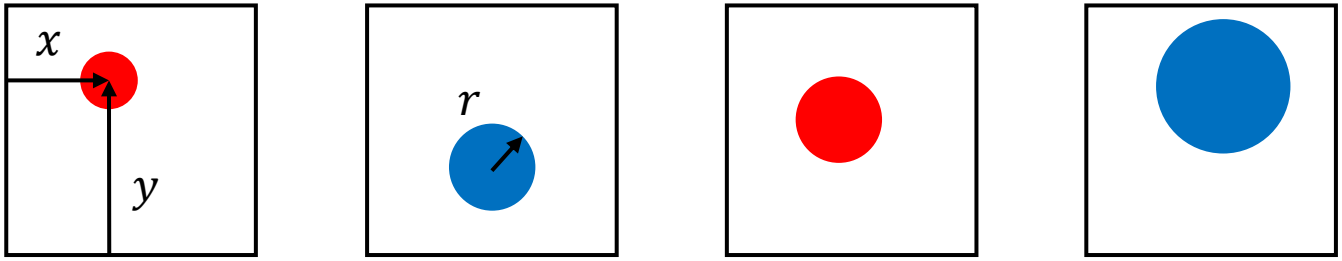

  - Bio informatics



Feature extraction

> Ex) distinguishing a red ball and a blue ball

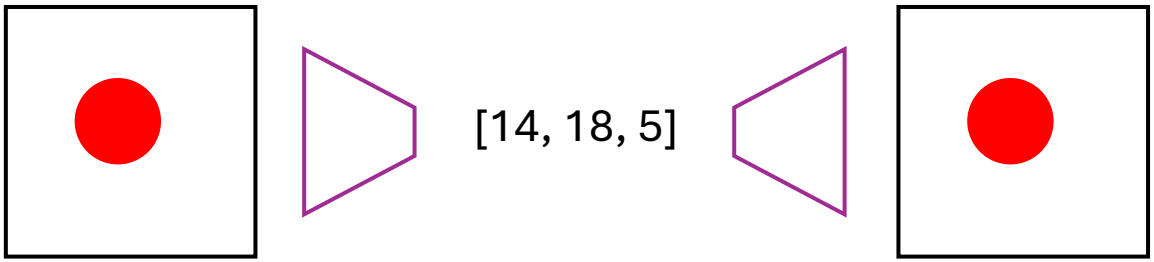
- image inputs (32*32*3=3072 dims)



- features: position (x, y), radius (r), color (a.k.a latent vector)

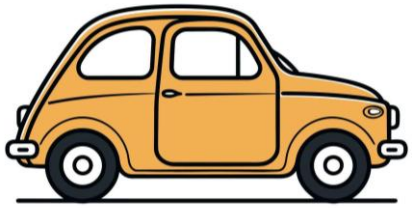
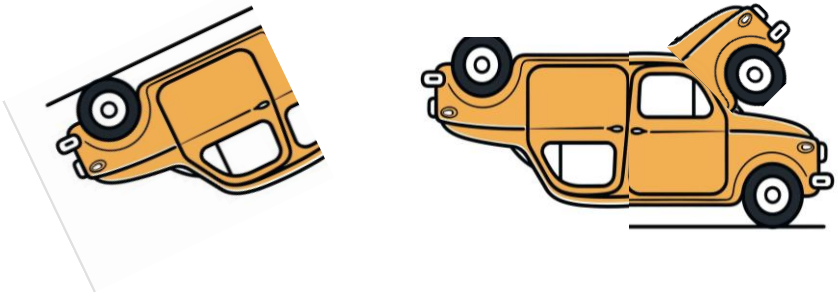
[13, 23, 3] [14, 8, 5] [14, 18, 5] [18, 21, 7]

- Auto-encoder structure (embedding and reconstruction)



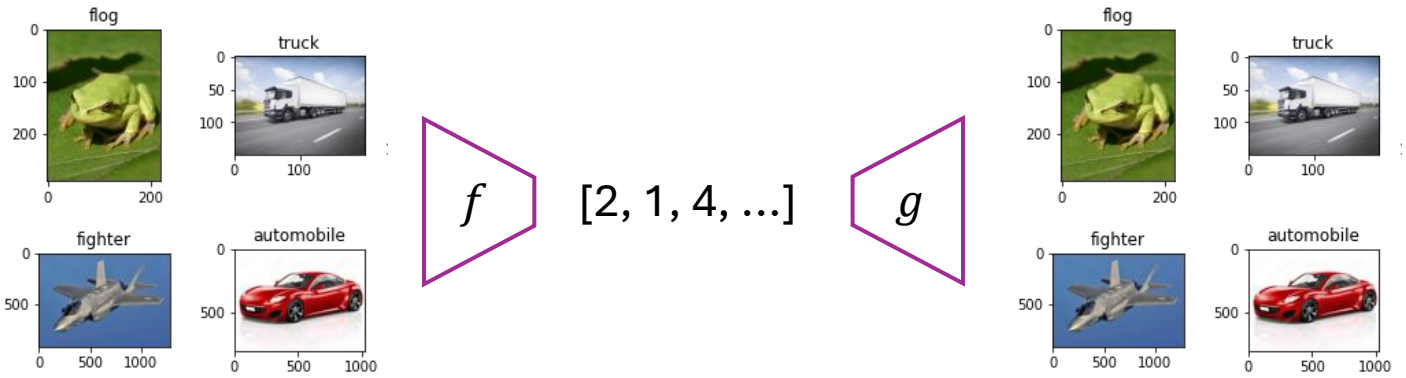
Feature extraction

> A feature can be task-specific (e.g., classification)
  - an individual measurable property or characteristics of a data sample

- car  : wheels, windows, doors, headlights, …
- dog  : legs, tail, fur texture, ears, nose, …
-  Are they still cars?

Feature extraction

> Auto-encoder: $\text{minimize}_\theta \left\| x - g\left(f(x)\right) \right\|^2$



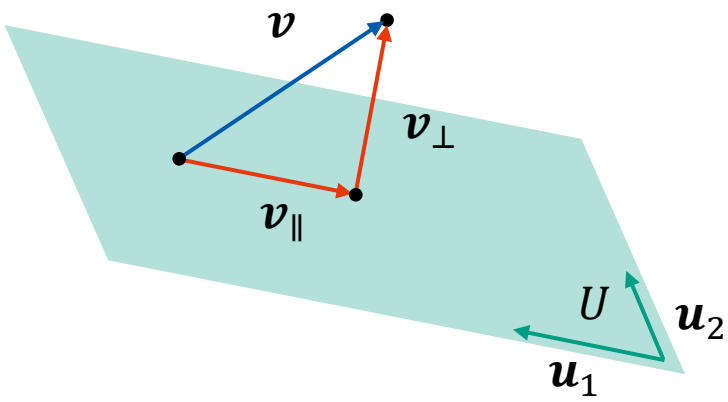
- There is no labels
- It is an unsupervised learning
- objective: while preserving the essential structure and information of the data, learn low-dimensional representations

- dimension reduction → feature extraction → representation learning

Linear dimension reduction

- Map high-dim. data into a low-dim. space using linear transformations
  - $z = W^\top x,$ where $W \in \mathbb{R}^{D \times d}$, $\mathrm{d} < D$
  - W is called a projection matrix

  - recall linear projection (from linear algebra)

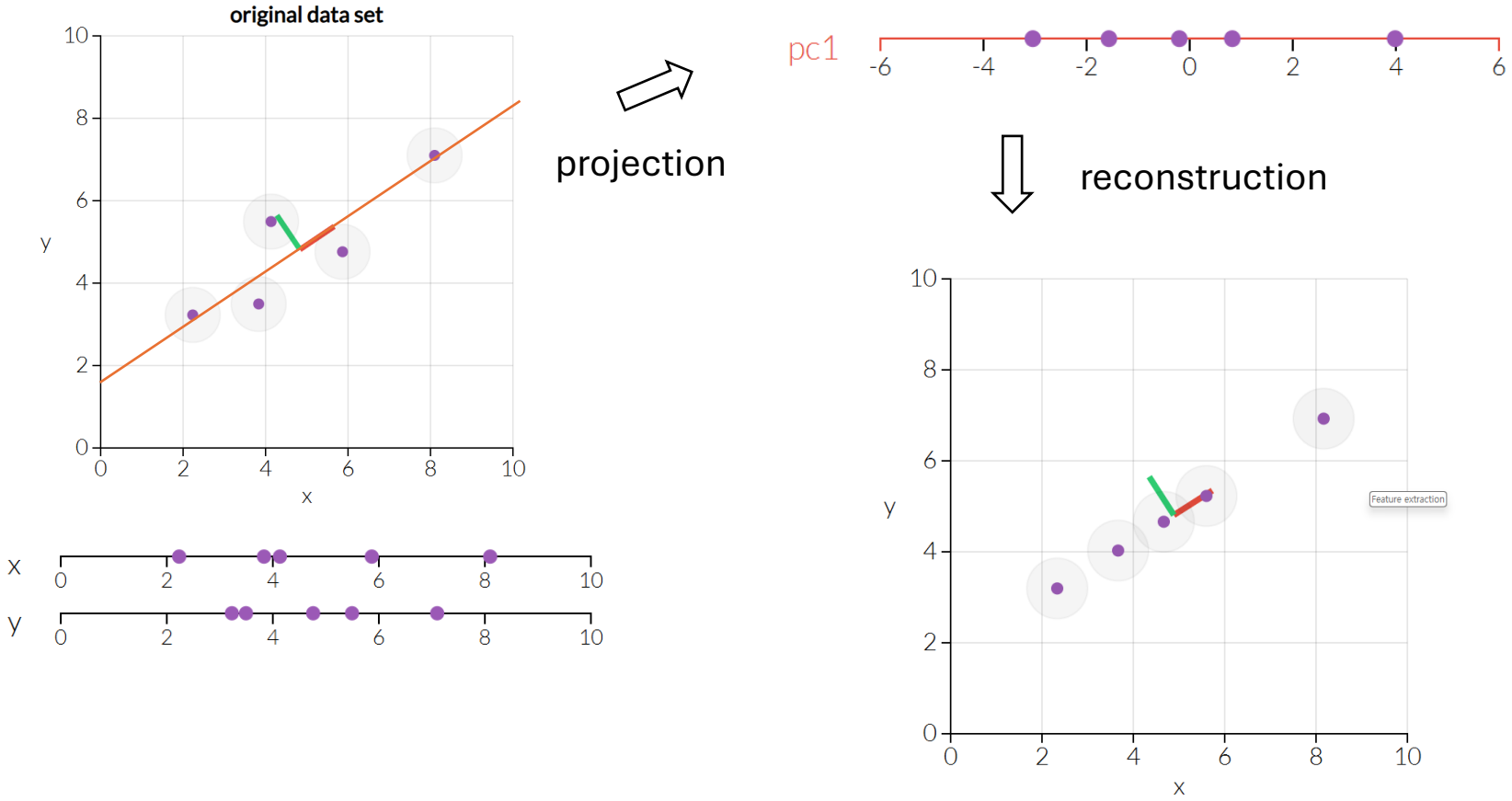


$$Q = [\boldsymbol{u}_1 \quad \dots \quad \boldsymbol{u}_k]$$

$$\boldsymbol{v}_\parallel = Q(Q^\top Q)^{-1} Q^\top \boldsymbol{v}$$

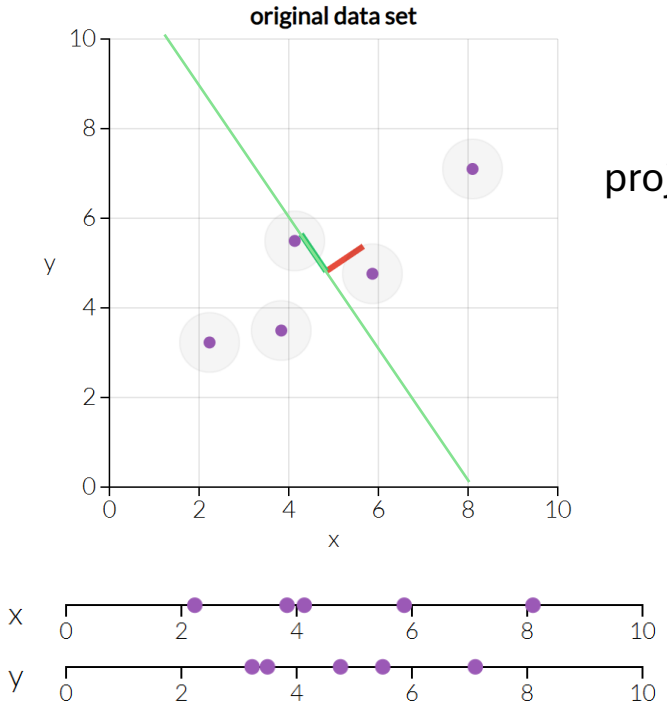
Linear dimension reduction

> linear projection onto 1-dimension

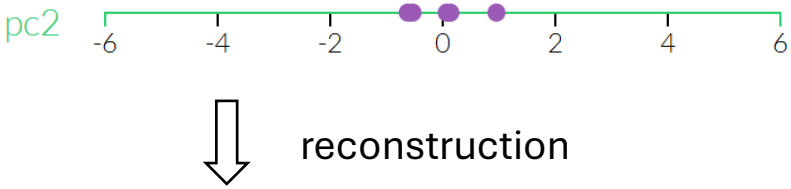


Linear dimension reduction

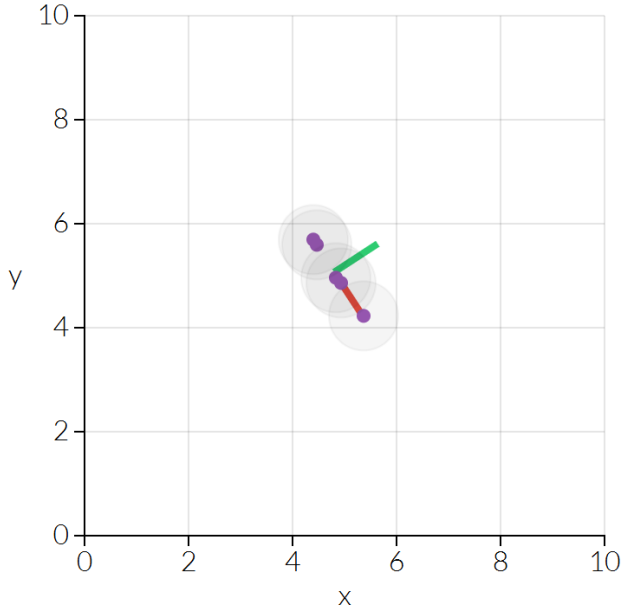
> linear projection onto 1-dimension



projection

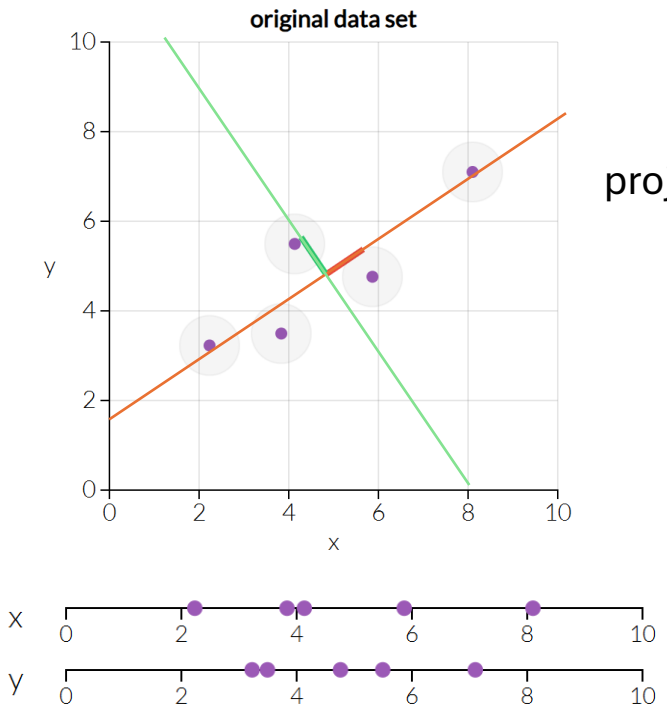


reconstruction



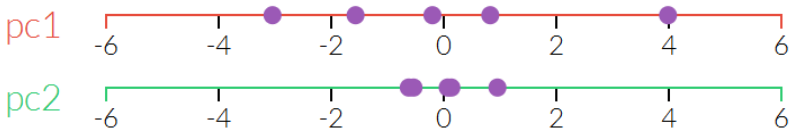
Linear dimension reduction

> linear projection onto 2-dimension



Linear dimension reduction

- linear projection
  - if I had to choose one of those vectors, which do I prefer?



- Principal component analysis (PCA)
  - project d-dimensional data into k-dimensional space using linear transformation while preserving as much as possible
  - choose projection with minimum reconstruction error

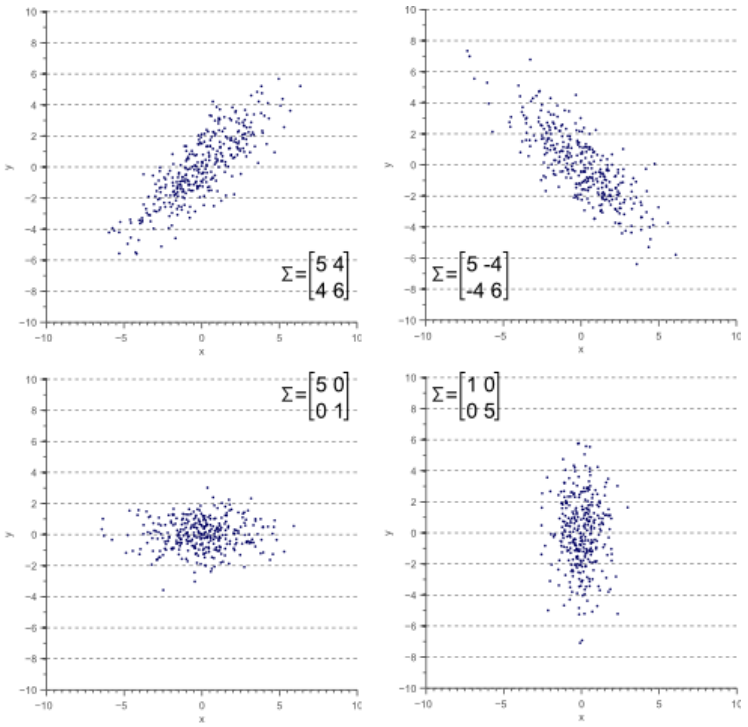
PCA

- 1) preprocessing data
  - data is centered, so that every variable has mean 0
  - (optional) we can normalize the data to have variance 1
    not usual (variance can be meaningful)
    yes, when we have different units of measurement

  - $\boldsymbol{x_i} \leftarrow \boldsymbol{x_i} - \boldsymbol{\mu}$
  - covariance matrix $\boldsymbol{v} = \frac{\boldsymbol{x}^\top \boldsymbol{x}}{\boldsymbol{n}}$

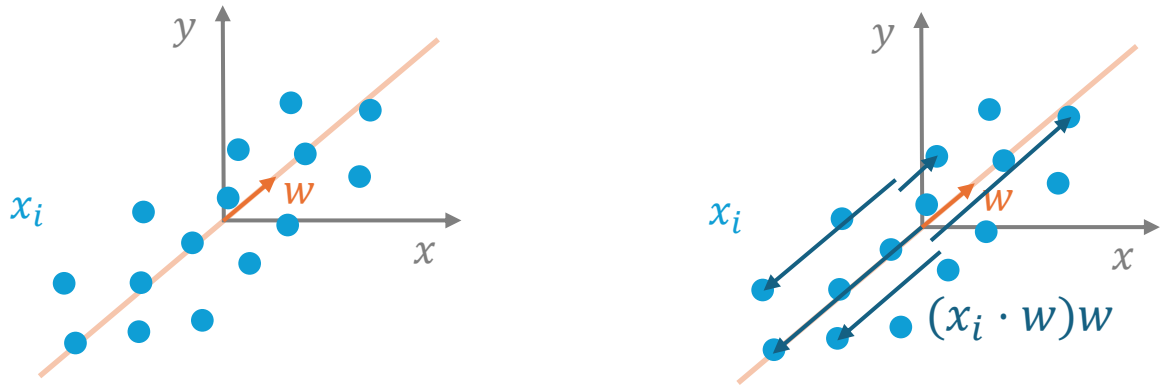
$$Var(X) = \frac{\sum(X_i - X)^2}{N} = \frac{\sum x_i}{N}$$

$$Cov(X, Y) = \frac{\sum(X_i - X)(Y_i - Y)}{N} = \frac{\sum x_i y_i}{N}$$



PCA

> 2) project to 1-dim vector
  - project x_i on to a line through the origin with direction w (unit vector)
  - distance of the projection from the origin: $(x_i \cdot w)w$



  - residual of the projection: $\|x_i - (x_i \cdot w)w\|^2 = x_i \cdot x_i - (x_i \cdot w)^2$

PCA

- 2) project to 1-dim vector
  - residual of the projection: $\|x_i - (x_i \cdot w)w\|^2 = x_i \cdot x_i - (x_i \cdot w)^2$
  - Add those residuals up across all the vectors
    $MSE(w) = \frac{1}{n}\sum_{i=1}^{n}[x_i \cdot x_i - (x_i \cdot w)^2]$

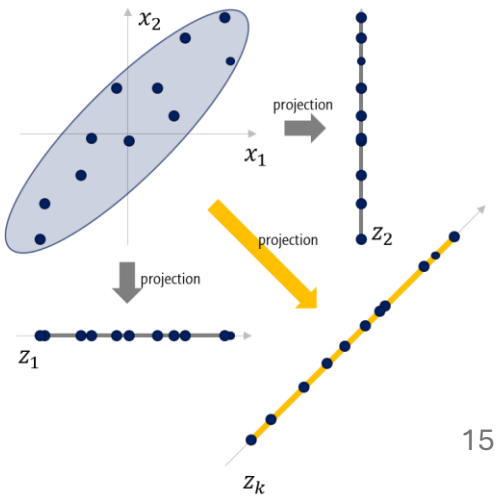
$$= \frac{1}{n}\sum_{i=1}^{n}\|x_i\|^2 - \left(\frac{1}{n}\sum_{i=1}^{n} x_i \cdot w\right)^2 - Var[x_i \cdot w]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\|x_i\|^2 - Var[x_i \cdot w] \quad \text{(the mean of the projection is zero)}$$

  - minimizing the residual sum of squares
    = maximizing the variance of the projections

  - In general, we project on to multiple principal components those are orthogonal, having the unit vectors $w_1, w_2, \ldots, w_k$

Find space z_k which maximize variance of projected data



PCA

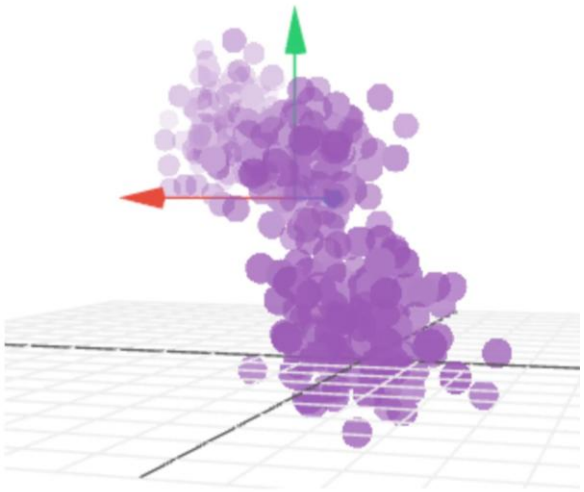
- 3) maximizing the variance
  - let $\boldsymbol{x} = [x_1 \;\; x_2 \;\; \dots \;\; x_n]^\top \in \mathbb{R}^{n\times p}, \;\; w \in \mathbb{R}^{p\times 1}$
  - variance $\sigma_w^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i \cdot w)^2 = \frac{1}{n} w^\top \boldsymbol{x}^\top \boldsymbol{x} w = w^\top v w, \; v = \mathrm{cov}(x)$
  - we want to choose w so as to maximize σ_w^2
  - Eigenvalue decomposition (diagonalization)
    - since covariance matrix is symmetric and positive semi-definite
    - $\boldsymbol{v} = Q\Lambda Q^\top$ where Q is an orthogonal matrix, composed of eigenvectors
      $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \dots, \lambda_p), \;\; \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$
  - Any vector can be represented by eigenvectors (basis)
    - $w = Qa, \quad \|w\|^2 = 1 \rightarrow \|a\|^2 = 1$
  - $w^\top v w = a^\top Q^\top (Q\Lambda Q^\top) Q a = a^\top \Lambda a$
  - maximizing $a^\top \Lambda a \;\; s.t. \|a\|^2 = 1 \;\; \rightarrow \;\; a = [1, 0, 0, \dots, 0]^\top$
    - $w = Qa = q_1$: normalized eigenvector corresponding to the maximum eigenvalue

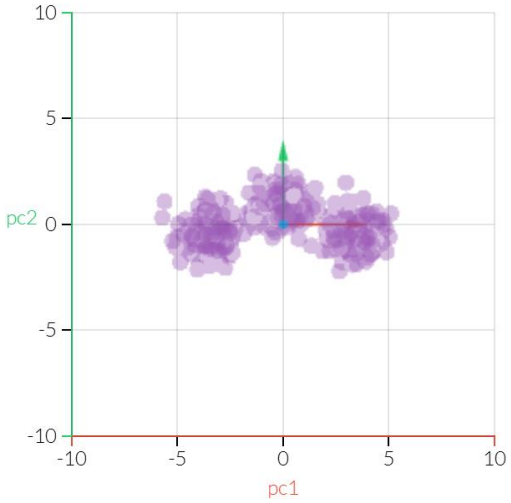
PCA

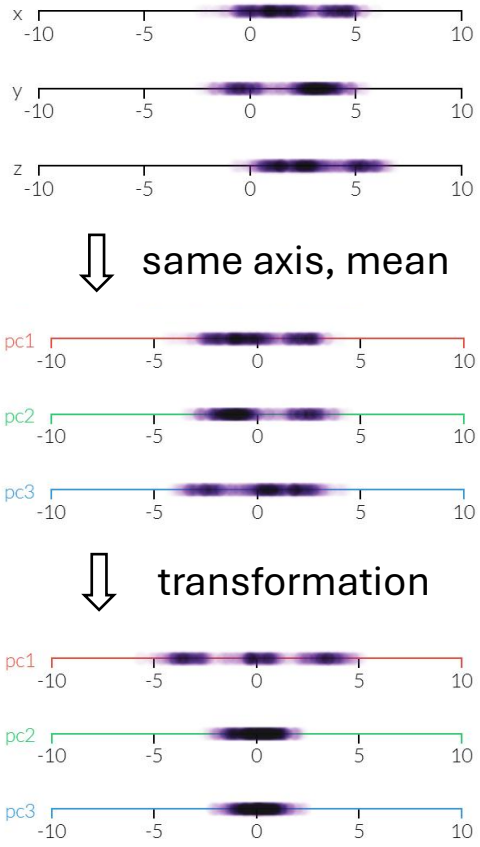
- Basic PCA algorithm
  - 1) re-center: subtract mean from each row of x, $x_i \leftarrow x_i - \mu$
  - 2) compute covariance matrix: $v = \frac{1}{n} x^\top x$
  - 3) find eigenvectors q_i and eigenvalues λ_i of v
  - 4) principal components: k eigenvectors with the highest eigenvalues
    $\hat{x}_i = \sum_{j=1}^{k} (q_j \cdot x_i) q_j = x_i^\top Q_k$ where $Q_k = [q_1 \ q_2 \ \ldots q_k]^\top$

PCA

> 3D example

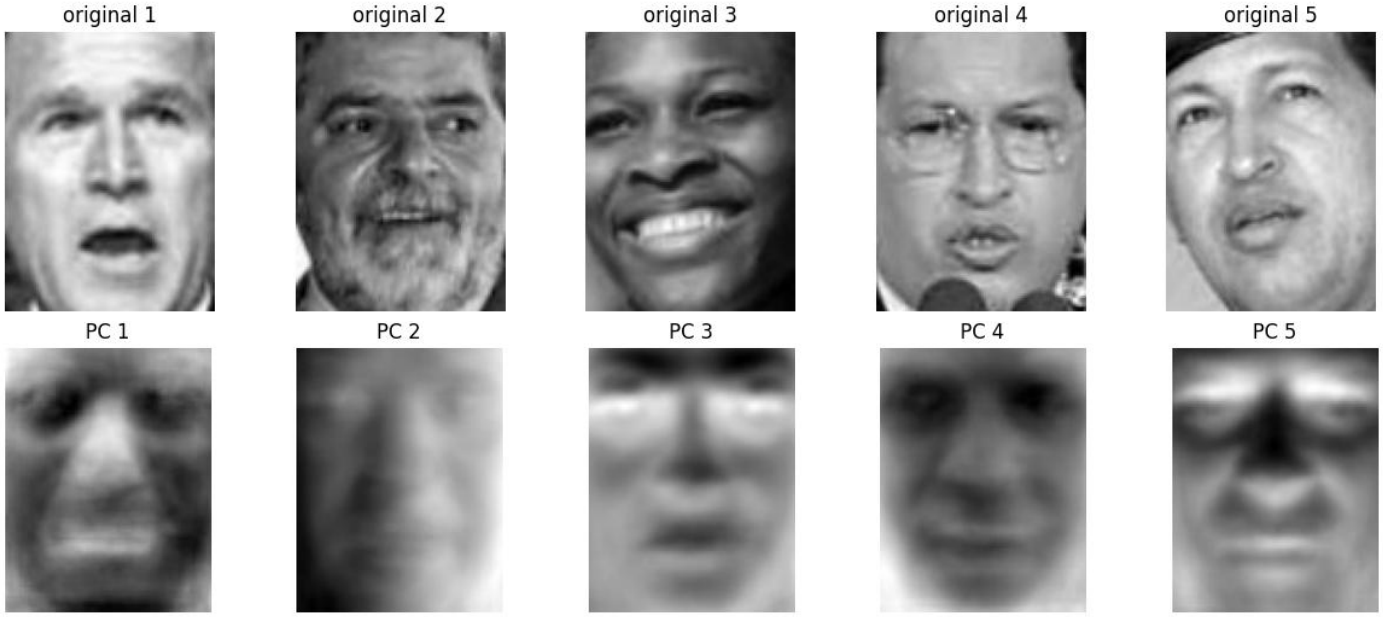


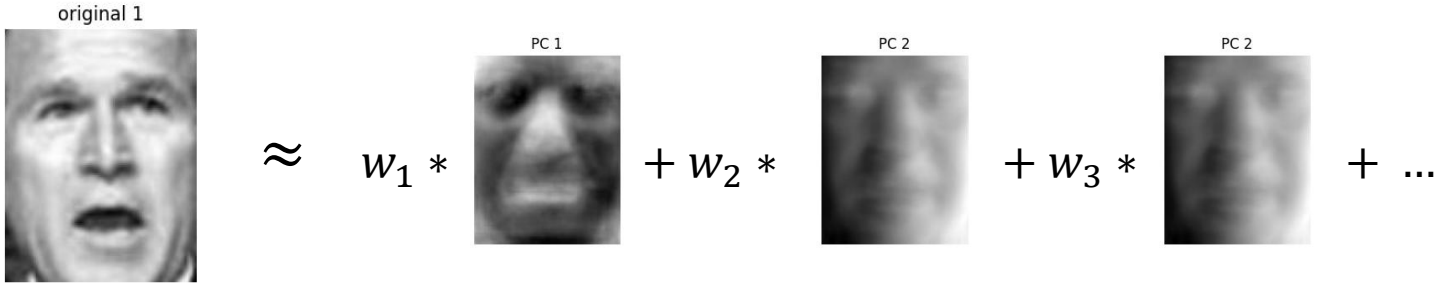




PCA

> Application: eigenface



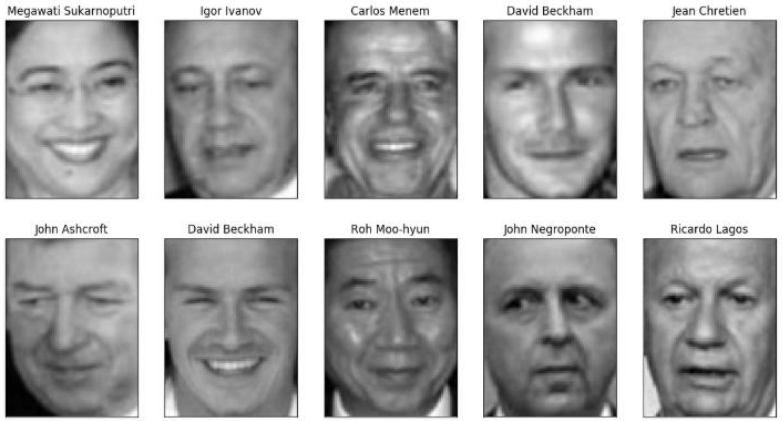


PCA

> Application: eigenface

PCA

- Application: eigenface
  - reconstruction error occurs when we have minor features



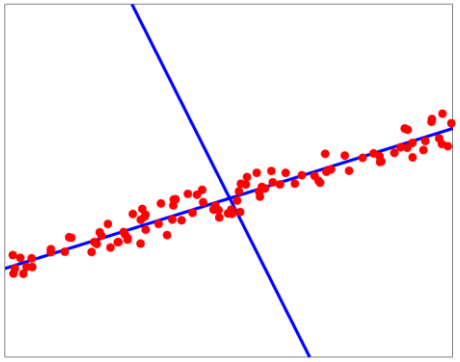
Best reconstructions

Worst reconstructions

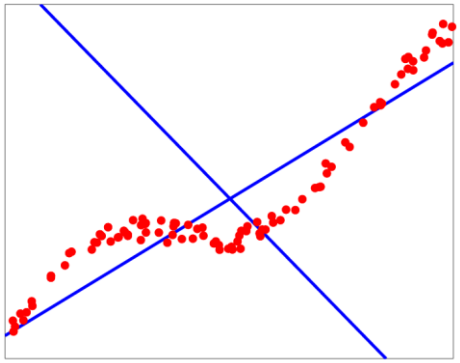
PCA

- Working examples
  - the first eigenvalue accounts for most variable, so the dimensionality is 1

Example: $\lambda_1 = 0.0938, \lambda_2 = 0.0007$

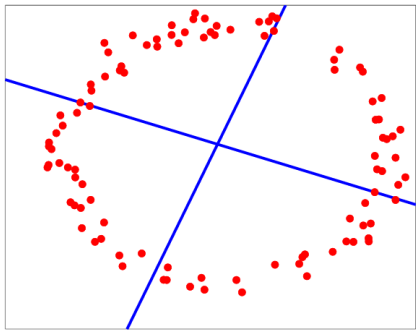


Example: $\lambda_1 = 0.1260, \lambda_2 = 0.0054$



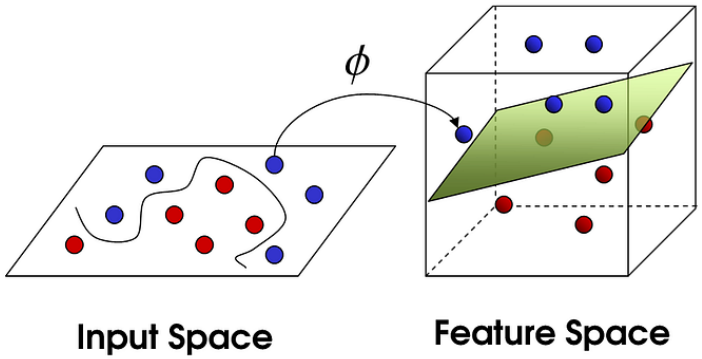
- Difficult examples
  - true non-linear dimension of the data is 1 (using polar coordinates)

Example: $\lambda_1 = 0.0884, \lambda_2 = 0.0725$



Kernel PCA

- Making PCA non-linear
  - instead of using x_i, we use some different feature space $\phi(x_i) \in \mathbb{R}^p$
  - e.g. using polar coordinates instead of cartesian coordinates
  - covariance matrix $v = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i)\phi(x_i)^\top = \frac{1}{n}\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x})$
  - Kernel: $K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$
    - we don't have to know the $\phi(x_i)$, we only need $K(x_i, x_j)$
    - given any algorithm that can be expressed solely in terms of dot products, this kernel trick allows us to construct difficult nonlinear versions of it



Kernel PCA

- 3-c) maximizing the variance
  - let $\boldsymbol{\phi}(\boldsymbol{x}) = [\phi(x_1)\ \ \phi(x_2)\ \ \ldots\ \ \phi(x_n)]^\top \in \mathbb{R}^{n\times p},\quad w \in \mathbb{R}^{p\times 1}$
  - $\sigma_w^2 = \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)\cdot w)^2 = \frac{1}{n} w^\top \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}) w = w^\top v_\phi w,\ v_\phi = \mathrm{cov}(\boldsymbol{\phi}(\boldsymbol{x}))$
  - we want to choose w so as to maximize σ_w^2
  - Since $\phi(\mathrm{x})$ has (infinite dimensional) basis
    - Any vector can be represented by eigenvectors (basis)
    - $w = \boldsymbol{\phi}(\boldsymbol{x})^\top a, \quad \|w\|^2 = 1 \rightarrow a^\top \boldsymbol{\phi}(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})^\top a = 1 \rightarrow a^\top \boldsymbol{K} a = 1$
  - $w^\top v_\phi\, w = a^\top \boldsymbol{\phi}(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})^\top a = a^\top \boldsymbol{K}^2 a$
  - maximizing $a^\top \boldsymbol{K} a \quad s.t.\, a^\top \boldsymbol{K} a = 1$
    - $\boldsymbol{K} a = n\lambda a$, pick the largest eigenvalue and corresponding eigenvector a and normalize it to satisfy $a^\top \boldsymbol{K} a = 1$

Kernel PCA

- Popular kernels
  - Polynomial: $K(x_i, x_j) = (x_i^\top x_j + c)^d$
  - RBF (Gaussian): $K(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$
  - Sigmoid: $K(x_i, x_j) = \tanh(\alpha x_i^\top x_j + c)$
  - Fisher, Neural tangent, Laplacian, Bessel, …

- Normalizing the feature space
  - $\tilde{\phi}(x_i) = \phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\phi(x_k)$
  - $\tilde{K}(x_i, x_j) = \tilde{\phi}(x_i)^\top \tilde{\phi}(x_j)$
  - $\tilde{\boldsymbol{K}} = \boldsymbol{K} - \frac{2}{n}\sum_{k=1}^{n} K(x_i, x_k) + \frac{1}{n^2}\sum_{j,k=1}^{n} K(x_j, x_k)$
    $= \boldsymbol{K} - 2\,\mathbf{1}_{1/n}\boldsymbol{K} + \mathbf{1}_{1/n}\boldsymbol{K}\mathbf{1}_{1/n}$
    where $\mathbf{1}_{1/n}$ is a matrix with all elements 1/n

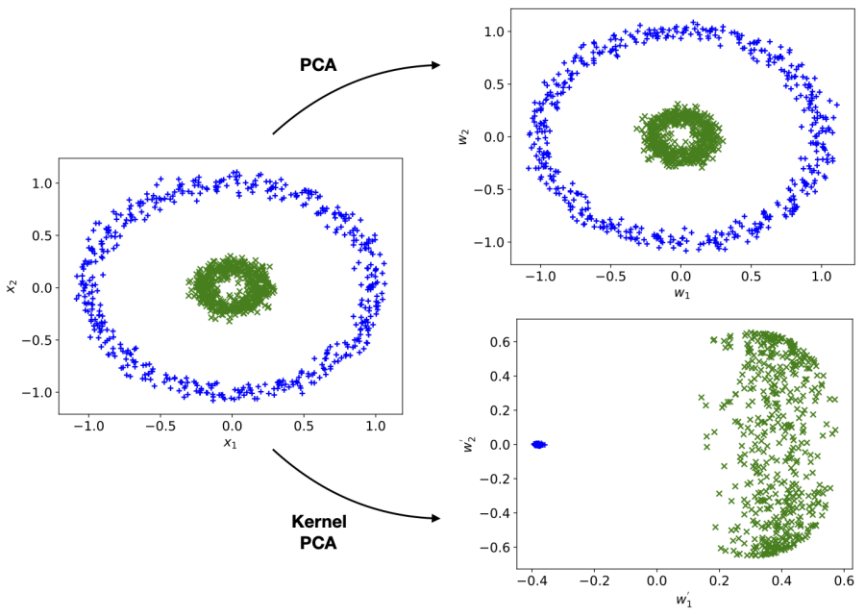
Kernel PCA

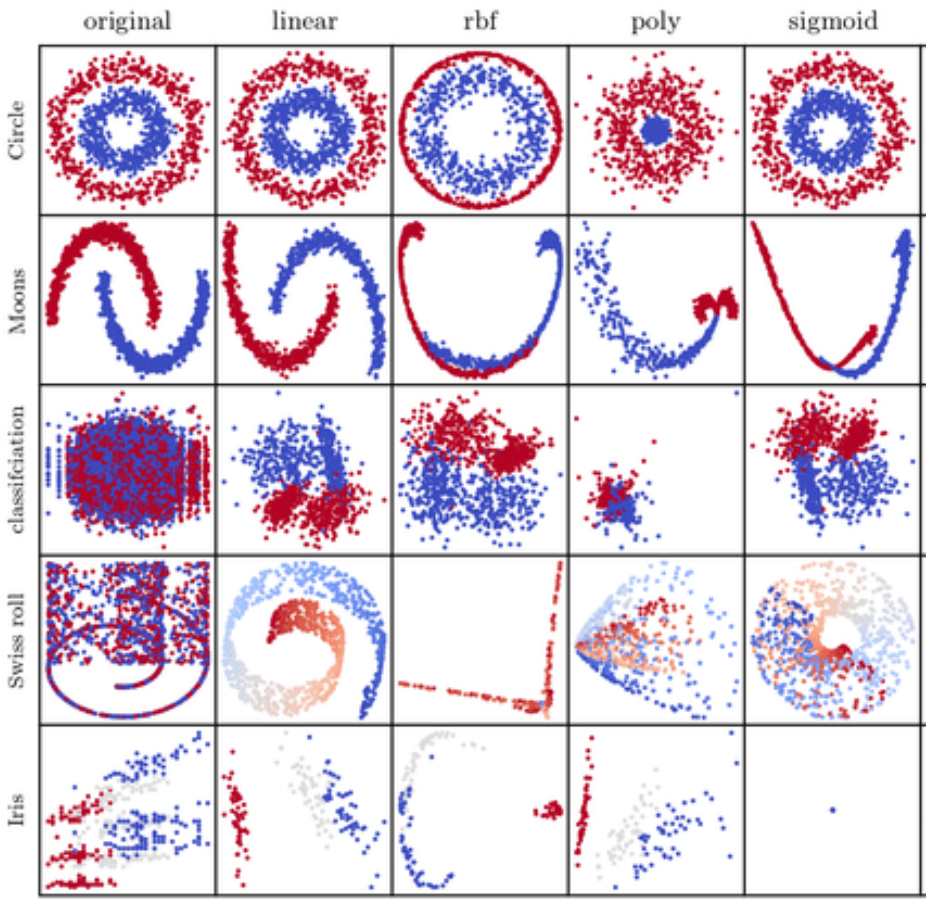
- Algorithm
  - 1) pick a kernel
  - 2) construct the normalized kernel matrix $\widetilde{\boldsymbol{K}}$ of the data
  - 3) find eigenvectors q_i and eigenvalues λ_i of $\widetilde{\boldsymbol{K}}$
  - 4) principal components: k eigenvectors $q_1, \ldots, q_k$ with the highest eigenvalues $\hat{z}_i(x) = \sum_{j=1}^{n} q_{ij} K(x_j, x)$

- Comparison to basic PCA
  - can capture complex nonlinear structure
  - works well for clustering and visualization
  - computationally expensive
  - requires storing full kernel matrix
  - difficult to interpret components
  - can't easily reconstruct original data

Kernel PCA

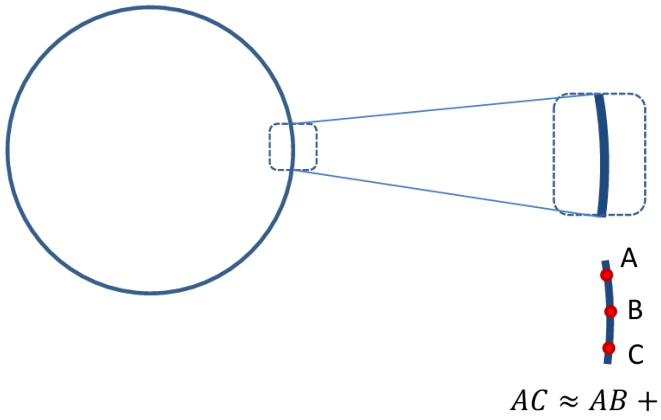
> Comparison to basic PCA / depending on the choice of kernel functions

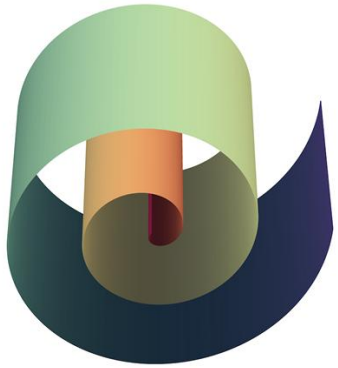




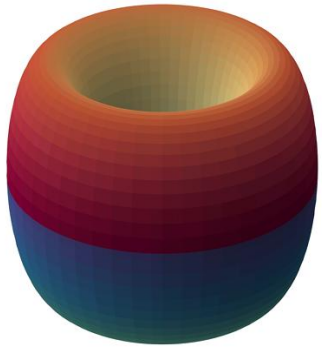
Nonlinear dimension reduction

- Assuming data are supported on smooth nonlinear low dimensional geometric objects
- Manifold: a generalization of curves and surfaces
  - a subset of points in the high-dimensional space that locally looks like a low-dimensional space





(a) Swiss Roll

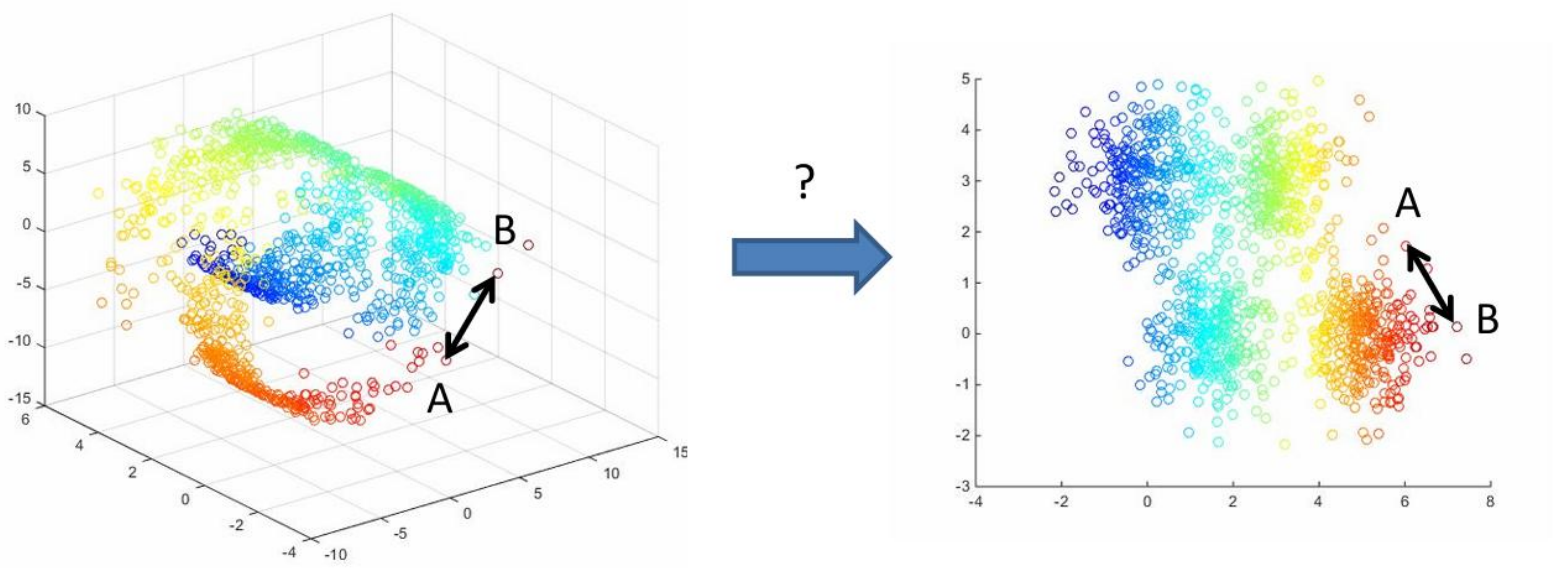


(b) Torus

  - idea is that once the manifold is unfolded, the analysis becomes easy

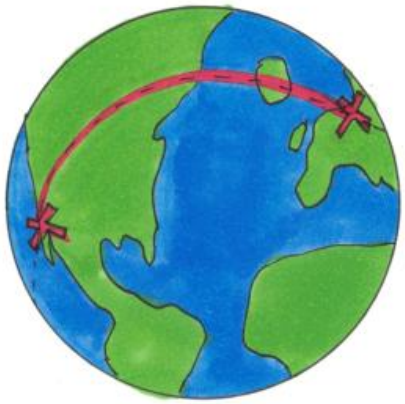
Nonlinear dimension reduction

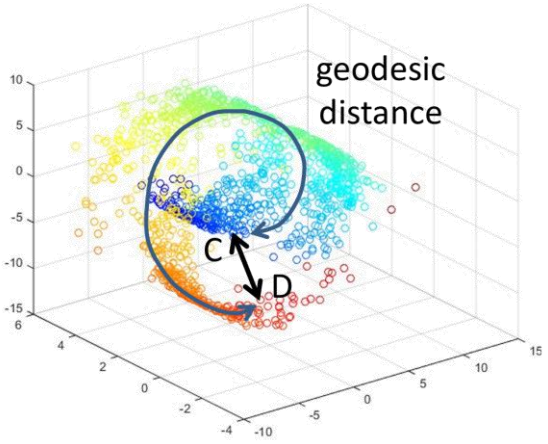
- Key assumption: high dimensional data actually resides in a lower dimensional space that is locally Euclidean
  - find a lower dimensional representation of data that preserves distances between points (MDS)



Geodesic distance

- Multidimensional scaling (MDS)
  - find a lower dimensional representation of data that preserves distances between points
  - it doesn't have to be Euclidean distance

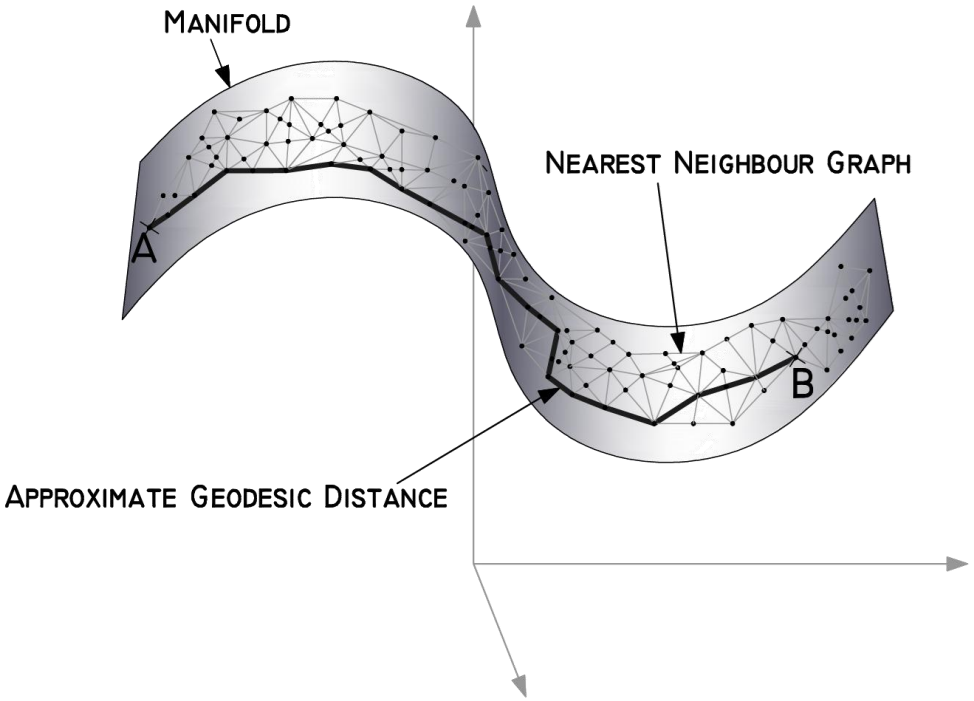




  - we are interested in preserving distances along the manifold (geodesic distances)

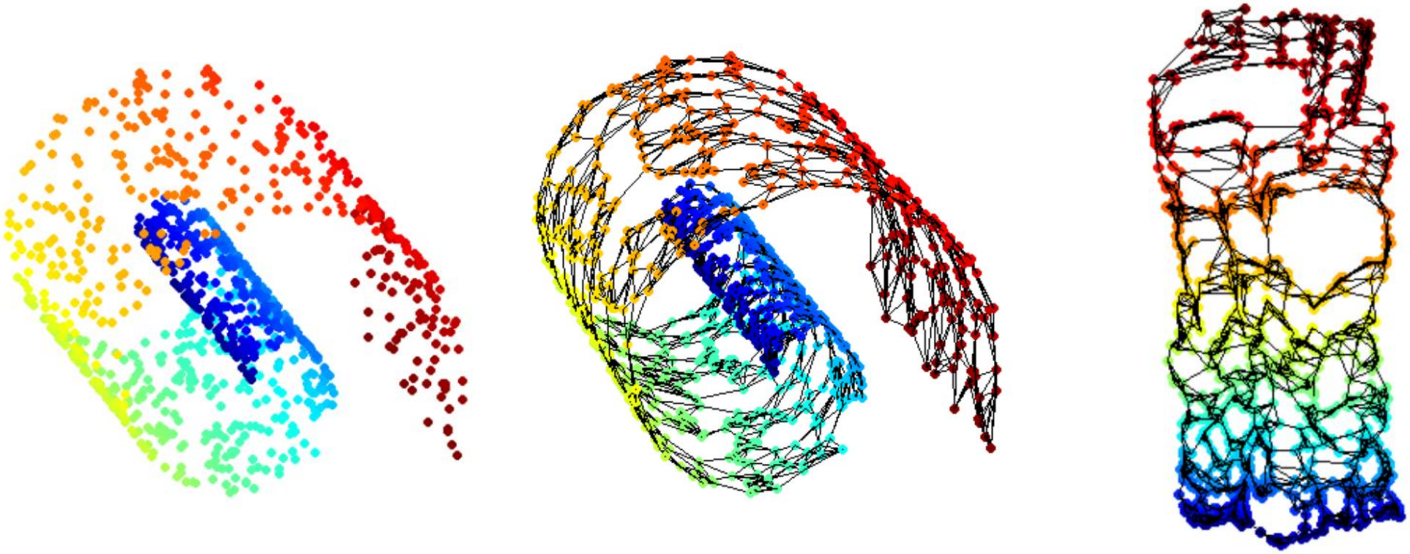
Geodesic distance

> Geodesic distances can be approximated using a graph in which vertices represent data points
  - option1: define a local radius and connect vertices within the radius
  - option2: connect each point to its k nearest neighbors



Isomap

- Algorithm
  - construct the similarity graph
  - compute geodesic distances by finding shortest paths on the graph
  - construct the geodesic distance matrix
  - apply MDS

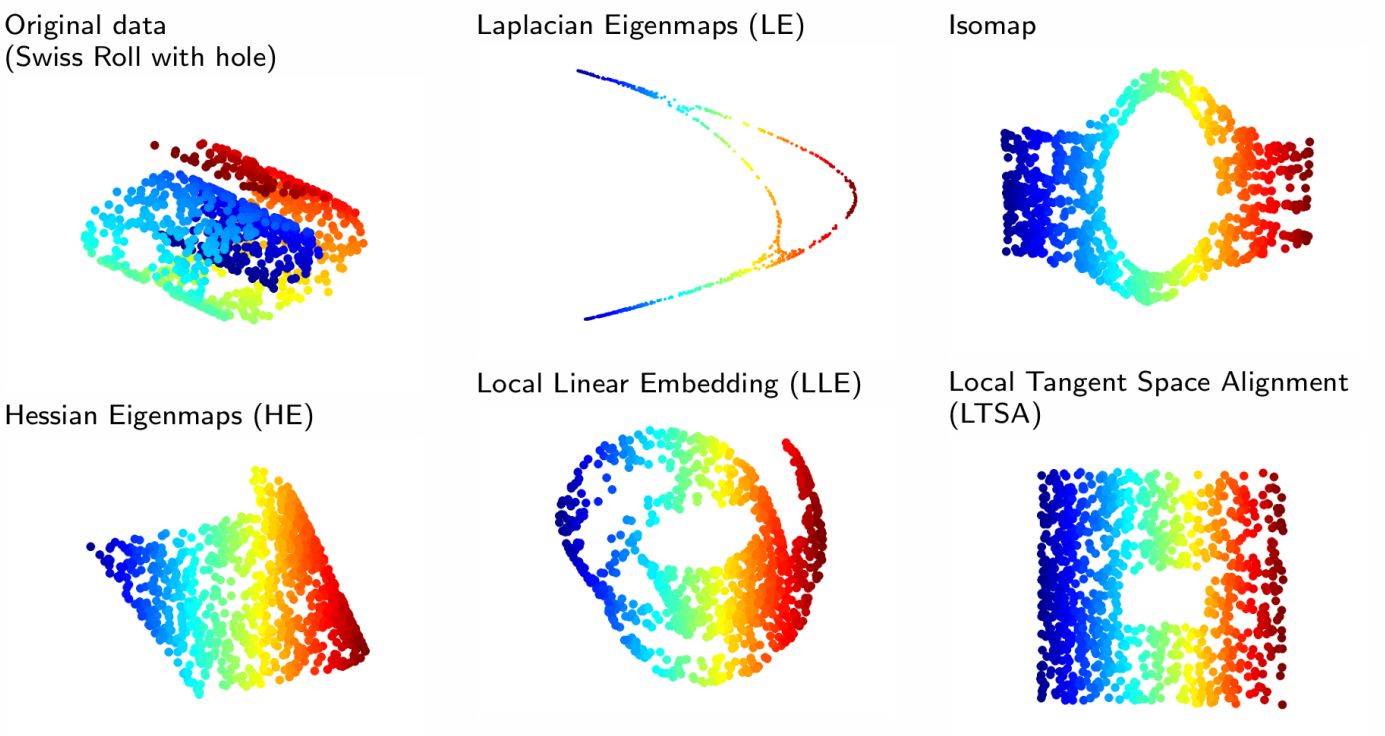


Manifold learning

- Starting from neighborhood graph and distance matrix
- Locally-linear embedding (LLE)
  - describe each point as a linear combination of it neighbors
  - finds a low-dimensional embedding that preserves these local relationships via eigenvalue decomposition
- Local tangent space alignment (LTSA)
  - estimate the tangent space at each point by performing PCA
  - produce an embedding in which the tangent spaces are aligned
- Laplacian eigenmaps (LE)
  - compute the Laplacian of the graph
  - solve the eigenvalue problem for the graph Laplacian

Manifold learning

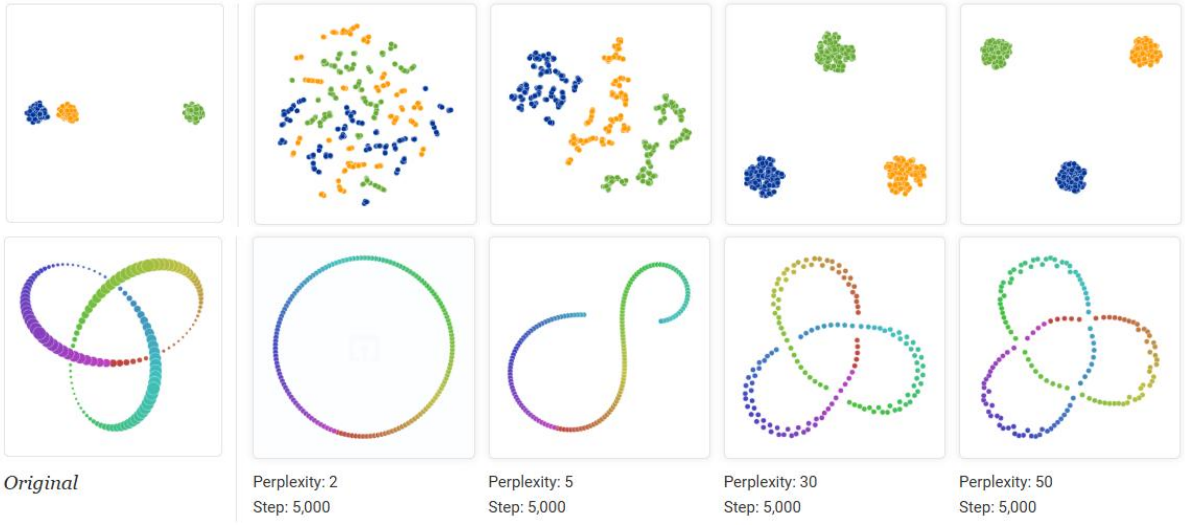
- Pros and cons
  - Isomap – best for flat manifolds with no holes, small data
  - LLE – simple and fast
  - LSTA – correct at boundaries
  - LE – local smoothness, but only topology



Manifold learning

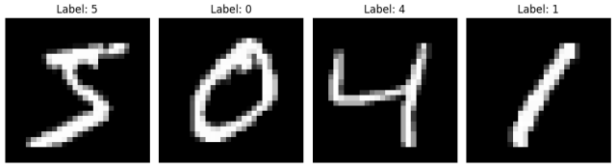
- Blur the line between dimension reduction and data visualization
  - axes don't correspond to anything in the input space
  - often can't transform new data
  - t-SNE, UMAP, LargeVis

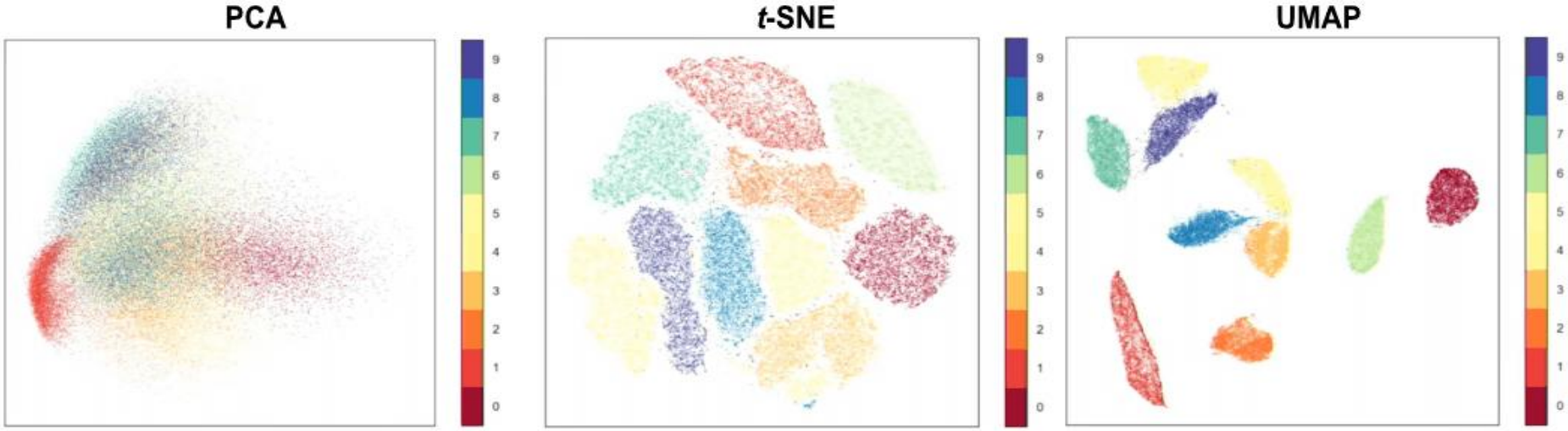
- How to use t-SNE effectively
  - https://distill.pub/2016/misread-tsne/



Manifold learning

> MNIST dataset visualization





Reference

- Dimension reduction
  - https://amueller.github.io/COMS4995-s20/slides/aml-13-dimensionality-reduction
- PCA
  - https://cs229.stanford.edu/notes2022fall/pca.pdf
  - https://setosa.io/ev/principal-component-analysis/
- Manifold learning
  - https://arxiv.org/pdf/2311.03757
  - https://sites.stat.washington.edu/mmp/Talks/mani-fields22-notes.pdf
  - https://trevorcohn.github.io/comp90051-2017/slides/16_manifold_learning.pdf